Competencies_text prediction using Job description text and its Title

Author – Anshu Aditya

Data Preprocessing

Given Train/Test files are full of irrelevant information (essentially noise),So we need to remove the irrelevant information from the title and job desc columns.

1. As there are few rows in training data which has missing **competencies_text**. So I removed those rows form training set and saved file as *Train_data_1.csv.*
2. I removed irrelevant words like stopwords,punctuations from training set.For title and job_desc I intentionaly considered only Nouns and Adjectives as POS.
3. For missing value imputataion I imputed job_desc missing value with job_desc of the row which has same title,If that job_desc is also none then impute with title itself.After this my program will save Training data as *pre_process_Train_data_1.csv*

Model concept

As this problem doesn't fall in any of the pre-specified category,So After a long research I decide to use Concept of Probability.

The idea is to develop rules that assign tags to certain words in the titles and job desc.Eg..

Hadoop -> distributed system

eclipse -> java

The question that needs to be answered is given that word A appears in a job_desc, what is the probability that tag B will also appear in that job_desc?

To get this probability, we need to count the number of job_desc in which word A appears and the also the number of job_desc in which both word A and tag B appear. The desired probability can then be calculated as

P(B|A) = |Co(A,B)| / |A|

where Co(A,B) is the number of titles/job_desc where word A and tag B co-occur and $|A|$ is the number of titles/job_desc where word A occurs. So, if $P(B|A)$ is above a certain threshold, we can then generate the association rule A-->B.
P(B|A) will also depends on the sample size.Therefore we will consider one more factor |Co(A,B)| as support.Hence support thresholding will also be done.
Note - We will aplly this concept individually in title and job_desc.

Algorithm

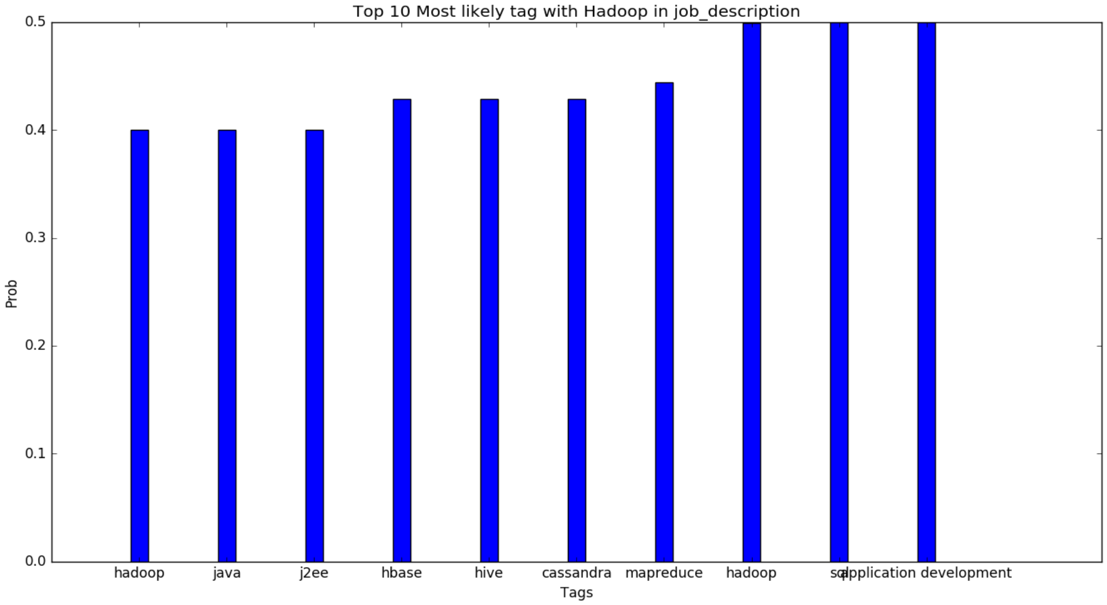
Step-1 : Find all possible combination of words and tags.I have also considered bigrams for both titles and job_desc.

Step-2 : Count All Combinations.

Step-3 : Count overall occurence of all words.

Step-4 : Calculate desired probabilities.

Example –
Bar chart showing Top 10 most likey tag which occurs with Hadoop in Job_Desc.



Parameter Tunning

For finding the optimum threshold I have also implemented a bruteforce approach,which takes various combinations of threshold and find best among themself using F1-Score.

Data Storage

1. *MongodB* -

I have used MongodB to store documents which is in the format {word,tag,score} eg.

```
{u'word': u'teradata/hadoop', u'_id': ObjectId('5856f49beb0a192892534e89'),
u'tag': u'sql', u'score': 0.5}.
where score are the associated prob of word with tag.
```

2. *tmp/pickle*-

I have also saved unigram and bigrams as .*tmp* file and for titles I have saved **word_count** and **word_tag_count** as .*pickle* file.

Project Structure

1. Edge_challenge – Parent directory consists all codes.
2. Data – Inside Edge_challenge data directory contains given data/pre-processed-data and some intermediate data.

Running procedure

1. Run *pre-process.py* for all kind of pre-processing.
2. Run *model.py* for prediction.
3. Final prediction file will be saved as ***predicted_final.csv***.

Note

I have saved more that 5 intermediate files so that computation can be faster.Instead of loading all files directly in main memory an attempt has made to process data row wise.

Results

F1-Score = 0.7819
At threshold 0.7 and support of 0.9

Future-Work

Classical Method -

Create a very sparse binary valued matrix from the training set where each feature is a word that could appear in a title/job_desc (assign it a value of 1 if it does, 0 if is doesn't). Then use well-known classification methods to model the relationship between the input matrix and the output tags.