Machine Learning for Automatic Classification of Web Service Interface Descriptions

Amel Bennaceur[1], Valérie Issarny[1], Richard Johansson[4], Alessandro Moschitti[3], Romina Spalazzese[2], and Daniel Sykes[1]

[1] Inria, Paris-Rocquencourt, France
firstname.lastname@inria.fr
[2] Università degli Studi dell'Aquila, Italy
romina.spalazzese@di.univaq.it
[3] DISI, University of Trento, Italy
moschitti@disi.unitn.it
[4] Centre for Language Technology, University of Gothenburg, Sweden
richard.johansson@gu.se

Abstract. We argue that automatic classification of web service interface description into a predefined set of categories can considerably speed up the task of finding compatible web services. This can greatly improve to establish automatic connection between networked systems. To carry out the classification, we show that techniques derived from automatic document classification can be successfully applied. We describe how the standard document classification techniques need to be adapted for our task, and finally present the results of a number of experiments in classification of web service interface descriptions.

1 Introduction

Systems interoperability is becoming more and more important given the large increase of both types and use of connecting devices, e.g., smart phones, laptop, tablets, and so on. This fact, along with the large variety of services that can be offered to the user, highly increases the communication complexity. In this perspective, automated solutions for establishing interoperability between the networked systems appear to be the only viable approach to achieve the required level of flexibility and scalability. Unfortunately, traditional solutions to determine compatibility between systems are rather expensive in terms of computational cost, especially when, these are applied to systems in unrelated domains. Indeed, a compatibility assessment requires in-depth analyses considering the interface and conversational protocol of the two target connecting systems.

One way to speed up the assessment above is to apply machine learning methods to automatically classify the high-level functionality of a system based on its interface description. This allows for restricting the scope of compatibility checks and consequently providing an overall performance gain when conducting matchmaking between systems.

In this paper, we describe how the interface description classifiers are implemented by applying machine learning: inducing the classification function from a set of examples. We describe how the standard document classification techniques need to be changed in order to be adapted to work with interface descriptions: most importantly, the feature extraction function needs to process the semi-structured data that is available in files written in the Web Services Description Language (WSDL)[5].

[5] `http://www.w3.org/TR/wsdl`

We carried out a number of experiments that evaluate the effect of several design parameters in the implementation of the categorization system, such as the design of the feature extraction function and the choice of machine learning method.

In the reminder of this paper, Section 2 describes the target of the automated classifiers, i.e., the web services description files, Section 3 describes the machine learning methods we apply, i.e., automated text categorization, specialized for the target task, Section 4 presents our experiments on the classification of description files, and, finally, Section 5 derives the conclusions.

2 Web Service Interface Descriptions and the WSDL Format

Web services normally expose a description of their programmatic interface (API) using the standard WSDL (Web Service Description Language[6]) format. This description details all the operations which can be performed by the web service, and data types (in XML Schema[7]) that are inputs or outputs of the operations. WSDL also permits the inclusion of human-readable documentation and has some support for specifying the *semantics* of operations (and data) through the ontological annotations supported in SA-WSDL[8]. There is however no structured support for specifying what the service does at a high level, i.e. the abstract category to which the service belongs.

In the Connect project we define a language for specifying these categories by reference to ontology concepts [2] in order to facilitate the identification of services with similar, compatible functionality. Services in the same category are expected to have similar functionality, which may be provided to the environment, or required from it, and which allows the services to be composed and interact. However, legacy services do not make use of the explicit Connect description language. Without information about the high-level functionality of services, it is necessary to employ time-consuming syntactic and behavioural analyses that determine compatibility as a very fine-grained level of detail. Since Connect aims to overcome interoperability issues at runtime, computationally expensive procedures must be avoided. Service categories provide a cheap means to determine approximate compatibility, before applying more detailed checks where necessary. Consequently we need a means to determine the high-level functionality of a service given only the WSDL description.

Figure 2 shows a partial WSDL description for a weather service, taken from the web. It lists a number of operations with names such as " GetWeatherByZipCode" and " GetWeatherByPlaceName". Each operation refers to messages which in turn determine the input and output data of the operation (defined elsewhere in the file). Each operation also includes a short piece of documentation, although this latter part is not always present. It is however obvious that this service has functionality related to the weather, and when presented with a taxonomy of categories a human would likely be able to assign this service to one of them: this is the process we seek to automate.

3 Applying Document Classification Techniques for Classification of Web Interface Descriptions

In order to build automatic classifiers of web service interfaces, we will build on the considerable amount of research that has been carried out on the topic of automat-

[6] `http://www.w3.org/TR/wsdl`
[7] `http://www.w3.org/XML/Schema`
[8] `http://www.w3.org/2002/ws/sawsdl/`

```
<wsdl:portType name="WeatherForecastSoap">
  <wsdl:operation name="GetWeatherByZipCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a valid Zip Code (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByZipCodeSoapIn" />
    <wsdl:output message="tns:GetWeatherByZipCodeSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetWeatherByPlaceName">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a place name (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameSoapIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="WeatherForecastHttpGet">
  <wsdl:operation name="GetWeatherByZipCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a valid Zip Code (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByZipCodeHttpGetIn" />
    <wsdl:output message="tns:GetWeatherByZipCodeHttpGetOut" />
  </wsdl:operation>
  <wsdl:operation name="GetWeatherByPlaceName">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week  weather forecast for a place name (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameHttpGetIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameHttpGetOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="WeatherForecastHttpPost">
  <wsdl:operation name="GetWeatherByZipCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a valid Zip Code (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByZipCodeHttpPostIn" />
    <wsdl:output message="tns:GetWeatherByZipCodeHttpPostOut" />
  </wsdl:operation>
  <wsdl:operation name="GetWeatherByPlaceName">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a place name (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameHttpPostIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameHttpPostOut" />
  </wsdl:operation>
</wsdl:portType>
```

Fig. 1. WSDL fragment for a weather service.

ically assigning a category tag to a given text document. This is a task with many practical applications in the real world [1].

We give an introduction to the topic of automatic classification of text documents, and describe how machine learning techniques are typically applied to solve this task. We then show how these techniques can be adapted at very little effort for the task of interface description classification. The most significant change from standard document classification methods is that the *feature extractor* – the function that determines the attributes used to distinguish the categories – needs to be tailored for the specifics of web service interface descriptions.

3.1 An Introduction to Automatic Classification of Documents

The complexity of the category systems may vary depending on the application. The simplest would be a binary classification such as spam filtering. Slightly more complex category system can be seen in tasks such as sentiment classification of reviews [13], where the task of the classifier would be to predict the number of stars assigned by the reviewer. The largest category systems are typically hierarchically organized, such as the categories used in well-known Reuters dataset [9], and we may well imagine even more advanced categories such as the structured classifications used in library science [14].

3.2 Machine Learning for Automatic Document Classification

The task of categorizing documents is usually tackled by applying classifiers that have been automatically induced by estimation on a collection of document. We refer to the process of automatically inducing a classification function from data as *machine learning*, and the collection of documents on which the estimation is carried out is called the *training set* [10].

We may divide the set of machine learning methods into two broad categories: *supervised learning*, where each document in the training set is associated with a document category assigned by a human and the task of the machine learner is to induce a function that produces similar labelings, and *unsupervised learning*, where the documents are not labeled a priori and the machine learning method must take responsibility for finding a meaningful division into categories. This paper focuses on the former method, which has generally been much more successful in most studies.

There are a very large variety of methods to carry out supervised machine learning of classifiers. For classification of documents the most popular learning methods are based on the idea of associating classes of documents with regions in a *vector space*. Training a classifier becomes equivalent to describing the *decision surface*, the boundary between the regions in the space. In most cases this is described using a linear function, so the decision surface becomes a hyperplane. Methods for inducing the linear separator include the well-known perceptron algorithm [15]. The most notable recent advance in machine learning by inducing linear separator is the *support vector machine* (SVM) [3], which has been very successfully applied to the task of document classification [8]. The support vector classification approach is based on finding the maximal separation between the classes – the *maximal margin*. In case the classes are not fully separable, *soft margins* are introduced, which permit a small number of violations of the separation constraint. Figure 2 shows an illustration of a soft-margin support vector machine. Note that this is much simpler than in realistic document classification, where the documents may be represented as points in a vector space of millions of dimensions rather than just two.

In addition to the simple task of assigning a category label to a document, similar machine learning techniques can also be applied in very complex categorization

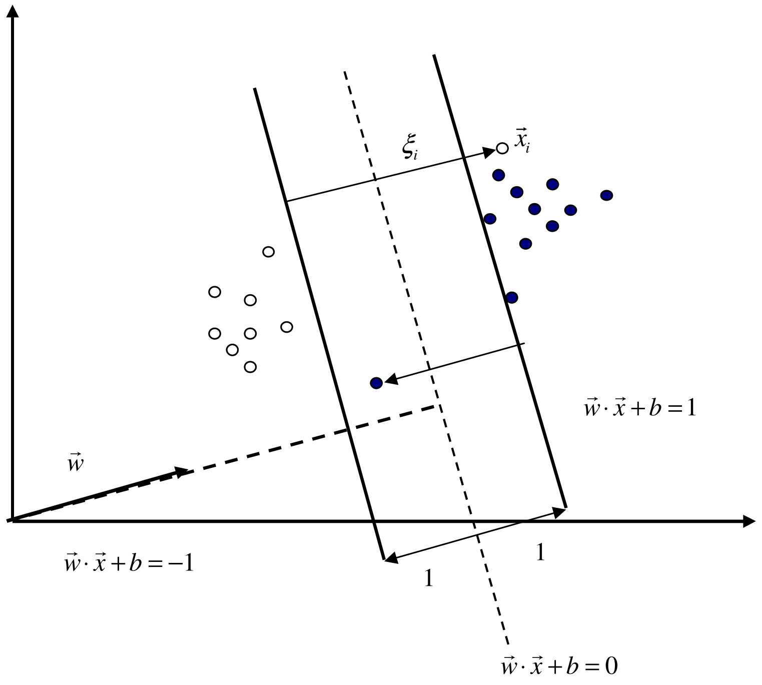
ξ_i
$\vec{x}_i$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}$
$\vec{w}\cdot\vec{x}+b=-1$
1
1
$\vec{w}\cdot\vec{x}+b=0$

Fig. 2. Example of the decision boundary (dashed line) and the margins in a soft-margin support vector machine. There are two violations of the margin constraints.

tasks. An important type of such categorization tasks are those requiring a classification of a *pair* of documents, for instance determining whether a text is a good answer to a given question [11].

3.3 Extraction of Features

In order to be able to assign objects into categories, an automatic classifier needs to determine the salient properties of those objects. This process is called *feature extraction.* A good feature extractor should extract exactly those features that make it easy to distinguish the categories. As it has been noted repeatedly, designing feature extractors is an art more than a science, and requires a good understanding of the classification task.

For the task of document classification, the most common feature extraction method is called the *bag-of-words* representation [16]. In this method, a document is converted into a point in a vector space; every word in the vocabulary is associated with a dimension of the vector space, allowing the document to be mapped into the vector space simply by computing the occurrence frequencies of each word. The bag-of-words representation is considered the standard representation underlying most document classification approaches, and attempts to incorporate more complex structural information have mostly been unsuccessful for the task of categorization of single documents [12].

3.4 Feature Extraction in Web Service Interface Descriptions

WSDL documents generally contain a significant amount of text, and this text can be directly processed using standard a bag-of-words representation method. However, the most informative part of the WSDL interface descriptions normally consists of *structured data*: names of methods, objects, parameters. These elements are represented as a tree-structured XML structure. It should be noted that very little textual documentation may be available if the identifiers are informatively named and their purpose obvious.

This means that the various semi-structured identifiers that are part of the WSDL interface description XML documents should be added to the bag-of-words feature representation. Most importantly, the representation should include the names of the method and input parameters defined by the interface. The inclusion of identifiers will be important since: (1) the textual content of the identifiers is often highly informative of the functionality provided by the respective methods; and (2) the free text documentation is not mandatory and may not always be present.

To extract useful word tokens from the identifiers, we split them into pieces based on the presence of underscores or CamelCase. All tokens were then normalized to lowercase. For instance, consider the following piece of WSDL code.

```
<wsdl:message name="GetWeatherByZipCodeSoapIn">
    <wsdl:part name="parameters"
          element="tns:GetWeatherByZipCode" />
</wsdl:message>
 <wsdl:message name="GetWeatherByZipCodeSoapOut">
   <wsdl:part name="parameters"
         element="tns:GetWeatherByZipCodeResponse" />
</wsdl:message>
```

In this example, we split the CamelCased identifier `GetWeatherByZipCode` into the tokens `get`, `weather`, `by`, `zip`, and `code`, so the complete bag-of-words vector for the example will be `[get:4, weather:4, by:4, zip:4, code:4, soap:2, in:1, out:1, response:1]`.

4 Experiments

We carried out a large number of experiments to find the best way to implement the categorizer of web service interface descriptions.

As described in Sec. 3.2, we implemented the classifiers by automatically inducing them from labeled data. For this purpose, we used a collection of WSDL documents[9] [7]. We selected the 10 most frequent categories, in total 397 documents.

To train the classifiers, we used support vector machines that we trained using the LibLinear machine learning software [6].

The following subsections describe the experiments. All results have been obtained using a 10-fold cross-validation procedure: split the data into 10 pieces; form 10 different training sets by excluding each piece; train 10 classifiers; evaluate on each piece and combine the result.

4.1 Classification Results

To evaluate the performance of the classifiers, we computed a number of different evaluation meaures. The first one is the overall classification *accuracy*, which is

[9] `http://www.andreas-hess.info/projects/annotator/ws2003.html`

Category	Number of instances
CountryInfo	64
Money	54
Converter	49
Finder	46
Communication	45
Web	39
Developers	37
News	30
Business	23
Mathematics	10
Total	397

Table 1. Statistics for the data collection.

defined as the proportion of correctly classified interface descriptions. Our best implementation correctly classified 236 out of 397 descriptions, giving us an accuracy of 59.4%.

In addition to the overall accuracy, we evaluated the classification performance on individual categories. Here, we used the *precision* and *recall* measures. For a category C, we define the precision is defined as the number the classifier correctly predicted C divided by the number of times it predicted C at all. Conversely, we define the recall as the number of correctly predicted C divided by the number of C in the dataset. Finally, it is very common to present the harmonic mean of precision and recall, which is referred to as the F-measure.

In most situations, there is a tradeoff relationship between the precision and recall measures: if we often predict C, then we would also find many C (higher recall) but also overgenerate (lower precision). By varying a class sensitivity parameter when training, we can tune the precision/recall tradeoff, and plot the relationship in a graph. Figure 3 shows such plots for the four largest classes of interfaces: CountryInfo, Money, Converter and Finder. In this kind of plot, overall prediction quality for a class is determined by how close the plot is to the upper right corner. In our case, we see that the overall prediction quality for the four classes tends to be correlated with the size of the class: CountryInfo class is the class for which the plot is closest to the upper right corner.

In addition to the classwise precision and recall evaluations, we computed the *macro precision and recall*, which are precision and recall values averaged over all classes. In this macro evaluation, our classifier achieved a precision of 58.0, a recall of 52.8, and an F-measure of 55.3.

4.2 The Effect of the Design of the Feature Extractor

The challenge in interface description classification compared to traditional document classification is the feature design problem, and in particular the problem of making use of the WSDL structure.

As a baseline, we used traditional bag-of-word feature extractors that are normally used in document classification. We applied the baseline feature extractors to the text available in the documents: the code documentation and the comments.

As an alternative to the raw text, we extracted features from the structured text, i.e. the identifiers used in the WSDL code. As discussed in Sec. 3.4, we used an identifier splitting heuristic based on the presence of CamelCase. However, we also tried out an identifier-based feature representation that did not split the identifiers.

Finally, we used a feature representation that combined the BOW and WSDL identifier features. Here, we evaluated two different representations. In the first

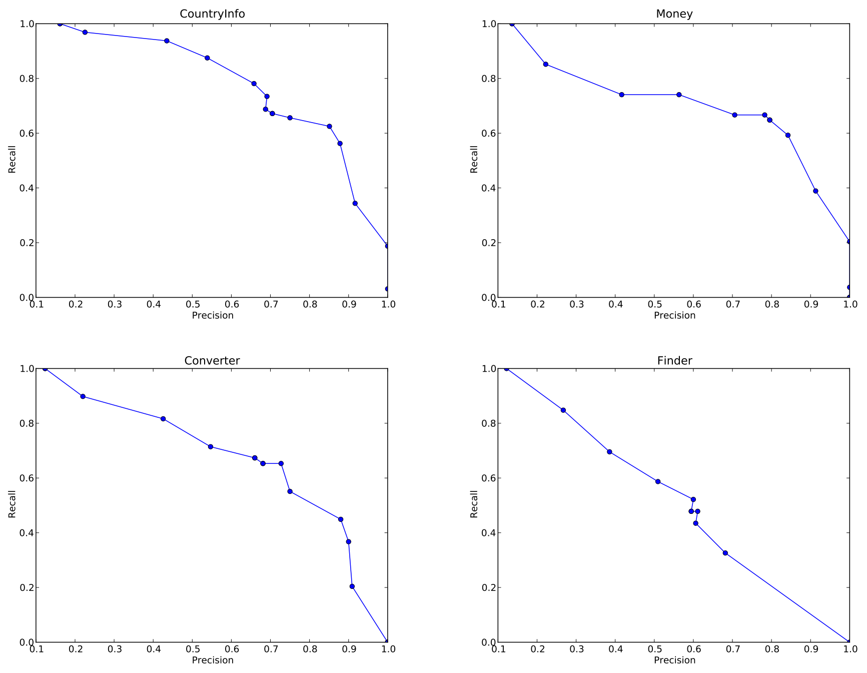

Fig. 3. Precision / recall plots for the four most frequent classes: CountryInfo, Money, Converter and Finder.

Representation	Accuracy	Macro precision	Macro recall	Macro F-measure
BOW (doc. only)	24.9	39.5	28.4	33.0
BOW (doc. and comments)	25.2	36.1	27.8	31.5
WSDL Identifiers	41.6	33.7	32.8	33.2
WSDL Identifiers, CC splitting	58.2	54.8	51.6	53.2
BOW + Indentifiers (separate)	57.9	55.1	51.1	53.0
BOW + Indentifiers (mixed)	59.4	58.0	52.8	55.3

Table 2. Evaluation of feature extraction methods.

approach we used two separate vector spaces for the two types of features. In the second one, there was only one vector space, so for instance if the word *withdraw* would appear in the documentation or in an identifier, this would result in the same feature being enabled.

To see the effect of the feature extraction design choices, we carried out an evaluation of several different feature extractors. The result of this experiment is shown in Table 2. We show the overall classification performance using accuracy and macro precision/recall/F-measure.

The experiment shows very clearly that purely text-based feature representations are not sufficient for achieving a good classification performance. It is crucial to extract features from the WSDL identifiers, and they need to be split in order to be useful. The best representation was the mixed combination of BOW and identifier features; the combination by separate vector spaces seems to lead to feature sparsity, which is very problematic when using small training sets.

Learning method	Accuracy	Macro precision	Macro recall	Macro F-measure
Binarized L2-SVM	59.4	58.0	52.8	55.3
Binarized L1-SVM	58.9	55.3	49.5	52.3
Multiclass SVM	59.4	55.0	52.9	53.9
Logistic regression	49.9	47.1	39.3	42.9
Perceptron	51.1	51.0	45.6	48.1
Passive–aggressive	58.4	53.3	51.4	52.4

Table 3. Evaluation of machine learning methods.

4.3 The Choice of Machine Learning Algorithm

In addition to the feature extraction mechanism, another crucial parameter when building a classifier is the selection of a machine learning algorithm for training. We evaluated a wide range of learning algorithms. Our primary approach was support vector machines (SVMs) since this algorithm has been very successful in text categorization [8].

We used several variations of the SVM learning algorithm. The original SVM formulation [3] was restricted to two-class classification. This means that if we want to apply SVMs to classification problems with more than two classes, we must apply a tranformation of the problem – a *binarization*. The most common binarization method is called *one-versus-all*: create one SVM for each class, and select the highest-scoring class at test time. We tried two binarized SVM variants: L_2-loss and L_1-loss SVMs; L_2-loss is the standard formulation and L_1-loss is a newer variant that typically uses much smaller feature sets [17]. In addition to the binarized SVM variants, we used a recent SVM formulation that can solve multiclass problems directly [5].

Apart from the SVMs, we evaluated classifiers trained using the logistic regression, perceptron [15], and passive–aggressive [4] learning methods. All SVMs, as well as the logistic regression classifiers, were implemented using LibLinear. The perceptron and passive–aggressive classifiers were based on our own implementations.

Table 3 shows the result of the machine learning method comparison. We notice that all SVM variants outperform the other learning algorithms. The best-performing SVM is the most commonly used variant: the L_2-loss SVM with standard binarization.

4.4 Generating Multiple Hypotheses

The purpose of a categorizer of web services is to reduce the time it takes to decide whether we can automatically connect two different web services. While we would certainly like to have a classifier that perfecly predicts the correct class, this is of course not realistic; however, if a classifier that is allowed to make multiple predictions, the probability of the correct class being found is much higher. Such a classifier would also save considerable connection time.

We measured the performance of a classifier that is allowed to output k different category labels. In this evaluation, a prediction is counted as correct if the set of guesses contains the correct answer. The relationship between the k value and the performance is shown in Table 4. We see that even with a small k, we can get very high classification performance.

k	Accuracy	Macro precision	Macro recall	Macro F-measure
1	59.4	58.0	52.8	55.3
2	71.5	71.0	65.6	68.2
3	79.3	79.1	75.0	77.0
4	85.1	85.1	81.3	83.2
5	87.7	88.0	83.9	85.9

Table 4. Classification performance when predicting k possible hypotheses.

5 Conclusion

There is a considerable need for automatic composition of web services. The CONNECT project addresses, in particular, interoperability issues arising from the need to compose heterogeneous systems at runtime. The first step in composing such systems is determining whether, and to what degree, the systems are compatible. At the highest level of abstraction, systems in the same domain category, e.g. weather, have the potential to interact.

Automatic classification of web service interface descriptions is a technique that can speed up the service matching procedure considerably by allowing us to avoid expensive behavioral analyses that encumber the runtime composition of services.

We described how methods derived from automatic document classification based on machine learning can be used to build categorizers of web service interface descriptions.

We carried out a number of experiments in automatic categorization of interface descriptions, to determine the best way to implement an automatic system to carry out this kind of categorization. We evaluated the effect of the choice of machine learning method, and we saw that support vector machines gave the best performance.

Most importantly, we evaluated how the performance is influenced by the design of the feature extraction component of the classifier. We saw very clearly that standard document classification methods are not directly applicable: such an approach leads to very low performance. Instead, we need to use a feature representation that is tailored to the task of interface description classification by using the specific structure of the WSDL code, in particular its identifiers.

We saw that a classifier that predicts a number of possible alternatives (not just one) achieves very high performance levels. We believe that an approach using multiple hypothesis can also be useful for service matching.

Acknowledgements

This research has been supported by the EU FP7 projects: CONNECT – Emergent Connectors for Eternal Software Intensive Networking Systems (project number FP7 231167), EternalS – "Trustworthy Eternal Systems via Evolving Software, Data and Knowledge" (project number FP7 247758) and by the EC Project, LIMOSINE – Linguistically Motivated Semantic aggregation engiNes (project number FP7 288024).

References

1. R. Basili and A. Moschitti. *Automatic Text Categorization: from Information Retrieval to Support Vector Learning.* Aracne editrice, Rome, Italy, 2005.

2. A. Bennaceur, G. S. Blair, F. Chauvel, N. Georgantas, P. Grace, V. Nundloll, M. Paolucci, R. Saadi, and D. Sykes. Intermediate connect architecture. Technical Report D1.2, Connect ICT FET IP Project, February 2011.
3. B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Computational Learning Theory*, pages 144–152, Pittsburgh, United States, 1992.
4. K. Crammer, O. Dekel, J. Keshet, S. Shalev-Schwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 2006(7):551–585, 2006.
5. K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2001(2):265–585, 2001.
6. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.
7. A. Heß and N. Kushmerick. Learning to attach semantic metadata to web services. In *International Semantic Web Conference*, pages 258–273, 2003.
8. T. Joachims. *Learning to Classify Text using Support Vector Machines.* Kluwer/Springer, Boston, 2002.
9. D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.
10. T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.
11. A. Moschitti. Kernel methods, syntax and semantics for relational text categorization. In *Proceedings of ACM 17th Conference on Information and Knowledge Management (CIKM)*, Napa Valley, United States, 2008.
12. A. Moschitti and R. Basili. Complex linguistic features for text classification: A comprehensive study. In *Proceedings of the 26th European Conference on Information Retrieval Research (ECIR 2004)*, pages 181–196, Sunderland, United Kingdom, 2004.
13. B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? Sentiment classification using machine learning techniques. In *Proceedings of EMNLP*, 2002.
14. S. R. Ranganathan. *Colon Classification.* Ess Ess Publications, Delhi, 2006.
15. F. Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(5):386–408, 1958.
16. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. Technical Report TR74-218, Department of Computer Science, Cornell University, Ithaca, New York, 1974.
17. B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2002.