Sentence Duplicate Detection

AY20/21 Sem 1 CS3244 Project

Group 45

Jeremy Lee, Lee Penn Han, Loke Kay Chi,

Lua Jun An, Neaton Ang, Swa Yong Shen

Table of Contents

01

Problem Statement

Why is identifying duplicate questions important?

300,000,000

Monthly active users

https://www.vox.com/recode/2019/5/16/18627157/quora-value-billion-question-answer

Reasons for Question Duplication

01 Users do not check if their question has been asked before

02 Users feel that an existing question may be too old and outdated

Problems with Question Duplication

- For knowledge-sharing platforms such as Quora and Stackoverflow, a central location for each post facilitate more efficient discussion.
- Duplicate questions negatively affects user experience as discussion for the same question is segmented into different posts

Difficulty in finding appropriate answers

Varying consensus across different posts

Miss out on important discussion

Identifying duplicate questions is HARD.

- "Who is the leader of the free world?" VS "Who is the president of the USA?"
  - Similar meaning, but very few common words!
- "How much does an apple cost?" VS "How much does an Apple iPhone cost?"
  - Different meaning, but many common words!

Motivation

	Quora	Stack Overflow	Reddit
Active users	300 million monthly[1]	100+ million yearly[2]	430 million monthly[3]

[1] https://www.vox.com/recode/2019/5/16/18627157/quora-value-billion-question-answer
[2] https://stackoverflow.blog/2019/01/18/state-of-the-stack-2019-a-year-in-review/
[3] https://www.theverge.com/2020/12/1/21754984/reddit-dau-daily-users-revealed

02

Related Work

Past work done on this area of research

Related Works

Team/Author	Method	F1	Accuracy	Log loss
DL guys[1]	● Siamese LSTM pretrained with gloVe. ● Decomposable attention neural network.[2] ● Stacking ensemble	-	-	0.116
ashwin4glory[3]	Logistic regression, SVM	-	-	0.46
Elier Cohen[4]	MaLSTM	-	0.82	0.130

[1]https://www.kaggle.com/c/quora-question-pairs/discussion/34355
[2]https://arxiv.org/abs/1606.01933
[3]https://github.com/ashwin4glory/Quora-Question-Pair-Similarity/blob/master/4.ML_models.ipynb
[4]https://blog.mlreview.com/implementing-malstm-on-kaggles-quora-question-pairs-competition-8b31b0b16a07

03

Project Workflow

Our project timeline

Project Timeline

Week	5	6	Recess	7	8	9	10	11	12	13	Reading
EDA	Investigating the Quora dataset										
Feature Engineering		Cosine Dist, Common Tokens, Common Substring etc.									
Individual Model				LR, SVM, RFC, XGBoost, RNN							
Final Model Design						Siamese LSTM in RNN with Feature Engineering					
Presentation Slide Creation						Create and update presentation slides for past work					
Finalization										Record presentation	

04

Exploratory Data Analysis & Feature Engineering

Data cleaning and analysis forms the backbone of machine learning

Importance of EDA & Feature Engineering

- EDA allows us to **analyse the distribution** of our dataset based on selected features.
- Together with feature engineering, we are able to **identify** through histogram plots **which features would be more useful** in prediction of classes.
- Ultimately, Quora Question Pairs is a **classification** problem → **Data needs to be separable.**

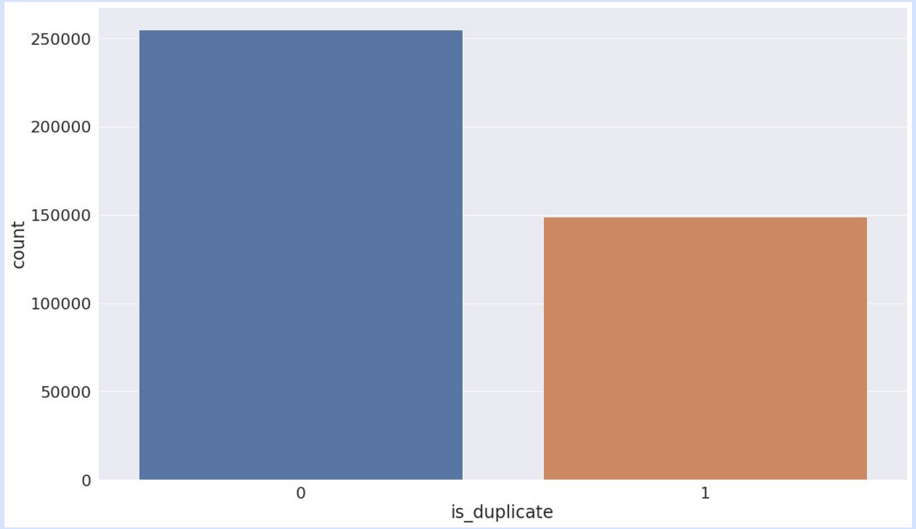
Checking for Imbalanced Dataset

Legend:

- 0: Non duplicates
- 1: Duplicates

Analysis:

Since the dataset is skewed towards non-duplicates, undersampling is considered as a method to balance the data.



Data Preparation

Removing stop words

- Remove common words across the document, like **articles** (e.g. the, a) and **pronouns** (e.g. he, she)

Decontracting words

- Convert **contractions** (e.g. aren't -> are not) into their long form to allow detection by GloVe embedding

Replacing special characters

- Convert **special characters** (e.g. % —> percent) into their word equivalents

Why Feature Engineering?

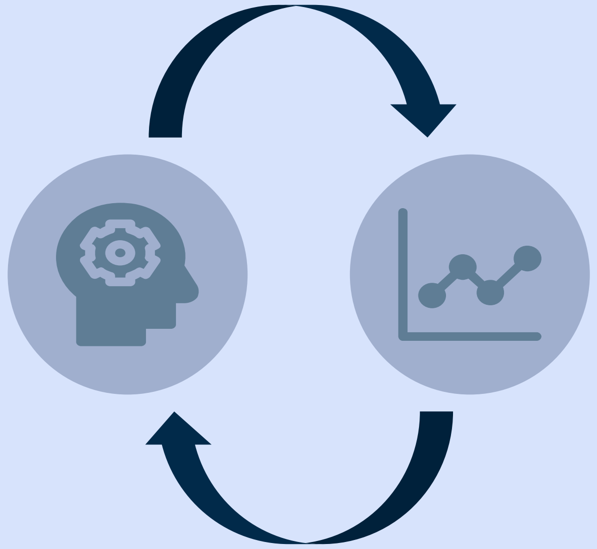
Text input

Dataset is in text form, need to convert to **numeric form**

Lack of information

Raw data only has **two** questions in text. Difficult to quantify their values

Choosing Features



Research

Identify common features chosen in NLP models

E.g. Fuzzywuzzy, POS tagging, TF-IDF, BoW

Data Exploration

Study and visualise the relationship of each feature.

Part-Of-Speech (POS) Tagging

- POS tagging is a common practice in NLP to **grammatically categorise** parts of the text
- The **nouns** and **adjectives** could be informative in telling us about the semantics of a given text
- We decided to use these features in our model training:
  - **Common adjectives min**
  - **Common adjectives max**
  - **Common noun min**
  - **Common noun max**

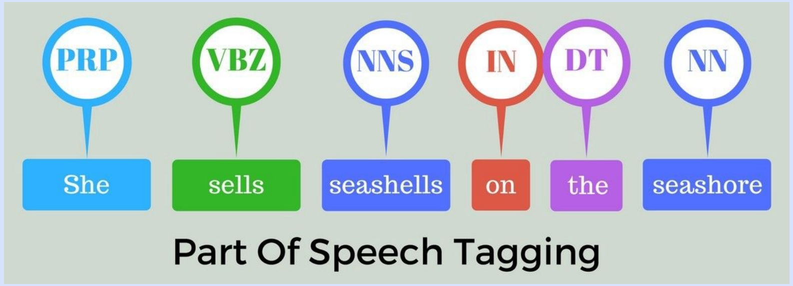


Image credits:
https://medium.com/@muddaprince456/categorizing-and-pos-tagging-with-nltk-python-28f2bc9312c3

Word Embeddings / Sentence Encoders

- Embeddings / Vectors can be **compared to using cosine distance** as a similarity metric
- Libraries / Embeddings explored: **GloVe, Word2Vec, Universal Sentence Encoder, Doc2Vec**
- We decided to use a NLP pre-trained word vector dataset by Stanford University: Global Vector for Word Representation (GloVe) Embeddings
- GloVe Embeddings **capture the meaning** of a word or sentence in a matrix form that allows us to **mathematically compare** their meanings
- The word vector data set has 840 billion tokens and 300 dimensions

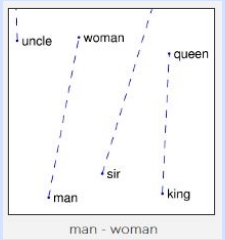


Image credits:
https://nlp.stanford.edu/projects/glove/

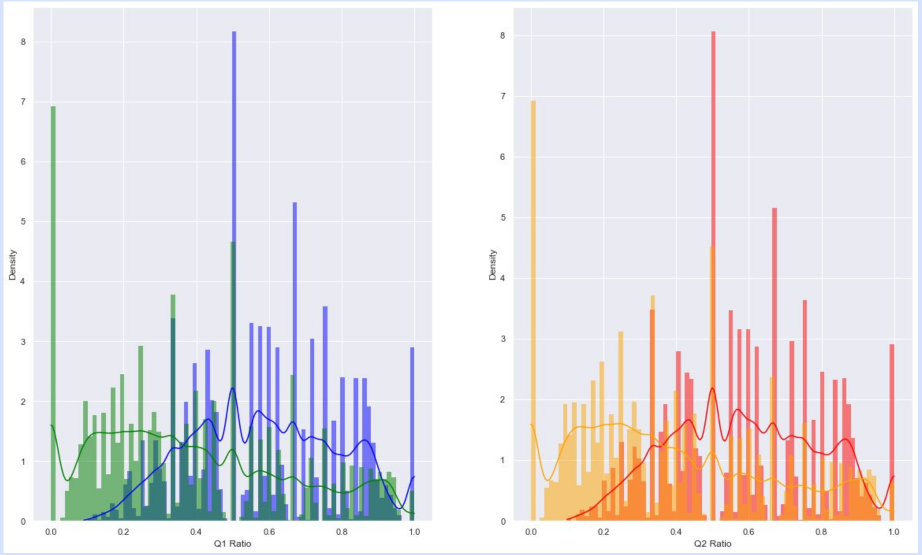
Overlapping Words Ratio

Legend:

- Blue/Red: Duplicates
- Green/Yellow: Non-duplicates

Analysis:

- From the histogram plots of common word ratios, the duplicated questions tend to have a higher overlapping words ratio, as observed by the Blue and Red sections (skewed to the right)
- We can extract this defining feature to be used in our modelling



Fuzzywuzzy

- Fuzzywuzzy is a python package known for **string matching**, and helps to compare words even if there are different spellings (e.g. misspellings, incomplete words)
- It uses the **Levenshtein Distance** to calculate the differences between sequences and patterns within a sentence
- Since we are dealing with **semantics** in our project, extracting these fuzzy features might be significant to our model
- We decided to use these features in training our model: **fuzz ratio, fuzz partial ratio, fuzz token sort ratio, fuzz token set ratio**

```
fuzz.ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Mets") => 45

fuzz.partial_ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Mets") => 45

fuzz.token_sort_ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Mets") => 100
```

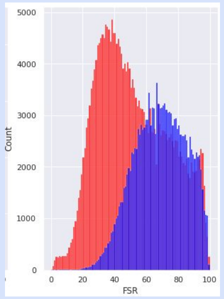
Fuzzy Ratio, Partial, Sort, Set

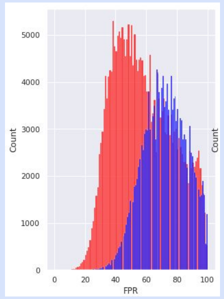
Fuzz Ratio (FSR):
The ratio of matching tokens in a comparison of 2 strings of which order of tokens matter

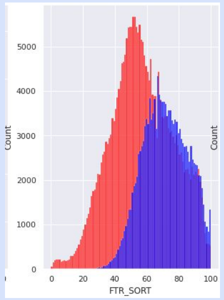
Fuzz partial ratio (FPR):
The ratio of matching tokens in a comparison of subsets of 2 strings of which order of tokens matter.

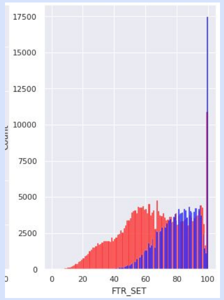
Fuzz token sort ratio (FTR_SORT):
The ratio of matching tokens in a comparison of 2 strings of which order of tokens does not matter.

Fuzz token set ratio (FTR_SET):
The ratio of matching tokens in a comparison of 2 strings of which repeated tokens and order of tokens does not matter.





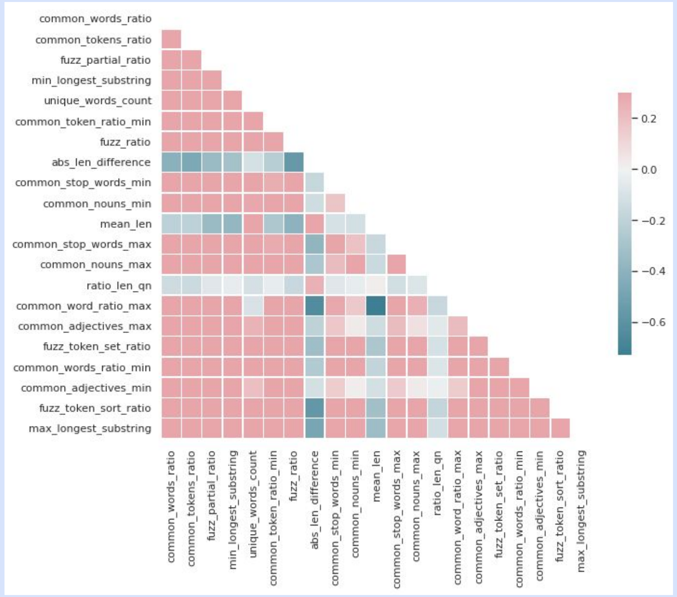




Feature Engineering

No.	Feature	No.	Feature
1	Number of unique words	12	Common Noun (Maximum)
2	Ratio of Common Words to Total Words	13	Fuzzy Wuzzy Fuzz Ratio
3	Common Word Ratio (Minimum)	14	Fuzzy Wuzzy Fuzz Partial Ratio
4	Common Word Ratio (Maximum)	15	Fuzzy Wuzzy Token Sort Ratio
5	Common Stop Words (Min)	16	Fuzzy Wuzzy Token Set Ratio
6	Common Stop Words (Max)	17	Mean Length of 2 Questions
7	Common Tokens (Min)	18	Ratio of Length of Questions
8	Common Tokens (Max)	19	Absolute Length Difference
9	Common Adjectives (Minimum)	20	Longest Matching Substring (Minimum)
10.	Common Adjective (Maximum)	21	Longest Matching Substring (Maximum)
11.	Common Noun (Minimum)	22	Embedded Cosine Distance

Correlation Matrix



Analysis Of Correlation Matrix:

- After running through the pairwise correlation of all the engineered features, we ensure that there are features that are highly collinear to one another **(> 0.8 or < -0.8).**

- Features that have high collinearity would **drown out the effects of other features**, hence they should be removed from the dataset before performing machine learning.

05

Our Models

Logistic Regression, Random Forest Classifier,
Support Vector Machine, XGBoost,
Recurrent Neural Network

Summary of Model Performance

Model	Accuracy	Precision	Recall	F1
LR	0.680	0.576	0.500	0.536
SVM	0.702	0.559	0.903	0.690
RFC	0.710	0.580	0.760	0.660
XGBoost	0.737	0.597	0.867	0.708
RNN-LSTM	0.828	0.759	0.781	0.770

- Baseline models allow us to set expectations for subsequent model performances (e.g. accuracy)
- By considering these baseline models, it allows us to quickly eliminate more advanced models that perform worse than these baseline models

Logistic Regression Implementation

1. Split the training data into a 70-30 using train_test_split
2. Undersampled train set using Random Undersampling
3. Cross validation (CV) with k = 5 to check if model is overfitting
   → CV scores are quite consistent so there should not be major overfitting
4. RandomizedSearchCV, tuning **penalty** and **solver** (algorithm used in the optimisation problem)
   - Penalty: None
   - Solver: Sag

Logistic Regression

Model Overview

Evaluation	Value
Accuracy	0.680
Precision	0.576
Recall	0.500
F1	0.536

Model Analysis

- A simple and easy model to implement, provides us inference about each feature
- Serves as a good baseline for binary classification problems
- Linear decision surface, can't solve nonlinear problems *(https://towardsdatascience.com/when-logistic-regression-simply-doesnt-work-8cd8f2f9d997)*
- Prone to overfitting in high dimensional dataset *(http://eointravers.com/post/logistic-overfit/)*
- NLP is a complex problem → Only baseline

SVM Implementation

1. Random split of data into a 80-20 train test split for training of the actual SVM model and 50-50 train test split for GridSearchCV parameter tuning (lower number of training data as GridSearch is slow).
2. Normalize each training and testing set using scikit-learn's StandardScaler.
3. Use scikit-learn's GridSearchCV to find the best permutation of the following parameters for our final SVM model. This also runs 5-fold cross validation for each permutation of model trained.

Kernel
rbf 🟩
sigmoid
linear

C (regularisation param)
0.1
1
10 🟩
100

Gamma
1
0.1
0.01 🟩
0.001

4. Train an SVM model based on best performing parameters found in step 3 [C: 10, Gamma: 0.01, Kernel: rbf].
5. Evaluate performance of SVM model.

Support Vector Machine (SVM)

Model Overview

Evaluation	Value
Accuracy	0.702
Precision	0.559
Recall	0.903
F1	0.690

Model Analysis

- Allows for separation of the classes through a kernel even if data is not linearly separable *(https://core.ac.uk/reader/6302770)*
- Provides good out of sample generalisation through regularisation to prevent overfitting on the dataset
- Long training time if data is large *(https://stats.stackexchange.com/questions/314329/can-support-vector-machine-be-used-in-large-data)*
- Under-represented classes may skew the decision boundary due to high variance *(https://www.quora.com/Why-does-SVM-not-perform-well-for-imbalanced-data)*

RFC Implementation

- Random split of data into an 80-20 train test split.
- Experimented with undersample of the train set
- Created the RFC Object with SKLearn
- Using n_estimators = 200, any additional trees might cause overfitting and too few trees might not fit the data well
- Capping the depth of the tree at 12, it prevents overfitting as a large depth might fit perfectly for train data but does not generalize the test data well
- Fit and predict test results from train test split
- Check accuracy score

Random Forest Classifier

Model Overview

Evaluation	Value
Accuracy	0.710
Precision	0.580
Recall	0.760
F1	0.660

Model Analysis

- Adjusting n_estimators and max_depth of tree mitigated overfitting and helped to improve overall accuracy.
- Able to handle large dataset and dimensionality
- Computationally inefficient due to large number of decision trees
- Difficult to guarantee that the optimal tree can be found due to the ensemble of Decision Trees

XGBoost Implementation

- Used GloVe word embeddings to attempt to capture meaning of words
- Random undersampling
- Used XGBoost’s DMatrix
  - An internal representation of matrices XGBoost, which optimises training efficiency and memory consumption
- Used XGBoost’s native cross validation with k = 5
- **Tuned parameters:**
  - **eta** (learning rate)
  - **max_depth** (affects how deep the tree is, hence affecting overfitting)
  - **min_child_weight** (minimum weight needed to create a new node in the tree - smaller weights allow for more splits and hence possible overfitting)
  - **sub_sample** (number of rows of data to use at each step, helps control overfitting)
  - **colsample_bytree** (number of features to use at each step)

XGBoost

Model Overview

Evaluation	Value
Accuracy	0.737
Precision	0.597
Recall	0.867
F1	0.708

Model Analysis

- Possible reason for better performance: **ensemble** method that **builds on weaker learners to train stronger learners** (boosting)
    - **Ensemble method helps with reducing variance** and hence overfitting
    - **Learn from misclassified output to improve existing model**
- However, **difficult to capture semantic meaning**
    - E.g. “Who is the leader of the free world” vs “Who is the President of the USA”
        - **Similar meaning, but few common words**!
    - E.g. “How much does an apple cost?” vs “How much does an Apple iphone cost?”
        - **Vastly different meaning, but many common words**

Generally,

- Models that relied heavily on our extracted/engineered data did not do as well (e.g. LR, RFC, SVM, XGBoost)

- Cannot be certain that we have extracted useful information that helps in identifying duplicate pairs, **requires specific domain knowledge in linguistics**
    - Potentially difficult to extract good features

- **Extracted features tend to describe the properties** of a pair of sentences (e.g. number of common words), which **ignores contextual meaning**

RNN Implementation

- Data processing was not done to balance the dataset as an experimental model with a randomly undersampled dataset resulted in worse performance, this is likely due to the fact that there was fewer training data
- Each question was converted into a weighted vector using pre-trained word embeddings from GloVe.
- The question pairs were also preprocessed into engineered features such as Cosine Distance, Fuzz Ratio and Common Tokens
- The weighted vectors and engineered features were inputs to a LSTM network using the Keras Implementation library in Python.
- **Tuned Parameters**
    - Dropout Value
    - LSTM Regularization
    - Dense Layer Regularization
    - No. of Neurons per layer

Recurrent Neural Network

Model Overview

Evaluation	Value
Accuracy	0.828
Precision	0.759
Recall	0.781
F1	0.770

Model Analysis

- **Recurrent Neural Networks**
  - Is able to loop back previous predictions to affect subsequent neurons
  - We are able to retain the semantics of a question to better analyse 2 questions

- **Long Short Term Memory (LSTM) Layer**
  - This added **LSTM** layer is able to better capture semantic information and predict semantic relatedness of two sentences

Evaluation Metrics Analysis

Accuracy

Ratio of correct predictions to the total number of predictions

Precision

Ratio of correctly predicted positive observations to the total number of predicted positive observations.

Recall

Ratio of correctly predicted positive observations to the total number of true positive observations, also known as the true positive rate.

F1

Weighted average of the Precision and Recall score.

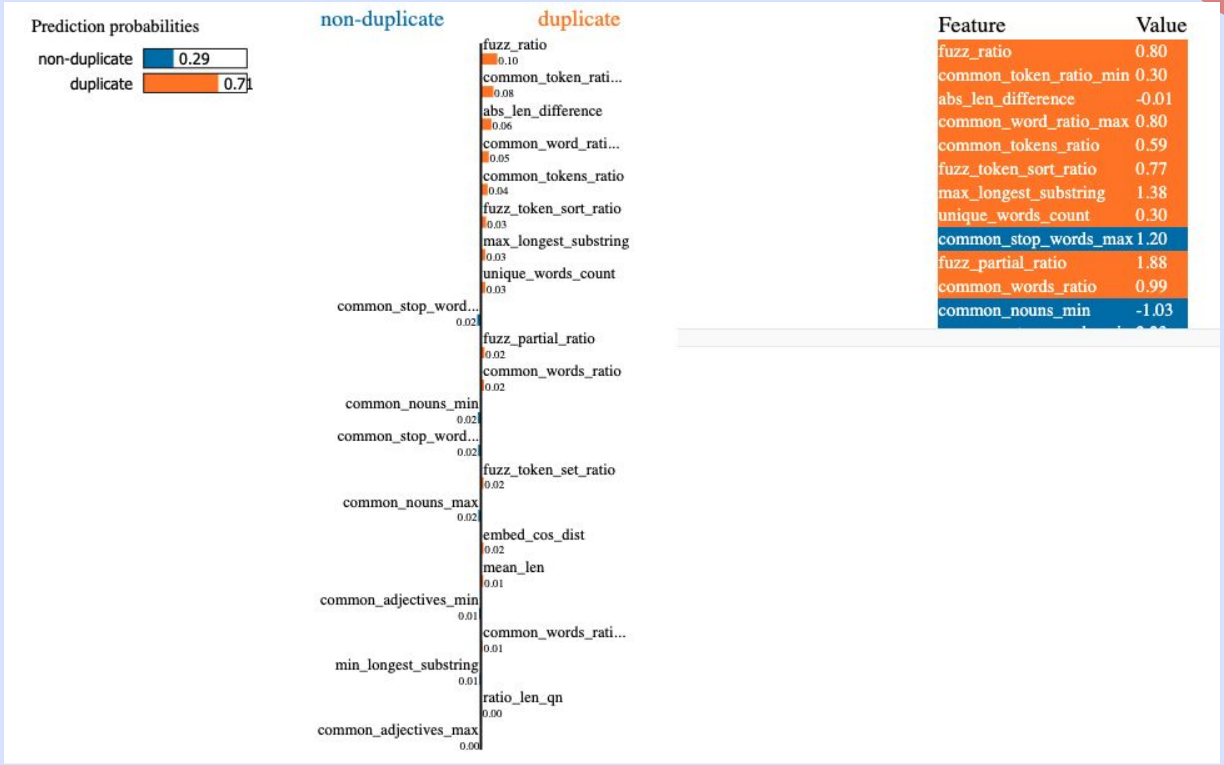
Why Precision?

- Precision is the ratio of correctly predicted positive observations to the total number of predicted positive observations.
- In our case, the penalty for **false positives** is greater than the penalty for **false negatives.**
- This would mean that **a new and unique question may get deleted/removed** which would result in information loss for the stakeholders.

False positive	When a non-duplicated question gets identified as a duplicated question

XAI For XGBoost (LIME)

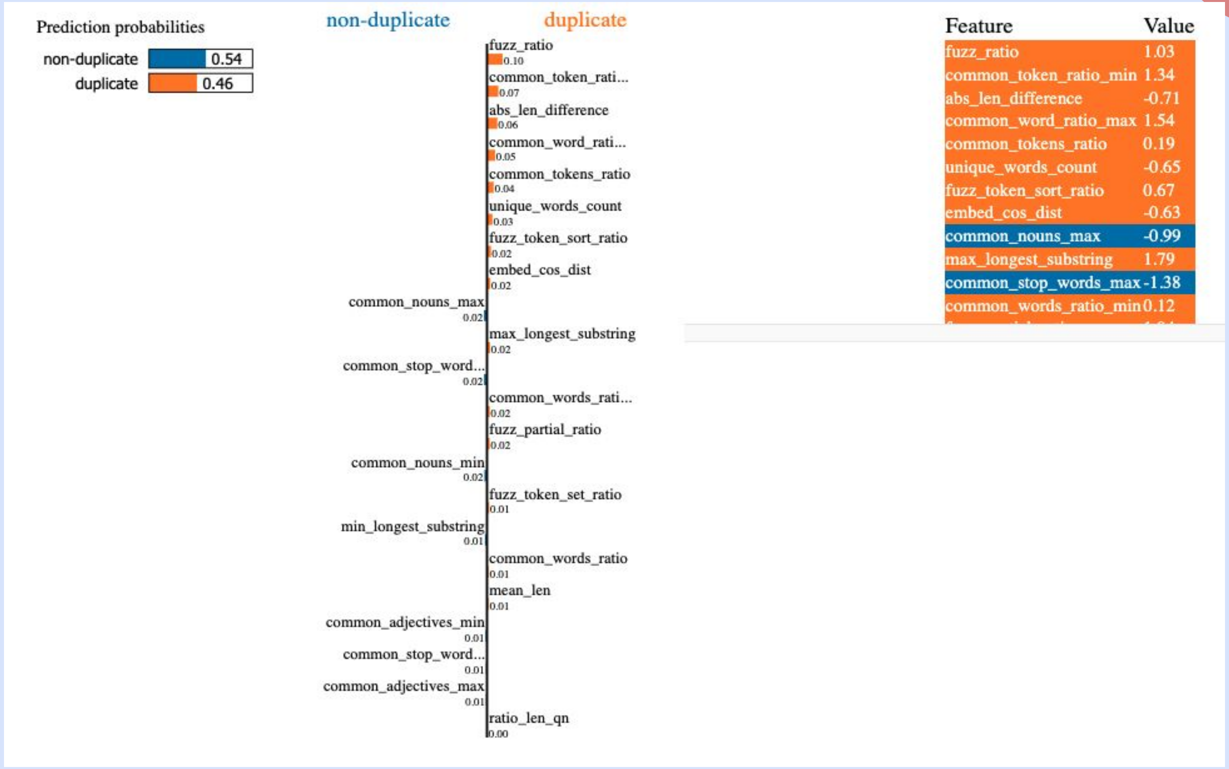
- Taking our first test input as reference, we can see that **fuzz_ratio, common_token_ratio_min, abs_len_difference and common_word_ratio_max** have a **larger weight** in deciding whether 2 sentences are duplicates.
- The **orange/blue** signifies that the feature has a **positive/negative correlation to the output** respectively.



Feature	Value
fuzz_ratio	0.80
common_token_ratio_min	0.30
abs_len_difference	-0.01
common_word_ratio_max	0.80
common_tokens_ratio	0.59
fuzz_token_sort_ratio	0.77
max_longest_substring	1.38
unique_words_count	0.30
common_stop_words_max	1.20
fuzz_partial_ratio	1.88
common_words_ratio	0.99
common_nouns_min	-1.03

XAI For XGBoost (LIME)

- Taking our second test input as reference, we can see that **fuzz_ratio, common_token_ratio_min, abs_len_difference and common_word_ratio_max** have a **larger weight** in deciding whether 2 sentences are duplicates.
- The **orange/blue** signifies that the feature has a **positive/negative correlation to the output** respectively.



Feature	Value
fuzz_ratio	1.03
common_token_ratio_min	1.34
abs_len_difference	-0.71
common_word_ratio_max	1.54
common_tokens_ratio	0.19
unique_words_count	-0.65
fuzz_token_sort_ratio	0.67
embed_cos_dist	-0.63
common_nouns_max	-0.99
max_longest_substring	1.79
common_stop_words_max	-1.38
common_words_ratio_min	0.12

Advantages & Disadvantages of using LIME

Advantages:

- We can see that the weights of the top features in the LIME explainer do make sense as they are factors that we deem would be crucial to explaining whether 2 questions are duplicates.
- The **LIME explainer is also much faster** (shorter runtime) and is able to churn more explanations within a shorter period of time.

Disadvantages:

- Since LIME creates a local linear model around the area that we are interested in, **the assumption is that the boundary can be linearly interpreted**, which might not always be the case.
- The LIME interpretation is also **unstable as it greatly depends on the sampling of the points** near the boundaries. Due to the instability, we cannot fully trust LIME to provide a full story of a complex model.

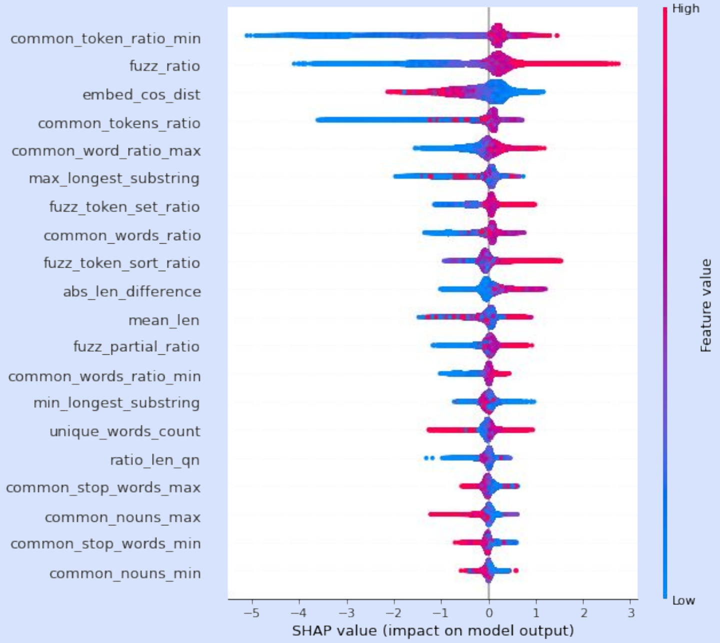
XAI For XGBoost (SHAP)

SHAP Analysis:

Top 5 Features are:

1. **Common Token Ratio Min**
2. **Fuzz Ratio**
3. **Embedded Cosine Distance**
4. **Common Tokens Ratio**
5. **Common Word Ratio Max**

From the SHAP summary plot, we can see that 1,2,4,5 are positively correlated to the output. In other words, the higher the values of these features, the more likely the questions are duplicates. On the flip side, a lower embedded cosine distance would mean that the questions are more likely to be duplicates.



Advantages & Disadvantages of using SHAP

Advantages:

- **SHAP is more stable than LIME** because it takes into account all the different permutations and combinations of the features to give a clearer representation of how each feature affects the prediction.

Disadvantages:

- SHAP takes **extremely long to compute** because it takes into account all the different permutations and combinations of the features of a given prediction.

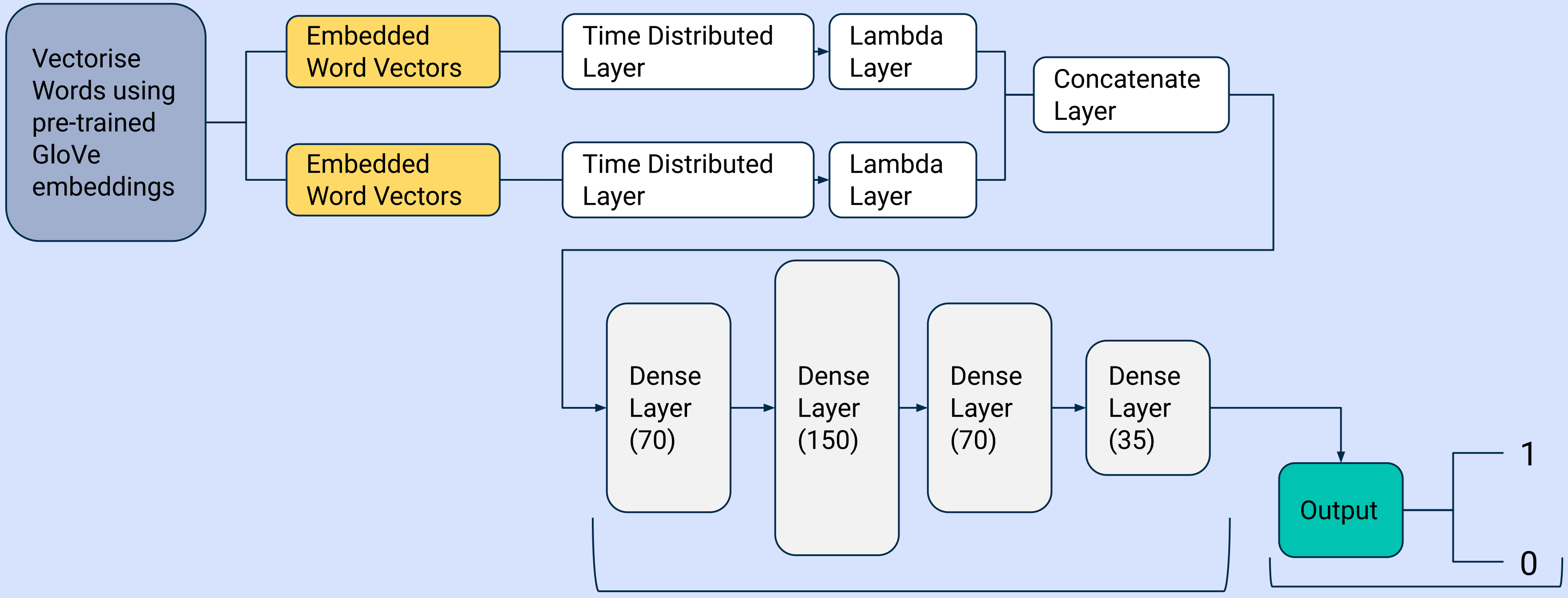
06

Final Model

Siamese Neural Network with
Engineered Features

Initial RNN Model

Recurrent Neural Network



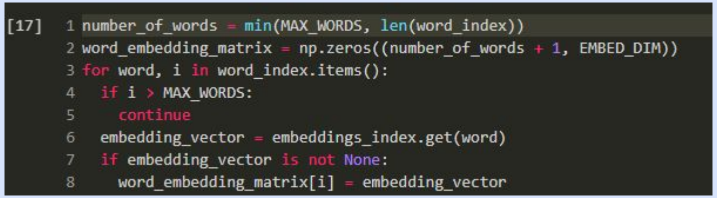
Activation: ReLu
Dropout: 0.1

Activation: Sigmoid
Loss

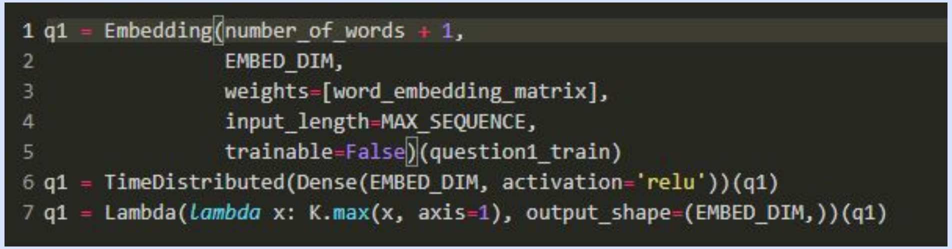
Initial RNN Model: Design

Data Preparation:
Each duplicate question pair which have been preprocessed sentences from EDA are tokenized into 2 vectors using a Tokenizer.
These vectors are weighted using the GloVe embedding matrix.

Neural Network Input Preparation:
Each Input is passed into a Embedding Layer to weight each vector. The output is then mapped to select the maximum value of each column in the vector using a Lambda function.


[17]

```python
number_of_words = min(MAX_WORDS, len(word_index))
word_embedding_matrix = np.zeros((number_of_words + 1, EMBED_DIM))
for word, i in word_index.items():
  if i > MAX_WORDS:
    continue
  embedding_vector = embeddings_index.get(word)
  if embedding_vector is not None:
    word_embedding_matrix[i] = embedding_vector
```



```python
q1 = Embedding(number_of_words + 1,
                 EMBED_DIM,
                 weights=[word_embedding_matrix],
                 input_length=MAX_SEQUENCE,
                 trainable=False)(question1_train)
q1 = TimeDistributed(Dense(EMBED_DIM, activation='relu'))(q1)
q1 = Lambda(lambda x: K.max(x, axis=1), output_shape=(EMBED_DIM,))(q1)
```

Initial RNN Model: Design

Parameters	Samples
Training Set	363839
Validation Set	10% of training Set
Testing Set	40427
Dropout	0.1
Regularization	No Regularization
Training Time	80.2 minutes for 50 epochs

Layers Implemented

Embedding Layer: The embedding layer maps the input question vector with its equivalent weights in the pre-trained word embedding matrix.
Time Distribution Layer: The time distributed layer adds a “time” dimension to the input.
Concatenate Layer: Joins the two embedding layer to pass as 1 input into the neural network
Within each Dense Layer:
Dense Layer: A layer with a user-specified number of neurons for neural network training, ReLu activation is applied on each neuron.
Dropout Layer: A layer that deactivates neurons with a user-specified probability as a means of reducing noise.
Batch Normalization Layer: A layer that will normalise mean output to be close to 0 and output standard deviation to 1 to help solve internal covariate shift.
Final Layer:
Dense Layer: A layer with 1 neuron. A sigmoid activation is used to summarise the duplicate prediction as a value between 0 to 1.

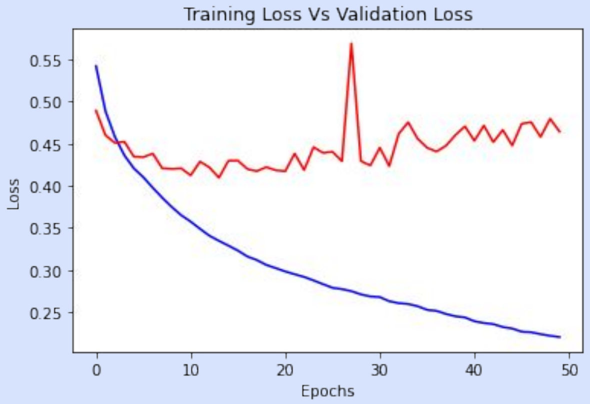
Initial RNN Model: Evaluation

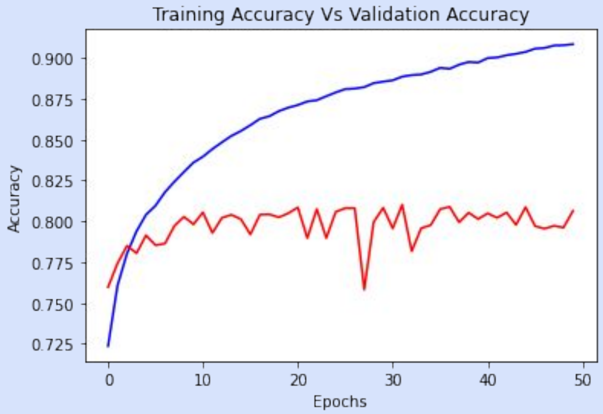
Model Description

The model overfits from the 5th epoch. The training accuracy overfits to reach 90% by 50th epoch while the validation loss generally stagnates regardless of the number of epochs.

Although the accuracy is higher than other models, it is undesirable due to the large overfitting since the model might be doing exceptionally well on the validation set. This causes the accuracy measure to be unreliable on unseen data.

Evaluation	Value
Accuracy	0.809
Precision	0.751
Recall	0.720
F1	0.736



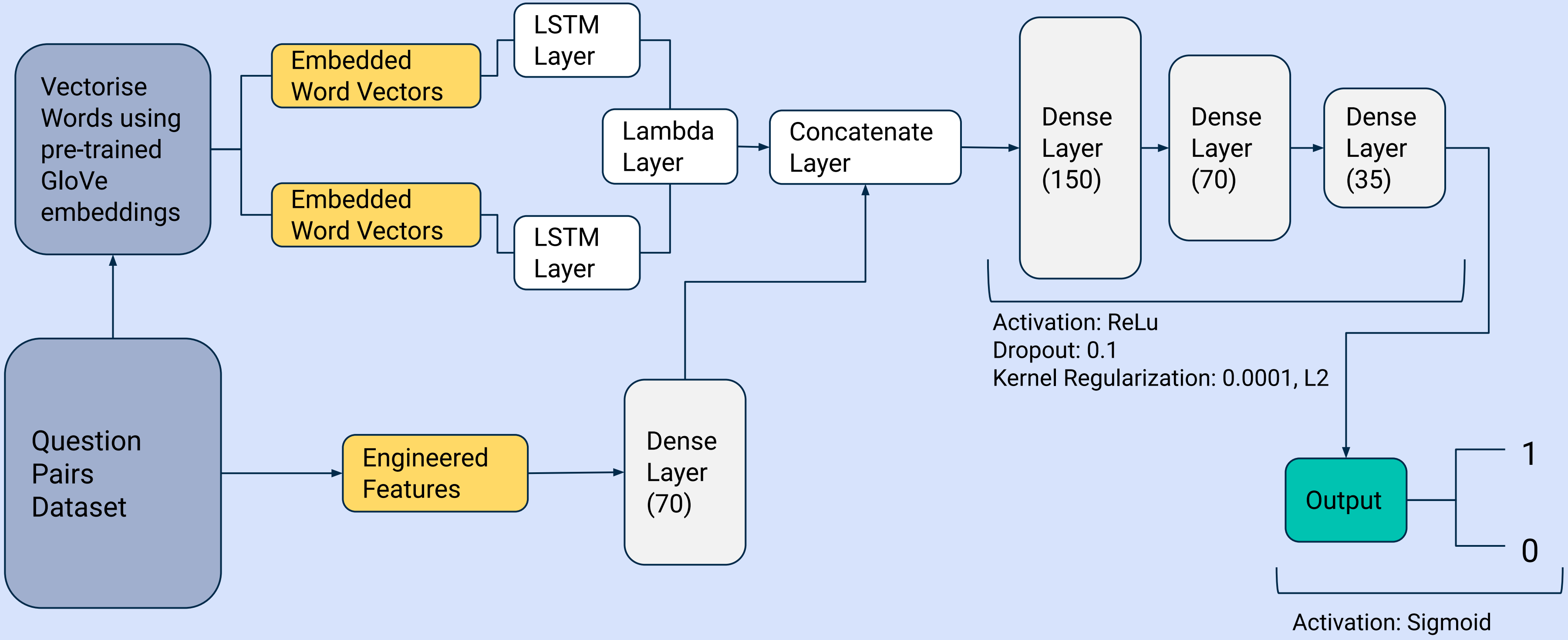


Attempted Adjustments

Model	Description	Remarks
LSTM RNN with Model Tuning	Dropout and Regularization value was increased to work on mitigating the overfit of training set.	Overfitting was better but still apparent after several epochs
Bidirectional LSTM method	Bidirectional LSTM was implemented to attempt learning the sentence semantics better since Bidirectional LSTM is .	Overfitting was still apparent. Accuracy did not improve by a large margin
Siamese LSTM method	Siamese LSTM was implemented to compare the semantic similarity of the question pairs.	Accuracy did not improve by a large margin. Overfitting was still apparent
Siamese LSTM method with Engineered Features	Engineered Features was added as a new Input layer to be concatenated with the LSTM layers. This was an attempt to improve the accuracy.	Accuracy improved by a sizeable margin and after some tuning, overfitting was less apparent.

Tuned RNN Model

Siamese LSTM Recurrent Neural Network with Engineered Features



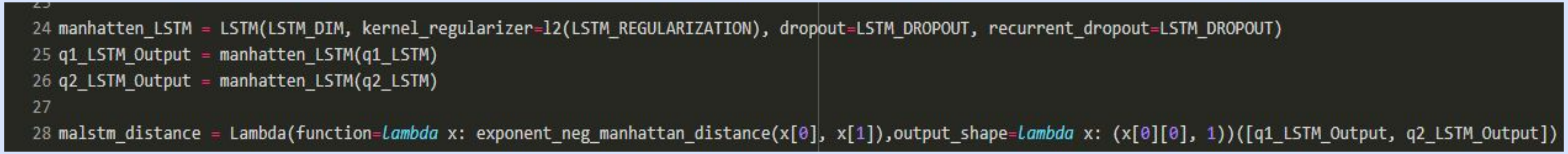
Tuned RNN Model: Design

Data Preparation:

Question pairs were processed similarly as per the initial model

Neural Network Input Preparation:

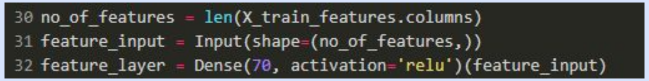
For both question vectors of a question pair, a LSTM layer is prepared with the same parameters to be applied on both question inputs. Both LSTM layers is then passed into a Lambda Layer which calculates the manhattan distance of the two vectors.



```
manhatten_LSTM = LSTM(LSTM_DIM, kernel_regularizer=l2(LSTM_REGULARIZATION), dropout=LSTM_DROPOUT, recurrent_dropout=LSTM_DROPOUT)
q1_LSTM_Output = manhatten_LSTM(q1_LSTM)
q2_LSTM_Output = manhatten_LSTM(q2_LSTM)

malstm_distance = Lambda(function=lambda x: exponent_neg_manhattan_distance(x[0], x[1]),output_shape=lambda x: (x[0][0], 1))([q1_LSTM_Output, q2_LSTM_Output])
```

The engineered features are passed into the Neural Network and trained on a Dense Layer of 70 Neurons before concatenation with the Siamese Network layer.



```
no_of_features = len(X_train_features.columns)
feature_input = Input(shape=(no_of_features,))
feature_layer = Dense(70, activation='relu')(feature_input)
```

Tuned RNN Model: Design

Parameters	Initial Model	Tuned Model
Training Set	363839	363839
Validation Set	10% of training Set	10% of training Set
Testing Set	40427	40427
Dropout	0.1	0.1
Regularization	0.0	0.0001
Training Time	80.2 minutes for 50 epochs	675.5 minutes for 50 epochs

New Layers Implemented

LSTM Layer:

Implements a layer which contains "forget" gates that will retain past states as part of memory to be involved in the next layer. This helps on analysing sentence semantics. The LSTM layer also mitigates the vanishing gradient problem due to the presence of "forget" gates within the layer.

Lambda Layer:

Maps the LSTM vector layer to calculate the manhattan distance of the two question inputs. The smaller the manhattan distance, the more similar the two questions are.

Justifications on Model Design

Manhattan Distance

- Manhattan Distance performs better as Euclidean distance is prone to undesirable plateaus in the objective function. This is due to the vanishing gradient problem for Euclidean Distance. (Mueller, J., & Thyagarajan, A. (2016, March))

Concatenate engineered features

As the accuracy for a pure LSTM network hovered around 81%, engineered features were considered to improve duplicate detection by analysing the similarity of sentence tokens and word orderings.

Dropout and Regularization values

- Varying Dropout values from 0.0 to 0.6 were tested for several epochs in the model. As the dropout increased, the accuracy decreased. This is likely due to the layer deactivating too many neuron due to the dropout value. 0.1 was a balance between learning the data and discarding noise

- Varying Regularization values from 0.00001 to 0.1 were tested. As regularization increased, the validation accuracy fell to below 80%. This was likely due to regularization affecting the parameters for the model, causing it to discard both noise and data.

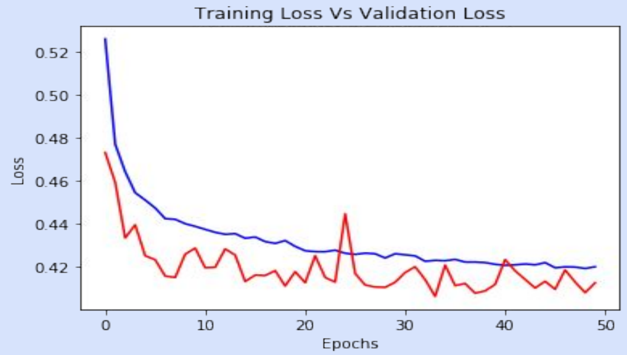
Tuned RNN Model

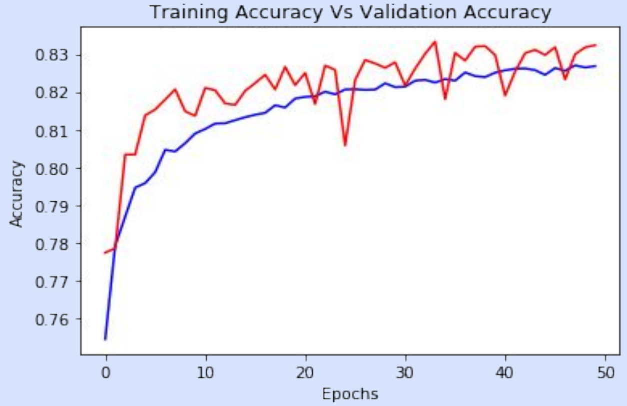
Model Description

With the implementation of Dropout Layer and Regularization, the training accuracy grows slower, reaching 83% at 50th epoch. However, the validation accuracy improves together with training accuracy.

The model performance also improved, likely due to addition of pairwise learning of the questions' similarity using Manhattan Distance.

Evaluation	Value
Accuracy	0.828 (+0.019)
Precision	0.759 (+0.008)
Recall	0.781 (+0.061)
F1	0.770 (+0.034)





Model Comparison

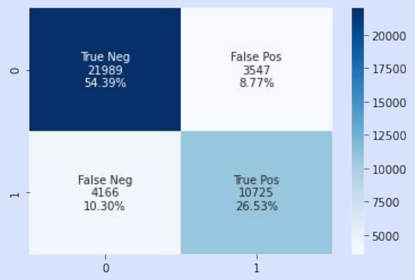


Fig 1: Initial Model

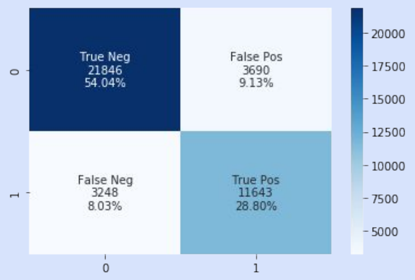


Fig 2: Tuned Model

- By comparing the confusion matrices, it is observable that our tuned model has predicted more positive classes than the initial model **(Initial Positives: 14,275, Tuned Positives: 15,333).**
- This has resulted in a **higher false positive count** with the tuned model.
- However, we also observe that the true positives has increased by a larger proportion than the false positives, causing Precision **to increase from 75.1% to 75.9% (reducing proportion of false positives).**
- Hence, our group decided to go with the final tuned model which is able to achieve an **improved overall accuracy of 82.8% as compared to the initial RNN model which has an accuracy of 80.9%.**

Did the Tuned Model work better?

Benefits:

- Performance increased compared to baseline model.
  - The Siamese Network design provided the model with the ability to compare the question semantics
  - Addition of engineered features allowed the model to compare general patterns such as common tokens between the question pair.

- Tuning of the model greatly reduced overfitting
  - Test accuracy is close to Training accuracy
  - Validation accuracy gradually improves together with training accuracy

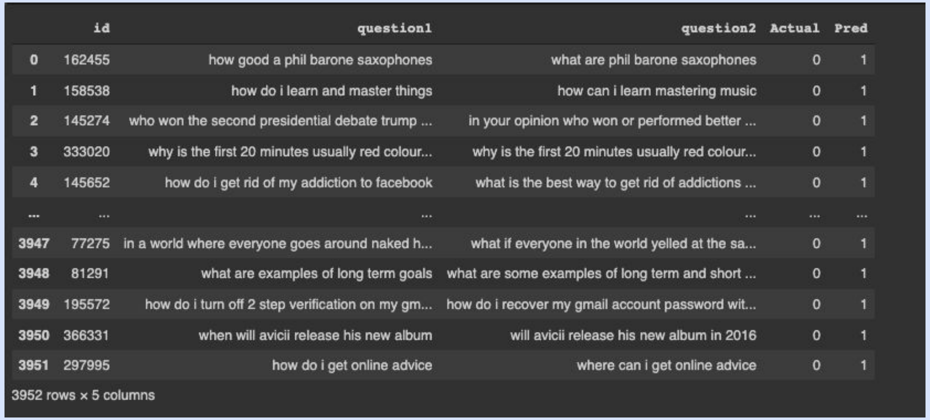
Shortfalls:

- Siamese Neural Network requires much more training time **(>10 hours)** than the usual RNN **(~1 hour)**. This is likely due to high computation time for pairwise learning as compared to pointwise learning.
  https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Blackbox Observation of Output

- As mentioned earlier, we prioritised on **false positives** as we hope to decrease false positive due to the detrimental effect it has to the user experience

- We extracted the **false positives** after predicting the test set on the Tuned Siamese RNN-LSTM Model to take a closer look at these misclassifications.



	id	question1	question2	Actual	Pred
0	162455	how good a phil barone saxophones	what are phil barone saxophones	0	1
1	158538	how do i learn and master things	how can i learn mastering music	0	1
2	145274	who won the second presidential debate trump ...	in your opinion who won or performed better ...	0	1
3	333020	why is the first 20 minutes usually red colour...	why is the first 20 minutes usually red colour...	0	1
4	145652	how do i get rid of my addiction to facebook	what is the best way to get rid of addictions ...	0	1
...	...	...	...	...	...
3947	77275	in a world where everyone goes around naked h...	what if everyone in the world yelled at the sa...	0	1
3948	81291	what are examples of long term goals	what are some examples of long term and short ...	0	1
3949	195572	how do i turn off 2 step verification on my gm...	how do i recover my gmail account password wit...	0	1
3950	366331	when will avicii release his new album	will avicii release his new album in 2016	0	1
3951	297995	how do i get online advice	where can i get online advice	0	1

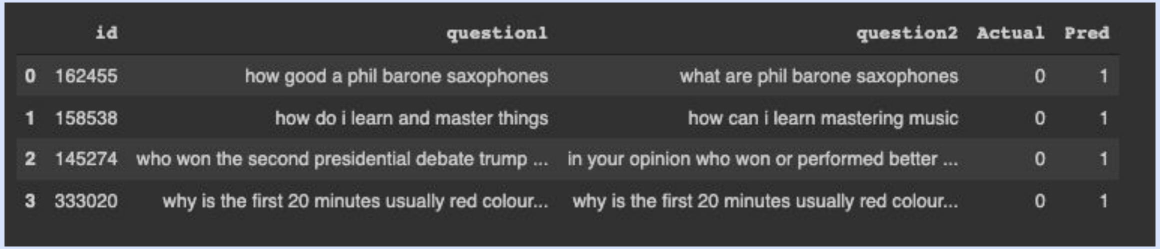
3952 rows × 5 columns

Blackbox Observation of Output

- Analysing the outputs of wrongly classified question pairs, it is observable that the questions pairs have a high number of common words and tokens.

- Our Siamese RNN-LSTM model includes the engineered features which contains **ratio of common words to total words, common tokens, etc**.

- The model may have balanced between these general features and sentence semantics wrongly due to the high feature values, which resulted in a wrong prediction.



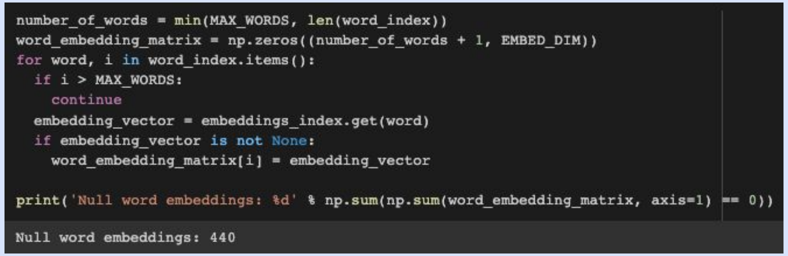
	id	question1	question2	Actual	Pred
0	162455	how good a phil barone saxophones	what are phil barone saxophones	0	1
1	158538	how do i learn and master things	how can i learn mastering music	0	1
2	145274	who won the second presidential debate trump ...	in your opinion who won or performed better ...	0	1
3	333020	why is the first 20 minutes usually red colour...	why is the first 20 minutes usually red colour...	0	1

07

Future Recommendations

Siamese LSTM Recurrent Neural Network with Engineered Features

Future Recommendations


```
number_of_words = min(MAX_WORDS, len(word_index))
word_embedding_matrix = np.zeros((number_of_words + 1, EMBED_DIM))
for word, i in word_index.items():
  if i > MAX_WORDS:
    continue
  embedding_vector = embeddings_index.get(word)
  if embedding_vector is not None:
    word_embedding_matrix[i] = embedding_vector

print('Null word embeddings: %d' % np.sum(np.sum(word_embedding_matrix, axis=1) == 0))
```

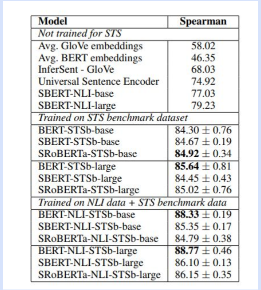
```
Null word embeddings: 440
```

- Even with the largest GloVe embeddings, we are still **experiencing 440 null word embeddings in the dataset.**
- This might cause certain **information loss** due to the missing embeddings.
- In the future when larger embeddings are available, we would be able to acquire 0 null word embeddings and hence, **achieve a better classification score with the information retained.**

Future Recommendation

- Nouns are more likely to be the subject of the sentence.
  https://www.grammarly.com/blog/nouns/
  Assign higher weights to nouns to better match similar subjects. This may better classify questions and identify duplicates.

- Use a more state of the art word embedder that can better identify sentence semantic similarity.
  - Currently, the project uses GloVe embedding.
  - More recently word embedders have been able to capture more semantic similarity when fine tuned with a Semantic Textual Similarity (STS) dataset.



Model	**Spearman**
Not trained for STS	
Avg. GloVe embeddings	58.02
Avg. BERT embeddings	46.35
InferSent - GloVe	68.03
Universal Sentence Encoder	74.92
SBERT-NLI-base	77.03
SBERT-NLI-large	79.23
Trained on STS benchmark dataset	
BERT-STSb-base	84.30 ± 0.76
SBERT-STSb-base	84.67 ± 0.19
SRoBERTa-STSb-base	**84.92** ± 0.34
BERT-STSb-large	**85.64** ± 0.81
SBERT-STSb-large	84.45 ± 0.43
SRoBERTa-STSb-large	85.02 ± 0.76
Trained on NLI data + STS benchmark data	
BERT-NLI-STSb-base	**88.33** ± 0.19
SBERT-NLI-STSb-base	85.35 ± 0.17
SRoBERTa-NLI-STSb-base	84.79 ± 0.38
BERT-NLI-STSb-large	**88.77** ± 0.46
SBERT-NLI-STSb-large	86.10 ± 0.13
SRoBERTa-NLI-STSb-large	86.15 ± 0.35

Table retrieved from:
Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.

Conclusion

- Started off working on baseline models with engineered features before moving on to implement more complex models like the RNN and Siamese Neural Network.
- We can see that the RNN models are better at modelling this complex NLP problem as compared to the baseline classifiers.
- We hope that various stakeholders would be able to deploy our model to identify duplicated questions on their platform in order to centralise the answers to these questions.
- By minimising duplicated threads, we hope to reduce confusion by reaching a common consensus under one centralised question.

-The End-