Knowledge-Based Systems 149 (2018) 110–123

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

An automated text categorization framework based on hyperparameter optimization

Eric S. Tellez[a,c], Daniela Moctezuma[a,b,*], Sabino Miranda-Jiménez[a,c], Mario Graff[a,c]

[a] CONACyT Consejo Nacional de Ciencia y Tecnología, Dirección de Cátedras, Insurgentes Sur 1582, Crédito Constructor, Ciudad de México 03940 México
[b] CentroGEO Centro de Investigación en Ciencias de Información Geoespacial, Circuito Tecnopolo Norte No. 117, Col. Tecnopolo Pocitos II, C.P., Aguascalientes, Ags 20313 México
[c] INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación, Circuito Tecnopolo Sur No 112, Fracc. Tecnopolo Pocitos II, Aguascalientes 20313, México

ARTICLE INFO

Article history:
Received 20 September 2017
Revised 10 January 2018
Accepted 1 March 2018
Available online 2 March 2018

Keywords:
Text classification
Hyperparameter optimization
Text modelling

ABSTRACT

A great variety of text tasks such as topic or spam identification, user profiling, and sentiment analysis can be posed as a supervised learning problem and tackled using a text classifier. A text classifier consists of several subprocesses, some of them are general enough to be applied to any supervised learning problem, whereas others are specifically designed to tackle a particular task using complex and computational expensive processes such as lemmatization, syntactic analysis, etc. Contrary to traditional approaches, we propose a minimalist and multi-propose text-classifier able to tackle tasks independently of domain and language. We named our approach μTC. Our approach is composed of several easy-to-implement text transformations, text representations, and a supervised learning algorithm. These pieces produce a competitive classifier in several challenging domains such as informally written text. We provide a detailed description of μTC along with an extensive experimental comparison with relevant state-of-the-art methods, i.e., μTC was compared on 30 different datasets. Regarding accuracy, μTC obtained the best performance in 20 datasets while achieves competitive results in the remaining ones. The compared datasets include several problems like topic and polarity classification, spam detection, user profiling and authorship attribution. Furthermore, our approach allows the usage of the technology even without an in-depth knowledge of machine learning and natural language processing.

1. Introduction

Due to the large and continuously growing volume of textual data, automated text classification methods have taken an increasing interest in the research community. Although many efforts have been proposed in this direction, it remains as an open problem. The arrival of massive data sources, like micro-blogging platforms, introduces new challenges where many of the prior techniques failed. Among the new challenges are the volume and noisy nature of the data, the shortness of texts that implies little context, the informal style is also plagued of misspellings and lexical errors, etc.

These new data sources have made popular tasks such as *sentiment analysis* and *user profiling*. The sentiment analysis problem consists in determining the polarity of a given text, which can be a global polarity (about the whole text) or about a particular subject or entity. The user profiling task consists in, given a text, predicting some facts about the author, like her/his demographic information (e.g., gender, age, language or region). Such is the importance of these problems that in the research community several international competitions have been carried out in recent years. For example, SemEval[1], TASS[2] and SENTIPOLC[3] are challenges for sentiment classifiers for Twitter data traditionally in English, Spanish, and Italian languages, respectively. PAN[4] also opens calls for author profiling systems for English, Spanish and German languages. These problems are closely related to traditional text classification applications such as *topic classification* (e.g., classifying a news-like text into sports, politics, or economy), *authorship attribution* (e.g., identifying the author of a given text) and *spam detection*.

* Corresponding author at: CentroGEO, Circuito Tecnopolo Norte s/n, Col. Hacienda Nueva, Aguascalientes, Ags 20313, Mexico.
E-mail addresses: eric.tellez@infotec.mx (E.S. Tellez), dmoctezuma@centrogeo.edu.mx (D. Moctezuma), sabino.miranda@infotec.mx (S. Miranda-Jiménez), mario.graff@infotec.mx (M. Graff).

https://doi.org/10.1016/j.knosys.2018.03.003
0950-7051/

[1] http://alt.qcri.org/semeval2017/.
[2] http://www.sepln.org/workshops/tass/2016/tass2016.php.
[3] http://www.di.unito.it/~tutreeb/sentipolc-evalita16/.
[4] http://pan.webis.de/.

Usually, each of aforementioned problems is treated in a particular way, i.e., a method is proposed to solve adequately one classification task. Traditionally, this approach cannot generalize to other related tasks, and, consequently, the methods are dependent on the problem; however, it is worth to mention that this specialization produces a lot of insight about the problem's domain. Conversely, in this contribution, we proposed a framework to create a text classifier regardless of both the domain and the language, and based only a training set of labeled examples.

The idea of creating a text classifier almost independent of the language and domain is not novel, in fact, in our previous work [1], we introduced a combinatorial framework for sentiment analysis. There, aspects of language were considered such as stopwords and tokenizers with particular attention to lexical structures for negations. Also, particularities of the domain like *emoticons* and *emojis* are considered. The presented manuscript is a generalization and formalization of our previous work; this allows us to simplify the entire framework to work independently of both the language and the particular task, and empower the use of more sophisticated text treatments whenever it is possible and necessary.

As stated above, we tackle the problem of creating text classifiers that work regardless of both the domain and the language, with nothing more than a training set to be learned. The general idea is to orchestrate a number of simple text transformations, tokenizers, a set of weighting schemes, along with a Support Vector Machine (SVM) as classifier to produce effective text classification. More detailed, we look at the problem of creating effective text classifiers as a combinatorial optimization problem; where there is a search space containing all possible combinations of different text transformations, tokenizers, and weighting procedures with their respective parameters, and, on this search space, a meta-heuristic is used to search for a configuration that produces a highly effective text classifier. This model selection procedure is commonly named in the literature as *hyper-parameter optimization*. To emphasize the simplicity of the approach, we named it *micro* Text Classification or simply μTC.C.

This manuscript is organized as follows. The related work is presented in Section 2. Section 3 describes our contribution in depth. In Section 4, all the experimental details are described. In Section 5, we show an extensive experimental comparison of our approach with the relevant state-of-the-art methods over 30 different benchmarks. Finally, the conclusions are listed in Section 6.

2. Related work

Let us start by describing a typical text classifier which can be summarized as a set of few, but complex, parts [2]. Firstly, the input text is passed to a lexical analyzer that both parses and normalizes the text, it outputs a list of tokens that represent the input text. The lexical analyzer typically includes some simple transformation functions like the removal of diacritic symbols and lower casing the text, but it also can make use of sophisticated techniques like stemming, lemmatization, misspelling correction, etc. Whereas, the tokens are commonly represented by words, pairs or triplets of adjacent words (bigrams or trigrams), and in general, sequences of words (word n-grams). It is also possible to extend this approach to sequences of characters (character n-grams). When it is allowed to drop the middle words of word n-grams, we obtain skip-grams. The usage of these techniques is driven by the human knowledge of the particular problem being tackled. Also, it is worth to mention that the entire process is tightly linked to the input language.

Secondly, the output of the lexical analyzer is commonly used to create high dimensional vectors where each token of the vocabulary has a corresponding coordinate in the vector. So, the value of each coordinate is associated with the weight of that token. The traditional way of weighting is to use the local and global statistics of tokens, popular examples of this approach are *TF, IDF, TFIDF*, and *Okapi BM25*; alternatively, some information measures like the entropy are commonly used as weight terms. Many times it is desirable to reduce the dimension of the vector space, and several techniques can be used for that purpose, just like *PCA* [3] (Principal Component Analysis), and *LSI* [4] (Latent Semantic Indexing).

Finally, the output of the weighting scheme is used to create a training set which can be learned by a classifier. A classifier is a machine learning algorithm that learns the instances of a training set $\mathbb{T}$. In more detailed, the training set is a finite number of inputs and outputs, i.e., $\mathbb{T} = \{(x_i, y_i) \mid i = 1 \dots n\}$ where x_i represents the ith input, and y_i is the associated output. The objective is to find a function ψ such that $\forall_{(x,y)\in\mathbb{T}}\ \psi(x) = y$ and that could be evaluated in any element x of the input space. In general, it is not possible to find a function ψ that learns $\mathbb{T}$, perfectly. Consequently, a good classifier finds a function ψ that minimizes an error function or maximizes a score function.

Perhaps, one of the first generic text classifiers was proposed by Rocchio [5] that works by generating object prototypes based on centroids of a Voronoi partition over *TFIDF* vectors. This strategy shows the effort to reduce the necessary memory to fit in the hardware available at that time. Rocchio uses the nearest neighbor classifiers over prototypes to perform the predictions, the preprocessing of the text was left to the expertise of the user. Rocchio was the baseline and the study object in the area for a long time; such is the case of the work presented by Joachims [6], which describes a probabilistic analysis of the Rocchio algorithm.

With the purpose of improving the quality of the text classification task, Cardoso-Cachopo [7] proposes the use of centroids to enhance the power of several typical classifiers, such as kNN (k-nearest neighbors) and SVM (Support Vector Machines). Also, Cardoso-Cachopo published a number of datasets in various preprocessing stages, which are popular among the text classification community because using them allows focussing on the weighting and classification algorithms, avoiding to tackle the text processing problem.

In [8], a machine learning algorithm is used to create a spam detector. The proposed method uses a combination of features, preprocessing steps or setup details, such as using lemmatization, stop-list, keywords patterns, varying the length of the training corpus, etc. Similar work is presented by Androutsopoulos et al. in [9].

In the topic classification task, Debole and Sebastiani [10] presents an experimental scheme with Reuters dataset and three machine learning methods (Rocchio algorithm, k-NN, and SVM), and also, three-term selection functions (information gain, chi-square and gain ratio). In [11] it is proposed a topic modeling algorithm based on Latent Dirichlet Allocation (LDA) which assign one topic to an unlabeled document; also, the paper also experiments with the combination of LDA and Expectation-Maximization (EM) algorithm.

Another approach to text classification is to move the focus from text processing and text classification, to improve the term-weighting; this is a successful strategy followed by recent works. Cummins [12] proposes a method based on Genetic Programming to determine and evaluate several term weighting schemes for the vector space model. Escalante et al. [13] present an approach to improve the performance of traditional term-weighting schemes using genetic programming. Their approach outperforms standard schemes, based on an extensive experimental comparison. The authors also compare the Cummins [12] approach over their benchmarks.

Lai et al. [14] use both recurrent and convolutional neural networks to produce a term-weighting scheme that captures semantics from the text. Similarly to word embeddings [15,16], the authors represent words based on their context and, also, they use

skip-grams for text representation. The experimental results show higher values of macro-F1 in comparison to other state-of-the-art methods.

Vilares et al. [17] introduce an unsupervised approach for multilingual sentiment analysis driven by syntax-based rules; the words are weighted based on the analysis of syntax-graphs. The authors provide experimental support for English, Spanish, and German. However, to support an additional language, it is needed to implement several rules and a proper syntax parser.

Mozetič et al. [18] study the effect of the agreement among human taggers in the performance of sentiment classifiers. In this way, they compare several classifiers over a traditional text normalization and a vector representation with *TFIDF* weighting. They provide 14 tagged datasets for European languages; we selected some of them for our benchmarks.

Author profiling is another important task related to text categorization, where several advances have been proposed. In [19] the authors report their approach to performing author profiling; in particular, they describe the best classifier of the PAN'13 contest that consists on a distributional word representation based on the membership to each class along with a number of text standard text preprocessing, see [20]. Recently, in PAN'17 [21], some current works related to user profiling are presented. In this case, user profiling is related to gender and language-region classification. In this aspect, in [22], an SVM with linear kernel, in combination with word unigrams, character 3- to 5-grams and POS (Part of Speech) features are employed. In [23] the features were selected as word and POS n-grams, the number of emojis in the text, document sentiment, character flooding (counting the number of times that three or more identical character sequence appears in the text). Finally, the authors also use a lexicon of relevant words to improve their system.

The efforts to solve the text classification problem are varied, and we attempt to test our approach with this variety of solutions. All the related work could be split into the approaches employed like term weighting, feature engineering, feature selection, deep learning, and others. For instance, [10,12,13,24–26] use term weighting to cope the text classification problem. Deep learning (neural networks with many hidden layers) are used in [14]. Also, feature selection and feature engineering is used by Meina et al. [27], Martinc et al. [23], Androutsopoulos et al. [28] and also Li and Huang [26]. Finally, the rest of the related work use a tranditional, yet disparate, approach [7,11,18,22,29–31]. Our approach focuses on model selection, and more detailed, our models are based on a large number of traditional schemes like text processing and normalization, a set of multiple tokenizers, and in selecting a proper term weighting scheme for a specific task.

2.1. Benchmark description

In the other hand, the text classification literature has a myriad of datasets, performance measures, and validation schemes. We select several prominent and popular benchmark configurations in the literature; for instance, we select to work with topic classification, spam identification, author profiling, authorship attribution, and sentiment analysis. The diversity of benchmarks and validation schemes help us to prove the functionality of our approach in many circumstances.

The Reuters-21578[5] is one of the most used collection for text categorization research. The documents were manually labeled by personnel from Reuters Ltd. The 20Newsgroup[6] dataset is very popular in text classification area and it contains news related to different topics originally collected by Ken Lang. The WebKB dataset[7] contains university webpages. This dataset is composed of the webpages classified in seven different categories: student, faculty, staff, department, course, project and other. We use the four most popular classes in our experiments. The CADE dataset [7] is another collection of webpages, specifically Brazilian webpages classified by human experts. This collection contains a total of 12 classes, e.g. services, sports, science, education, news, among others. The PU [9] is a collection of emails written in English and other languages, classified as spam and non-spam messages; this collection contains the following datasets: PUA, PU1, PU2 and PU3. Ling-Spam dataset [32] is also a spam dataset. PAN contest [20,21] has several tasks, between them are author identification and author profiling. The author profiling task is a forensic linguistics problem that consists in detecting gender and age for the author (PAN'13). For the PAN'17, age identification task was replaced by the task of determining the language variety of the writter, also, the number of different languages was increased up to four. As listed in Table 1, the official dataset is undisclosed, and each algorithm must be evaluated with the TIRA evaluation platform.[8] The authorship attribution datasets [13] are a set of different types of topics: CCA, NFL, Business, Poetry, Travel and Cricket. The objective of these datasets is to determine the authorship of each document. The multilingual sentiment analysis are a set of tweets in different languages: Arabic, German, Portuguese, Russian, Swedish and Spanish. The purpose of these datasets is classifying each tweet as negative, neutral, or positive polarity.

A detailed description of all these datasets is provided in Table 1, where there can be found some particularities of the dataset like the written language, the number of documents, the kind of evaluation (independent train-test sets or *k*-folds), the number of classes, and the performance measure optimized by μTC. It is worth to mention that in order to avoid implementation mistakes, we directly use the reported performances by the literature.

3. μTC: A combinatorial framework for text classification

Our approach consists in finding a competitive text classifier for a given task among a (possibly large) set of candidates classifiers. A text classifier is represented by the parameters that determine the classifier's functionality along with the input dataset. The search of the desired text classifier should be performed efficiently and accurately, in the sense that the final classifier should be competitive concerning the best possible classifier in the defined space of classifiers.

In the first part of this section, we will describe the structure of our approach, that is, we state the parameters defining our configuration space. Then, we define the μTC graph, which is the core structure used by the meta-heuristics implemented to find a good performing text classifier for a given task. In the road, we also describe the `score` function that encapsulates the functionality of the classifier and provides a numerical output necessary to maximize the efficiency of the classifier.

3.1. The configuration space

As mentioned previously, a text classifier consists of well differentiated parts. For our purposes, a classifier has the following parts: i) a list of functions that normalize and transform the input text to the input of tokenizers, ii) a set of tokenizer functions that transform the given text into a multiset of tokens, iii) a function

[5] http://www.daviddlewis.com/resources/testcollections/reuters21578/.
[6] http://people.csail.mit.edu/jrennie/20Newsgroups.
[7] http://www.cs.cmu.edu/~webkb/.
[8] https://tira.io .

Table 1
Description of the benchmarks and its associated performance measure.

Name	Language	#documents			#classes	Performance
		Total	Train	Test		measure
Topic classification						
R8	English	7674	70%	30%	8	macro-F1
R10	English	8008	70%	30%	10	macro-F1
R52	English	9100	70%	30%	52	macro-F1
News-4	English	13,919	70%	30%	4	macro-F1
News-20	English	20,000	70%	30%	20	macro-F1
WebKB	English	4199	70%	30%	4	macro-F1
CADE	Portuguese	40,983	70%	30%	12	macro-F1
Spam identification						
Ling-Spam	English	2893	— 10-fold —		2	macro-F1
PUA	English[a]	1142	— 10-fold —		2	macro-F1
PU1	English[a]	1099	— 10-fold —		2	macro-F1
PU2	English[a]	721	— 10-fold —		2	macro-F1
PU3	mixed[a]	4139	— 10-fold —		2	macro-F1
Author profiling						
PAN'13 Gender & Age group	English	242,040	236,600	25,440	2 & 3	accuracy
	Spanish	84,060	75,900	8160	2 & 3	accuracy
PAN'17[b] Gender & Language Variety	Arabic	–	2400	–	2 & 4	accuracy
	English	–	3600	–	2 & 6	accuracy
	Spanish	–	4200	–	2 & 7	accuracy
	Portuguese	–	1200	–	2 & 2	accuracy
Authorship attribution						
CCA	English	1000	500	500	10	macro-F1
NFL	English	97	52	42	3	macro-F1
Business	English	175	85	90	6	macro-F1
Poetry	English	200	145	55	6	macro-F1
Travel	English	172	112	60	4	macro-F1
Cricket	English	158	98	60	4	macro-F1
Multilingual sentiment analysis						
Arabic	Arabic	2000	— 10-folds —		3	macro-F1
German	German	91,502	— 10-folds —		3	macro-F1
Portuguese	Portuguese	86,062	— 10-folds —		3	macro-F1
Russian	Russian	69,100	— 10-folds —		3	macro-F1
Spanish	Spanish	19,767	— 10-folds —		3	macro-F1
Swedish	Swedish	49,255	— 10-folds —		3	macro-F1

[a] These datasets are encoded in a way that the original text is loss, however it preserves the document's distribution.
[b] Here, the documents are Twitter's profiles, each user is described 100–300 single entries for a total of 1,265,898 tweets for all languages in the training set.

that generates a vector from the multiset of tokens; and finally, iv) a classifier that knows how to assign a label to a given vector. These pieces define a μTC space of configurations, which is defined by the tuple $(\mathcal{T}, \mathcal{G}, \mathcal{H}, \Psi)$. In the following paragraphs a more detailed description is given.

1. $\mathcal{T} = \{T_i\}$ is the space of transformation functions, where T_i is defined as the identity function I and a set of related functions, mutually exclusive.[9] We define the function $f(S) = (f_{|\mathcal{T}|} \circ \ldots \circ f_1)(S)$ such that $f_i \in T_i$, where the parameter S is a text, i.e., a string of symbols.
2. $\mathcal{G} = \{G_i\}$ is the set of tokenizer functions. Each G_i is defined as either a function that returns $\emptyset$ or a simple tokenizer function, i.e., a tokenizer function is a function that extracts a list of subsequences of the given argument. More precisely, the function $g(S) = g_1(S) \cup \ldots \cup g_{|\mathcal{G}|}(S)$ is defined; where g_i such that $g_i \in G_i$, extracts a list of subsequences of S. The final multiset is named as *bag of tokens*.
3. $\mathcal{H}$ is a set of functions that transform a bag of tokens v into a vector $\vec{v}$ of dimension d, i.e., $h : \{S\}^+ \rightarrow \mathbb{R}^d$ where S is a non empty string, $h \in \mathcal{H}$. The proper value of each vector's coordinate is also determined by h; the later task is commonly known as *weighting scheme*.
4. Finally, Ψ is a set of functions that create a classifier for a given labeled dataset as knowledge source.

[9] The identity function is defined as $I(S) = S$.

Now, let $\mathcal{C}$ be the set of all possible configurations of the μTC space; therefore, it is defined as follows:

$$\mathcal{C} = T_1 \times \ldots \times T_{|\mathcal{T}|} \times G_1 \times \ldots \times G_{|\mathcal{G}|} \times \mathcal{H} \times \Psi$$

then, the size of $\mathcal{C}$ is described by

$$|\mathcal{C}| = \left(\prod_{i=1}^{|\mathcal{T}|} |T_i| \right) \cdot 2^{|\mathcal{G}|} \cdot |\mathcal{H}| \cdot |\Psi|$$

Without loss of generality, the size of the search space can be summarized as $(2 + O(1))^{|\mathcal{T}|+|\mathcal{G}|} \cdot |\mathcal{H}| \cdot |\Psi|$, where the $O(1)$ term captures the effect of T_is with more than two member functions. This means that $|\mathcal{C}|$ is lower bounded by $2^{|\mathcal{T}|+|\mathcal{G}|}$, i.e., all T_is are binary and both $\mathcal{H}$ and Ψ are singletons. Even on the simplest setup, the configuration space grows exponentially with the number of possible transformations and tokenizers. Thus, in order to find the best item, it is necessary to evaluate the entire space; this is computationally not feasible.[10] A typical configuration space can contain billions of configurations such that the exhaustive evaluation is not feasible in current computers. To remain as a practical approach, instead of performing an exhaustive evaluation of $\mathcal{C}$ to find the best configuration, we soften the problem to find a (very) competitive configuration; then it can be solved as a combinato-

[10] For instance, evaluating each configuration takes about 10 min on a commodity workstation; more about this will be detailed in the experimental section.

rial optimization problem, in particular, as the maximization of a score function.

Section 4 details the implemented configuration space that produce our experimental results, also a bunch of examples is given to help in the understanding of our approach. In the rest of this section, we describe more details about the statement of the combinatorial problem behind μTC.

3.2. The configuration graph

In order to solve the combinatorial problem with local search-based meta-heuristics, it is necessary to create a graph where the vertex set corresponds to $\mathcal{C}$, and the edge set corresponds to the neighborhood of each vertex, $\{N(c) \subseteq \mathcal{C}^+ \mid c \in \mathcal{C}\}$. The edges are simply denoted by the neighborhood function N, so $(\mathcal{C}, N)$ is a μTC graph.

Our main assumption is simple and feasible, the function score slowly varies on similar configurations, such that we can assume some degree of *locally concaveness*, in the sense that a well-performing local maximum can be reached using greedy decisions at some given point. Even when this is not true in general, the solver algorithm should be robust enough to get a good approximation even when the assumption is valid only to a certain degree. To induce the search properties, the neighborhood N should be defined in such a way that neighborhoods describe only similar configurations. For this matter, we should define a distance function between configurations. First, we must define a comparison function,

$$\Delta(a, b) = \begin{cases} 1 & \text{if } a \text{ and } b \text{ are the same function} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Since each configuration is a tuple of functions, the Hamming distance over configurations is naturally defined as follows

$$d_H(u, v) = \sum_{i=1}^{|\mathcal{T}|+|\mathcal{G}|+2} \Delta(u_i, v_i). \tag{2}$$

Now, we can define $N(c, r_{\max}) = \{u \in \mathcal{C} \mid 0 < d_H(u, c) \leq r_{\max}\}$, for any $r_{\max}$ and a c configuration . However, the number of items grows exponentially with the radius, and therefore, the notion of locality will be rapidly degraded. To maintain the locality, we define the neighborhood as:

$$N(c) = \{u \in \mathcal{C} \mid d_H(c, u) = 1\}. \tag{3}$$

Under this construction scheme, the diameter of $(\mathcal{C}, N)$ is determined by the length of the configuration tuple, i.e., $O(\log|\mathcal{C}|)$, the diameter determines the number of hops in the μTC graph that an optimal opt algorithm will perform, in the worst case. However, since the best configuration is unknown, we must use *score* as an *oracle* that leads our navigation at each step.

3.3. The score function

The score function evaluates the performance of the text classifier defined by the configuration with the given training and test sets. Without loss of generality, the evaluation of a configuration $c \in \mathcal{C}$ can be described by three main steps:

1. The dataset $\mathcal{D}$ is divided into two datasets D_{train} and D_{test}.
2. The μTC algorithm described by c learns from D_{train}.
3. The prediction performance of c is evaluated using the dataset D_{test}, more details are given below.

These steps can be modified to support cross-validation, schemes like k-folds or bagging, which provide a more robust way to measure the performance of a classifier. The details of these measurement strategies are beyond the scope of this manuscript, the interested reader is referenced to Ch. 9 of [33].

Now, please recall from Section 3.1 that c contains the parameters for a number of functions that transform the input text into its associated label. Given a configuration c, a classifier ψ is created using the labeled dataset transforming all texts in the training set into its corresponding vector form, i.e., $h \circ g \circ f(S)$ for each $S \in D_{\text{train}}$. Once the classifier ψ is trained, the associated label for all $S \in D_{\text{test}}$ is computed as $\psi \circ h \circ g \circ f(S)$. Finally, the performance of c is computed comparing the predicted labels against the actual ones. Typical score functions used are F1 (macro, micro, or weighted), accuracy, precision, or recall to measure the quality of the text classifier [34].

The metrics are defined as follows;

$$\begin{aligned}
\text{accuracy} &= \frac{\text{TP} + \text{TN}}{\#\text{samples}} \\
\text{precision}_{(k)} &= \frac{\text{TP}_{(k)}}{\text{TP}_{(k)} + \text{FP}_{(k)}} \\
\text{recall}_{(k)} &= \frac{\text{TP}_{(k)}}{\text{TP}_{(k)} + \text{FN}_{(k)}} \\
F_{1,k} &= \frac{2 \cdot \text{precision}_{(k)} \cdot \text{recall}_{(k)}}{\text{precision}_{(k)} + \text{recall}_{(k)}} \\
\text{micro-}F_1 &= \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \\
\text{(macro or weighted)-}F_1 &= \sum_{k \in \mathcal{L}} w_k \cdot F_{1,k} \\
w_k &= \begin{cases} \text{macro} \rightarrow & 1/\mathcal{L} \\ \text{weigthed} \rightarrow & \frac{|\{u \in \mathcal{D} | u = k\}|}{|\mathcal{D}|} \end{cases}
\end{aligned}$$

Here, TP denotes the number of true positives, TN the true negatives; FP and FN are the number of false positives and false negatives, respectively. All these measures can be parametrized by class k. $\mathcal{L}$ defines a set of classes. The *accuracy* is calculated by dividing the total of correctly classified samples by the total number of samples. The $F_{1,k}$ is defined as the harmonic mean of the *precision* and the *recall*, per class. When the weights are uniform, no matter the population of each class, the measure is known as macro-F_1; when the weights are given by the population of each class, the measure is named weighted-F_1. The micro-F_1 is defined as the harmonic mean between precision and recall, computed globally (not for classes). The interested reader is referenced to an excellent survey of text classifiers and performance metrics [33,34].

3.4. Optimization process

The core idea to solve the optimization problem is to navigate the graph $(\mathcal{C}, N)$ using a combination of two meta-heuristics. In the following paragraphs, we briefly review the techniques we used to solve the combinatorial problem, a proper survey of the area is beyond the scope of this manuscript. However, the interested reader is referred to [35,36].

To maintain μTC in practical computational requirements, we select two types of fast meta-heuristics, *Random Search* [37] and *Hill Climbing* [35,36] algorithms. The former consists in selecting the best performing configuration among the set $\mathcal{C}'$ randomly chosen from $\mathcal{C}$, that is,

$$\arg\max_{c \in \mathcal{C}'} \text{score}(c),$$

where the size of $\mathcal{C}'$ is an open parameter dependent on the task. On the other hand, the core idea behind Hill Climbing is to explore the configuration's neighborhood $N(c)$ of an initial setup c and then greedily update c to be the best performing configuration in $N(c)$.

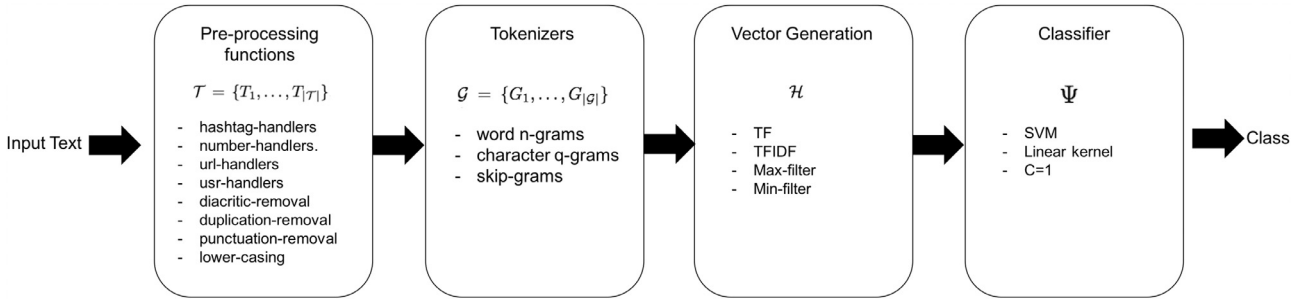


Fig. 1. The processing pipeline of μTC. The input text is firstly normalized and transformed with $\mathcal{T}$; then, it is converted into a multiset applying the selected tokenizers $\mathcal{G}$. This multiset is then represented as a vector using $\mathcal{H}$, applying the precise filters and weighting schemes; finally, the vector is classified using the model Ψ.

The process is repeated until no improvement is possible, that is,

$$\text{score}(c) \geq \max_{u \in N(c)} \text{score}(u).$$

We improve the whole optimization process applying a Hill Climbing procedure over the best configuration found by a Random Search. We also add memory to avoid a configuration to be evaluated twice.[11]

Summarizing, the optimization process is driven by the tuple $(\mathcal{C}, \mathcal{D}, \text{score}, \text{opt})$, where i) C is the μTC space, ii) $\mathcal{D}$ means the training set of labeled texts, iii) score is the function to be maximized, and finally, iv) opt is a combinatorial optimization algorithm that uses score and $\mathcal{D}$ to find an almost optimal configuration in $\mathcal{C}$.

4. A practical instantiation of a μTC space

This section describes the setup used to characterize and compare our method with the related state-of-the-art approaches. In particular, we define the set of functions used to create our μTC space. The reader will also find an example of how this pieces are applied to transform a text into a mathematical object useful for traditional classifiers.

As stated before, μTC is a framework to create text classifiers searching for best models in a configuration space. This space can be adjusted for any particular problem, but here, we consider a general enough space to match a disparity of benchmarks (listed in Section 2.1).

When the knowledge about the domain is low, then a broad and generic configuration space should be used. It could be tempting to learn about the domain using the information found by the optimization process; this is possible. However, it is encouraged to take into account that the search process will make decisions to match the particular dataset, not the domain, and a domain expert must curate any generalization of the knowledge. It is important to mention that large configuration spaces will consume a lot of computational time to be optimized.

On the other hand, a hand-crafted configuration space for a given problem can yield to very fast processing times; however, a vast knowledge of the domain is required to reach this state. In this case, we discard the possibility of discovering new knowledge on the domain and take advantage of the particularities of the dataset that a more general configuration space can provide.

To tackle with the disparate list of benchmarks, we select a generic large configuration space defined in the following paragraphs. The general flow of computation is illustrated in Fig. 1. In the following paragraphs, each component is described.

[11] In principle, this is similar to Tabu search; however, our implementation is simpler than a typical implementation of Tabu search.

Preprocessing functions $\mathcal{T} = \{T_1, \ldots, T_{|\mathcal{T}|}\}$. We associate T_i to the following function sets.

- **hashtag-handlers.** Defined as $\{remove\_htags, group\_htags, identity\}$, the idea is to allow to remove or group into a single tag all hash tags, for *remove_htags* and *group_htags*, respectively; the *identity* function lets the text unmodified. The format of a hash tag is that introduced by Twitter *#words*, but now popular along many data sources.
- **number-handlers.** Defined as $\{remove\_num, group\_num, identity\}$, this function set contains functions to remove, group, or left untouched numbers in the text.
- **url-handlers.** Defined as $\{remove\_urls, group\_urls, identity\}$, this function set contains functions to remove, group, or left untouched numbers in the text.
- **usr-handlers.** Defined as $\{remove\_usr, group\_usr, identity\}$, this function set contains functions to remove, group, or left untouched users and host domains in the text. The pattern being tackled is `@user` this is a popular way to denote users in several social networks; the pattern also matches naturally with the domain part of email addresses.
- **diacritic-removal.** Defined as $\{remove\_diac, identity\}$, this function set contains functions to remove, or left untouched, diacritic symbols in the text. The objective is to reduce composed symbols like `á,ä,ã,â,` or `à` to simply `a`. This is a well known source of errors in informal text written in languages making hard use of diacritic symbols
- **duplication-removal.** Defined as $\{remove\_dup, identity\}$, this function set contains functions to remove, or left untouched, duplicated contiguous symbols in the text.
- **punctuation-removal.** Defined as $\{remove\_punc, identity\}$, this function set contains functions to remove, or left untouched, duplicated punctuation symbols in the text. The list of punctuation symbols includes several symbols like `;,:,.,-,','',(,),[,],{,},~, <,>,?,!,` among others.
- **lower-casing.** Defined as $\{lower\_case, identity\}$ contains functions to normalize the case of the text or left untouched.

The list of tokenizers $\mathcal{G} = \{G_1, \ldots, G_{|\mathcal{G}|}\}$. After all text normalization and transformation, a list of tokens should be extracted. We use three schemes for our tokenizers.

- **Word n-grams.** This family of tokenizers firstly tokenizes the text into words, and then, produces $m - n + 1$ tokens for m words, i.e. word n-grams. An n-gram is a string of n consecutive words. For example, The red car is in front of the tree creates the following 3-grams: The red car, red car is, car is in, is in front, in front of, front of the, of the tree.
- **Character n-grams.** This family of tokenizers does not assume anything about the text and splits the input text to all n-

sized substrings, i.e., $m - n + 1$ substrings of characters for a text of m characters. For example, the character 4-grams of `I like the red car` are `I_li, _lik, like, ike_, ke_t, e_th, _the, the_, he_r, e_re, _red, red_, ed_c, d_ca, _car`. We use the symbol _ to show the symbol space.

Skip-grams. Skip-grams are similar to word n-grams but allowing to *skip* the middle parts. For example, the (2, 1) skip-grams[12] of the previous example are `I-the, like-red, the-car`. The idea behind this family of tokenizers is to capture the occurrence of related words that are separated by some unrelated words.

For this matter, instead of selecting one or another tokenizer scheme, we allow to select any of the available tokenizers and perform the union of the final multisets of tokens. For instance, our configuration space considers three word n-grams tokenizers ($n = 1, 2, 3$), six character n-grams ($n = 1, 2, 3, 5, 7, 9$), and four skip-grams (2, 1), (2, 2), (3, 1) and (3, 2). We use one possible combination of tokenizers per configuration, in this case, one among $2^{13} - 1$ combinations.

Weighting schemes $\mathcal{H}$. After we obtained a multiset (bag of tokens) from the tokenizers, we must create a vector space. We selected a small set of frequency filters and the TFIDF scheme to weight the coordinates of the vector. On one hand, we consider a sequential list of filters `max-filter` and `min-filter`, and then, we select to use the term frequency (TF) or the TFIDF as weight. For the `max-filter` we delete all tokens surpassing the frequency threshold of $\alpha \cdot \text{max-freq}$, where `max-freq` is the maximum frequency of a token in the collection. We consider three filters, for instance we use $\alpha \in \{0.5, 0.9, 1.0\}$. For the `min-filter` we delete all tokens not reaching the frequency threshold of `freq`, for instance we use, $\text{freq} \in \{1, 3, 5, 10\}$. Notice that $\alpha = 1.0$ and $\text{freq} = 1$ does not perform any filtering. So, we have embedded 24 different configurations for weighting.

Classifier Ψ. We decide to use a singleton set populated with an SVM with a linear kernel. It is known that SVM excels for large dimensional input (which is our case), and the linear kernel also performs quite well under these conditions. We do not optimize the parameters of the classifier since we are interested in the rest of the process; however, in some sense, our model is also selected to maximize the performance of the given classifier. We use the SVM classifier from *liblinear* (C parameter equals to 1), Fan et al. [38]. Nevertheless, μTC can work with any classifier; but we recommend to use a classifier with good performance on high dimensional data.

4.1. A step-by-step example

In a complete implementation, some configurations are selected by the optimizer, and its performance is measured using some variant of cross-validation; of course, at each probe, the optimizer will try to improve the best-known configuration. When each configuration is tested, the described workflow is applied to the entire set, and the resulting vectors are used to train the classifier Ψ. In this point, the instantiated configuration is called a model.

A model is then capable of classifying new texts, using the processing and tokenization schemes specified in the configuration, then, the representing vector is computed as the model indicates, and finally, the text is labeled using the classifier of the model.

[12] Two words, skipping one in the middle.

Perhaps the largest, and cumbersome, part of our approach is in both the list of text transformations and the combination of tokenizers. However, it is essential to the functionality of our approach. A step by step example of these two transformations is given below.

For instance, suppose a configuration that specifies to remove hashtags, numbers, URLs, users, diacritics, duplicated contiguous characters, and punctuations; also, whenever it applies, characters are transformed to lower case. Then, a multiset is extracted using the selected tokenizers, in this example, we will use 3-grams, 1-words, 2-words, and (2,1)-skip-grams (sequences of two words with a word as a gap).

Table 2 shows the transformation pipeline for a short text example. At each step, the text is slightly transformed using a simple function, indicated in the configuration of the model. Then, the resulting text, `cada dia es mejor`, becomes the input of the tokenizers:

3-grams → {`cad,ada,da_,a_d,_di,dia,ia_,a_e, _es,es_,s_m,_me, mej,ejo,jor`}
1-words → {`cada,dia,es,mejor`}
2-words → {`cada_dia,dia_es,es_mejor`}
(2,1)-skip-grams → {`cada_es,dia_mejor`}

As commented, the multiset union of these tokenizers becomes the input of the vectorizer $\mathcal{H}$, the vectorizer uses a weighting scheme to produce the output and this resulting vector is passed to Ψ. Note that the following steps are tightly dependent on a particular training set.

4.1.1. About the final configuration space

The task of finding the best model for the space of configurations is hard. The number of possible configurations of $\mathcal{T}$ is 1296 (i.e., four trivalent functions sets and four bivalent function sets). From the above configuration, the number of combinations of tokenizers, excluding the empty set, is 8191; also, we have 24 different weighting combinations. So, the configuration space contains more than 254 million configurations. For instance, a configuration needs close to ten minutes to be evaluated, i.e., a sentiment analysis benchmark with ten thousand tweets. Therefore, an exhaustive evaluation of the configuration space will need more than 4800 years. Even implementing it in a large distributed cluster the process requires too much time to complete. Such power of computing is not easily accessible. Nonetheless, if we soften the problem to finding not the best model but an excellent one, we can use an algorithm for combinatorial optimization, as explained in Section 3.

4.2. On the preparation of the input text

Since μTC considers the preprocessing step among its parts, we tried to collect all datasets in raw text, without any kind of preprocessing transformations. This was not possible in the general case, mostly due to the aging of datasets; we consider the following text preparation states, in the style of Cardoso-Cachopo [7]:

- the *raw* text corresponds to the original, non-formatted text,
- the *all-terms* converts all text into lowercase, also, all diacritic symbols and punctuation marks are removed, and all spacing symbols are normalized to a single space,
- the *no-short* dataset removes all terms having less or equal than three characters,
- the *no-stopwords* dataset also removes all non discriminant words for English (adjetives, adverbs, conjunctions, articles, etc.)

Table 2
Example of a text representation pipeline.

Input text		`Cada día es MEEEJOR!! @user #ViviendoAlMaximo http://some/url`
		tr. *Every day is better!! @user #LivingToTheFullest http://some/url*
Remove hashtags	→	`Cada día es MEEEJOR!! @user http://some/url`
Remove numbers	→	`Cada día es MEEEJOR!! @user http://some/url`
Remove urls	→	`Cada día es MEEEJOR!! @user`
Remove users	→	`Cada día es MEEEJOR!!`
Remove diacritics	→	`Cada dia es MEEEJOR!!`
Remove duplicates	→	`Cada dia es MEJOR!`
Remove punctuation	→	`Cada dia es MEJOR`
Lower case	→	`cada dia es mejor`

Table 3
Authorship attribution data sets.

Dataset	macro-F1			Dataset	Accuracy		
	Cummins and O'Riordan [12] and Escalante et al. [13]	Escalante et al. [13]	μTC		Cummins and O'Riordan [12] and Escalante et al. [13]	Escalante et al. [13]	μTC
CCA	0.0182	0.7032	**0.7633**	CCA	0.1000	0.7372	**0.7660**
NFL	**0.7654**	0.7637	0.7422	NFL	0.7778	**0.8376**	0.7555
Business	0.7548	0.7808	**0.8199**	Business	0.7556	**0.8358**	0.8222
Poetry	0.4489	0.7003	**0.7135**	Poetry	0.5636	0.7405	**0.7272**
Travel	0.6758	0.7392	**0.8621**	Ravel	0.6833	0.7845	**0.8667**
Cricket	0.9170	0.8810	**0.9665**	Cricket	0.9167	0.9206	**0.9667**

- finally, after the previous steps, all words were transformed by the Porter's stemmer for English [39] to generate the *stemmed* dataset.

For instance, we use the *all-terms* for R8, R10, R52 and WebKB; for CADE we use the *stemmed* version. In these cases, we used the datasets prepared by Cardoso-Cachopo [7]. In other cases, we use the raw text. The effect of using one or another state is studied in Section 5.1.

5. Experimental results

This section is dedicated to comparing our work with the relevant state-of-the-art methods described above. Also, we characterize the generalization power in terms of the validation scheme.

All the experiments were run in an Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60 GHz with 32 threads and 192GiB of RAM running CentOS Linux 7.1. We implemented μTC[13] on Python. To characterize the performance of μTC and compare it to the relevant state of the art, we selected a number of popular benchmarks in the literature; these datasets are described below. It is worth to mention that we bias our selection to benchmarks coming from popular international challenges. With the purpose of avoiding overfitting, we performed the model selection using score as a 3-fold cross-validation of the specified performance measure, see Table 1. We decided to use cross-validation for this stage because we observed overfitting for small datasets, like those found in authorship attribution, when we use a static train-test partitions to perform model selection. A brief experimental study of the effect of the validation schemes is presented in Section 5.3.

The first task analyzed is authorship attribution, Table 3 shows the macro-F1 and accuracy performances for a set of authorship attribution benchmarks. Here, we compare μTC with two term-weighting schemes [12,13]. The pre-processing stage of the μTC's input is *all-terms*; others use the *stemmed* stage. The best performing classifiers are created by μTC, except for NFL where alternatives perform better. In the case of Business, Escalante et. al [13] performs slightly better only in terms of accuracy. Please notice that NFL and Bussiness are among the smaller dataset we tested, the low performance of μTC can be produced by the low number of exemplars, while alternative schemes take advantage of the few samples to compute better weights.

Table 4
Performance of μTC on the author profiling task of the PAN'13 competition; all values are the accuracy score in the specified subtask.

	Task	English	Spanish	Avg.
μTC	Age	0.6605	0.6897	**0.6751**
	Gender	0.5867	0.6750	**0.6309**
	Joint	0.3946	0.4587	**0.4267**
Pastor Lopez-Monroy et al. [24]	Age	0.6572	0.6558	0.6565
	Gender	0.5690	0.6299	0.5995
	Joint	0.3813	0.4158	0.3985
Santosh et al. [29]	Age	0.6408	0.6430	0.6419
	Gender	0.5652	0.6473	0.6063
	Joint	0.3508	0.4208	0.3858
Meina et al. [27]	Age	0.6491	0.4930	0.5711
	Gender	0.5921	0.5287	0.5604
	Joint	0.3894	0.2549	0.3222

In Table 4 the results of PAN'13 competition are presented. According to the contest report [20], the best results were achieved by Pastor, Santosh, and Meina. In this benchmark, μTC produces the best result in all average cases. In a fine-grained comparison, only Meina surpasses μTC on the gender identification for English.

Table 5 shows the performance of μTC in the PAN'17 benchmark. The table also lists the best three results of the challenge, reported as statistically equivalent in [21], these works are detailed in Section 2. Please note that the result by Tellez et al. [40] was generated with μTC but using a special term-weighting scheme based on entropy instead of TFIDF (or TF). The details of the entropy based term-weighting scheme are beyond the scope of this contribution; the interested reader is referenced to [40]. The plain μTC, as described in this manuscript, achieves accuracies of 0.7880

[13] Available under Apache 2 license at https://github.com/INGEOTEC/microTC .

Table 5
Author profiling: PAN2017 benchmark [21], all methods were scored with the official gold standard. All scores are based on the accuracy computation over the specified subset of items.

Method	Task	Arabic	English	Spanish	Portuguese	Avg.
μTC	Gender	0.7569	0.7938	0.7975	0.8038	0.7880
	Variety	0.7894	0.8388	0.9364	0.9750	0.8849
	Joint	0.6081	0.6704	0.7518	0.7850	0.7038
Basile et al. [22]	Gender	0.8006	0.8233	0.8321	0.8450	**0.8253**
	Variety	0.8313	0.8988	0.9621	0.9813	**0.9184**
	Joint	0.6831	0.7429	0.8036	0.8288	**0.7646**
Martinc et al. [23]	Gender	0.8031	0.8071	0.8193	0.8600	0.8224
	Variety	0.8288	0.8688	0.9525	0.9838	0.9085
	Joint	0.6825	0.7042	0.7850	0.8463	0.7545
Tellez et al. [40]	Gender	0.7838	0.8054	0.7957	0.8538	0.8097
	Variety	0.8275	0.9004	0.9554	0.9850	0.9171
	Joint	0.6713	0.7267	0.7621	0.8425	0.7507

Table 6
Topic classification datasets.

	macro-F1						
	Reuters-8C	Reuters-10C	Reuters-52C	News-4C	News-20C	WebKB	CADE
Debole and Sebastiani [10]	–	–	–	–	–	–	–
Escalante Escalante et al. [13]	0.9135	0.9184	–	–	0.6797	0.8879	0.4103
Cummins and O'Riordan [12] and Escalante et al. [13]	0.8830	0.8759	–	–	0.6645	0.7197	–
Lai et al. CNN [14]	–	–	–	0.9479	–	–	–
Lai et al. RNN [14]	–	–	–	**0.9649**	–	–	–
Hingmire et al. [11]	–	–	–	–	–	0.7190	–
Cardoso-Cachopo [7]	–	–	–	–	–	–	–
μTC	**0.9698**	**0.9662**	**0.6746**	0.9432	**0.8269**	**0.9098**	**0.5687**
	accuracy						
	Reuters-8C	Reuters-10C	Reuters-52C	News-4C	News-20C	WebKB	CADE
Debole and Sebastiani [10]	–	0.7040	–	–	–	–	–
Escalante et al. [13]	0.9056	0.8821	–	–	0.6623	0.8912	0.5380
Cummins and O'Riordan [12] and Escalante et al. [13]	0.7440	0.7659	–	–	0.6578	0.7542	–
Lai et al. CNN [14]	–	–	–	–	–	–	–
Lai et al. RNN [14]	–	–	–	–	–	–	–
Hingmire et al. [11]	–	–	–	0.9360	–	–	–
Cardoso-Cachopo [7]	0.9049	–	0.8482	–	**0.8460**	0.8300	0.5071
μTC	**0.9214**	**0.9236**	**0.9376**	**0.9390**	0.8348	**0.9191**	**0.6174**

and 0.8849, respectively for gender and variety identification. The joint prediction of both classes achieves an accuracy of 0.7038. These score values locate the plain μTC in the eighth position in the official rank, see [21].

Table 6 reports the performance over topic classification benchmarks. This experiments considered several *news* datasets.[14] Our approach, μTC, reaches best results in most of the datasets with exception of News-20 and News-4 where μTC reaches second and third best performance.

In sentiment analysis task we compared the datasets reported in [30,31]. Moreover, we reported the results obtained with the B4MSA approach [1]. B4MSA is a method for multilingual polarity classification considered as a baseline to build more complex approaches[15]. It is important to note that from each dataset reported in [30,31], both approaches, B4MSA and μTC, use a subset specified in Table 7; e.g. in Arabic language we used 100%, in German we used 80% of the dataset and so on (all specified in table).

In Table 7, it can be seen that best results were obtained with B4MSA and μTC in all the cases, and both results are very close.

Finally, Table 8 shows the results of spam classification task. Here, it can be seen that best results in the macro-F1 measure were obtained with our approach μTC; nevertheless, the best results in the accuracy score were achieved by Androutsopoulos et al. [28] except in Ling-Spam dataset where μTC reached the best performance.

[14] Please refer to Table 1 for the detailed description of each benchmark.
[15] https://github.com/INGEOTEC/b4msa.

Table 7
Performance for multilingual sentiment analysis.

Language		macro-F_1	Accuracy
Arabic	Salameh et al. [30]	–	0.787
	Saif et al. [31]	–	0.794
	B4MSA (100%)	0.642	**0.799**
	μTC (100%)	0.641	0.792
German	Mozetič et al. [18]	–	0.610
	B4MSA (89%)	0.621	0.668
	μTC (89%)	0.614	**0.672**
Portuguese	Mozetič et al. [18]	–	0.507
	B4MSA (58%)	0.557	0.561
	μTC (58%)	0.562	**0.566**
Russian	Mozetič et al. [18]	–	0.603
	B4MSA (69%)	0.754	0.750
	μTC (69%)	0.754	**0.751**
Swedish	Mozetič et al. [18]	–	0.616
	B4MSA (93%)	0.680	**0.691**
	μTC (93%)	0.679	0.688
Spanish	B4MSA	0.657	**0.784**
	μTC	0.649	0.780

5.1. About the pre-processing state of the input text

Here, the pre-processing step is analyzed; for this, Table 9 shows different performances that correspond to the News benchmark in various stages of the normalization process, as used as inputs for μTC. We found that μTC achieves high performances without using additional sophisticated pre-processing steps, almost all of them, language dependent. For instance, using the raw

Table 8
Performance for spam classification.

Data set	macro-F1			
	Androutsopoulos et al. [28]	Sakkis et al. [25]	Li and Huang [26]	μTC
Ling-Spam	–	0.8957	0.9870	**0.9979**
PUA	0.8897	–	–	**0.9478**
PU1	0.9149	–	**0.983**	0.9664
PU2	0.6794	–	–	**0.9044**
PU3	0.9265	–	**0.977**	0.9701
		Accuracy		
Data set	Androutsopoulos et al. [8]	Sakkis et al. [25]	Li and Huang [26]	μTC
Ling-Spam	–	–	0.9800	**0.9993**
PUA	**0.9600**	–	–	0.9482
PU1	**0.9750**	–	0.971	0.9706
PU2	**0.9839**	–	–	0.9634
PU3	**0.9778**	–	0.968	0.9738

Table 9
The performance of μTC for text collections being in different stages of text normalization for News benchmark.

Kind of preprocessing	Actual accuracy	Actual macro-F1	Pred accuracy	Pred macro-F1
Raw	0.8265	0.8199	0.8968	0.8963
All-terms	0.8340	0.8260	0.9075	0.9056
No-short	0.8310	0.8235	0.9052	0.9034
No-stopwords	0.8373	0.8300	0.9099	0.9082
Stemmed	0.8413	0.8344	0.9071	0.9058

text is just below 0.0148 points than the performance using the *stemmed* collection; thus, μTC barely needs the human intervention to prepare the input text. Alternatively, methods like Escalante et al. [13] and Cardoso-Cachopo [7] need to use the stemmed version of the dataset to achieve its optimal performance, i.e., accuracy values ranging from 0.6623 to 0.8460, for more details see Table 6.

5.2. Selected configurations

In this section, the selected configurations are presented and analyzed. Table 10 shows the selected configurations for our benchmarks. In favor of simplicity, both PAN13 and PAN17 were left out of the analysis since they need too many configurations (see Table 1), one per language and sub-problem. The configurations in the table were selected using macro-F_1 and 3-folds cross-validation.

We grouped benchmarks by related-task to highlight the underlying patterns around and marked with "–" those parameters not being of use for that dataset. For instance, hashtag handlers are only necessary for *multilingual sentiment analysis*; and also, since the text of PUA, PU1, PU2, and PU3 is offuscated with some cyphering, almost every text pre-processing functions useless for them.

Firstly, we will describe the parameters related to text transformations and term weighting. In topic classification, note that the *remove* columns are barely used but almost other parameters are consistent with the selection. The spam identification task is also pretty uniform in the parameters; please note that Ling-Spam is the unique dataset with real words in the spam section. For authorship attribution and multilingual sentiment analysis, we observe a more varied selection of parameters. Please note that tfidf weighting is almost always selected and frequency filters set to $\alpha = 1.0$ and min-filter set to 1 (i.e., no filtering work); it could be tempting to assume that they should be fixed; however, our approach encourages the fine-tuning of all these parameters to fit the particular benchmark. Moreover, the hashtag handler is only selected to *none*, and it could be linked to the fact that they contain information encoded on them; many times they are full of information that can be extracted using the correct set of tokenizers. Again, for the presented datasets, the hashtag handler should be set to *none* not means that other tasks do not benefit from having a hashtag handler. For instance, if we set the hashtag handler to *group* and also use *tf* as weighting scheme, then we obtain a simple way to capture the number of hashtags in a tweet; the same occurs with URLs and users.

The rest of our configuration is dedicated to the set of tokenizers. Table 11 lists the set of tokenizers used by the best model, for each benchmark. We found that 1-words is used in almost any benchmark, of course, this is consistent with the significant trend of tokenizing by words. For topic classification, the selected tokenizers are spread out the entire table, except large q-grams (7 and 9), 3-words, and large skip-grams. The spam identification is divided, Ling-Spam finds out that using only 3-grams and 5-grams give the best performance while the rest of the spam benchmarks, only use n-words and skip-grams, mainly because q-grams do not have sense under the cyphering of these datasets. On the authorship attribution, the number of tokenizers used by each configuration is larger than any other task; even 2-words and 3-words are quite popular among the best configurations. The reason behind selecting both small q-grams and large n-words can be that both captures precise marks for each author, small ones because they can capture some morphological features, as each author prefers them, and large n-words capture accurate sentences and expression marks of each author. Finally, we observed that q-grams are quite crucial in sentiment analysis, specifically, on Twitter; this may be produced by the number of errors and variants of words found in social networks. Besides, 1-grams and 2-grams capture emojis and emoticons in a natural way.

5.3. On the robustness of the score function

The score function leads the model selection procedure to fulfill the requirements of the task. In this process, it is necessary to determine which precise quality's measure is needed, e.g., macro-F1 or accuracy. As any learning algorithm, it is necessary to protect the score with some validation schemes to avoid the latent overfitting. On this matter, we consider the use of two validation schemes: i) stratified k-folds and ii) a random binary partition of size βn for the train set and $(1-\beta)n$ for the test set, for a (training) collection of size n.

To learn how to choose the right criteria, we review both the *predicted* and the *actual* performance (macro-F1, for instance) of these two validation schemes. The predicted macro-F1 is the performance achieved by the model selection procedure using some of

Table 10
The selected configurations for our benchmarks. To improve the visual analysis, we replace "no" cells with blanks, and non-applicable values with "–".

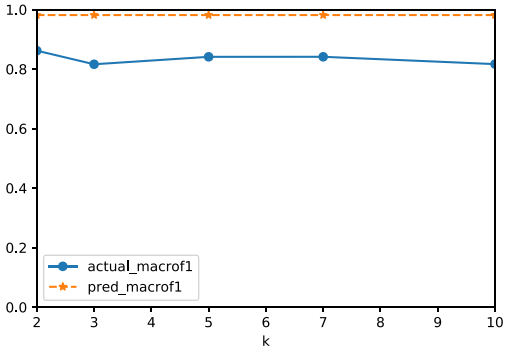
benchmark	Remove			Handlers				Lower	Freq. filters		
	diac.	dup.	punc.	hashtag	url	usr	num.	case	max.	min.	tfidf
Topic classification											
R8		yes	–	–	delete	none	group	yes	1.0	1	yes
R10			–	–	delete	none	group	yes	1.0	1	yes
R52			–	–	delete	none	group	yes	1.0	1	yes
News-4	yes	yes	yes	–	group	group	none	yes	1.0	1	yes
News-20		yes	yes	–	delete	none	group	yes	1.0	1	yes
WebKB	yes		–	–	none	group	delete	yes	0.5	5	yes
CADE		yes	–	–	delete	none	group		0.9	1	yes
Spam identification											
Ling-Spam		yes	yes	–	delete	none	group		0.9	1	yes
PUA	–	–	–	–	–	–	none	–	1.0	1	yes
PU1	–	–	–	–	–	–	none	–	0.5	1	
PU2	–	–	–	–	–	–	none	–	1.0	1	yes
PU3	–	–	–	–	–	–	none	–	1.0	1	yes
Authorship attribution											
CCA	yes			–	delete	none	none		0.5	5	
NFL	yes		yes	–	group	group	delete		1.0	1	yes
Business		yes		–	delete	group	group	yes	1.0	1	yes
Poetry	yes			–	group	group	delete		1.0	1	yes
Travel		yes		–	group	none	delete	yes	1.0	1	yes
Cricket				–	delete	none	group		0.5	1	yes
Multilingual sentiment analysis											
Arabic	yes	yes	yes	none	group	group	delete		0.9	1	yes
Spanish	yes	yes		none	group	group	none	yes	0.5	1	
German				none	group	delete	delete	yes	0.9	1	yes
Portuguese		yes		none	group	group	group	yes	0.9	1	yes
Russian	yes			none	none	group	delete	yes	0.5	1	yes
Swedish				none	group	group	none	yes	0.9	1	yes

Table 11
The selected configurations for our benchmarks. We replace "no" entries with blank cells to improve readability.

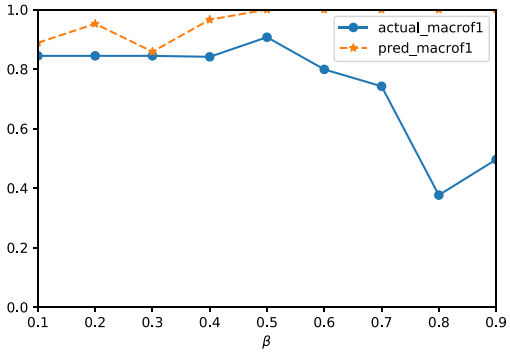
benchmark	q-grams						n-words			skip-grams			
	1	2	3	5	7	9	1	2	3	(2,1)	(2,2)	(3,1)	(3,2)
Topic classification													
R8		yes					yes						
R10			yes				yes						
R52	yes						yes						
News-4	yes	yes		yes			yes			yes			
News-20		yes		yes			yes			yes			
WebKB							yes	yes		yes	yes		
CADE			yes	yes			yes	yes		yes			
Spam identification													
Ling-Spam			yes	yes									
PUA	–	–	–	–	–	–	yes						
PU1	–	–	–	–	–	–	yes	yes			yes		
PU2	–	–	–	–	–	–	yes						
PU3	–	–	–	–	–	–	yes	yes					
Authorship attribution													
CCA		yes		yes	yes	yes		yes	yes		yes		
NFL	yes	yes					yes						
Business		yes	yes				yes	yes					
Poetry			yes						yes			yes	
Travel	yes		yes		yes	yes	yes		yes	yes			
Cricket	yes	yes	yes	yes									
Multilingual sentiment analysis													
Arabic	yes	yes	yes										
German		yes	yes		yes		yes				yes		
Portuguese	yes	yes	yes				yes						
Russian	yes	yes	yes		yes		yes						
Spanish	yes		yes		yes		yes			yes			
Swedish		yes	yes	yes	yes		yes						

the two mentioned validation schemes. The actual performance is the one obtained directly evaluating the gold-standard collection.

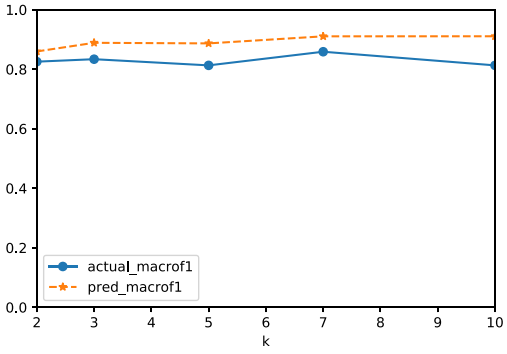
Fig. 2 shows the performance of μTC on small databases. The stability of k-folds in terms of predicted and actual performance is supported by Fig. 2(a), (c) and (e). This is also true for larger datasets like those depicted in Fig. 3(a), (c) and (e). The figures show that even on $k = 2$ the μTC achieves almost its optimal actual performance; even when the predicted performance is most of the times better for larger k values. On the other hand, the binary partition method is prone to overfitting, especially on small datasets and small $1 - \beta$ values (i.e., small test sets). For instance, Fig. 2(b) shows the performance for NFL; please note how $\beta = 0.5$



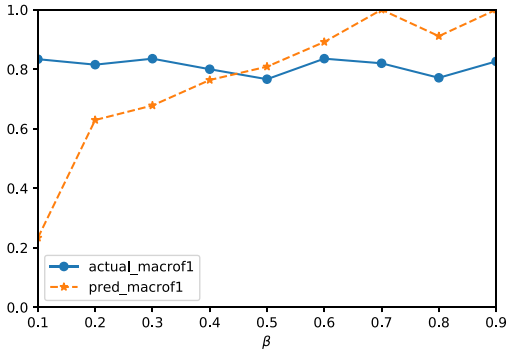
(a) Authors NFL – k-folds



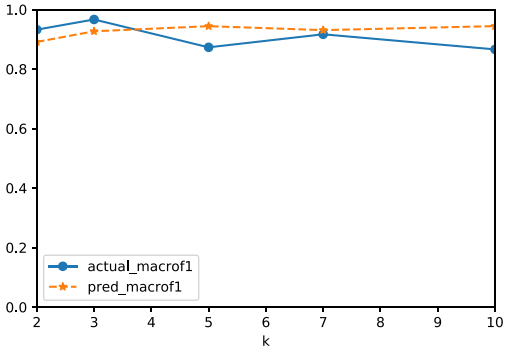
(b) Authors NFL – binary partition



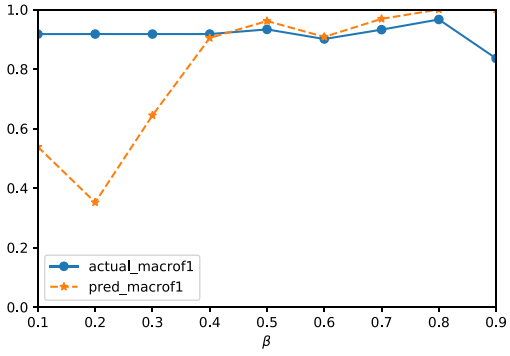
(c) Authors Business – k-folds



(d) Authors Business – binary partition



(e) Authors Cricket – k-folds

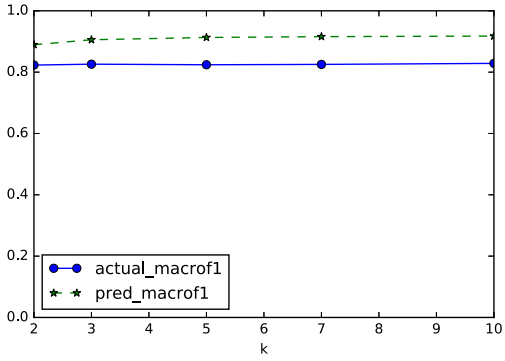


(f) Authors Cricket – binary partition

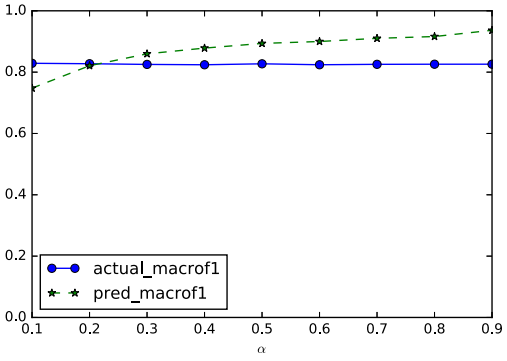
Fig. 2. The final performance in small datasets as a function of the validation's stage of the score function of μTC; we consider two validation schemes for this purpose: i) k-folds and ii) random binary partitions of sizes βn and $(1-\beta)n$, for training and testing subsets respectively.

yields to very competitive performances, i.e., higher than 0.9 for both macro-F1 and accuracy. These performances are pretty higher than those achieved by the alternatives (see Table 3); however, $\beta > 0.5$ produces low real performances, contrasting the perfect predicted performance. A similar case happens for the Business dataset, Fig. 2(d); but in this case, the actual performance is relatively stable. The behaviour of binary partition in larger dataset is less prone to overfit, like Fig. 3(b) and (d) illustrate. Nonetheless, the case of R52, Fig. 3(f), shows that the overfitting issue is still latent; however, it barely affects the actual performance since the score function is applied to a large enough test set.

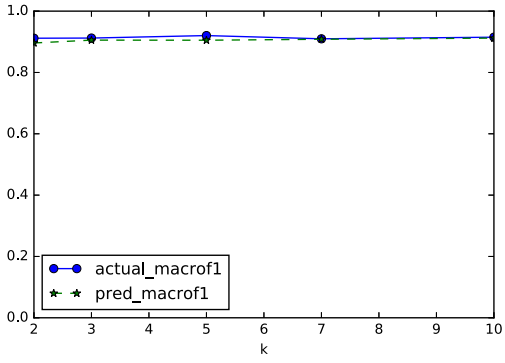
As rule of thumb, it is safe to use k-fold cross-validation to compute score in the model selection stage. We encourage the use of small k values (e.g., 2, 3 or 5) since the actual performance is relatively stable and the computational cost is kept low. Please notice that k-folds procedure introduces a factor of k to the computational cost of score, and, algorithms to solve the underlying combinatorial optimization problem need to evaluate a considerable number of configurations to achieve good results. In cases where the number of samples is pretty large, or a rapid solution is required, the binary partition method is also a good choice, especially for high $1-\beta$ values. The last setup corresponds to prepare robust score functions at the cost of reducing the train set in the model selection stage. The reduction of the training set is not a significant problem for the actual performance, as it is illustrated by experiments corresponding to binary partition performances, see Figs. 2 and 3. Please remember, at this stage, we are just selecting a proper configuration, and in a subsequent step, the final model is



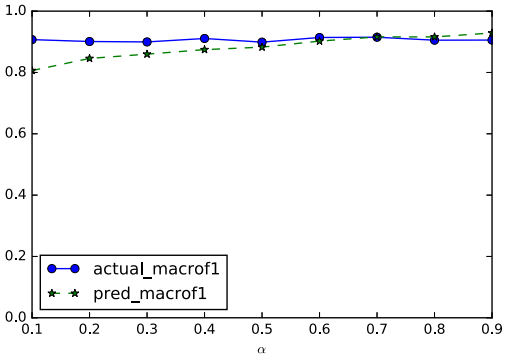
(a) News – k-folds



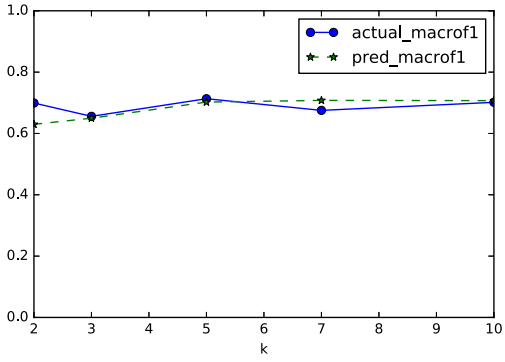
(b) News – binary partition



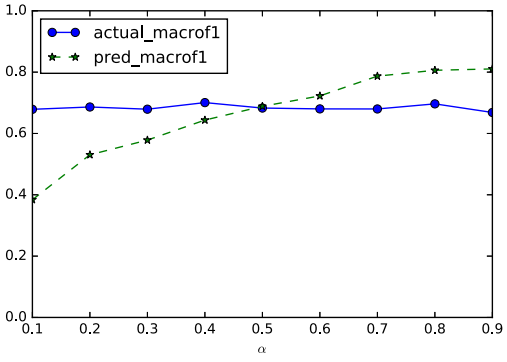
(c) WebKB – k-folds



(d) WebKB – binary partition



(e) R52 – k-folds



(f) R52 – binary partition

Fig. 3. The final performance on medium sized datasets as a function of the validation's stage of the score function of μTC; we consider two validation schemes for this purpose: i) k-folds and ii) random binary partitions of sizes βn and $(1-\beta)n$, for training and testing subsets respectively.

computed using this configuration and the entire training dataset $\mathcal{D}$.

6. Conclusions

In this work, a minimalist and global approach to text classification is proposed. Moreover, our approach was evaluated in a broad range of classification tasks such as topic classification, sentiment analysis, spam detection, and user profiling; for this matter, a total of 30 databases related with these tasks were employed. In order to evaluate the performance of our approach, the results obtained in each task were compared to the state-of-the-art methods, related to each task. Additionally, we analyze the effect of the pre-processing stage. In this experiment, we observed that our approach is competitive with the alternative methods even using the raw text as input, without a penalty in the performance; therefore, it is possible to use μTC to create text classifiers with a little knowledge of natural language processing and machine learning techniques. We also studied some simple strategies to avoid overfitting problem; we consider using a k-fold cross-validation scheme and a binary partition to perform the model selection. Based on our experimental observation, our μTC can both properly fit the dataset and speed up the construction step using small k values in cross-validation schemes and small training sets when we use binary random partitions. We also found that perform k-folds can be the preferred validation scheme on small to medium sized datasets, but very large datasets can use the binary partition scheme without a significant reduction of the performance, and also, keeping the cost the entire process low.

References

[1] E.S. Tellez, S. Miranda-Jiménez, M. Graff, D. Moctezuma, R.R. Suárez, O.S. Siordia, A simple approach to multilingual polarity classification in twitter, Pattern Recognit. Lett. (2017).

[2] A. Khan, B. Baharudin, L.H. Lee, K. Khan, A review of machine learning algorithms for text-documents classification, J.Adv.Inf.Technol. 1 (1) (2010) 4–20.

[3] S. Wold, K. Esbensen, P. Geladi, Principal component analysis, Chemom. Intell. Lab. Syst. 2 (1) (1987) 37–52.

[4] T.K. Landauer, P.W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse Process. 25 (2–3) (1998) 259–284.

[5] J.J. Rocchio, Relevance Feedback in Information Retrieval, 1971.

[6] T. Joachims, A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization, Technical Report, 1996. DTIC Document

[7] A. Cardoso-Cachopo, Improving Methods for Single-label Text Categorization, Instituto Superior Técnico, Portugal, 2007 Ph.D. thesis.

[8] I. Androutsopoulos, G. Paliouras, E. Michelakis, Learning to Filter Unsolicited Commercial e-mail, "DEMOKRITOS", National Center for Scientific Research, 2004.

[9] I. Androutsopoulos, J. Koutsias, K.V. Chandrinos, C.D. Spyropoulos, An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages, in: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, in: SIGIR '00, ACM, New York, NY, USA, 2000, pp. 160–167.

[10] F. Debole, F. Sebastiani, Supervised Term Weighting for Automated Text Categorization, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 81–97.

[11] S. Hingmire, S. Chougule, G.K. Palshikar, S. Chakraborti, Document classification by topic labeling, in: Proceedings of the 36th international ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2013, pp. 877–880.

[12] R. Cummins, C. O'Riordan, Evolving local and global weighting schemes in information retrieval, Inf. Retr. 9 (3) (2006) 311–330.

[13] H.J. Escalante, M.A. Garcia-Limon, A. Morales-Reyes, M. Graff, M.M. y Gomez, E.F. Morales, J. Martinez-Carranza, Term-weighting learning via genetic programming for text classification, Knowl. Based Syst. 83 (2015) 176–189.

[14] S. Lai, L. Xu, K. Liu, J. Zhao, Recurrent convolutional neural networks for text classification., in: AAAI, 2015, pp. 2267–2273.

[15] J. Pennington, R. Socher, C.D. Manning, Glove: global vectors for word representation, in: Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.

[16] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[17] D. Vilares, C. Gómez-Rodríguez, M.A. Alonso, Universal, unsupervised (rule-based), uncovered sentiment analysis, Knowl. Based Syst. 118 (2017) 45–55.

[18] I. Mozetič, M. Grčar, J. Smailović, Multilingual twitter sentiment classification: the role of human annotators, PLoS One 11 (5) (2016) e0155036.

[19] A.P. Lopez-Monroy, M.M. y Gomez, H.J. Escalante, L.V. nor Pineda, E. Stamatatos, Discriminative subprofile-specific representations for author profiling in social media, Knowl. Based Syst. 89 (2015) 134–147.

[20] F. Rangel, P. Rosso, M. Moshe Koppel, E. Stamatatos, G. Inches, Overview of the author profiling task at pan 2013, in: CLEF Conference on Multilingual and Multimodal Information Access Evaluation, CELCT, 2013, pp. 352–365.

[21] F. Rangel, P. Rosso, M. Potthast, B. Stein, Overview of the 5th author profiling task at pan 2017: gender and language variety identification in twitter, CLEF, 2017.

[22] A. Basile, G. Dwyer, M. Medvedeva, J. Rawee, H. Haagsma, M. Nissim, N-gram: new groningen author-profiling model, CLEF, 2017.

[23] M. Martinc, I. Å krjanec, K. Zupan, S. Pollak, Pan 2017: author profiling - gender and language variety prediction, CLEF, 2017.

[24] A. Pastor Lopez-Monroy, H.J. Escalante, M. Montes-Y-Gomez, E. Villatoro-Tello, INAOE participation at PAN'13: author profiling task, PAN 2013, CLEF, 2013. Working Notes.

[25] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C.D. Spyropoulos, P. Stamatopoulos, A memory-based approach to anti-spam filtering for mailing lists, Inf. Retr. 6 (1) (2003) 49–73.

[26] C.H. Li, J.X. Huang, Spam filtering using semantic similarity approach and adaptive {BPNN}, Neurocomputing 92 (2012) 88–97. Data Mining Applications and Case Study

[27] M. Meina, B. Celemer, M. Czoków, M. Patera, J. Pezacki., K. Brodzinska, M. Wilk, Ensemble-based classification for author profiling using various features, PAN 2013, CLEF, 2013. Working Notes.

[28] I. Androutsopoulos, G. Paliouras, E. Michelakis, Learning to Filter Unsolicited Commercial e-mail, 2004.

[29] M.S. K Santosh, R. Bansal, V. Varma, Author profiling: predicting age and gender from blogs, PAN 2013, CLEF, 2013. Working Notes

[30] M. Salameh, S. Mohammad, S. Kiritchenko, Sentiment after translation: a case-study on arabic social media posts, in: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Denver, Colorado, 2015, pp. 767–777.

[31] S.M. Mohammad, M. Salameh, S. Kiritchenko, How translation alters sentiment, J. Artif. Intell. Res. 55 (2016) 95–130.

[32] I. Androutsopoulos, J. Koutsias, K.V. Chandrinos, G. Paliouras, C.D. Spyropoulos, An evaluation of naive Bayesian anti-spam filtering, (2000). arXiv preprint cs/0006013.

[33] D.L. Olson, D. Delen, Advanced Data Mining Techniques, 1st edition, Springer Publishing Company, Incorporated, 2008.

[34] F. Sebastiani, Machine learning in automated text categorization, ACM Comput. Surv. 34 (2002) 1–47.

[35] E.K. Burke, G. Kendall, et al., Search Methodologies, Springer, 2005.

[36] R. Battiti, M. Brunato, F. Mascia, Reactive Search and Intelligent Optimization, 45, Springer Science & Business Media, 2008.

[37] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (February) (2012) 281–305.

[38] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, Liblinear: a library for large linear classification, J.Mach.Learn.Res. 9 (August) (2008) 1871–1874.

[39] S. Bird, E. Klein, E. Loper, Natural Language Processing with Python, O'Reilly Media, 2009.

[40] E.S. Tellez, S. Miranda-Jiménez, M. Graff, D. Moctezuma, Gender and language variety identification with microtc, PAN 2017, CLEF, 2017. Working Notes.