Quora Question Duplication Detection & Generative Question Modelling

Nguyen V. Binh
School of Computing
National University of Singapore
nguyen.binh@u.nus.edu

Eugene Chong
School of Computing
National University of Singapore
eugene.chong@u.nus.edu

Arnold Lek Jian Ming
College of Humanities and Sciences
National University of Singapore
e0726350@u.nus.edu

Chang Jin Lynn
College of Humanities and Sciences
National University of Singapore
e0774041@u.nus.edu

Huang HongRui
College of Humanities and Sciences
National University of Singapore
e0726131@u.nus.edu

Ramasamy Shreya
College of Humanities and Sciences
National University of Singapore
rsshreya12@u.nus.edu

Abstract

This project presents a two-fold approach employing advanced machine learning techniques. Firstly, we introduce a classification model, built upon diverse neural network architectures, for detecting duplicate questions, with the best model achieving 90.13% accuracy on unseen samples. Secondly, we explore the integration of generative AI methods to not only enhance user experience but also stimulate curiosity by generating related questions on specific topics. We rigorously evaluate both models using a comprehensive set of metrics, demonstrating their efficacy. Our results chart a promising course for future research and potential advancements in content management on Quora. This work contributes to the ongoing evolution of machine learning applications for enhancing content quality and user engagement on online platforms.

1 Introduction

Quora is a community-driven question-and-answer (Q&A) website, where users can ask questions and collaborate in giving and editing responses. With a high amount of visitor traffic (over 100 million visitors a month), it is a common occurrence for users to ask questions that are similar to or repeat questions that have already been asked on the site previously. Having multiple questions with the same content and intent can greatly diminish the users' experience, as seekers will have to peruse multiple instances to find the best answer to their queries, while writers will be pressured to provide the same answers for each new instance of the same question.

As such, this project focuses on machine learning solutions to detect when any two questions are duplicates, using Natural Language Processing (NLP) methods. In addition, we also experiments with the use of text generation in order to supplement and enhance the dataset, which could improve the classification model's performance in detection.

2 Related Work

Word2Vec Word2Vec, introduced by Mikolov et al.[1] in 2013, has been a cornerstone in the field of Natural Language Processing (NLP). It is an unsupervised learning algorithm that learns vector representations of words in a high-dimensional space. These representations capture semantic and syntactic similarities among words, making Word2Vec a powerful tool for tasks such as semantic similarity, analogy detection, and named entity recognition. The algorithm comes in two flavors: Continuous Bag of Words (CBOW) and Skip-Gram, each with its own strengths depending on the task and data at hand. Word2Vec's success has inspired a plethora of subsequent work in word embeddings, including GloVe[2] and fastText,[3] and has paved the way for more complex models like BERT. In our work, we leverage the power of Word2Vec for feature extraction in the task of question duplication detection.

Recurrent Neural Networks (RNN) First conceptualized in the 1980s, RNNs have undergone various adaptations and improvements over the years. Unlike traditional feedforward neural networks, RNNs possess the ability to retain information from previous inputs through recurrent connections in their architecture. This intrinsic memory enables them to effectively model sequences, making them suitable for applications such as time series analysis, language modeling, and speech recognition. Despite their expressive power, RNNs face challenges such as vanishing gradients, limiting their effectiveness in capturing long-term dependencies. Notable variants, like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), have been introduced to address these issues, enhancing the capability of RNNs in preserving and utilizing information over extended sequences. Our research extends the utility of RNNs in the domain of question duplication detection, harnessing their sequential modeling prowess to discern nuanced similarities in textual data.

Siamese Neural Networks Siamese neural networks represent a significant development in the field of machine learning, particularly for tasks involving the comparison of two distinct inputs. Introduced by Bromley and LeCun in the early 1990s for signature verification,[4] the architecture has since been adapted for a variety of tasks. The Siamese network's unique architecture, which involves two parallel networks sharing the same parameters, allows it to effectively measure the similarity between inputs. This makes it particularly suited for tasks such as duplicate detection, facial recognition, and one-shot learning. Recent facial recognition and verification models like DeepFace[5] incorporate more complex networks, such as Convolutional Neural Networks (CNNs) further enhancing its capabilities. Our work builds upon these developments, employing a Siamese network with LSTM for the task of question duplication detection.

Transformers The Transformer architecture, introduced in the seminal paper "Attention is All You Need" by Vaswani et al.[6] in 2017 has revolutionized the field of Natural Language Processing (NLP). The paper proposed a novel architecture that, unlike its predecessors, relies entirely on self-attention mechanisms, eliminating the need for recurrence and convolutions. This allows the model to capture dependencies regardless of their distance in the input sequences, leading to improved translation performance. The Transformer model's architecture consists of an encoder and decoder, each composed of multiple identical layers with self-attention and point-wise feed-forward networks. The introduction of multi-head attention allows the model to focus on different positions, capturing various aspects of the input sequence. This model has paved the way for several subsequent models like BERT and GPT, which have achieved state-of-the-art results on numerous NLP tasks. In our work, we fine-tuned BERT[7], a Transformer-based model, for the task of question duplication detection, leveraging the power of the Transformer's attention mechanism to understand the semantic similarity between questions.

3 Dataset Description

The dataset used in this project is the Quora Question Pair dataset, a constituent dataset of the General Language Understanding Evaluation (GLUE) benchmark, specifically designed for the assessment of general language understanding tasks. This dataset comprises of question pairs from the community question-answering website Quora. The inherent objective is to determine the semantic equivalence between pairs of questions, constituting a binary classification task.

The dataset consists of 3 files: `train.tsv`, which consists of 363,846 labeled samples, `dev.tsv`, which consists of 40,430 labeled samples, and `test.tsv`, which consists of 390,965 unlabeled samples. Notably, labels in the `train.tsv` and `dev.tsv` files denote the duplicity of the paired questions, whereas the `test.tsv` file lacks such labels.

Obtaining accuracy on the `test.tsv` samples mandates the submission of our predictions to the GLUE leaderboard. However, constraints related to submission limitations and the requirement to submit predictions for all GLUE tasks render this process impractical within our temporal and resource confines. Therefore, we adopt an alternative approach by designating the data in the `dev.tsv` file as our test set on which we benchmark our model performance.

The `train.csv` samples will be split into a training set for model development and a development set for cross-validation with a train-to-dev ratio of 9:1. In light of this, we will exclusively evaluate the performances of the implemented models in an intragroup context instead of comparing with the performances of the models on the GLUE leaderboard. This approach is deliberately adopted to ensure the fairness of our models' evaluations.

3.1 Imbalance in Dataset

As depicted in Figure 1, we observe a slight imbalance in the dataset, with the positive class constituting approximately 37% and the negative class approximately 63%. While this is not a perfect balance, it is not a severe imbalance that would necessitate the implementation of methods like oversampling or undersampling. Oversampling and undersampling techniques are typically used when there is a significant class imbalance, which can lead to a model that is biased towards the majority class. However, in our case, the imbalance is relatively minor and our models were able to learn from both classes effectively.

Moreover, oversampling is particularly challenging with text data. Unlike numerical data, generating additional synthetic text samples that are coherent and meaningful is non-trivial and could introduce noise into the data.

Due to this slight imbalance, we were cautious in our choice of evaluation metrics. While accuracy is a useful metric, it can be misleading in imbalanced datasets as a model could achieve high accuracy by simply predicting the majority class. Therefore, we also considered the F1-score, which takes into account both precision and recall, and provides a more balanced measure of a model's performance on imbalanced datasets. By considering both these metrics, we were able to gain a more comprehensive understanding of our models' performance.

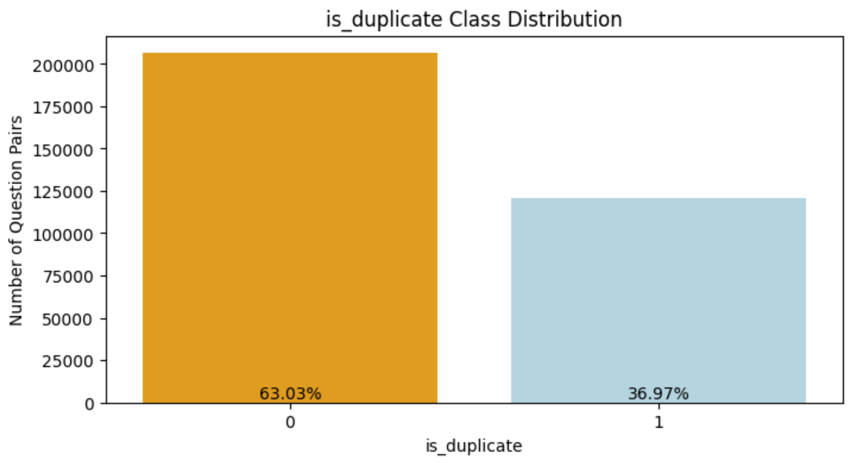


Figure 1: Distribution of target class `is_duplicate`

4 Question Duplication Identification

4.1 Simple Classifiers

In the realm of machine learning, it is often beneficial to start with simple models before progressing to more complex ones. This approach allows us to establish a baseline performance and gain a better understanding of the problem at hand. In this study, we initially employed simple classifiers such as k-Nearest Neighbors (k-NN), Support Vector Machines (SVMs), logistic regression and random forests.

These models, despite their simplicity, are powerful tools that can handle a wide range of classification tasks. They are computationally efficient, easy to interpret, and provide a solid foundation upon which we can build.

4.1.1 Preprocessing

To utilise these models, we first had to transform our text data so that meaningful numerical features can be extracted from it. This involved several steps of preprocessing, such as lower-casing the questions, removing punctuation and non-ASCII characters in the questions, tokenizing the questions, removing stopwords, and lemmatizing the words in each question.

The Natural Language Toolkit (NLTK) library's `word_tokenize()` was used to tokenize each question, allowing for the analysis of each question at the word level to understand its components better. The other preprocessing steps collectively helped in reducing the complexity of the text data. For example, punctuation and non-ASCII characters do not usually contribute to the semantic meaning of a sentence and were identified as noise, hence they were removed. Lemmatization played a significant role in decreasing the size of the corpus vocabulary. By reducing words to their root form, different forms of the same word were combined, which not only simplified the data but also proved beneficial in computing the degree of similarity between two questions based on word frequency.

The impact of these preprocessing steps is further illustrated in the section below detailing the extracted features, underscoring their importance in the task of detecting duplicate questions.

4.1.2 Feature Engineering

After the preprocessing steps outlined above, the following numerical features were extracted from each question pair. These features capture various aspects of the text data and provide a numerical representation that can be processed by our models.

- Number of tokens in `question1`.
- Number of tokens in `question2`.
- Difference in number of characters of both questions.
- Difference in number of tokens of both questions.
- Jaccard similarity of tokens in both questions.
- Jaccard similarity of tokens in both questions with stopwords removed.
- Term Frequency-Inverse Document Frequency (TF-IDF) similarity of both questions.

The difference in number of characters and question length can prove to be a useful feature. This is predicated on the intuitive assumption that a significant discrepancy in length between two questions could suggest they are not duplicates, as one question might encompass more detailed information than the other. However, it is important to note that this assumption, while often useful, is not universally applicable and there may be exceptions where duplicate questions differ substantially in length. Therefore, this feature is used in conjunction with others as described below.

Measuring the Jaccard similarity of tokens in both the questions provides a simple yet effective way to quantify the similarity of two pieces of text, that is through a measure of the proportion of shared tokens. To further enhance the effectiveness of this feature, the stopwords were then removed from the questions before computing the Jaccard similarity. This is to reduce the impact of common and semantically insignificant words, allowing for the computation of a slightly more meaningful metric.

While the Jaccard similarity provides a straightforward measure of overlap between two questions, it does not account for the relative importance of each word in the context of the entire dataset. This is a limitation that motivates the use of TF-IDF similarity. TF-IDF assigns a weight to each word in a question, with higher weights attributed to words that are more unique to the question. This weighting scheme allows for a more nuanced representation of question similarity, where words that are potentially more informative and discriminative exert a greater influence on the similarity score.

In Figure 2, the relationship between the duplicity of the question pairs and the extracted features can be observed. As we progress from character difference, Jaccard similarity to TF-IDF, there are slight increases in correlation. This suggests that these features are increasingly better at predicting whether a question pair is a duplicate.

The difference in number of characters and question length, despite its simplicity, shows a slight negative correlation (-0.2). This supports our initial assumption that a significant discrepancy in length between questions could suggest they are not duplicates.

The Jaccard similarity shows a higher positive correlation (0.34) with the target variable. The correlation is significantly increased (0.4) when stopwords are removed which indicates that the removal of semantically irrelevant words can help the model determine the duplicity of a question pair.

Finally, the TF-IDF similarity feature shows the highest correlation with the target variable, albeit not significantly higher than the correlation of Jaccard similarity without stopwords. This underscores the effectiveness of TF-IDF in capturing the semantic similarity between questions over simpler measures like character length difference and Jaccard similarity.

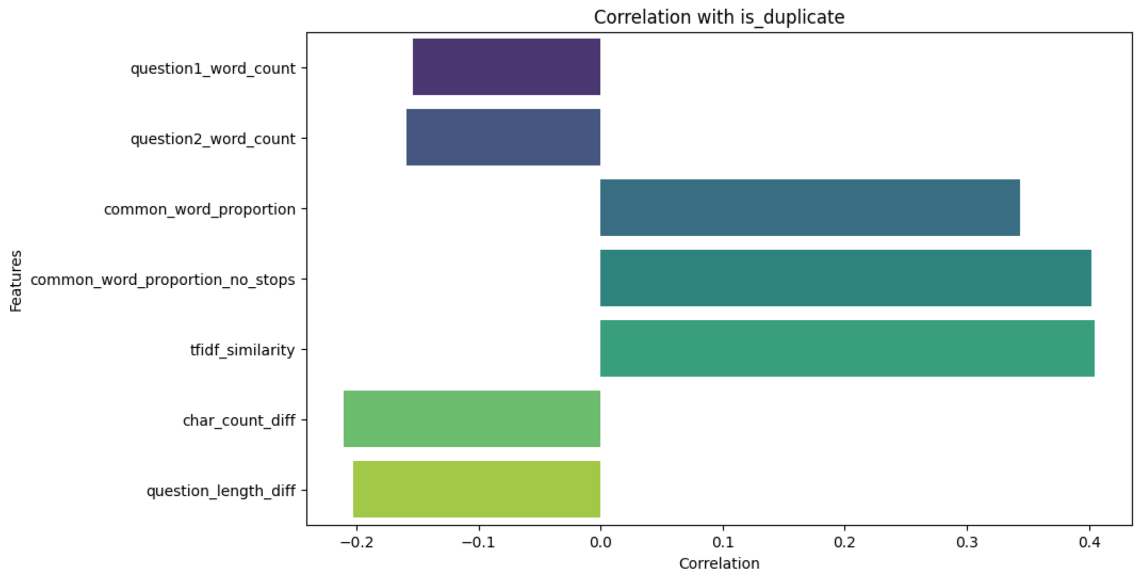


Figure 2: Correlation coefficient barchart between numerical features and `is_duplicate`

4.1.3 Scaling Numerical Features

In the process of model optimization, a noticeable enhancement in the performance of both the k-NN and SVM[8] models (with RBF kernel) was observed upon the application of numerical data scaling using scikit-learn's `StandardScaler`. This improvement can be attributed to the disparity in the range of values between different features. Specifically, the TF-IDF similarity feature, which has a strong correlation with the duplicity of the question pairs, ranges from 0 to 1. On the other hand, the differences in number of characters and question length can take on values significantly larger than 1.

Given that both k-NN and SVMs with RBF kernels rely on distance metrics for their operation, the large numerical differences between these features could potentially skew the models' performance. The SVM with RBF kernel, in particular, uses a measure of similarity between two data points, computed as an exponential function of the Euclidean distance between the data points. The RBF kernel is defined as:

$$K(x_1, x_2) = \exp(-\gamma ||x_1 - x_2||^2)$$

where x_1 and x_2 are two data point, $||x_1 - x_2||^2$ is the squared Euclidean distance between them, and γ is a hyperparameter that controls the width of the Gaussian.

By scaling the data, each feature was given equal importance, thereby allowing the models to better capture the underlying patterns in the data. This hypothesis is supported by the observed improvement in the models' performance post-scaling (3 to 4% increase in accuracy), underscoring the importance of appropriate data preprocessing and feature engineering for this machine learning task.

4.1.4 Results

We employed grid search with cross-validation to select the appropriate hyperparameters for each model. This systematic approach allowed us to explore a range of potential hyperparameters and identify the ones that resulted in the best accuracy. Despite the slight imbalance in the dataset, the hyperparameters were chosen based on accuracy scores of the models. Accuracy is the most straightforward and intuitive metric to gauge the performance of each model. However, it can be misleading if the model predicts the majority class all the time, hence we ensured that the models surpassed this baseline accuracy of 63%, indicating that they were able to learn useful patterns from the data.

For instance, in the k-NN model, the chosen hyperparameters were a neighborhood size of 75 and a distance metric of Euclidean distance. Each point in the neighborhood was also weighted equally in the computation of a datapoint's nearest neighbors. Similarly, for the SVM model with RBF kernel, the chosen hyperparameter was a C value of 1.5. The value of γ was chosen automatically to be $1/d\sigma^2$, where d is the number of features and σ^2 is the variance of the input data X. For the logistic regression model, a C value of 1.5 was chosen with SAGA as its solver. The maximum number of iterations needed for convergence of the logistic regression model's weights was 500. Lastly, the random forest model performed best with 250 estimators, a maximum depth of 20 per tree, a minimum of 5 samples to split an internal node, and a log-loss splitting criterion.

Table 1 below shows the performance of the simple classifiers.

Table 1: Performance of Simple Classifiers

Model	**Train Acc. (%)**	**Validation Acc. (%)**	**Test Acc. (%)**	**Test F1-score**
k-NN	73.4	72.2	71.7	0.62
SVM (RBF Kernel)	71.7	71.6	71.1	0.60
Logistic Regression	66.3	66.1	65.9	0.49
Random Forest	82.4	73.0	72.60	0.64

The results obtained showed that the train and test accuracies were quite close to each other. This small gap between train and test accuracy indicates that there was little overfitting. Overfitting occurs when a model learns the training data too well, to the point where it performs poorly on unseen data. The close train and test accuracies in our models suggest that they have learned to generalize to unseen data, which is a desirable property in machine learning models.

4.1.5 Interpretation

The performance of the simple classifiers (as assessed by the test F1-scores) was found to be sub-optimal, which can be attributed to several factors related to the nature of the features and the inherent complexity of the task.

Firstly, the numerical features extracted, such as Jaccard similarity and TF-IDF, do not capture any notion of ordering in the questions. The order of words in a sentence can significantly alter its meaning, and this is particularly important when dealing with questions. For instance, the question ""Did you see the movie before reading the book?" shares similar words with the question "Did you read the book before seeing the movie?" but they have totally different meanings due to the order of the words.

Secondly, these features do not capture the semantic similarity between words. Words with nearly identical meanings or synonyms are treated as distinct words by these models. This lack of semantic understanding can lead to an underestimation of the similarity between questions. The example from the test dataset, consisting of the question pair “How can I see someone’s private Instagram account?” and “How do I view someone’s private Instagram pictures?”, underscores this limitation of the extracted features. Despite the semantic similarity between the words “view” and “see”, as well as “Instagram account” and “Instagram pictures”, these pairs of words are treated as distinct by the Jaccard and TF-IDF similarity measures. Consequently, these questions have a low Jaccard and TF-IDF similarity, leading them to be predicted as non-duplicates, despite being semantically similar and essentially asking the same thing.

Lastly, the models may struggle with questions where a single word (or a minor part of the question) changes the meaning significantly. For example, consider these questions extracted from the test dataset – "What should I eat for breakfast?" and "What do you eat for breakfast usually?". Despite having a high Jaccard and TF-IDF similarity, these questions have different meanings due to the word "usually". The high similarity measures lead this question pair to being incorrectly predicted as duplicates.

These examples provide an interpretation of the sub-optimal performance of the simple classifiers, as well as highlight the inherent limitations of relying solely on numerical features for text classification tasks. They underscore the necessity for more sophisticated techniques capable of capturing the nuanced semantic relationships present in textual data.

4.2 Neural Networks

In this section, we delve into the exploration of neural networks for the task of detecting duplicate questions. Recognizing the limitations of using solely numerical features for text classification, we turn to neural networks for their ability to capture complex patterns and semantic nuances in text data. We explore techniques for tokenizing questions, specific methodologies for representing individual tokens as vector embeddings, and approaches to aggregate these embeddings for each constituent word to construct a comprehensive representation of each question as a singular vector.

We investigate the use of the Siamese neural network architecture, which specialises in comparison tasks, with Multi-Layer Perceptrons (MLPs) and Long Short-Term Memory (LSTM) networks. We also study the effectiveness of Recurrent Neural Networks (RNNs) and its variants, LSTMs and Gated Recurrent Unit (GRU) that are particularly suited for sequential text data. Lastly, we explore the power of the attention mechanism used in state-of-the-art Transformer architectures, specifically BERT.

We detail the architecture of these networks, the process of training them, and the performance they achieve on our task. This section serves to highlight the potential of these advanced models in improving the performance of duplicate question detection. It underscores our systematic approach of progressing from simpler to more complex models, each building upon the insights gained from the previous, in tackling this challenging task.

4.2.1 Siamese Networks

We adopted a different approach to preprocessing the questions for its use with the Siamese networks. The preprocessing stage only involved lower-casing the questions, removing punctuation, and tokenizing the questions. We consciously chose not to apply lemmatization or remove stopwords to preserve the semantics of each question. Lemmatization and stopword removal can sometimes oversimplify the text and strip away nuances that could be important for understanding the meaning of the questions. We employed the use of Natural Language Toolkit's (NLTK's) `word_tokenize()` or the split() function, as they provided a simple and effective way to break down text into individual words, which was suitable for use with pretrained Word2Vec embeddings. The use of pretrained Word2Vec embeddings is elaborated and justified in Appendix 7.

The Siamese network architecture, as depicted in Figure 3, is an architecture specialised for comparing and measuring similarity between two input samples. In the context of question duplication detection, the inputs to the network are the embeddings for the questions in each question pair. The Siamese network utilises an identical (shared) network to process each input question separately, producing a fixed-size vector representation (encoding) for both questions. The encodings of both questions are subsequently used to compute a dissimilarity measure between the questions. A common approach is to use a distance metric,

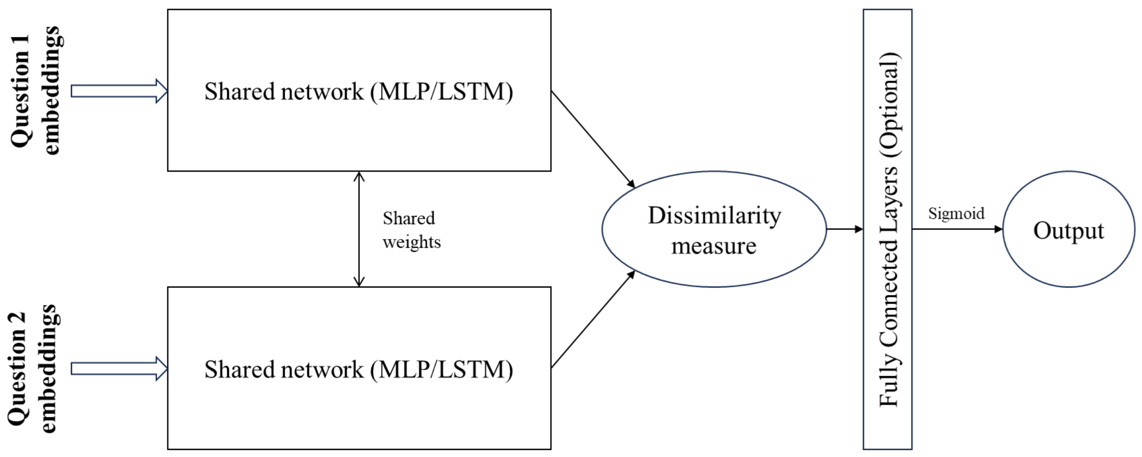


Figure 3: Siamese network architecture

such as Euclidean distance (L2-distance) or Manhattan distance (L1-distance) to measure the dissimilarity between the encodings. We employ the use of the Manhattan distance for our Siamese network.

The computed dissimilarity is then either interpreted directly as the final prediction or passed through a fully connected network which outputs the duplicity of the questions in the question pair. Hence, a Siamese network can be thought of as learning a mapping from the input space (individual questions) to a vector space where the vector representations for semantically similar questions exhibit a dissimilarity measure that is discernible from that of semantically different questions.

4.2.1.1 Siamese Multi-Layer Perceptrons (MLPs)

The first shared network we explored was the MLP. As briefly mentioned above, the embedding generation process involved the utilization of Word2Vec embeddings for individual tokens within a question. Subsequently, a straightforward aggregation technique was employed, entailing the summation of embeddings in an element-wise manner across each token. This summative approach served as a means to consolidate the contextual information encoded in the individual embeddings, facilitating a somewhat comprehensive (but less sophisticated) representation of the entire question. The single “sum-aggregated” embedding is the input the Siamese MLP network. The network consists of:

- **Input Layer:** Two inputs, corresponding to tokenized and embedded question pairs, with dimensions of 300 matching the Word2Vec embeddings.
- **Hidden Layers:** Two layers, each with 512 units and a Rectified Linear Unit (ReLU) activation functions, resulting in two 512-dimensional vectors representing the encoded values of each question.

The dissimilarity between the questions’ encodings is measured using the L1-distance. The final prediction is determined by applying a threshold to the computed distance.

To determine the model’s parameters, a grid search with a validation set was employed. Some noteable observations were made. Firstly, models trained with a large learning rate (e.g., Keras library’s default of 0.01) failed to converge. We postulate that the elevated learning rate led to the occurrence of “overshooting,” causing the model to surpass the minimum of the loss function.

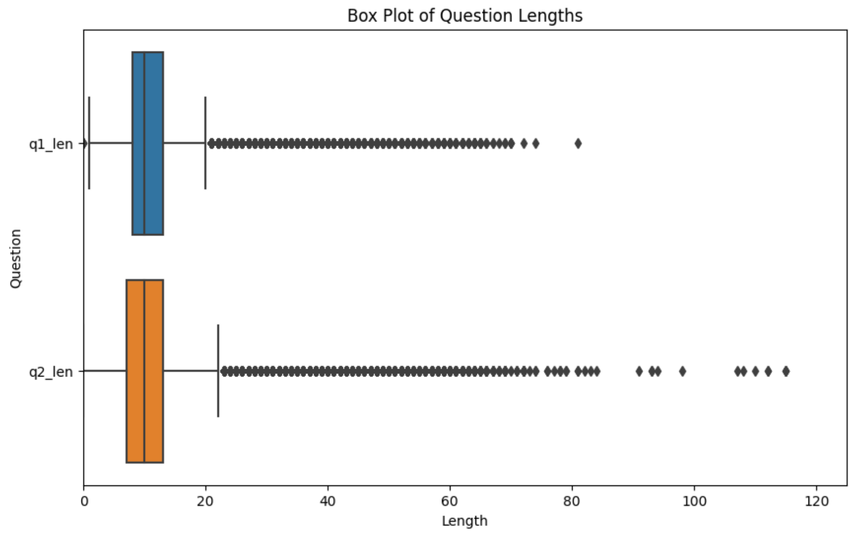


Figure 4: Distribution of question lengths in the training dataset

Secondly, following an extensive grid search, numerous candidate models emerged with approximately comparable validation accuracy. The selection of the aforementioned model was grounded in two key considerations. Primarily, it boasted the highest validation accuracy among the contenders. Additionally, its comparative simplicity, in contrast to other candidate models, contributed to the decision. This dual rationale instilled confidence in the model's potential for generalization to the test set.

4.2.1.2 Siamese LSTM

We transition to the use of a neural network suited for processing sequential data like text, that is an RNN variant, the LSTM network. The LSTM is used as the underlying shared network in the Siamese architecture. It is rather well-suited for the task of question duplication detection due to its ability to capture long-term dependencies in text, which is crucial for understanding the semantic similarity between questions.

We defined a maximum sequence length of 30 based on the observed distribution of question lengths in the dataset as depicted in Figure 4. The need for a maximum sequence length arises from the requirement to feed fixed-length sequences into the LSTM. Using the Tukey outlier[9] method, a length of 22 was considered as an outlier in the dataset, hence a sequence length of 30 seemed most appropriate. This choice ensures that most questions are represented fully while keeping the computational complexity manageable.

Tokens that were part of the Google News Word2Vec vocabulary were assigned the pretrained Word2Vec embeddings while tokens that were out-of-vocabulary (OOV) were assigned a vector embedding of zeros. It is also important to note that having a max sequence length of 30 meant that questions longer than 30 tokens were truncated, which is a trade-off we accepted for computational efficiency.

The resulting network's architecture is described as follows:

- **LSTM (Shared network):** The LSTM consisted of 2 layers, each with a hidden size of 300 neurons.

- **Absolute Element-wise Difference Layer**: The absolute element-wise difference between the 300-dimensional encodings generated for each question was computed and passed through a fully-connected network.
- **Fully-connected Network:** The FCN consisted of 4 layers, each with 300 neurons, ReLU activations, and a sigmoid activation for the final prediction.

The learning rate was chosen to be 0.001, a value that was found to provide a good balance between training speed and convergence stability. The selection of these hyperparameters was guided by a process of early stopping based on validation loss. This approach involved monitoring the model's performance on a separate validation set during training. The training process was halted when the validation loss ceased to improve, ensuring that the chosen hyperparameters resulted in optimal model performance and prevented overfitting to the training data.

4.2.2 RNN Variants

Next, we employ Recurrent Neural Network (RNN) variants, specifically RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) architectures. To preprocess the textual data, we lowercase and tokenize sentences into words using the Natural Language Toolkit (nltk) library. The questions are then concatenated with the `<>SEP</>` token serving as a separator, and `<>UNK</>` is used to encode unknown words. Each word is mapped to a unique identifier, and an embedding layer is utilized to map words to a lower-dimensional representation.

Subsequently, the embedded sequences are passed through RNN, LSTM, or GRU layers to capture sequential dependencies. A fully connected layer processes the output, and a sigmoid activation function is applied for binary classification. This architecture aims to discern the similarity between two questions.

Hyperparameter tuning is conducted with early stopping to optimize model performance. Hyperparameters such as the number of hidden neurons, the count of RNN and linear layers, the number of training epochs, and the learning rate are chosen via cross-validation based on the validation accuracy.

We also tried to use pre-trained fastText word embedding weights for the embedding layer instead of learning it during the training process. However, this does not yield significant improvement due to a relatively small vocabulary compared to the number of training samples available.

4.2.3 BERT-Family Models

The main mechanism of BERT, Bidirectional Encoder Representations from Transformers, is its self-attention and bidirectional properties.

Intuitively, self-attention works like a lookup table. Three weights are trained and multiplied to the input, giving 3 matrices – key (K), query (Q) and value (V). Each key has a score to some query and a large dot product value between the key and query vector gives higher activation. In the context of NLP, each word in a sentence is the key and we look at the dependencies between words. Below is the equation for attention:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

We attribute the higher performance of BERT to the fact that it is bidirectional, incorporating context from both directions, unlike RNN which only stores information from the left of the current input.

HuggingFace's pretrained tokenizer `BertTokenizer` and classification model `BertForSequenceClassification` were used with `bert-base-uncased` consisting of

110M parameters. `BertTokenizer` is a subword-based tokenizer that splits unknown words into smaller words or characters like stemming, thus handles rare and out-of-vocabulary words well, reducing the need to treat them as unknown tokens. `BertForSequenceClassification` implements the self-attention mechanism with the following hyperparameters:

- Number of layers: 12
- Hidden size: 768
- Number of attention heads: 12

The model consists of skip connection which helps with the vanishing gradient problem. A Gaussian Error Linear Unit (GeLU) activation is also used, which is an activation function smoother than the ReLU, helping with optimization and convergence. A pretrained model was used but fine-tuning significantly improved the accuracy from 51.1% to 90.1%.

4.3 Results

Table 2 shows the performance of each model used in this classification task.

Table 2: Performance of Classification Models

Model	**Train Acc. (%)**	**Validation Acc. (%)**	**Test Acc. (%)**	**Test F1-score**
Simple Classifier (RF)	82.4	73.0	72.60	0.64
Siamese MLP	91.8	82.5	82.2	0.76
Siamese LSTM	94.0	84.5	84.1	0.79
RNN	82.3	82.3	82.3	0.75
LSTM	84.8	84.5	84.5	0.79
GRU	85.9	85.9	85.9	0.80
BERT	**92.0**	**90.4**	**90.1**	**0.87**

Simple Classifiers: While the random forest model achieved the highest accuracy among the simple classifiers, its overall accuracy is notably lower compared to more sophisticated neural network models. This underscores the necessity for complex architectures.

Siamese Networks: The Siamese LSTM outperforms the Siamese MLP in both validation and test accuracies, emphasizing the advantage of employing LSTM for capturing intricate relationships between question pairs.

RNN Variants: The performance progression from RNN to LSTM and GRU highlights the challenges of traditional RNNs in retaining information over long sequences. The gated mechanisms in LSTM and GRU effectively address this issue, with GRU surpassing LSTM in both accuracy and F1-score, indicating its efficacy for the task.

BERT: BERT emerges as the top-performing model, showcasing exceptional capabilities in understanding and contextualizing textual data. The substantial performance gap between BERT and other models underscores the transformative impact of pre-trained language models on complex natural language processing tasks. BERT's bidirectional nature and attention mechanisms allow it to capture nuanced dependencies, providing a significant advantage over traditional and sophisticated neural network architectures.

These observations collectively highlight the importance of leveraging advanced neural network architectures and pre-trained language models for achieving state-of-the-art results in tasks requiring a deep understanding of language semantics.

5 Generative Language Modelling

5.1 Overview

The objective of the Generative Language Modeling component within this project is to create a language model capable of generating novel questions that 1) advances the boundaries of knowledge exploration, and 2) contributes to the augmentation of the existing training set for the Question Duplication Detection task, thereby enhancing the overall robustness and diversity of the dataset.

Generative Language Modeling is a task that involves predicting the next token in a sequence based on the context provided by the preceding tokens. In our exploration, the primary performance metric for assessing the effectiveness of the language models is **perplexity** (PP). Perplexity quantifies how well the model predicts a sequence of tokens and is calculated using the formula:

$$PP(W) = \left(\prod_{i=1}^{N} P(w_i)\right)^{-\frac{1}{N}} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\ln\left(P(w_i)\right)\right)$$

We concatenate questions in the `train.tsv` and `test.tsv` files and split them to train, validation, and test set with a train:val:test ratio of 9:1:1. During the development phase, perplexity on the validation set is used for cross-validation.

The encoding approach adopted for tokenization is byte-pair encoding, implemented using Google's `sentencepiece`[10] with vocabulary size of 8000. We explore various model architectures, including n-grams, Multi-Layer Perceptrons (MLP), Recurrent Neural Networks (RNN), and transformer models. The models undergo hyperparameter tuning based on cross-validation using validation perplexity.

For generation, at each step, the models will calculate the probability of occurrence of each token based on previous tokens (context). The next token will be sampled from the vocabulary with the calculated probability and concatenated to the previous tokens to form the new context.

5.2 Models

N-gram Model: The n-gram model[11] divides text into series of adjacent tokens of length n. During the training step, it constructs a table of n-gram counts, and the next token will be sampled with probabilities calculated based on these counts Cross-validation is employed to select the optimal n value. Laplace smoothing[12] is applied to handle unknown tokens when calculating perplexity, ensuring robust performance in the presence of unseen vocabulary.

MLP and RNN Models: Both Multi-Layer Perceptron (MLP)[13] and Recurrent Neural Network (RNN) architectures are explored for generative language modeling. MLPs provide a non-linear mapping between input and output, capturing intricate relationships within the sequential data. RNNs leverage sequential information to model dependencies across tokens, allowing for the capture of contextual nuances in token prediction.

Transformer Model: The Transformer model, known for its self-attention mechanisms, is incorporated for generative language modeling. Causal attention is utilized, restricting the model to consider only tokens to the left when predicting the next token. Due to the intensive computational requirements, the model is trained with a random set of hyperparameters without cross-validation.

5.3 Results

Table 3: Performance of Generative Language Models

Model	Test Perplexity
N-gram (bi-gram)	76.83
MLP	**72.07**
RNN	107.13
Transformer	124.58

The performance of the generative language models is assessed based on test perplexity, as shown in Table 3. The following observations provide insights into the effectiveness of each model:

The Multi-Layer Perceptron (MLP) emerges as the best-performing model, exhibiting the smallest test perplexity of 72.07. This result highlights the effectiveness of MLPs in capturing complex patterns within the dataset, showcasing their capability in generative language modeling.

Despite its simplicity, the n-gram model, specifically with bi-grams, demonstrates remarkable performance with a test perplexity of 76.83. This can be attributed to the dataset's characteristics, which is relatively small, and the question samples exhibit similarities. The simplicity of the n-gram model proves effective in this context.

The Transformer model yields a test perplexity of 124.58. It's important to note that the performance of the Transformer may improve with cross-validation. However, the current result underscores that the Transformer excels in scenarios with large and diverse datasets, while the dataset in this context is comparatively smaller and less diverse.

Samples of generated questions can be found in Appendix 7.

6 Extensions and Future Works

Having successfully developed both the Question Duplication Detection and Generative Question Modeling models with robust performance, our project lays a solid foundation for further exploration and enhancement. Despite our achievements, several avenues remain unexplored due to time and resource constraints. The following outlines potential extensions and future works that could amplify the impact and capabilities of our models:

6.1 Model Diversification and Error Analysis

While our current models exhibit strong performance, there is room for further exploration in terms of model diversity. Future endeavors could involve experimenting with additional models beyond those initially explored. Furthermore, a more comprehensive error analysis could unveil specific question pairs that pose challenges to our models (for example, non-English questions, questions with a lot of emojis, etc.). This nuanced understanding would facilitate targeted improvements and refinement of the models.

6.2 Hyperparameter Tuning for Large Models

The deployment of larger and more sophisticated models, such as those from the BERT family for the Question Duplication Detection task and transformer models for the Generative Question Modeling task, holds the potential for substantial performance gains. Due to the time and compute resource constraints, we only use default recommended hyperparameters for those models. Conducting extensive hyperparameter tuning on these advanced models could uncover optimal configurations, pushing the boundaries of model capabilities and achieving even higher accuracy.

6.3 Utilising Generated Questions for Training Set Augmentation

An intriguing avenue for future work involves leveraging the questions generated during the second phase of our project to augment the training set. Specifically, augmenting the dataset with the questions generated could contribute to addressing imbalances, particularly by increasing the number of duplicate question pairs in the original dataset. This augmentation strategy holds promise for enhancing the model's ability to handle minority sets and further improving performance on challenging instances.

6.4 Integration of External Knowledge Sources

Expanding the scope of our models by integrating external knowledge sources could enhance their contextual understanding. Incorporating domain-specific knowledge bases or leveraging pretrained embeddings from external sources might enable our models to generate more informed and contextually relevant questions.

6.5 Prompt Engineering

An avenue we managed to explore in the spirit of fun and curiosity is in experimenting with OpenAI's `gpt-3.5-turbo` API for the duplication task. By applying simple prompt engineering techniques, we asked the model to determine if two questions in each question pair are duplicates. This approach mirrors the trend in the modern startup industry, where many AI products are built primarily on OpenAI's GPT API. The philosophical significance of this approach lies in its reflection of the democratization of AI, where powerful models are made accessible to startups and individual developers alike. Our decision to test this approach was driven by a desire to mimic these startups, providing us with valuable insights into the practical applications of these advanced models.

Despite the potential of the OpenAI API, it did not perform particularly well on our task, achieving significantly lower performance (close to 71% accuracy and an F1-score of 0.51) than the neural networks we implemented. This could be attributed to several factors, one of which is the quality of prompt engineering, which is a crucial aspect of working with such models. In our case, more sophisticated prompt engineering could potentially have improved the performance of the API. This could involve experimenting with different ways of phrasing the prompts or providing more context to the model.

7 Conclusion

In this project, we presented a comprehensive exploration into the realms of Question Duplication Detection and Generative Question Modeling within the context of the Quora ecosystem. Our dual-pronged approach aimed to advance the frontiers of knowledge-seeking by not only discerning duplicate questions but also by generatively modeling novel inquiries. Through rigorous experimentation and model development, we achieved robust performance in both tasks, providing valuable contributions to the field of natural language understanding.

Acknowledgments

We express our sincere gratitude to Professors Brian Lim and Xavier Bresson for providing us with the invaluable opportunity to work on this project during the semester. Their teachings have been instrumental in shaping the direction of our project, and we are truly appreciative of the machine learning knowledge they shared. Additionally, we extend our thanks to the CS3244 Teaching Assistants for their assistance and support throughout the course. Special appreciation goes to Mario Michelessa for generously sponsoring $50 worth of Google Cloud credits, which contributed to meeting our computing requirements. Furthermore, we would like to thank TA Lin Xiaoqiang for his constructive feedback, which played a crucial role in refining the quality of our work. These collective contributions have greatly enriched our project experience and have been integral to the success of this project.

References

[1] Mikolov, T., Chen, K., Corrado, G.S., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *International Conference on Learning Representations*.

[2] Jeffrey Pennington, Richard Socher, and Christopher Manning. (2014). GloVe: Global Vectors for Word Representation. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[3] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics, 5*. pages 135-146.

[4] Bromley, J., Bentz, J.W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Säckinger, E. and Shah, R. (1993). Signature Verification using a "Siamese" Time Delay Neural Network. *International Journal of Pattern Recognition and Artificial Intelligence*, 07(04), pp. 669–688. American Telephone and Telegraph Company.

[5] Y. Taigman, M. Yang, M. Ranzato and L. Wolf (2009). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.

[6] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2017). Attention is All you Need. *Neural Information Processing Systems*.

[7] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *North American Chapter of the Association for Computational Linguistics*.

[8] Cortes, C., Vapnik, V. Support-vector networks. *Mach Learn 20*, pp. 273–297 (1995). https://doi.org/10.1007/BF00994018

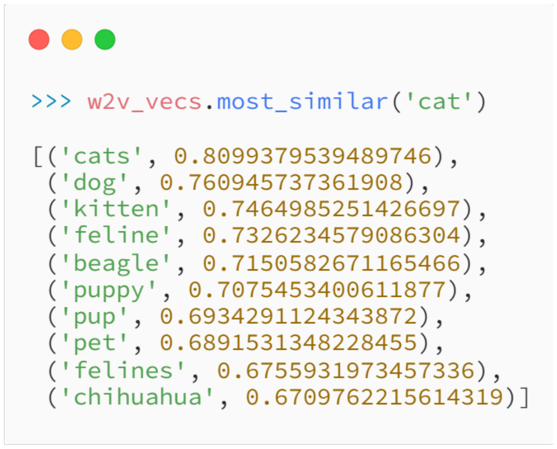
[9] Hoaglin, D. C., Iglewicz, B., and Tukey, J. W. (1986). Performance of Some Resistant Rules for Outlier Labeling. *Journal of the American Statistical Association*, 81(396), 991–999. https://doi.org/10.1080/01621459.1986.10478363

[10] Kudo, T., and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for *Neural Text Processing. Conference on Empirical Methods in Natural Language Processing*.

[11] Jurafsky, Daniel; Martin, James H. (2023). *Speech and Language Processing* (3rd ed.). Prentice Hall. pp. 31-57. ISBN 978-0-13-187321-6.

[12] C.D. Manning, P. Raghavan and H. Schütze (2008). *Introduction to Information Retrieval. Cambridge University Press*, p. 260.

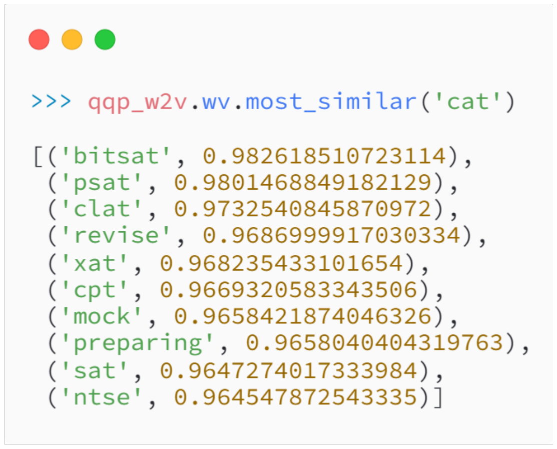
[13] Bengio, Y., Ducharme, Réjean, and Vincent, Pascal. (2000). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*.



```
>>> w2v_vecs.most_similar('cat')

[('cats', 0.8099379539489746),
 ('dog', 0.760945737361908),
 ('kitten', 0.7464985251426697),
 ('feline', 0.7326234579086304),
 ('beagle', 0.7150582671165466),
 ('puppy', 0.7075453400611877),
 ('pup', 0.6934291124343872),
 ('pet', 0.6891531348228455),
 ('felines', 0.6755931973457336),
 ('chihuahua', 0.6709762215614319)]
```

(a) Similar words to 'cat' in pretrained embeddings



```
>>> qqp_w2v.wv.most_similar('cat')

[('bitsat', 0.982618510723114),
 ('psat', 0.9801468849182129),
 ('clat', 0.9732540845870972),
 ('revise', 0.9686999917030334),
 ('xat', 0.968235433101654),
 ('cpt', 0.9669320583343506),
 ('mock', 0.9658421874046326),
 ('preparing', 0.9658040404319763),
 ('sat', 0.9647274017333984),
 ('ntse', 0.964547872543335)]
```

(b) Similar words to 'cat' in QQP-trained embeddings

Figure 5: Comparison of pretrained Word2Vec and QQP-trained embeddings

Appendix

Word Embeddings (Word2Vec)

To facilitate the training of the Siamese networks, we leveraged the use of Word2Vec embeddings trained on Google News data. (`word2vec-google-news-300`) A 300-dimensional vector embedding was extracted for each token within a tokenized question. Consequently, each question could be effectively represented as an $l \times 300$ tensor, where l corresponds to the length of the question, or the maximum sequence length predetermined by our specifications.

Our decision to use pretrained Word2Vec embeddings instead of training a Word2Vec model on our dataset was driven by the limited vocabulary of our dataset (around $75,000$ words) compared to the pretrained vectors (around $3,000,000$ words). We believed this would be more appropriate for generalizing to unknown tokens which may appear in new questions.

The pretrained Word2Vec embeddings also address an issue of semantic similarity between words, as previously described. A simple demonstration of the mentioned improvement is depicted in Figure 5. When we use the Word2Vec `most_similar()` method on the word 'cat', the pretrained Word2Vec embeddings (`w2v_vecs`) return words that are actually similar to 'cat', demonstrating the richness of these embeddings. In contrast, the Word2Vec model (`qqp_w2v`), trained on our dataset, does not return any relevant words. This underscores the usefulness of the pretrained embeddings for this task and their ability to capture semantic nuances in the text data.

Samples of Generated Questions

"Can you increase your age on YouTube?"

"How important are suggestions in depression?"

"What are Best computer science engineering student have before graduating?"

"Why do so?"

“What is your review of time?”

“How will you stop reading election?”

“Does Donald Trump lose weight?”

“Is plug of poissh is mostin Chuy River than girls?”

“Which is a ways at them?”

“What are some disadvantages of three points according to an SCE 75% for indo from birth, the military, FP?”

“What is the originating pump and anti-16?”

“Why can you should I download movies that will be discontinued? What are blue shirts on Quora I study online java?”

“Why did the egg glitch?”

“Which is the future?”

“What are some year?”

“How long does it becomes president?”

“How does one know how to make cut off someone on Quora need improvement?”

“What are some of the best city in India?”

“What is funniest joke you have resonance’ used in which ecole-6 cups that can easily find the publisher using the Borgi mix from biting and Moriarty will you eat eggs? What are the job interview?”