Language Understanding System mid-term project

Marco Mancini
University of Trento
marco.mancini@studenti.unitn.it

Abstract

This report documents all the work done for the Language Understanding System mid-term project. It consisted in develop a Spoken Language Understanding module for the Movie Domain. The report starts from the analysis of the data, pass through all the developed version of the module and ends with the evaluation and comparison of all the versions developed.

1 Introduction

The objective of this mid-term project for the course of Language Understanding System (Giuseppe, 2016) is to develop a Spoken Language Understanding Module for Movie Domain. The module in question must take as input a test file composed by sentences, and produce as output a prediction of the concepts for each word of the sentences passed as input.

The following document report all the work done, the first section describe the data on which the project has been developed, with some interesting statistics which helped me during the development. The objective pursued is to increment the performances of the algorithm as much as possible, for this reason from the second section starts the description of all the different implementations I've done, starting from the basic one and arriving to the most advanced one. The last section regard the evaluation and the comparison between the implementations.

2 Data Analysis

The data provided were split into two main sets:

- Basic set
- Feature set

Both the sets contained training and test data, the first one basically contained sentences with words and related concepts, and the second set contained also other features like Lemmas and Part of Speech tags.

The first interesting distribution to observe is the one regarding the words, this analysis can help to understand if the dataset follows the distribution of the words in Spoken Language, also known as Zipf's law (Li, 1992). The given training set was populated with 3338 sentences and 21453 tokens (1728 unique), moreover the test set was composed by 7117 tokens (1039 unique) and 1091 phrases. How is possible to see in Figure 1 the distribution of the words in the training set follows the Zipf's law, in fact the lower rank words (i.e "the", "in", "of",..) appears with an higher frequency (~5/6%) with respect to the higher rank words (i.e. "mcgovern", "restrictions"). Since we're dealing with a movie-specific dataset, in the most frequent words there are also present context-words like "movie", "movies", "produced". The same analysis has been done over the lemmas and the results are more or less the same, but the lemma "movie" has the higher frequency. Analyzing the dataset I've also noticed that some words are incorrect, I'll show how I've tried to manage these incorrectness in the implementations using two different cut-off strategies.

The next analysis regards the distribution of concepts. The concepts are given with the IOB notation (Wikipedia, 2001) and their frequency can be easily seen in Table 1. The first column shows the concepts, the remaining two columns show the frequency of the concept in the training set, specifically the 2nd column contains the frequencies including the Os (no concept), instead the 3rd column shows the distribution without Os. How can be easily seen the *movie.name* concept is very wide distributed with respect to other concepts, instead there are a lot of concepts with small frequencies (~1 %) . Is also easy to see how the

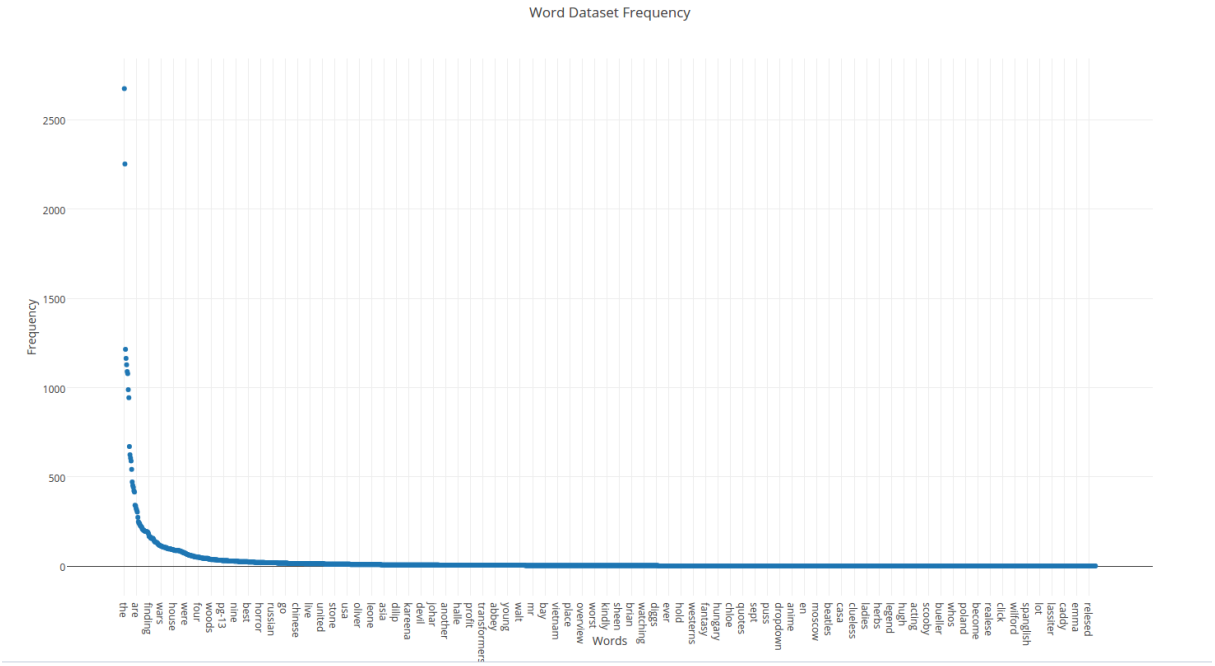


Figure 1: Word frequency on training set

concept "O" is very very spread in the dataset, we'll see later how these informations has been exploited in order to have better performances.

The last ratio proposed in this section is the Out Of Vocabulary ratio (OOV). This ratio indicates how many words in the test set are not present in the training set, then in the lexicon of the module. The OOV ratio of the test set is 23.67% over 1039 unique tokens of the test set, which seems be high and this can cause several errors (empirically each OOV word cause 1.5/2 errors).

3 Basic Project

The first basic version of the project has been developed using only the basic sets of training and test data, then the only features I used were the words. In a very small nutshell the objective is to map words into concept, these concepts can be seen like classes which the module should predict. In order to achieve this objective I used the Bayesian Decision rule, which, given an example (i.e. the word) aims to choose the class (i.e. the concept) maximizing the posterior probability:

$$P(Y = y_i \mid X = x_k) = \frac{P(Y = y_i)P(X = x_k \mid Y = y_i)}{P(X = x_k)}$$

The generative model used to complete this concept-tagging task are the Stochastic Weighted

Concept	Freq (w/ O)	Freq (! O)
O	82.618 %	0 %
movie.name	7.52590 %	43.29833 %
director.name	1.27220 %	7.31933 %
actor.name	1.21853 %	7.01050 %
rating.name	1.07359 %	6.17665 %
producer.name	1.04675 %	6.02223 %
country.name	1.01991 %	5.86781 %
movie.language	0.93402 %	5.37368 %
movie.subject	0.92329 %	5.31192 %
person.name	0.81056 %	4.66337 %
movie.genre	0.50458 %	2.90302 %
movie.release_date	0.46701 %	2.68684 %
character.name	0.26303 %	1.51327 %
movie.location	0.09125 %	0.52501 %
award.ceremony	0.05367 %	0.30883 %
movie.release_region	0.04831 %	0.27794 %
movie.gross_revenue	0.04294 %	0.24706 %
actor.nationality	0.00268 %	0.15441 %
actor.type	0.01610 %	0.09264 %
director.nationality	0.01073 %	0.06176 %
movie.description	0.01073 %	0.06176 %
person.nationality	0.01073 %	0.06176 %
movie.star_rating	0.00536 %	0.03088 %
award.category	0.00536 %	0.03088 %

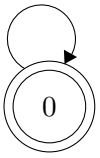
Table 1: Concepts distribution

Finite State Transducers (Mehryar Mohri, 2001), using this model I had to engineer and create some transducers which now I'm going to explain. This first basic project required me only to create two transducers:

- Word to concept transducer
- Language Model transducer

The word-to-concept transducer basically encodes the meaning of the likelihood in the bayesian decision theory, then it must model the probability of a word given a concept. The transducer is composed by only one state, it takes as input a word, outputs a concept and each transition has a specific weight.

$$word : concept/weight$$



All the values needed to create this transducer have been taken from the training set, basically each possible word-concept pair has been extracted from the set and used in this machine with the weight calculated according to the Maximum Likelihood estimation as:

$$P(w_i|c_i) = \frac{C(w_i, c_i)}{C(c_i)}$$

Since this machine has been created with no tool, I've also managed the unknown words (words not already seen in the training set) since we'll see that in order to tag we need to compose different machines and, if a word is not mapped into this machine, then the composition cause an empty set as result and we do not want this happens. This problem has been addressed into 3 different ways:

- Same probabilities: I've basically added one entry unk -concept for each concept and assigned the weight as the fraction 1/#concepts.
- Frequency cut-off: Given as hyperparameter a threshold T, I've cut off all the words appearing less or equal to such a threshold, accumulating a counter of deleted words. Then I've subdivided the accumulated counter between all the concepts and used them to compute the probabilities. This approach can help to cut-off also the errors made in the training set, since an erroneous word has an high probability to be cut off.
- Noise cut-off: The procedure is the same as explained above, but the threshold T is used on the pairs word-concept, and no more on the words only. This kind of cut-off can help to eliminate the noise, since word-concept pairs which appears with small frequencies do not give us a lot of informations.

The language model machine encodes the knowledge about the concepts, and basically represents the prior in the Bayesian Decision rule. This machine can be modelled in different ways, in fact it is possible to vary the smoothing algorithm and the size of the grams to use. The smoothing algorithms are used in order to take into account about the unknown words and the problem of sparseness of probabilities. The grams size is useful to change the meaning of the "prior", in fact playing on this size we can take into account only the current concept (size one) or the previous ones. The transducer modeling the Language Model has been created using both OpenGRM[1] and OpenFST[2].

Now we have both the machines which must be used in order to tag. Given a new sentence the only thing to do is to craft the transducer related to such a sentence (a simply chain-transducer with no weights and output labels equal to input labels) and compose it with the Word-Concept transducer and then, compose the result with the Language Model transducer. Since the resulting machine is composed by all the possible meanings but we're interested in the most probable, we simply take the shortest path (since the weights are expressed as -ln(probability), then we need to minimize the cost in order to maximize the probability). We'll see in the last section the results given by this version of the project.

4 Advanced Project 1

The first attempt to optimize the performances of the module has been done using the additional features which are Lemmas and Part Of Speech tags. Since they are additional features and are present

[1] http://www.openfst.org/twiki/bin/view/GRM/NGramLibrary

[2] http://www.openfst.org/twiki/bin/view/FST/WebHome

also in the test set, I've not modeled other transducers but, instead, I've simply decided to concatenate them with the words and train and test the module using these concatenations. With respect to the basic project and depending on the feature selected, the likelihood modeled change meaning. For example if the feature to use is the PoS tag, the probability to model will become P(word, PoS | concept). It is useful to notice that modeling these likelihoods instead of the previous ones produce spread results. The combinations of features selected for this second version of the project are:

- Lemmas
- Word + PoS
- Lemma + PoS

The remaining combinations have been discarded, since they do not add any relevant information (i.e. Word + PoS + Lemma cause only loss of informations and the PoS alone are not too much to permit a good discrimination of the concepts).

5 Advanced Project 2

Since, how we'll see later in the evaluation, with some group of concepts the classifier is not very able to discriminate, and confuse them one to each other (i.e. director.name, character.name, person.name), I've implemented another version of the project which aims to solve this issue. This version basically add the concept of "cluster of concepts", instead of pass from words directly to concept I've inserted an intermediate step in which the words are grouped into clusters of concepts and then these clusters are re-mapped into the concepts. The transducers needed for this version are 3:

- Word-Cluster: This transducer map words into cluster of concept (i.e input=words, output=cluster). The weights associated with the transitions are calculated as -ln(P(Word | cluster) (i.e. -ln[Counter(Word,Cluster)/Counter(Cluster)]). Obviously the words which are not mapped to some cluster of concepts are mapped into the concepts themselves.
- Cluster-Concept: It re-map clusters into concepts (i.e. input=cluster, output=concept), the weights associated are calculated as -ln(P(cluster | concept)) (i.e. -ln[Counter(cluster,concept) / Counter(cluster)]).
- Language Model: The usual language model already explained in Section 3.

Given a new sentence the composition follows the order on which the transducers are presented and the shortest path is used to retrieve the most-probable tagging. Unfortunately, how we'll see in the results of the evaluation, this version of the project did not work very well in terms of performances, then it has not been inserted as testable part of the project.

6 Advanced Project 3

As explained in Section 2 the amount of Os (no-concept) is very very high in the training set (>82%), then what I've done was to analyze better the data in order to understand if was possible to find some interesting insight to use in order to embed prior in the data, and have better results. For each word preceding each concept I've computed the ratio of appearances of such a words behind such a concepts, over the whole appearances of the word in the training set. This ratio helped me to understand, for each concept, which are the most related preceding words and how much these words interfere with other concepts (i.e. the higher the ratio, the most is the word related to the concept and the less interfere, since it appears the most of the times behind such concept). This reasoning was also extended to 2 preceding words, and I've extrapolated very interesting ratios. For example I've discovered that the words "produced by" were used a lot of times behind *producer.name*, the words "directed by" for *director.name*, the word "starring" preceded plenty of times the concept *actor.name*, and a lot of others insights.

Then I used these insights in order to embed prior in the machine. To perform this embedding what I've done is simply to change the concepts related to the words analyzed, from "O" to the word itself. In this manner I've added "dummy" classes which, together with the Language Model transducer, helped me to exploit all the insights found and have better performances. After this reasoning I've also tried to modify all the words into concepts (instead of use only part of them as concepts as explained until now), and this last trial, together

Fold	Accuracy	Precision	Recall	F1
1	94.44%	82.72 %	82.46%	82.59%
2	94.41%	82.32 %	83.33%	82.83%
3	94.73%	81.38 %	83.90%	82.62%
4	94.98%	82.04 %	83.79%	82.90%
5	95.03%	82.57 %	82.57%	82.57%
6	94.16%	81.99 %	81.23%	81.61%
7	94.66%	83.08 %	84,88%	83.97%
8	93.42%	81.60 %	81.85%	81.72%
9	93.23%	80.48 %	81.96%	81.21%
10	95.46%	88.07 %	88.89%	88.48%

Table 2: 10-fold validation results on Advanced Project 3

with 4-grams and Kneser Ney as smoothing algorithm for the Language Model, gave me the best results. A first premonition about this technique was the risk of over-fitting, since I've added a lot of relations between the concepts which could lead the model to hard-follow the data, having a bad generalization. In order to check this property I've shuffled the training set and performed a K-cross validation, the results are the ones in Table 2

As is possible to see, the results hint that there is no overfitting and the model seems have good performances, but we'll analyze them into the next section.

7 Evaluation and Comparison

The evaluation of the different versions of the module has been done using as metrics accuracy, precision, recall and F1-measure. The most important metrics to look for is the F1-measure (harmonic mean between precision and recall) since the accuracy do not care about the unbalancing between the classes. Before starting to discuss about the performances of the versions of the project, is useful to set a baseline so to have some base results to start from. As baseline I've decided to use the basic version of the project with the minimal settings possible, then *unsmoothing* and unigrams regarding the Language Model and no cut off set. With this settings the performances the module have are:

- Accuracy: 88.94 %
- Precision: 54.97 %
- Recall: 60.31 %
- F1: 57.52 %

The first version analyzed is the basic one. Playing on the different hyperparameters of the project (e.g. order for n-grams, smoothing algorithm, kind of cut-off and threshold) I've collected several results and I've derived the following conclusions:

- The best smoothing algorithms to use are the *witten bell* and *kneser ney*, with respect to the others algorithms (unsmoothing, presmoothing, absolute, katz) they give a gain of 6/7 % on the F1. Probably this gain is due to the fact that are two interpolation algorithms, then they take into account all the n-grams sizes (weighted) until the one specified by the order of the Language Model.
- The best results are reached using bi-grams in the language model. Changing from bi-grams to tri-grams or uni-grams comport in average a drop of 3/4 % of F1-measure.
- The cut-off strategies do not help to improve the performances, both the noise cut-off and frequency cut-off explained in Section 3 produce a drop of 2/3 % of F1-measure. It is interesting to see how increasing the size of the threshold, the F1 (and also the other metrics) decrease. I deduced that, even if there are less frequent words and word-concept pairs, these contain useful informations for the classification and the best way to tract the unknown words is to avoid the usage of cut-off strategies and, instead use the *Same probabilities* strategy already explained.

From all these conclusions I claim that the best results for this first basic project are:

1. **Witten bell + 2-grams + no cut-off** => Accuracy: 92.68% Precision: 78.51% Recall: 74.34% F1: 76.37%
2. **Kneser ney + 2-grams + no cut-off** => Accuracy: 92.68% Precision: 78.41% Recall: 74.24% F1: 76.27%

The second version analyzed is the "Advanced Project 1" explained in Section 4. This version of the project extended the previous one simply using, instead of the words only, also other features like Lemmas and PoS tags. As explained, the additional features were used only with specific pairs (i.e. Lemmas, Words + PoS and Lemmas + PoS) and not with all the possible combinations.

In all the three cases the conclusions previously explained hold, in fact also in this case the *witten bell* and *kneser ney* resulted the bests smoothing algorithms, the bi-grams won again and also the cut-off strategies gave me a worst result. The best result are the following:

1. **Witten bell + 2-grams + no cut-off + (Lemma + PoS)** => Accuracy: 92.19% Precision: 77.56% Recall: 72.23% FB1: 74.80%
2. **Witten bell + 2-grams + no cut-off + (Word + PoS)** => Accuracy: 92.17% Precision: 77.48% Recall: 72.23% FB1: 74.76%
3. **Kneser ney + 2-grams + no cut-off + (Lemma + PoS)** => Accuracy: 92.15% Precision: 77.26% Recall: 71.95% FB1: 74.51%

Unfortunately no one between the features added gave me better results with respect to the previous version of the project, but they were all very so close.

From this version I've noticed that the classifier confused concepts like *producer.name*, *person.name* and *director.name* which are also in the top 10 of the most distributed concepts in the training set. In fact in the succeeding version of the project (Advanced Project 2), as already explained, I've tried to solve this issue grouping the concepts into clusters of concepts, and then remap the clusters into concepts. Unfortunately also this trial gave me no good results, in fact the best performances has been achieved using again *witten bell* and *kneser ney* with bi-grams and no cut-off but, this time, I had a drop of 4/5 % on the F1-measure. Then I tried to cluster other confused concepts, but again with bad performances.

The last version of the project (Advanced Project 3) is the one which gave me the best results with respect to all the other versions. In this case the best results are reached again using *kneser ney* as smoothing algorithm but with 4-grams and no cut-off strategies, these settings gave me a best of accuracy:94.96%, precision: 82.44%, recall:83.04% and F1:82.74%. With respect to the best result reached until now (Basic project + witten bell + bi-grams and no cut-off), this model increased the F1-measure of more than 6% and all the other statistics of at least 2/3 %. It is important also to notice that also all the other settings of hyperparameters give better results with respect to all the previous results. As already explained, for this version I've also run a 10-cross validation to check the possibility of overfitting, but the validation gave me optimal results.

Comparing the different version of the project I can draw the following conclusions:

- Strangely the usage of Lemmas and PoS do not increase the performances as expected but, instead decrease them of 1/2% at least.
- All the cut-off strategies implemented are only resulted in a loss of performances. Probably the data are not too much to permit a winning cut-off strategy.
- Create group of confused concepts did not helped to increase the performances but, probably, merging the idea implemented in the last version with the idea of the clusters could improve the performances a little bit.
- The last version of the project gave me very good results and increased the F1-measure of 25% with respect to the baseline. This is probably due to the high frequency of Os concepts (no-concept) which in this version has been completely translated into concepts, giving more detailed informations and useful statistics to the classifier.

Appendices

The project can be found on GitHub at `https://github.com/marcomanciniunitn/LUS-Proj`

References

Riccardi Giuseppe. 2016. Language understanding system course [unitn]. `http://disi.unitn.it/~riccardi/page7/page13/page13.html`.

W. Li. 1992. Random texts exhibit zipf's-law-like word frequency distribution. *IEEE*.

Michael Riley Mehryar Mohri, Fernando Pereira. 2001. Weighted finite-state transducers in speech recognition.

Wikipedia. 2001. Iob notation. `https://en.wikipedia.org/wiki/Inside_Outside_Beginning`.