# Autonomous Tagging of Stack Overflow Questions

CSCE 5290: Natural Language Processing, Spring 2022

**Instructor:** *Dr Sayed Khushal Shah* (sayed.shah@unt.edu)

**Team**: Alekhya Vachakarla (alekhyavachakarla@my.unt.edu)

**Github Link**:

https://github.com/AlekhyaVachakarla1225/Autonomous-tagging-stackoverflow-questions.git

## Introduction

Online responsive discussions, for example, Stack Exchange and Quora are turning into an undeniably well known asset for training. Key to the usefulness of a large number of these discussions is the idea of labeling, by which a client names his/her post with a proper arrangement of subjects that portray the post, to such an extent that it is more effortlessly recovered and coordinated. The quantity of data is increasing day by day on these forums but there is no productive or automated way to classify the data currently.We propose a classification that naturally labels clients inquiries to upgrade client experience.

# Motivation

Stack Overflow is the biggest platform where different people like developers, students, designers etc. use to share, learn and develop their skills and knowledge and build their careers. Most of them use stack overflow while developing a technical feature or to resolve any issues or errors in their code/programs. Every month, more than 50 million engineers come to Stack Overflow to learn and share their insight. The platform allows the users to ask and answer the questions at same time. Many people also vote and edit answers.

# Significance

- Why is Autonomous tagging of stack overflow questions Important?

  A question posted on stack overflow contains three different segments title, description and tags/labels. By using the title and description, the system should be able to automatically suggest the label related to the subject posted on the website. These labels are very important for the efficient working of the stack overflow platform.

- Tagging of the questions is specifically useful for indexing data based on the tags.
- If the questions posted are not segregated or grouped correctly, then the questions will lose all sense of direction in a pool of un-addressed questions.
- Assigning labels to the questions will also help in clubbing the similar questions.

- Assigning tags/labels correctly to the questions will make it more likely to reach the proper audience and thus there will be a higher opportunity in getting the question answered.
- Incorrect labeling of the questions will also impact the experience of the users on stack overflow.

# Objective & Features

The objective of my project is to build different models which classify the category of the question posted or inputted. With the help of deep learning models and bert models we can implement this mechanism with more precision and accuracy

- Using different NLP techniques to prepare and process the data.

- Creating a Fully trained model and training the model with the dataset

- Evaluate the performance of the model and check whether the model is able to determine the category of the given input text correctly or not.

# Related Work

1. **Autonomous tagging of stack overflow questions**, this paper was written by Mihail Eric, Ana Klimovic, Victor Zhong. They proposed multi class classification of stack overflow questions using the dataset from kaggle. Text classification is both a multiclass and multi-label classification problem,there are many classes to choose when labeling an input and each input may according to more than one class. For tackling multi-label classification there are mainly two common approaches.one-v-rest is one of

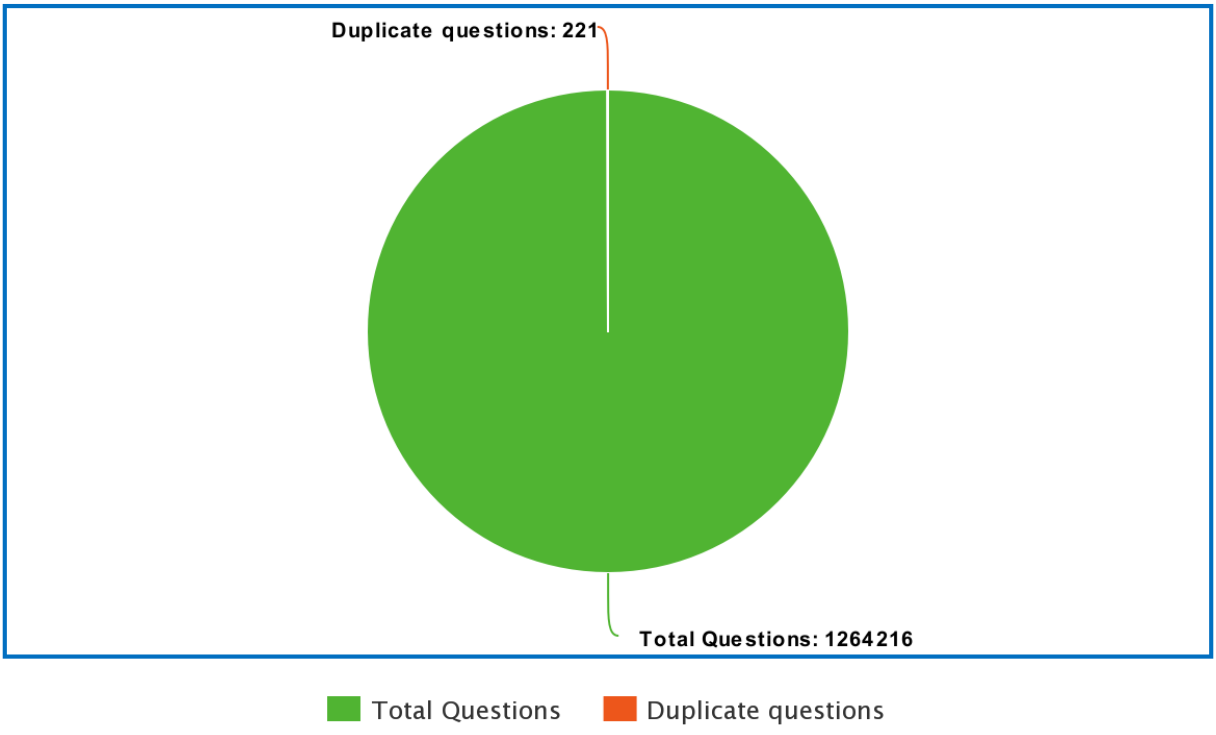
the common approaches. It will decompose the classification problem into k independent binary classification problems,which will train a separate classifier for all the k possible output labels. Another approach for multi-label classifications is known as the set of adaptive algorithms,which will predict all labels at once with  a confidence ranking associated with each label. Boosting and random forest are examples of adaptive algorithms.

# Dataset

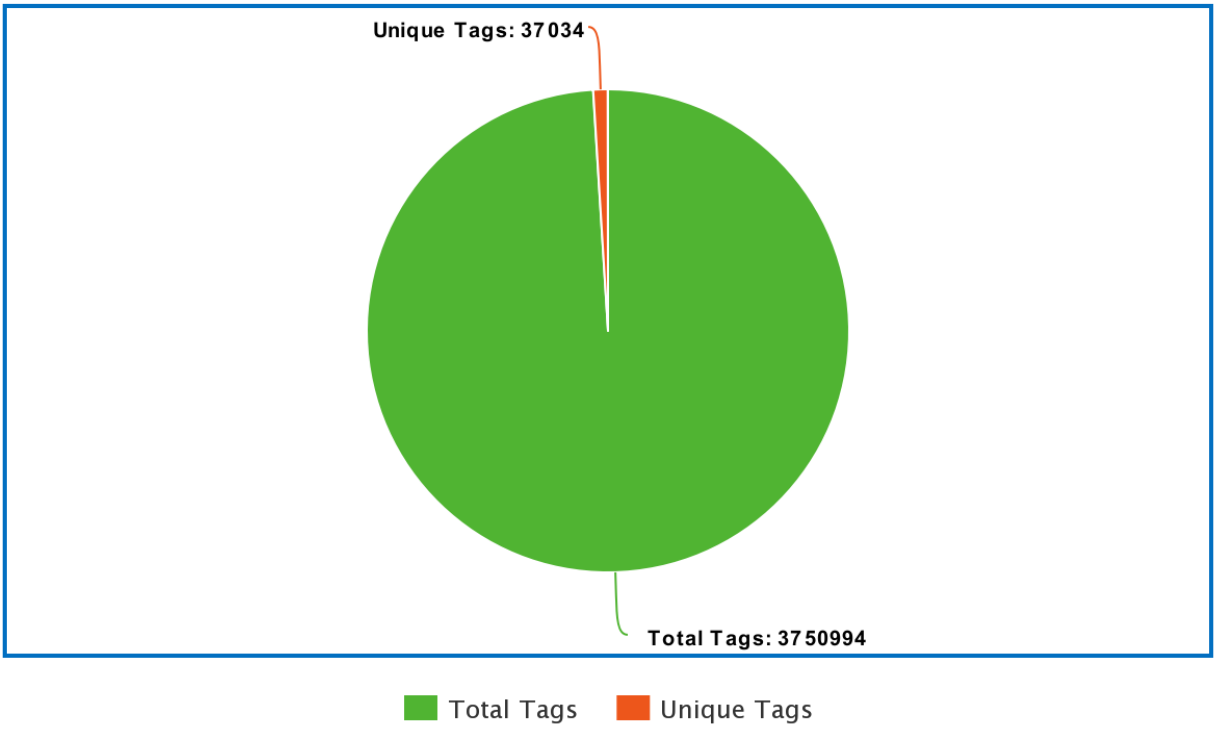
The dataset used in the project to create the classifier is from a Kaggle. The dataset has questions which are assigned tags based on the body and title of that particular question. The dataset has approximately 1,264,204 unique questions which have a total number of 40,000 unique tags taken from the stack overflow site. The data is divided into two different files in which one consists of questions and other consists of tags and both are related by a unique identifier using which we combine the data from both files. Every question has  Id, OwnerUserId, CreationDate, ClosedDate, Score, Title, Body. For each record in the tags file it has two columns Id and Tag. We combine the files into one to make the process easier and row in the data frame has an identifier, title, body, tag, score which we are going to consider mainly in the process of classifying. We ignore other rows as they are not needed in training the classifier. This large number of records creates a problem in computing, so for that reason we consider only some amount of data based on conditions and train the classifier. Hence the data is restricted to some part of records in data. This guarantees that we have adequately different training data so we can learn statistics for the tags.

Here are the below graphs which shows the statistics of questions and tags from the data frame which i have used in project,

Total questions Vs Duplicate questions




Total tags Vs Unique Tags



# Detail Design of Features

**Data Cleaning / Pre-Processing**

- First step, we group both the data frames into one based on the unique identifier present in both the files.
- In order to minimize the computing power, We consider the rows/data which have scores greater than 20.
- Another reason for choosing posts that contain scores greater than 20 is that probably has a better quality and also can be better tagged as they is contains many upvotes
- We then, drop the unwanted fields and clean the data using different NLP techniques to make the data fit to train a classification model
- Check for null values/duplicate values and remove if there are any.
- We then split the words in tags and transform it into list of tags
- We calculate the tags which are most common and take only the top 100 tags, then from the revised data frame we check for null values and remove rows if there are any using ‘dropna’.
- Then we incorporate a column in the dataframe called tag count based on the length of the list for each row in the tags column.

- After reducing the data frame size, we remove the html format in the body column of the data frame.
- We first convert each and every one of the alphabets of the body to lowercase letters from capitalized letters to standardize the text and reduce the variety as the calculations might be case sensitive.
- Then, transform the abbreviations for example, what's converted to what is. It is a very useful technique which helps the program to understand the text.
- We then take out all of the punctuation and uncommon characters(except #) from the text to reduce the varieties a lot further
- We then, at that point, use lemmatization on the text and later dispense with stop-words to basically diminish the un supportive words and reduce estimation cost during training.

## Model Training

- After data preparation and cleaning, we use tokenizer to convert the ‘body’ text to matrix using predefined methods from tokenizer
- Then we encode the tags using LabelBinarizer and split the data into train and test in the ratio of 80% and 20%
- Model training is the process of feeding an algorithm or a model with data to help distinguish and learn great qualities for all attributes involved.

- We plan to use different models in the project and analyze the output of models.

### Evaluation of models

- For Evaluation, we are going to calculate the accuracy of the models and also try to predict a tag based on the question given as input by users.
- We use, evaluate function from the model and calculate the accuracy on the test set.

# Analysis

For further analysis, I have created some visualizations of the data to easily analyze the statistics of the data, there is one or multiple tags maximum upto 5 for each question posted, for example,

Q1 = {t1,t2,t3,t4}
Q2 = {t1,t2,t3}
Q3 = {t1,t2}

So, the classifier should be able to predict multiple tags for the question posted. This process is called multi-label classification. It assigns a set of labels to each sample.

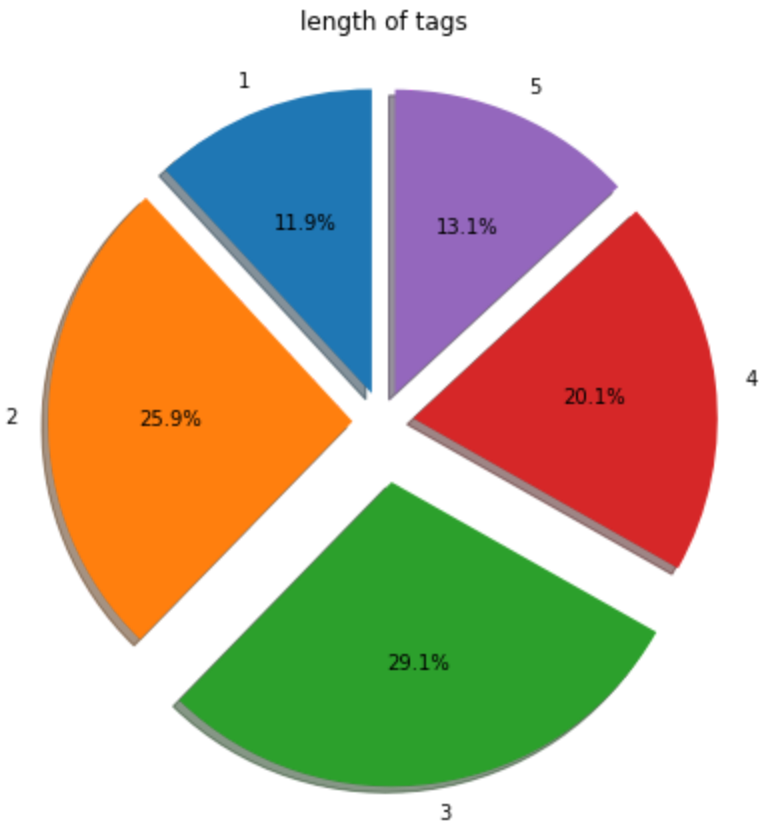



Fig.1: Percentage of tags

Figure 1, shows the percentage of the length of tags, As the number of questions tagged with one label are 11.9, number of questions tagged with two labels are 25.9%, number of questions tagged with three labels are 29.1% which is the highest among all of them, number of questions tagged with four labels are 25.9% and number of questions tagged with five labels are 13.1%

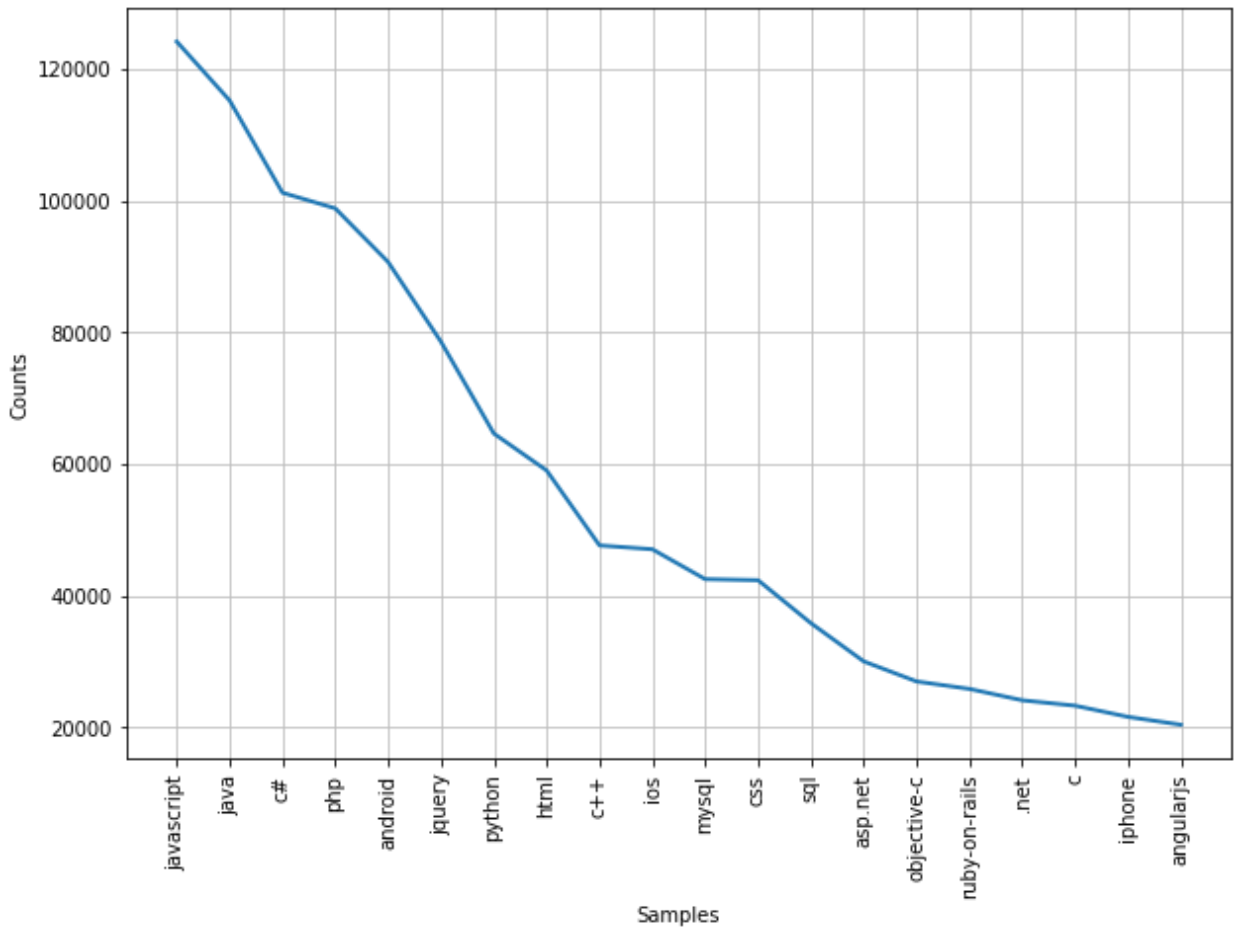



Fig.2: Samples Vs Counts

Figure 2, shows the top twenty tags in the data set and the number of questions tagged with those top twenty tags.

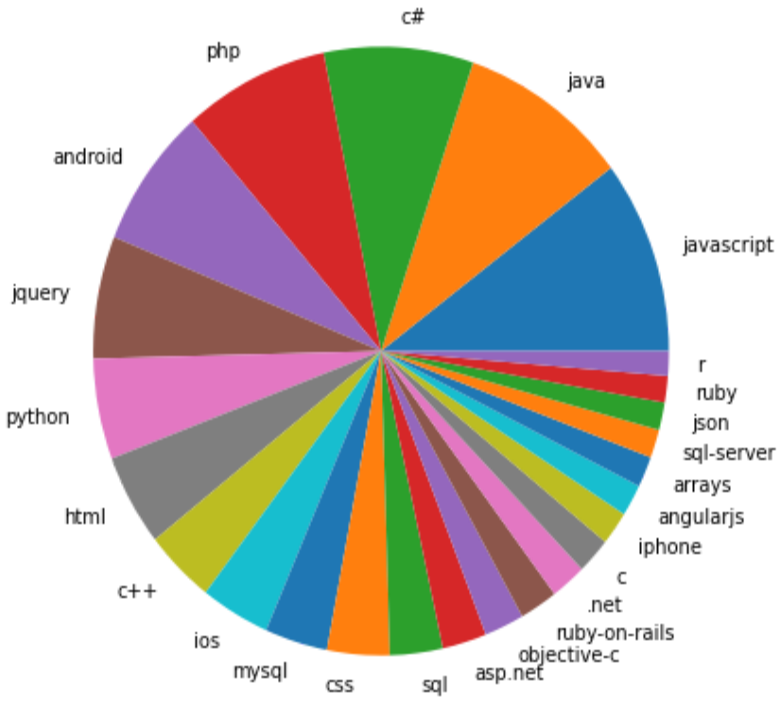



Fig.3: Tags and questions

Figure 3, shows the same as above but, top twenty five tags and number of questions tagged.


**Fig 4**: word cloud

Figure 4, shows the top hundred tags from the dataset that is being used.

# Implementation

## Models

The first model that I am going to use is a basic deep learning model i.e sequential model from keras with some layers embedded. There are a total of three dense layers, three activation layers and two drop out layers.

The below figure shows the visualization of the layers which I used in the model. The model will be able to tag multiple labels to the question based on the array values generated. The activation function that is being used is ‘relu’ and loss is ‘binary cross entropy’

The rectified linear activation or ReLU linear function that will generate an output as input directly if it is positive, otherwise, it will generate an output as zero.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               7680512

 activation (Activation)     (None, 512)               0

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 activation_1 (Activation)   (None, 512)               0

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 512)               262656

 activation_2 (Activation)   (None, 512)               0

 dropout_2 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 512)               262656

 activation_3 (Activation)   (None, 512)               0

 dropout_3 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 5102)              2617326

 activation_4 (Activation)   (None, 5102)              0

=================================================================
```

Fig.5: layers of model

```
encoder =  MultiLabelBinarizer()
encoder.fit(train_tags)
y_train = encoder.transform(train_tags)
y_test = encoder.transform(test_tags)
```

MultilabelBinarizer is used to encode the tags. Multilabelbinarizer allows to encode many labels per instance.

Then training the model,

```
num_epochs =60
batch_size = 500
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=num_epochs,
                    verbose=2,
                    validation_split=0.1)
```

```
Epoch 3/60
21/21 - 1s - loss: 0.0068 - accuracy: 0.0285 - val_loss: 0.0069 - val_accuracy: 0.0292 - 743ms/epoch - 35ms/step
Epoch 4/60
21/21 - 1s - loss: 0.0054 - accuracy: 0.0333 - val_loss: 0.0060 - val_accuracy: 0.0292 - 758ms/epoch - 36ms/step
Epoch 5/60
21/21 - 1s - loss: 0.0049 - accuracy: 0.0348 - val_loss: 0.0057 - val_accuracy: 0.0345 - 753ms/epoch - 36ms/step
Epoch 6/60
21/21 - 1s - loss: 0.0047 - accuracy: 0.0333 - val_loss: 0.0056 - val_accuracy: 0.0292 - 754ms/epoch - 36ms/step
Epoch 7/60
21/21 - 1s - loss: 0.0045 - accuracy: 0.0382 - val_loss: 0.0056 - val_accuracy: 0.0283 - 743ms/epoch - 35ms/step
Epoch 8/60
21/21 - 1s - loss: 0.0044 - accuracy: 0.0382 - val_loss: 0.0055 - val_accuracy: 0.0319 - 739ms/epoch - 35ms/step
Epoch 9/60
21/21 - 1s - loss: 0.0043 - accuracy: 0.0375 - val_loss: 0.0055 - val_accuracy: 0.0301 - 753ms/epoch - 36ms/step
Epoch 10/60
21/21 - 1s - loss: 0.0043 - accuracy: 0.0376 - val_loss: 0.0054 - val_accuracy: 0.0345 - 741ms/epoch - 35ms/step
Epoch 11/60
21/21 - 1s - loss: 0.0042 - accuracy: 0.0366 - val_loss: 0.0054 - val_accuracy: 0.0372 - 755ms/epoch - 36ms/step
Epoch 12/60
21/21 - 1s - loss: 0.0042 - accuracy: 0.0408 - val_loss: 0.0055 - val_accuracy: 0.0327 - 821ms/epoch - 39ms/step
Epoch 13/60
21/21 - 1s - loss: 0.0042 - accuracy: 0.0373 - val_loss: 0.0053 - val_accuracy: 0.0372 - 826ms/epoch - 39ms/step
Epoch 14/60
```

**Random Forest Classifier**

Second model implemented in the project is a random forest classifier which is one of the best classifiers for classifying multiple labels. Sci-kit learn provides inbuilt support of multi-label classification for Random Forest. Random forest builds decision trees based on multiple samples and will take their majority votes for classification.

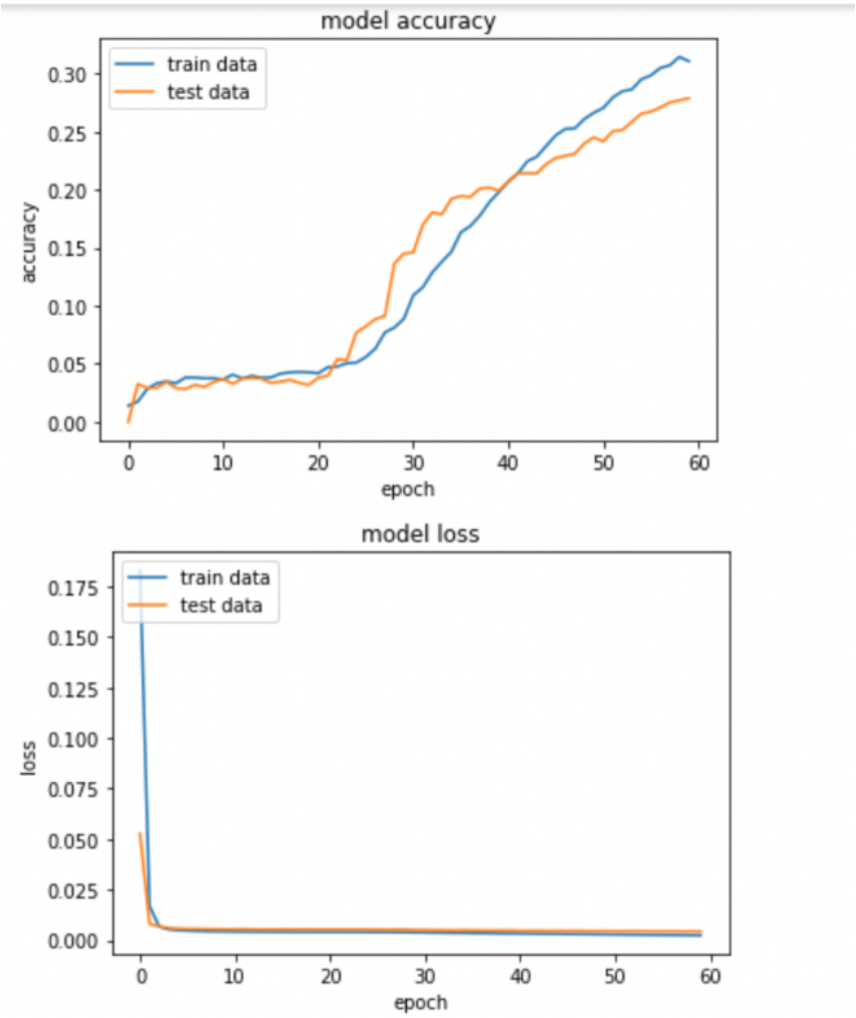
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)
```

# Preliminary Results

**Sequential model:**




The above figure shows the graphs of train vs test accuracy and train vs test loss. It also predicts the tags based on the input given.

```
predict("Why is processing a sorted array faster than processing an unsorted array? Here is a piece of C++ code that shows some very peculiar behavior. For
```

```
text: Why is processing a sorted array faster than processing an unsorted array? Here is a piece of C++ code that shows some very peculiar behavior. For some
Predicted label: algorithm
Predicted label: string
Predicted label: c++
```

```
Actual label: ['matplotlib', 'marker', 'scatter']
Predicted label:  python, matplotlib, numpy,
Actual label: ['maven', 'markdown', 'maven-site-plugin']
Predicted label:  git, github, version-control,
Actual label: ['python']
Predicted label:  python, generator, algorithm,
Actual label: ['android', 'android-fragments']
Predicted label:  android, android-fragments, android-activity,
Actual label: ['javascript', 'angularjs', 'checkbox', 'repeat']
Predicted label:  jquery, asp.net-mvc-3, asp.net-mvc,
Actual label: ['html5', 'html5-audio']
Predicted label:  javascript, html, html5,
Actual label: ['amazon-web-services', 'nosql', 'amazon-dynamodb']
Predicted label:  mysql, sql, database,
Actual label: ['java', 'unit-testing', 'mockito']
Predicted label:  java, junit, multithreading,
Actual label: ['java', 'collections']
Predicted label:  java, c#, collections,
Actual label: ['ruby', 'string', 'substring', 'slice', 'idiomatic']
Predicted label:  ruby, regex, string,
Actual label: ['chef', 'chef-recipe']
Predicted label:  python, ruby, r,
Actual label: ['php', 'arrays', 'loops', 'foreach']
Predicted label:  haskell, php, c#,
Actual label: ['ruby', 'median']
Predicted label:  string, arrays, algorithm,
Actual label: ['java', 'spring', 'spring-mvc', 'configuration']
Predicted label:  java, spring, spring-mvc,
Actual label: ['android', 'android-textview', 'linkify', 'selectable']
Predicted label:  android, android-layout, iphone,
Actual label: ['ios', 'uiviewcontroller', 'push-notification', 'apple-push-notific
```

The above picture shows the Actual tags and predicted tags of the data present in the test dataset.

**Random Forest Results:**

```python
from sklearn.metrics import hamming_loss
from sklearn.metrics import jaccard_score

# Function to get jacard score
def get_jacard():

    jscore = jaccard_score(y_test, y_pred, average='samples')

    return jscore

# Function to call jacard score and get hamming loss
def get_score():

    print("Jacard score: {}".format(get_jacard()))
    print("Hamming loss: {}".format(hamming_loss(y_pred, y_test)*100))
```

Hamming loss is used to find out the fraction of incorrect predictions of a given model.

```
get_score()

Jacard score: 0.185375
Hamming loss: 0.1568776628119294
---
```

The above picture shows that hamming loss is 15% which indicates 15% of the data are predicted incorrectly.

# Project Management

Work Completed so far,

| Team Member | Responsibility |
|---|---|
| Alekhya Vachakarla | 1) Brainstorming topics on classification.<br>2) Research on Project Idea, collection of dataset from kaggle.<br>3) Added different visualizations of data.<br>4) Data preparation/ cleaning.<br>5) Implemented a basic sequential model with deep learning layers.<br>6) Implemented Random Forest Classifier<br>7) Project report documentation. |

Work to be done,

| Team Member | Responsibility |
|---|---|
| Alekhya Vachakarla | 1. Implement best multi label classification models like oneVsRest classifier<br>2. Implement GRU model<br>3. Add Evaluation metrics to the models implemented<br>4. Document the Implementation and Results of above mentioned models. |

# References


- Mihail Eric, Ana Klimovic, Victor Zhong. Autonomous Tagging of Stack Overflow Questions. Paper published on December 8, 2014 in stanford edu. http://cs229.stanford.edu/proj2014/Mihail%20Eric,%20Ana%20Klimovic,%20Victor%20Zhong,MLNLP-Autonomous%20Tagging%20Of%20Stack%20Overflow%20Posts.pdf

- Virik Jain, Jash Lodhavia. Automatic Question Tagging using k-Nearest

Neighbors and Random Forest. 2020 International Conference on Intelligent Systems and Computer Vision 10.1109/ISCV49265.2020.9204309

- Meghashyam Chinta, Predicting Tags for the Questions in Stack Overflow

  https://medium.datadriveninvestor.com/predicting-tags-for-the-questions-in-stack-overflow-29438367261e