

---

# Devoir 2 IFT3335

## Report - Using Classification for Word Sense Disambiguation

---

**Steve Levesque**  
Département d'informatique et  
de recherche opérationnelle  
Université de Montréal  
steve.levesque@umontreal.ca

**Weiyue Cai**  
Département d'informatique et  
de recherche opérationnelle  
Université de Montréal  
weiyue.cai@umontreal.ca

### Abstract

This practical work aims to try out classification algorithms (machine learning) on NLP. We will deal with the word sense disambiguation problem, which aims to classify a word used in a context into the appropriate sense class.

## 1 Introduction

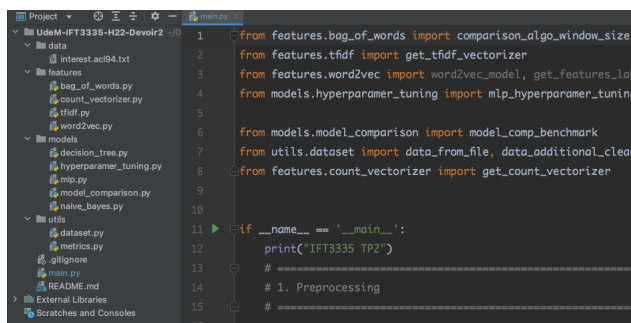
### 1.1 Description of Report

This work will discuss multiple algorithms, with established characteristics (stopwords removal + stemming) and interesting sets of features. Following the implementation, multiple aspects will be discussed: a comparison in performance from multiple algorithms (i.e. Naive Bayes, Decision Tree and MultiLayerPerceptron), same for them towards their sets of features, feature engineering with word2vec and possible features that would of been worth exploring.

### 1.2 How to Execute Program

The scripts should work "out of the box" if all modules are installed.

In any IDE supporting traditional Python, the run configuration should reference the script `main.py` and the full project should have every files given in the submission (i.e., all scripts, the data file `interest.ac194.txt` in the data folder).



The screenshot shows a code editor with a project structure on the left and the `main.py` file open in the editor. The project structure includes folders for `data`, `features`, `models`, and `utils`. The `main.py` file contains the following code:

```
1 from features.bag_of_words import comparison_algo_window_size
2 from features.tfidf import get_tfidf_vectorizer
3 from features.word2vec import word2vec_model, get_features_label
4 from models.hyperparameter_tuning import mlp_hyperparameter_tuning
5
6 from models.model_comparison import model_comp_benchmark
7 from utils.dataset import data_from_file, data_additional_clean
8 from features.count_vectorizer import get_count_vectorizer
9
10
11 if __name__ == '__main__':
12     print("IFT3335 TP2")
13     # 1. Preprocessing
14     # ...
```

## 2 Preprocessing

The preprocessing consists of getting the data ready for the interpretations of algorithms. In such matter, it can slightly differ from one to another. Here are the key steps in the transformation for important algorithm steps:

### 2.1 For all Algorithms

The NP (Noun Phrase) structural marks on the sentence are removed for all steps.

```
[ yields/NNS ] on/IN => yields/NNS on/IN
```

### 2.2 Bag of Words (BoW)

The grammatical category is removed and the words are left alone.

```
[ yields/NNS ] on/IN => yields, on
```

### 2.3 Categories

Respectively, the opposite from BoW is done, which gives us only the category keyword without the word in question.

```
[ yields/NNS ] on/IN => NNS, IN
```

## 3 Naive Feature Extraction: Using Whole Sentence as a Feature [Implemented]

The naive feature consist of only using the whole sentence (with established characteristics like stopwords removal + stemming) for the algorithms.

It is an overall simple and fast method. Based on the results of experiments, Count Vectorizer is more effective than Tfidf Vectorizer even though Tfidf is normally considered better than count vectorizer.

3. Naive feature extraction						
	naive bayes		decision tree		MLP	
	accuracy	f1	accuracy	f1	accuracy	f1
Count Vectorizer	0.8354	0.8236	0.8249	0.8248	<b>0.8565</b>	<b>0.8512</b>
Tfidf Vectorizer	0.6646	0.5801	0.7890	0.7915	0.8270	0.8186

In Count Vectorizer, we only count the number of times a word appears in the document which results in biasing in favour of most frequent words. this ends up in ignoring rare words which could have helped is in processing our data more efficiently. <sup>1</sup>

However, in a large text corpus, some words will be very present (e.g. "the", "a", "is" in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms. In order to re-weight the count features into floating point values suitable for usage by a classifier it is very common to use the tf-idf transform to consider overall document weight of a word. It helps us in dealing with most frequent words. <sup>2</sup>

In our case, since our corpus is very small, Count Vectorizer can be better than Tfidf.

<sup>1</sup><https://www.quora.com/What-is-the-difference-between-TfidfVectorizer-and-CountVectorizer-1>

<sup>2</sup>[https://scikit-learn.org/stable/modules/feature\\_extraction.html#text-feature-extraction](https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction)

## 4 Algorithms Comparison: Using Bag of Words (BoW) and Categories of the Adjacent Words (POS) as 2 Sets of Features [Implemented]

The algorithms are now using data which contain 2 sets of features: the words of the sentence and the their respective categories. A window size is taken into account to define the scope of words included in the disambiguation.

As given in the homework instructions, the words are from around the subjected word through the outbounds (incremented by each window size unit) as such:

```
window_size = 2
'NNS', 'IN', 'NNS', '.'
=> (C-2=NNS, C-1=IN, C1=NNS, and C2=.)*
```

\*The most outbound being 2, and the "." being the strict outbound of the sentence.

A outbound most left or most right is an empty string if it does not exist. It can happen if such an outbound goes "outside" a sentence's maximal count of words. Such an example is possible with the first sentence:

```
window size = 2 => [['nns', 'in', 'nns', '.']]
window size = 3 => [['jj', 'nns', 'in', 'nns', '.', '']]
```

Every empty value is considered as zeros in the created vector. It is observable in relation of this property that a window size too large will add bias with lots of zeros, which lowers the accuracy and f1 score.

After having obtained the BoW and POS, we converted the texts into vectors by using Count Vectorizer or Tfidf Vectorizer; and we used onehot encoding to transform categories of words into numerical values.

### 4.1 Count Vectorizer

A window size of 1 to 3 is appropriate in this case. This range probably has very few trailing zeros added from the discussion earlier.

4.1. Count Vectorizer						
window size	naïve bayes		decision tree		MLP	
	accuracy	f1	accuracy	f1	accuracy	f1
1	0.8755	0.8711	0.8840	0.8862	0.9072	<b>0.9065</b>
2	0.8819	0.8746	0.8460	0.8454	0.8966	0.8948
3	0.8882	0.8789	0.8586	0.8596	0.9051	0.9028
4	0.8608	0.8497	0.8565	0.8560	<b>0.9093</b>	0.9045
5	0.8544	0.8431	0.8207	0.8223	0.8861	0.8808
6	0.8439	0.8297	0.7911	0.7950	0.8776	0.8710
7	0.8165	0.7954	0.7954	0.7970	0.8481	0.8419
8	0.8059	0.7834	0.7869	0.7877	0.8586	0.8510
9	0.8059	0.7843	0.7722	0.7744	0.8713	0.8653
10	0.8038	0.7844	0.7890	0.7879	0.8671	0.8610

### 4.2 Tf-Idf Vectorizer

It seems that a window size of 1 does not does much difference with Tf-Idf, probably with the fact we don't use enough context. However, the performance drops compared to count vectorizer when window size becomes larger.

4.2. Tfidf Vectorizer						
window size	naive bayes		decision tree		MLP	
	accuracy	f1	accuracy	f1	accuracy	f1
1	0.8565	0.8492	0.8692	0.8698	<b>0.9093</b>	<b>0.9087</b>
2	0.8460	0.8334	0.8671	0.8668	0.8755	0.8732
3	0.8143	0.7976	0.8481	0.8480	0.8903	0.8855
4	0.8017	0.7768	0.8228	0.8235	0.9008	0.8958
5	0.7722	0.7377	0.8207	0.8276	0.8650	0.8585
6	0.7405	0.6988	0.7827	0.7867	0.8565	0.8500
7	0.7236	0.6704	0.7743	0.7776	0.8249	0.8176
8	0.7236	0.6703	0.7257	0.7309	0.8186	0.8074
9	0.7089	0.6541	0.7405	0.7367	0.8418	0.8339
10	0.7025	0.6477	0.7489	0.7512	0.8228	0.8143

## 5 Feature Engineering: Using Word2vec to Represent with Vectors [Implemented]

In this section, we transformed every word into a vector of size 100 with customized word2vec model and 'glove-wiki-gigaword-100' pre-trained model. Since our corpus is very small, we decided to use pre-trained model which gave us a better result.

For simplicity of implementation, stopwords removal and stemming has been omitted. However, we can compare our previous results and see that BoW with Word2Vec is more effective. There is clearly an advantage to use extraction of features with Word2vec.

### 5.1 Whole Sentence

5.1. MLP + word2vec for whole sentence	
accuracy	f1
0.7869	0.7773

This "disastrous" result is caused by the fact that we set 100 as maximal length of each sentence so that we can transform all the texts into a matrix of size [2368, 10000]. If the length of sentence is smaller than 100, we fill all empties with zeros. If the length of sentence is larger than 100, we truncate the sentence to keep the first 100 words.

Here is the implementation details:

```
Sentences : [1] = 20 words, [2] = 10, [3] = 200, [4] = 100, ...
max_length = 100
fill all empties with zeros
for example, [1] => [a vector of 10000 elements]
first 2000 elements representing 20 words
and we fill the empties with zeros.
[3] => [a vector of 10000 elements]
these 10000 elements represent the first 100 words
because we truncate the last 100 words.
```

We can observe lots of zeros where there should not be and loss of information when the sentence is too long, which explains the drop of performance.

### 5.2 Bag of Words

Results without stopwords removal and stemming. Better than BoW with 2 set of features.

5.2. Bag of Words		
window size	MLP	
	accuracy	f1
1	0.8987	0.8976
2	0.8966	0.8946
3	<b>0.9156</b>	<b>0.9127</b>
4	0.8882	0.8851
5	0.8840	0.8806
6	0.8797	0.8760
7	0.8650	0.8608
8	0.8523	0.8481
9	0.8565	0.8556
10	0.8418	0.8366

## 6 Other Possible Features that Would of Been Worth Exploring

Here are some features that are interesting to discuss in our opinion. However, because of high complexity and minimal time left, none of them are implemented as for the submission.

### 6.1 Basic feature engineering for NLP [Not Implemented]

In this section, we will list some features<sup>3</sup> which can be used to improve the metric.

1. Number of Characters: count the number of characters present in a tweet.
2. Number of words: count the number of words present in a tweet.
3. Number of capital characters: count the number of capital characters present in a tweet.
4. Number of capital words: count the number of capital words present in a tweet.
5. Count the number of punctuations
6. Count of stopwords
7. Calculating average word length This can be calculated by dividing the counts of characters by counts of words.
8. unique words vs word count feature: the ratio of unique words to a total number of words.
9. stopwords count vs words counts feature: the ratio of counts of stopwords to the total number of words.

We can add these features above and the categories of words that we obtained in section 4 (POS Tagging) into the dataset that we have already pre-processed using word2vec.

### 6.2 NP Context (Noun/Nominal Phrase) [Not Implemented]

This feature exploits the sentence nominal phrase context to give a sence of relation for words that could hardly be related without this information:

```
i.e.
[fresh hot dog] => the food with a bun and sausage
--
Interest:
[Those expensive interests] => referring to stocks high probably
```

Those are simple NP definitions, but lets retake an example we used previously:

```
window_size=2, like previous example with C-2,C-1,C1,C2
[ further/JJ declines/NNS ] in/IN [ interest_6/NN rates/NNS ] ./
(added the whole left side NP for explanation)
```

<sup>3</sup><https://www.analyticsvidhya.com/blog/2021/04/a-guide-to-feature-engineering-in-nlp/>

The representation could be of this sort, where the set of feature is a vector with a numerical value. The length would be the size of the sentence and the values would be between 0 (not in a NP) to 1 within a NP scope to value intensity from the center of a NP.

Why?, because we use a window size and information would probably be lost if we use size 1 or 2.

```
We lose the full NP and the word further with window_size=2:
[1, 0, 1, 0]
if window_size=3:
[1, 1, 0, 1, 0, 0]
```

We could just make them all 1's for simplicity if some information loss for huge NPs is acceptable, because our example above does not have a drawback.

```
[1, 0, 1, 0]
```

However, here is one where a cut is made in the NP, but we have a way to "know" it is partial:

```
[fresh hot dog] interests me a lot
window_size=3 => [2, 1, 2, 0, 0, 0]
window_size=2 => [1, 2, 0, 0]
```

## 7 Hyperparameter Tuning: Number of Hidden Neurons

The best choice of algorithms and parameters are used for the tuning on the neurons. Keeping in mind that the best disambiguation setting is a crucial criteria.

Complete model with parameters : MLP + word2vec + BoW with window\_size=3 + 200 as number of hidden neurons. We obtained 0.9156 as accuracy and 0.9127 as f1 score.

7. Hyperparameter Tuning: Number of Hidden Neurons		
hidden layer sizes	accuracy	f1
100	0.9030	0.9008
150	0.9072	0.9048
200	<b>0.9156</b>	<b>0.9127</b>
250	0.9030	0.9009
300	0.9051	0.9035
350	0.9051	0.9027
400	0.9093	0.9070
450	0.9072	0.9048
500	0.9093	0.9071
550	0.9135	0.9107
600	0.9114	0.9088
650	0.9114	0.9091
700	0.9135	0.9111
750	0.9072	0.9043
800	0.9030	0.9008
850	0.9093	0.9069
900	0.9051	0.9030
950	0.9135	0.9115

## 8 Conclusion

To conclude, we can compare all algorithms and see that a word's disambiguation has a lot of information from its neighbors and that the formulation of the data is really important both for speed and accuracy.

## References

- [1] Word Sense Disambiguation - Wikipedia  
[https://en.wikipedia.org/wiki/Word-sense\\_disambiguation](https://en.wikipedia.org/wiki/Word-sense_disambiguation)
- [2] Noun Phrase Semantic Segmentation  
[https://intellabs.github.io/nlp-architect/np\\_segmentation.html](https://intellabs.github.io/nlp-architect/np_segmentation.html)
- [3] What is the difference between TfidfVectorizer and CountVectorizer?  
<https://www.quora.com/What-is-the-difference-between-TfidfVectorizer-and-CountVectorizer-1>
- [4] Scikit-learn  
<https://scikit-learn.org/stable/>
- [5] Gensim word2vec  
<https://radimrehurek.com/gensim/models/word2vec.html>
- [6] nltk.tokenize package  
<https://www.nltk.org/api/nltk.tokenize.html>