University Of Pisa

Doctoral Thesis

Posterior Probabilities, Active Learning, and Transfer Learning in Technology-Assisted Review

Author:
Alessio Molinari

Supervisors:
Dr. Andrea Esuli
Dr. Fabrizio Sebastiani

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in

Computer Science

2023

Abstract

In several subfields of data science, the term "review" refers to the activity, carried out by a human annotator (also called *reviewer*), of assigning the correct class labels to unlabelled data items in a dataset, possibly replacing wrong labels assigned by an automatic process (a *classifier*). Review might have different goals. For instance, the aim could be that of labelling data to be used for training a classifier. In other cases, the aim of the review process might be that of finding the set of "relevant" documents of the dataset: this is the case, e.g., in e-discovery, an important part of the civil litigation process in the US and other countries, where the goal of the producing party is to find all the items (e.g., emails, or other digitally stored documents) relevant to the object of the litigation. Another example is the production of a "systematic review" in evidence-based medicine, i.e., a comprehensive survey of all the medical literature relevant to a specific research question.

In many cases, the amount of documents to review may be enormous, which means that manually labelling the entire collection is an infeasible and extremely expensive operation. For this reason, the review process is usually supported by human-in-the-loop machine learning algorithms, which go under the name of Technology-Assisted Review (TAR). The goal of TAR algorithms should be that of ultimately improving the cost-effectiveness of the review process, i.e., to minimize the time and money spent on reviewing, while maximizing the accuracy and quality of the review. More specifically, reviews are carried out via an *active learning* (AL) algorithm, i.e., an iterative process which prioritizes documents for the human to review (usually via a classifier). At every iteration, the newly labelled documents are fed back to the classifier and the process repeats until a stopping condition is met (e.g., a target recall has been achieved).

However, the AL algorithms typically used in TAR naturally tend to generate *prior probability shift* (PPS), i.e., the fact that the prior probabilities $\Pr_L(y)$ on the labelled set (greatly) diverge from those on the unlabelled set $\Pr_U(y)$. This phenomenon can significantly compromise the performance of the classifier, potentially jeopardizing the quality of the review itself.

One of the most well-known procedures to adjust both the prior and posterior probabilities in PPS scenarios is the Saerens-Latinne-Decaestecker algorithm (SLD), an instance of expectation-maximization. In this thesis, we try to leverage the SLD algorithm to improve both the prior and posterior probabilities of a classifier trained in PPS scenarios: our goal is that of (i) improving the posteriors fed as input to a risk-minimization framework for e-discovery called MINECORE, and (ii) improving the prevalence estimates in order to halt the review process as soon as a target recall is achieved.

Regarding (i), we first give a thorough and comprehensive analysis of the SLD algorithm in simulated PPS scenarios, based on extensive experimental results. We then try to improve MINECORE performance, first by using an AL algorithm to generate the training set used by the framework, and later by applying SLD to its classifier posteriors.

Regarding (ii), we first give an analysis of the shortcomings of SLD, previously found in our MINECORE optimization work. We then attempt to solve these issues (albeit targeting prevalence estimation), proposing a novel modification to the SLD algorithm, called SAL_τ: the experimental results show that SAL_τ is able to stop the TAR process well in advance of the current state-of-the-art algorithms, while still reaching the given target recall.

Finally, we also study and explore the portability of machine learned models in TAR, as well as the employment of recent deep learning architectures in these scenarios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past two decades, machine and deep learning algorithms have become increasingly popular. They have not only pervaded and integrated with many disciplines, but also gained a primary role in several areas of society: machine learning models have been extensively and increasingly used in social networks, telephony, medicine, security and even art.

In the literature, machine learning algorithms are usually categorized based on the level of supervision a model can rely on. Two of these categories are supervised and unsupervised learning: in the former, we rely on the availability of labelled data (a training set) to train a model which later generalizes to unseen (and unlabelled) data; in the latter, the model is trained on data where labels, when needed, can be somehow automatically crafted. While it could be argued that much of the success of recent unsupervised models (such as Bert (Devlin et al., 2019) and GPT (Brown et al., 2020)) is due to the enormous amount of readily available data on the internet, in many real-world applications training datasets are not available for supervised algorithms. In-domain experts are usually required to undertake an annotation effort to label a (often limited) number of data items. The expert annotating the data sample is often called the "annotator" or "reviewer", and we refer to the process of assigning a label to an item either as "labelling", "reviewing" or "annotating". Clearly, having a human expert reviewing a data sample is an expensive operation, both in terms of time and costs: the number of data items that can be annotated are usually limited by either the availability of the reviewer, the time available, or the money one is willing to invest in the process (the *annotation budget*). Rather than annotating a uniform random sample of the data (which might be suboptimal), machine learning practitioners usually rely on a plethora of techniques which goes under the name of "Active Learning" (Dasgupta, Hsu, 2008; Huang et al., 2014; Lewis, Gale, 1994) (or AL for short): by active learning we mean an iterative process whose goal is that of prioritizing documents for review. More precisely, at every iteration the AL algorithm selects (usually via an automated classifier) a batch of documents for the reviewer to label: the newly labelled documents are fed back to the learning algorithm, and the process repeats until a stopping condition is met.

In several cases, the reviewing process is not aimed at simply building a training set for later use (e.g., labelling images to eventually build a classifier): the user commissioning or carrying out the review might be looking for some specific items (documents, in the case of textual data) in order to answer a specific research question. In these scenarios, the human-in-the-loop annotation workflow is usually referred to as Technology-Assisted Review (TAR) (Cormack et al., 2010; Grossman,

Cormack, 2011b; Kanoulas et al., 2019): TAR algorithms are mostly relevant in e-discovery in the legal domain, systematic reviews in empirical medicine, and online content moderation. The common ground in all the different TAR fields of application is that users need to review a large number of textual documents, in order to find those which are relevant to a specific information need (e.g., finding hate speech in user-generated online content). Moreover, users usually have a requirement to reach high or very high recall levels (i.e., the fraction of relevant documents annotated with respect to the total number of relevant documents).[1] Furthermore, as anticipated, users are limited by time or money: their goal is thus to maximize the review cost-effectiveness, that is, to make sure that review costs are kept to a minimum, while guaranteeing that most relevant documents have been found/annotated.

In these scenarios, reviews are usually conducted in two or more stages (multi-stage review), often requiring the combined effort of multiple review teams.

Multi-stage reviews: By multi-stage review we mean a labelling effort carried out in N stages, where each stage n may be conducted with: (i) a different information need, i.e., the concept of what is relevant in stage n might change in stage $n+1$; (ii) a different review team, possibly with different hourly (or per document) rate, making each stage more or less expensive with respect to the others; (iii) a different depth and width of the analysis of the data, e.g. reading (and labelling) many short abstracts vs reading the entirety of fewer articles/papers. These are all critical aspects that should be taken into consideration when estimating the costs of a multi-stage review (we will see how the recent literature approached this in Section 2.5.2).

Finally, we should notice that multi-stage reviews are still conducted with one common end-goal: in the fields of application we have mentioned, this might be the production of a systematic review relevant to a certain research question, or the production of documents relevant to a certain litigation matter in e-discovery. That is, there usually is a consequentiality between the different stages of the review: the purpose of stage n might be that of quickly filtering out irrelevant documents, so that stage $n+1$ can afford more time (and money) on fewer important items; likewise, documents may be passed on to stage $n+1$ only if they are deemed relevant in stage n.

1.1 Technology-Assisted Review applications

1.1.1 E-discovery

E-discovery is an important aspect of the civil litigation in many (but not only) common law countries: in e-discovery a large number of documents (which we call the pool P)[2] need to be reviewed in order to find all items "responsive" (i.e., relevant) to the object of the litigation. The documents labelled as responsive are "produced" by the producing party, and disclosed to the other party in the civil litigation. However, the producing party holds the right to keep some of these documents "hidden": this is only allowed if the "logged" documents are deemed to contain "privileged" information (e.g., intellectual property, sensitive data). Documents can finally be assigned to one out of three categories: **P**roduce, when a document is responsive and not privileged, **L**og, when a document is responsive and privileged, and **W**ithhold, when a document is not responsive. Following Oard et al. (2018)'s notation, we use c_P, c_L and c_W to indicate the produce, log and withhold

[1] It is not uncommon to have recall requirements of 100%, transforming the task in a "total recall" one.

[2] In Table 1.1 we show the mathematical notation that we use throughout this thesis.

Notation	Meaning
$p(y)$	prevalence of class y
$\Pr(y\|x)$	posterior prob. of y given x
$\Pr(\oplus\|x)$	posterior prob. of positive class given x
$\Pr(\ominus\|x)$	posterior prob. of negative class given x
$\Pr(y)$	prior probability of y
$y(x)$	true class of a document/item x
Y	the set of possible labels
L	training (labelled) set
U	test (unlabelled) set
S	seed set (initial training for active learning)
P	documents pool (active learning document pool)
D	a set of documents
D_r	the relevant subset of D (also used with L_r or U_r)
R	target recall
$\hat{R}$	estimated recall
R_s	recall at stopping (for a TAR process)
ϕ	a classifier
B	set of documents in a batch
b	batch size
β_i	binned set of items (i.e., the ith bin)
c_P, c_L, c_W	Privilege, Log and Withhold class (e-discovery/MINECORE)
ϕ_r	classifier for responsiveness (or relevant items)
ϕ_p	classifier for privilege
λ_p^a, λ_r^a	annotation costs for privilege/responsiveness
λ_{ij}^m	misclassification costs of assigning c_i to a document $d \in c_j$ ($c_i, c_j \in \{c_P, c_L, c_W\}$)
$\mathcal{Q}$	set of tasks (e.g. for systematic reviews)
q	single task
$R(q)$	target recall for task q
Ψ	Number of documents necessary to reach a target recall

Table 1.1: Notational conventions used throughout the thesis.

classes respectively. Making different misclassification errors usually brings about different costs for the producing party, based on the severity of the error committed: for instance, producing (i.e., assigning class c_P) a document which is responsive and yet contains privileged information (i.e., should have been logged c_L) is usually a more serious mistake than producing a document which is not responsive (i.e., belonging to class c_W). It is worth noticing, however, that there are only few works which really take e-discovery costs into consideration (to the best of our knowledge, Oard et al. (2018); Yang et al. (2021b); see also Section 2.5.2).

Usually, the review happens in two stages: (i) documents are first reviewed by responsiveness (i.e., relevancy) by a team of junior reviewers; (ii) the documents judged as responsive are then passed on to a second team of senior reviewers (with an hourly rate which can be several times higher than the junior team's), who mainly re-review the documents by privilege. As it can be inferred, annotating documents by privilege is usually a much more costly and delicate operation than annotating by responsiveness (see e.g., Oard et al. (2018); Yang et al. (2021b)).

1.1.2 Production of systematic reviews

In empirical medicine, a systematic review discusses (ideally) all medical literature relevant to a given research question. The production of a systematic review is usually carried out by one or more physicians, over the course of (possibly) years and can cost hundreds of thousands of U.S. dollars (Michelson, Reuter, 2019; Shemilt et al., 2016). A systematic review usually collects a large set of documents by issuing a boolean query on a (medical literature) search engine, such as PubMed.[3] Then, similarly to e-discovery, systematic reviews are usually conducted in two stages: (i) a first one, where the abstracts of the documents are reviewed in order to determine their probable relevance and (ii) a second one, where documents which passed the first phase are reviewed in their entirety. One critical aspect in the production of systematic reviews is the so-called "risk of bias" (ROB) (Viswanathan et al., 2017; Whiting et al., 2016), i.e., the fact that missing many relevant documents may jeopardize the quality of the review itself, bringing to (or reporting) biased results and conclusions: for instance, if the identification and selection of studies are not properly formalized, the document pool might be too homogenous, possibly causing the review to miss other relevant key works in the literature. Notice, however, that the risk of bias is extremely difficult to formalize and/or quantify: as such, to the best of our knowledge, there is no current work in the information retrieval literature that has taken ROB into consideration.

That said, the production of systematic reviews has recently attracted the interest of the IR community (Callaghan, Müller-Hansen, 2020; Lease et al., 2016; O'Mara-Eves et al., 2015; Wang et al., 2022), which has focused on several aspects of the process, from improving the query formulation issued to search engines, to finding the optimal stopping criterion, reducing the annotation costs (and the time spent on a systematic review). More in details, given a research question, a systematic review is conducted in the following way:

1. The reviewer prepares a query which is issued on one or more search engines (for medical literature);
2. An initial pool of documents P (based on the search engine ranking) is retrieved;
3. The reviewer reads P abstracts. Only the subset of documents L_r which are deemed relevant is kept;

[3] `https://pubmed.ncbi.nlm.nih.gov/`

4. The L_r documents are now read and reviewed in their entirety. Again, we filter out all the non-relevant documents;

5. Finally, the remaining documents will form the set of documents included in the systematic review.

TAR algorithms are usually involved in step 3 and 4 (but the literature has also targeted steps 1 and 2). More specifically, the TAR process is structured in the following manner:[4]

1. An initial set of labelled documents S is provided. This set should contain at least one positive (and it often consists exclusively of this single positive) document;

2. A machine learning algorithm is trained on S and outputs scores (or probabilities) on the remaining $P \setminus S$ documents;

3. An active learning policy (e.g. Continuous Active Learning, see Section 2.1.2 and Section 2.3.1) chooses, based on these scores, which and how many documents to show to the reviewer;

4. The reviewer reads the selected documents abstracts, deciding whether they are relevant or not;

5. The set S (which we may call now the labelled set L) is augmented with the new labelled documents and the process starts again, until a review budget is exhausted or a stopping rule condition is met.

We give a graphical representation of the process in Figure 1.1.

1.2 What this thesis is about

The main focus of this thesis is on improving *a posteriori* and *a priori* (posteriors and priors) probabilities estimates in TAR workflows. The motivations behind this work are: (i) having higher quality posteriors is particularly important when our decisions are entirely based on these estimates, as it is for the risk-minimization framework for e-discovery called MINECORE (Oard et al. (2018), which we will see in Section 2.3.3); (ii) having better priors (prevalence) estimates trivially means that we can better quantify the achieved recall, possibly stopping the annotation process as soon as we reach a target recall R.

More in details, this thesis studies, analyzes and proposes improvements to the Saerens-Latinne-Decaestecker algorithm (SLD, Saerens et al. (2002)), applied in TAR contexts: SLD is an instance of expectation maximization whose goal is that of adjusting prior and posterior probabilities in *prior probability shift* scenarios (PPS, see Section 2.1.1 and 2.4). Our decision to focus particularly on this algorithm originates from our master thesis work (Molinari, 2019b), of which this PhD thesis is a follow-up. There, we highlighted how, despite being seemingly suited for such a task, the SLD algorithm failed to deliver consistent results when applied to the input posterior probabilities of the MINECORE framework, actually deteriorating its performance.

The thesis is structured as follows: in Chapter 2 we give an overview of the TAR state of the art for one-phase and two-phase workflows, as well as a detailed explanation of the SLD algorithm;

[4]Notice that this process is substantially similar to the one for e-discovery, where we review documents for responsiveness first and privilege afterwards. See also Lease et al. (2016).

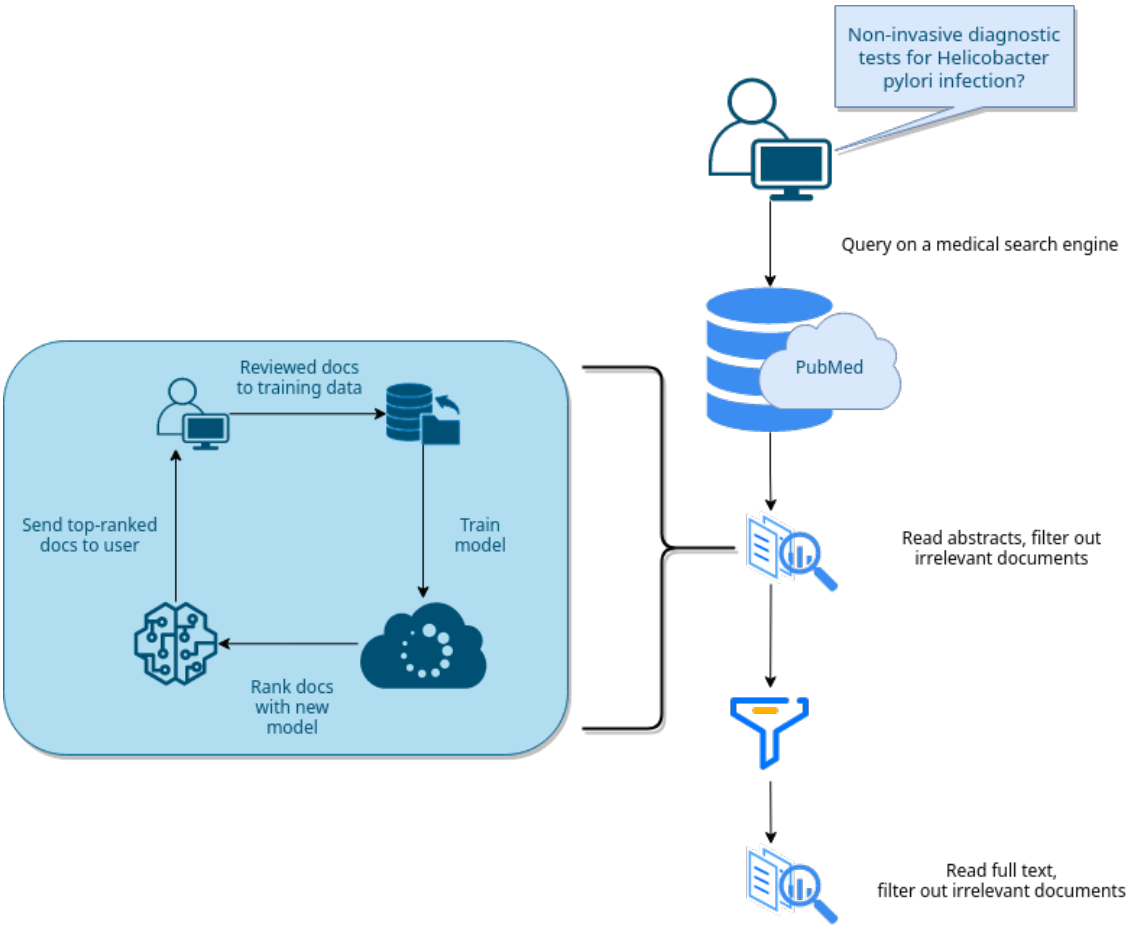


Figure 1.1: Diagram of a technology-assisted systematic review process.

in Chapter 3 we give an in-depth analysis of the SLD algorithms for posterior improvements under prior probability shift. Our analysis shows that the algorithm is capable of improving the classifier probabilities for binary classification tasks. In Chapter 4, we analyze and evaluate which active learning procedure is best in order to minimize MINECORE costs, and whether SLD can bring improvements in this regard. Our results show that active learning can indeed bring to a better training set for the MINECORE framework; regarding the SLD algorithm, we show that its usage in active learning scenario brings to a consistent deterioration of the classifier probabilities. In Chapter 5, we propose our own adjustment to the SLD algorithm, a new method called SAL_τ, which enables the usage of SLD in active learning procedures: we show that SAL_τ is able to stop the TAR process well before the other baselines, while still reaching the target recall. Furthermore, in Chapter 6 we study whether transfer learning can be successful in TAR for systematic reviews: we analyze transformer-like architectures, their capability to do zero-shot classification and ranking in this context, and whether they can be continuously trained in an active learning procedure. We show that, while models such as BioBERT can have interesting zero-shot performance, the difficulties of continuously train these models in active learning scenarios make their adoption challenging for TAR tasks. Finally, in Chapter 7 we conclude this work.

1.3 List of publications

We list below the papers we published throughout the PhD, i.e., since late 2019 to early 2023:

- Esuli, Andrea, Alessio Molinari, and Fabrizio Sebastiani. "A critical reassessment of the Saerens-Latinne-Decaestecker algorithm for posterior probability adjustment." ACM Transactions on Information Systems (TOIS) 39.2 (2021): 1-34 (Esuli et al., 2021);

- Molinari, Alessio, Andrea Esuli, and Fabrizio Sebastiani. "Active learning and the Saerens-Latinne-Decaestecker algorithm: an evaluation." Proceedings of the 2nd Joint Conference of the Information Retrieval Communities in Europe (CIRCLE 2022), Samatan, France. 2022 (Esuli et al., 2022);

- Molinari, Alessio, Andrea Esuli, and Fabrizio Sebastiani. "Improved Risk Minimization Algorithms for Technology-Assisted Review." Intelligent Systems with Applications (2023): 200209 (Molinari et al., 2023);

- Molinari, Alessio, and Andrea Esuli. "SAL_τ: Efficiently Stopping TAR by Improving Priors Estimates." Submitted to Data Mining and Knowledge Discovery, currently under review;

- Molinari, Alessio, and Evangelos Kanoulas. "Transferring knowledge between topics in systematic reviews." Intelligent Systems with Applications 16 (2022): 200150 (Molinari, Kanoulas, 2022).

Chapter 2

Background

Technology-Assisted Review algorithms and frameworks mostly find their application in three different fields: e-discovery, in the legal domain, systematic reviews in empirical medicine, and finally online content moderation. IR literature has mostly focused on the e-discovery domain, and recently on systematic reviews, proposing a plethora of methods that cover the whole pipeline, from the query formulation to the different annotation stages usually present in TAR tasks. In this chapter, we will give an overview of the background and the previous works in TAR application context, as well as of key algorithms and phenomena, whose understanding is fundamental for the following chapters. Regarding the TAR application contexts, we will exclusively focus on e-discovery and systematic reviews. We will also explain the difference between the so-called one-phase and two-phase TAR algorithms/frameworks. Finally, we will give an overview of the different evaluation metrics proposed in the literature over the years, especially focusing on those we employ in Chapters 3, 4, 5 and 6.

2.1 Preliminaries

2.1.1 Dataset shift

Dataset shift is a key phenomenon in TAR, due to TAR algorithms/frameworks heavily relying on some active learning policy to label the documents pool P (see next section). As a matter of fact, AL procedures may generate several types of shift, such as prior probability shift (PPS) or covariate shift. In order to distinguish different types of dataset shift, Moreno-Torres et al. (2012) distinguish (along with Fawcett, Flach (2005)) between "$X \to Y$ problems" and "$Y \to X$ problems".

Problems of type $X \to Y$ are ones in which it is the values of the features in $\mathbf{x}$ that stochastically determine the class $y = t(\mathbf{x})$ to which $\mathbf{x}$ belongs. An example of a $X \to Y$ learning problem is weather forecasting, since it is a number of climatic conditions (for instance, pressure, temperature, humidity, etc., that can be represented in a feature vector $\mathbf{x}$) that determine whether it is going to snow or not (a fact that can be represented by a binary dependent variable y). In these cases, it is useful to write the joint distribution $\Pr(\mathbf{x}, y)$ as

$$\Pr(\mathbf{x}, y) = \Pr(y|\mathbf{x}) \Pr(\mathbf{x}) \tag{2.1}$$

Equation 2.1 suggests that there are two phenomena (or, of course, a combination of both) that can cause $\Pr(y)$ to vary across L and U, i.e.,

1. *Covariate shift*, defined as the case in which $\Pr_L(y|\mathbf{x}) = \Pr_U(y|\mathbf{x})$ and $\Pr_L(\mathbf{x}) \neq \Pr_U(\mathbf{x})$;
2. *Concept shift*, defined as the case in which $\Pr_L(y|\mathbf{x}) \neq \Pr_U(y|\mathbf{x})$ and $\Pr_L(\mathbf{x}) = \Pr_U(\mathbf{x})$.

For instance, in the example above, if the distribution of climatic conditions change, the probability that it is going to snow changes too; this is a case of covariate shift. Instead, if the causal relationship between climatic conditions and snowing were to change (an admittedly unlikely case), this would be a case of concept shift.

Problems of type $Y \rightarrow X$ are instead ones in which the class $y = t(\mathbf{x})$ to which document $\mathbf{x}$ belongs stochastically determines the values of the features in vector $\mathbf{x}$. An example of a $Y \rightarrow X$ learning problem is authorship attribution, i.e., the task of determining the author (from a set of $|Y|$ candidate authors) of a text of unknown or disputed paternity (Koppel et al., 2009). This task is usually carried out by using as features a number of "stylistic" traits that tend to characterize an author's writing style. Authorship attribution is a $Y \rightarrow X$ problem, since it is the fact that a certain text is, say, Shakespeare's, that causes it to have certain stylistic characteristics, and not the other way around. In these cases, the joint distribution $\Pr(\mathbf{x}, y)$ can be usefully written as

$$\Pr(\mathbf{x}, y) = \Pr(\mathbf{x}|y)\Pr(y) \tag{2.2}$$

Here, $\Pr(y)$ can vary for independent reasons (since y is a cause, and not an effect), a phenomenon which is usually called *prior probability shift*. For instance, in Stratford-upon-Avon's municipal library there might proportionally be more books by Shakespeare than in any other municipal library.[1]

Finally, it is worth noticing that Moreno-Torres et al. (2012, §6.1) indicate *sample selection bias* as one of the key reasons behind PPS. When using active learning techniques, we intentionally introduce a sample selection bias, thus generating PPS. However, active learning also causes another phenomenon, which some of the literature (e.g., Dasgupta, Hsu (2008); Krishnan et al. (2021)) refers to by the name of *sampling bias* (we will also use this name in Chapters 4 and 5). By *sampling bias*, we mean the phenomenon whereby not only our prior probabilities diverge ($\Pr_L(y) \neq \Pr_U(y)$), but so does the distribution of the covariates conditioned on the class label, i.e., $\Pr_L(x|y) \neq \Pr_U(x|y)$.[2] In other words, when using active learning, we draw similar and uninformative examples in our training set, possibly completely ignoring entire clusters of items (we will analyze the effects of this phenomenon on a classifier in Chapters 4 and 5).

2.1.2 Optimizing documents revision: active learning in TAR

Having a human expert reviewing the whole set of documents P can be an extremely expensive operation, sometimes infeasible when P consists of more than a few thousands of documents. TAR

[1] Notice, however, that it is not always easy to characterize with certainty a given problem as being of type $X \rightarrow Y$ or of type $Y \rightarrow X$; sometimes this question looks a bit akin to wondering which of chicken and egg came first. As a result, different types of dataset shift (covariate shift, concept shift, prior probability shift) that concur in causing dataset shift may be at play at the same time.

[2] Notice that this is not properly a *concept shift*, since the same item x would still be assigned to the same class y, regardless of whether $x \in L$ or $x \in U$. Moreover, since we also have PPS, it does not hold that $\Pr_L(y) = \Pr_U(y)$. Moreno-Torres et al. (2012, §4.4) acknowledge the existence of this shift, reporting that it is a rare and very hard to solve situation, which has hardly been addressed in the literature.

practitioners have thus resolved to use an active learning procedure, i.e. an iterative process where an algorithm prioritizes, at each iteration, a batch B of documents for the reviewer to annotate, and often attempts to stop the review once a target recall R is reached. In both the e-discovery and the systematic reviews fields, Active Learning via Relevance Sampling (ALvRS) (Lewis, Gale, 1994; Rocchio, 1971) has established as the main active learning procedure used; TAR practitioners actually use a variation of ALvRS, called Continuous Active Learning (CAL), developed and adapted for TAR by Cormack and Grossman (Cormack, Grossman, 2014, 2015a, 2016b): in CAL based strategies, the review is usually carried on until deemed complete. Other active learning techniques have also been used or proposed over the years, such as Active Learning via Uncertainty Sampling (ALvUS, Lewis, Catlett (1994); Lewis, Gale (1994)), or more recently Callaghan, Müller-Hansen (2020)'s procedure which alternates between ALvRS and random sampling (see Section 2.3.4), or Li, Kanoulas (2020)'s autostop framework (see Section 2.3.5); that said, CAL has without doubt become the standard AL algorithm for TAR.

We give an overview of ALvRS and ALvUS, as they are the key AL procedures used in TAR (or form the basis for other AL policies, such as CAL).

Active Learning via Relevance Sampling (ALvRS): ALvRS is an interactive process which, given a data pool of unlabelled documents P, asks the reviewer to annotate an initial "seed" set of documents $S \subset P$, uses S as the training set L to train a binary classifier ϕ, and uses ϕ to rank the documents in $(P \setminus L)$ in decreasing order of their posterior probability of relevance $\Pr(\oplus|\mathbf{x})$. Then, the reviewer is asked to annotate the b documents for which $\Pr(\oplus|\mathbf{x})$ is highest (with b the *batch size*), which, once annotated, are added to the training set L. Finally, we retrain our classifier on the new training set and repeat the process, until a predefined number of documents (the annotation budget) have been reviewed. ALvRS is most-effective and has been mostly used when we are interested in finding all the items relevant to a given information need, as quickly as possible.

Active Learning via Uncertainty Sampling (ALvUS): The ALvUS policy is a variation of ALvRS, where we review the documents not in decreasing order of $\Pr(\oplus|\mathbf{x})$ but in increasing order of $|\Pr(\oplus|\mathbf{x}) - 0.5|$, i.e., we top-rank the documents which the classifier is most uncertain about. ALvUS can be useful when we want to build a high-quality training set to later train a machine learning model on it.

We illustrate an AL workflow with a stopping condition in Algorithm 1.

2.1.3 The *Rand* policy: A pseudo-oracle sampling-bias-free policy

As we will later see in Chapter 4 and 5, ALvRS and ALvUS are affected by what is called *sampling bias* (Dasgupta, Hsu, 2008; Krishnan et al., 2021), i.e. the fact that, due to the document selection policy and the initial seed S, the two active learning policies tend to draw very similar (and thus uninformative) documents from the pool P (see also Section 2.1.1): this will in turn cause the labelled set L to be anything but a representative set of the underlying data distribution, diverging both from U and P.[3] Sampling bias can cause several issues, such as bringing to an overly confident classifier, which we will analyze in Chapter 4 and, more in details, in Chapter 5. In order to better understand and study the effects of this bias, we introduced in Esuli et al. (2022) a pseudo-oracle

[3]Notice, for completeness, that U also diverges from P.

Algorithm 1: Schema of an AL process.

Input : Pool of documents P to be reviewed; Batch size $b = 100$; budget $t = |P|$; target recall R; AL policy *pol*;

$i \leftarrow 0$;
$S \leftarrow$ `initial_seed()` ;
$L \leftarrow S$;
$U \leftarrow P \setminus L$;
do
  $i \leftarrow i + 1$;
  $\phi_i \leftarrow$ `train_clf(`L`)`;
  $B_i \leftarrow$ `select_via_pol(`pol, ϕ_i, U, b`)` ;
  $L \leftarrow L \cup B_i$;
  $U \leftarrow P \setminus L$;
while `should_not_stop(`ϕ_i, B_i, L, U, t, R`)`;

policy called *Rand*, whose goal is that of generating a random sample of P, while keeping the labelled and unlabelled set prevalences as generated by ALvRS or ALvUS. More in details, the *Rand* policy observes the prevalence of labels in the L set and in the U set generated by either ALvRS or ALvUS, and draws two random samples L^{Rand} and U^{Rand}, of size $|L|$ and $|U|$, where the prevalences $p_{L^{Rand}}(y)$ and $p_{U^{Rand}}(y)$ are identical to $p_L(y)$ and $p_U(y)$; we call *Rand*(RS) and *Rand*(US) the controlled random samples generated from ALvRS and ALvUS respectively.

The *Rand* policy should allow us to understand whether some specific classifier behaviours are due to the document selection policy or rather to the *prior probability shift* (PPS), which we naturally generate with active learning techniques. More specifically, ALvRS and ALvUS generate a type of dataset shift where $\Pr_L(y) \neq \Pr_U(y)$ and $\Pr_L(x|y) \neq \Pr_U(x|y)$. With the *Rand* policy, instead, we only generate PPS, i.e., $\Pr_L(y) \neq \Pr_U(y)$ but $\Pr_L(x|y) = \Pr_U(x|y)$.

2.2 Using supervised machine learning

The human-in-the-loop, machine learning aided review, has been used in civil litigation since the mid-2000s (Baron et al., 2007). While early works were mainly focusing on the query formulation (replacing the classical boolean query with text classifiers), a pervasive usage of machine learning across the whole TAR pipeline has been advised (and evaluated as superior) by Cormack and Grossman (Cormack, Grossman, 2014; Cormack, Mojdeh, 2009) since 2009.

TAR workflows are typically characterized by some active learning strategy, which usually also comprises a stopping methodology, i.e., the TAR framework is also responsible to stop the review process once a target recall R (known beforehand) is achieved. TAR algorithms are usually divided into one-phase and two-phase workflows: despite part of the literature has promoted the usage of one-phase workflows as superior (Cormack, Grossman, 2014, 2016b, 2020; Cormack, Mojdeh, 2009), or defined two-phase workflows as "TAR 1.0" (Tredennick, 2015), we argue together with Yang et al. (2021b) that the superiority of one of the two types of workflows is dependent on the type and costs of the review (see Section 2.5.2). Moreover, the US Department of Justice Antitrust Division suggests a two-phase workflow, where the responsiveness decisions in phase two are taken exclusively by an automated classifier (Yang et al., 2021b); Keeling et al. (2020) have reaffirmed the need for human supervision, but their argument was later rejected by Grossman, Cormack (2020).

2.2.1 Assuming infallible reviewers: a common simplifying assumption, and a limitation of this thesis

In this thesis, and in many of the works we present in this Chapter, the reviewers emulated in the in vitro experiments are assumed to be infallible: that is, it is assumed that the human reviewer always assigns the correct label to each document. Of course, in reality this assumption cannot hold, and human reviews are subject to many mistakes.

Concerning this thesis, this simplifying assumption is made by the MINECORE framework, which is the framework we study and experiment with in Chapter 4: as a result, we also make this assumption in that chapter. Furthermore, we also rely on infallible reviewers in our works presented in Chapter 5 and 6, making, in fact, this assumption an underlying limitation of our thesis.

That said, however, the infallible reviewer assumption is a common one, made by many of the works we present here, such as Callaghan, Müller-Hansen (2020); Cormack, Grossman (2016a); Li, Kanoulas (2020); Oard et al. (2018); Yang et al. (2021a). Nonetheless, Cormack and Grossman have been raising awareness on this since 2011 (Grossman, Cormack, 2011a), as well as presenting one of the few works, to the best of our knowledge, which deals with the infallibility of reviewers (Cormack, Grossman, 2017).

2.3 One-phase and two-phase TAR workflows

By one-phase TAR we mean a workflow where we employ an active learning strategy in order to find the highest number of relevant documents in the least amount of time possible, hopefully stopping the review once a target recall R is reached. The classifier trained in the active learning process is discarded and not re-used for subsequent tasks:[4] our focus is then not on creating/training the best classifier possible, but rather on finding the best possible strategy to achieve the target recall as soon as possible.

On the other hand, in two-phase TAR workflows we first collect a training set (first phase) via an active learning process (or via random sampling) from the pool P. This labelled data will then be used (second phase) to train a classifier ϕ, which should help reviewers finding relevant documents in the pool. The review team labelling documents in the first phase is often a different team than the one in the second phase (also, with different hourly rates).

Yang et al. (2021b) argued that when the review costs are uniform (i.e., positive and negative documents are equally expensive, regardless of the review stage) a one-phase TAR workflow is optimal. Two-phase workflows might instead achieve better results (and lower costs) when reviewing documents (and especially positives) is more expensive in the first phase than in the second phase. Recent and notable algorithms which work inside a one-phase workflow are the Knee and the Budget methods (Cormack, Grossman, 2016a), the Callaghan, Müller-Hansen (2020)'s (CMH) method, Li, Kanoulas (2020)'s autostop method and Yang et al. (2021a)'s QuantCI. Despite literature has focused much more on one-phase workflows, an important and recent framework for two-phase TAR, that we will see in Section 2.3.3, is Oard et al. (2018)'s MINECORE.

[4] As a matter of fact, in e-discovery review by privilege is usually not carried out via active learning. One of the main exceptions is MINECORE (Oard et al., 2018).

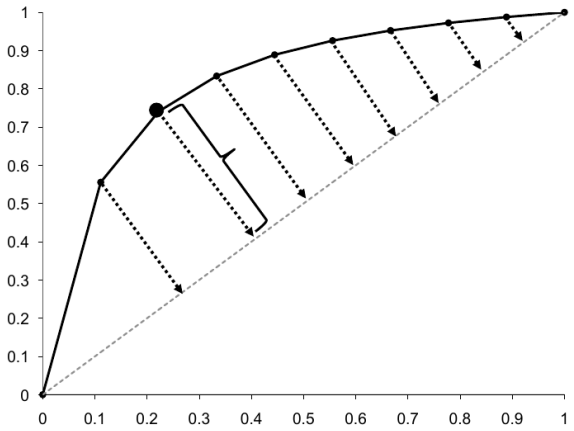


Figure 2.1: The knee detection method by Satopaa et al. (2011), as reported by Cormack, Grossman (2016a).

2.3.1 The Continuous Active Learning (CAL) strategy

The standard active learning strategy used in most one-phase TAR workflows is based on the so-called Relevance Feedback (Rocchio, 1971) and on ALvRS. This strategy has been further elaborated and adapted to TAR by Cormack and Grossman, and goes under the name of Continuous Active Learning (CAL) (Cormack, Grossman, 2015a, 2016b). The Knee, the Budget and the QuantCI methods all work inside a CAL process. Continuous Active Learning (CAL) characteristics are:

- an initial seed set S consisting of a single positive instance (which can often be the query text), and a number n of randomly sampled instances, temporarily labelled as negative;
- the batch size b can be incrementally defined as $b = b + \lceil \frac{b}{m} \rceil$, where m is a constant value (e.g., $m = 10$).

Nonetheless, other sampling strategies have been proposed over the years: in the recent literature, the previously mentioned CMH (Callaghan, Müller-Hansen, 2020), which combines CAL with random sampling, and Li and Kanoulas' autostop framework (Li, Kanoulas, 2020).

2.3.2 The Knee and the Budget methods

The Knee method was first proposed by Cormack, Grossman (2016a). The method is based on a gain curve for a one-phase TAR workflow, i.e., a plot of how the number of positive documents increases as more documents are reviewed during the AL process. The method, based on Satopaa et al. (2011), empirically finds "knees" in the plot, ideally stopping the process when the effort of continuing to review documents is not supported by the retrieval of a sufficient amount of positive documents. We reproduce Cormack, Grossman (2016a)'s knee plot in Figure 2.1. The Budget method (Cormack, Grossman, 2016a) is a heuristic variant of the knee method, where the process is stopped no earlier than when at least 70% of the document collection has been reviewed. This follows the observation that, if we were to review by random sampling, we would expect to achieve a recall of 0.7 when reviewing 70% of the collection; by using an AL technique, we expect the

recall to be much higher. After the 70% threshold has been reached, the Budget method stops the review if a knee test passes (detailed in Cormack, Grossman (2016a); Yang et al. (2021a)), and if the number of relevant items found $|L_r|$ is somewhat large, i.e.: $|L| \geq 10\ \frac{P}{|L_r|}$. Finally, notice that both methods do not allow users to specify a target recall.

2.3.3 The MINECORE framework

MINECORE (Oard et al., 2018) is a recently proposed two-phase decision-theoretic algorithm for technology-assisted review that attempts to minimize the expected cost (i.e., the risk) of review for responsiveness and privilege in e-discovery. MINECORE is a notable exception in the TAR world for two main reasons:

1. MINECORE is, to the best of our knowledge, the only TAR framework which treats e-discovery costs as first-class citizens: that is, its risk-minimization approach, as well as its stopping condition, are entirely based on estimates of the expected costs of review. More recently, e-discovery costs have also been addressed by Yang et al. (2021b), who formalized the IC metric (see Section 2.5.2).

2. MINECORE is also the only e-discovery framework which jointly addresses review by responsiveness and privilege (as a matter of fact, most TAR algorithms only deal with the responsiveness stage).

Given a set P (the *pool*) of documents that must each be assigned to a class in $\{c_P, c_L, c_W\}$ (where the meaning of these three classes is as discussed in the introduction), the goal of MINECORE is to determine, for each document $x \in P$, whether manually reviewing $\mathbf{x}$ for responsiveness and/or privilege is expected to be cost-effective or not. This determination is based

1. on the ("posterior") probabilities of class membership (written as $\Pr(c_r|\mathbf{x})$ and $\Pr(c_p|\mathbf{x})$) returned by automated classifiers ϕ_r (that classifies documents by responsiveness) and ϕ_p (that classifies documents by privilege);

2. on the costs of manually reviewing a document for responsiveness (λ_r^a) or for privilege (λ_p^a), where superscript a stands for "annotation";

3. on the costs λ_{ij}^m incurred when assigning class c_i to a document which should be assigned class c_j, where $c_i, c_j \in \{c_P, c_L, c_W\}$ and superscript m stands for "misclassification".

Concerning Bullet 2, the fact that costs λ_r^a and λ_p^a are different is due to the fact that, as previously mentioned, annotation by responsiveness can usually be assigned to junior personnel, while annotation by privilege requires more subtle expertise, and is usually entrusted to senior lawyers. Concerning Bullet 3, the fact that costs λ_{ij}^m are different for different $c_i, c_j \in \{c_P, c_L, c_W\}$ is due to the fact that, in e-discovery, not all misclassifications are equally serious; for instance, inadvertently disclosing a privileged document to the other party is typically a very serious mistake, while inadvertently disclosing a nonresponsive nonprivileged document is usually a less serious one.

We assume that our pool P is partitioned into a set L of labelled (i.e., manually reviewed for both responsiveness and privilege) documents and a set U of unlabelled documents.

The MINECORE workflow is articulated in three steps, which we summarize below.

In **Step 1** we train from L the two classifiers ϕ_r and ϕ_p described in Bullet 1 above, and use them to generate, for each document $\mathbf{x} \in U$, the two posteriors $\Pr(c_r|\mathbf{x})$ and $\Pr(c_p|\mathbf{x})$ mentioned in Bullet

1. We can reasonably assume c_r and c_p to be stochastically independent, which implies that we may assume $\Pr(c_P|\mathbf{x}) = \Pr(c_r|\mathbf{x})\Pr(\overline{c}_p|\mathbf{x})$, $\Pr(c_L|\mathbf{x}) = \Pr(c_r|\mathbf{x})\Pr(c_p|\mathbf{x})$, and $\Pr(c_W|\mathbf{x}) = \Pr(\overline{c}_r|\mathbf{x})$. MINECORE takes a *risk minimization* approach, i.e., it assigns each document $\mathbf{x} \in U$ to the class

$$\begin{aligned}\phi(\mathbf{x}) &= \arg \min_{c_i \in \{c_P, c_L, c_W\}} R(\mathbf{x}, c_i) \\ &= \arg \min_{c_i \in \{c_P, c_L, c_W\}} \sum_{j \in \{P,L,W\}} \lambda_{ij}^m \Pr(c_j|\mathbf{x})\end{aligned} \tag{2.3}$$

where $R(\mathbf{x}, c_i)$ is the *risk* associated with assigning $\mathbf{x}$ to class $c_i \in \{c_P, c_L, c_W\}$. In other words, MINECORE assigns to each document $\mathbf{x}$ the class that brings about the minimum misclassification risk, thus avoiding assignments which would bring about a high expected misclassification cost. The function for measuring the global misclassification cost (that derives from an assignment of labels in $\{c_P, c_L, c_W\}$ to the documents in U) is thus

$$\begin{aligned}K^m(U) &= \sum_{\mathbf{x} \in U} K^m(\mathbf{x}) \\ &= \sum_{\mathbf{x} \in U} \sum_{i,j \in \{P,L,W\}} \lambda_{ij}^m \cdot \mathbf{1}[\mathbf{x} \in U_{ij}]\end{aligned} \tag{2.4}$$

where $\mathbf{1}[\cdot]$ is the characteristic function that returns 1 if its argument is true and 0 if it is false, and U_{ij} is the set of documents $\mathbf{x} \in U$ that are assigned to c_i and whose true class (which we denote by $t(\mathbf{x})$) is c_j. Note that $K^m(\mathbf{x})$ is the misclassification cost brought about by document $\mathbf{x}$, and that the global misclassification cost is simply the sum of document-wise misclassification costs, i.e., MINECORE assumes that misclassification costs are *linear*.[5]

Step 2 is based on the consideration that, if τ_r documents are manually reviewed for responsiveness and τ_p documents are manually reviewed for privilege, the overall cost $K^o(U)$ of the entire process is

$$\begin{aligned}K^o(U) &= K^m(U) + K^a(U) \\ &= \sum_{\mathbf{x} \in U} K^m(\mathbf{x}) + \sum_{\mathbf{x} \in U} K^a(\mathbf{x}) \\ &= \sum_{\mathbf{x} \in U} K^m(\mathbf{x}) + \lambda_r^a \tau_r + \lambda_p^a \tau_p\end{aligned} \tag{2.5}$$

where by $K^a(U)$ we indicate the global annotation cost. Similarly to the above, note that $K^a(\mathbf{x})$ is the annotation cost brought about by document $\mathbf{x}$, and that the global annotation cost is simply the sum of document-wise annotation costs, i.e., MINECORE is based on linear annotation costs. Since both misclassification costs and annotation costs are linear, overall costs are also linear, i.e.,

$$K^o(U) = \sum_{\mathbf{x} \in U} K^o(\mathbf{x}) \tag{2.6}$$

[5]This simplifying assumption is probably a limitation, since in many e-discovery contexts a few mistakes of a certain kind might be without any consequence while more mistakes of the same kind might give rise to major negative consequences, with the relationship between number of mistakes of this type and consequences of these mistakes not being linear. However, dealing with this problem is not within the scope of this thesis, and is a potential topic of future research.

If document $\mathbf{x} \in U$ is reviewed for, say, responsiveness, this has the effect of removing (assuming infallible reviewers) any uncertainty about whether $\mathbf{x}$ is responsive or not. In other words, if by subscript $(n) \in \{(1), (2), (3)\}$ we indicate the value of a given quantity after Step n has been carried out (so that, e.g., $\phi_{(2)}$ and $\Pr_{(2)}(y|\mathbf{x})$ will indicate the classifier ϕ and the posterior $\Pr(y|\mathbf{x})$ resulting from the completion of Step 2), reviewing $\mathbf{x}$ for responsiveness during Step 2 means that $\Pr_{(2)}(c_r|\mathbf{x})$ will be either 0 or 1. As a result, if during Step 2 document $\mathbf{x} \in U$ is reviewed for responsiveness, it will in general hold that $\Pr_{(1)}(c_r|\mathbf{x}) \neq \Pr_{(2)}(c_r|\mathbf{x})$, $\phi_{(1)}(\mathbf{x}) \neq \phi_{(2)}(\mathbf{x})$ (where ϕ is the cost-sensitive classifier of Equation 2.3), and $K^m_{(1)}(\mathbf{x}) \geq K^m_{(2)}(\mathbf{x})$. Since reviewing $\mathbf{x}$ for responsiveness brings about an annotation cost λ^a_r, it is worthwhile to annotate $\mathbf{x}$ only if, as a result of the annotation, $K^o_{(2)}(\mathbf{x}) \leq K^o_{(1)}(\mathbf{x})$, i.e., $K^m_{(2)}(\mathbf{x}) + \lambda^a_r \leq K^m_{(1)}(\mathbf{x})$; in other words, the additional annotation cost λ^a_r must be offset by a reduction $(K^m_{(1)}(\mathbf{x}) - K^m_{(2)}(\mathbf{x}))$ in misclassification cost of greater or equal magnitude. Of course, computing precisely whether annotating $\mathbf{x}$ by responsiveness is going to bring about such a reduction is not possible, because at the time of deciding whether $\mathbf{x}$ should be annotated by responsiveness or not we do not know the value of $y_r(\mathbf{x})$ (a binary variable that indicates whether the reviewer will annotate $\mathbf{x}$ as responsive or not), and we do not know the true label $t(\mathbf{x})$ of $\mathbf{x}$. However, it is possible to compute an expectation of this reduction over the $y_r(\mathbf{x})$ and $t(\mathbf{x})$ variables; when this expected value exceeds λ^a_r, MINECORE decides that $\mathbf{x}$ should be annotated by responsiveness. Since MINECORE computes the expectation of this reduction for all documents in U, this means that MINECORE

- can *rank* the documents in U (where the top-ranked document is the one with the highest expected reduction), so that by proceeding from the top downwards the annotator reviews first the documents whose annotation brings about the highest expected benefit;
- provides the annotator with a *stopping criterion*, which coincides with the position in the ranked list when the reduction $(K^m_{(1)}(\mathbf{x}) - K^m_{(2)}(\mathbf{x}))$ (actually: its expected value) has become smaller than λ^a_r.

We refer the reader to (Oard et al., 2018, §3) for details on how the above expected value is computed, and for a full mathematical specification of MINECORE.

Step 3 is essentially identical to Step 2, the only difference being that, while Step 2 focuses on responsiveness, Step 3 focuses on privilege and uses the posteriors $\Pr_{(2)}(c_r|\mathbf{x})$ resulting from Step 2. Note that responsiveness is tackled first because we assume that $\lambda^a_r < \lambda^a_p$; should it be the case that $\lambda^a_r > \lambda^a_p$, MINECORE would deal with privilege in Step 2 and with responsiveness in Step 3.

This concludes the MINECORE workflow. Notice that, in addition to what we highlighted earlier, one further difference between MINECORE and most one-phase TAR systems is that, in the latter, only documents that have been manually annotated are produced to the other party; for this reason, these systems lead the annotator to identify as many relevant (i.e., responsive and nonprivileged) documents as early as possible. In MINECORE, instead, the documents that are produced to the other party may or may not have been manually annotated; unlike the above systems, MINECORE leads the annotator to annotate as early as possible the documents that, if not manually annotated, bring about the highest expected misclassification cost.

2.3.4 Callaghan Müller-Hansen method

Callaghan, Müller-Hansen (2020) proposed a stopping heuristic based on an estimation of the probability of having reached the target recall, which is then compared against a confidence level. The CMH method consists of two phases:

1. first, documents are screened and reviewed via an ALvRS policy until the target recall is achieved with a given confidence level. Once this confidence level is reached, the process stops and the second phase starts;

2. in the second phase, the review is carried out via random sampling and a higher confidence level. This should give stronger guarantees of having actually reached the required target recall.

CMH heuristic treats batches of previously screened documents as if they were random samples (an assumption somewhat similar to the one we make in Section 5.4.1); for subsets $A_i = \{d_{N_{seen}-1}, ..., d_{N_{seen}-i}\}$ of these documents they compute $p = \Pr(X \leq k)$, where
$X \sim \text{Hypergeometric}(N, K_{tar}, n)$: n is the size of the subsample, N is the total number of documents and $K_{tar} = \lfloor \frac{\rho_{seen}}{R} - \rho_{AL} + 1 \rfloor$ represents the minimum number of relevant documents remaining at the start of sampling. This is done for all sets A_i with $i \in N_{seen} - 1...1$; p_{min} is the value where the null-hypothesis (i.e., recall being below target) is lowest. The review of documents proceeds with AL until $p_{min} < 1 - \alpha$; α is a confidence level, which is set to 95%.

2.3.5 The autostop framework

Li, Kanoulas (2020) proposed a new TAR framework (mainly aimed to systematic reviews, but perfectly suitable for other TAR applications as well). The framework is based on an active learning process and an estimator: the review process is stopped when the estimator deems the target recall R to be achieved; we reproduce Li, Kanoulas (2020, Fig. 1) in Figure 2.2, which gives an overview of the autostop framework.

The framework starts with an initial document seed set S, which consists solely of the description of the systematic review topic. However, S is augmented with k documents randomly sampled (without replacement) from the pool P, which are temporarily labelled as non-relevant (similarly to Cormack, Grossman (2015a)): this forms the training set at iteration 0, L_0. A classifier ϕ is trained on L_0 and produces a ranking of all the documents in P: a sampling distribution $\mathcal{P}_0$ is built based on the ranking (the AP Prior distribution, see Li, Kanoulas (2020, §3.2)). A batch B_0 of documents are sampled with replacement from $\mathcal{P}_0$, and reviewed by the human annotator. Once the reviewer's assessments are collected, the total number of relevant documents $|\hat{L}_r|$ (and its variance) is estimated via the Horvitz-Thompson and the Hansen-Hurwitz estimators (see Li, Kanoulas (2020, §3.3,3.4)). A more or less optimistic (i.e., with higher or lower confidence) strategy is then employed to decide whether to stop or not the reviewing process, which otherwise goes on to the next iteration.

2.3.6 Quant and QuantCI

The QuantCI method proposed by Yang et al. (2021a) leverages the classifier predictions (a logistic regression) to estimate the current recall, computes a confidence interval based on variance in the predictions, and finally stops the reviewing process when the lower bound of the confidence interval reaches the target recall.

More specifically, the estimated recall $\hat{R}$ is computed as:

$$\hat{R} = \frac{\widehat{|L_r|}}{\widehat{|P_r|}} = \frac{\sum_x^{|L|} \Pr(\oplus|x)}{\sum_x^{|P|} \Pr(\oplus|x)} \tag{2.7}$$

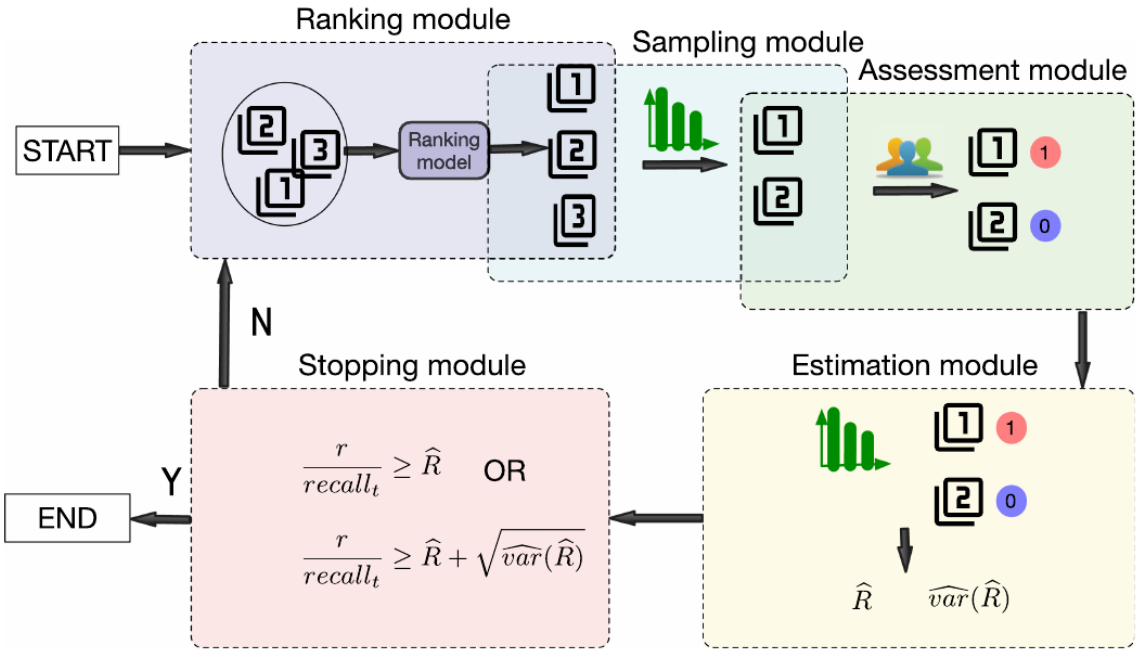


Figure 2.2: Li and Kanoulas autostop framework (Li, Kanoulas, 2020, Figure 1).

This estimate is based on modeling the relevance of a document i as the outcome of a Bernoulli random variable $D_i \sim Bernoulli(\Pr(\oplus|x))$. Equation 2.7 can then also be written as:

$$\hat{R} = \mathbb{E}\left[\frac{D_L}{D_P}\right] \tag{2.8}$$

Yang et al. (2021a) propose to estimate this quantity using a Taylor series truncated to the first order. The 95% confidence interval (CI) is then computed as:

$$\pm 2\sqrt{\frac{1}{\widehat{|P_r|}^2} Var(D_L) + \frac{\widehat{|L_r|}^2}{\widehat{|P_r|}^4}\left(Var(D_L) + Var(D_U)\right)} \tag{2.9}$$

Where with U we indicate the set of unlabelled documents (i.e. $P \setminus L$). We refer the reader to Yang et al. (2021a, §4) for a more detailed explanation of the QuantCI baseline.

The authors tested their method with and without the confidence interval (i.e., using the recall estimate as is) resulting in two stopping techniques, called QuantCI and Quant.

2.4 The SLD algorithm

As previously mentioned, most of our work focuses on the Saerens-Latinne-Decaestecker (SLD) algorithm, deeply analyzing its possible usage for technology-assisted review. We give hereby an overview of the algorithm.

We assume a training set L of labelled examples and a set $U = \{(\mathbf{x}_1, y(\mathbf{x}_1)), \ldots, (\mathbf{x}_{|U|}, y(\mathbf{x}_{|U|}))\}$ of unlabelled examples, i.e., examples whose true labels $y(\mathbf{x}_i) \in Y = \{y_1, \ldots, y_{|Y|}\}$ are unknown to the system.

SLD, proposed by Saerens et al. (2002), is an instance of Expectation Maximization (Dempster et al., 1977), a well-known iterative algorithm for finding maximum-likelihood estimates of parameters (in our case: the class prior probabilities) for models that depend on unobserved variables (in our case: the class labels). Pseudocode of the SLD algorithm is here included as Algorithm 2.

Algorithm 2: The SLD algorithm Saerens et al. (2002).

Input : Class priors $\Pr_L(y_j)$ on L, for all $y_j \in Y$;
Posterior probabilities $\Pr(y_j|\mathbf{x}_i)$, for all $y_j \in Y$ and for all $\mathbf{x}_i \in U$;
Output: Estimates $\hat{\Pr}_U(y_j)$ of class prevalences on U, for all $y_j \in Y$;
Updated posterior probabilities $\Pr(y_j|\mathbf{x}_i)$, for all $y_j \in Y$ and for all $\mathbf{x}_i \in U$;

1 // Initialization
2 $s \leftarrow 0$;
3 **for** $y_j \in Y$ **do**
4 $\quad \hat{\Pr}_U^{(s)}(y_j) \leftarrow \Pr_L(y_j)$; // Initialize the prior estimates
5 $\quad$ **for** $\mathbf{x}_i \in U$ **do**
6 $\quad\quad \Pr^{(s)}(y_j|\mathbf{x}_i) \leftarrow \Pr(y_j|\mathbf{x}_i)$; // Initialize the posteriors
7 $\quad$ **end**
8 **end**
9 // Main Iteration Cycle
10 **while** *stopping condition = false* **do**
11 $\quad s \leftarrow s + 1$;
12 $\quad$ **for** $y_j \in Y$ **do**
13 $\quad\quad \hat{\Pr}_U^{(s)}(y_j) \leftarrow \frac{1}{|U|} \sum_{\mathbf{x}_i \in U} \Pr^{(s-1)}(y_j|\mathbf{x}_i)$; // Update the prior estimates
14 $\quad\quad$ **for** $\mathbf{x}_i \in U$ **do**
15 $\quad\quad\quad \Pr^{(s)}(y_j|\mathbf{x}_i) \leftarrow \frac{\frac{\hat{\Pr}_U^{(s)}(y_j)}{\hat{\Pr}_U^{(0)}(y_j)} \cdot \Pr^{(0)}(y_j|\mathbf{x}_i)}{\sum_{y_j \in Y} \frac{\hat{\Pr}_U^{(s)}(y_j)}{\hat{\Pr}_U^{(0)}(y_j)} \cdot \Pr^{(0)}(y_j|\mathbf{x}_i)}$ // Update the posteriors
16 $\quad\quad$ **end**
17 $\quad$ **end**
18 **end**
19 // Generate output
20 **for** $y_j \in Y$ **do**
21 $\quad \hat{\Pr}_U(y_j) \leftarrow \hat{\Pr}_U^{(s)}(y_j)$; // Return the prior estimates
22 $\quad$ **for** $\mathbf{x}_i \in U$ **do**
23 $\quad\quad \Pr(y_j|\mathbf{x}_i) \leftarrow \Pr^{(s)}(y_j|\mathbf{x}_i)$ // Return the adjusted posteriors
24 $\quad$ **end**
25 **end**

Essentially, SLD iteratively updates (Line 13) the class priors by using the posterior probabilities

computed in the previous iteration, and updates (Line 15) the posterior probabilities by using the class priors computed in the present iteration, in a mutually recursive fashion. The main goal is to adjust the posteriors and re-estimate the priors in such a way that they are consistent with each other, where this "mutual consistency" means that they should be such that

$$\Pr_U(y_j) = \frac{1}{|U|} \sum_{\mathbf{x}_i \in U} \Pr(y_j|\mathbf{x}_i) \tag{2.10}$$

In Section 3.6 we show that Equation 2.10 is a necessary (albeit not sufficient) condition for the posteriors $\Pr(y_j|\mathbf{x}_i)$ of the documents $\mathbf{x}_i \in U$ to be calibrated. SLD may thus be viewed as making a step towards calibrating these posteriors (regarding calibration, see Section 3.1).

The algorithm iterates until convergence, i.e., until the class priors become stable and Equation 2.10 is satisfied. The convergence of SLD may be tested by computing how the distribution of the priors at iteration $(s-1)$ and that at iteration s still diverge; this can be evaluated, for instance, in terms of absolute error, i.e.,

$$\mathrm{AE}(\hat{p}_U^{(s-1)}, \hat{p}_U^{(s)}) = \frac{1}{|Y|} \sum_{j=1}^{|Y|} |\hat{\Pr}_U^{(s)}(y_j) - \hat{\Pr}_U^{(s-1)}(y_j)| \tag{2.11}$$

In the experiments for most of the work presented here, we decree that convergence has been reached when $\mathrm{AE}(\hat{p}_U^{(s-1)}, \hat{p}_U^{(s)}) < 10^{-6}$; we stop SLD when we have reached either convergence or the maximum number of iterations (that we set to 1000).

At each iteration of the algorithm, all the posteriors relative to class y_j are multiplied by the same amount $\hat{\Pr}_U^{(s)}(y_j)/\hat{\Pr}_U^{(0)}(y_j)$. As a consequence, the net effect of SLD is to multiply all these posteriors by the same amount $\hat{\Pr}_U(y_j)/\hat{\Pr}_U^{(0)}(y_j)$ so that the resulting posteriors $\Pr(y_j|\mathbf{x}_i)$ are consistent with the resulting class prior $\hat{\Pr}_U(y_j)$, i.e., so that Equation 2.10 is satisfied; in other words, SLD is an iterative *rescaling* algorithm. The posteriors for different classes, though, do not get multiplied by the same amount; this is somehow obvious, since at the end of the process the posteriors for document $\mathbf{x}_i$ must all sum up to 1, which means that if the posteriors for a class y' all end up increasing, there must be at least a class y'' whose posteriors all end up decreasing.

SLD, as proposed by Saerens et al. (2002) and as described here, addresses single-label classification, i.e., the task in which exactly 1 out of $|Y|$ classes must be assigned to each document. This means that SLD can be used for *binary* classification (which is single-label classification with $|Y| = 2$), for *single-label multiclass classification* (which is single-label classification with $|Y| > 2$), and for *multi-label* classification (which is the task in which any number of classes in Y can be assigned to a document), since multi-label classification can be trivially recast into $|Y|$ independent binary classification tasks.

It is worth pointing out something which Saerens et al. (2002) did not observe, i.e., that the combination of (i) a learner that trains classifiers to return posterior probabilities, and (ii) the SLD algorithm that improves the quality of the posterior probabilities for a given set of unlabelled documents U, might be called a *transductive* algorithm (Vapnik, 1998), since it uses training documents to infer posterior probabilities only for a specific, finite set of unlabelled documents known at training time. This is different from standard *inductive* algorithms, that use training documents to infer a general-purpose hypothesis that can later be applied to the entire domain. One aspect of this transductive nature is that SLD must operate "holistically", i.e., on entire *sets* of unlabelled

documents, and cannot, for instance, update the posteriors of individual unlabelled documents in isolation of each other; another aspect is that, as (Saerens et al., 2002, p. 35) put it, "the model has to be completely refitted each time it is applied to a new data set".

Interestingly enough, SLD was originally designed with the goal of improving the posteriors, so as to improve the accuracy of classification (by means of Equation 3.1) in the presence of PPS. The fact that it also allows estimating the priors in a more accurate way than by just "classifying and counting" was considered a by-product by its authors. However, in the years that followed, thanks to increased interest in the "quantification" task (i.e., the task of estimating the prevalence of a class), SLD became a popular baseline for algorithms whose goal was the estimation of the priors.

In Saerens et al. (2002), the quality of the posteriors generated by means of SLD was measured in terms of error rate, i.e., the fraction of classification decisions that are wrong. However, a major difference between error rate and the measures we will instead use for the same purpose (see Section 3.3.1) is that the former, unlike the latter, evaluates not the posterior probabilities *per se* but the classification decisions that are based on them. Error rate is thus only an "indirect" measure of the quality of the posteriors, and a coarse one too. To see this, let us assume we are dealing with binary classification, and let us consider a document $\mathbf{x}_i$ such that its true class is y_1. According to Equation 3.1, posteriors $\Pr(y_1|\mathbf{x}_i) = .51$ and $\Pr(y_2|\mathbf{x}_i) = .49$ would lead to $\mathbf{x}_i$ being correctly classified into y_1, and so would posteriors $\Pr(y_1|\mathbf{x}_i) = .99$ and $\Pr(y_2|\mathbf{x}_i) = .01$. The former set of posteriors is equivalent to the latter set as far as error rate is concerned; however, we intuitively consider the latter set "better" than the former set, and the measures we discuss in Section 3.3.1 indeed consider it as such. Note also that classification (as implemented by means of Equation 3.1), is just a downstream application of the posteriors, and there are many such potential applications, such as (as already recalled in the introduction) ranking and cost-sensitive classification; rather than evaluating the posteriors by evaluating one of their potential applications, it seems more sensible to evaluate them directly, which can be done by means of the measures of Section 3.3.1.

Finally, notice that in parallel with our work (Esuli et al., 2021), here presented in Chapter 3, two papers affirmed the key and central role played by calibration in SLD:[6] Alexandari et al. (2020) showed that the maximum likelihood function optimized by SLD is concave and that SLD thus converges to a global maximum; Garg et al. (2020) proved instead that calibration brings consistency to SLD results. Both Alexandari et al. (2020) and Garg et al. (2020) independently explored the use SLD with calibrated and uncalibrated posteriors, and other priors estimation methods, in the context of neural network classifiers, finding that SLD with proper calibration consistently obtains better results.

2.4.1 Quantification: estimating class priors

While most chapters in this thesis are concerned with the posterior probabilities' adjustment capabilities of SLD, the algorithm has often been used for its adjusted prior probabilities. Indeed, a classifier trained in a prior probability shift scenario (see Section 2.1.1) can be heavily biased on the training set class priors $\Pr_L(y)$. In order to estimate the class priors on the test set $\Pr_U(y)$, a simple "classify and count" (CC) methodology, i.e., by simply counting the items predicted to be in y, might not output the best result: the SLD algorithm has instead shown to output better quality prior estimates.

The task of estimating the class priors (or prevalence) is called "quantification", and we will propose our own modified version of SLD (called SAL_τ) in Chapter 5 in order to estimate the

[6] We will see what calibration is in Chapter 3.

class priors. Many quantification algorithms have been proposed over the years: we will here briefly discuss some of them, i.e., Probabilistic Classify and Count (PCC, Bella et al. (2010); Lewis (1995)), Adjusted Classify and Count (ACC, Forman (2005)), Probabilistic Adjusted Classify and Count (PACC, Bella et al. (2010)), and HDy (a method based on the Hellinger Distance, González-Castro et al. (2013)).

In PCC, the class prior $\Pr_U(y)$ is computed using the classifier posterior probabilities (we might call this "soft classification"), rather than using "hard" decision as in CC; in other words, $\Pr_U(y) = \frac{1}{U}\sum_{x\in U}\Pr(y|x)$.

ACC is instead based on the observation that (let us consider the binary case for simplicity):

$$\Pr(\hat{Y}=\oplus) = \Pr(\hat{Y}=\oplus|Y=\oplus)\cdot\Pr(\oplus) + \Pr(\hat{Y}=\oplus|Y=\ominus)\cdot\Pr(\ominus) \tag{2.12}$$

This can be rewritten as:

$$\hat{\Pr}^{\text{CC}}(\oplus) = \text{tpr}_\phi\cdot\Pr(\oplus) + \text{fpr}_\phi\cdot\Pr(\ominus), \tag{2.13}$$

where tpr_ϕ and fpr_ϕ are the true and false positive rates. ACC then computes the new $\Pr^{\text{ACC}}(\oplus)$ with:

$$\Pr^{\text{ACC}}(\oplus) = \frac{\Pr^{\text{CC}}(y_1) - \hat{\text{fpr}}_\phi}{\hat{\text{tpr}}_\phi - \hat{\text{fpr}}_\phi} \tag{2.14}$$

PACC, as it could be inferred, substitutes $\Pr^{\text{PCC}}$ to $\Pr^{\text{CC}}$ in ACC.

Finally, HDy is a method based on comparing two distributions with the Hellinger distance (HD):

$$\begin{aligned}\hat{\Pr}_U^{\text{HDy}}(\oplus) &= \text{HDy}(f^L_\oplus, f^L_\ominus, f^U)\\ &= \arg\min_{0\le\alpha\le1}\{HD(\alpha f^L_\oplus + (1-\alpha)f^L_\ominus, f^U)\}\end{aligned} \tag{2.15}$$

where $f^L_\oplus$ and $f^L_\ominus$ are the probability density functions of scores for the positive and negative samples of L; f^U is the distribution of scores obtained for U by the classifier trained on L. For a more in-depth explanation of all the methods presented here we refer the reader to the original works, and to the recently published book on quantification by Esuli et al. (2023).

2.5 Evaluation measures for TAR

Technology-Assisted Review applications deal with a wide range of different sub-tasks:

1. As previously mentioned, the first step in many TAR workflows is the collection of the data pool via the formulation of a query. This step is not too different from a classical information retrieval task, albeit the user and the researcher/engineer usually has no control over the search engine;

2. The second step is usually to review the highest number of relevant documents from the pool P (this does not necessarily hold for two-phase workflows, see Section 2.3). This is usually done via an active learning algorithm, and can be evaluated on:

(a) the recall R_s achieved when the process is stopped, which usually has to match an input target recall R provided by the user;

(b) the work saved with the active learning process with respect to annotating a random sample of the pool P;

(c) the annotation and misclassification costs brought about by the review process. This is an important aspect of TAR applications, as their main goal should be a joint minimization of both costs (or in other words, the maximization of cost-effectiveness).

3. The third and final step, when present, is a second review stage: the difference with the second step is that the review team usually changes (e.g., second step is fulfilled by junior reviewers and third step by seniors), and so do their hourly rates. When this third step is present, evaluating the algorithm on its capability to jointly minimize annotation and misclassification costs become even more crucial.

Given this, it is clear that TAR applications should be evaluated with a diverse range of metrics, targeting each of the aspects we have just illustrated. That said, however, most research works usually focus on one or two steps of the TAR pipeline (and are thus evaluated accordingly);[7] moreover, despite a few works have recently raised attention on the importance of TAR costs (Oard et al., 2018; Yang et al., 2021b), this aspect has been often overlooked in the literature.

2.5.1 Evaluating the review method: Recall matching and work saved over sampling

TAR tasks are usually defined, by the one-phase TAR literature, as high recall retrieval tasks, i.e. the goal is to review most (if not all) relevant documents in P. While reviewing the whole pool of documents would certainly guarantee the recall target (assuming infallible reviewers, see Section 2.2.1), this is often not feasible as P can consist of several thousands of documents. For this reason, one-phase TAR algorithms focus on reviewing relevant documents as soon as possible, and on stopping the review process once the target recall R has been achieved.

Hence, TAR algorithms need to be evaluated on their capability of properly stopping the review process once the recall goal is reached. The most used metrics in the literature are:

- the Mean Squared Error (MSE) between the recall at stopping R_s and the target recall R;
- similarly, the Relative Error (RE) between R_s and R is also often used;
- the Work Saved over Sampling (WSS) metric is another standard metric used in most of the TAR related literature;
- for total recall tasks (i.e., $R = 1$), loss_{re} has been proposed in Cormack, Grossman (2016a) and used, for instance, by Li, Kanoulas (2020);
- finally, the reliability metric was proposed by Cormack, Grossman (2016a), and it can be used to evaluate very high recall targets.

[7] In this thesis, we exclusively focus on the second and third steps.

Mean Squared Error (MSE) and Relative Error (RE) in TAR applications: The Mean Squared Error (MSE) is a metric used in many machine learning applications, often even used as a loss function in deep learning. MSE is computed as:

$$\text{MSE} = (R - R_s)^2 \tag{2.16}$$

together with the Relative Error (RE) metric, its goal is to measure how distant the recall at stopping R_s is from the target recall R, equally penalizing algorithms which stop too early or too late.

Relative error is computed as:

$$\text{RE} = \frac{|R - R_s|}{R} \tag{2.17}$$

and has been recently used to evaluate TAR algorithms in Li, Kanoulas (2020). We will use both metrics in Chapter 5 to evaluate our SAL_τ method.

Work Saved over Sampling (WSS): Work Saved over Sampling (WSS) was first proposed in Cohen et al. (2006) in the context of systematic review screening, and it measures the reduction in human workload by using automation tools (i.e., whether it is beneficial to use TAR algorithms with respect to annotating a random sample). WSS is usually measured at a target recall in percentage $t\%$ (WSS@$t\%$, t stands for threshold); in other words, since we often have a target recall to achieve, we are interested in measuring how much work we have saved by using a specific algorithm over using random sampling. WSS@$t\%$ is defined as:

$$\text{WSS@}t\% = \frac{TN + FN}{|P|} - (1 - t), \tag{2.18}$$

where TN is the number of true negatives and FN the number of false negatives.

loss_{re} metric for total recall tasks: A metric proposed in Cormack, Grossman (2016a) for total recall tasks is loss_{re}:

$$\text{loss}_{re} = (1 - R_s)^2 + \left(\frac{\overline{r}}{P}\right)^2 \left(\frac{|L|}{|L_r| + \overline{r}}\right)^2, \tag{2.19}$$

where $|L_r|$ indicates the number of relevant documents annotated in the training set. This metric was recently used by Li, Kanoulas (2020). For better comprehension, we can break down the metric in several parts: $(1 - R_s)^2$ is the squared error between the total recall and the recall at stopping. The second part of the equation is heuristic: $\frac{|L|}{|L_r| + \overline{r}}$ indicates how much of the annotation effort was actually spent annotating relevant documents, where $\overline{r}$ represents the maximum amount of non-relevant documents we are willing to annotate in the process (due to classifier mistakes); $\left(\frac{\overline{r}}{P}\right)^2$ decides how much weight to give to this second part of the equation. As it may be inferred, $\overline{r}$ has to be set heuristically: Cormack, Grossman (2016a) suggest a value of $\overline{r} < 1000$, and Li, Kanoulas (2020) use a value of 100.

The reliability metric: The reliability metric was proposed in Cormack, Grossman (2016a) and measures the percentage of fulfilled target recalls by a method across different tasks $\mathcal{Q}$ (e.g., different systematic review topics). More formally, reliability is expressed as:

$$\text{reliability} = \frac{|\{q|R_s(q) \leq R(q), q \in \mathcal{Q}\}}{|\mathcal{Q}|}, \tag{2.20}$$

where q is a specific topic/task, while $R_s(q)$ and $R(q)$ are the recall at stopping and the target recall for task q, respectively. This metric was recently used by Li, Kanoulas (2020), but was rejected by Yang et al. (2021a) as the metric has no formal penalty for algorithms, which, in an extreme case, would never stop the review (thus reaching the total recall). The metric should thus be complemented with a cost-aware one: a method which achieves a higher recall with respect to another should be of course considered better, costs being equal.

2.5.2 Evaluating annotation and misclassification costs

As previously mentioned, the review cost is one of the most critical aspects to analyze in order to understand whether a proposed TAR algorithm is truly effective or not: the review effort can in fact span over many years (Michelson, Reuter, 2019; Shemilt et al., 2016) and, in the case of systematic reviews in empirical medicine, each review can cost about $141,194.80 U.S. dollars, according to Michelson, Reuter (2019). Beside annotation costs, misclassification errors can also play a critical role: in e-discovery, for instance, producing a document which actually had to be logged can have much worse consequences than producing a document that had to be withdrawn (see Section 1.1 and 2.3.3); similarly, in systematic reviews missing too many relevant documents can jeopardize the quality of the review itself.

Despite the primary role that review costs should have when assessing a TAR algorithm (or framework), literature has often overlooked this aspect. The two rare exceptions, to the best of our knowledge, are Oard et al. (2018); Yang et al. (2021b) (Yang and Lewis would then later evaluate one of their methods (Yang et al., 2021a) based on their metric illustrated in Yang et al. (2021b)). In Oard et al. (2018), the authors propose an evaluation of TAR algorithms, along with their novel MINECORE framework (see Section 2.3.3), based on several cost structures that they elicited from e-discovery experts. Yang et al. (2021b) propose instead an "idealized" cost metric suited to evaluate any type of technology-assisted review, albeit not giving an expert-elicited cost structure like Oard et al. (2018) did (which is reasonable, as they target multiple TAR domains with their metric).

Cost structures, annotation and misclassification costs for e-discovery: In Section 1.1, we illustrated the different categories to which each document can be assigned: **P**roduce (c_P), **L**og (c_L) and **W**ithdraw (c_W). Different types of misclassification errors bring about different costs: for instance, disclosing highly confidential intellectual property to the other party (i.e., assigning a document to c_P instead of c_L) is usually the worst case scenario.

In Section 2.3.3, we have seen how Oard et al. (2018) defined the linear costs assumed by their MINECORE framework. These cost metrics are not, however, exclusively exploitable by MINECORE. Indeed, any TAR framework can be flawlessly evaluated with the same metric. As anticipated, Oard et al. (2018) define a cost matrix Λ^m for misclassification costs (Table 2.1): any λ^m_{ij} is the misclassification cost of assigning class c_i to a document actually belonging to c_j.

		actual		
		c_P	c_L	c_W
pred	c_P	0	λ^m_{PL}	λ^m_{PW}
	c_L	λ^m_{LP}	0	λ^m_{LW}
	c_W	λ^m_{WP}	λ^m_{WL}	0

Table 2.1: the cost matrix Λ^m defined in Oard et al. (2018).

	λ^a_r	λ^a_p	λ^m_{PL}	λ^m_{PW}	λ^m_{LP}	λ^m_{LW}	λ^m_{WP}	λ^m_{WL}
CostStructure1	1.00	5.00	600.00	5.00	150.00	3.00	15.00	15.00
CostStructure2	1.00	5.00	100.00	0.03	10.00	2.00	8.00	8.00
CostStructure3	1.00	5.00	1000.00	0.10	1.00	1.00	1.00	1.00

Table 2.2: Cost structures defined in Oard et al. (2018), as elicited from different experts. Costs are expressed in US$.

Annotation costs are instead defined as unit costs λ^a_r and λ^a_p, the cost of annotating a document by responsiveness or privilege respectively.

Oard et al. (2018) elicited three cost structures from TAR and e-discovery experts, which we report in Table 2.2. Let us assume we have reviewed the subsets D_r of documents for responsiveness, and D_p for privilege (usually, $D_p \subset D_r$). Our labelled set L is then $L = D_r \cup D_p$. The total annotation and misclassification costs can then be computed as:

$$K^a(L) = \lambda^a_r|D_r| + \lambda^a_p|D_p| \tag{2.21}$$

$$K^m(L) = \sum_{i,j\in\{P,L,W\}} \lambda^m_{ij}|D_{ij}|, \tag{2.22}$$

where with $|D_{ij}|$ we indicate the number of documents assigned to class c_i, when in truth belonging to class c_j. The overall cost is then simply defined as $K^o(L) = K^a(L) + K^m(L)$.

An idealized cost structure for TAR: The Idealized Cost (IC) was recently proposed by Yang et al. (2021b) as a cost-based metric to evaluate any TAR algorithm. As previously mentioned, TAR algorithms cannot be evaluated solely on their ability to find relevant documents or to stop the review at a given target recall: a very crucial aspect is to be able to quantify the expected costs of the application of a TAR algorithm/framework over another. Considered this, Yang et al. (2021b) propose a new metric similar, in principle, to Oard et al. (2018)'s. The IC metric is defined as follows: it uses a cost structure, a four-tuple $s = (\alpha_r, \alpha_n, \beta_r, \beta_n)$. Subscript r and n indicate the cost of reviewing positive (relevant) and negative (non-relevant) documents; α and β represents the costs of reviewing a document in a first or second phase (see Section 2.3): in a one-phase TAR process, this "second" phase is referred to as the *failure penalty*. That is, it would be the cost of continuing the review with an optimal second phase, by ranking documents with the model trained in the first one.

Let Ψ be the minimum number of documents to review to reach the recall target R. Say we review batches of size b and we stop at iteration t: let L_r be the number of positive (relevant)

documents we reviewed before the method stopped the review. If $L_r < \Psi$, we have a deficit of $\Psi - L_r$ positive documents; let ρ_t be the minimum number of documents that need be reviewed to find the additional $\Psi - L_r$ positive documents. The total cost of our review is then:

$$\text{IC} = \alpha_r L_r + \alpha_n(bt - L_r) + I[L_r < \Psi]\,(\beta_r(\Psi - L_r) + \beta_n(\rho_t - \Psi + L_r)) \tag{2.23}$$

Where $I[L_r < \Psi]$ is 0 if Ψ documents were found in the first t iterations and 1 otherwise.

As anticipated, Yang et al. (2021b) do not propose cost structures elicited from experts like Oard et al. (2018) did. Nonetheless, they still defined different cost structures for different situations that often occur in TAR applications. Following both Yang et al. (2021b) and Oard et al. (2018), we use (in Chapter 5) three cost structures for the IC metric:

- a uniform cost structure, where $s = (1, 1, 1, 1)$, which we call Cost_u. This assumes that there is no difference between the different phases of review, and that reviewing positive and negative documents have the same cost (we keep this latter assumption in all our cost structures). As argued by Yang et al. (2021b, §4.1), this cost structure is common in many review scenarios;
- the expensive training cost structure, where $s = (10, 10, 1, 1)$, which we call Cost_e. This assumes that reviewing a document in the first phase is 10 times more expensive than in the second phase. According to Yang et al. (2021b, §4.2) this is fairly common in systematic reviews in empirical medicine (see Section 1.1);
- a MINECORE-like cost structure that we propose, where $s = (1, 1, 5, 5)$, which we call Cost_m. This cost structure reflects MINECORE cost structure 2 (see Table 2.2), where it is assumed that reviewing in the second stage is 5 times as expensive as in the first stage.[8]

Notice that Yang et al. (2021b) defined several other cost structures (which we are not going to use in our evaluations), where reviewing positives is more expensive than reviewing negatives: intuitively, this makes sense since positive items usually require more time to be assessed.[9] For this scenario, Yang et al. propose an additional cost v for each positive document. The structure is then $s = (\alpha + v, \alpha, \beta + v, \beta)$. The cost function when $L_r \leq \Psi$ is defined as:

$$\begin{aligned} Cost(t) &= ((\alpha + v) - \alpha - (\beta + v) + \beta)L_r + \\ &\quad + \alpha bt + \beta\rho_t + ((\beta + v) - \beta)\Psi \\ &= \alpha bt + \beta\rho_t + v\Psi \end{aligned} \tag{2.24}$$

which becomes $\alpha bt + v\Psi + v(L_r - \Psi)$ when $L_r \geq \Psi$. We refer the reader to (Yang et al., 2021b, §4.3) for a more comprehensive explanation of the different cost structures.

2.6 Datasets for e-discovery and systematic reviews

In e-discovery, the producing party is legally required to find and assign to either c_P, c_L or c_W all documents in the pool P. Real e-discovery datasets are usually not publicly available, as privileged

[8] Notice, however, that in Yang et al. (2021b) the two "phases" refer to the one-two phase TAR workflows (see Section 2.3), whereas in Oard et al. (2018) they refer to review by responsiveness and privilege. Nonetheless, we believe adding this cost structure allows to consider all possible scenarios (i.e., uniform costs, a more expensive first phase, and a more expensive second phase).

[9] Moreover, in e-discovery, "positive documents may require review for factors (e.g., attorney-client privilege) not applicable to negative documents." (Yang et al., 2021b, §4.3.1).

information would then be public for everyone to download. Recently, however, Sayed et al. (2020) published an e-discovery dataset, an annotation of the Avocado Research Email Collection distributed by the Linguistic Data Consortium, on a restricted research license. Due to the restricted availability, however, we do not use this dataset in the experiments run for this thesis: we rather fall back to emulating the responsiveness and privilege annotations on the RCV1-v2 dataset (this was also used with the same purpose in, e.g., Oard et al. (2018); Yang et al. (2021a)).

Regarding the production of systematic reviews in empirical medicine, real-case datasets are instead available, albeit only for the first reviewing stage (i.e., the abstract screening): as a matter of fact, full-length articles are often under a paywall and thus not freely available to download. In our experiments for Chapters 5 and 6, we employ the CLEF 2019 EMED dataset.

2.6.1 RCV1-v2

RCV1-v2 (Lewis et al., 2004) is a publicly available collection of 804,414 news stories from the late nineties, published on the Reuters website. RCV1-v2 is integrated and easily accessible from the Scikit-Learn Python library (Pedregosa et al., 2011). RCV1-v2 is a multi-label multi-class collection, i.e., every document can be assigned to one or more classes from a set $\mathcal{C}$ of 103 classes. Since for our experiments we need binary classification datasets (i.e., a document can either be relevant or not), for each class $c \in \mathcal{C}$ we consider each document d as either belonging to c or not, thus obtaining 103 binary datasets. Moreover, for text classification purposes, RCV1-v2 is traditionally split into a training set consisting of the (chronologically) first 23,149 documents (the ones written in Aug 1996), and a test set consisting of the last 781,265 documents (the ones written from Sep 1996 onwards). Throughout the experiments of this thesis, we use different subsets of RCV1-v2, which will be explained in detail in the respective sections.

2.6.2 CLEF EMED

The CLEF EMED datasets were made publicly available[10] for the TAR in EMED tasks ran from 2017 to 2019. The goal of the task was to assess TAR algorithms aimed at supporting the production of systematic reviews in empirical medicine. For the 2019 dataset, three different type of reviews were made available: Diagnostic Test Accuracy (DTA) reviews, also available in previous years; Intervention, Prognosis and Qualitative reviews. Following Li, Kanoulas (2020), we use the Diagnostic Test Accuracy (DTA) reviews part of the dataset, working with the abstract relevance assessments (i.e., the first phase of the review, where the physician only assesses abstracts): the dataset consists of 72 "Training" topics and 8 "Testing" topics.

The texts of the reviewed documents are not available for download on the GitHub platform: they must be downloaded from PubMED,[11] an online search engine with more than 34 million citations for biomedical literature. While an HTTP API is available, the full text of documents are often under a paywall: hence the choice of focusing on the abstract reviews only. Moreover, we have encountered several issues in downloading some of the abstracts (that is, API errors), thus being unable to retrieve the whole dataset: in total we have retrieved abstracts for 60 topics (between "Training" and "Testing"), downloading a collection of 264,750 documents. We published the dataset at (Molinari, 2022).

[10] `https://github.com/CLEF-TAR/tar`
[11] `https://pubmed.ncbi.nlm.nih.gov/`

Chapter 3

A reassessment of the SLD algorithm for posterior probabilities improvement

As explained in Chapters 1 and 2, TAR workflows usually rely on an active learning strategy to train a classifier, which aids the reviewer in labelling the document pool P. Active learning strategies naturally generate high *prior probability shift* (PPS), i.e. the phenomenon for which the prior probability $\Pr_L(y_j)$ of a class $y_j \in Y$ in the training set is (more or less) different than the prior probability $\Pr_U(y_j)$ for the same class in the test set (see Section 2.1.1). The Saerens-Latinne-Decaestecker (SLD) algorithm, here illustrated in Section 2.4, was proposed in 2002 as a methodology to improve both prior and posterior probabilities estimates in PPS scenarios: in this chapter, we conduct an extensive analysis of the SLD algorithm for posterior probabilities adjustment, with the end-goal of leveraging this procedure to improve both our prior and posterior estimates in TAR workflows (which we will attempt in Chapter 4 and 5). This work was published in Esuli et al. (2021).

3.1 Introduction

Single-label text classification is the task of training a text classifier $\phi : X \to Y$ that labels each document $\mathbf{x}_i \in X$ with a class $\phi(\mathbf{x}_i) \in Y$; X is a (possibly infinite) set of documents (the *domain*), while $Y = \{y_1, ..., y_{|Y|}\}$ is a finite set of classes (the *codeframe*, or *classification scheme*).

The classifiers trained by means of modern machine learning methods usually return, together with the class assigned to the document, a vector $(s(\mathbf{x}_i, y_1), ..., s(\mathbf{x}_i, y_{|Y|}))$ of *confidence scores*, where $s(\mathbf{x}_i, y_j)$ by and large represents the confidence (or the strength of belief) that the classifier has in the fact that $\mathbf{x}_i$ belongs to y_j; the class $\phi(\mathbf{x}_i)$ assigned to document $\mathbf{x}_i$ is thus the one with the highest confidence score, i.e.,

$$\phi(\mathbf{x}_i) = \arg\max_{y_j \in Y} s(\mathbf{x}_i, y_j) \tag{3.1}$$

Classifiers that return confidence scores are sometimes called *scoring classifiers* (Fawcett, 2006).

Without loss of generality[1] we may assume that these confidence scores are actual probabilities (if so, these are called *posterior probabilities*, or simply *posteriors*), i.e., we may assume that the vector being returned has the form $(\Pr(y_1|\mathbf{x}_i), ..., \Pr(y_{|Y|}|\mathbf{x}_i))$, where $\sum_{j=1}^{|Y|} \Pr(y_j|\mathbf{x}_i) = 1$ and $\Pr(y_j|\mathbf{x}_i)$ represents the probability that the classifier "subjectively" attributes to the fact that x_i belongs to class y_j. Rather than simple classifiers, these models are full-blown *probability estimators*.

The posteriors play an important role in several tasks, a role that goes beyond allowing to take a classification decision by means of Equation 3.1. One of these tasks is *document ranking*, as when the documents are ranked in decreasing order of the probability $\Pr(y_j|\mathbf{x}_i)$ that they belong to a certain class y_j; ranking is useful, for instance, when performing active learning by means of relevance sampling (Lewis, Gale, 1994), or when one needs to choose the best k documents for a certain class, or when one needs to choose the best k classes for a certain document. Another such task is *cost-sensitive classification*, where classification is performed in such a way that

$$\phi(\mathbf{x}_i) = \operatorname*{arg\,min}_{y_j \in Y} \sum_{y_l \in Y} \lambda_{jl} \cdot \Pr(y_l|\mathbf{x}_i) \tag{3.2}$$

where λ_{jl} represents the "cost" of classifying a document in class y_j when it should have been classified in class y_l (this cost is equal to 0 when $j = l$ and higher than 0 when $j \neq l$); in other words, $\mathbf{x}_i$ is assigned to the class such that the expected cost (i.e., the risk) of assigning $\mathbf{x}_i$ to it is minimum. Example applications of cost-sensitive text classification may be found, for instance, in spam filtering (Cormack, 2008), or in technology-assisted review, as we have seen with the MINECORE framework (Oard et al., 2018).

Of course, in order to guarantee that single-label multiclass classification, ranking, cost-sensitive classification, and other such tasks, are executed with high accuracy, the posteriors must be accurate too. An intuition of what "accurate posteriors" means can be provided by the following example. For instance, if 10% (resp., 90%) of all the documents $\mathbf{x}_i$ for which $\Pr(y_j|\mathbf{x}_i) = 0.5$ indeed belong to y_j, we can say that the classifier has overestimated (resp., underestimated) the probability that these documents belong to y_j, and that their posteriors are thus inaccurate. Indeed, we say (see for instance Flach (2017)) that the posteriors $\Pr(y_j|\mathbf{x}_i)$, where $\mathbf{x}_i$ belongs to a set $\Upsilon = \{\mathbf{x}_1, ..., \mathbf{x}_{|\Upsilon|}\}$, are (perfectly) *calibrated* (i.e., accurate) when, for all $a \in [0, 1]$, it holds that[2]

$$\frac{|\{\mathbf{x}_i \in \Upsilon \cap y_j | \Pr(y_j|\mathbf{x}_i) = a\}|}{|\{\mathbf{x}_i \in \Upsilon | \Pr(y_j|\mathbf{x}_i) = a\}|} = a \tag{3.3}$$

The classifiers trained by means of some learners (such as logistic regression) are known to return reasonably well calibrated probabilities. Those trained by means of some other learners (such as Naïve Bayes) return probabilities which are known to be not well calibrated (Domingos, Pazzani, 1996). Yet other learners (such as SVMs or AdaBoost) train classifiers that return confidence scores that are not probabilities (i.e., that do not range on [0,1] and/or that do not sum up to 1). In order to address these two latter cases, *probability calibration* mechanisms exist (see e.g., Niculescu-Mizil, Caruana (2005a,b); Platt (2000); Wu et al. (2004); Zadrozny, Elkan (2002)) that convert the outputs of these classifiers into well calibrated probabilities.

However, even when using text classifiers that tend to return well calibrated probabilities, or even when using the probability calibration methods mentioned above, the accuracy of the posteriors

[1] See the discussion on probability calibration mechanisms later in this section.

[2] Perfect calibration is usually unattainable on any non-trivial dataset; however, calibration comes in degrees (and the quality of calibration can indeed be measured – see Section 3.3.1), so efforts can be made to obtain posteriors which are as close as possible to their perfectly calibrated counterparts.

tends to be low if the problem setting exhibits *dataset shift* (see e.g., Quiñonero-Candela et al. (2009) and Section 2.1.1). To see why this is the case, take the probabilistic classifier

$$\Pr(y_j|\mathbf{x}_i) = \frac{\Pr(\mathbf{x}_i|y_j)\Pr(y_j)}{\Pr(\mathbf{x}_i)} \tag{3.4}$$

and note that the posterior $\Pr(y_j|\mathbf{x}_i)$ on the left-hand side directly depends on the prior $\Pr(y_j)$ on the right-hand side. Since the prior $\Pr(y_j)$ has been estimated on the training set (i.e., its value has been set to $\Pr_L(y_j)$, which is distributed as $p_L(y)$), if $\Pr_L(y_j)$ is higher (resp., lower) than $\Pr_U(y_j)$ (which is distributed as $p_U(y)$), then the posteriors $\Pr(y_j|\mathbf{x}_i)$ of the documents in U will be overestimated (resp., underestimated).[3] Ideally, in order to have well calibrated posteriors even in the presence of dataset shift, we would need to set $\Pr(y_j)$ in Equation 3.4 to $\Pr_U(y_j)$, and not to $\Pr_L(y_j)$. But this is impossible, since $\Pr_U(y_j)$ is unknown at training time. The only known way out of this conundrum is provided by the Saerens-Latinne-Decaestecker algorithm[4], an algorithm that iteratively re-estimates the priors $\Pr_U(y_j)$ of the unlabelled set and adjusts the posteriors $\Pr(y_j|\mathbf{x}_i)$, in a mutually recursive way (Saerens et al. (2002), see Section 2.4). This algorithm is essentially unique in its kind, and, to the best of our knowledge, no other algorithm that attempts to adjust the posteriors in the presence of prior probability shift has been proposed since its publication. An exception is the algorithm described in Sun, Cho (2018); in Section 3.2 we discuss why we do not consider it as a contender. As a result, SLD has become a standard, and is frequently used in scenarios characterized by PPS, either when the goal is improving the accuracy of the posteriors, or when the goal is obtaining estimates of the priors more accurate than can be obtained by the trivial "classify and count" method (the latter task is known as *supervised prevalence estimation*, or *quantification* González et al. (2017)).

However, in recent experiments aimed at improving the quality of cost-sensitive text classification in technology-assisted review (Molinari, 2019a,b), SLD has not delivered any measurable improvement in the quality of the posteriors. Since these experiments were limited in scope, we have then decided to engage in a large-scale experimentation of SLD, with the goal of reassessing its true ability at (i) accurately re-estimating the priors $\Pr_U(y_j)$ of the unlabelled set, and (ii) improving the quality of the posteriors $\Pr(y_j|\mathbf{x}_i)$ of the unlabelled documents. Note that goal (ii) is more important than goal (i), since the ability of SLD at estimating the priors has been systematically tested in previous works (e.g., Esuli et al. (2018)), and since (as mentioned before) SLD is essentially the only known algorithm for improving the quality of already calibrated posteriors, while there are many alternatives to it (see the extensive review by González et al. (2017)) when it comes to estimating the priors. We thus present systematic experiments involving different learners, different datasets, and different amounts of PPS, in which we try to assess the real benefits of using SLD. Notice that in Saerens et al. (2002), SLD was subjected to a small-scale experimentation, which involved the binary case only. The experiments we conduct in this chapter are instead carried out on a very large scale, and involve both binary and multiclass classification.

The rest of the chapter is structured as follows. In Section 3.3 we present the systematic experimentation to which we have subjected SLD, and in Section 3.4 we present its results, while in

[3] In other words, that prior probability shift brings about a low quality of the posteriors is due to the fact that PPS, as all types of dataset shift, invalidates the iid assumption (according to which the training examples and the unlabelled examples are drawn from the same distribution), on which probability calibration methods rely.

[4] In a number of other publications (Esuli et al., 2018; Gao, Sebastiani, 2016; Molinari, 2019a,b) the same algorithm was called EMQ, standing for "Expectation Maximization for Quantification"; in yet other publications (Bequé et al., 2017) it is called RS, standing for "rescaling algorithm".

Section 3.5 we discuss exactly which kinds of dataset shift we target in our experiments. Section 3.2 discusses some related work, while Section 3.7 concludes.

3.2 Related work

Despite having been proposed more than 15 years ago, SLD remains an algorithm unique in its kind, since at the same time it updates the posterior probabilities *and* the class prior probability estimates returned by the classifier.

As discussed in the previous sections (and, later, in Section 3.6), SLD bears strong relations to probability calibration. While several calibration methods have been proposed in the last 20 years (e.g., Alasalmi et al. (2020); Bequé et al. (2017); Coussement, Buckinx (2011); Naeini et al. (2015)), none of them actually deals with calibrating the posterior probabilities of the unlabelled set *in the presence of prior probability shift*.

As already mentioned, dataset shift (and PPS in particular) is central to SLD's concerns. Dataset shift is a multifaceted phenomenon and a largely unexplored territory, and only in the last ten years or so the machine learning community has started to address it systematically (Quiñonero-Candela et al., 2009). The task of estimating class prior probabilities in the presence of PPS has, since about 2005, evolved as a task of its own, called *quantification* (González et al., 2017), and many algorithms alternative to SLD have been proposed (see Esuli et al. (2020); Fernandes Vaz et al. (2019); Pérez-Gállego et al. (2019); Spence et al. (2019) for a few recent examples). However, while these algorithms are interesting alternatives to SLD as far as estimating class prior probabilities goes, there are no current alternatives to SLD when it comes to *adjusting the posterior probabilities* in the presence of PPS. To the best of our knowledge, the only alternative to SLD that has ever been proposed for adjusting the posterior probabilities in the presence of PPS is the algorithm in Sun, Cho (2018), based on the idea of binning the unlabelled documents based on an invariance property of ROC curves. However, this algorithm *assumes that the true class priors in the unlabelled set are known*; this is an assumption which is not verified in practice (because, in the presence of distribution shift, these class priors are different from the ones in the training set), which means that this algorithm cannot be used in practice.[5]

3.3 Experiments

In this section we report systematic experiments in which, using a variety of datasets, learners, and amounts of PPS, we compare the quality of the priors and (above all) of the posteriors *before* the application of SLD, with that *after* the application of SLD. This allows us to see when and in what conditions the application of SLD is beneficial.

3.3.1 Evaluation measures

We evaluate SLD in terms of two main criteria, i.e., (i) the ability to improve the accuracy of the estimated class priors with respect to the trivial "Classify & Count" estimator, and (ii) the ability

[5]Indeed, the experiments reported in Sun, Cho (2018) use an oracle that provides to the algorithm the true class priors of the unlabelled set; but this oracle, as all oracles, is not available in practice, so the utility of this algorithm is extremely questionable.

to improve the accuracy of the posterior probabilities with respect to the ones originally returned by the classifier.

Evaluating the Priors

For evaluating the quality of the estimated class priors we use *normalized absolute error* (NAE) (see e.g., Sebastiani (2020, §4.2)), defined as

$$\text{NAE}(p_U, \hat{p}_U) = \frac{\sum_{j=1}^{|Y|} |\Pr_U(y_j) - \hat{\Pr}_U(y_j)|}{2(1 - \min_{y_j \in Y} \Pr_U(y_j))} \tag{3.5}$$

where p_U and $\hat{p}_U$ indicate the true class distribution and the predicted class distribution, resp., on the set U of unlabelled documents. The reason we use NAE is that, besides its simplicity, it is also (as argued in Sebastiani (2020)) one of the theoretically most satisfying measures for evaluating the quality of class priors; NAE ranges between 0 (best) and 1 (worst). In all the tables of results that we include in Section 3.4, we compare the estimates of the class priors before applying SLD, computed by "classifying and counting", i.e., as

$$\hat{\Pr}_U(y_j) = \frac{1}{|U|}|\{\mathbf{x}_i \in U, \phi(\mathbf{x}_i) = y_j\}|$$

with the same estimates after applying SLD (which are the values of $\hat{\Pr}_U(y_j)$ resulting from Line 21 of Algorithm 2).

Evaluating the Posteriors

For evaluating the quality of the posterior probabilities, the measure we use is the *Brier score* (Brier, 1950). Given a set $U = \{(\mathbf{x}_1, t(\mathbf{x}_1)), \ldots, (\mathbf{x}_{|U|}, t(\mathbf{x}_{|U|}))\}$ of unlabelled documents to be labelled according to codeframe Y, the Brier score is defined as

$$\text{BS} = \frac{1}{|Y| \cdot |U|} \sum_{j=1}^{|Y|} \sum_{i=1}^{|U|} (I(t(\mathbf{x}_i) = y_j) - \Pr(y_j|\mathbf{x}_i))^2 \tag{3.6}$$

where $I(\cdot)$ is a function that returns 1 if its argument is true and 0 otherwise. The Brier score ranges between 0 (best) and 1 (worst), i.e., it is a measure of error, and not of accuracy. It rewards classifiers that return a high posterior for the true class of $\mathbf{x}_i$ and low posteriors for all classes other than the true class of $\mathbf{x}_i$. The Brier score is an example of so-called *strictly proper scoring rules* (Gneiting, Raftery, 2007), defined as loss functions which are minimized only when $\Pr(y_j|\mathbf{x}_i)$ equals 1 for $y_j = t(\mathbf{x}_i)$.

It is useful to analyze the Brier score in a more fine-grained way. For class y_j, let the [0,1] interval be partitioned into an ordered sequence of ξ intervals $I_{1j}, ..., I_{\xi j}$, and let us define bins $\beta_{1j}, ..., \beta_{\xi j}$ such that $\mathbf{x}_i \in \beta_{kj}$ iff $\Pr(y_j|\mathbf{x}_i) \in I_{kj}$. (DeGroot, Fienberg, 1983, §4) show[6] that the

[6]The formulation of the Brier score originally given in DeGroot, Fienberg (1983, §4) is slightly different since the authors assume that a posterior may only take up one of a small, fixed number of values, which makes intervals and bins not necessary for the formulation of BS. While this assumption is reasonable when posteriors are returned by human beings, this is not when they are returned by automatic probabilistic classifiers; as a result, we here reformulate BS by using intervals and bins.

Brier score can be written as

$$\text{BS} = \text{CE} + \text{RE} \tag{3.7}$$

with

$$\text{CE} = \frac{1}{|Y| \cdot \xi} \sum_{j=1}^{|Y|} \sum_{k=1}^{\xi} \nu(\beta_{kj}, U) \cdot (\pi(\beta_{kj}) - \rho(y_j, \beta_{kj}))^2 \tag{3.8}$$

$$\text{RE} = \frac{1}{|Y| \cdot \xi} \sum_{j=1}^{|Y|} \sum_{k=1}^{\xi} \nu(\beta_{kj}, U) \cdot \rho(y_j, \beta_{kj}) \cdot (1 - \rho(y_j, \beta_{kj})) \tag{3.9}$$

where

- $\nu(\beta_{kj}, U)$ is the prevalence of β_{kj} in U, i.e., the fraction $\frac{|\beta_{kj}|}{|U|}$ of documents $\mathbf{x}_i$ in U that are in β_{kj};
- $\pi(B_{kj})$ is the expected value $\frac{1}{|B_{kj}|} \sum_{\mathbf{x}_i \in B_{kj}} \Pr(y_j|\mathbf{x}_i)$ of the posteriors for the documents in B_{kj};
- $\rho(y_j, B_{kj})$ is the prevalence of y_j in B_{kj}, i.e., the fraction $\frac{1}{|B_{kj}|} \sum_{\mathbf{x}_i \in B_{kj}} I(t(\mathbf{x}_i) = y_j)$ of documents $\mathbf{x}_i$ in B_{kj} that belong to class y_j.

Here, CE is a measure of the *calibration error* of the posterior probabilities; in fact, it is easy to see that its value is 0 if and only if Equation 3.3 is verified for each $S \in \{B_{1j}, ..., B_{bj}\}$. RE is instead a measure of what DeGroot, Fienberg (1983) call the *refinement error* of the classifier, i.e., of the lack of confidence of its predictions; its value is 0 if and only if all the posteriors it returns have a value of 0 or 1, while its value is 1 if and only if the classifier always "sits on the fence", i.e., if all the posteriors it returns have a value equal to the prevalence of y_j in U.[7]

As an example, in a binary setting consider a perfectly balanced unlabelled set U, consider a ("perfect") classifier ϕ' that returns $\Pr(y_j|\mathbf{x}_i) = 1$ for all $\mathbf{x}_i$ whose true class is y_j and $\Pr(y_j|\mathbf{x}_i) = 0$ for all $\mathbf{x}_i$ whose true class is not y_j, and consider a classifier ϕ'' that returns $\Pr(y_j|\mathbf{x}_i) = .50$ for all $\mathbf{x}_i \in U$. Classifiers ϕ' and ϕ'' are equivalent as far as CE is concerned (they both get a score of 0), but they are not for RE, which is equal to 0 for ϕ' and to .50 for ϕ''. Conversely, consider the same set U and the same ("perfect") classifier ϕ' of the previous example, and consider a ("perverse") classifier ϕ''' that returns $\Pr(y_j|\mathbf{x}_i) = 0$ for all $\mathbf{x}_i$ whose true class is y_j and $\Pr(y_j|\mathbf{x}_i) = 1$ for all $\mathbf{x}_i$ whose true class is not y_j. Classifiers ϕ' and ϕ''' are equivalent as far as RE is concerned (they both get a score of 0), but they are not for CE, which is equal to 0 for ϕ' and to 1 for ϕ''. Prediction power (which, in this case, manifests itself in the form of good-quality posteriors) thus requires calibration *and* refinement. Another way of saying this is that BS measures the classifier's *knowledge*, which is a combination of the classifier's *introspection*, or *self-awareness* (which is measured by CE), and of the classifier's *confidence* (which is measured by RE).

In this chapter we define and use two variants of the Brier score, i.e.,

[7] The decomposition of BS into CE and RE was originally introduced by Murphy (1973), who actually used the terms *reliability* and *resolution* to denote CE and RE, respectively; the terminology we use in this chapter is the one now current.

- the *Isometric Brier Score* (here shortened as BS_L, where L stands for "length"), which is obtained by partitioning U into intervals $I_{1j}, ..., I_{\xi j}$ of equal length; for instance, if $\xi = 10$ then $I_{1j} = [.0, .1)$, $I_{2j} = [.1, .2)$,..., $I_{\xi j} = [.9, 1.0]$;
- the *Isomerous Brier Score* (here shortened as BS_N, where N stands for "number"), which is obtained by partitioning U into intervals $I_{1j}, ..., I_{\xi j}$ such that the corresponding bins $\beta_{1j}, ..., \beta_{\xi j}$ have equal size, i.e., are such that (a) $\mathbf{x}' \in \beta_{sj}$ and $\mathbf{x}'' \in \beta_{tj}$ with $s < t$ implies that $\Pr(y_j|\mathbf{x}') \leq \Pr(y_j|\mathbf{x}'')$, and (b) $|\beta_{sj}| = |\beta_{tj}|$ for any $s, t \in \{1, ..., \xi\}$. Note that, when partitioning U this way, $\nu(\beta_{kj}, U)$ is the same for all $1 \leq k \leq \xi$.

The advantage of BS_N over BS_L is that all bins are guaranteed to have a high enough number of elements, which reduces the risk that the difference between $\rho(y_j, \beta_{kj})$ and $\pi(\beta_{kj})$ is extreme due to sparsity. In this chapter we use the BS_L variant for compatibility with previous literature (which mostly uses the BS_L variant – e.g., Bequé et al. (2017); Stephenson et al. (2008)), and the BS_N variant because, as argued, it seems to have superior formal properties.

In the experiments reported in this chapter we use $\xi = 10$.

3.3.2 Dataset

As explained in Section 2.6, RCV1-v2 is multi-label, i.e., a document may belong to several classes at the same time; since in this chapter we are interested in single-label classification, we select its "single-label fragment", i.e., the subset of RCV1-v2 documents that have exactly 1 label. In order to do so, (a) we remove all "derived" labels, leaving only "primitive" labels[8], and (b) we remove from the collection all documents that do not have exactly one "primitive" label.

For reasons that will be clear in the next paragraph, in our experiments we consider only the 37 classes with at least 2000 (training or test) positive examples; of these, 31 are "leaf" classes while the remaining 6 classes correspond to internal nodes of the hierarchy.[9] We also remove all documents that do not belong to any of these 37 classes, which leaves us with 517,978 documents.

Generating samples with controlled amounts of PPS

RCV1-v2 exhibits very little PPS between training set and test set. In fact, if we compute the normalized absolute error between p_L (the class distribution in the training set) and p_U (the class distribution in the unlabelled documents), i.e.,

$$\text{NAE}(p_L, p_U) = \frac{\sum_{j=1}^{|Y|} |\Pr_L(y_j) - \Pr_U(y_j)|}{2(1 - \min_{y_j \in Y} \Pr_L(y_j))} \tag{3.10}$$

for RCV1-v2 we obtain NAE = .0026, which is an extremely low value (since NAE always ranges between 0 – indicating no shift – and 1 – indicating maximum shift).

We instead want to test the SLD algorithm on a variety of distribution shift values, thus simulating a variety of possible application scenarios. In order to do so, by using the protocol described

[8] The RCV1-v2 codeframe has a hierarchical structure. As a result, when a document is labelled with class y_j, it is also labelled with all classes that are ancestors of y_j in the RCV1-v2 tree. Whenever a document has two labels y' and y'' such that y' is an ancestor of y'', we remove this "derived" label y' from its labels; we are thus left with "primitive" labels (i.e., labels y_j such that the document has no label which is a descendant of y_j).

[9] Each of these latter 6 classes has at least 2000 positive examples "of its own", i.e., such that none of its descendant classes has any of these examples.

below we extract from RCV1-v2 k different samples, each consisting of a training set and a test set sampled from different class distributions; all the results of our experiments will thus be average values across these k samples.

We run binary, "one-against-the-rest" classification experiments, i.e., experiments in which, for each class y_j, all the examples not belonging to y_j are considered negative examples of y_j. For these experiments:

1. We generate two random vectors $\Pi_L = (\pi_1^L, \pi_2^L)$ and $\Pi_U = (\pi_1^U, \pi_2^U)$ of class priors, i.e., two vectors such that $0 \leq \pi_j^L, \pi_j^U \leq 1$ for each $1 \leq j \leq 2$ and such that $\sum_{j=1}^{2} \pi_j^L = \sum_{j=1}^{2} \pi_j^U = 1$;[10]

2. We generate a training set σ_L by drawing $m_L = |\sigma_L|$ different documents (with m_L a parameter to be fixed beforehand), where at each draw we pick with probability π_j^L a random document among those belonging to class y_j, and with probability $(1 - \pi_j^L)$ a random document among those not belonging to y_j. We then generate a test set σ_U by first removing from the pool the documents drawn for σ_L, and then by drawing $m_U = |\sigma_U|$ different documents (with m_U a parameter to be fixed beforehand), where at each draw we pick with probability π_j^U a random document among those belonging to class y_j, and with probability $(1 - \pi_j^U)$ a random document among those not belonging to y_j. We thus obtain a sample $\sigma = (\sigma_L, \sigma_U)$ with which we run a train-and-test experiment.

3. We repeat the two steps above k times for each class $y_j \in Y$ and average the results across these $37 \times k$ train-and-test experiments.

We also run single-label multiclass classification experiments, using varying number of classes. For these experiments

1. given a desired number n of classes, we randomly choose n of our 37 RCV1-v2 classes, thus obtaining codeframe Y, with $|Y| = n$;

2. we generate two random vectors $\Pi_L = (\pi_1^L, ..., \pi_{|Y|}^L)$ and $\Pi_U = (\pi_1^U, ..., \pi_{|Y|}^U)$ of class priors, i.e., two vectors such that $0 \leq \pi_j^L, \pi_j^U \leq 1$ for each $1 \leq j \leq |Y|$ and such that $\sum_{j=1}^{|Y|} \pi_j^L = \sum_{j=1}^{|Y|} \pi_j^U = 1$;

3. we generate a training set σ_L (resp., a test set σ_U) by drawing $m_L = |\sigma_L|$ (resp., $m_U = |\sigma_U|$) different documents (with m_L and m_U two parameters to be fixed beforehand), where at each draw we pick with probability π_j^L (resp., π_j^U) a document belonging to class y_j. We thus obtain a sample $\sigma = (\sigma_L, \sigma_U)$ with which we run a train-and-test experiment;

4. we repeat the three steps above k times and average the results across these k train-and-test experiments.

In the experiments we run in this chapter we use $m_L = m_U = 1000$, and $k = 500$. The fact that, as previously specified, we only consider classes with at least 2000 positive examples allows us to use $m_L = m_U = 1000$, i.e., there would be enough positive training examples even if, in some of

[10] The method we use for generating each such vector is to pick two random real numbers in [0,1] and normalizing them so that they sum to 1. We have specified this since different methods to generate random vectors of class priors (for instance, picking a random number x in [0,1] and using $(x, (1 - x))$ as the vector) are possible, and may yield different results. The same method is also used in Step 2 of the analogous process for multiclass experiments that we are going to describe next, of course using vectors with dimensionality equal to the number of classes considered.

the k draws, π_j^L and π_j^U were both 1 for some y_j.[11] We run multiclass experiments for all values of $|Y| \in \{5, 10, 20, 37\}$.

Thanks to the use of randomly generated drawing probabilities, the class distributions of both the training set and the test set of each sample are random, each class distribution is equiprobable, and the value of PPS (as measured by NAE) between the training set and the test set of each sample we generate is also random. The set of samples that we generate with this method is, since k is large enough, fairly representative of the entire spectrum of shift values.

Note that this strategy for generating samples characterized by random values of PPS is radically different from the one adopted, for instance, in Esuli et al. (2018); Forman (2008). In these latter works there is no random component in picking class distributions or PPS values, and an equal number of samples is generated for all possible class distributions such that each class prior belongs to a finite set of values (e.g., $\{.00, .01, ..., .99, 1.00\}$). However, those works deal only with the binary case, where the number of all possible such class distributions is small. In the general multiclass case (i.e., when $|Y| > 2$) this number is much higher, since it grows exponentially with $|Y|$; therefore, even generating a single sample for all possible class distributions such that each class prior is in $\{.00, .01, ..., .99, 1.00\}$, would be prohibitive even for small numbers of $|Y|$. The random strategy we adopt in this chapter thus allows us to avoid this pitfall.

3.3.3 Representing text

We preprocess text by using stop word removal and no stemming. As the weighting criterion we use a version of the well-known *tfidf* method, expressed as

$$\textit{tfidf}(f, \mathbf{x}_i) = \log \#(f, \mathbf{x}_i) \times \log \frac{|L|}{|\mathbf{x} \in L : \#(f, \mathbf{x}) > 0|} \tag{3.11}$$

where $\#(f, \mathbf{x}_i)$ is the raw number of occurrences of feature f in document $\mathbf{x}_i$; weights are then normalized by means of cosine normalization.

3.3.4 Learners

In our experiments we use four different learners, i.e., support vector machines (SVMs), logistic regression (LR), multinomial naive Bayes (MNB), and random forests (RFs). For all of them we rely on the implementations available from the `scikit-learn` package.[12] For all of them we use the default parameters of the `scikit-learn` implementation, since the possible accuracy improvements resulting from a parameter optimization based on k-fold cross-validation would be obtained at the expense of a very large computational cost.[13] This possible accuracy improvement would bring about no evident benefit to our study, since the goal of this work is not squeezing every possible drop of accuracy from our classifiers, but comparing the pre-SLD results with the post-SLD results in the same experimental conditions. The default values are as follows:

- SVMs: we use soft-margin SVMs with linear kernel, L2 regularization with $C = 1$;

[11] Note that documents are drawn from RCV1-v2 in its entirety, disregarding the "traditional" split of RCV1-v2 into 23,149 training documents and 781,265 test documents.

[12] https://scikit-learn.org/stable/index.html

[13] No parameter has been optimized because it would have been too expensive to do it individually for each of the 500 samples per dataset mentioned in Section 3.3.2, and because doing it on just one of the 500 samples and using the obtained parameter values for the other 499 would have been of dubious utility.

- LR: we use L2 regularization with a regularization coefficient $C = 1$;
- MNB: We use Laplace smoothing, with $a = 1$ as the additive factor;
- RFs: we use 100 trees per forest, Gini impurity as the splitting function, no max depth, no pruning.

For each of these learners but SVMs we use two versions, one with post-calibration of the posteriors that the learner returns (CALIB), and the other without calibration (NOCALIB). SVMs are an exception because, as is well-known, the confidence scores they return are not probabilities, and the only way to have SVMs return probabilities in `scikit-learn` is to invoke a calibration routine; as a result, the only version of SVMs we experiment with is one with post-calibration.

We perform calibration using the method proposed by Platt (2000), sometimes known as "Platt scaling".[14] Given a confidence score $s(\mathbf{x}_i, y_j)$ produced by a classifier, either in the form of a non-probabilistic score or of a non-calibrated probability, we transform it into a calibrated probability $\Pr(y_j|\mathbf{x}_i)$ by applying the logistic transformation

$$\Pr(y_j|\mathbf{x}_i) = \frac{1}{1+\exp(\alpha \cdot s(\mathbf{x}_i, y_j) + \beta)} \tag{3.12}$$

where the parameters α and β are determined by fitting a maximum-likelihood model on a set of scores $S_{\text{Calib}} = \{s(\mathbf{x}, y_j)|\mathbf{x} \in Tr_{\text{Calib}}\}$ produced by the classifier on some training documents Tr_{Calib}. If the same training documents that are used to train the classifier are also used for calibration, overfitting may happen. Held-out documents may be used, but this requires additional labelled documents. To avoid overfitting without requiring held-out documents, Platt suggests to collect the set of scores S_{Calib} by performing cross-validation on the training documents. We have implemented this k-fold cross-validation procedure, performing 10-fold cross-validation on the training documents, using the same learning algorithm that is separately used on the entire training set in order to learn the actual classifier. We obtain the scores S^f_{Calib} for each validation fold $f \in [1, \ldots, 10]$ and then optimize the parameters of Equation 3.12 on the resulting set of scores $S_{\text{Calib}} = \bigcup_f S^f_{\text{Calib}}$. We then apply the optimized Equation 3.12 to the scores of the classifier trained on the entire training set; we refer to this process as the CALIB version of the learner.

3.4 Results

This section presents the results of our experiments. The code for reproducing them is available at `https://github.com/HLT-ISTI/SLD-reassessment`. At `https://hlt-isti.github.io/SLD-visualization/` we also make available a visualization tool that shows, for various combinations of (number of classes, sample, learner, class) from our experiments,

1. how the prior of the chosen class as estimated by SLD evolves as a function of the number of iterations;
2. as the values of the four evaluation metrics (as computed on this sample only) evolve as a function of the number of iterations.

[14] We have implemented Platt scaling ourselves since the version available from `scikit-learn` turns out to be not a faithful implementation of Platt's algorithm; see `https://github.com/scikit-learn/scikit-learn/issues/16145` for a discussion. The code of our implementation is available at `https://github.com/aesuli/scikit-learn/tree/platt`

Figure 3.1 shows sample plots as generated by our visualization tool.

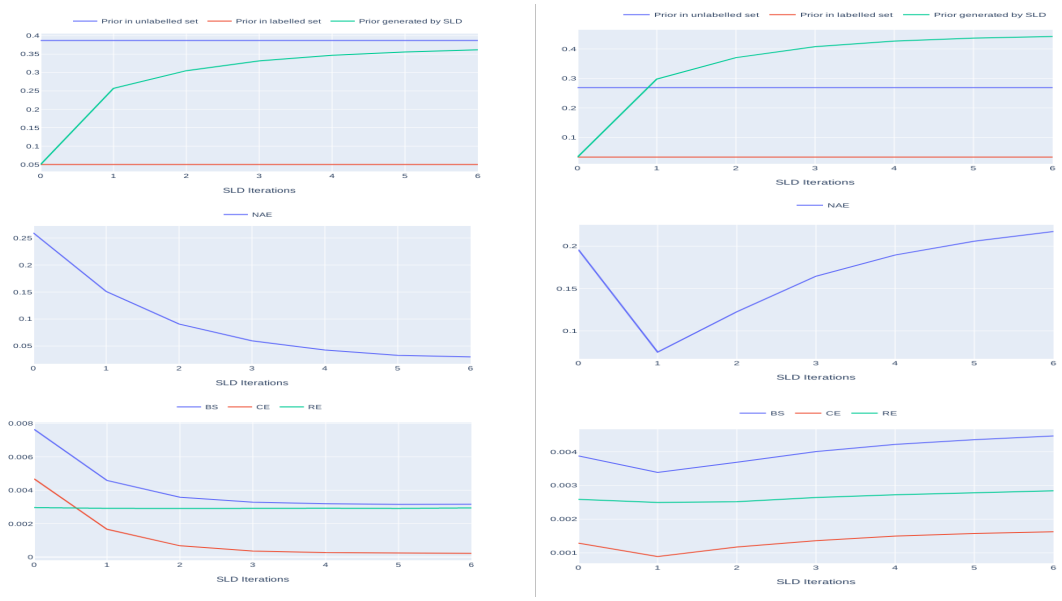


Figure 3.1: Plots from the visualization tool at `https://hlt-isti.github.io/SLD-visualization/`, showing the evolution, as a function of the number of SLD iterations, (top) of the prior of a chosen class as estimated by SLD, (center) of NAE, and (bottom) of BS, CE and RE. The plots on the left show a successful application of SLD (3 of the 4 metrics improve and the 4th stays constant), and the distance between the prior generated by SLD and the true prior in the unlabelled set U diminishes, while the plots on the right show an unsuccessful such application (all 4 metrics worsen, and the distance between the prior generated by SLD and the true prior in the unlabelled set U increases).

3.4.1 Results of binary classification experiments

The upper half of Table 3.1 reports, for our binary classification experiments, the values of NAE and of the isometric variants of BS, CE, RE, before ("Pre-SLD") and after ("Post-SLD") the application of SLD. In other words, Pre-SLD indicates the values computed directly on the outputs of the classifier, while Post-SLD indicates the values after these outputs have been updated by SLD. Since NAE, BS, CE, RE are all error measures, differences between Pre-SLD and Post-SLD are indicated in terms of *relative error reduction* $R_E = \frac{B-A}{B}$, where $E \in \{\text{NAE}, \text{BS}, \text{CE}, \text{RE}\}$ is the specific error measure, B and A are the Pre-SLD and Post-SLD values of E, respectively, and where the values of R_E are reported (for simplicity) as percentages instead of as fractions.[15] Note that

[15] In this table and in all the other tables in this chapter, some values of R_E might not appear to be completely justified; for instance, when the transition from a Pre-SLD value of .001 to a Post-SLD value of .000 is indicated to correspond to a value $R_E = +79.3\%$. Of course, this value of R_E derives from using the real Pre-SLD and Post-SLD values in much higher precision. We use the standard notation (e.g., .027) rather than the more precise E notation (e.g., 2.7E-3) for higher legibility.

			Priors			Posteriors								
			NAE			BS			CE			RE		
			Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction
Isometric	NoCalib	LR	.005	.013	-136.9%	.011	.011	-4.2%	.008	.006	+21.4%	.003	.005	-68.8%
		MNB	.116	.186	-60.3%	.020	.025	-26.0%	.013	.015	-15.6%	.007	.011	-44.4%
		RF	.016	.039	-142.1%	.010	.007	+26.2%	.006	.003	+46.9%	.003	.004	-12.1%
		Avg	.046	.079	-72.7%	.013	.015	-7.8%	.009	.008	+9.6%	.005	.006	-41.8%
	Calib	SVM	.005	.004	+21.5%	.003	.002	+31.7%	.001	.000	+79.3%	.002	.002	+8.6%
		LR	.008	.002	+78.2%	.004	.003	+29.1%	.001	.000	+75.8%	.002	.002	+7.6%
		MNB	.023	.014	+39.1%	.009	.006	+35.3%	.004	.001	+72.8%	.005	.005	+8.3%
		RF	.002	.002	+17.5%	.005	.003	+32.7%	.002	.000	+81.6%	.003	.003	+7.7%
		Avg	.009	.005	+43.5%	.005	.004	+33.0%	.002	.000	+76.1%	.003	.003	+8.1%
Isomerous	NoCalib	LR	.005	.013	-136.9%	.012	.012	-3.0%	.009	.009	-4.1%	.003	.003	+0.0%
		MNB	.116	.186	-60.3%	.022	.028	-26.5%	.017	.023	-35.3%	.005	.005	-0.0%
		RF	.016	.039	-142.1%	.011	.008	+28.0%	.007	.004	+44.4%	.004	.004	+0.0%
		Avg	.046	.079	-72.7%	.015	.016	-7.0%	.011	.012	-9.8%	.004	.004	+0.0%
	Calib	SVM	.005	.004	+21.5%	.004	.003	+23.7%	.001	.000	+90.1%	.003	.003	+0.0%
		LR	.008	.002	+78.2%	.004	.003	+22.8%	.001	.000	+86.5%	.003	.003	+0.0%
		MNB	.023	.014	+39.1%	.010	.007	+33.1%	.004	.001	+73.5%	.005	.005	-0.0%
		RF	.002	.002	+17.5%	.006	.004	+28.0%	.002	.000	+86.8%	.004	.004	+0.0%
		Avg	.009	.005	+43.5%	.006	.004	+28.4%	.002	.000	+80.3%	.004	.004	-0.0%

Table 3.1: Values of NAE, BS, CE, RE, before and after the application of SLD, for binary classification experiments.

for R_E a positive value indicates an improvement (i.e., that SLD had a beneficial effect) while a negative value indicates a deterioration. The rows of the table each correspond to one of the learners of Section 3.3.4, grouped into learners with post-calibration of the posteriors that the learner returns (Calib) and ones without such calibration (NoCalib). As indicated in Section 3.3.2, every row of this table is the result of 37×500=18,500 train-and-test runs; given that each of the 7 rows accounts for a different learner, this is a total of 18,500×7=129,500 train-and-test runs.

There are a number of observations that can be derived from the top part of Table 3.1:

- In terms of the quality of the estimated priors (as measured by NAE), there is a very substantial difference between the performance of non-calibrated learners and that of calibrated learners: for the former, the application of SLD brings about an extremely large deterioration (an average of 72.7% across all tested learners), while it brings about a very good improvement (an average of 43.5% across all tested learners) for the latter.[16] This adds to the fact that calibrated learners have, on average, a much better NAE right from the start (.009, instead of

[16]This confirms an observation of Saerens et al. (2002), according to whom "In order to obtain good a priori probability estimates [by means of SLD], it is necessary that the a posteriori probabilities relative to the training set are reasonably well approximated (i.e., sufficiently well estimated by the model)".

.046 for the non-calibrated ones); this means that *calibrating one's learner is a win-win move*, given that it brings about much better posterior probabilities *and* that these posteriors have much larger margins of improvement by means of the application of SLD.[17]

- The very large magnitude of these improvements / deteriorations is not mirrored by analogous magnitudes when it comes to the quality of the posteriors. It is still true that deteriorations are observed in the case of non-calibrated classifiers (7.8% on average) and improvements are instead observed for the calibrated classifiers (33.0% on average), but the magnitudes of these variations are smaller.

- SLD seems to have a much more beneficial effect in terms of calibration than in terms of refinement; in fact improvements in BS, when present, are largely the responsibility of CE, while deteriorations in BS, when present, are largely the responsibility of RE.

- While there are (even substantial) *quantitative* differences among learners belonging to the same category (non-calibrated or calibrated), there are very few *qualitative* differences, i.e., when a learner exhibits a deterioration in one of the measures, all other learners (with few exceptions) also exhibit a deterioration for the same measure. This seems to indicate that the results derive from inherent properties of the SLD algorithm, rather than from peculiarities of the individual learning algorithms.

The lower half of Table 3.1 presents the analogous results for the isomerous variants of BS, CE, RE. (The results for NAE are the same as in the upper half, since the distinction isometric/isomerous does not apply to NAE.) The observations that can be made by looking at the lower half the table are essentially the same as those derived from the upper half, since the results are qualitatively similar. There is one important difference, though, i.e., the fact that, when measured by means of the isomerous variant, RE is always 0 or very close to 0, which is far from being the case when using isometric RE. That this should be so is an obvious consequence of the definition of RE, as from Equation 3.9. In fact, since all bins are equally populated, it is clear from Equation 3.9 that RE only depends, for all bins β_{kj} ($1 \leq k \leq b$) and for all classes $y_j \in Y$, on the fraction $\rho(y_j, \beta_{kj})$ of documents in the bin that belong to the class. However, for all bins, that fraction is the same in the Pre-SLD and Post-SLD distributions, because, as observed in Section 2.4, SLD is just a *rescaling* algorithm, that multiplies all the posteriors for a given class for the same constant but does not change the composition of the bins. That RE is not 0 when using the isometric variant is thus due to the fact that rescaling changes the compositions of the bins; for instance, a document that was in the bin corresponding to the [.9, 1.0] interval before the application of SLD, after SLD has been applied might be in the [.8, .9) bin if SLD has multiplied all the posteriors for that class by a factor smaller than 1. In this case, rescaling not only changes the composition of the bins, but also changes the number of documents they contain, thus potentially generating also very sparse bins.

Interestingly, the fact that SLD could not reduce RE is reminiscent of an observation by DeGroot, Fienberg (1983):

[17] Niculescu-Mizil, Caruana (2005b) state that "For learning methods that make well calibrated predictions such as neural nets, bagged trees, and logistic regression, neither Platt Scaling nor Isotonic Regression yields much improvement in performance even when the calibration set is very large. With these methods calibration is not beneficial, and actually hurts performance when the calibration sets are small." Our large-scale experimentation indicates that, while this might be true in the absence of PPS, when PPS is present any calibrated learner works better than its non-calibrated counterpart. In fact, note that the Pre-SLD values of NAE, BS and CE for the calibrated learners are always substantially better than the values of the corresponding non-calibrated learners, and this also includes logistic regression, a learner that is known to return well calibrated probabilities.

> We then study the question of when an observer can use a forecaster's predictions to obtain a better score than the forecaster himself, and show that such an improvement can be achieved by the observer essentially if and only if the forecaster is not well calibrated.

Here, the forecaster is the classifier and the observer is SLD, who tries to obtain a better score (in terms of Brier score) than the classifier by "piggybacking" on the classifier's predictions. DeGroot, Fienberg (1983) state that what SLD can at most hope for, is to improve on the classifier's *calibration* error, but not on its *refinement* error. This shows that SLD is, in essence, a *re-calibration algorithm*, i.e., an algorithm for re-calibrating the posterior probabilities of documents belonging to an unlabelled set U, where these posteriors have been returned by a classifier already calibrated on a training set L, and where the re-calibration is made necessary by the fact that a prior probability shift between L and U has occurred.

3.4.2 Results of multiclass classification experiments

			Priors			Posteriors								
			NAE			BS			CE			RE		
			Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction
Isometric	NoCalib	LR	.007	.003	+50.1%	.005	.007	-33.9%	.004	.004	-25.2%	.002	.003	-49.3%
		MNB	.022	.030	-36.7%	.009	.012	-36.9%	.005	.006	-25.3%	.004	.006	-49.6%
		RF	.014	.036	-154.9%	.005	.005	+11.0%	.003	.002	+31.9%	.002	.003	-22.4%
		Avg	.014	.023	-61.6%	.007	.008	-22.9%	.004	.004	-8.6%	.003	.004	-42.7%
	Calib	SVM	.009	.006	+37.5%	.003	.003	+2.6%	.001	.001	-2.5%	.002	.002	+5.0%
		LR	.004	.009	-133.0%	.002	.002	-31.6%	.001	.001	-48.2%	.001	.001	-22.4%
		MNB	.015	.017	-15.1%	.005	.004	+21.7%	.002	.001	+40.1%	.003	.003	+10.7%
		RF	.005	.005	-4.8%	.003	.002	+24.4%	.001	.000	+53.5%	.002	.002	+9.7%
		Avg	.008	.009	-12.7%	.003	.003	+9.5%	.001	.001	+19.9%	.002	.002	+3.8%
Isomerous	NoCalib	LR	.007	.003	+50.1%	.006	.008	-28.2%	.004	.006	-42.5%	.003	.003	-5.7%
		MNB	.022	.030	-36.7%	.010	.014	-35.4%	.006	.009	-51.7%	.004	.004	-10.7%
		RF	.014	.036	-154.9%	.006	.006	+12.8%	.004	.003	+29.2%	.003	.003	-6.9%
		Avg	.014	.023	-61.6%	.008	.009	-19.8%	.005	.006	-27.8%	.003	.003	-8.2%
	Calib	SVM	.009	.006	+37.5%	.004	.004	-0.5%	.001	.001	+4.2%	.003	.003	-2.0%
		LR	.004	.009	-133.0%	.003	.004	-20.2%	.001	.001	-88.9%	.002	.002	-1.8%
		MNB	.015	.017	-15.1%	.006	.005	+18.4%	.002	.001	+43.8%	.004	.003	+4.7%
		RF	.005	.005	-4.8%	.004	.003	+16.5%	.001	.000	+60.6%	.003	.003	+2.7%
		Avg	.008	.009	-12.7%	.004	.004	+6.4%	.001	.001	+19.7%	.003	.003	+1.3%

Table 3.2: As Table 3.1, but for multiclass classification (5 classes).

Tables 3.2 to 3.5 report the results of our experiments on multiclass classification. As indicated in Section 3.3.2, we run multiclass experiments with varying number of classes, starting from $|Y| = 5$

			Priors			Posteriors								
			NAE			BS			CE			RE		
			Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction
Isometric	NoCalib	LR	.014	.030	-107.5%	.005	.007	-61.0%	.002	.004	-55.1%	.002	.004	-67.4%
		MNB	.014	.035	-143.5%	.006	.009	-51.8%	.002	.004	-59.9%	.004	.005	-46.5%
		RF	.018	.068	-281.6%	.004	.004	+0.3%	.002	.001	+26.3%	.002	.003	-24.7%
		Avg	.016	.044	-185.4%	.005	.007	-40.2%	.002	.003	-32.7%	.003	.004	-46.7%
	Calib	SVM	.015	.022	-41.3%	.002	.004	-43.1%	.001	.001	-97.5%	.002	.002	-21.9%
		LR	.018	.026	-45.1%	.002	.004	-93.6%	.001	.002	-173.3%	.001	.002	-58.7%
		MNB	.017	.020	-19.3%	.003	.004	-18.1%	.001	.001	-38.0%	.002	.003	-9.9%
		RF	.008	.020	-143.7%	.002	.003	-22.4%	.001	.001	-52.3%	.002	.002	-12.3%
		Avg	.015	.022	-50.4%	.003	.003	-39.6%	.001	.001	-83.3%	.002	.002	-22.4%
Isomerous	NoCalib	LR	.014	.030	-107.5%	.005	.008	-52.0%	.003	.006	-93.7%	.003	.003	-4.8%
		MNB	.014	.035	-143.5%	.007	.010	-48.6%	.003	.006	-87.0%	.004	.004	-14.2%
		RF	.018	.068	-281.6%	.005	.005	+3.0%	.002	.002	+16.5%	.003	.003	-6.6%
		Avg	.016	.044	-185.4%	.006	.008	-34.9%	.003	.004	-63.4%	.003	.003	-9.1%
	Calib	SVM	.015	.022	-41.3%	.003	.005	-31.5%	.001	.002	-158.8%	.003	.003	-1.8%
		LR	.018	.026	-45.1%	.003	.005	-58.8%	.000	.002	-319.4%	.002	.003	-6.1%
		MNB	.017	.020	-19.3%	.004	.005	-14.0%	.001	.002	-65.0%	.003	.003	+1.4%
		RF	.008	.020	-143.7%	.003	.004	-16.1%	.001	.001	-102.1%	.003	.003	+0.3%
		Avg	.015	.022	-50.4%	.003	.004	-28.2%	.001	.002	-142.3%	.003	.003	-1.3%

Table 3.3: As Table 3.2, but with 10 classes.

classes (Table 3.2) and moving up to $|Y| = 10$ (Table 3.3), $|Y| = 20$ (Table 3.4), and $|Y| = 37$ (Table 3.5), which is the total number of classes in our dataset. For $|Y| \in \{5, 10, 20\}$, the classes are randomly sampled from the entire set of 37 RCV1-v2 classes. As indicated in Section 3.3.2, every row of these 4 tables is the result of 500 train-and-test runs; given that each of the 7 rows accounts for a different learner, this is a total of 4×500×7=14,000 train-and-test runs.

There are several observations we can make by looking at these tables:

- The main fact that emerges is that *all quality indicators of SLD* (i.e., the values of error reduction, for each of the four error measures we consider) *drastically deteriorate when* $|Y|$ *grows*, for all learners, calibrated or not. Table 3.5, that reports results for $|Y| = 37$, indicates disastrous performance on the part of SLD on all counts.

- Concerning SLD's impact on the priors, while the binary experiments had indicated a very positive impact (at least: for the calibrated learners), the multiclass experiments indicate a negative impact for $|Y| = 5$ (12.7% average deterioration across all calibrated learners) and an even more negative impact for $|Y| \in \{10, 20, 37\}$, with the average deterioration across all calibrated learners reaching up to 251.0% for $|Y| = 37$.

- Concerning SLD's impact on the posteriors, while the binary experiments had indicated a

			Priors			Posteriors								
			NAE			BS			CE			RE		
			Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction
Isometric	NoCalib	LR	.017	.081	-382.2%	.003	.006	-72.3%	.001	.002	-95.6%	.002	.003	-59.2%
		MNB	.021	.075	-260.7%	.004	.006	-64.6%	.001	.002	-130.9%	.003	.004	-39.0%
		RF	.025	.118	-379.4%	.003	.003	-11.0%	.001	.001	+17.0%	.002	.002	-27.4%
		Avg	.021	.091	-340.5%	.003	.005	-52.2%	.001	.002	-71.3%	.002	.003	-42.6%
	Calib	SVM	.030	.062	-110.5%	.002	.004	-86.3%	.001	.001	-175.3%	.002	.002	-54.4%
		LR	.024	.047	-96.7%	.002	.003	-114.5%	.000	.001	-230.7%	.001	.002	-74.7%
		MNB	.015	.047	-205.2%	.002	.004	-85.2%	.000	.002	-234.7%	.002	.002	-38.4%
		RF	.015	.040	-162.8%	.002	.003	-87.2%	.000	.001	-265.7%	.001	.002	-40.7%
		Avg	.021	.049	-133.4%	.002	.004	-92.3%	.000	.001	-222.4%	.001	.002	-50.9%
Isomerous	NoCalib	LR	.017	.081	-382.2%	.004	.006	-61.8%	.002	.004	-149.6%	.002	.002	-3.7%
		MNB	.021	.075	-260.7%	.004	.007	-58.9%	.001	.003	-144.7%	.003	.003	-13.6%
		RF	.025	.118	-379.4%	.003	.004	-6.1%	.001	.001	-9.8%	.003	.003	-4.8%
		Avg	.021	.091	-340.5%	.004	.006	-44.1%	.001	.003	-115.7%	.003	.003	-7.6%
	Calib	SVM	.030	.062	-110.5%	.003	.005	-57.3%	.000	.002	-395.2%	.002	.003	-1.4%
		LR	.024	.047	-96.7%	.003	.004	-62.9%	.000	.002	-502.6%	.002	.002	-4.3%
		MNB	.015	.047	-205.2%	.003	.005	-56.2%	.000	.002	-407.9%	.003	.003	-0.6%
		RF	.015	.040	-162.8%	.003	.004	-52.1%	.000	.002	-491.1%	.002	.002	-1.0%
		Avg	.021	.049	-133.4%	.003	.004	-57.0%	.000	.002	-441.6%	.002	.002	-1.8%

Table 3.4: As Table 3.2, but with 20 classes.

very positive impact (at least: for the calibrated learners), the multiclass experiments indicate that this impact is still mildly positive for $|Y| = 5$ but becomes negative for $|Y| = 10$ and deteriorates even more for $|Y| \in \{20, 37\}$. For instance, BS in the isomerous variant has, thanks to SLD, an average improvement across the calibrated learners by 28.0% for $|Y| = 2$ and 6.4% for $|Y| = 5$, but this improvement becomes a deterioration for $|Y| = 10$ (28.2%) and for $|Y| \in \{20, 37\}$ (e.g., 72.2% for $|Y| = 37$). This trend is even more marked for CE, and indicates an improvement for $|Y| = 2$ (80.3%, average across the calibrated learners) and $|Y| = 5$ (19.7%) but a deterioration for higher values of $|Y|$, with the amount of deterioration reaching up to 937.2% for $|Y| = 37$.

3.4.3 Analyzing the results by amount of shift

In this section we analyze the relations between error and PPS. Our goal is that of highlighting, in the results of the experiments discussed in Sections 3.4.1 and 3.4.2, any noteworthy correlation between error reduction, for any of our four measures, and PPS.

In our analysis of the results, we have not been able to detect any significant correlation between NAE and PPS, or between RE and PPS. As a result, from here onwards we only concentrate on discussing BS and CE. Figure 3.2 plots the values of relative error reduction for the BS and CE

			Priors			Posteriors								
			NAE			BS			CE			RE		
			Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction	Pre-SLD	Post-SLD	Error Reduction
Isometric	NoCalib	LR	.014	.137	-913.5%	.002	.003	-50.5%	.000	.001	-149.9%	.002	.002	-25.6%
		MNB	.022	.115	-411.6%	.002	.004	-67.8%	.000	.002	-326.1%	.002	.002	-19.2%
		RF	.028	.159	-458.5%	.002	.002	-19.6%	.000	.000	+3.7%	.001	.002	-28.4%
		Avg	.021	.137	-537.7%	.002	.003	-47.9%	.000	.001	-140.3%	.002	.002	-23.9%
	Calib	SVM	.031	.091	-193.1%	.002	.003	-115.1%	.000	.001	-278.9%	.001	.002	-67.9%
		LR	.026	.067	-160.4%	.001	.003	-112.8%	.000	.001	-269.0%	.001	.002	-69.5%
		MNB	.015	.071	-368.2%	.001	.003	-135.0%	.000	.002	-447.1%	.001	.002	-56.9%
		RF	.016	.080	-398.3%	.001	.003	-138.2%	.000	.001	-501.4%	.001	.002	-62.1%
		Avg	.022	.077	-251.0%	.001	.003	-125.2%	.000	.001	-363.9%	.001	.002	-64.0%
Isomerous	NoCalib	LR	.014	.137	-913.5%	.002	.004	-45.7%	.001	.002	-184.1%	.002	.002	-1.7%
		MNB	.022	.115	-411.6%	.003	.004	-64.3%	.001	.002	-271.6%	.002	.002	-7.5%
		RF	.028	.159	-458.5%	.002	.003	-12.8%	.000	.001	-83.0%	.002	.002	-2.4%
		Avg	.021	.137	-537.7%	.002	.003	-41.9%	.000	.001	-196.4%	.002	.002	-3.9%
	Calib	SVM	.031	.091	-193.1%	.002	.004	-74.0%	.000	.002	-803.6%	.002	.002	-0.7%
		LR	.026	.067	-160.4%	.002	.003	-57.9%	.000	.001	-790.7%	.002	.002	-1.1%
		MNB	.015	.071	-368.2%	.002	.004	-79.7%	.000	.002	-1009.1%	.002	.002	-0.9%
		RF	.016	.080	-398.3%	.002	.004	-76.8%	.000	.002	-1206.9%	.002	.002	-0.9%
		Avg	.022	.077	-251.0%	.002	.004	-72.2%	.000	.002	-937.2%	.002	.002	-0.9%

Table 3.5: As Table 3.2, but with 37 classes.

measures (we here use the isomerous variants; the isometric variants return similar results) for the same experiments as discussed in Sections 3.4.1 and 3.4.2, for each of our 4 calibrated classifiers,[18] but with the samples binned into four quartiles according to how much PPS between the training set and test set the sample exhibits. The 1st quartile contains the samples characterized by the lowest amounts of PPS, and the 4th contains the samples characterized by the highest such amounts. The actual values of PPS (expressed in terms of NAE) that characterize the samples in each quartile are reported in Table 3.6. Each result reported in the plots is the average across all samples that belong to the bin.

A clear pattern emerges from the analysis of BS and CE values: for both measures, for both the binary and multiclass cases, and for all numbers of classes considered in the multiclass experiments ($|Y| \in \{5, 10, 20, 37\}$), *performance tends to improve monotonically with the amount of PPS*. (Exceptions do exist for individual classifiers, but the average values across the 4 classifiers exhibits strict monotonicity). This happens both for the cases (binary case + multiclass case with $|Y| = 5$) in which SLD has a positive impact (i.e., error diminishes as a result of its application), and for the cases (multiclass case with $|Y| > 5$) in which the impact of SLD is negative (i.e., error increases as a result of its application); in the former cases the magnitude of error reduction increases with the

[18] We omit discussing the non-calibrated classifiers since the experiments of Sections 3.4.1 and 3.4.2 have clearly indicated that SLD requires, in order to perform well, calibrated classifiers.

	2 classes		5 classes		10 classes		20 classes		37 classes	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
1st quartile	.000	.168	.063	.253	.116	.274	.177	.297	.224	.311
2nd quartile	.168	.343	.254	.334	.274	.329	.297	.335	.311	.342
3rd quartile	.343	.557	.334	.413	.329	.387	.336	.373	.342	.372
4th quartile	.557	.993	.414	.746	.387	.644	.374	.534	.372	.483

Table 3.6: Values of PPS (expressed in terms of NAE) for the samples in each of the four quartiles in which all samples are binned; for each quartile we indicate minimum shift and maximum shift of the samples the quartiles actually contain.

increase in shift, while in the latter cases the magnitude of error amplification diminishes with the increase in shift. Case $|Y| = 5$ seems the threshold here, with SLD yielding a decrease in error for the two quartiles representing low shift and an increase in error for the two quartiles representing high shift.

Together with the analyses presented in Sections 3.4.1 and 3.4.2, this observation suggests that SLD should be used to improve the quality of the prior probability estimates and of the posterior probabilities, only (a) when the classifier has been calibrated, *and* (b) the number of classes in the codeframe Y is low (say, $|Y| \leq 5$), *and* (c) when the amount of distribution shift is high enough.[19]

3.4.4 Analyzing the distributions produced by SLD

In the previous sections we have evaluated, among other things, the impact of SLD on the difference between the predicted class distribution and the true class distribution, by using the NAE measure. In this section we instead look at the impact of SLD on two intrinsic characteristics of class distributions, i.e., their entropy and their shape. In order to do so we compare, for a given train-and-test run, the four class distributions involved: (a) the true class distribution of L, (b) the true class distribution of U, (c) the class distribution of U predicted by the classifier (which SLD receives as input), and (d) the class distribution of U returned by SLD. This allows us to better understand the impact of SLD, and the reasons behind some of the patterns highlighted in Sections 3.4.1 through 3.4.3.

Average entropy of class distributions

For each of the 129,500+14,000=143,500 train-and-test runs we have discussed in Sections 3.4.1 and 3.4.2, we measure the entropy

$$E(Y) = -\sum_{i=1}^{|Y|} \Pr(y_i) \log_{|Y|} \Pr(y_i) \tag{3.13}$$

of the four class distributions (a) to (d) mentioned in the previous paragraph. In each case we set the base of the logarithm to the number $|Y|$ of classes of the distribution being observed, so that entropy values always range between 0 and 1. A low entropy value means that most of the

[19] Statistical tests are indeed available that allow to detect how much PPS there is between the training documents and the unlabelled documents; one such test (a likelihood ratio test) is presented in Saerens et al. (2002, §3).

documents in the sample belong to one or few classes, while a high entropy value means that the documents are spread fairly evenly across the entire set of classes.

Table 3.7 shows the average value of the entropy of the four class distributions (a) to (d) (which in this and in the following tables will be indicated as L, U, Pre-SLD, and Post-SLD, respectively) across all the 143,500 train-and-test runs.

	L	U	Pre-SLD	Post-SLD
Entropy	.902	.902	.906	.638

Table 3.7: Values of the entropy of the four class distributions, averaged across all train-and-test runs.

From this table we can observe that the values for L and U are the same. This is intuitive, since, even though the training set and the test set of a given sample have in general two different class distributions, the sampling method for generating training sets and test sets is the same and the pool of documents from which to sample is the same, so training sets and test sets will exhibit, on average, the same class distribution. In the following we will not report the entropy values for L, since those for U are always practically identical.

The average entropy value of Pre-SLD class distributions is slightly higher than those for L and U, but not substantially so. However, what immediately jumps to the eye is that the average entropy value for Post-SLD is much lower than the values for L, U, and Pre-SLD. Similar distributions exhibit similar entropy values, so a difference in entropy values is a clear indicator of a dissimilarity between the two observed distributions. The sharp difference between the Pre-SLD and Post-SLD average entropy values thus unequivocally indicates that *SLD substantially alters the Pre-SLD class distribution*, and the sharp difference between the U and Post-SLD values indicates that *this alteration is detrimental*. Since a high entropy value indicates a highly uniform distribution, the above results indicate that SLD has a tendency to sharply diminish this uniformity, and label most of the documents with one or few classes.

Table 3.8 reports again entropy values of class distributions, but averaged across all runs characterized by the same number of classes in the codeframe. An analysis of this table shows that

# of classes	U	Pre-SLD	Post-SLD
2	.821	.819	.704
5	.894	.891	.787
10	.919	.917	.721
20	.935	.935	.574
37	.943	.946	.405

Table 3.8: Values of the entropy of the four class distributions, averaged across all train-and-test runs with the same number of classes in the codeframe.

values for U tend to increase as the number $|Y|$ of classes in the codeframe increases. This is due to the sampling method, that generates prior probabilities with mean equal to $\frac{1}{|Y|}$ and variance equal to $\frac{1}{|Y|^2}$ (as will also be evident from Figures 3.4 to 3.7). The values for Pre-SLD follow the same trend as values for U, but the values for Post-SLD have an almost *opposite* trend: as the number

$|Y|$ of classes in the codeframe increases, SLD decreases the uniformity of class distributions.

Table 3.9 reports again entropy values of class distributions, but averaged across all runs that use the same learning algorithm. The application of SLD following the use of the No-Calib learners

		U	Pre-SLD	Post-SLD
No-Calib	LR	.902	.925	.501
	MNB	.902	.802	.395
	RF	.902	.923	.751
	Avg	.902	.883	.549
Calib	SVM	.902	.914	.698
	LR	.902	.919	.719
	MNB	.902	.906	.687
	RF	.902	.921	.719
	Avg	.902	.915	.706

Table 3.9: Values of the entropy of the four class distributions, averaged across all train-and-test runs obtained by means of the same learning algorithm.

brings about an even stronger divergence between the U values and the Post-SLD values than the corresponding Calib versions, thus confirming that non-calibrated classifiers are not fit for use with SLD. The application of SLD drastically reduces the average entropy for all learners, thus indicating that the decrease in uniformity of the distributions is less related to the chosen learning algorithm than to the number $|Y|$ of classes in the codeframe (see also Section 3.4.4).

Table 3.10 shows the average entropy values of the class distributions for all the possible combinations of number of classes and learners. From this table we can identify the very few cases in which the U class distributions have an average entropy value closer to the Post-SLD value than to the Pre-SLD value: this happens only for $|Y| = 2$ with calibrated SVMs, MNB, and RF.

Histogram-based representations of class distributions

In this section we display and comment on histograms that indicate how class prevalences are distributed in U, Pre-SLD, and Post-SLD class distributions resulting from the use of specific learners and with specific numbers of classes. Figure 3.3 does this for $|Y| = 2$, i.e., the binary classification case. As an example, the histogram in its left bottom subfigure ("Pre-SLD - Random Forests") shows that, if we pool together all the 37×500=18,500 train-and-test runs where RF was used as the learner, the results returned by the classifier (i.e., Pre-SLD) are such that a high number of classes have a prevalence of about 50% (i.e., $\Pr(y) = .5$), a slightly lower number of classes have a prevalence of 40%, ..., and a very small number of classes have a prevalence of 0%. Every subfigure of Figure 3.3 is, of course, bilaterally symmetric, since we are in the binary case, in which $\Pr(y) = \alpha$ entails $\Pr(\overline{y}) = (1 - \alpha)$. The top row of the figure (orange colour) refers to the U class distributions (the left and right histograms are the same); the other histograms in the left column refer to Pre-SLD class distributions, one for each of the 7 learning algorithms, while the other histograms in the right column refer to Post-SLD class distributions for the same algorithms. Figures 3.4 to 3.7 do the same for $|Y| \in \{5, 10, 20, 37\}$; these histograms are, of course, not bilaterally symmetric.[20]

[20]Note that, for higher legibility, the X axis displays a shorter interval than [0,1] when there are no classes with prevalence outside that interval.

# of classes		No-Calib			Calib			
		LR	MNB	RF	SVM	LR	MNB	RF
2	U	.821	.821	.821	.821	.821	.821	.821
	Pre-SLD	.866	.631	.868	.840	.828	.846	.851
	Post-SLD	.577	.409	.693	.815	.799	.820	.813
5	U	.894	.894	.894	.894	.894	.894	.894
	Pre-SLD	.921	.772	.922	.905	.913	.896	.909
	Post-SLD	.708	.567	.804	.858	.851	.854	.869
10	U	.919	.919	.919	.919	.919	.919	.919
	Pre-SLD	.939	.821	.934	.931	.940	.914	.938
	Post-SLD	.552	.479	.789	.809	.780	.798	.842
20	U	.935	.935	.935	.935	.935	.935	.935
	Pre-SLD	.947	.874	.943	.946	.953	.932	.951
	Post-SLD	.351	.321	.762	.635	.657	.614	.678
37	U	.943	.943	.943	.943	.943	.943	.943
	Pre-SLD	.953	.912	.949	.951	.959	.942	.958
	Post-SLD	.318	.198	.705	.370	.507	.346	.393

Table 3.10: Values of the entropy of the four class distributions, averaged across all train-and-test runs with the same number of classes *and* obtained by means of the same learning algorithm.

Figure 3.3 shows that all the methods produce class prevalences that are distributed more uniformly than the true ones, i.e., many Pre-SLD or Post-SLD distributions generate many class prevalences with very low or very high values. What is more important, though, is that for each learning method the difference between the U histogram and the Post-SLD histogram is larger than the difference between the U histogram and the Pre-SLD histogram; in other words, this confirms that SLD alters the Pre-SLD class distribution, and that this alteration is detrimental. However, what we learn from these histograms, and that we had not learned from the entropy study of the previous section, is *how* SLD alters this distribution: it does so by generating fewer class priors with mid values, i.e., close to 50%, and more class priors with extreme values, i.e., close to 0% or 1% (to see this better, note that the Y axes of the left subfigure and the right subfigure are often not on the same scale).

It is evident from Figure 3.3 that SLD's impact in altering the distribution is substantial for each of the four calibrated learners (4th to 8th rows), and it is even more for the non-calibrated ones (2nd to 4th rows). When SLD is run on the posteriors generated by these latter learners, all class priors except 0 and 1 become much more frequent, and class priors equal to 0 and 1 increase dramatically with respect to the Pre-SLD case.

In Figures 3.4 to 3.7, which represent the multiclass case with $|Y| \in \{5, 10, 20, 37\}$, these trends are increasingly evident, and the deterioration introduced by SLD reaches disastrous levels for $|Y| = 37$. Post-SLD average class distribution become increasingly skewed when $|Y|$ grows, and this concerns both calibrated and non-calibrated learners (although the latter are impacted to a much higher degree). While for the $|Y| = 2$ case class priors equal to 0 and class priors equal to 1 were both prevalent, in the multiclass cases class priors equal to 1 are practically absent and, as $|Y|$ grows, the histogram becomes increasingly skewed and class priors equal to 0 become the overwhelming majority. Overall, what all these histograms show, aligns very well with our

experimental findings of Sections 3.4.1 and 3.4.2, i.e., with the facts that SLD works better with calibrated than with non-calibrated classifiers, and with the fact that it works better for small values of $|Y|$ and (much) worse for high values of $|Y|$. They also show something more, i.e., that the reason of the bad behaviour is the fact that SLD has, especially when $|Y|$ is high and/or when classifiers are non-calibrated, a tendency to return many class priors equal to 0 and few class priors different from 0.

The fact that the SLD algorithm can bring to an extremisation of the posterior and prior probabilities, e.g., that most class priors are 0, will be central to most analyses of the algorithm in this thesis. As a matter of fact, this phenomenon was first noticed by Du Plessis, Sugiyama (2014), who showed (Du Plessis, Sugiyama, 2014, §3.3) that SLD is indeed solving a convex problem, but that it is not guaranteed to converge to a unique optimal value. Rather, the authors show that SLD can get "stuck" in a degenerate fixed point. That said, notice that the extremisation phenomenon which we will see and discuss in Chapters 4 and 5 is mainly due to the active learning policy which builds the training set, rather than to the phenomenon discussed here.

3.4.5 On the speed of convergence of SLD

As indicated in Section 2.4, in our experiments we stop SLD when we have reached either convergence (which we take to mean that $\text{AE}(\hat{p}_U^{(s-1)}, \hat{p}_U^{(s)}) < 10^{-6}$) or the maximum number of iterations (that we set to 1000). Table 3.11 reports, for each learner and for each number $|Y|$ of classes,

		2 classes		5 classes		10 classes		20 classes		37 classes	
		#	%	#	%	#	%	#	%	#	%
No-Calib	LR	60.86	0.18%	138.92	0.20%	476.69	12.40%	939.06	77.60%	999.62	99.80%
	MNB	29.27	0.00%	53.74	0.00%	136.46	0.60%	296.92	5.80%	469.28	14.00%
	RF	32.38	0.01%	83.65	0.20%	223.70	1.40%	446.88	5.80%	636.25	14.80%
	Avg	40.84	0.06%	92.10	0.13%	278.95	4.80%	560.95	29.73%	701.72	42.87%
Calib	SVM	9.29	0.00%	29.02	0.00%	109.75	0.00%	288.73	0.80%	353.37	1.40%
	LR	9.39	0.00%	27.68	0.00%	95.87	0.00%	249.06	1.00%	368.32	4.00%
	MNB	35.69	0.82%	58.95	0.40%	162.36	1.00%	308.40	4.20%	507.15	20.60%
	RF	12.66	0.00%	26.85	0.00%	86.00	0.00%	223.57	1.20%	432.19	11.00%
	Avg	16.76	0.20%	35.62	0.10%	113.49	0.25%	267.44	1.80%	415.26	9.25%

Table 3.11: Average number of iterations needed to reach convergence ("#") and percentage of cases in which convergence was not reached ("%") for all combinations of learner and number $|Y|$ of classes.

- the average number of iterations (column "#") SLD required to reach convergence (when convergence was actually reached – the value 1000 is used whan convergence was not reached), where the average is computed across all the train-and-test runs we have performed;
- the percentage of cases (column "%") in which convergence was not reached, and processing had to be stopped after 1000 iterations.

There are three conclusions that can be reached from this table, i.e.,

1. For a given number of classes, convergence tends to be quicker when the Pre-SLD posteriors have been obtained by calibrated learners; this is always true for LR and RF, although it is

always false for MNB. The difference between the two versions (non-calibrated and calibrated) of LR is somehow surprising, since LR is often presented as an algorithm that naturally returns calibrated probabilities, i.e., a classifier which does not need post-calibration; our results throughout this chapter instead show that post-calibration is beneficial for LR too.

2. For a given learner, the number of iterations required to reach convergence grows monotonically with the number $|Y|$ of classes considered.

3. For a given learner, the percentage of cases in which convergence is not reached grows monotonically with the number $|Y|$ of classes considered.

These findings constitute yet another argument in favour of calibrated learners, and yet another reason why the use of SLD should be contemplated only when the number $|Y|$ of classes is small.

3.5 What kind of dataset shift do we simulate in our experiments?

In Section 3.3.2 we stated that, in extracting from RCV1-v2 a number of samples with training sets and test sets characterized by random class distributions, our goal is to test the SLD algorithm on a variety of PPS values (see Section 2.1.1). But what kind of dataset shift are we simulating, exactly?

If our dataset is from a $X \rightarrow Y$ problem, we are certainly simulating covariate shift but *not* concept shift; in fact, we are selectively removing documents (which means that $\Pr(\mathbf{x})$ changes) but we are not making the causal relationship between X and Y change (which means that $\Pr(y|\mathbf{x})$ does not change), since the documents that are not removed still have the same class label. Conversely, if our dataset is from a $Y \rightarrow X$ problem, we are simulating prior probability shift, because by selectively removing documents we are making $\Pr(y)$ change.

So, what we are simulating with the sampling strategy of Section 3.3.2 is covariate shift and/or prior probability shift, but not concept shift.

There are two reasons for this:

- While a strategy that also simulates concept shift might have been better, since it would have allowed us to test the SLD algorithm in a broader set of challenging situations, it is not clear how concept shift should be simulated, since this would involve changing the class labels of documents that are included in a sample, and it is unclear whether there are sensible policies for doing it.

- SLD was conceived for handling not concept shift but prior probability shift; it would thus probably make no sense to simulate situations for which SLD is intentionally unequipped.[21]

[21] Saerens et al. (2002) explicitly assume

> "that the generation of the observations within the classes, and thus the within-class densities, $p(\mathbf{x}|y)$, does not change from the training set to the new data set (only the relative proportion of measurements observed from each class has changed). This is a natural requirement; it supposes that we choose the training set examples only on the basis of the class labels y, not on the basis of $\mathbf{x}$."

Our method to generate samples, detailed in Section 3.3.2, is indeed based on choosing the training set examples only on the basis of the class labels.

3.6 On SLD and the mutual consistency of the posteriors and priors of U

We here show that SLD may be viewed as an attempt to enforce a necessary condition for the posteriors $\Pr(y_j|\mathbf{x}_i)$ of the documents $\mathbf{x}_i \in U$ to be calibrated. In order to show this, let us define

- U_a to be the set of documents $\mathbf{x}_i \in U$ such that $\Pr(y_j|\mathbf{x}_i) = a$;
- U^j to be the set of documents $\mathbf{x}_i \in U$ such that $\mathbf{x}_i \in y_j$;
- U_a^j to be the set of documents $\mathbf{x}_i \in U_a \cap U^j$.

Recall from Section 3.1 that the posteriors $\Pr(y_j|\mathbf{x}_i)$, with $\mathbf{x}_i \in U$, are perfectly calibrated when, for all $a \in [0,1]$, it holds that $\frac{|U_a^j|}{|U_a|} = a$. If so, then it holds that

$$\begin{aligned} |U_a^j| &= |U_a| \cdot a \\ &= \sum_{\mathbf{x}_i \in U_a} a \\ &= \sum_{\mathbf{x}_i \in U_a} \Pr(y_j|\mathbf{x}_i) \end{aligned} \tag{3.14}$$

Since U is finite, there is a finite set A of values that the posteriors of the documents in U take. From Equation 3.14 it follows that

$$\sum_{a \in A} |U_a^j| = \sum_{a \in A} \sum_{\mathbf{x}_i \in U_a} \Pr(y_j|\mathbf{x}_i) \tag{3.15}$$

which can be rewritten as

$$|U^j| = \sum_{\mathbf{x}_i \in U} \Pr(y_j|\mathbf{x}_i) \tag{3.16}$$

By multiplying both sides by $\frac{1}{|U|}$ we obtain

$$\Pr_U(y_j) = \frac{1}{|U|} \sum_{\mathbf{x}_i \in U} \Pr(y_j|\mathbf{x}_i) \tag{3.17}$$

which is exactly the condition on the "mutual consistency" of priors and posteriors that SLD tries to enforce (see Equation 2.10 and Step 13 of Algorithm 2), and that holds after SLD has converged.

In sum, for the posteriors $\Pr(y_j|\mathbf{x}_i)$ of the documents $\mathbf{x}_i \in U$ to be calibrated, Equation 3.17 must hold. While SLD is not a full-fledged attempt to calibrate the posteriors in U (which would be impossible, since we do not know the label of any document in U), it may nevertheless be seen as a step in that direction.

3.7 Discussion

We present a thorough reassessment of SLD, a well-known algorithm that, given a machine-learned single-label classifier and a set of unlabelled documents characterized by prior probability shift with respect to the labelled documents the classifier has been trained on, adjusts the posterior probabilities and class prior probability estimates returned by the classifier, in an iterative, mutually recursive way, with the goal of making both more accurate. Since its publication more than 15 years ago, SLD has become the standard algorithm for improving the quality of posterior probabilities, and is still considered a contender when it comes to estimating the class prior probabilities on unlabelled sets. However, its real effectiveness at improving the quality of posterior probabilities has been questioned. Studying SLD is thus not just an academic exercise, and is still important, since no other algorithm for adjusting the posterior probabilities returned by a classifier in the presence of PPS is known, and since the quality of posterior probabilities is of key importance for a number of document management tasks, including document ranking and cost-sensitive text classification.

We here present the results of a large scale experimentation that uses multiple learners and a very large, publicly available dataset for text classification, on which multiple amounts of PPS have been simulated. In total, the experimentation consists of 129,500 train-and-test runs for the binary case and 14,000 such runs for the multiclass case. In these experiments we are especially interested in SLD's ability at improving the quality of posterior probabilities, something which Saerens et al. (2002) evaluated only indirectly, i.e., in terms of the accuracy of (cost-insensitive) classification that results from using the posteriors SLD generates.

Our study allows three main conclusions. The first conclusion is that SLD is ineffective, and often detrimental, when the classifier has not been previously calibrated; in this latter case, an additional disadvantage is that the speed of convergence of SLD is slower, and the probability that the computation does not even converge is higher. The second conclusion is that, in any situation, the improvements that SLD brings about are higher (or the deterioration it brings about is lower) when PPS is higher. The third conclusion is that the improvements that SLD brings about are higher (or the deterioration it brings about is lower) when the number of classes in the codeframe is small; binary classification is thus the most apt context for the use of SLD, which should instead be used with prudence in multiclass classification with small numbers of classes, and completely avoided in multiclass classification with high numbers of classes. An additional disadvantage of working with a high number of classes is that, as for non-calibrated classifiers, the speed of convergence of SLD is much slower, and the probability that the computation does not even converge is much higher.

Our results also show that, concerning the improvements in the quality of the posteriors that have been found in the binary case (and, to a lesser extent, in the multiclass case when the codeframe is small), these are due to a reduction of the calibration error, and not to a reduction of the refinement error. This shows that SLD is, in essence, a re-calibration algorithm, i.e., an algorithm for re-calibrating the posterior probabilities of documents belonging to an unlabelled set U, where these posteriors have been returned by a classifier already calibrated on a training set L and where the re-calibration is made necessary by the presence of prior probability shift. For this kind of use, and when the number of classes $|Y|$ is small and the classifiers have been calibrated beforehand, the use of SLD is still recommended.

Given this, in the next chapter we will attempt to use the SLD algorithm in order to improve the posterior probabilities of a classifier trained in the active learning scenario typical of TAR workflows:

the end goal is that of using the new posteriors to help the MINECORE framework (Oard et al., 2018) (see also Section 2.3.3) make better informed decisions.

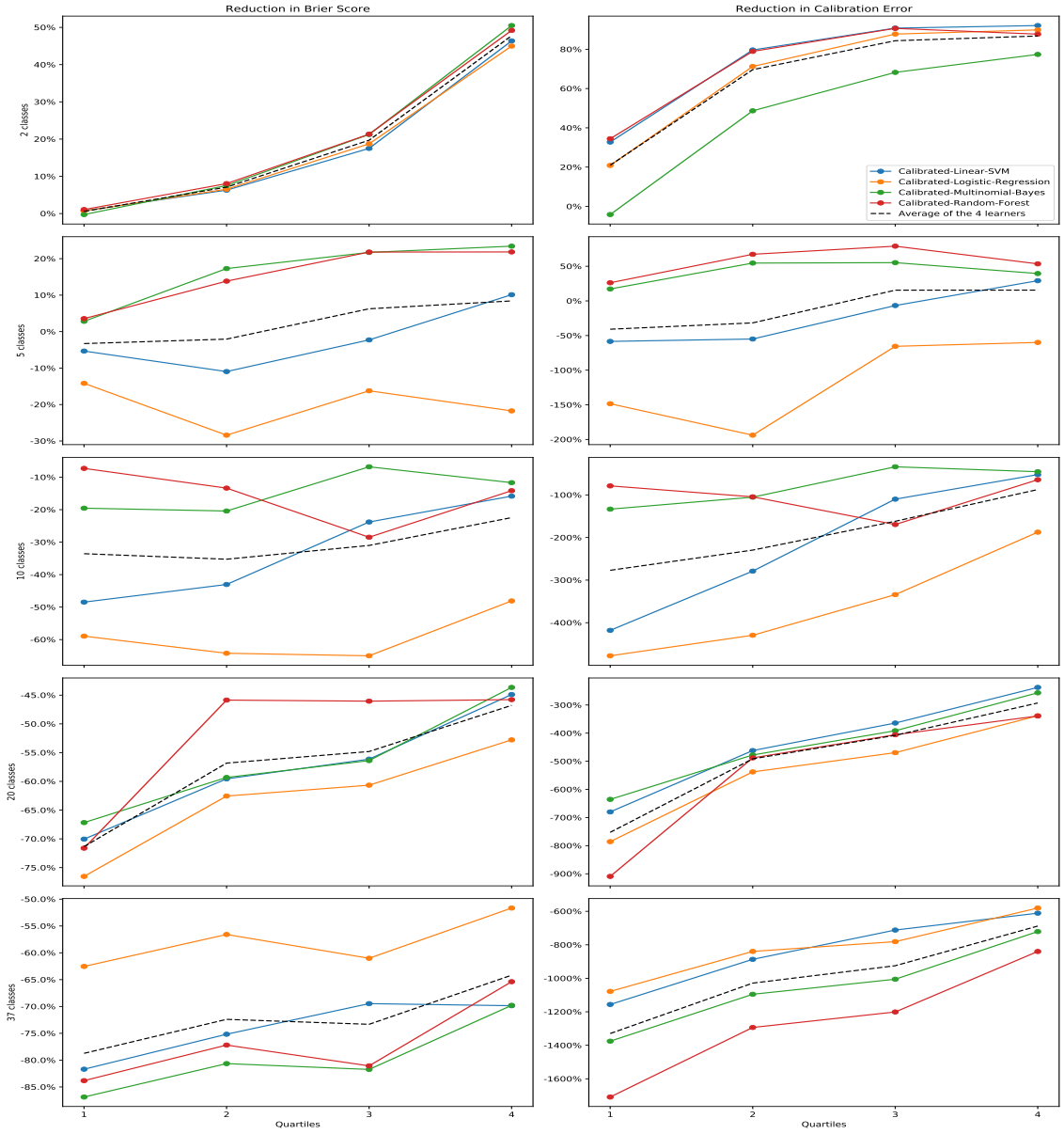


Figure 3.2: Error reduction for the isomerous variants of Brier Score (left) and Calibration Error (right) for the four different quartiles into which samples have been binned; in each of the ten subfigures, quartiles are arranged with low-shift quartiles on the left and high-shift quartiles on the right. Subfigures are sorted top-to-bottom as a function of the number of classes considered, from $|Y| = 2$ (top) to $|Y| = 37$ (bottom).

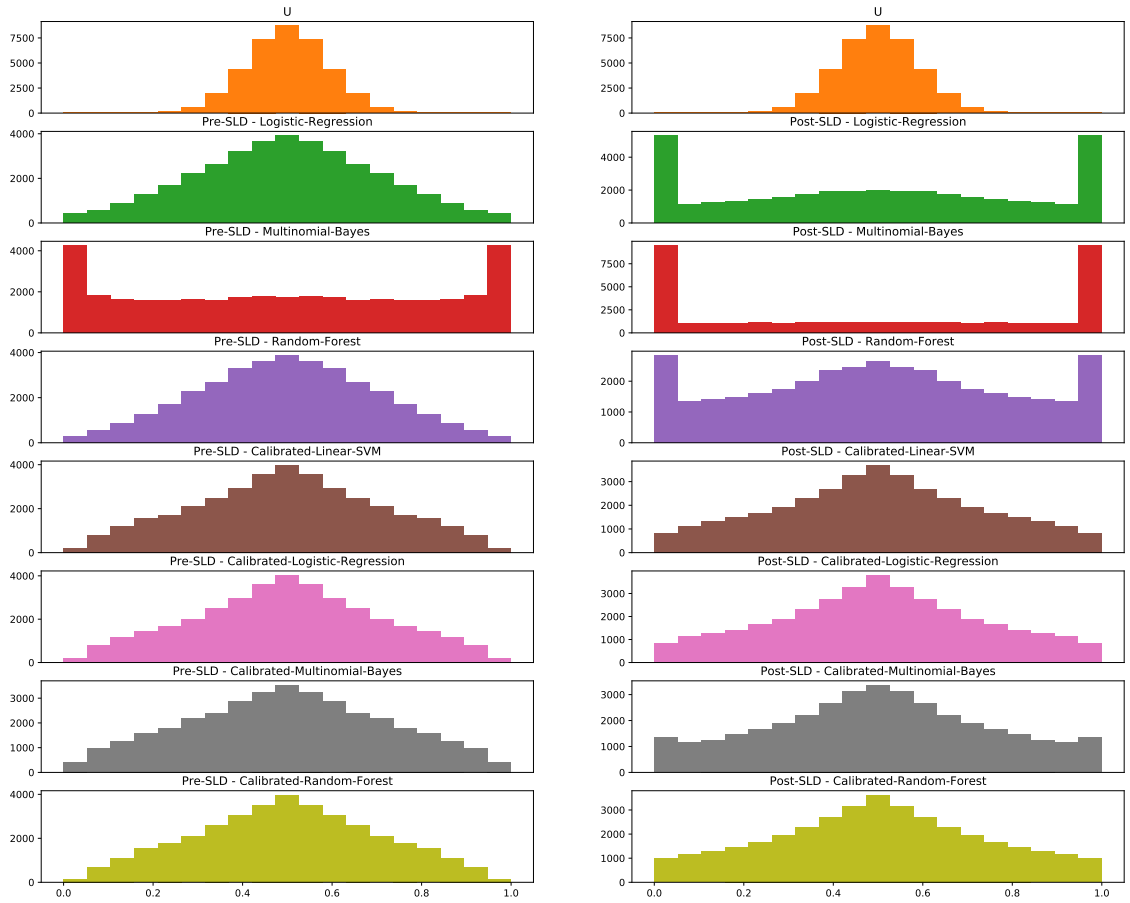


Figure 3.3: Histograms showing various distributions of the class priors for $|Y| = 2$ experiments. The two top subfigures show the true distribution in the unlabelled set U; the other subfigures show, for different classifiers, the distribution of predicted class priors before SLD is applied (i.e., as computed on the classifier output) and the distribution of predicted class priors after the application of SLD.

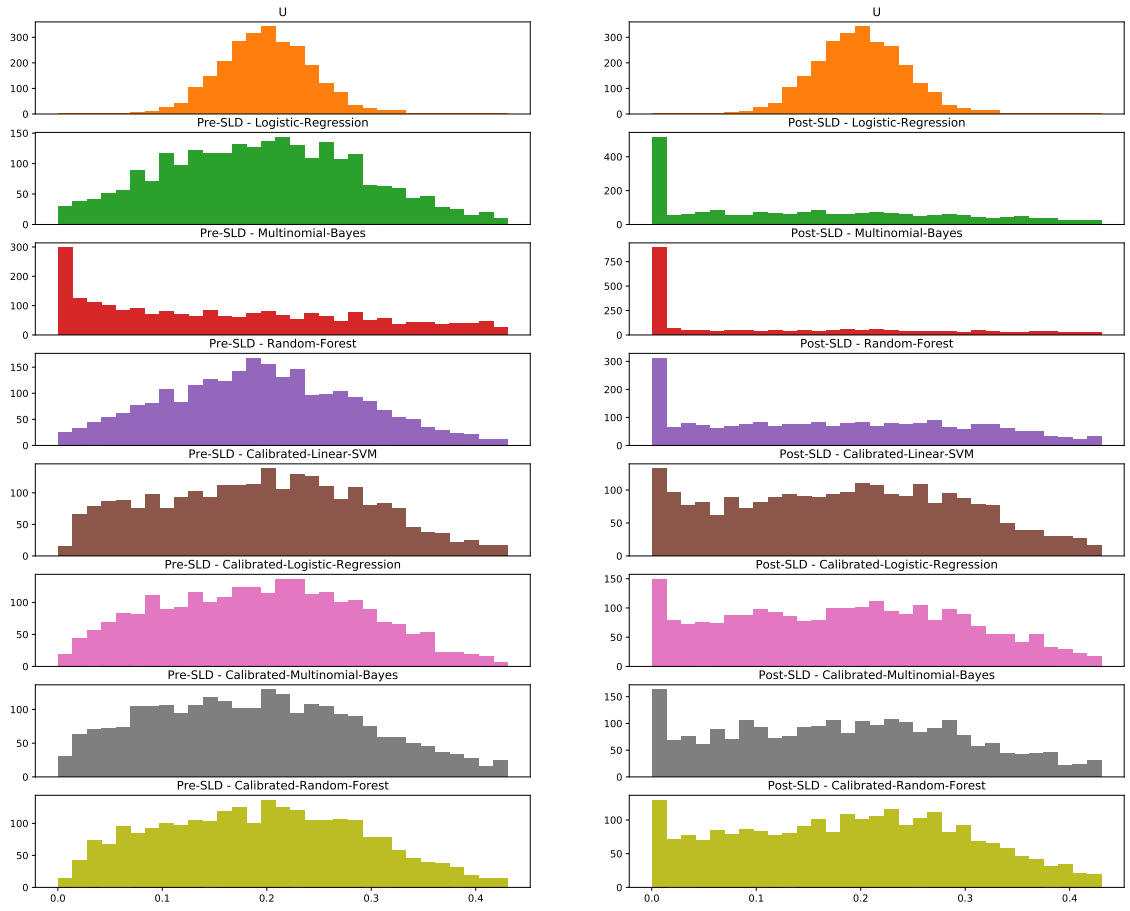


Figure 3.4: As in Figure 3.3 but with $|Y| = 5$.

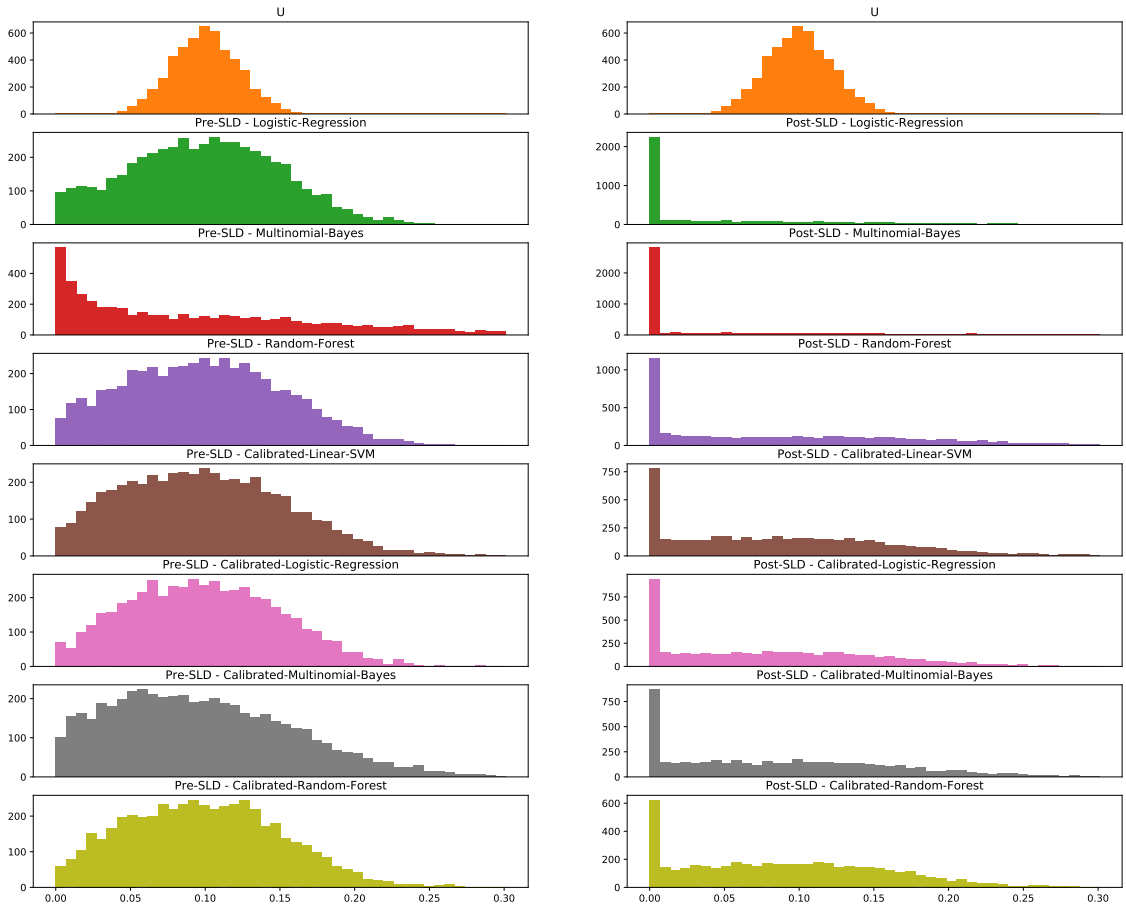


Figure 3.5: As in Figure 3.3 but with $|Y| = 10$.

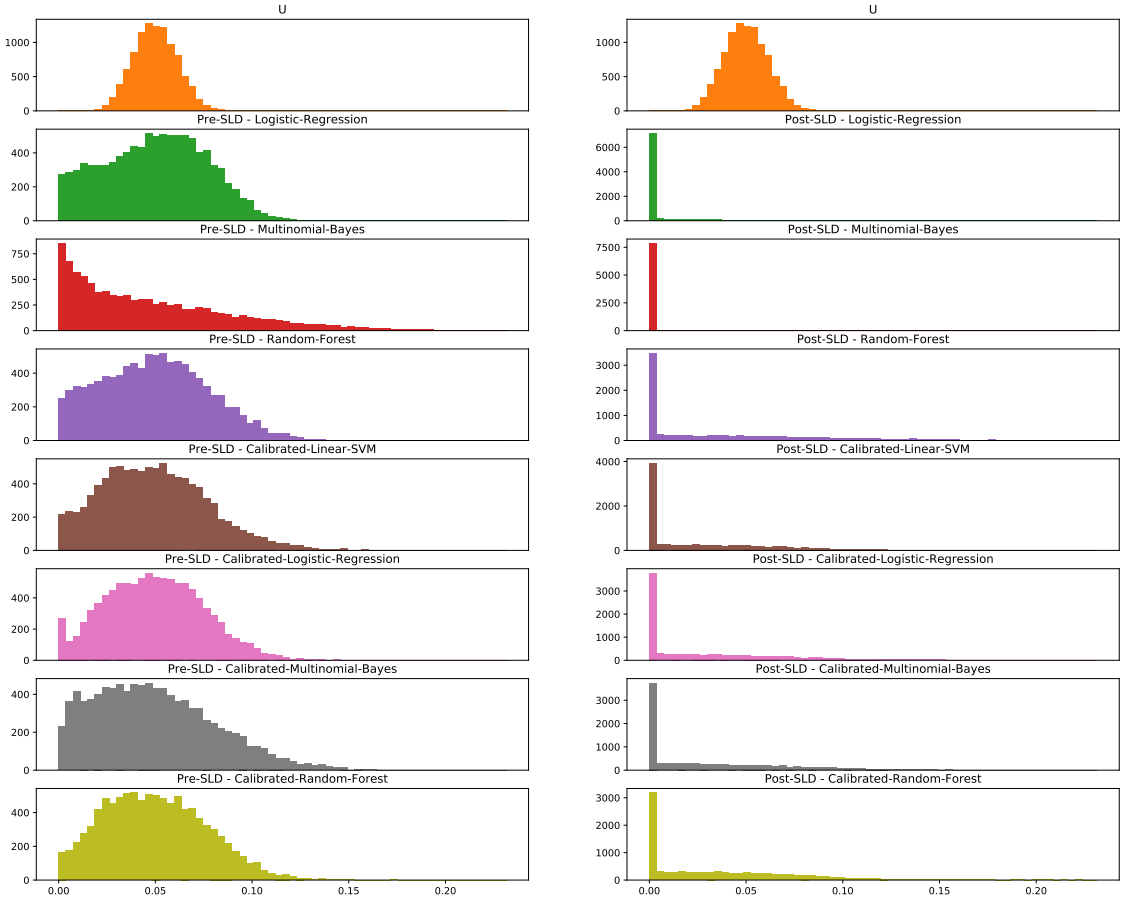


Figure 3.6: As in Figure 3.3 but with $|Y| = 20$.

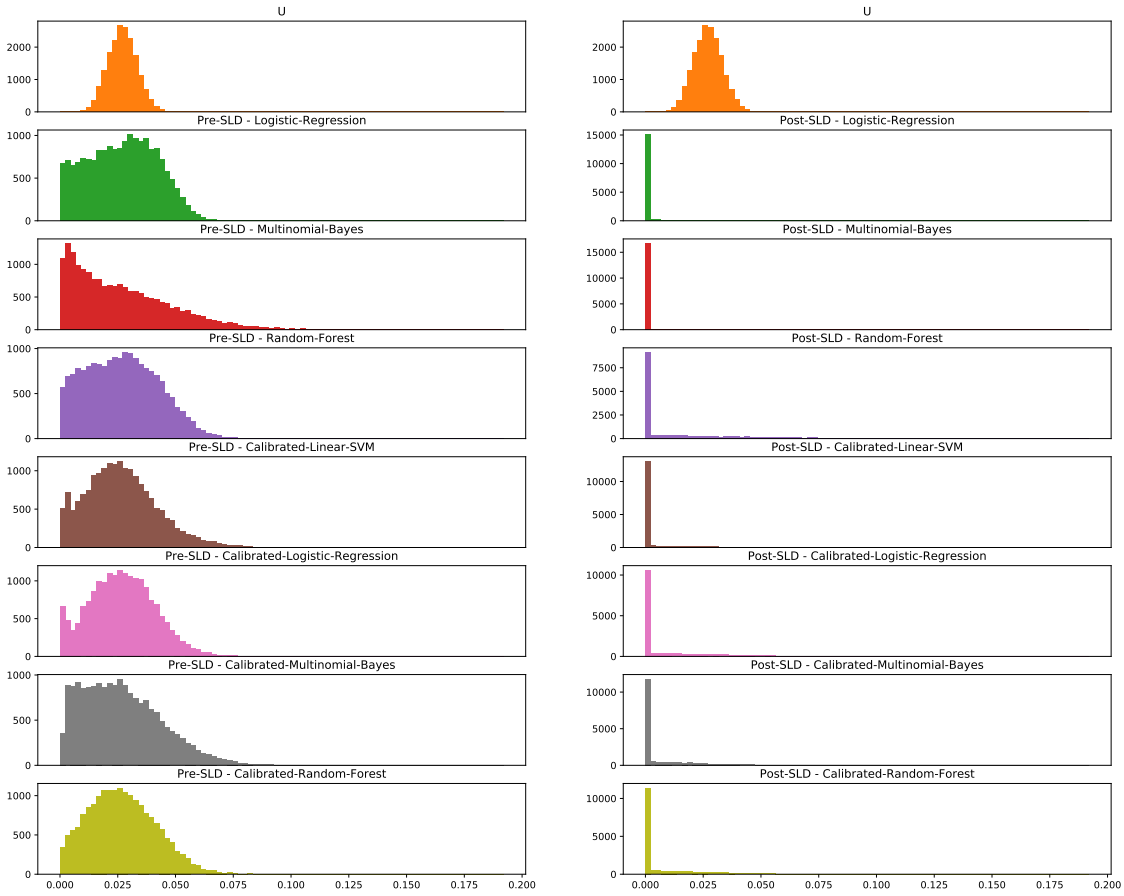


Figure 3.7: As in Figure 3.3 but with $|Y| = 37$.

Chapter 4

Improving MINECORE posterior probabilities

The quality of the classifier posterior probabilities estimates is one of the key elements, in MINECORE framework, to assess and minimize the risk brought by a misclassification error (see Section 2.3.3). In this chapter, we will analyze which active learning technique, among ALvRS, ALvUS (two of the most well-known and used active learning strategies) and a newly proposed ALvRUS policy, builds the best training set, thus delivering better probability estimates.

Building a training set via an AL policy inevitably generates a more or less substantial Prior Probability Shift (PPS): given the evaluations and analysis of Chapter 3, we might then expect the SLD algorithm to further improve our posterior probabilities, resulting in even more accurate risk assessments by the MINECORE framework.

This chapter tries to answer these two research questions (i.e., which AL policy is better for MINECORE, and whether SLD brings improvements or not) with an extensive experimental protocol, and a thorough analysis of the results; the code to reproduce our experiments is publicly available at `https://github.com/levnikmyskin/improved_risk_min_tar`. Finally, the work presented in this chapter was published in Molinari et al. (2023).

4.1 Research question # 1: How should we label the training set?

The MINECORE experiments that were presented in Oard et al. (2018) used *passive learning* (PL), i.e., the labelled set L that was used for training the classifiers ϕ_r and ϕ_p was (conceptually) a random sample of the pool U. This is somehow at odds with the standard practice of the TAR field, according to which the labelled set L is usually annotated via *active learning*. Active learning is a class of techniques whereby the machine takes an active role in choosing the examples that should be part of the training set L; the most popular such class (and the only class we will consider here) is that of *pool-based* AL techniques, whereby the machine chooses the examples from an available pool P of unlabelled examples (rather than, say, generating artificial examples with pre-specified properties) and asks the annotators to label them. We will consider two popular forms of pool-based AL here, *active learning via uncertainty sampling* (ALvUS) and *active learning via*

relevance sampling (ALvRS), along with a new ALvRUS (*active learning via Relevance/Uncertainty sampling* policy; see Section 2.1.2 and 4.3.3 for a more detailed explanation). We choose ALvUS because it is probably the most widely used AL technique in machine learning at large, while we choose ALvRS (which is essentially a form of *relevance feedback* (Rocchio, 1971)) because it is probably the most widely used AL technique in TAR, under the name of *continuous active learning* (Cormack, Grossman (2015b), see also Section 2.3.1).

In this chapter we want to answer the following research question:

> **RQ1: Assuming we use MINECORE as the TAR system, how should we annotate the training set L? Is it advantageous to annotate it via passive learning or via an active learning policy?**

The question is non-trivial since the three policies (a) typically give rise to classifiers that, for the same number $|L|$ of training examples, are characterized by different levels of accuracy, and (b) typically give rise to different sets $U \equiv P \setminus L$ of unlabelled sets that the probabilistic classifiers need to rank. This means that one policy may generate a better training set L (i.e., one that delivers a better classifier) but also generate a set $U \equiv P \setminus L$ harder to rank accurately, and the interactions between these two factors are difficult to predict.

Note that past results showing that AL delivers more accurate classifiers than PL are usually based on testing the two techniques *on the same test set* U. This is not representative of our scenario, where the set U that the classifiers need to classify changes when the training set L changes, because $U \equiv P \setminus L$. More in detail, we may expect the ALvUS policy to produce, as the AL literature suggests, the best classifiers (see e.g., Esuli et al. (2019)). However, we may expect the PL policy to generate the easiest unlabelled set U to rank, since when using PL the sets L and U are independently and identically distributed (IID), which is not the case when using US or RS. Concerning the RS policy, we may expect it to lead to a high number of relevant (i.e., responsive / privileged) documents being manually (i.e., correctly) labelled, which helps increase recall, which is an important goal in TAR; however, the RS policy is usually the worst of the three in terms of deviation from the IID condition, which means that we may expect it to generate a set U of documents very hard to rank.

When the training set L has been obtained from P via active learning, the fact that L and U may not be IID is usually true, since in these cases L is anything but a random sample of P (i.e., it is a biased sample, suffering from *sample selection bias*), which means that $U \equiv P \setminus L$ is also not a random sample of P. As a consequence, the relationship between L and U is one of *dataset shift* (Moreno-Torres et al., 2012; Quiñonero-Candela et al., 2009), and in particular of *prior probability shift* (see Section 2.1.1 and the discussion in Chapter 3). As a result, a classifier trained on L is unlikely to perform well on U.

All in all, this says that it is not easy to predict which of the three methodologies is the most beneficial for MINECORE, and that our research question is an interesting one.

4.2 Research question # 2: Should we try to improve the posteriors via SLD?

MINECORE receives as input the posteriors $\Pr(y_r|\mathbf{x})$ and $\Pr(y_p|\mathbf{x})$ returned by the two probabilistic classifiers ϕ_r and ϕ_p (see Section 2.3.3). The quality of these posteriors is thus of key importance for the performance of MINECORE. In order to ensure this quality, Oard et al. (Oard et al.,

2018) subject ϕ_r and ϕ_p to a *calibration* step so that they return well-calibrated probabilities. Calibration routines use k-fold cross-validation, i.e., they tune the classifier in such a way that, when classifying the training documents $\mathbf{x} \in L$, the classifier returns posterior probabilities $\Pr(y|\mathbf{x})$ that are well-calibrated. In reality, what we are really interested in is that the posteriors of the *unlabelled* documents in U, and not those of the labelled documents in L, are calibrated; if the sets L and U are IID, though, if the posteriors of the documents in L are calibrated, those of the documents in U are too. Indeed, virtually all probability calibration routines, including the one used in Oard et al. (2018), assume that L and U are IID. If L and U are not IID, though, the fact that the posteriors of the documents in U are well-calibrated is not guaranteed even after the intervention of these routines; in particular, if L and U suffer from *prior probability shift*, the posteriors of the documents in U will not be calibrated.[1] ALvUS and (especially) ALvRS generate substantial PPS (Settles, 2012);[2] this means that, if we had to use one of them for generating our training sets L, the quality of the posteriors we would obtain from the resulting classifiers would be a concern.

The SLD algorithm is a well-known algorithm (and the only known algorithm) that attempts to improve the quality of the posteriors returned by probabilistic classifiers. The reassessment of SLD we conducted in Chapter 3 has shown that SLD may be very effective at improving the quality of the posteriors in situations in which binary classification is performed under high PPS. This would suggest using SLD to improve the quality of the posteriors that are to be fed to MINECORE. However, it should be remarked that the experimentation of Chapter 3 never used active learning for generating the training sets L; this means that it is not clear how SLD might perform in our scenario if we had to use an active learning policy.

In addition to RQ1, in this chapter we want to answer the following research question:

> **RQ2: Assuming we use MINECORE as the TAR system, can we obtain benefits from using SLD for updating the posterior probabilities that MINECORE receives as input from the two probabilistic classifiers?**

The above question may have different answers depending on the answer to RQ1, i.e., depending on whether we annotate the labelled set L via PL, via ALvUS, via ALvRS, or via ALvRUS, because PL generates no PPS while ALvUS and (especially) ALvRS (and ALvRUS) generate high dataset shift.

4.3 Experiments

4.3.1 The dataset

We will use RCV1-v2 as the only dataset for our experiments (see Section 2.6.1). For computational reasons, however, we will only use the first 100,000 documents of RCV1-v2 collection, which also comprises the 23,149 documents used for training in Lewis et al. (2004) In Oard et al. (2018) a

[1] This is a direct consequence of Equation 3.3.

[2] The reason why ALvUS generates PPS is that it encourages the annotators to annotate documents that are close to the decision threshold; when the dataset is imbalanced, this means that the annotators end up annotating a fairly large proportion of members of the minority class, which means that the prevalence of the minority class in L ends up being larger that its prevalence in U. Instead, the reason why ALvRS generates PPS is that it encourages the annotators to annotate documents belonging to the minority class; this means that the prevalence of the minority class in L ends up being *much* larger than its prevalence in U.

subsample of RCV1-v2 class pairs was chosen to simulate the classes of responsive documents (c_r) and privileged documents (c_p). More specifically, every pair (c_r, c_p) was selected such that the prevalence (i.e., relative frequency) of c_r in the entire RCV1-v2 collection is in the [0.03,0.07] interval and the prevalence of c_p in the responsive documents is in [0.01,0.20]. This broad range is actually seen in e-discovery practice, with some classification tasks run in "needle in a haystack" conditions, and others run on collections that have been prescreened when they were acquired to have as high a responsiveness prevalence as can be achieved (Oard, Webber, 2013). For each of the 24 responsiveness classes that meet the prevalence criterion, the authors of Oard et al. (2018) randomly selected 5 privilege classes that meet the respective prevalence criterion: this gives rise to 120 class pairs. The experiments described in this chapter are run on the very same set of class pairs as used in Oard et al. (2018).

4.3.2 The active learning methods

In our experiments we test MINECORE on training sets generated by three active learning policies (see Section 2.1.2), i.e., *Active Learning via Uncertainty Sampling* (ALvUS), *Active Learning via Relevance Sampling* (ALvRS) and a middle-ground policy which we present in the next subsection, called *Active Learning via Relevance/Uncertainty Sampling* (ALvRUS). Other, more recent, active learning algorithms such as those presented in Dasgupta, Hsu (2008); Huang et al. (2014) were explored and dismissed due to unfeasibly expensive computational costs (see Section 4.3.6 for more on this).

4.3.3 Active learning via relevance/uncertainty sampling

The ALvRUS policy asks the reviewer to annotate, at each iteration, the $b/2$ documents in U for which $\Pr(y_i|\mathbf{x})$ is closest to 0.5 *and* the $b/2$ documents in U for which $\Pr(y_i|\mathbf{x})$ is highest. In other words, this policy is a mix of ALvUS and ALvRS (hence its name), since it attempts to satisfy both goals at the same time, i.e., providing feedback on the regions of the instance space in which the classifier is weakest (as in ALvUS) and adding many examples from the minority class to L (as in ALvRS).

4.3.4 Passive learning

By Passive Learning (or PL) we mean a simple strategy that generates a training set by uniformly sampling from the data pool. This was the technique used in Oard et al. (2018) to emulate the annotated training sets and, as such, we use it as a baseline.

4.3.5 The Rand(RS), Rand(US), and Rand(RUS) policies

We add to our experiments three "oracle-like" policies (*Rand(RS)*, *Rand(US)*, *Rand(RUS)*), which will serve as testing grounds for the other algorithms run for RQ1 and RQ2. The goal of a *Rand* policy is that of building a training set L and a test set U, from the same pool P where the other AL policies operate, by performing a "controlled" random sampling of the documents. That is, controlled in such a way that it yields the same prevalence value of the positive class that we would have obtained had we used one of the AL policies (i.e., ALvRS, ALvUS or ALvRUS). For more details on the *Rand* policy, we refer the reader to Section 2.1.3.

4.3.6 Exploring other policies

Motivated by the advances in the active learning literature, we decided to explore other more recent and sophisticated AL policies. Unfortunately, the ones we considered were computationally expensive or prohibitive to run, and we eventually decided not to use them in our experiments. The following is an overview of the two policies we considered and of the reasons why they proved too challenging to run.

Active Learning by Querying Informative and Representative Examples (QUIRE). QUIRE (Huang et al., 2014) is a recent (with respect to ALvUS and ALvRS) active learning algorithm whose goal is not only to annotate documents for which the classifier exhibits the strongest uncertainty (as in ALvUS), but also to maximise representativeness (i.e., diversity) of the examples annotated at every iteration. For more technical details we refer the reader to Huang et al. (2014).[3]

However, this technique is problematic since it requires (a) the computation of an $m \times m$ kernel matrix K, where m is the number of documents in pool P, and (b) the storage of another $m \times m$ matrix $L = (K + \lambda I)^{-1}$ (where I is the identity matrix). This is clearly unfeasible in our case, where m=100,000 (assuming one byte for each element of each matrix, just storing K and L would require approximately 160GB); note also that, in real e-discovery scenarios, P may contain many more documents than the 100,000 documents we use in our experiments.

Active Learning via Hierarchical Sampling (ALvHS). ALvHS was first presented in Dasgupta, Hsu (2008). The goal of this algorithm is to avoid sampling bias, i.e., the fact that the set of labelled documents $\mathcal{L}$ may not be representative of the remaining set of unlabelled documents U, which happens when using AL strategies such as ALvRS, ALvUS or ALvRUS. The basic step of this policy consists of partitioning the data into hierarchical clusters and later sampling from them (for a more in-depth presentation of the algorithm, see Dasgupta, Hsu (2008)).

However, given m items to be clustered, hierarchical clustering algorithms have a complexity of $\mathcal{O}(m^3)$, which can be reduced to $\mathcal{O}(m^2)$ or $\mathcal{O}(m^2 \log m)$ only in some specific cases (Patel et al., 2015). In any case, the algorithm results in unfeasibly expensive computation costs in our application scenario.

4.3.7 The experimental setup

In order to answer RQ1, we compare our different active / passive learning policies for training the two classifiers ϕ_r and ϕ_p used in Step 1 of MINECORE.

For these experiments, we take the first 100,000 documents of the RCV1-v2 collection as the pool of documents P to which MINECORE is applied. As already mentioned in Section 2.6, for higher consistency with the experiments carried out in Oard et al. (2018), we use the same RCV1-v2 pairs of classes used in Oard et al. (2018) to play the role of the responsive class y_r and the privileged class y_p; this is a set of 120 pairs of RCV1-v2 classes (see Section 2.6 and Oard et al. (2018, §4.1) for details).

For each active / passive learning policy we test, we run different experiments in which we vary the size s of the training set that the process eventually creates; we test all sizes $s \in \{2000, 4000, 8000, 16000, 23149\}$; the reason why we use the fairly peculiar size $s = 23149$ is that this is the size used in Oard et al. (2018).

We seed all the active learning processes with a set S of 1000 initial training documents randomly sampled from our pool P, train (for each $y \in \{y_r, y_p\}$) an SVM classifier on S, calibrate it, and

[3] An implementation of this algorithm is available at `https://libact.readthedocs.io/en/latest/libact.query_strategies.html#module-libact.query_strategies.quire`.

apply it to all the remaining unlabelled documents in $U \equiv P \setminus S$ to obtain posterior probabilities $\Pr(y|\mathbf{x})$ for each of them.[4] We constrain this initial training set S to have at least 2 positive instances, which are the bare minimum in order to calibrate the classifier.

We then iterate the active learning process on the remaining unlabelled documents with a batch size $b = 1000$. The active learning process simulates the work of *infallible* reviewers, i.e., at each iteration the unlabelled documents are ranked based on their assigned posteriors, the b unlabelled documents which are ranked highest are added to the training set together with their true label (which simulates the infallible reviewer's annotation) and removed from the unlabelled set U, the classifier is retrained, recalibrated, and applied again to all the remaining unlabelled documents to obtain updated posterior probabilities for each of them.

As mentioned in Section 4.3.5, for each training set size $s \in \{2000, 4000, 8000, 16000, 23149\}$ we also generate training sets of size s via the *Rand*(US), *Rand*(RS), *Rand*(RUS) and PL policies; for each such policy all sets are obtained each time anew via random sampling (constrained for *Rand*(US), *Rand*(RS), *Rand*(RUS) – see Section 4.3.5 and 2.1.3, unconstrained for PL), i.e., it is *not* the case that, for a given policy, the smaller training sets are contained in the larger ones. Here too, the classifier is trained, calibrated, and applied to all the remaining unlabelled documents, after which these latter are ranked based on their newly obtained posterior probabilities.

For any of these policies, the finally obtained posterior probabilities for each of the remaining unlabelled documents are fed to Step 2 of the MINECORE workflow. We run Steps 2 and 3 of the MINECORE workflow for any of the above policies and for each cost structure (Table 2.2), and evaluate MINECORE as explained in Section 2.3.3 and 2.5.2, so as to ascertain which policy brings about the smallest overall cost. We will show and comment these results in Section 4.4.1.

In order to answer RQ2, instead, we run SLD on the posterior probabilities (here indicated as $\Pr^{\text{Pre-SLD}}(y|\mathbf{x})$) obtained from each of the above policies, thus obtaining $\Pr^{\text{Post-SLD}}(y|\mathbf{x})$ posterior probabilities, and we compare, for each policy and each cost structure, the overall cost resulting from using Pre-SLD posteriors with the overall cost resulting from using, in place of them, Post-SLD posteriors. These results are commented on in Section 4.4.2.

4.4 Results

In this section we show and analyse our results for RQ1 (Section 4.4.1) and RQ2 (Section 4.4.2). For RQ1 we present our results in Tables 4.2 and 4.3. For RQ2 we illustrate our results in Tables 4.4, 4.5, 4.7, 4.8, 4.9, and 4.10. Finally, we show some insightful plots concerning the AL policies in Figures 4.1 and 4.2, and other plots concerning the effects of SLD on the distribution of the posterior probabilities in Figures 4.3 to 4.5.

4.4.1 RQ1

The goal of our first research question (RQ1) is that of understanding which among the policies described in Sections 4.3.2 to 4.3.5 generates the best training sets on which to train our two classifiers. ALvRS (in its CAL incarnation) is the standard active learning methodology used in one-phase TAR systems. In these contexts, we are given an unlabelled pool of documents and our goal is to find the highest number of relevant (i.e., positive class) documents in the least possible

[4] Since we use SVMs as the base learner, calibration is strictly necessary, since SVMs return confidence scores that are not probabilities; our calibration step thus maps these confidence scores into calibrated posterior probabilities. In all of our experiments we use Platt's calibration method (Platt, 2000).

	$\lvert\mathcal{L}\rvert$	Recall	$\Pr_{\mathcal{L}}(y)$	$\Pr_{\mathcal{U}}(y)$
ALvRS	2,000	0.240 ± 0.113	0.698 ± 0.056	0.080 ± 0.099
	4,000	0.509 ± 0.218	0.762 ± 0.113	0.064 ± 0.100
	8,000	0.766 ± 0.231	0.637 ± 0.203	0.045 ± 0.095
	16,000	0.894 ± 0.158	0.427 ± 0.252	0.028 ± 0.077
	23,149	0.936 ± 0.107	0.336 ± 0.254	0.019 ± 0.061
ALvUS	2,000	0.099 ± 0.044	0.300 ± 0.047	0.088 ± 0.099
	4,000	0.196 ± 0.074	0.308 ± 0.068	0.083 ± 0.099
	8,000	0.431 ± 0.161	0.337 ± 0.072	0.071 ± 0.101
	16,000	0.597 ± 0.186	0.251 ± 0.089	0.062 ± 0.102
	23,149	0.670 ± 0.183	0.204 ± 0.091	0.059 ± 0.102
ALvRUS	2,000	0.135 ± 0.060	0.408 ± 0.067	0.086 ± 0.099
	4,000	0.381 ± 0.176	0.560 ± 0.057	0.073 ± 0.100
	8,000	0.699 ± 0.249	0.551 ± 0.129	0.052 ± 0.100
	16,000	0.863 ± 0.197	0.388 ± 0.181	0.036 ± 0.091
	23,149	0.914 ± 0.150	0.307 ± 0.191	0.027 ± 0.081

Table 4.1: Average recall and training/test class prevalence values at different training set sizes for the three active learning policies.

amount of time; given this goal, relevance sampling is usually preferred to uncertainty sampling, as it encourages the annotators to review documents that are likely to be relevant (see Section 2.3).

However, MINECORE operates according to a two-phase TAR workflow, where in the 1st phase we create a training set which is later used to train the classifier, and where the posteriors returned by this classifier are then used as input by MINECORE in the 2nd phase in order to prioritize the documents that the annotator should review. That is, in this case ALvRS might be less effective than ALvUS, since, by repeatedly improving the classifier in the regions of instance space on which the classifier is most uncertain, ALvUS might eventually obtain higher-quality posteriors. Moreover, we might expect the ALvRUS policy, which stands as a middle ground between ALvRS and ALvUS, to also improve MINECORE's results by building a higher quality classifier than ALvRS.

In Table 4.1 we report the average recall[5] and the average class prevalence in the training set $\mathcal{L}$ and in the set of unlabelled documents $\mathcal{U}$ as deriving from the application of the different AL policies, where the averages are computed across y_r and y_p. All these figures are reported at different training set sizes (i.e., 2000, 4000, 8000, 16000, 23149); this should give a clearer picture of the overall scenarios generated by the different active learning policies.

The results in Table 4.2 show that, for any given cost structure, ALvRUS is the most effective among the policies we have studied. Regarding the other policies, ALvUS appears to be the second-best policy for two cost structures out of three (Λ_1 and Λ_2), whereas ALvRS achieves second-best results on Λ_3. Finally, the passive learning strategy is the worst one in two cases out of three (Λ_2 and Λ_3).

Furthermore, the *Rand*(RS), *Rand*(US) and *Rand*(RUS) results of Table 4.3 tell us that a higher prevalence value of the positive class in the test set, and an overall better balance between training

[5]In this context, by *recall* we mean the ratio $\Pr_{\mathcal{L}}(y)/\Pr_{\mathcal{P}}(y)$ between the number of positive documents in the training set $\mathcal{L}$ and the total number of positive documents in the entire pool $\mathcal{P}$.

Λ	PL	ALvRS	ALvUS	ALvRUS
Λ_1	38,589	39,373	32,511	**19,890**
Λ_2	13,047	10,398	6,721	**5,766**
Λ_3	5,318	2,742	3,749	**2,129**

Table 4.2: Average overall costs resulting from running MINECORE when different policies have been used for generating the training sets. Values in **boldface** indicate the best results for the given cost structure. The reported values are obtained by averaging across the experiments run with different training set sizes (2000, 4000, 8000, 16000, 23149). Superscripts † and ‡ indicate whether the second-best method is not statistically significantly different from the best one, according to a Wilcoxon signed-rank test at different confidence levels: symbol † indicates $0.001 < p < 0.05$, while ‡ indicates $p \geq 0.05$. In this table, all differences are statistically significant, hence no instances of † and ‡ are present.

Λ	Rand(RS)	Rand(US)	Rand(RUS)
Λ_1	63,109	**41,284**	41,298
Λ_2	14,708	**8,397**	9,819
Λ_3	4,102	4,087	**2,964**

Table 4.3: Same as Table 4.2, but with different policies for generating the training sets.

and test class prevalence values (which we obtain when using uncertainty sampling), can steer the results in favour of ALvUS if, as with all the Rand policies, L and U do not suffer from sampling bias. Also, despite the fact that ALvRUS is the best policy in the results of Table 4.2, its corresponding Rand policy does not achieve the best results for two cost structures out of three (Λ_1 and Λ_2), albeit still obtaining better results than the *Rand*(RS) policy.

That said, we conclude that *the best strategy for training the ϕ_r and ϕ_p classifiers on which Step 1 of MINECORE hinges is, by a wide margin, ALvRUS.*

We end this discussion by noting that an obvious way to try to improve on ALvRUS would consist in adding to it a parameter α, so that the reviewers are asked to annotate, at each iteration, the $\alpha \cdot b$ documents in $\mathcal{U}$ for which $\Pr(y_i|\mathbf{x})$ is closest to 0.5 and the $(1-\alpha) \cdot b$ documents in $\mathcal{U}$ for which $\Pr(y_i|\mathbf{x})$ is highest (with $i \in \{r, p\}$); optimising this parameter (say, on a held-out dataset) would allow striking the best possible balance between "exploration" (the US component) and "exploitation" (the RS component).

4.4.2 RQ2

The goal of our second research question (RQ2) is that of understanding whether the application of SLD can (i) improve the quality of the posterior probabilities that are input to Step 2 of the MINECORE algorithm, and thus (ii) generate a reduction in overall cost.

The active learning strategies that we use in this chapter (and ALvUS, the "winner" in the previous batch of experiments, is no exception) tend to generate PPS between the training set and the set of unlabelled data; more specifically, they tend to generate situations in which the class prevalence value $\Pr_L(y)$ (with $y \in \{y_r, y_p\}$) can be much larger than the class prevalence value

	Λ	PL		ALvRS		ALvUS		ALvRUS	
Λ_1	Pre-SLD	38,589		39,373		32,511		**19,890**	
	Post-SLD	38,620	+0.08%	28,694	-37.22%	**25,309**‡	-28.46%	33,504	+40.63%
Λ_2	Pre-SLD	13,047		10,398		6,721		**5,766**	
	Post-SLD	13,073	+0.20%	14,040	+35.03%	**7,299**	+8.6%	14,612	+153.42%
Λ_3	Pre-SLD	5,318		2,742		3,749		**2,129**	
	Post-SLD	5,317	-0.02%	**3,735**	+36.21	8,452	+125.42%	8,748	+310.90%

Table 4.4: Average overall costs resulting from running MINECORE on posterior probabilities coming from either the classifier (Pre-SLD) or the SLD algorithm (Post-SLD). Notational conventions are as in Table 4.2. A value in **boldface** indicates the best result on the given row (i.e., combination of a cost structure and a choice between Pre-SLD and Post-SLD), whereas a value in underline indicates the best result for the given cost structure.

$\Pr_U(y)$. In such a scenario, given the findings of Esuli et al. (2021), we would expect the application of SLD to bring about substantial improvements to the quality of the posterior probabilities.

The results of our experiments are displayed in Table 4.4. Something we can see from these figures is that the results of the application of SLD are uneven; in some cases this application brings about a reduction in overall cost, while in other cases overall cost increases. In particular, SLD always brings about a deterioration when ALvRUS (the "winning" policy of our previous section) has been used to generate the training set. In sum, there is no clear answer to RQ2.

However, it is important to notice that, for all three cost structures, the best result is obtained by using ALvRUS and *not* using SLD. We thus have a clear answer to RQ1 and RQ2 altogether, i.e., that *the best course of action is to avoid using SLD and stick to the "Pre-SLD" posterior probabilities generated by classifiers trained via ALvRUS.*

In case we wonder what are the reasons for the failure of SLD to systematically improve the quality of our posterior probabilities, the answer comes from examining Table 4.5, which presents the results of experiments analogous to the ones of Table 4.4 but using the Rand active learning policies instead of the original ones. Table 4.5 says that, for all Rand policies and for all cost structures, the SLD algorithm brings about a *drastic* reduction in overall cost, unlike what happened for the original active learning policies. It is thus easy to conclude that *SLD copes well with PPS* (which the Rand policies generate) *but not with sampling bias* (which the original active learning policies, unlike the Rand policies, generate). Indeed, a close examination of SLD (see Algorithm 2 in Appendix 2.4) shows that nothing in it caters for sampling bias. Conversely, SLD does cater for PPS; indeed, if we assumed that there is no PPS between L and U, there would be no need to re-estimate the priors (see Line 13 of Algorithm 2) and, consequently, to update the posteriors (see Line 15), as SLD instead does.

Incidentally, the fact that Rand(RS) and Rand(RUS) are the two best overall algorithms confirms the observations of (Esuli et al., 2021) that the greater the shift between training set and test set, the better the performance of SLD; indeed, Rand(RS) and, to a lesser degree, Rand(RUS), tend to generate more PPS than Rand(US).

In order to provide a finer-grained analysis of the above results, we further

	Λ	Rand(RS)		Rand(US)		Rand(RUS)	
Λ_1	Pre-SLD	63,109		**41,284**		41,298	
	Post-SLD	14,394	-77.19%	19,819	-51.99%	**12,311**	-70.19%
Λ_2	Pre-SLD	14,708		**8,397**		9,819	
	Post-SLD	**2,586**	-82.42%	4,641	-44.73%	3,065	-68.79 %
Λ_3	Pre-SLD	4,102		4,087		**2,964**	
	Post-SLD	1,736	-57.68%	2,857	-30.10%	**1,586**	-46.49%

Table 4.5: Same as Table 4.4, but with different policies for generating the training sets.

1. Bin the classes by class prevalence value, in order to analyse whether the results may depend on the prevalence value of the class;

2. generate a visualization of the results of the different selection strategies (i.e., ALvRS, ALvUS, ALvRUS, PL and Rand), so as to see where the documents they select are picked from the data distribution;

3. Plot the distributions of the posteriors before SLD and after SLD, in order to visually understand how the SLD algorithm is adjusting these distributions.

Effects of class prevalence value

Regarding Point 1, we bin the 28 classes in quartiles by prevalence value; we accordingly call these quartiles "Low", "Medium-Low", "Medium-High", "High". In Table 4.6 we show which bin each class belongs to and whether the class is used to simulate Responsiveness (R), Privilege (P), or both (R+P).

We show in Tables 4.7 and 4.8 the results for each of the four bins, for responsiveness and privilege, respectively; these results are consistent with those of Table 4.4, with the PL strategy only slightly negatively affected by SLD and the three AL strategies suffering the strongest effects. Notice how the bins of classes with lower prevalence values (Low and Medium-Low) tend to be the ones where SLD performs comparatively better; given that SLD works better in high-shift scenarios (Esuli et al., 2021), this was to be expected, since the AL strategies will cause a much stronger PPS if the overall class prevalence value is low (since the few examples of the positive class tend to end up quickly in the training set). Also, notice how the above-described effect is stronger for ALvRS than for ALvUS; this was also to be expected, as ALvUS generates less extreme PPS than ALvRS. Finally, the results of Table 4.8 indicate that the deterioration brought about by SLD is higher for the privilege classes than it was for the responsiveness classes (Table 4.7); again, this was to be expected, since the privilege class has a greater impact on the final cost of a review than the responsiveness class, since $\lambda_p^a > \lambda_r^a$.

As we can see from Tables 4.9 and 4.10, the situation is completely reversed when considering the Rand policies: SLD performs consistently well, across all bins and cost structures, be it for responsiveness or privilege.

Class	Prevalence	Quartile	Used for
C21	0.032	Low	R+P
M12	0.032	Low	R+P
M132	0.033	Low	R+P
E12	0.034	Low	R+P
E212	0.034	Low	R+P
M131	0.035	Low	R+P
C24	0.040	Medium-Low	R+P
GCRIM	0.040	Medium-Low	R+P
GVIO	0.041	Medium-Low	R+P
C13	0.047	Medium-Low	R
GDIP	0.047	Medium-Low	R+P
C31	0.050	Medium-Low	R+P
C17	0.052	Medium-High	R+P
E21	0.054	Medium-High	R+P
C181	0.054	Medium-High	R+P
M141	0.059	Medium-High	R+P
M11	0.061	Medium-High	R+P
C18	0.066	Medium-High	R+P
M13	0.067	High	R+P
GPOL	0.071	High	R+P
C152	0.091	High	R+P
C151	0.102	High	R+P
M14	0.106	High	R+P
ECAT	0.149	High	R+P
C15	0.189	High	P
MCAT	0.255	High	P
GCAT	0.297	High	P
CCAT	0.474	High	P

Table 4.6: The RCV1-v2 classes that we use in our experiments, binned into quartiles according to prevalence value. The last column indicates whether we use the class to represent responsiveness (R), privilege (P), or both (R+P). We use these quartiles to bin our results in Tables 4.7, 4.8, 4.9, 4.10.

Λ	Bin	PL			ALvRS			ALvUS			ALvRUS		
		Pre-SLD	Post-SLD		Pre-SLD	Post-SLD		Pre-SLD	Post-SLD		Pre-SLD	Post-SLD	
Λ_1	Low	26,230	26,288‡	+0.22%	25,796	12,125	-112.74%	21,395	13,106	-63.24%	10,834	14,541	+25.49%
	Med-Low	40,001	40,056‡	+0.14%	44,419	23,677	-87.60%	35,639	26,164	-36.22%	20,245	25,562	+20.80%
	Med-High	38,936	38,943‡	+0.02%	32,438	25,167	-28.89%	28,872	24,368	-18.48%	16,885	29,179	+42.13%
	High	49,190	49,193‡	+0.01%	54,839	53,807‡	-1.92%	44,137	37,600	-17.38%	31,597	64,734	+51.19%
Λ_2	Low	8,955	9,002‡	+0.53%	5,262	5,668‡	+7.16%	3,607	4,289‡	+15.90%	3,149	7,062	+55.40%
	Med-Low	12,449	12,446‡	-0.03%	10,715	11,378‡	+5.82%	6,948	9,386‡	+25.97%	5,639	13,329	+57.69%
	Med-High	14,426	14,433‡	+0.05%	9,638	12,689	+24.05%	6,972	6,288	-10.88%	5,641	12,642	+55.37%
	High	16,357	16,412‡	+0.33%	15,976	26,426	+39.54%	9,355	9,234‡	-1.32%	8,634	25,414	+66.02%
Λ_3	Low	3,181	3,195‡	+0.45%	1,383	958	-44.43%	1,951	2,050‡	+4.83%	963	1,869	+48.46%
	Med-Low	4,501	4,500‡	-0.03%	2,442	1,826	-33.74%	3,087	5,097‡	+39.43%	1,674	1,840	+9.01%
	Med-High	5,603	5,602‡	-0.03%	2,459	2,084	-18.02%	3,669	9,310‡	+60.59%	1,851	6,879	+73.08%
	High	7,986	7,970‡	-0.19%	4,682	10,072‡	+53.51%	6,289	17,350‡	+63.75%	4,025	24,406	+83.50%

Table 4.7: Average MINECORE overall costs with responsiveness classes binned by prevalence value. A positive increment indicates higher costs resulting from the application of SLD. Superscript † and ‡ denote whether the Post-SLD results are not statistically significantly different from the Pre-SLD results, according to a Wilcoxon signed-rank test at different confidence levels: symbol † indicates $0.001 < p < 0.05$, while ‡ indicates $p \geq 0.05$.

Λ	Bin	PL			ALvRS			ALvUS			ALvRUS		
		Pre-SLD	Post-SLD		Pre-SLD	Post-SLD		Pre-SLD	Post-SLD		Pre-SLD	Post-SLD	
Λ_1	Low	30,904	31,097‡	+0.62%	30,142	19,869	-51.704%	24,815	18,459	-34.426%	14,180	25,302	+43.956%
	Med-Low	41,944	41,837‡	-0.26%	46,985	34,855	-34.800%	37,836	31,207†	-21.240%	23,771	45,192	+47.401%
	Med-High	41,004	41,159‡	+0.38%	42,307	34,937†	-21.095%	33,776	23,078	-46.356%	21,788	39,309	+44.572%
	High	39,971	39,921‡	-0.12%	38,626	26,842	-43.901%	33,216	27,003	-23.005%	19,935	27,806	+28.305%
Λ_2	Low	11,284	11,313‡	+0.25%	8,215	9,759†	+15.81%	4,385	4,698‡	+6.67%	3,833	10,025	+61.76%
	Med-Low	14,111	14,148‡	+0.26%	12,914	17,108†	+24.52%	7,221	7,871‡	+8.26%	6,281	17,933	+64.97%
	Med-High	13,984	14,020‡	+0.26%	10,070	17,376	+42.04%	6,109	6,634‡	+7.92%	5,407	17,559	+69.21%
	High	12,957	12,970‡	+0.10%	10,303	12,921	+20.26%	8,224	8,954‡	+8.16%	6,861	13,732	+50.03%
Λ_3	Low	4,559	4,564‡	+0.11%	2,113	2,106†	-0.31%	3,030	11,085‡	+72.66%	1,642	10,235†	+83.952%
	Med-Low	5,791	5,776‡	-0.25%	3,300	5,659†	+41.68%	4,542	17,351‡	+73.82%	2,687	18,538	+85.50%
	Med-High	5,996	6,003‡	+0.12%	3,158	6,252†	+49.48%	4,362	4,219‡	-3.40%	2,578	7,388	+65.10%
	High	5,119	5,119‡	-0.00%	2,546	2,120	-20.10%	3,346	3,144‡	-6.44%	1,823	2,011	+9.35%

Table 4.8: Same as Table 4.7, with privilege classes binned by prevalence.

Λ	Bin	Rand(RS) Pre-SLD	Rand(RS) Post-SLD		Rand(US) Pre-SLD	Rand(US) Post-SLD		Rand(RUS) Pre-SLD	Rand(RUS) Post-SLD	
Λ_1	Low	40,902	7,773	-426%	28,551	11,200	-155%	26,616	6,875	-287%
	Med-Low	67,945	15,605	-335%	47,435	20,328	-133%	46,674	13,261	-252%
	Med-High	56,956	13,192	-332%	37,320	19,270	-94%	36,330	10,872	-234%
	High	86,633	21,005	-312%	51,832	28,479	-82%	55,571	18,235	-205%
Λ_2	Low	8,207	1,162	-606%	4,915	2,366	-107%	5,757	1,513	-280%
	Med-Low	15,548	2,696	-476%	8,843	4,395	-101%	10,713	3,137	-241%
	Med-High	14,615	2,519	-480%	8,409	4,914	-71%	9,494	3,022	-214%
	High	20,463	3,969	-415%	11,420	6,887	-65%	13,312	4,588	-190%
Λ_3	Low	2,218	779	-185%	2,217	1,306	-70%	1,515	721	-110%
	Med-Low	3,939	1,599	-146%	3,545	2,404	-47%	2,718	1,407	-93%
	Med-High	3,958	1,697	-133%	3,998	3,011	-33%	2,741	1,553	-76%
	High	6,291	2,870	-119%	6,587	4,704	-40%	4,881	2,663	-83%

Table 4.9: Same as Table 4.7, but with different policies for generating the training sets.

Λ	Bin	Rand(RS) Pre-SLD	Rand(RS) Post-SLD		Rand(US) Pre-SLD	Rand(US) Post-SLD		Rand(RUS) Pre-SLD	Rand(RUS) Post-SLD	
Λ_1	Low	46,015	8,946	-414%	32,852	12,249	-168%	30,473	7,605	-301%
	Med-Low	74,367	15,846	-369%	47,843	21,558	-122%	49,346	13,611	-262%
	Med-High	70,036	14,447	-385%	42,347	19,984	-112%	44,094	12,380	-256%
	High	62,800	16,899	-272%	41,751	23,436	-78%	41,359	14,431	-187%
Λ_2	Low	8,903	1,153	-672%	5,820	2,237	-160%	6,440	1,455	-343%
	Med-Low	16,300	2,445	-566%	9,509	4,443	-114%	11,479	2,941	-290%
	Med-High	14,419	2,298	-527%	7,624	4,337	-76%	9,060	2,759	-228%
	High	17,538	3,760	-366%	9,735	6,484	-50%	11,300	4,350	-160%
Λ_3	Low	2,889	1,153	-150%	3,334	1,910	-75%	2,149	1,077	-100%
	Med-Low	4,569	2,025	-126%	4,761	3,295	-44%	3,466	1,862	-86%
	Med-High	4,439	1,740	-155%	4,624	3,044	-52%	3,310	1,634	-102%
	High	4,385	1,917	-129%	3,827	3,070	-25%	2,963	1,703	-74%

Table 4.10: Same as Table 4.8, but with different policies for generating the training sets.

In conclusion, the results of this analysis confirms that no matter the class prevalence value and the cost structure, SLD tends to have a clear negative effect on the quality of the posteriors generated by classifier trained via active learning techniques.

Visualising the behaviour of different selection strategies

Let us now move to Point 2, i.e., visualizing visualising the effects of the different selection strategies. As we have mentioned before, the results we have just commented say that the reasons behind the failure of SLD to improve the posteriors have to be found in the document selection criteria enacted by our three original active learning strategies. In order to better understand how these different strategies select their documents from the pool P, we provide a visualization of the first 1000 documents that each policy selects: in order to do this, we remove the initial seed set S, use the LSA algorithm (Landauer et al., 1998) on the TF-IDF matrix to reduce its dimensionality to the

two largest singular values, pick two random classes (namely the C17 and M14 classes of RCV1-v2), and show the corresponding scatter plots for the two components in Figure 4.1. The plots show, with different colours, the positive and negative documents of $\mathcal{P}$, and the positive and negative documents that the given strategy selected to be included in $\mathcal{L}$.[6]

It is indeed interesting to see how the ALvRS policy selects mostly positive instances, and very few negative ones, and that all of these positive instances tend to come from the same region of the instance space, thereby bringing about a training set characterised by very little diversity; this pattern is especially evident for the C17 class, and is less evident but still present for the M14 class. This is a visualisation of the sampling bias brought about by ALvRS (see also (Dasgupta, Hsu, 2008, §2) and (Krishnan et al., 2021)). The same visualisation also shows how the *Rand*(RS) policy selects instead the (same amount of) positives in a much more uniformly distributed and unbiased way. Similar effects, although less marked, can be noted for ALvRUS and ALvUS (and for the corresponding Rand policies). The plots for the passive learning (PL) strategy show instead the intrinsic limitations of this policy in TAR contexts (also highlighted in (Cormack, Grossman, 2014) for one-phase TAR systems); given the substantial imbalance between the positive and negative examples, a raw random sampling of the data results in the selection of very few positive documents; this is evident for both C17 and M14.

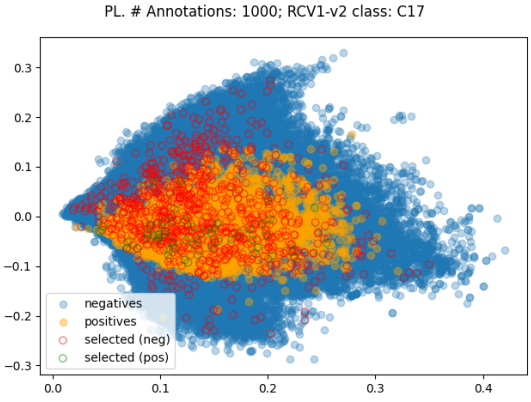
Analysing the distributions of the posteriors

Let us now discuss Point 3. Despite the fact that the LSA plots can help us visualise the effects of the different selection strategies, they cannot tell us much about the consequences of each strategy and why these strategies cause SLD to fail in delivering better-quality posterior probabilities. A much more helpful insight might instead emerge by plotting the distributions of the Pre-SLD and Post-SLD posteriors of the unlabelled documents as returned by the classifiers trained via the different selection strategies that we consider. For doing this we pick one of the two classes of our previous example (the C17 class) and plot (see Figures 4.3 and 4.4) the above-mentioned distributions for classifiers trained on 2,000 documents and on 23,149 documents, respectively. We also show the CCAT class, one of the most populated RCV1-v2 classes, for comparison (see Figure 4.5).[7] We use a logarithmic scale for the Y axis as this helps to visualise the distributions better.

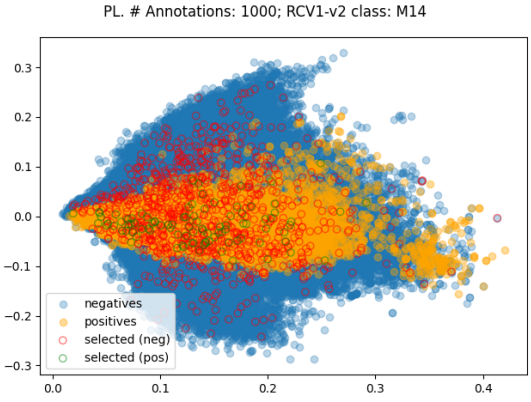
Looking at Figure 4.4 we can observe that the Pre-SLD distributions for all three AL policies are extremely skewed towards zero (i.e., most of the mass of the probability distribution is close to the zero value), whereas the distributions for the respective Rand policies are more uniformly spread across the [0,1] interval. The hypothesis for the cause of SLD's extremisation of the posteriors distribution might then arise from these plots. The SLD algorithm iteratively updates the posterior

[6]Notice that given the scale and the density of the documents, it may look as if some negative selected points are among the positive ones. This is just a limitation of the 2-D projection.

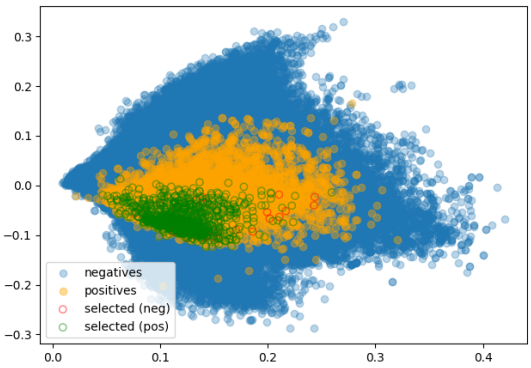
[7]Notice that for the CCAT class we only plot the distributions for the AL classifiers. Distributions for the *Rand* classifiers were fairly similar and thus not very interesting.


PL. # Annotations: 1000; RCV1-v2 class: C17
negatives
positives
selected (neg)
selected (pos)

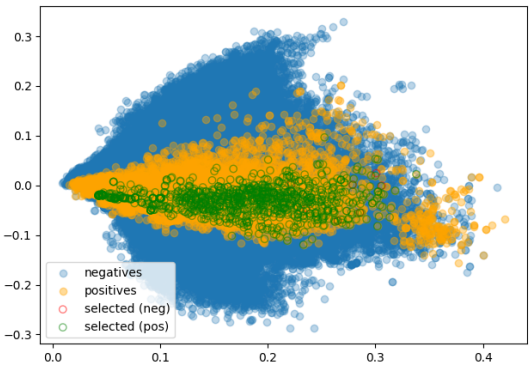
(a)


PL. # Annotations: 1000; RCV1-v2 class: M14
negatives
positives
selected (neg)
selected (pos)

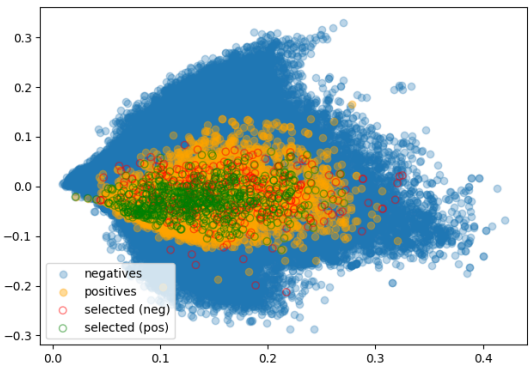
(b)


ALvRS. # Annotations: 1000; RCV1-v2 class: C17
negatives
positives
selected (neg)
selected (pos)

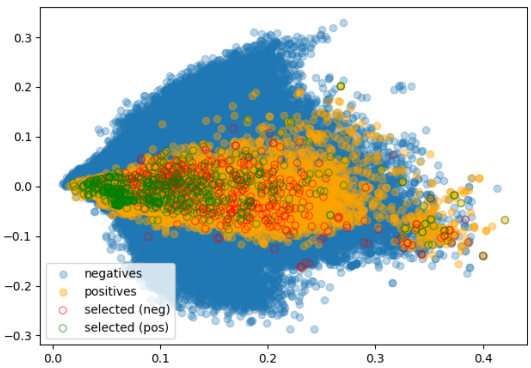
(c)


ALvRS. # Annotations: 1000; RCV1-v2 class: M14
negatives
positives
selected (neg)
selected (pos)

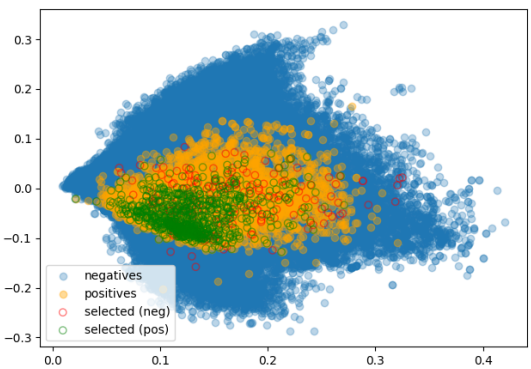
(d)


ALvUS. # Annotations: 1000; RCV1-v2 class: C17
negatives
positives
selected (neg)
selected (pos)

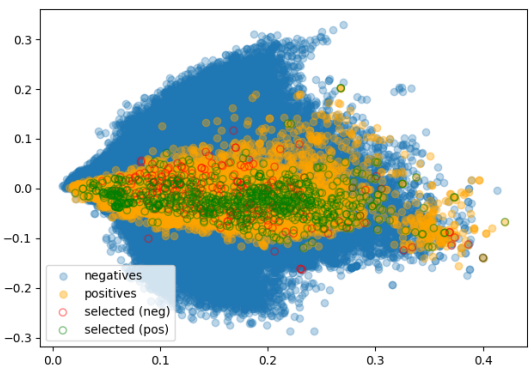
(e)


ALvUS. # Annotations: 1000; RCV1-v2 class: M14
negatives
positives
selected (neg)
selected (pos)

(f)


ALvRUS. # Annotations: 1000; RCV1-v2 class: C17
negatives
positives
selected (neg)
selected (pos)

(g)


ALvRUS. # Annotations: 1000; RCV1-v2 class: M14
negatives
positives
selected (neg)
selected (pos)

(h)

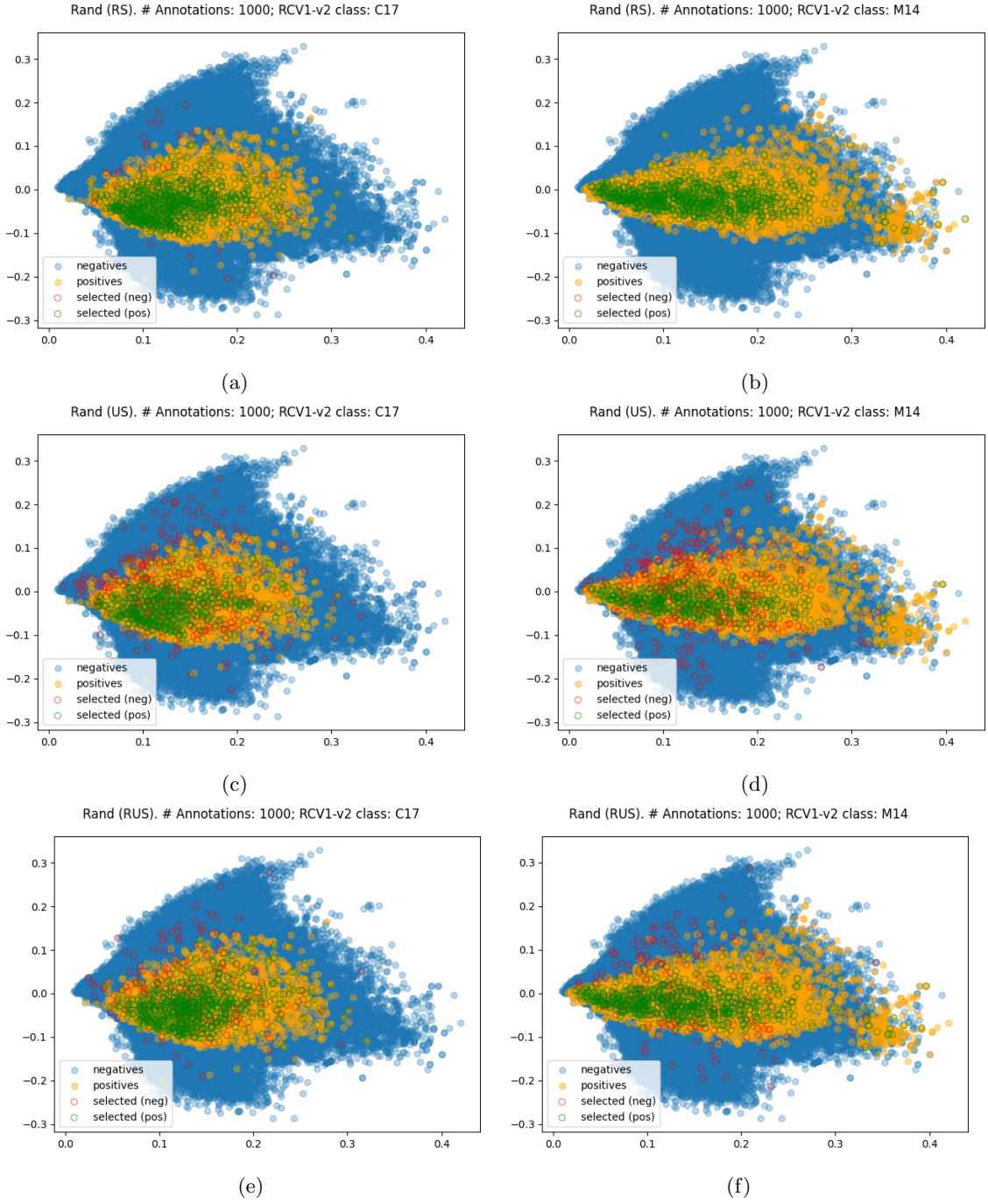


Figure 4.2: Same as Figure 4.1, but with the Rand policies in place of the original passive and active learning policies.

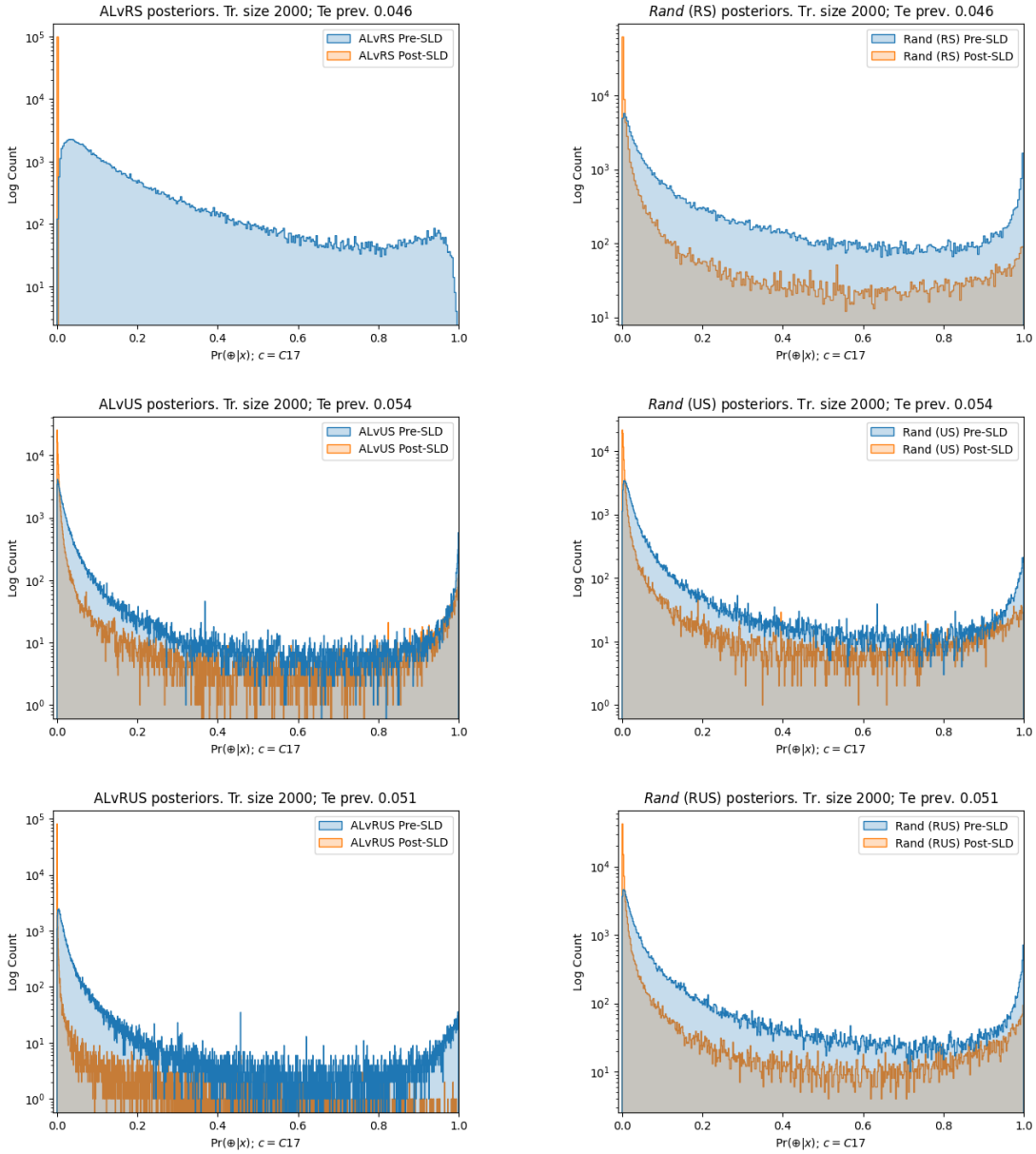


Figure 4.3: Distribution of the posteriors of the unlabelled documents generated by classifiers trained with various training document selection policies (indicated in the captions above each subfigure), using a set of $|\mathcal{L}|$=2,000 training documents.

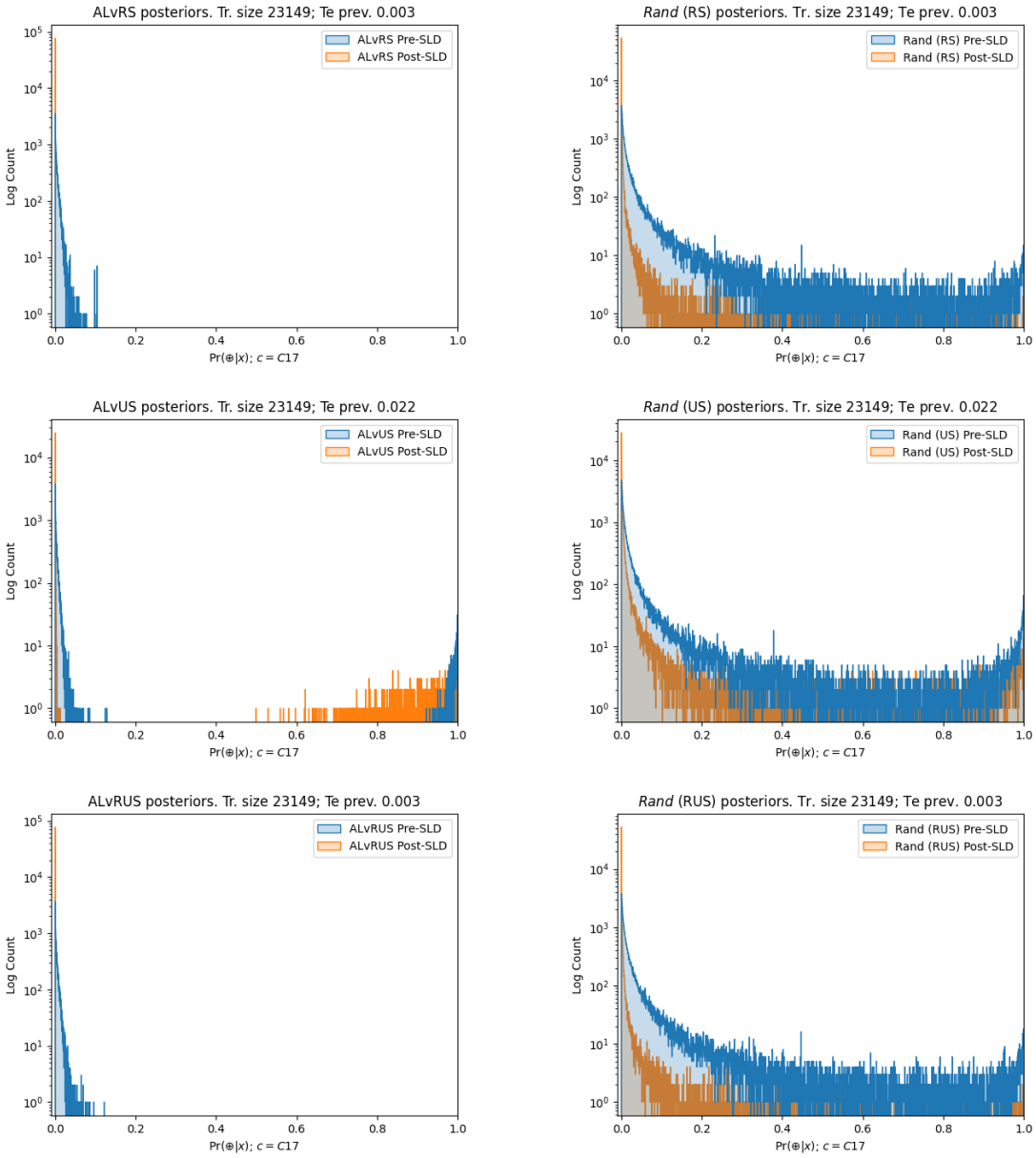


Figure 4.4: As Figure 4.3, but with 23,149 training documents instead of 2,000.

and prior probabilities using equations

$$\hat{\Pr}_U^{(s)}(y) = \frac{1}{U}\sum_{\mathbf{x}\in U}\overset{(s-1)}{\Pr}(y|\mathbf{x})$$

$$\Pr^{(s)}(y|\mathbf{x}) = \frac{\frac{\hat{\Pr}_U^{(s)}(y)}{\Pr_L(y)}\cdot\overset{(0)}{\Pr}(y|\mathbf{x})}{\sum_{y\in Y}\frac{\hat{\Pr}_U^{(s)}(y)}{\Pr_L(y)}\cdot\overset{(0)}{\Pr}(y|\mathbf{x})} \tag{4.1}$$

From these equations we notice that

$$\lim_{\hat{\Pr}_U^{(s)}(y)\to 0}\Pr^{(s)}(y|x) = 0 \tag{4.2}$$

That is, when the average of $\Pr(\oplus|\mathbf{x})$ is close to 0, then we iteratively drag the distribution towards 0. Indeed, this would be the maximisation of the expectation we can make given the data: i.e., there is likely no positive item remaining (which is also not too far from the truth for the ALvRS and ALvRUS policies, as we can see from the test class prevalence value (Te prev.) in the plots). Yet, this also results in pushing an already skewed posteriors distribution even more toward the extreme where most of the probability mass already lies, producing a distribution of probabilities composed almost entirely of zero values.

SLD does not skew the posteriors distribution in case of more balanced classes, e.g., for the C17 and CCAT classes, at least when the L is still small, e.g., $|L| = 2000$ (see Figure 4.3). In these cases SLD does not shift the distribution towards its extremes, but it is also seemingly not doing much (indeed, for the ALvUS the two distributions are almost indistinguishable from each other). Notice that CCAT is one of the most populated RCV1-v2 classes. This means that, even when we draw many positives from P, we are still not going to generate a too extreme PPS in the first iterations. Indeed, most of the positive documents would still be in the unlabelled set U. However, this also brings SLD outside of its main scope of application, i.e., correcting high PPS: the algorithm will thus bring no significant benefit.

So, if the *Rand* and AL strategies are working with the same training/test class prevalence values, why are the classifier output probabilities so much more skewed in the latter case and not in the former? In order to understand this, we need to consider the sampling bias (which, again, is the main difference between the *Rand* and the AL policies): as a matter of fact, due to how the policies work (especially for ALvRS) we will tend to annotate many positive but similar items (see Figure 4.1) and the classifier will be trained on these positive examples only. This in turn will result in the classifier being particularly good at classifying those type of positives, as well as being particularly sure of the negative label of the other documents; since the *Rand* policies do not suffer from sampling bias, this does not happen with these pseudo-oracle policies. This is indeed what we see in Figure 4.4.

If we consider the AL strategy used and the overall prevalence value of the class in P we can then predict when SLD will perform a correct rescaling of the posterior probabilities or not:[8]

1. If the prevalence in P is low, and we use a strategy based on relevance sampling, we will remain with a very low number of positive items in the unlabelled set. Plus, since our classifier is

[8] Of course this is not possible in real scenarios, where we do not know $\Pr(y)$.

suffering from sampling bias, its predictions will be particularly skewed towards the negative class.

2. If the prevalence in P is low, and we use a strategy based on uncertainty sampling, we expect to have less skewed distributed posteriors up until a certain size of annotated documents. Eventually, though, the number of positives will shrink and, with our classifier still suffering from sampling bias, we will end up in the previous scenario.

3. Finally, if the prevalence in P is fairly high and the number of annotations relatively low, then we expect the posteriors from the ALvRS/ALvRUS policy to be not very skewed.

SLD will eventually fail in all three of these scenarios, either because we have no PPS between the labelled and unlabelled sets or because sooner or later our classifiers will suffer from a strong sampling bias, which paired to the shrinking number of positives, will bring to the "posterior extremisation" phenomenon that we witness in the plots.

We then conclude this analysis arguing that SLD cannot be used "as is" in contexts where the training and test sets are resulting from AL strategies such as ALvRS, ALvRUS and ALvUS. We propose to further investigate this issue in future works, to explore possible solutions that might enable the usage of the SLD algorithm.

4.5 Discussion

In this chapter we have explored and analysed different strategies for improving the performance of the MINECORE risk minimisation framework for technology-assisted review in e-discovery (Oard et al., 2018). Specifically, we have concentrated on strategies for improving the posterior probabilities that Step 1 of the MINECORE workflow provides as input to Step 2 of the same workflow, an improvement that we measure in terms of reduction in the overall cost of the review process that MINECORE brings about. We have formulated two research questions (RQ1 and RQ2), that correspond to two possible strategies for improving these probabilities.

RQ1 poses the problem of which policy is the best for training the two classifiers that return these posterior probabilities; the policies we consider are passive learning (PL), active learning via relevance sampling (ALvRS), active learning via uncertainty sampling (ALvUS), and a combination of the two latter policies that we call active learning via relevance and uncertainty sampling (ALvRUS). The results of our experiments show that ALvRUS is unquestionably the best such policy, thus indicating that reaching a balance between "exploration" (the US component) and "exploitation" (the RS component) proves a key step in generating better training sets for MINECORE. Passive learning proves instead the worst such policy, which confirms the analogous results obtained for one-phase TAR systems (see Cormack, Grossman (2014)).

In RQ2 we instead pose the problem whether an application of the well-known SLD algorithm (Saerens et al., 2002), whose goal is to improve the quality of the posterior probabilities in contexts affected by prior probability shift (PPS), could indeed prove beneficial for MINECORE. Here, the results are less uniform, and show that the application of SLD often decreases (instead of increasing) the quality of the posterior probabilities, especially when some active learning policy has been used to train the classifiers; unfortunately, SLD always brings about a deterioration in the quality of these probabilities when ALvRUS (that had proved the "winning" policy for RQ1) has been used to train the classifiers. Additional experiments that we have run unequivocally show that

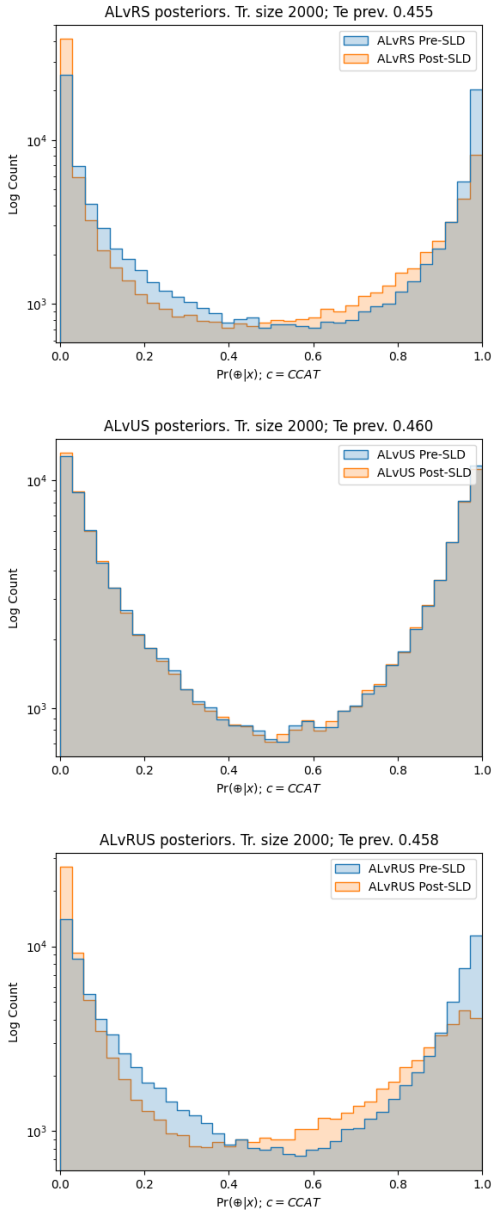


Figure 4.5: ALvRS, ALvUS, and ALvRUS posteriors for the positive class, before and after the application of SLD. $|\mathcal{L}| = 2000$.

the reason why SLD tends to perform badly when the classifiers have been trained via active learning, is that active learning generates not only prior probability shift (which SLD has been designed for) but also sampling bias (for which SLD is not equipped). This raises the question whether using other active learning strategies that attempt to maximise diversity / representativeness of the training data (thereby doing away with sampling bias) could allow SLD to be used profitably; this proves a difficult path to follow, though, since active learning techniques that maximise diversity tend to be too expensive, from a computational point of view, for the typical problem sizes encountered in TAR.

However, when we take RQ1 and RQ2 altogether, the answer returned by our experiments is unequivocal: the best course of action consists of (i) using ALvRUS for training the classifiers, and (ii) *not* using SLD in the attempt to further improve the posterior probabilities.

In future work we would like to investigate (for RQ1) the use of active learning techniques that are both computationally efficient and diversity-preserving, as well as (for RQ2) variants of the SLD algorithm that are also robust to the presence of sampling bias.

Chapter 5

SAL_τ: Efficiently stopping TAR via SLD

In Chapter 4, we have unsuccessfully tried to leverage the SLD algorithm in an AL process in order to improve both the posterior and prior estimates of a classifier. SLD showed detrimental behaviours, bringing to an extremization of the probability distribution, i.e. the fact that most, if not all, posterior probabilities are pushed to either 1 or 0. In this chapter, we give an extensive analysis of the phenomenon, and propose a solution to adjust and enable the SLD algorithm in AL scenarios. We leverage our modified version of SLD (called "SLD for Active Learning", or SAL_τ) to create a new stopping rule for TAR systems: we show that our algorithm is able to deliver good prevalence estimations, which in turn can better quantify the number of relevant items remaining in the unlabelled set; we use the new prevalence estimates to stop the TAR process well in advance with respect to previous state-of-the-art baselines, while keeping the same capabilities of reaching the target recall. A preliminary analysis on SLD and active learning was published in Esuli et al. (2022). The follow-up paper, which introduces SAL_τ, is instead currently under review at the Data Mining and Knowledge Discovery journal.[1] The code to reproduce our experiments is publicly available at `https://github.com/levnikmyskin/salt`.

5.1 Introduction

As previously mentioned in Chapters 1 and 2, one of the most challenging issues in TAR applications is the so-called "when-to-stop" problem: deciding when to stop the AL process, in order to jointly minimize the annotation effort and satisfy the information need, e.g., finding all (or almost all) documents relevant to a research question. Recently, IR literature has proposed many stopping methods (Cormack, Grossman, 2016a; Li, Kanoulas, 2020; Oard et al., 2018; Yang et al., 2021a): the when-to-stop issue is usually tackled by either changing the sampling policy of the AL algorithm, by crafting task-specific heuristics and/or by estimating the current achieved recall (see Chapter 2, Sections 2.3.4 and 2.3.5). In this chapter, we focus on the latter approach and propose a new technique based on the Saerens-Latinne-Decaestecker (SLD) algorithm (Saerens et al. (2002), see also Section 2.4), adapting it to the CAL workflow typically adopted in TAR processes.

[1] `https://www.springer.com/journal/10618/?IFA`

This chapter is structured as follows: in Section 5.2 we analyze the shortcomings of the SLD algorithms when used "as-is" in AL scenarios. We then propose a solution to this problem, our own method in Section 5.3. Experiments and results are discussed in Sections 5.4 and 5.5. Section 5.6 concludes.

5.2 An analysis of the shortcomings of SLD in active learning scenarios

Chapter 4 (as well as our work in Esuli et al. (2022)) has shown how SLD can have disastrous behaviours in AL contexts, leading to an extremization of the posterior probability distribution, i.e., the fact that most (if not all) posterior probabilities $\Pr(\oplus|x)$ are dragged to either 0 or 1. Experiments from Esuli et al. (2022) and Chapter 4 show that when SLD is applied to the *Rand*(*pol*) version of an AL policy, be it either ALvRS or ALvUS, the phenomenon of corruption of posteriors does not happen. The cause of the issue is thus to be found in the document selection policy, and in the fact that both ALvRS and ALvUS produce a *sampling bias* (Dasgupta, Hsu, 2008; Krishnan et al., 2021), which leads to a dataset shift where $\Pr_L(y) \neq \Pr_U(y)$ and $\Pr_L(x|y) \neq \Pr_U(x|y)$.

Sampling bias emerges from AL algorithms due to both the selection policy *pol* and the initial seed S. S is usually very small (in many TAR applications, it may consist of a single positive instance). Considering the ALvRS policy,[2] the classifier trained on S will have high confidence on documents similar to the few ones contained in S. As the AL process continues, the training set L will diverge from the underlying data distribution. A classifier trained on such a biased training set can hardly make sense of the whole document pool. Indeed, Dasgupta, Hsu (2008, §2) show that the classifier can be overly confident in attributing the negative label to a cluster of data which actually contains several positive instances. A visual representation of the sampling bias is shown in Figure 4.1c in Chapter 4, which shows how the document selection is extremely focused on one small region of the positive instances.

When it comes to the classifier capability of estimating the prevalence of relevant documents in the unlabelled set U, this means that its estimates are going to be much lower than those of a classifier trained on a random and representative sample of the same population: we can see this in Table 5.1, where we compare the prevalence estimates of a calibrated SVM classifier trained on the ALvRS training set (at different sizes) and the same classifier trained on a controlled random sample of the pool, with same size and same positive prevalence (we call ALvRS SVM and *Rand*(RS) SVM, the two SVMs trained on the two different training sets); these results were measured on the datasets generated in the experiments for Chapter 4. The estimate is the average of the posterior probabilities for the relevant class $\Pr(\oplus|x)$ for all $x \in U$. The last two columns of the table report the true prevalence of the positive class in L and U, showing the strong PPS generated by AL techniques; clearly, the shift is stronger if the prevalence of the positive class $p_P(\oplus)$ is already fairly low to start with (which is usually the case in many TAR applications). In this scenario, the classifier usually overestimates $p_U(\oplus)$, given its bias on $p_L(\oplus)$ prevalence, which is what we see for *Rand*(RS) SVM. The values in the table seem to indicate that the ALvRS-based estimates are better than the *Rand*(RS) ones. We argue that this is actually due to the sampling bias: the classifier is very likely underestimating the prevalence of those positive clusters it does not know

[2] As mentioned several times throughout this thesis, ALvRS (or rather CAL) is the preferred AL policy in TAR applications. Hence, we usually write about (or give more attention to) ALvRS rather than the other AL policies in this chapter.

Size of L	$\hat{p}_U(\oplus)$ ALvRS	$\hat{p}_U(\oplus)$ *Rand*(RS)	$p_L(\oplus)$	$p_U(\oplus)$
2000	0.048	0.141	0.500	0.058
4000	0.065	0.158	0.709	0.040
8000	0.063	0.121	0.686	0.013
16000	0.008	0.064	0.405	0.003
23149	0.004	0.048	0.284	0.002

Table 5.1: Prevalence estimates of an SVM classifier trained on a ALvRS and *Rand*(RS) training set respectively, compared to true prevalences of the L and U sets. These results were measured on the datasets generated in the experiments for Chapter 4.

about; the end result is that the output prevalence estimate is much lower than the *Rand*(RS) and incidentally closer to the real prevalence of U.

In order to better visualize how sampling bias affects the AL trained classifier, we give a visual representation of this phenomenon on a synthetic dataset:

1. We generate an artificial dataset consisting of 10,000 data items with four clusters (blue, yellow, purple and pink in Figure 5.1). Blue and yellow clusters are the positive clusters (i.e., every item in these clusters has a positive $\oplus$ label); purple and pink clusters are the negative clusters (i.e., every item in these clusters has a negative $\ominus$ label). Notice that negative clusters are much more populated than positive ones (i.e., the overall positive prevalence is low);

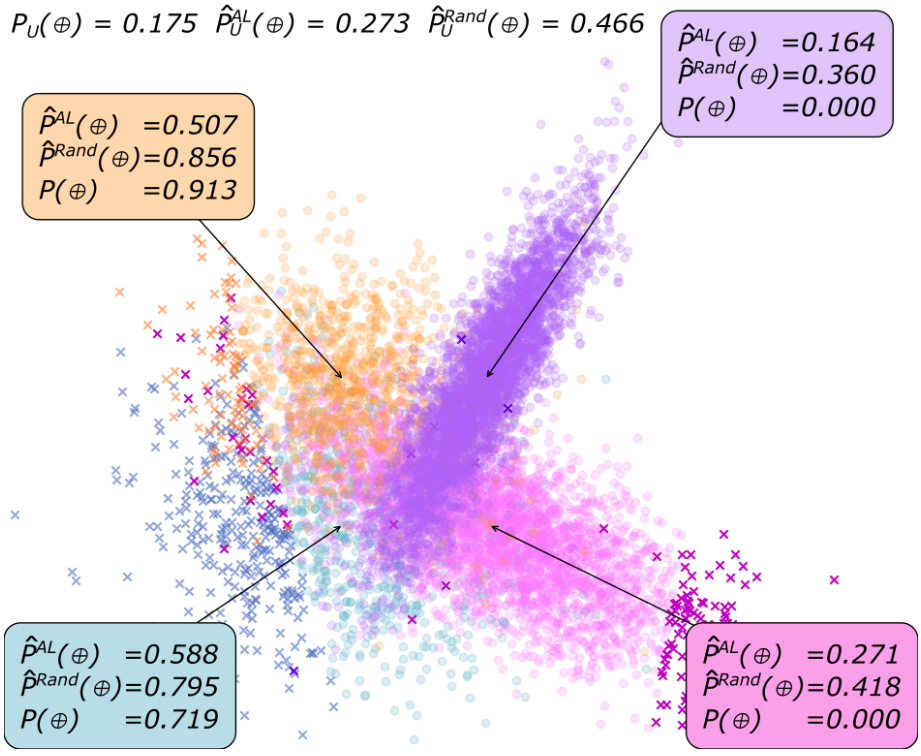
2. We start the active learning process with two positive items coming from one of the positive clusters and 10 negative items, randomly sampled from the negative clusters. We then annotate 500 documents with the ALvRS policy and generate an analogous training set with the *Rand*(RS) policy. The two training sets are shown with "X" markers in Figure 5.1a and 5.1b, for ALvRS and *Rand*(RS) respectively;

3. We show the estimated (and the true) proportion of positive items remaining in each cluster, for a Calibrated SVM trained on the ALvRS and *Rand*(RS) training sets respectively. Furthermore, we show in the title the true $p_U(\oplus)$ and estimated prevalence ($p_U^{\text{ALvRS}}(\oplus)$ and $p_U^{Rand}(\oplus)$) on the test set.

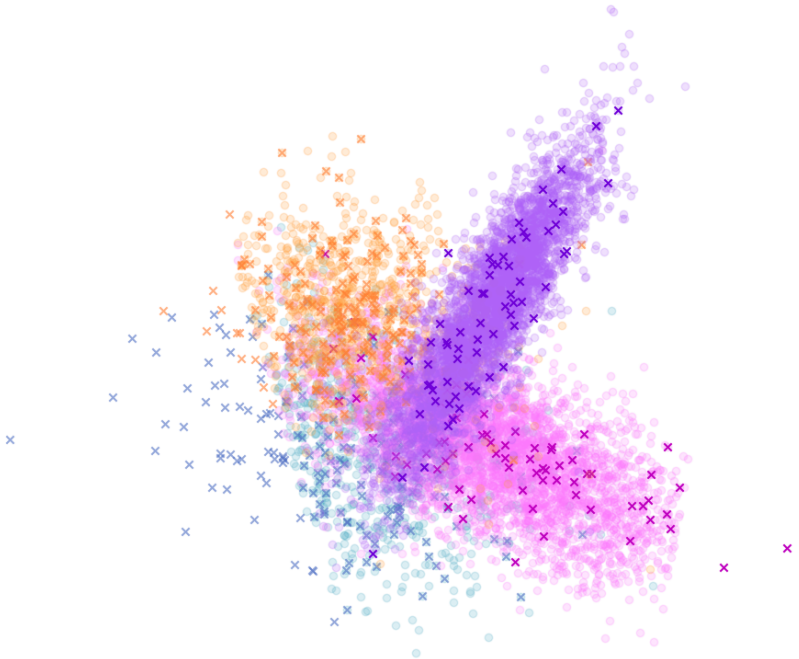
In Figure 5.1a we see how the AL process annotates a specific subregion of positive items. This in turn "misleads" the classifier to output a much lower prevalence than the *Rand*(RS) classifier for the clusters it has never seen during training; notice that, being the overall positive prevalence quite low, the prevalence estimates of the AL classifier seem better than the *Rand*'s.

5.2.1 How is sampling bias related to SLD failures in active learning contexts?

The reason why sampling bias is a key element in our analysis lies beneath the main assumptions made in SLD on the posterior probability distribution of the classifier: SLD reasonably assumes to be in the scenario represented by the *Rand*(RS) policy rather than the AL one. More precisely, it assumes that there is no dataset shift on the conditional probability $\Pr(x|y)$ between the labelled



(a) ALvRS training set (marked with X) and prevalence estimation of an SVM trained on this training set. We report a *Rand*(RS) trained classifier estimation as well for completeness. The blue and yellow clusters are the "positive" clusters, whereas the pink and purple ones are the "negative" ones.



(b) The *Rand*(RS) policy training set (marked with X), used for the *Rand* classifier in Figure 5.1a.

Figure 5.1: ALvRS and *Rand*(RS) applied to synthetic data.

set L and U, i.e. that $\Pr_L(x|y) = \Pr_U(x|y)$. If this assumption holds, the trained classifier will have a bias on L which will "uniformly" translate (i.e., in our artificial example, the bias is consistent for all regions of the graph) to the posterior probabilities $\Pr_U(\oplus|x)$ on the unlabelled set. In a PPS scenario, this means that the classifier estimate of the prevalence (i.e., the average of the classifier posteriors) will be closer to L, and that they can be "adjusted" consistently across the whole distribution. Nonetheless, when using an active learning policy such as ALvRS or ALvUS, we not only generate prior probability shift, but we also affect the distribution of the conditional probability $\Pr(x|y)$, such that $\Pr_L(x|y) \neq \Pr_U(x|y)$.[3]

Let us now focus on one of the key updates in SLD: the priors ratio (Line 13 of Algorithm 2), which is later multiplied by the posteriors. This is defined as:

$$\frac{\hat{\Pr}_U^{(s)}(\oplus)}{\Pr_L(\oplus)} \tag{5.1}$$

This linear relation is represented by the red line of Figure 5.2. When $\hat{\Pr}_U(\oplus) = \Pr_L(\oplus)$, the ratio is 1, and posteriors do not change. However, as $\hat{\Pr}_U(\oplus)$ drifts further away from $\Pr_L(\oplus)$ (recall that this latter quantity is constant for all iterations in SLD), the ratio becomes progressively smaller, resulting in a multiplication of $\Pr_U(\oplus|x)$ by a number very close to 0.[4] We argue this is one of the main culprits of degenerated outputs from SLD.

In other words, let us assume that $\Pr_L(x|y) = \Pr_U(x|y)$, and that L is then a representative sample of U. A classifier trained and biased on L will tend to shift any prevalence estimate toward $\Pr_L(\oplus)$. When the classifier estimated $\hat{\Pr}_U(\oplus)$ is lower (or higher) than $\Pr_L(\oplus)$, SLD deems the true $\Pr_U(\oplus)$ to be even lower (or higher). Indeed, this works very well when the previous assumption holds, e.g., for *Rand*(pol). As a matter of fact, SLD has been a state-of-the-art technique for prior and posterior probabilities adjustments in PPS for 20 years. However, ALvRS and similar techniques generate a $\Pr_L(x|y) \neq \Pr_U(x|y)$ type of shift (as well as PPS), and, as a result, we cannot apply SLD update with confidence: we should rather find a way to apply a milder correction, when possible, or no correction at all when we have no way of estimating how "far" we are from the former assumption.

5.3 Adapting the SLD algorithm to active learning

As discussed in the previous section, in AL scenarios, the priors ratio defined in the SLD algorithm should be milder in order to avoid extreme behaviours. A simple but possibly effective action is to directly add a correction factor τ to the priors ratio equation, so that:

- when $\tau = 1$, we get the original SLD algorithm;
- when $\tau = 0$, the ratio always equals 1, i.e., we do not apply any correction;
- all other intermediate values adjust the ratio, making it milder with respect to SLD original ratio.

[3] As noticed in Section 2.1.1, this is not properly a "concept shift": with concept shift we also assume that, given the same x, the label y might change between L and U. Moreover, concept shift also assumes that $\Pr_L(y) = \Pr_U(y)$; this is clearly not the case, as we also generate PPS (see also Moreno-Torres et al. (2012, §4.4)).

[4] This is the case for low prevalence scenarios, typical of TAR. The opposite case is the one with very high prevalence, in which the posteriors for the relevant class are all pushed to 1.

We thus define a correction to the priors ratio of SLD:

$$\delta = -\left[\tau \cdot \left(-\frac{\hat{\Pr}_U^{(s)}(y)}{\hat{\Pr}_U^{(0)}(y)} + 1\right) - 1\right] \tag{5.2}$$

the new ratio, which we call δ, will then be multiplied by the posteriors at every SLD iteration. We show how τ affects the slope of the ratio in Figure 5.2. We call our method "**S**LD for **A**ctive **L**earning" or SAL$_\tau$ for short. The complete algorithm is reported in Algorithm 3. In the next section we detail how we set the value of τ.

5.3.1 Estimating SAL$_\tau$ τ across active learning iterations

We have seen that SLD works on the output of a *Rand*(RS)-based classifier (see Esuli et al. (2022) and Chapter 4). We can build our estimate of τ for SAL$_\tau$ by measuring how much the ALvRS classifier posteriors are more or less distributed like those of a *Rand*(RS) classifier. The more ALvRS posteriors diverge from *Rand*(RS) ones, the milder SLD correction should be, up to the point where we do not use SLD at all.

Let us consider the posteriors for the relevant class, i.e., $\Pr(\oplus|x)$: we collect a vector $\mathcal{A}$ of posteriors $\Pr(\oplus|x)$ for all documents in U for the classifier trained on the ALvRS training set, and an analogous one $\mathcal{R}$ for the classifier trained on the *Rand*(RS) training set. We define τ as the cosine similarity between these two vectors:

$$\tau = \text{cosine similarity}(\mathcal{A}, \mathcal{R}) = \frac{\mathcal{A} \cdot \mathcal{R}}{||\mathcal{A}||\ ||\mathcal{R}||} \tag{5.3}$$

Since $\Pr(\oplus|x) \geq 0$ by definition of probability, the cosine similarity is naturally bounded between 0 and 1, a required property of our τ parameter. In other words, we apply the SLD update when AL posteriors are similar to posteriors for which we know the assumption made in SLD holds (i.e., $\Pr_L(x|y) = \Pr_U(x|y)$, see Section 5.2); we apply an accordingly milder correction the further the AL posteriors are from this assumption.

Equation 5.3 would be a good solution, as well as an impossible one, since the *Rand*(pol) policy requires knowledge of the labels (e.g., relevancy) for the entire data pool.

We thus resort to an heuristic formulation based on the evolution of the classifier during the iterations of the AL process.[5] Given the batch of documents B_i reviewed at the i-th iteration, we define $\mathcal{A}_{\phi_i}$ and $\mathcal{A}_{\phi_{i-1}}$ as:

$$\mathcal{A}_{\phi_i} = \Pr^{(i)}(\oplus, x) = \phi_i(x)|x \in B_i \tag{5.4}$$

$$\mathcal{A}_{\phi_{i-1}} = \Pr^{(i-1)}(\oplus, x) = \phi_{i-1}(x)|x \in B_i \tag{5.5}$$

i.e., the posteriors on B_i returned by the classifiers ϕ_i and ϕ_{i-1}. The τ parameter is then defined as the cosine similarity between $\mathcal{A}_{\phi_i}$ and $\mathcal{A}_{\phi i-1}$. The assumption we make is that:

- at early iterations, there will not likely be substantial differences between the two vectors (thus making the cosine similarity useless);

[5] This idea of leveraging previously annotated batches is more or less similar to what Callaghan, Müller-Hansen (2020) proposed in their method (see also Section 2.3.4).

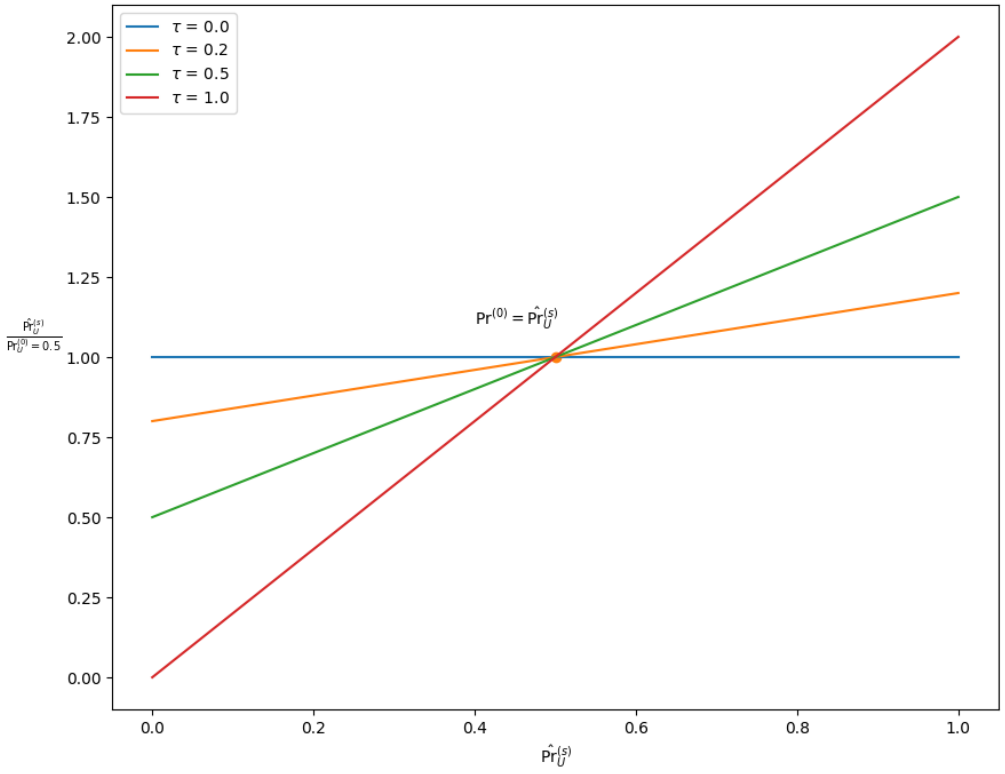


Figure 5.2: The τ-based correction to the priors ratio of SLD that we propose. The value of this ratio (i.e., the y axis) is multiplied by the posteriors during SLD iterations. Notice that when $\tau = 1$ we get the SLD original ratio, whereas when $\tau = 0$ we multiply the posteriors by 1, i.e. we do not change the classifier posteriors.

Algorithm 3: The SAL_τ algorithm. Changes with respect to SLD are highlighted.

Input : Class priors $\Pr_L(y_j)$ on L, for all $y_j \in Y$;
Posterior probabilities $\Pr(y_j|\mathbf{x}_i)$, for all $y_j \in Y$ and for all $\mathbf{x}_i \in U$;
Correction factor τ;

Output: Estimates $\hat{\Pr}_U(y_j)$ on U, for all $y_j \in Y$;
Updated posterior probabilities $\Pr(y_j|\mathbf{x}_i)$, for all $y_j \in Y$ and for all $\mathbf{x}_i \in U$;

```
// Initialization
```
$s \leftarrow 0$;
for $y_j \in Y$ **do**
  $\hat{\Pr}_U^{(s)}(y_j) \leftarrow \Pr_L(y_j)$; `// Initialize the prior estimates`
  for $\mathbf{x}_i \in U$ **do**
    $\Pr^{(s)}(y_j|\mathbf{x}_i) \leftarrow \Pr(y_j|\mathbf{x}_i)$; `// Initialize the posteriors`
  end
end
`// Main Iteration Cycle`
while *stopping condition = false* **do**
  $s \leftarrow s + 1$;
  for $y_j \in Y$ **do**
    $\hat{\Pr}_U^{(s)}(y_j) \leftarrow \frac{1}{|U|} \sum_{\mathbf{x}_i \in U} \Pr^{(s-1)}(y_j|\mathbf{x}_i)$; `// Update the prior estimates`
    $\delta \leftarrow -\left[\tau \cdot \left(-\frac{\hat{\Pr}_U^{(s)}(y)}{\hat{Pr}_U^{(0)}(y)} + 1\right) - 1\right]$;
    for $\mathbf{x}_i \in U$ **do**
      `// Update the posteriors`
      $\Pr^{(s)}(y_j|\mathbf{x}_i) \leftarrow \frac{\delta \cdot \Pr^{(0)}(y_j|\mathbf{x}_i)}{\sum_{y_j \in Y} \delta \cdot \Pr^{(0)}(y_j|\mathbf{x}_i)}$;
    end
  end
end
`// Generate output`
for $y_j \in Y$ **do**
  $\hat{\Pr}_U(y_j) \leftarrow \hat{\Pr}_U^{(s)}(y_j)$; `// Return the prior estimates`
  for $\mathbf{x}_i \in U$ **do**
    $\Pr(y_j|\mathbf{x}_i) \leftarrow \Pr^{(s)}(y_j|\mathbf{x}_i)$; `// Return the adjusted posteriors`
  end
end

- however, later on in the AL process this could help us obtain an estimate of how much sampling bias is affecting the classifier.

Let us consider the ALvRS policy, in a scenario similar to the one depicted in Figure 5.1a, that is, overall prevalence is low and we start with a small seed set: in the first iterations we are likely going to find many positive items; that is, when we compare ϕ_i and ϕ_{i-1} predictions on B_i they are likely to be similar, since the "clusters" of data available to ϕ_i are probably the same that were available to ϕ_{i-1}. However, as we review all the positive items in a cluster, the process is forced to explore items in the neighbour clusters. This is when the cosine similarity should be effective: by comparing the posterior distribution of ϕ_i to that of ϕ_{i-1} for the same B_i documents, we should be able to assess the impact of the new documents on the classifier. Indeed, if ϕ_i accessed previously unseen clusters, the posteriors on B_i might radically change and, in turn, roughly give us an estimate of how much sampling bias was affecting ϕ_{i-1} predictions.

Finally, let us consider one last issue: we said that in the first AL iterations $\mathcal{A}_{\phi_i}$ will likely be similar to $\mathcal{A}_{\phi_{i-1}}$, and their distance cannot be used as an indication of sampling bias. How can we establish when to use our method and when to fallback to the classifier posteriors? In lack of better solutions (which we defer to future works), we introduce a hyperparameter α:

- at every iteration i, we apply SAL_τ and obtain a new set of posterior probabilities $\Pr^{\text{SAL}_\tau}(\oplus|x)$ and a prevalence estimation $\Pr^{\text{SAL}_\tau}(\oplus)$ (computed as $\frac{\sum_x^X \Pr^{\text{SAL}_\tau}(\oplus|x)}{|X|}$);
- we measure the Normalized Absolute Error (NAE) between the estimated prevalence and the true prevalence of batch B_i, where $\text{NAE}(\Pr_{B_i}(y), \hat{\Pr}_{B_i}^{\text{SAL}_\tau}(y))$ is defined as (notice we used NAE in Chapter 3 as well, see Equation 3.10):

$$\text{NAE} = \frac{\sum_{j=1}^{Y} |\Pr_{B_i}(y) - \hat{\Pr}_{B_i}^{\text{SAL}_\tau}(y_j)|}{2(1 - \min_{y_j \in Y} \Pr_{B_i}^{\text{SAL}_\tau}(y_j))} \tag{5.6}$$

- if $\text{NAE} > \alpha$ we do not use our SAL_τ method.

The simple intuition behind this heuristic is that when the NAE between the true prevalence and the prevalence estimation of SAL_τ is too high, then the estimates of SAL_τ are likely going to be poor on the rest of the pool as well.

To recap, we give below an overview of how SAL_τ integrates into the active learning process (see Algorithm 4):

1. at each iteration i we employ an active learning policy, annotating a batch B_i of b documents, which are added to the training set L;
2. we train a classifier ϕ_i on L;
3. at each iteration $i > 1$, we compute the cosine similarity between the two vectors of scores $\mathcal{A}_{\phi_i} = \langle \Pr^{\phi_i}(\oplus|x) \ \forall \ x \in B_i \rangle$ and $\mathcal{A}_{\phi_{i-1}} = \langle \Pr^{\phi_{i-1}}(\oplus|x) \ \forall \ x \in B_i \rangle$. The cosine similarity will be used as the τ parameter in SAL_τ;
   (a) we obtain a new set of posterior probabilities $\Pr^{\text{SAL}_\tau}(\oplus|x)$ and a new prevalence estimate $\Pr^{\text{SAL}_\tau}(\oplus)$ on U, using SAL_τ on the posteriors coming from ϕ_{i-1};

4. we compute NAE between SAL$_\tau$ prevalence estimate for B_i and the true prevalence of B_i. If this is lower than a threshold α, we consider SAL$_\tau$-based probabilities to be the correct ones, otherwise we fall back to ϕ_i-based probabilities. In a TAR process this means that we can use SAL$_\tau$-based probabilities to estimate the recall and decide when to stop reviewing documents.

Algorithm 4: SAL$_\tau$ integration within an active learning process

Input: Pool of documents P to be reviewed; Active learning policy a; Initial seed set S; Batch size b; Budget t; threshold value α;

$i \leftarrow 0$;
$L \leftarrow S$;
$\phi_i \leftarrow$ `train_clf(`L`)`;
$B_i \leftarrow$ `select_via_policy(`ϕ_i, a, P, b`)`;
$L \leftarrow L \cup B_i$;
while $|L| < t$ **do**
  $i \leftarrow i + 1$;
  $U \leftarrow P \setminus L$;
  $\phi_i \leftarrow$ `train_clf(`L`)`;
  $A_i \leftarrow \langle \Pr^{\phi_i}(\oplus|x)\ \forall x \in B_{i-1} \rangle$;
  $A_{i-1} \leftarrow \langle \Pr^{\phi_{i-1}}(\oplus|x)\ \forall x \in B_{i-1} \rangle$;
  $\tau \leftarrow$ `cosine_similarity(`A_i, A_{i-1}`)`;
  $\Pr^{\text{SAL}_\tau}(\oplus|x) \leftarrow$ `SAL`$_\tau$`(`$\Pr^{\phi_{i-1}}(\oplus|x)\ \forall x \in U, \hat{\Pr}_L^{\phi_{i-1}}(\oplus), \tau$`)`;
  if `NAE(`$\Pr_{B_{i-1}}(\oplus), \hat{\Pr}_{B_{i-1}}^{\text{SAL}_\tau}(\oplus)$`)` $< \alpha$ **then**
    `// The new posteriors can be used to, e.g., estimate recall`
    $\Pr(\oplus|x) \leftarrow \Pr^{\text{SAL}_\tau}(\oplus|x)\ \forall x \in U \cup \Pr^{\phi_i}(\oplus|x)\ \forall x \in L$;
    $\hat{R} \leftarrow \frac{\sum_x^L \Pr(\oplus|x)}{\sum_x^P \Pr(\oplus|x)}$;
  end
  $B_i \leftarrow$ `select_via_policy(`ϕ_i, a, P, b`)`;
  $L \leftarrow L \cup B_i$;
end

5.3.2 Mitigating SAL$_\tau$ recall overestimation: SAL$_\tau^m$

As it will be clear in the results section (Section 5.5), SAL$_\tau$ achieves significant improvements with respect to the compared methods. However, SAL$_\tau$ also tends, in some cases and especially for higher recall targets, to overestimate the recall, stopping the TAR process too early. Leaving the study of more complex approaches to future work, we propose a simple way to mitigate the issue of recall overestimation by raising by a margin value the target recall given in input to the method. We call this variant SAL$_\tau^m$ (SAL$_\tau$ with margin m), which trades off an increment in annotation costs for a safer TAR process that reduces the early stops. SAL$_\tau^m$ is actually SAL$_\tau$ with the only difference being the usage of a target recall R^m determined as a function of the target recall R and the margin m:

$$R^m = R + (1 - R)m \tag{5.7}$$

The margin m ranges from 0 to 1. When $m = 0$, $\text{SAL}_\tau^m = \text{SAL}_\tau$; when $m = 1$, $R^m = 1$. A low value means fully trusting SAL_τ, a high value means accepting to label more documents to avoid early stops. In order to avoid adding a free parameter to the method, we decided to set $m = R$, following the intuition that it is crucial to guarantee a given target recall R, the closer R is to 1. Equation 5.7 thus becomes:

$$R^m = R + (1 - R)R = 2R - R^2 \tag{5.8}$$

We call this configuration SAL_τ^R and comment upon its results in Section 5.5. We defer to future works the exploration of more informed methods to set m, which could possibly enable a more convenient trade-off between annotation costs and proper recall targeting.

5.4 Experiments

5.4.1 Using SAL_τ to stop a TAR process

The SAL_τ algorithm can be tested in any AL scenario where we need to improve priors and posteriors. In this chapter, we focus on testing SAL_τ capabilities for TAR: our goal is to stop the review process as soon as a target recall R is reached, lowering the review cost.

We test the SAL_τ algorithm with three configurations:

1. the SAL_τ formulation of Algorithm 4;
2. SAL_τ^R, i.e., with margin, as described in Section 5.3.2;
3. SAL_τCI, a variant of SAL_τ that uses the confidence interval heuristic from the QuantCI technique (Yang et al., 2021a).

Notice also that, following Yang et al. (2021a), we only focus on one-phase TAR baselines (SAL_τ is also a one-phase algorithm) which work inside a CAL process and, as such, do not change the sampling policy. As observed in Yang et al. (2021a, §2), we argue that these latter methods (called interventional methods) are (i) less efficient than AL (i.e., they usually trade off annotation costs for a safer recall estimation) and (ii) less applicable in real case scenarios, where reviewers are often limited to using a specific AL policy (i.e., usually CAL) provided by a specific software. Given this, we compare against the following well-known methods (see Chapter 2):

1. The Knee method by Cormack et al. (Cormack, Grossman, 2015a);
2. The Budget method by Cormack et al. (Cormack, Grossman, 2015a)
3. The CHM method by Callaghan et al. (Callaghan, Müller-Hansen, 2020);
4. The QuantCI method by Yang et al. (Yang et al., 2021a);

Following an approach similar to Yang et al. (2021a), we will not use the random sampling part of CMH in our experiments and only use its heuristic method as a stopping rule; notice, however, that the random sampling and the confidence level estimation which follows the heuristic stopping criterion is applicable to any of the other methods we explore in this chapter (including our own SAL_τ method).

5.4.2 The active learning workflow

We run the same active learning workflow for all tested methods. In most TAR applications (Cormack, Grossman, 2015a; Yang et al., 2021a), we usually have only a single positive document to start the active learning with, called the initial seed: for our experiments, we decide to seed the active learning process with an additional negative document randomly sampled from the document pool P; that is, our initial seed set S consists of a positive and a negative document, randomly sampled from P.

As mentioned earlier, the active learning policy we pick is CAL (see Section 2.3.1) since this is the most common policy used in TAR tasks. As the batch size b, we follow Yang et al. (2021a) and set it to 100. The classifier we use in all our experiments is a standard Logistic Regression, as this is also the classifier of choice in most (if not all) one-phase TAR applications. We test the target recalls values $\{0.8, 0.9, 0.95\}$.

5.4.3 Datasets

We run our experiments on two well-known datasets, which we illustrated in Section 2.6: the RCV1-v2 and the CLEF Technology-Assisted Reviews in Empirical Medicine (EMED) datasets (specifically, the dataset made available for the CLEF 2019 edition). Both datasets have been already used to test TAR frameworks and algorithms, e.g., the MINECORE framework (Oard et al., 2018), the QuantCI stopping technique (Yang et al., 2021a) and Li, Kanoulas (2020)'s autostop framework. Regarding RCV1-v2, we use (in these experiments) a random sample of 10,000 documents, both for computational reasons, and to keep RCV1 pool size close to that of CLEF. We also use a sample of the Jeb Bush Email collection to set SAL$_\tau$ α hyperparameter.

All text is converted into vector representation first converting it to lowercase, removing English stopwords and any term occurring in more than 90% of the documents in P; vectors are weighted using $tf - idf$.

Jeb Bush Email collection

The Jeb Bush's emails collection consists of 290,099 emails sent and received by the former governor of Florida Jeb Bush. We used the subset published by Grossman et al.:[6] the sample consists of 9 topics, with 50,000 documents. For each document and topic, a relevance judgment is available. We do not run experiments on this sample, but only use it to set the hyperparameter α (see Section 5.3.1).

5.5 Results

We run each of the experiments defined in the previous section 20 times, using a different randomly generated seed set S each time (the same random seed set for all methods compared); we set $\alpha = 0.3$ (Section 5.3.1), as this was the best-performing value in a hyperparameter search conducted on the Jeb Bush dataset. Thus, the results reported for a dataset are the average of the evaluation measure applied to all seed sets and all classes for that dataset. We also defined three equal-sized bins sorted by class prevalence (Very low, Low, and Medium) to show how the different methods performed with different level of imbalance between relevant and non-relevant documents. The average prevalence

[6] `https://github.com/hical/sample-dataset`

for the classes in the three bins of RCV1 is 0.002, 0.012, and 0.084, and for CLEF is 0.005, 0.027, and 0.117.

Tables 5.2 and 5.3 report the MSE and RE values, tables 5.4 and 5.5 report the IC values (see Section 2.5). The three most competitive models are SAL_τ, SAL_τ^R and the Budget method. For the MSE and RE metrics our SAL_τ and SAL_τ^R stopping rules bring about consistent and substantial improvements for lower target recalls (0.8 and 0.9). When $R = 0.95$, the Budget method performs slightly better instead.

With respect to the IC measure, SAL_τ shows the lowest costs in almost all configurations, closely followed by SAL_τ^R. The most relevant reduction of cost shown by SAL_τ and SAL_τ^R, with respect to the compared methods, is for the Very Low prevalence bin. This is an important result, as the "needle in a haystack" scenario is the most common in TAR. The Budget method has comparable costs for high target recall values and in higher prevalence bins.

As we anticipated, SAL_τ tends to stop the TAR process too early. This can be seen in the box plots in Figure 5.3, which show at which real recall values the different stopping rules decided to halt the review process. The average recall value for SAL_τ is always higher than the target recall, i.e. the expected value of recall satisfies the target recall requirement. Yet, it is evident how for higher recall targets, the distribution of recall values produced by SAL_τ goes under the target not only for the tail of the distribution, as it happens for Budget and Knee, but also for a portion of the center part of the distribution. Most TAR tasks require to match the target recall in a much larger portion of the cases. By simply adding a margin that is a function of the target recall, SAL_τ^R shifts the distribution of the reached recall values up. Its distribution is similar to the Budget method's, with slightly increasing costs with respect to SAL_τ but still lower than the other tested methods.

To sum up, SAL_τ makes SLD usable in AL and TAR processes, solving the issues observed in Chapter 4. SAL_τ brings consistent and substantial improvements, especially for medium/high (0.8, 0.9) target recalls; for higher targets, it tends to stop too early. SAL_τ^R solves this issue without a significant increase in annotation costs with respect to SAL_τ, finding a very good trade-off between annotation costs and proper target recall matching. In future works we propose to investigate more informed methods to mitigate SAL_τ target recall overestimation, as well as testing SAL_τ and SAL_τ^R with other sampling techniques (e.g. Li, Kanoulas (2020)).

5.6 Discussion

In this chapter, we introduced a new method called SAL_τ (and its variations SAL_τ^R and SAL_τCI) to improve posterior probabilities and prevalence estimates in an active learning review process; more specifically, we tested our method as a "when-to-stop" stopping rule for TAR tasks. SAL_τ is a variant of the well-known SLD algorithm: our algorithm was designed to enable the usage of SLD in AL and TAR processes, solving the issues observed in Chapter 4. Experiments have shown that SAL_τ still tends to slightly overestimate the true recall as it gets close to one, stopping the process too early. SAL_τ^R solves this issue, without significantly increasing the review cost compared to SAL_τ. In the experiments, SAL_τ^R has consistently improved over state-of-the-art methods by improving the estimation of prevalence and thus stopping the TAR process much earlier than other methods.

For future work, we propose to investigate more informed methods to mitigate the overestimation of target recall of SAL_τ, as well as to test SAL_τ and SAL_τ^R with other sampling techniques (e.g., Li, Kanoulas (2020)).

	All		Very Low		Low		Medium	
	MSE	RE	MSE	RE	MSE	RE	MSE	RE
Recall=0.8								
Budget	0.032	0.222	0.038	0.243	0.030	0.215	0.028	0.208
CHM	0.036	0.238	0.038	0.245	0.037	0.239	0.034	0.229
Knee	0.034	0.229	0.040	0.249	0.034	0.230	0.028	0.208
Quant	0.039	0.248	0.040	0.250	0.040	0.249	0.038	0.245
QuantCI	0.040	0.249	0.040	0.250	0.040	0.250	0.039	0.247
SAL_τ	0.029	**0.182**	0.048	0.233	**0.026**	**0.185**	**0.013**	**0.127**
SAL_τ^R	**0.027**	0.194	**0.031**	**0.208**	0.028	0.199	0.021	0.176
SAL_τCI	0.031	0.213	0.040	0.250	0.038	0.243	0.016	0.146
Recall=0.9								
Budget	0.007	0.087	0.009	0.105	0.006	0.080	0.005	0.076
CHM	0.009	0.107	0.010	0.111	0.010	0.109	0.008	0.102
Knee	0.008	0.094	0.010	0.111	0.008	0.094	0.005	0.076
Quant	0.010	0.111	0.010	0.111	0.010	0.111	0.010	0.110
QuantCI	0.010	0.111	0.010	0.111	0.010	0.111	0.010	0.111
SAL_τ	0.010	**0.076**	0.019	0.100	0.006	**0.073**	**0.003**	**0.053**
SAL_τ^R	**0.006**	0.078	**0.007**	**0.085**	**0.006**	0.075	0.005	0.072
SAL_τCI	0.009	0.103	0.010	0.111	0.010	0.111	0.007	0.087
Recall=0.95								
Budget	**0.001**	**0.035**	0.002	0.048	**0.001**	**0.030**	**0.001**	**0.027**
CHM	0.002	0.051	0.003	0.053	0.003	0.053	0.002	0.049
Knee	0.002	0.040	0.002	0.052	0.002	0.040	0.001	0.028
Quant	0.002	0.052	0.003	0.053	0.003	0.053	0.002	0.052
QuantCI	0.002	0.053	0.003	0.053	0.003	0.053	0.002	0.052
SAL_τ	0.006	0.044	0.012	0.057	0.003	0.041	0.002	0.034
SAL_τ^R	0.002	0.037	**0.002**	**0.040**	0.002	0.039	0.001	0.031
SAL_τCI	0.002	0.051	0.003	0.053	0.003	0.053	0.002	0.048

Table 5.2: MSE and RE results on RCV1 (**best result**, second).

	All		Very Low		Low		Medium	
	MSE	RE	MSE	RE	MSE	RE	MSE	RE
Recall=0.8								
Budget	0.035	0.230	0.035	0.229	0.031	0.219	0.038	0.242
CHM	0.038	0.245	0.039	0.248	0.038	0.242	0.038	0.244
Knee	0.039	0.245	0.040	0.249	0.037	0.241	0.039	0.245
Quant	0.039	0.247	0.040	0.249	0.039	0.248	0.038	0.243
QuantCI	0.040	0.249	0.040	0.250	0.040	0.250	0.039	0.248
SAL_τ	**0.026**	**0.173**	0.032	**0.173**	**0.017**	**0.146**	**0.029**	**0.201**
SAL_τ^R	0.028	0.192	**0.027**	0.177	0.026	0.195	0.030	0.204
SAL_τCI	0.036	0.235	0.040	0.250	0.034	0.224	0.035	0.230
Recall=0.9								
Budget	0.008	0.093	0.008	0.093	0.006	0.084	0.009	0.104
CHM	0.010	0.109	0.010	0.111	0.010	0.109	0.010	0.108
Knee	0.009	0.107	0.010	0.111	0.009	0.103	0.009	0.107
Quant	0.010	0.110	0.010	0.111	0.010	0.110	0.010	0.108
QuantCI	0.010	0.111	0.010	0.111	0.010	0.111	0.010	0.111
SAL_τ	0.014	0.093	0.024	0.110	**0.005**	**0.068**	0.013	0.101
SAL_τ^R	**0.007**	**0.088**	**0.007**	**0.081**	0.006	0.084	**0.008**	**0.099**
SAL_τCI	0.010	0.110	0.010	0.111	0.010	0.109	0.010	0.111
Recall=0.95								
Budget	**0.002**	**0.042**	**0.002**	**0.050**	**0.001**	**0.030**	**0.002**	**0.046**
CHM	0.002	0.052	0.003	0.053	0.002	0.052	0.002	0.051
Knee	0.002	0.049	0.002	0.052	0.002	0.045	0.002	0.049
Quant	0.002	0.052	0.003	0.053	0.003	0.053	0.002	0.052
QuantCI	0.003	0.053	0.003	0.053	0.003	0.053	0.003	0.053
SAL_τ	0.010	0.064	0.021	0.094	0.002	0.037	0.007	0.060
SAL_τ^R	0.003	0.047	0.004	0.059	0.001	0.034	0.002	0.048
SAL_τCI	0.003	0.053	0.003	0.053	0.003	0.053	0.003	0.053

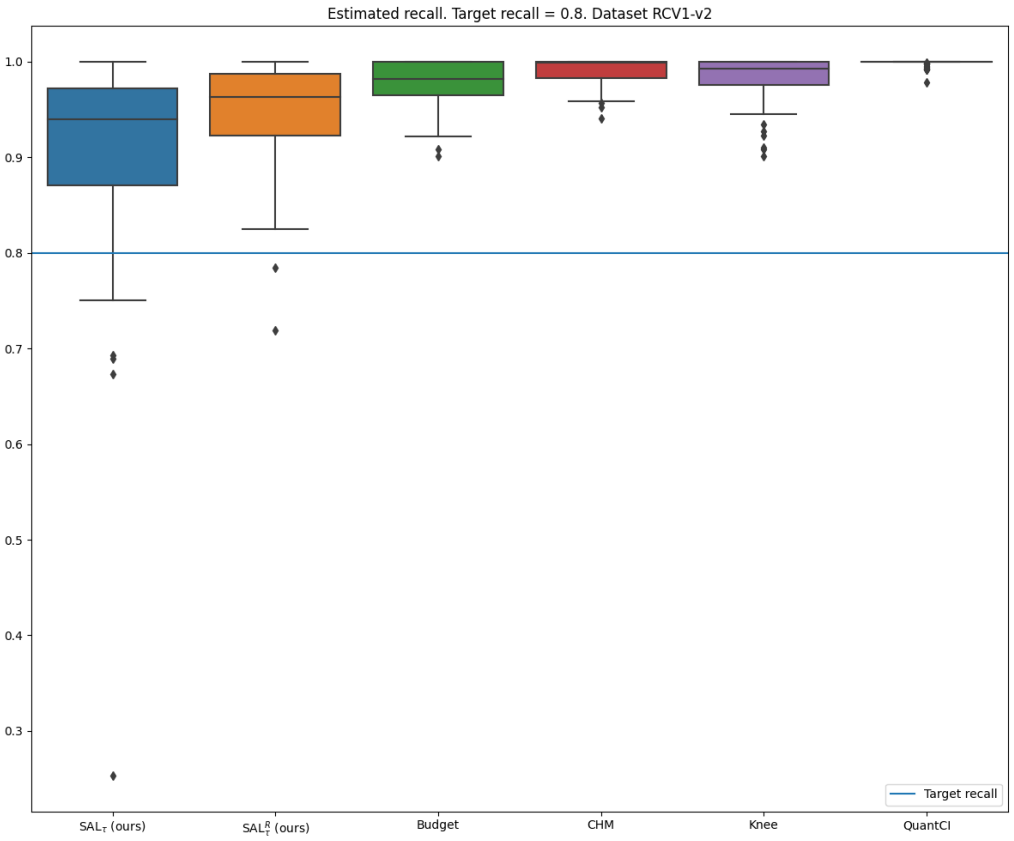
Table 5.3: MSE and RE results on CLEF.

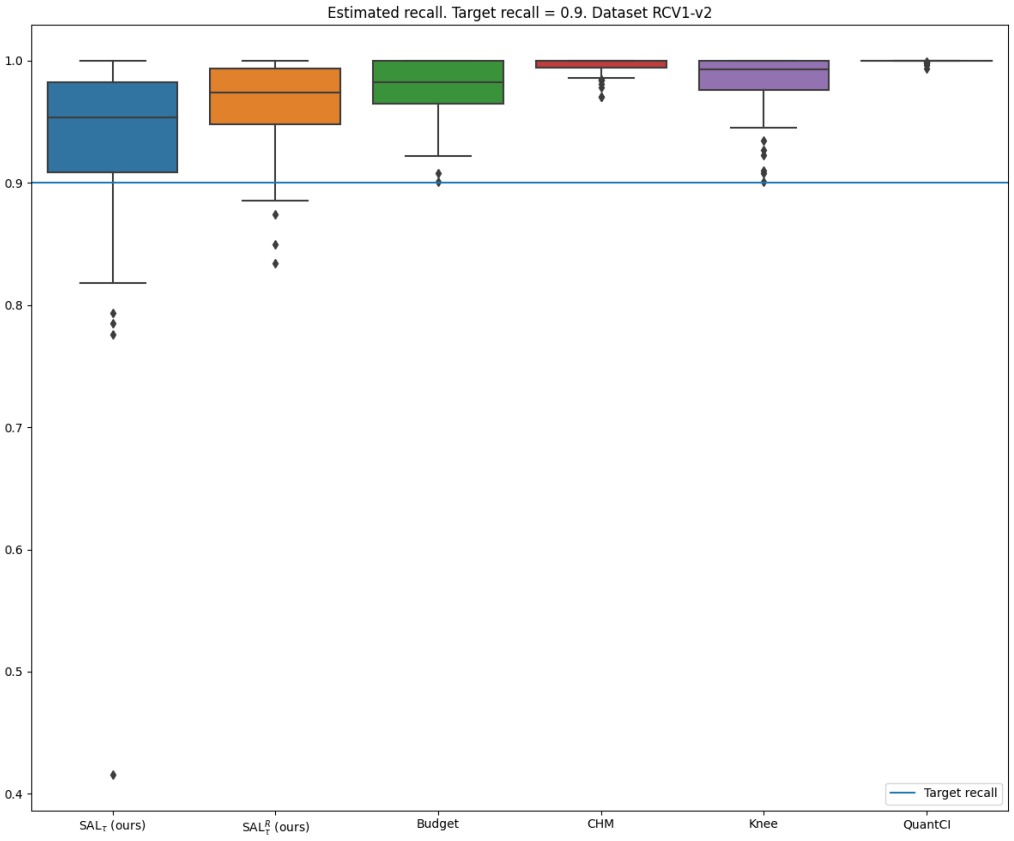
	All			Very Low			Low			Medium		
	C_u	C_e	C_m	C_u	C_e	C_m	C_u	C_e	C_m	C_u	C_e	C_m
Recall=0.8												
Budget	2855	28550	2855	4865	48653	4865	1258	12581	1258	2393	23932	2393
CHM	3738	37383	3738	5742	57424	5742	2545	25453	2545	2891	28911	2891
Knee	4303	43026	4303	7574	75741	7574	2868	28683	2868	2422	24219	2423
Quant	6355	63550	6355	7660	76596	7660	6439	64392	6439	4969	49687	4969
QuantCI	8714	87140	8714	10000	100000	10000	9813	98129	9813	6362	63623	6362
SAL_τ	**961**	**9403**	**1051**	**639**	**6091**	774	**775**	**7724**	**787**	**1462**	**14342**	**1586**
SAL_τ^R	1153	11518	1160	697	6936	**713**	939	9386	942	1817	18167	1817
SAL_τCI	6444	64436	6447	10000	100000	10000	7766	77660	7766	1607	16049	1614
Recall=0.9												
Budget	2856	28551	2859	4865	48653	4865	1258	12581	1259	2395	23934	2404
CHM	5228	52281	5228	7943	79434	7943	4049	40487	4049	3656	36564	3656
Knee	4304	43027	4308	7574	75741	7574	2869	28684	2869	2425	24221	2436
Quant	7764	77637	7764	8842	88420	8842	7952	79518	7952	6503	65029	6503
QuantCI	9682	96821	9682	10000	100000	10000	10000	100000	10000	9056	90559	9056
SAL_τ	**1135**	**10526**	**1502**	**814**	**6727**	**1441**	**895**	**8785**	**967**	**1690**	**16014**	**2083**
SAL_τ^R	1667	16223	1867	1761	16428	2285	1016	10059	1061	2205	21993	2231
SAL_τCI	8688	86861	8696	10000	100000	10000	10000	100000	10000	6103	60981	6127
Recall=0.95												
Budget	2915	28610	3154	4904	48691	5057	1343	12666	**1684**	2450	23989	**2676**
CHM	6754	67539	6754	9665	96650	9665	5950	59503	5950	4622	46222	4622
Knee	4332	43055	4448	7575	75742	7578	2897	28712	3012	2480	24276	2711
Quant	8694	86944	8694	9466	94656	9466	8895	88948	8895	7729	77290	7729
QuantCI	9927	99266	9927	10000	100000	10000	10000	100000	10000	9782	97821	9782
SAL_τ	**1445**	**11575**	**2723**	**1174**	**7316**	**3138**	**1123**	**9490**	1894	**2029**	**17856**	3111
SAL_τ^R	2259	21098	2920	2910	26272	4165	1308	11855	1855	2529	24888	2709
SAL_τCI	9712	97066	9738	10000	100000	10000	10000	100000	10000	9146	91287	9223

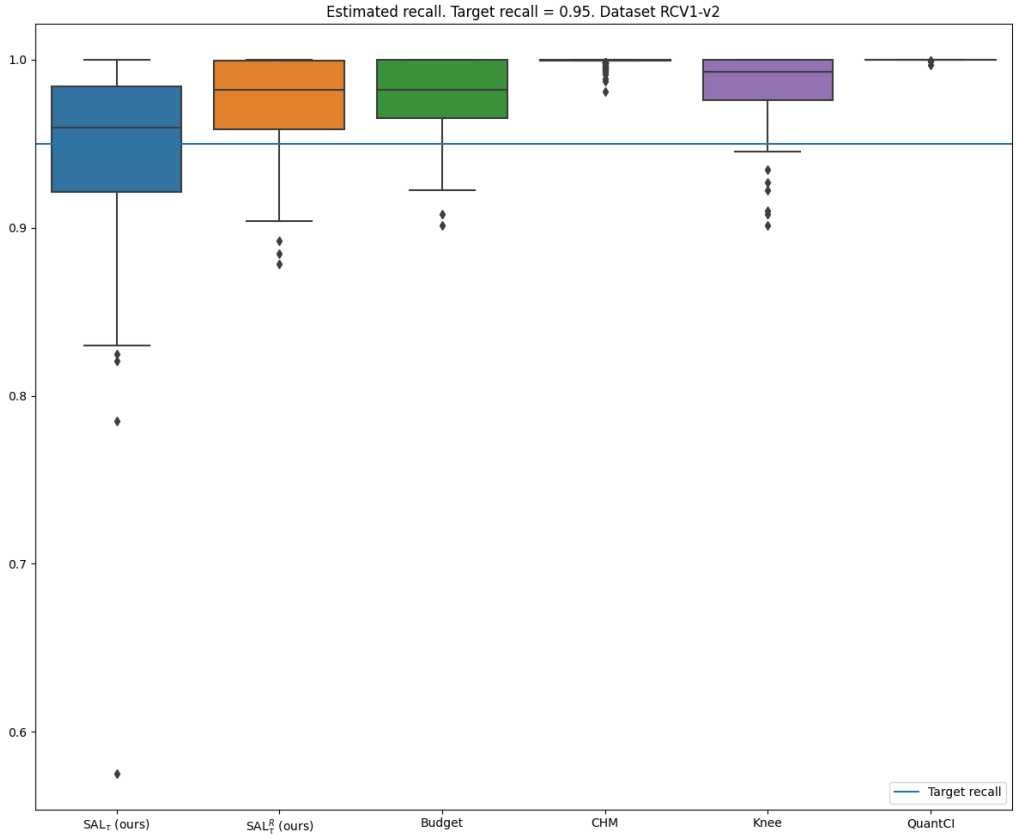
Table 5.4: IC measure on RCV1. Regarding the meaning of the different cost structures (C_u, C_e and C_m), we refer the reader to Section 2.5.2.

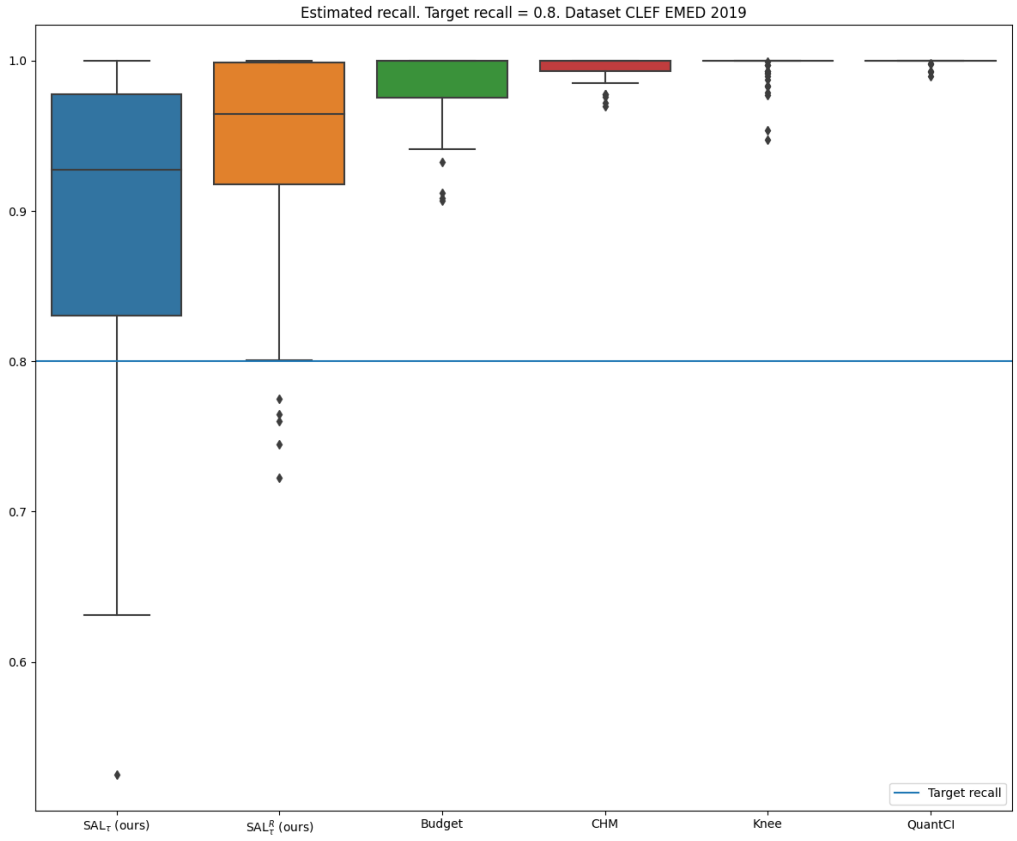
	All			Very Low			Low			Medium		
	C_u	C_e	C_m	C_u	C_e	C_m	C_u	C_e	C_m	C_u	C_e	C_m
Recall=0.8												
Budget	1656	16556	1656	2986	29860	2986	1125	11253	1125	855	8554	855
CHM	2442	24418	2442	4768	47678	4768	1733	17328	1733	825	8248	825
Knee	3173	31726	3173	6219	62189	6219	2295	22947	2295	1004	10042	1004
Quant	3329	33289	3329	6512	65120	6512	2575	25750	2575	900	8998	900
QuantCI	5145	51449	5145	10095	100954	10095	4172	41715	4172	1168	11678	1168
SAL_τ	**601**	**5666**	**753**	**685**	**6451**	**860**	**628**	**6190**	**668**	**490**	**4357**	730
SAL_τ^R	732	7222	773	789	7686	882	858	8583	858	547	5397	**580**
SAL_τCI	4657	46438	4714	10095	100954	10095	3131	31304	3136	743	7056	912
Recall=0.9												
Budget	1656	16556	1656	2986	29860	2986	1125	11253	1125	855	8554	855
CHM	3271	32706	3271	6435	64351	6435	2430	24300	2430	947	9466	947
Knee	3173	31726	3173	6219	62189	6219	2295	22947	2295	1004	10042	1004
Quant	4187	41870	4187	8053	80532	8053	3397	33970	3397	1111	11108	1111
QuantCI	5524	55238	5524	10095	100954	10095	4956	49556	4956	1520	15204	1520
SAL_τ	**777**	**6447**	1366	**996**	**7316**	**2172**	**774**	**7345**	**948**	**561**	**4679**	978
SAL_τ^R	1027	9547	**1348**	1324	11087	2282	1052	10518	1054	704	7036	**707**
SAL_τCI	5336	53355	5336	10095	100954	10095	4391	43908	4391	1520	15204	1520
Recall=0.95												
Budget	1729	16629	**2025**	3206	30080	**4085**	1127	11255	**1134**	855	8554	856
CHM	4016	40155	4016	7849	78490	7849	3106	31058	3106	1092	10918	1092
Knee	3173	31726	3174	6219	62189	6219	2295	22948	2298	1004	10042	1004
Quant	4767	47671	4767	9010	90100	9010	4022	40215	4022	1270	12698	1270
QuantCI	5537	55371	5537	10095	100954	10095	4995	49954	4995	1520	15204	1520
SAL_τ	**999**	**7235**	2225	**1435**	**8158**	4186	**925**	**8310**	1344	**638**	**5238**	1146
SAL_τ^R	1309	11302	2105	1974	14482	4313	1148	11370	1196	805	8053	**806**
SAL_τCI	5537	55371	5537	10095	100954	10095	4995	49954	4995	1520	15204	1520

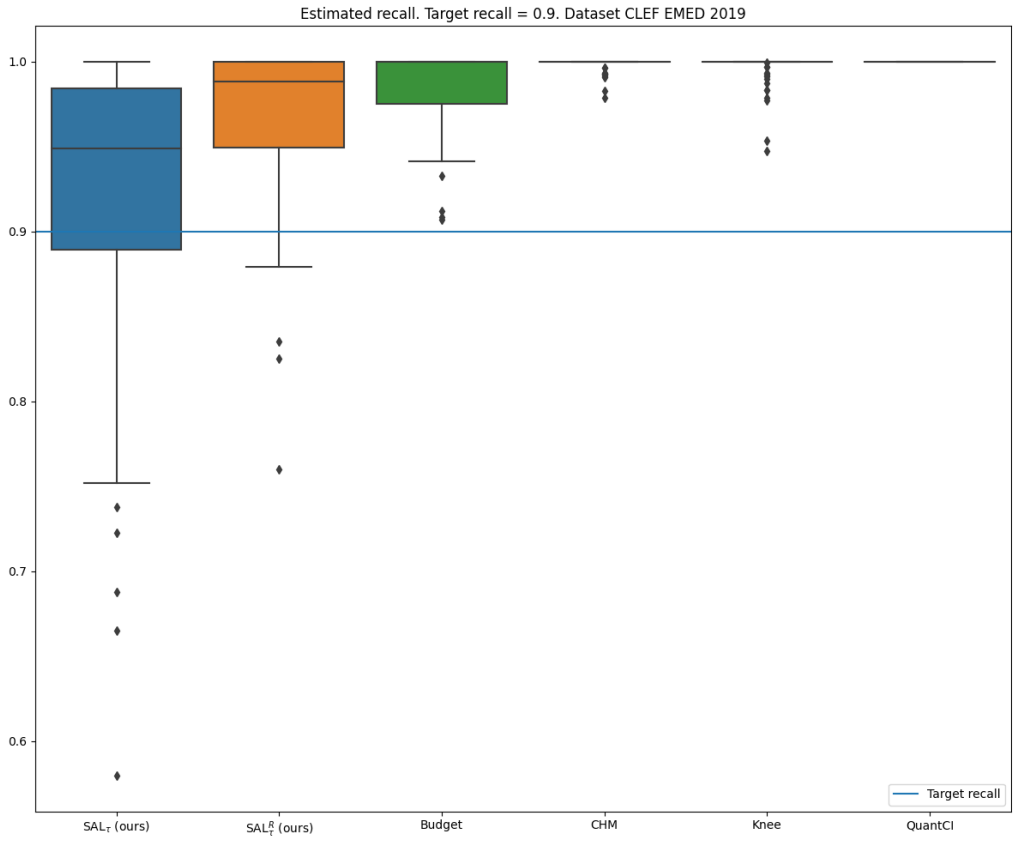
Table 5.5: IC measure on CLEF.


Estimated recall. Target recall = 0.8. Dataset RCV1-v2
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Target recall
SALτ (ours)
SALτR (ours)
Budget
CHM
Knee
QuantCI


Estimated recall. Target recall = 0.9. Dataset RCV1-v2
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Target recall
SALτ (ours)
SALτR (ours)
Budget
CHM
Knee
QuantCI


Estimated recall. Target recall = 0.95. Dataset RCV1-v2
1.0
0.9
0.8
0.7
0.6
Target recall
SAL$_T$ (ours)
SAL$_t^R$ (ours)
Budget
CHM
Knee
QuantCI


Estimated recall. Target recall = 0.8. Dataset CLEF EMED 2019
1.0
0.9
0.8
0.7
0.6
Target recall
SAL$_T$ (ours)
SAL$_t^R$ (ours)
Budget
CHM
Knee
QuantCI



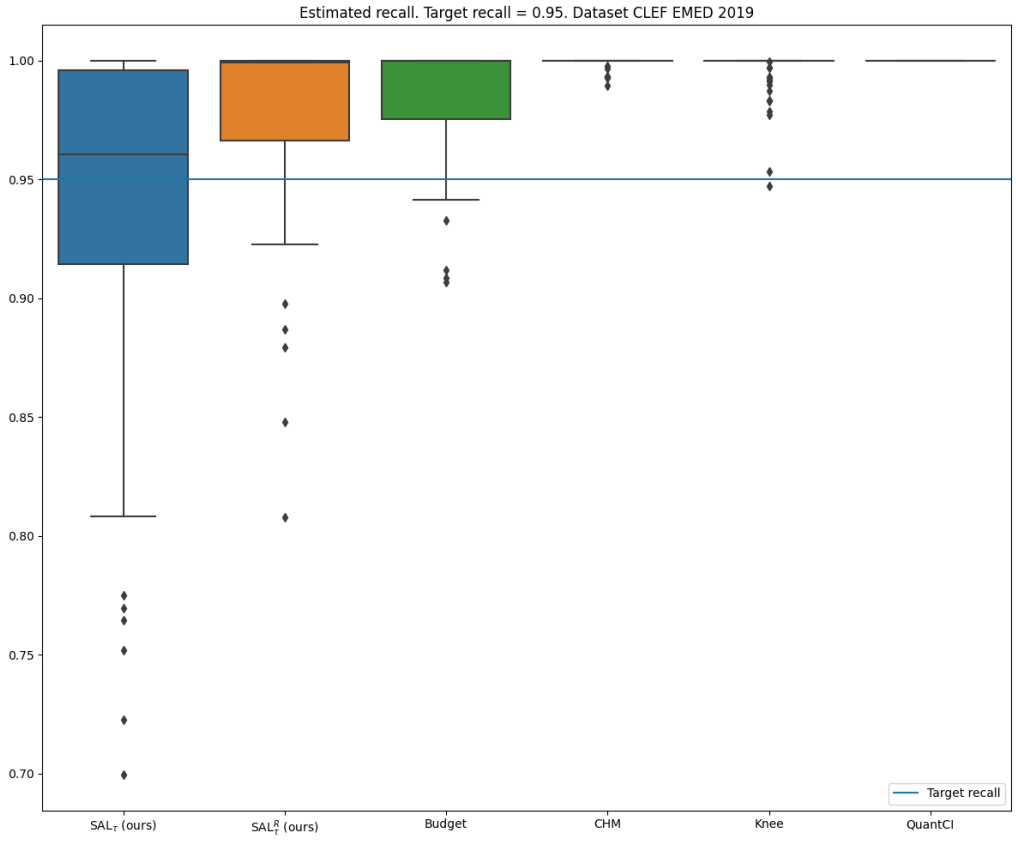


Figure 5.3: Box plots of actual recall reached by the methods, given a target recall: 0.8 (left), 0.90 (center), 0.95 (right). First three plots are measured on RCV1, last three on CLEF.

This chapter also concludes our work on the SLD algorithm; in the next and last chapter before conclusions, we will focus exclusively on systematic reviews, and specifically on the possibility of transferring knowledge between systematic review topics, studying whether it is possible to leverage zero-shot rankings to enhance the document retrieval process.

Chapter 6

Transfer learning for model portability in TAR

In this last chapter of our thesis, we focus exclusively on systematic reviews. One of the substantial differences between e-discovery and the production of systematic reviews is that, for the latter, we can find real-world datasets, i.e., previously conducted systematic reviews. Can we leverage these previous review efforts to pre-train a (deep learning) model? Would this bring to a good zero-shot (i.e., without training the model with in-domain data) initial ranking? The recent literature has shown how big deep learning architectures (such as BERT or GPT) are also good zero-shot learners (Wei et al., 2021; Zhong et al., 2021). Does this hold in systematic reviews, where we have very small training sets, and "needle-in-a-haystack" scenarios? And, if it does, can we leverage a zero-shot ranking to bootstrap the CAL process? Finally, can we continuously train deep learning models in the active learning process? We try to answer these questions in this chapter, showing that deep learning models such as BioBERT are indeed capable of transferring knowledge between SR topics, albeit failing to deliver consistent results when continuously trained in an active learning scenario. Nonetheless, in our concluding analysis we show that there might be several promising aspects where future research might focus in order to enable the active training of deep learning models. The work presented in this chapter was published in Molinari, Kanoulas (2022).

6.1 Introduction

In this chapter we explore the possibility of using previous systematic reviews (SR) in order to pre-train a model, which is later used for a zero/few shot classification (or ranking) of a new, unseen, review topic. More specifically, we investigate two research questions:

RQ1. Can we transfer the knowledge acquired on previous systematic reviews, and if so, to which extent?

RQ2. Can we keep training our pre-trained models in the active learning process?

We will test a classical logistic regression and two deep learning architectures: one is the well-known BioBERT (Lee et al., 2020) model (i.e., BERT trained on biomedical data), and the other is a transformer-based architecture which computes attention between documents instead of tokens

(Pobrotyn et al. (2020), see also Section 6.3). All of our experiments in this chapter are conducted on the CLEF 2019 dataset (see Section 2.6.2).

6.2 Related work

Regarding the usage of pre-existing systematic reviews to pretrain a model which is later applied to a new systematic review topic, an interesting work (as well as the first one attempting such an experiment, to the best of our knowledge) is Cohen et al. (2009). In this study, the authors train an SVM algorithm on 23 previously conducted SRs and later apply it to one unseen topic, using a Leave-One-Out setup; the experimental results show how such an approach is able to consistently deliver better results than the baseline. That said, they assume the availability of topic-specific training data, which we do not in this chapter (shifting the context to a real zero-shot scenario).

A more recent work where a transfer learning approach is leveraged is Lagopoulos, Tsoumakas (2020). Here, the pre-training of a model on previous SRs is part of a larger framework, which deals with all the stages of a systematic review. The experiments conducted are focused on the whole pipeline rather than the effectiveness of the training procedure. Nonetheless, their results show that pre-training can indeed be useful and that knowledge can be transferred between different topics. Pickens (2021) conducts several experiments to measure whether "portable" models can be trusted (and whether they are effective) in TAR applications; however, their experimentation mostly focuses on in-topic training (i.e., the training set (source) is coming from the same distribution of the test (target) set).

Most of these approaches experiment with machine learning algorithms such as SVM or logistic regression: Deep Learning (DL) models are usually not considered fit for such a task, as they depend on too many parameters (and data is usually scarce) and are easily outperformed by classic ML algorithms (Yang et al., 2021c). That said, there has been some work on testing DL models for TAR applications: Yang et al. (2021c) tried to use a just-right fine-tuned BERT (Devlin et al., 2019) in the active learning process for e-discovery. BERT is first fine-tuned with the masked language task on the available documents, and later continuously trained for a fixed number of epochs during the active learning review process. Their results show how in some cases BERT can actually achieve slightly better results than linear models, but postpone further experimentation to future works. Similarly, Zhao et al. (2021) explored whether several models (BERT, logistic regression) trained (or fine-tuned) on a given training set can perform well on new data not seen during training: they showed how pre-training can usually bring to good performances, but the results are not consistent across all tested datasets (i.e., the pre-trained models can completely fail to transfer knowledge in some cases).

The main difference between this chapter and most works we found in the literature is that, while the different learning architectures that we test (see Section 6.3.1) are first trained/fine-tuned on a set of previous systematic reviews topics for which we have the reviewers' final decisions, they are later tested on a previously unseen SR topic. We assume not to have any kind of training data for this latter test topic. The reviewer's opinion is elicited via an active learning approach: in particular, we employ the CAL algorithm as the main and only active learning approach to emulate data annotation in our experiments. As we will see in more details in Section 6.5, the CAL algorithm will be both used as a baseline (using a classical logistic regression as its classifier) and as the active learning methodology to continuously train our transfer-learning models. Regarding its implementation, we follow the configuration described in Cormack, Grossman (2015a) (unless

differently stated), where we use an exponentially increasing batch size b, starting from $b = 1$, and using the title of the topic as the seed (and only known) positive document.

6.3 Methodology and experimental design

6.3.1 Learning algorithms

With the goal of understanding whether we can transfer information between different systematic review topics, we explore and employ different learning architectures:

- a classical **Logistic Regression (LR)** classifier, which is one of the well-established standard learning models used in TAR for systematic reviews literature;
- the **BioBERT** architecture (Lee et al., 2020), a BERT (Devlin et al., 2019) based model specifically trained on scientific and medical data which could thus be expected to achieve good performances on our task. We fine-tune it with a pairwise loss;
- a deep learning model based on the transformer architecture (Vaswani et al., 2017) where the self-attention mechanism is actually computed between documents and not between tokens. This model was first proposed in Pobrotyn et al. (2020); we refer the reader to the original work for a more thorough and in-depth explanation of this model. We test the model by maximizing the NDCG metric, more specifically we implement the deterministic NeuralNDCG (Pobrotyn, Białobrzeski, 2021). We call this model the **DL Ranker or Ranker**;
- we also test the same architecture with a cross-entropy loss; we call this model the **DL Classifier or Classifier**;

The reasons behind our choice to test another deep learning architecture other than BioBERT are strongly tied to the learning setting we are confronted with in systematic reviews: firstly, despite having a decently sized training set when we merge together past systematic reviews, the testing and fine-tuning of our models is done on single topics, whose sizes may be very small (i.e., few hundreds of documents, with a very low positive prevalence); we believe that having less parameters to learn might ease the learning/fine-tuning of models when continuously trained in an active learning process. The second reason that motivates us in using the ranker model of Pobrotyn et al. (2020) is that we would also like to compare the more traditional classification and/or pairwise approaches (i.e., the logistic regression and BioBERT) to a list-wise capable architecture.

6.3.2 Data preprocessing

As explained in Section 6.3.1 we experiment with different architectures, which require different data representation and organization:

- The Logistic Regression, the DL Ranker and the DL Classifier are trained with the same data representation. We see this in detail in Section 6.3.2;
- We train the BioBERT model with a pairwise loss, where the aim of the model is that of correctly prioritizing one of the documents in the pair. For this reason we merge together pairs of documents as explained in Section 6.3.2;

- Since we compare with the CAL baseline (as implemented in Cormack, Grossman (2015a)), we also need a representation suited to its logistic regression; we simply default to a standard TF-IDF representation.

We will now look more in details to the different techniques we use to preprocess our data before feeding it to the models.

Document embeddings and list-wise structure

For the LR, the DL Ranker and DL Classifier models, we preprocess data by tokenizing each document and transforming it into a fixed-dimensional vector of features. The title of the topic (i.e., the research question of the SR) is prepended to the text of each document. More precisely, for each document we have a feature vector v of size $E = 768$. This vector v is the average of the non-finetuned BioBERT embeddings for the tokens of a document $d \in D$.

The architecture used for the DL Ranker and Classifier requires a matrix $s \times E$ of document embeddings as input (where s stands for sequence). Ideally, s should be equal to the number of documents we need to rank (i.e., $s = |L|$ or $s = |U|$) but the computational costs would be unsustainable when we have thousands of documents to rank. We have thus to pick a value of $s < |\{L, U\}|$. Due to hardware constraints, we choose $s = 1000$ in our experiments. Moreover, we also fix the batch size $b = 512$ such that the DL Ranker and Classifier are finally fed a $b \times s \times E$ matrix as input, and return a $b \times s \times o$ matrix as output, where o is the number of neurons in the output layer (this is always 2 for the Classifier and we set it at 100 for the Ranker, see also Pobrotyn et al. (2020); Pobrotyn, Białobrzeski (2021) for the original implementation). For the DL Ranker, we then take the average on the third dimension, such that we have a matrix $b \times s$ of scores for each document. For the DL Classifier, we also "augment" the training set L with a pre-defined number of documents randomly sampled from other topics (we sample a number equal to 20% of $|L|$ size): this should help the classifier to generalize better over the different topics; clearly, this cannot be done with the DL Ranker, as it would not make sense to rank documents for a topic q_i higher than a topic q_j or viceversa.

Finally, each sequence s is built by randomly sampling s examples from the dataset without replacement. Notice that since we only elaborate a sequence s of documents for every batch, when applying the model to the test topics we first classify/rank all batches and then aggregate the scores together to obtain the ranking/classification output for each $d \in U$.

Pairwise document representation

We use this pairwise representation for the BioBERT fine-tuning. For n times, we randomly pick a relevant and an irrelevant example and we combine them together: a "new" document is then formed, where we have the title of the SR topic separated by a [SEP] token from the text of the first document, which is in turn separated by another [SEP] token from the text of the second document. Whether the relevant document is first or last is decided by a fair coin flip. Since BioBERT needs many data points to be fine-tuned, for any $L \in Q$, we set $n = 1.2 \cdot |L|$, creating a training set which is 20% larger than the original size of L.

TF-IDF

The classical TF-IDF representation is only used for the logistic regression trained in the CAL algorithm used as a baseline. This representation is built using the TFIDFVECTORIZER class exposed

by SCIKIT-LEARN[1] API.

6.3.3 Rankings and evaluation measures

As explained in Section 6.2, we use the CAL algorithm as a baseline in our experiments. We use a standard non-pretrained logistic regression (which we call **NP Logistic**) as the CAL's learner. Notice that, clearly, comparing CAL's performances with the zero-shot rankings is not a fair comparison since the NP Logistic has access to and is continuously trained on in-topic data.

Furthermore, throughout our experiments, we will report metrics and results on two different types of ranking:

- A ranking which is the output of a model (be it a set of probabilities or a vector of scores) on the set of all documents to be ranked. We call this **full reranking**;
- The ordering of the documents resulting from the CAL process, that we call the **CAL ordering**. More precisely: at every iteration of CAL, we take the top k documents and have a reviewer annotating them. What we call CAL ordering is then the order in which the reviewer annotates the documents throughout the process.

Evaluation metrics

In order to evaluate the models rankings or the CAL ordering, we use two of the most well-known metrics in TAR literature: Mean Average Precision (MAP) and Work Saved Over Sampling (WSS, see Section 2.5). Average Precision (AP) is computed as:

$$\text{AP} = \frac{1}{rel} \sum_{j} \text{Precision}(j), \tag{6.1}$$

where rel is the number of relevant documents and Precision(j) is the precision at the jth item. We take the average of this metric over all the testing topics, calling it MAP.

6.4 Implementation details

The aim of our experimentation is to answer research questions RQ1 and RQ2 (see Section 6.1). Unless otherwise stated, we train our DL Ranker and DL Classifier models for 500 epochs, using the Adam optimizer (Kingma, Ba, 2015). BioBERT[2] is instead fine-tuned with a classification head with two output neurons for a maximum of 10 epochs. However, we employ a typical early stopping strategy with patience on the loss set to 10 update steps, which in practice usually stops the training set before reaching 10 epochs. The DL models (except for BioBERT) are implemented using the PyTorch Python library,[3] whereas for the Logistic Regression we use the standard scikit-learn library[4] implementation (code will be made available in the near future).

As anticipated, for the DL Ranker and Classifier we experiment with two different losses: we use (i) a Cross-Entropy loss for the DL Classifier model; given the extremely unbalanced datasets

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

[2] The pre-trained model is downloaded from the HuggingFace Hub, available at `https://huggingface.co/models`

[3] `https://pytorch.org/`

[4] `https://scikit-learn.org/stable/`

(see Section 2.6), we also use fixed class weights of 0.2 and 0.8 for the negative and positive class respectively. Furthermore, for all of our experiments with the DL models, we use only one transformer encoder layer; we leave experimentation with different number of encoders and attention heads to future works. (ii) For the ranking loss function, we directly maximize an approximation of the NDCG metric, more precisely the deterministic Neural-NDCG (Pobrotyn, Białobrzeski, 2021). While we refer the reader to Pobrotyn, Białobrzeski (2021) for implementation details, we notice that in order to approximate the permutation matrix that would sort the input, we have a temperature parameter τ which we can use to control the degree of approximation (when $\tau = 0$ we get the exact permutation matrix): we found this parameter to be highly susceptible to the size of the data fed to the model, but we postpone any precise analysis on this to future works.

For the fine-tuning of the BioBERT model, we use a pairwise loss function usually known as the Margin (or Margin Ranking) loss. That is, given two scores x_i and x_j, and a label $y = 1$ when x_i should be higher than x_j and $y = -1$ viceversa, the loss is computed as:

$$L(x_i, x_j, y) = \max(0, -y \cdot (x_i - x_j) + m) \tag{6.2}$$

where m is the margin, which we actually set to 0, following the PyTorch implementation default[5] as of version 1.10.1.

All of our experiments are run on a maximum of two NVIDIA Tesla T4 GPU (with 16GB of RAM each) on a multi-processor machine (kindly made available by the Computer Science department of the University of Pisa). Training times are not particularly demanding, as the BioBERT fine-tuning took around 36 hours, whereas 20 minutes were enough for the DL models and as little as 28 seconds for the logistic regression.

6.5 Results

We show in this section the experiment results both for our first and second research questions (RQ1 and RQ2, see Section 6.1). We also present the results of a hyperparameter search for the DL models (Section 6.5.3): this last section should serve as a basis for further research, in order to understand how and where future works might focus to successfully continuously train DL models in active learning scenarios.

6.5.1 RQ1: Can we transfer knowledge?

In order to answer this first question, we train our models on our training topics and apply them to the testing topics without any further training, to see whether we can actually obtain a good zero-shot ranking of the documents.

Evaluating the zero-shot ranking As stated in Section 6.3.3, we compare our results (full reranking) with the document ordering coming from Cormack's CAL classical implementation (what we called the CAL ordering). We show the Mean Average Precision (MAP) of the zero-shot rankings in Table 6.1; we also show the MAP for the CAL ordering and the CAL's NP Logistic ranking after 10 documents have been annotated (which we might call a few-shots NP Logistic). Notice how all the zero-shot models achieve rather good performances, obtaining a better MAP than the few-shots NP Logistic. BioBERT seems to be the best model, closely followed by the Logistic Regression

[5] `https://pytorch.org/docs/stable/generated/torch.nn.MarginRankingLoss.html`

	CAL ordering	NP Logistic (no pre-training)	Zero-shot rankings			
			DL Rank.	DL Class.	BioBERT	LR
MAP	**0.240** ±0.178	0.126 ± 0.132	0.211 ± 0.168	0.185 ± 0.163	<u>0.226</u> ±0.162	0.219 ± 0.166

Table 6.1: MAP for the zero-shot rankings. We show the CAL ordering and CAL's NP Logistic full reranking after 10 documents have been annotated for comparison. Best result overall is in **bold**, whereas the best result among the zero-shot rankings is <u>underlined</u>.

	NP Logistic	DL Rank.	DL Class.	BioBERT	LR
R@10	0.012 ± 0.017	0.028 ± 0.022	0.029 ± 0.040	**0.069** ±0.055	0.046 ± 0.032

Table 6.2: Recall@10 (R@10) for the different pre-trained models and the NP Logistic baseline after annotating 10 documents. Notice the recall is measured on what we called the full reranking and not on the CAL ordering. All the pre-trained zero-shot models obtain a higher Recall@10.

(LR). As expected, the CAL baseline is able to achieve a stronger MAP than the zero-shot rankings' (since its logistic regression is being trained on in-topic data). Nonetheless, this shows that the pre-trained models are able to successfully transfer knowledge between topics.

Jump-starting the CAL algorithm To test if our zero-shot rankings are beneficial to the reviewing process (i.e., if we can achieve a higher recall earlier), we propose to jump-start the CAL algorithm from the top-10 documents coming from our zero-shot rankings. With "jump-starting" CAL we mean:

1. we pre-train a model on the training topics;
2. we rank the current new (and unseen) topic and take the model top-10 documents;
3. we obtain the labels for these 10 documents;
4. we train CAL's logistic regression on these 10 documents (using TF-IDF features) and start the CAL algorithm from there, following Cormack, Grossman (2015a) thereafter.

Notice that, for some topics, the pre-trained models failed to retrieve any positive instance in the top-10 documents: the DL Classifier failed on 3 topics out of 7, of which the DL Ranker failed on 2 and BioBERT and the LR failed on 1; hence, we show results averaged on 4 topics out of 7. We first show the Recall@10 (on the full reranking) in Table 6.2: notice how the zero-shot rankings effectively jump-start the CAL process from a higher recall; BioBERT proves to be the most effective algorithm to jump-start with. We show the WSS@{85, 95, 100}% and the MAP averaged across the topics in Table 6.3. Notice that the "ranking" here is actually the ordering of the documents collected at the end of the CAL process (CAL ordering). The results show how the higher initial recall translates to better performances on the average WSS scores with respect to the NP Logistic baseline, even though they are not consistently in line with the metrics taken on the zero-shot setup and the Recall@10: i.e., the top-10 documents coming from the DL Classifier or the pre-trained LR seem to be able to better jump-start the CAL process, despite BioBERT was the

	CAL ordering				
	NP Logistic	DL Rank.	DL Class.	BioBERT	LR
WSS@85%	0.494 ± 0.241	0.501 ± 0.245	0.499 ± 0.243	0.501 ± 0.243	**0.508** ± 0.254
WSS@95%	0.475 ± 0.304	0.466 ± 0.205	**0.480** ± 0.159	0.457 ± 0.328	0.460 ± 0.334
WSS@100%	0.372 ± 0.306	0.371 ± 0.297	0.373 ± 0.304	0.369 ± 0.307	**0.378** ± 0.304
MAP	0.357 ± 0.134	0.378 ± 0.151	0.387 ± 0.136	**0.468** ± 0.124	0.399 ± 0.133

Table 6.3: WSS@{85, 95, 100}% and MAP for the jump-started CAL. The NP Logistic is the classical CAL implementation, starting from a seed document. The other columns indicate from which ranking we take the top-10 documents that jump-start the CAL algorithm. Average is on 4 out of 7 topics.

best model in terms of MAP (Table 6.1) and Recall@10 (Table 6.2). Furthermore, the DL Classifier was the worst of the three pre-trained models in both Tables 6.1 and 6.2, but the CAL process jumpstarted from its top-10 documents shows better WSS performances at the 95% thresholds than the other models. Regarding the MAP, the pre-trained models can effectively jump-start the CAL algorithm as seen for the WSS; notice that the top-10 documents from BioBERT manage to keep the advantage we saw in Table 6.1 for the MAP metric. From these results, overall, we conclude that the pre-training can actually improve on the baseline performances both in terms of MAP and WSS; however, as reported by Zhao et al. (2021) as well, knowledge transfer can fail in some cases.

6.5.2 RQ2: Can we keep training our DL models in the active learning process?

In our experiments so far, we have showed results on the zero-shot rankings from our models, or when using them to jump-start the CAL process. We did not, however, leverage the pre-trained DL models in the active learning process: can these models actually be trained in such a scenario? To understand this, we run another set of experiments with the same setup as before, but where we actually keep training our DL models during the active learning review process. Training a DL model in such a scenario is not a trivial task, since many hyperparameters have to be taken into account: epochs, cross-entropy class weights (to counteract class imbalance) and learning rate are just some of the hyperparameters we deal with. Regarding epochs, Yang et al. (2021c) fine-tune BERT in the AL process for 10 and 30 epochs (based on the dataset), albeit with no clear rationale behind the choice of the number of epochs; however, they also point out how crucial it is to have "just-right" tuning of the model.

Lacking a validation set, however, we run a first batch of experiments where we arbitrarily set these hyperparameters. Due to the high computational costs of fine-tuning BioBERT at every iteration, we decided against using it in this part of the experiments for RQ2; moreover, we argue that these very large language models are impractical to fine-tune in such a scenario, both due to their computational costs and to the disproportion between the high number of parameters to fine-tune and the size of training data. Regarding the DL Ranker and Classifier:

- we train the models for 50 epochs at each CAL iteration;
- we keep the class weights in the Cross-entropy loss at 0.2 and 0.8 for the negative and positive class respectively;

	CAL ordering			
	NP Logistic	DL Rank.	DL Class.	LR
WSS@85%	0.494 ± 0.241	0.504 ± 0.242	0.468 ± 0.234	**0.508** ± 0.254
WSS@95%	**0.475** ± 0.304	0.460 ± 0.321	**0.475** ± 0.204	0.460 ± 0.334
WSS@100%	0.372 ± 0.306	0.249 ± 0.207	0.297 ± 0.131	**0.378** ± 0.304
MAP	0.357 ± 0.134	0.351 ± 0.120	0.368 ± 0.138	**0.399** ± 0.133

Table 6.4: WSS@{85, 95, 100}% and MAP for the CAL orderings where we keep training the DL models inside the CAL process. Notice that the LR is not continuously trained and results are the same as reported in Table 6.3. Average is still on 4 topics out of 7.

- we use a learning rate of 0.001.

We show the results of such experiments in terms of WSS and MAP on the CAL ordering (Table 6.4). As we can clearly see from the table, continuously training these models during the CAL process has inconsistent effects on the metrics: with respect to the jump-started CAL results (Table 6.3), the DL Ranker only improves for the WSS@85% metric, showing slight to substantial decrease in performances for all other metrics. The DL Classifier is no different and exhibits a consistent loss of performances for all metrics. In summary, fine-tuning these models in an active learning process seems unadvisable: we think this might be due to (i) the small number of documents we usually have for fine-tuning (especially in the first CAL iterations), (ii) the training set size constantly changing (possibly too slowly), (iii) a number of parameters to update which is too large with respect to the training data, (iv) many hyperparameters which might need better adjustment in such a scenario.

We believe that a much better solution in this case might be to employ Adapter modules (Houlsby et al., 2019), freezing the rest of the network. This also allows us, in terms of computational costs, to fine-tune BioBERT.

Notice Given the amount of hyperparameters involved, the absence of a validation set, and the small set of training topics, we have decided to show the impact of different hyperparameters directly on the testing topics, when using (and not using) adapter layers. These results should serve as a basis for further research on the matter and as a mean to better understand whether it is possible at all to properly train such big models (especially in BioBERT case) in a CAL setting, where the overall number of documents span from a few hundreds to a few thousands.

6.5.3 Hyperparameter search

As mentioned, we conduct a hyperparameter search study where we analyze the variation in Mean Average Precision due to the learning rate, the number of epochs, and the percentage of documents assessed at every CAL iteration. We conduct this hyperparameter search directly on the testing topics: these experiments should be taken as an effort to understand why the DL models failed when continuously trained (see Section 6.5.2) and, possibly, where to look for a solution in future works; in other words, the aim of these experiments is not to compete with a baseline (which would not be fair, since we are testing hyperparameters directly on the test set), but rather to show the most promising directions to take in order to enable DL models to be actively trained. For this reason,

we sometimes omit results when they are not particularly good or interesting (i.e., not exhibiting a pattern that we might exploit in the future), as to avoid cluttering the chapter with too many figures.

That said, we evaluate the effect of these hyperparameters both when training the whole neural network and when using adapter layers. For the former case, we show results for the DL Ranker only, as it was the best DL zero-shot model (not considering BioBERT). For the latter case, we also show BioBERT results where we vary the learning rate. We test with different configurations:

- the learning rate values range from the default value used in training of 1×10^{-3}, to 1×10^{-5}. Being BioBERT a completely different model, we test here with the default learning rate[6] of 5×10^{-5} and the value suggested by the AdapterHub library[7] of 1×10^{-4}. Epochs are fixed at 60 for the Ranker and at 5 for BioBERT;

- for the DL Ranker only, we also test the model by training for 10, 30, 60 and an adaptive number of epochs (see below) at every CAL iteration. Learning rate is fixed at 1×10^{-4};

- finally, we also train the DL Ranker annotating 5% and 20% of the documents at every CAL iteration. We indicate the percentage of documents we take at every iteration with Δ_d. BioBERT is fine-tuned with $\Delta_d = 5\%$ only.

By "adaptive number of epochs" we mean that the number of epochs change at every active learning iteration, as a function of the number of training documents we have collected so far. For these experiments, we have empirically defined this as:

$$\text{epochs} = \min(|L_i| \cdot 0.3, 500) \tag{6.3}$$

where with $|L_i|$ we indicate the size of the available training documents at a given iteration i. That is, the number of epochs is equal to 30% of the training documents, with an upper bound set at 500 (the number of epochs used during the pre-training of the models). We will now comment upon the learning rates experiments first, and the epochs experiments afterwards.

For the DL Ranker without adapter layers, we show the MAP for the different learning rate setups; we also show the NP Logistic as a baseline. The average is on all testing topics (as opposed to results in Section 6.5.1 and 6.5.2). More precisely, we continuously train and evaluate the models with the following procedure:

1. At iteration $i = 0$ (i.e., no document has been reviewed yet), all documents are ranked according to the pre-trained model zero-shot ranking;

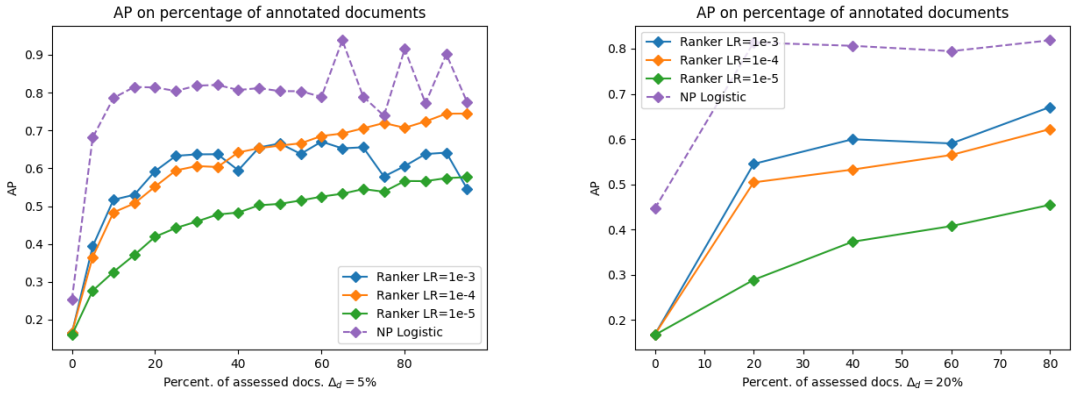
2. We compute the AP of this ranking;

3. We review the top Δ_d documents and re-train the model;

4. We re-rank the whole pool of documents again and re-compute AP;

5. We repeat this process until all documents have been reviewed.

[6] This was one of the learning rates used in Devlin et al. (2019), as well as the default in the HuggingFace library.
[7] `https://docs.adapterhub.ml/training.html`



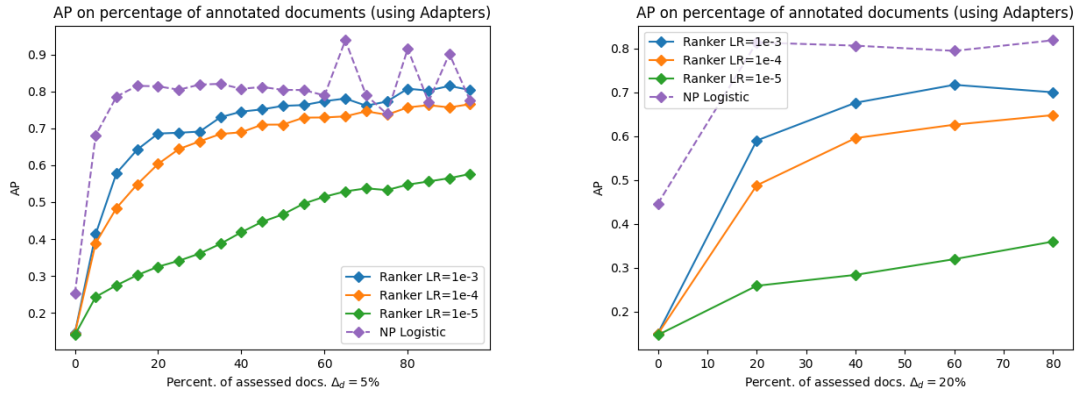
(a) (b)

Figure 6.1: Variation of Mean Average Precision with different learning rates, annotating 5% (left) and 20% (right) of the documents at each iteration.

Clearly, the NP logistic is an exception, using the systematic review topic query as the initial seed document and following Cormack, Grossman (2015a) procedure (as it did so far in our experiments, unless otherwise stated). In other words, at each iteration we take the full reranking (not the CAL ordering, see Section 6.3.3) and evaluate it. This is useful to understand whether the models under examination can indeed learn and improve on the previous iteration. Since we cannot exactly take 5% or 20% of the documents for all topics, we bin the results by number of annotated documents and plot the average of the bins. The results are plotted in Figure 6.1. When $\Delta_d = 5\%$, we notice how the default learning rate of 1×10^{-3} causes instability for the Mean Average Precision as the training set grows. The other two learning rates seem to be much more stable across CAL iterations, and a learning rate of 1×10^{-4} is capable of achieving MAP values close to the baseline's at later stages of the reviewing process. That said, the NP Logistic baseline is clearly the better algorithm here, achieving and keeping a higher MAP across all iterations. Moreover, Figure 6.1b shows that reviewing 20% of the documents at every iteration is suboptimal, leading to much lower values of MAP across all iterations.

Regarding adapter layers, we show the results in Figure 6.2 for the DL Ranker. The plots show how, when using adapters and $\Delta_d = 5\%$, a higher learning rate is able to achieve better performances. As a matter of fact, a learning rate of 1×10^{-3} obtains greater values of MAP than the baseline, at later stages of the CAL process. Overall, adapter layers seem to bring greater stability to the learning capability of the model (which is expected, having less parameters to learn). Finally, we notice once again how using $\Delta_d = 20\%$ brings to overall worse performances than with $\Delta_d = 5\%$.

We will now move on to analyze the effect of the number of epochs on the performances of the DL Ranker. As mentioned before, we test with 10, 30, 60 and an empirically defined adaptive strategy (see Equation 6.3), that we call "Adaptive" in the plots. Since results for the DL Ranker without adapters were, similarly to the learning rate ones, not particularly interesting, we show the variation of MAP when using adapters only (Figure 6.3). As we have seen for the learning rate figures, using adapter layers can indeed bring to a substantial improvement on the average



(a) (b)

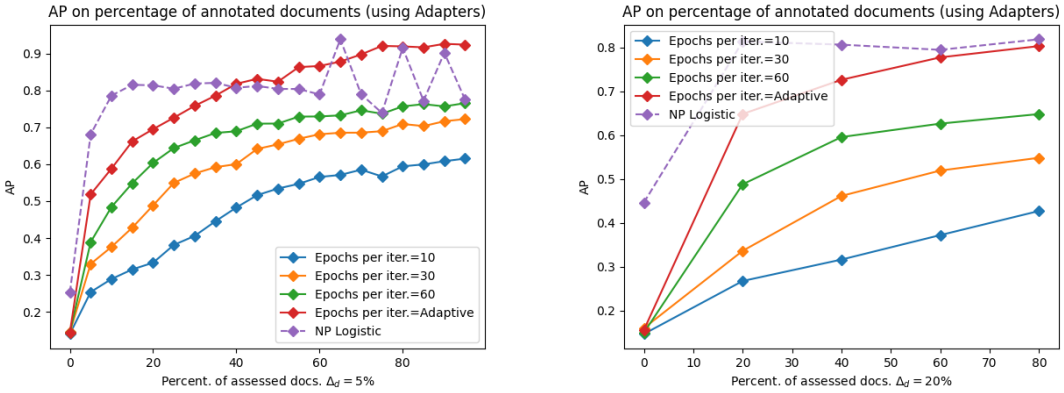
Figure 6.2: Variation of the Average Precision with different learning rates, annotating 5% (left) and 20% (right) of the documents at each iteration. We only train Adapter layers and freeze the rest of the network.

precision metric. As a matter of fact, the adaptive number of epochs can, at later stages, achieve a better MAP than the NP Logistic baseline; again, we notice that overall the gain in performance is much more consistent with the growth in training set size when using adapters.[8] In conclusion, we could say that, especially when using adapters, the number of epochs is a particularly sensitive hyperparameter (with respect to learning rate) which must be correctly adapted to the growing size of the training set; we believe future research on this topic might give new and interesting prospectives on the trainability of DL models in active learning scenarios.

Regarding the fine-tuning with adapters of the BioBERT algorithm, we only experimented with two different learning rates: (i) the default BioBERT learning rate in the HuggingFace library, i.e. 5e-5; (ii) the default learning rate in the AdapterHub library, i.e. 1e-4. The number of epochs, on the other hand, is fixed at 5 and $\Delta_d = 5\%$. This was done mainly for computational reasons, since, even with adapter layers, fine-tuning BioBERT can take very long times. Moreover, the results we were seeing from this initial set of experiments were not promising enough, and we decided against running further experimentation. As a matter of fact, looking at BioBERT results in Figure 6.4 we notice very poor performances, raising the question whether it is actually possible to train large language models when the dataset is relatively small, and the update is done in a continual fashion: indeed, despite testing with two very different learning rates, the results seem to be just slightly affected, with 5×10^{-5} being the better of the two, even though not significantly. That said, further experimentation with different number of epochs might give more promising results.

Finally, so far we have shown metrics on the full reranking, but we have not shown their CAL's ordering (which is what a reviewer would actually see). We plot how these orderings change as a function of the number of epochs (or learning rate, in BioBERT case) when $\Delta_d = 5\%$ and using adapter layers, to keep the number of plots to a minimum. We can see these orderings in Figure 6.5. Unsurprisingly, these plots show similar results to the previous ones, with the adaptive

[8] We also point out that this setup can achieve WSS@95% close to the baseline, albeit not as consistently as the baseline can.



(a) (b)

Figure 6.3: Variation of the Average Precision with different epochs, annotating 5% (left) and 20% (right) of the documents at each iteration. We only train Adapter layers and freeze the rest of the network.

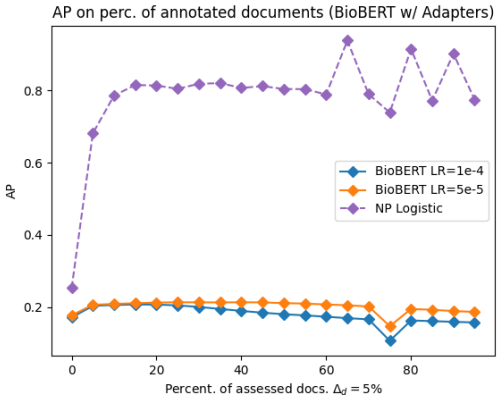
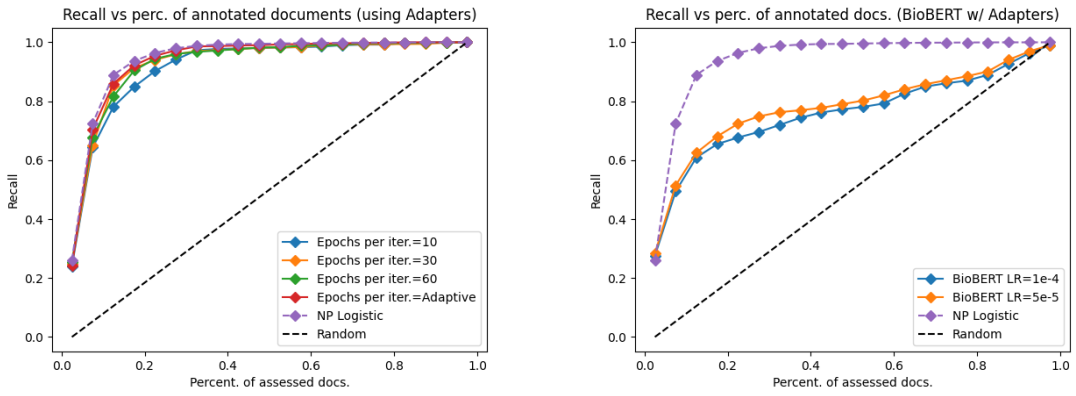


Figure 6.4: Variation of the Mean Average Precision with different learning rates for BioBERT, annotating 5% of the documents at each iteration. We only train Adapter layers and freeze the rest of the network.



(a) (b)

Figure 6.5: Variation on recall vs percentage of assessed documents due to different learning rates. Models have been trained with $\Delta_d = 5\%$. We show the CAL ordering of the different continuously trained models.

epochs obtaining the best results. That said, differences between the several epoch values tested are much smaller.[9] Furthermore, as we were seeing for the plots on BioBERT rankings, its CAL's ordering is also showing rather poor performances.

Finally, one critical aspect to consider when further studying the applicability of large language models to TAR is also the unavoidable increase in training times: at each active learning iteration we need to re-train the model, whose cost, when dealing with so many parameters, can be non-negligible. Indeed, even when only training adapters as we did here, training times can substantially increase: in order to complete the experiments (on all testing topics), BioBERT with adapters took about 20 hours; the DL ranker with adapters needed less than 7 minutes and the LR just 20 seconds.

In conclusion, this hyperparameter search can help us give a first tentative picture of what works and what does not, as well as finding directions for future works:

1. deep learning models, be them very large models or tinier ones, cannot be simply updated in an active learning process. Despite starting from a more or less good zero-shot capability, their performances quickly deteriorate when trained in these scenarios;

2. adapter layers can be a good solution for fine-tuning, except when the underlying frozen model is excessively large (especially compared to the size of the dataset). Indeed, models such as BioBERT are rather good at transfer learning, but cannot seemingly be fine-tuned in the CAL process;

3. some of the hyperparameters can be of key importance to the success of the models fine-tuning. Understanding how to adapt hyperparameters such as the learning rate and the number of epochs to the increasing training set size, as well as being able to assess how many documents

[9] This is somehow expected, since, at every iteration, the updated model can only have an impact on the top-k documents reviewed in the next batch, and not on the previous ones.

we review at each iteration, can make the difference between a decent model that can compete with current state of the art and a rather poor one.

Regarding our bullet point 2, it would be interesting to explore, in future works, which type of adapter layers (e.g., He et al. (2021); Houlsby et al. (2019); Pfeiffer et al. (2020)) can bring about the most promising increase in performances, and if the peculiar active learning scenario might require further adaptations or modifications of these techniques to fully leverage the zero-shot knowledge that we were seeing in Tables 6.1 and 6.2.

Finally, for bullet point 3, the correct choice of hyperparameters can be truly problematic since we lack a validation set. Indeed, it could be possible to extract a validation set before starting the review process, but this seems to make sense only when the dataset is large enough (e.g. not when we are dealing with a few hundreds of documents) and should be compared to a baseline which is also taking into account the presence of such a validation set. That said, it would be interesting to explore whether a more or less empirical rule of thumb can be found, which could allow one to select and/or adapt the hyperparameters without necessarily looking at a validation set; in this sense, adapting the number of epochs to the size of the current training set seems to be particularly effective and should be further studied.

6.6 Discussion

In this chapter, we explored whether using previous Systematic Reviews (SR) to pre-train machine learning models can actually bring better performances for a new SR topic, compared to no pre-training at all. Specifically, we also investigated whether deep learning models such as BioBERT or other transformer-like architectures can be effective, and to which extent. We conducted experiments on the CLEF TAR 2019 Task 2 dataset, and the results clearly show that pre-trained models can obtain good zero-shot rankings on both the Mean Average Precision and Work Saved over Sampling metrics (Section 6.5.1). When used with the CAL algorithm, we also see that jump-starting the active learning process from these zero-shot rankings can actually bring to a higher recall earlier in the assessment process (Section 6.5.2). Finally, we also noticed how continuously training our deep learning models results in inconsistent performances (usually, with a detrimental effects on the evaluation metrics): we then decided to conduct an extensive analysis on a hyperparameter search (Section 6.5.3). The aim of these latter experiments was to understand how and what we would need to change or research to effectively train deep learning models in an active learning process. Our results show that future works should focus on finding and (at least empirically) define a set of rules to adapt hyperparameters (e.g. epochs and learning rate) to the growing training set size.

Chapter 7

Conclusion

In this thesis, we have presented four main contributions, developed and published (or currently under revision) throughout the course of my PhD. All our works are focused on the Technology-Assisted Review (TAR) field: the main goal of TAR algorithms and frameworks is that of aiding human reviewers, usually via machine learning, to review a set of documents (usually referred to as "the pool"), minimizing the time (and cost) spent on the review, as well as maximizing the number of relevant items found (i.e., the recall). In other words, TAR aims at maximizing the cost-effectiveness of the review. Our work has exclusively focused on two of the most well-known application contexts in TAR, that is, e-discovery in civil litigation and the production of systematic reviews in empirical medicine. Both fields are characterized by a similar two-stage review: in e-discovery, the first stage focuses on finding responsive (i.e., relevant) documents, and it is usually conducted by junior reviewers; in systematic reviews, the reviewer reads only the abstract of documents, filtering out irrelevant documents. The second stage, in e-discovery, aims at finding privileged documents (i.e., containing privileged/sensitive information), and it is usually conducted by senior (more expensive) reviewers. In systematic reviews, the physician reads documents in their entirety, thus making the second stage more time-consuming. The review process is usually carried out with an active learning algorithm: the most used is currently Cormack and Grossman's Continuous Active Learning (CAL) (Cormack, Grossman, 2014), a variation of the relevance feedback policy (Rocchio, 1965) and of the active learning via relevance sampling (ALvRS (Lewis, Gale, 1994)) specifically adapted to e-discovery.

One important difference between the two fields is that, while for e-discovery real-world datasets are usually not publicly available (as they deal with delicate and privileged information), in systematic reviews we have freely available datasets (albeit mostly for the first stage only, as whole papers are usually under a paywall): we used one of these datasets in our experiments in both Chapter 5 and 6.

7.1 A summary of this thesis

Our work in this thesis has mainly focused on the application of the Saerens-Latinne-Decaestecker (SLD, Saerens et al. (2002)) algorithm to TAR workflows (Chapters 3, 4 and 5): SLD is an iterative algorithm whose goal is that of adjusting the *a posteriori* (posteriors) and *a priori* (priors) probabilities, coming from a classifier, in Prior Probability Shift (PPS) scenarios (i.e., when the

prior probability of the labelled set $\Pr_L(y)$ diverges from that in the unlabelled set $\Pr_U(y)$). Active learning policies such as CAL tend to generate substantial PPS, and thus SLD could be leveraged to improve our classifier estimates. However, in the experiments conducted during our work in Molinari (2019b), SLD showed disappointing performances when applied to MINECORE (Oard et al., 2018), a recent e-discovery cost-sensitive framework.

Chapter 3: In Chapter 3 we gave an in-depth analysis of the SLD algorithm for posterior probabilities adjustments (whose results were published in Esuli et al. (2021)): using the RCV1-v2 dataset to emulate single and multi class scenarios, we ran several experiments where we tested SLD against a variety of simulated PPS scenarios (from low to high drift). While SLD showed poor performances for single-label multi-class scenarios, it also obtained good results in the binary classification task (i.e., the one we are interested in for TAR applications), when we have a moderate or high PPS. It could thus be suitable for applications in active learning scenarios.

Chapter 4: Following up from the previous chapter, in Chapter 4 we first tried to establish the best active learning algorithm to generate the training set for MINECORE, comparing two of the most famous and used policies, i.e., active learning via relevance sampling (ALvRS) and uncertainty sampling (ALvUS), plus a newly presented ALvRUS policy (active learning via relevance/uncertainty sampling, see Section 4.3.3). This work was published in Molinari et al. (2023). Experimental results showed that ALvRUS is able to deliver a better training set, which in turns improve the classifier performances on the unlabelled set as well: moreover, as expected, it also generates a substantial PPS between the two sets. As argued in Chapter 3, high PPS and binary classification tasks seem to be the main ingredients for a successful SLD application. Nonetheless, when applied to AL generated datasets, SLD clearly showed poor performances and, more specifically, brought to an extremization of the posterior probability distribution, i.e., all (or almost all) posteriors are either pushed very close to 1 or 0. In order to analyze the problem, we introduced a new pseudo-oracle policy called *Rand*, whose goal is that of generating a labelled and unlabelled sets via a controlled random sampling: that is, given the prevalence $p_L(y)$ on the labelled set and $p_U(y)$ on the unlabelled set, we generate two analogous sets L^{Rand} and U^{Rand} with the same prevalences, but with a completely random document selection policy. This should enable us to understand whether SLD behaviours are due to some specific PPS or to the document selection policy: indeed, we find out that SLD extremization does not happen for the *Rand* policy. We thus hypothesize that SLD failures are due to the AL policy, and more specifically to what literature calls *sampling bias* (Dasgupta, Hsu, 2008; Krishnan et al., 2021): in ALvRS or ALvUS (as well as in ALvRUS) the document selection is highly influenced by the initial seed set; this means that the AL policy will draw very similar (and likely uninformative) documents from the pool, hiding from the classifier whole clusters of relevant items. Of course, this also makes the L set completely diverge from both the pool P and the U set (i.e., $\Pr_L(y) \neq \Pr_U(y)$ and $\Pr_L(x|y) \neq \Pr_U(x|y)$). Part of this analysis was also published in Esuli et al. (2022).

Chapter 5: In Chapter 5 we give an in-depth analysis of this phenomenon, showing how sampling bias strongly modifies the posterior distribution with respect to a similar sampling-bias-free dataset (i.e., generated with the *Rand* policy): we show that one of the key equations of SLD and its assumptions cannot hold in an active learning scenario. SLD posteriors update is thus shown to be too extreme. Hence, we propose a modification to the original SLD algorithm, introducing a new parameter τ, in order to avoid these extreme behaviours in SLD: this seems to effectively enable the

usage of SLD with AL generated datasets. We call our new method "SLD for Active Learning" (or SAL_τ). SAL_τ leverages the different classifiers generated at each active learning iteration, in order to estimate the sampling bias effects on the posterior probabilities. SAL_τ consequently adjusts the τ parameter, deciding whether to use the original SLD, a "milder" update or no update at all. Our final goal is then to use SAL_τ in order to get a better prevalence estimation (with respect to the classifier or SLD) during the active learning iterations: using the improved estimation, we can stop the active learning process earlier, while still achieving the required recall targets, resulting in lower annotation costs. From our experiments, the newly proposed SAL_τ method, and its variant SAL_τ^R, seem to indeed be able to achieve this goal. That said, future works should focus on better tackling SAL_τ early stopping issues, possibly defining a more statistically informed method than what we proposed with SAL_τ^R; moreover, it would also be interesting to adapt SAL_τ and SAL_τ^R to other sampling policies. Finally, the paper describing this work is currently under review at the Data Mining and Knowledge Discovery journal.

Chapter 6: In our last Chapter 6 we focused exclusively on the production of systematic reviews in empirical medicine. This work was published in Molinari, Kanoulas (2022). In this chapter, we explored the possibility of leveraging previously conducted systematic reviews to train deep learning models, which can eventually be applied to new and unseen systematic review topics. As a matter of fact, while in e-discovery we usually do not possess data from previous reviews, in empirical medicine datasets are publicly available (at least for the abstract screening stage). We thus investigated two main research questions: (i) is it possible to transfer knowledge between systematic review topics, and (if yes) to which extent? (ii) can we keep training these deep learning models in the active learning process? Regarding the first research question, we pre-trained a BioBERT model and another transformer-like architecture, as well as a classical logistic regression on the CLEF dataset. Results show that the deep learning models (and the logistic regression) are indeed able to transfer knowledge acquired on previous reviews, obtaining a zero-shot ranking with performances close to a logistic regression trained on in-domain data. We then proposed to "jump-start" the CAL process from the top-10 documents of the zero-shot rankings: however, despite starting from a higher recall, the jump started CAL process do not keep a substantial advantage over the classical CAL (although still showing improved performances). Regarding the second research question, our experiments have shown that deep learning models are subject to a great deterioration of their performance when continuously trained via active learning: in particular, BioBERT is the model whose performance deteriorates the most, making it basically worthless for the task at hand. Given this, we decided to run experiments using adapter modules (Houlsby et al., 2019), following the intuition that given the scarce amount of data items, having fewer parameters should deliver better results. Moreover, we decided to run a hyperparameter analysis to see whether changing or adapting both the learning rate and the number of epochs may bring improvements. Our results show that, when running with adapter modules and when adapting the number of epochs to the size of the training set, smaller deep learning models can be effectively trained in the active learning process, achieving results close to a standard logistic regression. We finally argue that future work should focus on more in-depth analysis of which hyperparameters matter most, and how to tune them in such a low-resource scenario. This would be of key importance, since leveraging previous reviews could bring to a non-negligible amount of saved annotation effort.

7.2 Limitations

We believe there are two main limitations to this thesis work:

1. The ubiquitous assumption of having infallible reviewers (Section 2.2.1), i.e., the fact that the human reviewer is assumed to always give the correct label to the correct item. This assumption is particularly relevant in Chapters 4 and 5. Chapter 4 analyses and studies the use of active learning policies in order to build the training set of MINECORE (Oard et al., 2018), as well as the effects of the SLD algorithm on a classifier trained on this training data. Hence, the simplifying assumption on reviewers made by Oard et al. (2018) is kept in our Chapter 4.

   Regarding Chapter 5, the assumption is made mostly to keep consistency, and in order to be able to compare with the baselines, where this assumption is also made.

2. The lack of real data for the e-discovery context. Indeed, while we tried to leverage real systematic reviews in Chapters 5 and 6, in Chapter 4 we used the same experimental setting of Oard et al. (2018) and, as such, the e-discovery data was simulated via the RCV1-v2 dataset (see Section 2.6). This was done in Chapter 5 as well, where the algorithms were tested both on the CLEF EMED dataset and the RCV1-v2 dataset. Of course, this poses the question of whether, when used in real e-discovery scenarios, the relative performance of the different algorithms tested (and in particular, of our SAL_τ method) would remain unchanged. Unfortunately, this limitation is common to most TAR literature and, hence, to the works presented in this thesis as well.

7.3 Future works and conclusions

In future works, we propose to further investigate the SAL_τ method in order to: (i) define a less empirical solution to its overestimation issues (with respect to SAL_τ^R); (ii) test its capabilities of improving posterior probabilities coming from active learning classifiers, both in terms of classification metrics (such as F1) and calibration metrics (e.g., using brier decomposition, see Section 6.3.3). Furthermore, regarding the deployment of deep learning algorithms in active learning processes, we believe further research would be necessary to understand how these models can be effectively integrated and leveraged in active learning algorithms: indeed, this could lead to larger initial seed set of documents in low-resource annotation scenarios, as well as to consistently achieve higher recall targets, earlier.

In conclusion, we believe this thesis has effectively contributed an in-depth analysis of SLD (for both posteriors and priors adjustment, in passive or active learning scenarios), as well as an effective method to save consistent annotation effort in TAR workflows; moreover, our SAL_τ method can also be employed to obtain better prevalence estimates in any active learning scenario. Finally, this thesis also gives a contribution towards enabling deep learning algorithms in TAR systems, in order to leverage their zero-shot capabilities.

Bibliography

Alasalmi Tuomo, Suutala Jaakko, Koskimäki Heli, Röning Juha. Better classifier calibration for small data sets // ACM Transactions on Knowledge Discovery from Data. 2020. 14, 3. 1–19.

Alexandari Amr, Kundaje Anshul, Shrikumar Avanti. Maximum likelihood with bias-corrected calibration is hard-to-beat at label shift adaptation // Proceedings of the 37th International Conference on Machine Learning (ICML 2020). Virtual Event, 2020. 222–232.

Baron Jason R, Braman R, Withers K, Allman T, Daley M, Paul G. The Sedona conference® best practices commentary on the use of search and information retrieval methods in e-discovery // The Sedona conference journal. 8, 2. 2007.

Bella Antonio, Ferri Cèsar, Hernández-Orallo José, Ramírez-Quintana María José. Quantification via probability estimators // Proceedings of the 11th IEEE International Conference on Data Mining (ICDM 2010). Sydney, AU, 2010. 737–742.

Bequé Artem, Coussement Kristof, Gayler Ross W., Lessmann Stefan. Approaches for credit scorecard calibration: An empirical analysis // Knowledge-Based Systems. 2017. 134. 213–227.

Brier Glenn W. Verification of forecasts expressed in terms of probability // Monthly Weather Review. 1950. 78, 1. 1–3.

Brown Tom, Mann Benjamin, Ryder Nick, Subbiah Melanie, Kaplan Jared D, Dhariwal Prafulla, Neelakantan Arvind, Shyam Pranav, Sastry Girish, Askell Amanda, others . Language models are few-shot learners // Advances in neural information processing systems. 2020. 33. 1877–1901.

Callaghan Max W., Müller-Hansen Finn. Statistical stopping criteria for automated screening in systematic reviews // Systematic Reviews. 2020. 9, 1. 1–14.

Cohen A. M., Hersh W. R., Peterson K., Yen Po-Yin. Reducing Workload in Systematic Review Preparation Using Automated Citation Classification // Journal of the American Medical Informatics Association. 03 2006. 13, 2. 206–219.

Cohen Aaron M., Ambert Kyle, McDonagh Marian. Cross-Topic Learning for Work Prioritization in Systematic Review Creation and Update // Journal of the American Medical Informatics Association. 2009. 16, 5. 690–704.

Cormack Gordon V. Email spam filtering: A systematic review // Foundations and Trends in Information Retrieval. 2008. 1, 4. 335–455.

Cormack Gordon V., Grossman Maura R. Evaluation of machine-learning protocols for technology-assisted review in electronic discovery // Proceedings of the 37th ACM Conference on Research and Development in Information Retrieval (SIGIR 2014). Gold Coast, AU, 2014. 153–162.

Cormack Gordon V., Grossman Maura R. Autonomy and reliability of continuous active learning for technology-assisted review. CoRR abs/1504.06868. 2015a.

Cormack Gordon V., Grossman Maura R. Multi-faceted recall of continuous active learning for technology-assisted review // Proceedings of the 38th ACM Conference on Research and Development in Information Retrieval (SIGIR 2015). Santiago, CL, 2015b. 763–766.

Cormack Gordon V., Grossman Maura R. Engineering quality and reliability in technology-assisted review // Proceedings of the 39th ACM Conference on Research and Development in Information Retrieval (SIGIR 2016). Tokyo, JP, 2016a. 75–84.

Cormack Gordon V., Grossman Maura R. Scalability of continuous active learning for reliable high-recall text classification // Proceedings of the 25th ACM Conference on Information and Knowledge Management (CIKM 2016). 2016b. 1039–1048.

Cormack Gordon V, Grossman Maura R. Navigating imprecision in relevance assessments on the road to total recall: Roger and me // Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval. 2017. 5–14.

Cormack Gordon V, Grossman Maura R. Systems and methods for a scalable continuous active learning approach to information classification. 2020. US Patent 10,671,675.

Cormack Gordon V., Grossman Maura R., Hedin Bruce, Oard Douglas W. Overview of the TREC 2010 Legal Track // Proceedings of the 19th Text Retrieval Conference (TREC 2010). 2010.

Cormack Gordon V., Mojdeh Mona. Machine learning for information retrieval: TREC 2009 web, relevance feedback and legal tracks // Proceedings of the 18th Text Retrieval Conference (TREC 2009). Gaithersburg, US, 2009.

Coussement Kristof, Buckinx Wouter. A probability-mapping algorithm for calibrating the posterior probabilities: A direct marketing application // European Journal of Operational Research. 2011. 214, 3. 732–738.

Dasgupta Sanjoy, Hsu Daniel. Hierarchical sampling for active learning // Proceedings of the 25th International Conference on Machine Learning (ICML 2018). Stockholm, SE, 2008. 208–215.

DeGroot Morris H., Fienberg Stephen E. The comparison and evaluation of forecasters // The Statistician. 1983. 32, 1/2. 12–22.

Dempster Arthur P., Laird Nan M., Rubin Donald B. Maximum likelihood from incomplete data via the EM algorithm // Journal of the Royal Statistical Society, B. 1977. 39, 1. 1–38.

Devlin Jacob, Chang Ming-Wei, Lee Kenton, Toutanova Kristina. BERT: Pre-training of deep bidirectional transformers for language understanding // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2019). Minneapolis, US, 2019. 4171–4186.

Domingos Pedro M., Pazzani Michael J. Beyond independence: Conditions for the optimality of the simple Bayesian classifier // Proceedings of the 13th International Conference on Machine Learning (ICML 1996). Bari, IT, 1996. 105–112.

Du Plessis Marthinus Christoffel, Sugiyama Masashi. Semi-supervised learning of class balance under class-prior change by distribution matching // Neural Networks. 2014. 50. 110–119.

Esuli Andrea, Fabris Alessandro, Moreo Alejandro, Sebastiani Fabrizio. Learning to Quantify. 47. 2023.

Esuli Andrea, Molinari Alessio, Sebastiani Fabrizio. A critical reassessment of the Saerens-Latinne-Decaestecker algorithm for posterior probability adjustment // ACM Transactions on Information Systems. 2021. 39, 2. Article 19.

Esuli Andrea, Molinari Alessio, Sebastiani Fabrizio. Active Learning and the Saerens-Latinne-Decaestecker Algorithm: An Evaluation // CIRCLE 2022: 2nd Joint Conference of the Information Retrieval Communities in Europe. 2022.

Esuli Andrea, Moreo Alejandro, Sebastiani Fabrizio. A recurrent neural network for sentiment quantification // Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM 2018). Torino, IT, 2018. 1775–1778.

Esuli Andrea, Moreo Alejandro, Sebastiani Fabrizio. Building automated survey coders via interactive machine learning // International Journal of Market Research. 2019. 61, 4. 408–429.

Esuli Andrea, Moreo Alejandro, Sebastiani Fabrizio. Cross-lingual sentiment quantification // IEEE Intelligent Systems. 2020. 35, 3. 106–114.

Fawcett Tom. An introduction to ROC analysis // Pattern Recognition Letters. 2006. 27. 861–874.

Fawcett Tom, Flach Peter. A response to Webb and Ting's 'On the application of ROC analysis to predict classification performance under varying class distributions' // Machine Learning. 2005. 58, 1. 33–38.

Fernandes Vaz Afonso, Izbicki Rafael, Bassi Stern Rafael. Quantification under prior probability shift: The ratio estimator and its extensions // Journal of Machine Learning Research. 2019. 20. 79:1–79:33.

Flach Peter A. Classifier calibration // Encyclopedia of Machine Learning. Heidelberg, DE: Springer, 2017. 2nd. 212–219.

Forman George. Counting positives accurately despite inaccurate classification // Proceedings of the 16th European Conference on Machine Learning (ECML 2005). Porto, PT, 2005. 564–575.

Forman George. Quantifying counts and costs via classification // Data Mining and Knowledge Discovery. 2008. 17, 2. 164–206.

Gao Wei, Sebastiani Fabrizio. From classification to quantification in tweet sentiment analysis // Social Network Analysis and Mining. 2016. 6, 19. 1–22.

Garg Saurabh, Wu Yifan, Balakrishnan Sivaraman, Lipton Zachary. A unified view of label shift estimation // Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Virtual Event, 2020. 3290–3300.

Gneiting Tilmann, Raftery Adrian E. Strictly proper scoring rules, prediction, and estimation // Journal of the American Statistical Association. 2007. 102, 477. 359–378.

González Pablo, Castaño Alberto, Chawla Nitesh V., Coz Juan José del. A review on quantification learning // ACM Computing Surveys. 2017. 50, 5. 74:1–74:40.

González-Castro Víctor, Alaiz-Rodríguez Rocío, Alegre Enrique. Class distribution estimation based on the Hellinger distance // Information Sciences. 2013. 218. 146–164.

Grossman Maura R, Cormack Gordon V. Inconsistent assessment of responsiveness in e-discovery: difference of opinion or human error // DESI IV: The ICAIL Workshop on Setting Standards for Searching Electronically Stored Information in Discovery Proceedings. 2011a. 1–11.

Grossman Maura R., Cormack Gordon V. Technology-assisted review in e-discovery can be more effective and more efficient than exhaustive manual review // Richmond Journal of Law and Technology. 2011b. 17, 3. Article 5.

Grossman Maura R, Cormack Gordon V. 'Reaffirming the Superiority of Human Attorneys in Legal Document Review and Examining the Limitations of Algorithmic Approaches to Discovery': Not So Fast // Rich. JL & Tech. 2020. 27. 1.

He Junxian, Zhou Chunting, Ma Xuezhe, Berg-Kirkpatrick Taylor, Neubig Graham. Towards a unified view of parameter-efficient transfer learning // arXiv preprint arXiv:2110.04366. 2021.

Houlsby Neil, Giurgiu Andrei, Jastrzebski Stanislaw, Morrone Bruna, De Laroussilhe Quentin, Gesmundo Andrea, Attariyan Mona, Gelly Sylvain. Parameter-efficient transfer learning for NLP // International Conference on Machine Learning. 2019. 2790–2799.

Huang Sheng-Jun, Jin Rong, Zhou Zhi-Hua. Active learning by querying informative and representative examples // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2014. 36, 10. 1936–1949.

Kanoulas Evangelos, Li Dan, Azzopardi Leif, Spijker René. CLEF 2019 Technology Assisted Reviews in Empirical Medicine Overview // Working Notes of the Conference and Labs of the Evaluation Forum (CLEF 2019). Lugano, CH, 2019.

Keeling Robert, Chhatwal Rishi, Gronvall Peter, Huber-Fliflet Nathaniel. Humans against the Machines: Reaffirming the Superiority of Human Attorneys in Legal Document Review and Examining the Limitations of Algorithmic Approaches to Discovery // Rich. JL & Tech. 2020. 26. 1.

Kingma Diederik P., Ba Jimmy. Adam: A method for stochastic optimization // Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). San Diego, US, 2015.

Koppel Moshe, Schler Jonathan, Argamon Shlomo. Computational methods in authorship attribution // Journal of the American Society for Information Science and Technology. 2009. 60, 1. 9–26.

Krishnan Ranganath, Sinha Alok, Ahuja Nilesh, Subedar Mahesh, Tickoo Omesh, Iyer Ravi. Mitigating Sampling Bias and Improving Robustness in Active Learning // arXiv preprint arXiv:2109.06321. 2021.

Lagopoulos Athanasios, Tsoumakas Grigorios. From Protocol to Screening: A Hybrid Learning Approach for Technology-Assisted Systematic Literature Reviews // arXiv preprint arXiv:2011.09752. 2020.

Landauer Thomas K., Foltz Peter W., Laham Darrell. An introduction to latent semantic analysis // Discourse Processes. 1998. 25, 2-3. 259–284.

Lease Matthew, Cormack Gordon V., Nguyen An Thanh, Trikalinos Thomas A., Wallace Byron C. Systematic review is e-discovery in doctor's clothing // Proceedings of the SIGIR 2016 Medical Information Retrieval Workshop (MedIR 2016). Pisa, IT, 2016.

Lee Jinhyuk, Yoon Wonjin, Kim Sungdong, Kim Donghyeon, Kim Sunkyu, So Chan Ho, Kang Jaewoo. BioBERT: a pre-trained biomedical language representation model for biomedical text mining // Bioinformatics. 2020. 36, 4. 1234–1240.

Lewis David D. Evaluating and optimizing autonomous text classification systems // Proceedings of the 18th ACM International Conference on Research and Development in Information Retrieval (SIGIR 1995). Seattle, US, 1995. 246–254.

Lewis David D., Catlett Jason. Heterogeneous uncertainty sampling for supervised learning // Proceedings of 11th International Conference on Machine Learning (ICML 1994). New Brunswick, US, 1994. 148–156.

Lewis David D., Gale William A. A sequential algorithm for training text classifiers // Proceedings of the 17th ACM International Conference on Research and Development in Information Retrieval (SIGIR 1994). Dublin, IE, 1994. 3–12.

Lewis David D., Yang Yiming, Rose Tony G., Li Fan. RCV1: A new benchmark collection for text categorization research // Journal of Machine Learning Research. 2004. 5. 361–397.

Li Dan, Kanoulas Evangelos. When to stop reviewing in technology-assisted reviews: Sampling from an adaptive distribution to estimate residual relevant documents // ACM Transactions on Information Systems. 2020. 38, 4. 41:1–41:36.

Michelson Matthew, Reuter Katja. The significant cost of systematic reviews and meta-analyses: a call for greater involvement of machine learning to assess the promise of clinical trials // Contemporary clinical trials communications. 2019. 16. 100443.

Molinari Alessio. Leveraging the transductive nature of e-discovery in cost-sensitive technology-assisted review // Proceedings of the 8th BCS-IRSG Symposium on Future Directions in Information Access (FDIA 2019). Milano, IT, 2019a. 72–78.

Molinari Alessio. Risk minimization models for technology-assisted review and their application to e-discovery. Pisa, IT, 2019b.

Molinari Alessio. CLEF EMED 2019 Dataset. X 2022.

Molinari Alessio, Esuli Andrea, Sebastiani Fabrizio. Improved Risk Minimization Algorithms for Technology-Assisted Review // Intelligent Systems with Applications. 2023. 200209.

Molinari Alessio, Kanoulas Evangelos. Transferring knowledge between topics in systematic reviews // Intelligent Systems with Applications. 2022. 200150.

Moreno-Torres Jose G., Raeder Troy, Alaíz-Rodríguez Rocío, Chawla Nitesh V., Herrera Francisco. A unifying view on dataset shift in classification // Pattern Recognition. 2012. 45, 1. 521–530.

Murphy Allan H. A new vector partition of the probability score // Journal of Applied Meteorology. 1973. 12, 4. 595—-600.

Naeini Mahdi P., Cooper Gregory F., Hauskrecht Milos. Obtaining well calibrated probabilities using Bayesian binning // Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015). Austin, US, 2015. 2901–2907.

Niculescu-Mizil Alexandru, Caruana Rich. Obtaining calibrated probabilities from boosting // Proceedings of the 21st Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI 2005). Arlington, US, 2005a. 413–420.

Niculescu-Mizil Alexandru, Caruana Rich. Predicting good probabilities with supervised learning // Proceedings of the 22nd International Conference on Machine Learning (ICML 2005). Bonn, DE, 2005b. 625–632.

Oard Douglas W., Sebastiani Fabrizio, Vinjumur Jyothi K. Jointly minimizing the expected costs of review for responsiveness and privilege in e-discovery // ACM Transactions on Information Systems. 2018. 37, 1. 11:1–11:35.

Oard Douglas W., Webber William. Information retrieval for e-discovery // Foundations and Trends in Information Retrieval. 2013. 7, 2/3. 99–237.

O'Mara-Eves Alison, Thomas James, McNaught John, Miwa Makoto, Ananiadou Sophia. Using text mining for study identification in systematic reviews: A systematic review of current approaches // Systematic Reviews. 2015. 4, 5. 1–22.

Patel Sakshi, Sihmar Shivani, Jatain Aman. A study of hierarchical clustering algorithms // 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom). 2015. 537–541.

Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay E. Scikit-learn: Machine learning in Python // Journal of Machine Learning Research. 2011. 12. 2825–2830.

Pfeiffer Jonas, Vulić Ivan, Gurevych Iryna, Ruder Sebastian. Mad-x: An adapter-based framework for multi-task cross-lingual transfer // arXiv preprint arXiv:2005.00052. 2020.

Pickens Jeremy. On the Effectiveness of Portable Models versus Human Expertise under Continuous Active Learning // 2nd International Workshop on AI and Intelligent Assistance for Legal Professionals in the Digital Workplace (LegalAIIA). 2021.

Platt John C. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods // Advances in Large Margin Classifiers. Cambridge, MA: The MIT Press, 2000. 61–74.

Pobrotyn Przemysław, Bartczak Tomasz, Synowiec Mikołaj, Białobrzeski Radosław, Bojar Jarosław. Context-Aware Learning to Rank with Self-Attention // arXiv:2005.10084 [cs]. VII 2020. arXiv: 2005.10084.

Pobrotyn Przemysław, Białobrzeski Radosław. NeuralNDCG: Direct Optimisation of a Ranking Metric via Differentiable Relaxation of Sorting // arXiv:2102.07831 [cs]. II 2021. arXiv: 2102.07831.

Pérez-Gállego Pablo, Castaño Alberto, Quevedo José Ramón, del Coz Juan José. Dynamic ensemble selection for quantification tasks // Information Fusion. 2019. 45. 1–15.

Quiñonero-Candela Joaquin, Sugiyama Masashi, Schwaighofer Anton, Lawrence Neil D. Dataset shift in machine learning. Cambridge, US: The MIT Press, 2009.

Rocchio J. J. Relevance feedback in information retrieval // The SMART Retrieval System: Experiments in Automatic Document Processing. Englewood Cliffs, US: Prentice-Hall, 1971. 313–323.

Rocchio Joseph J. Relevance feedback in information retrieval // Information Storage and Retrieval: Scientific Report No. ISR-9. 1965. XXIII.

Saerens Marco, Latinne Patrice, Decaestecker Christine. Adjusting the outputs of a classifier to new a priori probabilities: A simple procedure // Neural Computation. 2002. 14, 1. 21–41.

Satopaa Ville, Albrecht Jeannie, Irwin David, Raghavan Barath. Finding a" kneedle" in a haystack: Detecting knee points in system behavior // 2011 31st international conference on distributed computing systems workshops. 2011. 166–171.

Sayed Mahmoud F., Cox William, Rivera Jonah L., Christian-Lamb Caitlin, Iqbal Modassir, Oard Douglas W., Shilton Katie. A test collection for relevance and sensitivity // Proceedings of the 43rd ACM Conference on Research and Development in Information RetrievaL (SIGIR 2020). Xi'an, CN, 2020. 1605–1608.

Sebastiani Fabrizio. Evaluation measures for quantification: An axiomatic approach // Information Retrieval Journal. 2020. 23, 3. 255–288.

Settles Burr. Active learning. San Rafael, US: Morgan & Claypool Publishers, 2012.

Shemilt Ian, Khan Nada, Park Sophie, Thomas James. Use of cost-effectiveness analysis to compare the efficiency of study identification methods in systematic reviews // Systematic Reviews. 2016. 5, 140. 1–13.

Spence David, Inskip Christopher, Quadrianto Novi, Weir David. Quantification under class-conditional dataset shift // Proceedings of the 11th International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2019). Vancouver, CA, 2019. 528–529.

Stephenson D. B., Coelho C. A. S., Jolliffe I. T. Two extra components in the Brier score decomposition // Weather and Forecasting. 2008. 23, 4. 752–757.

Sun Meesun, Cho Sungzoon. Obtaining calibrated probability using ROC binning // Pattern Analysis and Applications. 2018. 21, 2. 307–322.

Tredennick John. TAR for Smart People // Catalyst. 2015.

Vapnik Vladimir. Statistical learning theory. New York, US: Wiley, 1998.

Vaswani Ashish, Shazeer Noam, Parmar Niki, Uszkoreit Jakob, Jones Llion, Gomez Aidan N., Kaiser Lukasz, Polosukhin Illia. Attention is all you need // Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS 2017). Long Beach, US, 2017. 5998–6008.

Viswanathan Meera, Patnode Carrie D, Berkman Nancy D, Bass Eric B, Chang Stephanie, Hartling Lisa, Murad M Hassan, Treadwell Jonathan R, Kane Robert L. Assessing the risk of bias in systematic reviews of health care interventions // Methods guide for effectiveness and comparative effectiveness reviews [Internet]. 2017.

Wang Shuai, Scells Harrisen, Mourad Ahmed, Zuccon Guido. Seed-Driven Document Ranking for Systematic Reviews: A Reproducibility Study // European Conference on Information Retrieval. 2022. 686–700.

Wei Jason, Bosma Maarten, Zhao Vincent Y, Guu Kelvin, Yu Adams Wei, Lester Brian, Du Nan, Dai Andrew M, Le Quoc V. Finetuned language models are zero-shot learners // arXiv preprint arXiv:2109.01652. 2021.

Whiting Penny, Savović Jelena, Higgins Julian PT, Caldwell Deborah M, Reeves Barnaby C, Shea Beverley, Davies Philippa, Kleijnen Jos, Churchill Rachel, others . ROBIS: a new tool to assess risk of bias in systematic reviews was developed // Journal of clinical epidemiology. 2016. 69. 225–234.

Wu Ting-Fan, Lin Chih-Jen, Weng Ruby C. Probability estimates for multi-class classification by pairwise coupling // Journal of Machine Learning Research. 2004. 5. 975–1005.

Yang Eugene, Lewis David D., Frieder Ophir. Heuristic stopping rules for technology-assisted review // Proceedings of the 21st ACM Symposium on Document Engineering (DocEng 2021). Limerick, IE, 2021a. 31:1–31:10.

Yang Eugene, Lewis David D, Frieder Ophir. On minimizing cost in legal document review workflows // Proceedings of the 21st ACM Symposium on Document Engineering. 2021b. 1–10.

Yang Eugene, MacAvaney Sean, Lewis David D, Frieder Ophir. Goldilocks: Just-Right Tuning of BERT for Technology-Assisted Review // arXiv preprint arXiv:2105.01044. 2021c.

Zadrozny Bianca, Elkan Charles. Transforming classifier scores into accurate multiclass probability estimates // Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2002). Edmonton, CA, 2002. 694–699.

Zhao Haozhen, Ye Shi, Yang Jingchao. An Empirical Study on Transfer Learning for Privilege Review // 2021 IEEE International Conference on Big Data (Big Data). 2021. 2729–2733.

Zhong Ruiqi, Lee Kristy, Zhang Zheng, Klein Dan. Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections // arXiv preprint arXiv:2104.04670. 2021.