Language Understanding Systems Mid-Term Project 2017
FST & GRM Tools for SLU

Nicola Garau, 189086

Abstract

The aim of this project is to develop a **SLU** (Spoken Language Understanding) module capable to tag words with tags from a **Movie Domain** and to analyse its performance with different configurations. To achieve these goals, a language model was trained to predict unseen utterances from a testing dataset. For both the training and testing phase, a set of natural language utterances for movie domain was used.

The results show that, in this context, training the SLU on **n-grams** of order greater than five tend to overfit the data, leading to a lower Accuracy and F1-Score.

1 Introduction

A **Concept Tagger** is a model that assigns a token, called tag, to different words belonging to sentences. In order for it to work, it needs to be trained with several training data composed of couples word-tag.

For the purposes of this project, the following were given:

- A data set for movie domain, split into train and test set
- A data set with additional features, split into train and test set
- A Perl script to analyse the tagger performances and give different scores as output (like **Accuracy**, **Precision**, **Recall** and **F1-Score**)

2 Data Analysis

As said before, a data set for the Concept Tagger training and testing was used. The given dataset is the Microsoft **NL-SPARQL** data set.

All the data was already split into train (3338 utterances) and test (1084 utterances). Four files were given:

- **NLSPARQL.train.data**, containing training utterances composed of tab-separated couples of word-concept tag
- **NLSPARQL.train.feats.txt**, containing training utterances composed of tab-separated triples of word-POS tag-lemma
- **NLSPARQL.test.data**, containing, containing test utterances composed of tab-separated couples of word-concept tag
- **NLSPARQL.test.feats.txt**, containing test utterances composed of tab-separated triples of word-POS tag-lemma

Each utterance in the files is separated by the others with an empty line.

Here is an example of utterance from the given dataset:

- From the data file

```
who         O
's          O
in          O
star        B-movie.name
```

```
wars          I-movie.name
episode       I-movie.name
four          I-movie.name
```

- From the feats file

```
who           WP    who
's            VBZ   be
in            IN    in
star          NN    star
wars          NNS   war
episode       NN    episode
four          CD    four
```

2.1 Concept tags

Concept tags are defined using an **IOB Notation**. The notation (the prefix notation B-NP is the one used in the dataset) is used to label multi-word spans in token-per-line format.

The prefixes are:

- **B** : beginning of a span
- **E** : end of a span
- **I** : inside of a span
- **O** : outside of span

2.2 POS (Part Of Speech) tags and lemmas

Both POS tags (explained at https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html) and lemmas in the feats files are additional features that can be used to improve the training of the concept tagger.

2.3 Dataset issues

During the development phase there were two main misunderstandings with respect to the additional features data sets.

The first was that it was not clear how to cope with the fact that the same word could be mapped to different lemmas ('s-be and 's-'s for example) in the transduction from word to lemma. Then it was clear that that transduction was not necessary because it led to a worst training performance.

The second issue was that was not clear how to take advantage of the test set **NLSPARQL.test.feats.txt**. The fact is, probably it can't be exploited during the training phase without "cheating".

3 Development, Structure and Implementation

The whole system was entirely developed using a combination of **Bash** and **Python 2.7**. No third party libraries were used, apart from:

- OpenFST library
- OpenGrm NGram library

The main script, called `conceptTagger.sh`, can be split into 5 parts:

1) Variables definition and clearing of the previous computation's files and folders

2) Utterances and tags sets extraction: two lists of sentences and of the associated concept tags were needed in order to create a far archive and extract FSTs for every single utterance later. The sets can be found inside the *data* folder. The script responsible for the extraction is called *extractSets.py*.

3) Lexicon creation: a lexicon (list of coupes element-index) was created, where each element was chosen amongst (training set):
   a. Words
   b. POS tags
   c. Concept tags
   The lexicon can be found inside the *data* folder. The script responsible for the lexicon creation is called *createLexicon.py*.

4) **Word-To-Concept transducer creation**: a transducer which maps words into Movie Domain concept tags was created using a Python script called *wordToLemma.py* and then compiled using OpenFst's command *fstcompile*.
   During this phase, probably the most important tweak was applied to the "standard" transducer: the cost of the transition from a word to the corresponding concept tag was calculated as follows:

$$Cost = -\log(P)$$

   Where P is the tweaked probability of the concept. It was calculated by taking into account the lemmas and POS-tags occurrences other than the words and concept tags ones:

$$P = \frac{C(word, lemma, POS_{tag}, Concept_{tag})}{C(Councept_{tag})}$$

Where *C* stands for counts. This tweak allowed to improve the overall performance of the **Concept Tagger**, reducing also the overfitting effect that appeared before, especially for n-grams with order greater than 3. In fact, with this probability, the best F1 score was reached with a 5-gram, absolute discounting smoothing method.

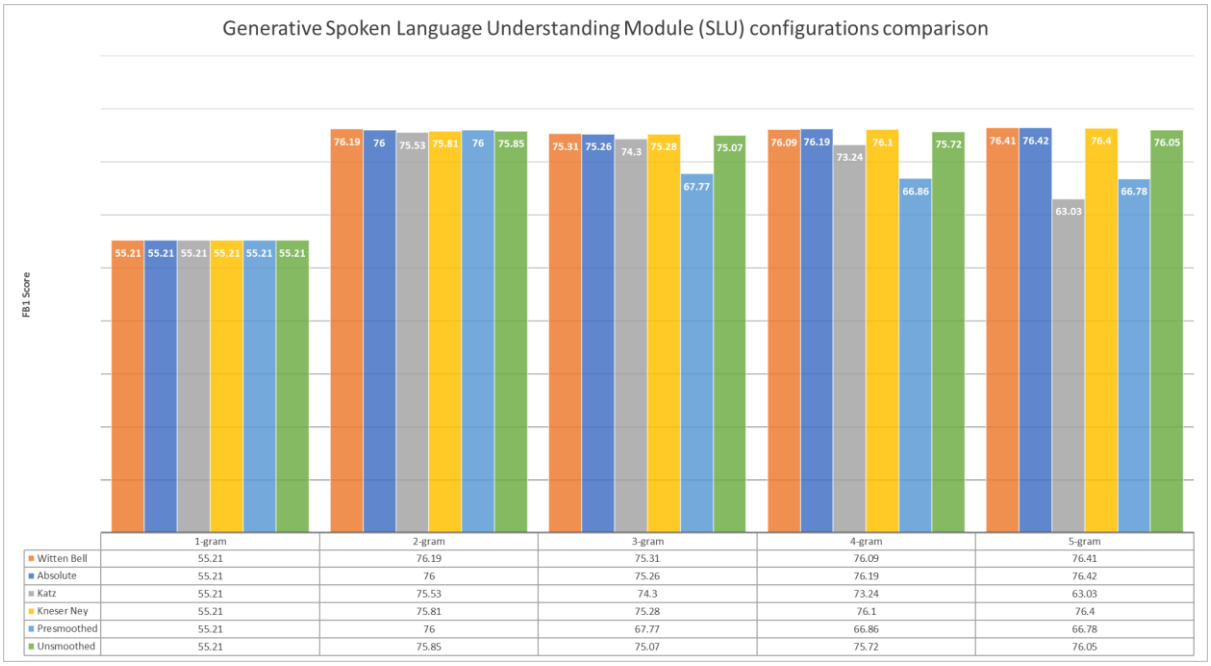
5) Methods and n-gram orders iterations, tagging and evaluation: for each smoothing method offered by OpenFst (*witten_bell*, *absolute*, *katz*, *kneser_ney*, *presmoothed* and *unsmoothed*) and for each n-gram order from 1 to 10, the following pipeline was applied for each utterance in the dataset, using the appropriate generated **language model**:
    a. Composition of the current utterance FST with the Word-To-Concept FST
    b. Composition of the FST from the previous point with the language model (for the m-th method and the i-th n-gram order)
    c. Epsilon removal
    d. Shortest path calculation
    e. Topological sort
    f. Creation of the input file for the Perl script and evaluation of all the utterances for the m-th method and i-th n-gram order

Each result for all the combinations methods-n-gram order were saved in a file. All the results can be found in the folder *results/outputData*. The points 2,3 and 4 are defined by calls of Python scripts from the main Bash script. To call the main script is necessary to execute the command "bash conceptTagger.sh" from inside the *scripts* folder.

4 Failed attempts

Different method were tested before reaching the best result. Among them, two are worth mentioning:

1) Creation of intermediate transducers, like **Word-To-Lemma** and **Lemma-To-Concept** or **Word-To-POS** and **POS-To-Concept**. All those transducers only led to a loss of information and therefore of performance, especially when trying to map a Part-Of-Speech information into a Concept tag (intuitively, because are two unlinked kind of information). The creation of a Word-To-Lemma seemed like a good way to generalise and therefore improve the overall performance but instead led to a worst F1 score for each combination.
2) Taking into account prior probabilities and assigning non-uniform probabilities to the



	1-gram	2-gram	3-gram	4-gram	5-gram
Witten Bell	55.21	76.19	75.31	76.09	76.41
Absolute	55.21	76	75.26	76.19	76.42
Katz	55.21	75.53	74.3	73.24	63.03
Kneser Ney	55.21	75.81	75.28	76.1	76.4
Presmoothed	55.21	76	67.77	66.86	66.78
Unsmoothed	55.21	75.85	75.07	75.72	76.05

transitions from **<unk>** words to concept tags. In the end, assigning cost zero to all the unknown transitions produced better results overall.

5 Evaluation

As mentioned before, the Perl script *conlleval.pl* gives as output a text file containing different evaluation measures, some of which are:

$$Accuracy = 100 * \frac{TP + TN}{FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_\beta = (1 + \beta^2) \frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$$

In particular, in the script, with $\beta = 1$,

$$F_1 = 2 \frac{Precision * Recall}{Precision + Recall}$$

In the diagram above is it possible to see all the best obtained F1 scores of the possible combinations method - n-gram order for the first 5 orders of n-grams.

6 Comparison

A lot of combinations and methods were tested before reaching the current results. Generally speaking, it can be said that the best results can be achieved simply by keeping the system clean, without redirecting data from one transducer to another.

By having just one transducer (Word-To-Concept) it was possible to minimize the information loss and include all the additional features in the cost calculation, increasing mostly the overall accuracy. The best overall F1 score was oddly achieved by the Absolute Discounting method with an order of n-gram of 5. It scored 76.42. Another method worth mentioning is the Witten Bell one, which reached approximately the same score as the Absolute discounting one (76.41) with 5-grams and reached a good 76.19 with 2-grams.

The method described in this document was used to reach the best possible F1 score, but it penalizes the performance of bigrams and trigrams. If a good bigrams and trigrams performance is needed, it is sufficient to exclude the lemmas and the POS tags counting during the calculation of the probabilities at section 3.

6.1 Feature Sets

As said before, the feature sets were used to maximize the F1 score, even if including them in the probability calculation reduced a little the performance when using bigrams and trigrams. On the other hand, by including them, the performance of orders of n-grams greater than 5 increased significantly, while before they tended to drop after the third order.

6.2 Training Parameters

By increasing the order of n-grams, we tend to overfit the training data, thus obtaining worst performances in many cases, even if this phenomenon is reduced by applying the "tweaked" probability calculation described before.

As for the other training parameter (the smoothing methods), the methods which behaved better than the others are the **Absolute Discounting** and **Witten Bell** ones. The **Kats** and **Presmoothed** ones were the one to behave worst with higher n-gram orders.

6.3 Lexicon frequency cut-off

Surprisingly enough, in each configuration, applying a lexicon frequency cut-off (both lower bound and upper bound) only and always led to a worst training of the concept tagger.

Also a combination of frequency cut-off from the lower and upper bound didn't improve the system's performance at all.

7 Conclusions

The method provided by this document describes a method for Movie Domain concept tagging given a training data set, which allows to obtain a high F1 score even for higher order n-grams by taking into account additional feature sets like POS tags and lemmas.

It can be also easily modified to increase bi-grams and trigrams performances by discarding the additional features during the calculation of the cost in the *wordToLemma.py* script.

8 References

- Microsoft NL-SPARQL data set
  - https://www.microsoft.com/en-us/research/project/nl-sparql-a-dialog-system-challenge-set-for-converting-natural-language-to-structured-queries
- Penn Treebank part-of-speech tags:
  - https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
- OpenFst and OpenGrm libraries
  - www.openfst.org