IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 19, 2022, accepted June 6, 2022, date of publication June 17, 2022, date of current version June 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3184009

ICS: Total Freedom in Manual Text Classification Supported by Unobtrusive Machine Learning

ANDREA ESULI
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo," 56124 Pisa, Italy
e-mail: andrea.esuli@isti.cnr.it

This work was supported in part by the ARIADNEplus Project funded by the European Commission through the H2020 Program INFRAIA-2018-1 under Grant 823914, in part by the SoBigdata++ Project funded by the European Commission through the H2020 Program INFRAIA-2019-1 under Grant 871042, and in part by the AI4Media Project funded by the European Commission through the H2020 Program ICT-48-2020 under Grant 951911. The authors' opinions do not necessarily reect those of the European Commission.

ABSTRACT We present the Interactive Classification System (ICS), a web-based application that supports the activity of manual text classification. The application uses machine learning to continuously fit automatic classification models that are in turn used to actively support its users with classification suggestions. The key requirement we have established for the development of ICS is to give its users total freedom of action: they can at any time modify any classification schema and any label assignment, possibly reusing any relevant information from previous activities. We investigate how this requirement challenges the typical scenarios faced in machine learning research, which instead give no active role to humans or place them into very constrained roles, e.g., on-demand labeling in active learning processes, and always assume some degree of batch processing of data. We satisfy the "total freedom" requirement by designing an unobtrusive machine learning model, i.e., the machine learning component of ICS acts as an unobtrusive observer of the users, that never interrupts them, continuously adapts and updates its models in response to their actions, and it is always available to perform automatic classifications. Our efficient implementation of the unobtrusive machine learning model combines various machine learning methods and technologies, such as hash-based feature mapping, random indexing, online learning, active learning, and asynchronous processing.

INDEX TERMS Active learning, automatic text classification, online learning, machine learning.

I. INTRODUCTION

The task of text classification consists of selecting labels that are relevant to the content of a document. This label assignment process gives an explicit and structured form to the information that is latent and represented in an unstructured way in the text.

Classification enables the successive use of information processing/mining tools that otherwise would not be directly applicable to the original information represented using natural language. For example, it is possible to classify a stream of social posts to mark those relevant to a certain political topic, doing it for a set of topics emerging from an ongoing political debate. The availability of this classification enables to perform various tasks on the data, e.g.: to measure the variation of the engagement of the public over time and topics in order to identify which are the most relevant ones; to profile users' interests, possibly targeting each different profile with different messages; to select the content that is relevant to one specific topic in order to perform further analysis, e.g., sentiment analysis.

The classification of documents is an intellectual task that requires giving a semantic to the concepts represented by the labels and recognizing such concepts in the content of documents. Some concepts may be simple to define and recognize, e.g., the mention of a brand name, others may be much harder to give a clear and shared definition, e.g., the expression of sarcasm. With the exception of trivial tasks, i.e., those that can be solved by simple string matching, the effort required to read, understand the document, and match it to the relevant labels makes classification performed by humans a low-productivity activity that is expensive to scale. This is why the automatic classification of texts is a research topic that has a long history in computer science [1].

The leading approach to automatic classification is based on supervised machine learning [2], [3]. Supervised

The associate editor coordinating the review of this manuscript and approving it for publication was Pasquale De Meo.

information, i.e., a set of documents with their associated labels, is exploited by the machine learning algorithm to fit a classification function that is then applied to unlabeled documents to produce their automatic classification.

Developing an accurate automatic classifier thus always requires a human effort to define the classification schema and label the training examples for the learning algorithm. Yet, the machine learning research focus is on the learning algorithms, and the human effort of defining the classification schema, and the manual labeling activity are assumed to be already done and they are typically outside the focus of the research activity. Whenever human users are considered to have a role in the machine learning process, they are usually constrained to a limited set of actions to be performed in specific moments of the learning process, e.g., on-demand labeling in active learning processes [4]. These constraints mostly derive from the need to model the user within a rigorous scientific investigation, and we contend that these limits must be removed when implementing a practical application.

In this work, we define a ''total freedom of action'' scenario, where users are allowed to perform any action, e.g., label any document, modify the classification schema, at any moment. Giving such freedom to users is especially relevant in the early phases of the classification process when the concepts to be labeled emerge and are refined while the classification progresses. It is thus very important in these phases to allow users to freely create and modify the classification schema and label assignments.

The freedom of action we seek also includes the requirement that any machine learning-based functionality involved in the process should never interrupt users with the need to perform any machine learning-related action or let them wait for any machine learning-related process to complete. The machine learning component has to act as an observer, that eagerly follows the actions of the human classifiers and rapidly updates its automatic classifiers. The machine learning component must be always available to provide automatic classification for any document. Such automatic classifications can be used as suggestions presented to the users during their manual classification activity, thus implementing an active learning process, or to produce, once its accuracy is deemed to be adequate, the automatic classification of an entire dataset. We call this model of interaction between human and machine ''unobtrusive'' machine learning, because of the role of an always-available, non-interrupting observer that is played by the machine.

In this context, we developed, and present in this paper, the Interactive Classification System (ICS), a web-based application for manual text classification. ICS provides users with many interfaces and functionalities to perform manual text classification, all adhering to the requirement of giving total freedom of action to its users. ICS uses machine learning to support users in their activities and to produce automatic classifiers. Any machine learning element in ICS is implemented following the ''unobtrusive'' machine learning approach. As detailed in the rest of the paper, this approach is a challenge that asks for original machine learning methods to effectively and efficiently solve the problems it poses. The machine learning solution we implement in ICS is our first proposal for the problem, which we hope will lead to more interest in the topic.

ICS is implemented in Python. The source code is publicly available under an open-source license on GitHub (https://github.com/aesuli/ics) [5]. The software is published also as a package on the Python Package Index (https://pypi.org/project/ics-pkg/).

The main contributions in this article can be summarized as follows:

- We present the Interactive Classification System (ICS), a web-based application for manual text classification designed and implemented following the ''total freedom of action'' scenario.
- We motivate the need for the total freedom scenario, discussing how it challenges the assumptions typically made in machine learning research (e.g., batch indexing, batch learning, fixed schema…), and why existing machine learning methods do not fit this scenario.
- We propose an ''unobtrusive'' machine learning solution, based on an original combination of technological and theoretical solutions, which satisfies all the requirements set for the machine learning component of ICS.
- We show with experiments that the proposed solution is efficient, effective, and competitive against traditional approaches that are not constrained to the requirements of ICS.

The rest of the paper is organized as follows. Section II presents the related work, framing the context of our work and comparing ICS with similar existing systems. Section III describes the interfaces and functionalities provided by ICS to the users. Section IV details on the architecture and implementation of ICS, with a special attention on the machine learning component of ICS. In Section V we present experiments that evaluate the machine learning component of ICS, on four relevant classification problems (i.e., single-label classification, multi-label classification, binary sentiment classification, transfer learning). Conclusions are drawn in Section VI.

II. RELATED WORK

A. TERMINOLOGY

This paper follows the terminology currently used in machine learning research. Yet, given that its subject can be of interest to a diverse audience we will briefly discuss here the terms used over time and across disciplines to refer to the process of labeling documents according to a classification schema, before moving on to the specific topics of our work. The activity of labeling documents has been called in many different ways: text filtering [6]–[8], text routing [9], text categorization [10]–[12], text classification [13]–[15], text coding [16], [17]. The terms filtering and routing somewhat imply the goal of the classification process, i.e., to filter out

non-relevant documents or to route documents to different processing channels depending on their content. However, most papers that use these terms just focus on the accurate assignment of labels to documents, with no actual interest in the subsequent processing. The other three terms do not denote any assumption about the use of the assigned labels. In the domain of computer science, and especially in machine learning, the terms categorization and classification are used almost as synonyms. The term classifier is typically used to denote an actual instance of an automatic method that assigns labels, while the terms category and class are the ones most used to refer to the concept and properties represented by a label. The term label is often used as a synonym for the term category, especially when defining the constraints on how to assign labels, i.e.: in a single-label classification, a document must be assigned with one and only one label from the set of available labels, in a multi-label classification a document can be assigned with zero, one, or more labels. The last term, coding, is more frequently used in social sciences and market research, fields in which the classification activity is mostly carried out by humans (called coders) and rarely by means of automatic methods.

B. CLASSIFICATION SCENARIOS FOR MACHINE LEARNING

Research on automatic text classification addresses several machine learning problems and scenarios, each defined by a specific set of boundary conditions on the task.

The dominating research scenario is the one that considers a predetermined and fixed classification schema, a predetermined and fixed training set of labeled documents, and a predetermined and fixed test set of documents to be labeled. In this scenario, the human effort of defining the classification schema, and the manual labeling activity are assumed to be already done and are outside the focus of the research activity. This scenario dominates machine learning research because, by using fixed shared corpora and removing any human intervention from the process, it focuses the attention on the machine learning methods, supporting a rigorous, repeatable/reproducible, and comparable evaluation.

Other scenarios do consider some intervention of humans, i.e., in order to provide labels for a selection of previously unlabeled documents that an automatic learning process deems to have a relevant impact on the learned model. These scenarios are typically related to research on active learning [4], [18], which can be considered to belong to the emerging class of ''human in the loop'' processes [19]. The goal of the active learning process can be maximizing the accuracy given a fixed amount of human effort or minimizing the human effort required to reach a given level of accuracy. In both cases, the research focus is on the machine learning process, with humans playing the role of a resource that is expected to act as required by the process. [20] highlights some challenges that make the theoretical active learning process more realistic and close to an actual human-centered classification experience. Relevant to our work is the problem of considering the human as an oracle that acts on-demand, waiting for interrogations by the system. Batch-mode active learning methods [21], [22] let the human work on sets of documents at a time, before waiting for the retrain and the definition of a new batch. Even though some batch-mode active learning methods have shown competitive results with respect to sequential sampling [23], they do not satisfy our requirements, given that the learning algorithm still dictates the actions of the human actor. Relevant to our goals is the ''machine in the loop'' scenario [24], which switches the role of the human and the machine. This scenario puts the human at the center of the action, while a machine learning component observes and suggests actions to humans, collecting some eventual feedback to improve its models (see Section IV-B).

Some scenarios deal with the scarcity of labeled documents. In this context, some approaches assume the availability of labeled documents for very similar tasks, exploiting transfer learning methods [25]. This assumption may hold especially in sentiment analysis. Any language has sentiment-related expressions that are independent of the domain, making it possible to use a sentiment training set for a domain (e.g., product reviews) to perform sentiment classification on a different domain (e.g., posts on social networks) [26]. Other approaches focus instead on bootstrapping a classifier from external information sources [27]. An extreme approach is to generate a classifier leveraging only the names of the classes in the classification schema [28], [29]. In Section V-B we show how the use of online learning algorithms supports the reuse and adaptation of an already available automatic classifier trained on problems and domains similar to the one on which a new classifier has to be bootstrapped.

Research on supervised text classification always worked on a predefined classification schema, as the classification schema is the defining element of the task. The scenario in which a classification schema is not defined for a collection of documents is usually faced as a distinct problem that involves unsupervised learning methods, such as clustering [30], [31]. Yet, this is a scenario of interest in practical text classification problems as nowadays many events, expected or unexpected, can result in huge amounts of information published on the web, especially on social platforms. The timely analysis of such events can benefit from the classification of content, even though at the beginning of data collection there could be no clear idea of the classes the content will fit into. Our system supports this scenario by providing powerful tools to modify classifiers at any time (see Section III-B).

C. CLASSIFICATION INTERFACES

With respect to research on user interfaces for data labeling, there is limited literature on the topic, especially regarding text classification. A well-known tool for text annotation is GATE (General Architecture for Text Engineering, [32]). GATE has been originally designed for annotation of passages of text, e.g., traditional named entities [33], but its suite

of machine learning tools supports building complex NLP pipelines, e.g., to perform complex social media analysis [34] that also include document classification. GATE does not offer interactive functionalities in which models are updated live while humans produce annotations.

INCEpTION [35] is a web application that follows an approach that is very similar to ours, though working on text annotation rather than classification. The INCEpTION platform allows multiple users to collaboratively upload documents and annotate them, using complex annotation layers. An annotation layer can be connected to a recommender, i.e., a machine learning algorithm that is continuously updated on new annotations and can be used to present suggestions on text yet to be annotated. This is an example of unobtrusive machine learning, as the annotators are never imposed to perform a task by the machine learning component of the system. INCEpTION provides some support for modification of the annotation schema, yet the system does not check the consistency of previous annotations with the new schema, nor can adapt a recommender trained on the previous version of the schema.

BRAT [36] is an earlier example of a web application focused on manual text annotation. It supported complex annotation schema and collaborative annotation of multiple users. With respect to machine learning, it included only functionality for the automatic annotation of the current document by means of externally pre-trained machine learning models.

The Verbatim Coding System (VCS, [37], [38]) is a text classification application that exploits both human labeling and machine learning. When compared with the aims of this work, VCS follows a rather rigid and traditional approach to the problem, with the machine learning component at the center of its design. In VCS the classification schema must be defined in advance of any classification activity. Any change in the classification schema requires restarting the classification process from scratch. Machine learning training follows a typical batch processing approach: the machine learning classifiers are trained on request (e.g., when the user makes a guess that a sufficient number of documents have been labeled), and the training process takes some time to complete (because batch learning algorithms are used). A trained classifier can be used to perform training data cleaning [39], and/or active learning [40], yet these activities do result in an updated machine learning classifier only after an explicit request by the user, which then must wait for the training process to complete. Classification of unlabeled documents is performed in batches too, so the architecture of the system is not fit to produce on-the-fly classification suggestions.

Table 1 compares the systems described above and ICS. Each column identifies a functionality: fitting a machine learning model from a training set (batch learning); updating an existing model using novel examples (online learning); the possibility of implementing an active learning process; creating a model starting from an existing one; modifying a schema, i.e., adding/renaming/removing a label; combining parts of fitted models into a new model (merge of models, see Sections III-B3 and IV-B2); classifying/annotating a set of documents (batch predictions); quickly classify/annotate a single document (live predictions).

In the crowdsourcing model, a requester distributes many small, quick - and low-paid - jobs to a large population of workers, then collects the outcome of every single task to obtain a large set of processed data. The most famous platform for such activities is Amazon Mechanical Turk [41]. Classification, of different media types including text, is a frequent task in crowdsourcing. A worker is presented with some content to be classified and the set of possible labels from which to select those that are relevant. The requester must define the batch of classification tasks to be distributed to workers in advance. Therefore the classification schema must be also defined in advance, and workers have no means to change it. Due to this batch processing design, and other aspects related to trust in the workers' reliability, no crowdsourcing platform supports continuous interaction between humans and automatic classifiers. An online survey [42] collected feedback from users of online annotation platforms, asking for the ideal characteristics of ''your dream annotation platform'', one of the most desired features was ''Flexibility (ability to customize annotation project, labels, tasks, schema, in-/output formats)'', which is a requirement of this work.

III. ICS: ELEMENTS AND OPERATIONS

The activities carried out on the system revolve around two entities: datasets and classifiers. Both such entities have dedicated interfaces (Figures 1 and 7) that list their instances and give access to the actions that are relevant to them.

A. DATASETS

A dataset is a dynamic set of documents. The collection of documents forming a dataset can change over time, e.g., adding new tweets from a running filtered stream, even when classification is already ongoing. A user can perform manual classification of a dataset in two ways: ''browse and code'', and ''live classification''.

1) DATA UPLOAD

Documents can be added to a dataset in batches or single instances. The web application interface allows uploading a CSV file, in which a document is represented as an external unique identifier (to link it to its external source) and its text. The web service API has methods to upload a CSV file or to send text data directly in the POST request. This second method is used for example by a Twitter filtered stream script included in the package, which continuously collects tweets from Twitter and populates datasets on ICS.

2) BROWSE AND CODE

In the ''browse and code'' interface (Figure 2) the user is presented with the content of a document and the possibility to classify it using any selection of classifiers. The user

TABLE 1. Comparison of the machine learning functionalities implemented by the systems described in Section II and ICS. Notes: (*) requires re-training on the whole training set; in ICS active learning works together with online learning, not requiring a full re-train. (†) requires access to all the documents in the training set of the source domain; in ICS the transfer learning works using just the parameters of the model of the source domain, thanks to the combined use of an online learning algorithm, random indexing, and feature hashing (see Section IV-A).

System	Task	Batch learning	Online learning	Active learning	Transfer learning	Modify a schema	Merge of models	Batch predictions	Live prediction
GATE	Annotation, Classification	Yes	No	No	No	No	No	Yes	No
BRAT	Annotation	Yes	No	No	No	No	No	Yes	No
INCEpTION	Annotation	Yes	Yes	Yes	No	No	No	Yes	Yes
VCS	Classification	Yes	No	Yes*	Yes†	No	No	Yes	No
ICS	Classification	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

TABLE 1. Comparison of the machine learning functionalities implemented by the systems described in Section II and ICS. Notes: (*) requires re-training on

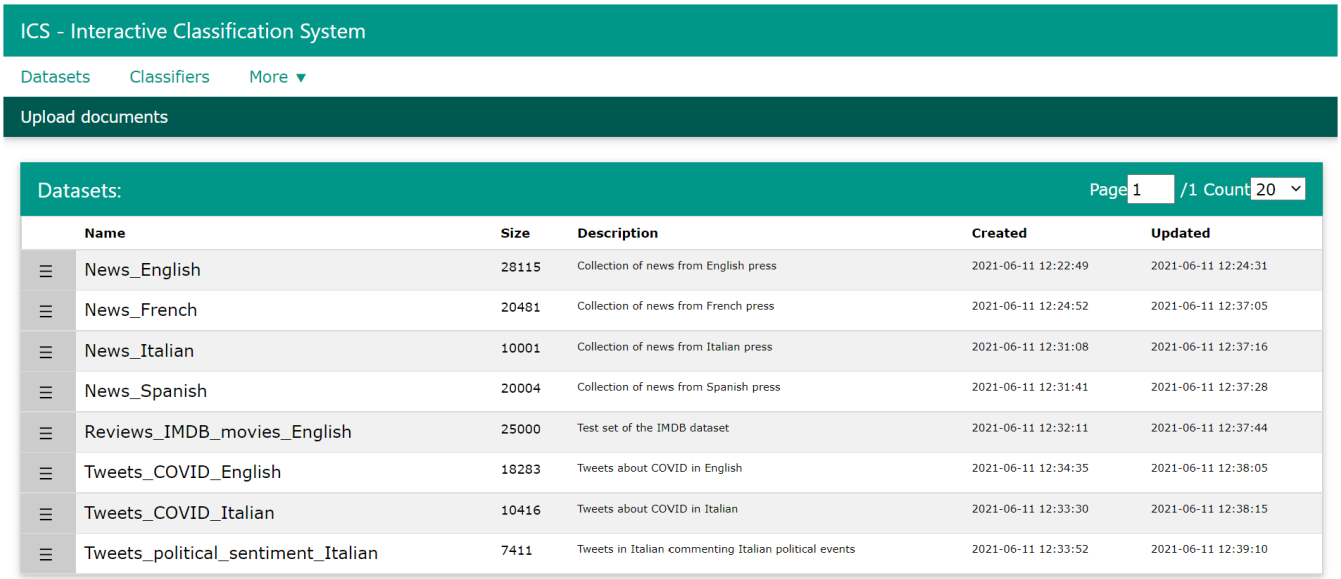

FIGURE 1. Main view of the dataset management interface. The list shows the names of datasets along with other information, e.g., number of documents, description. The three bars symbol on the left opens a menu of possible actions, e.g., go to browse and code mode, rename, delete, download.

can classify documents in any order. Any document can be accessed at any time by using a unique numeric id assigned at load time. A "Filter" text input field enables to work only on the subset of documents in the dataset that match a given query text. An "unlabeled" option restricts the browsing to documents that have an incomplete label assignment with respect to the current selection of classifiers. A number of custom browsing functions are implemented, from browsing documents in the same order they were loaded into the system, inverted order, random order, or following an active learning policy.

The selection of the classifiers to work with is completely free, and there are no constraints on which have to be actually used for a document. For example, two classifiers "brand" and "sentiment" are selected; if unsure about the "sentiment" classification, the user can classify the document only for "brand", moving to the next document and leaving the other decision to a later moment. Similarly, for a multi-label classifier (which is a collection of binary classifiers) there are no constraints on how to perform the labeling. For example, for a multi-label classifier with six labels, a user can mark a label as not relevant, other two as relevant, and skip the decision on the remaining three. All the cases of missing labels can be retrieved easily using the "unlabeled" filtering option of the browsing mode.

The actual classification by the user is performed by clicking/touching the labels to be assigned, which are then highlighted with a marked color. Note that if a document has been already classified, the assigned labels are shown and the user can eventually change the classification by clicking on a different label.

Multiple users can work simultaneously on the same dataset, each user free to use any browsing mode, and any selection of classifiers. Any classification produced by a user is immediately visible by any other user working on the same information (see Section IV). Any classification produced by a user is immediately notified to the machine learning component that updates the relevant automatic classifier accordingly (see Section IV-B).

3) ACTIVE LEARNING

The presence of the active learning browsing mode is one of the few hints about the existence of a machine

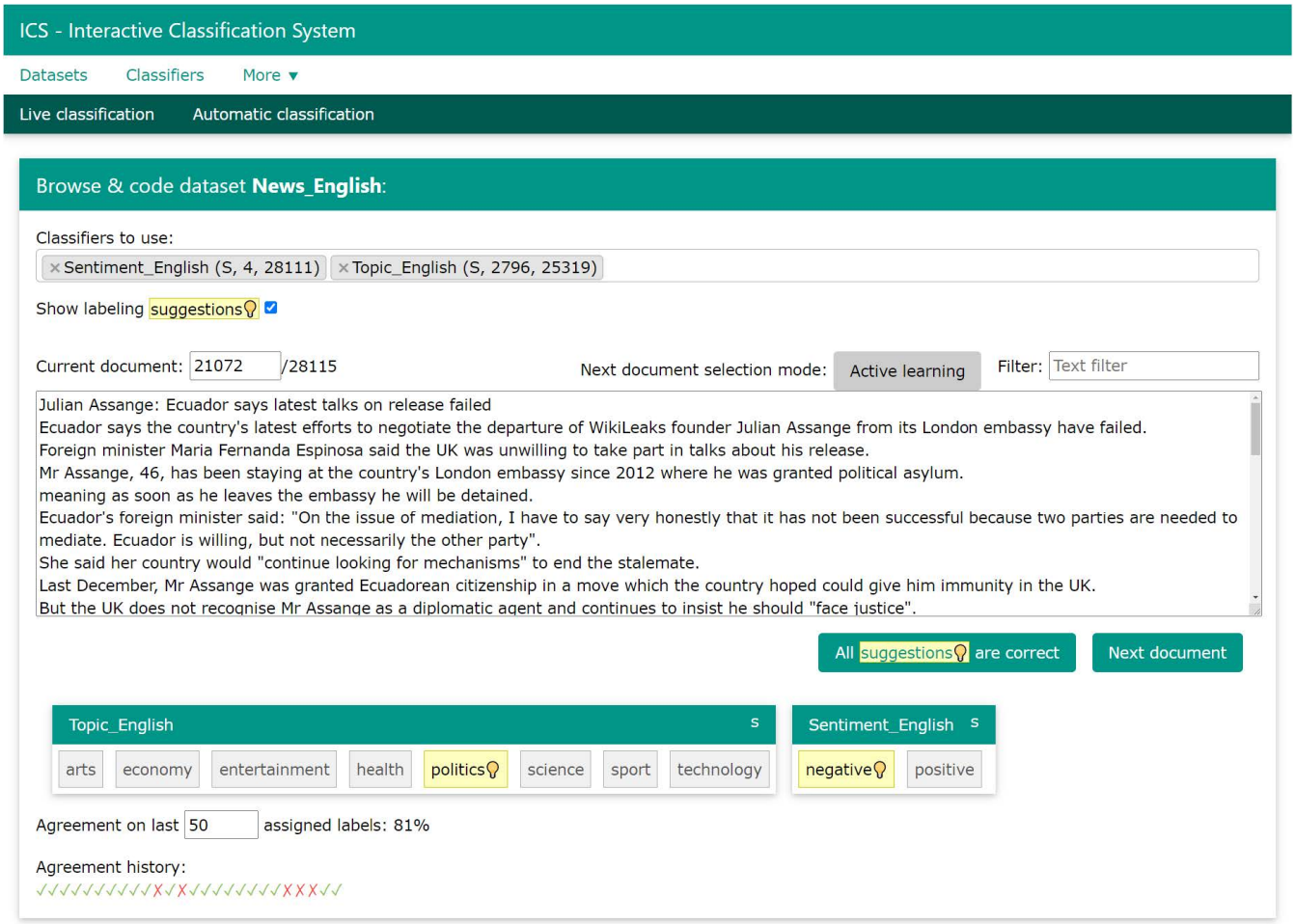

FIGURE 2. The "browse and code" interface. The upper part lets the users select the classifiers to be used and select the browsing mode. The central part shows the content of the document to be classified. The lower part shows the classifiers in use, with any eventual suggestion or already assigned label. Classification is done by directly clicking/touching the labels.

learning component in the system (the other two being label suggestions and automatic classification). Yet, active learning is implemented in a way that makes it no different in the interaction with the user from any other browsing method. The active learning browsing mode presents the user with a document that is deemed to contribute the most to improving the accuracy of the automatic classifier. When the user requests the next document to classify, a large random sample of documents in the dataset that are still unlabeled is quickly classified by the system and the one with the lowest confidence in the prediction is presented to the user as the next document to be classified. The user is not obliged in any way to classify the document, completely or partially, and can request another one. This selection policy is the uncertainty sampling policy, which is the best policy for text among the simpler, and faster active learning policies [40]. The use of a sample instead of the full dataset to perform the selection is dictated by performance requirements, as datasets may be arbitrarily large. Note that the sample is different for every request, thus the dataset is fully explored as requests are made.

4) LABEL SUGGESTION

As mentioned above, the machine learning component of the system can provide users with suggestions of which labels should be assigned to a document for the currently selected classifiers. These suggestions come in the form of a symbol and a color hint on the suggested labels (see Figures 3 and 4). Suggestions are always available, independently of which browsing mode is in use.

Whenever an automatic classifier is updated, any suggestion produced by that automatic classifier that is currently shown to any user is updated accordingly to the output of the updated version.

The user is not required to act any differently than when label suggestions are not shown. When label suggestions are shown, the interface also shows a bar with the history of agreements/disagreements and the agreement percentage (i.e., accuracy) over the last n labeling actions (see Figure 2, lower part).

The purpose of suggestions is twofold. The most obvious one is that label suggestions from a good automatic classifier can ease the work of the user. Less obvious, but of great

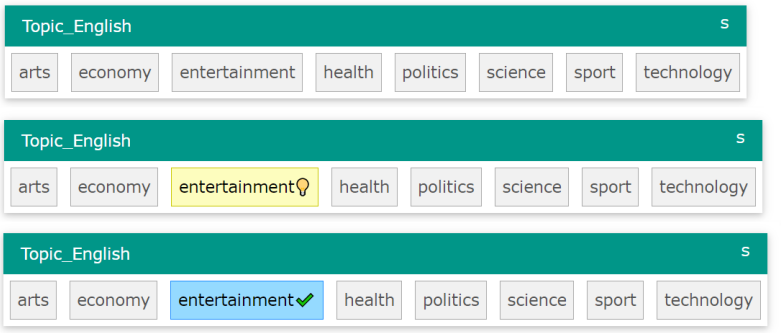

FIGURE 3. Visualization of classifiers and label assignments in the browse and code mode, single-label classifier. Top to bottom: no suggestion shown and no label assigned by users; suggestion from the automatic classifier shown; classification made by users.

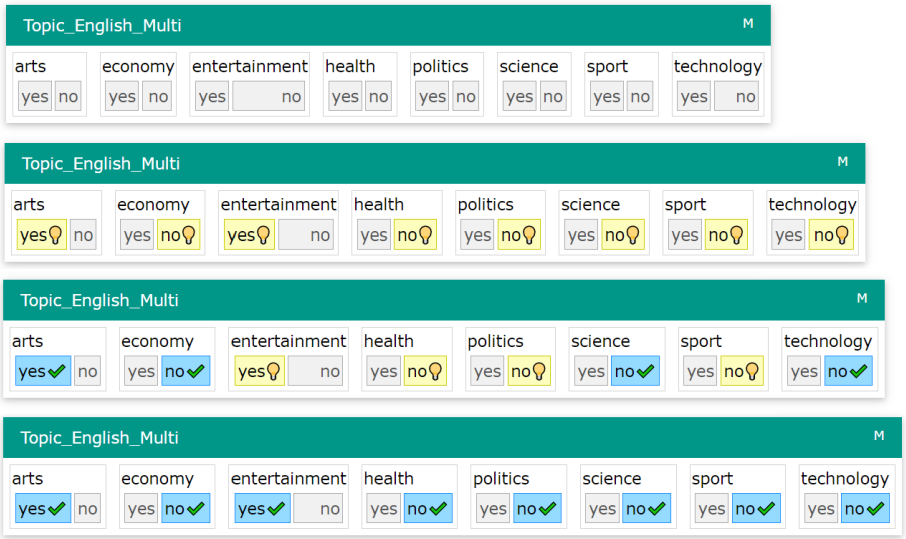

FIGURE 4. Visualization of classifiers and label assignments in the browse and code mode, multi-label classifier. Top to bottom: no suggestions shown and no labels assigned by users; suggestion shown; partial classification by users; complete classification by users.

importance, is allowing the user to collect evidence about the accuracy and robustness of the automatic classifier. This workflow, with an unobtrusive machine learning component in the loop, is meant to be a trust-building experience in the automatic classifier. Starting from the first suggestions, which would likely be almost random, users can see how their manual classification work translates into more accurate suggestions and increased agreement with automatic suggestions, to the point where enough trust is built on the automatic classifier to use it to produce automatic classifications of unlabeled documents. At any time during this process, nothing prevents the use of rigorous evaluation based on solid experimental protocols, e.g., k-fold validation, to quantitatively assess the quality of the automatic classifiers.

5) LIVE CLASSIFICATION

The "live classification" interface is a paged view that lists a number of documents (10, 20, 50…) alongside the labels assigned for a classifier (see Figure 5). Similarly to the browse and code mode, the user-assigned labels, or those suggested by the automatic classifier, are highlighted with specific colors and kept updated in the interface as new information is available. This classification mode may be preferred when focusing on a single classifier and exploiting the "Filter" function to quickly search and classify specific sets of documents.

6) AUTOMATIC CLASSIFICATION

The automatic classification mode is the only place in the dataset management part of the system that is actively focused on the automatic classifiers (see Figure 6). Given a dataset, it is possible to select a classifier and request the entire dataset to be automatically classified. The automatic classification is performed in the background, with the state of the process updated live in the interface. Once the process completes, a copy of the dataset with labels assigned to every document by the machine learning model can be downloaded.

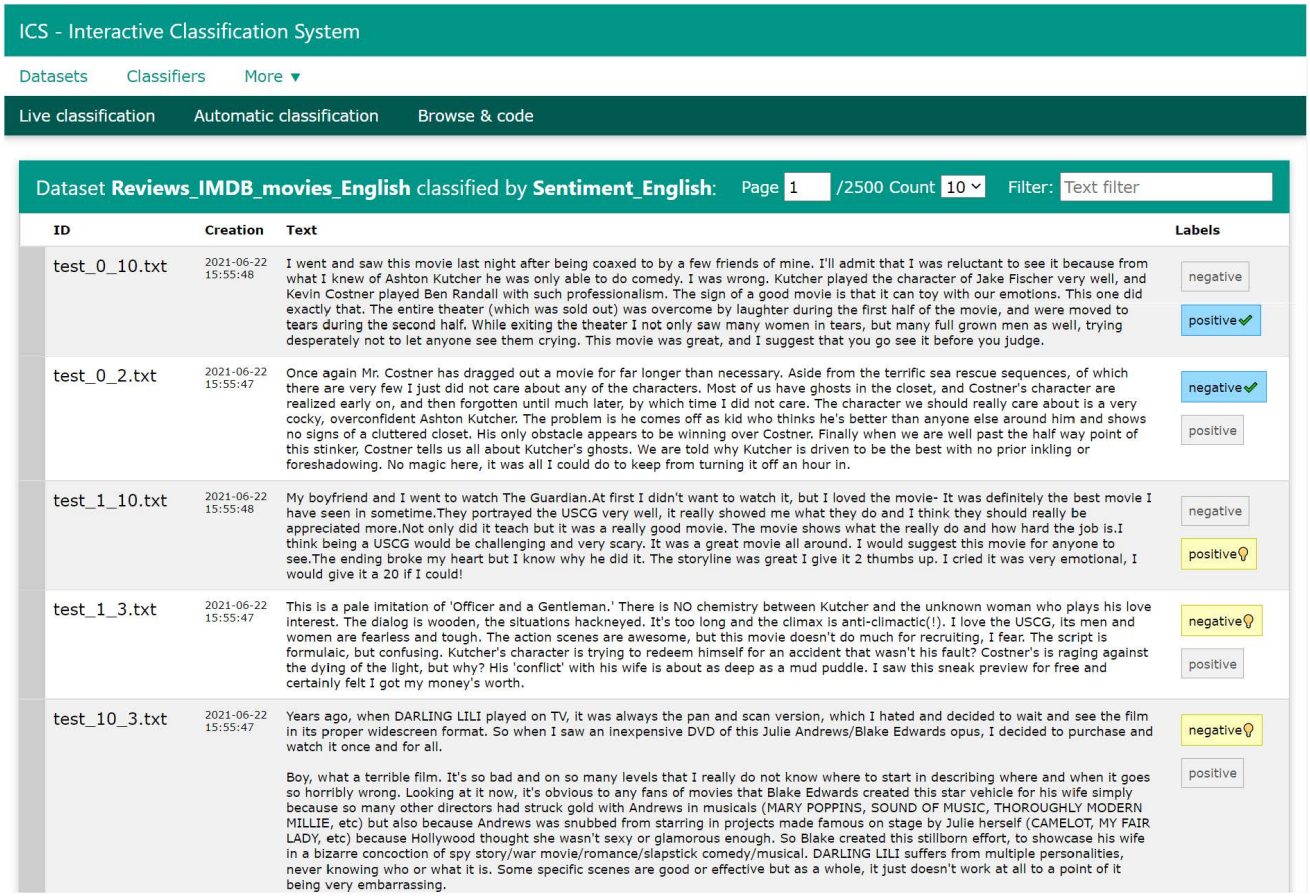

FIGURE 5. Live classification interface. Automatic suggestions and classification by users are shown similarly to the browse and code view. Any label assignment can be made or changed by clicking/touching on the label to be assigned. The text filter box allows the user to work on the subset of documents matching a given query.

The downloaded data also marks the source of any label, i.e., if it comes from a user or from an automatic classifier. Any user-assigned label has precedence over automatic-assigned ones. Note that automatic classification does not alter the user classification stored on the system. The classification it produces is only an output of the system.[1]

B. CLASSIFIERS

A classifier is defined by a set of labels and a labeling constraint (i.e., single-label or multi-label). For example, a user can define a "News topic" classifier with the set of labels "Politics, Economy, Entertainment, Sport, Health" and a single-label constraint. The web application (see Figure 7) allows users to create and modify classifiers. For every classifier, the system holds a machine learning model (detailed in Section IV-B) and its training data, which are updated whenever an event on the system produces relevant information.

The first event that is relevant for a classifier is its creation. A new classifier consists of an empty training set and a machine learning model that is initialized with random parameters. Datasets of already labeled documents, in the form of CSV files with text and assigned labels on each row, can be uploaded at any time into the system, resulting in the immediate update of the relevant classifiers.

A new classifier may be created also from existing classifiers. This is a key feature with respect to the possibility of reusing any information that has been accumulated in the past and that could be relevant for a current task. This refers not only to the simple reuse of an existing classifier as is but also to the interactive learning process [43] that enables the continuous evolution and adaptation of an automatic classifier to a new task. In Section V we show how reusing a classifier already trained for a task that is similar to a new one can boost the learning speed and accuracy of the new classifier.

The creation of new classifiers from existing ones is implemented by three different actions: copy, extraction, and merge.

1) COPY

The copy operation clones the information stored for the original classifier into the new one. The new classifier is thus immediately available to produce suggestions and to be

[1]Such output uses the same data format that is used to input training data (see Section III-B). This simplifies the process of uploading back into the system any externally validated data as additional training data for the classifier.

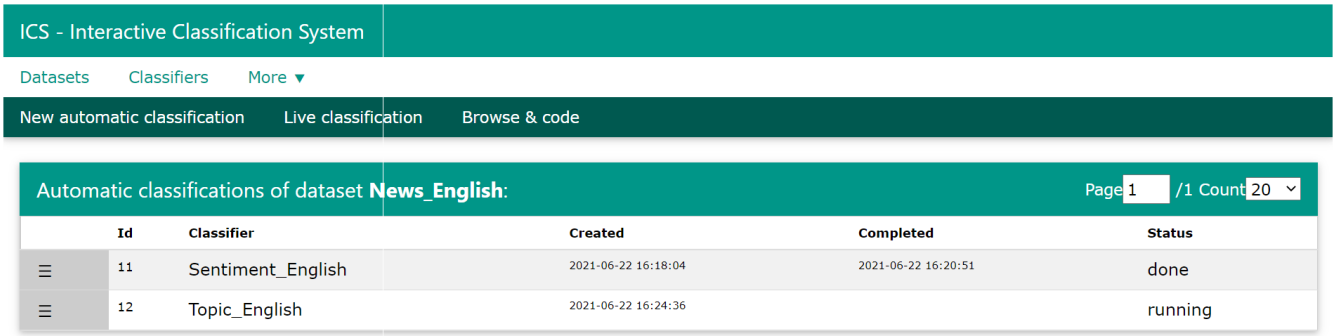

FIGURE 6. The automatic classification interface lists ongoing and completed automatic classifications. Automatically classified documents can be then downloaded as a CSV formatted file.

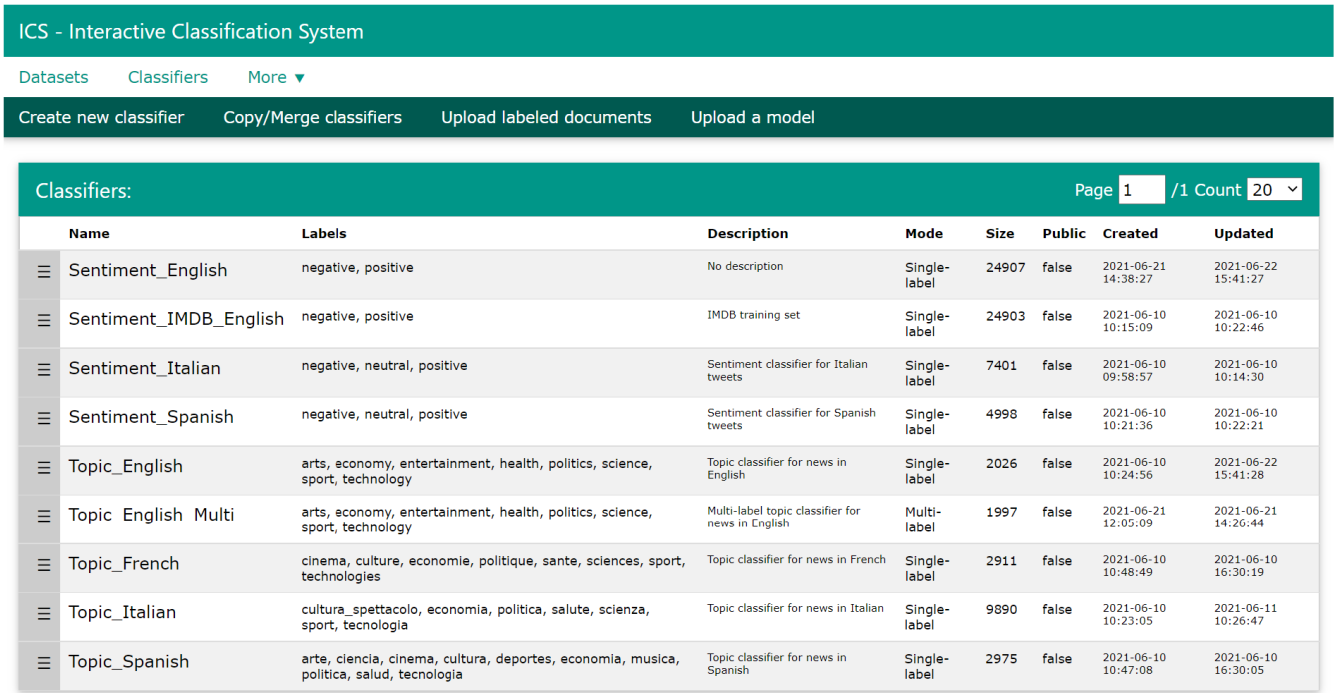

FIGURE 7. The interface for creation and management of classifiers. Similarly to the dataset interface (see Figure 1), it shows classifiers and their properties, each with a menu of action to modify them, e.g., rename/delete classifier, add/rename/delete labels, download training data or the classification model.

updated using new training data, independently of the original one. The copy operation is useful to perform versioning of a classifier, and whenever there is a transfer learning scenario. For example, a copy of a sentiment classifier trained on product reviews can be the starting point to work on a new dataset composed of tweets. The machine learning model of the original classifier will likely obtain a decent performance right from the beginning of the classification activity, reducing the effort required to obtain a good automatic classifier for the domain of the new dataset.

2) EXTRACTION

The extraction operation let the user select a label from a classifier and creates a new multi-label classifier with that single label, i.e., a classifier that classifies documents as either relevant or not relevant for the concept represented by the original label. The classification model of the new classifier consists of the part of the model of the original classifier that is devoted to the selected label. Similarly to the copy operation, the extracted model is immediately available for use. The extraction operation enables isolating a classifier for a specific label. For example, a classifier for the ''Economy'' class can be extracted from a single label classifier that classifies news by multiple topics, obtaining a binary classifier that labels documents as either relevant for ''Economy'' or not relevant. The extracted classifier can be then merged with other classifiers to create another new classifier, using the merge operation.

3) MERGE

The merge operation takes as input any number of classifiers, of any type, and combines them into a new classifier. The merge operation also takes as input the type of

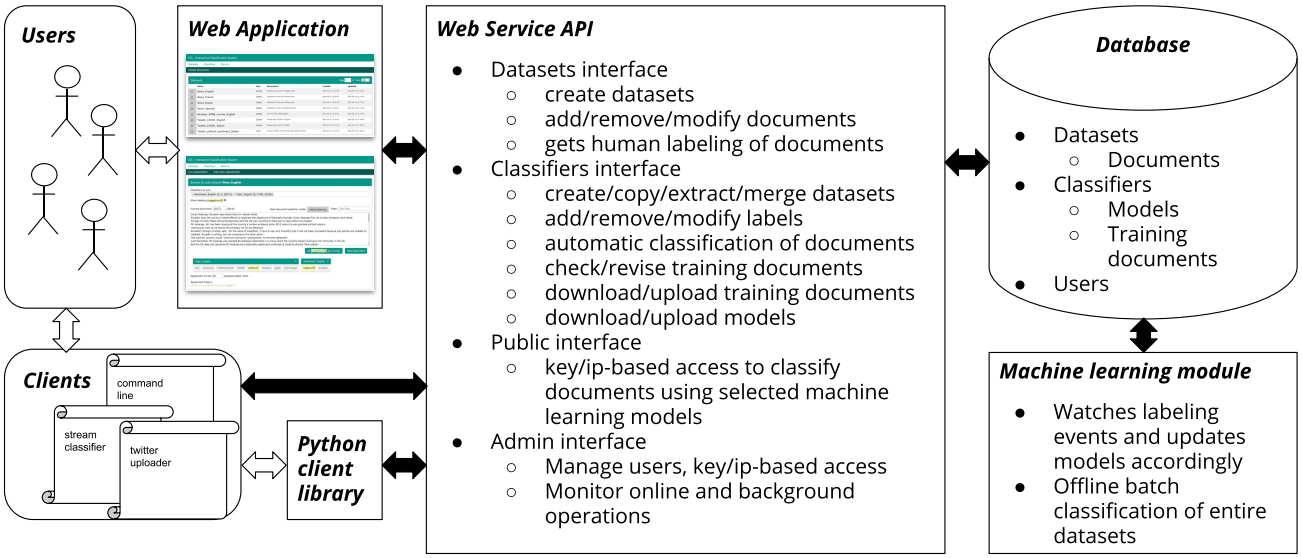

FIGURE 8. Architecture of the system.

labeling constraint (i.e., single-label or multi-label). The set of labels for the new classifier is the union of all the labels of the classifiers being merged. The merge operation is the most complex operation among the three described here, as it must combine information from models and training sets from different sources. Its implementation is detailed, along with the other machine learning-related operations, in Section IV-B.

The copy, extraction, and merge operations, along with add, rename, and remove label operations, enable users to quickly adapt at any time their classification schema to any new and changing classification task they face.

IV. SYSTEM ARCHITECTURE AND IMPLEMENTATION

The core of ICS consists of four main components, as depicted in Figure 8:

- A database that holds all the system information: documents, collections of documents, labels, classification schemata, automatic classification models, users, and all the relations that occur among them, e.g., assigned labels, ownership of objects, and access rights.
- A stateless[2] web service API that publishes all the functionalities of ICS. The methods are grouped into sub-services that pertain to a specific type of information: datasets, classifiers, users, and admin functionalities.
- A user interface, in the form of a web application that connects to the web-service API.
- A machine learning module that continuously observes classification events and updates automatic classification models accordingly.

Being stateless, the web services, the web application, and the machine learning modules can run in multiple instances and on different hardware, making them easily scalable. The database is the only element of the system that conceptually exists in a single instance, yet modern database technology allows to easily adopt solutions, e.g., replication, partitioning, to scale the system.

Additional components may be easily integrated using the web service API. The software package currently includes two applications:

- A command-line interface that implements all the methods published by the web service API. This allows browser-less access to the system and also provides a reference implementation in Python of all the requests that can be made to the system.
- A Twitter API[3] based application that continuously populates ICS datasets as tweets are collected in the background from a given set of filtered stream queries.

A. TEXT INDEXING

The scenario of use we set for ICS imposes relevant constraints on both the text indexing and the machine learning components.

The machine learning algorithm, which is described in the next Section, works with the vector representations of text used by most statistical machine learning methods, i.e., real-valued, high dimensional vectors $\mathbf{x} \in \mathcal{V} = \mathbb{R}^n$ where $\mathcal{V}$ is a high dimensional vector space. The transformation of a document into a vector consists of two phases: feature extraction and indexing.

The feature extraction process identifies in the text all the linguistic features (e.g., words, lemmas, n-grams, PoS, entities…) that may result in useful information for the machine learning algorithm. The set D of all the linguistic features

[2]The web service uses only a session token to manage user authentication.

[3]https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction

observed in the training set by feature extraction is called the feature space.

In this initial release of the software we adopt a simple (yet effective, as shown by experiments in Section V-A), language-agnostic processing that consists of splitting the text string on blank characters (e.g., space, tab) and punctuation. The feature extraction function also extracts word bi-grams and tri-grams, and character five-grams. Character n-grams are helpful with languages that do not use blanks as word separators (e.g. Chinese and Japanese). Some typical elements of text from social platforms (i.e., hyperlinks, @-based mentions, hashtags) are also recognized.

The typical text indexing process defines a vector space $\mathcal{V}$ as a one-to-one mapping with the feature space D. Every feature is mapped to a distinct, independent dimension of the vector space, $n = |D|$. All the subsequent elements of the machine learning pipeline use this mapping to properly link the features observed in a text to the parameters of the machine learning model.

Our training sets are expected to evolve continuously: when a single label is assigned to a document, and when batches of examples are added (via upload, or during the merge operations, see Section IV-B2). This makes the problem we face different from most experimental setups in text classification research in which a fixed training set of documents is assumed to be available since the beginning of the learning process. A fixed training set allows to perform a batch processing, defining the feature space, and thus the vector space, once and for all by means of a feature extraction pass on the whole training set.

Our scenario is closer to an active learning one, in which new examples are added to the training set one by one or in small batches. This means that the set of documents from which the training documents are sampled may evolve continuously. We thus cannot assume to be able to perform text indexing on a fixed dataset of documents (labeled or not) a single time, and then always use the vector space the indexing produced. However, indexing the training set every time it changes is not a viable solution, due to the cost of indexing, and also due to the need to not change the vector space if we want to use efficient online learning algorithms (see Sections IV-B and V-A).

We solved the above issues using a stateless indexing function based on Random Indexing [44], [45]. In random indexing, the feature space and the vector space are decoupled by a random projection function that maps every feature into a random vector in a vector space with a given number of dimensions, independent of the size of the feature space. A document is described as a vector in such space by summing up the random vectors for all the features that are extracted from it.

With respect to our goals, random indexing brings several advantages. It lets us control the size of the vector space, which directly determines the size of the machine learning models, allowing us to explore the trade-off between efficiency/high bias of smaller models and higher computational cost/low bias of larger models. It supports online learning algorithms, which require a fixed vector space. Compared to traditional indexing and to other projection methods, such as Latent Semantic Analysis (LSA, [46]), random indexing is much more efficient to compute, and also it produces a vector space in which learning algorithm run quicker, performing comparably in terms of accuracy [47]. More specifically, we adopted the Lightweight Random Indexing method (LRI, [47]), in which the random vector is constrained to have exactly two non-zero dimensions. This definition of random vectors maximizes the probability of orthogonality among vectors assigned to features. In experiments on text classification [47] LRI performed better, both in terms of efficiency and effectiveness, than any other random indexing method.

Algorithm 1 The Lightweight Random Indexing function

```
def LRI(features: list, n: int):
  # features is the list of features extracted from a document.
  # v is initialized as a vector of length n, the number
      of dimensions of the vector space, filled
      with zero values.
  v = zeros(n)
  for feature in features:
    # hash is the murmurhash3 hashing function.
    h1 = hash(feature, seed=0)
    # The hash values are 32-bit signed integers, with
      an equal probability of being greater or lesser
      than zero.
    # Modulus of hash determines the non-zero dimension.
    # We use the sign of the hash to determine the sign
      of the random component.
    if h1>0:
      v[h1%n] += 1
    else:
      v[h1%n] -= 1
    # We determine the second component of the
      random vector of the feature by changing the
      seed value.
    h2 = hash(feature, seed=1)
    if h2>0:
      v[h2%n] += 1
    else:
      v[h2%n] -= 1
  return v
```

A basic implementation of LRI uses a dictionary that maps each feature to its random vector. Whenever a feature is extracted from text, the dictionary is checked to retrieve the random vector, if it is missing a random vector is generated by means of some random number generator and added to the dictionary for future use. This implementation has a memory cost, to store the dictionary, i.e., feature-vector pairs, and a computational cost, to retrieve the vector given a feature.

The more efficient implementation replaces the use of an explicit, memorized dictionary, with the use of an implicit

dictionary based on hashing functions, a method known as feature hashing [48], or hashing trick. Feature hashing determines the vector representing a feature by means of a hashing function that takes in input the representation of the feature and returns a numeric value. That numeric value is then mapped, typically via modulus operation, to a dimension in the vector space, in which a $+1$ or -1 value (the sign is also determined from the hash) is set. Feature hashing removes the need to store the dictionary and is in theory able to handle feature spaces of infinite size. Algorithm 1 shows the pseudo-code that implements the LRI method based on feature hashing. Note that the hashing function is called twice, to determine the two non-zero dimensions of the random vector, using two different seeds, as different seeds determine completely independent outputs.

We use the same hashing function for all the classifiers. More specifically, our python implementation of LRI relies on the FeatureHasher class of scikit-learn package [49], which uses the Murmurhash3 [50] hashing function. Murmurhash3 takes in input an integer seed value to prime the random number generator. We always use the same seed value on the system. The vector space is thus the same for all classifiers and documents. This makes it possible to perform the merge operations among different classifiers without the need to perform remapping or projection operations among their vector spaces.

An interesting aspect of using LRI and feature hashing is that a working model can be shared without the need to share neither the training set nor the feature dictionary. Being the hashing function not reversible, and considering also the use of two non-zero dimensions by LRI, it is not possible to univocally map a single dimension to a specific feature in the feature space. This property of the implementation we adopted can be thus an aspect of interest in situations in which there is the will to share a classifier but there are privacy concerns or intellectual property constraints about sharing information, even single features, contained in the training set.

For example, documents in the training set of a sentiment classifier built on customers' feedback could contain names of customers, phone numbers, and similar critical information. An attacker trying to get information about the content in the training set may proceed only by guessing potential features, generating vectors for them, and checking how they correlate with the parameters of the model. The only features for which an attacker can get hints about their presence in the training data are those that have a high correlation with the supervised learning task, thus getting high weights in the model. These features are typically less critical with respect to privacy concerns as they model the concept that is the goal of the classification task (shall also those features be critical, it would mean that also the model should not be shared). Referring to the example, names and phone numbers in a sentiment classifier likely have irrelevant weights in the model, making them almost indistinguishable from any never observed feature, that would get a random vector picking some random weights in the model.

Another interesting possibility deriving from the use of hashing is that a user can deliberately change the seed value used for a new classifier, scrambling its feature mapping. That classifier can be publicly shared, but only those knowing the associated seed will get the real predictions, any other seed resulting in senseless predictions.

B. THE MACHINE LEARNING MODEL

The machine learning model used by the system is a vector space-based model, more specifically every label ℓ of a classifier is associated with a linear classification model, i.e., a vector of weights $\mathbf{w}_\ell \in \mathbb{R}^n$ that produces a prediction $\hat{y}_\ell$ for an input document x by means of a simple dot product based linear combination:

$$\hat{y}_\ell = f_\ell(\mathbf{x}) = \begin{cases} +1 & \mathbf{w}_\ell \cdot \mathbf{x} > 0 \\ -1 & \mathbf{w}_\ell \cdot \mathbf{x} \leq 0 \end{cases} \tag{1}$$

where $\hat{y}_\ell = +1$ means that the prediction is to assign the label to the document and $\hat{y}_\ell = -1$ means that prediction is to not assign the label to the document. Basically, the sign of $\mathbf{w}_\ell \cdot \mathbf{x}$ determines the assignment of the label and the magnitude $|\mathbf{w}_\ell \cdot \mathbf{x}|$ indicates the degree of confidence of the prediction.

A single instance of the model is thus a binary classification decision for a label of the classifier. The complete prediction for a classifier $\mathscr{C}$, likely composed of more than one label, is determined by combining the predictions of all the labels of the classifier, taking into account the classification constraints. The prediction of a multi-label classifier is simply union of all the predictions for the labels of the classifier, i.e.:

$$\hat{Y}^{\mathrm{m}}_{\mathscr{C}} = F^{\mathrm{m}}_{\mathscr{C}}(\mathbf{x}) = \bigcup_{\ell \in \mathscr{C}} f_\ell(\mathbf{x}) \tag{2}$$

The prediction of a single-label classifier assigns the label whose model returns the highest classification score, i.e.:

$$\hat{Y}^{\mathrm{s}}_{\mathscr{C}} = F^{\mathrm{s}}_{\mathscr{C}}(\mathbf{x}) = \arg\max_{\ell \in \mathscr{C}} \ \mathbf{w}_\ell \cdot \mathbf{x} \tag{3}$$

The learning algorithm used to fit the linear model is the Passive-Aggressive (PA) algorithm [51]. PA is an online learning algorithm that uses a hinge loss function:

$$l(y, \mathbf{x}, \mathbf{w}) = \begin{cases} 0 & y(\mathbf{w} \cdot \mathbf{x}) \geq 1 \\ 1 - y(\mathbf{w} \cdot \mathbf{x}) & y(\mathbf{w} \cdot \mathbf{x}) < 1 \end{cases} \tag{4}$$

The passive-aggressive name derives from the update rule of the algorithm, which does not change the model if the prediction is correct (passive case), while it updates the weights aggressively to correct any incorrect prediction. We adopt the PA-I version of the update rule:

$$\begin{aligned} \mathbf{w}_{t+1} = & \arg\min_{\mathbf{w} \in \mathbb{R}^n} \ \frac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + C\xi \\ & \text{s.t. } \xi \geq max(l(y, \mathbf{x}, \mathbf{w}), 0) \end{aligned} \tag{5}$$

which introduces a slack variable ξ and an aggressiveness parameter C that allows performing non-perfect correction of the weight when the update would require changing them too much. This makes the algorithm robust to outliers that

would corrupt an already well-fitted model to adapt to a single anomalous case.

1) TRAINING OF CLASSIFIERS

A newly created classifier is initialized with random weights, and thus it produces random label assignments, as expected from an untrained classifier that has never observed a training example.

Any label assignment made by a user with respect to any piece of text triggers both the update of the training set of the classifier that the label belongs to and the update of the machine learning model.

The training set logs all the pieces of text that have a label assignment from a user. The purpose of the training set is twofold. First, any request for a prediction that finds an exact match in the training set gets as the answer the user-assigned label, with a flag that tells that the assignment is from a user and not from a machine learning model.[4] The rationale is that whenever human-generated information is available, it is preferable to the machine learning one. Note also, that this approach can be considered a trivial machine learning model, that is able only to memorize and not to generalize. Whenever memorization fails (text was never seen before), generalization is used (the machine learning model makes a prediction). The second purpose of the training set is obviously to store the supervised information produced during labeling so as to be able to inspect it in the future and eventually make corrections (see Section III-A5), and to use it to train new models in all the cases in which it is better to train a new model from scratch rather than cloning existing models (see Section IV-B2).

For the machine learning model, the update process is asynchronous. A machine learning module (lower right corner of Figure 8) watches a queue of labeling events on the database. When a labeling event occurs, the relative classifier is retrieved from the database and updated. This is the core functionality of the machine in the loop scenario we aimed at.

The update process changes with respect to the labeling constraints. In the multi-label case, each label is handled separately from the others. Given a label ℓ of a classifier, the user may have clicked the ''yes'' or the ''no'' value for that label (see Figure 4), this determines if the example is a positive or a negative one. Only the model $\mathbf{w}_\ell$ is retrieved from the database, updated using the PA-I learning algorithm, and the updated model is loaded back to the database, replacing the previous one. In the single-label case, if a user selects a label (see Figure 3) it means that a positive example is generated for that label but also a negative example is generated for all the other labels. This means that multiple update processes will occur, eventually in parallel, as many as the number of labels in the classifier.

[4]This flag is by default not returned on the public classification interface, to avoid giving away information about the composition of the training set.

The machine learning module consists of multiple instances that work in parallel. Minimal locking is required to avoid the simultaneous update by two instances of the same model. Locks concern only the update operations, and models are always available to make predictions. The current machine learning model is used to make predictions until an updated version is available, which instantly replaces the previous model.

The other cases that trigger a training process are the upload of a training set and the merge operation. The case of upload is handled as the case of a single document labeling described above, with the only difference that many label assignments are available at the same time, and grouped in batches for processing by the machine learning module. The case of merge requires dedicated processing, as described in the next section.

2) MERGE

The merge operation is the only operation on classifiers that may require training of the new machine learning model. The extraction operation simply copies the relevant part of a model as is, while rename and deletion do not create new models.

In a merge operation, training from documents is not required when none of the source classifiers have any label in common with the others. In this case, each label of the new classifier is independent of the others and there are no training sets for a shared label to be merged, which would require training a label classifier on the union of the source training sets for that label. The machine learning models for every label of the merged classifier can thus be produced by copying the models for every label of the source classifiers.

If some source classifiers have any label in common with others, then each of the new models for those labels must be produced by performing training on the union of the training sets for the respective labels. For example, two classifiers, one with labels ''Ford'' and ''Renault'', the other with labels ''Fiat'' and ''Ford'' are merged. The new classifiers will have as the models for the labels ''Renault'' and ''Fiat'' the copy of the respective models from the relevant source classifiers. The model for the label ''Ford'' will be produced by training a new model on the union of the training sets stored for that label for each of the two source classifiers.

The merge operation works with incomplete information. There is no complete information about label assignments for all the documents in the training sets of source classifiers with respect to the entire set of labels of the target classifier. In the case of a multi-label constraint, a document in the training set of a source classifier may be relevant also for a label of a different source classifier. In the case of a single-label constraint, a label assignment for a document is also implicitly a negative example for all the other labels. This may be not the case for a label that originally belonged to a different classifier. Any policy that tries to fill in the missing labels, e.g., by assuming they are not assigned, introduces a baseless bias in the resulting model. For this reason, the actual

implementation of the merge operation, as described above, does not make any assumption about the missing labels, leaving the improvement of the target classifier to successive activities of revision of training data and labeling of more documents.

V. EXPERIMENTS

The accuracy of classification is always a key metric to evaluate an automatic classification method. In the case of systems such as ICS, in which the attention is on the experience of the human classifier, also the efficiency and responsiveness of the system play a relevant role. We have run experiments[5] to compare different active learning (Section V-A) and batch learning algorithms when deployed as the machine learning component of the system. We have also run experiments on using the system to perform transfer learning, i.e., leveraging an already fitted model to bootstrap a new classifier that is targeted for a slightly different problem or domain (Section V-B).

A. ACTIVE LEARNING

These experiments are focused on measuring which configuration of machine learning algorithm and browsing mode achieves the best performance, in terms of accuracy and efficiency of the learning process.

1) CONFIGURATION

The experiments are performed on three datasets selected to represent different labeling scenarios, covering topic and sentiment classification, binary, single-label, and multi-label classification:

- IMDB [52], a dataset of 50,000 movie reviews with binary positive/negative sentiment classification.
- 20 Newsgroups [53], a dataset of 20,000 newsgroup posts single-labeled on 20 topic labels.
- Reuters 21578 [54], a dataset of 21,578 news published on Reuters Newswire, multi-labeled on 113 topic labels.

We simulated a user that starts classifying the dataset from scratch, labeling the documents adopting the active learning browsing mode described in Section III-A3. We tested a number of different configurations:

- SVM - A linear SVM.[6] Every time a new training example is available the SVM retrains the model from scratch on the whole set of training examples that have been labeled so far. This batch learning method is expected to perform well in terms of accuracy, but it has a higher computational cost than the online learning algorithms.
- PA-1 - The PA algorithm performs a straightforward online learning step, i.e., an incremental update of its model executing the training function on the new labeled document.
- PA-L-*N* - When a new training example is available, the PA algorithm performs an online learning step on a "mini-batch" of training documents composed of the last N labeled documents.[7] We tested the values 10 and 100 for *N*, in this and the next configuration.
- PA-R-*N* - Similar to PA-L-*N*, but the mini-batch of training documents is composed of the newly labeled document plus $N - 1$ documents randomly selected among the previously labeled ones[7].

Given a dataset, all the documents in the dataset are loaded into the system, and then we simulated the labeling of 1,000 documents by the user. After each simulation step we measured the time required by the learning algorithm to update the model and the accuracy of classification on the whole dataset, i.e., using the human label for the documents labeled until the current step, and prediction from the current model for the documents still unlabeled. For each method, the active learning browsing mode uses the classification scores returned by the method itself. We also tested the use of a random browsing mode, in which the next document to label is randomly selected among the unlabeled ones. We ran each experimental setting 10 times and averaged the results.

Both the SVM and the PA algorithms have various parameters to fine-tune their behavior to the specific data being processed. The fine-tuned value of the parameters is usually determined when the training set creation process is completed and the user aims at obtaining the best performance from it. The fine-tuning process involves repeated experiments that explore a portion of the many configurations parameters can have, usually adopting a k-fold validation protocol or similar protocols. The fine-tuning process is thus a costly process that does not fit in the scenario we are facing in this work.[8] For this reason, we set any learning algorithm tested in our experiments to use the default value for any parameter not explicitly mentioned in this paper.

2) RESULTS

Tables 2, 3, and 4 report the accuracy figures for the various tested configurations. IMDB and 20 Newsgroups are single-label datasets, so accuracy and macro-averaged measures for precision, recall, and F_1 are reported. Reuters 21578 is a multi-label dataset, so micro- and macro-averaged measures for precision, recall, and F_1 are reported. Reuters 21578 has only 5 labels that are assigned to more than 5% of documents in the dataset, and 62 labels that are assigned to less than 1.5% of documents in the dataset [55]. In cases like this one, of a multi-label setup with high imbalance, the accuracy measure is not informative, as it is dominated by the vast majority of "simple" predictions of not-assigned labels. This

[5]The code to replicate the experiments is available at https://github.com/aesuli/ics-exp

[6]In preliminary experiments we tested other learning algorithms, i.e., Polynomial SVM, Random Forest, Multinomial Naïve Bayes. Linear SVM performed better in almost any test and thus we selected it as the reference method for batch learning.

[7]If the set of labeled documents is smaller than N, then all the available labeled documents are used.

[8]Eventually, fine-tuning may happen once a large enough training set is created using our system, to give a final boost to the automatic classifier.

TABLE 2. Results on the IMDB dataset of active/random learning experiments using different learners and configurations. Measured after 1,000 documents have been labeled.

Browsing mode	Learner	Accuracy	macro- Precision	macro- Recall	macro- F_1
Active	SVM	0.806	0.807	0.806	0.806
	PA-1	0.680	0.732	0.680	0.658
	PA-L-10	0.763	0.772	0.763	0.761
	PA-R-10	0.781	0.791	0.781	0.779
	PA-L-100	0.789	0.790	0.789	0.788
	PA-R-100	0.797	0.801	0.797	0.797
Random	SVM	0.785	0.786	0.785	0.785
	PA-1	0.723	0.746	0.723	0.716
	PA-L-10	0.748	0.752	0.748	0.747
	PA-R-10	0.765	0.773	0.765	0.763
	PA-L-100	0.758	0.763	0.758	0.756
	PA-R-100	0.774	0.778	0.774	0.773

TABLE 3. Results on the 20 Newsgroups dataset of active/random learning experiments using different learners and configurations. Measured after 1,000 documents have been labeled.

Browsing mode	Learner	Accuracy	macro- Precision	macro- Recall	macro- F_1
Active	SVM	0.461	0.474	0.454	0.454
	PA-1	0.279	0.384	0.273	0.257
	PA-L-10	0.367	0.444	0.363	0.366
	PA-R-10	0.442	0.474	0.436	0.436
	PA-L-100	0.440	0.458	0.435	0.435
	PA-R-100	0.473	0.490	0.467	0.469
Random	SVM	0.465	0.458	0.455	0.447
	PA-1	0.262	0.393	0.258	0.236
	PA-L-10	0.318	0.415	0.313	0.303
	PA-R-10	0.418	0.449	0.410	0.406
	PA-L-100	0.396	0.429	0.390	0.386
	PA-R-100	0.450	0.451	0.442	0.438

TABLE 4. Results on the Reuters 21578 dataset of active/random learning experiments using different learners and configurations. The upper half of the table reports micro-averaged measures, the lower half reports macro-averaged measures. Measured after 1,000 documents have been labeled.

Browsing mode	Learner	micro- Precision	micro- Recall	micro- F_1
Active	SVM	0.770	0.644	0.701
	PA-1	0.749	0.610	0.672
	PA-L-10	0.718	0.654	0.684
	PA-R-10	0.738	0.686	0.711
	PA-L-100	0.710	0.693	0.702
	PA-R-100	0.735	0.691	0.712
Random	SVM	0.772	0.388	0.516
	PA-1	0.740	0.363	0.486
	PA-L-10	0.644	0.476	0.547
	PA-R-10	0.697	0.508	0.587
	PA-L-100	0.658	0.522	0.582
	PA-R-100	0.681	0.529	0.595
		macro- Precision	macro- Recall	macro- F_1
Active	SVM	0.373	0.182	0.225
	PA-1	0.210	0.134	0.152
	PA-L-10	0.231	0.187	0.195
	PA-R-10	0.253	0.199	0.212
	PA-L-100	0.229	0.216	0.216
	PA-R-100	0.252	0.216	0.226
Random	SVM	0.161	0.041	0.056
	PA-1	0.117	0.034	0.043
	PA-L-10	0.159	0.084	0.094
	PA-R-10	0.199	0.094	0.113
	PA-L-100	0.184	0.104	0.117
	PA-R-100	0.203	0.111	0.129

is confirmed in our experiments on Reuters 21578, in which accuracy is higher than 99% for all the experiments we run.

Comparing the two browsing modes, the Active one performs better than the Random one in 17 of the 18 tested configurations (6 learners by 3 datasets). Averages by datasets are largely in favor of the active learning browsing mode. The most relevant improvements are observed for the macro-averaged measures on Reuters 21578. Reuters 21578 consists of 113 binary imbalanced classification problems, the differences observed on IMDB are amplified, with the Active configurations performing much better than Random ones. On Reuters 21578 it is important to note that the active learning browsing mode results in a better balance between precision and recall than the random one. In both browsing modes precision is higher than recall. This is somewhat expected, given the high imbalance of the dataset that makes it much harder to find positive examples for rarer labels.

On IMDB, SVM always performs better than any PA-based configuration, with PA-R-100 being the second-best method. On 20 Newsgroups and Reuters 21578, PA-R-100 is the only configuration that performs better than SVM.

Within the PA-based configurations, the PA-R-100 one always performs better than the others. The "R" strategy for mini-batch-based learning always performs better than the "L" strategy. This indicates that the learning algorithm benefits from being exposed to varied samples of documents from the training pool. A larger size of the mini-batch always produces better results.

Figure 9 shows how accuracy/macro-F_1 varies step after step of the labeling simulation. PA-L-*N* configurations are not shown, their curves are similar to the relative PA-R-*N* versions, only a bit lower. The differences observed in Tables 2, 3, and 4, are confirmed across the whole learning curves. On IMDB and 20 Newsgroups, PA-R-100 and SVM follow very similar paths. On Reuters 21578, SVM reaches a better macro-F_1 earlier, to be reached by PA-R-100 only toward the end of the simulation. A key aspect in the comparison of SVM and PA-based configurations is thus the computational cost.

Table 5 shows the total time required to update the classification models for the 1,000 labeling steps. The time required to update the classification models at each step in the labeling process is shown in Figure 10. Training times for PA-based configurations are significantly lower than those required by SVM. SVM requires every time to train from scratch on the whole training set and thus it has a linearly increasing training time with respect to the size of the training set. The plots clearly show how the PA-based configurations have a stable

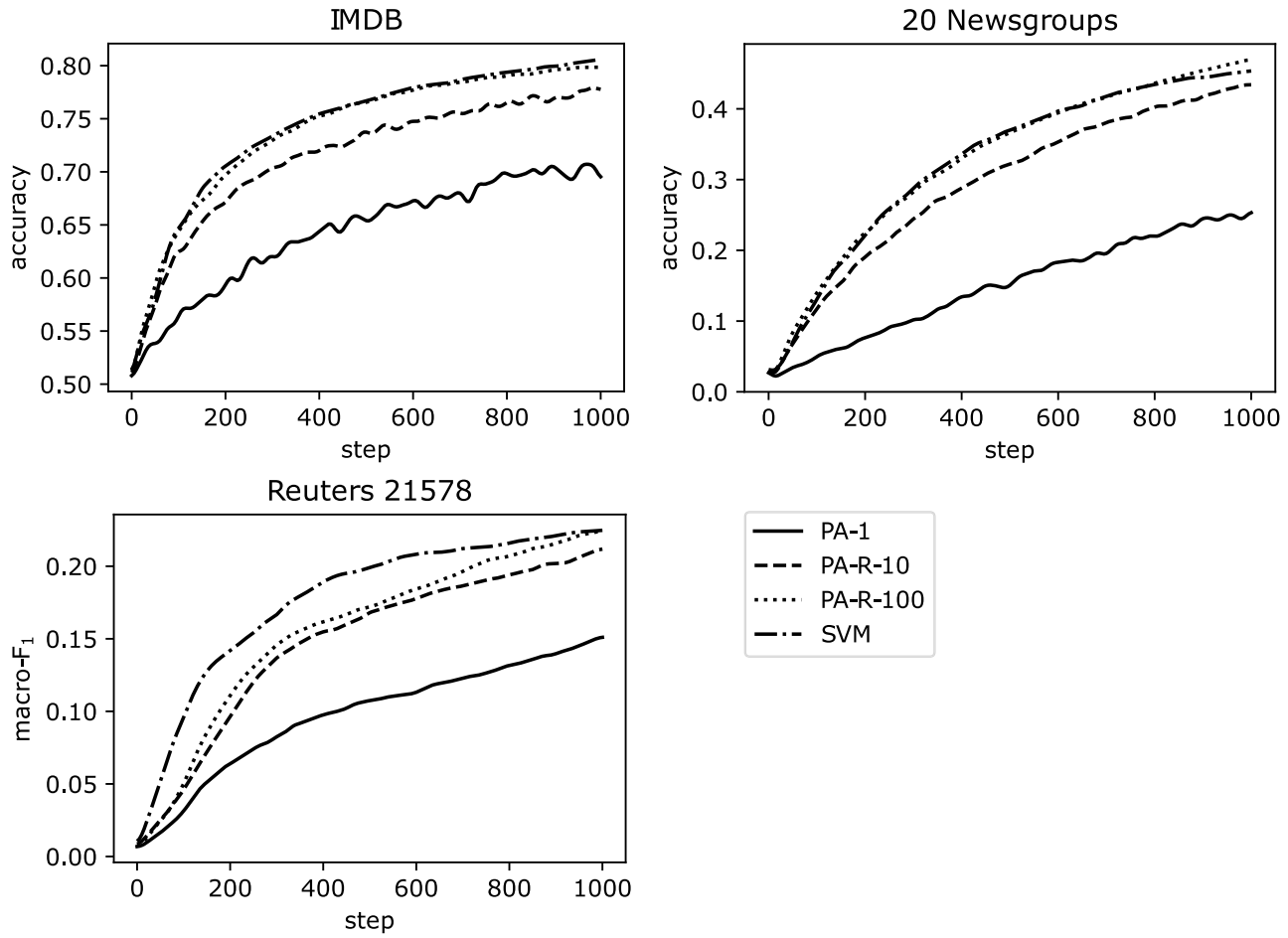

FIGURE 9. Learning curves for SVM, PA-1, and PA-R-*N* configurations using the active learning browsing mode. Simulation of 1,000 labeling steps.

update time, due to the fact that the training set has a fixed size. Larger batch sizes for PA-based configurations require more time than smaller batch sizes, yet sensibly less than SVM.

Considering that the mini-batch size can be changed at any moment, that there is no large drop accuracy/macro-F_1 moving from a mini-batch size of 100 to 10, and that instead training times drop sensibly, it is reasonable to think to implement in the system the use of a variable mini-batch size determined by the current load on the system. When there are few concurrent model update requests on the system, a high mini-batch size can be used (even larger than 100). Whenever a spike of requests generates a high load on the system, the mini-batch size can be temporarily lowered, allowing the system to remain responsive to the actions of users.

B. TRANSFER LEARNING

We have also run experiments to evaluate the performance of the system in a transfer learning/domain adaptation setup. The scenario is about facing a new classification problem when a well-performing classifier for a very similar task is available. This is the case of sentiment classification, where many expressions of evaluation, sentiment, and attitude are common to most domains (e.g., "I like it"). Each domain has also its own sentiment expressions, with a specific sentiment valence. For example, "unpredictable" typically has a positive valence when it describes the plot of a movie, and a negative valence when it describes an electronic appliance. In this scenario, instead of creating a new sentiment classifier for product reviews, a user can create a copy of an already trained classifier for movie reviews. The initial performance of the copied classifier on the new domain will be not perfect but better than a new random classifier. As more training examples are produced by users, the copied classifier will achieve good accuracy much faster (i.e., requiring fewer human-labeled examples) than one created anew.

TABLE 5. Cumulative time (in seconds) required by the various configurations to update the machine learning models in the 1,000 steps of the labeling simulation.

Learner	IMDB	20 Newsgroups	Reuters 21578
SVM	100.48	284.28	5714.55
PA-1	5.70	7.47	52.42
PA-L-10	7.18	11.20	87.66
PA-R-10	7.11	11.81	85.24
PA-L-100	22.18	44.00	482.18
PA-R-100	21.69	50.48	415.67

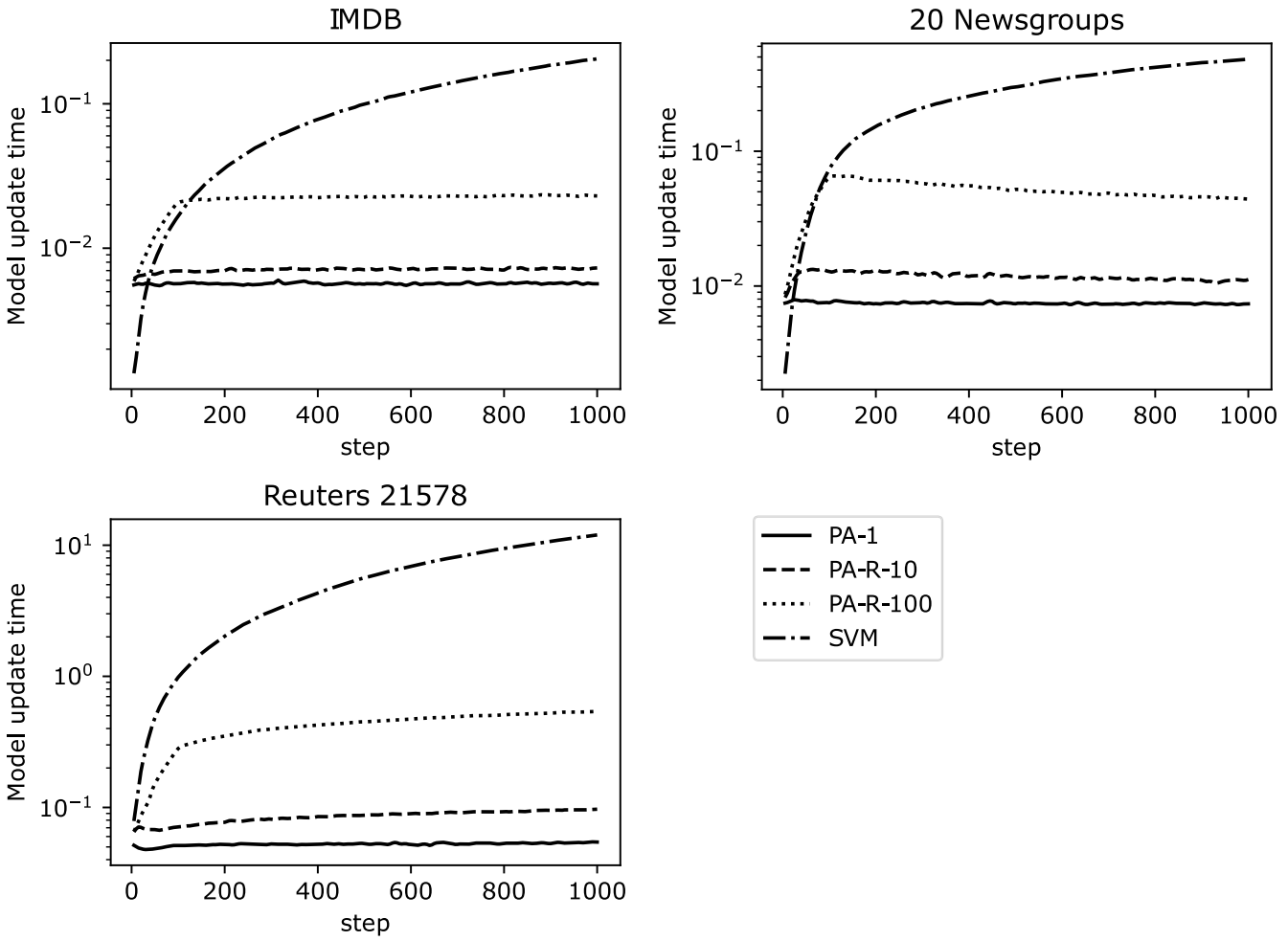

FIGURE 10. Time (seconds, logarithmic scale) required to update the classification models for SVM, PA-1, and PA-R-*N* configurations using the active learning browsing mode. Simulation of 1,000 labeling steps.

1) CONFIGURATION

The datasets used in these experiments are the IMDB dataset already described and the Fine Foods dataset [56]. The Fine Foods dataset consists of 568,454 Amazon reviews for food products. The star rating associated with each review is used to derive a sentiment label. The label is positive if the review has four or five stars, and negative if it has three stars or less. For our experiments, we extract from the original Fine Foods dataset a subset of 50,000 reviews, with a perfect balance between positive and negative labels, similar to the composition of the IMDB dataset.

The experiments consist of simulating a user labeling 1,000 documents of a "target" dataset (i.e., one among IMDB and Fine Foods) starting from two initial conditions:

- New(target) - A new classifier with labels Positive and Negative is created just before the first label is assigned to documents in the target dataset.
- Copy(source, target) - A copy of a classifier created simulating 1,000 labeling steps on a "source" dataset (different yet similar to the target dataset) is created just before the first label is assigned to documents in the target dataset.

The Copy setup thus models the transfer learning case, in which there exists a classifier for a similar domain/task to the one faced on the target dataset. We tested both datasets in both roles of source and target datasets, i.e., Copy(IMDB, Fine Foods) and Copy(Fine Foods,IMDB).

As the learning algorithms, we tested the SVM and PA-R-100 algorithms, both using the Active browsing mode. The SVM algorithm cannot be used in the Copy learning mode, as it does not support incremental learning. We ran each experimental setting 10 times with random initialization and averaged the results. For every run, we randomly sampled a new subset from the Fine Foods dataset.

2) RESULTS

Figure 11 shows the learning curves for all the tested configurations. Table 6 shows the accuracy values at various steps. The New configurations start from an accuracy of 0.5, which is the expected accuracy of a random binary classifier on a balanced dataset. The Copy configurations start from a much higher accuracy, over 65%, confirming that the classifier trained on the source dataset models a classification problem that is close to the one of the target dataset. It is interesting

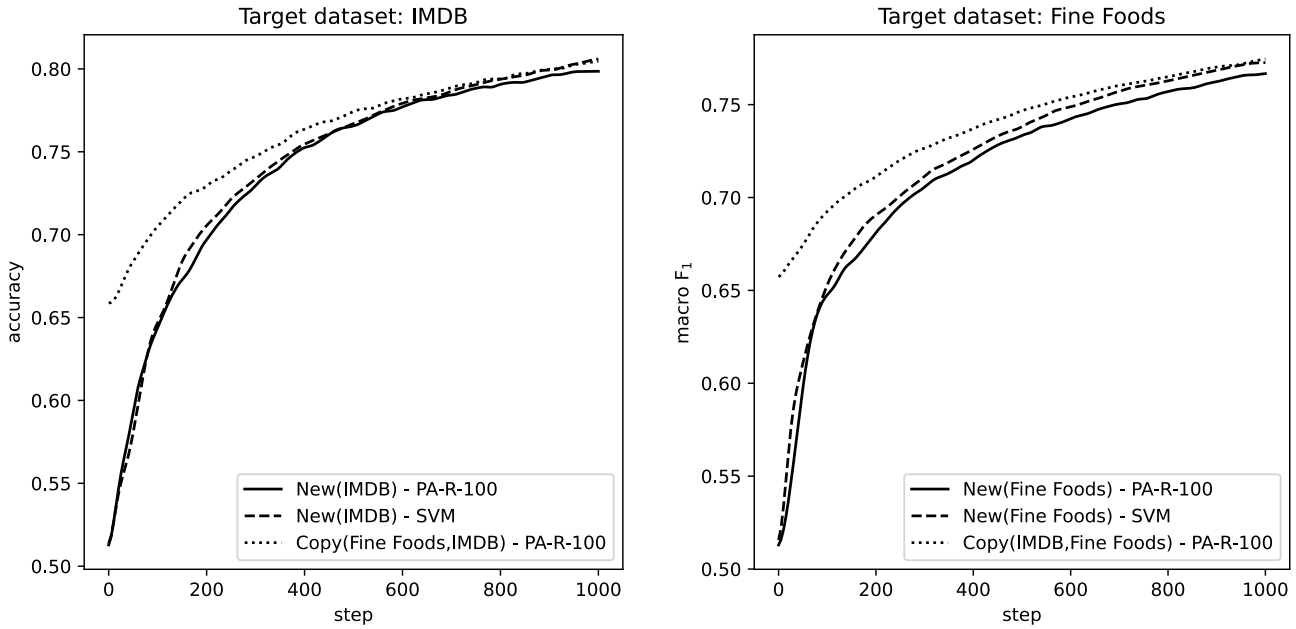

FIGURE 11. Learning curves for the transfer learning experiments on IMDB and Fine Foods datasets. Simulation of 1,000 labeling steps using Active browsing.

TABLE 6. Comparison of accuracy for various number of labeling steps for transfer learning experiments. FF=Fine Foods.

		Step				
		10	50	100	500	1000
Configuration	Algorithm	Accuracy				
New(FF)	SVM	0.531	0.616	0.657	0.741	0.773
New(FF)	PA-R-100	0.525	0.604	0.639	0.733	0.767
Copy(IMDB,FF)	PA-R-100	0.655	0.680	0.695	0.747	0.774
New(IMDB)	SVM	0.513	0.594	0.645	0.769	0.805
New(IMDB)	PA-R-100	0.511	0.584	0.637	0.761	0.800
Copy(FF,IMDB)	PA-R-100	0.673	0.689	0.706	0.774	0.804

to see that the New/PA-R-100 configuration seems not able to reach the accuracy of the Copy/PA-R-100 configuration. The New configuration that uses SVM tends to reach the accuracy of the Copy/PA-R-100 configuration in the long run. Yet, the Copy/PA-R-100 has better performance at any step. Note that the computational cost of Copy/PA-R-100 and New/PA-R-100 is identical. Thus, whenever possible, Copy is preferable for the obvious advantage of resulting in higher accuracy, but also for the not so evident advantage of getting from the beginning better suggestions, which should result in quicker labeling decisions by users.

VI. CONCLUSION

We presented ICS, a web-based application that supports the activity of manual text classification. ICS has been designed and implemented to give its users total freedom of action. This is an innovation with respect to the typical approach of machine learning research applied to text, which focuses on the algorithms, and assigns the human actors to constrained roles within the workflow of the algorithm.

Online learning methods, especially when coupled with active learning, do give some freedom to their users. They also bring in the advantage of having usable models for prediction since the early steps of training set construction. Yet, they still do not cover the additional freedom of action we require for our system, e.g., adding/removing labels, and merging existing models to define new, different models that leverage the already acquired supervised information. To implement such solutions, we combined the flexibility of online learning methods with additional theoretical and technological tools, such as feature hashing, random indexing, and asynchronous processing. The resulting system satisfies our requirements, and also shows new avenues for the development of classification systems.

A typical barrier to access to machine learning is the need for training data. The perceived cost of labeling a training set without any hint of the potential quality of the resulting automatic classifier until the training set labeling work is done can be a psychological barrier that hinders the adoption of machine learning methods. A counterargument to this observation is that if some labeling work must be done, the eventual switch to machine learning has no impact on the cost of the manual labeling activity, which must be done regardless. Yet, drawing from personal experience, once some relevant manual labeling work is done, it is not uncommon to find opposition to the use of machine learning with the motivation "we have been able to do this work up to now without machine learning, we can continue without it". This is the manifestation of a bias that confers a higher cost to the manual labeling already done, and a lower cost to the one to be still done. In this way, the perceived cost-benefit trade-off is shifted against the adoption of machine learning.

By shifting the focus from the machine learning algorithm to the user, the machine learning component assumes the role of a virtual collaborator who, from the earliest stages

of the labeling activity, supports the user by giving labeling suggestions of increasing quality as the labeling proceeds. In this setup, the user perceives a contribution of the machine learning component to the labeling effort since the beginning, contrasting the bias in the estimation of the cost-benefit trade-off. This process, in addition to actually easing the user's work, and producing better automatic models (as shown in Section V-A), also helps the user to build trust in the automatic classifier, assess the quality of the automatic model, and decide when to switch to fully automatic classification.

REFERENCES

[1] M. E. Maron, "Automatic indexing: An experimental inquiry," *J. ACM*, vol. 8, no. 3, pp. 404–417, Jul. 1961, doi: 10.1145/321075.321084.

[2] C. C. Aggarwal and C. Zhai, *A Survey of Text Classification Algorithms*. Heidelberg, Germany: Springer, 2012, pp. 163–222, doi: 10.1007/978-1-4614-3223-4_6.

[3] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, 2002, doi: 10.1145/505282.505283.

[4] B. Settles, "Active learning literature survey," Dept. Comput. Sci., Univ. Wisconsin-Madison, Wisconsin, WI, USA, Tech. Rep. TR1648, 2009. [Online]. Available: http://digital.library.wisc.edu/1793/60660

[5] A. Esuli, "Interactive classification system," Zenodo, 2022, doi: 10.5281/zenodo.6586244.

[6] D. W. Oard, "The state of the art in text filtering," *User Model. User-Adapted Interact.*, vol. 7, no. 3, pp. 141–178, Jun. 1997.

[7] R. E. Schapire, Y. Singer, and A. Singhal, "Boosting and Rocchio applied to text filtering," in *Proc. 21st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, vol. 98, Aug. 1998, pp. 215–223, doi: 10.1145/290941.290996.

[8] I. Soboroff and C. K. Nicholas, "Combining content and collaboration in text filtering," in *Proc. Workshop Mach. Learn. Inf. Filtering (IJCAI)*, 1999, pp. 86–91. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.8019

[9] S. Wermter, "Neural network agents for learning semantic text classification," *Inf. Retr.*, vol. 3, no. 2, pp. 87–103, 2000, doi: 10.1023/A:1009942513170.

[10] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proc. Eur. Conf. Mach. Learn.*, 1998, pp. 137–142, doi: 10.1007/BFb0026683.

[11] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *Proc. 22nd Annu. Int. ACM Conf. Res. Develop. Inf. Retr. (SIGIR)*, 1999, pp. 42–49, doi: 10.1145/312624.312647.

[12] R. Johnson and T. Zhang, "Semi-supervised convolutional neural networks for text categorization via region embedding," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2015, pp. 919–927. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/acc3e0404646c57502b480dc052c4fe1-Abstract.html

[13] A. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," in *Proc. AAAI Workshop Learn. Text Categorization*, vol. 752, no. 1, 1998, pp. 41–48.

[14] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using EM," *Mach. Learn.*, vol. 39, nos. 2–3, pp. 103–134, 2000, doi: 10.1023/A:1007692713085.

[15] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics*, vol. 2, 2017, pp. 427–431. [Online]. Available: https://aclanthology.org/E17-2068

[16] D. J. Hruschka, D. Schwartz, D. C. S. John, E. Picone-Decaro, R. A. Jenkins, and J. W. Carey, "Reliability in coding open-ended data: Lessons learned from HIV behavioral research," *Field Methods*, vol. 16, no. 3, pp. 307–331, Aug. 2004, doi: 10.1177/1525822X04266540.

[17] M. Cope, "Coding transcripts and diaries," *Key Methods Geography*, vol. 440, pp. 440–452, Jan. 2010.

[18] Y. Fu, X. Zhu, and B. Li, "A survey on instance selection for active learning," *Knowl. Inf. Syst.*, vol. 35, no. 2, pp. 249–283, May 2013, doi: 10.1007/s10115-012-0507-8.

[19] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop," 2015, *arXiv:1506.03365*.

[20] B. Settles, "From theories to queries: Active learning in practice," in *Proc. Act. Learn. Exp. Des. Workshop Conjunct. (AISTATS PMLR)*, 2011, pp. 1–18. [Online]. Available: https://proceedings.mlr.press/v16/settles11a.html

[21] S. C. Hoi, R. Jin, J. Zhu, and M. R. Lyu, "Batch mode active learning and its application to medical image classification," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 417–424, doi: 10.1145/1143844.1143897.

[22] Y. Guo and D. Schuurmans, "Discriminative batch mode active learning," in *Proc. 20th Int. Conf. Neural Inf. Process. Syst.*, vol. 20, 2007, pp. 593–600. [Online]. Available: https://proceedings.neurips.cc/paper/2007/file/ccc0aa1b81bf81e16c676ddb977c5881-Paper.pdf

[23] Y. Chen and A. Krause, "Near-optimal batch mode active learning and adaptive submodular optimization," in *Proc. ICML*, vol. 28, 2013, pp. 160–168. [Online]. Available: https://proceedings.mlr.press/v28/chen13b.html

[24] E. Clark, A. S. Ross, C. Tan, Y. Ji, and N. A. Smith, "Creative writing with a machine in the loop: Case studies on slogans and stories," in *Proc. 23rd Int. Conf. Intell. Interfaces (IUI)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 329–340, doi: 10.1145/3172944.3172983.

[25] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2009, doi: 10.1109/TKDE.2009.191.

[26] A. M. Fernández, A. Esuli, and F. Sebastiani, "Distributional correspondence indexing for cross-lingual and cross-domain sentiment classification," *J. Artif. Int. Res.*, vol. 55, no. 1, pp. 131–163, Jan. 2016, doi: 10.1613/jair.4762.

[27] R. Kawase, M. Fisichella, B. P. Nunes, K.-H. Ha, and M. Bick, "Automatic classification of documents in cold-start scenarios," in *Proc. 3rd Int. Conf. Web Intell., Mining Semantics*, New York, NY, USA: ACM, 2013, pp. 19:1–19:10, doi: 10.1145/2479787.2479789.

[28] A. Gliozzo, C. Strapparava, and I. Dagan, "Investigating unsupervised learning for text categorization bootstrapping," in *Proc. Conf. Hum. Lang. Technol. Empirical Methods Natural Lang. Process. (HLT)*, 2005, pp. 129–136, doi: 10.3115/1220575.1220592.

[29] L. Barak, I. Dagan, and E. Shnarch, "Text categorization from category name via lexical reference," in *Proc. Hum. Lang. Technol., Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Companion Volume, Short Papers*, Boulder, CO, USA: Association for Computational Linguistics, Jun. 2009, pp. 33–36. [Online]. Available: https://aclanthology.org/N09-2009

[30] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining Text Data*. Cham, Switzerland: Springer, 2012, pp. 77–128, doi: 10.1007/978-1-4614-3223-4_4.

[31] C. Wang, Y. Song, D. Roth, M. Zhang, and J. Han, "World knowledge as indirect supervision for document clustering," in *Proc. ACM Trans. Knowl. Discov. Data*, Dec. 2016, vol. 11, no. 2, pp. 13:1–13:36, doi: 10.1145/2953881.

[32] H. Cunningham, "GATE a general architecture for text engineering," *Comput. Humanities*, vol. 36, no. 2, pp. 223–254, 2002, doi: 10.1023/A:1014348124664.

[33] D. D. Maynard, D. K. Bontcheva, and D. H. Cunningham, "Towards a semantic extraction of named entities," Dept. Comput. Sci., Univ. Sheffield, Sheffield, U.K., Tech. Rep. S1 4DP, 2003. [Online]. Available: https://gate.ac.U.K./sale/ranlp03/maynard.pdf

[34] D. Maynard, I. Roberts, M. A. Greenwood, D. Rout, and K. Bontcheva, "A framework for real-time semantic social media analysis," *J. Web Semantics*, vol. 44, pp. 75–88, May 2017, doi: 10.1016/j.websem.2017.05.002.

[35] J.-C. Klie, M. Bugert, B. Boullosa, R. E. de Castilho, and I. Gurevych, "The inception platform: Machine-assisted and knowledge-oriented interactive annotation," in *Proc. 27th Int. Conf. Comput. Linguistics, Syst. Demonstrations*, Santa Fe, NM, USA: Association for Computational Linguistics, Jun. 2018, pp. 5–9. [Online]. Available: https://aclanthology.org/C18-2002

[36] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "BRAT: A web-based tool for NLP-assisted text annotation," in *Proc. Demonstrations 13th Conf. Eur. Chapter Assoc. Comput. Linguistics*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 102–107. [Online]. Available: https://aclanthology.org/E12-2021.pdf

[37] T. Macer, M. Pearson, and F. Sebastiani, "Cracking the code: What customers say, in their own words," in *Proc. 50th Annu. Conf. Market Res. Soc. (MRS)*, Brighton, U.K., 2007. [Online]. Available: https://meaning.U.K.com/resources/articles_papers/files/macer-pearson-sebastiani-2007.pdf
[38] A. Esuli and F. Sebastiani, "Machines that learn how to code open-ended survey data," *Int. J. Market Res.*, vol. 52, no. 6, pp. 775–800, Nov. 2010, doi: 10.2501/S147078531020165X.
[39] A. Esuli and F. Sebastiani, "Training data cleaning for text classification," in *Proc. Conf. Theory Inf. Retr.*, 2009, pp. 29–41, doi: 10.1007/978-3-642-04417-5_4.
[40] A. Esuli and F. Sebastiani, "Active learning strategies for multi-label text classification," in *Proc. 31st Eur. Conf. Inf. Retr. (ECIR)*, Toulouse, France, 2009, pp. 102–113, doi: 10.1007/978-3-642-00958-7_12.
[41] G. Paolacci, J. Chandler, and P. G. Ipeirotis, "Running experiments on Amazon mechanical Turk," *Judgment Decis. Making*, vol. 5, no. 5, pp. 411–419, 2010. [Online]. Available: https://ssrn.com/abstract=1626226
[42] L. Tan. (2020). *A Survey of NLP Annotation Platforms*. [Online]. Available: https://github.com/alvations/annotate-questionnaire
[43] A. Moreo, A. Esuli, and F. Sebastiani, "Building automated survey coders via interactive machine learning," *Int. J. Market Res.*, vol. 61, no. 4, pp. 408–429, Jul. 2019, doi: 10.1177/1470785318824244.
[44] P. Kanerva, J. Kristoferson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proc. Annu. Meeting Cognit. Sci. Soc.*, 2000, vol. 22, no. 22, pp. 1–2. [Online]. Available: https://escholarship.org/uc/item/5644k0w6
[45] M. Sahlgren, "An introduction to random indexing," in *Proc. Methods Appl. Semantic Indexing Workshop, 7th Int. Conf. Terminol. Knowl. Eng.*, 2005, pp. 1–9. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1041127/FULLTEXT01.pdf
[46] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990, doi: 10.1002/(SICI)1097-4571(199009)41:6%3C391::AID-ASI1%3E3.0.CO;2-9.
[47] A. M. Fernández, A. Esuli, and F. Sebastiani, "Lightweight random indexing for polylingual text classification," *J. Artif. Intell. Res.*, vol. 57, pp. 151–185, Oct. 2016, doi: 10.1613/jair.5194.
[48] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1113–1120, doi: 10.1145/1553374.1553516.
[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Jan. 2011. [Online]. Available: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf
[50] A. Appleby. (2016). *Murmurhash3, 2016*. [Online]. Available: https://github.com/aappleby/smhasher/wiki/MurmurHash3
[51] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *J. Mach. Learn. Res.*, vol. 7, pp. 551–585, Dec. 2006. [Online]. Available: https://www.jmlr.org/papers/volume7/crammer06a/crammer06a.pdf
[52] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meet. Assoc. Comput. Linguistics, Hum. Lang. Technol.*, Portland, OR, USA, Jun. 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015
[53] K. Lang, "Newsweeder: Learning to filter netnews," in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 331–339.
[54] D. Lewis. (1997). *Reuters-21578 Text Categorization Test Collection, Distribution 1.0*. [Online]. Available: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html
[55] A. Moreo, A. Esuli, and F. Sebastiani, "Distributional random oversampling for imbalanced text classification," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jul. 2016, pp. 805–808, doi: 10.1145/2911451.2914722.
[56] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews," in *Proc. 22nd Int. Conf. World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 897–908, doi: 10.1145/2488388.2488466.

ANDREA ESULI received the M.Sc. and Ph.D. degrees from the University of Pisa, in 2001 and 2008, respectively.

He is currently a Senior Researcher with the Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo," which is a part of the Italian National Research Council (CNR). His research interests include machine learning and information retrieval.

Dr. Esuli won the European "Cor Baayen Award" as a Promising Young Researcher in computer science and applied mathematics, in 2010.