QCRI at SemEval-2016 Task 4: Probabilistic Methods for Binary and Ordinal Quantification

Giovanni Da San Martino, Wei Gao, Fabrizio Sebastiani*
Qatar Computing Research Institute
Hamad bin Khalifa University
PO Box 5825, Doha, Qatar
Email: {gmartino,wgao,fsebastiani}@qf.org.qa

Abstract

We describe the systems we have used for participating in Subtasks D (binary quantification) and E (ordinal quantification) of SemEval-2016 Task 4 "Sentiment Analysis in Twitter". The binary quantification system uses a "Probabilistic Classify and Count" (PCC) approach that leverages the calibrated probabilities obtained from the output of an SVM. The ordinal quantification approach uses an ordinal tree of PCC binary quantifiers, where the tree is generated via a splitting criterion that minimizes the ordinal quantification loss.

1 Introduction

This document describes the systems we have used for participating in Subtasks D (binary quantification) and E (ordinal quantification) of SemEval-2016 Task 4 "Sentiment Analysis in Twitter". In the runs we have submitted no training data was used other than the officially provided ones (indeed, the only "external" data used were the sentiment lexicons mentioned in Section 2).

Like a classification system, a system for performing quantification consists of two main components: (i) an algorithm for converting the objects of interest (tweets, in our case) into vectorial representations that can be interpreted both by the learning algorithm and, once it has been trained, by the quantifier itself, and (ii) an algorithm for training quantifiers from vectorial representations of training objects. Section 2 will be devoted to discussing component (i), while Sections 3 and 4 will be devoted to discussing the two learning algorithms we have deployed for the two tasks.

2 Features for detecting tweet sentiment

As in (Gao and Sebastiani, 2015; Gao and Sebastiani, 2016), for building vectorial representations of tweets we have followed the approach discussed in (Kiritchenko et al., 2014, Section 5.2.1), since the representations presented therein are those used in the systems that performed best at both the SemEval 2013 (Mohammad et al., 2013) and SemEval 2014 (Zhu et al., 2014) tweet sentiment classification shared tasks.

The text is preprocessed by normalizing URLs and mentions of users to the constants `http://someurl` and `@someuser`, resp., after which tokenisation and POS tagging is performed. The binary features used (i.e., features denoting presence or absence in the tweet) include word n-grams, for $n \in \{1, 2, 3, 4\}$, and character n-grams, for $n \in \{3, 4, 5\}$, whether the last token contains an exclamation and/or a question mark, whether the last token is a positive or a negative emoticon and, for each of the 1000 word clusters produced with the CMU Twitter NLP tool[1], whether any token from the cluster is present. Integer-valued features include the number of all-caps tokens, the number of tokens for each POS tag, the number of hashtags, the number of negated contexts, the number of sequences of exclamation and/or question marks, and the number of elongated words (e.g., `cooooool`).

*Fabrizio Sebastiani is currently on leave from Consiglio Nazionale delle Ricerche, Italy.

[1] `http://www.ark.cs.cmu.edu/TweetNLP/`

Proceedings of SemEval-2016, pages 58–63,
San Diego, California, June 16-17, 2016.

A key addition to the above is represented by features derived from both automatically generated and manually generated sentiment lexicons; for these features, we use the same sentiment lexicons as used in (Kiritchenko et al., 2014), which are all publicly available. We omit further details concerning our vectorial representations (and, in particular, how the sentiment lexicons contribute to them), both for brevity reasons and because these vectorial representations are not the central focus of this paper; the interested reader is invited to consult (Kiritchenko et al., 2014, Section 5.2.1) for details.

3 Subtask D: Tweet quantification according to a two-point scale

For the binary quantification task, we have performed a thorough set of preliminary experiments using 8 quantification methods from the literature, i.e., Classify and Count (CC), Probabilistic Classify and Count (PCC) (Bella et al., 2010), Adjusted Classify and Count (ACC) (Gart and Buck, 1966), Probabilistic Adjusted Classify and Count (PACC) (Bella et al., 2010), Expectation Maximization for Quantification (EMQ) (Saerens et al., 2002), SVMs optimized for KLD (SVM(KLD)) (Esuli and Sebastiani, 2015), SVMs optimized for $NKLD$ (SVM(NKLD)) (Esuli and Sebastiani, 2014), and SVMs optimized for Q (SVM(Q)) (Barranquero et al., 2015).

All 8 methods are described in detail in (Gao and Sebastiani, 2016), where we test them on a ternary[2] tweet sentiment quantification task using 11 datasets and 6 evaluation measures. The aim of (Gao and Sebastiani, 2016) was to test whether the conclusions drawn from a previous experiment (Esuli and Sebastiani, 2015), where quantification was according to topic and where texts were significantly longer than tweets, were confirmed also in a context in which quantification is according to sentiment and the items are significantly shorter.

The preliminary experiments we performed for the present work were carried out by training our models on TRAIN+DEV and testing on DEVTEST. For the first 5 methods mentioned at the beginning of this section we make use of a standard SVM with a linear kernel, in the implementation made available in the LIBSVM system[3] (Chang and Lin, 2011). For the other 3 methods we make use of an SVM for structured output prediction, in the implementation made available in the SVM-perf system[4] (Joachims, 2005). For all 8 methods we optimize the C parameter (which sets the tradeoff between the training error and the margin) directly on DEVTEST by performing a grid search on all values of type 10^x with $x \in \{-6, ..., 7\}$; we instead leave the other parameters at their default value. The PCC, PACC, EMQ methods require the classifier to also generate posterior probabilities; since SVMs do not natively generate posterior probabilities, for these three methods we use the `-b` option of LIBSVM, which converts the scores originally generated by SVMs into posterior probabilities according to the algorithm of (Wu et al., 2004).

The results of these preliminary experiments, which are reported in Table 1, indicated PCC as the best performer. These experiments by and large confirmed the results of (Gao and Sebastiani, 2016), where PCC was the best performer for 34 of the 66 combinations of 11 datasets $\times$ 6 evaluation measures. Instead, for none of the 66 combinations SVM(KLD), which had been the best performer in the experiments of (Esuli and Sebastiani, 2015) (where it also outperformed PCC), was the best performer. In (Gao and Sebastiani, 2016) we conjectured that this difference may be due to the fact that in quantification by sentiment, class prevalences tend to be fairly high ($> .10$), and that the experiments of (Esuli and Sebastiani, 2015) mostly concerned classes with low prevalence ($< .10$) or very low prevalence ($< .01$), which tend to be the norm in classification by topic.

As a result of all this, in this work we decided to use PCC; Section 3.1 describes the PCC method in detail.

[2] Differently from the present task, the datasets we used in (Gao and Sebastiani, 2016) also used the Neutral class.

[3] LIBSVM is available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[4] SVM-perf is available from `http://svmlight.joachims.org/svm_struct.html` . The modules that customize it to KLD and $NKLD$ were made available by Andrea Esuli, while the module that customizes it to Q was made available by José Barranquero.

Table 1: Results of our preliminary experiments for 8 quantification methods. The 2nd column indicates accuracy as measured via KLD (lower values are better), while the 3rd column indicates the value of the C parameter used, and which had proved optimal when training on TRAIN+DEV and testing on DEVTEST; **boldface** indicates the best performer.

	KLD	C
CC	0.0528275	0.00001
PCC	**0.0278864**	0.000001
ACC	0.0844489	0.0001
PACC	0.0509028	0.0001
EMQ	0.2215310	0.001
SVM(KLD)	0.0356832	0.001
SVM(NKLD)	0.1415660	0.001
SVM(Q)	0.0989133	0.1

3.1 Probabilistic Classify and Count (PCC)

The PCC method, originally introduced in (Bella et al., 2010), consists in generating a classifier from Tr, classifying the objects in Te, and computing $p_{Te}(c)$ as the *expected* fraction of objects predicted to belong to c. If by $p(c|\mathbf{x})$ we indicate the posterior probability, i.e., the probability of membership in c of test object $\mathbf{x}$ as estimated by the classifier, and by $E[x]$ we indicate the expected value of x, this corresponds to computing

$$\begin{aligned}\hat{p}_{Te}^{PCC}(c) &= E[p_{Te}(\hat{c})] \\ &= \frac{1}{|Te|}\sum_{\mathbf{x}\in Te} p(c|\mathbf{x})\end{aligned} \quad (1)$$

where $\hat{p}_S^M(c)$ indicates the prevalence of class c in set S as estimated via method M (the "hat" symbol indicates estimation). The rationale of PCC is that posterior probabilities contain richer information than binary decisions, which are usually obtained from posterior probabilities by thresholding.

For our final run, we have retrained the system on TRAIN+DEV+DEVTEST, using the parameter values which had performed best on DEVTEST in the preliminary experiments. On the official test set (Nakov et al., 2016) we obtained a KLD score of 0.055, and thus ranked 5th in a set of 14 participating teams.

4 Subtask E: Tweet quantification according to a five-point scale

Our goal in tackling the ordinal quantification task has been to devise a new learning algorithm for ordinal quantification. We decided to aim for an algorithm that (a) leverages the information inherent in the class ordering, and (b) performs quantification according to the *Probabilistic Classify and Count* (PCC) method ((Bella et al., 2010) – see also (Gao and Sebastiani, 2016, §4.2)), since this has proven the best-performing method in the tweet quantification experiments of (Gao and Sebastiani, 2016).

Ordinal quantification will be tackled by arranging the classes in the totally ordered set $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$ into a binary tree. Given any $j \in \{1, \ldots, (|\mathcal{C}|-1)\}$, $\underline{\mathcal{C}}_j = \{c_1, \ldots, c_j\}$ will be called a *prefix* of $\mathcal{C}$, and $\overline{\mathcal{C}}_j = \{c_{j+1}, \ldots, c_{|\mathcal{C}|}\}$ will be called a *suffix* of $\mathcal{C}$. Given any $j \in \{1, \ldots, (|\mathcal{C}| - 1)\}$ and a set S of items labeled according to $\mathcal{C}$, by $\underline{S}_j$ we denote the set of items in S whose class is in $\underline{\mathcal{C}}_j$, and by $\overline{S}_j$ we denote the set of items in S whose class is in $\overline{\mathcal{C}}_j$.

4.1 Generating a quantification tree

The algorithm for training a quantification tree is described in concise form as Algorithm 1, and goes as follows. Assume we have a training set Tr and a held-out validation set Va of items labelled according to $\mathcal{C}$.

The first step (Line 3) consists in training $(|\mathcal{C}|-1)$ binary classifiers h_j, for $j \in \{1, \ldots, (|\mathcal{C}| - 1)\}$. Each of these classifiers must discriminate between $\underline{\mathcal{C}}_j$ and $\overline{\mathcal{C}}_j$; for training h_j we will take the items in $\underline{Tr}_j$ as the negative training examples and the items in $\overline{Tr}_j$ as the positive training examples. We require that these classifiers, aside from taking binary decisions (i.e., predicting if a test item is in $\underline{\mathcal{C}}_j$ or in $\overline{\mathcal{C}}_j$), also output posterior probabilities, i.e., probabilities $p(\underline{\mathcal{C}}_j|\mathbf{x})$ and $p(\overline{\mathcal{C}}_j|\mathbf{x}) = (1 - p(\underline{\mathcal{C}}_j|\mathbf{x}))$, where $p(c|\mathbf{x})$ indicates the probability of membership in c of test object $\mathbf{x}$ as estimated by the classifier[5].

The second step (Line 5) is building the ordinal quantification (binary) tree. In order to do this,

[5] If the classifier only returns confidence scores that are not probabilities (as is the case with many non-probabilistic classifiers), the former must be converted into true probabilities. If the score is a monotonically increasing function of the classifier's confidence in the fact that the object belongs to the class, the conversion may be obtained by applying a logistic function. *Well-calibrated* probabilities (defined as the probabilities such that the prevalence $p_S(c)$ of a class c in a set S is equal to $\sum_{\mathbf{x}\in S} p(c|\mathbf{x})$) may be obtained by using a *generalized* logistic function; see e.g., (Berardi et al., 2015, Section 4.4) for details.

1 **Function** GenerateTree ($\mathcal{C}, Tr, Va$);
/* Generates the quantification tree */
Input : Ordered set $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$;
Set Tr of labelled items;
Set Va of labelled items;
Output: Quantification tree $T_{\mathcal{C}}$.
2 **for** $j \in \{1, ..., (|\mathcal{C}| - 1)\}$ **do**
3 Train classifier h_j from $\underline{Tr}_j$ and $\overline{Tr}_j$;
4 **end**
5 $T_{\mathcal{C}} \leftarrow$ Tree($\mathcal{C}, \{h_j\}, Va$);
6 **return** $T_{\mathcal{C}}$;

7 **Function** Tree ($\mathcal{C}, \mathcal{H}_{\mathcal{C}}, Va$);
/* Recursive subroutine for generating the quantification tree */
Input : Ordered set $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$ of classes;
Set of classifiers $\mathcal{H}_{\mathcal{C}} = \{h_1, ..., h_{(|\mathcal{C}|-1)}\}$;
Output: Quantification tree $T_{\mathcal{C}}$.
8 **if** $\mathcal{C} = \{c\}$ **then**
9 Generate a leaf node T_c;
10 **return** T_c;
11 **else**
12 $h_t \leftarrow \arg\min_{h_j \in \mathcal{H}_{\mathcal{C}}} KLD(p, \hat{p}, h_j, Va)$;
13 Generate a node $T_{\mathcal{C}}$ and associate h_t to it;
14 $\mathcal{H}'_{\mathcal{C}} \leftarrow \{h_1, ..., h_{(t-1)}\}$;
15 $\mathcal{H}''_{\mathcal{C}} \leftarrow \{h_{(t+1)}, ..., h_{(|\mathcal{C}|-1)}\}$;
16 $LChild(T_{\mathcal{C}}) \leftarrow$ Tree($\underline{\mathcal{C}}_t, \mathcal{H}'_{\mathcal{C}}, Va$);
17 $RChild(T_{\mathcal{C}}) \leftarrow$ Tree($\overline{\mathcal{C}}_t, \mathcal{H}''_{\mathcal{C}}, Va$);
18 **return** $T_{\mathcal{C}}$;
19 **end**

Algorithm 1: Function GenerateTree for generating an ordinal quantification tree.

among the classifiers h_j we pick the one (let us assume it is h_t) that displays the highest quantification accuracy (Line 12) on the validation set Va, and we place it at the root of the binary tree. We then repeat the process recursively on the left and on the right branches of the binary tree (Lines 14 to 17), thus building a fully grown quantification tree. Quantification is performed according to the PCC method described in Section 3.1. We measure the quantification accuracy of classifier h_j via *Kullback-Leibler Divergence* (KLD), defined as

$$KLD(p, \hat{p}, h_j, S) = \sum_{c \in \mathcal{C}} p_S(c) \log \frac{p_S(c)}{\hat{p}_S(c)} \qquad (2)$$

where $\hat{p}$ is the distribution estimated via PCC using the posterior probabilities generated by h_j.

4.2 Estimating class prevalences via an ordinal quantification tree

The algorithm for estimating class prevalences by using an ordinal quantification tree is described in concise form as Algorithm 2, and goes as follows. Essentially, for each item $\mathbf{x} \in Te$ and for each class $c \in \mathcal{C}$, we compute (Line 6) the posterior probability $p(c|\mathbf{x})$; the estimate $\hat{p}_{Te}(c)$ is computed as the average, across all $\mathbf{x} \in Te$, of $p(c|\mathbf{x})$. The posterior probability $p(c|\mathbf{x})$ is computed in a recursive, hierarchical way (Lines 13 to 18), i.e., as the probability that the binary classifiers that lie on the path from the root to leaf c, would classify item $\mathbf{x}$ exactly in leaf c (i.e., that they would route $\mathbf{x}$ exactly to leaf c). This probability is computed as the product of the posterior probabilities returned by the classifiers that lie on the path from the root to leaf c.

An example quantification tree for a set of $|\mathcal{C}| = 6$ classes is displayed in Figure 1; for brevity, classes are represented by natural numbers, the total order defined on them is the order defined on the natural numbers, and sets of classes are represented by sequences of natural numbers. Note that, as exemplified in Figure 1, our algorithm generates trees for which (a) there is a 1-to-1 correspondence between classes and leaves of the tree, (b) leaves are ordered left to right in the same order of the classes in $\mathcal{C}$, and (c) each internal node represents a decision between a suffix and a prefix of $\mathcal{C}$.

Point (c) is interesting, and deserves some discussion. Indeed, internal node "1234 vs. 56" is trained by using items labelled as 1, or 2, or 3, or 4 as negative examples and items labelled as 5, or 6 as positive examples; however, by looking at Figure 1, it would seem intuitive that items labelled as 6 should *not* be used, since the node is root to a subtree where class 6 is not an option anymore. The reason why we do use items labelled as 6 (which is the reason the node is labelled "1234 vs. 56" and not "1234 vs. 5") is that, during the classification stage, the classifier associated with the node might be asked to classify an item whose true label is 6, and which has thus been misclassified up higher in the tree. In this case, it would be important that this item be classified as 5, since this minimizes the contribution of this item to misclassification error; and the likelihood that this happens is increased if the classifier is trained to choose

1 **Function** QuantifyViaHierarchicalPCC ($Te, T_{\mathcal{C}}$);
`/* Estimates class prevalences on` Te
`using the quantification tree */`
Input : Unlabelled set Te;
Quantification tree $T_{\mathcal{C}}$;
Output: Estimates $\hat{p}(c)$ for all $c \in \mathcal{C}$;

2 **for** $c \in \mathcal{C}$ **do**
3 $\quad \hat{p}(c) \leftarrow 0$
4 **end**
5 **for** $\mathbf{x} \in Te$ **do**
6 $\quad$ CPost($\mathbf{x}, T_{\mathcal{C}}, 1$); `/* Compute the` $\{p(c|\mathbf{x})\}$ `*/`
7 $\quad$ **for** $c \in \mathcal{C}$ **do**
8 $\quad\quad \hat{p}(c) \leftarrow \hat{p}(c) + \frac{p(c|\mathbf{x})}{|Te|}$;
9 $\quad$ **end**
10 **end**
11 **return** $\{\hat{p}(c)\}$

12 **Procedure** CPost ($\mathbf{x}, T_{\mathcal{C}}, SubP$);
`/* Recursive subroutine for computing`
$\{p(c|\mathbf{x})\}$ `all the posteriors */`
Input : Unlabelled item $\mathbf{x}$;
Quantification tree $T_{\mathcal{C}}$;
Probability $SubP$ of current subtree;
Output: Posteriors $\{p(c|\mathbf{x})\}$;

13 **if** $T_{\mathcal{C}} = \{c\}$ **then**
$\quad$ `/*` $T_{\mathcal{C}}$ `is a leaf, labelled by class` c `*/`
14 $\quad p(c|\mathbf{x}) \leftarrow SubP$;
15 **else**
16 $\quad$ CPost($\mathbf{x}, LChild(T_{\mathcal{C}}), p(\underline{\mathcal{C}}_t|\mathbf{x}) \cdot SubP$);
17 $\quad$ CPost($\mathbf{x}, RChild(T_{\mathcal{C}}), p(\overline{\mathcal{C}}_t|\mathbf{x}) \cdot SubP$);
$\quad$ `/*` $p(\underline{\mathcal{C}}_t|\mathbf{x})$ `and` $p(\overline{\mathcal{C}}_t|\mathbf{x})$ `are the posteriors returned by the classifier associated with the root of` $T_{\mathcal{C}}$ `*/`
18 **end**

Algorithm 2: Function QuantifyViaHierarchicalPCC for estimating prevalences via an ordinal quantification tree.

between 1234 and 56, rather than between 1234 and 5.

Note also that this is one aspect for which our algorithm is a true *ordinal* classification algorithm; if there were no order defined on the classes this policy would make no sense.

A second reason why our algorithm is an inherently ordinal quantification algorithm is that the groups of classes (such as 1234 and 56) between which a binary classifier needs to discriminate are

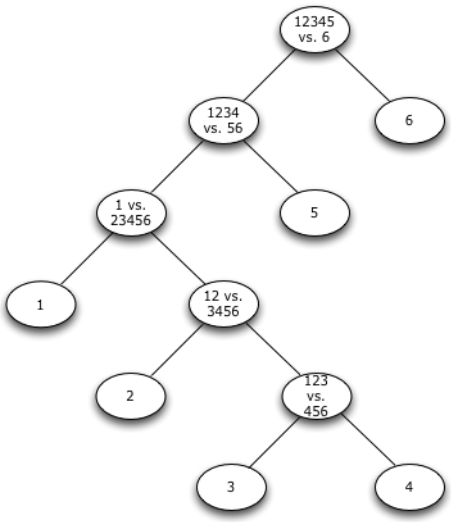

Figure 1: An example ordinal quantification tree.

groups of classes that are *contiguous* in the order defined on $\mathcal{C}$. It is because of this contiguity that the structure of the trees we generate makes sense: if, say, classes $\{1, ..., 6\}$ represent degrees of positivity of product reviews, with 1 representing most negative and 6 representing most positive, the group 56 may be taken to represent the positive reviews (to different degrees), while 1234 may be taken to represent the reviews that are not positive; a group such as, say, 256, would instead be very hard to interpret, since it is formed of non-contiguous classes that have little in common with each other.

Finally, we remark that our ordinal quantification algorithm does not depend on the fact that PCC is the chosen quantification method, and could be adapted to work with other such methods, such as e.g., SVM(KLD). Indeed, if SVM(KLD) is the chosen quantification method, in Algorithm 1 we only need to change the learning method we use (Line 3), and change the recursive subroutine CPost (Lines 12 to 17) in such a way that, by recursively making binary choices down the tree, it picks exactly one out of the $|\mathcal{C}|$ leaf classes instead of computing the posterior probabilities for all of them[6].

[6]The code that implements the PCC method for binary quantification, and our method for ordinal quantification, is available from `http://alt.qcri.org/tools/quantification/`

4.2.1 Our run

In our preliminary run over the DEVTEST set, our system obtained an EMD value of 0.210; for obtaining this, the optimization of the C parameter (see Section 3.1) was carried out using EMD as a criterion, i.e., the parameter that yielded the best EMD value on DEVTEST was chosen. For comparison, we also run on DEVTEST a baseline multiclass PCC system, i.e, one which performs quantification according to the PCC method and does not take the order on the classes into account; the baseline system, after parameter optimization, obtained an EMD value of 0.222, with a 5.64% deterioration over our system. As a result, we decided to tackle the unlabelled set with our system as described in Sections 4.1 and 4.2. Note that, unlike for Subtask D, in Subtask E we did not have a range of other datasets to perform preliminary experiments with; as a result, the only choice that could make sense here was using the system which had performed best on DEVTEST.

On the official test set (Nakov et al., 2016) we obtained an EMD score of 0.243, ranking 1st in a set of 10 participating systems, with a high margin over the other ones (systems from rank 2 to rank 8 obtained EMD scores between 0.316 and 0.366).

References

José Barranquero, Jorge Díez, and Juan José del Coz. 2015. Quantification-oriented learning based on reliable classifiers. *Pattern Recognition*, 48(2):591–604.

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

Giacomo Berardi, Andrea Esuli, and Fabrizio Sebastiani. 2015. Utility-theoretic ranking for semi-automated text classification. *ACM Transactions on Knowledge Discovery from Data*, 10(1):Article 6.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):Article 27.

Andrea Esuli and Fabrizio Sebastiani. 2014. Explicit loss minimization in quantification applications (preliminary draft). Presented at the *8th International Workshop on Information Filtering and Retrieval (DART 2014)*, Pisa, IT.

Andrea Esuli and Fabrizio Sebastiani. 2015. Optimizing text quantifiers for multivariate loss functions. *ACM Transactions on Knowledge Discovery and Data*, 9(4):Article 27.

Wei Gao and Fabrizio Sebastiani. 2015. Tweet sentiment: From classification to quantification. In *Proceedings of the 7th International Conference on Advances in Social Network Analysis and Mining (ASONAM 2015)*, pages 97–104, Paris, FR.

Wei Gao and Fabrizio Sebastiani. 2016. From classification to quantification in tweet sentiment analysis. *Social Network Analysis and Mining*. Forthcoming.

John J. Gart and Alfred A. Buck. 1966. Comparison of a screening test and a reference test in epidemiologic studies: II. A probabilistic model for the comparison of diagnostic tests. *American Journal of Epidemiology*, 83(3):593–602.

Thorsten Joachims. 2005. A support vector method for multivariate performance measures. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, pages 377–384, Bonn, DE.

Svetlana Kiritchenko, Xiaodan Zhu, and Saif M. Mohammad. 2014. Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, 50:723–762.

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. NRC-Canada: Building the state-of-the-art in sentiment analysis of tweets. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013)*, pages 321–327, Atlanta, US.

Preslav Nakov, Alan Ritter, Sara Rosenthal, Fabrizio Sebastiani, and Veselin Stoyanov. 2016. SemEval-2016 Task 4: Sentiment analysis in Twitter. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, San Diego, US. Forthcoming.

Marco Saerens, Patrice Latinne, and Christine Decaestecker. 2002. Adjusting the outputs of a classifier to new a priori probabilities: A simple procedure. *Neural Computation*, 14(1):21–41.

Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. 2004. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005.

Xiaodan Zhu, Svetlana Kiritchenko, and Saif M. Mohammad. 2014. NRC-Canada-2014: Recent improvements in the sentiment analysis of tweets. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 443–447, Dublin, IE.