Re-Ranking Models for Spoken Language Understanding

Marco Dinarelli
University of Trento
Italy
dinarelli@disi.unitn.it

Alessandro Moschitti
University of Trento
Italy
moschitti@disi.unitn.it

Giuseppe Riccardi
University of Trento
Italy
riccardi@disi.unitn.it

Abstract

Spoken Language Understanding aims at mapping a natural language spoken sentence into a semantic representation. In the last decade two main approaches have been pursued: generative and discriminative models. The former is more robust to overfitting whereas the latter is more robust to many irrelevant features. Additionally, the way in which these approaches encode prior knowledge is very different and their relative performance changes based on the task. In this paper we describe a machine learning framework where both models are used: a generative model produces a list of ranked hypotheses whereas a discriminative model based on structure kernels and Support Vector Machines, re-ranks such list. We tested our approach on the MEDIA corpus (human-machine dialogs) and on a new corpus (human-machine and human-human dialogs) produced in the European LUNA project. The results show a large improvement on the state-of-the-art in concept segmentation and labeling.

1 Introduction

In Spoken Dialog Systems, the Language Understanding module performs the task of translating a spoken sentence into its meaning representation based on semantic constituents. These are the units for meaning representation and are often referred to as concepts. Concepts are instantiated by sequences of words, therefore a Spoken Language Understanding (SLU) module finds the association between words and concepts.

In the last decade two major approaches have been proposed to find this correlation: (i) generative models, whose parameters refer to the joint probability of concepts and constituents; and (ii) discriminative models, which learn a classification function to map words into concepts based on geometric and statistical properties. An example of generative model is the Hidden Vector State model (HVS) (He and Young, 2005). This approach extends the discrete Markov model encoding the context of each state as a vector. State transitions are performed as stack shift operations followed by a push of a preterminal semantic category label. In this way the model can capture semantic hierarchical structures without the use of tree-structured data. Another simpler but effective generative model is the one based on Finite State Transducers. It performs SLU as a translation process from words to concepts using Finite State Transducers (FST). An example of discriminative model used for SLU is the one based on Support Vector Machines (SVMs) (Vapnik, 1995), as shown in (Raymond and Riccardi, 2007). In this approach, data are mapped into a vector space and SLU is performed as a classification problem using Maximal Margin Classifiers (Shawe-Taylor and Cristianini, 2004).

Generative models have the advantage to be more robust to overfitting on training data, while discriminative models are more robust to irrelevant features. Both approaches, used separately, have shown a good performance (Raymond and Riccardi, 2007), but they have very different characteristics and the way they encode prior knowledge is very different, thus designing models able to take into account characteristics of both approaches are particularly promising.

In this paper we propose a method for SLU based on generative and discriminative models: the former uses FSTs to generate a list of SLU hypotheses, which are re-ranked by SVMs. These exploit all possible word/concept subsequences (with gaps) of the spoken sentence as features (*i.e.* all possible n-grams). Gaps allow for the encod-

ing of long distance dependencies between words in relatively small n-grams. Given the huge size of this feature space, we adopted kernel methods and in particular sequence kernels (Shawe-Taylor and Cristianini, 2004) and tree kernels (Raymond and Riccardi, 2007; Moschitti and Bejan, 2004; Moschitti, 2006) to implicitly encode n-grams and other structural information in SVMs.

We experimented with different approaches for training the discriminative models and two different corpora: the well-known MEDIA corpus (Bonneau-Maynard et al., 2005) and a new corpus acquired in the European project LUNA[1] (Raymond et al., 2007). The results show a great improvement with respect to both the FST-based model and the SVM model alone, which are the current state-of-the-art for concept classification on such corpora. The rest of the paper is organized as follows: Sections 2 and 3 show the generative and discriminative models, respectively. The experiments and results are reported in Section 4 whereas the conclusions are drawn in Section 5.

2 Generative approach for concept classification

In the context of Spoken Language Understanding (SLU), concept classification is the task of associating the best sequence of concepts to a given sentence, *i.e.* word sequence. A concept is a class containing all the words carrying out the same semantic meaning with respect to the application domain. In SLU, concepts are used as semantic units and are represented with concept tags. The association between words and concepts is learned from an annotated corpus.

The Generative model used in our work for concept classification is the same used in (Raymond and Riccardi, 2007). Given a sequence of words as input, a translation process based on FST is performed to output a sequence of concept tags. The translation process involves three steps: (1) the mapping of words into classes (2) the mapping of classes into concepts and (3) the selection of the best concept sequence.

The first step is used to improve the generalization power of the model. The word classes at this level can be both domain-dependent, *e.g.* "Hotel" in MEDIA or "Software" in the LUNA corpus, or domain-independent, *e.g.* numbers, dates, months etc. The class of a word not belonging to any class is the word itself.

In the second step, classes are mapped into concepts. The mapping is not one-to-one: a class may be associated with more than one concept, *i.e.* more than one SLU hypothesis can be generated.

In the third step, the best or the m-best hypotheses are selected among those produced in the previous step. They are chosen according to the maximum probability evaluated by the Conceptual Language Model, described in the next section.

2.1 Stochastic Conceptual Language Model (SCLM)

An SCLM is an n-gram language model built on semantic tags. Using the same notation proposed in (Moschitti et al., 2007) and (Raymond and Riccardi, 2007), our SCLM trains joint probability $P(W, C)$ of word and concept sequences from an annotated corpus:

$$P(W, C) = \prod_{i=1}^{k} P(w_i, c_i | h_i),$$

where $W = w_1..w_k$, $C = c_1..c_k$ and $h_i = w_{i-1}c_{i-1}..w_1c_1$. Since we use a 3-gram conceptual language model, the history h_i is $\{w_{i-1}c_{i-1}, w_{i-2}c_{i-2}\}$.

All the steps of the translation process described here and above are implemented as Finite State Transducers (FST) using the AT&T FSM/GRM tools and the SRILM (Stolcke, 2002) tools. In particular the SCLM is trained using SRILM tools and then converted to an FST. This allows the use of a wide set of stochastic language models (both back-off and interpolated models with several discounting techniques like Good-Turing, Witten-Bell, Natural, Kneser-Ney, Unchanged Kneser-Ney etc). We represent the combination of all the translation steps as a transducer λ_{SLU} (Raymond and Riccardi, 2007) in terms of FST operations:

$$\lambda_{SLU} = \lambda_W \circ \lambda_{W2C} \circ \lambda_{SLM},$$

where λ_W is the transducer representation of the input sentence, λ_{W2C} is the transducer mapping words to classes and λ_{SLM} is the Semantic Language Model (SLM) described above. The best SLU hypothesis is given by

$$C = project_C(bestpath_1(\lambda_{SLU})),$$

where $bestpath_n$ (in this case n is 1 for the 1-best hypothesis) performs a Viterbi search on the FST

[1] Contract n. 33549

and outputs the n-best hypotheses and $project_C$ performs a projection of the FST on the output labels, in this case the concepts.

2.2 Generation of m-best concept labeling

Using the FSTs described above, we can generate *m* best hypotheses ranked by the joint probability of the SCLM.

After an analysis of the *m*-best hypotheses of our SLU model, we noticed that many times the hypothesis ranked first by the SCLM is not the closest to the correct concept sequence, *i.e.* its error rate using the Levenshtein alignment with the manual annotation of the corpus is not the lowest among the *m* hypotheses. This means that re-ranking the *m*-best hypotheses in a convenient way could improve the SLU performance. The best choice in this case is a discriminative model, since it allows for the use of informative features, which, in turn, can model easily feature dependencies (also if they are infrequent in the training set).

3 Discriminative re-ranking

Our discriminative re-ranking is based on SVMs or a perceptron trained with pairs of conceptually annotated sentences. The classifiers learn to select which annotation has an error rate lower than the others so that the *m*-best annotations can be sorted based on their correctness.

3.1 SVMs and Kernel Methods

Kernel Methods refer to a large class of learning algorithms based on inner product vector spaces, among which Support Vector Machines (SVMs) are one of the most well known algorithms. SVMs and perceptron learn a hyperplane $H(\vec{x}) = \vec{w}\vec{x} + b = 0$, where $\vec{x}$ is the feature vector representation of a classifying object *o*, $\vec{w} \in \mathbb{R}^n$ (a vector space) and $b \in \mathbb{R}$ are parameters (Vapnik, 1995). The classifying object *o* is mapped into $\vec{x}$ by a feature function ϕ. The kernel trick allows us to rewrite the decision hyperplane as $\sum_{i=1..l} y_i \alpha_i \phi(o_i)\phi(o) + b = 0$, where y_i is equal to 1 for positive and -1 for negative examples, $\alpha_i \in \mathbb{R}^+$, $o_i \forall i \in \{1..l\}$ are the training instances and the product $K(o_i, o) = \langle \phi(o_i)\phi(o) \rangle$ is the kernel function associated with the mapping ϕ. Note that we do not need to apply the mapping ϕ, we can use $K(o_i, o)$ directly (Shawe-Taylor and Cristianini, 2004). For example, next section shows a kernel function that counts the number of word sequences in common between two sentences, in the space of *n*-grams (for any *n*).

3.2 String Kernels

The String Kernels that we consider count the number of substrings containing gaps shared by two sequences, *i.e.* some of the symbols of the original string are skipped. Gaps modify the weight associated with the target substrings as shown in the following.

Let Σ be a finite alphabet, $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ is the set of all strings. Given a string $s \in \Sigma^*$, $|s|$ denotes the length of the strings and s_i its compounding symbols, i.e $s = s_1..s_{|s|}$, whereas $s[i:j]$ selects the substring $s_i s_{i+1}..s_{j-1}s_j$ from the i-th to the j-th character. u is a subsequence of s if there is a sequence of indexes $\vec{I} = (i_1, ..., i_{|u|})$, with $1 \leq i_1 < ... < i_{|u|} \leq |s|$, such that $u = s_{i_1}..s_{i_{|u|}}$ or $u = s[\vec{I}]$ for short. $d(\vec{I})$ is the distance between the first and last character of the subsequence u in s, i.e. $d(\vec{I}) = i_{|u|} - i_1 + 1$. Finally, given s_1, s_2 $\in \Sigma^*$, $s_1 s_2$ indicates their concatenation.

The set of all substrings of a text corpus forms a feature space denoted by $\mathcal{F} = \{u_1, u_2, ..\} \subset \Sigma^*$. To map a string s in $\mathbb{R}^\infty$ space, we can use the following functions: $\phi_u(s) = \sum_{\vec{I}:u=s[\vec{I}]} \lambda^{d(\vec{I})}$ for some $\lambda \leq 1$. These functions count the number of occurrences of u in the string s and assign them a weight $\lambda^{d(\vec{I})}$ proportional to their lengths. Hence, the inner product of the feature vectors for two strings s_1 and s_2 returns the sum of all common subsequences weighted according to their frequency of occurrences and lengths, i.e.

$$SK(s_1, s_2) = \sum_{u \in \Sigma^*} \phi_u(s_1) \cdot \phi_u(s_2) = \sum_{u \in \Sigma^*} \sum_{\vec{I}_1 : u = s_1[\vec{I}_1]} \lambda^{d(\vec{I}_1)}$$

$$\sum_{\vec{I}_2 : u = s_2[\vec{I}_2]} \lambda^{d(\vec{I}_2)} = \sum_{u \in \Sigma^*} \sum_{\vec{I}_1 : u = s_1[\vec{I}_1]} \sum_{\vec{I}_2 : u = s_2[\vec{I}_2]} \lambda^{d(\vec{I}_1) + d(\vec{I}_2)},$$

where $d(.)$ counts the number of characters in the substrings as well as the gaps that were skipped in the original string. It is worth noting that:

(a) longer subsequences receive lower weights;

(b) some characters can be omitted, *i.e.* gaps; and

(c) gaps determine a weight since the exponent of λ is the number of characters and gaps between the first and last character.

Characters in the sequences can be substituted with any set of symbols. In our study we preferred to use words so that we can obtain word sequences. For example, given the sentence: *How may I help you ?* sample substrings, extracted by the Sequence Kernel (SK), are: *How help you ?*, *How help ?*, *help you*, *may help you*, etc.

3.3 Tree kernels

Tree kernels represent trees in terms of their substructures (fragments). The kernel function detects if a tree subpart (common to both trees) belongs to the feature space that we intend to generate. For such purpose, the desired fragments need to be described. We consider two important characterizations: the syntactic tree (STF) and the partial tree (PTF) fragments.

3.3.1 Tree Fragment Types

An STF is a general subtree whose leaves can be non-terminal symbols. For example, Figure 1(a) shows 10 STFs (out of 17) of the subtree rooted in VP (of the left tree). The STFs satisfy the constraint that grammatical rules cannot be broken. For example, [VP [V NP]] is an STF, which has two non-terminal symbols, V and NP, as leaves whereas [VP [V]] is not an STF. If we relax the constraint over the STFs, we obtain more general substructures called *partial trees fragments* (PTFs). These can be generated by the application of partial production rules of the grammar, consequently [VP [V]] and [VP [NP]] are valid PTFs. Figure 1(b) shows that the number of PTFs derived from the same tree as before is still higher (*i.e.* 30 PTs).

3.4 Counting Shared SubTrees

The main idea of tree kernels is to compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. To evaluate the above kernels between two T_1 and T_2, we need to define a set $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$, *i.e.* a tree fragment space and an indicator function $I_i(n)$, equal to 1 if the target f_i is rooted at node n and equal to 0 otherwise. A tree-kernel function over T_1 and T_2 is $TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$, where N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_1) I_i(n_2)$. The latter is equal to the number of common fragments rooted in the n_1 and n_2 nodes. In the following sections we report the equation for the efficient evaluation of Δ for ST and PT kernels.

3.5 Syntactic Tree Kernels (STK)

The Δ function depends on the type of fragments that we consider as *basic* features. For example, to evaluate the fragments of type STF, it can be defined as:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (*i.e.* they are pre-terminals symbols) then $\Delta(n_1, n_2) = 1$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then

$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (\sigma + \Delta(c_{n_1}^j, c_{n_2}^j)) \quad (1)$$

where $\sigma \in \{0, 1\}$, $nc(n_1)$ is the number of children of n_1 and c_n^j is the j-th child of the node n. Note that, since the productions are the same, $nc(n_1) = nc(n_2)$. $\Delta(n_1, n_2)$ evaluates the number of STFs common to n_1 and n_2 as proved in (Collins and Duffy, 2002).

Moreover, a decay factor λ can be added by modifying steps (2) and (3) as follows[2]:

2. $\Delta(n_1, n_2) = \lambda$,
3. $\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (\sigma + \Delta(c_{n_1}^j, c_{n_2}^j))$.

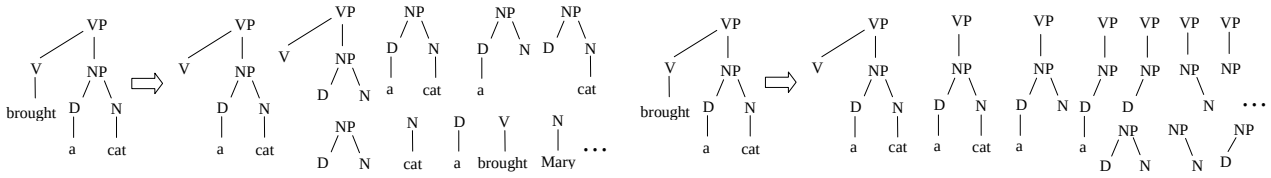
The computational complexity of Eq. 1 is $O(|N_{T_1}| \times |N_{T_2}|)$ but as shown in (Moschitti, 2006), the average running time tends to be linear, *i.e.* $O(|N_{T_1}| + |N_{T_2}|)$, for natural language syntactic trees.

3.6 The Partial Tree Kernel (PTK)

PTFs have been defined in (Moschitti, 2006). Their computation is carried out by the following Δ function:

1. if the node labels of n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. else $\Delta(n_1, n_2) =$
   $1 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1) = l(\vec{I}_2)} \prod_{j=1}^{l(\vec{I}_1)} \Delta(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))$

[2] To have a similarity score between 0 and 1, we also apply the normalization in the kernel space, *i.e.*: $K'(T_1, T_2) = \frac{TK(T_1, T_2)}{\sqrt{TK(T_1, T_1) \times TK(T_2, T_2)}}$.



(a) Syntactic Tree fragments (STF)

(b) Partial Tree fragments (PTF)

Figure 1: Examples of different classes of tree fragments.

where $\vec{I}_1 = \langle h_1, h_2, h_3, .. \rangle$ and $\vec{I}_2 = \langle k_1, k_2, k_3, .. \rangle$ are index sequences associated with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2, respectively, $\vec{I}_{1j}$ and $\vec{I}_{2j}$ point to the j-th child in the corresponding sequence, and, again, $l(\cdot)$ returns the sequence length, *i.e.* the number of children.

Furthermore, we add two decay factors: μ for the depth of the tree and λ for the length of the child subsequences with respect to the original sequence, *i.e.* we account for gaps. It follows that $\Delta(n_1, n_2) =$

$$\mu\Big(\lambda^2 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1)=l(\vec{I}_2)} \lambda^{d(\vec{I}_1)+d(\vec{I}_2)} \prod_{j=1}^{l(\vec{I}_1)} \Delta(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))\Big), \quad (2)$$

where $d(\vec{I}_1) = \vec{I}_{1l(\vec{I}_1)} - \vec{I}_{11}$ and $d(\vec{I}_2) = \vec{I}_{2l(\vec{I}_2)} - \vec{I}_{21}$. This way, we penalize both larger trees and child subsequences with gaps. Eq. 2 is more general than Eq. 1. Indeed, if we only consider the contribution of the longest child sequence from node pairs that have the same children, we implement the STK kernel.

3.7 Re-ranking models using sequences

The FST generates the *m* most likely concept annotations. These are used to build annotation pairs, $\langle s^i, s^j \rangle$, which are positive instances if s^i has a lower concept annotation error than s^j, with respect to the manual annotation in the corpus. Thus, a trained binary classifier can decide if s^i is more accurate than s^j. Each candidate annotation s^i is described by a word sequence where each word is followed by its concept annotation. For example, given the sentence:

ho (I have) un (a) problema (problem) con (with) la (the) scheda di rete (network card) ora (now)

a pair of annotations $\langle s^i, s^j \rangle$ could be

s^i: *ho* ***NULL*** *un* ***NULL*** *problema* ***PROBLEM-B*** *con* ***NULL*** *la* ***NULL*** *scheda* ***HW-B*** *di* ***HW-I*** *rete* ***HW-I*** *ora* ***RELATIVETIME-B***

s^j: *ho* ***NULL*** *un* ***NULL*** *problema* ***ACTION-B*** *con* ***NULL*** *la* ***NULL*** *scheda* ***HW-B*** *di* ***HW-B*** *rete* ***HW-B*** *ora* ***RELATIVETIME-B***

where **NULL**, **ACTION**, **RELATIVETIME**, and **HW** are the assigned concepts whereas **B** and **I** are the usual begin and internal tags for concept subparts. The second annotation is less accurate than the first since *problema* is annotated as an action and *"scheda di rete"* is split in three different concepts.

Given the above data, the sequence kernel is used to evaluate the number of common *n*-grams between s^i and s^j. Since the string kernel skips some elements of the target sequences, the counted *n*-grams include: concept sequences, word sequences and any subsequence of words and concepts at any distance in the sentence.

Such counts are used in our re-ranking function as follows: let e_i be the pair $\langle s_i^1, s_i^2 \rangle$ we evaluate the kernel:

$$\begin{aligned} K_R(e_1, e_2) &= SK(s_1^1, s_2^1) + SK(s_1^2, s_2^2) \\ &- SK(s_1^1, s_2^2) - SK(s_1^2, s_2^1) \end{aligned} \quad (3)$$

This schema, consisting in summing four different kernels, has been already applied in (Collins and Duffy, 2002) for syntactic parsing re-ranking, where the basic kernel was a tree kernel instead of SK and in (Moschitti et al., 2006), where, to rerank Semantic Role Labeling annotations, a tree kernel was used on a semantic tree similar to the one introduced in the next section.

3.8 Re-ranking models using trees

Since the aim in concept annotation re-ranking is to exploit innovative and effective source of information, we can use the power of tree kernels to generate correlation between concepts and word structures.

Fig. 2 describes the structural association between the concept and the word level. This kind of trees allows us to engineer new kernels and consequently new features (Moschitti et al., 2008),

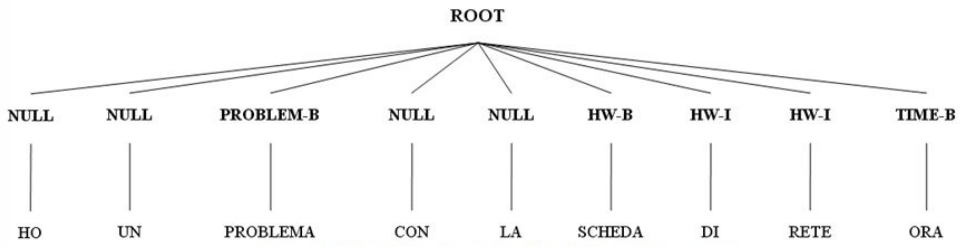


Figure 2: An example of the semantic tree used for STK or PTK

Corpus LUNA	Train set		Test set	
	words	concepts	words	concepts
Dialogs WOZ	183		67	
Dialogs HH	180		-	
Turns WOZ	1.019		373	
Turns HH	6.999		-	
Tokens WOZ	8.512	2.887	2.888	984
Tokens WOZ	62.639	17.423	-	-
Vocab. WOZ	1.172	34	-	-
Vocab. HH	4.692	49	-	-
OOV rate	-	-	3.2%	0.1%

Table 1: Statistics on the LUNA corpus

Corpus Media	Train set		Test set	
	words	concepts	words	concepts
Turns	12,922		3,518	
# of tokens	94,912	43,078	26,676	12,022
Vocabulary	5,307	80	-	-
OOV rate	-	-	0.01%	0.0%

Table 2: Statistics on the MEDIA corpus

e.g. their subparts extracted by STK or PTK, like the tree fragments in figures 1(a) and 1(b). These can be used in SVMs to learn the classification of words in concepts.

More specifically, in our approach, we use tree fragments to establish the order of correctness between two alternative annotations. Therefore, given two trees associated with two annotations, a re-ranker based on tree kernel, K_R, can be built in the same way of the sequence-based kernel by substituting SK in Eq. 3 with STK or PTK.

4 Experiments

In this section, we describe the corpora, parameters, models and results of our experiments of word chunking and concept classification. Our baseline relates to the error rate of systems based on only FST and SVMs. The re-ranking models are built on the FST output. Different ways of producing training data for the re-ranking models determine different results.

4.1 Corpora

We used two different speech corpora:

The corpus LUNA, produced in the homonymous European project is the first Italian corpus of spontaneous speech on spoken dialog: it is based on the help-desk conversation in the domain of software/hardware repairing (Raymond et al., 2007). The data are organized in transcriptions and annotations of speech based on a new multilevel protocol. Data acquisition is still in progress. Currently, 250 dialogs acquired with a WOZ approach and 180 Human-Human (HH) dialogs are available. Statistics on LUNA corpus are reported in Table 1.

The corpus MEDIA was collected within the French project MEDIA-EVALDA (Bonneau-Maynard et al., 2005) for development and evaluation of spoken understanding models and linguistic studies. The corpus is composed of 1257 dialogs, from 250 different speakers, acquired with a Wizard of Oz (WOZ) approach in the context of hotel room reservations and tourist information. Statistics on transcribed and conceptually annotated data are reported in Table 2.

4.2 Experimental setup

We defined two different training sets in the LUNA corpus: one using only the WOZ training dialogs and one merging them with the HH dialogs. Given the small size of LUNA corpus, we did not carried out parameterization on a development set but we used default or a priori parameters.

We experimented with LUNA WOZ and six rerankers obtained with the combination of SVMs and perceptron (PCT) with three different types of kernels: Syntactic Tree Kernel (STK), Partial Tree kernels (PTK) and the String Kernel (SK) described in Section 3.3.

Given the high number and the cost of these experiments, we ran only one model, *i.e.* the one

Corpus / Approach (STK)	LUNA WOZ+HH		MEDIA
	MT	ST	MT
FST	18.2	18.2	12.6
SVM	23.4	23.4	13.7
RR-A	**15.6**	17.0	**11.6**
RR-B	16.2	16.5	11.8
RR-C	16.1	16.4	11.7

Table 3: Results of experiments (CER) using FST and SVMs with the Sytntactic Tree Kernel (STK) on two different corpora: LUNA WOZ + HH, and MEDIA.

based on SVMs and STK[3] , on the largest datasets, *i.e.* WOZ merged with HH dialogs and Media. We trained all the SCLMs used in our experiments with the SRILM toolkit (Stolcke, 2002) and we used an interpolated model for probability estimation with the Kneser-Ney discount (Chen and Goodman, 1998). We then converted the model in an FST as described in Section 2.1.

The model used to obtain the SVM baseline for concept classification was trained using YamCHA (Kudo and Matsumoto, 2001). For the re-ranking models based on structure kernels, SVMs or perceptron, we used the SVM-Light-TK toolkit (available at dit.unitn.it/moschitti). For λ (see Section 3.2), cost-factor and trade-off parameters, we used, 0.4, 1 and 1, respectively.

4.3 Training approaches

The FST model generates the *m*-best annotations, *i.e.* the data used to train the re-ranker based on SVMs and perceptron. Different training approaches can be carried out based on the use of the corpus and the method to generate the *m*-best. We apply two different methods for training: **Monolithic Training** and **Split Training**.

In the former, FSTs are learned with the whole training set. The *m*-best hypotheses generated by such models are then used to train the re-ranker classifier. In Split Training, the training data are divided in two parts to avoid bias in the FST generation step. More in detail, we train FSTs on part 1 and generate the *m*-best hypotheses using part 2. Then, we re-apply these procedures inverting part 1 with part 2. Finally, we train the re-ranker on the merged *m*-best data. At the classification time, we generate the *m*-best of the test set using the FST trained on all training data.

[3]The number of parameters, models and training approaches make the exhaustive experimentation expensive in terms of processing time, which approximately requires 2 or 3 months.

WOZ	Monolithic Training					
	SVM			PCT		
	STK	PTK	SK	STK	PTK	SK
RR-A	**18.5**	19.3	19.1	24.2	28.3	23.3
RR-B	**18.5**	19.3	19.0	29.4	23.7	20.3
RR-C	**18.5**	19.3	19.1	31.5	30.0	20.2

Table 4: Results of experiments, in terms of Concept Error Rate (CER), on the LUNA WOZ corpus using Monolithic Training approach. The baseline with FST and SVMs used separately are **23.2%** and **26.7%** respectively.

WOZ	Split Training					
	SVM			PCT		
	STK	PTK	SK	STK	PTK	SK
RR-A	20.0	18.0	**16.1**	28.4	29.8	27.8
RR-B	19.0	19.0	19.0	26.3	30.0	25.6
RR-C	19.0	18.4	16.6	27.1	26.2	30.3

Table 5: Results of experiments, in terms of Concept Error Rate (CER), on the LUNA WOZ corpus using Split Training approach. The baseline with FST and SVMs used separately are **23.2%** and **26.7%** respectively.

Regarding the generation of the training instances $\langle s^i, s^j \rangle$, we set *m* to 10 and we choose one of the 10-best hypotheses as the second element of the pair, s_j, thus generating 10 different pairs.

The first element instead can be selected according to three different approaches:

(A): s_i is the manual annotation taken from the corpus;

(B) s_i is the most accurate annotation, in terms of the edit distance from the manual annotation, among the 10-best hypotheses of the FST model;

(C) as above but s_i is selected among the 100-best hypotheses. The pairs are also inverted to generate negative examples.

4.4 Re-ranking results

All the results of our experiments, expressed in terms of concept error rate (CER), are reported in Table 3, 4 and 5.

In Table 3, the corpora, *i.e.* LUNA (WOZ+HH) and Media, and the training approaches, *i.e.* Monolithic Training (MT) and Split Training (ST), are reported in the first and second row. Column 1 shows the concept classification model used, *i.e.* the baselines FST and SVMs, and the re-ranking models (RR) applied to FST. A, B and C refer to the three approaches for generating training instances described above. As already mentioned for these large datasets, SVMs only use STK.

We note that our re-rankers relevantly improve our baselines, *i.e.* the FST and SVM concept classifiers on both corpora. For example, SVM re-ranker using STK, MT and RR-A improves FST concept classifier of 23.2-15.6 = 7.6 points.

Moreover, the monolithic training seems the most appropriate to train the re-rankers whereas approach A is the best in producing training instances for the re-rankers. This is not surprising since method A considers the manual annotation as a referent gold standard and it always allows comparing candidate annotations with the perfect one.

Tables 4 and 5 have a similar structure of Table 3 but they only show experiments on LUNA WOZ corpus with respect to the monolithic and split training approach, respectively. In these tables, we also report the result for SVMs and perceptron (PCT) using STK, PTK and SK. We note that:

First, the small size of WOZ training set (only 1,019 turns) impacts on the accuracy of the systems, *e.g.* FST and SVMs, which achieved a CER of 18.2% and 23.4%, respectively, using also HH dialogs, with only the WOZ data, they obtain 23.2% and 26.7%, respectively.

Second, the perceptron algorithm appears to be ineffective for re-ranking. This is mainly due to the reduced size of the WOZ data, which clearly prevents an on line algorithm like PCT to adequately refine its model by observing many examples[4].

Third, the kernels which produce higher number of substructures, *i.e.* PTK and SK, improves the kernel less rich in terms of features, *i.e.* STK. For example, using split training and approach A, STK is improved by 20.0-16.1=3.9. This is an interesting result since it shows that (a) richer structures do produce better ranking models and (b) kernel methods give a remarkable help in feature design.

Next, although the training data is small, the re-rankers based on kernels appear to be very effective. This may also alleviate the burden of annotating a lot of data.

Finally, the experiments of MEDIA show a not so high improvement using re-rankers. This is due to: (a) the baseline, *i.e.* the FST model is very accurate since MEDIA is a large corpus thus the re-ranker can only *"correct"* small number of errors; and (b) we could only experiment with the less expensive but also less accurate models, *i.e.* monolithic training and STK.

Media also offers the possibility to compare with the state-of-the-art, which our re-rankers seem to improve. However, we need to consider that many Media corpus versions exist and this makes such comparisons not completely reliable. Future work on the paper research line appears to be very interesting: the assessment of our best models on Media and WOZ+HH as well as other corpora is required. More importantly, the structures that we have proposed for re-ranking are just two of the many possibilities to encode both word/concept statistical distributions and linguistic knowledge encoded in syntactic/semantic parse trees.

5 Conclusions

In this paper, we propose discriminative re-ranking of concept annotation to capitalize from the benefits of generative and discriminative approaches. Our generative approach is the state-of-the-art in concept classification since we used the same FST model used in (Raymond and Riccardi, 2007). We could improve it by 1% point in MEDIA and 7.6 points (until 30% of relative improvement) on LUNA, where the more limited availability of annotated data leaves a larger room for improvement.

It should be noted that to design the re-ranking model, we only used two different structures, *i.e.* one sequence and one tree. Kernel methods show that combinations of feature vectors, sequence kernels and other structural kernels, *e.g.* on shallow or deep syntactic parse trees, appear to be a promising research line (Moschitti, 2008). Also, the approach used in (Zanzotto and Moschitti, 2006) to define cross pair relations may be exploited to carry out a more effective pair re-ranking. Finally, the experimentation with automatic speech transcriptions is interesting to test the robustness of our models to transcription errors.

Acknowledgments

This work has been partially supported by the European Commission - LUNA project, contract n. 33549.

[4]We use only one iteration of the algorithm.

References

H. Bonneau-Maynard, S. Rosset, C. Ayache, A. Kuhn, and D. Mostefa. 2005. Semantic annotation of the french media dialog corpus. In *Proceedings of Interspeech2005*, Lisbon, Portugal.

S. F. Chen and J. Goodman. 1998. An empirical study of smoothing techniques for language modeling. In *Technical Report of Computer Science Group*, Harvard, USA.

M. Collins and N. Duffy. 2002. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete structures, and the voted perceptron. In *ACL02*, pages 263–270.

Y. He and S. Young. 2005. Semantic processing using the hidden vector state model. *Computer Speech and Language*, 19:85–106.

T. Kudo and Y. Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL2001*, Pittsburg, USA.

A. Moschitti and C. Bejan. 2004. A semantic kernel for predicate argument classification. In *CoNLL-2004*, Boston, MA, USA.

A. Moschitti, D. Pighin, and R. Basili. 2006. Semantic role labeling via tree kernel joint inference. In *Proceedings of CoNLL-X*, New York City.

A. Moschitti, G. Riccardi, and C. Raymond. 2007. Spoken language understanding with kernels for syntactic/semantic structures. In *Proceedings of ASRU2007*, Kyoto, Japan.

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

A. Moschitti. 2006. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of ECML 2006*, pages 318–329, Berlin, Germany.

A. Moschitti. 2008. Kernel methods, syntax and semantics for relational text categorization. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 253–262, New York, NY, USA. ACM.

C. Raymond and G. Riccardi. 2007. Generative and discriminative algorithms for spoken language understanding. In *Proceedings of Interspeech2007*, Antwerp,Belgium.

C. Raymond, G. Riccardi, K. J. Rodrigez, and J. Wisniewska. 2007. The luna corpus: an annotation scheme for a multi-domain multi-lingual dialogue corpus. In *Proceedings of Decalog2007*, Trento, Italy.

J. Shawe-Taylor and N. Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

A. Stolcke. 2002. Srilm: an extensible language modeling toolkit. In *Proceedings of SLP2002*, Denver, USA.

V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

F. M. Zanzotto and A. Moschitti. 2006. Automatic learning of textual entailments with cross-pair similarities. In *Proceedings of the 21st Coling and 44th ACL*, pages 401–408, Sydney, Australia, July.