**Assignment2 Report**

hp343

jz578

sg889

yt438

Table of Content

1. Approach 2

1.1. Dictionary-based WSD 2

1.1.1 Basic 2

1.1.2 Extension 2 4

1.1.3 Extension 3 4

1.2. Supervise WSD 4

1.2.1 Basic 4

1.2.2 Extension 6

2. Software 7

2.1. Dictionary-based WSD 7

2.2. Supervise WSD 8

3. Results 8

3.1. Dictionary-based WSD 8

3.2 Supervised WSD 9

4. Discussion 10

# Approach

## Dictionary-based WSD

### Basic

* Preprocessing

Convert dictionary.xml into this data structure:

|  |
| --- |
| dictionary [  "begin.v":{  num:1  senses:[  {  id:1  wn\_ids:[1,2,3,4]  gloss:["hello","world"]  examples:[  ["hello","world"],  ["how","are","you"]  ]  },  ...  ]  },  ...  ] |

During the conversion, we did a small adjustment to dictionary.xml, and named it dictionary-modified.xml. This modification aims to solve the quotation mark conflict in python’s xml parser. This adjustment does not interfere the accuracy of result.

And we convert validation.data into this data structure:

|  |
| --- |
| validation\_list [  {  word:"begin.v"  real\_sense:1  target\_word\_idx:15  sentence:['pressure', 'hello']  },  ...  ] |

During the preprocessing, we do the stemming and lemmatizing. Besides, we also remove stop words, for example “a”, “an”, we use a stop word dictionary named “Lucence” to remove all stop word in the word context, examples and gloss.

* Calculating

1. hit count

Count how many same words are there in the word context and in a sense’s gloss and examples.

2. Consecutive count

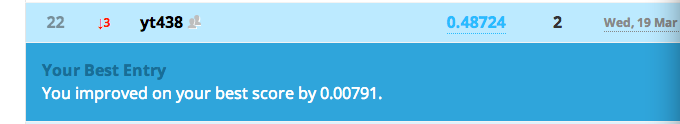
Count how many consecutive words are there in the word context and a sense’s gloss and examples. If there are two consecutive words, for example, in word context [“ab”, “cd”, “ef”] and in an example, [“ab”, “cd”, “gh”], then we add two to the consecutive count.

We add the hit count and consecutive count together, and return the sense with max sum as a best match.

* Scoring

If the best match sense calculated is the right one, then we add one to match count. After checking all the items, we divide match count by the total item number, and we get a score ranging from 0 to 1, inclusive.

* Screenshot of Kaggle



### Extension 2

We simply choose the second largest sum of hit count and consecutive count. And we change the hit rule, if the real sense is in one of the two senses, we call it a hit. And the hit rate goes up to 0.764201500536, increasing around 0.28.

### Extension 3

We redesign the score system, if the best sense matches, we add 0.7 to the match count and add 0.3 if a second best sense matches. After checking all the items, we divide the match count by 0.7 \* total item number. We will get a score ranging from 0 to 1, inclusive.

This score makes more sense because it evaluates the overall performance of our model, not only the best match result derived from the model. The second best match can also achieve some score.

## Supervise WSD

### Basic

* Preprocessing

We parse every entry in the training data or validation data into a simple form. For example,

*capital.n | 2 | … operation inside the Nicaraguan %% capital %% of Managua . And on at least one occasion…*

is parsed into an array

*[capital, 2, [operation, inside, Nicaraguan, Managua, least, occasion]]*

(the window size is 3, and stopping or functional words are removed, and the test has been lemmatized only for the reason mentioned in part 3.1)

Note every item in this array is a string or another array of strings.

Later on we transform the parsed entries into a dictionary to store learning data.

* Feature vector and calculation

First, we generate co-occurrence features in this project. We only considered *n* preceding and *n* upcoming words surrounding the target word, where *n* is the windowsize.

We strictly follow the formula in the instruction pdf to calculate the probability of a typical sense of word given the context as below

In the sense that, does not change w.r.t. , and then

To calculate each , we choose the most 1000 popular distinct words which are highly associated with one of the senses of the target words as our attributes. These words all occurred within the context in the training set. Then, the probability is calculated by

Here, add-one smoothing is used to make sure every attribute could have a probability of occurring in every sense of every word. More specifically,

So actually, we are assigning probabilities to every attribute that each of them occurs with the context of every sense of every target word. If attribute A never occurred within the context of word W whose sense is S, the probability that it occur is ((0+1) / (count(s) + V)), which is very close to but not zero.

To mention one point, we sum up log(P) instead of multiplying the probability directly one after another to avoid the overflow issue when doing the computation.

Then we add collocational features into the project. We consider the pre1-word, pre2-word, fol1-word and fol2-word. The basic calculation is the same as above, however, the main difference is that, instead of checking the count of a context word existing in the windowsize of the target word, we check the count of the context word existing in the specific location.

Finally, our probability of a typical sense S w.r.t. all the senses of word W, is calculated as below:

Where count(w) means the number of entries in the training set which are associated with word W.

* Scoring

As mentioned above,

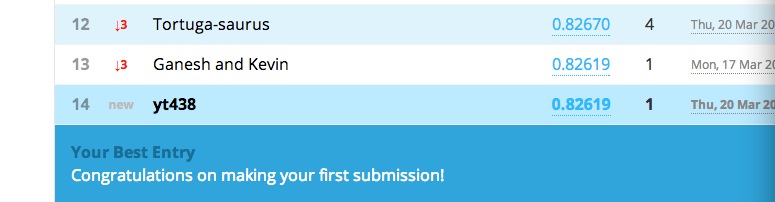
For instance, let the windowsize be 3. When evaluating the possibility of sense S given feature f, we actually view this feature f as a 1000-dimentional vector. We somehow want to compare them by the product of the probabilities. In order to do this, there are three steps:

1. Check to see if the six surrounding words are attributes we previously defined. Ignore the words that are not attributes.
2. Calculate the product of the probabilities of the remaining attributes
3. Rank the products and make decision.

By this way we could let these senses compete equally by having same number of attributes involved in the calculation. Even if none of the six words are attributes, the formula will still equal to P(s), which implies the most possible sense in the training set without considering features.

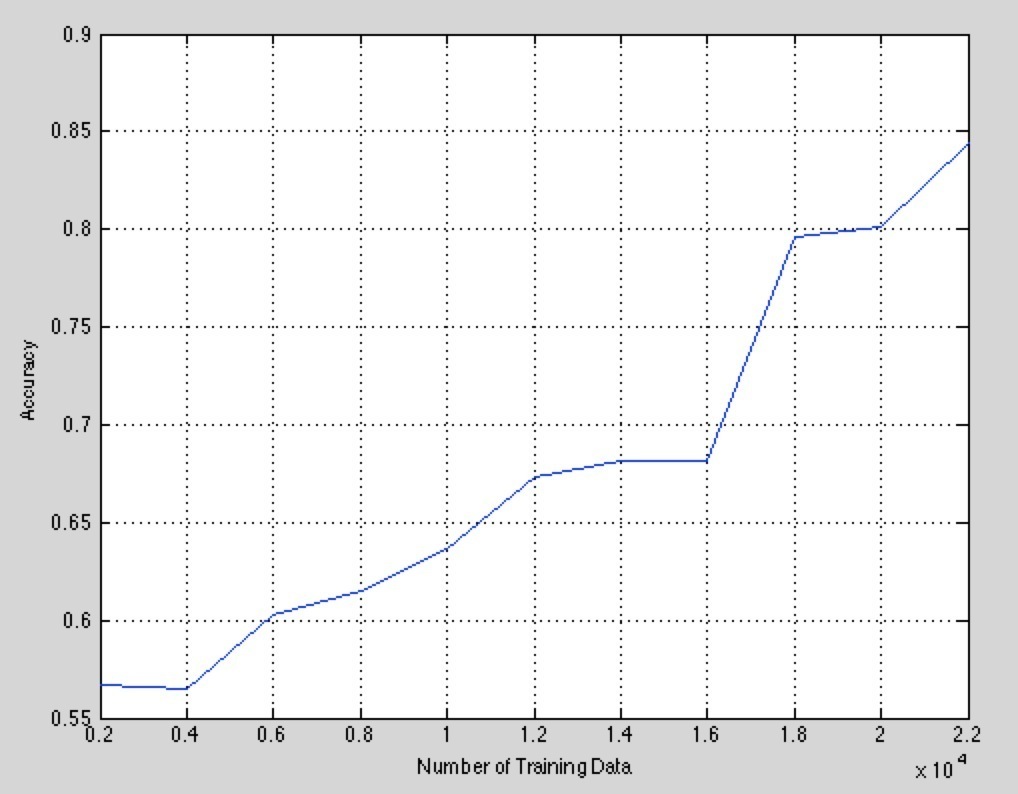
We mainly improve our results by finding the best windowsize, the suitable size of the top frequent context words, and adding collocational features.

* Screenshot of Kaggle



### Extension 4

In this part we did what is exactly required in option4 in the instruction document. Based on the validation data, the “learning curve” for the Naive Bayes approach is as below:



We can see that the accuracy increases with the size of the training set, which is because we can generate features more accurately based on large training data.

# Software

We use the LancasterStemmer and WordNetLemmatizer library from nltk to do stemming and lemmatizing. And we also use wordnet corpus in creating dictionary.

### Dictionary-based WSD

Change the **glob\_valid\_path** and **glob\_dict\_path** in dic\_preprocessing.py, so that these two variables contain the address of validation\_data.data (or test.data)and dictionary-modified.xml

If you want to run the basic version, just uncomment the line “**print basic\_score(dictionary, validation\_list)**”, and run “**dic\_wsd.py**” in command line tools.

If you want to run the extension 2 for dictionary-based WSD, uncomment the line “**print second\_best\_match(dictionary, validation\_list)**”, and run “**dic\_wsd.py**” in command line tools.

If you want to run the extension 3 for dictionary-based WSD, uncomment the line “**new\_score\_match(dictionary, validation\_list)**”, and run “**dic\_wsd.py**” in command line tools.

### Supervise WSD

In “**main.py**”, add **evaluateValidFile('validation\_data.data', 2)** tooutput the accuracy. The first attribute is the name of the validation file, and the second is the windowsize.  
  
Comment out evaluateValidFile() and add **decodeTestFile(testFileName, windowsize)** to output the best matching sense ID of the target word in each example into a file **called supervised\_test.output**.The first attribute **testFileName** is the name of the test file, and the second is the windowsize.

Run“**main.py**” in command line tools to get the result.

# Results

## Dictionary-based WSD

Basically, the Dictionary-based WSD achieves a score of 0.48724 on test data.

* Use stemming

As mentioned before, we do Lemmatizing does not change the word quite much. However, when it comes to stem, things are quite different. For example, in the word “begin.v”, in the examples, there are past tense, “began”, however, after stemming, it turns out to be “beg”, and “begin” remains to be “begin”. This is not we want, what we want is just convert “began” to “begin”. So we decided to remove stemming at first, however, after calculating the hit rate on validation.data, we found out that after stemming, the rates goes higher (score goes from 0.424437299035 to). So eventually, we add the stemming process into our program.

* Sum up hit count and consecutive count

This is not our first design, we used to take the max length of consecutive words as consecutive count, and it comes with highest priority. This means no matter how big the number of hit count of a sense is, if it have a very low consecutive count, it won’t be the best match. The basic assumption is that if we have a comparatively long consecutive sentence, it’s more likely to be the best match sense because even other sense has more hit count, the hit might be caused by very popular words. However, this approach does not work as well as we expected. After checking the result, we find out that the max consecutive word length is relatively small, usually 1-2, this may not be very distinguishable. Besides, we remove most stop words before, which makes it harder to find consecutive match words. This method achieves a basic score of 0.488745980707

## 3.2 Supervised WSD

* Use stemming

As mentioned before, we only consider doing Lemmatizing.

* Effect of the size of top frequent words

We generate a list of all the words occur in the training data and sort them from high frequency to low frequency. We tested different sizes, and found that with the size as 1000, it outperforms the size as 500 or 2000. It is because 500 words are not enough to be used as our dictionary, and 2000 words cause the problem of overfitting. So we decided to use those 1000 words as our co-occurrence features.

* Effect of windowsize

We only consider *n* preceding and *n* upcoming words surrounding the target word, where *n* is the windowsize. We tested our approaches on the validation file. We use 1000 most frequent words in the training set as our dictionary. Based on different windowsizes, we trained separate models per each target word in the training data. Then we tested our models on the validation data, and the accuracy is as below:

|  |  |
| --- | --- |
| Windowsize | Accuracy |
| 0 | 0.811361200429 |
| 1 | 0.840300107181 |
| 2 | 0.842443729904 |
| 3 | 0.841371918542 |
| 4 | 0.831725616292 |
| 5 | 0.834941050375 |
| 6 | 0.825294748124 |
| 7 | 0.818863879957 |
| 8 | 0.827438370847 |
| 9 | 0.827438370847 |
| 10 | 0.825294748124 |

As we can see, windowsize 2 gets the best performance with an accuracy of 84.24%.

* Effect of collocational features  
  We added pre1-word, pre2-word, fol1-word and fol2-word as our collocational features. With windowsize of 2 and 1000 co-occurrence features, the performance is as below:

|  |  |
| --- | --- |
| Feature | Accuracy |
| Only Co-occurrence | 0.842443729904 |
| Co-occurrence +  pre1-word + fol1-word | 0.84780278671 |
| Co-occurrence +  pre1-word + pre2-word +  fol1-word + fol2-word | 0.845659163987 |

We can see that adding collocational features helps increase the accuracy, but adding too many might cause the problem of overfitting. So we decided to add the pre1-word and fol1-word as our features.

* Summary

Our approach outperforms the baseline approach that always predicts the most frequent sense.

|  |  |
| --- | --- |
| Approach | Accuracy |
| Baseline | 0.811361200429 |
| Co-occurrence +  pre1-word + fol1-word | 0.84780278671 |

# Discussion

Dictionary focuses on analyzing the similarities between word context and different senses. So it more suits the situation where the we have enough examples and definition for different senses.

So, even the given validation set is very small, which means we do not have enough data to do training, we can also use dictionary based method, because we can use other dictionary, for example, Wordnet to give us data support. However, the accuracy of dictionary-based method does not change much as we enlarge the dictionary size. In this assignment, when we add Wordnet into our code, the score changed less than 0.002.

However, when the training data set is big enough, the accuracy of supervised WSD approach increases dramatically, which win a higher score than dictionary-based method. So, when we have enough data, we prefer supervised-WSD.

When we have enough structured training data, the learning-based supervised WSD approach is more suitable. In the supervised WSD approach, our performance mainly rely on the feature vectors we generate form the context. We chose the 1000 co-occurrence features and 2 collocational features as our feature vector, and optimize on the windowsize to get a good result. Based on our experiment, the co-occurrence feature, the pre1-word feature and the fol1-word feature are the top three most informative features. We looked into some examples in the validation file and analyzed them as below:

In Example A, we can see that the word ‘backlog’ occurs in the context. In our training data, ‘backlog’ is one of the 1000 co-occurrence features. When it occurs with the target word ‘order.n’, there is a high possibility that the target word is with the sense ID 6, and the example here proves that.

In Example B, the target word ‘rates.n’ with the first sense occurs immediately after the word ‘interest’, which matches with the training data (see line 708). This demonstrates the usefulness of the pre1-word feature.

In our generated model, it shows that if the fol1-word of the target word ‘go.v’ is ‘down’, the target word mostly means the first sense. The Example C is exactly the case, and that proves the fo1-word feature makes sense.

**Example A**

(line 522 of the validation file) order.n | 6 | But to traders , it looked like disaster on the 9:30 a.m. opening bell . The Dow Jones Industrial Average opened down 1.64 shortly after 9:30 . But most of the 30 blue-chip stocks in the average , including Eastman Kodak and General Motors , could n't trade because of the heavy backlog of sell %% orders %% left over from Friday 's late-afternoon rout . At 9:45 , Procter & Gamble -- one of the most important Dow bellwethers of late -- opened down 2 3\/4 to 117 . The Dow dropped to a quick 27-point loss , and to many traders it looked as if stocks were headed for yet another big tumble .

**Example B**

(line 175 of the validation file) rate.n | 1 | Thus , an institution obligated to make fixed-rate interest payments on debt swaps the payments with another making floating-rate payments . In most of the British transactions , the municipalities agreed to make floating-rate payments to banks , which would make fixed-rate payments . As interest %% rates %% rose , municipalities owed the banks more than the banks were paying them . The court hearing began in early October at the request of Anthony Hazell , district auditor for Hammersmith , who argued that local councils are n't vested with constitutional authority to engage in such capital-markets activities . The council backed the audit commission 's stand that the swap transactions are illegal .

**Example C**

(line 563 of the validation file) go.v | 1 | This may sound strangely optimistic . After all , until a few years ago , the stock market was viewed as a barometer of the national economy . When it %% went %% down , by all tradition , the economy followed . That has changed , partly because the two years following the worst stock-market plunge in history have been reasonably comfortable . The 1987 crash was `` a false alarm however you view it , '' says University of Chicago economist Victor Zarnowitz .