CSCI - B 659: Assignment #1

Due on Sunday, February 26, 2017

Prof. Cavar

poosingh/rraavi/vpatani

Contents

Text Corpus Selection **3**

Approach 1 - Bag of Words **4**
Model Selection . . . 4
Optimise Feature Extraction . . . 4
Making the Training/Testing Model . . . 4
Results . . . 4

Approach 2 - Collacational Features **5**
Model Selection . . . 5
Optimise Feature Extraction . . . 5
Making the Training/Testing Model . . . 5
Results . . . 5

Approach 3 - Naive Bayes **6**
Model Selection . . . 6
Optimisation Feature Extraction and Modeling Data . . . 6

Approach 4 - Naive Bayes - TF-IDF **7**
Optimisations and improvements . . . 7

Comparison **8**

References **9**

Text Corpus Selection

We selected our corpuse of text to be **Driver.**

From the below given meanings (**NLTK**):

1. The operator of a motor vehicle
2. Someone who drives animals that pull a vehicle
3. A golfer who hits the golf ball with a driver
4. (Computer science) a program that determines how a computer will communicate with a peripheral device
5. A golf club (a wood) with a near vertical face that is used for hitting long shots from the tee

We are only interested in the **first**, **second** and **fourth** meaning of the words for our disambiguation.
We have followed 4 approaches to disambiguate the given word:

- Bag Of Words Approach
- Collocation Features Approach
- Naive Bayes
- Naive Bayes with TF - IDF

In order to run the code, please refer the Readme.md file.
Also, our code can be found on: Github

Approach 1 - Bag of Words

Model Selection

Our Bag Of Words approach comes from the family of Lexical Sample Task. This contains a small set of pre selected words on which we train our model.
We first build a vocabulary of words at the begining, find synonyms and antonyms for it, and put them in a vocabulary collection. This is our feature set. Each vocabulary word (or its related meaning) represents a position in the feature vector.

Optimise Feature Extraction

While preparing the vocabulary, we use stemming (sometimes stemming adversely affects our output but we selected to go ahead with it anyway) and lemmatisation.
We pass each word of the vocabulary through Stemming and Lemmatisation, followed by looking for similar (synonyms and antonyms) words. This will also contain the word to be disambiguated.

Making the Training/Testing Model

We read each line of the training set and check for similar words from the vocabulary and if it does exist then we mark that location of feature vector as 1, which in the hindisght means that this sentence represents some relation (maybe it is negatively co-related, but it is) and it is good to learn.
We prepare the test set in a similar fashion.

Results

TimBL on receiving the input trains and tests the data.
We tried different algorithms and the best one it worked was for **TRIBL2** algorithm with **Overlap** distance metric.
The other closest was leaving the default values for TimBL with a difference of about 5 - 10% but it was consistent in producing results.

Please check comparison for more on results.

Approach 2 - Collacational Features

Model Selection

The collocational feature model, simply keeps in mind the immediate neighbours and their features.
More importantly we record the Part-Of-Speech tags of these neighbours and use it as a feature. We search for similar patterns and train our model. The model contains unique tags that have appeared and the more they appear the stronger is relation.
Each feature is represented by the unique tags found in train and testing together. If a tag is not seen before, it is discarded.

Optimise Feature Extraction

While preparing the data, we tried removing the stop words and seems like it is not a good idea. They represent a great deal of data, even though they appear in all sentences, they give you the presence of strong verbs and proper nouns. Frequency Profiles indicated they appear in large quantities. The performance difference was incremental on keeping stop words.

Making the Training/Testing Model

Training is marked by each feature represented as a unique tag (from the Parts-of-Speech Tag) and everytime a sentence of training (or testing) is scanned and contains any of these, it is marked as 1. Last column is given as predicted class.

Results

TimBL is interesting! The IB1 works pretty intuitevly if observed. We gave it dirty test data (i.e. it did not relate or minutely related to either of the trained data to check if we are going right) and obviously it failed horribly but still gave an accuracy of 10% - 20%. On the other hand, it trained well, while testing of class we applied (tested) a few of the parameters. We tried **IB1** with different distance measures and turns out **Cosine Distance** worked best for us.
We tried IGTree where our accuracy by about 35% with the same distance metric and other distance metric. Please check comparison for more on results.

Approach 3 - Naive Bayes

Model Selection

The Naive Bayes assumption comes with a small caveat that the independence assumption discards any relation between features (words) or such entities. We cannot strongly (or at all) represent relation between features and that may sometimes be a big problem but we have gone ahead and used it.

Optimisation Feature Extraction and Modeling Data

For this classification, we generate frequency profiles for each class and use term frequency for evaluating the scores. To generate the frequency profiles, we tokenize the words for which we use NLTK. Consider a scenario where token ends with a punctuation and the same token doesnt end in punctuation. While counting the frequency for this token, we might want to consider as one token occurring twice, instead of considering them as two different tokens altogether. For this reason, we use regular expressions to pick only the characters in the alphabet and tokenize them and then determine the term frequency. Also, we wont be calculating the term frequencies for stop words. Once we have the frequency class profiles, we proceed to try and classify the unknown text.
The unknown text will be tokenized and for each token of the unknown text, we find out the occurrence of that token in the class and then calculate accordingly. This calculation must be done for all the tokens of the unknown text against each class. Once we have two final scores, we evaluate to find out the higher score and decide which class the unknown text is more likely to fall under.

Approach 4 - Naive Bayes - TF-IDF

Optimisations and improvements

One way to optimize our classification is by refining how we calculate the frequency scoring. One other way to do this by using term frequency inverse document frequency, simply put tf-idf scoring.

$$TF - IDF = tf \times idf$$
$$idf = \log(N/df)$$
$$where, N \rightarrow Totalnumberofdocuments$$
$$df \rightarrow documentfrequency$$

The idea is that, there might be words occurring more frequently and having no significance to the context. To avoid having higher scores for such words, we use tf idf to limit. It reduces the scoring of such words, depending on the word/term appearing in multiple documents of collection. Also, in our case, the two text files will be like two Text Collections and each line is like one document.

Comparison

Table 1: **Comparison of All Algorithms Used**

Method	Baseline Accuraccy	Optimised Accuracy	Optimised Parameters
Bag Of Words	0% - 60%	35% to 70% use cosine distance for TRIBL2	For IB1 - no changes,
Collocational Features	50% - 98%	55% to 92.46%	IB1 - with Cosine Distance Metric
Naive Bayes (Base and TF)	Poorly classifies 4/10 times	-	Stemming and Lemmatisation
Naive Bayes with TF IDF	Poorly classifies 2/10 times	-	Remove Stop Words

These are the results we obtained. In order to run the code, please refer the Readme.md file.

References

1. Stanford Slides
2. PYWSD
3. Wikipedia - Naive Bayes
4. Stanford NLP
5. Professor Cavars Notebook for Feature Extraction

References continued on next page...