# Project_part2

Author: Yinghong Zhong(z5233608)

Date: 2019/11/17 (T3)

In part 2, I need to train a learning to rank model to disambiguate the mention. I use techniques like TF-IDF and F1score(precision, recall rate) to generate features.

## Step 1 Preprocess and prepare data

Before generating features, I construct 3 index class. Details are as below:

| Name | Functionality | Constructed Details |
|---|---|---|
| InvertedIndex() | 1. Construct inverted index for men_doc<br>2. Used to calculate tf-idf of tokens in men_docs | 1. Attributes: tf_tokens, idf_tokens<br>2. It is similar to the project part 1, but in this case, I only calculate the tf and idf of tokens. |
| wiki_InvertedIndex() | 1. Construct inverted index for parsed_entity_pages<br>2. Used to calculate the tf-idf of wiki page tokens of candidate entities | 1. Attributes: tf_tokens, idf_tokens<br>2. It is like the project part 1, and I only calculate the tf and idf of tokens.<br>3. I use token_lemma as the key of this index because I think token_lemma would be more common and easy to compare with tokens occurring in other files. |
| LocalIndex() | 1. Construct a dictionary of begin index and end index of a paragraph which a mention occurs in this paragraph of a men_doc<br>2. It is useful to find the relationship between the candidate entities and mentions in a specific situation, so I call it local_index. | 1. Attribute: sectionOffset<br>2. For each mention in a document including training set and test set, I use “offset” and “length” field to retrieve forward and backward in the document until I find a ‘\n’ which means that this is a small graph.<br>3. In my sectionOffset dictionary, the format is:<br>{doc_title: {(offset, length): (b_idx, e_idx)}}<br>b_idx is begin index of the men_doc<br>e_idx is end index of the men_doc |

After constructing indices, I preprocess the input data, setting each mention and one candidate as a pair. Each pair has the following format:

[mention_id, doc_title, mention, candidate, offset, length]

As for the training set, if candidate which equals to the label is the ground truth, I would set the label of this pair is 1, otherwise is 0.

## Step 2 Create features

The next step is creating features. After many attempts, finally I generated 7 features and use 6 of them to train my model.

Here is the description of my features.

**f_mention:** tf-idf score of mention tokens in men_docs

f_mention would not be used to train my model directly but used to compare with candidate entities in f3.

**f0:** tf-idf score of mention tokens in candidate entity's description pages

I think f0 indicates the relationship between a mention and every candidate entity. I use spacy to parse the token to find non-stop and non-punctuation tokens and then use wiki_InvertedIndex() to caluculate the tf-idf score.

**f1:** F1score between candidate_entity and mention

I find that sometimes if the candidate entity is same as the mention or there are more same tokens occur in both candidate entity and mention, this candidate entity has a higher portion to become the ground truth label. Hence, I calculate the F1score by using precision and recall rate as the f1 feature.

**f2:** tf-idf score of candidate_entity tokens in men_doc

I use InvertedIndex() to calculate tf-idf scores for each token of a candidate entity.

**f3:** difference between f_mention and f2

Because most of the mentions and candidate entities are short, I think that if a mention and its candidate entity have a smaller gap of tf-idf score in men_doc, it means that they may have a higher similarity.

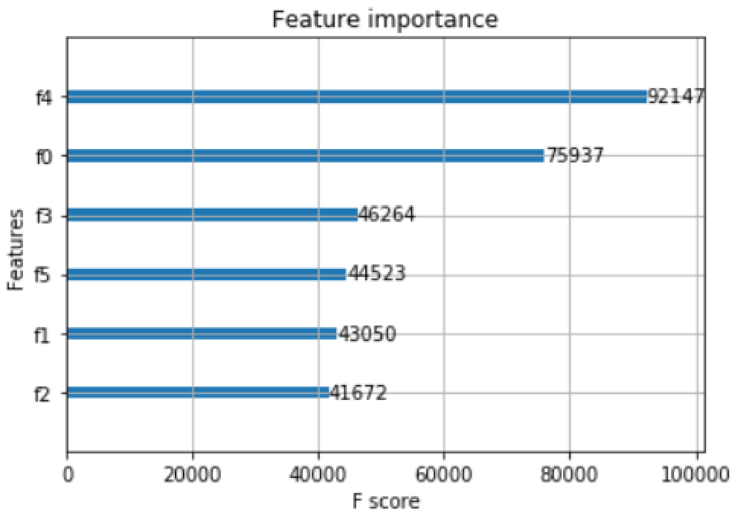
**f4:** tf-idf score of candidate_entity description in men_doc

I use InvertedIndex() to calculate tf-idf of all token_lemma occurring in men_doc.

**f5:** F1score of candidate entity tokens in local section

I use LocalIndex() to find the corresponding local paragraph in a men_doc for each mention and then calculate the F1score between candidate_entity and this local paragraph. I think maybe it would have a good performance because I find that most of the local paragraphs are not too long and if F1score is high, it means that this candidate entity matches this paragraph better.

The importance of my features is shown below. It seems that f4 and f0 has the best performance and the rest of features are also not bad.




## Step 3 Model training

In this case, I use Xgboost to train my model and choose the hyperparameters as below: param = {'max_depth': 8, 'eta': 0.05, 'silent': 1, 'objective': 'rank:pairwise', 'min_child_weight': 0.02, 'lambda':100, 'subsample': 0.8}

In my model, the accuracy of dev_1 is 91%, but for dev_2 is only 74%. I think my model is overfitting, and what I need to do is to find more features and not just stuck in the mud of tf-idf score. Actually I have tried to use BM25, but the performance is not good and the reason I guess is that most of the documents have similar length in our data set and the ratio of length of documents and average document length is about 1 which is not useful.