Metaphor Detection

Using Data Mining and Machine Learning

Haroon Qahtan, Yassin Alsalani(*Author*)
Department of Global Studies Institute
Georgia State University
Atlanta, Georgia
Email: hqahtan1@student.gsu.edu, yalsahlani1@student.gsu.edu

***Abstract*—This research paper dives deep into automatic metaphor detection using natural language processing and data mining. According to Google's dictionary, a metaphor; a met·a·phor is a figure of speech in which a word or phrase is applied to an object or action to which it is not literally applicable. An example of a metaphor is: "He drowned in a sea of grief." While an example of a literal sentence is: "The coffee is hot." This paper researches three methods of detecting metaphors through natural language processing. The first is using a One Class Support Vector Machine (SVM) approach. The second is using a Naïve Bayes Classifier Approach. Finally, the third is by using a Random Forest Classifier approach.**

***Keywords*—metaphor detection; automatic; sentences; Natural Language Processing; metaphorical**

I. Introduction

This paper describes three new approaches to metaphor detection using natural language processing. Two methods are supervised machine learning approaches which are Naive Bayes Classifier, and Random Forest Classifier, and one method is an unsupervised machine learning algorithm; One Class SVM. There have been many papers published that address metaphor detection through the process of machine learning, however, it has become clear to me that there could have been many different approaches on prepping the dataset to test the models on. Some approaches of metaphor detection from different papers include a Corpus-based Analysis approach where a huge corpus is annotated and used for the classifying certain types of words in many domains (MacWhinney and Fromm, 2014). Also another popular used method for metaphor detection does not use machine learning models, instead it decides the predicted output of a given sentence as metaphorical or not using the pre-set rules that it has been given such as verb and noun rules. However, using a corpus based method doesn't allow the ability to create automatic metaphor detection tools since after the information has been extracted from a corpus, it still needs an analyst to decide which collocations are likely to be metaphorical and which are likely to be literal (MacWhinney and Fromm, 2014). The purpose of this paper is to introduce three new proposed methods for classifying metaphors, but more importantly, have it so that it is automatic detection using machine learning models that increase in accuracy when more data is provided to train the models. Also, this research paper put a lot of emphasis on the preprocessing of the data, since which a clean dataset derived from reliable sources, we can obtain a higher accuracy that can be measured relative to data that has not been properly cleaned.

II. Obtaining the Datasets

One of the most challenging phases that have occurred in the research of this project is obtaining the dataset. There were not many corpora online that contained labeled metaphoric sentences and literal sentences. Therefore, the best option was to create our own dataset from scratch. This would have been a very tedious task for the metaphor labeled dataset, however, we were fortunate enough to find that The University of Virginia has a database of around 15,000 labeled metaphor sentences. The next step was to obtain these sentences, so I wrote a Python script that scraped the website and retrieved the metaphor sentences and outputted it in a csv file format. After doing so, the outputted dataset contained around 15,000 metaphoric sentences, however, they still needed to be preprocessed.

Moving on from the metaphor dataset, it was time to find a dataset of literal sentences. Finding the literal sentences was the most challenging thing to do in this research project. This is because there are no databases online that contained only literal sentences. Most databases that I came across contained a mix of hypernyms, metaphors, idioms etc. therefore, it was best for me to create my own dataset from online resources.

I tried to get very creative with how I'm creating this database, and one idea was to get sentences from history books since to my knowledge they were very literal. However, that plan soon failed since I could not find many history books that only contained literal sentences.

The other approach was to scrape Wikipedia and retrieve the first sentence from each random article the I requested. So, therefore, I wrote a Python script that requested random Wikipedia articles and retrieved the first sentence of that article. However, one issue that kept appearing is that some of these articles had metaphors in the first sentence and better yet, some were in completely different languages. Therefore, when I exported the sentences to a text file, there was an abundant amount of sentences that had Unicode characters that my program did not interpret.

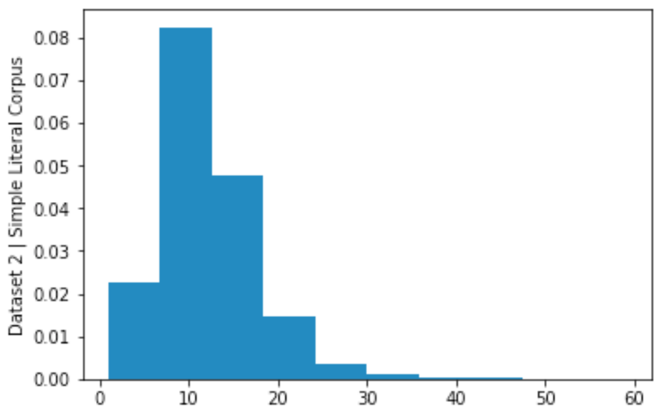
I finally found a solution when I found the Simple Wikipedia website. This is where Wikipedia has created simplified versions of its articles in simple English. This means that metaphoric sentences were less likely used and this allowed for a larger dataset of literal sentences. This was perfect to what I was looking for, so therefore, I ran my Python script to scrape random Simple Wikipedia pages and output the sentences to a text file.

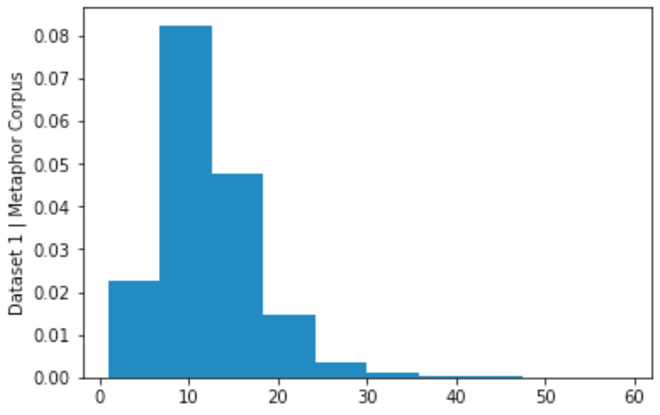
III. Cleaning The Data

Cleaning the data for this type of research project required a different approach compared to other machine learning problems. Since this research was focused on text data and natural language processing, I needed to use the Natural Language Processing Toolkit provided by Python in order to clean up the data so that it is ready for processing.

After loading both of my datasets (Metaphor sentences dataset, and literal simple sentences dataset) into a Pandas data frame (python's matrix computation library), it was ready for the first step. The datasets were loaded into a Pandas data frame and automatically labeled as being metaphoric (1) or literal (0). This is done because I have loaded 2 separate datasets so therefore, I knew in advance which dataset contained metaphor sentences and which one contained literal sentences.

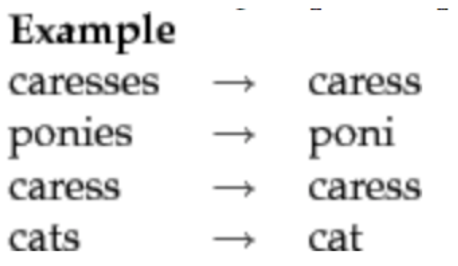
A problem that came up when using the two datasets is that they were not normalized. Meaning that each dataset had different number of words per sentence and the metaphor dataset had sentences that were of higher word average than the simple dataset. Therefore, to remove any biases, we normalized the two datasets by finding the average words of each sentence in the Simple sentence dataset and then for each item we match it with the larger metaphor dataset. In the end we toss out out all of the items from both data sets for which there isn't a match. What we'd have is a data set that contains pairs of items at each length where we'd be just as likely to at random grab a simple or a complex sentence for any given length. It would also yield a joint corpus with the same mean and standard deviation as shown in the histograms of the two datasets below.





Now that the two datasets are normalized, the next step was to clean out any HTML and URL links that may be located in the text. After that we clean the text by only keeping letters that are in the alphabet from (A-Z-a-z), that being capital and non capital letters. One of the reasons for doing this was to remove any punctuation marks as well as unrecognized Unicode characters. The next step was to lower all the words to lowercase letters.

After that the next step was to stem the words so that we retrieve the root of the word instead of keeping the different variation of that word. For example, as shown in the figure below from Stanford's NLP tutorial, the word "caresses" is now taken down to the root word "caress".



Example

caresses	$\rightarrow$	caress
ponies	$\rightarrow$	poni
caress	$\rightarrow$	caress
cats	$\rightarrow$	cat

In order to do this, the PorterStemmer provided by Python's Natural Language Processing Toolkit was used to separate each word in a sentene indivitualy and break them down to their root word.

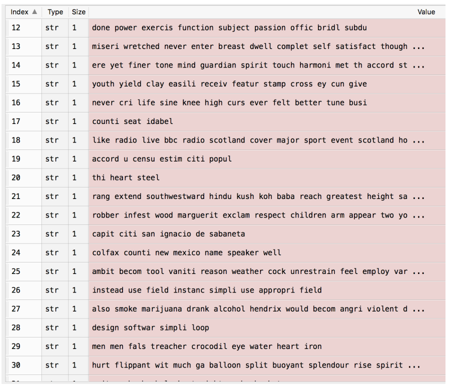
Once that process was complete, the next step was to remove all the stopwords in each sentence. This is done because you do not need the stop words in each sentence to have a comprehendable sentence. Since even after removing the stopwords, the sentences will still cointain their meaning.

Stopwords include words such as: a, an, are, and, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that. I used the Corpus module from the Natural Language Processing Toolkit to import the stopwords module and remove any stopwords that may appear in each of the sentences that were in both datasets (Metaphor and Literal Datasets).

Then for the final cleaning process, it was time to join all these words together and append them to a corpus dataset so that they are organized and ready for text processing.

IV. Converting The Data

The next step in the process of processing text is to convert the text from "text" to vectors so that the machine can undestand and that the selected machine learning algorithms can be applied appropriety. Up until this moment the corpus I have created as looked something like the figure shown below:



Index ▲	Type	Size	Value
12	str	1	done power exercis function subject passion offic bridl subdu
13	str	1	miseri wretched never enter breast dwell complet self satisfact though ...
14	str	1	ere yet finer tone mind guardian spirit touch harmoni met th accord st ...
15	str	1	youth yield clay easili receiv featur stamp cross ey cun give
16	str	1	never cri life sine knee high curs ever felt better tune busi
17	str	1	counti seat idabel
18	str	1	like radio live bbc radio scotland cover major sport event scotland ho ...
19	str	1	accord u censu estim citi popul
20	str	1	thi heart steel
21	str	1	rang extend southwestward hindu kush koh baba reach greatest height sa ...
22	str	1	robber infest wood marguerit exclam respect children arm appear two yo ...
23	str	1	capit citi san ignacio de sabaneta
24	str	1	colfax counti new mexico name speaker well
25	str	1	ambit becom tool vaniti reason weather cock unrestrain feel employ var ...
26	str	1	instead use field instanc simpli use appropri field
27	str	1	also smoke marijuana drank alcohol hendrix would becom angri violent d ...
28	str	1	design softwar simpli loop
29	str	1	men men fals treacher crocodil eye water heart iron
30	str	1	hurt flippant wit much ga balloon split buoyant splendour rise spirit ...

All the sentences were now cleaned and it was now time to convert all the sentences in the corpus into a Co-occurrancy matrix. The main purpose of the co-occurrancy matrix is to "calculate the abstractness level of a given word. The idea behind this matrix is that the co-occurring words are more likely to share an identical concept" (Shlomo and Last, 2014). An example of how a co-occurrence matrix looks like is show in the figure below:

$$
X = \begin{array}{c|cccccccc|}
 & I & like & enjoy & deep & learning & NLP & flying & . \\
\hline
I & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
like & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
enjoy & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
deep & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
learning & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
NLP & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
flying & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
. & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}
$$

We set the max feature limit of our co-occurrence matrix to be at 1000. After constructing our co-occurrence matrix, it was time to set out Y label equal to the second feature that was in our dataset, which was the binary number of 0 and 1. 0 being that the sentence was not metaphorical, and 1 being that the sentence is metaphorical.

Now that our data has been converted into a vector of numbers, it can now be used as input to a machine learning algorithm. However, before that I needed to split the dataset into a training and testing set. The testing size being 20% of the dataset and the training being 80% of the dataset.

V. Choosing The Right Classifier

At the beginning of this research project, it was very difficult to find a database that contains only literal sentences, so therefore, since I only had the metaphor dataset at the time, I research if it was possible to train a machine learning model on one set of label. In that example, for my project it would be training a machine learning model on the metaphor sentences dataset which only contained one label being a metaphor.

After much research online, I stumbled upon the first classifier that I used which is Sckikit's Learn One-class SVM which is "an unsupervised algorithm that learns a decision function for novelty detection: classifying new data as similar or different to the training set" (scikit, 2017). This allows for the metaphor dataset to be trained on itself so that anything outside the original data set is negative data determined by a gamma allowed error rate.

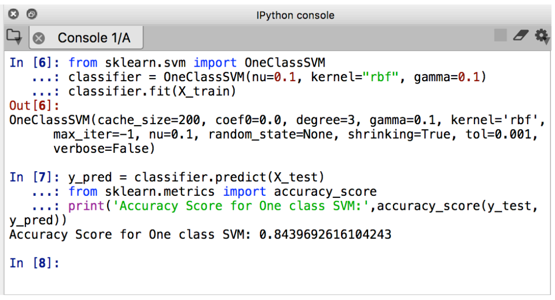
The second classifier that I tested out is Naïve Bayes Classifier. It is a classifier that is based on Bayes Theorem. I chose to test out the Naïve Bayes Classifier because NLP Stanford website, it describes that the Naïve Bayes Classifier is good with text classifications. Therefore, I decided to go forward and implement it in my design.

The third and Final Classifier that I used is the Random Forest Classifier. Random Forest is known to be a really good classification tool for most machine learning problems so therefore, I decided to use it for text classification.

VI. Results

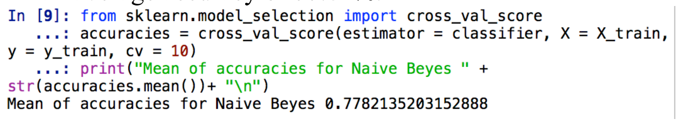
After experimenting with the three classifiers, I decided to test their accuracy based on my dataset.

For the One Class SVM Classifier, the results are shown in the next figure. The accuracy from the unsupervised classifier is around 84.4%. This was a one label test so therefore it was tested on only metaphorical sentences. The result of this classifier is either a 1 or a -1. 1 meaning that it is apart of the class and -1 meaning that it is not apart of the class.



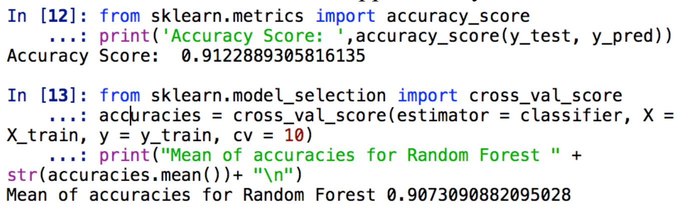
```
In [6]: from sklearn.svm import OneClassSVM
   ...: classifier = OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
   ...: classifier.fit(X_train)
Out[6]:
OneClassSVM(cache_size=200, coef0=0.0, degree=3, gamma=0.1, kernel='rbf',
      max_iter=-1, nu=0.1, random_state=None, shrinking=True, tol=0.001,
      verbose=False)

In [7]: y_pred = classifier.predict(X_test)
   ...: from sklearn.metrics import accuracy_score
   ...: print('Accuracy Score for One class SVM:',accuracy_score(y_test,
y_pred))
Accuracy Score for One class SVM: 0.8439692616104243

In [8]:
```

For the Naïve Bayes Classifier results shown in the next figure, the accuracy is a 78.72%. However, just to be sure of our results I conducted a 10-fold cross validation to get the mean of all the accuracies. What I ended up with is an average accuracy of 77.82%.



```
In [9]: from sklearn.model_selection import cross_val_score
   ...: accuracies = cross_val_score(estimator = classifier, X = X_train,
y = y_train, cv = 10)
   ...: print("Mean of accuracies for Naive Beyes " +
str(accuracies.mean())+ "\n")
Mean of accuracies for Naive Beyes 0.7782135203152888
```

For the Random Forest Classifier, the results greatly changed for the better. As shown in the next figure, the accuracy of the Random Forest Classifier is approximetly 91.23%. However, just to test the model again, I ran another K-cross validation which shows that the mean of all accuracies for the Random Forest Classifier is approximitly 90.73%.



```
In [12]: from sklearn.metrics import accuracy_score
    ...: print('Accuracy Score: ',accuracy_score(y_test, y_pred))
Accuracy Score:  0.9122889305816135

In [13]: from sklearn.model_selection import cross_val_score
    ...: accuracies = cross_val_score(estimator = classifier, X =
X_train, y = y_train, cv = 10)
    ...: print("Mean of accuracies for Random Forest " +
str(accuracies.mean())+ "\n")
Mean of accuracies for Random Forest 0.9073090882095028
```

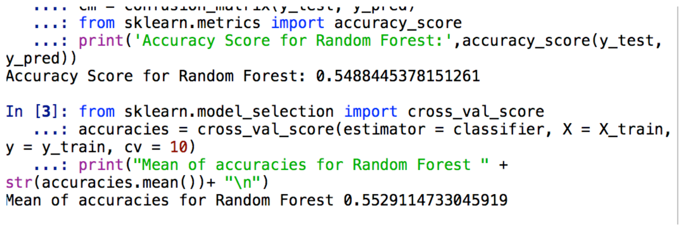
VII. Continuing On

In order to use the classifier, we built with other texts we decided to scrape propaganda websites such as American Renaissance and the Daily Stormer. After scraping the sites and cleaning the data by tokenizing it into sentences and then cleaning each sentence individually by the method stated earlier, we had about 787,929 sentences to work with.

We recalculated the co-occurrence matrix to account for the new data and then we trained part of the co-occurrence matrix on the previously labeled metaphor and literal data from the Virginia University dataset and the Wikipedia sentences. After recalculating and training the co-occurrence matrix on a Random Forest Classifier, we were then able to determine whether a sentence was a metaphor or not.

VIII. Classifying the Data by Catergories

The University of Virginia dataset already had the metaphors classified in 9 different categories which are: Government, Body, Population, Architecture, Mineral, Impressions, Writing, Light, and Animals. Therefore, we decided to create a Random Forest Classifier to classify our newly found metaphors into one of these categories. After cleaning the text and running the machine learning algorithm, we achieved an accuracy of 55.57% as well as a 10-fold accuracy of 55.29% as shown below.



```
   ...: cm = confusion_matrix(y_test, y_pred)
   ...: from sklearn.metrics import accuracy_score
   ...: print('Accuracy Score for Random Forest:',accuracy_score(y_test,
y_pred))
Accuracy Score for Random Forest: 0.5488445378151261

In [3]: from sklearn.model_selection import cross_val_score
   ...: accuracies = cross_val_score(estimator = classifier, X = X_train,
y = y_train, cv = 10)
   ...: print("Mean of accuracies for Random Forest " +
str(accuracies.mean())+ "\n")
Mean of accuracies for Random Forest 0.5529114733045919
```

Using the earlier method of recalculating the co-occurance matrix on the new text data, we were able to use this classifier to classify the metaphors based on the 9 different catergories.

IX. Conclusion Part 1

In this paper we have presented a novel way of classifying text into metaphorical and literal classifications. We have used the metaphor database from the University of Virginia as well as sentences from the simple Wikipedia pages to create our own dataset of literal sentences. We have cleaned up the data by removing the stop words in each sentence and then stemming each word to its root form. After that we have converted the text data into a vectored form by using the co-occurrence matrix to calculate the abstractness level of a given word compaerd to all other words in the dataset. This is then used as our input for our machine learning models which we chose three. The One Class SVM which worked with only metaphorical sentences and generated an accuracy of 84.4% classification. The Naïve Bayes Classifier which generated an average accuracy of 77.82% after cross validation. Finally, the Random Forest Classifier which generated an average accuracy of 90.73% after cross validation.

After identifying the metaphors from the classifier, we then used another Random Forest Classifier to classify the identified metaphors into their respected catergories. There were 9 catergories that we tested for and the accuracy for the 10-fold accuracy was 55.29%.

References

[1] Ben, Yosef & Last, Mark. (2015). MIL: Automatic Metaphor Identification by Statistical Learning. CEUR Workshop Proceedings. 1410.

[2] Brian Macwhinney and Davida Fromm. (2014). Two Approaches to Metaphor Detection. European Language Resources Association (ELRA).

[3] (n.d.). Retrieved from https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[4] (n.d.). Retrieved from https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html