Quora Insincere Questions Classification using Attention based Model

Snigdha Chakraborty[1], Megan Wilson[2], Sulaf Assi[2] and Dhiya Al-Jumeily OBE[1]

[1] School of Engineering and Technology, Liverpool John Moores University, L3 3AF, UK.
[2] School of Pharmacy and Bimolecular Science, Liverpool John Moores University, L3 3AF, UK.

Abstract. The online platform has evolved into an unparalleled storehouse of information. People use various social question-and-answer websites such as Quora, Form-spring, Stack-Overflow, Twitter, and Beepl to ask questions, clarify doubts, and share ideas and expertise with others. An increase in inappropriate and insincere comments by users without a genuine motive is a major issue with such Q & A websites. Individuals tend to share harmful and toxic content intended to make a statement rather than look for helpful answers. In the world of NLP, BERT has been a game-changer. It has dominated performance benchmarks and thereby pushed the limits of researchers' ability to experiment and produce similar models. This resulted in improvements in language models by introducing lighter models while maintaining efficiency and performance. This study aims to utilize pre-trained state-of-the-art language models and understand whether posted questions are sincere or insincere with limited computation. To overcome the high computation problem of NLP, the BERT, XLNet, StructBERT, and DeBERTa models are trained on three samples of data. The metrics proved that even with limited resources, recent transformer-based models outscore previous studies with remarkable results. Amongst the four, DeBERTa stands out with the highest balanced accuracy, macro,and weighted f1-score of 80%, 0.83 and 0.96, respectively.

1 Background

1.1 Introduction

The effort aims to develop an efficient system for classifying insincere questions on the Quora dataset. The questions are classified using the CRISP-DM (Cross-industry procedure for data mining) approach, which aids in the planning, organisation, and execution of data analysis initiatives.

1.2 Business Understanding

Business understanding the Internet and social media are excellent tools for exchanging information quickly. To gain input on their questions, people nowadays use community forums like Quora. Some people abuse such platforms to promote hostility or even false information by posing personal opinions as inquiries instead of genuine ones. Questions aimed at a certain community, gender, or race decrease the standard of content on Quora and are unsafe for the on-lookers who are truly seeking information.

As a result, it is critical to recognise these false queries before they reach a wider audience and restrict them for a better tomorrow.

1.3 Data Understanding

In this research we will use the Quora-insincere-questions classification provided by Kaggle (Dataset, 2019). The dataset consists of a train set and a test set, respectively. The train set consists of 1.3M rows with required labels corresponding to each row. The training data includes different questions asked by users in English and are not guaranteed to be perfect. Hence, there is a certain amount of noise in the dataset. Questions are marked as 0 (Sincere) and 1 (Insincere). The columns available in the dataset are: -

- qid - unique identifier for the question
- question_text - question text posted in Quora 33
- target - a question labelled as 1 for “insincere” and 0 for “sincere”

1.3.1 Exploratory Data Analysis

The training set consists of 1306122 unique values, whereas the test set has 375806. The dataset size may lead to higher processing time or memory issues. Based on target variable distribution, it's observed that the data is highly imbalanced with 1.2M belonging to sincere questions and just a small proportion of 80k belonging to target class 1 for insincere questions. The same needs to be handled properly before building a model to avoid biases towards the majority class. Data needs to be pre-processed before training a model upon it. There is no missing data found, thus removing the need for missing data imputation.

3.4 Data Pre-Processing

Data pre-processing is the modification, or encoding, of data before it is used for analysis. It is a data mining technique performed to ensure an improvement in performance. In other words, it is the process in which the raw data is transformed or formatted so that it becomes easier for a machine to parse. To train the model efficiently the textual queries will be pre-processed as follows: -

I. Remove special characters, numeric characters, punctuation, and duplicate records The question text column in the dataset can contain smileys, numeric or roman literals, punctuations, or special characters as in "?". Such characters are to be removed with the help of lambda expressions by utilising regex to filter the format.

II. The distinct contraction terms are enlarged with the use of python libraries such as expanding contractions using a pre-defined contractions dictionary map, such as replacing "couldn't" with "could not".

III. Remove all unwanted spaces from the text.

IV. Truncation of sequence length. Bert's maximum sequence length is 512, but for efficient memory usage, those can be truncated to a shorter length before being encoded by the tokenizer for use. After pre-processing, the cleaned question texts are tokenized and turned into vectors to suit the pre-trained transformer-based model's needed input format. To preserve the syntactical meaning of the questions, stemming or lemmatization will be avoided. This is done to ensure that the precious information of the data is restored.

1.4 Tokenization, Truncation & Padding

Tokenization is one of the main steps in breaking down a text into smaller chunks of words called tokens. The tokens are then converted into vectors of numbers. The build tensor is then fed to the model along with any additional input required to make the model work seamlessly. The splitting of text is done by a tool called a tokenizer. One can either construct their tokenizer class or can directly use the existing AutoTokenizer class. It is advised to use the associated pretrained tokenizer while using a pre-trained model. This ensures that the tokens are built the same way as done for the pre-training corpus and will use the same vocab during model pre-training. The pre-training vocab can be downloaded from the AutoTokenizer class. When the batch sequences are of different lengths, padding is used to maintain the length of the sequence. Similarly, when the length is longer than needed, truncation is used to shorten the length. Both are performed by specifying the value in the max_seq_length parameter.

1.5 Modelling

In this study the performance of the four state-of-art models viz. BERT, XLNet, StructBERT and DEBERTa on the dataset. All the mentioned models will be trained using python and its libraries in TPU (Tensor Processing Unit) or GPUs (Graphical Processing Unit). TPU is a powerful custom-built processor and provides a better performance against GPUs. To restrict memory issues model can be evaluated on a smaller batch size. Truncating the max length of the sequence will also help to utilize the memory efficiently. Based on model performance epochs and hyper tuning can be decided. The evaluation metrics of each model will be recorded for the outcome. The models to be used are briefly discussed in the following sections:

1.5.1 BERT

BERT (Devlin et al., 2018a; b) is a transformer-based machine learning framework for NLP that was pre-trained using only a plain text corpus by the Google AI team. The two variations of the original English-languages BERT model proposed are: BERTBASE consisting of 12 encoders with 12 bidirectional self-attention heads; and BERTLARGE consisting of 24 encoders with 16 bidirectional self-attention heads.

1.5.2 XLNet

The author of the research (Yang et al., 2019) presented XLNet, a generalised auto-regressive pre-training technique that incorporates the best aspects of Transformer-XL and auto-encoding while resolving their respective limitations. It enhances the pre-training architecture by integrating the segment recurrence mechanism and relative encoding scheme of TransformerXL. In terms of size, the XLNET base model is like BERT, with 12 layers, 768 hidden vector sizes, and 12 attention heads. On 20 tasks, it outperformed BERT, and on 18 NLP tasks, it reached SOTA results.

1.5.3 StructBERT

The author identified some scope of improvement in BERT and RoBERTa and extended it to build a new model, StructBERT in (Wang et al., 2019). The model proposes a new type of contextual representation that adds language structures into BERT pre-training with two novel linearization strategies. It enhances the BERT masking strategies by using sequential order at the word and sentence levels, which compels one to reconstruct the correct order of words and sentences as in Fig 1.4. The architecture uses a multilayer bi-directional transformer network, as explained in (Vaswani et al., 2017). It improved the SOTA on the GLUE benchmark to 89.0, the F1 score on SQuAD v1.1 question answering to 93.0, and the SNLI accuracy to 91.7.

1.5.4 DeBERTa

The model DeBERTa in (He et al., 2020) proposes a new architecture with disentangled attention and an enhanced mask decoder. In addition to BERT, disentangled attention accounts for position-to-content self-attention as well. It has two separate vectors representing content and position and self-attention is calculated between all possible pairs. The enhanced mask encoder provides both absolute and relative position information. And to enhance the generalization, a scale-invariant fine-tuning regularisation algorithm has been used. The model trained on half of the data used in RoBERTa achieved an improvement of +0.9% on MNLI, +2.3% on SQuAD V2.0, and +3.6% on RACE.

1.6 Evaluation

Model evaluation is an important part of the machine learning model development process. There are multiple ways to evaluate a model performance out of the four common metrics used are: I. Accuracy – This is the percentage of all the correct predictions made from the total number of predictions. This is the widely used most common metric to validate a model. However, for an imbalanced dataset, it's not a good indicator of the model's behaviour as it tends to incline towards the majority classes present in the dataset. Hence, even if the model accuracy is high, it is highly possible that the model may fail to generalize the pattern and will not perform better in an unseen dataset. Balanced accuracy is the mean of specificity and sensitivity.

Eq 1.1

$$\frac{TP + TN}{TP + FP + TN + FN}$$

II. Precision – This is the percentage of all the correct positives out of total predicted positives. In other words, it checks how much a model is able to generalize the behaviour in terms of quality. It is preferred where false negative has less impact such as YouTube ad recommendation.

Eq 1.2

$$\frac{\mathrm{TP}}{\mathrm{TP+FP}}$$

III. Recall – This is the percentage of predicted positives out of the total number of actual positives present in a dataset. In other words, it checks how much a model failed or missed to identify the correct class in terms of quantity. It is preferred where false negative are equally important such as clinical analysis

Eq 1.3

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

IV. F1-Score – This is the harmonic mean of both precision and recall. In other words, it identifies the performance by determining the actual positives predicted correctly and the model did not even misclassify positive to negative. Thus, the higher the F1 score better 39 the model. As the dataset is highly imbalanced, F1-Score will be used as a primary model evaluation metric. The Macro F1 Score is the arithmetic mean of the F1 score for each class label. A Weighted F1 Score is the sum of each class's F1 Score with respect to each class’s weight.

Eq 1.4

$$\frac{2}{\frac{1}{\mathrm{precision}}+\frac{1}{\mathrm{recall}}}=\frac{2 * \mathrm{precision} * \mathrm{recall}}{\mathrm{precision}+\mathrm{recall}}$$

2.0 Analysis

2.1 Data Preparation

The dataset is downloaded from Kaggle and loaded into the GPU-enabled Google Colaboratory. The implementation is done using Python libraries. The dataset contains 3 variables: a unique question id, actual question text, and a binary target variable indicating the sincerity of a question. The qid is used to uniquely identify a row and is not needed for our analysis, so the column is dropped. Thus, now the data contains one independent variable (the actual question posted by the user) and the target variable (sincere or insincere). There is no missing value found, and hence, missing value identification and imputation are not needed. The platform provided a respective dataset for training and testing. The training dataset is used for data analysis and data pre-processing, while the test set is used to validate the model's performance on unseen data.

2.2 Data Analysis

2.2.1 Word Cloud

Words like "work," "help," "book," "best," and so on are frequently used in standard questions. They are mostly neutral and indicate someone looking for advice or similar things. Insincere questions contain words like "atheist," "Trump," "Muslim," "black," etc. The difference is obvious. Instead of the generic advice topics (like relationships, work, education, etc.), the most frequently used words are all very political (for example, the frequent use of Donald Trump's name). This also shows why a simple sentiment analysis of words is not sufficient in this case.

2.2.2 N-Grams

Word clouds are a great tool to get a first impression of the word frequencies, but without looking at the actual values as well, they might be misleading. The N-grams method is used to find the most frequently used words and phrases in each question type. N-grams can be defined as continuous sequences of items in a text that help perform sentiment analysis on a word. A single word such as "bad" will be identified as discriminative. But when it gets combined with another token like "not bad," the label of sincerity changes to 44 positive.

2.2.3 Sentiment Analysis

Sentiment analysis helps us understand the mood and context behind the sentence. TextBlob, an NLP Python library, uses nltk to return the subjectivity and polarity of a sentence. Polarity can be defined as the orientation of the sentence. It has a range of [-1,1], with -1 denoting negative sentiment and 1 denoting positive feeling. Subjectivity refers to the semantic descriptors that aid in the fine-grained analysis of emoticons, exclamation marks, emojis, and so on. It has a range of [0,1]. The degree of opinion and factual data in a text is measured by subjectivity. The subjectivity value towards 1 conveys that the text contains opinion rather than factual data. The package calculates it by looking into the intensity of each word in a sentence based on a lexiconbased approach. The polarity and subjectivity of 0 indicate that text isn't the best.

Results higlighting polarity distribution in the insincere set shows that there is a tendency towards polarity within the insincere group. The distribution turns towards (-1). The mean polarity and subjectivity come out to be 0.03 and 0.36, respectively. When using the sentiment scores approach (from nltk.sentiment.vader), a concentration towards the negative values is found, and the mean was quite low at -0.099.

2.2.4 Feature Extraction

Meta-features are created as a part of feature extraction and their distribution between the classes are studied. The list of features constructed is as follows: - I. Number of words in the text II. Number of characters in the text III. Number of unique words in the text IV. Number of special characters in the text V. Number of stop words VI. Number of upper-case words VII. Mean, Max, Min length of the words. The spread of the extracted features among sincere and insincere classes is analysed with the help of box plot. It is observed that Insincere questions have a larger word count, unique word count, and special character count than sincere questions (apart from a few sincere outliers). The outliers in the boxplot shown in Fig 4.6 state question_length of sincere text is approximately 800. Insincere can be spam questions, ads, and so on. Based on the number of special symbols, insincere can be latex math formulas or questions containing icons or emojis and non-English characters. It is also observed that the sincere queries are longer than disingenuous questions, with a maximum length of 134.

2.3 Data Cleaning

Due to the self-attention mechanism, the models trained on huge corpus text data can adapt and learn features and generalize the behaviour of data on their own. Hence, the feature engineering steps are skipped and proceeded with the data cleaning process. The subsection includes different data cleaning steps performed before model training. Post data cleaning the values are stored in a new column named “cleaned_text”.

2.3.1 Expansion of Contractions

Contractions are the shorthand form of text by dropping letters and replacing them with apostrophes, such as "shouldn't" or "don't". To ensure text standardization and dimensionality, the contracted words present in question_text are expanded using the Python Contractions 48 Library. The library is installed and imported, and the question text is passed as an argument to the fix function.

2.3.2 Punctuations and Special Characters Removal

The question text in the dataset contains various emoticons, punctuation, emojis, symbols, and so on. These special characters are removed using regular expressions (regex). Lambda expressions with the tqdm progress_apply function are used on the question_text column to apply a regex filter to each text.

2.3.3 Removal of numbers and spaces

The question_text in the dataset contains numeric data. The numeric expressions are filtered out using lambda expressions and are removed from the text. After expanding the contractions and removing characters, extra spaces are found in the question text. All the leading, trailing, and double spaces resulting due to removal of punctuation are trimmed. Finally, the data is checked for the presence of any duplicate records and clean data along with the target variables is retained in a new CSV file for implementation.

2.4 Data Sampling

The platform provided different datasets for training and testing samples. And each set is large, with millions of records. To achieve better model performance and state-of-the-art results with efficient resources in hand and limited computational power, the models are trained by considering different fractions of data samples per iteration. The random sample of items is taken from the original dataset by using the sample () function of the Pandas DataFrame class. The function takes frac as an argument to return 5%, 10%, and 50% of random samples per iteration while keeping the target variable imbalance consistent.

2.5 Data Split

The extracted fraction of the dataset is again split into train and validation sets in the ratio of 80:20 using the train test split function from the sklearn library in Python. Models are trained on a training set and tested on a validation set. The testing set is used to predict the target outcome. Due to its expensive computational nature, K-fold cross-validation is avoided.

2.6 Hyperparameter Tuning

The libraries provide a list of hyperparameters to be tuned as per the dataset and task requirement. In this study, from an entire list, only a selected set of hyperparameters is used. The value is determined after multiple iterations and thorough consideration. The reason for selecting each hyperparameter and the final value is chosen for implementation is explained below: -

• Epoch - The transformer models are pre-trained on a large set of data, and finetuning the model with 2, 3, or 4 epochs yields outstanding results. When fine-tuning with 100k+ data, researchers found that training with greater epochs does not improve the outcomes considerably. Models are initially trained for seven epochs; however, it is discovered that after the fourth epoch, the models tend to overfit. As a result, epoch = 4 is used to train our models.
• Learning Rate - For text classification tasks, a learning rate of 1e-5 or 2e-5 is chosen to avoid model overshooting from local minima.
• Dropout - After studying the literature on text classification, the dropout value is determined. The models are trained with a dropout of 0.1 at the start. The models are found to perform well in the training data. The dropout value has been increased from 0.1 to 0.3 to improve performance. When the model with 0.1 dropout and 4 epochs is implemented, it is discovered that it generalised better. As a result, 0.1 is chosen as the dropout value for all models.
• Train Batch Size - The desirable batch size is determined since a smaller value result in a lengthier time for training and a highervalue result in non-convergence to global minima and memory issues. Hence, with 4 epochs, a train batch size of 32 is chosen to obtain a proper trade-off between model training duration and memory performance.

2.7 Implementation

The PyTorch Transformer, built on top of the HuggingFace (HuggingFace Transformers, 2022) and SimpleTransformer (Classification Models - Simple Transformers, 2022) libraries, provides an easy-to-use framework for pre-trained language modelling for a variety of NLP workloads. The Transformers provide APIs for easily downloading and using pre-trained models on a given text, as well as fine-tuning them on our own datasets, whereas the SimpleTransformers' ClassificationModel class is used for binary class text categorization. TensorFlow 2.0+ and PyTorch 1.1.0+ are both required for the implementation. The model's name and type are supplied as input parameters for each model trained. The implementation details of BERT, XLNet, DeBERTa, and StructBERT with tweaked hyperparameters are detailed in the subsections below.

2.7.1 Model Training

The pre-processed data saved in the CSV file is loaded in Google Colab. The different models are trained for three samples of data. The base models are trained with default arguments while the finetune models are trained with tuned hyperparameters. The implementation steps remain the same for all 4 models. The Python logging module is used to log model execution details, to help us to track any exception raised during training. The two different ways of implementation tried are HuggingFace PyTorch Transformer and Simple transformer. All the steps mentioned below are implemented for each dataset sample, i.e., 5%, 10%, and 50%. The 51 final trained model is saved, and the model is evaluated on the validation data loader while labels are predicted on an unseen test dataset

2.7.2 PyTorch Transformer

The model dependencies are installed, and the PyTorch and Tensor Flow versions are validated for smooth execution. This method is used to note the performance of the pre-trained models with passed arguments. For the StructBert model the Alice Mind code repository is cloned to download weights, configuration, and pre-trained model checkpoint to be used for the binary classification tasks. All the steps outlined below remains the same. The steps followed included: I. Tokenization: At the beginning and end of each sentence, the special tokens [CLS] and [SEP] are added. One represents the class of input and the other is used to separate the two sentences in the same input. The output label of classification is decided by the last hidden state of the [CLS] token. The sentences are then tokenized with the tokenize function, and each tokenized text is converted to token ids using pre-trained tokenizers as mentioned in Table 4.3. These tokens act as vectors for the embedding layer. II. Padding: The models expect all the input to be of the same length. To ensure the same, the maximum sequence length is initialized to 128 and the input ids are padded accordingly with the pad_sequences function. If a sentence is long, it's truncated, and if it is short, it is padded with 0. III. Attention mask: An attention mask is created with a mask of 1s for each token, followed by 0s for padding. These masks act as indices to determine which token the model needs to attend. IV. Data Loader: The input ids and the attention mask are split into training and validation sets, respectively. The respective training and validation inputs/masks are converted into torch tensors, the required data type for our model. The batch size is initialized to 32, and an iterator is created of our data with the torch DataLoader. This helps save memory during training because, unlike a for loop, with an iterator, the entire dataset does not need to be loaded into memory. V. Optimization and Training: The pre-trained model with a single linear classification layer is loaded and num_labels = 2 is passed as an argument. For optimization, a weight_decay_rate of 0.01 is passed along with a learning rate of 2e-5 to the AdamW optimizer with an epsilon of 1e-8. Binary Cross entropy with a dropout of 0.3 is the default selected loss function. Model arguments contained all the hyperparameter information for our training loop. The weight decay rate is applicable to all layers except all bias and layer norm layers in the AdamW optimizer. Finally, the model is trained for 2 or 4 epochs and the metrics are recorded

2.7.3 Simple Transformer

The SimpleTransformers library has two task-specific classification models. In this study, the ClassificationModel class is used to train the base and fine-tuned model by modifying the default set of arguments. The class provides extensive options for configuration to update the default model argument as per the use case. The configuration is passed as a Python dictionary to the ModelArgs class. In all Simple Transformer models, CUDA is enabled by default but can be disabled by passing a Boolean to the use_cuda parameter. The steps performed are detailed below. I. Initialization: The ClassificationModel class is created by passing the model_type and model_name as mentioned in Table 4.4. The model type must be one of the supported models, and the model's name assists in obtaining the precise architecture and trained weights for use. The model code is used to specify the model type.-
o To utilize the available GPU, use_cuda = True
o To override the default arguments with the arguments passed as-

▪ num_train_epochs = 4, the number of epochs the model will run for
▪ train_batch_size = 32, the training batch size ▪ max_seq_length = 128, the maximum sequence length for the model to support
▪ learning_rate = 1e-5 or 2e-5, the learning rate for training
▪ output_dir = output/, the directory to save model checkpoints and results
▪ overwrite_output_dir = True. If True, it will overwrite the output folder to store the latest model result. Due to multiple trials and to save memory space it is assigned a Boolean true in this study.
o For the rest, default arguments are used, and the parameters are not provided.
II. Split: The dataset sample is split in the ratio of 80:20 into training and evaluation data. III. Training: The initialised model is used to call the train_model () function, which instructs the model to train on the training dataset.

2.7.4 Evaluation of Model Performance

The classifiers are built using the training dataset and assessed using the performance evaluation matrix. The models are trained on three fractions of the dataset sample. This is a two-step procedure. In the first step, the best value in the training states across all models will be picked based on performance measures. In the second step, the performance of the models is compared to find the best model. For both the class labels, the confusion matrix and classification report are utilized to extract various measures such as accuracy, true positive, false positive, true negative, false negative, sensitivity or recall, precision, f1-score, Area Under Curve (AUC), Area Under Precision recall Curve (AUPRC), and Mathews Correlation Coefficient (MCC). The class labels "insincere" (1) and "sincere" (0) are classed as "Yes" and "No," respectively.

3.0 Results and Discussion

3.1 Model Evaluation

Following model training, each model is evaluated on a validation set. As the data is highly imbalanced, accuracy will not be the correct metric to understand the model's performance. Hence, F1-score is used for the same. The macro averaged f1-score is noted as it provides the average arithmetic mean of each class. The MCC is also a reliable statistical rate to understand the obtained prediction. It returns a high score if the prediction is good in all four quadrants of the confusion matrix. The results in all matrix categories are proportional to the size of the dataset belonging to each class. Balanced Accuracy is calculated as it will give the arithmetic average of sensitivity and specificity. 5.3 Experiment 1 - Transfer Learning with Original Hyper-Param Settings The first set of experiments is performed with default settings, without modifying the values of any hyperparameter. Initially, the models are trained and evaluated on 5% of the data, and the percentage is increased in subsequent iterations. The dataset contains 52177 training and 13045 validation records. Keeping the imbalance ratio consistent, the test set contains 12240 sincere and 805 insincere records. Each model is trained and validated, resulting in four executions in this experiment. Table 1.1 shows the standard value of the hyperparameter used in this specific execution. Table 1.2 depicts the performance metrics recorded from the validation set.

Table 1.1 Exp 1: Hyper-parameters - Values

Hyper-parameter	Value
adam_epsilon	1e-8
eval_batch_size	8
learning_rate	4e-5
max_seq_length	256
n_gpu	1
num_train_epochs	1
train_batch_size	8
optimizer	AdamW

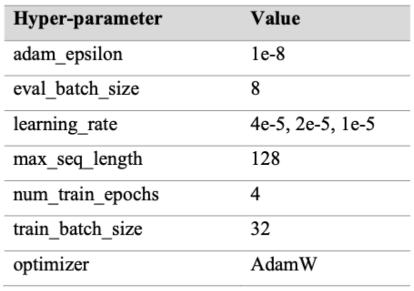
Table 1.2 Model Results- Default Settings

SL. No	Model	Acc	Pr	Recall	F1-Score	MCC	AU-ROC	AU-RPC	Train Time	Eval Loss
1	BERT	0.95	0.75	0.38	0.50	0.51	0.94	0.60	40	0.15
2	XLNET	0.95	0.62	0.52	0.56	0.61	0.95	0.68	40	0.16
3	DeBERTa	0.95	0.58	0.59	0.58	0.54	0.85	0.60	43	0.17
4	StrucctBERT	0.90	0.59	0.36	0.48	0.46	0.80	0.40	120	0.20

3.2 Experiment 2 - Transfer Learning with Modified Learning Rate (LR)

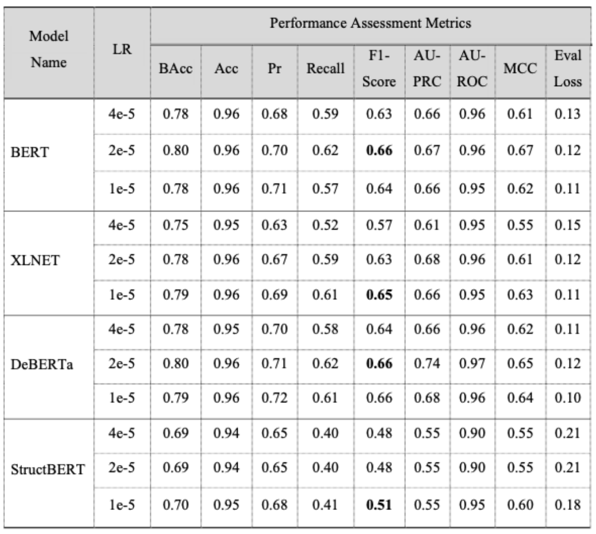
After performing the first set of experiments with the default model argument settings, the second set of experiments is done by changing the learning rate. The training batch size is kept at 32, the eval batch size at 8, the maximum sequence length is 128 and the number of training epochs is 4. The number of GPUs assigned is 1. With a 5% data sample, the distribution of data across the train and validation set remains the same as in experiment 1. In each iteration, the value of the learning rate is changed, and metrics are noted. Each model is tried with 3 different values, resulting in a total of 12 executions in exp-2. Table 1.3 shows the standard value of the hyperparameter used in this specific execution. Table 1.4 depicts the performance metrics recorded from the validation set for different learning rates used. Table 5.4 shows that BERT and DeBERTa produced better results with an LR of 2e-5. And XLNET and StructBERT produced it with an LR of 1e-5. Thus, the best-performing learning rate is chosen for the final set of evaluations with higher samples of data.

Table 1.3 Exp. 2: Hyper-parameters – Values



Hyper-parameter	Value
adam_epsilon	1e-8
eval_batch_size	8
learning_rate	4e-5, 2e-5, 1e-5
max_seq_length	128
num_train_epochs	4
train_batch_size	32
optimizer	AdamW

Table 1.4 LR vs Model Results



Model Name	LR	Performance Assessment Metrics								
		BAcc	Acc	Pr	Recall	F1-Score	AU-PRC	AU-ROC	MCC	Eval Loss
BERT	4e-5	0.78	0.96	0.68	0.59	0.63	0.66	0.96	0.61	0.13
	2e-5	0.80	0.96	0.70	0.62	**0.66**	0.67	0.96	0.67	0.12
	1e-5	0.78	0.96	0.71	0.57	0.64	0.66	0.95	0.62	0.11
XLNET	4e-5	0.75	0.95	0.63	0.52	0.57	0.61	0.95	0.55	0.15
	2e-5	0.78	0.96	0.67	0.59	0.63	0.68	0.96	0.61	0.12
	1e-5	0.79	0.96	0.69	0.61	**0.65**	0.66	0.95	0.63	0.11
DeBERTa	4e-5	0.78	0.95	0.70	0.58	0.64	0.66	0.96	0.62	0.11
	2e-5	0.80	0.96	0.71	0.62	**0.66**	0.74	0.97	0.65	0.12
	1e-5	0.79	0.96	0.72	0.61	0.66	0.68	0.96	0.64	0.10
StructBERT	4e-5	0.69	0.94	0.65	0.40	0.48	0.55	0.90	0.55	0.21
	2e-5	0.69	0.94	0.65	0.40	0.48	0.55	0.90	0.55	0.21
	1e-5	0.70	0.95	0.68	0.41	**0.51**	0.55	0.95	0.60	0.18

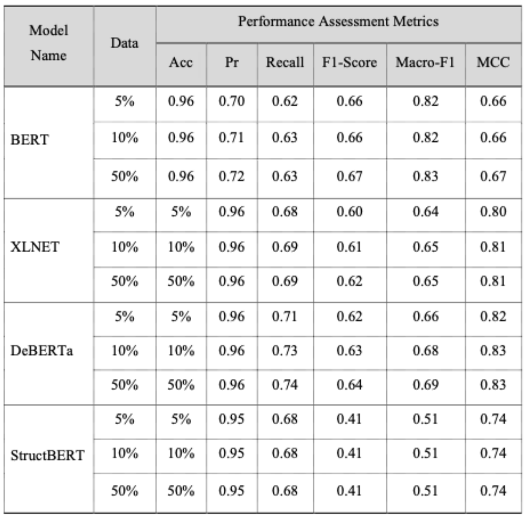
3.3 Experiment 3 - Transfer Learning with Modified Epochs

After performing the second set of experiments with the different learning rates, the third set of experiments is done by changing the training epochs. The training batch size is kept at 32, the eval batch size at 8, and the maximum sequence length is 128. The number of GPUs assigned is 1. With a 5% data sample, the distribution of data across the train and validation set remains the same as in experiment 1. In each iteration, the value of the epoch is changed, and metrics are noted. Each model is tried with 4 different values, resulting in a total of 16 executions in exp-3.

3.4 Experiment 4 - Transfer Learning with Modified Data Samples

After performing the third set of experiments with different epochs, the fourth and final set of experiments are done by changing the size of the dataset. The training batch size is kept at 32, the eval batch size at 8, and the maximum sequence length is 128. The number of GPUs assigned is 1. With a different percentage of data samples, the distribution of data across the train and validation set is highlighted in Table 5.8. In each iteration, the finetuned model performance on 5%, 10%, and 50% of the data with the best-chosen learning rate is noted. Each model is tried with 3 samples, resulting in a total of 12 executions in exp-4.

Table 1.5 Data Size vs Model Results



Model Name	Data	Performance Assessment Metrics					
		Acc	Pr	Recall	F1-Score	Macro-F1	MCC
BERT	5%	0.96	0.70	0.62	0.66	0.82	0.66
	10%	0.96	0.71	0.63	0.66	0.82	0.66
	50%	0.96	0.72	0.63	0.67	0.83	0.67
XLNET	5%	5%	0.96	0.68	0.60	0.64	0.80
	10%	10%	0.96	0.69	0.61	0.65	0.81
	50%	50%	0.96	0.69	0.62	0.65	0.81
DeBERTa	5%	5%	0.96	0.71	0.62	0.66	0.82
	10%	10%	0.96	0.73	0.63	0.68	0.83
	50%	50%	0.96	0.74	0.64	0.69	0.83
StructBERT	5%	5%	0.95	0.68	0.41	0.51	0.74
	10%	10%	0.95	0.68	0.41	0.51	0.74
	50%	50%	0.95	0.68	0.41	0.51	0.74

It is evident from Table 1.5 that there is only a slight to negligible increase in the model performance with the increase in the size of the data. It is observed that the model produced a macro f1 in the range of 0.74 - 0.82 with just 5% data. The model tends to generalize the pattern with 10% of the data within half the computational time of the entire dataset. Hence, results obtained from 10% of the data are considered for further analysis and discussion. In the following sub-section, the confusion matrix generated from each model with 5% and 10% data is explained

3.5.1 Confusion matrix - BERT

Taking sincere as positive class and insincere as a negative class the confusion matrix obtained with a learning rate 2e-5 and data of 5% and 10 %. With 5% data, the true positive of 12065 and the true negative of 456 depicts that the model can correctly predict most of the question. 340 records are predicted as insincere while they were observed to be sincere. Whereas 184 records are observed to be insincere but predicted as sincere. With 10% data, the true positive of 24089 and the true negative of 1093 depicts that the model can correctly predict most of the question. 409 records are predicted as insincere while they were observed to be sincere. Whereas 498 records are observed to be insincere but predicted as sincere.

3.5.2 Confusion matrix - XLNET

With 5% data, the true positive of 12030 and the true negative of 484 depicts that the model can correctly predict most of the question. 219 records are 64 predicted as insincere while they were observed to be sincere. Whereas 312 records are observed to be insincere but predicted as sincere. With 10% data, the true positive of 24034 and the true negative of 1070 depicts that the model can correctly predict most of the questions. 440 records are predicted as insincere while they were observed to be sincere. Whereas 545 records are observed to be insincere but predicted as sincere.

3.5.3 Confusion matrix - DeBERTa

With 5% data, the true positive of 12058 and the true negative of 483 illustrate that the model can correctly predict most of the questions. 191 records are predicted as insincere while they were observed to be sincere. Whereas 313 records are observed to be insincere but predicted to be sincere. With 10% data, the true positive of 24135 and the true negative of 1098 illustrate that the model can correctly predict most of the questions. 363 records are predicted as insincere while they were observed to be sincere. Whereas 493 65 records are observed to be insincere but predicted to be sincere.

3.5.4 Confusion matrix - StructBERT

With 5% data, the true positive of 12107and the true negative of 323 illustrate that the model can correctly predict most of the questions. 149 records are predicted as insincere while they were observed to be sincere. Whereas 466 records are observed to be insincere but predicted to be sincere. With 10% data, the true positive of 24213 and the true negative of 645 illustrate that the model can correctly predict most of the questions. 298 records are predicted as insincere while they were observed to be sincere. Whereas 973 records are observed to be insincere but predicted to be sincere.

3.6 Comparison of Model’s Performance

Model validation resulted in the generation of models with identical values of evaluation metrics to those of trained results. It is observed that the models did not record any significant differences between the two. This demonstrates that almost all the tested models performed well when tested in the eval dataset and the predicted-on test set. The total number of data points supported for evaluation and prediction is 26089 and 15000, respectively. When looking at the findings of all four models, it's clear that they can achieve state-of-the-art outcomes with only 5% of the data. The high rate of false positives is due to the imbalance in the dataset. When the dataset size is increased, the model's performance improves slightly. With 50 percentage data, there is just a minor rise in the F1 score and the macro average F1 score. Some of the assessment metrics discovered in question classification are compared between the classifiers to select the best or most resilient model among the proposed models. DeBERTa produced a macro f1-score of 0.83 with a 10% data sample, followed by BERT with a core of 0.82.

The MCC value of less than 0.5 clearly showed that the base models acted as random classifiers. The MCC shifting towards 0.7 in finetuned models showed that the model could understand the question text and generalize the class labels. It is observed that most finetuned models record similar values across all metrics. Based on the table, the balanced accuracy level is highest for DeBERTa and BERT with 80%. The StructBERT yielded the lowest balanced accuracy of all, with 70%. Even after multiple iterations with different learning rates, an increase in the F1 score is not observed. Due to the StructBERT model sizebeing equivalent to 1.25 GB, execution needed more space. Due to the limitation of computational size, the model is not validated for higher epochs.

For insincere labels, BERT and DeBERTa have the lowest false positive rate. The two models predicted the greatest precision for both class labels and macro and weighted averages. The average sensitivity for DeBERTa is given as 0.81 and for BERT as 0.80. The model DeBERTa resulted in an MCC of 0.66 and an AUPRC of 0.74 compared to BERT with 0.66 and 0.71, respectively. Thus, comparing all the performance measures, DeBERTa is chosen as the champion model over the rest.

4.0 Conclusion

The various types of questions posted on online social media platforms are not always intended to extend support or guidance. There are times when people use such platforms to promote hate or spread rumours to create restlessness among groups of people. This degrades the platform's quality, which in turn leads to a decline in its usage. Thus, the classification of destructive content before mass reading is equally essential and challenging. In the past, research was done using machine and deep learning models to restrain such content from posting. However, the models are highly dependent on feature engineering and the ratio of class labels in the dataset.

Starting with the most basic BERT model and progressing to the most esoteric 3 Bert-based transformer models, XLNET, DeBERTa, and StructBERT, the study shows how these models compete with one other. The basic model is BERTbase, which is one of the most stable models built on a big corpus of data. To classify the cleaned text and determine the outcome, all four models are applied. The entire implementation is done in Python programming language and the models are built in the Colab and Kaggle consoles using GPU and TPU for faster computation. 72 The dataset is taken from Kaggle and processed to form cleaned text. The lexicon and tokenizer for each of these models are used to train the model with the default argument on the cleaned dataset. Both PyTorch and Simple Transformer are used to demonstrate the implementation, as well as how to alter the standard parameter settings.

Multiple iterations are run, and the hyperparameters are chosen after a comprehensive examination. The preferred learning rate is determined by the metrics, and fine-tuned models with changed hyperparameters are trained on three separate dataset samples of 5%, 10%, and 50% of the total dataset. The class label imbalance ratio is guaranteed to be consistent across dataset samples. Model development, evaluation, and prediction are done in training, validation, and test sets, respectively, to maintain quality. As label imbalance is so crucial, the major assessment metric is a mix of f1 score, MCC, and balanced accuracy. To determine the best classifier, the outcomes of pre-trained and fine-tuned models are compared. DeBERTa and BERT surpassed all other models trained in this study. The results also outscored logistic regression or other deep learning models noted in the previous studies.

The objective of generating state-of-the-art results on a noisy dataset without using any class balancing approaches and with low computational capacity is met since all the models stated delivered outstanding results with only 10% of the data. When comparing DeBERTa with BERT, the former exceeded with a slightly higher F1-score, indicating that the study's research aim is justified. It also demonstrates that combining transfer learning with attention mechanisms can improve existing performance.

REFERENCES

1. Classification Models - Simple Transformers, https://simpletransformers.ai/docs/classification-models/ last accessed 2022/03/08.
2. HuggingFace Transformers, https://huggingface.co/docs/transformers/index last accessed 2021/03/08.
3. Quora Insincere Questions Classification | Kaggle, https://www.kaggle.com/c/quora-insincere-questions-classification/data last accessed 2021/11/02.
4. Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference, [online] 1, pp.4171–4186 (2018a).
5. Devlin, J., Chang, M.-W., Lee, K. and Toutanova,: K.BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference, [online] 1, pp.4171–4186 (2018b).
6. He, P., Liu, X., Gao, J. and Chen, W.: DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In: ICLR 2021 Conference Paper3690 (2020).
7. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I.:Attention is All you Need. In: I. Guyon, U. V Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds., Advances in Neural Information Processing Systems (2017).
8. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. and Le, Q. V.: (2019) XLNet: Generalized Autoregressive Pretraining for Language Understanding. Advances in Neural Information Processing Systems, (2019.