Introduction

Binary classification of paraphrases is an important topic being researched in the Nature Language Processing space. As humans it is easy to determine if two sentences hold the same meaning, however teaching a machine to the same is a great challenge.

This topic is crucial especially in the application of virtual assistants. Virtual assistants deal with questions and instructions posed to it multiple times a day, all over the world. Understanding the meaning behind these prompts will help virtual assistants relay information better.

This report summarizes an investigation into the creation of a rudimentary paraphrase classifier. Two main methods were employed; a rule-based baseline approach, and a KNN classifier accompanied by Bag of Words. Each methodology was described and explained and the results were discussed. Future improvements were also stated.

Dataset

About the Dataset

The dataset that was used was *Paraphrase and Semantic Similarity in Twitter*. This was a dataset used in an international competition at SemEval 2015. This was a workshop on Semantic Evaluation.

The dataset has multiple examples of two sentence inputs. A panel of five judges examined these two inputs and determined if they are paraphrases of each other or not. The decision of the judges was also provided in terms of votes. For example if three judges voted saying the two inputs are paraphrases and the other two did not, then the provided decision was `(3,2)`.

The entire dataset was used for the assignment. The dataset curator did suggest that inputs that have a "debatable" decision like `(3,2)` or `(2,3)` can be removed from any training and testing so that there is no uncertainty present in the training data. However, I chose to keep it in so that there would be more data to train on.

Converting Votes to a Binary Class

The method used to convert graded evaluations, the decisions made by judges, into a binary classification was relatively simple. The binary classification was decided based on majority. If there was a greater number of positive votes in favor of "paraphrase" then the class "P" was assigned. The converse is also true and the class "NP" was assigned in this case.

The below code was used to assign the new class. The previous votes were also tracked.

```
 c = []
p_votes = []
n_votes = []
for row in train_data.iterrows():
    row = row[1]
    p_votes.append(row['Label'][1])
    n_votes.append(row['Label'][4])
    if (int(row['Label'][1]) > int(row['Label'][4])):
        c.append("P")
    else:
        c.append("NP")

train_data['Class'] = c
train_data['Positive votes'] = p_votes
train_data['Negative votes'] = n_votes
```

Here are some examples of paraphrases ("P") and non-paraphrases ("NP"):

	Sentence 1	Sentence 2	Class
0	EJ Manuel the 1st QB to go in this draft	But my bro from the 757 EJ Manuel is the 1st Q...	P
1	EJ Manuel the 1st QB to go in this draft	Can believe EJ Manuel went as the 1st QB in th...	P
2	EJ Manuel the 1st QB to go in this draft	EJ MANUEL IS THE 1ST QB what	P
4	EJ Manuel the 1st QB to go in this draft	Manuel is the 1st QB to get drafted	P
5	EJ Manuel the 1st QB to go in this draft	EJ da 1st QB off da board	NP

	Sentence 1	Sentence 2	Class
6	EJ Manuel is the 1st QB in the draft	1st QB of the board is	NP
7	EJ Manuel is the 1st QB in the draft	Bills take EJ Manuel QBFlorida State 1st QB of...	NP
8	EJ Manuel is the 1st QB in the draft	EJ Manuel 1st QB off the board taken by the Bills	NP
9	EJ Manuel is the 1st QB in the draft	Lol and 1st QB off the board	NP

Methods

To predict if two sentence inputs are paraphrases or not, three different methods were used. The first was designated as the baseline algorithm as it was relatively simple and did not use any classification algorithms. The second and third algorithm are similar in terms of the classification algorithm that was employed, however the features used to train the model were different.

Baseline Algorithm

The baseline approach is straightforward. Each input sentence is transformed into lowercase form, and then the sentence is split into an array of words. The array is then converted into a set, so duplicate words are removed. Then the number of common words between the two sentences is found. This number if multiplied by two, since it occurs in the first sentence set as well as the second sentence set. It is then divide by the total number of words of the two sentences. When divided, if the output is greater than or equal to 0.5, then the sentences are classified as paraphrases of each other. Otherwise they are classified as non-paraphrases.

Here is the code that does what is described above:

```
def baseline(input1, input2):
    pred = []
    for row in zip(input1, input2):
        words1 = set(row[0].lower().split())
        words2 = set(row[1].lower().split())
        same = words1.intersection(words2)
        score = 2*len(same)/(len(words1) + len(words2))
        if score >= 0.5:
            pred.append('P')
        else:
            pred.append('NP')
    return pred
```

An example of the baseline approach is as follows:

1. Sentence 1 = "There was a motorist crossing the intersection", Sentence 2 = "A car was crossing the intersection"
2. The sentences is transformed to lowercase and split up. Sentence 1 = ['there', 'was, 'a', 'motorist', 'crossing', 'the', 'intersection'], Sentence 2 = ['a', 'car', 'was', 'crossing', 'the', 'intersection']
3. The intersection of the two sentences becomes ['a', 'was', 'crossing', 'the', 'intersection']
4. The score is calculated as `2*(number of common words) / total number of words in two sentences = 2*5/13 = 0.77` .
5. Given the score, the two sentences would be classified as paraphrases as one another.

Bag of Words and KNN Classifier

Bag of Words is a method that is used on a corpus. It takes into account the frequency of each word occurring in sentence and generates a vector accordingly.

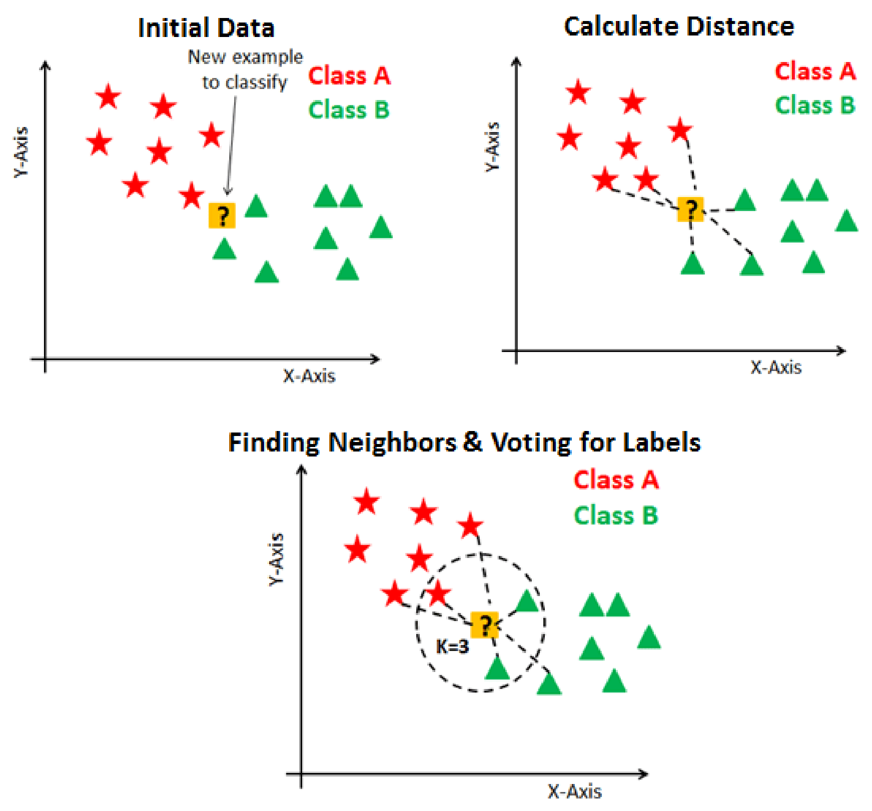
For example if the corpus was "Nice things don't happen in storybooks. Or when they do happen, something bad happens next. Because otherwise the story would be boring, and no one would read it." This blurb was taken from a book by Holly Black.

The vector transformation would look like this:

> [1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2]

Each number represents the frequency of a word like "nice" or "storybooks" occurring in the above corpus.

The K-Nearest Neighbours (KNN) classifier uses the nearest points of data (the neighbours) to classify a new data point. In the figure below, there is a new data point indicated by the "?". Since `K=3`, the classifier uses the three nearest neighbours. For our purposes we used the default value, `5`. From the three nearest neighbours, there are two class B points and one class A point. Here the classifier will classify the new data point as class B.



The reason the KNN Classifier was used as it is similar the vote based system used to generate the classes for paraphrase and non-paraphrase.

Bag of Words without Stop Words

The modification to the second algorithm is small but still significant. The same classifier was used, just the features used to train the classifier were slightly different.

In the previous Bag of Words approach, the frequency of every word in the sentence was taken into account. Words like "a", "to", etc, also known as stopwords, would naturally have higher frequencies and this could offset the classification model. To remedy this, the stopwords will need to be removed.

An example of the removal of stopwords for the above corpus is as follows:

> The corpus with no removal words: ['and','bad','be','because','boring','do','don','happen','happens','in','it','next','nice','no','one','or','otherwise','read','something','story','storybooks','the','they','things','when','would']
>
> The corpus with stopwords removed: ['bad','boring','don','happen','happens','nice','read','story','storybooks','things']

Results

Testing on the Dev Set

The results for the three separate algorithms is shown in the table below. The results are based of testing done on the Dev Set of the dataset. Surprisingly, the baseline algorithm performed better overall.

Algorithm	Precision	Recall	Accuracy
Baseline	0.6976	0.2966	0.7413
Bag of Words + KNN	0.3175	0.0544	0.6696
Bag of Words without stopwords + KNN	0.5183	0.0769	0.6907

There was some improvement when stopwords were removed from the Bag of Words vector, especially in regards to recall.

Testing on the Test Set

Since the best performing algorithm was the baseline algorithm, it was tested on the Test Set of the dataset.

The results are below:

Algorithm	Precision	Recall	Accuracy

Algorithm	Precision	Recall	Accuracy
Baseline	0.8220	0.3139	0.7603

■ Discuss the results, including a qualitative analysis (showing a few examples that the best method got right or wrong)

As expected, the baseline algorithm performed similarly on the Test Set as it did on the Dev Set. In fact, it performed a bit better but this is a coincidence as this is just a rule-based algorithm.

Here is an analysis of some examples of test cases for the baseline approach:

	Sentence 1	Sentence 2	Prediction	Actual	Analysis
1	The last rap battle in 8 mile though	But why were people watching the heat play when 8 mile is on	NP	NP	This is an example where the sentences are clearly different, and the only common words in the sentences are "8 mile"
2	Well at least 8 Mile is on	My favorite movie ever is 8 mile	NP	P	This an example where the sentences are different (non-paraphrases). The prediction is also indicative of this, however the correct result in this case should be paraphrase. In my opinion these sentences are non-paraphrases, however the judges felt they were paraphrases. This was most likely an instance where the sentences were debatable.
3	Ok good the end of 8 Mile is on	The end of 8 Mile makes me so happy	P	P	This example makes sense, both sentences are paraphrases. However, the baseline algorithm worked here because there are few words in the sentences. If there were more words then it would not work.
4	hahaha that sounds like me	That sounds totally reasonable to me	P	NP	Here we see that due to the low number of words, the baseline got the prediction wrong.

Conclusion

In a high level view, the baseline approach seems to work well on sentences that are a few words in length, since there is a higher chance of passing the 50% common word threshold.

I was also surprised that the KNN classifier did not work that well. One of the reasons which might have contributed to the lack of success could be that there were still too many features, and that might have led to less commonality between neighbors.

For future improvements, using part-of-speech tagging might be a great improvement. Also using unsupervised learning could be an alternative approach.

References

1. C. (n.d.). GitHub - cocoxu/SemEval-PIT2015: data and scripts for the shared task “Task 1: Paraphrase and Semantic Similarity in Twitter (PIT)” at SemEval 2015. GitHub. Retrieved February 26, 2022, from https://github.com/cocoxu/SemEval-PIT2015
2. Navlani, A. (2018, August 2). KNN Classification Tutorial using Sklearn Python. DataCamp Community. Retrieved February 26, 2022, from https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learnShah,
3. sklearn.neighbors.KNeighborsClassifier. (n.d.). Scikit-Learn. Retrieved February 26, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Source code:

1. H. (n.d.). ml-training/Building-a-Paraphrase-Classifier.ipynb at master · CoderHahs/ml-training. GitHub. Retrieved February 26, 2022, from https://github.com/CoderHahs/ml-training/blob/master/Knowledge/NLP/Notebooks/paraphrase_classifier/Building-a-Paraphrase-Classifier.ipynb