Supervised Learning for Relationship Extraction From Textual Documents

MSc Thesis
Extended Summary

João Pedro Lebre Magalhães Pereira*
*Departamento de Engenharia Informática and INESC-ID
Técnico Lisboa, ULisboa
Lisbon, Portugal
joaoplmpereira@ist.utl.pt

Abstract—**Information Extraction (IE) is the task of automatically extracting structured information from unstructured data, aiming to facilitate the use of said data by other applications. A typical sub-problem is the extraction of relationships from textual documents, which aims at identifying and classifying the relationships expressed between entities mentioned in the texts. Supervised machine learning techniques can be applied to relationship extraction. We used Support Vector Machines (SVM), which we trained with basis on online methods. Two specific modeling choices have been tested. The first one is a simple online solution that trains SVM models considering a single kernel. The second approach is based on the idea of online multiple kernel learning. With existing datasets and common pre-processing tools, we formulated a benchmark, which was then used to evaluate kernel-based methods. We then implemented state-of-the-art kernels, specifically designed for relationship extraction. The results show that an online multiple kernel learning solution obtains the best performance and that multiple kernel online solutions can perform better than heuristics solutions using a linear combinations of the same kernel sets.**

I. Introduction

Information Extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured textual documents. The structured information that we aim at extracting may correspond to entities, relationships, and/or attributes that describe said entities. Although IE concerns with processing human language texts by means of Natural Language Processing (NLP), traditionally relying on rule-based NLP methods, different areas of study have proposed alternatives and improvements. Examples include machine learning, information retrieval, databases, web information systems, and document analysis.

A specific IE sub-task is Relationship Extraction (RE). The objective of RE is to find semantic relationships between entities referenced in the text. Although more than two entities can participate in a relationship, this article focuses only on relationships between pairs of entities. For instance, from the sentence *Haifa, located 53 miles from Tel Aviv, will host ICML in 2010*, and considering the pair of entities *Haifa* tagged as a location and *ICML* as an organization, a system for RE should be able to find the relationship of *host* between *Haifa* and *ICML*.

The process that leads to relationship extraction starts with data pre-processing. A first step structures the unstructured text based on lexical, morphological and syntactic analysis. Then, an entity extraction step is needed to identify the candidate pairs of entities. Finally, a relationship extraction step is responsible for finding relationships between the entities identified. Relationship extraction is typically addressed as a classification problem for pairs of entities.

Different methods have been proposed to address RE. In contrast to the more traditional rule-based methods, this article explores Machine Learning methods. Machine Learning (ML) methods aim at analyzing patterns in data for the purpose of training a model, in this case for the extraction of relationships. ML methods for RE can be feature-based or kernel-based. Feature-based methods make a direct extraction of characteristics (also known as features) from the data, which are then used to train models. Kernel-based methods rely on functions that make use of characteristics from the data to compute a similarity value between pairs of data instances. Support Vectors Machines (SVMs) are machine learning models that can be trained using both a feature-based or a kernel-based method, and thus they are a natural choice for this work.

RE has been addressed in the scope of IE competitions that aim at evaluating different methods for solving a particular task. These competitions have resulted in a comparison between different methods, but they still have a few problems. Firstly, they focus on a limited group of relationship extraction methods. Secondly, the data sets that were used focus on particular domains and use specific types of relationships. Another issue concerns with the availability and technical characteristics of the pre-processing tools that are used for linguistic analysis.

This article develops a benchmark for relationship extraction, which attempts to overcome some of the issues found in the previous competitions focusing on the RE task. We were inspired by the work of Marrero et al. [1], who produced a benchmark for Named Entity Recognition (NER) systems. They developed their own validation metrics, as well as their own test and training corpora. This article aims at producing a similar benchmark for relationship extraction, using publicly available datasets and a common set of pre-processing tools. With this benchmark, our goal is to compare several kernel-based supervised machine learning methods from the literature. Moreover, using the benchmark, we aimed at proving the advantages of using an online approach for multiple kernel learning to train SVMs, by combining a set of kernels from state-of-the-art solutions for relationship extraction.

II. Related Work

This section presents current state-of-the-art kernel-based methods for relationship extraction.

A. Subsequence Kernel

Bunescu and Mooney [2] presented a kernel-based method based on subsequences. In a sentence, the word defining the relationship can appear not only between the two entities, but also before (e.g., activation of *P1* by *P2*) and/or after (e.g., *P1* and *P2* interact). Therefore, to take care of these cases, this kernel is separated into the three following aspects:

Fore–Between: Words before and between the two entity mentions are simultaneously used to express the relationship;

Between: Only words between the two entities are essential for exposing the relationship;

Between–After: Words between and after the two entity mentions are simultaneously used to express the relationship.

To explain the cases Fore-Between, Between, and Between-After, we first introduce a kernel from Lodhi et al. [3]. These authors describe a kernel to be applied to two text documents, where the idea is to compare them by means of the substrings they contain. The more substrings in common, the more similar the documents are. These substrings do not need to be contiguous, and the degree of contiguity of one such substring in a document determines how much weight it will have in the final kernel computation.

Formally, let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including the empty sequence. For strings s and t, we denote by $|s|$ and $|t|$ the length of the string $s = s_1...s_{|s|}$ and $t = t_1...t_{|t|}$. The sequence $s[i : j]$ is the contiguous subsequence $s_i...s_j$ of s. Let $\mathbf{i} = (i_1, ..., i_{|\mathbf{i}|})$ be a sequence of $|\mathbf{i}|$ indices in s, in ascending order. We define the length $l(\mathbf{i})$ of the index sequence $\mathbf{i}$ in s as $i_{|\mathbf{i}|} - i_1 + 1$. Similarly, $\mathbf{j}$ is a sequence of $|\mathbf{j}|$ indices in t.

Bunescu and Mooney [2] formalized the kernel shown in Equation 1, based on two fixed index sequences i and j, being both of length n. Given two feature vectors $x, y \in \Sigma$, let $c(x, y)$ denote the number of common features between x and y. The parameter λ is used as a decaying factor that penalizes longer subsequences. For sparse subsequences, this means that wider gaps are more penalized, which is exactly the desired behavior.

$$K_n(s,t,\lambda) = \sum_{\mathbf{i}:|\mathbf{i}|=n} \sum_{\mathbf{j}:|\mathbf{j}|=n} \prod_{k=1}^{n} c(s_{\mathbf{i}_k}, t_{\mathbf{j}_k}) \lambda^{l(\mathbf{i})+l(\mathbf{j})} \quad (1)$$

Given two strings s and t , let x_1 and x_2 be the entities in s, and let y_1 and y_2 the entities in t. Let us define s_b as the tokens between the entities of s, s_f the tokens before the entities, and s_a the tokens after. The same is applied to

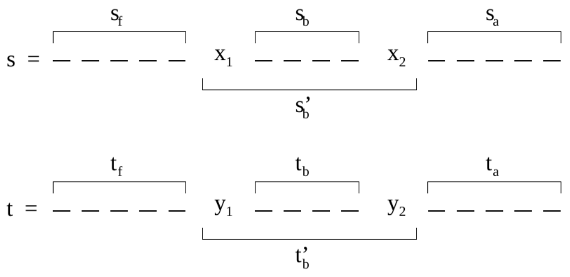

Figure 1: Computation of the final sequence kernel.

string t. This notation is represented graphically in Figure 1. The computation of the final kernel $rK(s,t)$ is expressed in Equation (6), which will contain the sum of the three aspects, Fore-Between as in Equation (3), Between as in Equation (4) and Between-After as in Equation (5). Both aspects share the computation of the substrings between the entities as in Equation (2).

$$bK_i(s,t) = K_i(s_b, t_b, 1) \cdot c(x_1, x_2) \cdot c(y_1, y_2) \cdot \lambda^{l(s_b)+l(t_b)+4} \quad (2)$$

$$fbK(s,t) = \sum_{i,j} bK_i(s,t) \cdot K_j(s_f, t_f), \quad 1 \le i, 1 \le j, i+j \le fb_{max} \quad (3)$$

$$bK(s,t) = \sum_{i} bK_i(s,t), \ 1 \le i \le b_{max} \quad (4)$$

$$baK(s,t) = \sum_{i,j} bK_i(s,t) \cdot K_j(s_a, t_a), \quad 1 \le i, 1 \le j, i+j \le fa_{max} \quad (5)$$

$$rK(s,t) = fbK(s,t) + bK(s,t) + baK(s,t) \quad (6)$$

B. Shortest Path Dependency Kernel

Bunescu and Mooney [4] presented a dependency kernel with the particularity of using the shortest path between entities. This kernel also tends to be fast when compared to other kernels that use dependencies graphs or trees.

In this kernel, the dependency path is represented as a sequence of tokens linked with arrows that indicate the orientation of each dependency. The tokens are a simple form of information so they can then be enrich with word categories called features. The authors used part-of-speech (POS) tags, general POS tags and the entity type for the entities tokens.

For two dependency paths x and y, the kernel computation is obtained by multiplying every number of common features between two words in the same position. Formally, if $x = x_1, x_2...x_m$ and $y = y_1, y_2, ..., y$ are two relation examples, where x_i denotes the set of word classes corresponding to position i, then the kernel is computed as in Equation (7).

$$K(x,y) = \begin{cases} 0, & m \neq n \\ \prod_{i=1}^{n} c(x_i, y_i), & m = n \end{cases} \tag{7}$$

where $c(x_i, y_i)$ is the number of common word classes between x_i and y_i.

As an examples, let us consider two phrases:

1) *his actions in Brcko* , and
2) *his arrival in Beijing.*

Their corresponding dependency path are:

1) his $\rightarrow$ actions $\leftarrow$ in $\leftarrow$ Brcko , and
2) his $\rightarrow$ arrival $\leftarrow$ in $\leftarrow$ Beijing.

Their representation as a sequence of set of word classes is given by:

1) $x = [x_1, x_2, x_3, x_4, x_5, x_6, x_7]$, where $x_1 = \{$ his, PPP, PERSON $\}$, $x_2 = \{\rightarrow\}$, $x_3 = \{$ actions, NNS, Noun $\}$, $x_4 = \{\leftarrow\}$, $x_5 = \{$in, IN$\}$, $x_6 = \{\leftarrow\}$, $x_7 = \{$ Brcko, NNP, Noun, LOCATION $\}$
2) $x = [y_1, y_2, y_3, y_4, y_5, y_6, y_7]$, where $y_1 = \{$ his, PRP, PERSON $\}$, $y_2 = \{\rightarrow\}$, $y_3 = \{$ arraival, NN, Noun $\}$, $y_4 = \{\leftarrow\}$, $y_5 = \{$in, IN$\}$, $y_6 = \{\leftarrow\}$, $y_7 = \{$ Beijing, NNP, Noun, LOCATION $\}$

The kernel computation of x and y is given from Equation (7) as $K(x,y) = 3 \times 1 \times 1 \times 1 \times 2 \times 1 \times 3 = 18$.

C. Bag-Of-N-Grams Kernel

Giuliano et al. [5] presented a kernel that combines both global and local context. The global context considers the whole sentence, while the local context takes into account the tokens around the entities.

The two main kernels, global and local can be subdivided in three and two kernels, respectively. Each sub-kernel is calculated using the 2-norm given in Equation (8), commonly used in other kernels in the literature.

$$K(s,t) = \frac{< \phi(s), \phi(t) >}{||\phi(s)||\ ||\phi(t)||} \tag{8}$$

Similar to the subsequence kernel previously presented in Section II-A, the global context kernel is the sum of three kernels. These kernels are similar but they are applied over different set of tokens. The pair of entities defines the borders, If the tokens are between, after and between, or before and between both entities.

The global context kernel is distinct from the subsequences kernel by using a bag-of-words instead of subsequences. A bag-of-words considers only the frequency of a token in a specific sequence or subsequence but it does not take into account its position. In this kernel, we have 3 bags-of-words one for each setup.

Formally, each bag-of-words kernel can be represented in Equation (9) and the final global kernel obtained by replacing them in Equation (8).

$$\phi_P(R) = (tf(t_1, P), tf(t_2, P), ..., tf(t_l, P)) \in \Re^l \tag{9}$$

In the formula, the function $tf(t_i, P)$ returns the frequency with which a token t_i is used in P.

Moreover, the authors found that it was useful to use the type of the interacting entities. Therefore in an attempt to find the type of an entity, the tokens around each entity are used to classify each entity. Two more kernels were developed, one for each entity, called the left local context and the right local context. Each local context uses a window around the entity and can also make use of some features. For instance, the authors use the token, the lemma of the token, part of speech tag of the token and orthographic features.

Given a sample R and a local context $L = t_{-w}, ...t_{-1}, t_0, t_{+1}, ..., t_{+w}$, a local context kernel can be defined as a vector by Equation (10).

$$\phi_L(R) = (f_1(L), f_2(L), ..., f_m(L)) \in \{0,1\}^m \tag{10}$$

In the formula, f_i is a feature function that returns 1 if the feature exists or 0 if it does not. Similar to the global context kernel, the local kernel is then computed by using the two vectors of each entity in Equation (8).

III. Relationship Extraction with Multiple Kernel Learning

The relationship extraction problem can be solved using different methods. The focus of this work is on supervised machine learning methods. We can extract relationships using a machine learning technique that classifies a candidate pair of entities with a relationship type.

We use a machine learning classifier, named Support Vector Machines, that weights the samples from the training data. In its basic form this classifier only works with binary problems. Therefore, for the cases where multiple relationship types are contained in a corpora, it is necessary to reduce the relationship extraction problem to a binary classification process. To solve this problem we assign a SVM classifier for each pair of classes (the set of classes include all relationship types plus a non-relationship class). Then, classification is performed by a max-wins voting strategy. When predicting a relationship, every classifier assigns the candidate sentence to one of the two classes, then the vote for the assigned class is increased by one. Finally, the class with the most votes determines the candidate pair of entities.

A characteristic of SVMs is that they can use kernels which are comparison functions between two candidate instances. In relationship extraction, a kernel allow us to introduce complex structures as input. Several methods for relationship extraction from the current state-of-the-art make use of kernels. Our first contribution in Section III-A is the implementation of an online learning approach, and then, developed to use a kernel.

In Section III-B we present our second contribution which includes the development and implementation of a multiple kernel learning approach. It is based on the first contribution because both use an online approach and train SVMs, but while the first contribution accepts only one kernel, our multiple kernel learning approach accepts and trains weights for a set of kernels.

The third contribution is in Section III-C where we detail the benchmark used to compare different kernels from the state-of-the-art, as presented in Section II. It also describes the datasets, the NLP preprocessing, and the evaluation procedure.

A. *Online Learning of Support Vectors Machines*

In order to solve the optimization problem associated to the training of SVMs, different approaches have been considered in the past. Some of them are associated to the training of online learning algorithms, which are iterative algorithms that solve the SVM problem through the successive analysis of new training instances. An advantage of these online approaches is that the model can be updated with new data. One method in this category is based on gradient descent and is known by the name of Pegasos [6]. Pegasos is a fast solver that is known to perform well when compared to other solutions. Therefore, we propose an approach based on Pegasos for solving the optimization problem using kernels. Although originally this method was proposed to solve a SVM problem with features and without considering a bias term, the authors suggested extensions in which a bias and kernels can be added to the Pegasos algorithm. In this section, we will first explain the base algorithm, and then the extensions that have led to the final algorithm used in this work.

The task of learning a support vector machine online is an unconstrained empirical loss minimization problem, with a penalty term for the norm of the classifier that is being learned. Formally, given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ of size m, where $\mathbf{x}_i \in \Re^n$ and $y_i \in \{+1, -1\}$, we want to solve the minimization problem, as in Equation (11), in order to obtain the weighting vector $\mathbf{w}$ that best describes the data.

$$\min_{\mathbf{w}} \left\{ \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{m} \sum_{(\mathbf{x},y)\in S} l(\mathbf{w}; (\mathbf{x}, y)) \right\} \tag{11}$$

The parameter λ corresponds to the weight or importance given to the L2-norm $||\mathbf{w}||^2$ in the objective function, against the loss function of every $(\mathbf{x}, y) \in S$, defined as being $l(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}\rangle\}$. In the literature, instead of a parameter λ, some support vector solvers use a parameter c associated to weighting the loss function. Note that the process of optimizing $\mathbf{w}$ using a L2-norm is called a L2 regularization, usually applied in machine learning to prevent overfitting.

Pegasos was originally designed to work with a mixture of descendant gradients and projections, using a variable k that corresponds to the number of examples that are used to calculate the sub-gradient. Two extremes of the algorithm correspond to setting k equal to the total number of samples, becoming only a projection based approach, or setting $k = 1$, this way behaving as a stochastic gradient method. In our implementation, we used $k = 1$..

Besides the parameter k, the algorithm allows another parameter to be given as input, namely a parameter T that corresponds to the maximum number of iterations to perform. Since the algorithm is iterative it can iterate over the same samples, the T parameter is also the number of times the samples are iterated.

To understand the algorithm, based on the minimization problem in Equation (11), we define the objective function that we want to minimize in Equation (12), where A_t represents a subset of samples with size k.

$$f(\mathbf{w}; A_t) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{k} \sum_{(\mathbf{x},y)\in A_t} l(\mathbf{w}; (\mathbf{x}, y)) \tag{12}$$

As we only consider $k = 1$, the set A_t will always consist of only one sample. Therefore, the sum can be omitted, as in Equation (13):

$$f(\mathbf{w}; (\mathbf{x}, y)) = \frac{\lambda}{2}||\mathbf{w}||^2 + l(\mathbf{w}; (\mathbf{x}, y)) \tag{13}$$

The main part of a SVM algorithm is to minimize the objective function resorting to optimization theory methods. Pegasos is based on a stochastic gradient descent approach. A stochastic gradient descendent defines a learning rate η_t depending on the current iteration, in our case given by $\eta_t = 1/(\lambda t)$. Intuitively, the samples are of greater importance at the beginning of the execution. Since we want to reach convergence in the objective function, after each sample, the new information continuously looses importance. Therefore, the stochastic gradient step can be written as $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t \nabla_{w_t} f(\mathbf{w}; (\mathbf{x}, y))$ where $\nabla_{w_t} f(\mathbf{w}; (\mathbf{x}, y))$ is the sub-gradient of $f(\mathbf{w}; (\mathbf{x}, y))$ at $\mathbf{w}_t$, as in Equation (14).

$$\nabla_{w_t} f(\mathbf{w}; (\mathbf{x}, y)) = \lambda \mathbf{w}_t - y\mathbf{x}, \quad (\mathbf{x}, y) \in A_t \tag{14}$$

In other words, the main objective is to find the optimal vector $\mathbf{w}_t$. After each iteration, $\mathbf{w}_t$ is scaled by $(1 - \eta_t \lambda)$. Moreover, if the sample A_t was not in the right margin of the support vectors, then the vector $y\eta_t x$ is added to $\mathbf{w}$, resulting in a new vector $\mathbf{w}_{t+\frac{1}{2}}$. In case A_t was in the right margin there is no correction to be made, and so $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}$.

After the gradient step, the algorithm applies a projection of $\mathbf{w}_{t+\frac{1}{2}}$ in order to agree with a constrain corresponding to $||\mathbf{w}|| \leq 1/\sqrt{\lambda}$ resulting in the final $\mathbf{w}_{t+1}$ of the iteration.

The algorithm will iterate over the samples until all the samples have been evaluated according to the objective function. The procedure repeats itself T times, or until the algorithm converges.

Considering only the simple version of Pegasos, there are two important limitations in what concerns the specific problem considered in this work. The first limitation is not being defined in Pegasos algorithm a kernel approach of the SVMs, which would enable us to use a kernel-based method instead of a features set. The second limitation is the lack of a bias term b that would augment the weight vector $\mathbf{w}$, the bias term is important when training over unbalanced data. In order to solve these limitations, two extensions were applied to Pegasos.

The first extension uses a bias term b. It means when predicting the result for an instance x, we will instead use $\langle \mathbf{w}, \mathbf{x} \rangle + b$. The new lost function is then defined as in Equation (15)

$$l((\mathbf{w}, b); (\mathbf{x}, y)) = max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)\} \quad (15)$$

To solve the bias limitation, we directly use Equation (15), while not penalizing for b. The new minimization problem is defined in Equation (16).

$$\min_{\mathbf{w}, b} \frac{\lambda}{2} ||\mathbf{w}||^2 + \sum_{(\mathbf{x}, y) \in S} [1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)] \quad (16)$$

The second extension relates to the Pegasos limitation to use kernels. Typical SVM solvers use a dual problem to address the issue of model training when considering kernels, but the Pegasos algorithm minimizes the primal problem. However, it is possible to incorporate kernels within Pegasos. Considering that for all appearances of $\mathbf{w}_t$, it can be written as $\mathbf{w}_t = \sum_{i \in I_t} \alpha_i \mathbf{x}_i$, where I_t is a subset of $\{1, ..., m\}$ and the coefficients α_i are the weights of the corresponding support vectors I_i. The transformation includes changing the inner product operations, using $\langle \mathbf{w}_t, \mathbf{x}_t \rangle = \sum_{i \in I_t} \alpha_i \langle \mathbf{x}_i, \mathbf{x}_t \rangle$, and evaluating the norm of $\mathbf{w}_t$ using $||\mathbf{w}_t||^2 = \sum_{i,j \in I_t} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. The final algorithm, with both extensions, is shown in the pseudocode for Algorithm 1.

```
input : $S, \lambda, T$
initialize : Chose $\alpha_1 = 0$;
for $t = 1, 2, ..., T$ do
    Choose $A_t \subset S$;
    Set $\eta_t = \frac{1}{\lambda t}$ ;
    for $i \subset t$ do
        Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda) \alpha_i$;
    end
    if $\{(x, y) \in A_t : y \sum_{i \in I_t} \alpha_i K(I_i, x) + b < 1\}$ then
        Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda) \alpha_i + \eta_t y$, $(x, y)_i \in A_t$;
    end
    for $l \subset t$ do
        Set $\alpha_{l+1} = min\{1, \frac{1/\sqrt{\lambda}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)}\} \alpha_{l+\frac{1}{2}}$ ;
    end
    Set $b_{t+1} = \frac{1/\sqrt{\lambda}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)} b_{t+\frac{1}{2}}$ ;
end
output: $\alpha_{T+1}$, $b_{T+1}$
```

Algorithm 1: Pegasos algorithm extended to use kernel and bias term.

B. Multiple Kernel Learning for Relationship Extraction

We used different kinds of kernels in this work. Some of them actually correspond to a mixture of simpler kernels. We developed and then evaluated an approach based on multiple kernel learning, in which the idea is to automatically learn the weights associated to a linear combination of multiple kernels. Therefore, we learn weights for the kernels, and learn at the same time the weights of the support vectors.

More formally, we used a procedure that finds an optimal set of kernel weights given by β, where a final kernel value is obtained through $K(x_1, x_2) = \beta_0 K_0(x_1, x_2) + \beta_1 K_1(x_1, x_2) + ... + \beta_N K_N(x_1, x_2)$. The scalar N is the total number of kernels. We can now define a new minimization problem as in Equation (17).

$$\min_{\alpha, \beta, b} \left\{ \frac{\lambda_\alpha}{2} ||\alpha||_2^2 + \lambda_\beta ||\beta||_1 + \frac{1}{m} \sum_{(x,y) \in S} l(\alpha, \beta, b; (x, y)) \right\} \quad (17)$$

Introducing these new ideas into the method explained in Section III-A, we can formulate a new objective function as shown in Equation (18), adding the minimization of the square L1-norm associated to vector β as being $||\beta||_1$.

$$f(\alpha, \beta, b; (\mathbf{x}, y) \in A_t) = \frac{\lambda_\alpha}{2} ||\alpha||_2^2 + \lambda_\beta ||\beta||_1 + l(\alpha, \beta, b; (x, y)) \quad (18)$$

A new loss function is also formulated as $l(\alpha, \beta, b; (x, y)) = \max\{0, 1 - y[\sum_{i \in I_t} \alpha_i \sum_{n=0}^{N} \beta_N K_n(x, I_i) + b]\}$.

While the α vector uses a L2 regularization procedure, we propose to use a L1 regularization on the vector β, this way encouraging models that use fewer kernels. A L1 regularization is the optimization procedure that uses a L1-norm, it is computed through the sum of the absolute values in a vector, in this case the β vector. A stochastic gradient descent step of β can be added to the algorithm. This step is written as in Equation (20), where the sub-gradient of $f(\alpha, \beta, b; (x, y))$ at β_{tn} is in Equation (19).

$$\nabla_{\beta_{tn}} f(\alpha, \beta, b; (x, y)) = y \sum_{i \in I_t} \alpha_i K_n(x, I_i) \quad (19)$$

$$\beta_{tn} = (1 - \eta_t \lambda_\beta) \beta_{tn} - \eta_t \nabla_{\beta_{tn}} f(\alpha, \beta, b; (x, y)), \quad n \in N \quad (20)$$

In a L1 regularization there are two usual problems that we did not have with a L2 regularization. The first one is the difficulty to get a value that is exactly equal to zero, in our case preventing the model from discarding useless kernels. The other problem is that the weights are sensible to the data which means, if a weight at a specific time can get a value different from zero, then it will not be discarded even if in the past it had the value zero for most of the time.

Tsuruoka et al. [7] proposed a method to overcome these two problems in solving a L1 regularization. The authors solved this problem for a L1 regularization of a feature vector, instead we apply it in β vector. The first solution presented solved the problem of the weights hardly getting a zero value. The solution simply demanded a weight to become zero, whenever its value change polarity. To solve the second problem, the authors suggest a cummulative penalty q_{ti}, that takes into account the past weight values. Note that in our solution, the elements of the vector β can not have values that are less than zero.

Defining q_{tn} as being the total L1 penalty that β_n has actually received up to the point $q_{tn} = \sum_{k=1}^{t}(\beta_{k+1n} - \beta_{k+\frac{1}{2}n})$, and u_t as being the absolute value of the total L1 penalty that each weight β could have received up to the point. $u_t = \lambda_\beta \sum_{k=1}^{t} \eta_k$. The final solution for the L1 regularization steps are given by Equation (21) and Equation (22), which substitute the regularization step defined before in Equation (20).

$$\beta_{t+\frac{1}{2}n} = \beta_{tn} + \eta_t \nabla_{\beta_{tn}} f(\alpha, \beta, b; (x, y)), \quad n \in N \qquad (21)$$

$$\beta_{t+1n} = \max(0, \beta_{t+\frac{1}{2}n} - (u_t + q_{t-1n})), \quad n \in N \qquad (22)$$

The final pseudocode is presented in Algorithm 2 .

```
input : $S, \lambda, T$
initialize : Choose $\alpha_1 = 0$ and $\beta_1 i = \frac{1}{\sqrt{\lambda_\beta * |\beta|}}$,
$i \in t = 1$;
for $t = 1, 2, ..., T$ do
    Choose $A_t \subset S$;
    Set $\eta_t = \frac{1}{\lambda_\alpha t}$ ;
    for $i \subset t$ do
        Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda_\alpha)\alpha_i$;
    end
    if $\{(x, y) \in A_t : y \sum_{i \in I_t} \alpha_i K(I_i, x) + b < 1\}$ then
        Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda_\alpha)\alpha_i + \eta_t y$, $(x, y) \in A_t$;
        for $n \subset N$ do
            Set $\beta_{t+\frac{1}{2}n} = \beta_{tn} + \eta_t y \sum_{j \in I_t} \alpha_{tj} K_n(I_j, x)$,
            $(x, y) \in A_t$;
        end
    end
    for $n \subset N$ do
        Set $\beta_{t+1n} = \max(0, \beta_{t+\frac{1}{2}n} - (u_t + q_{t-1n}))$ ;
        Set $q_{tn} = q_{t-1n} + (\beta_{t+1n} - \beta_{t+\frac{1}{2}n})$ ;
    end
    for $l \subset t$ do
        Set $\alpha_{l+1} = min\left\{1, \frac{1/\sqrt{\lambda}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)}\right\} \alpha_{l+\frac{1}{2}}$
        ;
    end
    Set $b_{t+1} = \frac{1/\sqrt{\lambda_\alpha}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)} b_{t+\frac{1}{2}}$ ;
    for $n \subset N$ do
        Set $\beta_{t+1n} = min\left\{1, \frac{1/\sqrt{\lambda_\beta}}{\sum_m^N \beta_m}\right\} \beta_{t+\frac{1}{2}n}$ ;
    end
end
output: $\alpha_{T+1}$, $b_{T+1}$
```

Algorithm 2: A Multiple Kernel Online Learner.

C. A Benchmark for Relationship Extraction Methods

Analogously to the benchmark for NER systems presented by Marrero et al. [1], this section describes the process involved in developing a benchmark for relationship extraction methods. This process can be separated to sub-processes. The first sub-process converts the datasets into a common format. The second sub-process takes care of the raw text contained inside the corpus. The raw text is unstructured and so, a pre-processing step is necessary to structure it. The pre-processing uses state-of-art techniques incorporated in existing open-source Natural Language processing tools. The third sub-process uses different relationship extraction methods form the state-of-the-art. These methods are machine learning techniques which make use of the pre-processed structures and annotated relationships to train a model. The trained model is then used to predict new instances of relationships given pairs of entities. The final sub-process evaluates the implemented techniques with metrics.

1) Corpora Available for Validation Experiments: This section describes two corpora used that were selected from available datasets from conferences and competitions. Table I presents a statistical characterization of the datasets.

AImed[1] is a protein-protein interaction dataset. It is a single relationship dataset, which means it covers only the problem of predicting whether there is a relationship, from a candidate pair of entities.

Semantic **Eval**uation [8] is an ongoing series of evaluations of computational semantic analysis systems. This competition, in 2010, considered a relationship extraction task that used a dataset[2] for what they called multi-way classification of semantic relations between pairs of nominals. The relationship types are Member-Collection, Cause-Effect, Component-Whole, Instrument-Agency, Entity-Destination, Product-Producer, Message-Topic, Entity-Origin and Content-Container.

Table I: Statistical characterization of the considered corpora.

Corpora	Sentences	Candidate Pairs of Entities	Entities	Relationships	Relationships Types
AImed	1159	5471	3754	996	1
SemEval	10717	10717	21434	8853	9

2) Linguistic Pre-processing tools:

Before we work on the relationship extraction module, it is important to use the same remaining modules for every solution made part of this Benchmark. Following the modules described before, the tools or techniques applied for each of them are as follows. The *Apache OpenNLP*[3] library, a machine learning based toolkit was used to split the text into sentences, to obtain tokens by processing these sentences and also for part of speech tagging. *Stanford CoreNLP*[4] libary was used for dependency parsing.

Some kernels form the literature require extra linguistic information. This information is used to enrich the token with word classes. An advantage is instead of comparing only tokens, we can compute a similarity function between tokens

[1]ftp://ftp.cs.utexas.edu/pub/mooney/bio-data/interactions.tar.gz
[2]http://semeval2.fbk.eu/semeval2.php?location=data
[3]http://opennlp.apache.org/
[4]http://nlp.stanford.edu/software/corenlp.shtml

by counting the common word classes. When applied we used the same types of word classes into the kernels. The implemented word classes types are:

Part of Speech Tags: computed during the pre-processing, this class represents the grammatical class of the word.

Generic Part of Speech Tags: are generalization over the Part of Speech Tags, it has less types of classes. It is computed by mapping the POS tags to only { Adjective Noun, Verb and Other }.

Stem: computed through the Porter Stemming Algorithm. This algorithm uses rules to reduce the work to his basic form.

Capitalization: reduces the words into two patterns. The first pattern reduces each char to a specific symbol if it is capital letter (A), a minuscule (a) or a number(0). The second pattern is similar to the first one but when there is one or more chars of the same type, the sequence is replaced by a symbol following a $+$.

3) Relationship Extraction Methods: The relationship extraction methods from the current state-of-the-art are based on machine learning approaches. The most indicated machine learning methods are the Support Vector Machines (SVMs). SVMs have the particularity to give good results in classification problems, and can either be used with feature-based methods or kernel-based methods.

Two SVM solvers are contained in our contribution, the simpler online learning from Section III-A which is used with single kernels solutions and the multiple kernel online learner Section III-B used with a set of kernels.

From the kernel-based methods in the literature, we selected and implemented the subsequence kernel described in Section II-A, the bag-of-n-grams kernel described in Section II-C and the shortest path dependency kernel described in Section II-B. These kernels are part of the benchmark and with the online learning solution from Section III-A they form the first three evaluated relationship extraction methods.

The next systems are based on these last three kernels. As it was described in these kernels descriptions, both the subsequence kernel and the bag-of-n-grams kernels are respectively composed of three and five sub-kernels. These kernels can be used separately with our multiple kernel learner described in Section III-B. This way we aim for a better performance of the system. Therefore, the benchmark includes three more systems that use the multiple kernel SVM solver. The first one, is composed by the three sub-kernels that compose the subsequence kernel, the second one similar to the first, used the five sub-kernels found in the bag-of-n-grams kernel and the last one uses all the sub-kernels from the subsequence kernel and the bag-of-n-grams kernel plus the shortest-path dependency kernel.

4) Evaluation Metrics for Relationship Extraction: In a benchmark study, it is very important to provide success measures in a way that supports the comparison of methods. This article is related to the usage of machine learning methods in a classification task. Typical measures to evaluate classification systems are *precision*, *recall* and the *f-measure*. Let us define r as a type of relationship. *Precision* is the fraction of correctly retrieved relationships of type r in the text, over the total number of retrieved relationships of type r. *Recall* is the fraction of relationships of type r that are correctly extracted, over the total number of relationships of type r present in the text. An attempt to combine these measures is the F_1*-Measure* , which corresponds to the harmonic mean of both Precision and Recall.

Being N the number of relationships types r, let us define:

Being N the number of relationships types r, let us define:

True Positives (TP_r) as the number of successfully extracted relationships of type r;

False Positives (FP_r) as the number of extracted relationships that are said to be of type r but are not from type r;

True Negatives (TN_r) as the number of success fully extracted relationships that were not of type r;

False Negatives (FN_r) as the number of extracted relationships that are of type r but are said to be other type;

Precision, Recall and F_1-Measure can accordingly be defined as follows:

$$P_r = \frac{TP_r}{TP_r + FP_r},\ R_r = \frac{TP_r}{TP_r + FN_r},\ F_1 r = \frac{2P_r R_r}{P_r + R_r} \tag{23}$$

These measures are naturally defined per class, but it can be useful to extend them in order to compose a final score in the case of multiclass classification problems. A direct extension is obtained by averaging over Precision and Recall, which is what Macro-average and Micro-average methods try to implement. *Macro-averages* given by Equations (24) and (25) take the average of the precision and recall of the system on different sets. *Micro-averages* given by Equations (26) and (27) sum up the individual true positives, false positives, and false negatives returned by the system for different classes, and the measures are computed by applying the formula of the single class measures (i.e. Precision or Recall) directly using the sums of the individuals.

$$\text{Macro-average Precision} = \frac{\sum_r^N P_r}{N} \tag{24}$$

$$\text{Macro-average Recall} = \frac{\sum_r^N R_r}{N} \tag{25}$$

$$\text{Micro-average Precision} = \frac{\sum_r^N TP_r}{\sum_r^N TP_r + \sum_r^N FP_r} \tag{26}$$

$$\text{Micro-average Recall} = \frac{\sum_r^N TP_r}{\sum_r^N TP_r + \sum_r^N FN_r} \tag{27}$$

IV. Validation

In this section, we present the results obtained when applying the benchmark described in Section III-C to the relationship extraction methods. In Section IV-A, we present the experimental setup, in particular the parameter values used and the tested kernel-based methods. Section IV-B holds the results of the experiments over the datasets.

A. Experimental Setup

This Section describes the experiments performed during this work by following the Benchmark described in Section III-C. First in Section IV-A1, we enumerate the different kernels, or combinations of kernels, and associate them either to the online learner from Section III-A or to the multiple kernel learner from Section III-B. Then, we present the fixed values for the SVM solver's parameters in Section IV-A2. Section IV-A3 describes the hardware and software configurations where we run our experiments.

1) Kernels used in the Experiments: The first three experiments are performed with the online learner described in Section III-A, while the other three are performed with the multiple kernel learner described in Section III-B. The kernels that constitute part of the experiments and that can accept a word class configuration are using {POS tags, GPOS tags, Stem, Capitalization pattern 1 and Capitalization pattern 2}. The kernel methods we experiment are:

Experiment 1: Subsequence Kernel, from Section II-A.

Experiment 2: Bag-Of-N-Grams Kernel, from Section II-C.

Experiment 3: Shortest Path kernel, from Section II-B.

Experiment 4: Multiple kernel learning over the following subsequences sub-kernels, subsequences fore-between kernel; subsequences between-after kernel; and the subsequences between kernel from Section II-A.

Experiment 5: Multiple kernel learning over the bag-of-n-grams sub-kernels described in Section II-C. The bag-of-n-grams is composed of two kernels. the global context Kernel and the local context kernel. Moreover, these two kernels can be decomposed in three and two sub-kernels respectively. Then the kernels contained in this set are the global context fore-between kernel; the global context between-after kernel; the global context between kernel; the local context left entity kernel; and the local context right entity kernel.

Experiment 6: Multiple kernel learning over the Shortest Path Kernel form Section II-B plus the sub-kernels used in the Experiments 4 and 5.

2) Parameters: The experiments were tested using each available dataset individually AImed and SemEval . Moreover, in the first half of the experiments we solve SVMs by using an online approach with a single kernel, while for the remaining experiments, we use an online multiple kernel approach with a set of kernels. For both datasets and both SVM solvers, a maximum number of iterations corresponding to the parameter T described in Section III-A and in Section III-B was fixed to the value of 100.

The parameters λ form the online learner in Section III-A, and the parameters λ_α and λ_β from the multiple kernel learner in Section III-B were fixed by finding the best performance depending on the experiment and on the dataset.

For the AImed dataset, experiments were tested using a 10-folds cross configuration. The parameter λ for Experiments 1, 2 and 3 was fixed to $\lambda = \frac{1}{50}$. As for the remaining experiments: Experiment 4 fixes $\lambda_\alpha = \frac{1}{50}$ and $\lambda_\beta = \frac{1}{50}$; Experiment 5 fixes $\lambda_\alpha = \frac{1}{30}$ and $\lambda_\beta = \frac{1}{30}$; and finally Experiment 6 fixes $\lambda_\alpha = \frac{1}{100}$ and $\lambda_\beta = \frac{1}{100}$.

The SemEval competition provided a train set and test sets. Unfortunately, SemEval is a bigger dataset than AImed and we could not perform as many experiences to fix the parameters as we did with AImed. The used parameters values were $\lambda = \frac{1}{50}$ for Experiment 1, 2 and 3. For Experiments 4, 5 and 6, we used $\lambda_\alpha = \frac{1}{50}$ and $\lambda_\beta = \frac{1}{50}$ in the multiple kernel learner.

3) Hardware and Software Configurations: The experiments were performed in a cluster environment with shared memory and composed by 13 machines, each with 8 Gb of memory RAM and equipped with an Intel Q6600 CPU, 4 cores of 2.4GHz. Our software was adapted in such a way that the training step could be executed by distributing the classifiers on the 13 machines. This configuration reduced the waiting time and we used MPJ Express[5], a Java library, to implement it.

B. Experimental Results

The experiments defined in Section IV-A were conducted over the AImed and SemEval datasets. The results are reported in Tables II and III for the AImed and SemEval datasets, respectively. The discussions about the results are in Sections IV-B1 and IV-B2 respectively for AImed and SemEval datasets.

1) AImed dataset: AImed is a one relationship dataset, containing interactions between proteins, being the content described in a medicine context. Therefore, the most descriptive measure of the performance is the F1-Measure. Through the observation of the results expressed in Table II, we can conclude that the Experiment 2 is the best in the range of experiments that used the online learner from Section III-A. Moreover, we conclude that Experiment 1 was in general the best experiment in terms of Precision. The worst performance experiment is Experiment 3, it uses the shortest path dependency kernel which is based on grammatical information between words. Due to AImed being a dataset with a specific domain of protein interactions, its documents are not composed of common structured English sentences. This fact can be the main cause of Experiment 3 bad performance.

Multiple kernel learning solutions are introduced in Experiments 4, 5 and 6. We expected them to score the best performances, and we observe a tendency for the multiple kernel learner, with the sub-kernels, to perform better than the linear composed solution, which are the cases of Experiment 1 vs Experiment 4 and Experiment 2 vs Experiment 5.

The best performance was obtained by Experiment 5, a multiple kernel learning solution. Note that Experiment 2 and Experiment 5 share the same kernels, the bag-of-n-grams, so, we can conclude it is the best kernel for the AImed dataset.

Experiment 6 which is the experiment with the biggest set of kernels, including the kernels from other experiments, have a lower performance than Experiment 5. It was expected that Experiment 6 would obtain the best performance because Experiment 5 uses a subset of kernels of Experiment 6. This behavior can be explained because Experiment 6 uses the

[5]http://mpj-express.org/

shortest path kernel which is also the kernel used in the worst experiment, Experiment 3. This kernel most probably is adding bad information and making it harder to learn the right SVMs.

Finally, the results of using AImed dataset show us that multiple kernel learning solutions will perform better than online learners using linear combination of the same kernels. The best performance was obtained by Experiment 5, a multiple kernel learning solution.

Table II: AImed results.

Experiment	**Average Precision**	**Average Recall**	**Average F1-Measure**
Experiment 1	**69.35%**	36.52%	42.29%
Experiment 2	60.08%	48.83%	52.21%
Experiment 3	56.96%	21.73%	29.20%
Experiment 4	58.61%	46.87%	50.27%
Experiment 5	59.08%	**50.92%**	**53.86%**
Experiment 6	61.89%	46.73%	52.23%

2) SemEval dataset: SemEval contains more than one relationship type, and so, the measures used are micro and macro averages of the standard measures (Precision, Recall and F1-measure). The SemEval competition evaluated the systems performance using the Macro measures. Therefore, We give more importance to the Macro F1-Measure in order to analyze the results of the experiments as in Table III

When we observe the Experiments 1, 2 and 3 that use single kernel online learning and we compare these three experiments, we conclude that in term of Macro F1-Measure, Experiment 2 obtains the best score. Moreover, Experiment 2 is better in terms of Micro Recall, Micro F1-Measure, Macro Recall and Macro Precision, and that Experiment 1 gets the best performance in Macro Precision.

We observe that Experiment 2 got the best result in the objective of the competition. Experiment 2 also scores the best performance in terms of Micro Recall, Micro F1-Measure and Macro Recall, while the Experiment 4 perform better for Micro Precision and Experiment 6 for Macro Precision.

From the hypothesis of this thesis, we want to prove that multiple kernel learning is a better solution than single kernel learning. As we observe in this dataset, only Experiment 4 fulfills this constrain. Experiment 4 has a better performance than Experiment 1, which is the multiple kernel solution using the same kernel. Experiment 2 scores the best result which is higher than Experiment 5 and 6 which use common sub-kernels.

From the analysis of the SemEval results, we can not claim for sure that a multiple kernel solution will always be better than a single kernel solution. Even tough, we believe this problem can be solved with a better parameter setup, as we explained before, for lack of time we could not perform as many experiences as we wanted to fix the parameters for SemEval.

Table III: SemEval results.

Experiment	**Micro Precision**	**Micro Recall**	**Micro F1-Measure**	**Macro Precision**	**Macro Recall**	**Macro F1-Measure**
Experiment 1	76.95%	55.46%	64.46%	74.73%	54.42%	62.98%
Experiment 2	76.00	**73.04%**	**74.49%**	75.01%	**72.24%**	**74.60%**
Experiment 3	74.51%	63.68%	68.67%	73.39%	63.04%	67.82%
Experiment 4	**77.07%**	67.12%	71.75%	75.16%	66.17%	70.38%
Experiment 5	76.46%	68.45%	72.23%	75.28%	67.58%	71.22%
Experiment 6	76.91%	69.33%	72.93%	**75.58%**	67.97%	71.57%

V. Conclusions and Related Work

This section presents the main conclusions of this dissertation. The purpose of this dissertation was to prove that a new solution for the relationship extraction problem could be developed. This solution was based on using multiple kernel learning classifiers. To prove this theory we elaborated a Benchmark, which included available datasets for relationship extraction and common pre-processing tools. Moreover, different methods from the state-of-the-art were implemented. These methods are the subsequence kernel, the shortest path dependency kernel and the bag-of-n-grams kernel.

We applied the selected kernels to an online SVM learner, and, by maintaining the specifications of the benchmark, we evaluated the kernel-based methods in common conditions. After, knowing that some of these kernels were composed by linear compositions of sub-kernels, we created sets of kernels and/or sub-kernels. These sets were then used with a multiple kernel SVM learner in the same benchmark. Through a performance analysis, we aimed to prove the advantages of using a solution based on multiple kernels.

Finally, the results showed us that for AImed dataset, we can conclude that multiple kernel learning solution proves to obtain better performances than the single kernel solutions. As for the case of the SemEva dataset, the best experiment is based on single kernel learning, and so, we could not assert the benefit of a multiple kernel based solution, over a single kernel one.

The main contributions of this article are:

Implementation of an online SVMs learner for relationship extraction, called Pegasos.

The development of two extensions for the online learner which allow us to use a kernel and a bias term.

Development and implementation of an online learner for relationship extraction which learns weights for a set of Kernels while learning the SVMs.

Development of a benchmark for relationship extraction methods. It include the use of two different context datasets. The experiments use a common set of pre-processing NLP tools. The benchmark also includes the implementation of three state-of-the-art kernel-based methods, the subsequences kernels, the bag-of-n-grams kernel and the shortest path dependency kernel.

Application of the developed benchmark for comparing a set of kernel-based methods for relationship extraction. The results from an extensive set of experiments showed that although the best experiments are based on multiple kernel learning, we could not always assert the benefit of a multiple kernel-based solution over a single kernel one.

It is usually impossible to claim that a work has come to an end. In fact, a set of possible improvements can be explored, in the future. We will consider the following six possible improvements as future work:

Datasets: Although we used two datasets with different domains, a more comprehensive benchmark can be developed by adding more, datasets, specially with more sentences and from distinct domains. We already implemented a loader for reACE[6], a dataset that contains news. Due to the size of this dataset and the lack of physical machines we were not able to execute the set of experiments before the dissertation's deadline.

Kernel Methods: We can incorporate more kernel-based methods from the literature in the benchmark. These implementations can produce different sets of kernels that can be used with the multiple kernel solution and improve the results.

Tokens enrichment: It was detailed in this dissertation that some kernels can use rich information associated to the token information with word classes, and then, when comparing pairs of tokens, a function would return the number of common word classes. We used a fixed set of word classes, so more experiences can be performed using different sets of word classes, for instance word classes obtained from the wordnet [9] are used in the literature.

Parameters Setup: Better performances can be achieved by adjusting the different parameters. Executing more experiences with different values can also lead to a better performance of the multiple kernel learning based solutions over the remaining ones.

Hierarchical Classification: In multiple classification problems using SVMs, we usually attribute a classifier to each class against the remaining classes or assign a classifier to each possible pair of classes. Furthermore, we can create a hierarchy using classifiers, for instance, when predicting a relationship type, we can first classify a pair of entities by containing, or not, a relationship. If it does, we then classify the type of the relationship. We can go further, for example, when an asymmetric type is predicted. After, we can classify the orientation. Actually, a solution based on hierarchical classification was implemented and tested over a dataset but because of time limitations we could not afford to proceed.

Kernel Weighting: In the process of kernel weighting, the fact that some kernels compute faster than others is not taken into account. This may be a problem, as it can be important to some software applications to design a less computational expensive model. A solution for this, can pass by re-designing the optimization function in order for it to attribute penalties to expensive kernels.

References

[1] M. Marrero, S. Sánchez-Cuadrado, J. M. Lara, and G. Andreadakis, "Evaluation of named entity extraction systems," *Advances in Computational Linguistics. Research in Computing Science*, vol. 41, pp. 47–58, 2009. [Online]. Available: http://site.cicling.org/2009/RCS-41/047-058.pdf

[2] R. Bunescu and R. J. Mooney, "Subsequence kernels for relation extraction," in *Proceedings of the 9th Conference on Natural Language Learning (CoNLL-2005)*, Ann Arbor, MI, July 2006, available at urlhttp://www.cs.utexas.edu/users/ml/publication/ie.html. [Online]. Available: http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=51413

[3] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *The Journal of Machine Learning Research*, vol. 2, pp. 419–444, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1162/153244302760200687

[4] R. C. Bunescu and R. J. Mooney, "A shortest path dependency kernel for relation extraction," in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, ser. HLT '05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, pp. 724–731. [Online]. Available: http://dx.doi.org/10.3115/1220575.1220666

[5] C. Giuliano, A. Lavelli, and L. Romano, "Exploiting shallow linguistic information for relation extraction from biomedical literature," in *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, ser. ACL '06, Trento, Italy, April 2006. [Online]. Available: citeseer.ist.psu.edu/giuliano06exploiting.html

[6] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for svm," in *Proceedings of the 24th international conference on Machine learning*, ser. ICML '07. New York, NY, USA: ACM, 2007, pp. 807–814. [Online]. Available: http://doi.acm.org/10.1145/1273496.1273598

[7] Y. Tsuruoka, J. Tsujii, and S. Ananiadou, "Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ser. ACL '09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 477–485. [Online]. Available: http://dl.acm.org/citation.cfm?id=1687878.1687946

[8] I. Hendrickx, S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, L. Romano, and S. Szpakowicz, "Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals," in *Proceedings of the 5th International Workshop on Semantic Evaluation*. Uppsala, Sweden: Association for Computational Linguistics, July 2010, pp. 33–38. [Online]. Available: http://www.aclweb.org/anthology/S10-1006

[9] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995. [Online]. Available: http://doi.acm.org/10.1145/219717.219748

[6]http://catalog.ldc.upenn.edu/LDC2011T08