The Palmisano system: measuring similarity between questions with a Takelab implementation

Lisa Vitolo
University La Sapienza of Rome
lisavitolo90@gmail.com

Abstract

The Palmisano system is going to be a live Question Answering system operating on Twitter. Starting from the original question posed by the user, Palmisano searches similar questions, complete with their answers, from QA databases. Extracted questions need to be evaluated so that the most similar to the original one is used to generate the final answer. This project developed a semantic similarity system based on the TakeLab *simple* system, which shall be used to assign a score to extracted questions, measuring their semantic similarity to the original one. TakeLab, however, was not designed for tweets, and so I will present a discussion on the possible issues and improvements for making this an efficient similarity system targeted to Twitter usage.

1 Introduction

A Semantic Textual Similarity system aims at assigning a degree of semantic similarity to two sentences. This is a problem that has many applications in the Natural Language Processing field. It has been presented for the first time as a pilot task in the Semeval 2012 (Task 6), and proposed again in the Semeval 2013. You can find a list of submitted systems, data and proceedings, in their official page: http://www.cs.york.ac.uk/semeval-2012/task6/.

Semantic similarity can be easily seen as a regression problem. With regression we try to learn a function $f : X \rightarrow \mathbb{R}$ where X is the sample space, in our case the infinite space of all the possible sentence pairs written in English. The values of this function are in the real domain. In this problem we actually talk about a restricted domain, lying in the range [0, 5]. This, however, doesn't affect its nature and possible solutions.

The remainder of this paper is structured in this way: section number 2 presents the TakeLab simple system, outlining all of its steps with references to recent works and studies in the field. The Related work section expands those references in order to give a summary of the state of the art for semantic similarity. Next, some practical details and issues regarding this project implementation are presented. Section 5 is concerned entirely with results and comparisons. Finally, section 6 delves into the task of applying this or similar systems to question answering in Twitter.

2 TakeLab

TakeLab (Saric 2012) is one of the STS systems presented for solving the Semeval 2012 task. Using the evaluation measures proposed in the presentation paper (Agirre 2012), and discussed in the Results, it ranked among the top fifth.

As mentioned above, solving the STS problem can be seen as solving a regression task. TakeLab, like most systems presented for the contest, embraced the machine learning approach, and Figure 1 shows a quick scheme of its organization.

TakeLab actually presented two systems, called *simple* and *syntax*. Their difference lies in the set of features they use, although some basic features are shared. The *syntax* system tries to exploit more syntactic dependencies and similarities (for instance looking at the dependency parse trees). The *simple* system is more focused on numerical values coming directly from "bag of words" features, or from semantic-related tools such as WordNet.

On average the *simple* system slightly outperformed *syntax*, so I decided to implement this one for the project.

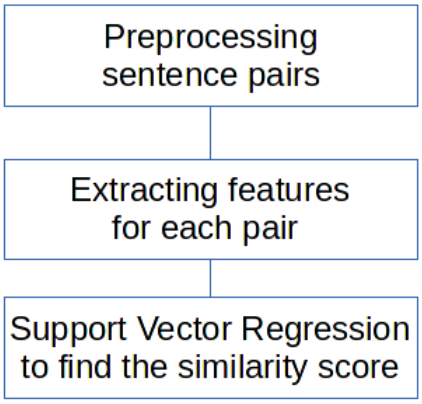

Figure 1: Organization of TakeLab

2.1 Similarity Scores

Similarity scores range from 0 (totally different sentences) to 5 (identical sentences). This is maintained in my implementation too.

Both the training and test set are shipped with a Gold Standard score for each sentence pair. However, semantic similarity is a concept that remains vague and open to discussions or contradictions. [CIT] recaps all the issues encountered while annotating.

2.2 Preprocessing steps

Before features are extracted, sentence pairs are preprocessed with the following rules:

- tokenization, Part-Of-Speech tagging and lemmatization;
- hyphens and slashes are removed;
- angular brackets appearing at the beginning or at the end of tokens are removed;
- some verb abbreviations typical of the English language, such as "n't" (see "cant"', "don't",...), are expanded to their non-abbreviated form;
- if a compound word appears as a single token in one sentence, but as two consecutive tokens in the other, then it is replaced by one token in both. This is useful since having tokens in common between the two sentences has a visible effect on more than one feature.
- stopwords are removed using a small list. Stopwords are words that are so common in the language that they bear no additional information about the semantics, and so can be ignored completely without loss of generality. Additionally, if two sentences have a lot of stopwords in common, this doesn't increase much their probability of being similar, so leaving them in place may actually confuse the system.

The first step is carried on in two ways: for the user question, by relying on the preceding stages of the Palmisano pipeline. For the questions extracted from the database the Stanford Core NLP pipeline is invoked.

All the other steps are taken care of internally, without use of external libraries.

2.3 Features

An introductive node: the overlap between two generic sets is a measure of how much elements they have in common. It ranges from 0 (no common element) to 1 (identical sets). Formally, for sets S_1 and S_2 their overlap is defined as:

$$o(S_1, S_2) = 2 \left(\frac{|S_1|}{|S_1 \cap S_2|} + \frac{|S_2|}{|S_1 \cap S_2|} \right)^{-1}$$

where $|S|$ is the set's cardinality.

NGram Overlap

Here we build two sets, one for each sentence, containing all of its unigrams. The first feature is the overlap degree between these two sets. The procedure is repeated for consecutive bigrams and trigrams, that is sequences of 2 and 3 tokens.

Three additional features are obtained from the same overlaps, but instead of using tokens as they are found in the sentences I use their lemmatization. This is not present in the original Takelab implementation, but it was added by the authors as a later improvement.

This overlap, or degree of coverage between the two sentences, deals with "raw" words or lemmas. Now we move to more sophisticated types of coverages, who try to take into account other semantic properties of the tokens in order to obtain a more realistic value.

WordNet-augmented word overlap

Here we try to take into account different words that have however a quite close meaning. Such

"closeness" is measured from the path length similarity of WordNet. Let's define the WordNet-augmented coverage:

$$P_{WN}(S_1, S_2) = \frac{1}{|S_2|} \sum_{w \in S_1} score(w_1, S_2)$$

$$score(w, S) = \begin{cases} 1 & w \in S \\ \max_{w' \in S} sim(w, w') & w \notin S \end{cases}$$

where sim is the path-length similarity mentioned before. The complete feature is a harmonic mean between the two correspondent coverages:

$$wawo = harmonic\left(P_{WN}(S_1, S_2),\ P_{WN}(S_2, S_1)\right)$$

Weighted word overlap

Here we assign more importance to words expressing more content inside a sentence. We define the information content of a token as:

$$ic(w) = \ln \frac{\sum_{w'} freq(w')}{freq(w)}$$

It's a measure of how frequently the token appears, compared with the total frequency of all the possible tokens. In order to obtain the frequency counts a corpus is needed. The Google Books NGram Corpus collects and counts tokens from millions of books written in English over a span of several decades.

Next we define the weighted word coverage of one sentence with respect to another:

$$wwc(S_1, S_2) = \frac{\sum_{w \in S_1 \cap S_2} ic(w)}{\sum_{w' \in S_2} ic(w')}$$

The feature added here is again an harmonic mean:

$$wwo = harmonic(wwc(S_1, S_2),\ wwc(S_2, S_1))$$

Vector space sentence similarity

In computational linguistics, the distributional hypothesis states that words that tend to appear in the same contexts are likely to be similar; this hypothesis is usually confirmed by the trends in human annotations. In TakeLab, words and the contexts are expressed with a term-document matrix. Word vectors are then summed together in order to compose sentence vectors. The first feature is a cosine similarity between the two sentence vectors. For the second feature, word vectors are weighted with the information content of the word before creating the sentence vectors.

Since the information content is considered a reliable measure of the importance of one word inside the sentence, it is natural to weight another feature with it.

Normalized differences

Two normalized differences between the sentences are added: sentence length and aggregate word information content.

$$slen = \frac{|len(S_1) - len(S_2)|}{\max\left(len(S_1),\ len(S_2)\right) + e^{-5}}$$

and similarly for the sum of all the information contents.

Number overlaps

This describes three features dealing entirely with numeric tokens, on the basis that the human annotators for the training set often gave low similarity scores to sentence pairs where the numbers differed a lot.

Named Entity features

Two types of Named Entity have been considered: words starting with a capital letter, and stock index symbols. In the first case of course there is a loss of precision due to the fact that not all words starting with a capital letter are actually named entities, but perhaps the authors thought it was a good tradeoff with respect to using an external recognizer.

Among the training sets provided for the contest, economics seems to be a recurring topic, and thus stock index symbols are mentioned in the sentences. Another overlap feature in the system is dedicated to them.

2.4 10-fold cross-validation

The Support Vector Regressor needs some parameters that control its internal behaviour; their number and nature depends mainly on which regression algorithm and kernel function is used. Choosing the wrong parameters can affect the system performances a lot: we want our system to be able to generalize well to a test set that may be

quite different from the training set. Indeed, cross-validation has been proved an efficient method in reducing the common issue of overfitting.

In order to identify the best choice of parameters from a span of alternatives, both the original TakeLab and my implementation use 10-fold cross-validation. Cross validation was performed on an union of all the training files.

A k-fold cross validation starts by randomly partitioning the data into k subsets. k-1 subsets are used to train a model with the current combination of parameters, while the remaining one is kept aside for validating it. The training is repeated so that every subset plays the role of the validation set once; this way, the cross validation ensures that the performance doesn't depend too much on the initial random partition. Furthermore, the cross validation tries to create subsets composed of samples that vary as much as possible in their target scores.

The mean of the results obtained on the k validation sets (usually measured in terms of correlation) is the performance measure for that particular combination of parameters.

This is repeated for all the possible choices of parameters we consider as options. The final choice used in the application is

$$\begin{pmatrix} C \\ p \\ \gamma \end{pmatrix} = \begin{pmatrix} 1 \\ 0.02 \\ 2 \end{pmatrix}$$

2.5 Support Vector Regression

Support Vector Regression is a supervised learning technique which was developed as an extension of support vector machines for regression problems. Since it's supervised, we assume a given dataset $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ with n input samples drawn from an input space X, and target values belonging to the real domain.

The goal of ε-SV regression (the one used in my system) is finding a function f(x) that has at most ε deviation from the real targets y_i for all the training data, while at the same time being as flat as possible. So ε is the maximum error the function is allowed to make while giving answers.

Let's start with a linear function f, described as:

$$f(x) = w \cdot x + b$$

with $w \in X$, $b \in \mathbb{R}$. Flatness means that we define the following minimization problem on w:

$$\min \frac{1}{2}\|w\|^2$$

subject to

$$|y_i - w \cdot x_i - b| \leq \varepsilon$$

(the constraint on the maximum permitted error). We are assuming that a function that solves this problem exists, but this is not always the case, and we may allow some violations of the error constraint. Introducing slack variables, the new problem formulation is:

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

where C describes the tradeoff between the flatness of f and how much we tolerate errors above the threshold ε.

Using Lagrange multipliers it can be proved that:

$$w = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)x_i$$

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)(x_i \cdot x) + b$$

where the α values are positive Lagrange multipliers. Note how w is now completely described with a linear combination of the training samples, and it's not needed to compute it explicitly. Moreover, the complexity of the function is independent of the dimension of the input space X.

At this point we need to generalize the procedure to nonlinear functions. This can be achieved by simply preprocessing the training inputs x_i with a function ϕ that brings them into another feature space F. Then the standard SV regression algorithm is applied. The SVR equations are rewritten as:

$$w = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\Phi(x_i)$$

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)k(x_i, x) + b$$

where k is the kernel function; admissible kernel functions must correspond to the dot product in some feature space F. Note that now w must be computed separately. My system uses the radial basis (RBF) kernel:

$$K(x_i, x_j) = e^{(-\gamma \|x_i - x_j\|^2)}, \quad \gamma > 0$$

2.5.1 SVR parameters

It is now immediate to explain the meaning of the parameters we considered during cross-validation:

- p is the threshold, or maximum allowed deviation, ε;
- C is the tradeoff factor in the minimization problem;
- γ is a parameter of the RBF kernel;

3 Related work

3.1 What is similarity?

(Dekang 1998) tried to give a complete and exhaustive definition of similarity, regardless of the kind of entities involved. They claim that most existing definitions are too tied to practical applications or knowledge bases. A meaningful example is the distance-based similarity: here, the similarity between two terms is measured looking at the path that joins them in a graph where an edge exists between each pair of related terms. This idea is very powerful: it posed the basis for WordNet and other machine-readable resources, but it is completely tied to the presence of such a network. Additionally, the concept of "related" terms is not so obvious: WordNet itself takes into account only a limited number of possible relationships between the synsets, cutting off a whole family of possible paths.

3.2 WordNet

Despite its limitations, distance-based similarity measures computed from WordNet has always played a primary role in the semantic similarity field. As you have seen, TakeLab is no exception in this. Most similarity measures start from the so-called path length similarity: two synsets are related by the number of steps it takes to get from one to the other. If no path exists, they are completely unrelated. (Budanitsky and Hirst 2001) present a evaluation of 5 distances based on WordNet path-length.

3.3 Latent Semantic Analysis

The distributional hypothesis of Natural Language is exploited by many similarity systems. They try to express linguistic entities (words, sentences or documents) as numerical vectors that should represent what the entity "talks about". Usually these vectors are created automatically from corpora, and stacked together to form document matrices. (Turney and Pantel 2010) describes the most common vector space models, underlining how these models are useful in automatically representing the semantics of concepts.

Broadly speaking, three kinds of matrices are used as semantic models:

1. **term-document**: here rows are terms (usually single tokens, sometimes with a POS tag attached) and columns are documents, e.g. Web pages. In its most basic form, each element contains a frequency count of the term in the document. Documents should vary enough in order to be representatives of several different contexts; they are treated as simple bags of words, so no syntactic structure is maintained. However, it is argued, this has not been proved to be a big limitation for the model.
2. **word-context**: quite similar to the one above, except that contexts have a less restricted representation;
3. **pair-pattern**: each row is a pair of tokens, while the columns represent possible relationships between them. Readers familiar with the Semantic Web may find some similarities with how RDF expresses knowledge about domains.

Such matrices tend to be huge, but also very sparse, since every document typically employs a small fraction of the total lexicon. Mathematical techniques such as Single Value Decomposition are usually applied in order to reduce the total dimensionality while maintaining the most relevant relationships.

These matrices give immediate information about words co-occurrences and semantics: we briefly passed over the issue of combining them together in order to get the same kind of meaningful information for a sentence (or for a bigger text). (Mitchell and Lapata 2008) presents different compositions of word vectors into sentence vectors, proving that averaging and summing are not good techniques because they don't take into account the syntactical structures words show inside the sentence, and therefore they end up assigning the same vector to sentences that happen

to share a vocabulary. They propose compositions based on multiplications.

Having a strong and reliable representations of semantics is useful in web-searching and information extraction tasks. (Deerwester 1990) starts with an historical way of indexing Web pages: keywords are assigned to each document to briefly summarize its content. Evaluating a query (a term, for simplicity) is simply a matter of matching it with the keywords and return the document(s) with the highest number of matches. This approach has evident issues: too much terms are synonims, or are closely related, to be able to rely only on exact matches. Latent semantics enters here to give an idea of what terms are also only "implied" by a query, rather than stated explicitly. If we are able to say that the occurrence of some words implies a relation with another word, this word can be added to the keyword list automatically.

With LSA we try to build a "semantic space" where term or documents are placed close if they are related with each other, regardless of which words appear in the document.

3.4 Notes on other systems

Reviewing semantic similarity systems presented for the same contest the machine learning approach clearly prevails. Furthermore the features tend to be drawn from the same ideas and studies, although they often differ in the actual formulae. Below are some of the most interesting ones found in systems other than TakeLab simple:

Syntax-related features

The relation between syntax and semantics has been deeply explored. Many systems try to infer a semantic relation from syntactical dependecies, often using syntax tree or dependency parse tree as machine-readable representations of syntax. Both TakeLab syntax and (Marsi 2013) contain good examples of this approach.

Specifically in the subject of assessing the similarity of questions extracted from large QA databases, (Wang 2009) describes new syntax-related features which, alone or combined with more "semantic" ones, achieve better results than simply relying on bag-of-words values. This improvement remain even if some noise is introduced in the questions in the form of grammatical errors, typical of many Internet users. It is not immediate to compare their results with those of TakeLab, however, due to the difference in their evaluation measures.

Random Indexing

Random Indexing is an alternative to LSA for distributional semantics which again tries to exploit the co-occurrences of words in the same documents, used for example by (Wu 2013).

Cross validation is often proposed as a valid techniques for understanding which features have the most relevant impact on the final performances, or which of them can be safely ignored.

3.5 Dataset gathering and annotations

When the pilot task on semantic similarity was first approved, the organizators found out data for sentence semantic similarity are a scarce resource. Existing sets were either too small, or focused on assesting the similarity of larger portions of text. (Agirre 2012) discusses in details how these issues were overcomed by building a new dataset, and their efforts to control the quality of the annotations.

3.6 The Question Answering approach

As Palmisano proposes itself to do, semantic similarity has been often applied to the field of question answering. Given a sufficiently large QA database, similar questions are searched and compared with the original question in order to find good answers. (Jeon 2005) however are not satisfied with approaches based on building feature vectors, typical of most systems. They find they often overfit on the training data and are hard to scale. Their mostly relies on information extraction techniques. In a nutshell, if two different queries have the same click log and retrieval results, they are likely to be similar. The database is searched extensively in order to build a huge collection of similar question pairs.

Once the collection of pairs is ready, it is easily interrogated in order to answer to new queries. Additionally, a pre-existing translation model can be applied to pairs in order to a infer a similarity model. The underlying concept is treating the pairs as if they were in different languages: if we train a translation model with those pairs as data, it will automatically become a semantic similarity model.

4 Implementation

4.1 External dependencies

My TakeLab implementation uses the following external libraries:

- Stanford Core NLP, for tokenization, Part-of-Speech tagging, and lemmatization;
- WS4j, a library for computing WordNet 3.0 similarity scores. It was picked because the JAR files includes an internal copy of the database, thus avoiding the need for external database files;
- LibSVM is an implementation of Support Vector Machines and Support Vector Regression, found at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/` and described in detail in (Chang and Lin 2011).

the lib/ subdirectory in the project root contains the JAR files for these libraries. The stanford-corenlp-models.jar file is needed to load the machine learning models used by the NLP pipeline.

4.2 Sample files

Training and test files were provided by the Semeval 2012 Task 6, so that all systems could be evaluated on the same samples. Each sample file was actually split in two: one text file for the sentence pairs, called STS.input.¡name¿.txt, and another with the annotated similarity scores, named STS.gs.¡name¿.txt.

In order to simplify command line arguments and I/O operations, the `putOutput.py` script merged the two of them into one text file organized this way:

<sentence1>TAB<sentence2>TAB<score>

This is also the only format recognized by my application. All the sample files you see in the subdirectories are already in this format, but `putOutput.py` is available in the src/scripts/ folder should you need to convert new samples.

4.3 The Google NGram Corpus

The Google NGram corpus is a huge resource, collecting frequency counts for English tokens spanning across several decades. Unfortunately it is not really optimized for a huge sequences of queries, so I applied some processing steps in order to reduce its volume to a manageable size, also in terms of occupied disk space:

- the frequency counts for punctuaction characters were not downloaded, since their information content is not really relevant for this semantic task;
- all the counts for the same token, but different year, were summed together.
- each row contains only the token, its POS tag, and the frequency count; additional information about where the occurrences were found were deleted.
- counts have been "indexed" by the first two initials. The googlebooks/ directory contains a subdirectory for each initial letter; these subdirectories contain in turn a text file for each pair of initial letters. So for instance the a/ folder contains "aa" for all the tokens starting with "aa", "ab", and so on. A special file oneLetter groups together 1-character tokens.
- All the tokens not appearing in an online English dictionary, stored in the dictionary/ folder, have been ignored. The comparison was case insensitive. The dictionary is the Ispell English Word Lists, chosen because it includes inflected forms of verbs and nouns (e.g. plurals). It can be downloaded from `http://wordlist.sourceforge.net/`.

These steps were applied using the `mergeYears.py` and `divideByInitials.py` Python scripts.

Thanks to this, plus some internal caching, lookups inside the corpus don't add much overhead to the execution time.

4.4 Usage

Refer to the README in the project root directory for instructions for compiling and running the program.

5 Results

The Pearson correlation is used to measure the accuracy of the semantic similarity system with respect to the Gold Standard annotations. I compute the correlation for each test file, plus two of the three aggregate measures described in (Agirre 2012): the ALL correlation is simply the correlation of all the test samples concatenated together.

The Mean correlation is an average weighted with the number of samples in each test file.

In Table 1, my results are compared with the original Takelab results, found in their presentation paper, and with the baseline for the Semeval task. The baseline scores are produced using just one feature, a simple word overlap.

	Mine	Takelab	Baseline
MSRpar.txt	0.64	0.73	0.43
MSRvid.txt	0.77	0.88	0.30
SMTeuroparl.txt	0.39	0.47	0.45
SMTnews.txt	0.40	0.40	0.39
OnWN.txt	0.61	0.68	0.58
ALL	0.71	0.81	0.31
Mean	0.60	0.67	0.43

Table 1: Default implementation

It appears that the low results obtained for the SMTeuroparl set are somewhat expected keeping in mind that systems for such complex tasks tend to overfit on the format of the training data. The Takelab authors state that the SMTeuroparl training set differs a lot from the corresponding test set: the latter tends towards significantly shortest sentences, and many short identical sentences are repeated multiple times.

Next, I performed small changes on the system to understand how this affected its performances. For simplicity, only the new aggregated correlations were reported.

	New	Default	Takelab	Baseline
ALL	0.7	0.71	0.81	0.31
Mean	0.58	0.60	0.67	0.43

Table 2: C=1, P=1, G=2

Table 2 shows that a not-so-big variation of P doesn't have an important influence on average.

	New	Default	Takelab	Baseline
ALL	0.6	0.71	0.81	0.31
Mean	0.57	0.60	0.67	0.43

Table 3: C=1, P=1, G=0.02

From Table 3: as expected, if also G is changed, it starts to matter.

	New	Default	Takelab	Baseline
ALL	0.46	0.71	0.81	0.31
Mean	0.35	0.60	0.67	0.43

Table 4: C=1000, P=0.02, G=2

Table 4 shows the biggest variation: C was set to the maximum value possible. Together with the terrible performances, it also resulted in a much slower training process. This is understandable looking at the value of P: the algorithm has to make an effort in order to keep the error below a quite restrictive threshold for almost all the training samples. It is my guess that such a complicated task is eventually managed by overfitting to the slightest details to the training set, hence the poor results on the test set.

	New	Default	Takelab	Baseline
ALL	0.56	0.71	0.81	0.31
Mean	0.42	0.60	0.67	0.43

Table 5: C=1000, P=1, G=2

The idea presented in Table 4 is partially verified by the results of Table 5. Now P was set to the highest possible value too, allowing some more freedom in the training process. Indeed the training took place at the usual speed again, but still there is too much overfitting for good performances.

	New	Default	Takelab	Baseline
ALL	0.60	0.71	0.81	0.31
Mean	0.53	0.60	0.67	0.43

Table 6: No information content features

	New	Default	Takelab	Baseline
ALL	0.47	0.71	0.81	0.31
Mean	0.37	0.60	0.67	0.43

Table 7: No preprocessing steps apart from tokenization, POS tagging and lemmatization

It must be noted that results obtained from removing the preprocessing steps and the features may not correspond entirely to reality. For a completely accurate comparison I should have run the cross validation on the new set of features, and then trained the model with the new parameters. This was avoided simply because the cross validation step is too much computationally expensive.

5.1 Issues

Although I've checked my code with great details and tried to follow the paper to the word, the results show that my implementation has slightly worse performances than the original TakeLab. This may be done to a variety of reasons:

- the Python code they present for extracting features slightly differs from the one described in the paper: namely, some features have different definitions and the preprocessing steps are not applied for every feature. However, I cannot be sure these are not changes added after the paper was written;
- I used different external data: LSA matrix, WordNet library, and Google NGram Corpus.
- personal errors.

6 Applying this system in Palmisano

From now on, I'm going to refer to the original TakeLab simple system. It is my personal opinion that it will not be suitable for being applied to Tweets "as it is", and I'm going to explain why.

First, some of the features and preprocessing functions were clearly designed with the specific training set in mind (e.g. the stock index overlap). When commenting on their results, the authors justify the poor performances on the SMTeuroparl or SMTnews test sets by observing that they differ a lot in structure and average length from the training data. Indeed those scores are pretty close to the baseline, which expresses the worst possible results. So my guess is that the system is not meant to be immediately generalized to such a different task as analyzing Twitter messages and Web questions.

Keeping Twitter and the Palmisano organization in mind, these are other issues that in my opinion will reduce the performances of this particular module:

- misspelled words: very typical of Twitter messages, but also of famous QA websites such as Yahoo Answers. Most features in TakeLab rely on words being spelled the correct way. Hopefully the negative effects of this will be partially reduced by the normalization stage;
- again on the subject of misspelling, I noticed that words starting with a capital letter tend to be immediately tagged as proper nouns by both Stanford Core NLP and other Part-of-Speech tagging tools. However, a lot of tweets I've seen use capital letters unduly, causing the named entity related features to fail.
- I used Stanford Core NLP instead of ArkTweetNLP because I needed the lemmatization step. Stanford NLP uses the Penn treebank tagset which is different from the one in Palmisano. The needed translation surely subtracts some relevant information in the questions.

On top of all, I believe that the system should be enriched with some features designed for dealing with questions before being inserted in the Palmisano pipeline. A possible idea is a feature comparing the motives behind a question (asking for a factual information vs. a personal opinion, asking for a reason, and so on). Furthermore, when more annotated tweets are available, the training and cross validation should be repeated on those.

7 Acknowledgements

I would like to thank Emanuele Bastianelli, Ph.D student at DIAG, for his help in providing LSA data and cross-validation suggestions. I would like also to thank David Jurgens for his support, especially for dealing with the Google Corpus.

References

Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. 2012 *SemEval-2012 Task 6: A pilot on semantic textual similarity.* In Proceedings of the 6th International Workshop on Semantic Evaluation (SemEval 2012), in conjunction with the First Joint Conference on Lexical and Computational Semantics (*SEM 2012). ACL

Alexander Budanitsky and Graeme Hirst. 2001 *Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures.* In Proceedings of ACL Workshop on WordNet and Other Lexical Resources, Pittsburgh, PA.

Chih-Chung Chang and Chih-Jen Lin. 2011 *LIBSVM: A library for support vector machines.* ACM Transactions on Intelligent Systems and Technology, 2:27:1-27:27.

Scott Deerwester and Susan T. Dumais and George W. Furnas and Thomas K. Landauer and Richard Harshman. 1990 *Indexing by latent semantic analysis*,

In Journal of the American Society for Information Science.

Dekang Lin. 1998 *An information-theoretic definition of similarity.* In Proceedings of the 15th international conference on Machine Learning, volume 1, pages 296-304. San Francisco

Jeon, Jiwoon. 2005 *Finding Similar Questions in Large Question and Answer Archives.* Computer Science Department Faculty Publication Series. Paper 138.

Marsi, Erwin; Moen, Hans; Bungum, Lars; Sizov, Gleb Valerjevich; Gamback, Bjorn; Lynum, Andre. 2013 *NTNU-CORE: Combining strong features for semantic similarity.* Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity.

Jeff Mitchell and Mirella Lapata. 2008 *Vector-based models of semantic composition.* Proceedings of ACL08: HLT, pages 236-244.

Saric, F., Glavas G., Karan M., Snajder J., Dalbelo Basic B. 2012 *TakeLab: Systems for Measuring Semantic Text Similarity.* In Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012). pp. 441-448. ACL, Montreal, Canada (7-8 June 2012).

Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003 *Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network.* In Proceedings of HLT-NAACL 2003, pp. 252-259.

Peter D. Turney and Patrick Pantel. 2010 *From frequency to meaning: Vector space models of semantics.* Journal of Artificial Intelligence Research, 37(1):141-188.

Wang, Kai and Ming, Zhaoyan and Chua, Tat-Seng. 2009 *A syntactic tree matching approach to finding similar questions in community-based QA services.* Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval.

Stephen Wu, Dongqing Zhu, Ben Carterette, Hongfang Liu. 2013 *MayoClinicNLP-CORE: Semantic representation for textual similarity.*