# Assignment 2

Ruosen Li: li.ruos@husky.neu.edu

## 1 Parsing

Codes for this problem is under folder "Problem1". Please read "Problem1/README.md" before running it. Code for this problem can only run in Linux environment. I would suggest Ubuntu 16.04. Also, original data and Stanford parsing tools are not packed in final submission due to the size limitation, but I keep folder structure and put corresponding README files in those folders. Please browse all folders before running codes.

### 1.1 Verb Percentage

The average number of verbs per sentences is around 3.597.

The part-of-speech tags that you are using to identify the verbs are all start with "V", which contains "VB", "VBD", "VBG", "VBN", "VBP", and "VBZ".

All implementation details and results can also be seen in Jupyter Notebook under "Problem1".

### 1.2 Number Of Sentence Parsed

By searching the "ROOT" in parsing files, the total number of sentence parsed is 14259.

All implementation details and results can also be seen in Jupyter Notebook under "Problem1".

### 1.3 Number of preposition in each file

The number of preposition in each file is too many to show here but I output it in Juypter notebook. The most three common three preposition overall are "of", "in", and "to".

All implementation details and results can also be seen in Jupyter Notebook under "Problem1".

### 1.4 Common errors

#### 1.4.1 Constituent Parsing

By viewing parsed files, I find two common errors in the constituent parsing. One is that the method cannot handle punctuation properly in sentences. For example:

Sentence:
The need was demonstrated recently by the statement from the British Government that

it is 'minded to grant' an export credit licence to a company involved in the construction of the Ilisu Dam in Turkey.

```
Parsed sentence (part):
(SBAR (IN that)
    (S
      (NP (PRP it))
      (VP (VBZ is) (“ ‘)
        (ADJP (JJ minded)
          (S
            (VP (TO to)
              (VP (VB grant) (” ’)
                (NP (DT an) (NN export) (NN credit) (NN licence))
                (PP (TO to)
```

Obviously, in this sentence, "minded to grant" should be parsed as a whole part. However, the parsing method treat "to grant" and following words as a clause, which is not correct. On possible explanation is that the parsing just ignore and remove punctuation before parsing. To solve the problem, I would suggest to add punctuation in syntax ad assign proper probabilities to those rules.

Another error is that the method cannot handle long parenthesis properly. For example:

Sentence:
I would like to ask the President-in-Office of the Council, after his few hours working in this Chamber with these Members of the European Parliament, whether he has come to any assessment of what they are worth in terms of pay?

```
Parsed sentence (part):
(NP
    (NP (DT the) (NN President-in-Office))
    (PP (IN of)
      (NP (DT the) (NNP Council))))
(, ,)
(PP (IN after)
    (NP
      (NP (PRP$ his) (JJ few) (NNS hours))
      (VP (VBG working)
        (PP (IN in)
          (NP (DT this) (NNP Chamber)))
        (PP (IN with)
          (NP
            (NP (DT these) (NNS Members))
            (PP (IN of)
              (NP (DT the) (NNP European) (NNP Parliament)))))
        (, ,)
        (SBAR (IN whether)
          (S
            (NP (PRP he))
            (VP (VBZ has)
```

Obviously, the second comma should stay at the same level as the first one. However, the method fails to recognize it and handles it improperly. In my opinion, in Treebank, the probability of such long parenthesis is far small than concatenate of two short sentence. One solution is to extract probability from a larger labeled corpus which may increase the probability of long parenthesis.

### 1.4.2 Dependency Parsing

After scanning dependency parsing file, two common errors are found. One is that the root of dependency parsing is mis-recognized. For example:

Sentence:
Secondly, as there is no mention of Structural Funds in the current draft could you ensure that this would be put back in?

Parsed sentence:
root(ROOT-0, Secondly-1)
mark(is-5, as-3)
expl(is-5, there-4)
dep(Secondly-1, is-5)
neg(mention-7, no-6)
nsubj(is-5, mention-7)
case(Funds-10, of-8)
amod(Funds-10, Structural-9)
nmod:of(mention-7, Funds-10)
case(current-13, in-11)
det(current-13, the-12)
nmod:in(Funds-10, current-13)
dep(ensure-17, could-15)
nsubj(ensure-17, you-16)
parataxis(Secondly-1, ensure-17)
mark(put-22, that-18)
nsubjpass(put-22, this-19)
aux(put-22, would-20)
auxpass(put-22, be-21)
ccomp(ensure-17, put-22)
compound:prt(put-22, back-23)
nmod(put-22, in-24)

It can be seen that "Secondly" should not be selected as root rather than choosing "ensure". The implementing algorithm might not identify "as" at the beginning of the sentence as relative pronouns and applied a wrong syntax on it. A solution would be adding additional information when parsing, for example, the weight of current word in a sentence.

Moreover another error is wrongly parsing noun to adjective. For example:

Sentence:
I wonder if you would answer a question concerning investment by EU companies in the

EU applicant countries.

Parsed sentence:
nsubj(wonder-2, I-1)
root(ROOT-0, wonder-2)
mark(answer-6, if-3)
nsubj(answer-6, you-4)
aux(answer-6, would-5)
advcl(wonder-2, answer-6)
det(question-8, a-7)
dobj(answer-6, question-8)
case(investment-10, concerning-9)
nmod:concerning(question-8, investment-10)
case(companies-13, by-11)
amod(companies-13, EU-12)
nmod:by(investment-10, companies-13)
case(countries-18, in-14)
det(countries-18, the-15)
compound(countries-18, EU-16)
compound(countries-18, applicant-17)
nmod:in(companies-13, countries-18)

As we can see, dependency relation of "EU" at position 12 and "EU" at position 16 are different. "EU" is abbreviate of European Union which is a noun. Thus, relation of both "EU - companies" and "EU - countries" should be parsed as "compound" rather than "amod". A possible explanation is that it is common to put an adjective and a noun together rather than concatenate two noun together. The algorithm prefer to choose the one with the highest probability, which decide to parse the relation between "EU" and "companies" as "amod". One possible solution would be adding constrains or specific rules for special words. In this way, we can always parse the dependency between "EU" and "noun" as "ccomp".

## 2 CKY parser

The answer can be seen in "Problem2.xlsx".

## 3 Sentiment Analysis

Code for this problem is under folder "Problem3". Please read "Problem3/README.md" before running it. In the program, I use scikit-learn tool for implementation. Google word2vec bin file is not packed in final submission due to the size limitation, but I keep folder structure and put corresponding README files in those folders. Please browse all folders before running codes.

### 3.1 Multi-layer Perceptron (MLP) with Cross-Validation

In this sub-problem, I use grid search on three parameters: neurons in first hidden layer, max iteration, and initial learning rate.

After vectorization, input layer is about 14000 neurons. Since the second hidden layer is 10, I deem that the best number of neurons of the first hidden layer is around

$$No.\ of\ the\ second\ layer \times \sqrt{\frac{No.\ of\ input\ layer}{No.\ of\ the\ second\ layer}}$$

The result is around 380. Therefore, the range of the first hidden layer is [370, 390] with a step of 10. The range of max iteration is [200, 600] with a step of 200. The range of initial learning rate is [0.001, 0.1] with a step of times 10.

After grid search, the best parameters are: first hidden layer - 370, max iteration - 600, and initial learning rate - 0.001. Best accuracy during validation is 0.875

All implementation details and results can also be seen in Jupyter Notebook under "Problem3".

### 3.2 Train on whole training set

After training on the whole training set. The model 100% fit the training data and the accuracy on it is 1.0.

All results can also be seen in Jupyter Notebook under "Problem3". All outputs are saved in "outputs.txt".

### 3.3 MLP with word embedding

In this sub-problem, I use the pre-trained Word2vec parameters and the average word embeddings to vetorize each file. Moreover, I use the same method of grid search in problem 3.1 to determine the range of parameters.

The range of first hidden layer is [45, 60] with a step of 3. The range of max iteration is [400, 600] with a step of 100. The range of initial learning rate is [0.001, 0.1] with a step of times 10.

After grid search, the best parameters are: first hidden layer - 56, max iteration - 200, and initial learning rate - 0.001. Best accuracy during the validation is 0.8509. Final accuracy on the whole training set is 0.8311.

All implementation details and results can also be seen in Jupyter Notebook under "Problem3".

### 3.4 MLP with word embedding and POS-tag distribution

Besides word embedding, word tags is also helpful on training, although the range of POS-tag feature (range [0, 1]) is a little bit different with word embedding (range [-1, 1]). I count on all Penn POS-tags and concatenate them with word embedding as total features.

Moreover, when considering how to concatenate POS-tag features on embeddings, there are three methods come to my mind. The first one is simply distribution; the second one

is standardization; the third one is normalization. After trying all of them, the first one performs the best so that I only use the POS-tag distribution to train the model. All other two functions can be seen in code notes.

Final accuracy on the whole training set is 0.8505747126436781.

All implementation details and results can also be seen in Jupyter Notebook under "Problem3".

### 3.5 Test on test set

By comparing all model scores, the model in problem 3.2 shows the best performance, which get a score of 1.0. Although I do not deem that the higher the better, I choose the model in 3.2 to predict the final result. The result saves to "Data/pred/pos.txt" and "Data/pred/neg.txt". Both files are packed in "save/labels.zip".

Moreover, I also predict test set on all models and calculate the percentage of "1" in all results. The result shows me that the percentage of predicted "1" of all models are all around 50% but model 3.4 predicts more "1" than others.

All implementation details and results can also be seen in Jupyter Notebook under "Problem3".

## 4 Summary Evaluation

Code for this problem is under folder "Problem4". Please read "README.md" before running it. Google word2vec bin file and Stanford parsing tools are not packed in final submission due to the size limitation, but I keep folder structure and put corresponding README files in those folders. Please browse all folders before running codes.

### 4.1 Building Non-Redundancy Scorers

#### 4.1.1

In this sub-problem, MSE is 0.1780 and the Pearson correlation is 0.7447.

All implementation details and results can also be seen in Jupyter Notebook under "Problem4".

#### 4.1.2

In this sub problem, I have tried two features which may provide improvement to the model: number of words in each sentence and ratio of stop words.

The reason I choose number of words in each sentence is that I believe the most accurate summary must have proper number of words. Too much words may increase the redundant of summary and too less words will not convey enough information and may cause repeating on the same issue. The result shows me that it works perfectly and reduce the mean square error from 0.1780 to 0.1756, also increase the Pearson correlation for about

0.007 to 0.7513. Therefore, by adding the new feature, the model get improved.

For the second feature, I assume the less stop words the less redundant, since non-redundant sentences would not have high proportion of stop words. The test result also proves my thought. The model's performance increases. The MSE is 0.1762 and the Pearson correlation is 0.7464.

All implementation details and results can also be seen in Jupyter Notebook under "Problem4".

## 4.2 Building Fluency Scorers

### 4.2.1

In this sub-problem, MSE is 0.2383 and the Pearson correlation is 0.2913.

All implementation details and results can also be seen in Jupyter Notebook under "Problem4".

### 4.2.2

In Jupyter notebook, we can see that the original model's performance is really poor. To improve the fluency, I deem that the average depth of PENN tree of each sentence in every summary is relevant to the fluency since people can easy understand simple structure sentences, which will improve the fluency of reading and understanding. The other feature is phrase (including preposition, noun, and verb) proportion. Proper proportion of phrase in a sentence may improve the fluency of sentences.

The experience shows me that all features bring positive result and improve the model significantly. The mean square error decreases for about 0.037 in average and the Pearson correlation score increases about two times. After adding the first feature, the MSE goes down to 0.2014 and the Pearson correlation grows up to 0.7026. After adding the second feature, the MSE decreases to 0.2008 and the Pearson correlation increases to 0.7042.

All implementation details and results can also be seen in Jupyter Notebook under "Problem4".