BINARY PARAPHRASES CLASSIFIERS

Jiacheng Hou
jhou013@uottawa.ca

1 Introduction

Paraphrases are two or more sentences with the same meaning but different expression methodologies. Paraphrasing is an essential field in Natural Language Processing (NLP), and it is widely used in paraphrases generation, paraphrases detection, similarity detection and Q&A. In this report, the author will explore three algorithms to detect whether two sentences are paraphrases or not. The author utilizes the dataset SemEval-2015 [1] to train and test various algorithms. For simplicity, the author assumes paraphrasing detection is a binary classification problem, where labels are "1" and "0". Label 1 means the two sentences are paraphrases; otherwise, they are not.

The rest of the report is organized as follows: Section 2 presents the dataset. Section 3 introduces three binary paraphrases classification algorithms. Section 4 shows the evaluations, and Section 5 concludes the report.

2 Dataset

The SemEval-2015 [1] dataset contains 13063 training data, 4727 validation data and 972 testing data. Each data contains a topic id, a topic name, a pair of sentences, a label, part-of-speech and named-entity tags of two sentences. For the training and validation dataset, labels are a tuple with two values, and the first is the number of votes that agree the two sentences are paraphrased. In contrast, the second is the number of votes of two sentences that do not paraphrase. All possible labels are: (0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0). Differently, the testing dataset has labels 0 to 5.

The author utilizes 2000 training data, 200 validation data, and 200 testing data in this report. It includes the first 2000 samples and 200 samples that labels are not (2, 3) in the file "train.data" and "dev.data", respectively. In addition, it contains the first 200 data samples that do not have label 3 in the file "test.data."

In order to transform the dataset labels into a binary label, the author considers labels (3, 2), (4, 1), (5, 0) as paraphrases and converts them to the binary label "1" for the training and validation dataset. In addition, the author treats labels (1, 4), (0, 5) as non-paraphrases, and the binary label is "0". Besides, the dataset does not include any data with labels (2, 3) because it is debatable. For testing data, the author considers labels 4 or 5 as paraphrases with binary labels "1", labels 0 or 1 or 2 as non-paraphrases, and their binary label is "0". The author also discards the data where the label is 3. Table 1 shows two examples of paraphrases and two examples of non-paraphrases in the training dataset.

It is worth noting that both training, validation and testing dataset are imbalanced. In the training dataset, there are 1309 samples with label 0 and 691 samples with label 1. In the validation dataset, 118 samples are in class 0, while 82 samples are in class 1. Class 0 has 175 samples in the testing dataset, but class 1 has only 25 samples.

Table 1: Paraphrases and non-paraphrases examples in the training dataset

Paraphrases	Sentence1	Sentence2
YES	EJ Manuel the 1st QB to go in this draft	My boy EJ Manuel being the 1st QB picked
YES	That amber alert was getting annoying	Ive gotten the same amber alert 3 times
NO	EJ Manuel is the 1st QB in the draft	1st QB of the board is
NO	What is an amber alert	OMG AMBER I FREAKING HATE YOU

3 Methodologies

In this section, the author will introduce three classifiers utilized in the report and show an example for each of them.

3.1 Baseline algorithm

The first algorithm is pretty simple, where the classifier calculates the percentage of the same characters among two sentences and compares the ratio with a threshold. The algorithm contains the following steps:

- Align the two sentences from the very left character (i.e. the beginning character of the two sentences).
- Count c: the number of the same characters between two sentences.
- Calculate the percentage $p = c/l$, where l is the number of characters in the shorter sentence.
- Define a threshold $\alpha = 0.7$, if $p \geq \alpha$, they are paraphrases. Otherwise, they are not.

For examples, two sentences (1): "I got it from amazon" and (2): "According to Amazon s review". Table 2 shows characters alignment of the two sentences. From the table, we can see that three characters (bold) are the same and thus $c = 3$. Sentence (1) has less number of characters, so $l = 20$. Therefore, $p = c/l = 3/20 = 0.15 < \alpha$. The algorithm classifies the two sentences as non-paraphrases.

Table 2: Sentences (1) and (2) characters alignment in the Baseline algorithm

(1)	I		g	**o**	t		**i**	t		f	r	**o**	m		a	m	a	z	o	n					
(2)	A	c	c	**o**	r	d	**i**	n	g		t	**o**		A	m	a	z	o	n	s	r	e	v	i	e	w

3.2 TF-IDF Vectors and Cosine Similarity

The second algorithm is to transform each pair of sentences into Term Frequency-Inverse Document Frequency (TF-IDF) [2] vectors and then compute the cosine similarity. The algorithm contains the following steps:

- The first term, "TF," calculates a specific word frequency in each sentence. It is mathematically formulated as $TF(t, s)$, where t is a unique term, and s is a specific sentence.
- The second term, "IDF," computes the logarithm of the total number of sentences divided by the number of sentences that contain a specific term t, formulated as $IDF(t)$.
- The TF-IDF value for each term t and sentence s is formulated as $TF - IDF(t, s) = TF(t, s) * IDF(t)$. Also, the TF-IDF vector for a sentence is a vector where each element is a TF-IDF value of a specific term.
- The cosine similarity of the two vectors achieved from the previous step is calculated. If the cosine similarity of the two sentences is higher than a threshold $\beta = 0.7$, they are paraphrases. Otherwise, they are not.

In this report, the author uses the scikit-learn package [2] to compute the TF-IDF vector for each sentence. For example, two sentences (1): "I got it from amazon" and (2): "According to Amazon s review." Table 3.2 shows the TF-IDF vectors for the two sentences, where sentence (1) is $[0, 0.58, 0.81, 0]$, and sentence (2) is $[0.63, 0.45, 0, 0.63]$. It is worth

Table 3: TF-IDF values of each term in sentences (1) and (2)

Sentence \ Word	according	amazon	got	review
(1)	0	0.58	0.81	0
(2)	0.63	0.45	0	0.63

noting that the terms like "I," "it," and "to" are ignored because they are stopping words in English and provide little information. Following this, the cosine similarity of the two vectors is computed, which is $0.26 < \beta$. Therefore, the two sentences are non-paraphrases.

3.3 Word2Vec Model and KNN

In this report, the author utilizes a pre-trained Word2Vec model 'glove-twitter-25', downloaded from GENSIM API [3]. The Word2Vec model is used to embed each word into a vector. Then the cosine similarity of two sets of word vectors is computed and fed into a K-Nearest Neighbours (KNN) [4] classifier. The algorithm includes the following steps:

- For each sentence, convert all words to lower cases and then extract a set of available words in the pre-trained model 'glove-twitter-25'. Compute the cosine similarity [5] between the two sets of words.
- Train the KNN using the training dataset, where the input is the cosine similarity computed from the previous step, and the output is a binary label 0 or 1.
- In the inference stage, the cosine similarity of each pair of sentences is fed into the KNN. A label 0 or 1 is predicted.

For example, two sentences (1): "I got it from amazon" and (2): "According to Amazon s review." It is worth noting that the Word2Vec model is case-sensitive. In order to extract both "Amazon" and "amazon," the author decides first to convert all words into lower cases. After converting all words into lower cases, two sets of words {'i', 'got', 'it', 'from", "amazon"} and {'according', 'to', 'amazon', "s", "review"} are extracted, respectively. And then, the cosine similarity of the two sets of words is 0.91. Finally, 0.91 is fed into the KNN to make a prediction.

4 Evaluations

Table 4 shows the Baseline, TF-IDF and the KNN algorithms performances on validation dataset. The performances metrics are True Positive (TP), False Positive (FP), False Negative (FN), Precision (Prec.), Recall (Rec.), F1 score (F1), a micro average of Prec., a micro average of Rec., a micro average of F1, a macro average of Prec., a macro average of Rec. and a macro average of F1. Each metric is mathmatically formulated as the following:

$$Prec. = \frac{TP}{TP + FP} \tag{1}$$

$$Rec. = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = 2 * \frac{Prec. \times Rec.}{Prec. + Rec.} = \frac{TP}{TP + \frac{1}{2} \times (FP + FN)} \tag{3}$$

$$Micro\ Prec., Micro\ Rec., Micro\ F1 = \frac{TP\_Class0 + TP\_Class1}{N} = Accuracy \tag{4}$$

where TP_Class0 is the TP for class 0 and TP_Class1 is the TP for class 1. N is the total number of samples in the dataset, and $Accuracy$ is the percentage of correctly classified data samples.

$$Macro\ Prec. = \frac{Prec\_Class0 + Prec\_Class1}{2} \tag{5}$$

where $Prec\_Class0$ is the Prec. for class 0 and $Prec\_Class1$ is the Prec. for class 1.

$$Macro\ Rec. = \frac{Rec\_Class0 + Rec\_Class1}{2} \tag{6}$$

where Rec_Class0 is the Rec. for class 0 and Rec_Class1 is the Rec. for class 1.

$$Macro\ F1. = \frac{F1\_Class0 + F1\_Class1}{2} \tag{7}$$

where $F1\_Class0$ is the F1 for class 0 and $F1\_Class1$ is the F1 for class 1.

From the table 4, it is obvious that the Baseline algorithm does a decent job regarding to Pre_Class1 and Rec_Class0. The reason is that the Baseline algorithm applies a rigorous rule of determining the two sentences are paraphrases. Therefore, the algorithm preferences do not classify two sentences as paraphrases. The low Rec_Class1 demonstrates the bias because the algorithm classifies most sentences non-paraphrases even though they are. It results in many FNs when label 1 is a positive class.

Table 4: Baseline, TF-IDF and KNN algorithms evaluation results on the validation dataset

Algorithms	Labels	TP	FP	FN	Prec.	Rec.	F1	Micro avg			Macro avg		
								Prec.	Rec.	F1	Prec.	Rec.	F1
Baseline	0	118	81	0	0.593	1	0.744	0.595	0.595	0.595	0.796	0.506	0.384
	1	1	0	81	1	0.012	0.024						
TF-IDF	0	117	79	1	0.597	0.992	0.745	0.6	0.6	0.6	0.673	0.514	0.407
	1	3	1	79	0.75	0.037	0.07						
KNN	0	106	64	12	0.624	0.898	0.736	0.62	0.62	0.62	0.612	0.559	0.529
	1	18	12	64	0.6	0.22	0.321						

Besides, the validation dataset is a little imbalanced, with more data samples labelled 0. The Baseline algorithm's bias toward classifying two sentences as non-paraphrased, together with the unbalanced data set, allows the baseline algorithm to achieve unimpressive $Accuracy$. However, this does not mean the Baseline algorithm performs a good job classifying paraphrases.

On the other hand, the macro average of Prec., Rec., and F1 treat each class equally rather than each sample. The differences between these three metrics are Prec. measures the error caused by FPs, whereas Rec. measures the extent of errors caused by FNs. F1 score evaluates both FP and FN and treats them in a balance. The author supposes FP and FN are equally undesirable in this problem. Therefore, the author prefers the macro F1 score as the final evaluation metric.

The TF-IDF algorithm achieves a better job distinguishing data samples in class 0 and 1 than the Baseline algorithm, reaching a higher macro F1 score. However, it still has a bias toward classifying two sentences as non-paraphrases because it focuses on word counting without considering word order and word meanings. Nevertheless, the word orders and meanings play an essential role in determining two sentences are paraphrases or not.

The KNN algorithm performs much better than the other two algorithms. For example, the TP is 34 when label 1 is a positive class. Even though it does not perform as well as the other two algorithms regarding $Prec\_Class0$, Rec_Class0 and $F1\_Class0$, it does not have much bias as the previous two algorithms. The input feature of the KNN algorithm is the cosine similarity of the sentence vectors achieved from the Word2Vec model, which can capture word meanings. It helps the KNN algorithm achieve a much higher macro F1 score than the other algorithms.

Therefore, the author considers the KNN algorithm as the best algorithm and shows the evaluation results on the testing dataset, referring to Table 5. It is worth noting that the testing dataset is extremely imbalanced, where the data samples of label 0 are seven times more than the data samples of label 1. The KNN algorithm achieves a macro F1 of 0.598 on this dataset. Table 6 shows four examples of KNN evaluation results on the testing dataset. Honestly, the fourth

sentence pair seems confused to the author. The author does not think the two sentences are paraphrases even though the dataset indicates.

Table 5: KNN evaluation results on testing dataset

Algorithm	Labels	TP	FP	FN	Prec.	Rec.	F1	Micro avg			Macro avg		
								Pre.	Rec.	F1	Pre.	Rec.	F1
KNN	0	168	20	7	0.894	0.96	0.926	0.865	0.865	0.865	0.655	0.58	0.598
	1	5	7	20	0.417	0.2	0.27						

Table 6: KNN evaluation examples on testing dataset

Actual Label	Predicted Label	Sentence1	Sentence2
1	1	Ok good the end of 8 Mile is on	The end of 8 Mile makes me so happy
0	1	Well at least 8 Mile is on	8 mile has been that movie
0	0	8 mile is on friday made	I missed the best part of 8 mile
1	0	Orr with a big hit on Chara	I keep waiting for the chara vs orr fight

5 Conclusion

In conclusion, the author introduced three classifiers to recognize two sentences are paraphrases or not. The author tested the three algorithms using the validation dataset, and the experimental results show that applying the Word2Vec model with the KNN classifier achieves the best performance. The author also explored the best classifier's performance on the testing dataset.

Paraphrasing detection is indeed a challenging problem, and even the authors have difficulty classifying it. In the future, the author plans to apply more training data samples to train the KNN classifier. One of the reasons that the KNN classifier does not perform satisfactorily is probably that the model is trained only on 2000 data samples, which is not enough for the model to learn the data distribution. Besides, the author is interested in exploring the Doc2Vec models [6] to predict the sentence similarity. Unlike the Word2Vec models, the Doc2Vec model generates an embedding of the sentence rather than a feature vector for each word. In addition, the author is interested in using deep learning-based models to generate sentences similarity. For example, the Sentence-BERT [7] model, which utilizes Siamese Bert-Network to generate sentence embeddings.

References

[1] Semeval-2015. `https://github.com/cocoxu/SemEval-PIT2015/`. Accessed: 2022-01-22.

[2] Term frequency-inverse document frequency. `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`. Accessed: 2022-01-22.

[3] Gensim. `https://radimrehurek.com/gensim/auto_examples/howtos/run_downloader_api.html`. Accessed: 2022-01-22.

[4] K-nearest neighbours classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`. Accessed: 2022-01-22.

[5] Word2vec cosine similarity. `https://tedboy.github.io/nlps/generated/generated/gensim.models.Word2Vec.n_similarity.html`. Accessed: 2022-01-22.

[6] Doc2vec. `https://radimrehurek.com/gensim/models/doc2vec.html`. Accessed: 2022-01-22.

[7] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.