A study of simple paraphrase classifier

Zixun Xiang
zxian010@uottawa.ca

1. Introduction

The binary classification of paraphrases is a way to do paraphrase detection and will provide a binary result that shows whether two sentences are paraphrases or not. In this study case, I developed one baseline and two advanced binary classification algorithms to do the paraphrase detection on the dataset SemEval-PIT2015.
Github: https://github.com/FSZ1X2/CSI5180SimpleParaphraseClassifier

2. Dataset

2.1 Part of the dataset used. I used the Dev and the Test set from the SemEval-PIT2015 dataset. Dev set contains one data file called “dev.data” that has 4727 pairs of sentences with the label that suggests whether this pair is a paraphrase or not. The Test set contains two data files, one is “test.data” and one is “test.label”. The first file has 972 pairs of sentences and the second file provides the paraphrase classification label for that 972 pairs.

2.2 Prepare data. Inside the dev.data file, each pair of sentences is in a format like “Topic_Id | Topic_Name | Sent_1 | Sent_2 | Label | Sent_1_tag | Sent_2_tag |” and the label indicates the classification of paraphrases. As the "Label" column is in a format like "(1, 4)", I build a function called checkExpection to change the label from graded evaluations to binary.

```
checkExpection(Label_contents): # take in the contents from the "Label" column
    positive_votes = int(Label_contents[1]) # save positive votes
    negative_votes = int(Label_contents[4]) # save negative votes
    # if positive votes are more than the negative votes
    if positive_votes > negative_votes:
        expectation = 'Yes' # the paraphrase classification is Yes
    else: expectation = 'No' # the paraphrase classification is No
```

Inside the test.label file, each line provides the graded evaluation and the graded ratio for each pair of sentences. The label file showed that if the ratio is larger than 0.6, the pair of sentences is a paraphrase. To change this to binary, I have a function called defineResult.

```
defineResult(contents): # take in the graded ratio from each line
    graded_ratio = contents # save the graded ratio
    if graded_ratio > 0.6: # if the ratio is larger than 0.6
        expectation = 'Yes' # the paraphrase classification is Yes
    else: expectation = 'No' # the paraphrase classification is No
```

2.3 Examples of paraphrases and non-paraphrases. The table below shows some examples of paraphrase and non-paraphrase pairs after my data preparation:

Sent_1	Sent_2	Label/Ratio	Paraphrase
ZBo playing no games	Gasol and ZBo are bullies	(1, 4)	No
ZBo playing no games	Now ZBo is cranking up	(4, 1)	Yes
The Spurs own ZBo	ZBo is a fucking scrub	0.2	No

3. Methods

3.1 Baseline algorithm. My baseline algorithm is to simply check the similarity between the two sentences. I used the SequenceMatcher function from the difflib package in Python to do this check. This function will automatically compare pairs of strings and provide a number to illustrate their similarity. If the similarity is larger than the given threshold, the baseline algorithm will consider this pair of sentences as paraphrases.

```
similarity = SequenceMatcher(None, inputSent1,inputSent2).ratio()
if similarity > threshold: # in this algorithm threshold = 0.8
      Output: "Yes" # inputSent1 and inputSent2 are paraphrase
else: Output: "No" # inputSent1 and inputSent2 are non-paraphrase
# Example
inputSent1 = "ZBo playing no games"
inputSent2 = "Gasol and ZBo are bullies"
similarity = similar(Sent1,Sent2) # in this example similarity = 0.356
similarity < threshold: # in this example 0.356 < 0.8
Output: "No" # in this example, paraphrase: No
```

Note that different threshold will provide different results.

3.2 Method 1 algorithm. My method1 algorithm is to compare the average edit distance for each word with the respect to the POS of it. The first step of this algorithm is to set up the weight for each POS and calculate the threshold. Below is the pseudo-code for this step.

```
# set up weight for POS based on Corenlp POS tags:
CC = 1; CD = 1; DT = 2 … Others = 1
# calculate the threshold:
threshold = (CC + CD + … + Others)/37 # this algorithm has 37 different POS
```

The second step is to build a POS table for each sentence in a pair. The input tags for each sentence are in a format like ”ZBo/O/IN/B-PP/O playing/O/VBG/B-VP/B-EVENT...”

```
rawString = InputSentTag.split() # split the input sentence tags by space
# loop through the string and get POS/word pairs from it
for i in range(0, length(rawString)):
   WordwithPOS = rawString[i].split('/') # split the POS/word pairs with '/'
   POS_list.append(POS) # save POS to a list of POS
   word_list.append(word) # save word to a list of word
# build a POS table for this sentence
d = {'POS':POS_list,'Word':word_list}
POSTable = pd.DataFrame(d)
return POSTable # the POS table for this sentence
```

After that, the algorithm builds a comparison table that combined the two POS tables for each sentence and calculate their average edit distance with the respect to the POS weights. The edit distance function is from the paper Definiton of Minimum Edit Distance and it will provide the edit distance between two strings.

```
# comparison table is the union of two POS tables
ComparisonTable = Sent1POSTable ∪ Sent2POSTable
# add a new column to store the weight for POS
```

```
weight_list = a list shows weights for the POS in the ComparisonTable
ComparisonTable['Weight']: weight_list
# calculte average edit distance with the respect to POS
for i in range(0, length(ComparisonTable)): # for each POS
   distance += editDistance(word1, word2) * WeightForRelatedPOS
average_distance = distance/length(ComparisonTable)
```

Finally, the algorithm use the average distance to detect paraphrase or not.

```
if average_distance < threshold: Paraphrase = 'Yes' # this pair are paraphrase
else: Paraphrase = 'No' # this pair of sentences are non-paraphrase
```

Below is an example for this algorithm.

```
# sentence 1: ZBo playing no games
inputSent1TAGs = "ZBo/IN playing/VBG no/DT games/NNS"
# sentence 2: Gasol and ZBo are bullies
inputSent2TAGs = "Gasol/NNP and/CC ZBo/NNP are/VBP bullies/NNS"
BuildPOSTable(Sent1) and BuildPOSTable(Sent2)
BuildCompareTables(Sent1POSTable, Sent2POSTable)
Calculate average distance
average_distance > threshold: # in this example 11.0 > 2.51
Output: "No" # in this example, paraphrase: No
```

3.3 Method 2 algorithm. My method2 algorithm is based on the method1 algorithm and improved the way of how the comparison table is generated. In the method1 algorithm, the comparison table is simply the union of two POS tables. For some cases, if two words in one sentence have the same POS, they will be combined into one word in the comparison table and influence the edit distance calculation. To avoid this situation, I developed a function called POSCombine to generate the comparison table.

```
# loop through each POS table to find duplicate POS/word pair and save them
for i in range(0,length(Sent1POSTable)): Sent1duplicate.add(POS/Word pair)
for i in range(0,length(Sent2POSTable)): Sent2duplicate.add(POS/Word pair)
# comparison table is the union of two POS tables the duplicate pairs
ComparisonTable = Sent1POSTable∪Sent2POSTable+Sent1duplicate+Sent2duplicate
```

In this way, the comparison table can provide a better view of words in different POS. For the same example showed above, if we have `inputSent1TAGs = "ZBo/IN playing/VBG no/DT games/NNS"` and `inputSent2TAGs = "Gasol/NNP and/CC ZBo/NNP are/VBP bullies/NNS"`. After building the POS table for each sentence, method1 will provide a comparison table like this:

IN	VBG	DT	NNS	NNP	CC	VBP
Zbo	playing	no	games			
			bullies	GasolZBo	and	are

While the method2 will provide a comparison table like this:

IN	VBG	DT	NNS	NNP	NNP	CC	VBP
Zbo	playing	no	games				
			bullies	Gasol	ZBo	and	are

It’s clear that with the method2 algorithm, the edit distance can be calculated more precisely.

```
Calculate average distance
average_distance > threshold: # in this example 11.5 > 2.51
Output: "No" # in this example, paraphrase: No
```

4. Results

4.1 Comparison results of 3 methods on Dev set

	Baseline	Method 1	Method 2
Precision	0.808	0.809	0.812
Recall	0.508	0.508	0.590
F1	0.624	0.624	0.683

4.2 Results of the chosen method on the Test set

Chosen Method	Precision	Recall	F1
Method 2	0.554	0.499	0.526

4.3 Discussion. The results provided a good overview of how each of my paraphrase detection algorithms performed. For the Dev set, the method2 algorithm provided the best F1-score and this is what I expected since the method2 algorithm is an advanced version of the method1 algorithm. The baseline algorithm also performs better than I thought. I think this is because the function I used to check similarity is a good one. For the Test set, the method2 algorithm performed not as well as I thought. I think this is because the weights for POS tags depend on the Dev set instead of the Test set. Therefore, the calculated threshold for the Test set might be not suitable.

5. Conclusion

5.1 Conclusion about the experiment. From all the experiments and their results, it is clear that the algorithm with more respect to the POS performs better than other algorithms. In conclusion, POS can provide a way for the paraphrase classifier to do a more in-depth checking or comparing and this will increase the accuracy of paraphrase classifiers.

5.2 Future possibilities. As future work, I think method 2 can be improved by implementing and testing on different datasets. In addition, I think building a model for the binary paraphrase classifier is a good idea. There are many different tools, like Word2Vec or Doc2Vec, that can help build the model and work for binary classification of paraphrases. It will be a great idea to explore those tools and build a better paraphrase classifier.

6. References

6.1 Use for my programming

difflib package: https://docs.python.org/3/library/difflib.html
edit distance source: https://web.stanford.edu/class/cs124/lec/med.pdf
POS tags: https://corenlp.run/
POS: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

6.2 Reference for the dataset

Dataset: https://github.com/cocoxu/SemEval-PIT2015
Dataset description: https://www.aclweb.org/anthology/S15-2001.pdf