Reverse Engineering of Tree Kernel Feature Spaces

Daniele Pighin
FBK-Irst, HLT
Via di Sommarive, 18 I-38100 Povo (TN) Italy
pighin@fbk.eu

Alessandro Moschitti
University of Trento, DISI
Via di Sommarive, 14 I-38100 Povo (TN) Italy
moschitti@disi.unitn.it

Abstract

We present a framework to extract the most important features (tree fragments) from a Tree Kernel (TK) space according to their importance in the target kernel-based machine, e.g. Support Vector Machines (SVMs). In particular, our mining algorithm selects the most relevant features based on SVM estimated weights and uses this information to automatically infer an explicit representation of the input data. The explicit features (a) improve our knowledge on the target problem domain and (b) make large-scale learning practical, improving training and test time, while yielding accuracy in line with traditional TK classifiers. Experiments on semantic role labeling and question classification illustrate the above claims.

1 Introduction

The last decade has seen a massive use of Support Vector Machines (SVMs) for carrying out NLP tasks. Indeed, their appealing properties such as 1) solid theoretical foundations, 2) robustness to irrelevant features and 3) outperforming accuracy have been exploited to design state-of-the-art language applications.

More recently, kernel functions, which implicitly represent data in some high dimensional space, have been employed to study and further improve many natural language systems, e.g. (Collins and Duffy, 2002), (Kudo and Matsumoto, 2003), (Cumby and Roth, 2003), (Cancedda et al., 2003), (Culotta and Sorensen, 2004), (Toutanova et al., 2004), (Kazama and Torisawa, 2005), (Shen et al., 2003), (Gliozzo et al., 2005), (Kudo et al., 2005), (Moschitti et al., 2008), (Diab et al., 2008). Unfortunately, the benefit to easily and effectively model the target linguistic phenomena is reduced by the the implicit nature of the kernel space, which prevents to directly observe the most relevant features. As a consequence, even very accurate models generally fail in providing useful feedback for improving our understanding of the problems at study. Moreover, the computational burden induced by high dimensional kernels makes the application of SVMs to large corpora still more problematic.

In (Pighin and Moschitti, 2009), we proposed a feature extraction algorithm for Tree Kernel (TK) spaces, which selects the most relevant features (tree fragments) according to the gradient components (weight vector) of the hyperplane learnt by an SVM, in line with current research, e.g. (Rakotomamonjy, 2003; Weston et al., 2003; Kudo and Matsumoto, 2003). In particular, we provided algorithmic solutions to deal with the huge dimensionality and, consequently, high computational complexity of the fragment space. Our experimental results showed that our approach reduces learning and classification processing time leaving the accuracy unchanged.

In this paper, we present a new version of such algorithm which, under the same parameterization, is almost three times as fast while producing the same results. Most importantly, we explored tree fragment spaces for two interesting natural language tasks: Semantic Role Labeling (SRL) and Question Classification (QC). The results show that: (a) on large data sets, our approach can improve training and test time while yielding almost unaffected classification accuracy, and (b) our framework can effectively exploit the ability of TKs and SVMs to, respectively, generate and recognize relevant structured features. In particular, we (i) study in more detail the relevant fragments identfied for the boundary classification task of SRL, (ii) closely observe the most relevant fragments for each QC class and (iii) look at the diverse syntactic patterns characterizing each ques-

tion category.

The rest of the paper is structured as follows: Section 2 will briefly review SVMs and TK functions; Section 3 will detail our proposal for the linearization of a TK feature space; Section 4 will review previous work on related subjects; Section 5 will detail the outcome of our experiments, and Section 6 will discuss some relevant aspects of the evaluation; finally, in Section 7 we will draw our conclusions.

2 Tree Kernel Functions

The decision function of an SVM is:

$$f(\vec{x}) = \vec{w} \cdot \vec{x} + b = \sum_{i=1}^{n} \alpha_i y_i \vec{x_i} \cdot \vec{x} + b \qquad (1)$$

where $\vec{x}$ is a classifying example and $\vec{w}$ and b are the separating hyperplane's *gradient* and its *bias*, respectively. The gradient is a linear combination of the training points $\vec{x_i}$, their labels y_i and their weights α_i. Applying the so-called *kernel trick* it is possible to replace the scalar product with a *kernel function* defined over pairs of *objects*:

$$f(o) = \sum_{i=1}^{n} \alpha_i y_i \mathrm{k}(o_i, o) + b$$

with the advantage that we do not need to provide an explicit mapping $\phi(\cdot)$ of our examples in a vector space.

A Tree Kernel function is a convolution kernel (Haussler, 1999) defined over pairs of trees. Practically speaking, the kernel between two trees evaluates the number of substructures (or *fragments*) they have in common, i.e. it is a measure of their overlap. The function can be computed recursively in closed form, and quite efficient implementations are available (Moschitti, 2006). Different TK functions are characterized by alternative fragment definitions, e.g. (Collins and Duffy, 2002) and (Kashima and Koyanagi, 2002). In the context of this paper we will be focusing on the SubSet Tree (SST) kernel described in (Collins and Duffy, 2002), which relies on a fragment definition that does not allow to break production rules (i.e. if any child of a node is included in a fragment, then also all the other children have to). As such, it is especially indicated for tasks involving constituency parsed texts.

Implicitly, a TK function establishes a correspondence between distinct fragments and dimensions in some *fragment space*, i.e. the space of all the possible fragments. To simplify, a tree t can be represented as a vector whose attributes count the occurrences of each fragment within the tree. The kernel between two trees is then equivalent to the scalar product between pairs of such vectors, as exemplified in Figure 1.

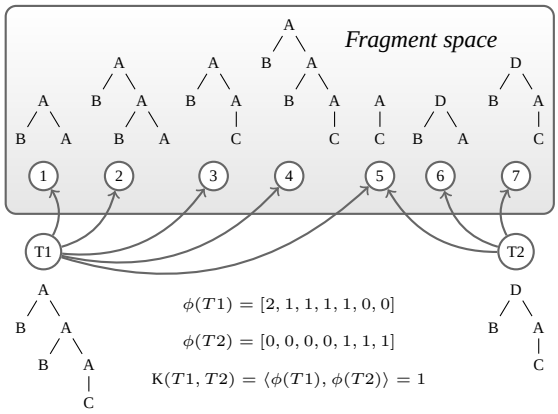

Figure 1: Esemplification of a fragment space and the kernel product between two trees.

3 Linearization of a TK function

Our objective is to efficiently mine the most relevant fragments from the huge fragment space, so that we can explicitly represent our input trees in terms of these fragments and learn fast and accurate linear classifiers.

The framework defines five distinct activities, detailed in the following paragraphs.

3.1 Kernel Space Learning (*KSL*)

The first step involves the generation of an approximation of the whole fragment space, i.e. we can consider only the trees that encode the most relevant fragments. To this end, we can partition our training data into S smaller sets, and use the SVM and the SST kernel to learn S models. We will only consider the fragments encoded by the support vectors of the S models. In the next stage, we will use the SVM estimated weights to drive our feature selection process.

Since time complexity of SVM training is approximately quadratic in the number of examples, by breaking training data into smaller sets we can considerably accelerate the process of filtering trees and estimating support vector weights. According to statistical learning theory, being trained on smaller subsets of the available data these models will be less robust with respect to the minimization of the empirical risk (Vapnik, 1998).

Algorithm 3.1: MINE_MODEL(M, L, λ)

```
global maxexp
prev ← ∅ ; CLEAR_INDEX()
for each ⟨αy, t⟩ ∈ M
do { T_i ← α · y/‖t‖
     for each n ∈ N_t
     do { f ← FRAG(n) ; rel = λ · T_i
          prev ← prev ∪ {f, rel}
          PUT(f, rel) }
best_pr ← BEST(L) ;
while true
do { next ← ∅
     for each ⟨f, rel⟩ ∈ prev if f ∈ best_pr
     do { X = EXPAND(f, maxexp)
          rel_exp ← λ · rel
          for each frag ∈ X
          do { temp = {frag, rel_exp}
               next ← next ∪ temp
               PUT(frag, rel_exp) }
     best ← BEST(L)
     if not CHANGED()
       then break
     best_pr ← best
     prev ← next }
F_L ← best_pr
return (F_L)
```

Nonetheless, since we do not need to employ them for classification (but just to direct our feature selection process, as we will describe shortly), we can accept to rely on sub-optimal weights. Furthermore, research results in the field of SVM parallelization using cascades of SVMs (Graf et al., 2004) suggest that support vectors collected from locally learnt models can encode many of the relevant features retained by models learnt globally. Henceforth, let M_s be the model associated with the s-th split, and $\mathcal{F}_s$ the fragment space that can describe all the trees in M_s.

3.2 Fragment Mining and Indexing (*FMI*)

In Equation 1 it is possible to isolate the gradient $\vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x_i}$, with $\vec{x_i} = [x_i^{(1)}, \ldots, x_i^{(N)}]$, N being the dimensionality of the feature space. For a tree kernel function, we can rewrite $x_i^{(j)}$ as:

$$x_i^{(j)} = \frac{t_{i,j}\lambda^{\ell(f_j)}}{\|t_i\|} = \frac{t_{i,j}\lambda^{\ell(f_j)}}{\sqrt{\sum_{k=1}^{N}(t_{i,k}\lambda^{\ell(f_k)})^2}} \quad (2)$$

where: $t_{i,j}$ is the number of occurrences of the fragment f_j, associated with the j-th dimension of the feature space, in the tree t_i; λ is the kernel decay factor; and $\ell(f_j)$ is the depth of the fragment.

The relevance $|w^{(j)}|$ of the fragment f_j can be measured as:

$$|w^{(j)}| = \left|\sum_{i=1}^{n} \alpha_i y_i x_i^{(j)}\right| = \frac{\left|\sum_{i=1}^{n} \alpha_i y_i t_{i,j}\lambda^{\ell(f_j)}\right|}{\|t_i\|} . \quad (3)$$

We fix a threshold L and from each model M_s (learnt during KSL) we select the L most relevant fragments, i.e. we build the set $\mathcal{F}_{s,L} = \cup_k\{f_k\}$ so that:

$$|\mathcal{F}_{s,L}| = L \text{ and } |w^{(k)}| \geq |w^{(i)}| \forall f_i \in \mathcal{F} \setminus \mathcal{F}_{s,L} .$$

To generate all the fragments encoded in a model, we adopt the greedy strategy described in Algorithm 3.1. Its arguments are: an SVM model M represented as $\langle \alpha y, t\rangle$ pairs, where t is a tree structure; the threshold value L; and the kernel decay factor λ.

The function FRAG(n) generates the smallest fragment rooted in node n (i.e. for an SST kernel, the fragment consisting of n and its direct children). We call such fragment a *base* fragment. The function EXPAND($f, maxexp$) generates all the fragments that can be derived from the fragment f by expanding, i.e. including in the fragment the direct children of some of its nodes. These fragments are *derived* from f. The parameter $maxexp$ limits fragment proliferation by setting the maximum number of nodes which can be expanded in a fragment expansion operation. For example, if there are 10 nodes which can be expanded in fragment f, then only the fragments where at most 3 of the 10 nodes are expanded will be generated by a call to EXPAND($f, 3$).

Every time we generate a fragment f, the function PUT(f, rel) saves the fragment along with its relevance rel in an *index*. The index keeps track of the cumulative relevance of a fragment, and its implementation has been optimized for fast insertions and spatial compactness.

A whole cycle of expansions is considered as an iteration of the mining process: we take into account all the fragments that have undergone k expansions and produce all the fragments that result from a further expansion, i.e. all the fragments expanded $k+1$ times.

We keep iterating until we reach a stop criterion, which we base on the threshold value L, i.e. the limit on the number of fragments that we are interested in mining from a model. During each iteration $k+1$, we only expand the best L fragments identified during the previous iteration k. When

the iteration is complete we re-evaluate the set of L best fragments in the index, and we stop only if the worst of them, i.e. the L-th ranked fragment at the step $k+1$, and its score are the same as at the end of the previous iteration. That is, we assume that if none of the fragments mined during the $(k+1)$-th iteration managed to affect the bottom of the pool of the L most relevant fragments, then none of their expansions is likely to succeed. In the algorithm, $\mathcal{N}_t$ is the set of nodes of the tree t; BEST(L) returns the L highest ranked fragments in the index; CHANGED$()$ verifies whether the bottom of the L-best set has been affected by the last iteration or not.

We call MINE_MODEL$(\cdot)$ on each of the models M_s that we learnt from the S initial splits. For each model, the function returns the set of L-best fragments in the model. The union of all the fragments harvested from each model is then saved into a dictionary $\mathcal{D}_L$ which will be used by the next stage.

3.2.1 Discussion on FMI algorithm

With respect to the algorithm presented in (Pighin and Moschitti, 2009), the one presented here has the following advantages:

- the process of building fragments is strictly small-to-large: fragments that span $n+1$ levels of the tree may be generated only after all those spanning n levels;
- the threshold value L is a parameter of the mining process, and it is used to prevent the algorithm from generating more fragments than necessary, thus making it more efficient;
- it has one less parameter ($maxdepth$) which was used to force fragments to span at-most a given number of levels. The new algorithm does not need it since the maximum number of iterations is implicitly set via L.

These differences result in improved efficiency for the FMI stage. For example, on the data for the boundary classification task (see Section 5), using comparable parameters the old algorithm required 85 minutes to mine the most relevant fragments, whereas the new one only takes 31, i.e. it is 2.74 times as fast.

3.3 Tree Fragment Extraction (*TFX*)

During this phase we actually linearize our data: a file encoding label-tree pairs $\langle y_i, t_i \rangle$ is transformed to encode label-vector pairs $\langle y_i, \vec{v_i} \rangle$. To do so, we generate the fragment space of t_i, using a variant of the mining algorithm described in Algorithm 3.1, and encode in $\vec{v_i}$ all and only the fragments $t_{i,j}$ so that $t_{i,j} \in \mathcal{D}_L$. The algorithm exploits labels and production rules found in the fragments listed in the dictionary to generate only the fragments that *may be* in the dictionary. For example, if the dictionary does not contain a fragment whose root is labeled N, then if a node N is encountered during TFX neither its base fragment nor its expansions are generated. The process is applied to the whole training *(TFX-train*) and test (*TFX-test*) sets. The fragment space is now *explicit*, as there is a mapping between the input vectors and the fragments they encode.

3.4 Explicit Space Learning (*ESL*)

Linearized training data is used to learn a very fast model by using all the available data and a linear kernel.

3.5 Explicit Space Classification (*ESC*)

The linear model is used to classify linearized test data and evaluate the accuracy of the resulting classifier.

4 Previous work

A rather comprehensive overview of feature selection techniques is carried out in (Guyon and Elisseeff, 2003). Non-filter approaches for SVMs and kernel machines are often concerned with polynomial and Gaussian kernels, e.g. (Weston et al., 2001) and (Neumann et al., 2005). Weston et al. (2003) use the ℓ_0 norm in the SVM optimizer to stress the feature selection capabilities of the learning algorithm. In (Kudo and Matsumoto, 2003), an extension of the PrefixSpan algorithm (Pei et al., 2001) is used to efficiently mine the features in a low degree polynomial kernel space. The authors discuss an approximation of their method that allows them to handle high degree polynomial kernels.

Suzuki and Isozaki (2005) present an embedded approach to feature selection for convolution kernels based on χ^2-driven relevance assessment. To our knowledge, this is the only published work clearly focusing on feature selection for tree kernel functions, and indeed has been one of the major sources of inspiration for our methodology. With respect to their work, the difference

in our approach is that we want to exploit the SVM optimizer to select the most relevant features instead of a relevance assessment measure that moves from different statistical assumptions than the learning algorithm.

In (Graf et al., 2004), an approach to SVM parallelization is presented which is based on a divide-et-impera strategy to reduce optimization time. The idea of using a compact graph representation to represent the support vectors of a TK function is explored in (Aiolli et al., 2006), where a Direct Acyclic Graph (DAG) is employed. In (Moschitti, 2006; Bloehdorn and Moschitti, 2007a; Bloehdorn and Moschitti, 2007b; Moschitti et al., 2007), the SST kernel along with other tree and combined kernels are employed for question classification and semantic role labeling with interesting results.

5 Experiments

We evaluated the capability of our model to extract relevant features on two data sets: the CoNLL 2005 shared task on Semantic Role Labeling (SRL) (Carreras and Màrquez, 2005), and the Question Classification (QC) task based on data from the TREC 10 QA competition (Voorhees, 2001). The next sections will detail the setup and outcome of the two sets of experiments.

All the experiments were run on a machine equipped with 4 Intel® Xeon® CPUs clocked at 1.6 GHz and 4 GB of RAM. As a supervised learning framework we used SVM-Light-TK[1], which extends the SVM-Light optimizer (Joachims, 2000) with tree kernel support. For each classification task, we compare the accuracy of a vanilla SST classifier against the corresponding linearized SST classifier (SST$_\ell$). For KSL and SST training we used the default decay factor $\lambda = 0.4$. For ESL, we use a non-normalized, linear kernel. No further parametrization of the learning algorithms is carried out. Indeed, our focus is on showing that, under the same conditions, our linearized tree kernel can be as accurate as the original kernel, and choosing of parameters may just bias such test.

5.1 Semantic Role Labeling

For our experiments on semantic role labeling we used PropBank annotations (Palmer et al., 2005) and automatic Charniak parse trees (Charniak, 2000) as provided for the CoNLL 2005 evaluation campaign (Carreras and Màrquez, 2005). SRL can be decomposed into two tasks: *boundary detection*, where the word sequences that are arguments of a predicate word w are identified, and *role classification*, where each argument is assigned the proper role. The former task requires a binary *Boundary Classifier* (BC), whereas the second involves a *Role Multi-class Classifier* (RM).

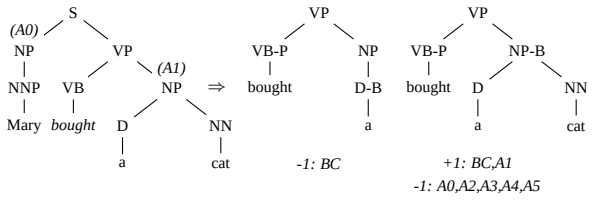

Figure 2: Examples of AST$_m$ structured features.

5.1.1 Setup

If the constituency parse tree t of a sentence s is available, we can look at all the pairs $\langle p, n_i \rangle$, where n_i is any node in the tree and p is the node dominating w, and decide whether n_i is an *argument node* or not, i.e. whether it exactly dominates all and only the words encoding any of w's arguments. The objects that we classify are subsets of the input parse tree that encompass both p and n_i. Namely, we use the AST$_m$ structure defined in (Moschitti et al., 2008), which is the minimal tree that covers all and only the words of p and n_i. In the AST$_m$, p and n_i are marked so that they can be distinguished from the other nodes. An AST$_m$ is regarded as a positive example for BC if n_i is an argument node, otherwise it is considered a negative example. Positive BC examples can be used to train an efficient RM: for each role r we can train a classifier whose positive examples are argument nodes whose label is exactly r, whereas negative examples are argument nodes labeled $r' \neq r$. Two AST$_m$s extracted from an example parse tree are shown in Figure 2: the first structure is a negative example for BC and is not part of the data set of RM, whereas the second is a positive instance for BC and A1.

To train BC we used PropBank sections 1 through 6, extracting AST$_m$ structures out of the first 1 million $\langle p, n_i \rangle$ pairs from the corresponding parse trees. As a test set we used the 149,140 instance collected from the annotations in Section 24. There are 61,062 positive examples in the training set (i.e. 6.1%) and 8,515 in the test set

[1] `http://disi.unitn.it/~moschitt/Tree-Kernel.htm`

(i.e. 5.7%).

For RM we considered all the argument nodes of any of the six PropBank core roles (i.e. A0, ..., A5) from all the available training sections, i.e. 2 through 21, for a total of 179,091 training instances. Similarly, we collected 5,928 test instances from the annotations of Section 24. Columns *Tr*$^+$ and *Te*$^+$ of Table 1 show the number of positive training and test examples, respectively, for BC and the role classifiers.

For all the linearized classifiers, we used 50 splits for the FMI stage and we set the threshold value $L = 50k$ and $maxexp = 1$ during FMI and TFX. We did not validate these parameters, which we know to be sub-optimal. These values were selected during the development of the software because, on a very small test bed, they resulted in a responsive and accurate system.

We should point out that other experiments have shown that linearization is very robust with respect to parametrization: due to the huge number and variety of fragments in the TK space, different choices of the parameters result in different explicit spaces and more or less efficient solutions, but in most cases the final accuracy of the linearized classifiers is affected only marginally. For example, it could be expected that reducing the number of splits during KSL would improve the final accuracy of a linearized classifier, as the weights used for FMI would then converge to the global optimum. Instead, we have observed that increasing the number of splits does not necessarily decrease the accuracy of the linearized classifier.

The evaluation on the whole SRL task using the official CoNLL'05 evaluator was not carried out because producing complete annotations requires several steps (e.g. overlap resolution, OvA or Pairwise combination of individual role classifiers) that would shade off the actual impact of the methodology on classification.

5.1.2 Results

The left side of Table 1 shows the distribution of positive data points in the training and test sets of each classifier. Columns *SST* and *SST*$_\ell$ compare side by side the F_1 measure of the non-linearized and linearized classifier for each class. The accuracy of the RM classifier is the percentage of correct class assignments.

We can see that the accuracy of linearized classifiers is always in line with vanilla SST, even if the selected linearization parameters generate a very rough approximation of the original fragment space, generally consisting of billions of fragments. BC$_\ell$ (i.e. the linearized BC) has an F_1 of 81.3, just 0.5% less than BC, i.e. 81.8. Concerning RM$_\ell$, its accuracy is the same as the non linearized classifier, i.e. 87.8.

	Data set		Accuracy	
Class	Tr$^+$	Te$^+$	SST	SST$_\ell$
BC	61,062	8,515	81.8	81.3
A0	60,900	2,014	91.6	91.1
A1	90,636	3,041	89.0	89.4
A2	21,291	697	73.1	73.0
A3	3,481	105	56.8	53.0
A4	2,713	69	69.1	67.9
A5	69	2	66.7	0.0
RM			87.8	87.8

Table 1: Number of positive training (Tr$^+$) and test (Te$^+$) examples in the SRL dataset. Accuracy of the non-linearized (SST) and linearized (SST$_\ell$) binary classifiers (i.e. BC, A0, ... A5) is F_1 measure. Accuracy of RM is the percentage of correct class assignments.

We should consider that the linearization framework can drastically improve the efficiency of learning and classification when dealing with large amounts of data. For a linearized classifier, we consider *training time* to be the overall time required to carry out the following activities: KSL, FMI, TFX on training data and ESL. Similarly, we consider test time the time necessary to perform TFX on test data and ESC. Training BC took more than two days of CPU time and testing about 4 hours, while training and testing the linearized boundary classifier required only 381 and 25 minutes, respectively. That is, on the same amount of data we can train a linearized classifier about 8 times as fast, and test it in about 1 tenth of the time. Concerning RM, sequential training of the 6 models took 2,596 minutes, while testing took 27 minutes. The linearized role multi classifier required 448 and 24 minutes for training and testing, respectively, i.e. training is about 5 times as fast while testing time is about the same. If compared with the boundary classifier, the improvement in efficiency is less evident: indeed, the relatively small size of the role classifiers data sets limits the positive effect of splitting training data into smaller chunks.

SRL fragment space. Table 3 lists the best fragments identified for the Boundary Classifier. We should remember that we are using AST$_m$ struc-

tures as input to our classifiers: nodes whose label end with "-P" are predicate nodes, while nodes whose label ends with "-B" are candidate argument nodes.

All the most relevant fragments encode the minimum sub-tree encompassing the predicate and the argument node. This kind of structured feature subsumes several features traditionally employed for explicit SRL models: the Path (i.e. the sequence of nodes connecting the predicate and the candidate argument node), Phrase Type (i.e. the label of the candidate argument node), Predicate POS (i.e. the POS of the predicate word), Position (i.e. whether the argument is to the left or to the right of the predicate) and Governing Category (i.e. the label of the common ancestor) defined in (Gildea and Jurafsky, 2002).

The linearized model for BC contains about 160 thousand fragments. Of these, about 70 and 33 thousand encompass the candidate argument or the predicate node, respectively. About 16 thousand fragments contain both.

5.2 Question Classification

For question classification we used the data set from the TREC 10 QA evaluation campaign[2], consisting of 5,500 training and 500 test questions.

5.2.1 Setup

Given a question, the QC task consists in selecting the most appropriate expected answer type from a given set of possibilities. We adopted the question taxonomy known as *coarse grained*, which has been described in (Zhang and Lee, 2003) and (Li and Roth, 2006), consisting of six non overlapping classes: Abbreviations (ABBR), Descriptions (DESC, e.g. definitions or explanations), Entity (ENTY, e.g. animal, body or color), Human (HUM, e.g. group or individual), Location (LOC, e.g. cities or countries) and Numeric (NUM, e.g. amounts or dates).

For each question, we generate the full parse of the sentence and use it to train SST and (linearized) SST_ℓ models. The automatic parses are obtained with the Stanford parser[3] (Klein and Manning, 2003). We actually have only 5,483 sentences in our training set, due to parsing issues with a few of them.

[2] `http://l2r.cs.uiuc.edu/cogcomp/Data/QA/QC/`

[3] `http://nlp.stanford.edu/software/lex-parser.shtml`

Class	Data set Tr^+	Data set Te^+	Accuracy SST	Accuracy SST_ℓ
ABBR	89	9	80.0	87.5
DESC	1,164	138	96.0	94.5
ENTY	1,269	94	63.9	63.5
HUM	1,231	65	88.1	87.2
LOC	834	81	77.6	77.9
NUM	896	113	80.4	80.8
Overall			86.2	86.6

Table 2: Number of positive training (Tr^+) and test (Te^+) examples in the QA dataset. Accuracy of the non-linearized (SST) and linearized (SST_ℓ) binary classifiers is F_1 measure. Overall accuracy is the percentage of correct class assignments.

The classifiers are arranged in a one-vs.-all (OvA) configuration, where each sentence is a positive example for one of the six classes, and negative for the other five. Given the very small size of the data set, we used $S = 1$ during KSL for the linearized classifier (i.e. we didn't partition training data). We carried out no validation of the parameters, and we used $maxexp = 4$ and $L = 50k$ in order to generate a rich fragment space.

5.2.2 Results

Table 2 shows the number of positive examples in the training and test set of each individual binary classifiers. Columns SST and SST_ℓ compare the F_1 measure of the vanilla and linearized classifiers on the individual classes, and the accuracy of the complete QC task (Row *Overall*) in terms of percentage of correct class assignments. Also in this case, we can notice that the accuracy of the linearized classifiers is always in line with non-linearized ones, e.g. 86.6 vs. 86.2 for the multi-classifiers. These results are lower than those derived in (Moschitti, 2006; Moschitti et al., 2007), i.e. 88.2 and 90.4, respectively, where the parameters for each classifier were carefully optimized.

QC Fragment space. Tables from 4 to 9 list the top fragments identified for each class [4].

As expected, for all the categories the domain lexical information is very relevant. For example, *film*, *color*, *book*, *novel* and *sport* for ENTY or *city*, *country*, *state* and *capital* for LOC. Of the six classes, ENTY (Table 6) is mostly characterized by lexical features. Interestingly, function words, which would have been eliminated by a pure Information Retrieval approach (i.e. by means of

[4] Some categories show meaningful syntactic fragments after the first 10, so for them we report more subtrees.

standard stop-list), are in the top positions, e.g.: *why* and *how* for DESC, *what* for ENTY, *who* for HUM, *where* for LOC and *when* for NUM. For the latter, also *how* seems to be important suggesting that features may strongly characterize more than one given class.

Characteristic syntactic features appear in the top positions for each class, for example: *(VP (VB (stand)) (PP))*, which suggests that *stand* should be followed by a prepositional phrase to characterize ABBR; or *(NP (NP (DT) (NN (abbreviation))) (PP))*, which suggests that, to be in a relevant pattern, *abbreviation* should be preceded by an article and followed by a PP. Also, the syntactic structure is useful to differentiate the use of the same important words, e.g. *(SBARQ (WHADVP (WRB (How))) (SQ) (.))* for DESC better characterizes the use of *how* with respect to NUM, in which a relevant use is *(WHADJP (WRB (How)) (JJ))*.

In (Moschitti et al., 2007) it was shown that the use of TK improves QC of 1.2 percent points, i.e. from 90.6 to 91.8: further analysis of these fragments may help us to device compact, less sparse syntactic features and design more accurate models for the task.

6 Discussion

The fact that our model doesn't always improve the accuracy of a standard SST model might be related to the process of splitting training data and employing locally estimated weights during FMI.

Concerning the experiments presented in this paper, this objection might apply to the results on SRL, where we used 50 splits to identify the most relevant fragments, but not to those on QC, where given the limited size of the data set we decided not to split training data at all as explained in Section 5.2. Furthermore, as we already discussed, we have evidence that there is no direct correlation between the number of splits used for KSL and the accuracy of the resulting classifier. After all, the optimization carried out during ESL is global, and we can assume that, if we mined enough fragments during FMI, than those actually retained by the global linear model would be by and large the same, regardless of the split configuration.

More in general, feature selection may give an improvement to some learning algorithm but if it can help SVMs is debatable, since its related theory show that they are robust to irrelevant features. In our specific case, we remove features

(ADJP(RB-B)(VBN-P))
(NP(VBN-P)(NNS-B))
(S(NP-B)(VP))
(VP(VBD-P(said))(SBAR))
(VP(VB-P)(NP-B))
(NP(VBG-P)(NNS-B))
(VP(VBD-P)(NP-B))
(VP(VBG-P)(NP-B))
(VP(VBZ-P)(NP-B))
(VP(VBN-P)(NP-B))
(VP(VBP-P)(NP-B))
(NP(NP-B)(VP))
(NP(VBG-P)(NN-B))
(S(S(VP(VBG-P)))(NP-B))

Table 3: Best fragments for SRL BC.

(NN(abbreviation))
(NP(DT)(NN(abbreviation)))
(NP(DT(the))(NN(abbreviation)))
(IN(for))
(VB(stand))
(VBZ(does))
(PP(IN))
(VP(VB(stand))(PP))
(NP(NP(DT)(NN(abbreviation)))(PP))
(SQ(VBZ)(NP)(VP(VB(stand))(PP)))
(SBARQ(WHNP)(SQ(VBZ)(NP)(VP(VB(stand))(PP)))(.))
(SQ(VBZ(does))(NP)(VP(VB(stand))(PP)))
(VP(VBZ)(NP(NP(DT)(NN(abbreviation)))(PP)))

Table 4: Best fragments for the ABBR class.

(WRB(Why))
(WHADVP(WRB(Why)))
(WHADVP(WRB(How)))
(WHADVP(WRB))
(VB(mean))
(VBZ(causes))
(VB(do))
(ROOT(SBARQ(WHADVP(WRB(How)))(SQ)(.)))
(ROOT(SBARQ(WHADVP(WRB(How)))(SQ)(.(?))))
(SBARQ(WHADVP(WRB(How)))(SQ))
(WRB(How))
(SBARQ(WHADVP(WRB(How)))(SQ)(.))
(SBARQ(WHADVP(WRB(How)))(SQ)(.(?)))
(SBARQ(WHADVP(WRB(Why)))(SQ))
(ROOT(SBARQ(WHADVP(WRB(Why)))(SQ)))
(SBARQ(WHADVP(WRB))(SQ))

Table 5: Best fragments for the DESC class.

(NN(film))
(NN(color))
(NN(book))
(NN(novel))
(NN(sport))
(WP(What))
(NN(fear))
(NN(movie))
(NN(word))
(VP(VBN(called)))
(NN(game))
(NP(DT)(NN(fear)))
(NP(NP(DT)(NN(fear)))(PP))

Table 6: Best fragments for the ENTY class.

(NN(company))
(WP(Who))
(WHNP(WP(Who)))
(NN(name))
(NN(team))
(NN(baseball))
(WHNP(WP))
(NN(character))
(NNP(President))
(NN(leader))
(NN(actor))
(NN(president))
(JJ(Whose))
(VP(VBD)(NP))
(NP(NP)(JJ)(NN(name)))
(VP(VBD)(VP))
(NN(organization))
(VP(VBD)(NP)(PP(IN)(NP)))
(SBARQ(WHNP(WP(Who)))(SQ)(.))
(ROOT(SBARQ(WHNP(WP(Who)))(SQ)(.)))
(ROOT(SBARQ(WHNP(WP(Who)))(SQ)(.(?))))
(SBARQ(WHNP(WP(Who)))(SQ)(.(?)))

Table 7: Best fragments for the HUM class.

(NN(city))
(NN(country))
(WRB(Where))
(NN(state))
(WHADVP(WRB(Where)))
(NN(capital))
(NP(NN(city)))
(NNS(countries))
(NP(NN(state)))
(PP(IN(in)))
(SBARQ(WHADVP(WRB(Where)))(SQ)(.(?)))
(SBARQ(WHADVP(WRB(Where)))(SQ)(.))
(ROOT(SBARQ(WHADVP(WRB(Where)))(SQ)(.)))
(ROOT(SBARQ(WHADVP(WRB(Where)))(SQ)(.(?))))
(NN(island))
(NN(address))
(NN(river))
(NN(mountain))
(ROOT(SBARQ(WHADVP(WRB(Where)))(SQ)))
(SBARQ(WHADVP(WRB(Where)))(SQ))

Table 8: Best fragments for the LOC class.

(WRB(How))
(WHADVP(WRB(When)))
(WRB(When))
(JJ(many))
(NN(year))
(WHADJP(WRB)(JJ))
(NP(NN(year)))
(WHADJP(WRB(How))(JJ))
(NN(date))
(SBARQ(WHADVP(WRB(When)))(SQ)(.(?)))
(SBARQ(WHADVP(WRB(When)))(SQ)(.))
(NN(day))
(NN(population))
(ROOT(SBARQ(WHADVP(WRB(When)))(SQ)(.)))
(ROOT(SBARQ(WHADVP(WRB(When)))(SQ)(.(?))))
(JJ(average))
(NN(number))

Table 9: Best fragments for the NUM class.

whose SVM weights are the lowest, i.e. those that are (almost) irrelevant for the SVM. Therefore, the chance of this resulting in an improvement is rather low.

With respect to cases where our model is less accurate than a standard SST, we should consider that our choice of parameters is sub-optimal and we adopt a *very* aggressive feature selection strategy, that only retains a few thousand features from a space where there are hundreds of millions of different features.

7 Conclusions

We introduced a novel framework for support vector classification that combines advantages of convolution kernels, i.e. the generation of a very high dimensional structure space, with the efficiency and clarity of explicit representations in a linear space.

For this paper, we focused on the SubSet Tree kernel and verified the potential of the proposed solution on two NLP tasks, i.e. semantic role labeling and question classification. The experiments show that our framework drastically reduces processing time, e.g. boundary classification for SRL, while preserving the accuracy.

We presented a selection of the most relevant fragments identified for the SRL boundary classifier as well as for each class of the coarse grained QC task. Our analysis shows that our framework can discover state-of-the-art features, e.g. the Path feature for SRL. We believe that sharing these fragments with the NLP community and studying them in more depth will be useful to identify new, relevant features for the characterization of several learning problems. For this purpose, we made available the fragment spaces at `http://danielepighin.net` and we will keep them updated with new set of experiments on new tasks, e.g. SRL based on FrameNet and VerbNet, e.g. (Giuglea and Moschitti, 2004).

In our future work, we plan to widen the list of covered tasks and to extend our algorithm to cope with different kernel families, such as the partial tree kernel and kernels defined over pairs of trees, e.g. the ones used for textual entailment in (Moschitti and Zanzotto, 2007). We also plan to move from mining fragments to mining classes of fragments, i.e. to identify prototypical fragments in the fragment space that generalize topological sub-classes of the most relevant fragments.

References

Fabio Aiolli, Giovanni Da San Martino, Alessandro Sperduti, and Alessandro Moschitti. 2006. Fast on-line kernel learning for trees. In *Proceedings of ICDM'06*.

Stephan Bloehdorn and Alessandro Moschitti. 2007a. Combined syntactic and semantic kernels for text classification. In *Proceedings of ECIR 2007, Rome, Italy*.

Stephan Bloehdorn and Alessandro Moschitti. 2007b. Structure and semantics for expressive text kernels. In *In Proceedings of CIKM '07*.

Nicola Cancedda, Eric Gaussier, Cyril Goutte, and Jean Michel Renders. 2003. Word sequence kernels. *Journal of Machine Learning Research*, 3:1059–1082.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proceedings of CoNLL'05*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL'00*.

Michael Collins and Nigel Duffy. 2002. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of ACL'02*.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency Tree Kernels for Relation Extraction. In *Proceedings of ACL'04*.

Chad Cumby and Dan Roth. 2003. Kernel Methods for Relational Learning. In *Proceedings of ICML 2003*.

Mona Diab, Alessandro Moschitti, and Daniele Pighin. 2008. Semantic role labeling systems for Arabic using kernel methods. In *Proceedings of ACL-08: HLT*, pages 798–806.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28:245–288.

Ana-Maria Giuglea and Alessandro Moschitti. 2004. Knowledge discovery using framenet, verbnet and propbank. In A. Meyers, editor, *Workshop on Ontology and Knowledge Discovering at ECML 2004*, Pisa, Italy.

Alfio Gliozzo, Claudio Giuliano, and Carlo Strapparava. 2005. Domain kernels for word sense disambiguation. In *Proceedings of ACL'05*, pages 403–410.

Hans P. Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. 2004. Parallel support vector machines: The cascade svm. In *Neural Information Processing Systems*.

Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

David Haussler. 1999. Convolution kernels on discrete structures. Technical report, Dept. of Computer Science, University of California at Santa Cruz.

T. Joachims. 2000. Estimating the generalization performance of a SVM efficiently. In *Proceedings of ICML'00*.

Hisashi Kashima and Teruo Koyanagi. 2002. Kernels for semi-structured data. In *Proceedings of ICML'02*.

Jun'ichi Kazama and Kentaro Torisawa. 2005. Speeding up training with tree kernels for node relation labeling. In *Proceedings of HLT-EMNLP'05*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL'03*, pages 423–430.

Taku Kudo and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proceedings of ACL'03*.

Taku Kudo, Jun Suzuki, and Hideki Isozaki. 2005. Boosting-based parse reranking with subtree features. In *Proceedings of ACL'05*.

Xin Li and Dan Roth. 2006. Learning question classifiers: the role of semantic information. *Natural Language Engineering*, 12(3):229–249.

Alessandro Moschitti and Fabio Massimo Zanzotto. 2007. Fast and effective kernels for relational learning from texts. In *ICML'07*.

Alessandro Moschitti, Silvia Quarteroni, Roberto Basili, and Suresh Manandhar. 2007. Exploiting syntactic and shallow semantic kernels for question/answer classification. In *Proceedings of ACL'07*.

Alessandro Moschitti, Daniele Pighin, and Roberto Basili. 2008. Tree kernels for semantic role labeling. *Computational Linguistics*, 34(2):193–224.

Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of ECML'06*, pages 318–329.

Julia Neumann, Christoph Schnorr, and Gabriele Steidl. 2005. Combined SVM-Based Feature Selection and Classification. *Machine Learning*, 61(1-3):129–150.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Comput. Linguist.*, 31(1):71–106.

J. Pei, J. Han, Mortazavi B. Asl, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu. 2001. PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In *Proceedings of ICDE'01*.

Daniele Pighin and Alessandro Moschitti. 2009. Efficient linearization of tree kernel functions. In *Proceedings of CoNLL'09*.

Alain Rakotomamonjy. 2003. Variable selection using SVM based criteria. *Journal of Machine Learning Research*, 3:1357–1370.

Libin Shen, Anoop Sarkar, and Aravind k. Joshi. 2003. Using LTAG Based Features in Parse Reranking. In *Proceedings of EMNLP'06*.

Jun Suzuki and Hideki Isozaki. 2005. Sequence and Tree Kernels with Statistical Feature Mining. In *Proceedings of NIPS'05*.

Kristina Toutanova, Penka Markova, and Christopher Manning. 2004. The Leaf Path Projection View of Parse Trees: Exploring String Kernels for HPSG Parse Selection. In *Proceedings of EMNLP 2004*.

Vladimir N. Vapnik. 1998. *Statistical Learning Theory*. Wiley-Interscience.

Ellen M. Voorhees. 2001. Overview of the trec 2001 question answering track. In *In Proceedings of the Tenth Text REtrieval Conference (TREC*, pages 42–51.

Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, and Vladimir Vapnik. 2001. Feature Selection for SVMs. In *Proceedings of NIPS'01*.

Jason Weston, André Elisseeff, Bernhard Schölkopf, and Mike Tipping. 2003. Use of the zero norm with linear models and kernel methods. *J. Mach. Learn. Res.*, 3:1439–1461.

Dell Zhang and Wee Sun Lee. 2003. Question classification using support vector machines. In *Proceedings of SIGIR'03*, pages 26–32.