Quora Insincere Questions Classification Leveraging Machine Learning Methods

An Independent Study (Eco 6398)

in

partial fulfillment of the requirements

for the degree of

Master of Science

in

Applied Economics and Predictive Analytics

by

Priyanka Asnani

(BTech, Uttar Pradesh Technical University, 2013)
(M.S., University Of Michigan, 2019)

Under the supervision of

Thomas B. Fomby

May 2, 2022

Contents

Chapter 1

Introduction

Quora is an American social question-and-answer website based in Mountain View, California. It was founded on June 25, 2009. It is one of the most popular knowledge sharing platforms where people can post questions and get responses from anyone on the Quora community, thereby helping people to learn from each other. Quora is very similar to Yahoo Answers where people from all over the world can ask simple, personal, and professional questions. One of the major challenges with the online platforms such as Reddit, Yahoo Answers, and Stack Exchange is identifying posted questions that violate the forum's guidelines. Holding online conversations from becoming toxic is one of today's most important internet challenges. Quora has about millions of monthly users and therefore it is of an utter importance to create safe content for its users which involves anything from reporting inappropriate videos to deleting toxic and insincere comments or questions. The key challenge for Quora is to identify 'insincere' questions on the platform. An insincere question is defined as a question intended to make a statement rather than look for helpful answers. Some characteristics that can signify that a question is insincere:

- Has a non-neutral tone
- Has an exaggerated tone to underscore a point about a group of people
- Is rhetorical and meant to imply a statement about a group of people
- Is disparaging or inflammatory
- Suggests a discriminatory idea against a protected class of people, or seeks confirmation of a stereotype
- Makes disparaging attacks/insults against a specific person or group of people
- Disparages against a characteristic that is not fixable and not measurable
- Based on an outlandish premise about a group of people

- Isn't grounded in reality
- Based on false information, or contains absurd assumptions
- Uses sexual content (incest, bestiality, pedophilia) for shock value, and not to seek genuine answers

As Quora grows, it becomes increasingly important to identify insincere questions and remove them as quickly as possible before they can harm their reputation. Due to the enormous size of the data on Quora, manually classifying questions is simply not feasible. This process of classifying insincere questions can be made efficient using Machine Learning methods. In this paper, we propose a system where we use machine learning classification models to classify insincere questions, thereby helping in maintaining the integrity of the website.

Supervised Machine Learning algorithms such as Naive Bayes, Logistic Regression, Support Vector Machine (SVM), Decision Tree, Random Forest, Voting Classifier and Deep learning methods such as BERT are implemented for this binary classification problem.

The paper is structured as follows: Section II data preparation and prepossessing methods have been discussed on detail. Section III discusses various text representation methods including static and dynamic representations. It further points out the pros and cons of the same. Section IV explains the various machine learning methods used and also discuses their implementation in detail. Section V explains the BERT model in detail including its architecture and training. Section VI compares the results of the models used in the study and highlights the methods of evaluation.

Chapter 2

Dataset

2.1 Overview of the Data

The dataset used in this project has been provided by Quora for the online competition on Kaggle (https://www.kaggle.com/). In this competition the task was to predict whether a question asked on Quora is sincere or not. The data consists of 1.3 million observations and 3 features. It can be challenging to train large datasets on a single machine as the computing resources such as CPU and RAM are limited. Training machine learning algorithms on large datasets can be computationally intensive, and therefore for this reason we use the high-performance computing environment at Southern Methodist University (SMU). We used SMU ManeFrame II medium-mem-1-m machine (36 cores and 750 GB memory) to submit the training jobs. The training was distributed across all cores to reduce the excessive training times. The new and more efficient architecture, high core count, and high memory capacities of the compute nodes provides significant improvements to existing computationally and memory intensive workflows. The sample observations from the dataset is presented in Fig 2.1. Here is the description of the three data fields given in the dataset:

- qid: unique question identifier
- question_text: Quora question text
- target: this field represents labels for each question. The value is 1 if question is insincere, and 0 otherwise.

2.2 Exploratory Data Analysis

Exploratory data analysis is one of the most important parts of any machine learning workflow and Natural Language Processing is no different. Exploratory Data Analysis is the process of exploring data, generating insights, testing hypotheses, checking assumptions and revealing underlying hidden patterns in the

	qid	question_text	target
0	00002165364db923c7e6	How did Quebec nationalists see their province...	0
1	000032939017120e6e44	Do you have an adopted dog, how would you enco...	0
2	0000412ca6e4628ce2cf	Why does velocity affect time? Does velocity a...	0
3	000042bf85aa498cd78e	How did Otto von Guericke used the Magdeburg h...	0
4	0000455dfa3e01eae3af	Can I convert montra helicon D to a mountain b...	0

Figure 2.1: Sample Observations

data. Distribution of target variable is presented in Fig 2.2. The dataset is extremely imbalanced, with only 6% of the question text being insincere. To explore the data further, we look at top unigrams (one word) and bigrams (two adjacent words) based on their word-frequency in sincere and insincere dataset. The count plot for unigrams is presented in Fig 2.3. The count plot for bigrams is presented in Fig 2.4.

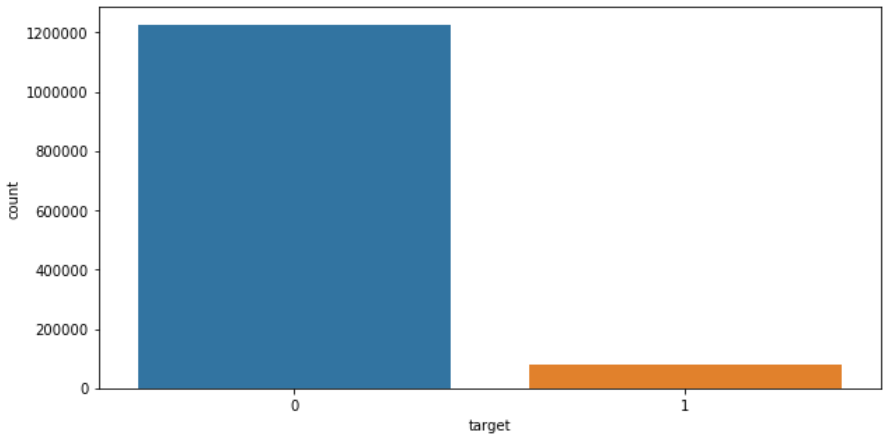

Figure 2.2: Distribution of Target Variable

2.3 Oversampling

To obtain good performance with machine learning models, the training dataset should be balanced across all classes; that is, we should have same number of observations for each class. In this problem, observations for rare class (insincere) is only 6% of the total observation, which is not enough to yield useful information about what distinguishes it from the dominant class. To deal with

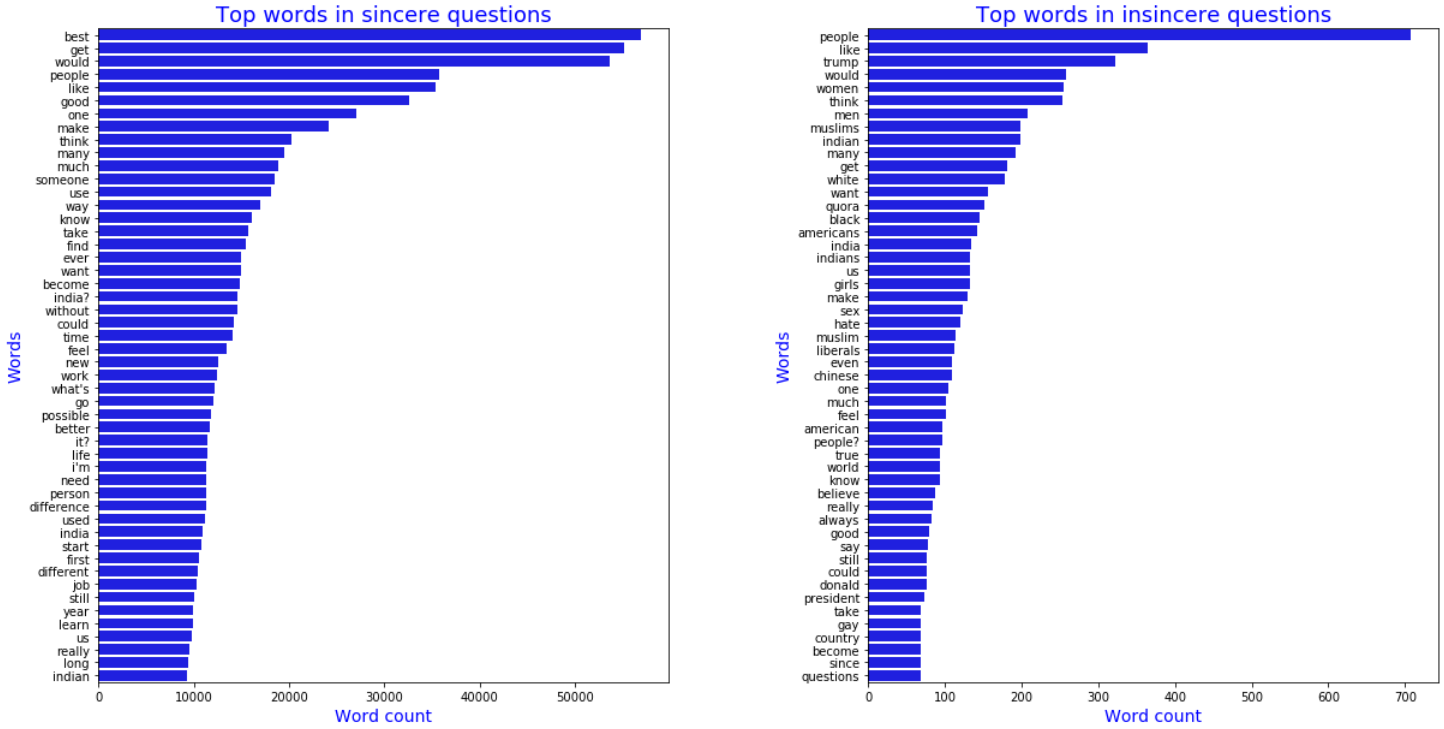

Figure 2.3: Top Unigrams in Sincere and Insincere Questions

such imbalanced dataset, we use an Oversampling technique to training set and validation sets. The training set is used to fit the machine learning models, while the validation set is used to evaluate models. In training set, we maintain an equal proportion of 'sincere' and 'insincere' classes by oversampling the records from the 'insincere' class. The Training set consists of 40,000 observations for the 'insincere' class and 40,000 observations for the 'sincere' class. In validation set, we use stratified sampling approach to maintain the same proportion of classes 'sincere' and 'insincere' as in the original dataset. Validation set consists of 40,000 observations for the 'insincere' class and the 600,000 observations for the 'sincere' class. The observations in the training and validation set are chosen using random sampling without replacement.

2.4 Data Preprocessing

2.4.1 Tokenization

The process of splitting a text sequence into words is called Tokenization. It is the first and most important step for text processing. The Natural Language

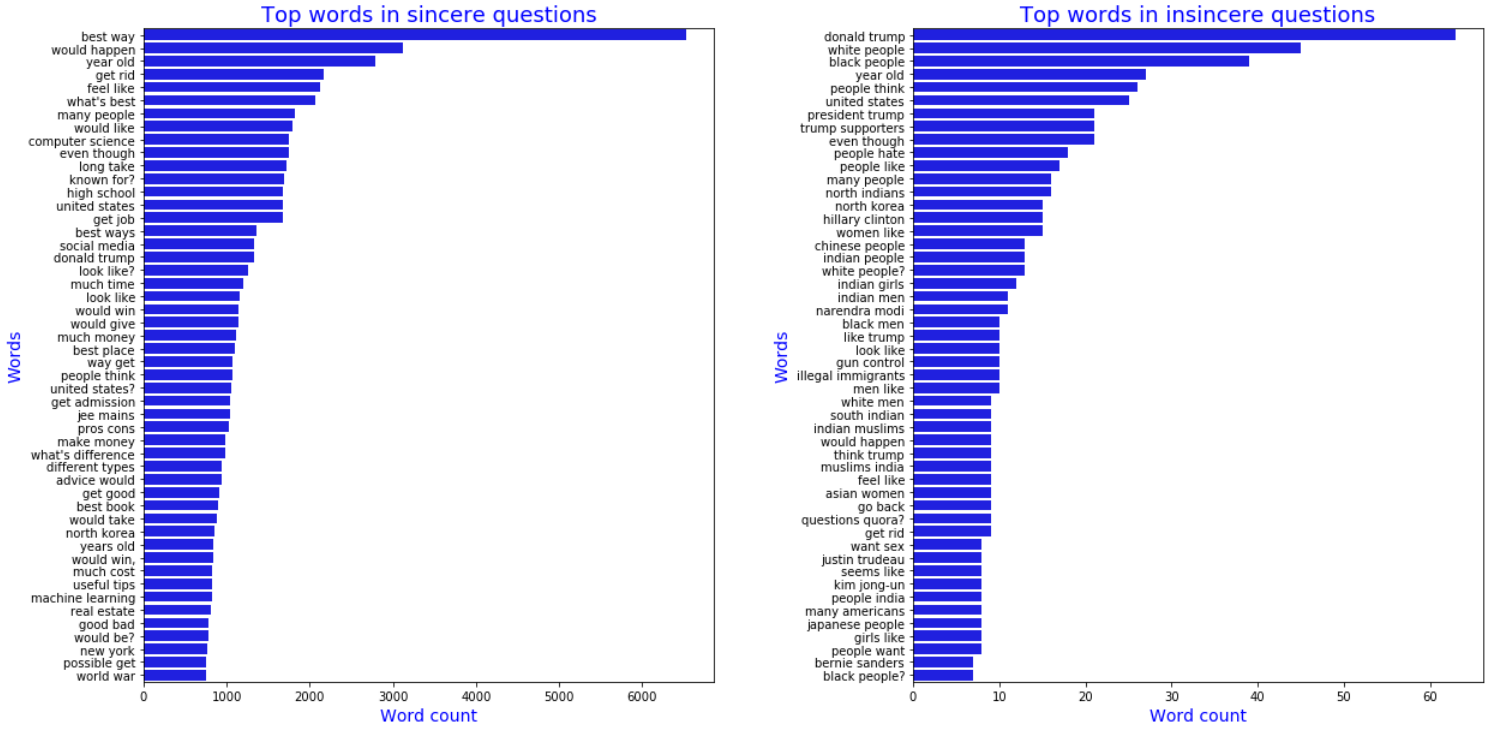

Figure 2.4: Top Bigrams in Sincere and Insincere Questions

Toolkit (NLTK) package in Python is used for this purpose. It splits the text on a delimiter such as a comma(,) and treats each individual word as a token.

2.4.2 Expand Contractions

Contractions are words or combination of words that are shortened by dropping letters and replacing them by an apostrophe. Contractions include words such as you'll, I'd, can't, won't, don't, etc. We wrote regular expressions to expand on the contractions. For example, you'll was replaced by you will, don't was replaced by do not. Converting each contraction to its expanded, original form helps with text standardization.

2.4.3 Remove Punctuations

Punctuation's do not provide any useful information, so we remove them as a part of the text cleaning and processing step in NLP. In addition to punctuation, the text also includes certain special characters such as '÷•à³‘´°£€™²_' which were removed from each text sequence.

2.4.4 Remove Stop Words

Stop words are the words that are used to add meaning to the sentence. We consider stop words as unimportant as they appear frequently in the corpus and do not provide any useful information. We eliminate stop words to reduce the noise in the data and thus allowing models to focus on the important words instead. The NLTK package in Python has pre-defined list of stop words that we removed from the data. Typically, these can be articles, conjunctions, prepositions and so on. Some examples of stopwords are a, an, to, and the. We can also add our own domain-specific stopwords as needed to the list.

2.4.5 Lemmatization

Lemmatization is a text normalization technique within the field of Natural Language Processing that are used to prepare text, words, and documents for further processing. We use WordNet Lemmatizer in the NLTK package to identify the lemma for each word. Lemmatizer analyzes full vocabulary to apply a morphological analysis to words. For example, the lemma of 'was' is 'be' and the lemma of 'mice' is 'mouse'. Lemmatization also helps reduce the dimensionality of the data which result in a better performance of models.

Chapter 3

Textual Representations

Representation learning, i.e., learning representations (features) from the data that makes it easier to derive useful insights from the unstructured text data when building classifiers, has been one of the key research areas since the developments of NLP. These representations are used in various natural language application such as in various semantic tasks such as information retrieval, text classification, or sentiment analysis. The performance of machine learning methods are dependent on the choice of data representations on which they are applied (Bengio et al., 2013). For this reason, most of the effort when building and deploying machine learning models, goes into designing prepossessing pipelines, feature engineering, and data transformations that can support effective machine learning.

The notion of word similarity is extremely useful in NLP. Knowing how similar two words can help us understand how similar two phrases or sentences are, which is an important concept in larger semantic tasks such as text summarization, question answering, and paraphrasing. The distributional hypothesis first formulated by linguists like Joos (1950), Harris (1954), and Firth (1957). The hypothesis suggests that Words that appear in similar contexts tend to have similar meanings; that is there is a link between similarity in how words are distributed in the text and similarity in what they mean.

Machine learning models take vectors (arrays of numbers) as input. When working with text, the first thing we must do is come up with a strategy to convert strings to numbers (or to "vectorize" the text) before feeding it to the model. In the early age of natural language processing, one of the most popular approaches to represent words was the term-document matrix approach, which was originally defined as means of finding similar documents for the document retrieval task (Salton 1971). Two documents that are similar will tend to have similar words, hence, their vectors will tend to be similar. In this model, each row represents a word in the vocabulary and each column represents a document from some collection of documents. For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors

because they tend to occur in similar documents. We do this by associating each word with a word vector— a row vector rather than a column vector.

3.1 Bag of Words

Representing words using the Bag of Words model was the very first approach that researchers used to vectorize the text. The bag-of-words featurization indicates the presence or absence of individual words or n-grams. The very naive approach is to represent each word in the vocabulary as "one-hot" word vector, in which we create a zero vector with length equal to length of vocabulary, then place one at the index corresponding to the word. Once we have, we can create the encoding for the entire sentence by concatenating the one-hot vectors for each word in the sentence. With this approach, all vectors are orthogonal to each other. Therefore, it is intractable to identify the semantic distance between words. For instance, the words apple, orange and book are equally similar to each other with the one-hot vector. To overcome this drawback of one-hot featurization, we can encode each word with a unique number instead. Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column).

An integer-encoding can be challenging for a model to interpret. A linear classifier, for example, learns a single weight for each feature. Because there is no relationship between the similarity of any two words and the similarity of their encodings, this feature-weight combination is not meaningful. This approach is also highly inefficient. A one-hot encoded vector is sparse (meaning, most indices are zero). Imagine you have 10,000 words in the vocabulary. To one-hot encode each word, you would create a vector where 99.99% of the elements are zero.

3.2 TF-IDF Model

One of the drawbacks Bag-of-words featurization is high dimensionalty of the vectors. Because the method treats each unique term as a separate dimension, this step leads to a large term space. To reduce the dimension space and generate a smaller vocabulary, we use term frequency inverse document frequency (Tf-idf) weighting scheme to score each word to signify its importance in the document and corpus. This statistic measures how important a word is to a document in a collection (or corpus) of documents. This method is a widely used technique in Information Retrieval and Text Mining. The TF-IDF weight is composed of two terms: the first computes the normalized Term Frequency (TF), which measures how frequently a term occurs in a document. It is possible that a term would appear much more times in long documents than shorter ones, so we normalize term frequency by dividing it by the document length i.e. the total number of words in the document. It is defined as the number of times a word appears in a document, divided by the total number of words in that document (Eq. 3.1);

the second term is the Inverse Document Frequency (IDF), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. Thus it weighs down the frequent terms while scales up the rare ones. It is computed as the logarithm of the number of the documents in the corpus N divided by the number of documents where the specific term appears df_t see (Eq. 3.2). Finally, we see in (Eq. 3.3) that we multiply term frequency and inverse document frequency weight of the word for estimating the tf-idf weights for individual terms. Intuitively, this weighting scheme assumes that the best descriptive terms for a given document are those that occur very often in the given document (term frequency) but not much in other documents (inverse document frequency) (Salton and Buckley, 1988). In this paper, we use tf-idf weighting approach to transform words. To reduce the dimensionlaity of tf-idf weighting matrix, we remove the words that appeared in less that 10 questions. As a result, our vocabulary consists of about 7000 words which were transformed using tf-idf vectorization approach and then fed into classification models.

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document} \tag{3.1}$$

$$IDF = log\left(\frac{N}{df_t}\right) \tag{3.2}$$

$$TF - IDF = TermFrequency\ (TF) * InverseDocumentFrequency\ (IDF) \tag{3.3}$$

3.3 Word Embeddings

Historically, researchers have used one-hot embeddings to encode words. However, the dimension of the one-hot embedding is large since each unique word takes one dimension. Although the one-hot and tf-idf featurization described above can work, it is still limited because it provides no information about how words are related to each other. For instance, the words "play", "playing", and "igloo" would have one-hot features [1,0,0], [0,1,0], and [0,0,1], which are all equally different, even though "play" and "playing" should intuitively have very similar representations.

A commonly used alternative to bag-of-words and tf-idf approach is word embeddings. The idea is to encode a word using a vector of real numbers. These fixed-length vector of real numbers is known as word embeddings. The diagram for a word embeddings is presented in Fig. 3.1. Each word is represented as a 4-dimensional vector of floating point values. Another way to think of an embedding is as "lookup table". After these weights have been learned, you can encode each word by looking up the dense vector it corresponds to in the table. Importantly, you do not have to specify this encoding by hand. An embedding is a dense vector of floating point values (the length of the vector

is a parameter you specify). Instead of specifying the values for the embedding manually, they are trainable parameters (weights learned by the model during training, in the same way a model learns weights for a dense layer). It is common to see word embeddings that are 8-dimensional (for small datasets), up to 1024-dimensions when working with large datasets. A higher dimensional embedding can capture fine-grained relationships between words, but takes more data to learn. The advantage of using word embeddings is that similar words can end up learning similar embeddings which allows the model to better encode the meaning of sentences. The concept is illustrated in Fig. 3.2. where word vectors are projected in two-dimensional space and similar words appear close to each other.

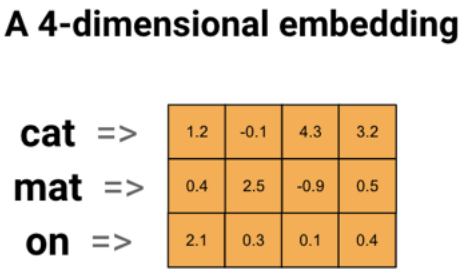

Figure 3.1: Word Embeddings for words cat, mat, and on[1]

The most widely used methods for word embedding are Word2Vec (Mikolov et al. 2013) and GloVe (Pennington et al. 2014), both of which are based on word co-occurrences. Such models take in a large number of texts and output a fixed vector for each word in the texts. The more frequently the two words co-occur, the more correlated two embeddings are. Once the model is trained, the embeddings of the words no long change, therefore we call these embeddings static embeddings. Such model generates the same embedding for one word no matter its context, although the meanings of the words can depend on the context in which this word occurs.

[1]https://www.tensorflow.org/text/guide/word_embeddings
[3]https://web.stanford.edu/ jurafsky/slp3/6.pdf

Figure 3.2: Two-dimensional projection of word embeddings showing words with similar context appear close in the vector space[3]

Chapter 4

Machine Learning Methods

In this paper, we solve a binary classification problem - given a corpus of questions from Quora. We need to classify whether the questions submitted to Quora are 'insincere' or not. The goal of classification is to take a single observation as an input, extract some useful features, and thereby classify the observation into one of a set of discrete classes. We handle this problem using supervised machine learning algorithms. To train supervised learning algorithms in natural language processing, we have a training set of N documents and set of output classes $c_1, c_2, ..., c_m$. The goal of the classifier is to predict class $c \in C$ for a new document d. Since machine learning classifiers cannot understand text, we encode each text document using tf-idf featurization and then then feed these encoded vectors to machine learning models. A probabilistic classifier like Naive Bayes and Logistic Regression will tell us the probability of the observation being in the class. We will be taking a closer look at several specific algorithms for supervised and unsupervised learning, starting here with naive Bayes classification.

4.1 Naive Bayes

Naive Bayes classifiers are built on Bayesian classification methods. The intuition behind Bayesian classification is to use Bayes rule in Eq. 4.1 to break down any conditional probability into three other probabilities.

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{4.1}$$

In Bayesian classification, we are interested in finding the probability of a class label given a document. The classifier computes the probability for each class $c \epsilon C$ then returns the class $\hat{c}$ with the maximum posterior probability.

$$\hat{c} = \underset{c \epsilon C}{argmax} P(c|d) = \underset{c \epsilon C}{argmax} \frac{P(d|c)P(c)}{P(d)} \tag{4.2}$$

We simplify the Eq. 4.2 by dropping P(d). This is possible because $\frac{P(d|c)P(c)}{P(d)}$ will be computed for each class but the probability of document P(d) will remain unchanged as we want to identify most likely class for same document.

$$\hat{c} = \underset{c \,\epsilon\, C}{argmax} P(c|d) = \underset{c \,\epsilon\, C}{argmax} P(d|c)P(c) \tag{4.3}$$

A document can be represented as set of features $(f1, f2, .., f_n)$:

$$\hat{c} = \underset{c \,\epsilon\, C}{argmax} P(c|(f_1, f_2, .., f_n)) = \underset{c \,\epsilon\, C}{argmax} P((f_1, f_2, .., f_n)|c)P(c) \tag{4.4}$$

This classifier makes an naive assumption that the probabilities $P(f_i|c)$ are independent, and hence each feature probability given class c can be multiplied as in Eq. 2.5. This assumption is called as naive Bayes assumption.

$$P((f_1, f_2, .., f_n)|c) = P(f_1|c).P(f_2|c).P(f_3|c)...P(f_n|c) \tag{4.5}$$

The calculations for probabilities for each feature is done in log space to avoid underflow and increase speed. The final equation for the class chosen by naive Bayes classifier is presented in Eq. 2.6.

$$\hat{c} = \underset{c \,\epsilon\, C}{argmax} logP(c) + \sum_{i=1}^{n} logP(f_i|c) \tag{4.6}$$

The Naive Bayes classifier is extremely fast and one of the simplest classification algorithms. Because they are so fast and have few hyper-parameters, they end up being very useful as a baseline models for classification problems.

4.2 Logistic Regression

Logistic Regression is a probabilistic classifier that is extremely useful is supervised machine learning tasks. The input to the logistic regression classifier is the set of documents, where each document is represented as a vector of features $[x_1, x_2, .., x_i]$. The logistic regression learns from the training set, the set of weights $[w_1, w_2, ..., w_i]$ and a bias term. Each weight w_i is associated with the one of the input feature x_i. The w_i represents the importance of an input feature to the classification decision. The bias term, also called the intercept, is the real number added to the weighted inputs. Once the logistic regression classifier learns the weights, it makes the classification decision by taking the dot product of the input feature vector and weights vector and adds the bias term to the weighted sum. The resulting number z lies between $-\infty \; to \; \infty$

$$z = (\sum_{i=1}^{n} w_i.x_i) + b \tag{4.7}$$

To obtain the probability, the value is z is passed through sigmoid function, also called logistic function The sigmoid function in Eq. 4.8. forces z to lie in range

$[0, 1]$. The value that we get from the squashing function is the probability of the document being insincere $P(y = 1|x)$ and the probability of the document being sincere $P(y = 0|x)$. The sum both probabilities sum to 1.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.8}$$

$$P(y = 1|x) = \sigma(w.x + b) \tag{4.9}$$

$$P(y = 0|x) = 1 - \sigma(w.x + b) \tag{4.10}$$

Once we have the probability for a given instance, we classify the instance as insincere if $P(y = 1|x)$ is greater than 0.5, which we call as the decision boundary. The classifier outputs 1; indicating that the observation belongs to class insincere and 0; indicating that the observation belongs to class sincere.

$$classify(x) = \begin{cases} 1, & \text{if } P(y = 1|x) > 0.5. \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

The logistic regression works well with large datasets with many correlated features. Like naive Bayes, it can be used a base line model for many applications in NLP.

4.3 Support Vector Machines

Support Vector Machines is a powerful machine learning algorithm suitable for classification and regression tasks. This classifier simply finds a line or curve (in two dimensions) or manifold (in multiple dimensions) that divides the classes from each other. The intuition is rather than simply drawing a zero-width line between the classes, we can draw around each line a margin of some width, up to the nearest point. In support vector machines, the line that maximizes this margin is the one we will choose as the optimal model. Support vector machines are an example of such a maximum margin estimator.These points are the pivotal elements of this fit, and are known as the support vectors, A key to this classifier's success is that for the fit, only the position of the support vectors matter; any points further from the margin which are on the correct side do not modify the fit. Technically, this is because these points do not contribute to the loss function used to fit the model, so their position and number do not matter so long as they do not cross the margin. This type of basis function transformation is known as a kernel transformation, as it is based on a similarity relationship (or kernel) between each pair of points. A potential problem with this strategy projecting points into dimensions is that it might become very computationally intensive as data grows large. This kernel trick is built into the SVM, and is one of the reasons the method is so powerful. In Scikit-Learn, we can apply kernelized SVM simply by changing our linear kernel to an RBF kernel, using the kernel model hyperparameter.

4.4 Decision Tree

Decision Trees provide simple classification rules to make the classification decision, and their decisions are easy to interpret. Decision trees can be used for both classification and regression problems. In this paper, we apply the decision tree classifier to a binary classification problem. The input to the decision tree classifier is the set of documents, where each document is represented as a vector of features $[x_1, x_2, .., x_n]$. The predictor variable used are considered to be continuous, categorical. The outcome variable will be a categorical variable Y which is equal to 1; if the document is insincere and 0; if document is sincere. The classifier recursively partitions the space of the predictor variables such that each space is homogeneous or as pure as possible. In other words, for each predictor, we search for a space where the records belong to only one class. First, one of the predictor variables is selected, say x_i, and a value of x_i, say v_i, is chosen to split the dimensional space into two parts: one part that contain all observations with $x_i < v_i$ and the other with all observations $x_i > v_i$. Then, we split the space on another predictor variable and its value.

The classifier computes the impurity for all possible split values for each predictor . These split points are then ranked according to their impurity factor, and the split chosen whose purity factor is maximum. The Impurity is measured using two popular techniques Gini Index and Entropy. Gini Index for a node is described by Eq. 4.12. p_k is the proportion of records that belongs to class k.

$$GiniIndex = 1 - \sum_{k=1}^{m} p_k^2 \tag{4.12}$$

Entropy is the another popular measure to compute impurity of a node. It is described in described by Eq. 4.13. P_k is the proportion of records that belongs to class k

$$Entropy = -\sum_{k=1}^{m} p_k \log_2(p_k) \tag{4.13}$$

The values for Gini Index and Entropy measure is 0 when all records belong to the same class. In binary classification, gini index and entropy measure are both maximized at $p_k = 0.5$, which means that impurity is maximum when there are equal number of records for each class.

4.5 Random Forest

Random Forest is an ensemble of Decision Trees. The idea is to train Decision Trees classifier on random subsets of training set and take the average of the predictions of all the decision trees. Random sampling can be performed with replacement or without replacement. When sampling performed with replacement, this method is called bootstrapping. Once all predictors are trained, the Random Forest classifier will make the prediction by simply aggregating the

predictions of all the trees. For classification, the class with most frequent prediction is chosen. For regression, the average of the predictions from all the predictors is computed. This aggregated result reduces both bias and variance. The hyperparameters of this model is same as Decision Tree, except that we need to specify the number of predictors, also known as estimators, on which to train the model on.

4.6 Voting Classifier

The Voting Classifier is an ensemble machine learning model that aggregated the predictions from multiple machine learning models. This is a technique that is used to improve the predictive performance; ideally outperforming the best model in the ensemble. The idea is instead of creating and evaluating separate classifiers, we create an ensemble of the classifiers like Naive Bayes, Logistic Regression, Decision Tree, Random Forest, and Support Vector Machines and predict the class with the majority vote. This majority-vote classifier is called hard voting classifier. We can also predict the class with the highest class probability, averaged over all the individual classifiers. This is called soft voting. In this paper, we use soft voting approach to predict the class.

Chapter 5

Deep Learning Methods

5.1 BERT

BERT stands for Bidirectional Encoder Representations from Transformers. BERT (Devlin et al. 2018) is the state-of-the-art natural language model published by the Google AI Language team in 2019. It has created a major breakthrough in the field of NLP by providing impressive results in many natural language processing tasks, such as question answering, text generation, sentence classification, and many more. BERT, unlike other popular embedding models is a context based model. This means it will understand the context and generate the embedding for a word based on the context. For example, the word 'bank' in sentence "We got loan from the bank" refers to the banking institution, while in sentence "She sat on the river bank", the word 'bank' refers to the river bank. So, for word 'bank' it will generate two different embeddings. More sophisticated methods such as Recurrent Neural Networks have some contextual understanding capabilities; it considers only the preceding words when generating the embedding for the word.

5.1.1 Model Architecture

BERT is pre-trained from unlabelled Wikipedia (2,500M words), and Book corpus (800M words) to obtain contextual embeddings. BERT is trained with a different number of layers,hidden units, and attention heads, while otherwise using the same hyperparameters and training procedure. The two most popular versions of BERT are: BERT-BASE (Layers=12, Hidden states‘=768, Attention Heads=12, Total Parameters=110M) and BERT-LARGE (Layers=24, Hidden states =1024, Attention Heads =16, Total Parameters=340M). The BERT-Large model requires significantly more memory than BERT-Base. It has long been known that increasing the model size will lead to continual improvements on large-scale tasks such as machine translation and language modeling. Further, the model also has case versions, where the the 'cased' version of the model preserves the original text cases for tokenization whereas the 'uncased'

BERT ensures that the text has been lower-cased before tokenization. For example, John Smith becomes john smith. The vocabulary size of BERT-Base (uncased) is 30,522 words. In this paper, we use BERT-BASE (uncased) for the classification task.

The model is based on Transformers (Vaswani et al., 2017) which uses attention mechanism. The transformer consists of encode-decoder architecture. The transformer's encoder architecture is presented in Fig 5.1. The encoder layer generates the representation of the input sentence. We see that all encoder blocks are identical. Each encoder block consists of two sub-layers:

1. Multi-head attention
2. Feedforward network

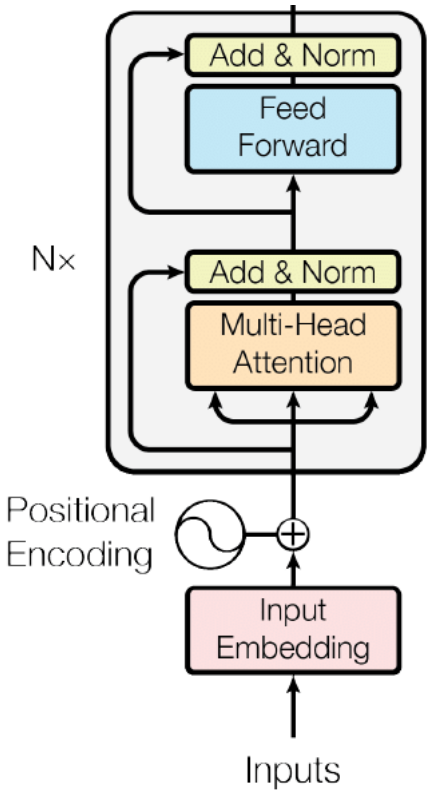

Figure 5.1: Transformer Encoder Architecture [1]

[1] https://arxiv.org/pdf/1706.03762.pdf

Multi-head attention

To compute the representation of a word, the self-attention mechanism relates the word to all other words in the sentence. First, we get the embeddings representation of each word. Note that embeddings are just the vector representations of words and values of the embeddings and values for embeddings will be learned during training. Then we can represent the input sequence as [*sequence length* $\times$ *embedding dimension*] input matrix. Next, we initialize three new weight matrices - query matrix Q, key matrix K, and value matrix V. These weights matrices are randomly initialized and their optimal values will be learned during training. The self-attention mechanism as illustrated in Fig. 5.2. follows a series of steps summarized as follows:

1. First, we compute the dot product between the query matrix and key matrix QK^T, and get the similarity scores, which helps us understand how similar each word in the sentence is to all other words.

2. Next, we divide the QK^T by the square root of the dimension of the key vector $\sqrt{d_k}$; this is useful in obtaining stable gradients

3. Next, we apply the softmax function to normalize the scores and obtain the score matrix; applying the softmax function would allow the score to range between 0 and 1, and sum of these scores equals to 1

4. At the end, we compute the attention matrix Z, by multiplying the score matrix by value matrix, V

In the encoder,the equation is applied to every input sequence in the batch. To ensure that our results are accurate, instead of a single attention matrix, We compute multiple-attention matrices and concatenate their results. The idea behind using multi-head attention is that instead of using single-attention head, if we use multiple attention heads, our attention matrix will be more accurate. The independent attention output scaled-dot product attention is

$$Z = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5.1}$$

Feedforward network

The feedforward neural network consists of two dense layers with RELU activations. It takes in the attention-matrix as output and returns the encoder representation. The parameters are the same over the different positions of the sentence and different over the encoder blocks.

[2]https://arxiv.org/pdf/1706.03762.pdf

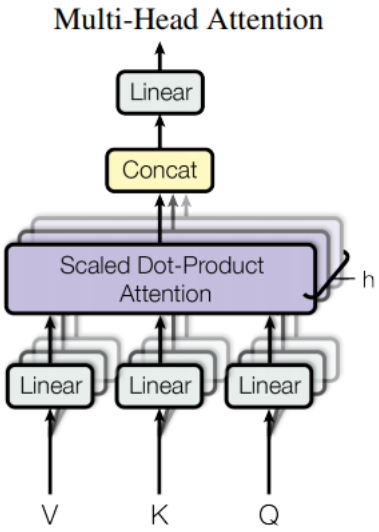

Figure 5.2: Multi-Head Attention [2]

5.1.2 Input Representations

To enable BERT to handle a variety of down-stream tasks, we transform the input into embeddings using three embedding layers - Token embeddings, Sentence embeddings, and Transformer positional embeddings. The embeddings from each layer is then aggregated and fed to the BERT model for training. The three embedding layers and their aggregation is illustrated in Fig. 5.3.

1. Token Embeddings: First, we tokenize the input sentence and obtain the tokens. Then, we add a new token called [CLS] at the beginning of the first sentence. Finally, we convert these tokens into embeddings using an embedding layer called token embedding. Note that the value of token embedding will be learned during training. We see in fig 5.3. the sentence "my dog is cute" is tokenized as tokens = [CLS, my, dog, is, cute]

2. Segment Embedding: Segment embedding is used to distinguish between two sentences. After we tokenize the two sentences, we add token [SEP] at the end of every sentence to indicate the end of a sentence and beginning of a new sentence. Apart from adding [SEP] token we have to give an indicator to the model to distinguish between the two sentences, we feed all the tokens to the segment embedding layer. The sentence embedding layer returns only either of the two embeddings, EA or EB as an output. That is, if token belongs to sentence A, then the token will be mapped to embedding EA, and if token belongs to sentence B, then it will be mapped to embedding EB.

3. Position Embedding: Position Embedding layer gives us the information about the position of a word in the sentence. A positional encoding is a dense vector that encodes the position of word within a sentence. The positional embedding is added to the word embedding of each word in the sentence.

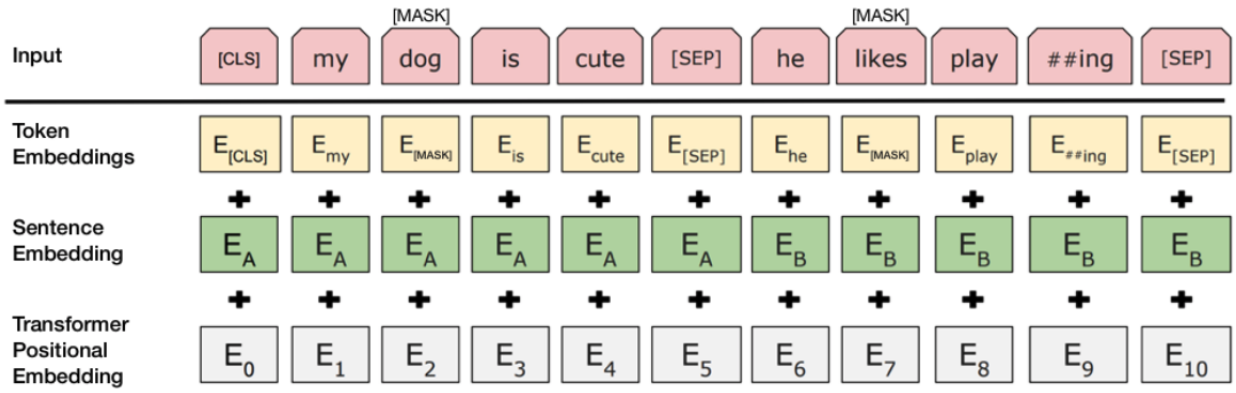

Figure 5.3: BERT Input Representation [3]

5.1.3 Fine-Tuning BERT

The self-attention mechanism in the transformer allows BERT to be fine-tuned for many downstream tasks such as text classification. In this paper, we fine-tune the model for the classification task Quora questions dataset. We first transform the inputs in a way that is accepted by BERT. The input is transformed the using three embeddings layers - token, segment, and position embedding. We then aggregate the embeddings from all three layers to compute the final representation of input which is then fed to the encoder layer of the transformer. The output from the encoder layer is fed into the classification layer, which is a sigmoid layer for binary classification and soft-max layer for multi-class classification. Since we solve a binary classification problem, we use sigmoid layer to output the probabilities of the question being in class sincere and insincere. The class having the highest probability is finally predicted. The fine-tuning process is also illustrated in Fig. 5.4. The fine-tuning tasks take much less time to train a new set of training data than training the model from scratch. The hyperparameters in Table 5.1. were tuned for the classification task. The batch size, a hyperparameter which controls the number of training samples to work through the model before updating model's weights, was set to 16. The length of the sequence was restricted to 60 words. It takes a long time to train longer sequences, while shorter sequences are faster to train. The

[3] https://arxiv.org/pdf/1810.04805.pdf
[4] https://aclanthology.org/2021.nllp-1.22.pdf

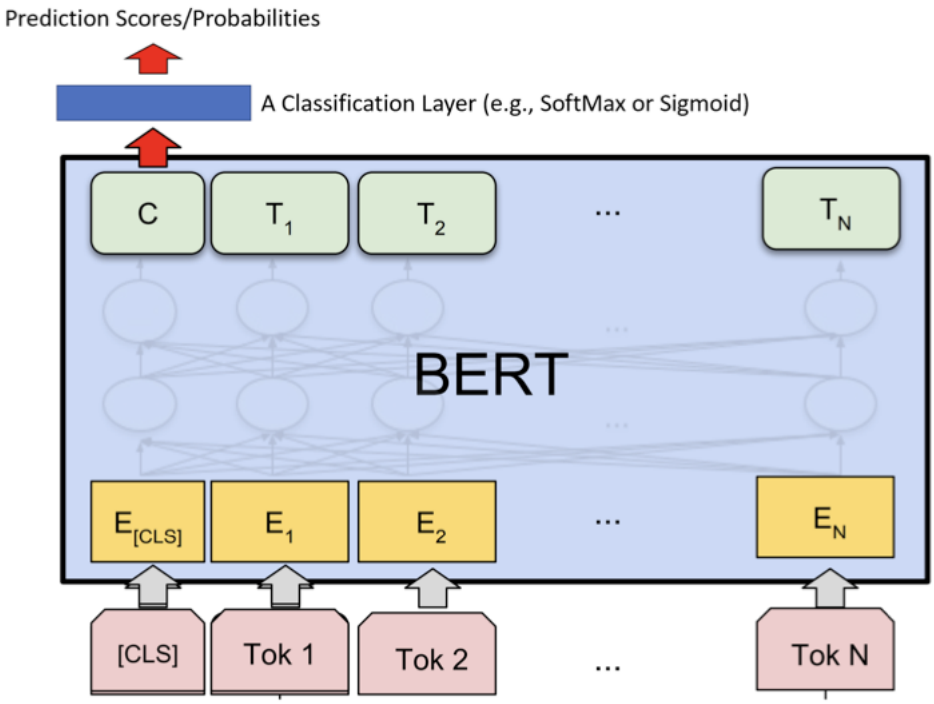

Figure 5.4: Fine-Tuning BERT model on a classification task [4]

maximum sequence length that can be processed by BERT is 512 words. The learning rate. The epochs, a hyperparameter that defines the number of passes of the entire training set the model will complete, was set to 3.

Hyperparameters	Value
batch size	16
sequence length	60
learning rate	2e-5
epochs	3
weight decay	0.01
optimizer	Adam

Table 5.1: Hyperparameters used for BERT fine-tuning

Chapter 6

Results

In this section, we discuss the scoring results for Logistic Regression, Naive Bayes, Support Vector Machines, Decision Tree, Random Forest, Voting Classifier, and BERT on the validation dataset (Table. 6.2). The dataset has been split into training and validation sets. The training set, comprising of 80800 observations, is used for training while the validation set, comprising of 673316 observations, is used for testing. We see that The performance of BERT model is slightly better than that of all other machine learning classifiers. Further, we perform grid search using the scikit-learn library in Python to tune the hyperparameters of the machine learning classifiers. We train the classification models with best set of hyperparameters (Table. 6.1); which gives the highest recall and f1-score.

Model	Hyperameters
Logistic Regression	C:3
Support Vector Machines	C: 1, gamma: 1, kernel: rbf
Decision Tree	maxdepth: 500, criterion: gini
Random Forest Classifier	maxdepth: 500, estimators: 2000, criterion: gini

Table 6.1: Hyperameters of Machine Learning classifiers

The metrics used for evaluating the classification models are precision, recall, f1-score, and accuracy. These metrics are measures of goodness of a model; this means it helps us assess how well the model is performing on the validation set and whether or not the model is able to generalize well on the validation set. Since this is an imbalanced dataset, accuracy is not a good measure for evaluation. Instead, the importance is given to measures like precision, recall, and f-1 score. In the context of this problem, we would want to lower false negatives (insincere questions classified as sincere), which corresponds with better recall.

1. Precision: measures the proportion of positively predicted labels that are actually correct. When there is uneven class distribution, there will be

a trade off between precision and recall. That is, improving precision typically reduces recall and vice versa.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{6.1}$$

2. Recall (Sensitivity): measures the proportion of instances of a class that were classified correctly. The table 6.2. reports the recall of insincere class (positive class in our case). The higher the recall score, the better the machine learning model is at identifying both positive and negative examples. Recall is also known as sensitivity or the true positive rate. A high recall score indicates that the model is good at identifying positive examples. Conversely, a low recall score indicates that the model is not good at identifying positive examples. The Recall ranges from 0.87 *to* 0.90 across all classifiers, which is really high given that insincere was a rare class in the dataset and that there were only 40,000 observations for this class while training. The highest Recall of 0.90 was reported by BERT.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{6.2}$$

3. F-1 Score: measures the weighted average of Precision and Recall. Therefore, its takes both False Positives and False Negatives into account. This measure becomes extremely useful when we have uneven class distribution. The recall ranges from 0.45 *to* 0.53 across all classifiers. The highest F1-score of 0.53 was reported by BERT.

$$F1 = \frac{2 * (Recall * Precision)}{Recall + Precision} \tag{6.3}$$

4. Accuracy: measures the proportion of instances that were classified correctly. It is defined as the ratio of true positives and true negatives to all positive and negative observations.

$$Accuracy = \frac{TruePositives + TrueNegatives}{TruePositives + TrueNegatives + FalsePositives + FalseNegatives} \tag{6.4}$$

Model	Precision	Recall	F-1 Score	Accuracy
Logistic Regression	0.31	0.87	0.45	0.88
Naive Bayes	0.22	0.89	0.35	0.80
Support Vector Machines	0.31	0.88	0.46	0.88
Decision Tree	0.23	0.81	0.36	0.83
Random Forest Classifier	0.25	0.88	0.39	0.83
Voting Classifier	0.28	0.89	0.43	0.86
BERT	0.37	0.90	0.53	0.90

Table 6.2: Results of Machine Learning classifiers & *BERT*

Chapter 7

Conclusion

Quora is one of the most popular platform to ask questions and connect with people who contribute unique insights. One of the key challenges that the website faces today is detection of toxic content in the posted questions. To address this challenge, Quora posted the dataset - consisting of 1.3 million questions, on Kaggle. The goal of this Kaggle challenge was to identify whether the question posted on Quora is insincere or not. To solve this challenge, we implemented various machine learning classifiers such as Naive Bayes, Logistic Regression, Decision Tree, Support Vector Machines, Random Forest, and Voting Classifier. We also implemented BERT, a state-of-art natural language processing model developed by Google AI team in 2019. We used Grid Search approach to optimize the machine learning classifiers. To optimize BERT, we tuned various hyperameters such as batch size, learning rate, epochs, weight decay. After implementing different models and parameter adjustment combinations, it can be stated that Bert model performed the best with 0.90 recall and 0.53 f1-score. Overall it can be said that the project is able to classify between sincere and insincere questions and overall got reasonably successful results.

Chapter 8

References

1. Bengio, Y., Courville, A., Vincent, P. (2013). Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8), 1798-1828.

2. Joos, M. (1950). Description of language design. The Journal of the Acoustical Society of America, 22(6), 701-707.

3. Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. Studies in linguistic analysis.

4. Salton, G. (1971). The SMART retrieval system—experiments in automatic document processing. Prentice-Hall, Inc.

5. Salton, G., Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information processing management, 24(5), 513-523.

6. Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

7. Pennington, J., Socher, R., Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

8. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

9. Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.