PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Automated Approaches to Community Question Answering

Antonio Uva

Advisor:

Prof. Alessandro Moschitti

Università degli Studi di Trento

April 2019

Abstract

Social Media applications, e.g., forums, social networks, allow users to pose questions about a given topic to a community of expert users. Although successful, these applications suffer from a major drawback: it is rather complex to find similar questions with traditional keyword-based search. Thus, Community Question Answering (cQA), a branch of QA, has been developed with the aim of automatically answering new user questions. Generally, cQA systems answer new user questions by (i) first looking at the questions most similar to the input question and (ii) selecting the best answer for the related question. Such systems require powerful machine learning algorithms that go beyond traditional approaches based on features. In recent years, tree kernels and neural networks have established as the state-of-the-art machine learning algorithms for solving such kinds of problems. Tree kernels are used to compute the similarity between two sentences encoded in form of trees that incorporate syntactic and semantic information. Neural networks map words into informative vectors called embeddings used to learn non-linear transformations of user inputs. In this work, we used these models for solving classification and ranking tasks needed to build automatic cQA systems. As a first step, we conceived structured input models able to automatically extract discriminative syntactic patterns for classifying relatedness between two questions. Then, we extended the previous work by presenting a new model for question similarity that combines semantic information of neural networks with structured information of tree kernels. We assess the performance of the new model on two tasks, i.e. question duplicate detection and question reranking, showing the advantages of injecting syntactic information in neural models. After that, we focus on more challenging tasks such as building a neural network architecture for ranking comments on a forum according to their relevance with respect to a new question. We show that neural models can benefit from being trained in multi-task learning setting, together with auxiliary tasks. This make possible to train cQA systems in an end-to-end fashion, which is convenient for industrial applications that needs to be easily deployed. Furthermore, we developed a novel intent detection model that combines state-of-the-art methods in relational text matching with the latest techniques in supervised clustering to make inference over a set of questions and automatically discover intent clusters. The latter can be used to quickly bootstrap Natural Language Understanding pipelines for dialog systems. To conclude, we study advantages and disadvantages of neural networks and tree kernel models when applied to cQA tasks. We show that neural networks perform effectively when data is abundant. Conversely, *tree kernels are more suitable in presence of data scarcity.*

Keywords Community Question Answering, Tree Kernels, Neural Networks

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this section, we present the motivations behind this work, and describe our contributions, chapter by chapter, including references to the corresponding publications.

1.1 Motivations

In recent years, there has been an exponential increase in social media applications such as social networks, photo-sharing and instant-messaging apps. These applications have quickly encountered the favor of the people, as they allow users to share media such as photos, videos and text with other people based on their common interests. One particular type of social media applications is represented by web forums and online community QA services, which allow users to pose questions about a given topic to a community of experts and users. Some very famous examples of such applications are Quora, Yahoo! Answers, Stack Overflow and Ask Ubuntu. Interestingly, questions and answers generated by users are not longer bound exclusively to cQA websites; nowadays we can find question threads even on places not originally conceived as QA websites. For example, modern e-commerce websites host questions (and answers) asked by users about products sold on their platform. It is no wonder that at certain point the NLP community started to probe the use of user-generated content to train systems able to automatically provide answers for questions asked by users on cQA websites. For many years, research on automatic QA has mainly focused on factoid questions, i.e. questions whose answers is a name. Unfortunately, the largest part of questions asked by users on social media are not factoid. They range from (i) polar yes/no questions to procedural how-questions to explanation why-questions. In order to be answered, this kind of questions require methodologies that should reflect the specificity of their intended domain of application. To make this more complicated, automatic cQA suffers from the problem that the text input by users is not phrased correctly and generally too complex to be characterized by a finite set of rules that specify how to match two pieces of text, such a question and a relevant answer comment on a forum. The number and complexity of rules needed in order to make accurate decisions tend to be very high and this may lead to the development of systems that are difficult to develop and maintain. So, while previous research on factoid QA show that systems based on heuristics can be very effective, such approaches are not feasible for cQA. Moreover, recent years have seen a growing interest in voice-controlled devices, such as Amazon Alexa, Microsoft Cortana, Apple Siri and Google Home. The success of such systems in the near future will very likely depend or their ability to quickly bootstrap Natural Language Understanding (NLU) components for answering new types of questions. Unfortunately, the need to quickly prototype NLU systems clashes with the high cost involved in engineering them. In order to extensively find a solution to this problem, many evaluation campaigns have been organized recently both from academia, e.g. SemEval [Nakov et al., 2016a], and companies, e.g. Quora[1] and Alibaba[2], with the goal of building automatic systems for cQA. Although manual approaches to this problems are doomed to fail, a viable solution is offered by the use of modern NLP methods, which we present and discuss in this work. Such approaches are aimed at reducing the engineering cost required for building automatic cQA systems, which can also potentially benefit Conversational Agents. In particular, we experimented with two state-of-the-art techniques for automatically engineering features: deep neural networks and structural kernels. The first approach encode pieces of text by generating informative embeddings starting from words. The latter approach represents text elements according to the text inner syntactic structure. Furthermore, we explore novel ways for combining deep neural networks (DNNs) with kernel technology in order to improve algorithms for relational text inference. In the final chapter, we show how we used these models that take advantage of automatic feature engineering capabilities to build a prototype of a system for automatically managing an Help Desk service. The systems was trained exclusively on data that have been generated by users.

1.2 Contributions and Structure of the Thesis

In Chapter 2 we introduce the reader to the different machine learning algorithms used in this thesis. First, we describe Support Vector Machines and kernel methods useful for training discriminative large-margin classifiers. Secondly, we provide a brief overview of different types of Neural Network architectures used for modeling the information in a sentence. Both methods are the basis of many of our contributions described in the

[1]`https://www.kaggle.com/c/quora-question-pairs`

[2]`https://102.alibaba.com/detail/?id=115&mtime=1528166091000`

following chapters.

Contribution 1 (Chapter 3): *Structural Models for automatic community QA.*

We first formalize the tasks related to the problem of building automatic systems for community QA (cQA). Then, we describe an SVM model that use structural representations and syntactic Tree Kernels (TKs) for solving suck tasks [Barrón-Cedeño et al., 2016]. Experiments carried out in the context of SemEval-2016 challenge show that our models deliver state-of-the-art performance on automatic cQA. Furthermore, we carry out extensive experiments to assess the ability of our structural model to improve the result obtained by advanced systems such as Google in reranking tasks, i.e., question-question and question-answer similarity [Da San Martino et al., 2016]. We show that our approach is robust in presence of noisy data and when combined with Google, provides new state-of-the-art results.

Contribution 2 (Chapter 4): *Combining Neural Networks and Kernel models for computing question-question similarity.*

We study the problem of building Neural Networks that model also syntactic information when measuring question-question similarity. To do so, we studied interaction between Tree Kernels and Neural Networks and propose a new approach to inject structural information in NNs Uva et al. [2018]. Briefly, the approach works by training a Neural Networks on a large corpus of unlabeled data, whose annotations have been provided by an SVM classifier operating on structural representations, and then fine-tuning on gold annotated data. We show that our approach consistently improves results on two datasets for question-question similarity: Quora and Qatar Living (QL).

Contribution 3 (Chapter 4): *Joint Model for solving the overall cQA task.*

We focus on the problem of learning a deep NN for solving the following task: predict if a comment submitted in response to a previous forum question contains a valid answer for a new out-of-forum question. Unfortunately, since we had very little training data, our preliminary results were very low. In order to solve this issue, we train a network with shared-weights in multi-task learning (MTL) setting on two auxiliary tasks, i.e. (i) question-to-question similarity and (ii) ranking answers with respect to related forum question. By doing so, we could exploit connections between inputs of related tasks, which allowed us to improve the final performance of our model.

Contribution 4 (Chapter 5): *Supervised clustering of questions for quick bootstrapping of Intent Ontologies.*

We study the problem of automatically clustering questions that correspond to the same user intent, in order to quickly bootstrap NLU pipelines Haponchyk et al. [2018]. Although very challenging, we provided a solution that combined (i) powerful semantic classifiers with (ii) novel structured output algorithms for supervised clustering. The results showed that our solution can achieve very good accuracy on intent clustering corpora and can be used for alleviating dialog manager engineering from the burden of manually annotating intents for new tasks and domains.

Contribution 5 (Chapter 6): *NLP Pipelines and demos.*

In this final chapter, we first present the multi-lingual UIMA-based NLP pipeline developed in the context of the European project Limosine [Uryupina et al., 2016]. Then, we describe our effort for training a high-performing constituency parser for the Italian language and reducing the accuracy gap with respect to the English language [Uva and Moschitti, 2016]. Training such parser allowed us to build the reliable structural representation of utterances required for solving higher-level semantic tasks in the Italian language. Finally, we used this representation to build two QA systems: (i) a more academic system for factoid QA that use the Italian Wikipedia corpus to search for answers [Uva and Moschitti, 2015] and (ii) a commercial cQA system that automatically addresses questions asked by users to operators of some Help Desk service [Uva et al., 2017].

Chapter 2

Machine Learning Methods

2.1 Support Vector Machines and Kernel Methods for relation text inference

In this section, with first define the concepts of supervised learning. Then, we introduce Support Vector Machines (SVMs), a class of machine learning algorithms that give state-of-the-art performances in many discriminative tasks. We provide important insights on the motivations behind the use of SVMs and the advantages of large-margin classification hyperplanes. In the end of the section, we describe the dual formulation of SVM and its most important development: the use of structural kernels in the so called kernel machines.

2.1.1 Supervised Learning

In Supervised learning, we are interested in learning a function that maps an input vector to its corresponding target vector. More in detail, given a training set $\mathcal{D} = \{(\mathbf{x_i}, \mathbf{y_i})\}_{i=1}^{n}$, we wish to learn a function $h \in \mathcal{H}$ from the space of possible functions $\mathcal{H}$. The function $h : X \to Y$ maps a input x from the input space X to an output y in the output space Y. The input can be a vector $x_i \in \mathbb{R}^d$ of dimension d, or a structured object. Te output y can be anything, but typically is one categorical variable y_i from a finite number of discrete categories $\{1, \ldots, C\}$. If the output consists of one or more continuous variables, then this yield to a regression problem. At test time, the function h returns the value y that gives the highest score, as follows:

$$h(x) = \arg\max_{y \,\in\, Y} f_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$$

where $f : X \times Y \to \mathbb{R}$ is a discriminant function that takes input a problem instance $\mathbf{x}$ together with a class y and output a numerical score. Typically, the function f is parameterized by a vector $\mathbf{w}$, learned on the train set, and is linear in the weight vector, like this:

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{\Psi}(\mathbf{x}, \mathbf{y}) \rangle \tag{2.1}$$

where $\langle \cdot, \cdot \rangle$ is a dot product and the function $\Psi : X \times Y \rightarrow \mathbb{R}^d$ maps each input example and its class into a feature vector in $\mathbb{R}^d$. Typically, we are interested in finding hypothesis that returned the expected answer in the majority of cases. One way to find a good hypothesis h^* among a fixed class of function from the hypothesis spaces $\mathcal{H}$, is to choose one for which the risk $R(h)$ is minimal.

2.1.2 Empirical risk minimization

Computing the risk requires the definition of a loss or discrepancy $\mathcal{L}(y, f(\mathbf{x}_i))$ between the response of f determined by the model parameters $\mathbf{w}$ and the actual label y_i [Vapnik, 1992]. Once we fixed a task-dependent loss $\mathcal{L}$, we can estimate parameters $\mathbf{w}$ of a model from training data by adopting the Empirical Risk Minimization principle. The latter states that the learning algorithm should choose a hypothesis h^* which minimizes the empirical risk $R(f)$ according to the defined task-dependent loss. However, since the functional risk cannot be directly optimized, we minimize the empirical risk $R_{emp}(f)$ evaluated on the training data.

$$h^* = \arg\min_{h \in \mathcal{H}} R_{emp}(f) = \operatorname{arc}\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(h(\mathbf{x_i}), \mathbf{y_i}) \tag{2.2}$$

2.1.3 Loss function

A loss function $\mathcal{L} : Y \times Y \rightarrow \mathbb{R}$ maps an event, generally defined over two variables, into a numerical cost associated with that specific event. The latter is typically the prediction of a model, and the loss specifies how much it should be penalized for incorrect predictions. Generally, a common loss used for training "maximum-margin" classifiers, such as SVMs, is the hinge loss. This loss is used in place of the $0 - 1$ loss, which is not convex and is not derivable at 0. The general formulation of the hinge loss is the following:

$$L_{hinge}(y^*, y) = \max\left(0, \max_{y \neq y^*}(\Delta(y, y^*) + \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle) - \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}^*) \rangle\right)$$

The term $\Delta(\cdot, \cdot)$ measures the discrepancy between the true label y^* and the predicted output y. It enforces the requirement that y^* should be scored higher than any other predicted value y by at least $\Delta(y, y^*)$.

2.1.4 Training discriminative models

After having defined the previous concepts, we can summarize the main components needed for training discriminative models. In particular, we have:

1. Training and test data from the input space $\mathbb{R}^d$, represented as vector of features x^d obtained by applying the feature map $\Psi(\cdot,\cdot)$. The design of the feature map Ψ is very important since the the expressiveness of the feature set critically impacts on the accuracy of the final model.

2. The output space, which depends on the task. It may be a binary variable, i.e. -1 or $+1$, for binary classification, or in case of multiclass classification, the confidence of one class over k classes.

3. A space of hypothesis, among which to select the function mapping an input example to output target.

4. A loss function $\mathcal{L}$ penalizing incorrect predictions. The loss needs to be carefully chosen depending on the task to solve.

5. An algorithm that estimates the parameters $\mathbf{w}$ of the model by minimizing the empirical risk.

6. An inference process that assigns an output label to an input.

2.1.5 Maximum Margin Classifiers

Support Vector Machines for two-class classification problems learn models of the form:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathrm{b} \tag{2.3}$$

where $\mathbf{w}$ is a weight vector in $\mathbb{R}^d$ and b represents the bias term. Typically, in order to train a SVM classifier we need training data. A training dataset comprises N input vectors $\mathbf{x}_1, \cdots, \mathbf{x}_N$, with corresponding target values, $y_1, \cdots, y_N$, where $y_n \in \{-1, 1\}$, and new data points $\mathbf{x}$ are classified according to the sign of $f(\mathbf{x})$. If the training data are linearly separable in the feature space, than there exists an assignment to the model parameters $\mathbf{w}$ and b such that the function 2.3 satisfies $\mathbf{w}^T\mathbf{x_n} + \mathrm{b} > 0$ for points having $y_n = +1$ and $\mathbf{w}^T\mathbf{x_n} + \mathrm{b} < 0$ for points having $y_n = -1$, so that $y_n(\mathbf{w}^T\mathbf{x_n} + \mathrm{b}) > 0$ for all training data points.

2.1.6 Hard margin SVM

When training data are linearly separable, we can train a hard margin SVM to select the hyperplane $\mathbf{w}$ that maximizes the separating margin between two classes. The confidence margin ρ of a classifier is defined as the minimal distance between the classifier hyperplane and the nearest examples from two classes.

$$\rho = \min_{(\mathbf{X},y)\in\mathcal{D}} y f(\mathbf{X})$$

However, there are infinite possible solution formulations for the same hyperplane, e.g.:

$$\begin{aligned} \mathbf{w}^T\mathbf{x} + \mathrm{b} &= 0 \\ \alpha(\mathbf{w}^T\mathbf{x} + \mathrm{b}) &= 0 \quad \forall\alpha \neq 0 \end{aligned}$$

Given this, we can choose the decision hyperplane that has confidence margin equal to 1, i.e. $\langle\mathbf{w},\mathbf{x}\rangle = 1$ for the closest points on the positive side (i.e. side of positive examples), and $\langle\mathbf{w},\mathbf{x}\rangle = -1$ for the closest points on the other side. Such hyperplane is called *canonical hyperplane*. As a result, the size of the margin band is equal to two times the canonical hyperplane *geometric margin* γ. Given that the hyperplane geometric margin γ is equal to $\frac{\rho}{\|\mathbf{w}\|}$ and the canonical hyperplane has confidence margin $\rho = 1$, it follows that $\gamma = \frac{1}{\|\mathbf{w}\|}$. Thus, maximizing the geometric margin γ corresponds to maximizing $\|\mathbf{w}\|^{-1}$, which is the same as minimizing $\|\mathbf{w}\|^2$. Finally, selecting the best separating hyperplane for hard margin SVMs corresponds to solving the following problem:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to:} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1 \end{aligned}$$

As it can bee seen, this is a *quadratic programming* problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints. The points lying on the minimal confidence (canonical) hyperplane are called *support vectors*. All the other points that do not contribute to the final decision function could have been removed from the training set, without affecting the classifier training. The resulting model is *sparse*, in the sense that classification of new points, i.e. problem instances, requires only few computations with respect to the model stored support vectors.

Soft margin SVM

In cases where the training data points are not linearly separable in feature space, e.g. due to overlapping class distributions or annotation errors, a more relaxed version of SVM can be used. An intuitive way of modifying SVM to handle such cases is to allow some of training points to be misclassified [Cortes and Vapnik, 1995; Bennett and Mangasarian, 1992]. This way, data points will be allowed to be on the wrong side of the margin boundary. So, we define a more relaxed version of SVM by introducing *slack variables*, $\xi_n \geqslant 0$, where there is one slack variable for each training data point. If $\xi_n = 0$, then the points are correctly classified and either they are on the margin, i.e. $y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) = 1$ or they are on the correct side of the margin $y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1$. Points for which $0 < \xi \leq 1$ line inside the margin (margin error), but on the correct side of the decision boundary, while points for which $\xi_n > 1$ lie on the wrong side of the decision boundary and are misclassified. The regularization parameter $C > 0$ controls the trade-off between slack variable penalty and margin, or in other words, controls the trade-off between minimizing the training errors and controlling model complexity. If the C parameters is large, the optimization will choose smaller-margin hyperplane that classify all training points correctly. However, this may result in poor generalization. Conversely, a small value of C causes the optimizer to look for separating hyperplanes with a larger margin. In such case, there is a reasonable cost to pay for training points falling inside the margin band, or the wrong side of hyperplane. Typically, this leads to better generalization and more robustness.

$$\begin{aligned} &\underset{\mathbf{w},b,\xi}{\text{minimize}} && \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ &\text{subject to:} && y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1 - \xi_i \qquad \xi_i \geq 0 \end{aligned} \tag{2.4}$$

2.1.7 Dual problem

The primal optimization problem of SVM can be solved with a number of methods for quadratic optimization problems, e.g. interior point methods, ellipsoids, simplex and many others. However, solving the primal is not convenient if the dimension d of training examples is larger than their number N, i.e. $d >> N$. One more efficient way to find the model parameters is by solving the Lagrangian dual of the quadratic optimization problem, enabled by the use of Representer Theorem [Kimeldorf and Wahba, 1970]. The latter says that a function f minimizing a regularized empirical risk function over a kernel Hilbert space can be represented as a combination of kernel products evaluated on the input points in the training dataset, i.e.:

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_i$$

If we substitute $\mathbf{w}$ in eq 2.3, we obtain:

$$f_\alpha(\mathbf{x}) = \left(\sum_{n=1} \alpha_n y_n \mathbf{x}_n\right)^T \mathbf{x} = \sum_{n=1} \alpha_n y_n \mathbf{x}_n^T \mathbf{x}$$

Here, we omit the derivation of the dual formulation with respect to the primal variables, and directly state the dual optimization problem:

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad & \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq C \end{aligned} \tag{2.5}$$

The dual formulation in equation 2.5 has several advantages in comparison to the primal form:

- in cases where $N << d$, it is more efficient to solve for the dual variables α_n than for the primal variables $\mathbf{w}$. By solving the dual optimization problem, we learn more compact models with sparser alphas, i.e. $\alpha_i \geq 0$. This way, the resulting SVM model will contain a number of support vectors smaller than the number of training instances.
- the dual formulation contains dot products only between input instances, which can be replaced with kernels.

2.1.8 Kernel trick (Kernel substitution)

Kernels make it possible to replace inner products between between input vectors $\mathbf{x}$ into inner products computer in an other *implicit* features spaces. There are two ways to apply kernels:

- Choosing a feature space mapping function $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ and then computing the inner product in the transformed space; or
- Constructing the kernel function directly by ensuring that the function we choose is a valid kernel, i.e. that the value $K(\mathbf{x}_i, \mathbf{x}_j)$ corresponds to scalar product in some feature space. Interestingly, in this case, a Kernel function between two input objects can be implicitly calculated, without requiring the computation of the coordinates of the data in the new *implicit* space. This is called "kernel trick" and is a result of the Mercer's Theorem [Shawe-Taylor et al., 2004]:

$$K(\mathrm{x}_i, \mathrm{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \tag{2.6}$$

While appealing, the second approach requires checking that a function is a valid kernel, which can be done, e.g., by explicitly constructing $\phi(\mathbf{x})$. Unfortunately, this is a very tedious process. As a solution, [Shawe-Taylor et al., 2004] show that a sufficient condition for a function $K(\mathbf{x}, \mathbf{x}')$ to be a valid kernel is that the Gram matrix K, whose elements are given by $k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semidefinite for all possible choices of the set $\mathbf{x}_n$, i.e.:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j) c_i c_j \geq 0, \ \forall c$$

At this point, the final SVM classifier can be defined as follows:

$$f_\alpha(\mathbf{x}) = \sum_i \alpha y_i K(\mathbf{x}_i, \mathbf{x})$$

Selecting the right kernel for a given problem requires some expertise. Some popular kernels on real-valued fixed-size vectors are:

- The Linear Kernel: $K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle$
- The Polynomial Kernel of degree d: $K(\mathbf{x}, \mathbf{y}) = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^d$, for any $d > 0$
- The Sigmoid Kernel: $K(\mathbf{x}, \mathbf{y}) = \tanh(a \langle \mathbf{x}, \mathbf{y} \rangle)$
- The Gaussian RBF: $K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2 / 2\sigma^2)$, for $\sigma > 0$

In the next section, we introduce some complex kernels operating on structured objects.

2.2 Structural Kernels

Structural kernels are kernels that operate on structured objects, e.g. strings, trees, graphs and so on. They have been successfully applied in many domains and problems, since they allow the user to incorporate knowledge about the structure of the data when computing kernel similarities. This is important especially for objects whose overall similarity is a function of the similarity of their subparts. In this section, we introduce general kernels that operate on structured objects such as strings and trees.

2.2.1 String Kernel

The String Kernel (SK) Lodhi et al. [2002] computes the similarity between two strings s_1 and s_2 by counting the number of common substrings that are shared between them. Some symbols in the strings may be skipped. This allows the skipgrams to contribute to the final similarity. The SK is defined by the following equation:

$$K_{SK}(s_1, s_2) = \sum_{u \in \Sigma^*} \phi_u(s_1) \cdot \phi_u(s_2) = \sum_{u \in \Sigma^*} \sum_{\vec{I_1}:u=s_1[\vec{I_1}]} \sum_{\vec{I_2}:u=s_2[\vec{I_2}]} \lambda^{d(\vec{I_1})+d(\vec{I_2})} \tag{2.7}$$

Here, $\Sigma^* = \cup_{n=0}^{\infty}\Sigma^n$ is the set of all possible strings, while $\vec{I_1}$ and $\vec{I_2}$ are the two sequences of indexes $\vec{I} = (i_1, \cdots i_{|u|})$, with $1 \leq i_1 < .. < i_{|u|} \leq |s|$, such that $u = s_{i_1}..s_{i_{|u|}}$, $d(\vec{I}) = i_{|u|} - i_1 + 1$ and $\lambda \in [0, 1]$ is a decay factor. The i indexes range from 1 to the length of substrings u, and u is shorter than the string length. $d(\vec{I})$ is the distance between the first and last character of the substring.

2.2.2 Convolution Tree Kernels

Tree kernels compute a similarity between tree structures by counting the number of common subtrees rooted at different nodes. There are many kinds of tree kernels and their difference is in richness of tree fragments generated. The main advantage of Tree Kernels (TKs) is that they compute the number of subtrees between two trees T_1 and T_2 without enumerating all the possible tree fragments, which would be very expansive operation. Let $\mathcal{T} = \{t_1, \cdots, t_{|\mathcal{T}|}\}$ be the set of all possible trees in the space of structures, and $\chi_i(n)$ and indicator function, which is equal to 1 if the target t_i is rooted at node n, and equal to 0 otherwise. We can defined a general tree kernel, over T_1 and T_2 as:

$$K_{TK}(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \tag{2.8}$$

N_{T_1} and N_{T_2} are a set of nodes of the T_1 and T_2 trees, and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{T}|} \chi_i(n_1)\chi_i(n_2) \tag{2.9}$$

computes the number of common tree fragments rooted at the n_1 and n_2 nodes. Depending on how the tree fragments are extracted and counted, Eq. 2.2.2 generates a number of tree kernels with different level of expressivity.

Syntactic Tree Kernel

The Syntactic Tree Kernel (STK) Collins and Duffy [2002] evaluates the number of common subtrees as follows:

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 (they are pre-terminals), 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 preterminals, then:
$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(c^j_{n_1}, c^j_{n_2}))$$
where $nc(\cdot)$ is a function that returns the number of children of the argument node, and c^j_n is the j-th child of node n. A decay factor can be introduced by modifying the steps (2) and (3) as follows[1]:

2. $\Delta(n_1, n_2) = \lambda$

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

   The running time of STK is $O(|N_{T_1} \times N_{T_2}|)$, but as shown in [Moschitti and Zanzotto, 2007] tends to be linear, i.e. $O(|N_{T_1}| + |N_{T_2}|)$ if (i) first, we sort subtrees by the lexicographical order of their production and (ii) secondly, we count kernel similarity between those trees. The main observation of STK is that the production rules of the grammar used to generate the tree will not be broken, i.e. children of a node are not separated.

Partial Tree Kernel

The Partial Tree Kernel (PTK) differs from the STK in the fact that it consider also partial tree fragments, i.e. subtrees where children can be separated. This means that production rules of the grammar generating the trees can be broken. PTK produces a greater number of fragments, and thus the feature spaces is more expressive. The Δ function of the PTK is the following:

1. if the node labels n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$; else

2. $\Delta(n_1, n_2) = 1 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1) = l(\vec{I}_2)} \prod_{j=1}^{l(\vec{I}_1)} \Delta_\sigma(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))$

   where $\vec{I}_1 = \langle h_1, h_2, \cdots \rangle$ and $\vec{I}_2 = \langle k_1, k_2, k_3, .. \rangle$ are sequences of indices synchronized with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2. $\vec{I}_{ik}$ and $\vec{I}_{2j}$ index the j-th child in the sequence, and $l(\cdot)$ returns the length of the index list, and therefore the number of children of a node.

We can extend the previous formula by adding two decay factors: u accounting for the tree depth and λ for the length of the child subsequences with respect to the original sequence, which accounts also for gaps:

[1]score can be normalized between 0 and 1 in kernel space: $TK_{norm}(T_1, T_2) = \frac{(T_1, T_2)}{TK(T_1, T_1) TK(T_2, T_2)}$

$$\Delta(n_1, n_2) = \mu(\lambda^2 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1)=l(\vec{I}_2)} \lambda^{d(\vec{I}_1)+d(\vec{I}_2)}) \prod_{j=1}^{l(\vec{I}_1)} \Delta(c_{n_1}(\vec{I}_{ij}), c_{n_2}(\vec{I}_{2j})) \tag{2.10}$$

where $d(\vec{I}_1) = \vec{I}_{1l(\vec{I}_1)} - \vec{I}_{11} + 1$ and $d(\vec{I}_2) = \vec{I}_{2l(\vec{I}_2)} - \vec{I}_{21} + 1$. The decay factor penalizes larger and deeper trees, and children that are far away from each other.

2.3 Neural Networks for Sentence Modeling

In this section, we present neural networks, a family of powerful machine learning models. Such models have been applied in many domain and problems, ranging from lucrative online advertising applications to image classification in computer visions and to machine translation in natural language processing. In the last years, the application of neural networks to all these fields allowed them to make huge steps forwards. In the remaining of the chapter, we provide some background about the use of deep learning methods in the NLP field, where they are considered the state of the art for many tasks. Then, we described the main network architectures used in our contributions, from simple feed-forward networks to convolutional and recurrent neural networks. Whenever possible, we include the motivation behind the design of different network architectures, highlighting advantages and limitations for each of them.

2.3.1 Rise of Deep Learning in NLP

Neural networks started to be adopted in NLP only recently. Generally, text has never been an easy task for computers and they have struggled considerably more with unstructured data compared to structured data. One reason is that for many years, the availability of annotated data for common NLP tasks has been very limited. As a result, discriminative approaches based on simple linear classifiers, such as SVMs, have always outperformed very complex classification function with millions of parameters. However, recent developments on neural networks as well as the availability of large datasets, e.g., created with crowd-sourcing methods has allowed computers to significantly improve their capability to fit complex functions, with beneficial effects for speech recognition, computer vision and NLP. Another driving factor of such improvement is the availability, today, of fairly large amount of semi-supervised data compared to the last decade, allowed from fast digitization of society. The time spent by people, for example, on social media and online apps, accessible from smartphones, made possible for machines to accumulate more and more data. Unfortunately, traditional learning algorithms were not ready to exploit such data effectively. Thus, a new generation of machine learning methods, i.e, deep learning, rose in order to take advantage from this new situation. Finally, the second major factor behind the success of deep learning is the availability of specialized hardware such as GPUs. This hardware, which provided the computational power required for fast matrix operations at the base of many neural network operations, made it possible to train very large models that can effectively benefit from data abundance.

2.3.2 Types of Neural Networks

Over the last years, many types of neural networks have been conceived for solving different kinds of applications. Generally, standard feed-forward NNs have been used for online advertising and real estate applications. Convolutional Neural Networks (CNNs) have been used for image classification, while Recurrent Neural Networks (RNNs) have been employed for handling sequential data, such as audio or sentences. In the following sections we describe the main network architectures that the reader will find in our contributions, from simple feed-forward to LSTM recurrent neural networks.

2.3.3 Standard feed-forward Neural Network (NN)

Neural networks can be shallow or deep, depending on the number of hidden layers contained by the network. By convention, deep networks, such as Standard feed-forward, are composed of 1 input layer, 1 output layer and at least one hidden layer. One of the simplest forms of feed-forward networks is the Multi-Layer Perceptron (MLP) (Figure 2.1). In a MLP every unit in a layer is connected to every unit in the next layer, except for the output layer, which returns the final network prediction.

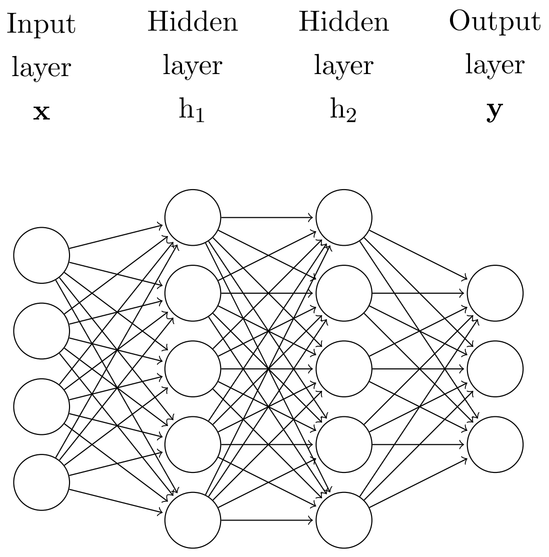


Figure 2.1: A Multilayer Perceptron (MLP) composed of 1 hidden layer, 2 hidden layers and 1 output layer

The input layer takes in input the $\mathbf{x}$ vector. The hidden layer values are obtained by applying a hidden non-linear transformation of the values from the previous layer. More in detail, the first hidden layer is computed by multiplying the input vector $\mathbf{x}$ with the weights of matrix corresponding to the $\mathbf{h}_1$ layer, and adding the bias to the result. Generally, a neural network model can be specified by a set of vector-matrix operations. Below, We report the equations specifying the MLP:

$$\begin{aligned}\hat{\mathbf{y}} &= f(\mathbf{x}) = o(h_2(h_1(\mathbf{x}))\\ h_1(\mathbf{x}) &= \sigma_1(\mathbf{W_1}\mathbf{x} + \mathbf{b_1})\\ h_2(\mathbf{x}) &= \sigma_2(\mathbf{W_2}\mathbf{x} + \mathbf{b_2})\\ o(\mathbf{x}) &= \sigma_3(\mathbf{W_3}\mathbf{x} + \mathbf{b_3})\end{aligned}$$

$$\begin{gathered}\mathbf{x} \in \mathbb{R}^{d_{in}}, \hat{\mathbf{y}} \in \mathbb{R}^{d_{out}}, f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}\\ \mathbf{W_1} \in \mathbb{R}^{d_1 \times d_{in}}, \mathbf{b_1} \in \mathbb{R}^{d_1},\\ \mathbf{W_2} \in \mathbb{R}^{d_2 \times d_1}, \mathbf{b_2} \in \mathbb{R}^{d_2},\\ \mathbf{W_3} \in \mathbb{R}^{d_{out} \times d_2}, \mathbf{b_3} \in \mathbb{R}^{out}\\ \sigma_i \in \{tanh, sigmoid, relu, ...\}, i \in \{1, 2, 3\}\end{gathered}$$

Here, d_{in} and d_{out} represent the input and output dimensions, while σ_i are non-linear activation functions applied to each element. In order to compute non-trivial functions a neural network needs to use non-linear activation functions. The reason is that composition of linear functions is still a linear function of the input, making de facto useless to have multiple hidden layers in a network. Regarding the activations σ_i, different non-linearities can be used across the hidden layers. As usual, the specific type of problem being solved guides the choice of details such as the number of outputs, the final output activation and the loss to minimize.

2.3.4 Activation function

Choosing the right activation function is very important when designing a neural network model. In table 2.1 we show some of the functions commonly used. The sigmoid function model the probability of a binary event output by a single neuron. This is the default choice when the desired output is between 0 ad 1. An other activation function is the hyperbolic tangent function (tanh), which returns an output between -1 and 1. The sigmoid and tanh functions are similar, but the latter is better, as it automatically centers the activation values to 0. For this reason, it often performs better then sigmoid when used in the hidden layers. Unfortunately, one problem of the sigmoid and tanh activations is that their gradient tend to vanish when the activation becomes very large or very small (gradient saturation). This results in slowing down gradient-based optimization algorithms at training time. To overcome this problem, the rectified linear (ReLU) units were proposed by Nair and Hinton [2010]. In ReLU the derivative is 1 as long as the activation is positive, otherwise is 0. Although the derivative is technically undefined when activation is equal to 0, this issue is solved by setting the derivative to 0. By using the ReLU, the network continues to learn even in case the activation values are very large or small. Nowadays, the ReLU unit is used in the hidden layers of the networks as default choice in the largest part of cases.

Logistic (or sigmoid)	$f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic Tangent (tanh)	$f(x) = \frac{2}{1+\epsilon^{-2x}} - 1$
Rectified Linear Unit (ReLU)	$f(x) = \max(0, x)$
Softmax	$f(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e_k^x}$ for $i = 1, \ldots, K$

Table 2.1: Common neuron activation functions

2.3.5 Training the network: Forward and Backward propagation

At classification time, the output of a neural network depends on the input and the parameters of the networks layers, i.e. the weight matrices and bias vector. These parameters, which are fixed during classification, first, need to be learned. During training, we wish to set the parameters in order to minimize the empirical risk on the training instances. The process by which the network learns the optimal parameter values happens in two steps: forward and backward passes. Typically, during the forward pass the network takes in input a training instance and produces an output under the current parameters. Then, the predicted output is compared with the true output, usually specified by the gold labels associated with the training instances in the training set. At this point, the difference between the true and the predicted value is computed and produces an error measure called loss. This measure is used during the backward pass for tuning the parameters of the network. The goal of the training process is to reduce the loss of the network on the next forward pass over the data to make the predictions of the network closer to the desired output.

2.3.6 Loss Functions

A loss function $\mathcal{L}$ is used to produce a single scalar value that measure the error made by the network on the predicted output y compared to the true output y^*. As usual, the loss has to be selected carefully based on the task to optimize, which depends on the desired output type. This can be a continuous value, a binary or a categorical variable. In the first case, where we want to model a continuous output, a suitable loss function can be the Root Mean Squared Error (RMSE).

$$\mathcal{L}_{RMSE}(y^*, y) = \sqrt{\frac{\sum_{i=1}^{n}(y_i^* - y_i)^2}{n}}$$

In the second case, where we model a binary variable, it is better to use the binary cross-entropy (also called logarithmic loss), which measures the performance of a classification model whose outputs are probabilities:

$$\mathcal{L}_{logloss}(y^*, y) = -\frac{1}{n}\sum_{i=1}^{n}[y_i^* \log y_i + (1 - y_i^*)\log(1 - y_i)]$$

A more general version of logarithmic loss is categorical cross-entropy. The latter is used when we need to classify an input into one of m possible classes, and m is greater than two. The cross-entropy measures how much two distributions diverge, i.e., the network predicted probability distribution vs. the ground truth probability distribution

$$\mathcal{L}_{cross-entropy}(y^*, y) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}^* \log(y_{ij})$$

2.3.7 Backpropagation

After having defined the network architecture and the loss, we need to train a model and update the network parameters to produce the desired output. To do so, we use the backpropagation algorithm [Rumelhart et al., 1988]. Backpropagation iteratively adjusts the parameters of the network in order to reduce the error quantified by the loss function. It works by computing the gradient of the loss function under the parameters of the network. Each term in the gradient quantifies the error of a single neuron during the computation of the network output. Once the gradient is computed, the weights of the networks are updated accordingly. For example, assuming we have a network composed of ℓ layers, i.e. $W^1, W^2, \ldots, W^\ell, b^1, b^2, \ldots, b^\ell$, we update them weights and biases as follows:

$$W^{|\ell|} = W^{|\ell|} - \alpha\frac{\partial\mathcal{L}}{\partial W^{|\ell|}}$$
$$b^{|\ell|} = b^{|\ell|} - \alpha\frac{\partial\mathcal{L}}{\partial b^{|\ell|}}$$

Backpropagation is called like this because it first computes the error starting from the last network layer, and, then proceeds backwards to estimating the errors of the previous layers. Popular gradient methods used in network optimization are gradient descent and its variants, e.g. Stochastic Gradient Descent (SGD) [Bottou, 2010]. Other more advanced techniques such as adaptive optimization methods [Duchi et al., 2011; Kingma and Ba, 2014] overcome some limitations of SGD by trading additional computational cost for faster convergence rate.

2.4 Convolutional Neural Network (CNN) for Sentence Modeling

Convolutional neural networks became popular after the work of Yann LeCun on optical character recognition at AT&T Bell Labs.. During his work, he found that that neural networks with locally-connected layers and shared weights outperformed fully-connected networks in image recognition tasks. This result, at the base of the LeNet-5 success [LeCun et al., 1998], is due to the ability of CNNs to extensively handle very large images. Indeed, previous models, such as fully-connected networks, could manage large images only after downsampling them. These models were very expensive in term of memory and difficult to regularize in order to prevent overfitting. The capability of CNNs to handle large images is the result of two features: (i) parameter sharing and (ii) sparsity of connections. Parameter sharing allow the use of the same feature maps in order to to detect patterns in different positions of the image, while sparsity of connections makes it possible to compute the output of a neuron based only on a subset of input features. Thus, by using these two mechanisms, a neural network with fewer parameters can be trained on smaller feature maps and it is less prone to overfitting.

2.4.1 CNNs for Natural Language Processing

As we've seen, CNNs have been been extensively used in computer vision problems thanks to their ability to accurately detect position of different objects appearing in an image. However, not only they are very popular in computer vision [Krizhevsky et al., 2012; Lawrence et al., 1997; Karpathy et al., 2014], but they proved to be effective in many different NLP tasks. Differently from computer vision, CNNs are employed in NLP for modeling the information in a sentence. Early example of CNNs in NLP were introduced by Collobert et al. [2011a]; Kalchbrenner et al. [2014] and Kim [2014]. Typically, they use a convolution operator to compute a vector for every possible phrase appearing in sentence. Phrases are represented regardless whether they are grammatically or not. Later, they combine the different phrases representations into a unique vector encoding the meaning of the entire sentence. Surprisingly, CNNs perform generally well on a number of different NLP tasks even though the inner process of the model is not very linguistically motivated.

Convolutional Feature Maps

Convolution feature maps are the building blocks of the CNN architectures. Here we describe how convolution filters are applied to a sentence matrix $\mathbf{S}$. Typically, in NLP we represent every words in a sentence with k-dimensional vectors $\mathbf{x}_i \in \mathbb{R}^k$ and then we model the entire sentence by concatenating the individual word vectors. Thus, a sentence is represented as follows:

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_n$$

where $\oplus$ is the concatenation operator. We refer to $\mathbf{x}_{i:i+j}$ as the concatenation of words in range (i, j), or said differently, from time step i to time step $i + j$. At this point, we apply a convolution filter of window size h and word vectors of size k. Convolutional filters are parameter vectors $\mathbf{w} \in \mathbb{R}^{hk}$ learned with gradient descent optimization methods. At each time step, the convolution filters look at h different word vectors of size k and combines them into a single feature, e.g., $c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$. Here, f is a non-linear activation function. After applying the filter $\mathbf{w}$ to all the possible windows, i.e. concatenated vectors, of length h: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \cdots, \mathbf{x}_{n-h+1:n}\}$, we obtain the following feature map:

$$\mathbf{c} = [c_1, c_2, \cdots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

Pooling operator

Generally, feature maps have different length depending on the window size h and the number of words appearing in a sentence. To overcome this problem, besides using zero padding, we typically employ a pooling operator such as max-over-time pooling layer (or max pooling layer), to capture the most important activation from the map $\hat{c} = \max\{\mathbf{c}\}$. This way, we select a feature $\hat{c}$ that has very large activation and ignore the rest of the sentence. As a general rule, CNNs employ multiple filters $\mathbf{w}$ of different length. These filters correspond to different feature maps, which extract unigrams, bigrams, trigrams, etc. It is proved that using multiple feature maps results in more accurate models.

Classification after one CNN layer

After having applied a number of convolution operators followed by max-pooling operations, we concatenate all the $\hat{c_i}$ and obtain a final feature vector $\mathbf{z} = [\hat{c}_1, \cdots \hat{c}_m] \in \mathbb{R}^m$. Each $\hat{c}_i$ is the result of one of m max-pooling operations obtained by convolving m different filters over the sentence. Generally, these $\hat{c}_i$ values are fed to a softmax function $y = \text{softmax}(\mathbf{W}\mathbf{z} + b)$ for training a multiclass classifier whose parameters are optimized by reducing the standard cross entropy error.

2.5 Recurrent Neural Networks (RNNs) for Sentence Modeling

2.5.1 Vanilla RNNs

CNNs are very effective in modeling sentences, but unfortunately they do not consider the grammatical structure of elements in a text. In addition, they completely ignore the inherent sequential nature of language. An alternative way to model sentence information in NLP tasks is by using recurrent models. These models have been explicitly conceived for modeling one-dimensional sequence data that span over time. An example of such models is the Vanilla RNN. When applied for modeling sentences, a Vanilla RNN consumes a sequence of vectors, each corresponding to the current word, one step at the time. Then, it updates its the internal state as a function of the new word and the previous previous state. By doing so, the model can condition what to predict next based on information in previous words. Below, we report the equations describing a simple Vanilla RNN:

$$\mathbf{h}_t = \sigma\left(\mathbf{W}^{(hh)}\mathbf{h}_{t-1} + \mathbf{W}^{(hx)}\mathbf{x}_{[t]}\right)$$
$$\hat{y}_t = \text{softmax}(W^{(S)}\mathbf{h}_t)$$

$$\mathbf{W}^{(hh)} \in \mathbb{R}^{D_h \times D_h} \qquad \mathbf{W}^{(hx)} \in \mathbb{R}^{D_h \times d} \qquad \mathbf{W}^{(S)} \in \mathbb{R}^{|V| \times D_h}$$

Here, $\mathbf{h}_t \in \mathbb{R}^{D_h}$ is the hidden state at time t, while $\mathbf{x}_{[t]}$ corresponds to the embedding of the t-th word in the sentence. Generally, we start (at time 0) by initializing the hidden state to a vector of all zeros. In the subsequent steps, we compute the new hidden states $\mathbf{h}_t$ by multiplying (i) the linear layer $\mathbf{W}^{(hh)}$ with the hidden state at time $t-1$ and (ii) the linear layer $\mathbf{W}^{(hx)}$ with the input word vector at time t. Then the resulting vectors are summed element-wise and a non-linearity is applied. Typically, as last step, the final hidden state $\mathbf{h}_t$ is fed to a standard softmax function to classify the input sentence into a predefined set of categories.

Despite being interesting from the theoretical point of view, Vanilla RNNs are not used in practice as they suffer from the vanishing and exploding gradient problem [Bengio et al., 1994]. This problem arises when the signal becomes either too weak or to strong during the computation of the gradients at different time steps. In order to solve this problem, many solutions have been proposed, such as gradient clipping [Pascanu et al., 2013] and the design of new types of recurrent models like LSTMs and GRUs.

2.5.2 Gated Recurrent Units (GRUs)

Gated Recurrent Units (GRUs) were introduced by Cho et al. [2014]. The main idea behind them is to create processing units that have gates. These gates make possible to build models that learn to capture long distance dependencies from previous words. In addition, they also allow the error messages to flow at different strengths depending on the inputs. A GRU can be described by the following equations:

$$\begin{aligned}
z_t &= \sigma(\mathbf{W}^{(z)}\mathbf{x}_t + \mathbf{U}^{(z)}\mathbf{h}_{t-1}) \\
r_t &= \sigma(\mathbf{W}^{(r)}\mathbf{x}_t + \mathbf{U}^{(r)}\mathbf{h}_{t-1}) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}\mathbf{x}_t + r_t \circ \mathbf{U}\mathbf{h}_{t-1}) \\
\mathbf{h}_t &= z_t \circ \mathbf{h}_{t-1} + (1 - z_t) \circ \tilde{\mathbf{h}}_t
\end{aligned}$$

The GRU cell first computes the values of two gates: (i) the *update gate* z_t and (ii) the *reset gate* r_t. If the reset gate r_t is close to 0, the GRU cell ignores previous hidden states. This, essentially corresponds to discarding the previous memory, i.e. the past, and setting the hidden state to the transformed current word vector $\mathbf{x}_t$. This way, the model drops information that is irrelevant in the future. After that, the reset gate r_t is element-wise multiplied by $\mathbf{U}\mathbf{h}_{t-1}$ and summed to $\mathbf{W}\mathbf{x}_t$, returning the intermediate memory content $\tilde{\mathbf{h}}_t$. Differently, the update gate z_t controls how much of the past state matters for the future prediction. The final memory at time step combines current and previous time steps: $\mathbf{h}_t = z_t \circ \mathbf{h}_{t-1} + (1 - z_t) \circ \tilde{\mathbf{h}}_t$. Differently, the update gate z_t controls how much the past state matters for future predictions. The way a GRU cell overcomes gradient problems is by smartly using the update gate: if z_t is close to 1, the unit just passes the previous activation to the next processing unit, without applying a non-linearity. As result, the final network is less affected by the vanishing/exploding gradient.

Long Short-Term Memories (LSTMs)

LSTMs [Hochreiter and Schmidhuber, 1997] are a more complex type of units used in recurrent neural models. Networks composed of LSTM units can learn long range connections better than GRUs. Nowadays, LSTMs are the default choice for most NLP tasks. An LSTM unit can be described by the following equations:

$$\begin{aligned}
i_t &= \sigma\left(\mathbf{W}^{(i)}\mathbf{x}_t + \mathbf{U}^{(i)}\mathbf{h}_{t-1}\right) \\
f_t &= \sigma\left(\mathbf{W}^{(f)}\mathbf{x}_t + \mathbf{U}^{(f)}\mathbf{h}_{t-1}\right) \\
o_t &= \sigma\left(\mathbf{W}^{(o)}\mathbf{x}_t + \mathbf{U}^{(o)}\mathbf{h}_{t-1}\right) \\
\tilde{\mathbf{c}}_t &= \tanh\left(\mathbf{W}^{(c)}\mathbf{x}_t + \mathbf{U}^{(c)}\mathbf{h}_{t-1}\right) \\
\mathbf{c}_t &= f_t \circ \mathbf{c}_{t-1} + i_t \circ \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= o_t \circ tanh(\mathbf{c}_t)
\end{aligned}$$

Briefly, an LSTM is composed of an *input gate* i_t, a *forget gate* f_t, an *output gate* o_t, a *memory cell* $\tilde{\mathbf{c}}_t$, a final *memory* $\mathbf{c}_t$ and a final *hidden state* $\mathbf{h}_t$. The basic input gate determines how much important is the current memory cell (or the current word vector). Instead, the forget gate decides weather to forget or keep the past. Typically, if the value of the gate is 0, the unit forgets the past. The output gate says how much the cell is exposed. Basically, it attempts to separate what is important to output at a certain time step compared what is important to keep for later predictions. The values of all the gates is used to the final memory memory cell $\mathbf{c}_t$. The latter is computed as the sum of (i) the product of forget gate f_t with the previous final memory cell $\mathbf{c}_{t-1}$ and (ii) the product of input gate i_t with the new memory cell $\tilde{\mathbf{c}}_t$. f_t specifies how much the network has to keep or forget from the past, whereas the input gate i_t says how much the network has to keep or ignore the current word. The final hidden state $\mathbf{h}_t$ is obtained by applying tanh to $\mathbf{c}_t$ and multiplying it by the output gate o_t. LSTMs are very powerful models, but have few limitations: (i) they require large amount data in order to work properly and (ii) they can consume input sequence only in one direction (typically, from left to right). To overcome the latter problem, Schuster and Paliwal [1997] devised Bidirectional RNNs, which consume input sequences in both directions and use them to compute both forward and backward hidden states. Also in this case, the last hidden state $\mathbf{h}_t$ of a bidirectional network contains the encoding of the entire sequence and can be used for the final prediction.

2.5.3 Summary

In this chapter, we introduced the background required for understanding the contributions exposed in the following parts of the thesis. We provided an overview of supervised learning and introduced the theory behind discriminative classifiers and kernel-based methods. Then, we introduced SVM, a powerful machine learning algorithm belonging to the family of maximum margin classifiers. We show the primal and dual formulation of SVMs, as well as the "kernel trick", which make SVMs able to use structured input objects like trees. In the second part of the chapter, we introduce neural models and present the most common types of neural architectures: feed-forward, convolution and recurrent models. We discussed the problems of vanilla RNNs, such as vanishing/exploding gradient, and showed how LSTMs and GRUs were conceived to overcome such problems.

Chapter 3

Community QA with Structural Kernels

In recent years, community Question Answering websites have gained popularity online. These systems allow users to freely ask any question to a community of users. One the positive side anybody can freely ask any question and expect some good, honest answers. On the negative side, these websites require that users go through all possible answers and making sense of them before finding one relevant answer. Unfortunately, this task is very time-consuming for humans. To overcome this problem, researchers propose a new cQA task at SemEval [1] with the purpose of automatizing the process of finding good answers to new questions posted in a community-created discussion forum. The main cQA task is defined as follows: given a large collection of question-comment threads created by a user community, predict what are the comments most useful for answering a new question. Trying to directly solving this task is difficult. As an example, let consider the test question: *Can I drive with an Australian driver's license in Qatar?*, which is supposed to be new with respect to the collection of previous question-answer pairs on a forum. In order to answer this question, one can (C) try to directly find a comment representing a valid answer for the new question, or (B) search for one or more forum *related* questions that are similar to the new question. Once a related questions is located, we can answer to a new question by (A) finding a comment providing a valid answer for a related forum question. Solving problems (A), (B) and (C) requires components able to effective measuring similarity between two pieces of text. Such approaches need to go beyond simple "bag-of-words" representations and "word matching" techniques in order to capture the new NLP phenomena connected with the community question answering scenario, e.g. relation between comments in a thread, relations between different threads

[1] http://alt.qcri.org/semeval2015/task3/

Q: Could someone advise the best psychiatrist/psycologist in DOHA?

R	*GS*	*Answer Text*
1	-1	c_1: i heard a good doctor in doha clinic...
2	-1	c_2: ok..shall check that out..thank you :)
3	+1	c_3: " Visit Psychiatrist clinic of Hamad Hospital located opposite " " The Center" " exactly facing KFC."
4	-1	c_4: Lot of Qlers are required to visit the psychiatrist including me so better to note down the above addresses
5	-1	c_5: Princess.I agree to equinox
6	-1	c_6: have any of you guys been there urself?
7	+1	c_7: I dint know hamad hospital was opp to The Centre :o
8	-1	c_8: but the problem with hamad hospital is getting appointment not later then 2 months time compare to private hospital like al ahly or doha hospital immediatly can see the doctor
9	-1	c_9: I have been. good treatment. .but why do u need. :)
10	+1	c_{10}: I heard dr. Ajju Clinic is also good one in doha with expereience

Table 3.1: An example of Task A, i.e. question-answer similarity task: The question **Q** is reported on the top of the table, while user answers are reported below in temporal order. For each answer, the position (R) and the gold standard (GS) are reported.

an so on. In the remaining of the chapter we first provide a formal definition of the three subtasks, i.e. (A), (B) and (C), at the base of community QA. Then, we present our models based on structural representations for solving them, which obtained results in line with state of the art in the context of the SemEval-2016 challenge [Nakov et al., 2016b].

3.1 Task A: Question-Comment Similarity

The first task that we are going to discuss is Task A, i.e. question-comment similarity (or question-comment relevancy). Table 3.4 presents an example of such task: a forum question is showed on the top, while the list of the comments submitted in response by some users is reported below. The task can be defined as follows: re-rank the comments in the thread according to their pertinence with respect to a forum question. Pertinence is defined according to three classes: (*i*) good: the comment is definitely relevant; (*ii*) potentially useful: the comment is not good, but it still contains related information worth checking; and (*iii*) bad: the comment is not relevant (e.g., it is part of a dialogue or unrelated to the topic). In our case, for evaluation purpose, we considered both potentially useful and bad comments as irrelevant.

3.1.1 Structural Representations for question-answer similarity

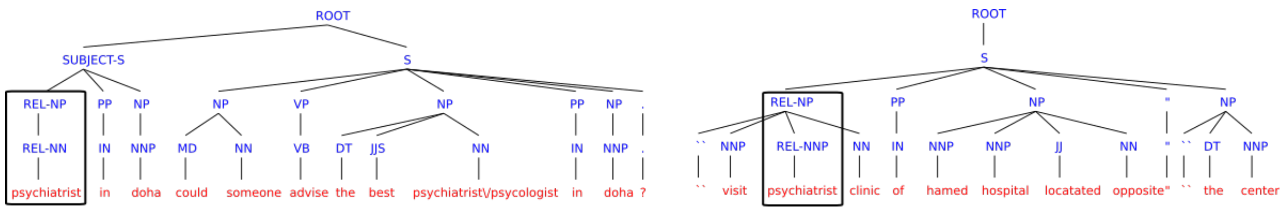


Figure 3.1: Shallow Tree representation for q/a pairs. The question and answer trees are depicted on the left and right side, respectively. The subtrees sharing the same lemma, i.e psychiatrist, are marked with the relational tag REL to encode information about their relatedness.

In order to build a model for computing question-answer similarity, we need to specify a way for representing question/answer pairs. Here, we decide to stick to the representation used in [Severyn and Moschitti, 2012], adjusted for this particular task. More specifically, we constructed a syntactic tree for each question and answer. Each tree is composed of five levels, organized as follows: At the bottom level there are the sentence lemmas, preceded by the their part-of-speech tags at the preterminal level. In the third level we have phrase chunks, i.e NPs, VPs, and PPs. The latter are in turn grouped into sentences at fourth level, represented with **S** tags. Finally, all the sentences are attached to a **ROOT** node, located in the top of the tree. Furthermore, in order to adjust the representation to the cQA setting, we enrich the structure with additional cQA-specific knowledge about question threads. The original representations of syntactic trees is modified as follows:

1. as forums questions are composed of a subject and a body, we put the question subject in a separate subtree under the **SUBJECT-S** root in the question tree.
2. we preserve the punctuation, differently from [Severyn et al., 2013], as some comments ask for additional information and thus contains a question mark. This is a strong feature that a new question is contained in the comment, and thus, is very unlikely to answer the original question.
3. we delete phrases that users employ to sign their own posts, e.g. *"The tough gets going..."*. Signatures create noise when measuring relevancy of answers with respect to questions. Here, we considered all the strings with length exceeding 20 characters that occur at the end of more than one comment by the same user, as signatures. We selected this heuristic empirically, by looking at training data and and selecting the number of characters that separate signature text from relevant answer content in most cases. This heuristic, far from being perfect, proved effective during our experiments.

As done in [Severyn and Moschitti, 2012], we linked subtrees in question-answer pairs that contain common phrases. In particular, we connect subtrees rooted at NP, PP and VP phrase nodes that span over words in a matching relation with lexicals of the other tree. Such link are marked with the presence of REL tag (see Figure 3.1). This tag indicates that the answer is relevant with respect to the question in a pair. By adding it, we encoded information about the relatedness of two subtrees. A more in-depth explanation of structures used for modeling the Task A can be found in [Tymoshenko et al., 2016a].

3.1.2 Convolutional network features

In addition to structural representation, we looked at novel deep learning models for automatically engineering features for Task A. To this purpose, we train the Convolutional Neural Network (CNN) by Severyn and Moschitti [2015b] to model information in question-answer pairs. The network is presented more accurately in Chapter 4, but it can be described very generally as a simple CNN augmented with relational feature embeddings. We use this network to train two sentence encoders returning the distributed representations of questions and answers in the form of embeddings. Also, we generate a joint embedding encoding the interactions between question and answers. Finally, we concatenate the question, comment and joint embeddings provided by the network into a single feature vector to be fed to our model.

3.1.3 Text similarity features

Text similarity features compute the degree of similarity between a pair of text elements. In our model, we include three kinds of similarity measures: lexical, syntactic and semantic features [Barrón-Cedeño et al., 2015; Nicosia et al., 2015]. More particularly, we compute 20 similarity features $sim(q_i, q_j)$ using word n-grams, after stopwords removal, greedy string tiling [Wise, 1996], longest common subsequences [Allison and Dix, 1986], Jaccard coefficient [Jaccard, 1901], word containment [Lyon et al., 2001], and cosine similarity.[2]

3.1.4 Context features

One important aspect to consider when modeling question-answer relatedness is the context in which a comment appears with respect to the other answers in the thread. Typically, comments in question threads are organized sequentially according to the time

[2]These features were computed by using the DKPro Code toolkit Eckart de Castilho and Gurevych [2014] for preprocessing the texts in English. The OpenNLP's tokenizer, POS-tagger, chunk annotator and Stanford's lemmatizer were used, all accessible through DKPro Core.

users post answers in response to a new question. Some important factors to take in consideration when assessing the value of comment is whether:

- the thread includes further comments by the user who originally asked the question
- if the same user is behind various comments in the threads
- the forum category the thread belongs to.

Thus, we've considered a set of feature describing a comment in the context of the entire thread. For example, we include boolean features characterizing potential dialogues that represent `irrelevant` side comments or the position of the comment in the thread. Other features considered are the categories of the questions in the forums, as well as the occurrence of specific strings (e.g. signatures) or the length of a comment. A complete description of these features can be found in Nicosia et al. [2015]

3.1.5 Our model for question-answer similarity

Here we present our model based on SVMs and structural representations for solving the Task A.

Datasets: For training and testing our model we used the official data[3] of the SemEval-2016 challenge for Task A. The dataset is composed of $2,361$ forum questions and each question is paired with first 10 comments submitted by users in chronological order. This results in $23,610$ question-comments pairs, organized as follows: $17,900$ pairs in the train set, $2,440$ pairs in dev. set and $3,270$ in test set. Each comment in the dataset was annotated with a label indicating its relevancy to the question.

Model: We learn a reranking model $r : Q \times C \to \mathbb{R}$, which given input a question in Q and a comment in C, returns a similarity score in $\{0, 1\}$. The model is learned by training an SVM operating on two kernels:

- a polynomial kernel of degree 3 applied to concatenation of context features (Section 3.1.4), similarity features (Section 3.1.3) and CNN features (Section 3.1.2)
- a the tree kernel applied on the syntactic trees (Section 3.1.1) of question-answer pairs $((q_1, c_1^i), (q_2, c_2^j))$:

$$\begin{aligned} K((q_1, c_1^i), (q_2, c_2^j)) &= PTK(t(q_1, c_1^i), t(q_2, c_2^j)) \\ &+ PTK(t(c_1^i, q_1), t(c_2^j, q_2))) \end{aligned} \tag{3.1}$$

[3]`http://alt.qcri.org/semeval2016/task3/index.php?id=data-and-tools`

In Equation 3.1.5, q_1 and q_2 are two generic forum questions, while c_1^i and c_2^j are the i-th and j-th comment appearing in question thread q_1 and q_2, respectively. The function $t(x, y)$ extracts the syntactic tree from text x, enriching it with REL tags computed with respect to y.

The SVM is trained on both training and development sets.

3.1.6 Experiments and Results

Table 3.2 shows the results obtained for Task A. Despite our model uses mainly automatically engineered features (compared to other participants), it achieved the second position in the SemEval-2016 challenge. Interestingly, as reported in [Barrón-Cedeño et al., 2016], while structural representations gave no impressive boost, they were still able to improve by 0.37 absolute points in MAP over a constrictive model that does not use structural information. In order to better study the results of the model, Tymoshenko et al. [2016c] investigated how the different approaches for automatically generating features, i.e. TKs and NNs, impact on the final performance of the system. Their results are reported in table 3.3. As can be seen, the CNN slightly outperforms the TK model by around 1 absolute point in MAP. In addition, we observe that the TK model augmented with context features, i.e. V_{CQA} and trained with PTK obtained the MAPs of 78.78 and 78.80 on dev. and test set, respectively. As both TK and CNN models achieve state-of-the-art results on cQA, the authors carried out further experiment to test if CNNs and TKs combined together could improve the final results. To do so, they added the question embedding V_{QE} and V_{CE} learned by a CNN with a feature vector encoding context features V_{CQA}. Then, they added these feature vectors to the TKs. The results are reported in Sec. 3 of Table 3.3. Experiments show that (i) CNN do not improve over TK models, but (ii) combining TK models with embedding features generated by CNNs improve the final performance of the base neural models. Once again, this proved the effectiveness of including syntactic representations when designing models for QA tasks.

3.1.7 Task B: Question-Question Similarity

Developing components able to automatically assess the similarity between two questions is critical to locate threads on a forum containing questions similar to new questions input by a user. Thus, the second task that we deal with in cQA is the Task B, i.e. question-question similarity. Table 3.4 shows an example of such task: a new out-of-forum question is reported in the top row of the table, while, a list of ten related forum questions retrieved by means of Google search engine is reported at the bottom. The goal is to to rerank the list of forum questions with respect to the new question by assessing if the former are (i)

A	MAP	AvgRec	MRR	P	R	F_1	Acc
ConvKN-primary (our model)[2]	77.66	88.05	84.93	75.56	**58.84**	**66.16**	**75.54**
best	**79.19**	**88.82**	**86.42**	**76.96**	55.30	64.36	75.11
baseline	59.53	72.60	67.83				

Table 3.2: Performance of our official primary submissions to SemEval Task A. Best-performing and baseline systems included for comparison. The super-index in the primary submission stands for the position in the challenge ranking. The baseline is based on the chronological order of the comments submitted by users in response to a form question.

perfect match: the new and forum questions request roughly the same information, (*ii*) relevant: the new and forum questions ask for similar information, or (*iii*) irrelevant: the new and forum questions are completely unrelated. In our case, for evaluation purpose, we consider both perfect match and relevant forum questions as relevant.

3.1.8 Structural Representations for question-question similarity

Similarly to Task A, we used structural representations for representing pairs of questions [Da San Martino et al., 2016; Martino et al., 2016]. In particular, we constructed syntactic trees for each question pair composed of a new question and a forum question. Also here, the syntactic trees are build by using the same representation in [Severyn and Moschitti, 2012]. However, differently from the latter, we adjusted the tree representation to the structure of the questions threads populating online web forums. Typically, a question contains a subject and body, which in turn are composed of several sentences including sub-questions, greetings, elaborations and so on. Thus, we connect the parse trees corresponding to all the sentences in the question Subject and Body with a fake root node.

In addition, we link the subtrees that contain matching lexical in (q_o, q_s) by connecting with a REL tag the NP, PP and VP phrases in the two two macro-trees. For example, given the original question q_o in Table 3.4 with the seventh candidate, q_{s_7}, we build the graph in Figure 3.2. As can be see from the picture, the top tree corresponds to the original question q_o composed of two sentences nodes: the subject and the body. At the bottom, there is the tree of the related question q_{s_7}, which in turn contains a subject and body. The subtrees containing same lemmas, e.g. *place*, *tourist*, *qatar* and *visit*, have been linked with the REL tag.

		DEV		TEST	
Model	Kernel	MAP	MRR	MAP	MRR
ConvKN-primary (Our model) [2]	P	-	-	77.66	84.93
CNN and TK models					
V$_{QF}$	P	63.45	70.51	73.50	82.98
CNN	n/a	67.41	73.46	77.12	83.85
TK	PTK	64.10	71.97	76.67	85.53
TK + V$_{CQA}$	PTK, P	68.45	74.49	**78.80**	86.16
Combining TK and CNN models					
$V_{QE\|CE}$	P	65.63	72.63	75.15	82.37
$V_{QE\|CE\|CQA}$	STK,P	68.17	75.32	77.22	82.98
TK + V$_{QE\|CE}$	STK, P	65.63	72.69	75.15	82.37
TK + $V_{QE\|CE\|CQA}$	STK,P	**68.92**	76.61	77.25	84.16

Table 3.3: Performance of TK and CNN combined with thread-level features V_{CQA}. The symbol V_{QE} refer to the question embedding, while V_{CE} indicates the answer embedding. The names in the second columns, i.e. P, PTK and STK refer the type of kernel used; they stand for polynomial, Partial Tree and Subset Tree kernel, respectively. The top section shows the performance of our primary submission, i.e. ConvKN-primary, while the bottom part of the table shows the performances of individual models when trained separately (section 2) and combined (section 3).

Q: What are the tourist places in Qatar? I'm likely to travel in the month of June. Just wanna know some good places to visit.

G	*GS*	*R*	*Question Text*
1	-1	8	q_{s_1}: The Qatar banana island will be transferred by the end of 2013 to 5 stars resort called Anantara. Has anyone seen this island? Where is it? Is it near to Corniche?
2	+1	2	q_{s_2}: Is there a good place here where I can spend some quality time with my friends?
3	-1	7	q_{s_3}: Where is the best beach in Qatar? Maybe a silent and romantic bay? Where to go for it?
4	-1	9	q_{s_4}: Any suggestions on what are the happenings in Qatar on Holidays? Something new and exciting suggestions please?
5	-1	3	q_{s_5}: Where in Qatar is the best place for Snorkeling? I'm planning to go out next friday but don't know where to go.
6	-1	6	q_{s_6}: Can you give me some nice places to go or fun things to do in Doha for children 17-18 years old? Where can we do some watersports (just for once, not as a member), or some quad driving? Let me know please. Thanks.
7	+1	1	q_{s_7}: Which all places are there for tourists to Qatar? My nephew 18 years on visit.
8	-1	10	q_{s_8}: Could you suggest the best holiday destination in the world?
9	-1	5	q_{s_9}: I really would like to know where the best place to catch fish here in Qatar is. But of course from the beach. I go every week to Umsaeed but rerly i catch somthing! So experianced people your reply will be appreciated.

Table 3.4: A question-question similarity reranking example, for each candidate the Google rank (G), the gold standard (GS) relevance and our rank (R) are reported.

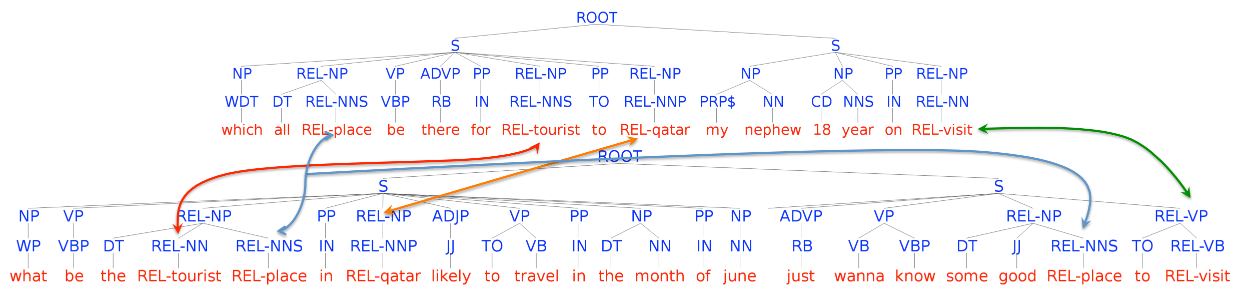

ROOT
S
S
NP REL-NP VP ADVP PP REL-NP PP REL-NP NP NP PP REL-NP
WDT DT REL-NNS VBP RB IN REL-NNS TO REL-NNP PRP$ NN CD NNS IN REL-NN
which all REL-place be there for REL-tourist to REL-qatar my nephew 18 year on REL-visit
ROOT
S
S
NP VP REL-NP PP REL-NP ADJP VP PP NP PP NP ADVP VP REL-NP REL-VP
WP VBP DT REL-NN REL-NNS IN REL-NNP JJ TO VB IN DT NN IN NN RB VB VBP DT JJ REL-NNS TO REL-VB
what be the REL-tourist REL-place in REL-qatar likely to travel in the month of june just wanna know some good REL-place to REL-visit

Figure 3.2: Our representation based on syntactic trees for the Q/Q pairs enriched with REL links.

3.1.9 Rank Feature

One interesting bit of information for modeling question-questions similarity is the rank of a question in the list of elements to rerank. To exploit such information we use a rank feature, which encodes meta-information about the position of related question thread with respect to new questions returned by the Google search engine. The experiments in Table 3.5 show that encoding the rank feature as the inverse of question position results in the best performance.

3.1.10 Our Model for question-question similarity

Here we describe our model using SVMs and syntactic structures for solving the Task B.

Data: As training and test material we use the part of SemEval-2016 dataset dedicated to Task B. The dataset contains 387 new questions, and for each new question, 10 related questions were retrieved. This results in $3,869$[4] pairs of questions, divided as follows: $2,669$ pairs in the train set, 500 pairs in dev. set and 700 pairs in the test set. Each pair in the dataset was annotated with a label indicating their similarity.

Model: We implemented a ranking function $r : Q \times Q \to \mathbb{R}$, which given in input two questions in Q, returns a similarity score in $\{0, 1\}$. The function was learned by training an SVM operating on three kernels:

- an RBF kernel on the similarity features (section 3.1.3),
- an RBF kernel on rank feature (section 3.1.9) and a

[4]For one new question, we could retrieve only 9 similar questions.

- a partial tree kernel (PTK) defined on question pairs:

$$\begin{aligned} K((q_o, q_s^i), (q_o, q_s^j)) &= PTK(t(q_o, q_s^i), t(q_o, q_s^j)) \\ &+ PTK(t(q_s^i, q_o), t(q_s^j, q_o)) \end{aligned} \tag{3.2}$$

  Here, q_o is a new out-of-forum question, while q_s^i and q_s^j are the forum questions ranked by Google at i-th and j-th positions, respectively. $t(x, y)$ extracts the syntactic tree from text x, enriching it with REL tags computed with respect to y. At training time the C parameter of the SVM was set to 1, while both tree and RBF kernels used some default values for the parameters. During our experiments, we tried different kernels, e.g. linear and RBF, and we selected the latter as it resulted in better results on the dev. set (see Table 3.5). The SVM is trained on union of the training and development sets. The results are reported in Table 3.7.

3.1.11 Experiments and Results

The Table 3.5 reports the results of our experiments on dev. and test data of SemEval-2016 Task B. In agreement with the challenge experimental setting, we evaluate our rankings with Mean Average Precision (MAP), average Recall (AvgRec) and Mean Reciprocal Rank (MRR). The top section shows the results of our model, i.e. ConvKN-primary, and compare its performance against that of the Google baseline and the best system in the challenge. As it can be seen, we were able to improve over the Google baseline (GR) by 1.27 absolute points in MAP, which resulted in our system scoring second at the SemEval-2016 challenge on question-question similarity. This result is very good if we consider the fact that Google provides strong baseline, as it is the product of many years of engineering in the field of Information Retrieval.

Feature Ablation Study

In order to better understand the final results of our system, we conducted some post-competition experiments to assess the impact of the different feature sets on the final task. In particular, we experimented with three different features for modeling the ranking function r:

- tree kernels applied to the syntactic structured of question pairs,
- similarity features computed between q_o and q_s, and
- rank feature, i.e. kernel over the question position in the rank, produced by the Google search engine (GR).

The Table 3.5 reports the results of such experiments. In the second section of the table are reported the experiments of the models using only a combination of similarity

	DEV						TEST					
Model	P	R	F_1	MAP	AvgRec	MRR	P	R	F_1	MAP	AvgRec	MRR
ConvKN-primary (Our model) [2]	-	-	-	-	-	-	68.58	66.52	67.54	76.02	90.70	84.64
best	-	-	-	-	-	-	63.53	69.53	66.39	76.70	90.31	83.02
IR baseline	83.33	11.68	20.49	71.35	86.11	76.67	49.64	59.66	54.19	74.75	88.30	83.79
Sim.	76.32	40.65	53.05	64.80	82.52	73.73	71.52	46.35	56.25	70.70	85.78	80.58
TK	73.10	58.41	64.94	69.97	86.86	77.73	67.44	62.23	64.73	73.98	88.90	82.55
TK + Sim	72.89	56.54	63.68	71.07	87.72	78.14	68.87	62.66	65.62	73.81	89.21	82.86
Linear Kernel												
Sim + *pos*	74.81	45.79	56.81	68.04	85.07	76.00	68.10	47.64	56.06	71.99	87.92	81.19
Sim + pos^{-1}	77.78	45.79	57.65	70.17	85.98	78.17	71.15	47.64	57.07	75.15	89.19	84.29
TK + *pos*	75.14	60.75	67.18	71.77	88.46	78.12	66.96	66.09	66.52	75.34	90.67	83.19
TK + pos^{-1}	73.99	59.81	66.15	72.64	87.69	75.58	68.66	63.95	66.22	76.18	90.62	84.62
RBF Kernel												
Sim. + *pos*	77.34	46.26	57.89	70.42	86.38	78.50	69.75	48.50	57.22	74.61	89.10	83.81
Sim. + pos^{-1}	77.95	46.26	58.06	69.82	85.91	77.17	70.25	47.64	56.78	74.58	89.09	83.57
TK + *pos*	75.43	61.68	67.87	72.93	87.95	77.54	67.40	65.67	66.52	75.72	90.80	83.86
TK + pos^{-1}	75.72	61.21	67.70	73.65	88.78	79.58	68.33	64.81	66.52	76.41	91.14	84.62

Table 3.5: Ranking-based features combined with linear and RBF kernels. In the top section we report the performance of our primary submission, i.e. ConvKN-primary, to SemEval-2016 Task 3 for Task B. Best-performing and baseline systems included for comparison. The super-index in the primary submission stands for the position in the challenging ranking. The baselines are provided by task organizers; they are based on Google search engine rankings.

features and TKs. As it can be seen, the results of these models are below the Google baseline. In contrast, when the rank feature is included, our best model outperforms the MAP of Google rank by 2.30 and 1.66 absolute percent points on the development and test sets respectively. At the bottom of the table, we report the results obtained by applying different kernels on the rank feature. Interestingly, better results are obtained when we apply an RBF kernel on the Position feature. This can be explained with the fact that RBF kernel can more effectively express higher similarity values when positions of questions are close.

3.2 Task C: New Question-Comment Similarity

The last task that we discuss is Task C. This is similar to task A, but in this case the relevance of one-hundred comments is assessed against a new out-of-forum question. Table 3.6 shows a ranking example for Task C. The new out-of-forum question is shown

R	GS	*Answer Text*
		$\mathbf{Q}_{\text{new}}$: how to extend the visit visa after 6 months and how long period?
		$\mathbf{Q}_{\text{rel}}$: Maximum period of a Visit Visa?
1	+1	c_1: MAXIMUM 6 MONTHS BROTHER AFTER THAT YOU HAVE TO EXIT FROM QATAR.
2	+1	c_2: After 6 months you can get an extension also....
3	-1	c_3: I am on tourist visa for 1 month.. can i also get extension? for how many months and how much? tnx a lot.
4	-1	c_4: You can't get an extension; may be i could be wrong; but extension is valid for family visa holders; after they have cleared their medical.
5	+1	c_5: no extension for tourist visa i think...
6	-1	c_6: ok. tnx a lot.. do you know any company hiring for office staffs?
7	+1	c_7: you can also extend your tourist visa but very costly as compared with extending a family visit visa. maybe what you can do is exit from qatar then; after three months; apply again for a family visit visa (if you have a relative here)
8	+1	c_8: you can extend a family visit visa for a maximum six months; but in any case that you have to extend it again after your 6 months limit; you can do that; too. i think upto 1 month; more. you just have to submit application form again to immigration with your flight booking details.
9	-1	c_9: hi rtaure..tnx.. what are the requirements in extending tourist visa and how much? if I exit; I can go back in qatar after 3 months?
10	+1	c_{10}: you guys are mixing Family visit visa and Tourist visit visa. Family visit visa (when you were sponsored by one of your family member)initially is valid for 1 month; and can be extended later for another 5 months (total 6 months stay) after undergoing Medical examination. Tourist visit visa is is valid for 30 days and can be extended for another 30 days ; 60 days stay in total.

Table 3.6: A new question-comment reranking example, for each candidate the chronological rank (R) and the gold standard (GS) relevance are reported.

in the top row of the table. The second row shows a related question retrieved by the Google search engine, using the new question as query. Then, a list of comments posted in response to a related forum question is reported. The goal is to predict relevancy of comments with respect to the new out-of-forum question. As in task A, three classes exist in this case: (*i*) good: the comment is relevant; (*ii*) potentially useful: the comment is not good, but it still contains related information worth checking; and (*iii*) bad: the comment is irrelevant. For evaluation purposes, both potentially useful and bad comments are considered irrelevant.

3.2.1 Structural Representations for new question-answer similarity

From a practical perspective, Task C is very similar to Task A, but differently from the latter, a comment needs to be reranked with respect to a fresh question. Thus, it would make sense to use the same structural representations for modeling question-answer pairs employed in Task A. Unfortunately, we did not have time to experiment with structural representations at SemEval-2016 due to the restricted timing of the challenge. Thus, in the remaining of the chapter, we describe the results of our experiments using only feature vectors. We leave the task of integrating s structural representation in our best model as future work.

3.2.2 Our Model for new question-answer similarity

Here, we present our SVM model for solving the Task C.

Data: For training and testing our model, we use the official data of the SemEval-2016 challenge for Task C. The dataset is composed of 387 new out-of-forum questions and each question is paired with 100 forum comments. This results in $38,690$ new question-comment pairs, organized as follows: $26,600$ pairs in the train set, $5,000$ pairs in dev. set and $7,000$ pairs in the test set. Each comment is labeled with its relevancy with respect to the new question.

Model: We learned a reranking function $r : Q \times C \to \mathbb{R}$, which given a question in Q and an answer comment in C, returns a similarity score in $\{0, 1\}$. The model is trained using an SVM operating on two RBF kernels:

- The first kernel acts on similarity features (see Section 3.1.3)
- The second kernel operates on two features:
  - the rank feature describing the position of the forum threads with respect to the new question (see Section 3.1.9)
  - the relevancy scores obtained from the prediction of a question-comment classifier trained on Task A, without tree kernels. The score or classifiers have been computed in cross-validation.

The SVM is trained on the union of the training part 2 and dev. set.

3.2.3 Experiments and Results

In Table 3.7 we report the results of our model on the SemEval-2016 Task C. In agreement with the challenge setting, we evaluate our reranking in terms of MAP, AvgRec and MRR. As it can bee seen from table, our model obtained a MAP of 47.15, ranking 8th at the SemEval-2016 challenge. This place corresponds to an average performance in the ranking of the participants to the challenge. It is important to mention that this is the only model that did not include tree kernels. However, experiments of other participants

C	MAP	AvgRec	MRR	P	R	F_1	Acc
ConvKN-primary^8	47.15	47.46	51.43	45.97	8.72	14.65	90.51
best	55.41	60.66	61.48	18.03	63.15	28.05	69.73
baseline	40.36	45.97	45.83				

Table 3.7: Performance of our official primary submissions to SemEval-2016 Task 3 for tasks C. Best-performing and baseline systems included for comparison. The super-index in the primary submission stands for the position in the challenge ranking. The baselines are as provided by the task organizers; they are based on Google search engine ranking.

[Mihaylova et al., 2016] show that a high-performing model for Task C can be assembled by opportunely combining the relevancy score returned by the question-comment classifier for Task A and the similarity score of question-question similarity classifier for Task B. This finding confirms the importance of building good models that can solve the two subtasks, i.e. A e B, over which the Task C (the main cQA task) factorizes.

3.2.4 Conclusions

In this chapter, we presented the results obtained by our models based on structural representation for community QA in the context of SemEval-2016. Our models achieved the second position in two out of three tasks, i.e. A and B. Their ranking in the challenge confirms the high-quality of our solutions, which are competitive with the best system submitted. This results show that tree kernels can achieve the state of the art when applied to cQA, even in those cases where questions are mainly non-factoid and the text is typically informal and noisy. Once more this confirms the importance of modeling syntactic information in relational text inference problems such as question-question similarity and question-answer relevancy.

Chapter 4

Neural models for Community Question Answering

As previously discussed, a critical aspect when implementing community Question Answering systems is the need to design modules that capture the most salient characteristics of pairs of text. In Chapter 3 we showed how to use structured models based on kernel methods to automatically learn relevant syntactic patterns between two pieces of text, e.g. question-question ad question-answer pairs. Unfortunately, while being effective, these models require complex pipelines with many components, e.g. part-of-speech taggers, constituency parsers, each of them being difficult to replicate. In addition, models based on structural kernels do not make it easy the task of building modular systems that can be trained to solve many different problems. Interestingly, an alternative direction is provided by deep-learning methods, which demonstrated to be very effective in many NLP tasks. These methods use a low-dimensional vector representation of words, known as word embeddings, as input features. Then, these embeddings are combined together by means of compositional operations into higher-semantic representation useful for solving a specific task. In this section, we show how to use neural networks for modeling relationships between pieces of text and solving the tasks A, B and C, introduced in the previous chapter. First, we train a baseline CNN model [Severyn and Moschitti, 2015b] for solving each task, individually. Then, we proposes a new multi-task learning (MTL) architecture, which we train *jointly* for solving all the tasks at the same time. This choice is motivated by the fact that the tasks A, B and C are deeply semantically connected, thus, the knowledge learned by modeling one problem can be useful also for solving the others. Finally, we report the results of our experiments, which confirm that the shared representation and jointly learning dramatically increase the performance on the main cQA task, especially in presence of scarce data.

4.0.1 Related Work

Learning components able to infer relations between two pieces of text is crucial for many NLP tasks. Typically, previous approaches to relational text inference consisted in feature vectors encoding a number of similarity features between two pieces of text. However, in recent years, new neural network models have been proposed for:

1. measuring question-question similarity; and

2. measuring question-answer relatedness.

In the following section, we present the related works on relational text inference, whose solutions are based on neural models.

Related Work of NNs for question-question similarity.

- **Siamese Networks**: These networks have been proposed for modeling tasks such as textual similarity, paraphrase identification and mention normalization problems. Some examples are:

  - The MaLSTM [Mueller and Thyagarajan, 2016] represents two sentences by using an LSTM encoder and learn a Manhattan distance between them. After that, the resulting feature representations given by the network are used to train a SVM classifier for recognizing textual entailment.
  - The SCQA convolutional network [Das et al., 2016] pre-train the sentence encoders with forum data to map questions and answers on web forums in the same space. Then, use the sentence encoders to compute the distributed representations of the two question. The pre-training steps improve the final results on the question-question similarity task.

- **Attention-based Networks**: Attentive networks [Parikh et al., 2016] use the attention mechanism to encode the representation of a sentence in a pair by also considering the other sentence.

- **Compare-Aggregate Networks**: These networks [Bian et al., 2017; Wang et al., 2017] extract multiple views from the same sentences, and then match each view by using a number of similarity functions. The results are then aggregated and used to produce a final similarity score.

4.0.2 Related Work of NNs for question-answer relevancy.

- **CNN + relational features**: Severyn and Moschitti [2015a] use a convolutional neural network for classifying the relevancy of an answer with respect to a question. The network use word overlap features for encoding relations between the two text elements.
- **CNN + Attention**: Yin et al. [2016] propose a convolutional network with attention for training sentence encoders that represent sentence information by taking in consideration their counterpart.
- **LSTMs and Bi-LSTM**: Tan et al. [2016] compare networks based on convolutional filters and recurrent units for modeling sentence information in QA. They show that a network having two LSTM sentence encoders and an attention layer on top consistently outperform other networks on two QA datasets. Similarly, Cohen and Croft [2016] propose a bidirectional LSTM network, but trained with a rank sensitive loss function for computing question-answer relevancy.
- **Pairwise Rank CNN** Rao et al. [2016] use two pointwise neural networks with a stacked on top a fully-connected layer. Each convolution network encode a question/answer pair. Typically, the network is feed with a positive and a negative pair and use a triple loss such that positive pairs (q, p^+) are assigned larger similarity scores than negative pairs (q, p^-).

4.1 Task A, Task B, Task C

In this section, we briefly recap the three tasks at the base of cQA and present our neural models for individually solving each task.

(A) predict if a comment produced in response to a forum question contains a valid answer;
(B) re-rank a set of questions according to their relevancy with respect to the original question; and
(C) predict if a comment produced in response to a previous question posed on the cQA forum represents a valid answer to a fresh out-of-forum question.

4.1.1 Preliminaries

Before starting to approach cQA with NNs, we looked for models that deliver state-of-the-art performances and whose results are fully reproducible. After some research, we selected the S&M model described in [Severyn and Moschitti, 2015b] to train neural models for the three cQA tasks. The choice of this model is motivated by the fact that this model is simple and well studied, fast to train, robust to the choice of hyperparameters, and its performance have been successfully reproduced and studied in several subsequent works [Rao et al., 2017; Chen et al., 2017; Sequiera et al., 2017]. In the next section, we describe the S&M network for relational text inference problems and use it to train individual models for solving the tasks already discussed.

4.1.2 The S&M neural model for relational text inference

We implemented the CNN model proposed by Severyn and Moschitti [2016] (Figure 4.1) and originally conceived for solving general QA tasks. The S&M model takes in input a question and a candidate answer passage, and learn a function $f : Q \times D \rightarrow \{0, 1\}$. The output correspond to the probability that the passage contains a right answer. The network learns f, using two separate sentence encoders $f_q : Q \rightarrow \mathbb{R}^n$ and $f_d : D \rightarrow \mathbb{R}^n$, which map a query or a question and a document into a fixed size dense vector of dimension n. The resulting vectors are fed to a hidden layer with a non-linearity, and the final soft-max layer performs the final classification. Each sentence is encoded into a fixed size vector using an embedding layer, a convolution operation and a global max pooling function. The embedding layer transforms the input sentence, a sequence of tokens, $X = [x_1, \cdots, x_i, \cdots, x_n]$, into a sentence matrix, $S \in \mathbb{R}^{m \times n}$, by concatenating the word embeddings w_i corresponding to the tokens x_i in the input sentence. One novelty introduced by Severyn and Moschitti [2016] in their model is the word overlap feature, which encode matches between two words in two pieced of text. In particular, each word w in the input sentences is associated with a *word overlap* index $o \in \{0, 1\}$, where $o = 1$ means that w is shared by both sentences, e.g. question and answer, $o = 0$ otherwise. This feature is represented with an embedding of 5 dimensions and serve the purpose of injecting relational information between the representations of two input texts.

The architecture in Figure 4.1 is general, thus it can be used in many different domains, data and tasks. In order to effectively model the different types of relations between two text elements in cQA (e.g. similarity, relatedness, etc...), we customized the implementations of the S&M network as required by each cQA task. In the next section, we discuss the modification that we applied to the network in our effort to adapt the learned model to each task.

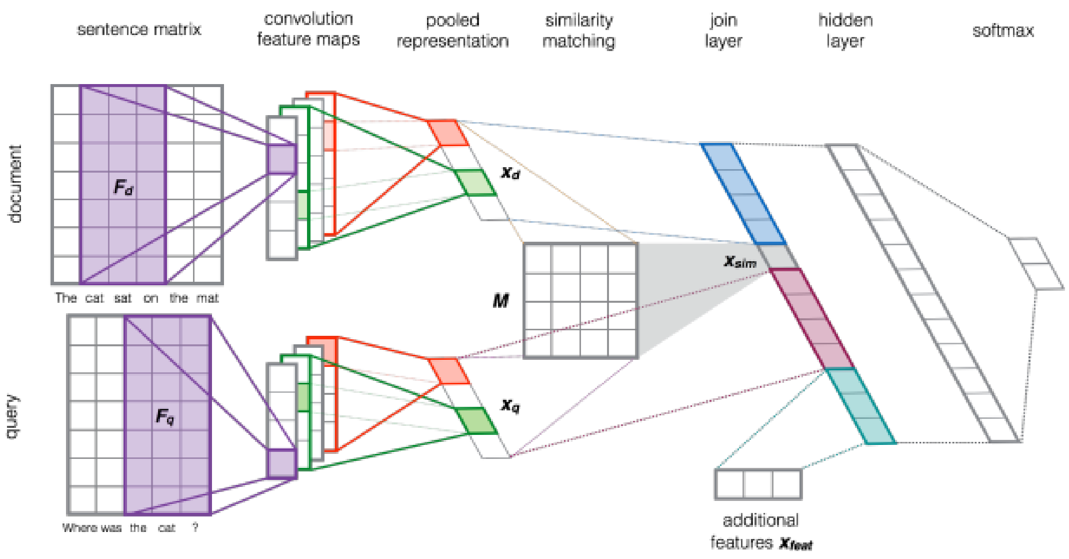


Figure 4.1: The CNN model from Severyn and Moschitti [2015b]

Model for Task A: question-comment similarity

For Task A, we trained two encoders, i.e. $f_{q_{rel}} : Q \rightarrow \mathbb{R}^n$ and $f_{c_{rel}} : C \rightarrow \mathbb{R}^n$, to encode both a forum question q_{new} and a comment c_{rel} into two dense vector representations. Also, we compute word overlap embeddings between q_{rel} and c_{rel}, encoding the relational information between the input sentences and concatenate them to the word embeddings. Then, we concatenated the representations of the question and the comment into the join layer, Teh join layer makes possible to inject additional features (see x_{feat} in Figure 4.1) in the network that can useful for solving the task. Finally, the output of the join layer is fed to a hidden layer and then passed to multi-layer perceptron with sigmoid activation. The latter produces a relevancy score of the answer comment with respect to the question. We trained the network by minimizing the binary cross entropy loss.

Model for Task B: question-question similarity

To model question-question similarity, we modified the original S&M network such that instead of using two separate sentence encoders, i.e. f_{q_1} and f_{q_2}, it only uses one $f_q : Q \rightarrow \mathbb{R}^n$. This way, f_q is used for encoding both questions q_{new} and q_{rel}. The final network architecture contains two *identical* sub-networks that returned the same representation for the same input sentence. Since Task B concerns re-ranking questions initially ranked by Google, a strong baseline is given by the Google rank. Thus, we decide to encode the Google rank by discretizing the rank values in different bins of different sizes, i.e. $[1-2]$, $[2-5]$, $[5-10]$. Also here, we added word overlaps between q_{new} and q_{rel} as an embedding vector $x_{overlap}$ of size 5. Also here, we added word overlap between the two questions. Then, we concatenate the representations of q_{new} and q_{rel} returned by the sentence encoder f_q. The output of the join layer is then fed to a hidden layer, whose output is passed to a multi-layer perceptron with sigmoid activation. The latter produces a similarity score between two questions. For training the network, we minimized the binary cross-entropy loss as objective function.

Model for Task C: new question-comment similarity

In the model for Task C, we trained two separate sentence encoder, i.e. $f_{q_{new}} : Q \to \mathbb{R}^n$ and $f_{c_{rel}} : C \to \mathbb{R}^n$ for mapping the new out-of-forum question and the answer comment into two distinct fixed-size dense vectors. Similarly to the original S&M network, the two encoders are kept separate, in order to model the different nature of question and answer comments. As for the other tasks, we added word overlaps between q_{new} and c_{rel} together with a rank embedding. The latter are inherited from the rank of the related forum question thread in which the comment appears.The intuition behind this is the following: answer comments for highly-related question in the top of the Google ranking should be scored higher than answers for questions in the bottom. Finally, these features are concatenated to the join layer with the distributed representations of the sentences. The join layer is fed to a multilayer perceptron, which produces a relevancy score between a new out-of-forum question q_{new} and a forum comment c_{rel}. Also in this case, during training, we minimized the binary-cross entropy loss.

4.1.3 Results of individual models and Discussions

In Table 4.1 we show the results of individual neural models for all tasks, in comparison with the Random and Information Retrieval baselines of the challenge (first grouped row) and the three-top systems of SemEval 2016, KeLP [Filice et al., 2016a], UH-PRHLT [Franco-Salvador et al., 2016], SUper-team [Mihaylova et al., 2016] (second grouped row). The third grouped row shows the performance of the individual models when trained on input pairs $\langle q_{rel}, c_{rel}\rangle$, $\langle q_{new}, q_{rel}\rangle$ and $\langle q_{new}, c_{rel}\rangle$. The model for the three tasks are the one described in the previous section. (Figure 4.1). These results show that the individual models can generalize well enough on all the tasks. However, as it can be seen from the Table, the results lie far behind the state of the art obtained by Tree Kernel-based systems, i.e. ConvKN and Kelp. More in detail, our model for Task A scored 5.24 absolute MAP points lower than the top-performing model Kelp. Similarly, our model for Task B performed 3 MAP points lower than best system UH-PRHLT. In this case, the network can barely approach the Google rank, which is a strong baseline to beat, even for systems top systems (UH-PRHLT) using sophisticated hand-engineered features. However, the largest gap in the results we obtained is on Task C, where we got 13.46 points lower than the best model (Super-team). One reason that may help to understand why our models have worse performances compared to the top systems in the competition is that neural networks suffer from the data scarcity problem. Although these models have been successfully applied to many text classification tasks [Goldberg, 2015] thanks to their capacity of automatically engineering features and achieve start-of-the-art performances, they still require a fairly large amount of training data compared to other machine learning approaches. This is even true when they are trained for solving high-level semantic tasks such as QA [Yu et al., 2014], for which, more traditional methods achieve comparable or even higher accuracy, e.g., [Tymoshenko et al., 2016b]. That's because neural models contain a huge set of parameters required for effectively modeling the interactions between the vector representations of words in order to solve the final task. In contrast, the lack of data results in poor generation of the trained model on new data. To solve this issue, in the next section, we present a new deep learning architecture that can alleviate the burden of data scarcity problem.

Models	Task A: question-comment similarity				Task B: question-question similarity				Task C: new question-comment similarity			
	DEV		TEST		DEV		TEST		DEV		TEST	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
Random	-	-	59.53	67.83	-	-	46.98	50.96	-	-	15.01	15.19
IR Baseline	-	-	52.80	58.71	71.35	76.67	74.75	83.79	-	-	40.36	45.83
Kelp	-	-	79.19	86.42	-	-	-	-	-	-	-	-
UH-PRHLT	-	-	-	-	-	-	76.70	83.02	-	-	-	-
SUper-team	-	-	-	-	-	-	-	-	-	-	55.41	61.48
$\langle q_{rel}, c_{rel} \rangle$	68.93	76.46	74.73	81.18	-	-	-	-	-	-	-	-
$\langle q_{new}, q_{rel} \rangle$	-	-	-	-	74.19	**83.26**	**73.70**	**82.13**	-	-	-	-
$\langle q_{new}, c_{rel} \rangle$	-	-	-	-	-	-	-	-	44.77	52.07	41.95	47.21

Table 4.1: Results on the validation and test set for the proposed models

4.2 Joint model

In this section, we introduce a new multi-task learning (MTL) architecture for solving the overall cQA task, i.e. given a new fresh out-of-forum question, predict if a comment produced in response to a previous forum question represents a valid answer . In order to overcome the data scarcity problem, our architecture exploit auxiliary tasks that are highly

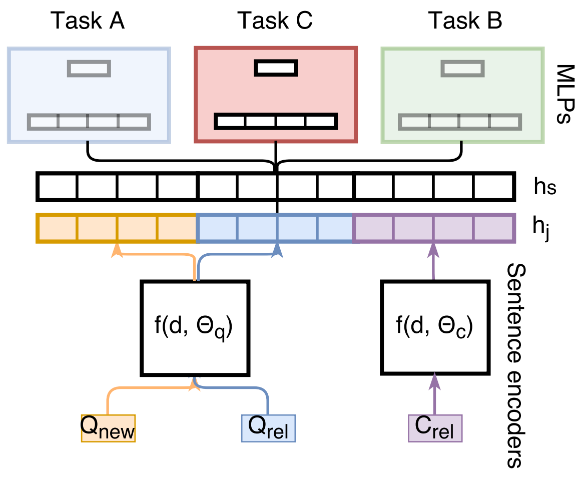


Figure 4.2: Our MTL architecture for cQA. Given the input sentences q_{new}, q_{rel} and c_{rel} (at the bottom), the NN passes them to the sentence encoders. Their output is concatenated into a new vector, h_j, and fed to a hidden layer, h_s, whose output is passed to three independent multi-layer perceptrons. The latter produce the scores for the individual tasks.

semantically connected with the main task. More in particular, we exploit the strong semantic connections between selection of comments relevant to (i) new questions and (ii) forum questions, which enable our model to learn global representations for comments, new and previous questions. The experiments of our model on the SemEval-2016 challenge dataset for cQA show a 20% relative improvement over standard deep neural networks (DNNs).

4.2.1 Related Work on Multi-Task learning (MTL) for NNs

Finding a general solution to the data scarcity issue is still an an open problem. However, for some class of applications, the problem can be alleviated by making use of multitask learning (MTL) [Caruana, 1997]. Recent work has shown that it is possible to *jointly train* a general DNN system for solving different tasks simultaneously. For example, Collobert et al. [2011b] show that MTL can be used to train a single neural network for carrying out many sequence labeling tasks (e.g., pos-tagging, name entity recognition, etc.), whereas Liu et al. [2015] trained a DNN with MTL to perform multi-domain query classification and reranking web search results with respect to user queries. This resulted in improved prediction accuracy for the task-specific models when compared to the models trained separately.

4.2.2 Our MTL model for cQA

MTL aims at learning several related tasks at the same time to improve some (or possibly all) tasks using joint information [Caruana, 1997]. In this section, we show how to apply MTL to model semantically connected tasks to the purpose of obtaining a larger improvements on the final cQA task. In our work, we found MTL particularly well-suited for modeling Task C as it is a composition of tasks A and B. Thus, it can benefit from having both questions q_{new} and q_{rel} in input to better model the interaction between the new question and the comment. More precisely, it can use the triplets, $\langle q_{new}, q_{rel}, c_{rel} \rangle$, in the learning process, where the interaction between triplet members is exploited during the joint training of the three models for the tasks A, B and C. In fact, a better model solving the auxiliary tasks, i.e. (i) question-comment similarity and (ii) question-question similarity, can lead to a more accurate model for new question-comment similarity (Task C).

Additionally, the SemEval dataset is organized in threads, and each of them is annotated with the labels for all the three tasks. Therefore, it is possible to apply joint learning directly (using a global loss), rather than training the network by optimizing the loss of the three single tasks independently as in previous works [Collobert et al., 2011b; Liu et al., 2015]. There, each example was annotated for only one task and, thus, training the model required to alternate examples from the different tasks.

Join Learning Architecture

Our joint learning architecture is depicted in Figure 4.2. It takes three pieces of text as input, i.e. a new question, q_{new}, the related question, q_{rel}, and its comments, c_{rel}, and produces three fixed sized representations $x_{q_{new}}$, $x_{q_{rel}}$ and $x_{c_{rel}}$ respectively. This process is performed using the sentence encoders $x_d = f(d, \theta_d)$, where d is the input text and σ_d is the set of parameters of the sentence encoders. In previous work, different sentence encoders have been proposed, e.g., CNNs with max-pooling [Kim, 2014; Severyn and Moschitti, 2015b], and Long-short term memory (LSTM) networks [Hochreiter and Schmidhuber, 1997]. Here, we concatenate the three representations $h_j = [x_{q_{new}}, x_{q_{rel}}, x_{c_{rel}}]$ and we fed them to a hidden layer to create a shared input representation for the three tasks, $h_s = \sigma(Wh_j + b)$. Next, we connect the output of h_s to three independent Multi-Layer Perceptrons (MLP), which produce the scores for the three tasks. At training time, we compute the global loss as the sum of the individual losses for the three tasks for each example, where each loss is computed as binary cross-entropy.

Model	MAP	MRR
LSTM	43.91	49.28
CNN	44.43	49.01
CNN Train	44.43	49.01
CNN Train + ED[1]	**44.77**	**52.07**

Table 4.2: Impact of CNN vs. LSTM sentence models on the baseline network for Task C.

Shared Sentence Models

By construction, the SemEval dataset contains ten time less new questions q_{new} than related questions q_{rel}. However, the two types of questions have the same nature (i.e., generated by forum users), thus we can share the parameters of their sentence models as depicted in Figure 4.2. Formally, let $x_d = f(d, \theta)$ be a sentence model for a text, d, with parameters, θ, i.e., the embeddings weights and the convolutional filters. In a standard setting, each sentence model uses a different set of parameters $\theta_{q_{new}}$, $\theta_{q_{rel}}$ and $\theta_{c_{rel}}$ In contrast, we proposed sentence models that encode both the questions q_{new} and q_{rel} that use the same set of parameters θ_q.

4.3 Experiments

4.3.1 Experiments of individual models

In this section, we describe the experimental setting used for evaluating the performance of the joint model on all the three tasks and report the obtained results.

Dataset: We used the same data already introduced in Chapter 3 and provided by the organizers of the SemEval-2016 challenge on cQA. Each of the three datasets is in turn divided in training, dev. and test sets. The label distribution of the different datasets is reported in Table 4.4. As it can be seen, the data for Task C presents a higher number of negative than positive examples. Usually, when trained on such dataset, a typical classifier that maximizes accuracy is likely to learn a model that will label all examples as negative. Hence, will perform poorly in terms of Precision and Recall. Addressing the problem of class-imbalance data, where the number of negative examples is much larger than the number of positive examples, is very important. Thus, we automatically extended the set of positive examples in our join MTL training set using the data from Task A. More specifically, we take pairs (q_{rel}, c_{rel}) from training set of Task A and create the triples (q_{rel}, q_{rel}, c_{rel}), where the label for question-question similarity (Task B) are

[1]Extended Dataset for Task C computed using questions from Task A.

obviously positive, whereas the labels for Task C are inherited from those of Task A. We ensure that the questions in the extended dataset do not overlap with questions from dev. and test sets. The resulting training data contains 34, 100 triples: its relevance label distribution is shown in the row, Train + ED, of Table 4.4[2].

Pre-processing: we tokenized both questions and comments in lowercase. In addition, we concatenated the question subject and body to create a unique text. For computational reasons, we limited the document size to 100 words. This did not cause degradation in accuracy.

Neural Networks: we mapped words to embedding of size 50, pre-initializing them with standard skipgram embedding of dimensionality 50. The latter embeddings were trained on the English Wikipedia dump using word2vec toolkit [Mikolov et al., 2013]. We encoded the input sentence with a fixed-sized vector, whose dimension are 100, using a convolutional operation of size 5 and a k-max pooling operation with $k = 1$. Table 4.2 shows the results of our preliminary experiments with the sentence models of CNN and LSTM, respectively, on the dev. set of Task C. In our further experiments, we opted for CNN since it produced better MAP and is computationally more efficient. For each MLP, each one producing the scores for the three tasks, we used a non-linear hidden layer (with hyperbolic tangent activation, tanh), whose size is equal to the size of the previous layer, i.e., 100. We included information such as word overlaps [Tymoshenko et al., 2016b] and rank position as embeddings with additional lookup table with vectors of size $d_{feat} = 5$. The rank feature is provided in the SemEval dataset and described the position of the question/comments in the search engine output.

Training: We trained our network using SGD with shuffled mini-batches using the rmsrop update rule [Tieleman and Hinton, 2012]. We set the training to iterate until the validation loss stops to improve, with patience $p = 10$, i.e., the number of epochs to wait before early stopping, if no progress on the validation set is obtained. We added dropout [Srivastava et al., 2014] between all layers of the network to improve generalization and avoid co-adaptation of features. We tested different dropout rates (0.2, 0.4) for the input and (0.3, 0.5, 0.7) for the hidden layers obtaining better results with the highest values, i.e., 0.4 and 0.7.

Results of individual models with joint input

Table 4.3 shows the results of our individual and MTL models on all the three tasks. Also in this case, we report the Random and Information Retrieval baselines of the challenge (first grouped row), and the three-top systems of SemEval 2016, KeLP Filice et al. [2016a], UH-PRHLT Franco-Salvador et al. [2016], SUper-team Mihaylova et al. [2016] (second

[2]MTL data available at `http://ikernels-portal.disi.unitn.it/repository/`

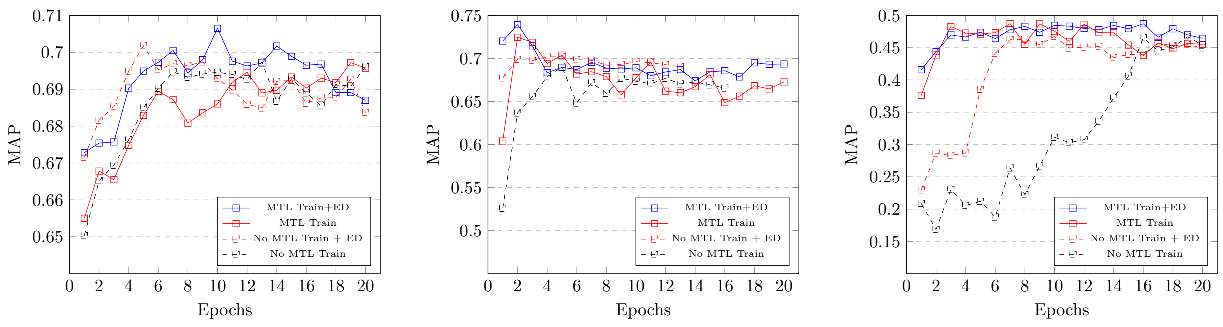
Models	Task A: question-comment similarity				Task B: question-question similarity				Task C: new question-comment similarity			
	DEV		TEST		DEV		TEST		DEV		TEST	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
Random	-	-	59.53	67.83	-	-	46.98	50.96	-	-	15.01	15.19
IR Baseline	-	-	52.80	58.71	71.35	76.67	74.75	83.79	-	-	40.36	45.83
Kelp	-	-	79.19	86.42	-	-	-	-	-	-	-	-
UH-PRHLT	-	-	-	-	-	-	76.70	83.02	-	-	-	-
SUper-team	-	-	-	-	-	-	-	-	-	-	55.41	61.48
$\langle q_{rel}, c_{rel} \rangle$	68.93	76.46	74.73	81.18	-	-	-	-	-	-	-	-
$\langle q_{new}, q_{rel} \rangle$	-	-	-	-	74.19	**83.26**	**73.70**	**82.13**	-	-	-	-
$\langle q_{new}, c_{rel} \rangle$	-	-	-	-	-	-	-	-	44.77	52.07	41.95	47.21
$\langle q_{new}, q_{rel}, c_{rel} \rangle$	-	-	-	-	-	-	-	-	45.59	51.04	46.99	55.64
$\langle q_{new}, q_{rel}, c_{rel} \rangle + \leftrightarrow$	70.69	77.19	**75.52**	82.11	72.92	80.20	72.88	80.58	47.82	53.03	46.45	51.72
MTL (BC)	-	-	-	-	**74.22**	80.40	73.68	81.59	47.80	52.31	48.58	**55.77**
MTL (AC)	70.11	76.50	75.43	**82.46**	-	-	-	-	46.34	51.54	48.49	54.01
MTL (ABC)	69.93	76.27	74.42	81.68	70.68	75.85	71.07	80.11	**49.63**	**55.47**	49.87	55.73
MTL (ABC)*	**70.70**	**77.48**	74.89	81.80	74.21	81.93	72.23	80.33	**49.63**	**55.47**	49.87	55.73
MTL (weighted score)	-	-	-	-	-	-	-	-	-	-	**52.67**	55.68

Table 4.3: Results on the validation and test set for the proposed models.

grouped row). The fourth grouped row of Table 4.3 illustrates the models exploiting the joint input, $\langle q_{new}, q_{rel}, c_{rel} \rangle$, but no joint learning is carried out, i.e., the networks for the different tasks are trained individually. The results show that a small degradation of performance happens in Task B, while Task A slightly improves. These variations may be due to the fact that tasks A and B can be efficiently solved using the standard pairwise approach, thus the extra text introduced in the model may just add some noise. However, using the shared sentence model for q_{new} and q_{rel} of the tasks B and C (indicated with $\leftrightarrow$) improves the overall performance.

4.3.2 Results of MTL models

The shared input representations show good results on all tasks, thus, in the last experiments, we jointly trained (i) tasks B and C, (ii) tasks A and C and finally (iii) the thee tasks together. The results are reported in the fifth grouped row of Table 4.3. The MTL architecture improves the performance in terms of MAP by 2 absolute points on DEV set and by 3 absolute points on the TEST set for Task C, while the performance on the other tasks tends to degrade. This is due to the fact that in the MTL setting, the models are not optimized on the outcome of a single task and so the individual objective functions converge at different epochs. Therefore, when the global loss reaches the



(a) Task A (b) Task B (c) Task C

Figure 4.3: Learning curves for all tasks on the dev. set; dotted and solid lines represent the individual and multi-task models, respectively.

	Task A	Task B	Task C
Train	37.51%	39.41%	9.9%
Train + ED	37.47%	64.38%	21.25%
Dev	33.52%	42.8%	6.9%
Test	40.64%	33.28%	9.3%

Table 4.4: Percentage of positive examples in the training datasets for each task.

minimum, it is possible that individual models are sub-optimal. Indeed, the comparison between the learning curves (on the development set) for Task B (Figure 4.3b) and Task C (Figure 4.3c) show that for the former, models achieve earlier convergence rate (epoch 2) while, for the latter, they converge later (epoch 16). Moreover, Figure 4.3a shows that the results on Task A are not badly affected by jointly training models with the other two tasks. Finally, the learning curves show that our networks trained in MTL tend to have faster convergence rate than the individual models: this is a very interesting result. We also experimented with shallower networks and SVMs using the predictions scores from the different classifiers in a stacking approach, and obtained results far below the baselines.

4.3.3 Results on the overall Task C

Although the findings for Task A ad Task B are very interesting, we are mainly interested on the final results on the overall cQA task. Table 4.5 shows the performances of our individual and MTL models on Task C, compared with Random and IR baseline of the challenge (first two rows), and the SemEval 2016 (row 3–12). Rows 13-15 illustrate the results of our models when trained only on Task C. $\langle q_{new}, c_{rel} \rangle$ corresponds to the basic

Model	DEV		TEST	
	MAP	MRR	MAP	MRR
Random	-	-	15.01	15.19
IR Baseline	-	-	40.36	45.83
SUper-team	-	-	55.41	61.48
KeLP	-	-	52.95	59.23
SemanticZ	-	-	51.68	55.96
MTE-NN	-	-	49.38	51.56
ICL00	-	-	49.19	53.89
SLS	-	-	49.09	55.98
ITNLP-AiKF	-	-	48.49	55.21
ConvKN	-	-	47.15	51.43
ECNU	-	-	46.47	51.41
UH-PRHLT	-	-	43.20	47.79
$\langle q_{new}, c_{rel} \rangle$	44.77	52.07	41.95	47.21
$\langle q_{new}, q_{rel}, c_{rel} \rangle$	45.59	51.04	46.99	55.64
$\langle q_{new}, q_{rel}, c_{rel} \rangle + \leftrightarrow$	47.82	53.03	46.45	51.72
MTL (BC)	47.80	52.31	48.58	55.77
MTL (AC)	46.34	51.54	48.49	54.01
MTL (ABC)	**49.63**	55.47	**49.87**	55.73
MTL + one feature	-	-	**52.67**	55.68

Table 4.5: Results on the validation and test set for the proposed models

model, i.e., the single network, whereas the $\langle q_{new}, q_{rel}, c_{rel}$ model only exploits the joint input, i.e., the availability of q_{rel}. Rows 16-18 report the MTL models combining Task C with the other two tasks. The difference with the previous group (rows 13-15) is in the training phase, which also operated on the instances from tasks A and B. As it can be noted (i) the single network for Task C cannot compete with the challenge systems, as it would be ranked at the last position, according to the official MAP score (test set results); (ii) the joint representation, $\langle q_{new}, q_{rel}, c_{rel} \rangle$ highly improves the MAP of the basic network from 41.95 to 46.99 on the test set. This confirms the importance of using same input encoders for encoding objects with closely related semantics and solving highly related tasks. (iii) The shared sentence model for q_{new} and q_{rel} (indicated with $\leftrightarrow$) improves MAP on the dev. set only. (iv) The MTL (ABC) provides the best MAP, improving BC and AC by 1.29 and 1.38 absolute points, respectively. Most importantly, it also outperforms $\langle q_{new}, q_{rel}, c_{rel} \rangle$ by 2.88 points, i.e. it improves the best model using the join representation and no training on auxiliary tasks. One important thing to consider is that our full MTL model would have ranked 4^{th} on Task C of the SemEval 2016 competition. This is an important result since all the other systems in the challenge made use of many manually engineered features whereas our model does not (except for the necessary initial rank). If we add the most powerful feature used by the top systems to our model, i.e. the weighted sum between the score of the Task A classifier and the Google Rank [Mihaylova et al., 2016] our system would achieve a MAP 52.67, i.e., very close to the second system.

4.3.4 Conclusions

In this section, we proposed an end-to-end architecture for solving the overall cQA task. We showed that our architecture can benefit from being trained in MTL setting on auxiliary tasks that are semantically connected with our main task. Our experiments on the dataset of SemEval 2016 Task 3 show that our MTL approach relatively improves the individual DNNs by almost 20%. Thanks to the shared representation as well as training on the instances of two auxiliary tasks, our network showed better accuracy a higher convergence rate than the models independently trained. Thanks to this, we could approach the performance of the models participating at the challenge.

4.4 Combining Neural and Kernel models for Task B

Despite the final performance of the joint model over the Task A and C considerably increased, the Task B did not benefit much from MTL. In this section, we propose a new method to fill the accuracy gap in question-question similarity between DNNs and top-performing SemEval models based on structural representations. Our approach is aimed at injecting syntactic information into a DNN model, useful in case training data is scarce. Effectively using syntactic parsing information in neural networks for relational text inference tasks such as question-question similarity is still an open problem. Previous research, [Linzen et al., 2016], shows that recurrent sequence models conceived for learning long dependencies are really effective only when a sufficient amount of supervision is provided to them, which in turn requires additional data. In this section we propose a solution aiming at injecting structural representations in NNs by (*i*) learning an SVM model using Tree Kernels (TKs) on relatively few pairs of questions (few thousands) as gold standard (GS) training data is typically scarce, (*ii*) predicting labels on a very large corpus of question pairs, and (*iii*) pre-training NNs on such larger corpus. We test our approach on Quora and SemEval question-similarity datasets and show that NNs trained with our approach can learn more accurate models, especially after fine tuning on GS.

4.4.1 Related Work

In recent years, both academia, e.g. SemEval [Nakov et al., 2016a, 2017] or companies, e.g. Quora[3], Alibaba[4], have proposed to build automatic systems for detecting duplicate questions. An interesting outcome of the SemEval challenge discussed in Chapter 3 is that syntactic information is essential to achieve high accuracy in question reranking tasks. Indeed, top-systems winning the competition used with structural kernels, which were applied to syntactic representation of question text [Filice et al., 2016b, 2017]. In chapter 3, we saw that SVMs are very effective in presence of datasets with few training examples, as in the case of the SemEval dataset. However, their inherent time complexity make them not very practical for training systems on large datasets. On the contrary, neural networks are fast compared to kernels. Thus, researchers start to wonder how to combine kernels with the neural networks in order to train state-of-the-art systems even in cases data is scarce. One viable solution seems to provide neural networks with additional information, such as syntactic information (modeled by structural Kernels), that can help to solve the task. Exploiting syntactic information in neural networks is a topic of ongoing debate. In recent years, recursive Neural Networks models such as Tree-LSTM Socher et al. [2013]; Tai et al. [2015] have been proposed to overcome this limitation and exploit syntactic information. However, subsequent research showed that such models can be outperformed by well-trained sequential models Li et al. [2015]. One interesting piece of work is the one by Hu et al. [2016], who tried to combine symbolic representations with NNs by transferring structured information of logic rules into the weights of NNs. However, our aim is rather different as we re interested in injecting syntactic, and not logic, information in NNs. The work most similar to our is the one by Croce et al. [2017], who use Nystrom methods to compact the TK representation in embedding vectors and use the latter to train a feed forward NNs. In contrast, we present a simpler approach, where NNs learn syntactic properties directly from data.

4.4.2 Overview/Introduction

Here, we describe our approach that aims at injecting syntactic information in NNs, but trying to keep the network architecture simple. It consists of the following steps: (*i*) train a TK-based model on a few thousands training examples; (*ii*) apply such classifier to a much larger set of unlabeled training examples to generate automatic annotation data; (*iii*) pretrain NNs on automatic data; and (*iv*) fine-tune on the GS data. We show that the performance of our model improved only when a very different classifier, i.e.,

[3]`https://www.kaggle.com/c/quora-question-pairs`

[4]`https://102.alibaba.com/detail/?id=115&mtime=1528166091000`

TK-based, are used to label additional data. Conversely, when using the same NN in a self-training fashion to label data, the procedure does not provide any improvement. At the same time, when SVM use standard similarity lexical features – without kernel on syntactic trees – no improvements are observed. The use of syntactic information is at the core of TKs-based models. Although further investigation is need to assess that NNs specifically learn syntax, the fact that only the transfer from TKs produces improvement is a significant evidence that injecting syntax information in NNs is a viable research direction worth to explore.

4.4.3 Injecting Structure in NNs

To inject structured information in a neural network, we employed a weak supervision technique:

- a classifier, i.e., Support Vector Machine with Tree Kernels, is trained on the original dataset.
- external data is classified using this classifier, creating *weakly supervised* data.
- a neural network is trained on such automatically labeled data to mimic the original classifier.

We experiment with the weak supervision technique using three different classifiers: the network itself (NN), an SVM over feature vectors (FV) and SVM with Tree Kernel (TK). The pre-trained network can be further refined on the original data. This fine tuning phase is usually performed using a smaller learning rate γ. The main reason to reduce the value of γ in fine tuning is to avoid catastrophic forgetting [Goodfellow et al., 2013] when training with a high learning rate.

SVM Vector Machines

As first thing, we learn a scoring function $r : \mathcal{Q} \times Q \rightarrow \{0, 1\}$ telling if two questions are similar or not. The function were learned on the training sets of Quora and SemEval containing gold-standard (GS) annotated data. These functions can also be used to rerank a set of forum question q_{rel} with respect to a new question q_{new} based on their similarity. In our experiments, we provide two implementations of the question-similarity function, e.g.: (i) a linear function on the feature vector representations of two questions; (ii) a kernel function applied to the syntactic structure of questions in a pair.

Feature Vector model: This model relies on a set of *text similarity* features that capture the relationships between two questions. More specifically, we compute a total of

20 similarities $sim(\langle q_1, q_2 \rangle)$ using word n-grams (n = [1,. . . ,4]) after stopwords removal, using greedy string tiling [Wise, 1996], longest common subsequences [Allison and Dix, 1986], Jaccard coefficient [Jaccard, 1901], word containment [Lyon et al., 2001], and cosine similarity.

Tree Kernel model: This model uses tree kernel functions to measure the similarity between the syntactic structures of two questions. We used the same representations described in Chapter 3 for modeling question pairs. We build two macro-trees, one for each question in the pair, containing the syntactic trees of the sentences within a question. In addition, we link two macro-trees by connecting the phrases, e.g. NP, VP, PP, etc., when there is a lexical match between the phrases of two questions. Then, we apply Subset Tree Kernel and obtain the following kernel: $K(\langle q_1, q_2 \rangle)^i, \langle q_1, q_2 \rangle^j = TK(t(q_1^i, q_1^j), t(q_2^i, q_2^j)) + TK(t(q_1^j, q_1^i), t(q_2^j, q_2^i))$, where $t(x, y)$ extract the syntactic tree from the text x, enriching it with REL tags.

NNs for question similarity

We inject syntactic knowledge into two well-known state-of-the-art networks for question similarity, enriching them with relational information. Then, we used our procedure for injecting TK-knowledge into the NN model. First, we implemented the Convolutional NN (CNN) model proposed by Severyn and Moschitti [2016] and described in Chapter 3. The model learns a similarity function between two sentences $f : Q \times Q \to (0, 1)$, using two separate sentence encoders $f_{q_1} : Q \to \mathbb{R}^n$ and $f_{q_2} : Q \to \mathbb{R}^n$, which map each question into a fixed size dense vector of dimension n. The resulting vectors are concatenated and passed to a Multi Layer Perceptron that performs the final classification. Each question is encoded into a fixed size vector using an embedding layer, a convolution operation and a global max pooling function. The embedding layer transforms the input question, i.e., a sequence of tokens, $X_q = [x_{q_1}, \cdots, x_{q_i}, \cdots, x_{q_n}]$, into a sentence matrix, $S_q \in \mathbb{R}^{m \times n}$, by concatenating the word embeddings w_i corresponding to the tokens x_{q_i} in the input sentence.

As a second model, we implemented a Bidirectional (BiLSTM) Graves et al. [2013, 2005], using the standard LSTM by Hochreiter and Schmidhuber [1997]. An LSTM iterates over the sentence one word at the time by creating a new word representation h_i by composing the representation of the previews word and the current word vector $h_i = \text{LSTM}(w_i, h_{i-1})$. A BiLSTM iterates over the sentence in both directions and the final representation is a concatenation of the hidden representations, h_N obtained after processing the whole sentence. We apply two sentence models (with different weights), one for each question, then we concatenate the two fixed-size representations and fed to a Multi-Layer Perceptron.

Relational Information

As in Severyn and Moschitti [2016] we include relational information by means of word overlap feature embeddings. We show that providing such information can highly improve accuracy. Thus, for both networks above, we mark each word with a binary feature indicating if a word from a question appears in the other pair question. This feature is encoded with a fixed size vector.

Learning NNs with structure

At this point, to inject structured information in the network: (*i*) we train SVM with TKs on GS data; (*ii*) use this model to classify additional unlabeled data, creating automatic data; and (*iii*) train a neural network on the latter data. Finally, the pre-trained network is fine-tuned on the GS data.

4.4.4 Experiments

We experiment with two datasets comparing models trained on gold and automatic data and their combination, before and after fine tuning.

Data

- **Quora dataset:** contains $384,358$ pairs in the training set and $10,000$ pairs both in the dev. and test sets. The latter two are balanced and contain the same number of positive and negative examples.
- **QL dataset:** contains $3,869$ question pairs divided in $2,669$, 500 and 700 pairs in train, dev. and test sets. We created 93k unlabeled pairs from the QL dump[5], retrieving 10 candidates with Lucene for $9,300$ query questions.

NN setup

As input features for our network, we use pre-initialized word embedding of dimensionality 50 jointly trained on English Wikipedia dump [Mikolov et al., 2013] and the jacana corpus[6]. The input sentences are encoded with fixed-sized vectors using a CNN with the following parameters: a window of size 5, an output of 100 dimensions, followed by a global max pooing. We use a single non-linear hidden layer, whose size is equal to the size

[5]In the context of SemEval-2016 challenge, the organizers released a large unannotated dataset from Qatar Living with 189,941 questions and 1,894,456 comments.

[6]Embeddings are available in the repository: `https://github.com/aseveryn/deep-qa`

Model	Automatic data	GS data	DEV	TEST
FV-10k	–	10k	0.7046	0.7023
TK-10k	–	10k	0.7405	0.7337
CNN-10k	–	10k	0.7646	0.7569
LSTM-10k	–	10k	0.7521	0.7450
CNN(CNN-10k)	50k	–	0.7666	0.7619
CNN(CNN-10k)*	50k	10k	0.7601	0.7598
CNN(FV-10k)	50k	–	0.6960	0.6931
CNN(FV-10k)*	50k	10k	0.7681	0.7565
CNN(TK-10k)	50k	–	0.7446	0.7370
CNN(TK-10k)*	50k	10k	0.7748	0.7652
LSTM(TK-10k)	50k	–	0.7478	0.7371
LSTM(TK-10k)*	50k	10k	0.7706	0.7505
TK-5k	–	5k	0.6859	0.6774
CNN-5k	–	5k	0.7532	0.7450
CNN(TK-5k)	50k	–	0.7239	0.7208
CNN(TK-5k)*	50k	5k	0.7574	0.7493
CNN(TK-10k)	375k	–	0.7524	0.7471
CNN(TK-10k)*	375k	10k	**0.7796**	**0.7728**
Voting(TK+CNN)	–	10k	0.7838	0.7792

Table 4.6: Accuracy on the Quora dataset.

of the sentence embeddings, i.e. 100. The word overlap embedding is set to 5 dimensions. The activation function for both convolution and hidden layers is ReLU. The model is trained by optimizing the binary cross-entropy loss. More in particular, we used SGD with Adam update rule, setting the learning rate γ to 10^{-4} and 10^{-5} for the pre-training and fine-tuning phase.

Results on Quora

Table 4.6 reports the results of our different models, FV, TK, CNN and LSTM described in the previous section. The suffixes -10k or 5k indicate the amount of GS data used to train them, and the name in parenthesis indicates the model used for generating data, e.g. CNN(TK-10k) means that a CNN has been pre-trained with the data labeled by a TK model trained on 10k GS data. The amount of data for pre-training is reported in second column, while the amount of GS data for training or fine tuning (indicated by *) is in the third column. Finally, the results on the dev. and test sets are in the fourth and fifth columns. We can observe that: first, NN trained on 10k of GS data obtain higher accuracy than FV and TKs on both dev. and test sets (see the first four lines); Second, CNNs pre-trained with the data generated by FV or in a self-training setting, i.e., CNN(CNN-19k), and also fine-tuned do not improve[7] on the baseline model, i.e., CNN-10k, (see the second part of the table). Third, when CNNS and LSTMs are trained on data labeled by TK model, they match the TK model accuracy (third part of the table). Most importantly, when they are fine-tuned on GS data, they obtain better results than the original models trained on the same amount of data, e.g., 1% accuracy over CNN-10k. Next, in the fourth part of the table we can see that our method gives an improvement when training TK (and fine-tuning the NNs) on less GS data, i.e., only 5k. Additionally, in the fifth section of the table we show high improvements by training NN on all available Quora data annotated by TK-10k (Tree kernel-based model trained on just 10k examples). This seems to suggest that NNs require more data to learn complex relational syntactic patterns expressed by TKs. However, the plot in Figure 4.4 shows that the improvement reaches a plateau around 100k examples. Finally, in the last row of the table, we report the result of a voting approach using a combination of the normalized scores of TK-10k and CNN-10k. The accuracy is almost the same than CNN(TK-10k)*. This shows that NNs completely learn the combination of a TK model, mainly exploiting syntax, and a CNN, only using lexical information. It must be noted that the voting model is heavy to deploy as it uses syntactic parsing and the kernel algorithm, which has a time complexity quadratic in the number of support vectors. Thus, our solution provides a considerable speedup in terms of time over the voting model.

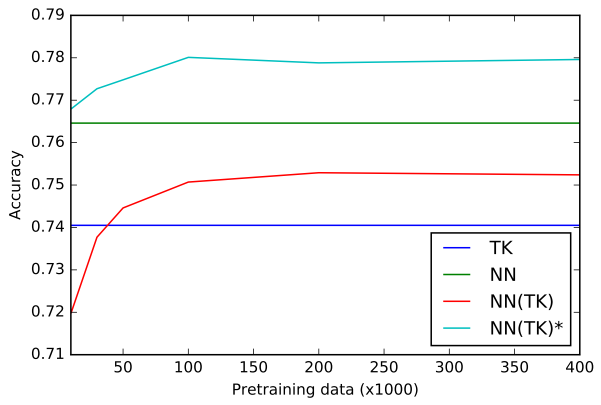


Figure 4.4: Impact of the pre-training data.

[7]The improvement of 0.5 is not statistically significant.

Model	Automatic Data	Dev	Dev (MAP)	Test	Test (MAP)
CNN		0.7000	0.6598	0.7514	0.7208
TK		0.7340	**0.6988**	0.7686	0.7424
CNN(TK)	50k	0.5580	0.6578	0.5428	0.7370
CNN(TK)*	50k	0.7160	0.6794	**0.7814**	0.7312
CNN(TK)	93k	0.7000	0.6782	0.6957	**0.7430**
CNN(TK)*	93k	**0.7380**	0.6782	0.7614	0.7320

Table 4.7: Accuracy on QL using all available GS data.

Results on Qatar Living

Table 4.7 reports the results when applying our technique to a smaller and differently dataset such as QL. As can be seen in the first grouped row, CNNs have lower performance than TK models as 2,669 pairs are not enough to train their parameters, ant the text is also rather noisy, i.e. there are a lot of spelling errors. Despite this problem, the results show that CNNs can approximate the TK models quite well, when using a large set of automatic data. For example, the CNN trained on 93k automatically annotated examples and then fine tuned exhibits 0.4% accuracy improvement on the dev. set and almost 3% on the test set over TK models. On the other hand, using too much automatically labeled data may hurt the performance on the test set. This may be due to the fact that the quality of information contained in the gold labeled data deteriorates. In other words, the right amount of weekly-supervised data to use during training must be carefully chosen.

4.4.5 Conclusion

In this section, we have trained TK-based models, which make use of structural information, on relatively small data and applied them to new data to produce a much larger automatically labeled dataset. Our experiments show that NNs trained on automatic data improve accuracy. We may speculate that NNs somehow learn relational information as (*i*) TK models mainly used syntactic structures to label data and (*ii*) other advanced models based on similarity feature vectors, e.g. CNN(FV) and CNN(CNN), do not produce any improvement. However, even if our conjecture were wrong, the bottom line would be that thanks to our approach, we can have NN models comparable to TK-based approaches, by avoiding the use of syntactic parsing and expensive TK processing at deployment time.

Chapter 5

Supervised Clustering of questions for fast bootstrapping of Intent Ontologies

Modern NLP applications such as Conversation Agents and Dialog systems, need to classify user request into a predefined set of categories or *semantic* categories. For example, a dialog system would classify the utterance *"Turn on the lights in the kitchen"* into the semantic category `turn_light_on`. This task is called intent detection and is typically solved by using supervised methods to learn a classifier, able to automatically pair a user request with its corresponding intent. While these methods can deliver state-of-the-art results, they require a large amount of labeled data annotated by experts. Unfortunately, these approach become less applicable as the number of data and intents increase, since a larger amount of human labor is needed. Furthermore, their applicability is very limited, as they cannot generalize to novel *unseen* intents. Approaching this problem, sometimes referred as zero-shot user intent detection or zero-shot user intent learning, is of extreme importance for fast bootstrapping of Natural Language Understanding (NLU) pipelines needed by modern Dialog Systems. In this chapter, we present a new supervised approach for clustering similar questions (or user requests) into semantic clusters. These clusters represent sets of questions that correspond to the same intent. Our approach, which performs global inference on a graph of questions connected by their textual similarity, is able to group the elements into new user intents better than other strong baseline approaches. In our experiments, we trained a supervised clustering function on a smaller annotated part of the dataset and apply it to the remaining part. This allowed us to let the system discover the latent structure of intents in the data, freeing us from the burden to manually go though all the questions and making sense of them.

5.1 Overview

Recently, we have seen a renewed interest in dialog systems, ranging fro help desks to more complex task-based to general purpose conversational agents, e.g., Alexa, Cortana or Siri. When using these devices, users expect to formulate complex information needs in natural language, with no limitation to their expressiveness. Due to this, modern automated dialog systems require complex dialog managers able to understand *user intent* triggered by high-level semantic questions and expressed by articulated natural language text. Unfortunately, current solutions to the intent detection problem consists in manually analyzing user questions and creating a taxonomy of intents to be attached to the appropriate actions. For example, when designing a conversational interface for booking flight, several semantically similar/identical questions regard `BookFlight` are expected. Thus, the designer has to build a category for all these questions. As can be easily imagined, this is a rather costly, difficult and time consuming task, which make virtually impossible to quickly prototype dialog systems even for small domains. Unfortunately, very little work has been dedicated to automatizing the process of building intent ontologies starting from a set of questions. A reason can be the fact that the underlying problem, i.e. semantic question paraphrasing is very challenging. However, recent initiatives for automatic question duplicate detection[1], question relatedness Nakov et al. [2016b, 2017] and semantic textual similarity Agirre et al. [2012]; Cer et al. [2017] have shown that current technology can achieve good accuracy in matching short text expressing similar semantics.

5.2 Our solution

In this section, we describe our new model for automatically grouping a given set of questions into clusters representing new intents. By automatically discovering new intents, this model can provide important insights into the design of dialog systems. It does so by clustering questions into user intent categories, which can help the design of dialog systems. The main advantage of our approach is that, given a notion of intent, explicitly provided by annotated data, our model can create clustering driven by such intrinsic definition. Thus, this is one of our major contributions: providing an effective supervised clustering approach, which can learn definitions from examples. Our approach combined (i) state-of-the-art methods for question similarity/paraphrasing with (ii) powerful supervised clustering algorithms. The former are obtained by exploiting previous research, e.g., on Quora, whereas the latter, are obtained by employing the structured output ma-

[1]Quora: https://www.kaggle.com/c/quora-question-pairs

chine learning methods use for coreference resolution (CR), e.g. [Yu and Joachims, 2009; Fernandes et al., 2014]. In order to train our model, we define a clustering corpus by automatically deriving question clusters from pairs of duplicate questions in Quora. We did this by exploiting transitive closure of *semantic matching property* implied by question pair annotation. In addition, to test the applicability of our approach across languages and domains, we run evaluation experiments on another intent-based corpus, a collection of FAQs for an Italian online service.

5.2.1 Question clustering algorithms

The problem of clustering questions into user intents can be defined in many ways. Here, we provide a formalization of the problem: given a set of questions Q, we want to split them into subsets (clusters), $c_i = {q^i_j}_{j=1}^{N_i}$, where q^i_j is the j-th question in the cluster i of size N_i and $\sqcup_i c_i = Q$. Each c_i is assumed to contain questions with the same intent, i.e. to represent a distinct intent. Generally, what algorithm to adopt in a clustering task depends on the amount of supervision available. Thus, based on on this information, we can distinguish between two tasks: (i) unsupervised clustering and (ii) supervised clustering.

- **Unsupervised clustering**: Approaches for unsupervised clustering attempt to group elements together based on some identified commonalities. Clustering new sets of questions Q in an unsupervised way may be generally troublesome due to the lack of information about the structure of Q and the target number of distinct intents in it. To overcome this problem, we learn a new clustering function from data annotated with gold question clusters.

- **Supervised clustering**: In this work, we pose the task as a supervised clustering problem, according to the same formulation by [Finley and Joachims, 2005]. Given a set of training examples of the form $(x_i, y_i)_{i=1}^{n}$, where each input x_i is a set of elements of some nature and y_i - the corresponding gold standard clustering of such a set, the goal is to learn a predictor $h : X \rightarrow Y$ from the space of sets X to the space of clustering Y.

Supervised clustering proved to be particularly effective for coreference resolution Yu and Joachims [2009]; Fernandes et al. [2014]. It is known that coreference is a a very hard NLP task. Supervised clustering models learn to infer optimal clustering $\mathbf{y}$ of an input set $\mathbf{x}$ in a structured way, i.e. as one output object optimizing a global scoring function $f : X \times Y \rightarrow \mathbb{R}$. Global models are different from local model. Indeed, while the latter aggregate multiple clustering decision taken with respect to pair of elements, global models draw predictions by finding:

$$\hat{y} = \underset{y \in Y}{\text{argmax}} f(\mathbf{x}, \mathbf{y}) \tag{5.1}$$

In the following, we provide the necessary details of the original approach of [Yu and Joachims, 2009] and then we show its adaptation for clustering questions.

5.2.2 Structured Output Clustering

To make inference of optimal clustering in Equation 5.1, Yu and Joachims [2009] represent clustering variable **y** using graph structures. For an input **x** they construct a fully-connected undirected graph G, whose nodes are elements x_i of the input **x** and edges are all the pairwise links between them (x_i, x_j). Any spanning forest **h** on G straightforwardly translated into a clustering **y**. In the representation, the nodes in each connected components of **h** are considered to belong to the same cluster. At this point, the authors incorporate the spanning forest structure **h** as latent variables and decompose the feature representation of input-output pair (**x**, **y**), which is extended with **h**, over the edges of **h**:

$$\Phi(\mathbf{x}, \mathbf{y}, \mathbf{h}) = \sum_{(x_i, x_j) \in \mathbf{h}} \phi(x_i, x_j) \tag{5.2}$$

Then, they employ the Kruskal's spanning algorithm to infer the optimal **h**, and, respectively, **y**. A linear model **w** is trained using the latent formulation of the structural SVM learner (LSSVM), to score the output clustering according to the function $f(\mathbf{x}, \mathbf{y}, \mathbf{h}) = \mathbf{w} \cdot \phi(\mathbf{x}, \mathbf{y}, \mathbf{h})$. The linear model **w** learns to score the edges since the structural feature vector decomposes over the edges. Imposing a structure onto the output is supposed to produce a better **w**, which we test in our experiments described in Section 5.4.

5.2.3 SVM Models

In order to work, our model for intent clustering relies on the pairwise similarity between questions (edge score). In the following section we describe text similarity features used for estimating similarities between pairs.

5.2.4 Pairwise question similarity classifier

To accurate estimate question-question similarity, we use state-of-the-art classifiers for computing semantic similarity for short text. More specifically, we refer to our previous works [Filice et al., 2016b, 2017], [Da San Martino et al., 2016], [Barrón-Cedeño et al., 2016] solutions/features shown effective in shared tasks by Nakov et al. [2016b, 2017];

Agirre et al. [2013, 2014]. In such works, a classifier is trained with SVMs, which learn a classification function $f : Q \times Q \rightarrow \{0, 1\}$, on duplicate vs. non-duplicate pairs of questions belonging to the question set Q. The score returned by the classifier is used to decide if two questions q_i and q_j are duplicate or not. We encode questions pairs as vectors of similarity features derived between two questions. Feature vectors are built for question pairs (q_i, q_j), using a set of *text similarity* features modeling the relations between two questions. More specifically, we compute 20 similarity features $sim(q_i, q_j)$ using word n-grams ($n = [1,\cdots,4]$), after stopwords removal, greedy string tiling Wise [1996], longest common subsequences Allison and Dix [1986], Jaccard coefficient Jaccard [1901], word containment Lyon et al. [2001], and cosine similarity.

5.2.5 Models

To perform supervised clustering we use: (i) the original implementation of the Latent SVM^{struct}[2] – LSSVM, and (ii) our implementation of the LSP algorithm based on the same clustering inference on undirected graphs using Kruskal's spanning algorithm. LSSVM and LSP require the tuning of a regularization parameter, C, and of a specific loss parameter, r (penalty for adding an incorrect edge), which we select on the dev. set. We pick up C from 1.0, 10.0, 100.0, 1000.0, and r values from 01.1, 0.5, 1.9.

5.2.6 Baselines

For comparison purposes, we employed two unsupervised clustering baselines: (i) spectral clustering [Ng et al., 2001], for which we employ the implementation from the *smile*[3] library, and (ii) relational k-means [Szalkai, 2013]. The former implementations takes a matrix of pairwise similarities between data points as input, whereas the latter approach is a generalization of k-means to an arbitrary matrix of pairwise distances. Thus, they can be run on the scores relatives to the question pairs (q_i, q_j). We provide two variants of such models based on the (q_i, q_j) score computations: first, we run both the methods on the scores obtained from a binary pairwise similarity classifier, described in Section 5.2.4. Second, we run clustering baselines on the tf-idf scores computed for the question pairs. Particularly interesting is the latter approach as it use the scores from a trained pairwise classifier. This introduces some supervision in standard unsupervised clustering approaches, originating new hybrid methods. As K-means and spectral clustering algorithms require the indication of the number of clusters k, we use the gold standard k of each example (clustering) in all our experiments. This corresponds to comparing with

[2]`www.cs.cornell.edu/~cnyu/latentssvm/`

[3]`http://haifengl.github.io/smile/`

an upper bound of the baselines.

5.3 Datasets: Building Intent clusters

In this section, we described the datasets used in this work, which we had to build explicitly for this task, as we could not find one suitable for our problem. Indeed, the datasets already available for benchmarking Natural Language Understanding systems [Coucke et al., 2017], were composed of a rather small set of relatively generic intents. Then, we detail our approach for converting resources available for similar tasks, e.g. question-similarity, into intent corpora, relying on an automatic followed by a manual post-annotation step. The intent corpora as well as the larger raw question cluster collections are available to the research community[4].

5.3.1 Quora Intent corpus

Quora question-similarity task: The Quora Intent corpus was derived from the original Quora question-similarity task. The original Quora task required detecting whether two questions are semantically duplicate or not. The associated dataset contains over $404,348$ pairs of questions, posted by users on the Quora website, labeled as duplicate pairs or not. For example, *How do you start a bakery?* and *How can one start a bakery business?* are duplicate, while *What are natural numbers?* and *What is a least natural number?* are not. One thing worth noting is the fact that coders label pairs in isolation, only having access to one pair to be labeled at time on Quora website. The pairs to be labeled were not selected randomly. In addition, to make the task more challenging, as well as more useful for practical applications, the organizers only offer pairs of questions that are somewhat semantically related

(5.3) q_1: How does an automobile works?
q_2: How does automobile R&D work?

(5.4) q_1: Will I lose weight if I fast ?
q_2: Why am I losing weight so fast ?
q_3: How can I lose my weight fast ?

In example 5.3, the lexical items ave very similar, yet the questions are rather distinct, as reflected in the Quora annotation. They also express very different user intents: while q_1 is a generic curiosity question about automobiles, q_2 is a practical request for information on R&D in the automotive industry. Example 5.4 shows why the Quora duplicate

[4]`https://ikernels-portal.disi.unitn.it/repository/intent-qa`

detection task is very challenging and requires a very good level of NLU Understanding: while these three questions are very similar on the surface level, they all convey distinct semantics.

Question clusters from Quora

Differently from the original question-similarity task, in this work, we are interested in automatically acquiring intents from large question repositories. Given this, we need a corpus that contains clusters of questions annotated by the underlying intents. To obtain such data, we approximate intent clusters with the clusters of similar questions from Quora. These can be obtained by exploiting the pairwise annotation and relying on the transitivity property of the duplicate relation: for each pair q_1, q_2 annotated as duplicate, we assign q_1 and q_2 to the same cluster; negative pairs (non-duplicate question) do not impact the the clustering in any way. This process has some obvious limitations by design (i) it will not give us any intent labels, only the clusters and (ii) it will not provide any hierarchy of intents or any general/large intent categories. Still, this method provides a large number of user-generated intents that manually labeling initiatives (e.g., Natural Language Understanding Benchmarks) cannot guarantee.

Manually annotating intent clusters

The procedure employed for deriving intents from the Quora dataset raises several potential issues[5]: (i) no consistency is enforced across labels, (ii) duplicate or very similar Quora answers potentially pollute the annotation for their corresponding questions, (iii) specific decisions may depend on availability and granularity of underlying answers, and (iv) the annotation of popular questions might be very spurious since the users have no access to all the other related questions. Moreover, we found numerous cases where the annotation does not respect the transitivity property:

(5.5) q_1: What are, if any, the medical benefits of fasting?
q_2: What are the benefits of water fasting?
q_3: What are the health benefits of fasting?

Here, the three independent coders have produced inconsistent labeling: although q_2 and q_3 are explicitly labeled as non-duplicate, they are both considered duplicates of q_1. The second issue arises when the answer base contains (near-)duplicate entries. For example, the following two very similar questions are considered non-duplicates since they lead to two distinct answers:

(5.6) q_1: Which is better - DC or Marvel?
q_2: DC VS Marvel: which do you like more? [non-duplicate]

[5]https://www.kaggle.com/c/quora-question-pairs/discussion/30435

Note that this typically happens for rather popular questions that are therefore important to be analyzed correctly, either manually or automatically. The third issue is extremely important for areas only partially covered by the answer DB. For example, for the set of questions, *Why is Saltwater Taffy candy imported in LOCATION?*, most LOCATIONs are covered by a generic answer, and all the corresponding questions are judged duplicates. However, some specific LOCATIONs, e.g., Fiji, have a dedicated answer and thus the corresponding questions form singleton clusters. Finally, the annotation coherence problem arises for very popular areas covering a lot of closely related questions. Thus, more than 100 questions cover different aspects of *Weight Loss*. Since the coders do not have any access to all the questions on the same topic, the individual decisions are not coordinated, which leads to rather arbitrary partitioning of the area into clusters:

(5.7) **Gold Quora Cluster 1:**
How can I lose weight ?
What is the easiest way to lose weight faster ?
How can you lose weight quickly ?
How do I lose 7kgs in 2 weeks ?
What a great diet to lose weight fast and not make you hungry or keep on measuring portions ?
.. many more
Gold Quora Cluster 2:
How can I lose 3 kg in one week?
Gold Quora Cluster 3:
What are the good diets for weight loss ?
What is the best diet plan for weight loss?

To overcome these issues, we manually re-annotated a portion of the original Quora dataset with intent-based clusters. Started from clusters automatically induced according to the procedure described above, we first re-assessed the partitioning and, then, we correct eventual mistakes. Finally, we assigned intent-based labels to our clusters. Our labels are hierarchical, thus allowing for a better flexibility when designing dialog managers: dialog managers can be defined in terms of generic (e.g., `Advice`) or more specific (e.g., `Advice-WeightLoss-diet`) intents, depending on different implementation considerations (query frequency for specific intents, overall importance for the application, difficulty of processing inter alia). Also, in order to follow recent trends in dialogue research, we annotate *slots*, where applicable. The latter are entities external to the intent, which yet play an essential role for the correct semantic interpretation of question. In Table 5.1 we show an example of cluster annotations based on intents.

Intent	**Slots**	**Questions**
Recommend-TourismCuisine	streetfood, Delhi	What are the best street food in Delhi ?
Recommend-TourismCuisine	streetfood, Delhi	What is the best street food in Delhi ?
Recommend-TourismCuisine	streetfood, Delhi	What are the best street foods in delhi ?
Recommend-TourismRestaurant	streetfood, Delhi	What are the best street food places of delhi ?

Table 5.1: Manually annotated intent clusters for Quora

Dataset	N. questions	N. clusters	Average cluster size
TRAIN	636	270	2.36
DEV	315	146	2.16
TEST (automatic)	383	211	1.81
TEST (manual)	199	68	2.93

Table 5.2: Statistics about the Quora intent corpus.

5.3.2 FAQ: Hype Intent corpus

The second corpus we used, Hype, allows for a more direct evaluation of intent clustering algorithms. The data have been collected from a set of questions asked by users to a conversational agent. Such questions have been processed for constructing a FAQ section for Hype – an online service that offers a credit card, a bank account number and an iBanking app to its customers. Unlike Quora, questions are explicitly assigned to clusters by human annotators, and these clusters corresponds to intents by construction. However, for these intents we do not have informative labels and there is therefore no associated hierarchy. While this corpus provides very valuable data for our study, the main disadvantage is a very limited number of questions. Some examples are reported below:

(5.8) q_0: Cos'é HYPE? (What is HYPE?)
q_1: Volevo dei chiarimenti di cos'é la app hype (I'd like to have more information about the hype app)
q_2: mi puó spiegare cose'é la app hype (could you please explain me what the hype app is?)
q_3: informazione applicazione hype (information about hype app)

At the current stage of our research, we use the FAQ/Hype corpus directly, with no automatic or manual adjustments as we did for Quora.

5.4 Experiments

Dataset	N. questions	N. clusters	Average cluster size
TRAIN	97	19	5.11
TEST	50	9	5.55

Table 5.3: Statistics about the FAQ-HYPE dataset

In this section, we first provide details about the corpus used and introduce the evaluation measures employed to assess the accuracy of clusters predicted by our model. Then, we report the results obtained on (i) the Quora and (ii) FAQ-HYPE dataset.

5.4.1 Setup

We used two different corpora, described in Section 5.3:

The Quora Intent corpus contains 270, 146 and 212 clusters in the training, development and test sets. The clusters contain different numbers of questions, ranging from singleton to groups of 100+ questions. The singletons are the dominant group in the Quora dataset. This is probably due to the inclusion of non-duplicate questions that appear in the original Quora dataset. Overall, ther are $1,334$ questions distributed in 628 clusters (an average of 2.12 questions/cluster). More details about the corpus are reported in Table 5.2.

The FAQ/Hype corpus contains no small-size (< 3) clusters by construction since smaller clusters are typically not selected as FAQ entries. The largest groups of clusters are those of size 8 and 9. Overall, the FAQ Intent corpus contains 147 questions spread in 28 intent-based clusters, which correspond to an average of 5.25 questions/cluster. In our experiments, we used 97 questions for training and 50 questions for testing, distributed in 19 and 9 clusters, respectively. Table 5.3 reports statistics statistics about the corpus.

5.4.2 Evaluation measures

We compare the output clustering $\hat{\mathbf{y}} = \bigsqcup_j \hat{c}_j$ to the ground truth $\mathbf{y}^* = \bigsqcup_i c_i$, where c_i, in our case, are either the clusters obtained with transitive closure from Quora annotation or the manually annotated categories (see Section 5.3.1). For evaluation purposes, we assign each cluster $\hat{c}_j$ to the most frequent gold class (cluster), i.e., $argmax_i|c_i \cap \hat{c}_j|$, and compute the average precision over the clustering as:

$$Precision = \frac{1}{N} \sum_{j=1}^{\hat{k}} max_i|c_i \cap \hat{c}_j|, \qquad (5.9)$$

where N is the number of questions to be clustered, and $\hat{k}$ is the number of output clusters. This number is exactly the standard clustering purity by Zhao and Karypis [2002]. Since the purity is known to favor the clustering outputs with the large number of clusters, we interchange the roles of output and gold clusters, which gives us the clustering

$$Recall = \frac{1}{N}\sum_{j=1}^{k} max_i|\hat{c}_i \cap c_j|, \tag{5.10}$$

where k is the number of gold standard clusters. We then compute F1 from the above measures. The defined majority-class based clustering measure allows assigning more than one cluster to the same gold cluster. The coreference resolution metric CEAF [Luo, 2005; Cai and Strube, 2010] solves this issue by finding one-to-one alignment between the clusters in the output and in the ground truth, based on which the final score is computed. We use CEAF$_e$, the variant with the entity-based similarity, as an alternative evaluation measure.[6] Note that, although we split the data into samples, all the clustering measures we use, the majority-based, defined by equations 5.9 and 5.10, and CEAF, are computed over the whole test sets (not by averaging scores separately for each sample). We evaluate the results as well in terms of the classification scores relative to the correctness of the models in detecting the pairs of questions with the same/different intent. This enables the comparison against the pairwise classification approaches and an evaluation of their impact. We compute the Precision, Recall, and Accuracy of question pairs with the same intent.

5.4.3 Experiments on Quora

Original question label-based evaluation: As first thing, we test the models on clustering data from the Quora corpus, derived as described in Section 5.3.1. We train LSSVM, LSP and the SVM classifier on the training part. The results of all the models on the test set are depicted in Table 5.4. As can be seen, in terms of clustering accuracy, the LSSVM approach outperforms all the clustering baselines, improving about 10 points the highest baseline mode, i.e., SVM + k-means both in terms of F1 and CEAF. LSP, while outperforming the baselines, shows a slightly worse F1 than LSSVM, producing a model with high recall. In order to assess the impact of the pairwise classifier, we consider all the pairs of questions predicted by the mustering approaches as belonging to the same cluster as positive and the rest – as negative. After that, we measure pairwise classification precision, recall, F1 and accuracy (right side of the table). Interestingly, only LSSVM, though, among all the clustering models, approaches the classifier in terms

[6]We used the version 8 of the official coreference scorer `conll.cemantix.org/2012/software.html`

Model	**Clustering**				**Pairwise Classification**			
	Precision	Recall	F1	CEAF$_e$	Precision	Recall	F1	Accuracy
LSSVM	**80.16**	77.81	**78.96**	**63.68**	**43.74**	32.00	36.96	88.41
LSP	66.06	**91.64**	76.78	51.50	20.36	**76.85**	32.19	65.62
SVM + spectral clustering	72.06	62.40	66.89	47.04	28.07	3.52	6.26	88.80
SVM + k-means	70.76	66.58	68.60	53.87	31.03	7.92	12.62	88.35
tfidf + spectral clustering	72.06	62.92	67.18	52.96	33.90	4.36	7.72	**88.94**
tfidf + k-means	69.19	65.01	67.04	50.94	29.95	5.33	9.04	88.62
SVM					26.25	72.23	**38.50**	75.50

Table 5.4: Supervised vs. unsupervised clustering models and pairwise classification baselines on the test set, where the gold labels are from the original Quora annotation. Note that pairwise classification does not provide a good estimation of clustering accuracy.

of classification F1. However, clustering accuracy depends on many factors in addition to pairwise classification accuracy.

Intent-based evaluation: In Table 5.5 we present the results obtained on the portion of the test set which we manually annotated with intent clusters. We apply the same LSSVM, LSP and SVM classifier model trained in the experiments of the previous paragraph. However, before proceeding we recomputed all the four unsupervised clustering baselines supplying them with the new k – the number of gold intent-based annotated clusters. Although these new models have been trained on data with different annotation style, which is potentially noisy, LSSVM is able to recover new intent categories between than other baseline approaches in terms of all clustering metrics. However, the difference from the closest unsupervised clustering approach, which is the same as in the previous experiments, is now reduced in terms of CEAF. The new information about the number of clusters in the ground truth impact severally on the accuracy. The LSP model score the best with respect the new annotation. An interesting thing to note is the fact that the LSSVM classification accuracy is lower with respect the SVM pairwise classifier. However, this result is expected as the cluster number changed notably with the new annotation.

5.4.4 Evaluation on the FAQ dataset

Since the FAQ HYPE dataset has limited size, we split it into two parts, each of which form one sample. We use the one containing 19 out of 28 clusters for training and the other with remaining 9 clusters for testing. The training sample is composed of 97 questions, while the test sample is composed of 50. The plots in Figure 5.1 show the performance

Model	Clustering				Pairwise Classification			
	Precision	Recall	F1	CEAF$_e$	Precision	Recall	F1	Accuracy
LSSVM	**84.92**	51.76	64.32	49.72	33.24	7.86	12.71	78.07
LSP	71.36	**89.45**	**79.38**	**59.99**	37.22	**83.46**	**51.48**	68.03
SVM + spectral clustering	62.31	43.22	51.04	33.04	35.11	2.95	5.44	79.17
SVM + k-means	68.84	47.24	56.03	48.62	42.71	6.48	11.25	**79.23**
tfidf + spectral clustering	65.83	45.23	53.62	35.46	38.18	3.62	6.62	**79.23**
tfidf + k-means	65.83	47.24	55.00	38.49	29.31	9.43	14.26	76.98
SVM					**40.35**	62.33	48.99	73.62

Table 5.5: Supervised vs. unsupervised clustering models and pairwise classification baselines on the test set, where the gold labels are provided by the intent-based manual annotation on a portion of the test set.

of LSSVM and LSP in terms of clustering F1 compared to the clustering baselines. In addition, we run the k-means and spectral clustering algorithms with k in the range (1,50), which covers all the possible numbers of clusters for the given test set size. As scan be seen LSSVM is better than the spectral clustering models with any k. k-means curves surpass LSSVM only in a narrow interval, showing high instability. This result suggests that guessing the right k value in a realistic scenario in the absence of supervision does not seem an easy task. However, it should be noted that we deal with very scarce training data. This also explains slightly insufficient accuracy of LSP compared to the k-means baseline.

5.4.5 Error Analysis and Discussion

Quora: As can be seen in tables 5.4 and 5.5, the structural output model consistently outperforms strong baselines such spectral clustering and k-means. The most prominent improvement comes from singleton clusters: questions that are not duplicate with any other entries. Recall that the original dataset is constructed in such a way that singleton clusters are somewhat similar or related to existing material, but are still considered distinct by Quora annotators. LSSVM correctly recovers 71% of singleton clusters, whereas other methods perform much worse (5-30%). In the question-answering setting, singleton clusters correspond to novel questions that require setting up of a new entry in the answer base. Accurate recognition of singletons would allow for a timely allocation of resources to keep the answer base up-to-date and in line with incoming user requests. Larger clusters are problematic for all the compared methods. Still, As evidenced by the

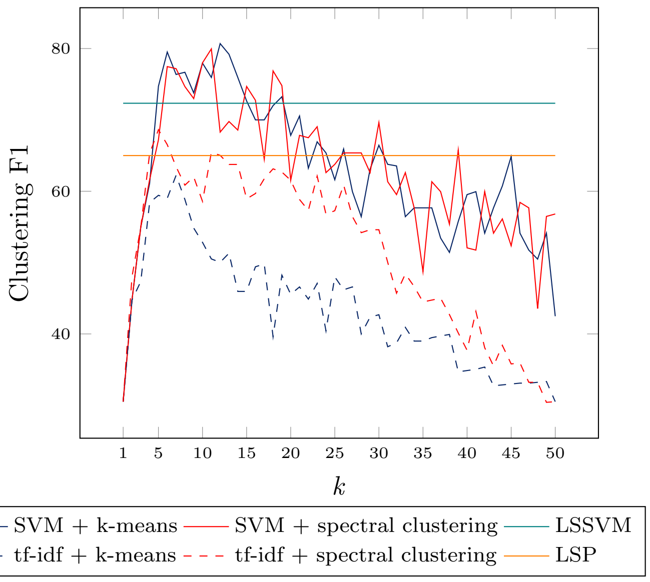


Figure 5.1: LSSVM and baseline clustering models; the latter vary with the cluster number k, on the FAQ HYPE test set.

CEAF[7] score, the structural clustering is doing a better job at recovering non-singleton clusters. This reflects our observations that even human annotators have difficulties in correctly and consistently detecting duplicates in complex over-populated semantic areas (see Example 5.11) in the absence of the global context (e.g., list of all the related questions). Finally, the clusters created by the LSSVM approach are more semantically related. Thus, 97% of all suggested clusters contain questions with the same intent, but, possibly, incorrect `slots`. For example, in the following question cluster:

(5.11) **gold cluster**
`Advice-Weightloss: fast,deadline`
q_1: How do I loose 50 lbs by Dec 2016?
q_2: How do I loose weight fast for operation ?
q_3: How can I lose 20 lbs super fast to audition for a small role in a movie ?
q_4: I want to lose weight for an event coming up in 2 weeks and I really don't care if I gain it back afterwords. What should I do ?

the user wants advice on losing their weight very fast by a specific deadline. LSSVM group these questions with some others, more generic queries on weight loss (*How do I loose weight fast?*). This means that LSSVM captures intent hierarchy well, providing meaningful clusters, although occasionally misses some important details. Other methods, on the contrary, form more poorly-related clusters (25-42% of clusters suggested by unsupervised approaches contain unrelated intents). Thus, questions from Example 5.11

[7]The reference scorer adopted by the coreference community discards singleton clusters.

get grouped by baselines with unrelated queries such as *How is it to be in true love?* (spectral clustering over tf-idf). It's important to node that neither LSSVM nor unsupervised approaches have any access to the cluster labels in the hierarchy in the training data, we only specify the clustering itself. Yet, by taking into account the global cluster structure, LSSVM method can uncover the underling hierarchy.
FAQ HYPE: In the FAQ setting, most clusters are mid size (5-9 questions). All the methods do a moderate job at recuperating the intent structure in this experiment. However, LSSVM shows better performance (see Section 5.4.4). Moreover, structural output is the only method capable to recuperate at least some clusters, e.g.:

(5.12) q_1: Non ricordo piú la password per accedere all'App (I don't remember the password for the App)
q_2: mi sono dimenticato la password (I forgot the password)
q_3: reimpostare la password (reset the password)
q_4: cambio password (change the password)

Here, LSSVM predicts the correct cluster exactly. In contrast, while k-means based approaches put q_1 – q_4 into the same cluster, they also merge them with *bloccare gli acquisti online (block the online purchases)*. Finally, spectral clustering does a poor job on this particular example, tearing either q_1 (tf-idf based spectral clustering) or q_2 (SVM pairwise-based spectral clustering) apart and introducing a lot of spurious material.

5.5 Conclusions

In this chapter we proposed structured output methods fed with semantic question paraphrasing models to automatically extract user intents from question repositories. Our approach provides clustering accuracy of 80% with respect to the original Quora annotation and still valuable accuracy of 65% with respect to one of the many interpretations of question intent of our dataset, carried out by our expert in dialog modeling. This line of research looks promising as it can potentially simplify and speed up the work of Dialog Manager engineers. Although a deeper study is required to assess the benefits of our approach, preliminary results suggests that automatic clusters, even if were not perfect, simplify the annotation work. Several future research directions are enabled by our study, ranging from the use of neural clustering models to the application of our models to fast and semi-automatic prototyping of Dialog Systems. For this purpose, we made our data and software available to the research community.

Chapter 6

NLP Pipelines and demos

In this chapter, we present a set of pipelines and NLP demos that we developed as side to this thesis work. First we present a Multi-lingual pipeline developed in the context of the Limosine European project (Section 6.2.1). The experience accumulated working on this pipeline allowed us to implement the cQA pipeline using structural models presented in Chapter 3. In addition, the availability of a multi-lingual NLP pipeline open interesting research directions such as the design of cross-lingual models for cQA. Then, we describe our effort aimed at improving the performance of constituency parsers for Italian language, which were very low compared to the English counterpart. The results of this work was partly used to build the syntactic representations employed in the automatic Help Desk system presented in Section 6.4. This system, designed with the help of a company operating in the IT field, has been successfully implemented to support operators working in an HD office to answer questions asked by clients in Italian language. As last thing, we implemented a system for factoid questions in Italian. Although factoid QA is not the focus of this work, we must consider that a small percentage of questions asked by users on discussion forums required simple named entities as answers. Thus, in last section, we proved the effectiveness of using structural representations for improving the performance of systems also for factoid QA.

6.1 Multilingual UIMA-based NLP Platform

In this section, we present a robust and efficient multilingual UIMA-based platform for automatically annotating textual inputs with different layers of linguistic description. The types of annotations returned by the pipeline range for surface level phenomena all the way down to deep discourse-level information. More particularly, the pipeline extract sentence tokens, entity mentions, syntactic information; opinionated expressions; relations between entity mentions co-reference chains ad wikified entities. The system is available in two versions: standalone distributions, expandable with new user-specific submodules and a server-client distribution allowing for high-performance NLP processing for higher-level tasks.

6.2 Overview

Nowadays, the growing amount of textual information require Natural Language Processing pipelines more scalable. Due to this, In recent years a lot of effect has been invested into the development of multi- and cross-lingual resources, which annotate textual inputs with various linguistic structures. To address this problem, we present the LiMoSINe pipeline – a platform developed by the FP7 EU project LiMoSINE: Linguistically Motivated Semantic aggregation engines Uryupina et al. [2016]. While many platforms and toolkits have been already made available to the research community in the past decades, e.g.. OpenNLP[1], FreeLing Padró and Stanilovsky [2012], and GATE Cunningham et al. [2011], these tools suffer from the following drawbacks:

- most of these tools require considerable effort for installation and configuration
- parallelism might be an issue
- for languages other than English modules are missing, while the existing ones have only a moderate performance level.

In the LiMoSINE project, we focused on high-performance NLP processing for four European languages: English, Italian, Spanish and Dutch. We combine state-of-the-art solutions with specifically designed in-house models to ensure reliable performance. Furthermore, the pipeline is based on the UIMA framework Ferrucci and Lally [2004], which allow for full parallelism when processing large amounts of data. The pipeline is available at: `http://ikernels-portal.disi.unitn.it/projects/limosine/`

6.2.1 LiMoSINe pipeline: overall structure

Our platform support various level of linguistic descriptions of document's semantics, obtained by combining the outputs of numerous linguistics preprocessors. The structure of our pipeline is shown in Figure 6.1. As it can be seen, the pipeline is composed by many preprocessors – designed by different project partners and stakeholders – whose input/output format has been unified to ensure interoperability among the components. This is achieved thanks to the modularity feature at the base of UIMA, which made it

[1] http://opennlp.apache.org

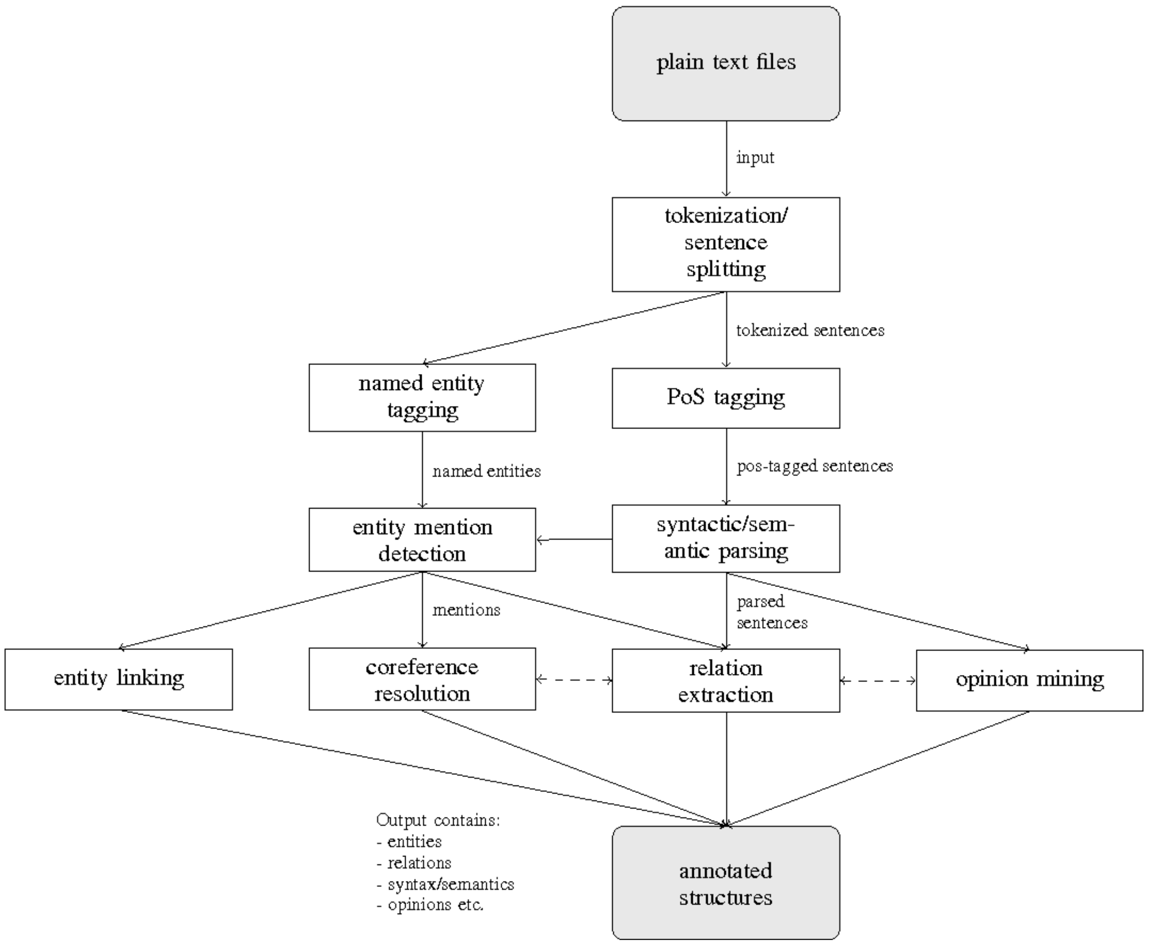


Figure 6.1: LiMoSINe pipeline architecture

Annotator	English	Italian	Spanish	Dutch
tokenizer	Stanford	TextPro	IXA	xTas/Frog
POS-tagger	Stanford	TextPro	IXA	xTas/Frog
NER	Stanford	TextPro	IXA	xTas/Frog
Parsing	Stanford, LTH	FBK-Berkeley	IXA	xTas/Alpino
Entity Mention Detection	BART	Bart-Ita	-	-
Opinion Mining	Johansson and Moschitti [2011]	-	-	-
Relation Extraction	RE-UNITN	RE-UNITN unlex	-	-
Coreference	BART	Bart-Ita	-	-
Entity Linking	Semanticizer	Semanticizer	Semanticizer	Semanticizer

Table 6.1: Supported modules for different languages

successfully adopted or a number of NLP projects, e.g. IBM Watson system Ferrucci et al. [2010]. During the processing, the individual annotators update the representation of document, which store in a CAS object. If a new annotation is required for an other task, UIMA allows for straightforward deployment of new components. In addition to parallelization, our pipeline can be deployed both locally or remotely. This avoid the user to download the pipeline locally for using it. Instead, the user can reach a client-server version of the pipeline installed on the LiMoSINe server. The client application can be download from the pipeline website, thus allowing the user to obtain annotation from components implements state-of-the-art algorithms for solving NLP tasks. The annotations are dispatch by the remote pipeline to the local application running on the client machine. This provide valuable support for projects focusing on higher-level tasks such as Question Answering, especially for languages other than English.

6.2.2 Integrated modules

Our multi-language pipeline has focused on four European language: English, Italian, Spanish and Dutch. For all these languages, we provides robust parallelizable NLP processing up to the syntactic parsing level. In addition, for some languages, we provide deeper semantic and discourse level processing, such as relation extraction, coreference, opinion mining and entity linking. Table 6.1 provides an overview of all currently supported modules.

English

Here we list the components integrated in our pipeline for processing English language.

Stanford tools. To provide basic preprocessing, required by our high-level components, we created UIMA wrappers for several Stanford NLP tools Manning et al. [2014]: the tokenizer, the parser and the named entity analyzer.
Entity Mention Detector. We developed an entity mention detector (EMD) [Uryupina et al., 2011, 2012] as part of BART (see below). The EMD extract *mentions*– textual units that correspond to real-word object–needed by both coreference resolver and relation extract. The EMD has been developed at the university of Trento and it is a rule-based system combining the outputs of a parser and an NE-tagger to extract mention boundaries and assign mention types (name, nominal or pronouns) and semantic classes (inferred from WordNet for common nouns, from NER label for proper nouns).
Opinion Mining. The opinion expression annotator is a system developed at the University of Trento by Johansson and Moschitti [2011]. It extracts fine-grained opinion expressions together with their polarity. Opinions are extracted by using a standard sequence labels for subjective expression markup, in a way similar to the approach by Breck et al. [2007]. The system was developed on the MQA corpus that contains news articles. It uses the syntactic/semantic LTH dependency parser of [Johansson and Nugues, 2008] and requires pre-tokenized and tagged input data formatted according to the CoNLL-2008 shared task
Relation Extraction. Our relation extractor (RE) is a tree-kernel based system developed at the University of Trento [Moschitti, 2006; Plank and Moschitti, 2013]. The Tree-kernel based methods have been shown to outperform feature-based RE approach [Nguyen et al., 2015]. The system takes in input mentions (together with their entity types, i.e. PERSON, LOCATION, ORGANIZATION or ENTITY) from the EMD module, and provide relations. The Relation Extractor includes two models. The first one use the relation extracted using the ACE 2004 data and output the following binary relations: Physical, Personal/Social, Employment/Membership, PER/ORG Affiliations and GPE Affiliation. The second model is an version of the Relation Extractor and include additional model trained o the CoNLL 2004 data [Roth and Yih, 2004] following the setup of [Giuliano et al., 2007]. The model uses composite kernels consisting of path-closed tree kernel on constituents and a linear vector encoding local and global contexts [Giuliano et al., 2007] Both models exhibit state-of-the art performance. For the ACE 2004 data, experiments are reported in Plank and Moschitti [2013]. For the CoNLL 2004 data, our model achieves results comparable to state of the art or more [Giuliano et al., 2007; Ghosh and Muresan, 2012].

Coreference Resolution. Our coreference resolution Analysis Engine is a wrapper around BART – a toolkit for Coreference Resolution developed at the University of Trento [Versley et al., 2008; Uryupina et al., 2012]. It is a modular anaphora resolution system that support state-of-the-art statistical approaches to the task. BART implements several models for anaphora resolution and has interface to different machine learners (MaxEnt, SVM, decision trees). In addition, provides a large set of linguistically motivate feature (MaxEnt, SVM, decision trees). In addition, provides a large set of linguistically motivated features.
Entity Linking. The Entity Linking Analysis Engine ("Semanticizer") makes use of the Entity Linking Web Service developed by the University of Amsterdam [Meij et al., 2012]. The web service supports automatic linking of an input text to Wikipedia articles: the output of the web service API is a list of IDs of recognized articles, together with confidence scores as well as the part of the input text that was matched. The entity linking module is cross-lingual, since mentions in documents in different languages are disambiguate and linked to Wikipedia articles. Each annotation unit corresponds to a span in the document and it is labeled labeled with the corresponding Wikipedia ID and the system's confidence.

Italian

Here we list the components integrated in our pipeline for processing the Italian language. We integrated language-specific processors for tokenization, sentence splitting, named entity recognition, parsing, mention detection and coreference. For relation extraction, we adapted the English model to Italian, transferring the unlexicalized learned model trained on the English data. A detailed description of our annotator for Italian is provided below.

TextPro wrapper. To provide basic levels of linguistic tic processing, we rely on TextProa suite of Natural Language Processing tools for analysis of Italian (and English) texts [Pianta et al., 2008]. The suite has been designed to integrate various NLP components developed by researchers at Fondazione Bruno Kessler (FBK). The TextPro suite has shown exceptional performance for several NLP tasks at multiple EvalIta competitions. Moreover, the toolkit is being constantly updated and developed further by FBK. We can therefore be sure that TextPro provides state-of-the-art processing for Italian. TextPro combines rule-based and statistical methods. It also allows for a straightforward integration of task-specific user-defined pre- and post-processing techniques. For example, one can customize TextPro to provide better segmentation for web data. TextPro is not a part of the LiMoSINe pipeline, it can be obtained from FBK and installed on any platform in a straightforward way. No TextPro installation is needed for the client

version of the semantic model.
Parsing. A parsing model has been trained for Italian on the Torino Treebank data[2] using the Berkeley parser by the Fondazione Bruno Kessler. Both the Torino TreeBank itself and the parsing model use specific tagsets that do not correspond to the Peen TreeBank tags of the English parser. To facilitate cross-lingual processing we have map the tagset of the Torino Treebank to the Penn TreeBank and vice versa. **Entity Mention Detection.** We have adjusted our Entity Mention Detection analysis engine to cover the Italian data. Similarly to the English module, we use BART to heuristically extract mention boundaries from parse trees. However, due to the specifics of the Torino Treebank annotation guidelines, we had to change the extraction rules substantially.
Relation Extraction Since there are no relation extraction datasets available for Italian, we have opted for a domain adaption solution, learning an unlexicalized model on the English RE data. This model aims to capture structural patterns characteristic for specific relations through tree kernel-based SVMs. To do so, we extract tree patterns for CoNLL-2004 relations from the unlexicalized variant of the English corpus and then run it on modified Italian parse trees. Although this model cannot provide robust and accurate annotations, it can be used as benchmark for supervised RE in Italian.
Coreference Resolution A coreference model for BART has been trained on the Italian portion of the SemEval-2010 Task 1 dataset [Uryupina et al., 2012]. Apart from retraining the model, we have incorporated some language-specific features to account, for example, for abbreviation and aliasing patterns in Italian. The Italian version of BART, therefore, is a high-performance language specific system. It has shown reliable performance for Italian, in particular, at SemEval-201- Task 1 [Broscheit et al., 2010] and at the EvalIta 2009 [Biggio et al., 2009].

Spanish

We have tested two publicly available toolkits supporting language processing in Spanish: OpenNLP and IXA [Agerri et al., 2014]. The latter has shown a better performance level and has therefore been integrated for the final release of the LiMoSINe pipeline. For tokenization, we rely on the `ixa-pipe-tok` library (version 1.5.0) from the IXA pipes project. Since it uses FSA technology for the tokenization and a rule-based segmenter, it is fast (tokenizing around 250K words/s) and expected to be valid across several dialects of Spanish [Agerri et al., 2014]. The POS tags are assigned by using the IXA model for Maximum Entropy POS tagging, and reported to provide 98.88% accuracy [Agerri et al., 2014]. Lemmatization uses the morphology-stemming toolkit, based on FSA for a lower memory footprint (up to 10% the size of a full-fledged dictionary). Named entities

[2]`http://www.di.unito.it/tutreeb/`

(PERSON, LOCATION, ORGANIZATION and MISC) are annotated using the Maximum Entropy model of IXA trained on the CONLL 2002 dataset and tags. Finally, the IXA pipeline provides a module for constituency parsing trained on the (Iberian) Spanish section of the AnCora corpus.

Dutch

For Dutch, we have been able to integrate language-specific processors for tokenization, sentence splitting, lemmatization, named entity recognition, dependency tree, and part-of-speech tagging. To provide basic levels of linguistic processing, we rely on xTasa text analysis suite for English and Dutch [De Rooij et al., 2012]. The suite has been designed to integrate various NLP components developed by researchers at University of Amsterdam and is ex extandable to work with components from other parties. xTas is designed to leverage distributed environments for speeding up computationally demanding NLP tasks and is available as a REST web service. xTas and instructions on how to install it and set it up can be found at `http://xtas.net`. Most of the Dutch processors at xTas come from Frog, a third-party module. Frog, formerly known as Tadpole, is an integration of memory-based NLP modules developed for Dutch [Bosch et al., 2007]. All NLP modules are based on Timbl, the Tilburg memory-based learning software package. Most modules were created in the 1990s at the ILK Research Group (Tilburg University, the Netherlands) and the CLiPS Research Centre (University of Antwerp, Belgium). Over the years they have been integrated into a single text processing tool. More recently, a dependency parser, a base phrase chunker, and a named entity recognizer module were added. For dependency parsing, xTas uses Alpino, a third-party module. Annotation typically starts with parsing a sentence with the Alpino parser, a wide coverage parser of Dutch text. The number of parses that is generated is reduced through interactive lexical analysis and constituent marking. The selection of the best parse is done efficiently with the parse selection tool.

6.2.3 Conclusion and Future Work

In this section, we have presented LiMoSINe, our multi-lingual UIMA-based pipeline, a platform supporting state-of-the-art NLP technology fro English, Italian, Spanish and Dutch. Being based on UIMA, it allows for efficient parallel processing of large volumes of text and can be distributed in two versions: (i) as a client applications oriented to potential users that need high-performance NLP processors at zero engineering cost; or (ii) a local version, which require some installation and configuration effort, but offers a great flexibility in implementing and integrating user-specified modules. The pipeline has been adopted by other parties, most importantly by the joint QCRI and MIT projects IYAS (Interactive sYstem for Answer Selection). The structural representations extracted by the pipeline proved to be effective in many tasks, e.g. Opinion mining on YouTube [Severyn et al., 2014], crossword puzzle resolution [Barlacchi et al., 2014] and Question Answering [Tymoshenko and Moschitti, 2015; Tymoshenko et al., 2014].

As future work it would be very interesting to implement new state-of-the-art deep-learning models for: (i) improving the overall performance of the pipeline among the different supported tasks and (ii) for replacing the components that are more difficult to configure when installing the local version of the pipeline.

6.3 Tree Kernels-based Discriminative Reranker for Italian Constituency Parsers

In this section we present our work Uva and Moschitti [2016] aimed at filling the gap between the accuracy of Italian and English constituency parsers: firstly, we adapt the Bllip parser, i.e. the most accuracy constituency parser for English, also known as Charniak parser, for Italian and trained it on the Turin University Treebank (TUT). Secondly, we design a parse reranker based on Support Vector Machines (SVMs) using tree kernels, where the latter can effectively generalize syntactic patterns, requiring little training data for training the model. We show that our approach outperforms the state of the art achieved by the Berkeley parser, improving it from 85.54 to 86.81 in labeled F1. Constituency Syntactic parsing is one of the most important research lines in Computational Linguistics as constituency parsing information is needed in many tasks, i.e. question similarity, semantic role labeling, question answering, etc... Consequently, a large body of work has been devoted to the design for the Italian language Bosco et al. [2007, 2009]; Bosco and Mazzei [2011]. However, the accuracy reported for the constituency best parsers for Italian language is still far behind the state of the art of other languages, e.g. English. This makes such technology to very useful for being used in more challenging semantic tasks, e.g. QA. One noticeable attempt to fill this technological gap was carried out in the Evalita challenge, which proposed a parsing track on both dependency and constituency parsing for Italian. Among the several participant systems, the Berkeley parser Petrov and Klein [2007] gave the best result Lavelli and Corazza [2009]; Lavelli [2011]. At the beginning the performance for constituency parsing computed on TUT Bosco et al. [2009] was much lower than the one obtained for English on the Penn Treebank Marcus et al. [1993]. In the last EvalIta edition, new systems were presented and the gap diminished as the Italian parser labeled F1 increased from 78.73% (EvalIta 2009) to 82.96% (EvalIta 2011). Some years later the parser F1 improved to 83.27% Bosco et al. [2013]. However, the performance of the best English statistical parser McClosky et al. [2006], i.e., 92.1% is still far away. The main reason for such gap is the difference in the amount of training data available for Italian compared to English. The main reason for such gap is the difference in the amount of training data available fir Italian compared to English. In fact, while Penn Treebank contains 49, 191 sentences/trees, TUT only contains 3, 542 sentences/trees. In presence of scarcity of training data, a general solution for increasing the accuracy of machine learning-based system is the use of more general features. This way, the probability of machine training and testing instance representation is larger, allowing the learning process to find more accurate optima. In case of syntactic parsing, we need to generalize either lexical or syntactic features, or possibly both. However, modeling such generalization in state-of-the-art parser algorithms such as the Bllip[3] Charniak [2000]; Charniak and Johnson [2005] is rather challenging. In particular, the space of all possible syntactic patterns is very large and cannot be explicitly coded in the model. In this section we present our effort in filling the gap between English and Italian constituency parsing accuracy: firstly, we adapted the Bllip parser, i.e., the most accurate statistical constituency parser for English, also known as Charniak parse, to Italian and trained it on TUT. We designed various configuration files for defining specific labels for TUT by also defining their type, although we did not encode head-finding rules for Italian, needed to complete the parser adaptation. Secondly, we apply rerankers based on Support Vector Machines (SVMs) using TKs by Moschitti [2006] to he k-best parses produced by Bllip, with the aim of selecting its best hypothesis. TKs allow use to represent data using the entire space of subtrees, which correspond to syntactic patterns of different level of generality. This representation enables the training of the ranker with little data. Finally, we tested our models on TUT, following the EvalIta setting and compare with other parsers. For example, we observed an improvement of about 2%, over the Berkeley parser, i.e., 86.81 vs. 84.54.

6.3.1 Bllip parser

In this section we briefly describe the model at he base of the Bllip parser. Bllip is a lexicalized probabilistic constituency parser. It can be considered a smoothed PCFG, whose non-terminals encode a wide variety of a manually chosen conditioning information, such as heads, governors, etc. Such information is used to derive probability distributions, which, in turn are utilized for computing the likelihood of constituency parse trees being generated. As described by McClosky et al. [2006], Bllip uses five distributions, i.e. the probabilities of (*i*) constituent heads, (*ii*) constituent part-of-speeches (PoS), (*iii*) head-constituents, (*iv*) left-of-head and (*v*) right-head-constituents. Each probability distribution is conditioned by five or more features and backed-off by the probability of lower-order models in case of rare feature configurations. The variety of information needed to properly train Bllip makes it much harder to configure than other parsers, e.g., The Berkeley' one. In contrast, Bllip is much faster to train than many other off-the-shelf parsers.

Adapting Bllip to Italian Language

Adapting Bllip to a new language require creating various configuration files. For examples, PoS and bracket labels observed in training and development must be defined in a

[3]https://github.com/BLIIP/blip-parser

file named *terms.txt*. As labels present in the TUT are different from those of the Penn Treebank., we added them in such a file. Then we specified the types of labels present in the data, i.e., constituent types, open-class Pos, punctuation, etc. Finally, it should be noted that, since Bllip is lexicalized, head-finding rules for Italian should be specified. However, in this work we used the default Bllip rules and leave this task as our short term future work.

6.3.2 Tree Kernel-based Reranker

We describe three types of TKs and Preference Reranker approach using them. Tree kernels free us from the burden to explicitly design features for many tasks, e.g. parse reranking Collins and Duffy [2002], as they implement scalar product between feature vectors as a similarity between two trees. Such scalar product is computed using efficient algorithms and it basically equal to the number of common subparts of the two trees.

Syntactic Tree Kernels (STK) count the number of common tree fragments, where the latter (i) contains more than two nodes and (ii) each node is connected to either all or none of its children. We also used a variant, called STK_b which ads the number of common leaves of the comparing trees in the final subpart count.

Partial Tree Kernels (PTK) counts a larger class of tree fragments, i.e., any subset of nodes, where the latter are connected in the original trees: clear PTK is a generalization of STK.

Preference Reranker

Here we use describe the preference reranking technique useful to train a classifier for reranking parse trees. In preference reranking we train a binary classifier for ranking tree hypothesis represented as pairs $\langle h_i, h_j \rangle$ The trained classifier is then used to decide if tree h_i is better than tree h_ji. Positive training examples are pairs $\langle h_1, h_j \rangle$ has the highest *F1* with respect to the gold standard among the candidate hypothesis. The negative examples are obtained inverting the hypothesis in the pairs, i.e. $\langle h_i, h_1 \rangle$ Hypothesis having the same score are not included in the training set. At classification time all pairs $\langle h_i, h_j \rangle$ generated from the k-best hypotheses are classified. A positive classification is a vote for h_i, whereas a negative classification is a vote for h_j. The hypothesis associated with the highest number of votes (or highest classifier scores) is selected as the best parse.

6.3.3 Experiments

In these experiments we report the performance of Bllip for Italian and compare it with the Berkeley parser. Then, we show that our parse reranker can be very effective, even in

Models	sentences $\leq$ 40 words				All sentences			
	LR	LP	LF	EMR	LR	LP	LF	EMR
Berkeley Bosco et al. [2013]	83.45	84.48	83.96	24.91	82.78	83.76	83.27	23.67
Berkeley (our model)	85.31	85.76	85.53	27.76	84.35	84.72	84.54	26.33
Bllip base model	85.90	86.67	86.28	29.54	85.26	85.97	85.61	28.00
STK	86.16	87.02	86.59	30.96	85.73	86.38	86.05	29.33
STK$_b$	86.36	87.21	86.78	**31.67**	85.89	86.53	86.21	**30.00**
PTK	**86.82**	**87.95**	**87.38**	30.96	**86.33**	**87.29**	**86.81**	29.67

Table 6.2: Comparative results on the test set. **LR/LP/LF** = labeled recall/precision/$F1$. **EMR** = percentage of sentences where recall and precision are 100%. STK- and STK$_b$-based rerankers use 20-best hypotheses, while PTK-based reranker use 30-best hypotheses.

case of use of small training data.

Experimental Setup

Parsing data. The data for training and testing the constituency parsers come from TUT project [4]. There have been several release of the dataset: we use the last version from EvalIta 2011 sentences composed of 3,542 sentences. The training set is composed of 3,542 sentences, while test set contains 300 sentences. The set of PoS-tags include 97 tags: 68 mophological features for pre-terminal symbols (e.g. ADJ, ADVB, NOUN, etc.) and 29 non-terminal sybols for phrase constituents (e.g. ADJP, ADVP, NP, etc.)
Reranking Data. To generate the reranker training data we apply 10-fold cross validation to the official TUT training set: we train the base parser on 9 folds and apply the model to the remaining fold to generate n-best trees for each of its sentences. Then, we merged all the 10-labeled folds to produce the training set of the reranker. Note that by following this procedure we avoid the bias a parser would have it applied to the data used for training it. For generating test data we simply apply the based parser trained on all the TUT training data to the TUT test set and generate n-hypothesis for each sentence.
SVM Reranker. We train the reranker using SVM-light-TK, which takes both trees and features vector to lean a classification model. More specifically, we use the following features for reranking constituency trees: (i) the probability and the (inverse) rank of the parse tree provided by Bllip and (ii) the entire syntactic trees used with two types of kernels, STK and PTK, described in Section 6.3.2
Measures. For evaluating the parsers we use the EVALB scoring program, which reports the Labeled Precision (LP), Labeled Recall (LR), Labeled *F1* (LF) and Exact Match Rate

[4]http://www.di.unito.it/ tutreeb/

Models	10-best				20-best				30-best			
	Tree		Tree + feat.		Tree		Tree + feat.		Tree		Tree + feat.	
	len≤40	All	≤40	All	len≤40	All	len≤40	All	len≤40	All	len≤40	All
Bllip base model	**87.06**	**86.25**	87.06	86.25	**87.06**	86.25	87.06	86.25	**87.06**	86.25	87.06	86.25
STK	85.40	85.02	86.85	86.26	85.14	84.64	87.32	86.80	85.36	84.87	87.33	86.80
STK$_b$	85.50	85.02	86.85	86.26	85.37	84.84	87.57	87.02	85.27	84.79	87.41	86.87
PTK	86.20	85.65	**87.78**	**87.06**	87.04	**86.51**	**88.44**	**87.80**	87.02	**86.52**	**88.18**	**87.57**

Table 6.3: Reranker performances: In the top are reported the number k of best parse trees used during training. Then, in the second row we report the group of features used: Tree or Tree + feat, while the third row shows the parse results for two sentence groups: sentences with $\leq$ 40 words and all sentences.

(EMR). We use the same evaluation setting adopted by the official EvalIta procedure for scoring participant system.

Bllip base parser results

We divided the training set in train and validation set, where the latter is composed of 300 sentences. We train the models on the training set and tune on the validation set. Then, we applied the best model on the tests set. Table 6.2 shows the results obtained by the Bllip parser compared to the other state-of-the-art Berkeley parser. Our parser obtained a LF of 86.28% for sentences with less than 40 words and a score of 85.61% for all sentences.

Comparison with the Berkeley parser

Tab 6.3 compare the results of the Berkeley parser obtained by Bosco et al. [2013] and our own version of the Berkeley parser trained for comparison purposes. We train the parser for 5 epochs and use a full PoS-tags scheme, as this configuration gave the best results on the dev. set. Our version of the Berkeley parser outperforms the version that of Bosco et al. [2013] by 1.2 absolute percent points (84.54 vs. 83.27). In addition, the Bllip parser outperforms the best results obtained by the Berkeley parser by 1.07% in LF, i.e. 85.61 vs. 84.54.

Reranking using different TKs

Table 6.3 reports the LF obtained by different reranking models, varying: (i) the type of TKs, (ii) the group of features used by the reranker (i.e. either trees or trees + features) and (iii) the number, n, of parse trees used to generate the reranker training data. More in particular, we experimented with three values for n, i.e., 10-, 20- and 30- best parse trees. As it can be seen from the table, PTK constantly outperforms STK and STK$_b$ for any number of parse trees generated by PTK. This proved that the subtree feature generates by PTK are very useful for improving parsing accuracy. One interesting thing is that the performance of all the models trained on 30-best trees give either word results (e.g. STK$_b$ and STK) or very little improvement (e.g. PTK) than training on 20-best parse trees. This may suggest that too many little negative examples can be detrimental. At the bottom of Table 6.3 we can see that the Bllip parser + reranker model shows an 1.2% absolute improvement in LF (from 85.61% to 86.81%) on all the sentences over the base-parser model when using the most powerful kernel, TK, and 30-best hypotheses. STK and STK$_b$ shows a lower improvement over the baseline of 0.44% and 0.6% respectively. Interestingly, while LF, STK and STK$_b$ give better performance in terms of EMS, i.e. percentage of sentence parse completely matching gold trees, then PTK. This can be intuitively explained by the fact that PTK, generating partial production rules, is better at capturing partial rules expressed by support vectors. In contrast, the precision in capturing complete patterns, i.e., regarding a complete tree, is decreased.

6.3.4 Conclusions

This work, was aim is to fill the gap in accuracy between English and Italian constituency parsing, was inspired by [Collins and Duffy, 2002] and [Collins and Koo, 2005], who explored discriminative approaches for ranking problems.

However, their studies were limited to WSJ. In this section we adapted the Charniak parser for Italian, gaining an improvement of 1.07% over the Berkeley model (indicated by EvalIta as the sate of the art for Italian). Then, our TK-based reranker further improved it up to 2 absolute percent points. It should also be noted that our best reranking results is 3.54 absolute points better than the best outcome reported in Bosco et al. [2013], i.e. 82.27.

6.4 cQA

In this section we describe our work on using current methods developed for Community Question Answering (cQA) of a commercial application focused on an Italian help desk Uva et al. [2017]. The approach we employed is based on a (*i*) a search engine to retrieve previously answered question candidates and (*ii*) kernel methods applied to most promising candidates. We show that methods developed for cQA work well also when applied to data generated in customer service scenarios, where the user seeks for explanation about products and a database of previously answered questions is available. The experiments demonstrate its suitability for an industrial scenario.

6.4.1 Overview

In recent years, large companies, e.g. IBM, Google, Facebook, Microsoft, ecc. invested a lot of resources in developing QA technologies for their commercial applications. However, medium and smaller enterprises cannot invest billions of dollars in achieving the desired QA accuracy: this limits the use of technology, especially for less supported languages , e.g. Italian. One alternative for smaller company involves the design of close-domain systems looking for answers in specific data such as customer documentation, which is often available in terms of unstructured text. However, even this scenario is complicated as reaching a satisfactory accuracy may require a lot of resources. An interesting alternative is provided by cQA technology, which as we saw in previous chapters, can be used for answering questions questions in specific forums. In addition, the fact that forums are divided by topics, which are rather restricted, make the retrieval task is easier. In this respect, cQA offers an even more interesting property: when a new question is asked in a forum, instead for directly searching for an answer, smarter technology would first rather try to look for a similar question. As it can be intuitively observed, the main advantage of this approach is that searching for similar questions is much easier than searching for text answering a given question. Due to this, challenges similar to SemEval 2017 Task 3 Nakov et al. [2016a] aimed at testing current cQA technology have been organized also for Italian language, e.g. QA4FAQ Caputo et al. [2016]. In this section, we show that companies can adopt cQA models to automatize the answering process. I particular, we describe a QA system developed for RGI, a software vendor specialized in the insurance businesses. One important task carried out by their help desk software regards answering customers' questions using a ticket system. Already answered tickets are stored in specialized databases but manually finding and routing them to the users in time consuming. We show that our approach, using standard search engines and advanced reranker based on machine learning and NLP technology, can achieve answer recall of almost 85% when

Table 6.4: An example of two similar tickets: the one used as query on the left and one retrieved by a search engine (only using question words) on the right.

Question$_{org}$	**Answer**$_{org}$	**Question**$_{rel}$	**Answer**$_{rel}$
Abbiamo bisogno delle credenziali di accesso al sistema. Grazie	Buongiorno, questo l'indirizzo mail al quale scrivere per avere le credenziali di accesso al sistema: xxx@xxx.xx Cordiali saluti	Buongiorno, non troviamo credenziali per accesso sistema. Potete aiutarci? Grazie	Buongiorno, questo l'indirizzo mail al quale scrivere per avere le credenziali di accesso al sistema: xxx@xxx.xx Cordiali saluti

considering the top three retrieved tickets. This results is particularly interesting because the experimented data and models are completely in Italian, demonstrating the maturity of this technology also for this language.

6.4.2 Related Work

From an industrial viewpoint, NLP is one of the hot topics of recent years, although it still mostly unexplored. Many platforms are emerging in the wide area of chatbot development, e.g., Wit.ai and Api.ai (proposed by Facebook and Google, respectively), which enable intent classification and entity extraction and Maya.ai, which can be used to developed rule-based chatbot systems. However, most of them are transparent to the final user and do not integrate QA models.

Task description

The scope of the experiments for this research is the evaluation of state-of-the-art QA models to automatize the operation of an help desk (HD) service.Typically, users interact contact the HD provided by a company in case of problems. A HD service is structured as a hierarchical organization of operators with different skill levels, which provide answers to the user requests, e.g., HD involves operators of Level 1 and regards basic knowledge; HD2 (Level 2) is managed by functional analysts with higher domain knowledge and so on. When a request is sent to an HD operator, a ticket is generated and stored in a trouble ticket system along with all the relevant information of that request: this includes a description of the problem and the detected solution. Such ticket is then managed, passed and eventually scaled by all the operators involved in the solution of the problem. In order to search and provide the right answer to the customer, each HD operator may use the following sources of information: **tickets** opened in the past; **Frequently Asked Questions** (FAQ) and their solutions, stored in a shared repository; a **forum**, where

Table 6.5: Results of the reranker obtained by combining Sim features with TKs.

	5-folds cv				
Model	MRR	MAP	P@1	P@2	P@3
IR baseline	70.85 ± 4.54	63.18 ± 3.37	57.67 ± 6.99	71.79 ± 3.98	77.86 ± 4.69
Sim	71.56 ± 4.16	63.90 ± 2.19	58.39 ± 8.04	72.44 ± 2.45	80.77 ± 3.31
TK	72.45 ± 2.19	67.09 ± 2.33	58.31 ± 3.42	75.34 ± 2.32	80.71 ± 3.36
TK + Sim	$\mathbf{75.07 \pm 1.67}$	$\mathbf{68.51 \pm 1.41}$	61.54 ± 1.86	$\mathbf{77.87 \pm 3.27}$	$\mathbf{84.57 \pm 2.57}$

HD operators share their knowledge; **user manuals** and other **domain knowledge** and expertise of the operator itself. Our objective is studying the impact of advanced QA systems for the automation of HD1, using FAQ and tickets data stored in the related repositories.

Data description

Data was gathered from the RGI HD support system, where technical issues are tracked and fixed. Basically, we have tickets organized in Question/Answer (Q/A) pairs, along with fields related to specific information, such as ticket ID and the domain problem. The original data size was around $40,000$ tickets but most of them do not provide useful information. Thus, we designed a preprocessing phase both to clean and prepare a valid data set: first, we designed a preprocessing phase both to clean and prepare a valid data set: first, we detected and filtered out spurious Question-Answer airs, concerning unanswered problems, using basic heuristics. Second, we extracted a subset of general-knowledge problems by selecting only tickets belonging to HD1 with a resolution time less than two days. In addition, our data was also reviewed by an expert team to filter out invalid tickets. As a result, the preprocessing with a dataset of 656 Q/A pairs spread over 10 question domains. Examples of our data are shown in Table 6.4.

6.4.3 Our QA System

Our system is constituted by (**i**) a search engine to retrieve questions (along with their associated tickets) similar to the new input question and (**ii**) a reranker built with state-of-the-art NLP and machine learning technology (see Chapter 3)

Question and Ticket Retrieval

We used a standard keyword-based Search Engine (SE) to retrieve a list of questions from our dataset similar to the input one. The score produced by SE is the standard cosine similarity between the vectors of the new and the candidate questions. In particular, we built our SE using Lucene TF-IDF based indexing, available in the open-source ElasticSearch platform. In order to improve the retrieval quality, we merged user request description (the question) and solution fields in a single joint text to build the ticket index. It should be noted that we only used the question text to build the query for SE as, in a real scenario, the asked question is not associated with an answer yet. For each question, in the filtered data mentioned above, we created a list of Question_original - Question_related pairs, by querying each ticket and collecting the first 10 relevant results. The obtained clustered data set resulted in a list $\langle q_{original}, q_{related} \rangle$ of 656 (tickets) $\times 10$ (retrieved questions). These pairs were annotated by a team of experts with relevant vs. irrelevant labels to create the training and test sets. For example, Table 6.4 shows a question pair: a original ticket with question and answer on the left, and a similar retrieved ticket on the right.

Reranking Pipeline

Given the initial rank provided by Se, we apply an advanced NLP pipeline to rerank the questions such that those having the highest probability to be similar to the query are ranked on top.

NLP pipeline. We used various NLP processors of TextPro Pianta et al. [2008] and embedded them in a UIMA pipeline, to analyze each ticket question as well as the questions of the tickets in the rank. The NLP components include part-of-speech tagging, chunking, named entity recognition, constituency and dependency parsing, etc. The result of the processing is used to produce syntactic representations of the ticket questions, which are then enhanced by relational links, e.g., between matching words of two questions of a pair.The resulting tree pairs are then sued to train a kernel-based reranker.

Kernel-based reranker. We train a kernel reranker function $r : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{R}$, where $\mathcal{Q}$, which tells if questions are similar or not and can be used to sort a set of questions q_r with respect to an original one q_o. The function was implemented similarly to model presented in Chapter 3: we used (i) a kernel function applied to the syntactic structure of the question pairs, together with (ii) some features capturing text similarity between questions.

6.4.4 Experiments

To evaluate our approach, we performed experiments on a dataset composed of $6,650$ pairs of ticket questions annotated with similarity judgment, i.e. Relevant and Irrelevant. We selected only questions having at least one answer in the first 10 retrieved tickets. We performed 5-fold cross-validation and used SVM-Light-TK[5] software to train 5 different reranking models, which combine both feature vectors and Tree Kernels. Then, we apply the models to all pairs of questions present in each test fold.

Results

We conducted there experiments to assess the effectiveness of the different feature sets, similarity features (Sim), TK and TK+Sim in the reranking model. The baseline is computed by means of the rank given by Lucene. Following previous work of the SemEval challenge, we evaluated our ranking with Mean Average Precision (MAP), Mean Reciprocal Rank (MRR) and Precision at k (P@k). The results are reported in Tab 6.5. As it can be seen, the best results are obtained by combining Sim and TK in the reranker, which improved the MRR and MAP of the IR baseline by 4.33 and 5.33 absolute points, respectively. In addition, P@1, P@2 and P@3 improved by 3.87, 6.07 and 6.71 absolute points, respectively. This shows the effectiveness of using syntactic structures in powerful algorithms such as TKs. We analyzed some selected errors of our system, focusing on the cases where the search engine performs better than our reranking model. We note that for each cluster of $question_{original}$-$question_{related}$ pairs, when the P@q is high, our model does not perform better than the search engine, or performs even worse. However, our reranking model always then to push relevant results on top.

6.4.5 Conclusions

In this section we have described our QA model for an Italian help desk in the field of insurance policies. Our main findings are: (i) the Italian NLP technology seems enough accurate to support advanced cQA technology based on syntactic structures; (ii) cQA model can boost the retrieval systems targeting text in Italian; and (iii) the achieved accuracy seems appropriate to create business at least in the field of help desk applications, although it should be consider that our results refer to only questions having an answer in our database.

[5] http://disi.unitn.it/moschitti/Tree-Kernel.htm

6.5 Italian QA pipeline

In this section we section we present our work on automatic feature engineering for an Italian QA system Uva and Moschitti [2015]. Our system use syntactic representations of questions and the answers pages derived by a syntactic parser. Then, apply Support Vector Machines using tree kernels to such trees for automatically generating relation syntactic patterns, which significantly improve on BM25 retrieval models.

6.5.1 Introduction

In section 3, we presented automatic feature engineering approach based on support vector machine using tree kernels for ranking answer passages. This approach consists of the following steps; (i) the set of possible candidate answers for all the input questions are retrieved by means of a search engine; (ii) each question is paired with all its candidate answer passages: positive pairs contain the correct answers and all the others are considered negative pairs; (iii) the pairs are present with two syntactic trees: one for the question and the other for the candidate answer; and (iv) an SVM classifier is trained for ranking the answer passages represented as trees. In this section, we present a similar system that can rerank answer passages for factoid questions in Italian. This system is built on top of the Unstructured Information Management Architecture (UIMA[6]) framework developed by IBM[7]. UIMA eases the tasks of assembling NLP pipelines by grouping together many text annotators to perform different types of analysis over multiple text documents. The derived analytics are then used to encode questions and answers as linguistic structures and train the reranking module for our QA pipeline.

6.5.2 Learning to rank relevant documents

QA system

The QA system has a simple architecture: it takes in input a question and retrieves a list of candidates passages from the indexed dump of the Italian Wikipedia. Such list is then reranked by its relevancy to the input question. The analysis of the question together with its candidate answers (e.g. PoS tags, Chunking, Named Entity, and many others) is performed by using the TextPro suite of NLP components for the Italian language. TextPro Pianta et al. [2008] has been integrated as a stand-alone annotator in our UIMA pipeline. The produced annotations are used to build the tree representations of both questions and answers. The resulting question/answer tree pairs are used to train a

[6]https://uima.apache.org/
[7]https://uima.apache.org/

classifier able to rank candidate passages according to their relevancy with the input question. The learned model is then used to improve the ordering of the answer passages provided by the search engine.

Answer reranking

Our goal is to rank text passages containing the correct answer higher in the list than irrelevant passages. For this purpose, we use the model we presented in Severyn et al. [2013], which is based on preference ranking Joachims [2002],. This treats the reranking problem as a binary classification task, where each problem instance is a pair, (p_1 , p_2), of question/answer pairs, i.e., $p_1 = (q, a_1)$ and $p_2 = (q, a_2)$. Positive training instances are pairs such that a_1 is a relevant passage and a_2 is an irrelevant passage otherwise (p_1 , p_2) is considered a negative example. These pairs can be used to train a binary classifier and build a reranking model. This is later used at classification time for reranking the q/a pairs representing the test instances by simply using the classifier as a voter: a positive classification is a vote for a_1 whereas negative outcome is a vote for a_2. The more an answer receives votes the higher its rank will be.

Q/A pair representation

In our model, questions and answer passages are encoded as shallow syntactic trees we introduced in Section 3. In each tree, the word lemmas constitute the terminal nodes and the Part-of-Speech(PoS) tags associated with each word constitute the pre-terminal nodes. Also, the words are organized in constituents by adding an additional layer of chunk nodes. As the chunk of text spans several words, the chunk node is connected to the PoS nodes of its words. The sentence node is located at the top level and it is linked to the chunk nodes. A ROOT node is used to connect several sentence nodes. In addition, we encoded the relationships between the question and answer trees by means of a special tag REL: if two trees share the same terminal node (word lemma) then we mark both the node parent and grandparent with the REL tag. Using the REL tag leads to more accurate results Severyn and Moschitti [2012].

Experiments

For our experiments we used factoid questions from the open-domain corpus TREC. We collected a subset of the questions from TREC 8, TREC 9, TREC 2000, TREC 2001 and TREC 2002 for a total of 1228 questions. An expert annotator translated the questions and answer gold keywords from English to Italian. The answers were searched in the Italian Wikipedia, thus we train our reranker on such data. Specifically, we split the

Wikipedia corpus in paragraphs and considered each of them as a separate document to be indexed by an off-the-shelf search engine. After performing some text cleaning, we were able to collect a total of 10 million documents. We used Lucene with the BM25 scoring function for indexing and retrieval. We trained our rerankers with the first 10 candidate answers retrieved by the search engine for each question of the train set. At test time, we retrieved a list of top 40 candidates for each test question and reranked them.

Metrics

In order to evaluate our systems we used the metrics most frequently used in QA: Precision at rank 1 (P@1) corresponds to the percentage of relevant documents ranked at position 1, Mean Reciprocal rank (MRR) and Mean Average Precision (MAP). The reported metrics are computed by conducting a 3-folds cross validation.

Results

Table 6.6 table reports the performance of the reranking models trained using different strategies:

1. the baseline model using the BM25 score of the search engine;

2. the reranker model trained only using feature vectors containing text similarity measures Bär et al. [2012];

Models	MAP	MRR	P@1
BM25	0.18	23.11	15.22
Feature vectors	0.21	26.85	18.23
Tree + Feature vectors	**0.25**	**30.74**	**22.29**

Table 6.6: The accuracy of the different ranking models

As it can be seen from the results reported in Tab 6.6, the reranking model using structural representations yields an improvement of about 3 absolute points in MAP, MRR and P@1 when compared with the vector model and about 7 absolute points when compared with the baseline model. It is interesting to note that we did not operate any adjustment of the tree kernel model, we simply build an Italian pipeline and trained our models.

6.5.3 Conclusions

In this section we showed an approach to QA requiring no manual feature engineering. Its main characteristic is the use of tree kernels for exploiting syntactic representations of question and answer passage pairs.

Chapter 7

Conclusion and Future Works

The rise of personal assistants and Conversational Agents – both task-based and open-domain – require the development of NLU components able to correctly interpret the *intent* triggered by questions expressed by user in natural language. Powerful algorithms such as (i) tree kernels and (ii) neural networks are useful for building effective systems for automatic community QA. The former work on syntactic and structures, while the latter are able to learn distributional representations currying out contextual word information. Inspired by these research lines and ideas we develop accurate models for solving the tasks involved in community QA. In Chapter 2 we introduce the two machine learning algorithms that at the base of many works referred in this thesis: Kernel-based Support Vector Machines and Neural Networks.

In Chapter 3 we demonstrate how to use structural kernels for relational text inference problems involved in building automatic systems for community Question Answering. Results proved that kernels were able to deliver state-of-the-art performances on Question-Answer similarity and Question-Question similarity tasks.

Chapter 4 presents our models based on the deep-learning architecture for community QA. First, we present a new end-to-end Neural Network model that can be *jointly* trained to solve all the three tasks together. It does not require task-specific processing pipelines and, in addition, its parameters are shared among the models used for solving the different tasks. Secondly, we present a model aimed at injecting syntactic knowledge into neural networks, while maintaining its architecture very simple. More specifically, we showed that training a network on data annotated by a Tree Kernel-based SVM classifier improve the performance on the question-question similarity task. Although this approach deserves further study, preliminary experiments showed that approaches aiming at transferring syntactic knowledge to NNs may be very beneficial for solving the final task.

Chapter 5 introduces the most original contribution of our thesis: supervised clustering of forum questions and FAQs into intents for fast bootstrapping of NLU pipelines. This work combines two research lines, i.e. structured output and relational text inference, in order to build models able to carry out to infer the most likely structure representing clusters of questions where relations are described by similarity features between pairs. We showed that the result of this work may help dialog system engineers in their task of rapidly prototyping intent ontologies.

In Chapter 6 we listed our NLP systems and demos. First, we present our work on building multi-lingual UIMA-based NLP pipeline for processing text in any language. Then, we describe our effort for filling the accuracy gap of constituency parsing models between Italian an English. Finally, we present two QA models that make use of structural representation for reranking candidate answers to new user questions. The first pipeline was used for answering factoid questions in Italian language by using Wikipedia as corpus from which we retrieve candidate answer passages. The the second pipeline has been employed by a company to automate the work of a help desk service. It uses the cQA paradigm for answering a user question: (i) search for similar questions among those already asked by previous users and (ii) return the answers for the related questions. Answers are searched among tickets stored in the ticketing system used by the Help Desks services.

Bibliography

Agerri, Rodrigo; Bermudez, Josu, and Rigau, German. Ixa pipeline: Efficient and ready to use multilingual nlp tools. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA), 2014. URL `http://www.lrec-conf.org/proceedings/lrec2014/pdf/775_Paper.pdf`.

Agirre, Eneko; Cer, Daniel; Diab, Mona, and Gonzalez-Agirre, Aitor. Semeval-2012 task 6: A pilot on semantic textual similarity. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 385–393. Association for Computational Linguistics, 2012. URL `http://www.aclweb.org/anthology/S12-1051`.

Agirre, Eneko; Cer, Daniel; Diab, Mona; Gonzalez-Agirre, Aitor, and Guo, Weiwei. *sem 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43. Association for Computational Linguistics, 2013. URL `http://www.aclweb.org/anthology/S13-1004`.

Agirre, Eneko; Banea, Carmen; Cardie, Claire; Cer, Daniel; Diab, Mona; Gonzalez-Agirre, Aitor; Guo, Weiwei; Mihalcea, Rada; Rigau, German, and Wiebe, Janyce. Semeval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 81–91. Association for Computational Linguistics, 2014. doi: 10.3115/v1/S14-2010. URL `http://www.aclweb.org/anthology/S14-2010`.

Allison, Lloyd and Dix, Trevor. A bit-string longest-common-subsequence algorithm. *Information Processing Letters*, 23(6):305–310, December 1986. ISSN 0020-0190.

Bär, Daniel; Biemann, Chris; Gurevych, Iryna, and Zesch, Torsten. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 435–440. Association for Computational Linguistics, 2012.

Barlacchi, Gianni; Nicosia, Massimo, and Moschitti, Alessandro. Learning to rank answer candidates for automatic resolution of crossword puzzles. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 39–48. Association for Computational Linguistics, 2014. doi: 10.3115/v1/W14-1605. URL `http://aclweb.org/anthology/W14-1605`.

Barrón-Cedeño, Alberto; Filice, Simone; Da San Martino, Giovanni; Joty, Shafiq; Màrquez, Lluís; Nakov, Preslav, and Moschitti, Alessandro. Thread-level information for comment classification in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguis-*

tics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pages 687–693. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-2113. URL `http://aclweb.org/anthology/P15-2113`.

Barrón-Cedeño, Alberto; Da San Martino, Giovanni; Joty, Shafiq; Moschitti, Alessandro; Al-Obaidli, Fahad; Romeo, Salvatore; Tymoshenko, Kateryna, and Uva, Antonio. Convkn at semeval-2016 task 3: Answer and question selection for question answering on arabic and english fora. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 896–903. Association for Computational Linguistics, 2016. doi: 10.18653/v1/S16-1138. URL `http://aclweb.org/anthology/S16-1138`.

Bengio, Yoshua; Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Bennett, Kristin P and Mangasarian, Olvi L. Robust linear programming discrimination of two linearly inseparable sets. *Optimization methods and software*, 1(1):23–34, 1992.

Bian, Weijie; Li, Si; Yang, Zhao; Chen, Guang, and Lin, Zhiqing. A compare-aggregate model with dynamic-clip attention for answer selection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 1987–1990, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4918-5. doi: 10.1145/3132847.3133089. URL `http://doi.acm.org/10.1145/3132847.3133089`.

Biggio, Silvana Marianela Bernaola; Giuliano, Claudio; Poesio, Massimo; Versley, Yannick; Uryupina, Olga, and Zanoli, Roberto. Local entity detection and recognition task. *Proc. of Evalita*, 9, 2009.

Bosch, Antal van den; Busser, Bertjan; Canisius, Sander, and Daelemans, Walter. An efficient memory-based morphosyntactic tagger and parser for dutch. *LOT Occasional Series*, 7:191–206, 2007.

Bosco, Cristina and Mazzei, Alessandro. The evalita 2011 parsing task: the constituency track. *Working Notes of EVALITA*, 2011.

Bosco, Cristina; Mazzei, Alessandro, and Lombardo, Vincenzo. Evalita parsing task: an analysis of the first parsing system contest for italian. *Intelligenza artificiale*, 12:30–33, 2007.

Bosco, Cristina; Mazzei, Alessandro, and Lombardo, Vincenzo. Evalita' 09 parsing task: constituency parsers and the penn format for italian. *Proceedings of EVALITA*, 9:1794–1801, 2009.

Bosco, Cristina; Mazzei, Alessandro, and Lavelli, Alberto. Looking back to the evalita constituency parsing task: 2007-2011. In *Evaluation of Natural Language and Speech Tools for Italian*, pages 46–57. Springer, 2013.

Bottou, Léon. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Breck, Eric; Choi, Yejin, and Cardie, Claire. Identifying expressions of opinion in context. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2683–2688, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1625275.1625707`.

Broscheit, Samuel; Poesio, Massimo; Ponzetto, Simone Paolo; Rodriguez, Kepa Joseba; Romano, Lorenza; Uryupina, Olga; Versley, Yannick, and Zanoli, Roberto. Bart: A multilingual anaphora resolution system. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 104–107, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1859664.1859685`.

Cai, Jie and Strube, Michael. Evaluation metrics for end-to-end coreference resolution systems. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, SIGDIAL '10, pages 28–36, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. ISBN 978-1-932432-85-5. URL `http://dl.acm.org/citation.cfm?id=1944506.1944511`.

Caputo, Annalina; de Gemmis, Marco; Lops, Pasquale; Lovecchio, Francesco; Manzari, Vito, and Spa, Acquedotto Pugliese AQP. Overview of the evalita 2016 question answering for frequently asked questions (qa4faq) task. In *CLiC-it/EVALITA*, 2016.

Caruana, Rich. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Cer, Daniel; Diab, Mona; Agirre, Eneko; Lopez-Gazpio, Inigo, and Specia, Lucia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14. Association for Computational Linguistics, 2017. doi: 10.18653/v1/S17-2001. URL `http://www.aclweb.org/anthology/S17-2001`.

Charniak, Eugene. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics, 2000.

Charniak, Eugene and Johnson, Mark. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics, 2005.

Chen, Ruey-Cheng; Yulianti, Evi; Sanderson, Mark, and Croft, W. Bruce. On the benefit of incorporating external features in a neural architecture for answer sentence selection. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1017–1020, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5022-8. doi: 10.1145/3077136.3080705. URL `http://doi.acm.org/10.1145/3077136.3080705`.

Cho, Kyunghyun; Van Merriënboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Cohen, Daniel and Croft, W. Bruce. End to end long short term memory networks for non-factoid question answering. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 143–146, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4497-5. doi: 10.1145/2970398.2970438. URL `http://doi.acm.org/10.1145/2970398.2970438`.

Collins, Michael and Duffy, Nigel. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 263–270. Association for Computational Linguistics, 2002.

Collins, Michael and Koo, Terry. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70, 2005.

Collobert, Ronan; Weston, Jason; Bottou, Léon; Karlen, Michael; Kavukcuoglu, Koray, and Kuksa, Pavel. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011a. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2078186`.

Collobert, Ronan; Weston, Jason; Bottou, Léon; Karlen, Michael; Kavukcuoglu, Koray, and Kuksa, Pavel. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011b.

Cortes, Corinna and Vapnik, Vladimir. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Coucke, Alice; Ball, Adrien; Delpuech, Clement; Doumouro, Clement; Raybaud, Sylvain; Gisselbrecht, Thibault, and Dureau, Joseph. Benchmarking natural language understanding systems: Google, Facebook, Microsoft, Amazon, and Snips, 2017. URL `https://medium.com/snips-ai/benchmarking-natural-language-understanding-systems-google-facebook-microsoft-and-snips-2b8ddcf9fb19`.

Croce, Danilo; Filice, Simone; Castellucci, Giuseppe, and Basili, Roberto. Deep learning in semantic kernel spaces. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 345–354, 2017.

Cunningham, Hamish; Maynard, Diana, and Bontcheva, Kalina. *Text processing with gate*. Gateway Press CA, 2011.

Da San Martino, Giovanni; Barrón Cedeño, Alberto; Romeo, Salvatore; Uva, Antonio, and Moschitti, Alessandro. Learning to re-rank questions in community question answering using advanced features. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1997–2000. ACM, 2016.

Das, Arpita; Yenala, Harish; Chinnakotla, Manoj, and Shrivastava, Manish. Together we stand: Siamese networks for similar question retrieval. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 378–387. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1036. URL `http://aclweb.org/anthology/P16-1036`.

De Rooij, Ork; Vishneuski, Andrei; De Rijke, Maarten, and others, . xtas: Text analysis in a timely manner. In *Dir*, volume 2012, page 12th. Citeseer, 2012.

Duchi, John; Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

Eckart de Castilho, Richard and Gurevych, Iryna. A broad-coverage collection of portable nlp components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University. URL `http://www.aclweb.org/anthology/W14-5201`.

Fernandes, Eraldo Rezende; dos Santos, Cícero Nogueira, and Milidiú, Ruy Luiz. Latent trees for coreference resolution. *Computational Linguistics*, 40(4):801–835, 2014.

Ferrucci, David and Lally, Adam. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

Ferrucci, David; Brown, Eric; Chu-Carroll, Jennifer; Fan, James; Gondek, David; Kalyanpur, Aditya A; Lally, Adam; Murdock, J William; Nyberg, Eric; Prager, John, and others, . Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

Filice, Simone; Croce, Danilo; Moschitti, Alessandro, and Basili, Roberto. Kelp at semeval-2016 task 3: Learning semantic relations between questions and answers. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1116–1123. Association for Computational Linguistics, 2016a. doi: 10.18653/v1/S16-1172. URL `http://aclweb.org/anthology/S16-1172`.

Filice, Simone; Croce, Danilo; Moschitti, Alessandro, and Basili, Roberto. KeLP at SemEval-2016 Task 3: Learning semantic relations between questions and answers. In *Proc. of the 10th Intl. Workshop on Semantic Evaluation*, SemEval '16, San Diego, California, USA, 2016b.

Filice, Simone; Martino, Giovanni Da San, and Moschitti, Alessandro. KeLP at SemEval-2017 task 3: Learning pairwise patterns in community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, SemEval '17, pages 327–334, Vancouver, Canada, 2017.

Finley, Thomas and Joachims, Thorsten. Supervised clustering with support vector machines. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 217–224, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102379. URL `http://portal.acm.org/citation.cfm?id=1102351.1102379`.

Franco-Salvador, Marc; Kar, Sudipta; Solorio, Thamar, and Rosso, Paolo. Uh-prhlt at semeval-2016 task 3: Combining lexical and semantic-based features for community question answering. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 814–821. Association for Computational Linguistics, 2016. doi: 10.18653/v1/S16-1126. URL `http://aclweb.org/anthology/S16-1126`.

Ghosh, Debanjan and Muresan, Smaranda. Relation classification using entity sequence kernels. In *Proceedings of COLING 2012: Posters*, pages 391–400. The COLING 2012 Organizing Committee, 2012. URL `http://aclweb.org/anthology/C12-2039`.

Giuliano, Claudio; Lavelli, Alberto, and Romano, Lorenza. Relation extraction and the influence of automatic named-entity recognition. *ACM Trans. Speech Lang. Process.*, 5(1):2:1–2:26, December 2007. ISSN 1550-4875. doi: 10.1145/1322391.1322393. URL `http://doi.acm.org/10.1145/1322391.1322393`.

Goldberg, Yoav. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

Goodfellow, Ian J; Mirza, Mehdi; Xiao, Da; Courville, Aaron, and Bengio, Yoshua. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Graves, Alex; Fernández, Santiago, and Schmidhuber, Jürgen. Bidirectional lstm networks for improved phoneme classification and recognition. In *International Conference on Artificial Neural Networks*, pages 799–804. Springer, 2005.

Graves, Alex; Jaitly, Navdeep, and Mohamed, Abdel-rahman. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

Haponchyk, Iryna; Uva, Antonio; Yu, Seunghak; Uryupina, Olga, and Moschitti, Alessandro. Supervised clustering of questions into intents for dialog system applications. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2310–2321. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/D18-1254`.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997. URL `https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735`.

Hu, Zhiting; Ma, Xuezhe; Liu, Zhengzhong; Hovy, Eduard, and Xing, Eric. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2410–2420. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1228. URL `http://www.aclweb.org/anthology/P16-1228`.

Jaccard, Paul. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 1901.

Joachims, Thorsten. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

Johansson, Richard and Moschitti, Alessandro. Extracting opinion expressions and their polarities – exploration of pipelines and joint models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 101–106. Association for Computational Linguistics, 2011. URL `http://aclweb.org/anthology/P11-2018`.

Johansson, Richard and Nugues, Pierre. Dependency-based semantic role labeling of propbank. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 69–78. Association for Computational Linguistics, 2008. URL `http://aclweb.org/anthology/D08-1008`.

Kalchbrenner, Nal; Grefenstette, Edward, and Blunsom, Phil. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665. Association for Computational Linguistics, 2014. doi: 10.3115/v1/P14-1062. URL `http://aclweb.org/anthology/P14-1062`.

Karpathy, Andrej; Toderici, George; Shetty, Sanketh; Leung, Thomas; Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

Kim, Yoon. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1181. URL `http://aclweb.org/anthology/D14-1181`.

Kimeldorf, George S and Wahba, Grace. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41(2):495–502, 1970.

Kingma, Diederik P and Ba, Jimmy Lei. Adam: Amethod for stochastic optimization. In *Proc. 3rd Int. Conf. Learn. Representations*, 2014.

Krizhevsky, Alex; Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL `http://dl.acm.org/citation.cfm?id=2999134.2999257`.

Lavelli, Alberto. The berkeley parser at the evalita 2011 constituency parsing task. In *Working Notes of EVALITA*, 2011.

Lavelli, Alberto and Corazza, Anna. The berkeley parser at the evalita 2009 constituency parsing task. In *EVALITA 2009 Workshop on Evaluation of NLP Tools for Italian*, 2009.

Lawrence, Steve; Giles, C Lee; Tsoi, Ah Chung, and Back, Andrew D. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

LeCun, Yann; Bottou, Léon; Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, Jiwei; Luong, Thang; Jurafsky, Dan, and Hovy, Eduard. When are tree structures necessary for deep learning of representations? In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2304–2314. Association for Computational Linguistics, 2015. doi: 10.18653/v1/D15-1278. URL `http://aclweb.org/anthology/D15-1278`.

Linzen, Tal; Dupoux, Emmanuel, and Goldberg, Yoav. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016. URL `http://aclweb.org/anthology/Q16-1037`.

Liu, Xiaodong; Gao, Jianfeng; He, Xiaodong; Deng, Li; Duh, Kevin, and Wang, Ye-Yi. Representation learning using multi-task deep neural networks for seman tic classification and information retrieval. In *Proc. NAACL*, 2015.

Lodhi, Huma; Saunders, Craig; Shawe-Taylor, John; Cristianini, Nello, and Watkins, Chris. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

Luo, Xiaoqiang. On coreference resolution performance metrics. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 25–32, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1220575.1220579. URL `http://dx.doi.org/10.3115/1220575.1220579`.

Lyon, Caroline; Malcolm, James, and Dickerson, Bob. Detecting short passages of similar text in large document collections. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, EMNLP, pages 118–125, Pittsburgh, PA, USA, 2001.

Manning, Christopher; Surdeanu, Mihai; Bauer, John; Finkel, Jenny; Bethard, Steven, and McClosky, David. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. Association for Computational Linguistics, 2014. doi: 10.3115/v1/P14-5010. URL `http://aclweb.org/anthology/P14-5010`.

Marcus, Mitchell P; Marcinkiewicz, Mary Ann, and Santorini, Beatrice. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Martino, Giovanni Da San; Barrón-Cedeño, Alberto; Romeo, Salvatore; Moschitti, Alessandro; Joty, Shafiq; Obaidli, Fahad A Al; Tymoshenko, Kateryna, and Uva, Antonio. Addressing community question answering in english and arabic. *arXiv preprint arXiv:1610.05522*, 2016.

McClosky, David; Charniak, Eugene, and Johnson, Mark. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics, 2006.

Meij, Edgar; Weerkamp, Wouter, and De Rijke, Maarten. Adding semantics to microblog posts. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 563–572. ACM, 2012.

Mihaylova, Tsvetomila; Gencheva, Pepa; Boyanov, Martin; Yovcheva, Ivana; Mihaylov, Todor; Hardalov, Momchil; Kiprov, Yasen; Balchev, Daniel; Koychev, Ivan; Nakov, Preslav; Nikolova, Ivelina, and Angelova, Galia. Super team at semeval-2016 task 3: Building a feature-rich system for community question answering. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 836–843. Association for Computational Linguistics, 2016. doi: 10.18653/v1/S16-1129. URL `http://aclweb.org/anthology/S16-1129`.

Mikolov, Tomas; Yih, Wen-tau, and Zweig, Geoffrey. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT '13, pages 746–751, Atlanta, GA, USA, 2013.

Moschitti, Alessandro. Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning*, pages 318–329. Springer, 2006.

Moschitti, Alessandro and Zanzotto, Fabio Massimo. Fast and effective kernels for relational learning from texts. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 649–656, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273578. URL `http://doi.acm.org/10.1145/1273496.1273578`.

Mueller, Jonas and Thyagarajan, Aditya. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2786–2792. AAAI Press, 2016. URL `http://dl.acm.org/citation.cfm?id=3016100.3016291`.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Nakov, Preslav; Màrquez, Lluís; Moschitti, Alessandro; Magdy, Walid; Mubarak, Hamdy; Freihat, abed Alhakim; Glass, Jim, and Randeree, Bilal. Semeval-2016 task 3: Community question answering. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 525–545. Association for Computational Linguistics, 2016a. doi: 10.18653/v1/S16-1083. URL `http://aclweb.org/anthology/S16-1083`.

Nakov, Preslav; Màrquez, Lluís; Moschitti, Alessandro; Magdy, Walid; Mubarak, Hamdy; Freihat, abed Alhakim; Glass, Jim, and Randeree, Bilal. Semeval-2016 task 3: Community question answering. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 525–545. Association for Computational Linguistics, 2016b. doi: 10.18653/v1/S16-1083. URL `http://aclweb.org/anthology/S16-1083`.

Nakov, Preslav; Hoogeveen, Doris; Màrquez, Lluís; Moschitti, Alessandro; Mubarak, Hamdy; Baldwin, Timothy, and Verspoor, Karin. Semeval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 27–48. Association for Computational Linguistics, 2017. doi: 10.18653/v1/S17-2003. URL `http://aclweb.org/anthology/S17-2003`.

Ng, Andrew Y.; Jordan, Michael I., and Weiss, Yair. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 849–856, Cambridge, MA, USA, 2001. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2980539.2980649`.

Nguyen, Thien Huu; Plank, Barbara, and Grishman, Ralph. Semantic representations for domain adaptation: A case study on the tree kernel-based method for relation extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 635–644. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-1062. URL `http://aclweb.org/anthology/P15-1062`.

Nicosia, Massimo; Filice, Simone; Barrón-Cedeño, Alberto; Saleh, Iman; Mubarak, Hamdy; Gao, Wei; Nakov, Preslav; Da San Martino, Giovanni; Moschitti, Alessandro; Darwish, Kareem; Màrquez, Lluís; Joty, Shafiq, and Magdy, Walid. Qcri: Answer selection for community question answering - experiments for arabic and english. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 203–209. Association for Computational Linguistics, 2015. doi: 10.18653/v1/S15-2036. URL `http://aclweb.org/anthology/S15-2036`.

Padró, Lluís and Stanilovsky, Evgeny. Freeling 3.0: Towards wider multilinguality. In *LREC2012*, 2012.

Parikh, Ankur; Täckström, Oscar; Das, Dipanjan, and Uszkoreit, Jakob. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1244. URL `http://aclweb.org/anthology/D16-1244`.

Pascanu, Razvan; Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

Petrov, Slav and Klein, Dan. Improved inference for unlexicalized parsing. In *HLT-NAACL*, volume 7, pages 404–411, 2007.

Pianta, Emanuele; Girardi, Christian, and Zanoli, Roberto. The textpro tool suite. In *LREC*. Citeseer, 2008.

Plank, Barbara and Moschitti, Alessandro. Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1498–1507. Association for Computational Linguistics, 2013. URL `http://aclweb.org/anthology/P13-1147`.

Rao, Jinfeng; He, Hua, and Lin, Jimmy. Noise-contrastive estimation for answer selection with deep neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1913–1916, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983872. URL `http://doi.acm.org/10.1145/2983323.2983872`.

Rao, Jinfeng; He, Hua, and Lin, Jimmy. Experiments with convolutional neural network models for answer selection. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1217–1220, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5022-8. doi: 10.1145/3077136.3080648. URL `http://doi.acm.org/10.1145/3077136.3080648`.

Roth, Dan and Yih, Wen-tau. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*, 2004. URL `http://aclweb.org/anthology/W04-2401`.

Rumelhart, David E.; Hinton, Geoffrey E., and Williams, Ronald J. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL `http://dl.acm.org/citation.cfm?id=65669.104451`.

Schuster, Mike and Paliwal, Kuldip K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Sequiera, Royal; Baruah, Gaurav; Tu, Zhucheng; Mohammed, Salman; Rao, Jinfeng; Zhang, Haotian, and Lin, Jimmy J. Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering. *CoRR*, abs/1707.07804, 2017. URL `http://arxiv.org/abs/1707.07804`.

Severyn, Aliaksei and Moschitti, Alessandro. Structural relationships for large-scale learning of answer re-ranking. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 741–750. ACM, 2012.

Severyn, Aliaksei and Moschitti, Alessandro. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 373–382, New York, NY, USA, 2015a. ACM. ISBN 978-1-4503-3621-5. doi: 10.1145/2766462.2767738. URL `http://doi.acm.org/10.1145/2766462.2767738`.

Severyn, Aliaksei and Moschitti, Alessandro. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 373–382. ACM, 2015b.

Severyn, Aliaksei and Moschitti, Alessandro. Modeling relational information in question-answer pairs with convolutional neural networks. *arXiv preprint arXiv:1604.01178*, 2016.

Severyn, Aliaksei; Nicosia, Massimo, and Moschitti, Alessandro. Learning adaptable patterns for passage reranking. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 75–83. Association for Computational Linguistics, 2013. URL `http://aclweb.org/anthology/W13-3509`.

Severyn, Aliaksei; Moschitti, Alessandro; Uryupina, Olga; Plank, Barbara, and Filippova, Katja. Opinion mining on youtube. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1252–1261. Association for Computational Linguistics, 2014. doi: 10.3115/v1/P14-1118. URL `http://aclweb.org/anthology/P14-1118`.

Shawe-Taylor, John; Cristianini, Nello, and others, . *Kernel methods for pattern analysis*. Cambridge university press, 2004.

Socher, Richard; Perelygin, Alex; Wu, Jean; Chuang, Jason; Manning, Christopher D; Ng, Andrew, and Potts, Christopher. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

Srivastava, Nitish; Hinton, Geoffrey; Krizhevsky, Alex; Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Szalkai, Balázs. An implementation of the relational k-means algorithm. *CoRR*, abs/1304.6899, 2013. URL `http://arxiv.org/abs/1304.6899`.

Tai, Kai Sheng; Socher, Richard, and Manning, Christopher D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Tan, Ming; dos Santos, Cicero; Xiang, Bing, and Zhou, Bowen. Improved representation learning for question answer matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 464–473. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1044. URL `http://aclweb.org/anthology/P16-1044`.

Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

Tymoshenko, Kateryna and Moschitti, Alessandro. Assessing the impact of syntactic and semantic structures for answer passages reranking. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1451–1460, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3794-6. doi: 10.1145/2806416.2806490. URL `http://doi.acm.org/10.1145/2806416.2806490`.

Tymoshenko, Kateryna; Moschitti, Alessandro, and Severyn, Aliaksei. Encoding semantic resources in syntactic structures for passage reranking. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 664–672. Association for Computational Linguistics, 2014. doi: 10.3115/v1/E14-1070. URL `http://aclweb.org/anthology/E14-1070`.

Tymoshenko, Kateryna; Bonadiman, Daniele, and Moschitti, Alessandro. Learning to rank non-factoid answers: Comment selection in web forums. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 2049–2052, New York, NY, USA, 2016a. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983906. URL `http://doi.acm.org/10.1145/2983323.2983906`.

Tymoshenko, Kateryna; Bonadiman, Daniele, and Moschitti, Alessandro. Convolutional neural networks vs. convolution kernels: Feature engineer ing for answer sentence reranking. In *Proceedings of NAACL-HLT*, pages 1268–1278, 2016b.

Tymoshenko, Kateryna; Bonadiman, Daniele, and Moschitti, Alessandro. Learning to rank non-factoid answers: Comment selection in web forums. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2049–2052. ACM, 2016c.

Uryupina, Olga; Saha, Sriparna; Ekbal, Asif, and Poesio, Massimo. Multi-metric optimization for coreference: The unitn / iitp / essex submission to the 2011 conll shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 61–65. Association for Computational Linguistics, 2011. URL `http://aclweb.org/anthology/W11-1908`.

Uryupina, Olga; Moschitti, Alessandro, and Poesio, Massimo. Bart goes multilingual: The unitn / essex submission to the conll-2012 shared task. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 122–128. Association for Computational Linguistics, 2012. URL `http://aclweb.org/anthology/W12-4515`.

Uryupina, Olga; Plank, Barbara; Barlacchi, Gianni; Valverde-Albacete, Francisco J; Tsagkias, Manos; Uva, Antonio, and Moschitti, Alessandro. Limosine pipeline: Multilingual uima-based nlp platform. In *Proceedings of ACL-2016 System Demonstrations*, pages 157–162. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-4027. URL `http://aclweb.org/anthology/P16-4027`.

Uva, Antonio and Moschitti, Alessandro. Tree kernels-based discriminative reranker for italian constituency parsers. *CLiC it*, page 303, 2016.

Uva, Antonio; Storch, Valerio; Carrino, Casimiro; Di Iorio, Ugo, and Moschitti, Alessandro. Commercial applications through community question answering technology. *CLiC-it 2017 11-12 December 2017, Rome*, page 333, 2017.

Uva, Antonio; Bonadiman, Daniele, and Moschitti, Alessandro. Injecting relational structural representation in neural networks for question similarity. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 285–291. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/P18-2046`.

Uva, Antonio E and Moschitti, Alessandro. Automatic feature engineering for italian question answering systems. In *IIR*, 2015.

Vapnik, Vladimir. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

Versley, Yannick; Ponzetto, Simone Paolo; Poesio, Massimo; Eidelman, Vladimir; Jern, Alan; Smith, Jason; Yang, Xiaofeng, and Moschitti, Alessandro. Bart: A modular toolkit for coreference resolution. In *Proceedings of the ACL-08: HLT Demo Session*, pages 9–12. Association for Computational Linguistics, 2008. URL `http://aclweb.org/anthology/P08-4003`.

Wang, Zhiguo; Hamza, Wael, and Florian, Radu. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 4144–4150. AAAI Press, 2017. ISBN 978-0-9992411-0-3. URL `http://dl.acm.org/citation.cfm?id=3171837.3171865`.

Wise, Michael J. Yap3: Improved detection of similarities in computer program and other texts. In *ACM SIGCSE Bulletin*, volume 28, pages 130–134. ACM, 1996.

Yin, Wenpeng; Schütze, Hinrich; Xiang, Bing, and Zhou, Bowen. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, 4:259–272, 2016. URL `http://aclweb.org/anthology/Q16-1019`.

Yu, Chun-Nam John and Joachims, Thorsten. Learning structural svms with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1169–1176, New York, NY, USA, June 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553523. URL `http://doi.acm.org/10.1145/1553374.1553523`.

Yu, Lei; Hermann, Karl Moritz; Blunsom, Phil, and Pulman, Stephen. Deep learning for answer sentence selection. *CoRR*, 2014.

Zhao, Ying and Karypis, George. Criterion functions for document clustering: Experiments and analysis. Technical report, 2002.