EXTENDED PRE-PROCESSING PIPELINE FOR TEXT CLASSIFICATION: ON THE ROLE OF META-FEATURES, SPARSIFICATION AND SELECTIVE SAMPLING

WASHINGTON LUIZ MIRANDA DA CUNHA

EXTENDED PRE-PROCESSING PIPELINE FOR TEXT CLASSIFICATION: ON THE ROLE OF META-FEATURES, SPARSIFICATION AND SELECTIVE SAMPLING

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Marcos André Gonçalves
Coorientador: Leonardo Chaves Dutra da Rocha

Belo Horizonte

Novembro de 2019

WASHINGTON LUIZ MIRANDA DA CUNHA

EXTENDED PRE-PROCESSING PIPELINE FOR TEXT CLASSIFICATION: ON THE ROLE OF META-FEATURES, SPARSIFICATION AND SELECTIVE SAMPLING

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Marcos André Gonçalves
Co-Advisor: Leonardo Chaves Dutra da Rocha

Belo Horizonte
November 2019

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa- CRB 6 - 1510

Cunha, Washington Luiz Miranda da.

C972e Extended pre-processing pipeline for text classification: on the role of meta-features, sparsification and selective sampling / Washington Luiz Miranda da Cunha — Belo Horizonte, 2019.
xxii, 46 f. : il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Marcos André Gonçalves
Coorientador: Leonardo Chaves Dutra da Rocha

1. Computação – Teses. 2. Aprendizado de máquina – Teses. 3. Pipelines de classificação de texto; – Teses. 4. Pré-processamento de dados – Teses. I. Orientador. II. Título.

CDU 519.6*82.10(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Extended Pre-Processing Pipeline For Text Classification: On the Role of Meta-Features, Sparsification and Selective Sampling

WASHINGTON LUIZ MIRANDA DA CUNHA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]

Prof. Marcos André Gonçalves - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]

Prof. Leonardo Chaves Dutra da Rocha - Coorientador
Departamento de Ciência da Computação - UFSJ

[illegible]

Profa. Jussara Marques de Almeida Gonçalves
Departamento de Ciência da Computação - UFMG

[illegible]

Prof. Anisio Mendes Lacerda
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 8 de Novembro de 2019.

This work is dedicated to all essential people present in my life.

Acknowledgments

First of all, I would like to express my gratitude to God for giving me health and protection all the time.

To my girlfriend Adriana, for all her help, love, affection, dedication and faith, which were fundamental for this project and for my life.

To my parents, Washington and Ivone, for encouragement, education and love.

To my brothers and friends for their abetment.

To Mr. Adilson and Mrs. Terezinha for the advices and care.

To Dona Maria Lucia for all the excitement and affection poured into me.

To my mentors and friends, Marcos and Leonardo, for all knowledge and help thats was crucial to this work.

Finally, my thanks to all those who contributed, directly or indirectly, to the development of this work.

“Quanto mais estudo, mais sinto que minha mente nisso é insaciável.”

(Ada Lovelave)

Resumo

Pipelines de classificação de texto são uma sequência de tarefas que devem ser executadas para classificar documentos em um conjunto de categorias predefinidas. A fase de pré-processamento (antes do treinamento) desses *pipelines* envolve diferentes maneiras de transformar e manipular os documentos para a próxima fase (aprendizado). Nesta dissertação, apresentamos *três* novas etapas na fase de pré-processamento dos *pipelines* de classificação de texto para melhorar a eficácia e reduzir os custos associados. A etapa de geração de *meta-features* (MFs) baseadas em distância visa reduzir a dimensionalidade da matriz termo-documento original, enquanto produz um espaço potencialmente mais informativo, o qual explora explicitamente as informações discriminativas sobre as categorias. O segundo passo é a esparsificação que visa tornar a representação do MF menos densa para reduzir os custos de treinamento. A terceira etapa é a amostragem seletiva (SS), destinada a remover linhas (documentos) da matriz obtida na etapa anterior, selecionando cuidadosamente os "melhores" documentos para a fase de aprendizado. Nossos experimentos mostram que o *pipeline* de pré-processamento estendido proposto pode obter ganhos significativos em eficácia quando comparado ao TF-IDF original (até 52 %) e às representações baseadas em *embeddings* (até 46 %), a um custo muito menor (até 9,7x mais rápido em alguns conjuntos de dados). Outra contribuição principal é uma avaliação completa e rigorosa do *trade-off* entre custo e eficácia associadas à introdução dessas novas etapas no *pipeline*.

Palavras-Chave: *pipelines* de classificação de texto; pré-processamento; *meta-features*; esparsificação; amostragem seletiva.

Abstract

Text Classification pipelines are a sequence of tasks needed to be performed to classify documents into a set of predefined categories. The pre-processing phase (before training) of these pipelines involve different ways of transforming and manipulating the documents for the next (learning) phase. In this dissertation, we introduce *three* new steps into the pre-processing phase of text classification pipelines to improve effectiveness while reducing the associated costs. The distance-based Meta-Features (MFs) generation step aims at reducing the dimensionality of the original term-document matrix while producing a potentially more informative space that explicitly exploits discriminative labeled information. The second step is a sparsification one aimed at making the MF representation less dense to reduce training costs. The third step is a selective sampling (SS) aimed at removing lines (documents) of the matrix obtained in the previous step, by carefully selecting the "best" documents for the learning phase. Our experiments show that the proposed extended pre-processing pipeline can achieve significant gains in effectiveness when compared to the original TF-IDF (up to 52%) and embedding-based representations (up to 46%), at a much lower cost (up to 9.7x faster in some datasets). Another main contribution is a thorough and rigorous evaluation of the trade-offs between cost and effectiveness associated with the introduction of these new steps into the pipeline.

Keywords: text classification pipelines; pre-processing; meta-features; sparsification; selective sampling.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Classification pipelines are a sequence of tasks needed to be performed to classify digital objects (e.g., textual documents, images) into a set of predefined categories. They are the main subject of a growing research area called AutoML [Feurer et al., 2018], whose main goal is to automatically recommend pipelines, algorithms or appropriate parameters without strong dependencies on user knowledge. In this dissertation, we are specially concerned with *pre-processing steps* of *text classification* pipelines.

Figure 1(a) illustrates a typical text classification pipeline. From the raw textual content of documents, we usually apply some pre-processing, such as lower-casing, punctuation, stopwords removal and stemming. Then, an additional feature selection step may be performed, keeping only the most informative words. After that, a $m \times n$ document-term matrix representation (or some latent term encoding) of the document is built. The most common representation of this matrix in text classification exploits the so-called TF-IDF paradigm. The major problems with the TF-IDF matrix representation have to do with its high dimensionality and sparseness. Other alternative representations that aim at producing a more compact space in terms of latent dimensions (e.g., distributional word embeddings) do exist [Mikolov et al., 2018a], but their unambiguous contribution to text classification tasks has not yet been fully established, as comparisons are not done with standard benchmarks following rigorous scientific procedures [Sculley et al., 2018] (e.g., with multiples runs to check variability and with statistical significance tests to reject the null hypothesis of equality of results)[1]. Indeed, some of our experimental results demonstrate that, when compared using proper and rigorous experimental procedures, (word) embedding-

[1]Issues regarding poor or inadequate experimental evaluation setups as well as improper or unfair baseline tuning have recently raised serious questions about the real value of complex deep learning solutions in areas related to information retrieval [Dacrema et al., 2019; Lin, 2019; Ludewig et al., 2019].

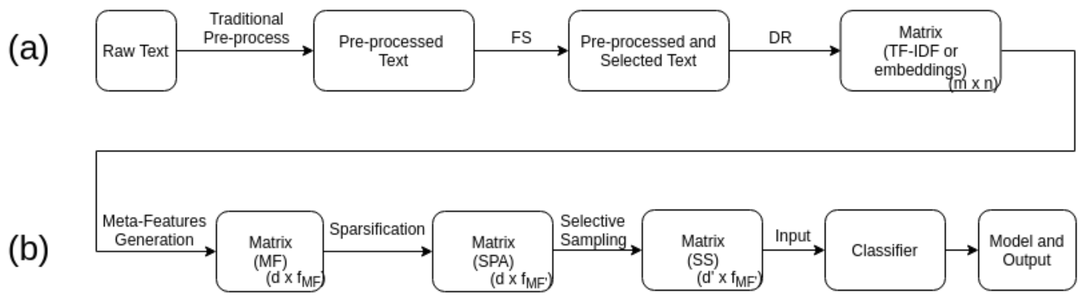


Figure 1.1: Text Classification Pipeline

based representations present no clear advantage in terms of effectiveness or cost than simpler TF-IDF representations, at least in the datasets we experimented with.

In any case, the complexity of the representation is directly associated with the costs involved in the whole classification process. This is particularly important given the development of recent techniques such as (word) embedding and deep neural networks, which require a lot of data and computational power to properly work. These methods cause large increases in the cost of training, validation and actual classification of unknown (test) instances.

It used to be the case, not in a distant past, that in text classification tasks the cost to train a model would not be a dominant factor in the choice of a particular algorithm or representation, as the training process would be run just once and in a batch mode. This scenario has changed in the last few years, mainly due to the tremendous increase in the complexity of the representations (mainly those based on word embeddings), the complexity of the models (mainly those based on deep learning) and the size of the datasets. For instance, as we shall see next, some of the most complex models (e.g., fully connected feed-forward neural networks) when compared to SVMs with standard TF-IDF representations, are sometimes 30x - 80x slower. This can make their application sometimes unfeasible for some tasks.

Although the computation power of a central processing unit (CPU) has made substantial progress in attaining a lower cost per computation and we can also use graphics processing units (GPUs) to train classification models, measuring the training performance, especially for industrial and commercial usage, remains necessary. A middle-sized company or a research group can only have a limited amount of resources it can invest in hardware improvements. Furthermore, if the accuracy is already sufficient for a given application, one may not need to run more complex models, which would require much more additional time or cost to train[2]. Finally, for some

[2]Indeed, some Industry estimates evaluate the cost of training state-of-the-art deep learning models, even pre-trained ones, such as BERT and XLNET, in hundreds of thousands of dollars in very powerful and costly computational architectures: https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/

applications in which there are rapid changes in some assumptions of the model (e.g., class distributions, term distributions, etc) or in real-time stream-based applications in which recent events may affect such assumptions, a periodical retraining is required [Guo et al., 2019], which makes the training time a non-negligible factor. In sum, *there is a tradeoff between effectiveness and cost* and this has been acknowledged by recent work [Kastrati et al., 2019; Strubell et al., 2019]. Therefore, in our work, we consider the training time as an important analysis factor and the gains in performance that can be obtained with the proposed pipeline without (or with minimal) losses in classification effectiveness.*This issue has been much neglected in the literature and constitutes one of our main contributions.*

1.1 Objectives

Turning back to our proposal, differently from generic AutoML solutions, we here are not trying to hide the complexity of configuring the classification pipeline. Instead, our concern is to manage issues related to the trade-off between classification effectiveness (e.g., high accuracy) and the costs involved to achieve it. This aligns with recent work [de Sá et al., 2017; Schoenfeld et al., 2018] that shows that by properly working on the pre-processing phase of the pipeline, one can achieve significant gains in performance, even if effectiveness is not improved. In this dissertation, we investigate issues related to the introduction of **three new steps** (Meta-Feature Generation, Sparsification and Selective Sampling - Figure 1(b)) into the standard pre-processing phase of text classification pipelines with the goal of improving effectiveness while, at the same time, reducing associated costs[3].

1.2 Contributions

The first contribution of this dissertation is the proposal of (distance-based) Meta-Feature (MF) generation as an *explicit* pre-processing step in a text classification pipeline. This step aims at reducing the dimensionality and sparsity of the original TF-IDF document-term matrix, by producing a potentially more informative denser space. Indeed, distance-based meta-features have been shown to be very effective in text classification tasks[Canuto et al., 2018]. However, they have been proposed as an end on itself, not as a means to achieve a more compact and richer representation

[3]As it will become clear later on, this sequencing corresponds to just one possible instantiation of pipelines that can be built with the proposed new steps.

in a text classification pipeline. As such, our contribution lies in re-thinking its role in a chain of data transformations before actual learning. By having this as an explicit goal, we introduce other contributions in this realm, such as: (i) investigations regarding the dimensionality and density of the MF representation; (ii) explicit studies on the effectiveness-cost trade-offs associated with the introduction of this step into the pipeline. *Those aspects have not been previously studied.*

In sum, by explicitly introducing this step into the pipeline, we aim at answering the following research questions:

- *(RQ1) What are the gains in terms of effectiveness of the MFs when compared to the original TF-IDF representation?* We revisit this question, first posed by some previous work[Canuto et al., 2018; Canuto et al., 2014], and observe some new interesting results.
- *(RQ2) What are the costs associated with this new step?*
- *(RQ3) How do the MFs compare with other alternative low-dimensional representations (e.g., word embeddings) in terms of effectiveness and incurred costs?*

The answers to these questions motivate the **second novel contribution** of this dissertation which is the introduction of a new *sparsification* step of the MF matrix. Although less dimensional and more informative (with gains, for instance, in memory consumption), the MF representation is dense, since all features have a non-zero value. This may have an impact on some classifiers' learning time, especially when applied to datasets with many classes due to the increase in density. For this reason, we propose a procedure to transform such a denser representation in a sparser one while keeping the same dimensionality. This aligns with recent work [Zamani et al., 2018] that aims at making dense latent embeddings representations sparser for the sake of performance in search tasks. As we shall see, this step results in significant reductions in cost/time, with no significant losses in effectiveness.

Finally, most of the previous related work (e.g., on feature selection) are column-oriented, but few are concerned with the lines (documents) of the design matrix. Thus, **our third novel contribution** is the introduction of a selective sampling step aimed at reducing the lines of the low-dimensional, denser matrices obtained in the previous MF step (with and without sparsification). For this, we propose to use a state-of-the-art compression-based selective sampling strategy–Cover [Silva et al., 2016]–that has been originally proposed for the learning-to-rank task but has never been used in the text classification realm. This method has several advantages towards our goals, including: (i) it is data-driven and independent of any learning framework; (ii) it is computationally cheap for dense representations and scales well to large datasets; and (iii) it is able to compress the original data in a much smaller representation without loss of information, by carefully selecting the "best" documents.

Considering the second and third contributions, the research questions we aim at answering include:

- *(RQ4) Can the more compact (SS) and less dense (SPA) MF representations reduce costs without loss of effectiveness?*
- *(RQ5) Is the previous step (MF) of creating a low-dimensional representation really necessary to apply the proposed SS technique? Or could we use it with the original TF-IDF matrix?*
- *(RQ6) Can the compact representation induced by the proposed pipeline benefit different state-of-the-art classifiers?*

Our experiments with four datasets in topic classification show that MF based representation can indeed be more effective (up to 52%) than TF-IDF (RQ1) and alternative embedding representations (RQ3) (up to 46%) at a much lower cost (RQ2 and RQ4) (4x-9.7x faster). We also show that reductions up to 430.8x in the size of the representations can be obtained with no loss in effectiveness (RQ4) by using MF, SPA and SS steps altogether. This comes with gains in the processing time of about 4-9.7 times (RQ4) over TF-IDF (which is already much faster to generate than the embedding representations). Moreover, the MF denser representation is essential to guarantee the scalability of the SS step (RQ5). Finally, we show that the generated representations are useful in several different classifiers based on completely different paradigms (RQ6).

To summarize, the main contributions of this dissertation include: (i) the study of MFs in a different role–as a step in pre-processing pipelines; (ii) rigorous comparisons with alternative representations (i.e., word embeddings) in this role; (iii) a study of MF generation impact in the whole pipeline; (iv) the proposal of a sparsification step aimed at reducing the classification related costs; (v) the proposal of a selective sampling step aimed at reducing classification costs without loss of information; (vi) the combination of all steps for a robust text classification pipeline; (vii) a thorough evaluation of effectiveness vs. efficiency trade-off associated with the introduction of those new steps.

Chapter 2

Related Work

2.1 Data Representation

Meta-features (MFs) correspond to n-dimensional document representations derived from more basic representations such as TF-IDF. The MFs' scores are obtained by functions that relate textual data with categories. Here, we focus on distance-based MFs as they have been shown to produce the best-reported results in the literature [Canuto et al., 2018]. Clustering techniques were among the first used to generate distance-based MFs [Kyriakopoulou and Kalamboukis, 2007]. In this approach, clusters represent higher-level "concepts" in the feature space and the features derived from them indicate the similarity of each example to these concepts.

Several works [Canuto et al., 2015, 2016, 2014; Gopal and Yang, 2010; Yang and Gopal, 2012] have proposed to use category centroids or kNN as the main tools to generate meta-features. These MFs differ from the previous ones derived from clusters because they explicitly capture information from the labeled set. Gopal and Yang [2010] reported good results by designing MFs that make a combined use of local information (through kNN-based features) and global information (through category centroids) in the training set. That work was extended by Yang and Gopal [2012], leveraging successful L2R retrieval algorithms over the meta-feature space for the multi-label classification problem. kNN-based meta-features are computationally expensive to generate for textual data. Canuto et al. [2014] proposed a massively parallel version of the kNN classifier for execution on manycore GPU architectures. Canuto et al. [2018] performed the most comprehensive in-depth analysis ever reported in the literature on the effectiveness of recently proposed distance-based MFs. They found that a few core meta-feature groups are responsible for achieving the best results. We here exploit the best sets of MFs found in [Canuto et al., 2018]. However, we do this as a step in a pre-processing pipeline, paying special attention to the effectiveness-efficiency trade-offs implied by this new application. *Such issues were not investigated in any previous work.*

Recent works pay attention to low-dimensional, dense textual data representations based on word embeddings, such as GloVe [Pennington et al., 2014], Word2Vec [Mikolov et al., 2013] and FastText [Mikolov et al., 2018a]. They relate to our proposal as they try to provide richer and more compact representations of textual information. Embedding models are mostly based on co-occurrence statistics of textual datasets. Words are represented as vectors so that their similarities correlate with semantic relatedness and contextualized positioning in texts (e.g., terms adjacent to a target one).

Similarly to GloVe, the unsupervised Word2Vec strategy aims at estimating the probability of two words occurring close to each other, achieved by a neural network trained with sequences of words that co-occur within a window of fixed size. Differently from distributional models, both Word2Vec and GloVe are prediction models, as they aim at predicting word occurrence. FastText [Mikolov et al., 2018a], on the other hand, learns vectors for the sub-words (ie., character n-grams) found within each word. At each training step in FastText, the mean of the target word and sub-word vectors are used for training. In experiments reported by Baroni et al. [2014], some of these representations outperformed BOW in several tasks. But in that, as well as in many other works, comparisons among the representations were not performed following rigorous and well established experimental procedures supported by statistical analyses of the results (e.g., statistical significance tests). *We investigate further this issue in our work obtaining different, somewhat surprising, results.*

One way to use embeddings for document representation is to create a centroid vector computed as the average of the vectors of all distinct words belonging to a document. In contrast to the simple centroid-based and inspired by works in computer vision, Lev et al. [2015] present a proposal based on pooling methods. The authors exploit an effective polling strategy called the Fisher Vector (FV) of single multivariate Gaussian distribution. The FV pooling representation was able to achieve great results in classification tasks and topic modeling. *We compare the MF and Fisher Vector representations in our work.*

Here, we pay special attention to the word embedding model called FastText [Mikolov et al., 2018b]. FastText learns vectors for the sub-words (ie., character n-grams) found within each word, as well as the complete word. At each training step in FastText, the mean of the target word and subword vectors are used for training. The adjustment that is calculated from the error is then used uniformly to update each of the vectors that were combined to form the target. This adds a lot of additional computation cost to the training step. The trade-off is a set of word-vectors that contain embedded sub-word information. Mikolov et al. [2018b] claim that the potential benefits of FastText are: (i) it generates better word embeddings for rare words; (ii) the usage of character embedding for downstream tasks have recently shown to boost the performance of those tasks compared to using word embedding like Word2Vec or GloVe. *FastText is one of our representation baselines.*

Tang et al. [2015] proposed a semi-supervised method, called Predictive Text Embedding (PTE), to learn document representations. PTE fills a gap of previous unsupervised methods that do not consider the labeled information when learning the representation. According to the authors, unsupervised text embeddings generalize to different tasks but have a weaker predictive power for a particular task. The method learns distributed representation of text by embedding the heterogeneous text network into a low dimensional space. This network is composed of three bipartite sub-networks: word-word, word-document, and word-label, with word vertices, shared across the three networks. *PTE is also one of our representation baselines.*

2.2 Selective Sampling

Most works in Selective Sampling for classification are related to the general area of Active Learning (AL). AL aims at choosing the most informative instances for labeling to reduce the labeling effort while maximizing the amount of useful (discriminative) information for learning a classifier. AL methods try to balance between two distinct objectives [Cheng et al., 2013]: (i) Exploration - How to select representative examples of the unlabeled set? (ii) Exploitation - How to find samples that are more useful to discriminate classes? To achieve the first objective, AL methods propose techniques to sample from all regions of the feature space to select instances that represent the diversity of the unlabeled set. The "exploitation" objective is to refine the decision border through some measure of "uncertainty". In classification tasks, the harder to classify instances are (possibly) the ones that may provide most information if labeled and added to the training set.

In this context, Long et al. [2010] propose a general AL framework based on expected loss optimization (ELO-QD). They use ensembles of learners to produce relevance scores and estimate predictive distributions for the documents in the AL set. They use a bootstrap technique that relies on a large initial labeled training set to learn the ensemble. Silva et al. [2011, 2014] proposed a lazy association rule-based active method–SSAR–to select a small training set from scratch based on their dissimilarity. The method is the state-of-the-art in L2R tasks and consists of two main steps: (i) "coverage" of the feature space with the minimum number of representative instances ("diversity" / "exploration"); and (ii) selection of "interesting" (i.e., "exploitation") instances from those remaining in the unlabeled set. However, due to its lazy nature, SSAR is extremely inefficient and does not scale to large datasets. Silva et al. [2016] proposed Cover–a selection method that does not rely on training expensive learning models to choose documents for labeling. They argue that different from traditional AL, Cover is "unsupervised"–it selects instances in one batch without any supervision. Cover interprets AL methods as compressors of the large set U of documents into a much smaller set D of documents to be labeled by annotators. The goal of Cover is to keep in D most of the information present in U. In experiments, Cover produced reductions of up to 95% in the size of the datasets while keeping effectiveness at levels similar to using the whole dataset. We will exploit Cover as the method of choice for selective sampling in the third step of our pipeline due to its strengths. Instead of using it to choose instances to label, we will use it to *reduce the amount of redundant information* in an already labeled dataset, with the goals of *reducing costs and potential noise* of the labeling process.

2.3 AutoML

The main goal of Automatic Machine Learning (AutoML) techniques is to recommend automatic pipelines, algorithms, and parameters for specific Machine Learning tasks [de Sá et al., 2017]. The goal is to automatize these tasks with many dependencies on user knowledge. Several research solutions have already been proposed for this problem such as [Kotthoff et al., 2019; Li et al., 2017; Olson and Moore, 2019] which are based on several distinct paradigms such as bayesian optimization, multi-armed bandits, and genetic programming (GP). One of the most interesting works in this area is RECIPE [de Sá et al., 2017], a GP-based approach that avoids the creation of invalid pipelines and the explosion of the search space by creating a context-free grammar that describes and limits the search space for the best pipelines. It not only avoids the creation of invalid pipelines but also accelerates considerably the search for the best solutions.

In any case, most of the aforementioned works are mostly concerned with the learning phase (e.g., classifier training and parameterization) without much attention paid to the necessary pre-processing tasks and their impact on the overall ML application. *Our work tries to cover such a gap by focusing exactly on the pre-processing* *phase.* One noticeable exception is the work described in [Schoenfeld et al., 2018]. The authors exploit an exhaustive search in the space of alternatives to analyze the effects of the pre-processing in several classification algorithms. They conclude that, in most cases, there are no significant improvements in the effectiveness, but positive effects can be observed in terms of execution time in several datasets. Here, we focus on the pre-processing phase of text classification pipelines and propose *three new steps* in the traditional pipeline. Differently from previous works, our focus is on *maximizing the effectiveness-efficiency trade-offs.* As we shall see, we find that both objectives can be maximized at the same time, sometimes by large margins, using our solutions.

2.4 Effectiveness vs. Efficiency trade-off

The effectiveness vs. efficiency trade-off is the object of study in a few works [Capannini et al., 2016; Dogan and Tanrikulu, 2013; Kastrati et al., 2019; Akhila et al., 2014]. Capannini et al. [2016] analyze the impact of training parameters on effectiveness and on efficiency (run time) of learning-to-rank models. In that work, the authors proposed a methodology to find the most effective ranker given a time budget. They concluded that there is no overall best algorithm for the learning-to-rank task which excels in both objectives (effectiveness and efficiency).

The work reported in [Dogan and Tanrikulu, 2013] is one of the first to explicitly take into account the efficiency(time)-accuracy trade-off in the study of classification methods. In that study, the authors compared three types of features on multiple datasets: (i) the original features; (ii) discretized features; (iii) best features chosen after applying principal component analysis to the original features. The authors also conducted an extensive experiment to analyze and identify strong relationships between dataset characteristics and the different type of features, with regard to their impact on accuracy and efficiency. Akhila et al. [2014] present a comparative study of four traditional classifications algorithms on the health problem, especially on a dataset about diabetes in women. The analyzed algorithms were Naive Bayes, MultiLayer Perceptron (MLP), SVM and C4.5. The evaluation criteria were execution time for training and classification accuracy. Finally, in [Kastrati et al., 2019], an analysis of the impact of a semantically enhanced representation, using manually created ontologies, in deep learners as well as in traditional classifiers is presented. Though the authors report the complexity of the deep learners in terms of number of parameters, the experimental comparison is limited to only accuracy (time is not an evaluation metric) in only one dataset with only one ontology and without proper statistical validation of the results. As the technique is very specific and the evaluation is limited, the results are hard to generalize.

Differently from those works, we propose an extended text classification pipeline that explores the use of meta-features in place of original bag-of-words representation and the use of a more compact and noisy-less subset of the training set to achieve gains in both effectiveness and efficiency in the processing of the whole pipeline.

Chapter 3

Extended Text Classification Pipeline

Text classification pipelines are defined as necessary steps to classify texts into predefined classes. Figure 1(a) illustrates a classical pipeline. Traditionally, classification pipelines are composed by four steps: (i) Pre-processing; (ii) Feature Selection; (iii) Data Representation; (iv) and Classification. In the Pre-processing step, different strategies may be applied to the raw text, such as lower-casing, punctuation, stopwords removal and stemming. Next, in order to consider just the most discriminative words for the classification task, Feature Selection algorithms [Viegas et al., 2018] may be applied to reduce dimensionality. The last step before classification consists of transforming the data into a representation that can be interpretable by classification algorithms. The most common representation is the $m \times n$ (terms) TF-IDF document-term matrix. We should note that the pipeline configuration is dynamic, each step may be instantiated using different methods, or, even omitted as a whole.

In this work, we introduce three new pre-processing steps in traditional classification pipelines: (i) MetaFeatures; (ii) Sparsification; and (iii) Selective Sampling. In the next sections, we will detail each of them.

3.1 Distance-Based Meta-Feature Generation

Recent work [Canuto et al., 2015, 2016; Yang and Gopal, 2012] proposed to collect information from distances between documents to improve the encoding text data representations. We follow this recent trend, using distance-based MFs, i.e., features derived from the original TF-IDF representations and from distances among labeled documents, as an additional pre-processing step for text classification. MFs have been studied before, but not as a means to achieve enhanced compact representations towards maximizing the effectiveness-efficiency trade-off. To achieve this goal, we introduce two new steps in the text classification pipeline: *sparsification* and *selective sampling.*

The meta-feature generation relies on a previous representation of textual data. As several pipelines for text classification, we assume that the textual data is previously encoded with TF-IDF scores [Salton and Buckley, 1988][1], which is a high dimensional representation. In this scenario, the inclusion of MFs in the pipeline can bring significant reductions on dimensionality, since distance-based MFs only represent the distance relationships between a document and other labeled documents instead of a representation of the original text of documents.

Formally, let $\mathcal{X}$ denote the TF-IDF feature space, and $\mathcal{C}$ the set of possible classes for each document $\vec{x_i} \in \mathcal{X}$. Given a set of labeled documents $\mathcal{D}_{train} = \{(x_i, c_i) \in \mathcal{X} \times \mathcal{C}\}|_{i=1}^{n}$, our goal is to generate distance-based metafeatures for an arbitrary document $\vec{x_f} \in \mathcal{X}$ using distance relationships between $\vec{x_f}$ and documents of $\mathcal{D}_{train}$. A vector of meta-features m_f is expressed as the concatenation of the sub-vectors below, which are defined for each example $x_f \in \mathcal{X}$ as:

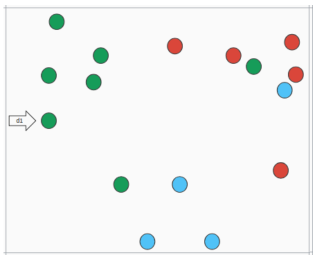
- $\vec{v}_{\vec{x}_f}^{kNN} = [dist(\vec{x}_{ij}, \vec{x}_f)]$: A $|\mathcal{C}|k$-dimensional vector produced by considering the k nearest neighbors of class $c_j \in \mathcal{C}$ to the target vector x_f. $\vec{x}_{ij}$ is the i-th ($i \leq k$) nearest neighbor to $\vec{x}_f$, and $dist(\vec{x}_{ij}, \vec{x}_f)$ is the (L2 or Cosine) distance score between them. A k-dimensional meta-level feature vector is generated for each class, given x_f. These meta-features compose the group of kNN MFs.

- $\vec{v}_{\vec{x}_f}^{Cent} = [dist(\vec{x}_j, \vec{x}_f)]$: A $|\mathcal{C}|$-dimensional vector where $\vec{x}_j$ is the centroid of category $c_j \in \mathcal{C}$. This vector contains the (L2 or Cosine) distance score between the document x_f and the centroid of c_j. These meta-features compose the group of Centroid MFs.

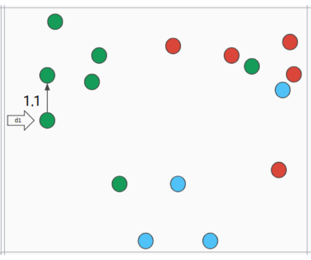
Figures 3.1 and 3.2 illustrate how to obtain kNN and Centroid MFs, respectively. Figure 3.1 (a) is the representation of a TF-IDF matrix of instances considering just two dimensions. The colors of elements in the graph represent the classes to which they belong (i.e. green represents class 1, red, class 2, and blue, class 3). The first step is to generate the MFs for the training set. For this, considering a training document d_1 of class 1, we calculate its distance (in the case, Euclidian Distance) for all other documents in class 1. Then, we select the k documents closest to d_1 (in the case, $k = 2$ – Figures 3.1 (b) and (c)) and generate the MFs mf_1 and mf_2 (Figure 3.1 (h)), in which the values correspond to the calculated distances. This process is repetead for

[1]MF-Based representations built from word embeddings may be considered in the future.

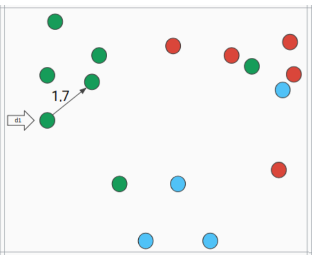
classes 2 (Figures 3.1 (d) and (e)) and 3 (Figures 3.1 (f) and (g)), generating MFs mf_3, mf_4, mf_5 and mf_6. This process is performed for all traning set. In the case of test set, the process is the same, however the class of each test document is unknow, obvially.



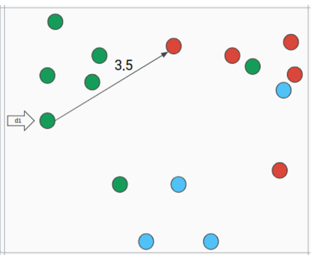
(a) Visual Data Repr.



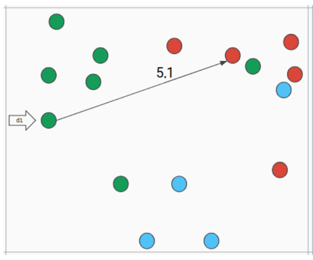
(b) Step 1



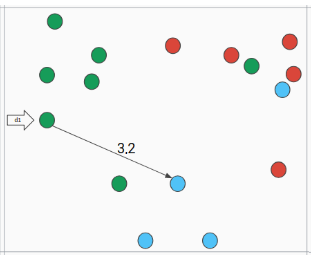
(c) Step 2



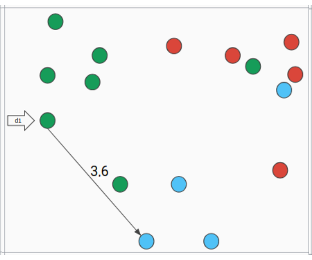
(d) Step 3



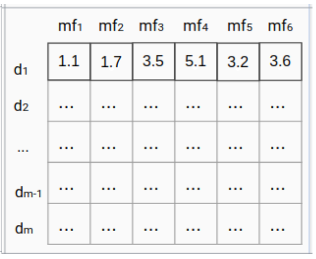
(e) Step 4



(f) Step 5



(g) Step 6

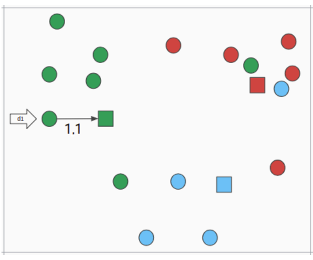


	mf_1	mf_2	mf_3	mf_4	mf_5	mf_6
d_1	1.1	1.7	3.5	5.1	3.2	3.6
d_2	...	...	...	...	...	...
...	...	...	...	...	...	...
d_{m-1}	...	...	...	...	...	...
d_m	...	...	...	...	...	...

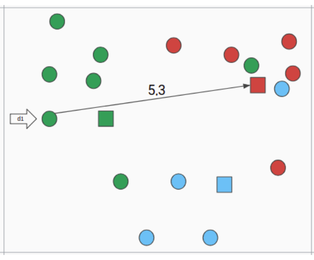
(h) Distance matrix

Figure 3.1: kNN MFs Example

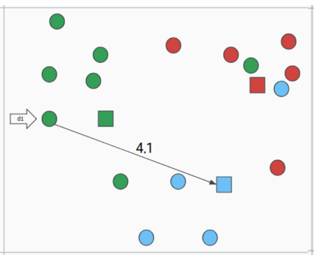
Regarding the Centroid MFs (Figure 3.2), first it is necessary to calculate the centroid for all classes, which are represented by squares. One more time, considering a training document d_1 of class 1, we calculate its distance (i.e. Euclidian Distance) for all centroids (Figures 3.2 (a), (b) e (c)), generating the MFs mf_1, mf_2 and mf_3 (Figure 3.2 (d)), in which the values correspond to the calculated distances. This process is also performed for all traning set. For test set, the process is the same, however the class of each test document is unknow.



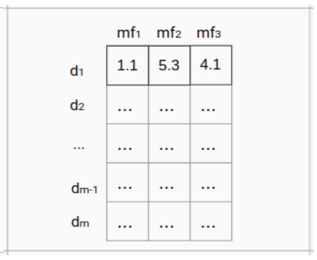
(a) Distance of d1 to centroid 1



(b) Distance of d1 to centroid 2



(c) Distance of d1 to centroid 3



	mf_1	mf_2	mf_3
d_1	1.1	5.3	4.1
d_2	...	...	...
...	...	...	...
d_{m-1}	...	...	...
d_m	...	...	...

(d) Computed distance matrix

Figure 3.2: Centroid MFs Example

The MFs we exploit here are the most effective ones across several datasets according to [Canuto et al., 2018], and also the cheapest to compute. Table 3.1

provides names for referencing the previously described meta-features according to the used distance measures and meta-feature vectors.

Meta-Feature Group	Description
CosKnn	Vector $\vec{v}_{\vec{x}_f}^{kNN}$ generated with the cosine similarity.
L2Knn	Vector $\vec{v}_{\vec{x}_f}^{kNN}$ generated with the L2 distance.
CosCent	Vector $\vec{v}_{\vec{x}_f}^{Cent}$ generated with the cosine similarity.
L2Cent	Vector $\vec{v}_{\vec{x}_f}^{Cent}$ generated with the L2 distance.
All	Concatenation of CosCent, L2Cent, CosKnn and L2Knn.
Cent	Concatenation of CosCent and L2Cent

Table 3.1: Given names for meta-features.

These meta-features make a combined use of the local information ($CosKnn$ and $L2Knn$) and global information ($L2Cent$ and $CosCent$). More specifically, each test example is directly compared to a set of nearest labeled examples and category centroids, which are assumed to be enough to effectively characterize and discriminate categories. The intuition behind these meta-features consists of the assumption that if the distances between an example to the nearest neighbors belonging to the category c (and its corresponding centroid) are small, then the example is likely to belong to c.

The MF step induces a low-dimensional space, but a potentially much denser one, since now all the features have some existing information (non-zero value) relating one instance to the defined classes. The drawback is the potential negative impact in learning runtime as all features have now non-zero values and cannot be ignored by the classifiers' optimization process in most implementations. To deal wth this, we introduce the sparsification step, discussed next.

3.2 Median-Based Sparsification

The sparsification (SPA) step aims at increasing the sparsity of the meta-feature vectors by zeroing the less informative similarity values, ultimately improving the learning runtime. Particularly, classifiers such as SVM and Random Forests can ignore such zeroed values when learning decision rules using sparse data representations.

The process is described in Algorithm 1, which requires as input the training and the test sets (i.e., $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$), represented with meta-features from the set of meta-feature groups. More specifically, each training or test document d is represented by a vector of meta-features $\vec{m_d}$, which is the concatenation of r sub-vectors $[m_d^1, m_d^2, ..., m_d^r]$, and each sub-vector m_d^g corresponds to a meta-feature group $g \in \{CosKnn, CosCent, L2Knn, L2Cent\}$. For each meta-feature group g, the

Algorithm 1 Sparsification Algorithm

```
Require: Training and test sets $\mathcal{D}_{train}$, $\mathcal{D}_{test}$ in meta-feature space.
Ensure: Sparsification of $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$.
1: for each meta-feature group $g$ do
2:     $g_{median} \leftarrow median(\mathcal{D}_{train}, g)$
3:     for each document $d \in \mathcal{D}_{train} \cup \mathcal{D}_{test}$ do
4:         $m_d^g \leftarrow$ meta-features from group $g$ in document $d$.
5:         for each meta-feature $i$ in $m_d^g$ do
6:             if $m_d^g[i] < g_{median}$ then
7:                 $m_d^g[i] \leftarrow 0$
```

algorithm calls the $median()$ function to compute the median value g_{median} for the set formed by the union of all values in sub-vector m_d^g in all documents in $\mathcal{D}_{train}$ (lines 1-2). Once the median value g_{median} of meta-feature group g has been computed, the algorithm compares the value of each group g meta-feature in each document d of the training or test set with g_{median}. If the meta-feature value is smaller than the median value, it receives zero (lines 3-5).

The running time complexity of Algorithm 1 is $\Theta(nMF(|\mathcal{D}_{train}| + |\mathcal{D}_{test}|))$, where nMF is the number of meta-features in the training and test sets. This bound in running time is due to the fact that for each meta-feature in each group g, the algorithm processes all documents in both training and test sets. The training set is processed twice[2] and each document in the test set once.

Depending on the number of groups of meta-features used, the total number of meta-features nMF may vary from $|\mathcal{C}|$, when only one (the smallest) meta-feature group is used, to the maximum $2(|\mathcal{C}| * k + k)$, when all groups of meta-features are utilized. As we show in Section 4.2.2, the running time of this sparsification process is very fast in all experimented datasets and does contribute significantly to reduce the overall running time of the whole pipeline.

3.3 Selective Sampling

In this pre-processing step, we apply the Cover selective sampling method [Silva et al., 2016] to obtain a reduced yet representative subset of the training examples. Cover was originally proposed for L2R problems. We adapt its use for text classification by applying it to document representations instead of pairs (query,document). In this case, the same premisses that work for L2R are valid for classification, such as the possibility of computing distances between documents and the use of a minimum

[2]Once for computing the median and another to decide whether the meta-feature m_d of a training document d must be zeroed.

spanning tree to summarize the dataset. Cover is computationally much cheaper than any compared alternative, scaling to larger collections while retaining most information from the original dataset, serving well as a pre-processing step.

We should stress that in the present scenario we are assuming that we receive a set of labeled data for training and our goal is to reduce it for performance reasons. In this case, Cover is used to obtain a smaller and more informative training set. Formally, let $\mathcal{D}$ be the training set in some feature space (e.g., TF-IDF or MF) from which we want to obtain a representative subset $\mathcal{D}'$. Cover assumes that documents in $\mathcal{D}$ that are "close enough" carry similar information. Thus, it groups "close enough" documents into <u>clusters of indistinguishable elements</u> in terms of the information they carry and then selects one representative from each cluster to generate the set $\mathcal{D}'$.

Let $d(\cdot,\cdot)$ be the Euclidean distance (ED) between two documents in the (normalized) feature space. Given a non-negative real value λ, the <u>cluster $\mathcal{C}_i \subseteq \mathcal{D}$ of indistinguishable elements</u> (CIE) containing $d_i \in \mathcal{D}$ is recursively defined as follows: (1) $d_i \in \mathcal{C}_i$; and (2) if $d' \in \mathcal{C}_i$ and $d(d', d) \leq \lambda$, then $d \in \mathcal{C}_i$. Given a value λ, any $d \in \mathcal{C}_i$ will yield the same cluster, and hence any element of a cluster can be used as a seed for the whole cluster. Let $G = (\mathcal{D}, E)$ be the complete graph whose vertices are the elements in $\mathcal{D}$, and each edge $e \in E$, connecting any two vertices, has weight equal to the ED between the two corresponding elements in the feature space. For a fixed value of λ, the definition of CIE induces a subgraph $G_\lambda = (\mathcal{D}, E_\lambda)$ of G where the set of edges $E_\lambda \subseteq E$ contains only those edges from E with weights at most λ. Thus, each connected component in the graph G_λ corresponds to a CIE.

For our purposes, the relevant edges of the induced subgraph G_λ are those strictly necessary to identify the connected components. A subset of the edges E_λ is redundant if, and only if, there is more than one distinct path between a pair of vertices in a connected component. This, in turn, is the definition of cycles in an undirected graph. By breaking all the cycles, we end up with a spanning tree of the cluster and, moreover, if we systematically remove the edges with the greatest weights of these cycles, our graph becomes the Minimum Spanning Tree (MST) of the component. Since the weights of the edges in graph $G = (\mathcal{D}, E)$ correspond to the EDs between pairs of elements in $\mathcal{D}$, our problem of identifying all clusters connected by edges of weights at most λ can be reduced to the problem of finding the Euclidean Minimum Spanning Tree (EMST) of G and then eliminating the longest edge iteratively until reach the desired number of clusters. The EMST can be computed efficiently with the Dual-Tree Boruvka [March et al., 2010] (DTB) algorithm. DBT solves this problem in $O(|\mathcal{D}| \log |\mathcal{D}|\ \alpha(|\mathcal{D}|))$ time, where $\alpha(|\mathcal{D}|)$ is the inverse Ackermann function, which has an extremely low growth-rate (e.g., $\alpha(10^{80}) \leq 4$).

Algorithm 2 Cover Algorithm

```
Require: The training set $\mathcal{D}$, number of desired samples $s$
Ensure: a representative subset $\mathcal{D}' \cup \mathcal{D}$, such that $|\mathcal{D}'| = s$.
 1: $n \leftarrow |\mathcal{D}|$
 2: $\mathcal{M} \leftarrow DualTreeBoruvka(\mathcal{D})$
 3: $G = (\mathcal{D}, E = \emptyset)$
 4: for all $i \in \{1, \dots, n-s\}$ do
 5:     $G.E \leftarrow G.E \cup \{M_i\}$
 6: $\mathcal{D}' = \emptyset$
 7: for all $i \in \{1, \dots, s\}$ do
 8:     $\mathcal{C}_i \leftarrow i\text{-}thCluster(G, i)$
 9:     $c_i \leftarrow GeometricCentroid(\mathcal{C}_i)$
10:     $p_i \leftarrow NearestNeighbour(\mathcal{C}_i, c_i)$
11:     $\mathcal{D}' \leftarrow \mathcal{D}' \cup \{p_i\}$
12: Return $\mathcal{D}'$
```

Algorithm 2 describes the Cover method. It receives as input the original training set $\mathcal{D}$ and the size of the expected reduced training set s. It produces as output the set $\mathcal{D}'$, where $|\mathcal{D}'| = s$. First, Cover executes the DTB algorithm to obtain the EMST for all points in $\mathcal{D}$ (Line 2). The set of edges in EMST, in ascending order of distances, is stored in set $\mathcal{M}$. For every element of the edge set, we can tell exactly how many document pairs will be selected for labeling if we include all edges up to it (e.g., if we have n points and include the first s edges of $\mathcal{M}$, we will have exactly $n - s$ clusters to choose from–Lines 4-6). Then, Cover computes the centroids c_i for each cluster $\mathcal{C}_i$ (Line 9) and determines the element p_i in $\mathcal{C}_i$ which is the nearest neighbor of c_i (Line 10). This element is then inserted in the output set $\mathcal{D}'$ (Lines 11-12).

The complexity of the DBT does not include the time to compute the ED between a pair of documents which is linear in the number of features composing each document. In the case of BOW representation of the dataset (using TF-IDF), this time complexity is $O(|V|)$, where V is the dataset's vocabulary. In the case where features are the MFs, the time to compute the ED is $\Theta(nMF)$, where nMF is the number of meta-features used[3]. These differences in the size of the feature space can strongly influence the execution time of DBT in the same collection represented with different types of features (BOW or MF). The greatest differences (in favor of MFs) occur in datasets where the number of meta-features nMF is much smaller than the size of V.

[3]See the discussion about the intervals for nMF in the last paragraph of Section 3.2.

3.4 Summary

In Figure 3.3 we present a summary of the three new pre-processing steps proposed to be used along with traditional classification pipelines. Each box represent a step. The continuous arrows indicate the order of application of the methods, while the dashed arrows correspond to the possible combinations that can be employed. Note that each box contains the step name as well as the instantiation proposed in this work. In the the next section, we evaluate the impact of instantiating each one of these steps, as well the impact of their joint combinations in the overall text classification process.

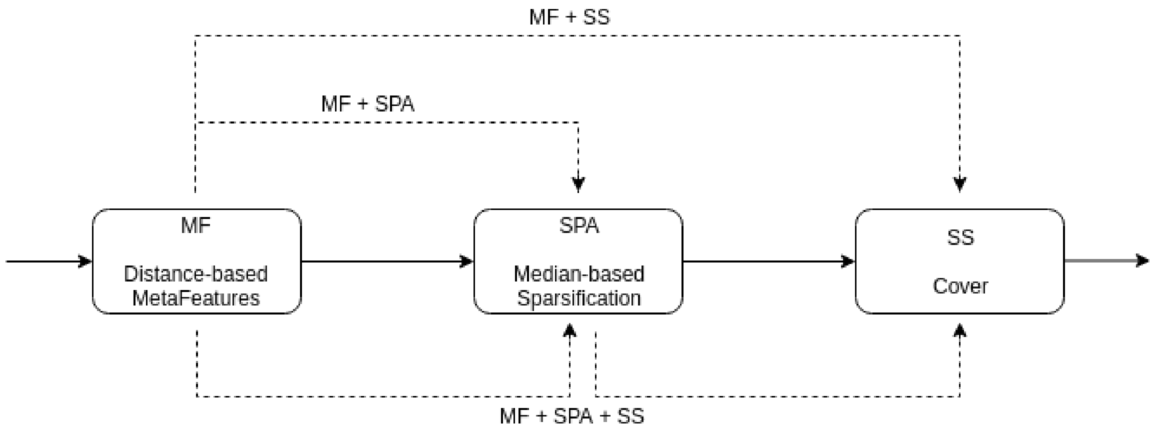


Figure 3.3: The three new pre-processing steps that extend traditional classification pipelines and their possible combinations.

Chapter 4

Experiments

4.1 Experimental Setup

4.1.1 Datasets and Notation

We consider the four real-world textual datasets used in [Canuto et al., 2018]: (i) WebKB, with $8,199$ documents (i.e. Web pages), collected from Computer Science departments of four universities, classified into 7 classes (such as student, faculty, course and project web pages, etc.); (ii) 20NewsGroup, with $18,846$ newsgroup documents, partitioned almost evenly across 20 different newsgroups categories; (iii) ACM, a subset of the ACM Digital Library that contains $24,897$ articles related to Computer Science, classified in 11 classes, according to the first level of the ACM taxonomy; (iv) Reuters, classical text dataset, composed of $13,327$ news articles collected and classified across 90 categories. Table 4.1 provides a brief characterization of the reference datasets, reporting the number of features (words), documents, the mean number of words per document (density) and the number of classes.

Dataset	**#Features**	**#Documents**	**Density**	**#Classes**
WebKB	23,090	8,199	101.6	7
20NewsGroup (20NG)	103,392	18,846	83.2	20
ACM	52,083	24,897	28.5	11
Reuters90	28,593	13,327	57.6	90

Table 4.1: Dataset characteristics

Regarding notation, $MF(Cent)$ corresponds to the joint use of CosCent and L2Cent centroid-based MF groups. Since such groups are the cheapest to compute, we always use them together in our experiments. $MF(Cent + CosKnn)$ corresponds to the Centroid-based and CosKnn Groups while in $MF(All)$ we use all groups, including $MF(L2Knn)$.

4.1.2 Evaluation Metrics, Methods and Parameterization

We evaluate the effectiveness of our proposal with Micro Averaged F1 (MicroF1) and Macro Averaged F1 (MacroF1) [Sokolova and Lapalme, 2009]. While MicroF1 measures the classification effectiveness overall decisions (i.e., the pooled contingency tables of all classes), MacroF1 measures the classification effectiveness for each individual class, averaging them.

All experiments were executed using a 10-fold cross-validation procedure. Parameters were set via cross-validation on the training set and the effectiveness of the algorithms running with distinct types of features was measured in the test partition. We adopted the LIBLINEAR [Fan et al., 2008] implementation of the SVM classifier, as it still is one of the best text classifiers capable of dealing with both high and low dimensional representations in large datasets. The regularization parameter was chosen among eleven values from 2^{-5} to 2^{15} using the validation sets. Experiments were run on an Intel superscript registered Core i7-5820K, running at 3.30GHz, with 32Gb RAM.

For feature selection (**FS**), we consider the importance score of Random Forests *applied to the original features (TF-IDF)* as the method of choice [Louppe et al., 2013], at a reduction rate of 30%, established again in preliminary experiments in the validation sets. Those numbers were based on the largest reduction possible without hurting effectiveness. For sparsification, a rate of 50% was established as the best choice. As for selective sampling, we used a reduction regarding the number of documents of 66% (2/3) in WebKB and 20NG, 33% (1/3) in ACM and 50% (1/2) in Reuters90. Those values were set based on results in the validation sets that produced the best trade-offs between effectiveness and efficiency.

To compare the average results on our cross-validation experiments, we assess the statistical significance employing a paired t-test with 95% confidence and Bonferroni correction to account for multiple tests. Some consider the Bonferroni correction too conservative. To account for that, in our analyses we also consider the Friedman-Nemenyi-Test [Zar, 2007] for multiple comparisons of mean rank sums. The Friedman test is used to compare multiple methods (in our context, data representations) considering different datasets. The test procedure is as follows: for each fold of a dataset in the cross-fold validation, for all datasets, the best performing method is ranked first (1), the second-best gets rank 2, and so on. The methods are compared based on the average of the assigned ranking across all folds. The $H0$ hypothesis (null-hypothesis) considers that all methods have the same average ranking position for a predefined p-value (i.e. 0.05). Nemenyi Post-Hoc is a procedure for pairwise rank comparison. To identify methods that perform better than others, we calculate the difference of the average assigned ranking, in our case calculated by Friedman, between each pair of methods and its standard error. Considering the results of a pair of methods, if the interval between their difference plus standard error and their difference minus standard error contains the value 0, their results are considered equivalent. We perform an analysis for each specific dataset (*within dataset*), considering the folded cross-validation, and a general analysis, considering all datasets (*cross-dataset*).

These tests assure that the best results are statistically superior to others or are clearly marked as ties. Finally, we report the execution time of each step aiming at analyzing the effectiveness-efficiency trade-offs in all steps of the proposed pipeline. The metric is the overall time in seconds (average of 10 folds).

Next, we present the results of all variants of the proposed pipeline and baselines, including: (**i**) Original TF-IDF representation; (**ii**) Feature selection applied to TF-IDF (FS); (**iii**) MF only (3 variants)–$MF(Cent)$, $MF(Cent + CosKnn)$ and $MF(All)$; (**iv**) Meta-features followed by selective sampling ($MF(*)+SS$)–3 variants; (**v**) Meta-features followed by sparsification of the MF matrix ($MF(*) + SPA$)–3 variants); (**vi**) complete pipeline, e.g., MF generation, followed by sparsification and selective sampling ($MF(*) + SPA + SS$)–3 variants. The overall best results for one dataset and one metric are highlighted in bold. If there are statistical ties for the best results, all ties are marked in bold. We use the symbols ▼, ▲, and ● to denote, respectively, statistical inferiority, superiority, and ties of the proposed variants concerning the TF-IDF representation in terms of effectiveness.

Before we proceed with our analyses, we point out that we only show the results of the feature selection (FS) step applied to the original TF-IDF representation. The application of FS before the generation of the MF representations produced losses in terms of effectiveness in most cases, with practically no gains in efficiency. For these reasons and the sake of simplicity and space, we do not present the results of the FS + MF variants.

4.2 Experimental Results

4.2.1 Effectiveness Analysis (RQ1)

We can see in Table 4.2 that the two most effective representations are those that consider the Knn-based meta-features (i.e., $MF(All)$ and $MF(Cent+CosKnn)$). These two groups produce the overall best results in all datasets considering both MicroF1 and MacroF1 (tying with TF-IDF only in WebKB, but never losing). For instance, $MF(All)$ achieves impressive results in Reuters90, gains of up to 52% over TF-IDF in

	WebKB			20NG		
	MicroF1	MacroF1	Time(s)	MicroF1	MacroF1	Time(s)
TFIDF	**82.26(0.9)**	**72.90(2.1)**	627.8	89.23(0.7)	89.07(0.7)	2,957.4
FS	**82.79(1.0)** ●	**73.41(2.4)** ●	518.7	88.82(0.7) ●	88.64(0.8) ●	2,490.7
MF(Cent)	78.29(1.1) ▼	68.84(2.5) ▼	46.3	83.56(0.5) ▼	83.12(0.6) ▼	465.0
MF(Cent) + SS	78.43(1.1) ▼	68.56(2.7) ▼	25.4	83.14(0.4) ▼	82.72(0.5) ▼	153.6
MF(Cent) + SPA	78.33(1.0) ▼	68.84(2.5) ▼	34.9	82.94(0.6) ▼	82.51(0.5) ▼	316.2
MF(Cent) + SPA + SS	78.37(1.1) ▼	68.05(1.9) ▼	21.6	82.77(0.4) ▼	82.35(0.5) ▼	149.5
MF(Cent+CosKnn)	**81.49(0.8)** ●	**71.51(2.4)** ●	372.4	**91.00(0.5)** ▲	**90.84(0.6)** ▲	1,302.1
MF(Cent+CosKnn)+SS	**80.66(1.0)** ●	**70.59(2.5)** ●	191.0	**90.60(0.5)** ▲	**90.48(0.5)** ▲	382.2
MF(Cent+CosKnn)+SPA	**81.22(0.9)** ●	**71.78(2.1)** ●	247.8	**90.77(0.5)** ▲	**90.63(0.5)** ▲	719.8
MF(Cent+CosKnn)+SPA+SS	**80.83(0.9)** ●	**70.71(2.1)** ●	129.9	**90.36(0.5)** ▲	**90.21(0.6)** ▲	305.2
MF(All)	**80.91(1.0)** ●	**71.62(2.4)** ●	644.0	**90.88(0.5)** ▲	**90.74(0.6)** ▲	2171.7
MF(All) + SS	**80.29(1.1)** ●	**69.87(2.4)** ●	323.0	**90.61(0.4)** ▲	**90.48(0.4)** ▲	653.0
MF(All) + SPA	**81.03(0.9)** ●	**71.59(1.7)** ●	387.5	**90.78(0.4)** ▲	**90.64(0.5)** ▲	1641.6
MF(All) + SPA + SS	**80.55(1.1)** ●	**70.51(2.1)** ●	198.2	**90.25(0.4)** ▲	**90.10(0.5)** ▲	568.7
	ACM			Reuters90		
	MicroF1	MacroF1	Time(s)	MicroF1	MacroF1	Time(s)
TFIDF	77.76(0.4)	67.95(0.7)	1,883.4	73.31(1.0)	31.86(3.2)	1,643.3
FS	77.41(0.4) ●	67.25(1.4) ●	1,501.6	73.54(1.1) ●	31.61(3.0) ●	1,317.4
MF(Cent)	72.55(0.9) ▼	62.91(1.2) ▼	727.0	72.71(0.9) ●	36.20(2.7) ▲	1,086.9
MF(Cent) + SS	72.02(0.8) ▼	62.08(1.2) ▼	54.0	73.07(1.1) ●	35.98(2.1) ▲	254.0
MF(Cent) + SPA	72.56(0.8) ▼	62.92(1.2) ▼	390.7	70.74(1.4) ●	34.24(2.6) ▲	871.8
MF(Cent) + SPA + SS	70.02(0.8) ▼	62.08(1.2) ▼	52.8	71.58(1.3) ●	34.57(2.1) ▲	191.8
MF(Cent+CosKnn)	**79.86(0.4)** ▲	**70.33(1.0)** ▲	4,105.5	78.77(0.9) ▲	**44.34(2.6)** ▲	4,027.0
MF(Cent+CosKnn)+SS	**79.28(0.5)** ▲	**69.83(1.3)** ▲	826.4	80.05(0.7) ▲	**45.53(2.8)** ▲	1,155.5
MF(Cent+CosKnn)+SPA	**79.99(0.4)** ▲	**70.47(1.0)** ▲	2,632.3	78.58(1.0) ▲	**44.13(2.5)** ▲	2,558.7
MF(Cent+CosKnn)+SPA+SS	**79.29(0.6)** ▲	**69.37(1.8)** ▲	678.8	78.26(1.0) ▲	**44.35(3.0)** ▲	750.7
MF(All)	**79.39(0.4)** ▲	**69.33(1.2)** ▲	8,075.0	**82.32(0.7)** ▲	**48.42(2.6)** ▲	7,426.1
MF(All) + SS	**79.71(0.7)** ▲	**69.57(1.4)** ▲	1,131.1	**82.56(0.8)** ▲	**48.74(2.8)** ▲	2,018.1
MF(All) + SPA	**79.39(0.5)** ▲	**69.47(1.2)** ▲	6,364.1	**81.86(1.0)** ▲	**48.33(3.3)** ▲	4,356.8
MF(All) + SPA + SS	**79.45(0.5)** ▲	**69.17(1.4)** ▲	1,063.5	**82.43(0.7)** ▲	**48.55(2.7)** ▲	1,201.2

Table 4.2: Effectiveness and Efficiency Results of All Variants of the Proposed Pipeline by t-test with Bonferroni correction.

MacroF1. $MF(Cent{+}CosKnn)$ also produces large improvements in this dataset with a tie in Macrof1 with $MF(All)$. Also notice that in all datasets, with both metrics, all versions that employ data reduction (i.e., that use SPA and/or SS) do not suffer from any statistically significant loss with regard to their "full" version. This is is important evidence for answering *RQ*4, as smaller representations are usually more efficient. We will further elaborate on this later on.

As mentioned, we also conducted an analysis with Friedman and Nemenyi post-hoc tests. As explained, the Friedman test is a non parametric alternative to the one-way ANOVA used to test differences among methods (in our case, the data representations). Given n blocks (repetitions) and k treatments (data representations), the Friedman test ranks each representation based on the mean of each repetition. Nemenyi is a post-hoc pairwise test for multiple comparisons of ranks . This test is able to find the groups that differ after a statistical significance test (in our case, Friedman test). Results are presented in Table 4.3, in which the values correspond to the average rank

Representation	Avg. Rank per dataset				Avg
	WebKB	20NG	ACM	Reuters90	Cross-Dataset
TFIDF	**3.70**	8.95	9.05	10.70	9.50
FS	**2.55**	10.00	9.70	9.90	9.50
MF(Cent)	12.25	11.20	11.50	11.45	11.40
MF(Cent) + SS	11.55	12.15	13.50	10.60	11.90
MF(Cent) + SPA	12.30	12.65	11.50	13.40	12.80
MF(Cent) + SPA + SS	12.15	14.00	13.50	12.95	13.90
MF(Cent+CosKnn)	**4.40**	**2.05**	**2.15**	**6.50**	**5.10**
MF(Cent+CosKnn)+SS	**6.75**	**5.20**	**5.65**	**5.10**	**6.30**
MF(Cent+CosKnn)+SPA	**4.55**	**3.20**	**1.50**	**7.05**	**6.30**
MF(Cent+CosKnn)+SPA+SS	**6.40**	**6.35**	**6.90**	**7.35**	**8.00**
MF(All)	**6.35**	**3.20**	**4.85**	**2.35**	**1.90**
MF(All) + SS	**8.60**	**5.00**	**5.15**	**1.80**	**2.90**
MF(All) + SPA	**5.65**	**3.55**	**4.90**	**3.45**	**2.10**
MF(All) + SPA + SS	**7.80**	**7.50**	**5.15**	**2.40**	**3.40**

Table 4.3: Rank per dataset and Cross-dataset Friedman Analysis on Micro-F1. **Bold** denotes rank ties by Nemenyi post-hoc.

position in the 10-fold cross-validation, considering *MicroF1*, of each representation in each dataset and cross-datasets (last column). **Bold** denotes rank ties based on the Nemenyi post-hoc test. As we can see in the Table 4.3, the Friedman-Nemenyi-tests agrees in most part with the t-tests with Bonferroni correction presented in Table 4.2.

One noticeable difference is that, according to the Friedman's test, $MF(Cent + CosKnn)$ and its variants now statistically tie with $MF(All)$ considering both, *MicroF1* and MacroF1 in Reuters90 – in the previous test, there were ties only in *MacroF1*. And confirming the previous test, $MF(Cent + CosKnn)$ and $MF(All)$ and their variants are the best representations, with ties with (TF-IDF and FS) only in WebKB.

Regarding the ranks themselves, we can see that in a *within dataset* analysis, in 3 out of 4 datasets (WebKB, 20NG and ACM) the $MF(Cent + CosKnn)$ and its variants usually get lower ranks that $MF(All)$, while the opposite behavior occurs in Reuters90. On the other hand, in the *cross-dataset* comparison, the $MF(All) + (*)$ have a slight tendency to have lower ranks, but from a statistical point of view, none of these differences are significant and both set of representations can be considered as equivalent according to the performed test.

Our results for the considered *MF representations* corroborate some of the results first obtained in [Canuto et al., 2018]: the *MF* space not only preserves the information in the original TF-IDF space but, in fact, enhances it by exploiting distance-based information from labeled data. In any case, the use of *MFs* as an explicit step in the pre-processing phase of text classification pipelines has not been discussed before, *mainly when considering the effectiveness-efficiency trade-offs imposed by its use in this context.* That is what we analyze next.

4.2.2 Efficiency Analysis (RQ2; RQ4)

In this section, we provide answers to research questions RQ2 and RQ4. In this analysis we will consider only the two most effective representations: $MF(All)$ and $MF(Cent + CosKnn)$. The overall total time for MF generation of these pipeline variants is shown in Table 4.2. It corresponds to the sum of the times for MF construction, sparsification, selective sampling and classifier learning (including parameter optimization).

Figure 4.1 presents the results in Macro-F1 by time for TFIDF, FS, MF (Cent + CosKnn) + (*) and MF (All) + (*). The idea is to be able to visualize the effectiveness of the methods in comparison with the time spent in training. On the y axis, we measure the Macro-F1 for the method, while on the x axis, we measure 1/time. Note that we use 1/time so that the most efficient methods are to the right of the axis. The lines correspond to the median values for the x and y axes (1/time and Macro-F1, respectively). Thus, the methods belonging to the first quadrant are those with better effectiveness than the median, but are more expensive. The second quadrant methods (clockwise) are considered the best in efficiency and effectiveness. Those in the third quadrant are the worst in MacroF1 but faster than the median. Finally, the fourth quadrant methods are worse than the median in time and effectiveness. To summarize, the more to the right and and the higher, the better the efficiency-effectiveness trade-off for the methods in question. Later on, in Table 4.5, we will break down these individual times for analysis purposes, but for now, we will concentrate on the overall time to analyze the trade-offs. All comparisons will be made against the faster version of TF-IDF, after applying feature selection (FS), since this is a standard procedure in traditional pipelines. FS caused a reduction of 30% of the features without diminishing effectiveness.

Figure 4.1 shows that, in all cases, both proposed steps (SPA and SS) produced significant efficiency gains, individually or when applied in conjunction, when compared to the MF versions. The SS step produces the largest time reductions when compared to SPA, though the later is also very efficient. When combined together, the largest cost reductions are usually observed. Note in Table 4.2, that although no effectiveness gains could be observed in WebKB with the use of the MFs, costs were reduced by a *factor of 4.0* in the case of $MF(Cent + CosKnn) + SPA + SS$. $MF(Cent + CosKnn)$ also produced the best trade-offs in 20NG with one of its variants ($MF(Cent + CosKnn) + SPA + SS$) producing statistically significant effectiveness gains over TF-IDF with *up to 9.7x processing time reductions.*

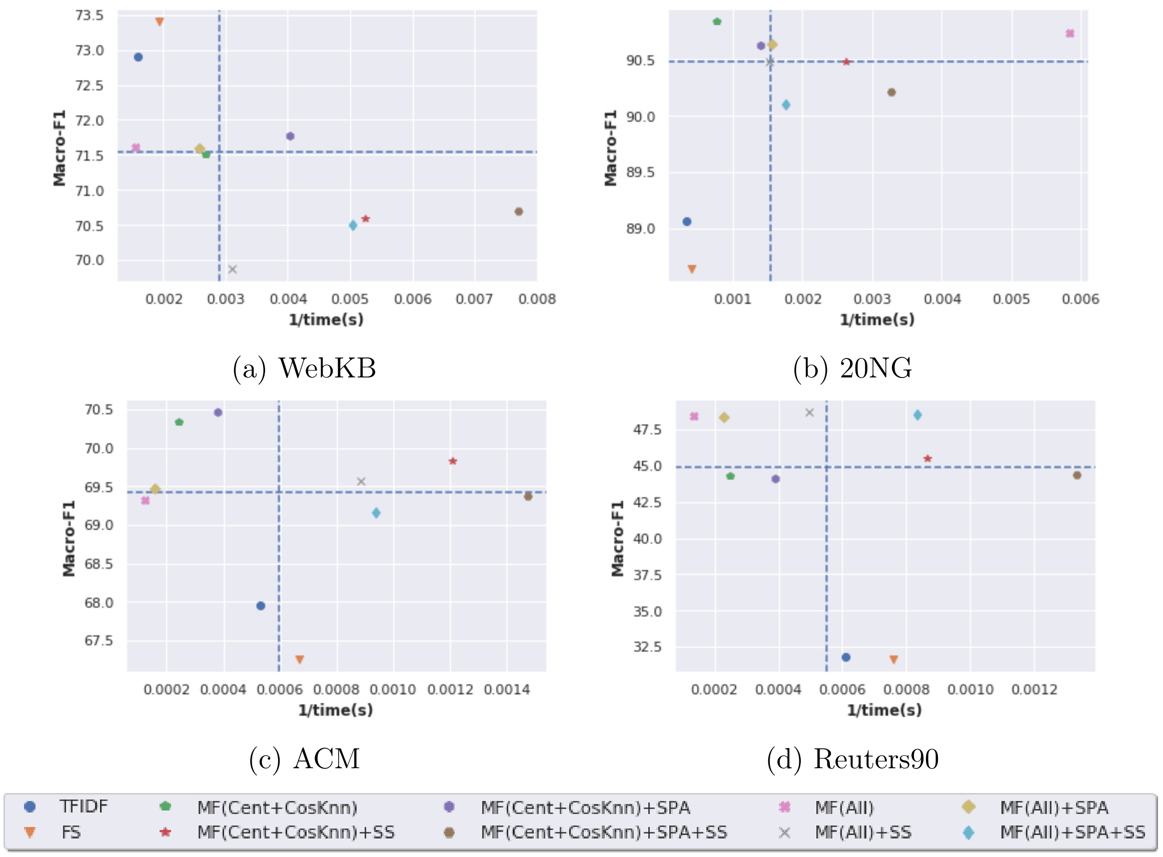


Figure 4.1: Trade-off between Macro-F1 (effectiveness) and training time (efficiency) for each dataset.

In ACM, we do have a clear winner: $MF(Cent+CosKnn)+SPA+SS$. Though slightly below the median, its results are statistically equivalent to the best ones. In other words, $MF(Cent+CosKnn)+SPA+SS$ produces effectiveness results that are better than TF-IDF and as good as any variant of $MF(All)$, at a much lower cost than any other alternative. ACM is an excellent example of the benefits SPA and SS bring together to the classification task. Each step in isolation produces some reduction in the overall costs, but when applied together they do reduce the overall time of $MF(Cent+CosKnn)$ by a factor of 6.0, resulting in an overall time almost 2.2x faster than TF-IDF with gains in effectiveness.

In Reuters90, we have a “good dilemma”. Both $MF(All)+SPA+SS$ and $MF(Cent+CosKnn)+SPA+SS$, belonging to the second quadrant, produce large effectiveness gains against TF-IDF, with the former having a slight tendency (not statistically significant) of producing better results in terms of MacroF1. As we can see, only $MF(All)$ and $MF(Cent+CosKnn)$ are slower in terms of efficiency. However, by applying the entire proposed pipeline, these costs are significantly reduced.

Overall, if cost is not an issue, we would recommend $MF(All) + SPA + SS$, otherwise $MF(Cent + CosKnn) + SPA + SS$ offers a better compromise, being more effective and faster than FS and TF-IDF. We emphasize that, in the particular case of Reuters90, even some $MF(Cent)$ variants are good alternatives. For instance, $MF(Cent) + SS$ has significant effectiveness gains in MacroF1 (with no loss in MicroF1) and is 5.2x faster than TF-IDF (even after FS). Considering all the results and effectiveness-efficiency trade-off analyses, our recommendation as a better compromise is $MF(Cent + CosKnn) + SPA + SS$.

These impressive results can be explained by reductions in dimensionality produced by the MF step, density reductions induced by the sparsification step (SPA), and selective sampling of instances to be used for training. The impact of these steps in each variant of the pipeline is detailed in Table 4.4.

	WebKB					20NG				
	Dim.	**Red.**	**Dens.**	**Inc.**	**#inst.**	**Dim.**	**Red.**	**Dens.**	**Inc.**	**#inst.**
TFIDF	23,090	-	101.6	-	7,379	103,392	-	83.2	-	16,961
FS	8,233	2.8x	88.6	0.87x	7,379	24,304	4.3x	74.0	0.89x	16,961
MF(Cent+CosKnn)	154	149.9x	154.0	1.5x	7,379	240	430.8x	240.0	2.9x	16,961
MF(Cent+CosKnn) + SS			154.0	1.5x	4,796			240.0	2.9x	11,024
MF(Cent+CosKnn) + SPA			150.1	1.5x	7,379			130.4	1.6x	16,961
MF(Cent+CosKnn) + SPA + SS			149.2	1.5x	4,796			132.5	1.6x	11,024
MF(All)	294	78.5x	294.0	2.9x	7,379	440	235.0x	440.0	5.3x	16,961
MF(All) + SS			294.0	2.9x	4,796			440.0	5.3x	11,024
MF(All) + SPA			290.1	2.9x	7,379			330.3	4.0x	16,961
MF(All) + SPA + SS			291.2	2.9x	4,796			333.4	4.0x	11,024

	ACM					Reuters90				
	Dim.	**Red.**	**Dens.**	**Inc.**	**#inst.**	**Dim.**	**Red.**	**Dens.**	**Inc.**	**#inst.**
TFIDF	52,083	-	28.5	-	22,407	28,593	-	57.6	-	11,994
FS	15,601	3.3x	25.8	0.90x	22,407	6,196	4.6x	51.9	0.90x	11,994
MF(Cent+CosKnn)	242	215.2x	242.0	8.5x	22,407	1,080	26.5x	1,080.0	18.8x	11,994
MF(Cent+CosKnn) + SS			242.0	8.5x	7,842			1,080.0	18.8x	5,997
MF(Cent+CosKnn) + SPA			187.1	6.6x	22,407			586.6	10.2x	11,994
MF(Cent+CosKnn) + SPA + SS			186.2	6.6x	7,842			586.0	10.2x	5,997
MF(All)	462	112.7x	462.0	16.2x	22,407	1,980	14.4x	1,980.0	34.4x	11,994
MF(All) + SS			462.0	16.2x	7,842			1,980.0	34.4x	5,997
MF(All) + SPA			406.8	14.3x	22,407			1486.2	25.8x	11,994
MF(All) + SPA + SS			407.3	14.3x	7,842			1485.8	25.8x	5,997

Table 4.4: Dimensionality, Density and Sampling Analysis

In fact, the reduction in the number of features (dimensionality) is considerable: for the two best representations, $MF(All)$ and $MF(Cent + CosKnn)$ (and its variants), it goes from a factor of 430.8x (maximum, in 20NG, with $MF(Cent+CosKnn)$) to 14.4x (minimum, in Reuters90, with $MF(All)$). We can also see in Table 4.4 that the proposed sparsification strategies (columns with $+SPA$) were able to reduce the density in the MF representations by a factor of 50% with no losses in effectiveness in all cases and large gains in learning time. The application of selective sampling (columns with *#inst.*) reduced the datasets sizes between 33% and 66%, with no losses in effectiveness. These reductions resulted in the aforementioned impressive gains in terms of efficiency of the final learning process.

Time	**WebKB**									
	MF			**SPA**	**SS**	**SVM**				**Total**
	Train	**Test**	**Test[avg]**			**Grid**	**Train**	**Test**	**Test[avg]**	
TFIDF	*	*	*	*	*	591.38	26.35	2.62	3.21e-03	627.8
FS	*	*	*	*	*	486.71	20.53	2.21	2.71e-03	518.7
MF(Cent+CosKnn)	3.76	0.49	5.99e-04	*	*	354.24	6.10	0.39	4.82e-04	372.4
MF(Cent+CosKnn)+SS				*	9.62	166.63	2.79	0.27	3.34e-04	191.0
MF(Cent+CosKnn)+SPA				0.02	*	230.30	5.44	0.32	3.85e-04	247.8
MF(Cent+CosKnn)+SPA+SS					4.63	115.33	2.66	0.22	2.71e-04	129.9
MF(All)	6.22	0.90	1.09e-03	*	*	617.64	10.89	0.87	1.06e-03	644.0
MF(All)+SS				*	18.12	285.52	4.31	0.51	6.26e-04	323.0
MF(All)+SPA				0.04	*	363.36	8.94	0.68	7.54e-04	387.5
MF(All)+SPA+SS					8.29	178.73	4.43	0.42	5.14e-04	198.2
	20NG									
TFIDF	*	*	*	*	*	2,801.90	127.30	11.37	6.03e-03	2,957.4
FS	*	*	*	*	*	2,360.50	94.58	9.88	5.24e-03	2,490.7
MF(Cent+CosKnn)	29.63	15.78	8.37e-03	*	*	1,229.50	8.37	1.96	1.04e-03	1,302.1
MF(Cent+CosKnn)+SS				*	37.50	277.25	4.06	1.10	5.86e-04	382,2
MF(Cent+CosKnn)+SPA				0.10	*	647.61	8.07	1.71	9.09e-04	719,8
MF(Cent+CosKnn)+SPA+SS					37.44	200.14	4.07	1.19	6.33e-04	305,2
MF(All)	49.83	22.19	1.18e-02	*	*	2,062.10	16.55	4.19	2.22e-03	2,171.7
MF(All)+SS				*	69.29	485.70	7.19	1.95	1.03e-03	653,0
MF(All)+SPA				0.19	*	1,534.80	14.47	3.21	1.70e-03	1,641.6
MF(All)+SPA+SS					69.01	400.13	8.22	2.23	1.18e-03	568,7
	ACM									
TFIDF	*	*	*	*	*	1,789.40	80.99	6.26	2.51e-03	1,883.4
FS	*	*	*	*	*	1,419.30	60.41	5.17	2.08e-03	1,501.6
MF(Cent+CosKnn)	28.71	9.52	3.82e-03	*	*	3,990.80	62.02	7.72	3.10e-03	4,105.5
MF(Cent+CosKnn)+SS				*	321.59	451.51	6.45	1.86	7.48e-04	826,4
MF(Cent+CosKnn)+SPA				0.12	*	2,542.80	39.60	4.76	1.91e-03	2,632.3
MF(Cent+CosKnn)+SPA+SS					176.76	430.00	22.57	4.38	1.76e-03	678,8
MF(All)	54.66	15.51	6.23e-03	*	*	7,864.20	119.78	14.10	5.66e-03	8,075.0
MF(All)+SS				*	339.87	698.38	12.22	3.71	1.49e-03	1,131.1
MF(All)+SPA				0.21	*	6,200.00	77.54	9.39	3.77e-03	6,364.1
MF(All)+SPA+SS					319.78	650.85	12.03	3.74	1.50e-03	1,063.5
	Reuters90									
TFIDF	*	*	*	*	*	1,544.90	84.86	5.75	4.31e-03	1,643,3
FS	*	*	*	*	*	1,209.80	60.72	5.05	3.79e-03	1,317.4
MF(Cent+CosKnn)	87.35	74.11	5.56e-02	*	*	3,810.10	38.00	9.57	7.18e-03	4,027.0
MF(Cent+CosKnn)+SS				*	118.83	847.33	13.94	6.08	4.56e-03	1,155.5
MF(Cent+CosKnn)+SPA				0.39	*	2,344.40	36.24	8.28	6.22e-03	2,558.6
MF(Cent+CosKnn)+SPA+SS					109.37	456.98	9.65	5.02	3.77e-03	750,7
MF(All)	136.57	110.84	8.32e-02	*	*	7,077.80	75.68	17.32	1.30e-02	7,426.1
MF(All)+SS				*	203.16	1,524.30	24.86	10.52	7.89e-03	2,018.1
MF(All)+SPA				0.57	*	4,024.20	63.08	13.66	1.03e-02	4,356.8
MF(All)+SPA+SS					118.44	801.73	16.85	8.30	6.68e-03	1,201.2

Table 4.5: Time Spent in Each Step of the Proposed Pipeline (* does not apply)

For our final analysis regarding these research questions, we now break down the overall time in the individual times taken by each step of the proposed pipeline. These times can be seen in Table 4.5. We can see that the time to generate the meta-features (MF column) in all datasets is at most 21.5% of the overall time.which is largely dominated by the time to train the SVM classifier (comprising parameterization and actual learning). In the cases in which the whole pipeline is applied

$(MF(Cent + CosKnn) + SPA + SS)$, the SVM learning time is usually between 66%-95% of the overall time. The SPA time is thus quite negligible in all cases, when compared to the total time. Also, the time for SS is no more than 35% of the overall time. In fact, in the largest and more difficult datasets (ACM and Reuters90) the time for SS is considerably lower than the time to create the MF representations. But as we shall see this representation is crucial for the application of SS step.

The most interesting results are again seen when applying the whole pipeline. The two additional steps of sparsification and selective sampling can reduce the overall time by a factor of 7.6-13.8 when compared to $MF(*)$ causing also the overall time to be much lower than that of the TF-IDF (up to 2.2 times faster) in all cases. These gains in efficiency come mainly due to the reduction of the SVM learning time as a result of the low-dimensional, "not-so-dense" (in terms of lines/documents) final representation produced by the proposed pipeline. Note that the test and test average[1] time in all cases are negligible when compared with the the other steps of the pipeline.

Therefore, our proposal can explore efficiency and effectiveness representations.

4.2.3 Comparison with Embedding-Based Representations (RQ3)

Remind that the research question we aim at answering in this section is "How do the MFs compare with other alternative low-dimensional representations (e.g., word embeddings) in terms of effectiveness and incurred costs?" In order to answer this, we run experiments with three embeddings-based representations described in section 2.2: PTE, FisherVector and FastText. As a way of keeping the comparison fair, we standardize the use of the SVM as default classifier, with the parameterization explained in Section 4.1.2. For generating each representation, best parameters were searched using Grid Search with the validation sets. We present in Table 4.6 the searched parameters for each method, as well as, the search values range. Rows that have only one value correspond to parameters for which we use the best values based on the original works [Lev et al., 2015; Mikolov et al., 2018a; Tang et al., 2015]. The best parameters found for each representation are marked in **bold**. The best found parameters were the same for all 4 considered datasets.

[1]In Table4.5, column **Test** stands for the average time to classify the complete test set, whereas **Test[avg]** corresponds to the average time taken to classify an instance of the test set.

	Parameter	Values
PTE	context window	[1, 2, 3, 4, **5**]
	samples	[100, 200, **300**]
	dimension	[100, **300**]
	min word count	1
FisherVector	pre-treined embedding	[GloVe, **GoogleNews**]
	dimension	[100, **300**]
FastText	context window	[1, 2, 3, 4, **5**]
	epochs	[100, **500**, 1000]
	dimension	[100, **300**]
	learning rate	[0.1]
	loss function	[Skipgram negative sampling (a.k.a. ns)]

Table 4.6: Embedding-Based Representations Parameterization Search.

The overall time includes the time to generate these representations. Results are shown in Table 4.7. As before, results in **bold** are the best among all and the arrow markers correspond to the comparisons with FS, the faster Bow-based representation. As we can see, in terms of effectiveness, FastText is clearly the worst among the embeddings representations (6 losses in 8 cases), while PTE and FisherVector are more competitive. In any case, all embeddings-based representations are worse than (or at most tie with) FS in all cases. In fact, there are even significant losses, for instance, with all embeddings approaches in WebKB and with FastText in ACM and 20NG. Indeed, embeddings are never competitive when compared to the MF representations. In all cases, there are significant losses.

In terms of efficiency, the embeddings results are even poorer: the overall times to generate them are 1.5x-31.1x slower than FS. When compared to the overall times after the application of our pipeline, the losses are even more significant – they reach up to 68.9x of slowdown (as in the case of Fisher vs $MF(Cent + CosKnn) + SPA + SS$ in ACM).

	WebKB			20NG		
	MicroF1	MacroF1	Time(s)	MicroF1	MacroF1	Time(s)
FS	**82.79(1.0)**	**73.41(2.4)**	518.7	88.82(0.7)	88.64(0.8)	2,490.7
MF(Cent+CosKnn)+SPA+SS	**80.83(0.9)** ●	**70.71(2.4)** ●	129.9	**90.36(0.5)** ▲	**90.21(0.6)** ▲	305.2
MF(All) + SPA + SS	**80.55(1.1)** ●	**70.51(2.1)** ●	198.2	**90.25(0.4)** ▲	**90.10(0.4)** ▲	568.7
PTE + SVM	72.0(1.1) ▼	59.10(2.9) ▼	765.9	88.96(0.5) ●	88.72(0.5) ●	2,480.7
FisherVector + SVM	74.81(0.7) ▼	64.45(1.1) ▼	16,121.0	87.88(2.8) ●	87.44(2.9) ●	38,497.8
FastText+SVM	76.80(1.2) ▼	69.17(2.8) ●	390.7	81.58(0.6) ▼	81.14(0.7) ▼	1,906.6

	ACM			Reuters90		
	MicroF1	MacroF1	Time(s)	MicroF1	MacroF1	Time(s)
FS	77.41(0.4)	67.25(1.4)	1,501.6	73.54(1.1)	31.61(3.1)	1,317.4
MF(Cent+CosKnn)+SPA+SS	**79.29(0.7)** ▲	**69.37(1.8)** ▲	678.8	**78.36(1.0)** ▲	**44.35(2.9)** ▲	750.7
MF(All) + SPA + SS	**79.45(0.5)** ▲	**69.17(1.4)** ▲	1,063.5	**82.43(0.7)** ▲	**48.55(2.7)** ▲	1,201.2
PTE + SVM	76.41(0.7) ●	63.71(1.2) ▼	2,912.1	72.54(0.9) ●	30.52(2.5) ●	1,719.5
FisherVector + SVM	75.34(1.1) ●	69.05(2.5) ●	46,675.2	63.28(0.7) ▼	28.37(3.8) ●	6,253.5
FastText + SVM	73.64(0.7) ▼	62.67(1.3) ▼	1,191.9	68.08(0.9) ▼	32.26(2.2) ●	1,571.5

Table 4.7: Comparison with Word Embeddings Approaches

To better understand this rather surprising results, we run additional experiments in which we use:

- the original classifiers with which the embedding representations were run, i.e., Linear Regression (LR) for PTE and FastText and SVM for FisherVector as already reported in Table 4.7 ;
- the TF-IDF original representation (without Feature selection) with Linear regression, since these were the baselines for PTE and FastText in the original works; as before, the LR classifier was tuned using the validation sets;
- Bow-based representation (FS and TF-IDF) without tuning of the Linear Regression classifier, since the original works were not clear about how they parameterized LR. The hypothesis here is that LR when applied to BOW-based representations may have been improperly tuned.

Results are shown in Table 4.8.

	WebKB			**20NG**		
	MicroF1	**MacroF1**	**Time(s)**	**MicroF1**	**MacroF1**	**Time(s)**
TFIDF + SVM	**82.26(0.9)** ●	**72.90(2.1)** ●	627.8	**89.23(0.7)** ●	**89.07(0.7)** ●	2,957.4
FS + SVM	**82.79(1.0)** ●	**73.41(2.4)** ●	518.7	**88.82(0.7)** ●	**88.64(0.8)** ●	2,490.7
TFIDF + LR	**82.24(0.6)** ●	**70.54(1.8)** ●	16.4	**89.65(0.6)** ●	**89.42(0.6)** ●	120.8
TFIDF + LR (without tunning)	79.79(0.7) ▼	63.78(1.4) ▼	0.8	86.73(0.6) ▼	86.05(0.7) ▼	6.7
FS + LR	**82.33(0.7)** ●	**70.94(2.3)** ●	12.9	**89.25(0.6)** ●	**89.03(0.6)** ●	67.8
FS + LR (without tunning)	79.78(0.8) ▼	63.88(1.6) ▼	0.6	86.25(0.6) ▼	85.50(0.7) ▼	4.2
PTE + SVM	71.92(0.9) ▼	59.40(2.2) ▼	765.9	**88.96(0.5)** ●	**88.72(0.5)** ●	2,480.7
PTE + LR	72.28(0.8) ▼	59.23(1.6) ▼	349.9	**89.19(0.5)** ●	**88.85(0.5)** ●	636.1
FisherVector + SVM	74.81(0.7) ▼	64.45(1.1) ▼	16,121.0	**87.88(2.8)** ●	**87.44(2.9)** ●	38,497.8
FastText + SVM	76.80(1.2) ▼	**69.17(2.8)** ●	390.7	81.58(0.6) ▼	81.14(0.7) ▼	1,906.6
FastText + LR	75.61(1.2) ▼	65.98(1.7) ▼	285.9	80.56(0.5) ▼	80.09(0.5) ▼	1,144.2
	ACM			**Reuters90**		
	MicroF1	**MacroF1**	**Time(s)**	**MicroF1**	**MacroF1**	**Time(s)**
TFIDF + SVM	**77.76(0.4)** ●	**67.95(0.7)** ●	1,883.4	**73.31(1.0)** ●	**31.86(3.2)** ●	1,643.3
FS + SVM	**77.41(0.4)** ●	**67.25(1.4)** ●	1,501.6	**73.54(1.1)** ●	**31.61(3.1)** ●	1,317.4
TFIDF + LR	**78.64(0.4)**	**65.54(1.1)** ●	45.1	66.85(0.9) ▼	**33.43(2.4)** ●	228.3
TFIDF + LR (without tunning)	76.60(0.7) ●	59.58(0.6) ▼	1.7	69.50(1.2) ▼	25.49(2.0) ▼	9.8
FS + LR	**78.02(0.4)** ●	64.69(0.7) ▼	31.3	67.92(0.9) ▼	**32.55(2.2)** ●	152.5
FS + LR (without tunning)	76.13(0.6) ●	59.20(0.6) ▼	1.2	66.71(1.1) ▼	25.44(2.1) ▼	7.4
PTE + SVM	76.41(0.7) ●	63.71(1.2) ▼	2,912.1	**72.54(0.9)** ●	**30.52(2.5)** ●	1,719.5
PTE + LR	76.18(0.6) ●	63.35(1.3) ▼	501.0	67.36(0.7) ▼	**31.08(2.5)** ●	1,192.9
FisherVector + SVM	75.34(1.1) ●	**69.05(2.5)** ●	46,675.2	63.28(0.7) ▼	**28.37(3.8)** ●	6,253.5
FastText + SVM	73.64(0.7) ▼	62.67(1.3) ▼	1,191.9	68.08(0.9) ▼	**32.26(2.2)** ●	1,571.5
FastText + LR	73.39(0.7) ▼	61.33(1.1) ▼	599.9	66.35(0.7) ▼	**30.58(2.2)** ●	2,294.9

Table 4.8: Additional Experiments with Linear Regression (LR) classifier and parametrization tunning

As can be seen, in Table 4.8, the results show that:

- Feature selection did not have much impact on effectiveness when used with LR. This is consistent with our previous results with SVM.

- LR produces equivalent or worse results than SVM in all cases (e.g., there are losses in Reuters90 and ACM with FS). In fact, our results with **PTE + SVM** are even higher in 20NG than those reported in the original work;
- Results of TF-IDF without LR tuning are consistent with those reported in the original papers when using the same datasets (e.g., 20NG).

In sum, the alleged claims of superiority of the embedding representations over the BoW-based ones previously reporteded may have been a combination of: (i) the use of weaker classifiers (e.g. LR); (ii) unfair tuning of the baselines; (iii) improper experimental protocols (all experiments were run with a single split of the datasets); (iv) lack of statistical significance tests to reject the null hypothesis of equality of results; (v) a combination of some or all the above options. Similar issues have been recently reported in comparative works in related areas (e.g., [Dacrema et al., 2019; Shen et al., 2018]).

We should stress that these results regard only ATC tasks in the tested datasets and cannot be generalized for all NLP tasks in which the experimented embeddings can be used.

4.2.4 Selective Sampling Applied to TF-IDF (RQ5)

A natural question that arises from our discussion is whether we could perform a selective sampling in the original high-dimensional TF-IDF representation before generating the MFs. Figure 4.2 shows the application of Cover on the TF-IDF representation after feature selection. We use the log_e scale due to the large time difference in both applications. We can see that the sampling time in this representation is *13.1x to 176.6x* slower than when applying it to $MF(All)$. In fact, when comparing with the results in Figure 4.2, we can see that SS applied to TF-IDF is *6.3x to 34.7x* slower than just using the original TF-IDF (with FS - Table 4.5) to learn the classifier, with no selective sampling at all.

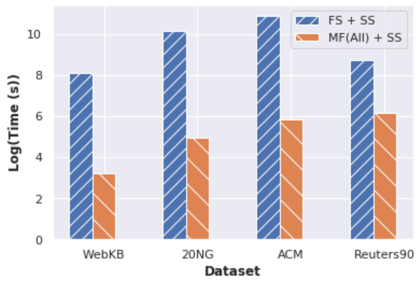


Figure 4.2: Log of the Time Spent in Cover on TF-IDF Representation.

Simply put, Cover, as designed, is not suitable to be applied to the original TF-IDF representation before generating MFs, since its complexity is highly influenced by the dimensionality size (see Section 3.3).

4.2.5 Impact on other Classifiers (RQ6)

In these experiments, we chose to show the results with $MF(All)$ (instead of $MF(Cent + CosKnn)$) and the complete pipeline $MF(All) + SPA + SS$ applied to it. We do so because $MF(All)$ is the most complete meta-level representation we experiment with. As we do not know, a priori, how the other classifiers would behave with the MF representation, we provide to them as much information as we could. As one of our main goals with these experiments is to check whether the observed gains in efficiency could be replicated with other classifiers, $MF(All)$ serves well our purpose. We leave for the future experimentations with all other variants. As before, all parameterization of learners and pipelines was performed in the validation sets.

As representatives of state-of-the-art classifiers we chose the Random Forest (RF) classifier and two recently proposed extensions of RF that has excelled in text classification tasks: BROOF [Salles et al., 2015] and BERT [Campos et al., 2017]. RF was proposed by Breiman [Breiman, 2001], and its fundamental aspect that guarantees the high effectiveness is the large set of low-correlated trees composing the forest, which is obtained by disturbing the data with series of random procedures, such as bagging of the training set and random attribute selection drawn from a randomly chosen subset of features. BROOF, a boosted RF strategy that combines boosting and bagging by exploiting RF as "weak learners", and BERT, which is BROOF-like and adds an additional source of randomization to reduce bias. Concerning those algorithms, we set the parameters as suggested in [Campos et al., 2017; Salles et al., 2015]. We also chose two simple neural network architectures (NNs). Their simplicity was targeted at efficiency, mainly for hyperparameter optimization, thus they are trained following a parameterization process with cross-validation, as the other classifiers. This guarantees a fair comparison. As shown in the literature [Le et al., 2018] deep models have not yet proven to be more effective than shallow models for ATC tasks, thus, both chosen NNs have few layers and simpler architecture for the sake of efficiency. Both architectures are feed-forward neural network: the first one, named Multilayer Perceptron (MLP) consists of one input and one output layer. The second, named Deep Multilayer Perceptron (DMLP) comprises one input layer, some hidden layers and one output layer. For parameter optimization (based on MacroF1), we use the Tree-of-Parzen-Estimators (TPE) algorithm with both NNs models [Bergstra et al., 2015]. The number of neurons, learning rate, activation function, type of regularization were optimized in both NNs, while dropout and number of layers were optimized only in the DMLP architecture. Our deep learning models were implemented using Keras and trained on an Intel superscript registered Core i7, 64 GB of memory and GPU Titan V. The number of epochs and batch size are set to 100 and 64, respectively. We also use early stopping criteria and set patience to 20 epochs. Figure 4.3 presents the results of *MicroF1*, *MacroF1* and time for each of the considered classifiers and representations.

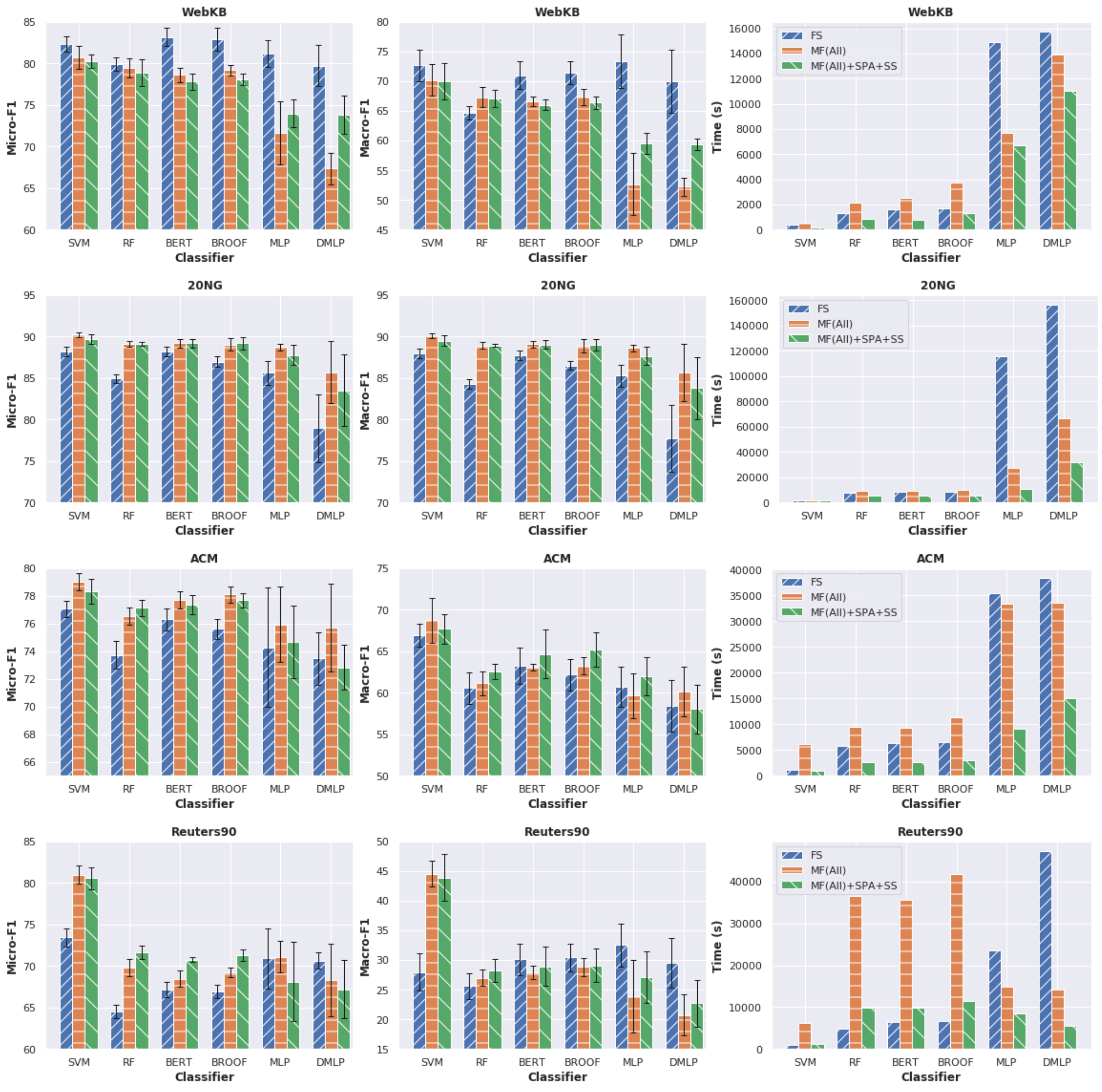


Figure 4.3: Proposed Pipeline Applied to Other State-of-The-Art Classifiers.

We start our analysis by observing that the overall best results among all classifiers in all datasets, considering all metrics, are still produced by SVM. As mentioned before, as far as we know, SVM is still one of the best-known text classifiers, at least in the tested datasets. Regarding the RF and RF-based classifiers (BERT and BROOF), we can see that the latter produced slightly better results in terms of effectiveness. In terms of efficiency, RF and BERT are almost equivalent, while BROOF is moderately more time consuming. But, more interestingly, when looking at the intra-classifier comparison between TF-IDF (with FS) and $MF(All) + SS + SPA$ with RF, BERT and BROOF, we observe no effectiveness losses for the latter representation (other than in MacroF1 in WebKB)[2] , and mainly several gains as seen on 20NG, ACM, and Reuters90 in MicroF1 and 20NG on MacroF1. We can observe (statistically) significant gains of $MF(All)+SS+SPA$ over TF-IDF in MicroF1 in 20NG, ACM and Reuters90, in MacroF1 in WebKB and in both metrics in 20NG and ACM. The largest overall improvements of the RF classifiers are observed in MicroF1 in Reuters90, almost 11% with RF. The second important observation is that the application of the complete pipeline produced significant gains in terms of efficiency. When comparing the overall times of $MF(All) + SPA + SS$ with those of TF-IDF (FS), we can see speedup gains between 1.3x-2.4x. When looking at the impact of the use of complete pipeline over $MF(All)$, the gains in efficiency achieve up to 3.8x. In fact, even some effectiveness gains may be observed with the use of the complete pipeline over $MF(All)$ such as in 20NG with both classifiers in both metrics. This may reveal that, with selective sampling, we are either removing noise or improving the generalization of the model, as it becomes less complex. We will investigate this issue further in the future.

Regarding the NNs, we see that they are no match for SVM in terms of the effectiveness-efficiency trade-off. However, the NNs are competitive with the RF classifiers in terms of effectiveness in some datasets, such as in 20NG and Reuters90, although much less efficient. When comparing both NN architectures, MLP is, in general, more effective and faster, probably due to the use of fewer layers. In any case, the application of both NN architectures to the original TF-IDF representation is too time-consuming. Also, disconsidering WebKB, which is perhaps too small for the NNs to properly work, the complete pipeline over $MF(All)$ not only reduced the overall time considerably (e.g. speedup of 10x in 20NG and 8x in Reuters90), with no effectiveness loss (in case of MLP) and even gains over TF-IDF, (e.g. in 20NG), confirming the enrichment of information brought by the MFs.

[2]The results of both RF-based classifiers in Reuters90 with all representations, including TF-IDF, are very far away from those of SVM. We believe this is due to the large number of classes of this dataset and the use of the one-against-all strategy of SVM.

To summarize, MF representations with sparsification and selective sampling makes it possible to apply NN architectures to text classification tasks with potential effectiveness gains and much improved scalability. Scalability and training costs are, in particular, important issues that have been mostly neglected or poorly discussed in applications of deep learning to text classification tasks.

Chapter 5

Conclusion and Future Work

We have provided evidence that, perhaps even more important than the classifier algorithm itself, is the adequate pre-processing of the data to achieve the best possible effectiveness results at the minimum cost (effectiveness-efficiency trade-off). We have introduced three new steps into the traditional text classification pipeline which were shown to produce large effectiveness gains or cost reductions (time), or both. By transforming the original textual representation, reducing dimensionality (MF generation step), increasing sparseness (SPA step) and reducing the number of training instances (SS step), we achieved improvements in terms of effectiveness with decreased costs. We verified this accomplishment using a carefully designed, statistically rigorous experimental framework, that highlight how each proposed pipeline step influences on effectiveness.Based on our experiments, We could conclude that, considering the analyzed effectiveness-efficiency trade-offs, $MF(Cent + CosKnn)$ and its variants produced the overall best results, after the application of the complete pipeline: it produces the best effectiveness with the largest reductions in overall time.

Our work has the potential to change the way text classification is performed. Our proposed pipeline is by no means a fixed one. In fact, we presented only one instantiation of it. Different or enhanced versions of the algorithms we have employed in each step could be considered. The order of the steps could be changed (e.g., SS before MF, if efficient solutions are devised for TF-IDF-based representations). Steps could be omitted (or included, e.g., FS) depending on the characteristics and goals (focus on effectiveness or efficiency).

As future work, we envision the construction of AutoML solutions that could incorporate the proposed steps according to the characteristics of the datasets and goals of the task. We also plan to evaluate our proposal on additional datasets, classifiers, and configurations. We also envision improvements in the individual steps of the current pipeline. In MF generation, we could take better advantage of the similarity information of pairs of neighbors, using common information (words) between them. Also, it is possible to increase sparsity levels of the representations with a *per document* strategy. In the SS step, we could better exploit the information of the positive classes in the selection of negative ones. We have not explored labeled information in the selection process at all. In sum, we have only started to grasp the possibilities opened up by our proposal.

Bibliography

Akhila, G., Madhu, G., Madhu, M., and Pooja, M. (2014). Comparative study of classification algorithms using data mining. In *Discovery Science*, volume 9, pages 17--21.

Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238--247, Baltimore, Maryland. Association for Computational Linguistics.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5--32. ISSN 0885-6125.

Campos, R., Canuto, S., Salles, T., de Sá, C. C., and Gonçalves, M. A. (2017). Stacking bagged and boosted forests for effective automated classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 105--114, New York, NY, USA. ACM.

Canuto, S., Gonçalves, M., Santos, W., Rosa, T., and Martins, W. (2015). An efficient and scalable metafeature-based document classification approach based on massively parallel computing. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 333--342, New York, NY, USA. ACM.

Canuto, S., Gonçalves, M. A., and Benevenuto, F. (2016). Exploiting new sentiment-based meta-level features for effective sentiment analysis. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, pages 53--62, New York, NY, USA. ACM.

Canuto, S., Salles, T., Gonçalves, M. A., Rocha, L., Ramos, G., Gonçalves, L., Rosa, T., and Martins, W. (2014). On efficient meta-level features for effective text classification. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 1709--1718, New York, NY, USA. ACM.

Canuto, S., Sousa, D. X., GonÃ§alves, M. A., and Rosa, T. C. (2018). A thorough evaluation of distance-based meta-features for automated text classification. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2242–2256. ISSN .

Capannini, G., Lucchese, C., Nardini, F. M., Orlando, S., Perego, R., and Tonellotto, N. (2016). Quality versus efficiency in document scoring with learning-to-rank models. *Information Processing & Management*, 52(6):1161 – 1177. ISSN 0306-4573.

Cheng, Y., Chen, Z., Liu, L., Wang, J., Agrawal, A., and Choudhary, A. (2013). Feedback-driven multiclass active learning for data streams. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 1311--1320, New York, NY, USA. ACM.

Dacrema, M. F., Cremonesi, P., and Jannach, D. (2019). Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019.*, pages 101--109.

de Sá, A. G. C., Pinto, W. J. G. S., Oliveira, L. O. V. B., and Pappa, G. L. (2017). Recipe: A grammar-based framework for automatically evolving classification pipelines. In McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., and García-Sánchez, P., editors, Genetic Programming, pages 246--261, Cham. Springer International Publishing.

Dogan, N. and Tanrikulu, Z. (2013). A comparative analysis of classification algorithms in data mining for accuracy, speed and robustness. *Information Technology and Management*, 14(2):105--124. ISSN 1573-7667.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871--1874. ISSN 1532-4435.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2018). Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, pages 1189--1232.

Gopal, S. and Yang, Y. (2010). Multilabel classification with meta-level features. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 315--322, New York, NY, USA. ACM.

Guo, L., Yin, H., Wang, Q., Chen, T., Zhou, A., and Quoc Viet Hung, N. (2019). Streaming session-based recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1569--1577, New York, NY, USA. ACM.

Kastrati, Z., Imran, A. S., and Yayilgan, S. Y. (2019). The impact of deep learning on document classification using semantically rich representations. *Information Processing & Management*, 56(5):1618 – 1632. ISSN 0306-4573.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2019). *Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA*, pages 81--95. Springer International Publishing, Cham.

Kyriakopoulou, A. and Kalamboukis, T. (2007). Using clustering to enhance text classification. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 805--806, New York, NY, USA. ACM.

Le, H., Cerisara, C., and Denis, A. (2018). Do convolutional networks need to be deep for text classification ?

Lev, G., Klein, B., and Wolf, L. (2015). *In Defense of Word Embedding for Generic Text Representation*, pages 35--50. Springer International Publishing, Cham.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765--6816. ISSN 1532-4435.

Lin, J. (2019). The neural hype and comparisons against weak baselines. *SIGIR Forum*, 52(2):40--51. ISSN 0163-5840.

Long, B., Chapelle, O., Zhang, Y., Chang, Y., Zheng, Z., and Tseng, B. (2010). Active learning for ranking through expected loss optimization. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 267--274, New York, NY, USA. ACM.

Louppe, G., Wehenkel, L., Sutera, A., and Geurts, P. (2013). Understanding variable importances in forests of randomized trees. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pages 431--439, USA. Curran Associates Inc.

Ludewig, M., Mauro, N., Latifi, S., and Jannach, D. (2019). Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, pages 462--466, New York, NY, USA. ACM.

March, W. B., Ram, P., and Gray, A. G. (2010). Fast euclidean minimum spanning tree: Algorithm, analysis, and applications. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 603--612, New York, NY, USA. ACM.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2018a). Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2018b). Advances in pre-training distributed word representations.

Olson, R. S. and Moore, J. H. (2019). *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning*, pages 151--160. Springer International Publishing, Cham.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532--1543, Doha, Qatar. Association for Computational Linguistics.

Salles, T., Gonçalves, M., Rodrigues, V., and Rocha, L. (2015). Broof: Exploiting out-of-bag errors, boosting and random forests for effective automated classification. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 353--362, New York, NY, USA. ACM.

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513--523. ISSN 0306-4573.

Schoenfeld, B., Giraud-Carrier, C. G., Poggemann, M., Christensen, J., and Seppi, K. D. (2018). Preprocessor selection for machine learning pipelines. *CoRR*, abs/1810.09942.

Sculley, D., Snoek, J., Wiltschko, A. B., and Rahimi, A. (2018). Winner's curse? on pace, progress, and empirical rigor. In *ICLR*.

Shen, D., Wang, G., Wang, W., Min, M. R., Su, Q., Zhang, Y., Li, C., Henao, R., and Carin, L. (2018). Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440--450, Melbourne, Australia. Association for Computational Linguistics.

Silva, R., Gonçalves, M. A., and Veloso, A. (2011). Rule-based active sampling for learning to rank. In *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III*, ECML PKDD'11, pages 240--255, Berlin, Heidelberg. Springer-Verlag.

Silva, R. M., Gomes, G. C., Alvim, M. S., and Gonçalves, M. A. (2016). Compression-based selective sampling for learning to rank. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 247--256, New York, NY, USA. ACM.

Silva, R. M., Gonçalves, M. A., and Veloso, A. (2014). A two-stage active learning method for learning to rank. *J. Assoc. Inf. Sci. Technol.*, 65(1):109--128. ISSN 2330-1635.

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage.*, 45(4):427--437. ISSN 0306-4573.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Tang, J., Qu, M., and Mei, Q. (2015). Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1165--1174, New York, NY, USA. ACM.

Viegas, F., da Rocha, L. C., Gonçalves, M. A., Mourão, F., Sá, G., Salles, T., Andrade, G., and Sandin, I. (2018). A genetic programming approach for feature selection in highly dimensional skewed data. *Neurocomputing*, 273:554--569.

Yang, Y. and Gopal, S. (2012). Multilabel classification with meta-level features in a learning-to-rank framework. *Mach. Learn.*, 88(1-2):47--68. ISSN 0885-6125.

Zamani, H., Dehghani, M., Croft, W. B., Learned-Miller, E., and Kamps, J. (2018). From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pages 497--506, New York, NY, USA. ACM.

Zar, J. H. (2007). *Biostatistical Analysis (5th Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA. ISBN 0131008463.