*Natural Language Processing****Project Report***

***GROUP-8***

- Parth Parashar

- Vansh Maheshwari

***Objective***

Develop predictive algorithms to evaluate the subjective quality aspects of question-answering based on the newly collected dataset from the CrowdSource team at Google Research. This dataset consists of question-answer pairs sourced from nearly 70 different websites, gathered in a "common-sense" manner, where raters applied their subjective interpretations to evaluate the quality of the answers.

***How Data looks***

* The shapes of the 3 files are:

Size of train\_data (6079, 41)

Size of test\_data (476, 11)

Size of sample\_submission (476, 31)

* The test file has 41 columns out of which 11 are independent variables and the remaining 30 are our target variables.
* We need to find out the relation of target variables with the independent variables using the [Spearman's correlation coefficient](https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient) for each target column and finally the mean of these values will be taken as the result for the submission file.

**Photo of dataset**

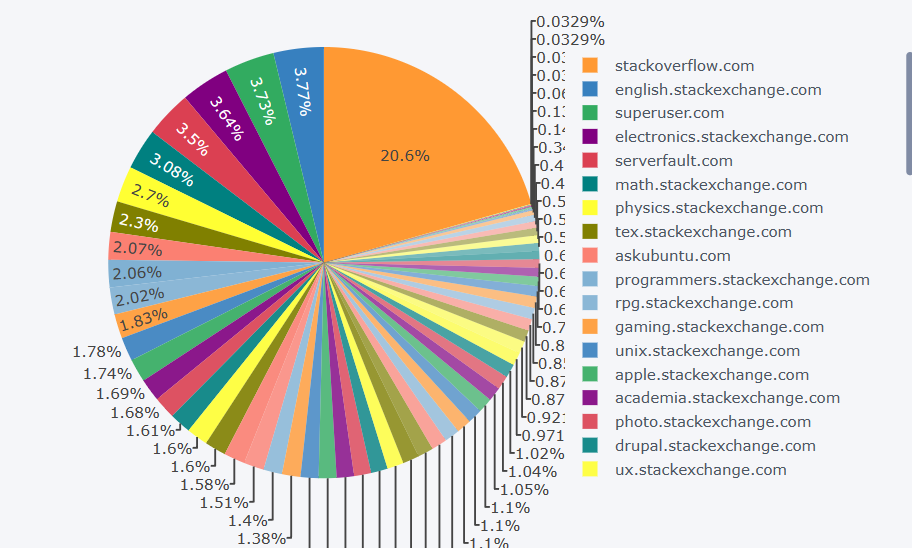
A screenshot of a computer

Description automatically generated

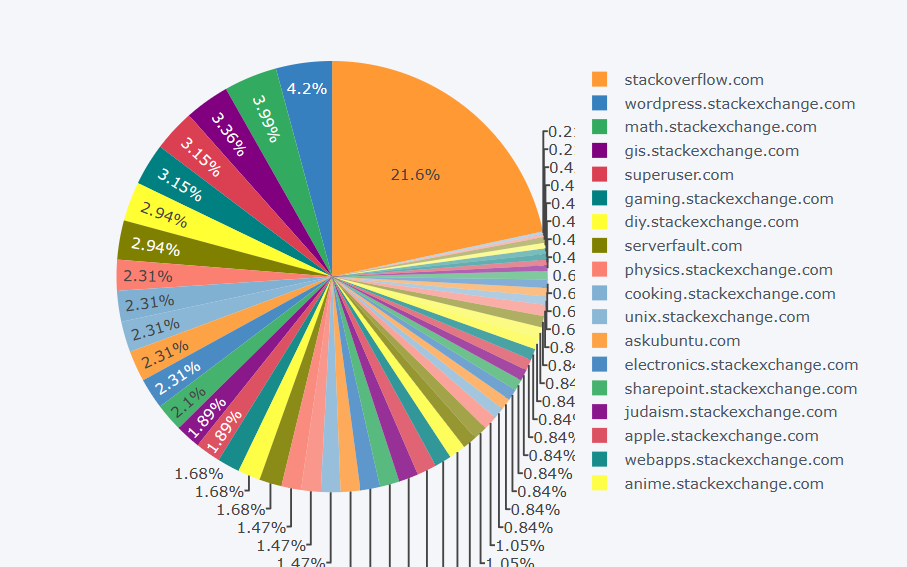
Analyzing Data

We began our project by first performing some EDA and thereby analyzing the useful data.

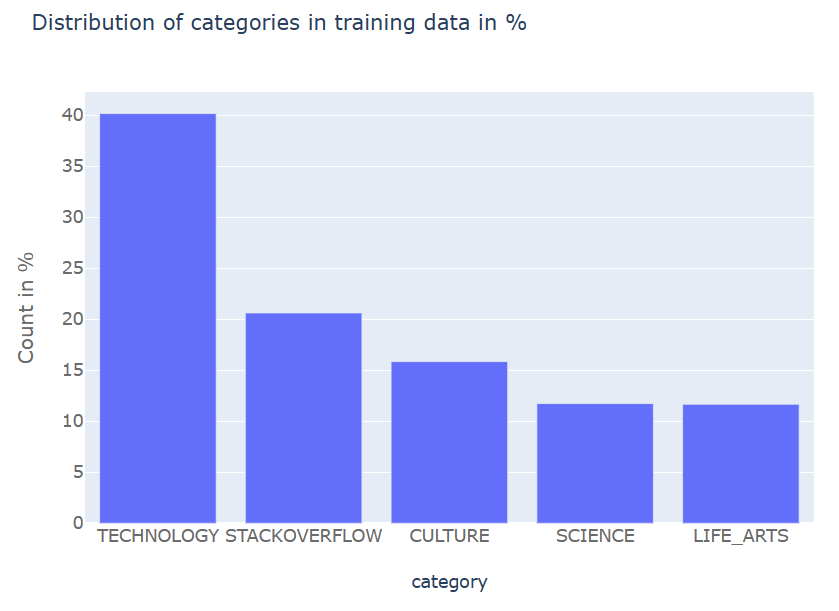
***Ratio of websites that have provided the data.***

***(Train Data)***

***Ratio of websites that have provided the data.***

***(Test data)***

***Distribution of categories in training data***



***Distribution for question title***

A graph of a distribution of data

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

***Distribution of question body***

**A graph of a number of data

Description automatically generated with medium confidence**

**A screenshot of a computer

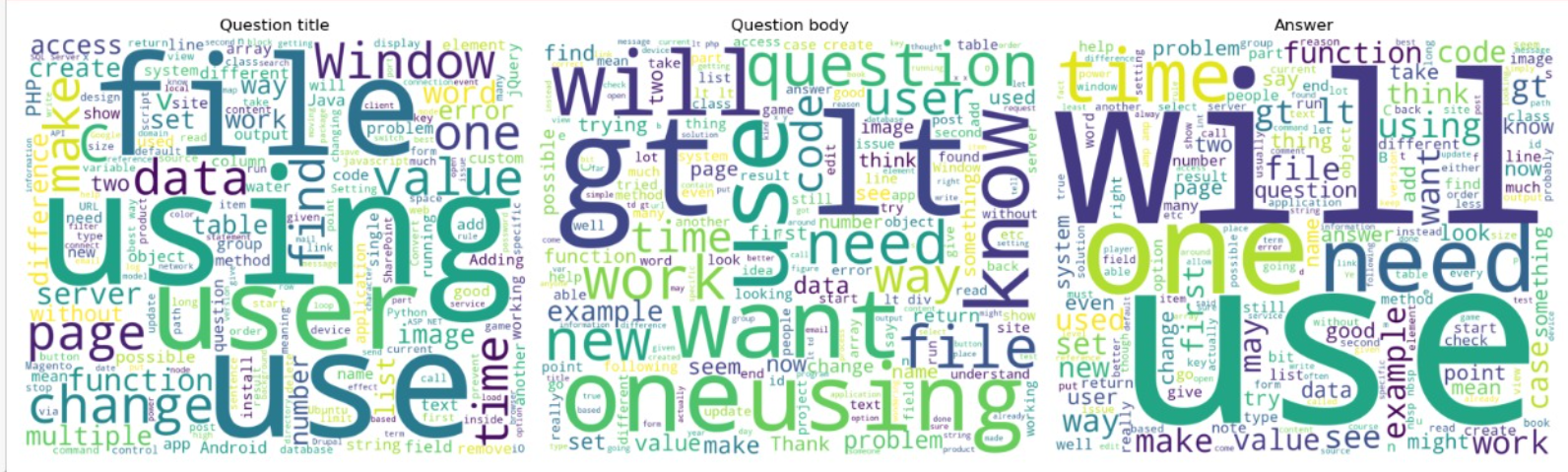
Description automatically generated**

***Distribution of answer*** ***A graph of distribution of data

Description automatically generated*A black background with white text

Description automatically generated**

***Word Cloud***

****

***What is Spearman’s rank correlation?***

* Spearman’s Rank Correlation is a statistical measure of the strength and direction of the monotonic relationship between two continuous variables.
* Can take values between -1 to +1.
* Represented by rho.
* Data is converted into ranks where the smallest value gets rank 1 and so on.

***An example***

A table of math equations

Description automatically generated

A black and white math symbol

Description automatically generated with medium confidence

***Reasons to use Spearman’s correlation***

* The Spearman correlation test examines whether two variables are correlated with one another or not. The Spearman's test can be used to analyze ordinal level, as well as continuous level data because it uses ranks instead of assumptions of normality.
* Spearman’s correlation is helpful in cases where outliers in data are present.
* Spearman's correlation measures the strength and direction of monotonic association between two variables.

***Data analysing***

* When we look at the data carefully we realise that we need only 3 columns as the input on which we need to train our model. These 3 columns are: question\_title, question\_body and answer.
* Anything other than these 3 columns was not making much impact on our results.
* In the model we use different ways by which we can make better use of the data provided by these 3 columns.

***Data Preprocessing***

* We first initialized a tokenizer based on the model we are going to implement. Since we are using 3 models Bert, Roberta, and XLNet, we need to use the tokenizer accordingly.
* Our data preprocessing class preprocesses the input text data, tokenizes it, trims it based on maximum sequence lengths, and prepares it for the model.
* The max\_sequence\_length was kept at 512 although a few issues were faced by us.
* The total input taken must be less than 512 tokens as length greater than that will be trimmed by the method based on the max size for every column i.e. t\_max\_len, q\_max\_len, a\_max\_len
* If after trimming, any of the text segments (Title, Body, Answer) becomes empty, a placeholder token ('\_') is used to ensure there's always some input for each segment.
* Depending on the tqa\_mode ('tq\_a', 't\_q', 't\_a'), the method decides how to combine the Title, Body, and Answer:

***'tq\_a****': Title + [TBSEP] + Body + Answer*

***'t\_q****': Only Title + Body*

***'t\_a’****:**Only Title + Answer*

* Encoding and Padding: Finally, it uses the tokenizer's encode\_plus method to convert the tokenized text segments into model inputs
* It also pads the sequences to MAX\_SEQUENCE\_LENGTH and returns tensors for input\_ids, attention\_mask, token\_type\_ids, and position\_ids.
* **use\_category**: If True, the category of the question is added to the Title.

***Example***

Suppose you have a row from the DataFrame with the following data:

* + ***question\_title****: "How to use Python?“*
  + ***question\_body****: "I want to learn Python for data analysis."*
  + ***answer****: "You can start with Python tutorials available online."*

If tqa\_mode is 'tq\_a' and use\_category is True, and you set MAX\_SEQUENCE\_LENGTH to 50, the preprocessing might produce:

* + ***Title****: "CAT\_TECHNOLOGY How to use Python?"*
  + ***Body****: "I want to learn Python for data analysis."*
  + ***Answer****: "You can start with Python tutorials available online."*

***Problems faced***

* We started working on Google colab as they provide us with T4 processors. But due to some issues colab was getting disconnected.
* We had to reduce max\_sequence\_length in order to avoid out of memory error
* Needed to reduce batch size.
* Jupyter on laptop took way too much time to run as CPU without accelerator is too slow.
* We finally used Kaggle’s workspace that gave us P100 processors which are faster.

***Model Architecture***

* Here we are using 3 different model architectures tailored for binary multi-label classification tasks:

1. BertModelForBinaryMultiLabelClassifier
2. RobertaModelForBinaryMultiLabelClassifier
3. XLNetModelForBinaryMultiLabelClassifier

***BERT Model***

**Initialization**:

* num\_labels: Number of labels/classes for the classification task.
* config\_path: Path to the model configuration file.
* state\_dict: Pre-trained model state dictionary.
* token\_size: Size of the token embeddings (optional).
* MAX\_SEQUENCE\_LENGTH: Maximum sequence length for input tokens.

**Linear Classifier**:

Linear layer with input size equal to the hidden size of the BertModel and output size equal to num\_labels.

**Dropout Layer:**

* Dropout layer with 20% dropout rate to prevent overfitting.

**Forward Method:**

Inputs:

* input\_ids: Tokenized input sequence.
* attention\_mask: Mask to indicate which tokens should be attended to.
* token\_type\_ids: Segment IDs for different segments in the input.
* position\_ids: Positional IDs for positional embeddings (optional).

Process:

* Token Embedding: The input tokens (input\_ids) are converted into embeddings by the base model.
* Attention Masking: attention\_mask helps the model focus only on non-padding tokens during attention calculation.
* Pooling: The pooled output is a mean of all token embeddings, producing a fixed-size representation of the input sequence.
* Classification: The pooled output is passed through a linear layer to produce logits, which are used to compute the probabilities of each class.
* Dropout: Dropout is applied to the pooled output to prevent overfitting.

**BERT Question Prediction Loop**

The BERT Answer Prediction Loop iterates over each fold and rank, loads the pre-trained model state, makes predictions on the test data, stores the predictions, and finally averages predictions across ranks. The loop outputs a dictionary bert\_fold\_prediction \_dict containing the combined predictions for each fold.

Loading Model and State Dict:

* We have used 5 folds.

For each fold and rank combination:

* Load the pre-trained model state dictionary specific to the fold and rank.
* Create an instance of BertModelForBinaryMultiLabelClassifier.
* Load the pre-trained state dictionary into the model.
* Move the model to the specified device (DEVICE).
* Create Test Dataset and DataLoader
* Make Predictions

***Similarly…***

* The BERT answer loop is trained in a similar way
* Also the 2 other models Roberta and XLNet have been trained the exact same way as the BERT model.
* The results of all 3 the models differ from each other but not by much.

**A screenshot of a computer

Description automatically generated*BERT PREDICTIONS***

***A screenshot of a computer

Description automatically generatedRoberta PREDICTIONS***

***XLNet PREDICTIONS***

**A screenshot of a computer

Description automatically generated**

***Predictions merged***

**A screenshot of a computer

Description automatically generated**

**CHANGES MADE**

During the presentation we were asked to make a few changes to our code that basically required us to show the predicted values from the model and the losses during training. We started to build the code again but this time only for the BERT model as we had only limited time. After writing the code again we have received the desired results and this python notebook will be submitted as a separate file along with the code we made before presentation. Hence a total of 2 python notebooks are submitted by us.

**Training Loss**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Final mean spearman coefficient**

****