INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

PROJECT REPORT

Natural Language Understanding with the Quora Question Pairs Dataset

COURSE CODE: STATISTICAL METHODS IN AI - CS7.403.M21

Advisor:
Prof. Anoop Namboodiri

Team Name:
Abraca-data

Project Mentor:
Adithya Jain

Project Representatives:
Aditya Kumar Singh - 2021701010
Dhaval Taunk - 2021701028
Dhruv Srivastava - 2021701021
Lakshya Khanna - 2021900004

Academic year:
2021-2022

Abstract: Our goal for this project was to determine whether two Quora questions are similar or not using various traditional ML and DL models with an extensive feature engineering. Till now, we've extracted **TF-IDF** and **N-Grams** features from pairs of questions and used them for training different variants of **Logistic Regression** as well as **SVM**. We tried to replicate the results as mentioned in the assigned paper and analysed the optimal hyperparameters found through **GridSearch Cross-validation**. Further, manual features are generated for **Tree-based** models upon which we'll be training our models like **Decision and Random Forest** in future. So far we're testing with ML models which, according to paper, serves as baseline models for the DL models like **CBOW**, **GloVe**, and **LSTM**. We expect to cover up all the model performances, comparison, analysis as mentioned in the paper along with a final touch on **Transformer** based models in our final evaluation report.

Contents

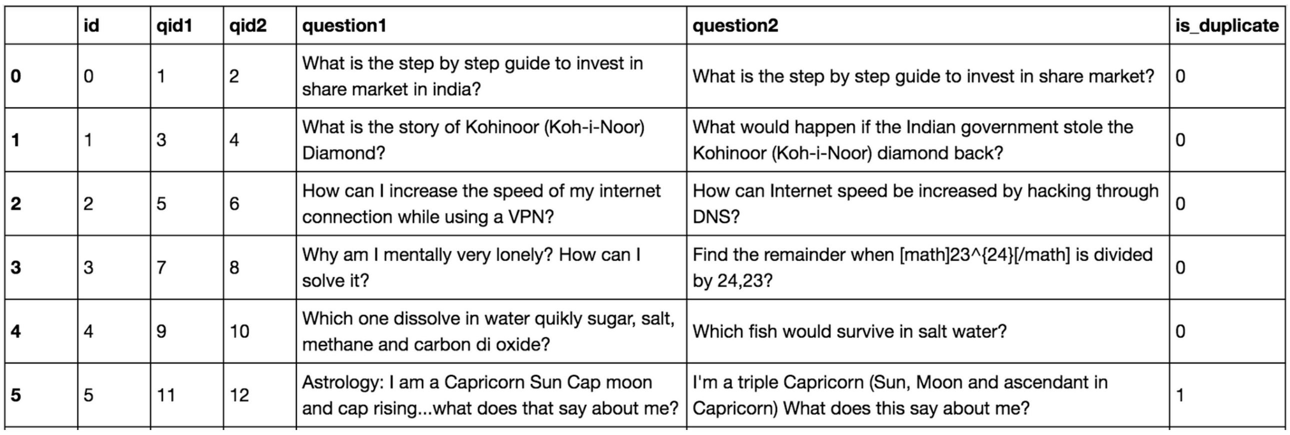
1. Project Description

The aim is to determine whether the two questions are duplicates of each other, i.e., whether they reflect the same meaning or not, using the Quora question pair dataset. As a result, this task is essentially a binary classification problem, with a 0/1 response dependent on whether or not the questions are comparable. This project is partially a replication of the cited paper [1].

Code link:- `https://github.com/DhavalTaunk08/smai_project`

2. Data Overview:

The dataset that is presently accessible has been found to be substantially unbalanced. 255,027 (63.08 %) of the 404,290 question pairings have a negative (0) label, while 149,263 (36.92 %) have a positive (1) label. The question pairs, their corresponding ids, sample id, and the accompanying label are shown in the sample from the dataset below.



	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
2	2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24}[/math] is divided by 24,23?	0
4	4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0
5	5	11	12	Astrology: I am a Capricorn Sun Cap moon and cap rising...what does that say about me?	I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me?	1

Figure 1: A sample from the available dataset

The dataset is being splitted into 3 sections namely train, val and test with a ratio of 70:20:10 in the similar way as mentioned by the authors.

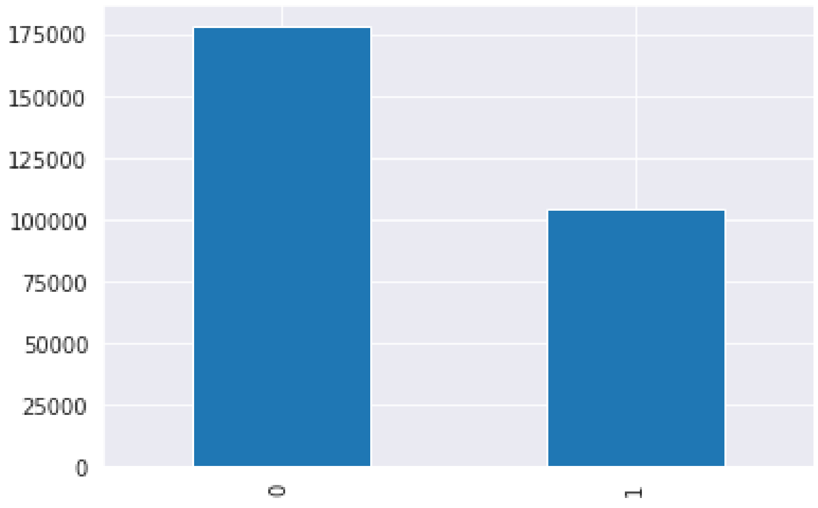
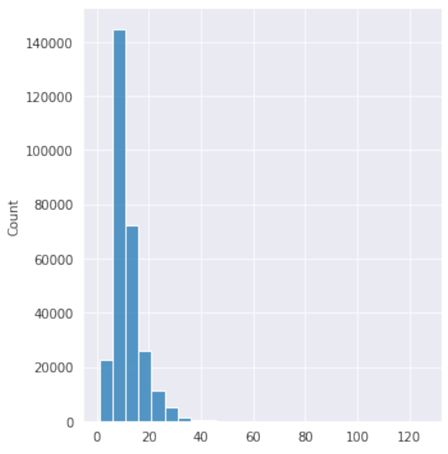
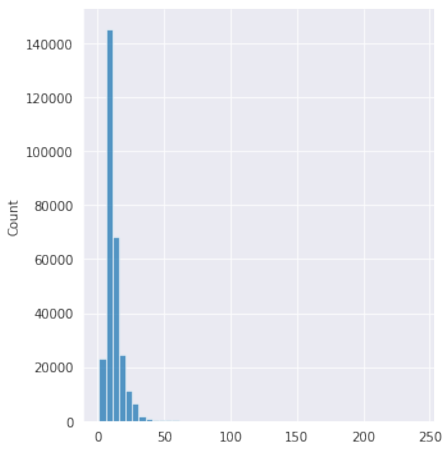


Figure 2: Class Distribution

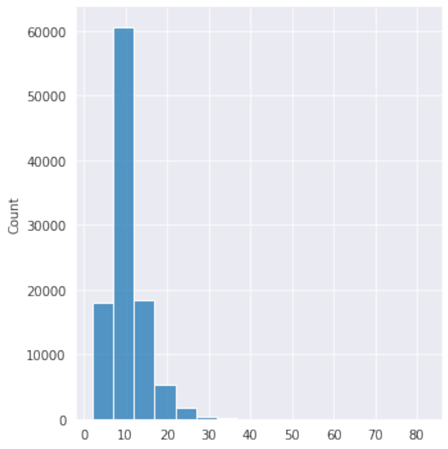


(a) Frequency of Question-1 Lengths

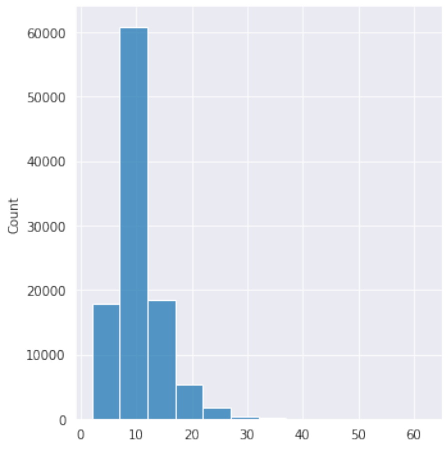


(b) Frequency of Question-2 Lengths

Figure 3: Frequency of Question Lengths

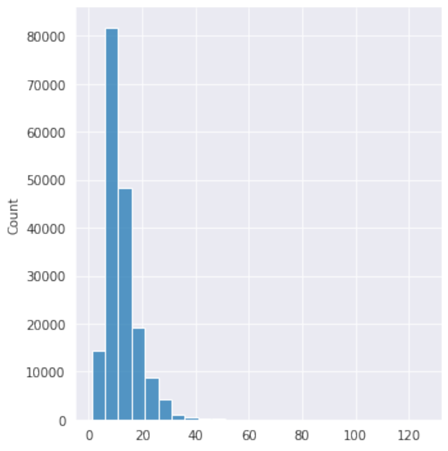


(a) Frequency of Duplicate Question-1

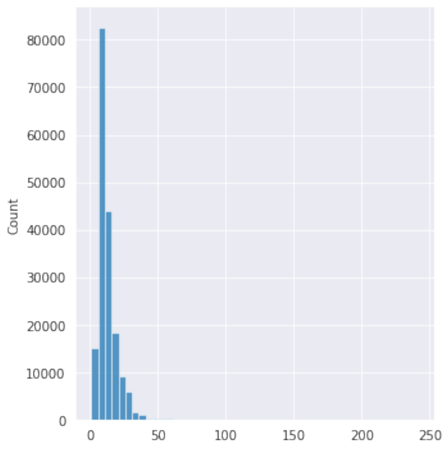


(b) Frequency of Duplicate Question-2

Figure 4: Frequency of Duplicate Questions



(a) Frequency of Non-Duplicate Question-1



(b) Frequency of Non-Duplicate Question-2

Figure 5: Frequency of Non-Duplicate Questions

3. Linear Models

3.1. Feature engineering

As mentioned in the given research paper, Linear models are trained on N-gram features (Unigrams, Bigrams and Trigrams).
An N-gram is a series of N words in natural language processing. For example, "Statistical" (n=1) is a **unigram**, "Methods in" (n=2) is a **bigram**, and "Artificial Intelligence Course" (n=3) is a **trigram**. The length of each feature vector is **128** hbf and is represented as a sparse-matrix.

3.2. Training Logistic Regression on N-gram features

3.2.1 Introduction

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative distribution function of logistic distribution.

3.2.2 Replicating Previous Work & Hyper-Parameter Tuning

According to the paper, three Logistic Regression (LR) model, namely

1. LR with Unigrams (Using Unigrams Features)
2. LR with Bigrams (Using Bigrams Features)
3. LR with Trigrams (Using Trigrams Features)

were trained each with

1. $L2$ regularization,
2. controlled by α,
3. trained with stochastic gradient descent (`SGDClassifier`) using scikit-learn's implementation (Pedregosa et al., 2011), and
4. an 'Optimal' learning rate $\eta = \dfrac{1}{\alpha(t + t_0)}$

Best parameters, as given in paper, is obtained with $\alpha = 0.00001$ **and** $n\_iter = 20$ **(i.e., iterations)** for **LR Unigram**.

To implement **LR** with **SGD** as optimizer with F_1 and *Accuracy* as our evaluation metric, we took help of `scikit-learn` library.

1. `sklearn.linear_model.SGDClassifier`
2. `sklearn.metrics.accuracy_score, f1_score`

Further, to search for *optimal* parameters we applied `GridSearch` along with 5-Fold `StratifiedCrossValidation` using following packages.

1. `sklearn.model_selection.GridSearchCV`
2. `sklearn.model_selection.StratifiedKFold`

Hyperparameters considered while implementing `GridSearchCV` are as follows (with the best one mentioned just beside it):

1. For `unigrams`: $\alpha = 0.001$
2. For `bigrams`: $\alpha = 0.0001$
3. For `trigrams`: $\alpha = 0.0001$; `tol` $= 0.001$

3.2.3 Results (LR)

N-Gram features	**Original Results**		**Logistic Regression (with SGD)**	
	Accuracy	**F_1 Score**	**Accuracy**	**F_1 Score**
Unigrams	75.4%	63.8%	68.1%	42.6%
Bigrams	79.5%	70.6%	66.9%	42.3%
Trigrams	80.8%	71.8%	64.9%	29.5%

Table 1: Replication Results

N-Gram features	**Original Results**		**Logistic Regression (with SGD)**	
	Accuracy	**F_1 Score**	**Accuracy**	**F_1 Score**
Unigrams	75.4%	63.8%	68.9%	44.4%
Bigrams	79.5%	70.6%	66.7%	36.2%
Trigrams	80.8%	71.8%	65.1%	31.9%

Table 2: Stratified 5-fold Cross-Validation Results with Hyper-Parameter Tuning

3.3. Training Support Vector Machine (SVM) on N-gram features

3.3.1 Introduction

The "Support Vector Machine" (SVM) is a supervised machine learning technique that can solve classification and regression problems. It is, however, mostly employed to solve categorization difficulties. Each data item is plotted as a point in n-dimensional space (where n is the number of features you have), with the value of each feature being the value of a certain coordinate in the SVM algorithm. Then we accomplish classification by locating the hyper-plane that clearly distinguishes the two classes (look at the below snapshot). More briefly this algorithm finds a hyperplane in an N-dimensional space that distinctly classifies the data points with maximum margin, i.e the maximum distance between data points of both classes.

3.3.2 Work done

Our base intention was to replicate the results mentioned in the given research-paper and hence a SVM model with same parameters was designed and tested.
It was observed that SVM with linear decision boundary can be implemented in two ways-

1. `sklearn.svm.LinearSVC`
2. `sklearn.svm.SVC` with `kernel=linear`

The given research paper had used `LinearSVC` and we were able to replicate the results and **even improved them for each of them**.

3.3.3 Results (Linear SVM)

Features used	**Original Results**		**Results from our implementation**	
	Accuracy	**F_1 Score**	**Accuracy**	**F_1 Score**
Unigrams	64.2%	-	**68.73% $\upuparrows$ 4.53%**	0.43134
Bigrams	65.1%	-	**66.59% $\upuparrows$ 1.49%**	0.3242%
Trigrams	65.9 %	-	**65.06%** $\downdownarrows$ 0.84%	0.2481%

Table 3: Stratified 5-fold Cross-Validation Results with Hyper-Parameter Tuning

4. Tree based Models

Tree-based is a family of **supervised Machine Learning** algorithms which performs classification and regression tasks by building a tree-like structure for deciding the target variable class or value according to the features. It is one of the popular Machine Learning algorithms used in predicting tabular and spatial/GIS datasets. Like **SVM** or **Logistic Regression**, these models approach problems by deconstructing them piece-by-piece through vertical and horizontal lines to minimize entropy, instead of finding one complex boundary that can separate the entire dataset.

Tree-based methods are **deterministic**, opposed to being probabilistic as in for Neural Network based models due to which Decision trees explicitly fit parameters to **direct the information flow** while Neural networks fit parameters to transform the input and **indirectly direct the activations** of following neurons. Conditional nodes that are activated in decision trees are analogous to neurons being activated (information flow). In general, tree-based methods are essentially simplified versions of neural networks.

In our case we'll be using **three** different tree-based models on dataset that is specially engineered for tree-based models. And we'll compare them upon their performance and analyse their results along with their hyper-parameter tuning.

1. Decision Tree Classifier
2. Random Forest Classifier
3. Gradient Boositng Classifier

4.1. Feature Engineering

For creating features for tree based models, we employed the same feature engineering methods as described by the authors. A brief overview is given below:

1. (L) Length based: length for question 1:- l1, and question 2:- l2, difference in length:- ($l1$ - $l2$), and ratio of lengths:- $\frac{l1}{l2}$.
2. (LC) Number of common lower-cased words: count, count / length of longest sentence.
3. (LCXS) Number of common lower-cased words, excluding stop-words: count, count / length of longest sentence.
4. (LW) Same last word.
5. (CAP) Number of common capitalized words: count, count / length of longest sentence.
6. (PRE) Number of common prefixes, for prefixes of length 3–6: count, count / length of longest sentence.
7. (PRE) Number of common prefixes, for prefixes of length 3–6: count, count / length of longest sentence.
8. (M) Misc: whether questions 1, 2, and both contain "not", both contain the same digit, and number of common lower-cased words after stemming, number of common lower-cased words after stemming / length of longest sentence.

These above features create a feature vector of size 25 for all the samples.

4.2. Decision Tree

It's a tree where each internal node represents an attribute, a branch represents a decision rule, and the leaf nodes represent an outcome. This works by splitting the data into separate partitions according to an attribute selection measure (which in our case is "gini").

4.2.1 Replicating previous work along with hyper-parameter tuning

1. **Paper Implementation:** As per paper they have specified `Max_depth` = 10, `min_samples_ per_leaf` = 5, whose results we have shown in the following table.
2. **Our Experiment:** We performed `Grid Search` + `5-Fold Stratified Cross-validation` where we prepared a parameter grid space over following parameters and got their corresponding best values as shown below:
   (a) `criterion`: `entropy` and `gini`. Got `entropy` as the best.
   (b) `max_depth`: $[10, 25, 50]$. Got **25** as the best which is reasonable too as we have 25 features then the height of a *fully grown tree* must also be **25**.
   (c) `min_samples_leaf`: $[1, 4, 16]$. Got **1** as best which we know at the border of parameter space but again we can't set count less than 1 as it should be a natural number.

   (d) `min_samples_split`: $[64, 128, 256]$. Got **128** as best.
   Fixed Parameters: `max_features` $= log2$

4.2.2 Results

Experiments	Original Results		Results from our implementation	
	Accuracy	F_1 Score	Accuracy	F_1 Score
Paper Implementation	73.2%	65.5%	72.0%	63.4%
Our Experiment	-	-	**72.8%**	**63.6%**

Table 4: Decision Tree Experiment Results

4.3. Random Forest

Random Forest, as the name suggests is constructed using multitude of decision trees that get trained in parallel fashion on **bootstrapped** data. And during inference the final output is taken by **aggregating** the decisions of all decision trees (generally by voting). Hence it's called **Bagging** (Bootstrap + aggregating) which is a type ensemble machine learning algorithm designed for both classification and regression. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Here we have, first, replicated the paper implementation followed by hyper-parameter tuning.

4.3.1 Replicating previous work along with hyper-parameter tuning

1. **Paper Implementation:** In paper they have specified `Max_depth` $= None$, `min_samples_ per_leaf` $= 5$, and `num_estimators` $= 50$, whose results we have shown in the following table.
2. **Our Experiment:** Here again we performed `Grid Search + 5-Fold Stratified Cross-validation` where we prepared a parameter grid space over following parameters and got their corresponding best values as shown below:
   (a) `criterion`: `entropy` and `gini`. Got `entropy` as the best.
   (b) `max_depth`: $[10, 25, 50]$. Got **25** as the best and why it's reasonable is also discussed in previous models' result.
   (c) `n_estimators`: $[100, 200, 400]$. Got **200** as the best.
   (d) `min_samples_split`: $[32, 64, 128]$. Got **64** as the best.
   (e) `min_samples_leaf`: $[1, 4, 16]$. Got **1** as the best.
   Fixed Parameters: `max_features` $= log2$

4.3.2 Results

Experiments	Original Results		Results from our implementation	
	Accuracy	F_1 Score	Accuracy	F_1 Score
Paper Implementation	75.7%	66.9%	73.5%	64.6%
Our Experiment	-	-	**73.7%**	**65.1%**

Table 5: Random Forest Experiment Results

4.4. Gradient Boosting Classifier

Gradient boosting is a machine learning technique for regression, classification, and other tasks, which produces a prediction model by combining weak "learners" into a single strong learner (ensemble of weak prediction models, typically decision trees) in an iterative fashion (and hence it's a **Boosting** ensemble method). In gradient boosted trees, a decision tree is usually a weak learner, which outperforms random forest. Here models are built in a stage-wise manner as other boosting methods do, and it generalizes them by optimizing an arbitrary differentiable loss function.

4.4.1 Replicating previous work along with hyper-parameter tuning

1. **Paper Implementation:** In paper they have specified `Max_depth` = 4 and `num_estimators` = 500, whose results we have shown in the following table.
2. **Our Experiment:** Here again we performed `Grid Search + 5-Fold Stratified Cross-validation` where we prepared a parameter grid space over following parameters and got their corresponding best values as shown below:
   (a) `loss`: `deviance` and `exponential`. Got `exponential` as the best.
   (b) `n_estimators`: $[20, 50, 75]$. Got **50** as the best.
   (c) `min_samples_split`: $[128, 200, 256]$. Got **256** as the best.
   Fixed Parameters: `max_features` = $log2$; `max_depth` = 25

4.4.2 Results

Experiments	Original Results		Results from our implementation	
	Accuracy	**F_1 Score**	**Accuracy**	**F_1 Score**
Paper Implementation	75.0%	66.5%	73.3%	64.6%
Our Experiment	-	-	**73.8%**	**65.0%**

Table 6: GBC Experiment Results

Clearly, `Random Forest` and `Gradient Boosting` performed well among all the tree-based models.

5. Neural Network based models

Neural Network based models have proven to be much efficient to perform modelling on unstructured data. A prominent reason behind this is the Universal Approximation nature of NN models to approximate any complex function. As part of this project, we have developed models ranging from simple Multilayer Perceptron Model till Sequence Based RNN Attention models and SOTA transformer based models. Different set of experiments were run to understand the behavior of the models and identfy the models and corresponding configuration that gives the best results among all.

5.1. CBOW + MLP

Implemented a Multilayer Perceptron Model (MLP) which which stacks a set of dense non-linear layers followed a by softmax operation. We used 3 layer MLP model and used 'Relu' activation function to induce non-linearity in the model. These 3 dense layers are followed by softmax layer to get the probabilities of two classes. Used 'categorical-crossentropy' to calculate the loss and 'Adam' optimiser as the optimisation algorithm. Adam optimiser uses the combination momentum based movement and learning rate decay technique for better and fast convergence.

5.1.1 Word Embeddings and usage

CBOW (Continuous Bag of Words) - CBOW is one of the major breakthrough and widely used word embedding technique in text processing space. CBOW embeddings are generated using a Feed Forward Neural Network where the objective is to predict the target word given the context words as input to the network. We used word2vec 300D embeddings for this project to generate the vectors of the words in the question.

Extraction of Sentence Embeddings We used the same technique as mentioned in paper to extract the embeddings from both questions and add them to build the relation between questions. We performed few experiments where we used used concatenation of embeddings obtained from sum, difference and element wise product of embeddings from two questions.

5.1.2 Experiments

We ran multiple experiments by changing the model configuration and tuning different hyper-parameters to identify the best performing model. Scope of tuning different parameters combinations is huge in MLP. However, owing to the time constraints, we tried few experiments which seemed more relevant. Below is the gist of the experiments done.

1. **Paper Implementation**: Built 3 layer MLP model followed by softmax layer. Used sum of the average of the embeddings generated for each question as input to the model. Used 300D word2vec embedding for text vectorisation.
2. **Sum, Difference, Point Wise Multiplication of Embeddings**: Instead of just sum of the embeddings generated for both question, we generated another two matrices - difference and point wise product of embeddings. Thereafter, we concatenated all the matrices and fed to the model.
3. **Added dropout to control overfitting**: The model was overfitting after few epochs. Added dropout in addition to L2 regularisation to control overfitting.
4. **Used He Initialisation**: As we were using large number of perceptron in each layer, because of the 900D input concatenated text vector, the chances of 50% of perceptrons being dead were high because of Relu activation. Therefore, used He initialisation that divides the weights of the layer by the number of perceptron in the previous layer.

5.1.3 Results

Experiments	**Original Results**		**Results from our implementation**	
	Accuracy	F_1 **Score**	**Accuracy**	F_1 **Score**
Paper Implementation	83.4%	77.8%	86.3%	79.7%
Added dropout to control overfitting	-	-	**87.92%**	**84.67%**
Sum, Difference, Point Wise Multiplication of Embeddings	-	-	86.7%	82.7%
Used He Initialisation	-	-	87.7%	83.22%

Table 7: CBOW + MLP Experiments Results

5.2. GloVe + LSTM

5.2.1 Initial preprocessing with GloVe word embeddings

We employed GloVe embeddings as input embeddings for the creation of an LSTM-based model, as specified in the paper.

GloVe:- GloVe is an unsupervised learning technique that generates word vector representations. The resulting representations highlight intriguing linear substructures of the word vector space, and training is based on aggregated global word-word co-occurrence statistics from a corpus.

Extraction of sentence embeddings:- We need to use a strategy to translate sentences into vector representations because GloVe provides word vector representations. We utilised the same approach as the writers. For each word in a question, we calculated the word vector representations and added them together. After that, the representations for both questions were concatenated, and the difference and element wise dot product were calculated. The LSTM-based model was then fed the concatenated version of these representations as input.

5.2.2 Replicating previous work along with hyper-parameter tuning

While training the model, we employed different hyper-parameter tuning to check the performance of the model. The summary of the results is described below:-

1. **Paper implementation**:- First we tried to replicate the exact paper implementation i.e. LSTM layer with dropout of 0.1 and L2 regularization with $\beta = 0.01$.
2. **Experiment 1**:- One experiment which we did is implementing the same model but without using the dropout and L2 regularization technique.
3. **Experiment 2**:- Another experiment which we performed is that we removed the dropout from the model but kept L2 regularization as it is.

In all the above configurations we used softmax as final classifiying layer. Apart from that we used Adam as an optimizer and categorical crossentropy as a loss function. Accuracy was used as a metric while training the model.

5.2.3 Results

Experiment	**Original Results**		**Results from our implementation**	
	Accuracy	**F_1 Score**	**Accuracy**	**F_1 Score**
Paper Implementation	81.4%	75.4%	74.66%	65.40%
Without dropout and L2 regularization	-	-	**80.87%**	**73.98%**
Without dropout and with L2 regularization	-	-	69.90%	63.87%

Table 8: GloVe + LSTM results with Hyper-Parameter Tuning

5.3. Glove + BiLSTM

5.3.1 Initial preprocessing with GloVe word embeddings

We employed GloVe embeddings as input embeddings for the creation of an BiLSTM-based model, as specified in the paper.

Extraction of sentence embeddings:- Extraction of sentence embeddings for BiLSTM model is same as we did for the LSTM model.

5.3.2 Replicating previous work along with hyper-parameter tuning

While training the model, we employed different hyper-parameter tuning similar to we did in LSTM based model to check the performance of the model. The summary of the results is described below:-

1. **Paper implementation**:- First we tried to replicate the exact paper implementation i.e. LSTM layer with dropout of 0.1 and L2 regularization with $\beta = 0.01$.
2. **Experiment 1**:- One experiment which we did is implementing the same model but without using the dropout and L2 regularization technique.
3. **Experiment 2**:- Another experiment which we performed is that we removed the dropout from the model but kept L2 regularization as it is.

In all the above configurations we used softmax as final classifiying layer. Apart from that we used Adam as an optimizer and categorical crossentropy as a loss function. Accuracy was used as a metric while training the model.

5.3.3 Results

Experiment	Original Results		Results from our implementation	
	Accuracy	F_1 Score	Accuracy	F_1 Score
Paper Implementation	82.1%	76.2%	73.40%	63.10%
Without dropout and L2 regularization	-	-	**81.40%**	**75.02%**
Without dropout and with L2 regularization	-	-	74.10%	63.06%

Table 9: GloVe + BiLSTM results with Hyper-Parameter Tuning

5.4. GloVe + LSTM + Attention

Attention mechanism is one of the major breakthrough in the deep learning space. The major idea behind attention is to calculate context vector that tells importance to be given to input token at the time of calculating the output. This is done by calculating an alignment score. In our project we calculated dot-product attention which is same as that of on used by the authors in the paper.

- Alignment score captures the importance of a word in a question1 w.r.t the to a word in question2 by doing the dot product of hidden states corresponding to words from question1 to hidden states from question2. For the ease of use, as LSTM in Keras does not give hidden states for all time steps, we used the outputs from hidden states from each time step. Let $\{\mathbf{u}_i\}_i^{l_u}$ and $\{\mathbf{v}_j\}_j^{l_v}$ be the hidden states from the RNN for the two questions, where l_u and l_v are the lengths of question 1 and 2 respectively. We calculate attention weights, e_{ij}, by simply taking a dot product. The dot-product matrix e_{ij} gives the alignment score matrix.
$$e_{ij} = \mathbf{u}_i^T \mathbf{v}_j, \ \forall \ i \in \{1, \ldots, l_u\} \text{ and } \forall \ j \in \{1, \ldots, l_v\} \tag{1}$$
- Then, we do softmax of this matrix across rows to find the softmax alignment score for question1 ($attention_u$). Similary, the softmax alignment score for question2 ($attention_v$) is calculated by doing softmax across column dimension.
$$softmax\_rows_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{l_v} \exp(e_{ik})} \tag{2}$$
$$softmax\_columns_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{l_u} \exp(e_{kj})} \tag{3}$$
- Thereafter, attention vector for question1 ($\overline{\mathbf{u}}_i$) is obtained by doing dot product of $attention_u$ with hidden states of question2 (i.e, v_j). Similarly, attention vector for question2 ($\overline{\mathbf{v}}_j$) is obtained by doing dot product of $attention_v$ with hidden states of question1 (i.e., u_i).
$$\overline{\mathbf{u}}_i = \sum_{j=1}^{l_v} softmax\_rows_{ij} \mathbf{v}_j \tag{4}$$
$$\overline{\mathbf{v}}_j = \sum_{i=1}^{l_u} softmax\_columns_{ij} \mathbf{u}_i \tag{5}$$
- To let the model decide on the proportion of value to be taken from each attention representations, both representation are passed to a **Feed Forward layer** followed by tanh non-linear activation. Output representation from the network is Final attention vector representation (vstar and ustar) of each question.
$$\mathbf{u}^* = \tanh\left(\mathbf{W}^u \overline{\mathbf{u}}_{l_u} + \mathbf{V}^u \mathbf{u}_{l_u}\right) \tag{6}$$
$$\mathbf{v}^* = \tanh\left(\mathbf{W}^v \overline{\mathbf{v}}_{l_v} + \mathbf{V}^v \mathbf{v}_{l_v}\right) \tag{7}$$

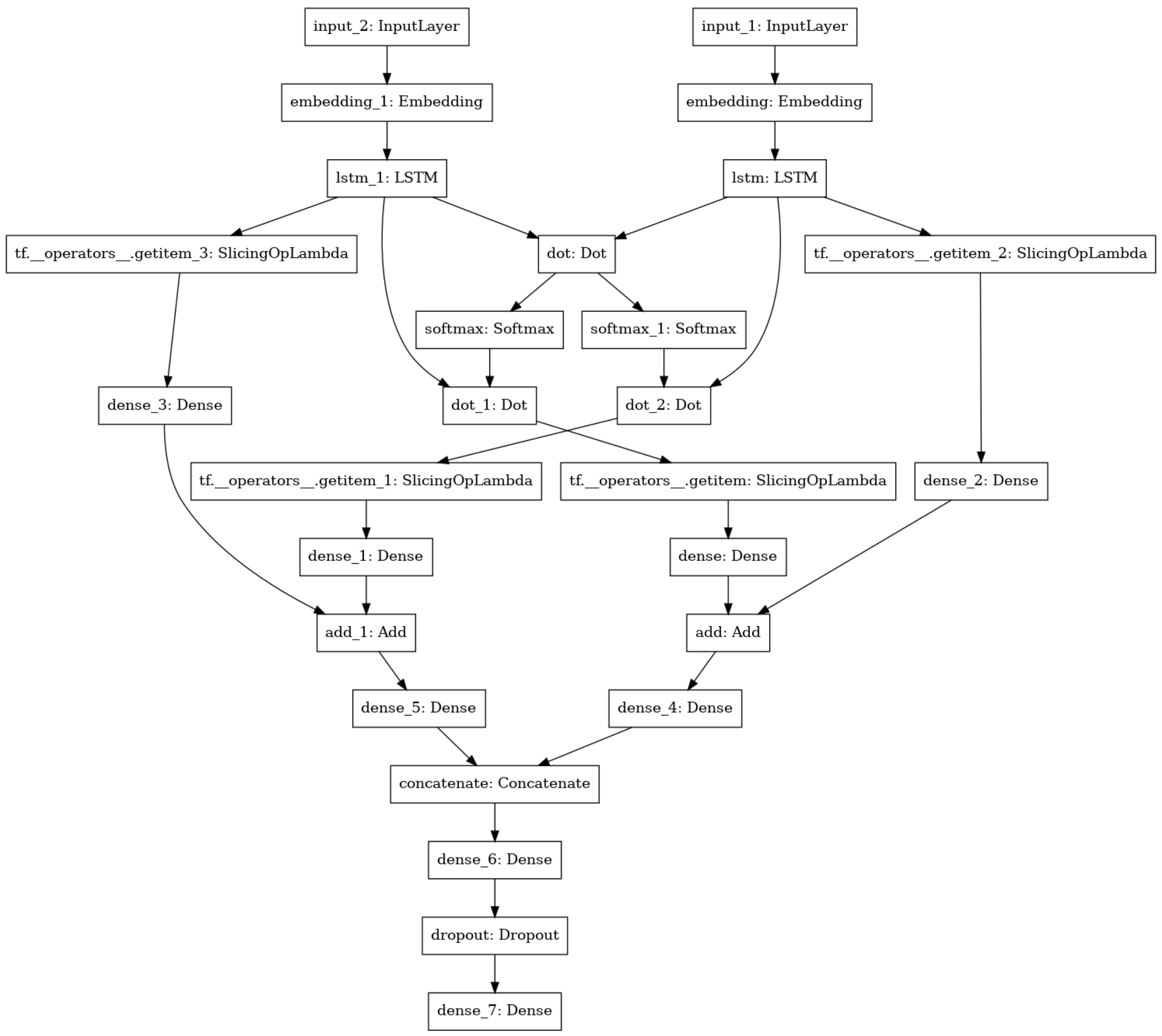


Figure 6: Attention Model

- These, **u*** and **v*** vectors are then concatenated and passed to Feed Forward Network of two layers followed by *softmax* layer.

5.4.1 Word Embeddings Usage and Extraction

We took the embeddings from the GLoVe embedding model in the similar way as we did for LSTM/BiLSTM models. Just a little difference is there is that instead of combining the representations of both the questions, we took both representations separately and then passed through LSTM layer to compute the dot product attention.

5.4.2 Results

Experiment	**Original Results**		**Results from our implementation**	
	Accuracy	F_1 **Score**	**Accuracy**	F_1 **Score**
Paper Implementation	81.8%	75.5%	77.71%	68.79%

Table 10: GloVe + LSTM + attention results

5.5. GloVe + BiLSTM + Attention

5.5.1 Initial pre-processing with GloVe word embeddings

BiLSTM adds an advantage of capturing both forward and backward context over LSTM. In this model, we replaced LSTM with BILSTM in addition to the attention mechanism. The architecture of the BiLSTM attention model is similar to that of LSTM attention model, except one change. While calculating u* an v*, we took the average of all the states from u_bar and v_bar matrix and and representation from LSTM instead of last states in LSTM + Attention.

5.5.2 Word Embeddings Usage and Extraction

We used the same process to calculate embeddings and dot product attention as we used in the LSTM + Attention model as explained in the previous sub-section.

5.5.3 Results

Experiment	**Original Results**		**Results from our implementation**	
	Accuracy	**F_1 Score**	**Accuracy**	**F_1 Score**
Paper Implementation	82.3%	76.4%	79.96%	71.99%

Table 11: GloVe + BiLSTM + attention results

5.6. Transformer based model (BERT)

Recently, transformers have proved to give the state of the art results for most of the NLP based tasks. Therefore, we used BERT (a transformer's encoder based model) to perform duplicate question detection on Quora dataset. A brief overview of the model, training and testing configuration and results are being shown below.

5.6.1 BERT

BERT which stands for Bidirectional Encoder Representations from Transformers is a paper by Google AI Language researchers. It has created a stir in the Machine Learning field by delivering cutting-edge findings in a range of NLP tasks, including as Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others. It is trained on 2 tasks which are Masked Language Modelling (MLM) and Next Sentence Predictions (NSP).

It leverages quite a new and useful concepts like positional encoding, multi-head self attention. These concepts helps BERT to understand importance of a word with respect to all other words in a sentence. It also helps the model to understand the context of a sentence in both the directions which is also evident from the model name.

5.6.2 Training Configuration

We used BERT model from huggingface.co and then put a linear layer on top of it with a dropout value of 0.1. The kept the final classifier layer on it to perform classification. We used Adam optimizer with cross-entropy loss.

5.6.3 Results

Experiment	**Results from our implementation**	
	Accuracy	**F_1 Score**
BERT + linear layer	89.69%	86.67%

Table 12: BERT results

6. Work distribution

Sl. No.	Model type	Model name	Contributors
1	Data Analysis		Dhaval
2	Linear Models	Logistic Regression	Dhruv, Lakshya
		SVM	Dhruv, Lakshya
3	Tree Based	Decision Tree	Aditya, Dhaval
		Random Forest	Aditya, Dhaval
		Gradient Boosting	Aditya, Dhaval
4	Neural Network Based	CBOW + MLP	Lakshya, Dhaval
		GloVe + LSTM	Dhaval, Dhruv
		GloVe + BiLSTM	Dhaval, Dhruv
		GloVe + LSTM + Attention	Lakshya, Aditya
		GloVe + BiLSTM + Attention	Lakshya, Aditya
5	Transformers Based	BERT	Dhaval, Lakshya

Table 13: Work Distribution

7. Results Summary

The below table shows the summary of results for all the models which we tried and tested. We are reporting the best results from our set of experiments below:-

Sl. No.	Model type	Model name	Accuracy (best)	F_1 Score (best)
1	Linear Models	Logistic Regression (Unigrams)	**68.9%**	**44.4%**
		Logistic Regression (Bigrams)	66.9%	42.3%
		Logistic Regression (Trigrams)	65.1%	31.9%
		SVM (Unigrams)	**68.73%**	**43.1%**
		SVM (Bigrams)	66.59%	32.4%
		SVM (Trigrams)	65.06%	24.1%
2	Tree Based	Decision Tree	72.8%	63.6%
		Random Forest	73.7%	**65.1%**
		Gradient Boosting	**73.8%**	65.0%
3	Neural Network Based	CBOW + MLP	**87.92%**	**84.67%**
		GloVe + LSTM	80.87%	73.98%
		GloVe + BiLSTM	81.40%	75.02%
		GloVe + LSTM + Attention	77.71%	68.79%
		GloVe + BiLSTM + Attention	79.96%	71.99%
4	Transformers Based	BERT	**89.69%**	**86.67%**

Table 14: Results Summary

8. Future Scope

The future task could be to try out more complex and newer models to get the better results. Also, the models can be tested on disjoint set as well as mentioned by the authors.

9. Conclusion

As part of this project, we built multiple models ranging from Linear, Tree based to Deep neural Network models and performed different experiments for each model. In Linear Models, SVM outperforms the Logistic Regression model by a significant margin. In Tree based models, ensemble models works better than a stand-alone decision tree. Performance of both Random Forest and Gradient are very similar, though their back-end ensemble mechanism and methodology is different. Furthermore, we built Deep learning based models that mimics the functionality of human brain through a deep neural network. We built MLP, Sequence Based and

Transformer based models. As per the paper, CBOW + MLP gave the best results. Our results resonates with that of the paper. Of all the Deep Learning based models, Transformer based BERT model gave the best results followed by MLP with CBOW embeddings.

References

[1] Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. Natural language understanding with the quora question pairs dataset. *arXiv preprint arXiv:1907.01041*, 2019.