First project report on WFST, HMM on LM classification

Anonymous ACL submission

Abstract

This report exposes the result in term of the harmonic mean of the precision and the recall of the classification of NLSPARQL database test using the related train dataset. We use openFST and openGRM in order to implement the different classifiers. The WFSTs are trained on WORD as TOKEN, LEMMA as TOKEN and POS as TOKEN and we test also check if we improve the performance if we set a cut-off frequency in order to reduce overfitting. The LM is trained with different n-Gram order and various smoothing method technique. The best result that is been found use a trained classifier on WORD as TOKEN without a cut-off frequency and a LM that use a 2-Gram and a Witten Bell smoothing method. This give an F-score of 76.31%. It is also done an analysis on the classification time and we realised that reducing the training dataset using only the WORD with a frequency greater than 1, reduced by an average of about 20 second the classification time but it reduced also by a few tenths the F-score performance.

1 Credits

This document is prepared for the course of Language Understanding System held by Professor Giuseppe Riccardi and by the researcher Evgeny Stepanov. I thank the community openFST and for openGRM for providing excellent tools to develop my work. Also, I want to thank the developers that wrote the programs included in the requirements of my project.

2 Introduction

”Spoken language understanding (SLU) is an emerging field in between the areas of speech processing and natural language processing. The term spoken language understanding has largely been coined for targeted understanding of human speech directed at machines.” [Microsoft]

As happen in classification, we assign a categorical label belonging to a Movie Domain to new observation on the bases of the categories present in the training data. Instead of treating each word as an independent classification task we use Hidden Markov Model in order to take inference for the classification of the present word given the previous categories. This gives in general better result. In this work, we used training and testing corpora CONCEPT tagged by a Microsoft's team.

First at all we define a weighted finite-state transducer that integrates into its weight the prior probability of the CONCEPT and the joint probability of (WORD,CONCEPT), (POS,CONCEPT) and (LEMMA,CONCEPT) in order to test which of this three features give the best performance. Then we took a more complex assumption. We test the first-order until the tenth-order Markov assumption. In order to implement this HMM, we use the farcompilestrings to build a finite-state archive in which we put all the sentences composed by CONCEPTs as finite-state machine. Then by the command ngramcount we compute the conditional probability of having t_i given the n previous t. Doing this we built the Language Model λ_C of order n and of type ”witten_bell”, ”absolute”, ”katz”, ”kneser_ney”, ”presmoothed” or ”unsmoothed”. This smoothing methods are used in order to fill all the zero probability gap. Then we decided to consider the unknown word, equal probable on the different CONCEPTs. Under this assumption, we build a new WFST that accept the <unk> words and has one state and forty-one arcs with the same weights. This machine is been used in union with the first WFST that integrate the prior and the class conditioned probability. It is important to do a closure after the union in order to accept <unk> words located also inside a sentence. In the decodification phase we use the concatenation of $\lambda_S \circ \lambda_{W2T} \lambda_{SCLM}$ to build the solution tree and by the Viterbi decoding algorithm we found the path that minimize the cost. Doing this we found the sequence $t_i, ...t_n$ that maximize $\prod_{i=1}^{n} P(w_i|t_i) * P(t_i|t_{i-1})$.

All this test can be replicated by a module written in bash that tests the performance using WORD as TOKEN, LEMMA as TOKEN and POS as TOKEN and for each of them it test ten different n-grams orders and for each order try all the different smoothing method. The last version of the module is the 1.5, that can be downloaded from the following github CONCEPTtaggingWFSTandHMM and allow to define in the classification.conf the search domain of the different classifiers and will produce in the output folder several performance plots: accuracy, mean precision, mean recall, F-score and the classification time.

3 Analysis of the dataset

The words in the training set that we used, have the following histogram.

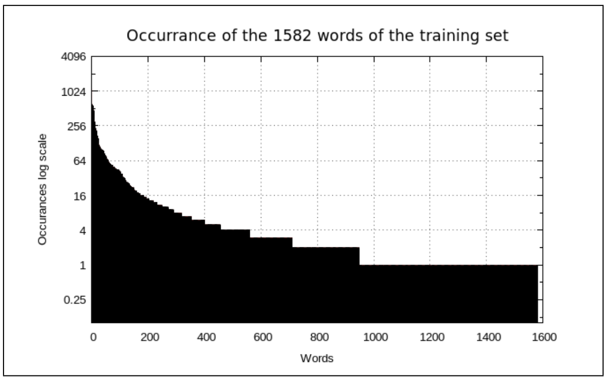

Figure 1: ”Histogram of the words in the dataset of training”

As it can be seen in the figure 1, about half of the words contained in the database present a frequency of one. These rare words contribute making the classier very data dependent and not general. For this reason, we train our WFST also on a reduced train set where the words that present a frequency equal to 1 had been deleted.

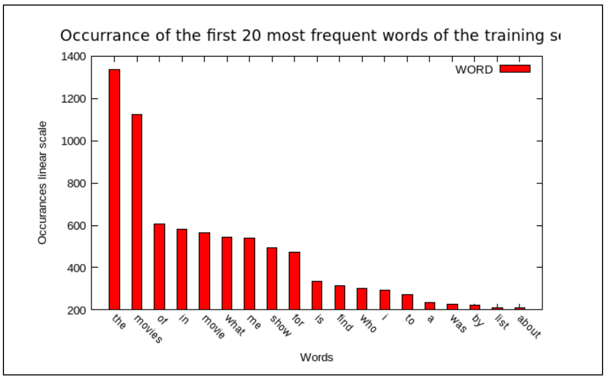

Figure 2: ”First twenty most frequent words of the NLSPARQL dataset.”

Going more deep in the analysis we can take a look at the figure 2 where are shown the twenty most frequent words.

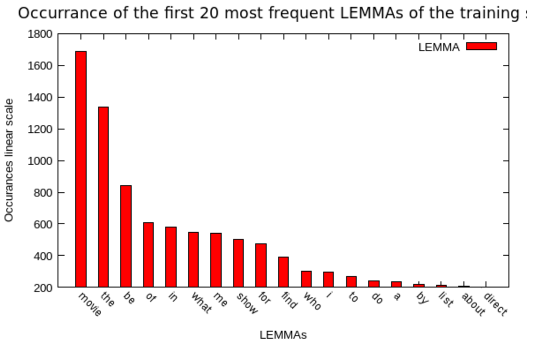

Figure 3: ”First twenty most frequent lemmas of the NLSPARQL dataset features.”

We can notice that differently from a general language database the second most frequent word is movies. And if we check in the figure 3 the first twenty most frequent lemmas, we found that movie is the first. This tells us that this training set is strong related to the Movie Domain.

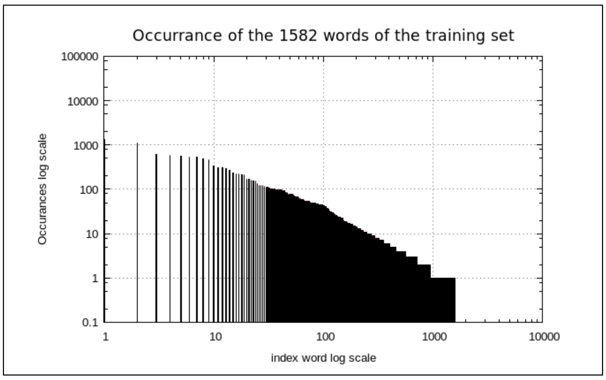

Figure 4: ”Loglog histogram shape that can be approximate by straight negative slope.”

From a shape point of view of the histogram, it is possible to assert that shows the classic characteristic of the distribution of the words in the English language. The hyperbolic shape of the zipfs law that handle for the most languages fit our data as we can see in the following loglog plot 4 where on the x-axis we put the index of the word and on the y-axis the occurrence. We do not show the entire distribution of the lemmas because of shows similar characteristic of the histogram of the words.

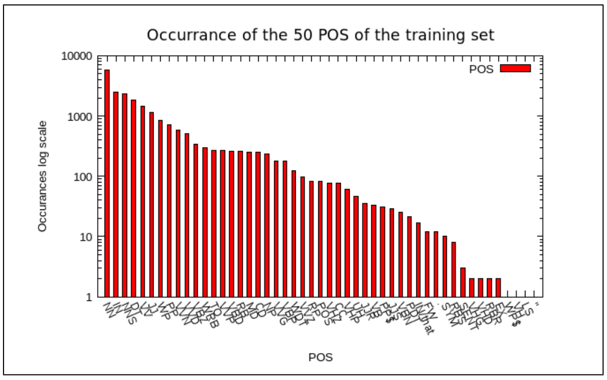

Figure 5: "Part-Of-Speech histogram."

The part of speech distribution shown in the figure 5 present an exponential histogram and none of it has a frequency of 1. So this information maybe can generate a more general model.

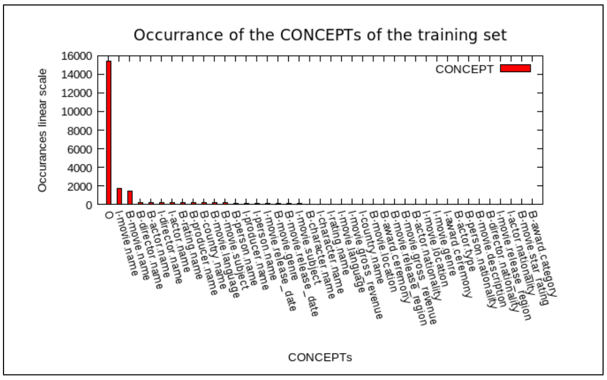

Figure 6: "CONCEPT's histogram."

Now we pass to take a look at the histogram of the class domain. In the next figure 6 it is possible to notice that the first 3 TAGs is much more frequent than the other. They are O (other), I.movie.name (inter-movie name's word) and B.movie.name (begin movie name's word).

4 Comparation of the different LM paramenters

As we said before the computation of the tagger WFST is standard: it uses the prior probability of the CONCEPT and the class conditional pdf. And in order to classify the unknown TOKEN it is assumed that the probability of the different CONCEPT is equal. Instead, the language model can be build using different n-grams order and different smoothing method that fill the zero value of the n-D space. Our simulation took into consideration the following smoothing methods and order:

- "witten_bell" n-grams of order 1, 2 ... ,10
- "absolute" n-grams of order 1, 2 ... ,10
- "katz" n-grams of order 1, 2 ... ,10
- "kneser_ney" n-grams of order 1, 2 ... ,10
- "presmoothed" n-grams of order 1, 2 ... ,10
- "unsmoothed" n-grams of order 1, 2 ... ,10

For each set of the parameters we test the performance considering:

- "WORD as TOKEN" with NO cut-off frequency **and** with cut-off frequency.
- "LEMMA as TOKEN" with NO cut-off frequency **and** with cut-off frequency.
- "POS as TOKEN" with NO cut-off frequency **and** with cut-off frequency.

4.1 WORD

WORD with NO cut-off frequency: Considering the word as the tokens to be classified, the result as we can see in the figure 7 are very good.

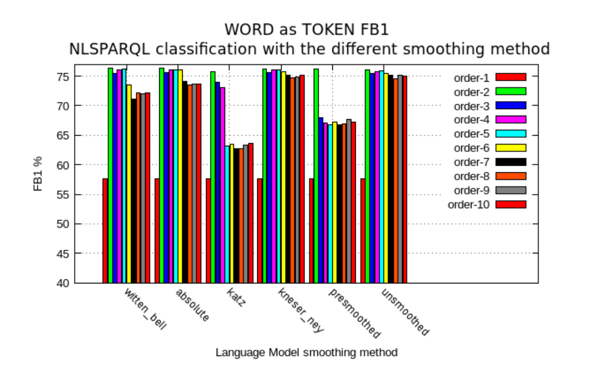

Figure 7: "F-score plot of different order for each smoothing method in the case of classification of WORD as TOKEN with NO cut-off frequency."

The best result we got with orders between 2 and 5 and with the following smoothing methods:

- "witten_bell" 57.62 **76.31** 75.52 **76.07** **76.17** 73.54 71.12 72.12 72.04 72.25 [FB1%]
- "absolute" 57.62 **76.31** 75.62 76.04 76.00 **76.03** 74.09 73.48 73.66 73.63 [FB1%]

- ”kneser_ney” 57.62 76.27 **75.67** 76.04 76.01 75.83 75.15 74.71 74.93 75.14 [FB1%]
- ”unsmoothed” 57.62 76.15 75.46 75.85 75.90 75.50 **75.17** 74.58 75.24 75.05 [FB1%]

We insert also the unsmoothed one to understand how much are the increasing of the performance. We can conclude that in term of F-score performance the smoothing method increases the overall F-score of about 0.15 %. Between this three methods the ones that performs also very good in term of low complexity are:

- ”witten_bell” 126 **127** 146 136 143 146 137 133 152 163 [sec]
- ”absolute” **122 127 130** 135 141 147 147 142 149 160 [sec]
- ”kneser_ney” 136 132 143 145 147 158 157 161 166 169 [sec]
- ”unsmoothed” 151 158 160 168 169 171 165 168 172 174 [sec]

The best two are the absolute and witten_bell smoothing methods that allow to classifying the entire test set in 2 minutes and 7 sec with trigrams language model. From a point of view of the time used for the classification, the LMs that **used a smoothing method** give an increase of performance of about 20 seconds.

WORD with cut-off frequency: If we decide to delete from the training set the words that present a frequency of 1, the trained classifier performs the following metrics. Respect to the previous result of figure 7 the F-score decrease a little bit.

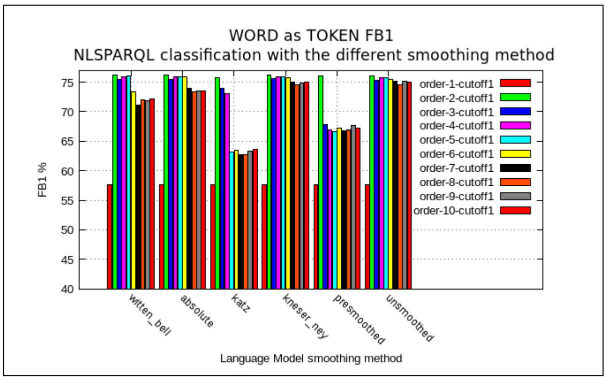

Figure 8: ”F-score plot of different order for each smoothing method in the case of classification of WORD as TOKEN considering a cut-off frequency.”

- ”witten_bell” 57.62 **76.21** 75.43 75.97 76.08 73.45 71.12 72.03 71.95 72.16 [FB1%]
- ”absolute” 57.62 **76.21** 75.52 **75.95** 75.90 75.94 74.00 73.39 73.57 73.54 [FB1%]
- ”kneser_ney” 57.62 76.18 **75.58 75.95** 75.91 75.74 75.06 74.62 74.84 75.05 [FB1%]
- ”unsmoothed” 57.62 76.06 75.37 75.76 75.80 75.50 75.17 74.58 75.24 75.05 [FB1%]

From a point of view of the time used for the classification, deleting the rare words improve the performance of about 30 seconds. It is a good result considering that we lost only 0.1 in the F-score metric.

- ”witten_bell” **94 96** 106 111 117 121 125 129 132 133 [sec]
- ”absolute” 100 103 105 111 116 120 124 130 132 133 [sec]
- ”kneser_ney” 100 101 106 **110 115** 120 125 128 132 135 [sec]
- ”unsmoothed” 103 102 **105** 111 116 120 125 130 132 133 [sec]

4.2 LEMMA

LEMMA with NO cut-off frequency: Considering the lemmas as the tokens for classification we got worse tagging than before. Taking in consideration the best parameter we lost around of 0.2 of F-score.

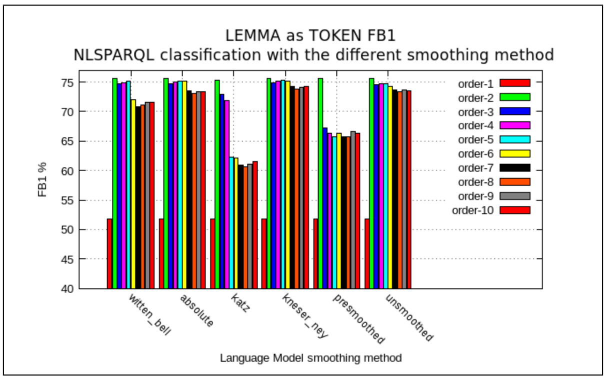

Figure 9: ”F-score plot of different order for each smoothing method in the case of classification of LEMMA as TOKEN with NO cut-off frequency.”

The best parameters in this case are witten_bell and absolute of order two with a F-score of 75.66%. In this case using witten_bell is less time consuming because it took 127 seconds respect to 140 seconds of the other.

- ”witten_bell” 51.75 **75.66** 74.70 74.94 75.16 72.06 70.82 71.19 71.56 71.65 [FB1 %]

- ”absolute” 51.75 **75.66** 74.66 75.03 75.10 **75.24** 73.53 73.00 73.35 73.34 [FB1 %]
- ”kneser_ney” 51.75 75.59 **74.85 75.20 75.37** 75.12 74.26 73.87 74.08 74.33 [FB1 %]
- ”unsmoothed” 51.75 75.57 74.64 74.71 74.79 74.30 73.65 73.39 73.61 73.52 [FB1 %]

LEMMA with cut-off frequency As before the F-score is reduced if we try to train a less specific classifier deleting the lemmas that have a frequency of one.

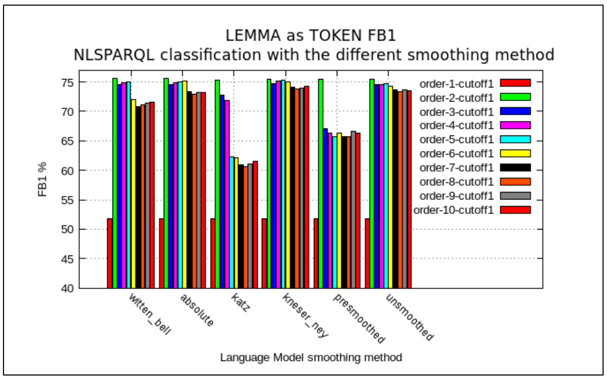

Figure 10: ”F-score plot of different order for each smoothing method in the case of classification of LEMMA as TOKEN considering a cut-off frequency.”

- ”witten_bell” 51.75 **75.57 74.60** 74.85 75.07 71.97 70.82 71.09 71.47 71.56 [FB1 %]
- ”absolute” 51.75 **75.57** 74.57 74.94 75.01 **75.15** 73.44 72.91 73.26 73.25 [FB1 %]
- ”kneser_ney” 51.75 75.49 74.76 **75.10 75.28** 75.02 **74.17** 73.78 73.99 74.24 [FB1 %]
- ”unsmoothed” 51.75 75.47 74.55 74.62 74.70 74.30 73.65 73.39 73.61 73.52 [FB1 %]

Also on a reduced train set the trigram with witten_bell and absolute gave the best result. Respect to the previous case with the entire train set we lost 0.09 %. The improvement in term of classification time is of 40 seconds for the absolute method and 20 seconds for the witten_bell.

4.3 POS

Tagging using POS as TOKEN gave us the worst performance. It is a complete mess. It doesn’t change if we use or not the cut-off frequency because there aren’t any POS that have a frequency of one. Then also in term of complexity or classification time performed very bad. For all the classification test with sixty combination of LM it took 08 hours 25 minutes and 27 seconds respect to 01 hour 57 minutes 35 seconds of WORD as TOKEN.

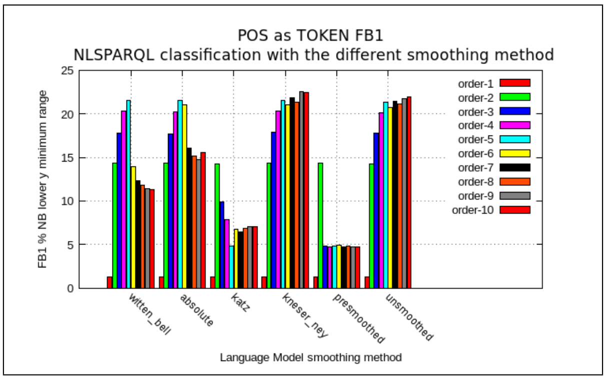

Figure 11: ”F-score plot of different order for each smoothing method in the case of classification of POS as TOKEN with NO cut-off frequency.”

5 Conclusion

We can conclude that the ”katz” and ”presmoothed” works very bad on our training set. Using the ”witten_bell” and the ”absolute” method we can reach the maximum performance in term of F-score. Then if we want to gain a little in term of classification time it is possible to delete the rare words losing some tens in term of F-score. Using the other features as TOKEN we got worse results. With LEMMA the results are not so bad as using the POS, but since the LEMMAS and the POS would have been classified before with their error probability, define a CONCEPT language model on them is definitely counterproductive. Maybe it can be possible to reach better results if we would define a language model that take into consideration not only the words and concept, but also the fact that this word is mapped in a given concept because belong to a given POS or to a given LEMMA.