A Benchmark for Relation Extraction Kernels

João L.M. Pereira(✉), Helena Galhardas, and Bruno Martins

INESC-ID and Instituto Superior Técnico, Universidade de Lisboa,
Lisbon, Portugal
{joaoplmpereira,helena.galhardas,bruno.g.martins}@tecnico.ulisboa.pt

Abstract. Relation extraction from textual documents is an important task in the context of information extraction. This task aims at identifying relations between pairs of named entities and assigning them a type. Relation extraction is often approached as a supervised classification problem, involving pre-processing steps such as text segmentation, entity recognition, and morphological and syntactic annotations. In previous studies, the way data is pre-processed differs among them, thus making the comparison of classification techniques for relation extraction unfair and inconclusive. Some of these classification techniques for relation extraction involve the use of kernels, which enable the comparison of complex structures. We propose a benchmark for the comparison of different kernels for relation extraction. Specifically, we propose the application of a common pre-processing stage, together with the use of an online learning algorithm to train Support Vector Machines with kernels designed for the classification of candidate pairs of related entities. We also report the results of the systematic experimental validation we have performed, using well known datasets in the area.

Keywords: Relation extraction · Benchmark · SVMs · Kernels · Online learning

1 Introduction

Textual corpora available in digital format are growing fast. These documents contain valuable information that, if properly identified and structured, can be used by several applications (e.g., in news aggregators). Information Extraction consists in automatically obtaining structured data from textual documents. This activity is typically composed of several tasks, namely segmentation, entity extraction, normalization, co-reference resolution and relation extraction [11].

This paper focuses on the Relation Extraction (RE) task. For example, given the text: *"The Taliban group Tehreek-e-Taliban Pakistan claimed the attack on the Karachi airport in southern Pakistan"*, a RE system should be able to identify the relation between the terrorist entity *"Tehreek-e-Taliban Pakistan"* and the location entity *"Karachi"*, and to classify this relation as being of the type outrage. The result will be the tuple ⟨*"Tehreek-e-Taliban Pakistan"*, *"Karachi"*⟩ of a relation with schema outrage(terrorist, location).

T. Morzy et al. (Eds.): ADBIS 2015, LNCS 9282, pp. 184–197, 2015.
DOI: 10.1007/978-3-319-23135-8_13

There are different types of techniques for RE. These techniques are usually divided in two main groups: (i) rule-based, that specify logical inferences manually designed by specialists; and (ii) Machine Learning (ML)-based, that extract relations through the automatic analysis of patterns and/or correlations in data [11]. Several techniques, specially the ML-based, depend on linguistic and/or lexico-syntactic data annotations (e.g., part-of-speech tags for words). The tasks for obtaining these annotations constitute a pre-processing step.

Currently, many of the techniques for RE are based on supervised ML which identifies patterns in the previously annotated data that constitute a training dataset. A RE task can be seen as a supervised ML problem by considering the representation of a pair of entities as a data instance, and the relation type (or a non-relation) as the class that a data instance can belong to. To provide the dataset in the format usually accepted by a supervised ML technique, we have to transform each data instance representing a pair of entities into a vector that contains the relevant features (e.g., words or grammatical dependencies) of the pair of entities. Representing such information in a vector is a very demanding task in terms of execution time and memory. Moreover, it results in large vectors because the space of all possible relevant features may be huge. Thus, supervised ML techniques, and the models that they generate, can be inefficient since their execution time highly depends on the length of the vectors. One way to circumvent this situation is to use complex data structures, which organize relevant features in a more efficient way, and give them directly as input to the supervised ML techniques and the generated models. Then, we use kernel functions that enable specific ML techniques to compare complex data structures. The use of kernel functions is a common practice in the ML community, in particular in RE systems. For example, Support Vector Machines (SVMs) constitute a supervised ML technique that can be based on kernels.

Over the years, some competitions (e.g., MUC [5], ACE [6] and SemEval [8]), aiming at comparing RE systems, took place. Several RE systems based on kernels were evaluated with datasets from these competitions. However, these competitions did not enable a fair comparison of RE kernels. In fact, these competitions used different pre-processing stages, distinct SVM training approaches, and datasets that belong to the same domain, thus making the comparison of kernels unfair. Inspired by the work of Marrero et al. [10], who proposed a benchmark for entity extraction systems, we propose, in this paper, a benchmark for kernels for the RE task. This benchmark was developed using a RE framework named REEL [1][1]. The benchmark is composed of: (i) two sets of documents from distinct domains: AImed[2] and SemEval[3]; (ii) a common pre-processing step, composed of segmentation, tokenization, normalization, capitalization, part-of-speech tagging and dependency analysis; (iii) a set of SVMs based on state-of-the-art RE kernels; (iv) an online learning algorithm that trains

[1] http://reel.cs.columbia.edu/
[2] ftp://ftp.cs.utexas.edu/pub/mooney/bio-data/interactions.tar.gz
[3] http://semeval2.fbk.eu/semeval2.php?location=data

the SVMs involving various kernels; and (v) a validation process that measures convergence, quality and execution time.

The rest of this paper is organized as follows. Section 2 presents the fundamental concepts. Section 3 describes the proposed benchmark. Section 4 describes the experiments we performed and discusses the obtained results. Finally, in Sect. 5, we conclude with final remarks and directions for future work.

2 Fundamental Concepts

In this section, we present the fundamental concepts required to understand the rest of the paper. In Sect. 2.1, we introduce SVMs and online learning. In Sect. 2.2, we describe the main RE kernels described in the literature.

2.1 SVMs and Online Learning

SVMs [9] constitute one of the most popular supervised ML techniques. A SVM is a binary classification model that uses an hyperplane to separate the data that belongs to two distinct classes. The biggest challenge when generating SVMs is to find an optimal hyperplane that properly separates the data in two classes. An hyperplane is represented by its normal vector $\mathbf{w}$ and the goal is to find the optimal vector, $\mathbf{w}^*$. For that, we define an objective function $f(S; \mathbf{w})$ that represents the proximity between two vectors $\mathbf{w}$ and $\mathbf{w}^*$ for a given training dataset S. Then, we apply optimization techniques [12] (e.g., gradient descent) to this objective function, to obtain a vector $\mathbf{w}$ close to the optimal vector $\mathbf{w}^*$.

The training dataset is composed of data instances. These data instances are pairs of entities. A data instance $(\mathbf{x}, y)$ from the training dataset S is constituted by an input vector $\mathbf{x}$ and a scalar y that represents the binary class the input vector belongs to. A position of the vector $\mathbf{x}$ corresponds to a relevant feature of the data instance. A SVM uses the vector $\mathbf{w}$ to predict the class of a data instance $\mathbf{x}$ by computing the inner product $\langle \mathbf{w}, \mathbf{x} \rangle$. The objective function is then given by $f(S; \mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \sum_{(\mathbf{x},y)\in S} loss(\mathbf{x}, y; \mathbf{w})$ where $||\mathbf{w}||$ is a regularization method applied to $\mathbf{w}$ that penalizes the model complexity degree (i.e., rewards fewer patterns in a model) to prevent overfitting, and $loss(\mathbf{x}, y; \mathbf{w})$ is a loss function given by $\max(0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle)$ that returns a penalty value if the model predicts the wrong class of a data training instance $\mathbf{x}$. In order to evaluate the model's predictive performance against the training dataset S, the objective function applies the loss function to each training data instance $(\mathbf{x}, y) \in S$ and sums the returned penalty values. The λ parameter is introduced by the user in order to adjust the two model components: the regularization and the loss function computed for the training dataset. In practice, it balances the model complexity degree with the model predictive performance for the training dataset.

In the context of RE, representing a data instance (i.e., a pair of entities and its features) through a vector is not a straightforward task. In order to obtain a vector, we need to transform a complex data structure that represents the data instance into a vector. In this process, we have to generalize and omit several

features that, comparatively to the original structure, result in weak representations of the data. A more satisfactory alternative than to relay on a single vector is to represent the data instances by more complex and informative data structures (i.e., graphs or sequences). A kernel [11] can then be used to compare two data instances represented by complex data structures. Kernels return a similarity value between two complex structures and they behave as inner products in non-linear feature spaces. Moreover, optimization techniques can compute a vector $\mathbf{w}$ close to the optimal vector $\mathbf{w}^*$ by calculating inner products between vectors without accessing directly to the vector positions. Since the optimization techniques do not directly access vector positions, we can replace the inner products by kernels and generate a kernel based model.

Online learning techniques are ML techniques that use online optimization algorithms to train a ML model. These algorithms find the best set of parameters for an objective function by processing the data instances in an online fashion (i.e., process a data instance at a time), thus enabling us to analyze the ML model at a specific point of the training process. Online optimization techniques are generally faster than other types of optimization techniques [12].

So far, we have presented binary SVM classifiers. However, most classification problems involve more than two classes. These problems are called multi-class classification problems and they are not solved directly using one SVM classifier. We can use heuristics that combine several SVM classifiers, for example *One-VS-One* or *One-VS-All* [9]. In Sect. 3.3, we provide more details about the *One-VS-One* heuristic that we used in this work.

2.2 Relation Extraction Kernels

This section describes three state-of-the-art kernels. Each data instance (i.e., pair of entities) is initially described by a sentence composed of words and the location of the words that compose each entity. This is an initial representation given as input to a pre-processing stage that outputs complex structures to be used by each kernel. In this paper, we considered the following kernels:

Subsequences Kernel (SSK): Bunescu and Mooney [4] developed a kernel based on subsequences. SSK is composed by the sum of three sub-kernels that compare sparse word subsequences (i.e., word sequences not necessarily contiguous) in different locations of a sentence: before, after and between the pair of entities. This kernel is defined by a function that computes the similarity between words based on the number of features that the words have in common (e.g., word stem or word grammatical category).

Shortest Path Kernel (SPK): Bunescu and Mooney [3] also proposed a kernel based on the comparison of dependency graphs. A dependency graph is a directed graph that represents the grammatical dependencies between words in a sentence. Each node represents a word and each edge represents a dependency between the two nodes. SPK has the particularity of using, for comparison, only the sub-graph that contains the shortest path between the entities. Only graphs

with the same number of nodes and similar edges in the shortest path are compared (i.e., for the others, the kernel returns zero). The final value returned by the kernel is calculated by multiplying the values returned by the comparison of the nodes. The nodes that are in the same position of the sub-graphs are compared using the same SSK function that computes the similarity between words. This function counts the number of features that the words have in common.

Bag-Of-N-Grams Kernel (BNK): Giuliano et al. [7] proposed a kernel that combines two simpler kernels, namely: (i) a global context kernel that considers textual information related to the whole sentence and (ii) a local context kernel that considers only the words around the entities. Similarly to SSK, the global context kernel is separated in three sub-kernels that evaluate the similarity of three independent sequences of words located before, between and after the pair of entities. This kernel is based on n-grams instead of subsequences. N-grams are small sets of contiguous words of size n (usually $n = 3$). Each sub-kernel compares the number of common n-grams between two sequences of words. The local context kernel compares the sequence of words centered at each entity and constituted by 6 words (i.e., 3 words before the entity and 3 words after the entity). The similarity between two words is obtained by the number of common features, analogously to SSK and SPK.

Table 1. Statistical characterization of the datasets.

	Sentences	Candidate entities pairs	Entities	Relations	Relation types
AImed	1159	5471	3754	996	1
SemEval	10717	10717	21434	8853	9

3 Benchmark

This section presents the development process of a benchmark for RE kernels. This benchmark enables an impartial evaluation process to assess the suitability of each kernel for the RE task. This evaluation process is composed of the following tasks: (i) pre-processing of datasets described in Sect. 3.1 using the techniques from the tools presented in Sect. 3.2; (ii) training and execution of SVMs based on kernels using the learning techniques described in Sect. 3.3; and (iii) evaluation of the kernels using the measures described in Sect. 3.4.

3.1 Datasets

This section describes the two datasets considered in the benchmark: AImed and SemEval. We chose these datasets because they belong to two distinct domains and they enclose different classification problems. Both datasets are composed of English documents. Table 1 presents a statistical summary for these datasets.

AImed: is composed of 255 Medline article abstracts, 200 of which referring to interactions between proteins, and with a remaining 25 which do not refer to any interaction. In total, AImed contains 5471 proteins pairs, of which 996 correspond to interactions and 4475 correspond to pairs without interaction.

SemEval (Semantic Evaluation) [8]**:** is an ongoing series of evaluations for computational semantic analysis systems. It proposed, in 2010, a classification challenge for relations between entity pairs. The dataset associated to this challenge is composed of 8000 training sentences and 2717 test sentences. Each sentence is clearly identified and contains a single annotated candidate pair. This challenge includes the following nine asymmetric relation types: *Member-Collection*, *Cause-Effect*, *Component-Whole*, *Instrument-Agency*, *Entity-Destination*, *Product-Producer*, *Message-Topic*, *Entity-Origin*, and *Content-Container*.

3.2 Linguistic Pre-processing Techniques

RE techniques generally use various types of linguistic and/or lexico-syntactic annotations. Therefore, it is necessary to pre-process the text in order to obtain these annotations. For pre-processing the datasets, we use the following techniques and tools:

1. **Sentence Segmentation:** Separates the text into sentences. We processed the text with the *Apache OpenNLP*[4] library, which is a tool for natural language processing based on ML.
2. **Tokenization:** Identifies the words of every sentence, keeping their order in the text. We processed the sentences with the *Apache OpenNLP* library.
3. **Token Normalization:** Labels the words with their stem, in such a way that words of the same family have the same label (e.g., *claimed* and *claims* have the same stem, *claim*). We used the *Porter Stemming* algorithm from the Snowball[5] framework, which finds the stem for each word.
4. **Capitalization:** Labels every word with two different representations, transforming them according to two patterns. The first pattern replaces each character by one of four specific symbols, depending on whether it is a capital letter, a lowercase character, a number or another character. In the second pattern, a sequence of characters of the same type is replaced by the symbol associated with the character type followed by the character +. For example, the normalization of the word *"Karachi"* results in *Cccccc* and *Cc+*. We used our own implementation of the two patterns[6].
5. **Part-of-Speech Tagging:** Produces Part-of-Speech (POS) tags and Generic Part-of-Speech (GPOS) tags for each word. A POS tag represents the grammatical class (e.g., *claimed* labeled with verb past tense) that this word has in the text, while a GPOS tag (e.g., verb) represents a high-level grammatical class. We directly map the POS tags into GPOS tag and we used the *Apache OpenNLP* library to obtain the POS tags.

[4] http://opennlp.apache.org/
[5] http://snowball.tartarus.org/
[6] http://web.tecnico.ulisboa.pt/joaoplmpereira/Capitalization.html

6. **Dependency Analysis:** Produces a dependency tree or graph for each sentence. We used the *Stanford CoreNLP*[7] library, which contains a dependency parser for identifying syntactical dependencies between words.

The *Apache OpenNLP* library uses a Maximum Entropy Model (MEM) [2] previously trained with data in English for each pre-processing task: a MEM for sentence segmentation[8], a MEM for tokenization[9], and a MEM for POS tagging[10]. We use sentence segmentation, tokenization, and dependency analysis to produce data structures that can be compared by kernels. Token normalization, capitalization and POS tagging are optional pre-processing steps introduced in the form of word features that improve the accuracy of the kernels.

3.3 Learning Techniques

We used an efficient online learning technique called Pegasos [12] to train the SVMs. This involves an iterative process that performs multiple passes over the training data.

In RE problems where the goal is to search for a single relation type (e.g., protein interaction extraction from AImed), it is sufficient to use a binary classifier. However, if the problem involves searching for various types of relations (e.g., the RE task in SemEval), it is necessary to use multi-class classifiers. To produce a multi-class classifier using binary classifiers, we used an *One-VS-One* heuristic. This technique makes use of as many binary classifiers as pairs of classes. We train each binary classifier with the subset of the training data that is annotated with the corresponding pair of classes. In the test phase, a data instance is evaluated by all the classifiers. Each classifier votes in the class associated to the binary class that it assigns to the data instance. Then, we choose the most voted class as the label.

3.4 Metrics

In this section, we present the metrics we use to evaluate the different kernels:

Convergence: Measures the proximity of a SVM to the optimal SVM. This proximity is given by applying the objective function for SVMs with a training dataset and the $\mathbf{w}$ of a SVM as arguments, as described in Sect. 2.1. This metric enables us to understand the impact of changing the parameters of the learning phase (i.e., λ and the number of iterations that indicate the number of passes over the data) on the generated SVMs.

Quality of Binary Classification: The measures commonly used to evaluate classification techniques are: precision, recall, and the F_1-measure. Defining r as

[7] http://nlp.stanford.edu/software/corenlp.shtml
[8] http://opennlp.sourceforge.net/models-1.5/en-sent.bin
[9] http://opennlp.sourceforge.net/models-1.5/en-token.bin
[10] http://opennlp.sourceforge.net/models-1.5/en-pos-maxent.bin

a relation type, *precision* is the fraction of correctly extracted relations of type r over the total number of extracted relations of type r. *Recall* is the fraction of correctly extracted relations of type r over the total number of relations of type r present in the dataset. These measures are usually contradictory: by increasing the recall value, we reduce the precision value and vice versa. To globally evaluate a classification technique, it is necessary to combine these two measures. A solution to combine these two scores is to use the F_1-*measure* that corresponds to the harmonic mean of precision and recall.

Quality of Multi-class Classification: In multi-class classification, we use new measures obtained by averaging the values of precision and recall over the multiple classes, namely macro-averages and micro-averages. *Macro-averages* calculate directly the system precision or recall averages for the various classes. To calculate the *Micro-averages*, first, we count, for all relations of type r, the values used in the formulas of precision and recall: the number of correctly extracted relations, the total number of extracted relations, and the total number of relations present in the dataset. We then compute the Micro-averages by using these summed values into the precision or the recall formulas.

Execution Times: We measure two execution times: the time to train a SVM and the time required to predict the class type for a pair of entities. Execution times are extracted in nano seconds of the CPU time. Then, we convert them into other units for easier comparison (i.e., seconds, minutes or hours).

4 Benchmark Results

In this section, we present the configurations and the results of the experiments we performed using the benchmark presented in Sect. 3. In Sect. 4.1, we describe the settings used in the experiments. In Sect. 4.2, we present the analysis of the convergence values. In Sect. 4.3, we analyze the quality of the results obtained for the various models. In Sect. 4.4, we analyze the execution times for training a SVM and for classifying a data instance.

4.1 Setup

The experimental setup that we have used is as follows:

Datasets: We used the datasets described in Sect. 3.1, namely AImed and SemEval. Both are pre-processed through the techniques and tools described in Sect. 3.2. The AImed dataset was split into 10 folds over which we applied a cross-validation process. The AImed dataset contains a single type of relation, which means that extracting relations in this dataset is a binary classification problem. The SemEval competition provided distinct training and testing splits. Thus, for SemEval, it was not necessary to use cross-validation. Extracting relations from SemEval is a multi-class problem. The data is annotated with 9 types of asymmetric relations, which gives a total of 19 distinct classes (two for each type plus another for a non-relation).

Learning Techniques: We used the learning techniques described in Sect. 3.3. In particular, we implemented the Pegasos technique extended to work directly with kernels[11]. We used the extended version of Pegasos to train the SVMs: one for the AImed dataset; and one for each of the 172 binary classifiers used by the *One-VS-One* heuristic for the SemEval dataset.

Kernels: We used the *SSK*, *SPK* and *BNK* kernels described in Sect. 2.2, since they use different data structures to represent the data instances.

Parameters: The parameter λ, which controls the importance of the regularization versus the loss function in the SVM training, took values in $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. The number of iterations T, which is introduced in the online learning technique to indicate how many passages should be made over the training instances, took values in $\{50, 100, 150, 200\}$.

Software and Hardware: We developed the benchmark using the framework for RE named REEL [1], which provides an implementation of SSK, SPK and BNK. We performed the experiments on a machine with an Intel Core i5 CPU M 460, 2.53 GHz and 4 GB of memory RAM.

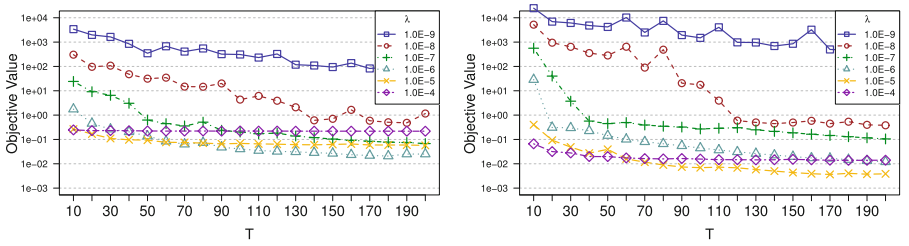
4.2 Convergence

In this section, we evaluate the impact of the λ and T parameters over the training of SVMs with RE kernels. To do this, we analyze the variation of the objective function values with these parameters (see Fig. 1). The experiments show that there is no substantial difference in the results obtained for different kernels. Therefore, we only present the results obtained for BNK.

In general, the number of iterations required to stabilize the objective function values increases when we decrease the λ parameter values. In fact, for small values of λ, we do not reach stable values for the objective function. Regarding the number of iterations T, the objective function values start to slowly decrease after iteration number 100, and stabilize after iteration 150. For the AImed dataset (see Fig. 1(a)), the models that obtain the lowest objective function values are the models trained with a λ value of 10^{-6}. For the SemEval dataset, the models that obtain the lowest objective function values are the models trained with a λ value of 10^{-5}. This situation occurs because, for datasets in technical language domains such as the AImed dataset, models tend to be more complex (i.e., enclose more patterns), since it is difficult to find a suitable generalization (i.e., few patterns that can explain the data).

From the results reported in this section, we conclude that: (i) models trained with very low values for the λ parameter obtain worse results in terms of convergence when compared to higher values, and (ii) after 150 iterations, there was no significant variation in the objective function values.

[11] http://web.tecnico.ulisboa.pt/joaoplmpereira/OnlineLearning.html

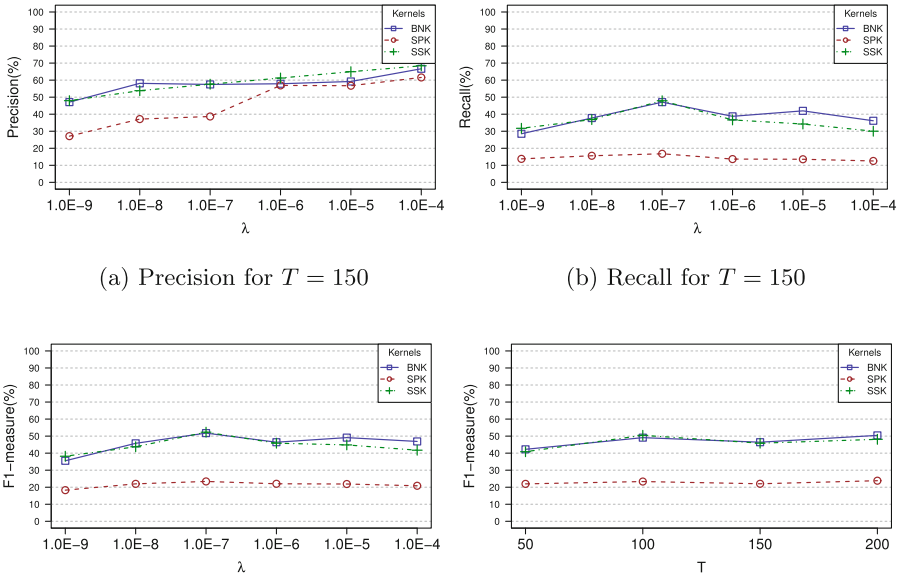


(a) (b)

Fig. 1. Variation of the objective function values for BNK models with parameters λ and T. (a) AImed; (b) Binary classifier for the pair of entities *Content-Container(e2,e1) vs Message-Topic(e1,e2)* in the SemEval dataset

4.3 Quality of the Obtained Extractions

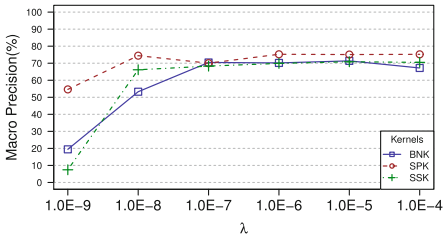
In this section, we evaluate the quality of the results obtained by the SVMs using the kernels BNK, SPK and SSK. We begin by evaluating the impact of the variation of the λ and T parameters in the quality of SVMs using the AImed dataset (see Fig. 2), and then using the SemEval dataset (see Fig. 3).



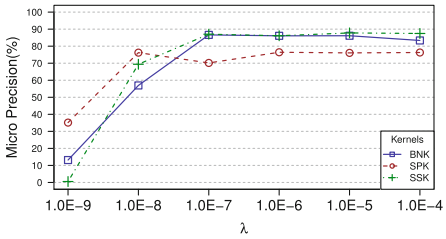
(a) Precision for $T = 150$ (b) Recall for $T = 150$

(c) F_1-measure for $T = 150$ (d) F_1-measure for $\lambda = 10^{-6}$

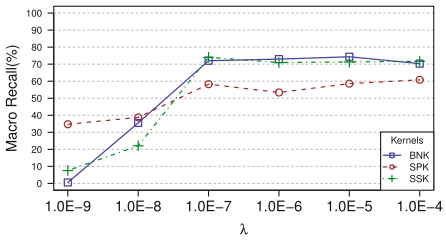
Fig. 2. Variation of the quality of the results obtained through SVMs with kernels BNK, SPK and SSK, for the AImed dataset.



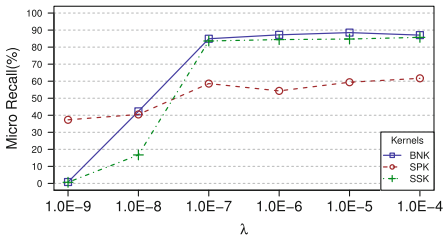
(a) Macro-average of precision



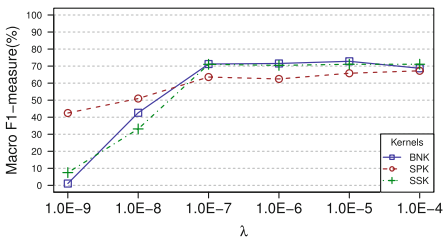
(b) Micro-average of precision



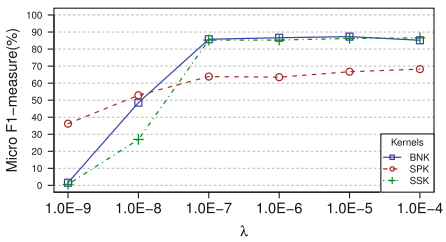
(c) Macro-average of recall



(d) Micro-average of recall



(e) Macro-average of F_1-measure



(f) Micro-average of F_1-measure

Fig. 3. Variation of the quality of the results obtained through SVMs over the SemEval test dataset for kernels BNK, SPK and SSK in $T = 150$.

For both datasets, BNK and SSK produce similar values for the F_1-measure (less than 5 % difference), as shown in Figs. 2(c) and (d), 3(e) and (f). The precision values of SSK are slightly better than the ones of BNK (from 1 % to 5 %) as observed in Figs. 2(a), 3(a) and (b), but the opposite occurs for the recall values (see Figs. 2(b), 3(c) and (d)). Generally, BNK presents more balanced precision and recall values than SSK. In order to explain the differences observed we need to consider the data structures that were used. Both kernels analyze the sentences, splitting them in a similar way. However, in SSK, the features are structured into subsequences which assign a fixed order to the words. BNK

is more flexible because it uses data structures with sets of n-grams, which are composed of n-grams independently of their position in the sentence.

In general, in both datasets, SPK gets worse results than the remaining kernels (e.g., between 5 % and 20 % less). For the AImed dataset, this difference is more significant than in the SemEval dataset. Since the precision values obtained by SPK are comparable to those of the other two kernels (see Figs. 2(a), 3(a) and (b)), the differences in terms of quality derive from its low recall. SPK obtains the best macro-average of precision for SemEval. SPK obtains good results in terms of precision because it compares shortest paths and it assigns zero to paths with different size. So, this kernel becomes more inflexible to compare data samples. For AImed, SPK obtains low quality results than SSK and BNK (less than 30 %), because this dataset is composed of technical medical documents, and the dependency parser used by SPK was trained with news articles. Therefore, the dependencies found are inappropriate for this dataset and consequently lead to the low quality performance of SPK. Due to the inadequate use of the dependency parser, we consider unfair to compare SPK with the other kernels in the AImed dataset.

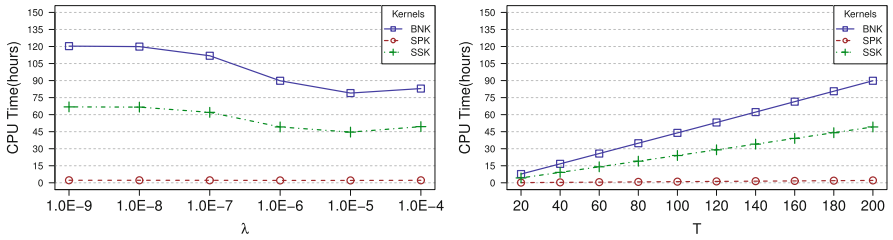
In this section, we conclude that: (i) BNK and SSK have similar behaviors; (ii) BNK is better in recall and returns more balanced precision and recall values; (iii) SPK is the kernel that obtains, in general, worse results, but nevertheless, stands out for the SemEval dataset in terms of precision.

4.4 Execution Time

In this section, we compare the kernels in terms of training and testing execution times. The pre-processing execution times were not considered because they heavily depend on the pre-processing techniques and tools used. For calculating the training execution times, we consider an estimate of the CPU time. We calculate this estimate by multiplying the number of calls to the kernel with the average execution time of comparing two data instances. We did not take the CPU time of the online optimization technique into consideration, because it is significantly lower than the time needed to calculate the kernels (i.e., less than 1 % of the total execution time). For the AImed dataset the execution times correspond to the classifier training time. For the SemEval dataset, the execution time is the sum of the training execution times of all classifiers used in the multi-class problem.

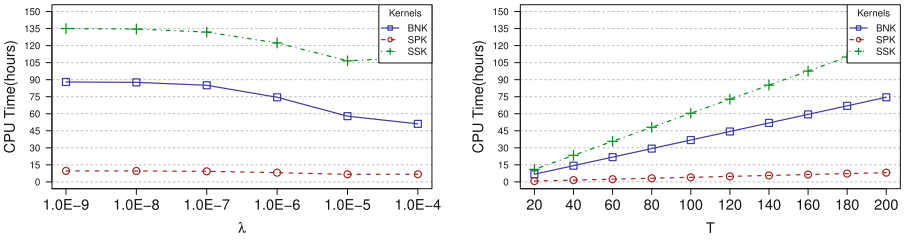
Figures 4 and 5 report the training execution times for AImed and SemEval, respectively. For both datasets, SPK is the fastest kernel. However, the pre-processing step of SPK is usually much more expensive than the pre-processing step of the other kernels, because it needs a dependency analysis process. SPK only takes into account the shortest path between the entities in the dependency graph therefore: (i) it compares less sentence features than the other two kernels; (ii) it ignores a large quantity of comparisons since it only compares sentences whose number of nodes in the shortest path between the two entities is equal.

For the AImed dataset, SSK is faster than BNK. However, for SemEval, the inverse situation occurs. This difference is due to the size of the sentences in each



(a) $T = 200$ (b) $\lambda = 10^{-6}$

Fig. 4. Average execution times for training over the 10 folds of AImed dataset.



(a) $T = 200$ (b) $\lambda = 10^{-6}$

Fig. 5. Execution times for training over the SemEval dataset.

of the datasets i.e., the average size of a sentence in AImed is significantly higher than in SemEval. SSK compares more subsets of tokens than BNK, but it uses a dynamic programming algorithm that performs faster comparisons, even when the sentence size increases. BNK is very penalized when the sentence length and the diversity of words increase, especially in the AImed dataset.

Regarding the variation of the λ parameter, we conclude that there is a relationship between the λ values and the training execution times. Models trained with lower λ values have lower execution times (see Figs. 4(a) and 5(a)). In Sect. 4.2, the λ parameter was associated with the stability of the objective function and convergence of the model. In fact, we observed that the objective function values stabilize for low values of λ. When we make the λ parameter constant and vary the number of iterations, we observe that the execution times grow linearly with the number of iterations (see Figs. 4(b) and 5(b)).

The testing execution times are generally constant for both datasets. With the AImed dataset, the testing execution time is 168 ms for BNK, 8 ms for SPK, and 290 ms for SSK. For the SemEval dataset, it takes 11 ms for BNK, 0.68 ms for SPK, and 6 ms for SSK to classify each data instance. Not surprisingly, the results obtained are similar to the training phase results.

In this section, we conclude that: (i) SPK is the fastest in training and testing, (ii) BNK and SSK have execution times with the same orders of magnitude. However, (iii) SSK is faster than BNK in datasets of technical domains, such as

AImed, and (iv) BNK is faster than SSK for datasets in the news domains such as the SemEval dataset.

5 Discussion and Future Work

In this paper, we proposed a benchmark to compare kernels for RE tasks, in which we train and analyze SVMs leveraging the kernels to be compared. We conducted an extensive experimental analysis using our benchmark over three state-of-the-art kernels, which enabled us to take the following main conclusions: (i) SVMs stabilize after $T = 100$ iterations for models trained with lower values of λ; (ii) BNK and SSK have a similar performance (i.e., quality and execution time) for the AImed and SemEval datasets; (iii) SPK obtains high quality results only when the dependency parser is trained with data of the same domain as the dataset; (iv) SPK is the kernel that has lowest execution times.

As future work, we plan to develop techniques that are able to effectively combine the various kernels for RE. This method should be domain independent and should be used with different sets of kernels.

Acknowledgements. We would like to thank Gonçalo Simões for the fruitful discussions, and for advice on preliminary versions of this paper.

This work was supported by *Fundação para a Ciência e a Tecnologia*, under Project UID/CEC/50021/2013, and under Project DataStorm (ref. EXCL/EEI-ESS/0257/2012).

References

1. Barrio, P., Simões, G., Galhardas, H., Gravano, L.: REEL: a relation extraction learning framework. In: JCDL (2014)
2. Berger, A.L., Pietra, V.J.D., Pietra, S.A.D.: A maximum entropy approach to natural language processing. Comput. Linguist. **22**, 39–71 (1996)
3. Bunescu, R., Mooney, R.J.: A shortest path dependency kernel for relation extraction. In: HLT-EMNLP (2005)
4. Bunescu, R., Mooney, R.J.: Subsequence kernels for relation extraction. In: CoNLL (2006)
5. Chinchor, N.A.: Named entity task definition. In: MUC-7 (1998)
6. Doddington, G.R., et al.: The automatic content extraction (ACE) program - tasks, data, and evaluation. In: LREC (2004)
7. Giuliano, C., Lavelli, A., Romano, L.: Exploiting shallow linguistic information for relation extraction from biomedical literature. In: EACL (2006)
8. Hendrickx, I., et al.: SemEval-2010 task 8: multi-way classification of semantic relations between pairs of nominals. In: SemEval (2010)
9. Hsu, C.W., Lin, C.J.: A comparison of methods for multiclass support vector machines. IEEE Trans. Neural Netw. **13**, 415–425 (2002)
10. Marrero, M., Sanchez-Cuadrado, S., Lara, J.M., Andreadakis, G.: Evaluation of named entity extraction systems. Res. Comput. Sci. **41**, 47–58 (2009)
11. Sarawagi, S.: Information extraction. Found. Trends Databases **1**, 261–377 (2008)
12. Shalev-Shwartz, S., Singer, Y., Srebro, N.: PEGASOS: primal estimated sub-GrAdient SOlver for SVM. In: ICML (2007)