PARAPHRASE DETECTION

Akanksha Malhotra akanksha.malhotra@colorado.edu
Amit Baran Roy amit.roy@colorado.edu
Aparajita Singh aparajita.singh@colorado.edu
Tanmai Gajula tanmai.gajula@colorado.edu

1 Introduction

The goal of the project is to detect whether two given sentences are paraphrase of each other or not. Two sentences are paraphrase of each other if their wordings are different, but they have the same meaning. There are generally three techniques to generate paraphrase: changing words, changing word forms, changing word order. Most of the paraphrase detection techniques use paraphrase corpora obtained using the first two techniques. In this project, we will focus on employing various machine learning techniques for detecting paraphrases generated from all three methods and find the merits and demerits of each algorithm. The aim is to suggest improvements through feature engineering in non-neural network machine learning algorithms and architectural improvements in neural network algorithms.

2 Motivation

The problem of paraphrase detection is fascinating as it has applications in many areas of natural language processing (NLP) like evaluation of extractive text summarizer, evaluation of machine translation systems, plagiarism detection, authorship authentication, question answering. The motivation behind choosing this project is two-fold. First, we present a comprehensive study of paraphrase detection problem. Second, by solving this problem, we can contribute to various existing NLP problems.

3 Dataset

We have used the Quora Question Pairs dataset. It consists of a training set of 404,290 question pairs, and a test set of 2,345,795 question pairs. Of the 404,290 question pairs, 255,027 (63.08%) have a negative label and 149,263 (36.92%) have a positive label.

3.1 Data Preprocessing

The quora dataset is human generated data which contains lots of anomalies. Before feature engineering, an important step is to do data preprocessing to reduce the anomalies and transform the data to useful and efficient format. The following are some of the pre-processing steps that we followed for non neural models:-

1. Unicode Normalization using NFKD- Characters are decomposed by compatibility, and multiple combining characters are arranged in a specific order.
2. Remove and replace Non-ASCII characters.
3. Remove punctuations and stopwords.
4. Word tokenization using NLTK.
5. Change all characters to lower case.
6. Expand contractions.
7. Change numeric data to word equivalent.

4 Feature Engineering

We have used a feature set of 39 features and we created features for a subset of training data consisting of 50000 examples. Features were created for 10000 examples of dev and 10000 examples of test. Our features can be grouped under 2 types as follows.

1. NLP based features:
- number of words, number of unique words, number of sentences, length of sentence with space, length of sentence without space for each pair of questions.
- difference of number of words, difference of number of unique words, difference of length of sentence with space, difference of length of sentence without space between a pair of questions.
- Bag of Words: Cosine Similarity between the term frequency of the two questions
- Bag of N-grams: Cosine similarity between the n-gram (unigram, bigram, trigram)
- Named Entity Feature: A named entity is a real world object such as locations, persons, companies etc. By extracting this feature, we can differentiate between two different questions such as "How do I invest in stock market?" and "How do I invest in stock market in the US?".
- Jacard distance, spacy similarity, word match share, first word similarity, idf, tfidf based features (ratio, sum, min, max, range)

2. Embeddings based features:
- Glove and Google's word2vec embeddings: ; it is a method to obtain vector representations for words by mapping words into a space where the distance between words is related to semantic similarity
- sense2vec embeddings: it is a technique that model words by using multiple vectors that are clustered based on context and multiple meanings or 'senses' of a single word.

We also analysed the importance of each feature by training the model on individual features and found that one of the most important feature was the weighted word match share. Another important feature was the first word similarity. Other important features were sense2vec and word2vec embeddings and these features together contributed significantly towards the overall accuracy.

5 Models and Hyperparameter Tuning

We trained the following models on 50000 training data having 39 features- Naive Bayes, Logistic Regression, Support Vector Machines, Decision Tree, Random Forest, Adaptive Boosting and Gradient Boosting. We used Scikit-Learn library to implement all the models.

5.1 Naive Bayes

A Naive Bayes classifier is a probabilistic classifier and is used as a baseline method for text categorization. In Gaussian Naive Bayes, the tuning of the parameter *var_smoothing* resulted

in good results. In Bernoulli Naive Bayes, the tuning of parameter *alpha* did not lead to any significant changes.

Model	Hyperparameter	Accuracy	F1 Score
Naive Bayes(Gaussian)	Var_Smoothing = 1e-09	46.65%	0.6343
Naive Bayes(Gaussian)	Var_Smoothing = 0.00001	63.92%	0.6938

Table 1: Naive Bayes Hyperparameters

5.2 Logistic Regression

Logistic regression is a statistical method used when the dependent variable (target) is categorical. In our case, the outcome we wanted was for the model to predict whether a pair of sentences is paraphrase or not. Higher values of the parameter C resulted in overfitting. Keeping the parameter C in the order of 0.00x resulted in the best accuracies on test data.

Hyperparameter	Values
C	**0.0021**, 0.001, 1, 2
Solver	newton-cg, lbfgs, **liblinear**, sag, saga

Table 2: Logistic Regression Hyperparameters

5.3 Support Vector Machines

Support Vector Machine (SVM) is a classification algorithm which can perform both linear and non-linear classification (by use of kernels). After finding the best kernel, we tuned the parameters that gave best results for that kernel. Increasing the value of C beyond 6 started to overfit the training data. The higher the *gamma* value, the more the algorithm tries to exactly fit the training data.

Hyperparameter	Values
max_iter	-1
kernel	linear, **rbf**, poly
C	1, 3, **6**, 7
gamma	**0.0001**, 0.001, 0.01, 0.1

Table 3: SVM Hyperparameters

5.4 Decision Tree

Decision trees are a non-parametric supervised learning method for classification and regression. Changing the values of the parameter *max_samples_leaf* had very little affect on the accuracy of the model. The higher the value of *max_depth* parameter, the more the model overfits.

Hyperparameter	Values
criterion	gini, **entropy**
splitter	random, **best**
max_features	no_of_features, sqrt, **log2**

Table 4: Decision Tree Hyperparameters

5.5 Random Forest

Random Forests are an ensemble learning method for classification and regression. When trees are made in this algorithm, each tree can only pick from a random subset of features. This forces variation amongst the trees and leads to creation of an uncorrelated forest of trees. The more the number of trees, the better is the learning. However, a large number of trees can lead to a decrease in training accuracy and an increase in training time. The parameter *min_samples_leaf* has the effect of 'smoothing' in the model.

Hyperparameter	Values
n_estimators	200, 250, **300**, 350, 400
criterion	entropy, **gini**
max_depth	1, 5, 10, 15 **None**
min_samples_split	0.1, 0.3, 0.5, **2**
min_samples_leaf	1, 3, **5**, 7
max_features	no_of_features, sqrt, **log2**

Table 5: Random Forest Hyperparameters

5.6 Adaptive Boosting

Adaptive boosting uses an ensemble of weak learners that are decision stumps. This algorithm is a sequential algorithm which creates a better model by paying attention to misclassified points of the previous model. The parameter *algorithm* had same results for both of its values *algorithm = 'SAMME'* or *algorithm = 'SAMME.R'* .

Hyperparameter	Values
n_estimators	500, 750, 950, **1000**, 1200
learning_rate	0.1, 0.001, **0.0001**, 0.00001

Table 6: Adaptive Boosting Hyperparameters

5.7 Gradient Boosting

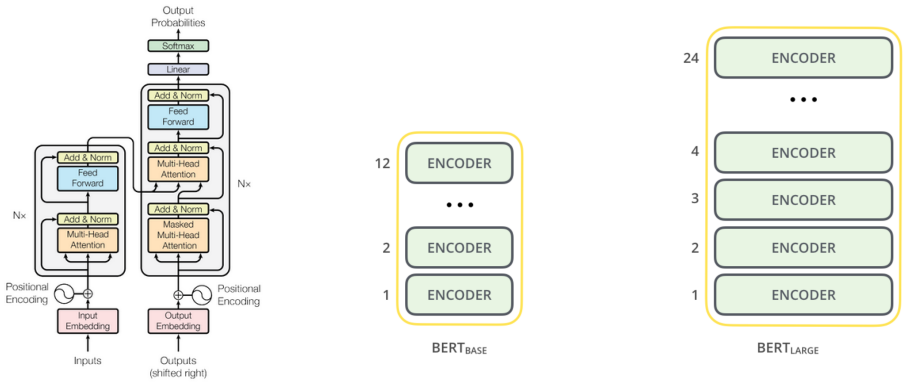
Gradient boosting is similar to Adaptive Boosting. The difference lies in what the algorithm does with the misclassified data points when learning. In Adaptive Boosting, the misclassified points are focused upon in next iteration by giving higher weight to them. In Gradient Boosting, the misclassified points are focused upon by using gradients.
The higher the value of learning rate, the more the model overfits. An increase *n_estimators* can lead to overfitting.

Hyperparameter	Values
loss	exponential, **deviance**
learning_rate	0.001, **0.1**, 0.2 , 0.5, 1
n_estimators	100, 200, **250**, 300, 350, 400
min_samples_split	2
min_impurity_split	1e-7
max_depth	2, 5, 7, **9**, 11
max_features	no_of_features, sqrt, **log2**

Table 7: Gradient Boosting Hyperparameters

6 Neural Network Models

6.1 BERT: Bidirectional Encoder Representations from Transformers



(a) Transformer Architecture (b) BERT Model Architecture

Figure 1: BERT

6.1.1 Model Overview

BERT or Bidirectional Encoder Representations from Transformers, is an unsupervised, deeply bidirectional system of pre-training language representations, which obtains state-of-the-art results on a wide array of NLP tasks. Pre-training language representations means training a general-purpose "language understanding" model on a large text corpus, that is then used for downstream NLP tasks i.e. fine tuning the pre-trained model for the specific task, which in our case is paraphrase detection.

6.1.2 Model Architecture

The architecture is based on a multi-layer bidirectional Transformer encoder. Currently, there are 2 variants of pre-trained models:

1. BERT Base: 12 layers (transformer blocks), 12 attention heads, 768 Hidden layer size and 110 million parameters.

2. BERT Large: 24 layers (transformer blocks), 16 attention heads, 1024 Hidden layer size and 340 million parameters.

6.1.3 Model Input

BERT receives input consisting of a sequence of tokens which may include a single sentence or a sentence pair. The input given to BERT is a combination of three types of embeddings:

1. Position embeddings: BERT learns and uses positional embeddings to express the positions of words in a sentence to capture "sequence" or "order" information.

2. Segment Embeddings: These are included to learn unique embeddings for each of the sentences to distinguish between the two. For this reason, special tokens - [CLS] (to mark the beginning of the sequence), [SEP] (to mark the end of each sentence) are used and the input sequence is in the format: [CLS] Sentence A [SEP] Sentence B [SEP].

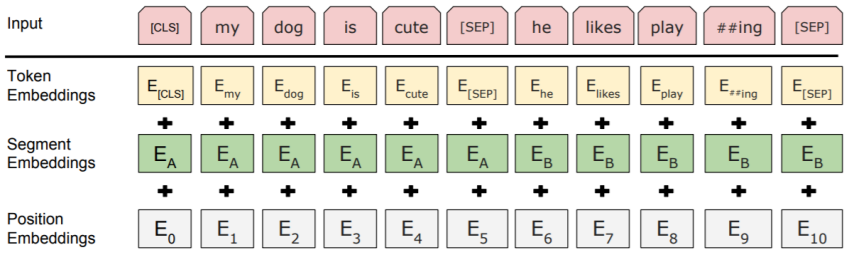


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Figure 2: BERT Model Input

3. Token embeddings: These embeddings are learnt for specific tokens from the wordpiece token vocabulary.

6.1.4 Model Pre-Training

BERT is pre-trained using 2 unsupervised tasks:

1. Masked Language modelling: In order to train a deep bidirectional representation, 15% of the input tokens are masked at random, and then the model is trained to predict those masked tokens. As the [MASK] token would never appear during fine tuning, the following strategy is adopted:
   (a) 80% of the time, the masked tokens are replaced with [MASK] token.
   (b) 10% of the time, the masked tokens are replaced with a random token.
   (c) 10% of the time, the masked tokens are left unchanged.
2. Next Sentence Prediction: BERT is also trained on the task of Next sentence prediction in order to understand relationships between sentences for specific NLP tasks. This is a binary classification task where the sentence pairs can be easily generated from any monolingual corpus, where 50% of the time, the second sentence would be the next sentence of the first sentence.

6.1.5 Model Fine-Tuning

The task specific inputs/outputs are plugged in to BERT model and all the pre-trained parameters are fine tuned from end to end. The only difference in architecture when compared to pre-training would be addition of an output layer (linear layer with softmax activation function), to which the [CLS] representation is given as input, for classification task.

6.1.6 Implementation and Experiments

We added a single fully connected layer to the output of the BERT model with a softmax layer to give us probabilities for class labels so that we can check the current examples are paraphrase of each other or not.

Train Batch Size	Eval Batch Size	Predict Batch Size	Max Seq Len	No of Train Epochs	Learning Rate
32	8	8	200	3	2e-5

6.2 BiMPM: Bilateral Multi-Perspective Matching

6.2.1 Model Overview

The paraphrase detection task can formally be redefined as estimating the conditional probability, $Pr(y|P,Q)$ based on training set and predicting the relationship for testing examples by $y_* = \text{argmax}_{x \in \mathcal{X}} Pr(y|P,Q)$, where P and Q are each of the sentences in the sentence pair example and y = 0, 1 where label 1 signifies that the sentences are paraphrases of each other and label 0, otherwise.

BiLateral Multi-Perspective Matching Model is a matching aggregation framework which matches 2 sentences P and Q in 2 directions (P $->$ Q and Q $->$ P) and this matching is done using multiple perspectives. The model estimates this probability distribution $Pr(y|P,Q)$, for a given pair of sentences, P and Q.

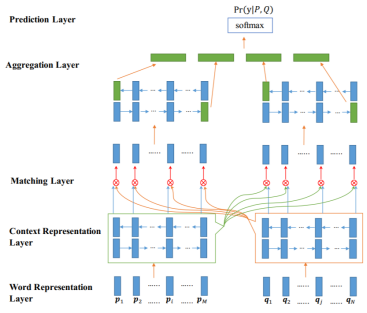


Figure 1: Architecture for Bilateral Multi-Perspective Matching (BiMPM) Model, where ⊗ is the multi-perspective matching operation described in sub-section 3.2.

(a) BiMPM Model Architecture

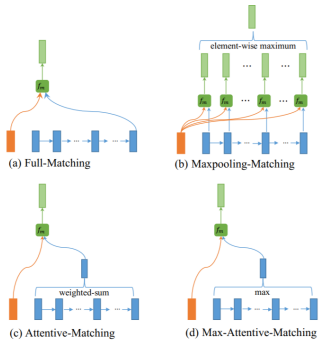


Figure 2: Diagrams for different matching strategies

(b) Multi-perspective matching strategies

Figure 3: BiMPM

6.2.2 Model Architecture

The architecture of BiMPM consists of 5 layers, which are described as follows:

1. Word Representation Layer:
   Each word in P and Q is represented as a combination of word embedding and character embedding. The word embeddings are pre-trained Glove or word2vec vectors. Character embeddings are learnt through a Long Short Term Memory network (LSTM).

2. Context Representation Layer:
   Here, a BiLSTM is applied to P as well as Q to encode contextual embeddings for each time step of P and Q.

3. Matching Layer:
   This Layer compares each contextual embedding or time-step of one sentence against all contextual embeddings or time steps of the other sentence. A multi-perspective matching operation is defined which uses 4 matching strategies listed as follows:

   (a) Full Matching: Each forward or backward contextual embedding is compared with the last time step of the forward or backward representation of the other sentence.

(b) Maxpooling-Matching: Same as full matching except only the maximum value of each dimension is retained.

(c) Attentive Matching: In attentive matching we first find attention weight through cosine similarity between both sentences context embedding, details in the equation. Then, we calculate weighted attention vector for an attentive vector for the entire sentence Q by weighted summing all the contextual embeddings of Q:

$$\overrightarrow{\alpha}_{i,j} = cosine(\overrightarrow{\boldsymbol{h}}_i^p, \overrightarrow{\boldsymbol{h}}_j^q) \qquad j = 1, ..., N$$
$$\overleftarrow{\alpha}_{i,j} = cosine(\overleftarrow{\boldsymbol{h}}_i^p, \overleftarrow{\boldsymbol{h}}_j^q) \qquad j = 1, ..., N$$

(a) Attention Calculation

$$\overrightarrow{\boldsymbol{h}}_i^{mean} = \frac{\sum_{j=1}^{N} \overrightarrow{\alpha}_{i,j} \cdot \overrightarrow{\boldsymbol{h}}_j^q}{\sum_{j=1}^{N} \overrightarrow{\alpha}_{i,j}}$$
$$\overleftarrow{\boldsymbol{h}}_i^{mean} = \frac{\sum_{j=1}^{N} \overleftarrow{\alpha}_{i,j} \cdot \overleftarrow{\boldsymbol{h}}_j^q}{\sum_{j=1}^{N} \overleftarrow{\alpha}_{i,j}}$$

(b) Weighted Attention Calculation

Figure 4: Attentive Matching

(d) Max-Attentive-Matching: This is the same as Attentive matching except for the formulation of attentive vector where contextual embedding with highest cosine similarity is taken instead of weighted sum of all contextual embeddings.

4. Aggregation Layer: Here, a BiLSTM model is used on each of the 2 sequences of matching vectors from previous step, individually. The output would be a fixed length matching vector obtained by concatenating the last time step of the model.

5. Prediction Layer: The output of the previous layer is fed as input to a two layer feed forward neural network with softmax as the activation function. The number of nodes in the output layer is a task specific parameter.

6.2.3 Implementation and Experiments

The final hyperparameters for BiMPM are given in the following table:

Batch Size	Character Dim	Character Hidden Size	Dropout	Word Embedding Dim	Learning Rate
16	20	50	0.1	300	0.001

We extended the model to replace word embedding with sense embedding. For this we tried two approaches:

1. Sense2Vec: We extracted sense2vec vectors for each word for a sentence. We generated the file with index of words in the vocabulary followed by the sense2vec vectors and transformed the data by replacing words with their ids. For character embedding, we used normal data. We extended the mode; by replacing word embedding with sense embedding. We implemented the model for only 150000 models and got 68% accuracy.

Batch Size	Character Dim	Character Hidden Size	Dropout	Word Embedding Dim	Learning Rate
16	20	50	0.2	300	0.001

2. BERT Embedding: We extracted the BERT embedding for each wordpieces by summing the output of last four encoder layers. As the hidden size of BERT is 768 and training data set was huge, so the embedding file created became huge and was not getting loaded into the memory.

7 Results

The results for generative, linear and tree based models are tabulated in the following table:

Model	Accuracy	F1 Score	Accuracy Reported in Paper[1]
Naive Bayes(Bernoulli)	57	0.6682	-
Naive Bayes(Gaussian)	63.92	0.6938	-
Logistic Regression	72.1	0.7267	80.1%
Support Vector Machines	**73**	**0.73**	80.9%
Decision Tree	70.66	0.6986	73.2%
Random Forest	71.9	0.6828	75.7%
Adaptive Boosting	69.38	0.7242	-
Gradient Boosting	71.79	0.6761	75.0%

Table 8: Accuracy and F1 score of various models

The results for neural network models are tabulated as follows:

Model	Dev Accuracy	Test Accuracy	Accuracy reported in paper [2][5]
BiMPM	87.3	87.3	88.17
BERT	90.09	-	71.2

Table 9: Accuracy and F1 score of various models

8 Model Analysis, Error Analysis and Conclusion

1. In the non neural models, SVM reportedly gave higher accuracy. In comparison to tree models, the linear models performed better. The results of the non neural models follows a similar trend as the results in paper [1] but the former results are lesser as we have trained on the subset of training data with 50000 examples having only 39 features while the baseline has been trained over a million features for over 4 million training examples.

2. For the two sentences
   Line 1: What software does Pixar use in its computer animations?
   Line 2: What kind of software do Pixar and Walt Disney use to animate pictures?
   The actual label is 1 for these pair of sentences. However, the predicted label is 0. This happens because we are using features like word-match-share, number of common words, sentence length and number of unique words. All these features will show high non-similarity score between both these sentences.
   These features work well in the following pair of sentences and predicts correctly:
   Line 1: How do I reset my WeChat password?
   Line 2: How can I reset my WeChat password?

3. Context free models such as Glove and Word2vec are static word embeddings in the sense that each word has a singular representation irrespective of the context in which it is used. For example, the word "broke" would have the same representation in the sentences: "I am broke" and "My car broke down". Bert generates contextual word embeddings which are represenations of each word based on the other words in a sentence.

4. Some of the existing models using pre-trained contextual representations introduced prior to BERT model such as Semi-supervised Sequence Learning, Generative Pre-Training, ELMo and ULMFit are only unidirectional or shallowly bidirectional. These methods are trained only using left-to-right context of a word in a sentence or vice versa (unidirectional), or shallowly concatenate the left-to-right and right-to-left contexts (shallowly bidirectional). Although, these contributed to a significant improvement over context free embeddings, they are still susceptible to errors due to loss of information. The design of BERT model combats these shortcomings by capturing contextual information from both left as well as right contexts of a word from the very first layer all the way through to the last layer of its neural network framework, thus making it deeply bidirectional.

5. One of the limitations of BERT model would be error introduced due to "max_seq_length" parameter used which signifies the maximum length of sequence to consider. This would mean, there would be loss of information for longer length sequences.

6. BERT's performance can be affected if there is a drastic difference between the data on which the model is initially trained and the data on which it is fine tuned.

7. BiMPM gave really good results since there are four different types of matching strategies used in the matching layer of the model's architecture.

9 Future Work

1. Add Bert embeddings as input to BiMPM and implement Zero-Shot Word Sense Disambiguation using Sense Definition Embeddings to obtain sense embeddings.

2. Build more robust non-neural network models by adding features to the current 39 features.

10 Acknowledgements

We would like to thank University of Colorado, Boulder for giving us the opportunity to work on this project. We would like to thank Professor Chenhao Tan for his guidance and the teaching assistants for their helpful inputs. Additionally, we would like to thank Kaggle and Quora for providing Quora Question Pairs dataset.

References

[1] Lakshay Sharma, Laura Graesser, Nikita Nangia and Utku Evci. *Natural Language Understanding with the Quora Question Pairs Dataset*, 2019.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019.

[3] Quora. Quora Question Pairs *https://www.kaggle.com/c/quora-question-pairs/data* , 2017.

[4] Hassan Shahmohammadi, MirHossein Dezfoulian and Muharram Mansoorizadeh *An Extensive Comparison of Feature Extraction Methods for Paraphrase Detection*, 2018.

[5] Zhiguo Wang, Wael Hamza, Radu Florian *Bilateral Multi-Perspective Matching for Natural Language Sentences*, 2017