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-stackoverfl

ow-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

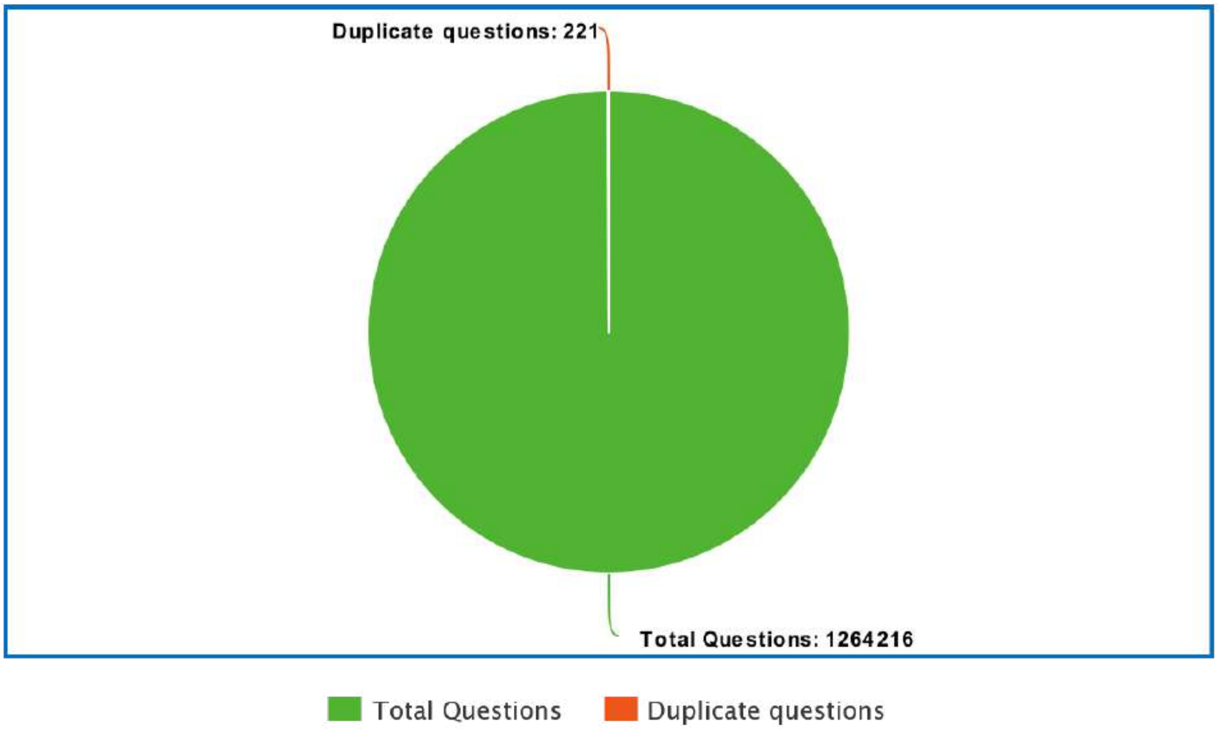
1. **Autonomous tagging of stack overflow questions**, this paper was written by Mihail Eric, Ana Klimovic, Victor Zhong. They proposed multi class classification of stack overflow questions using the dataset from kaggle. Text classification is both a multiclass and multi-label classification problem,there 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.

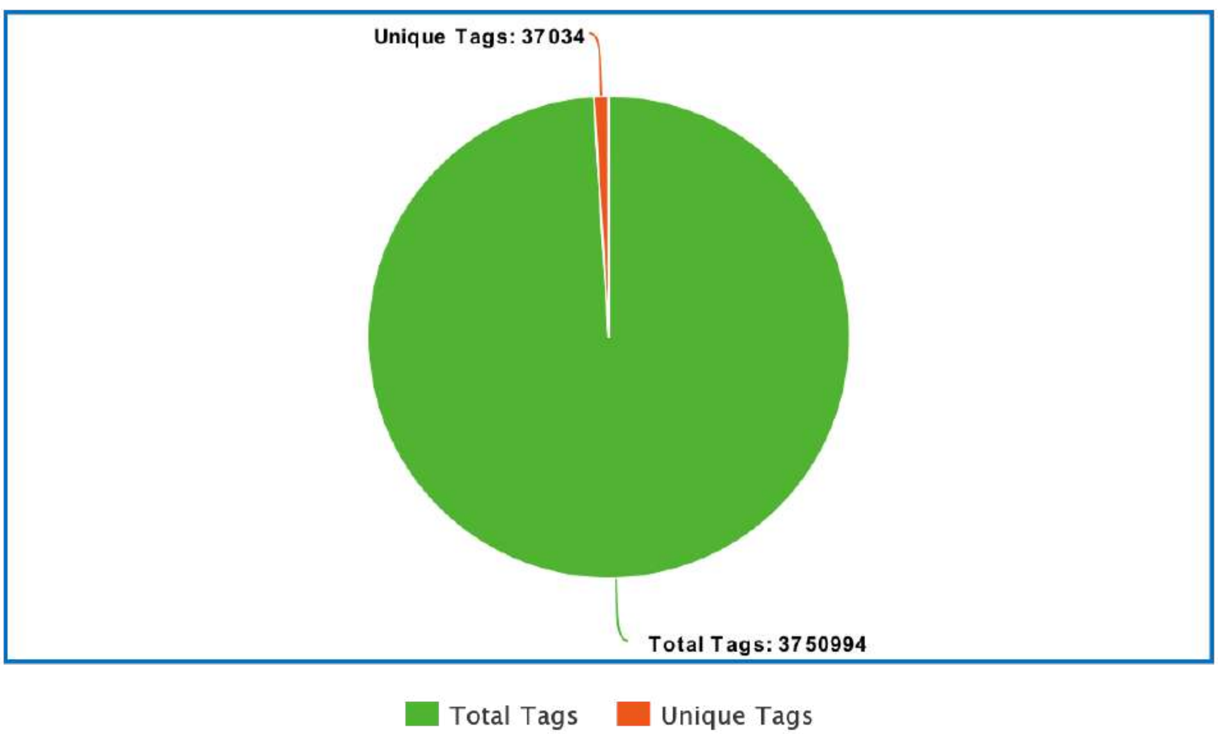
Dataset

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.

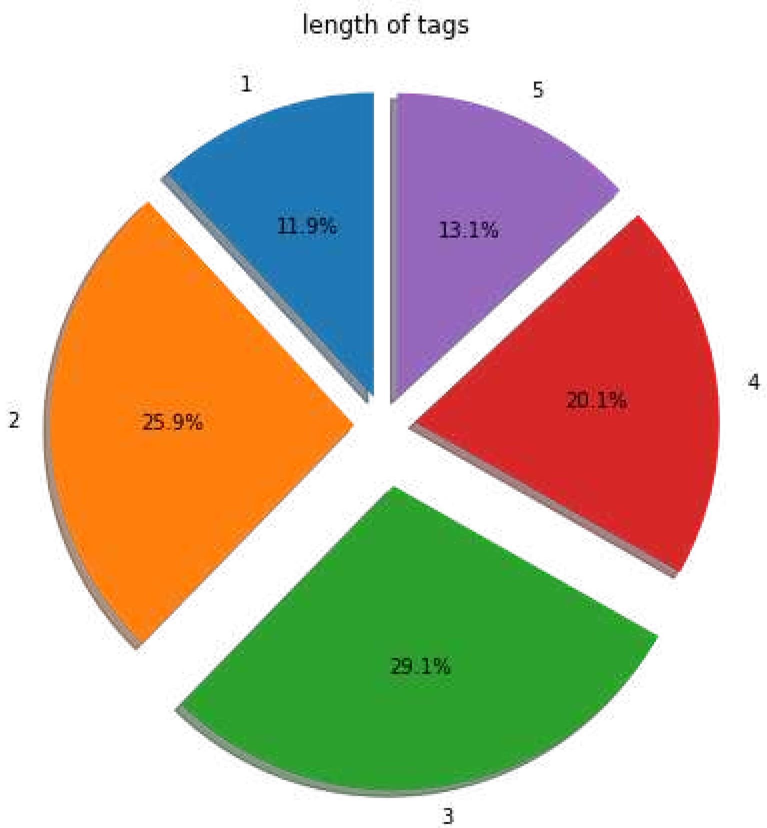


Fig.1: Percentage of tags

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%

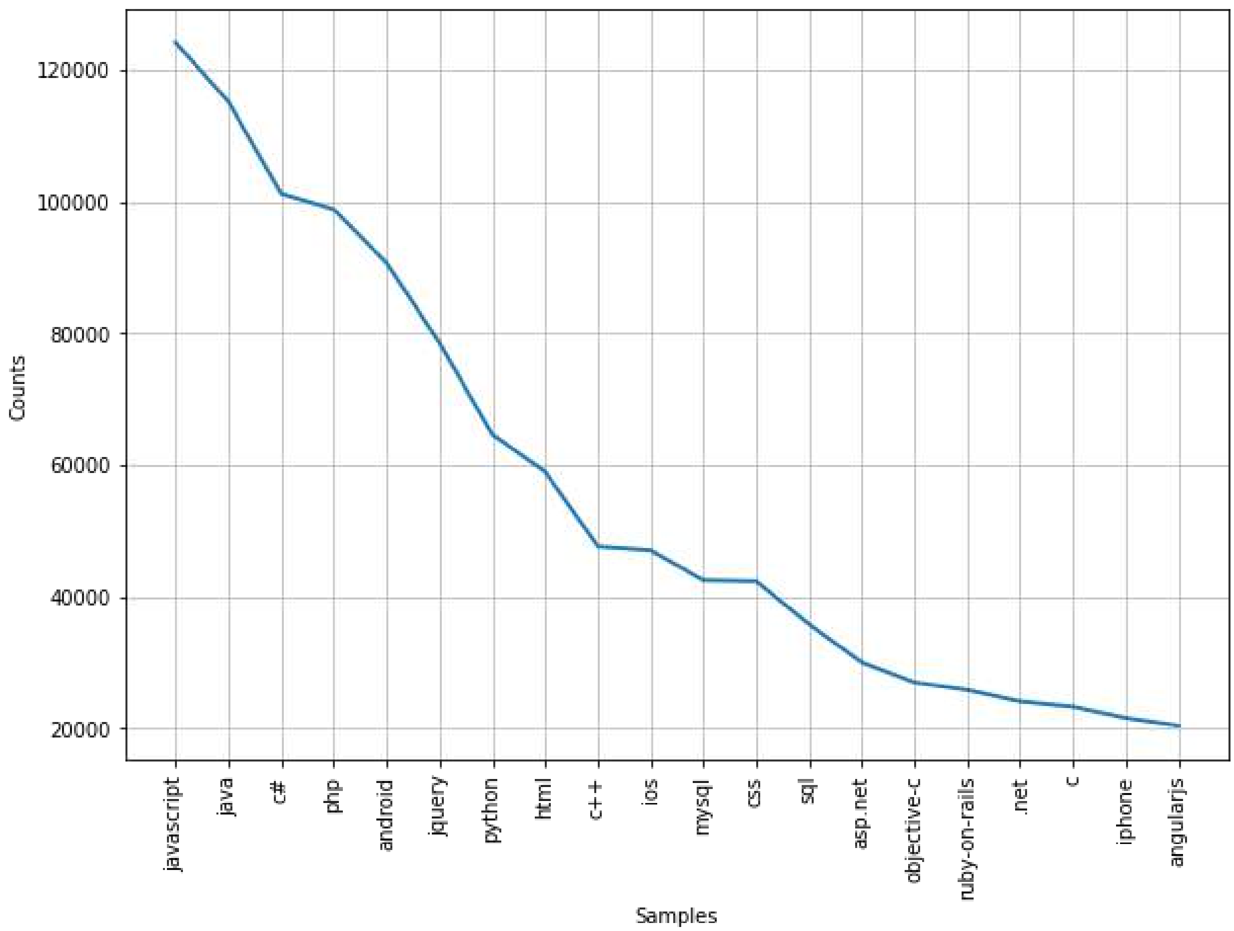


Fig.2: Samples Vs Counts

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

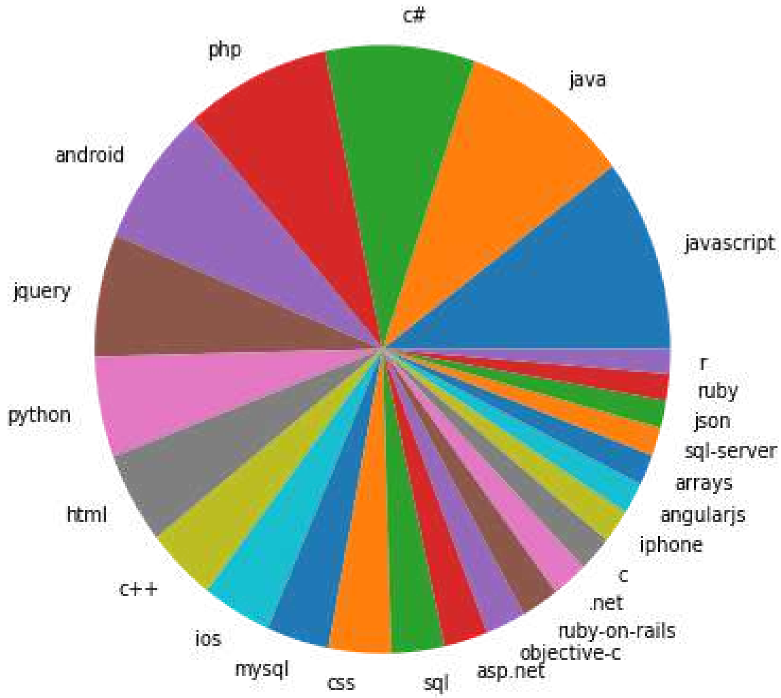


Fig.3: Tags and questions

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

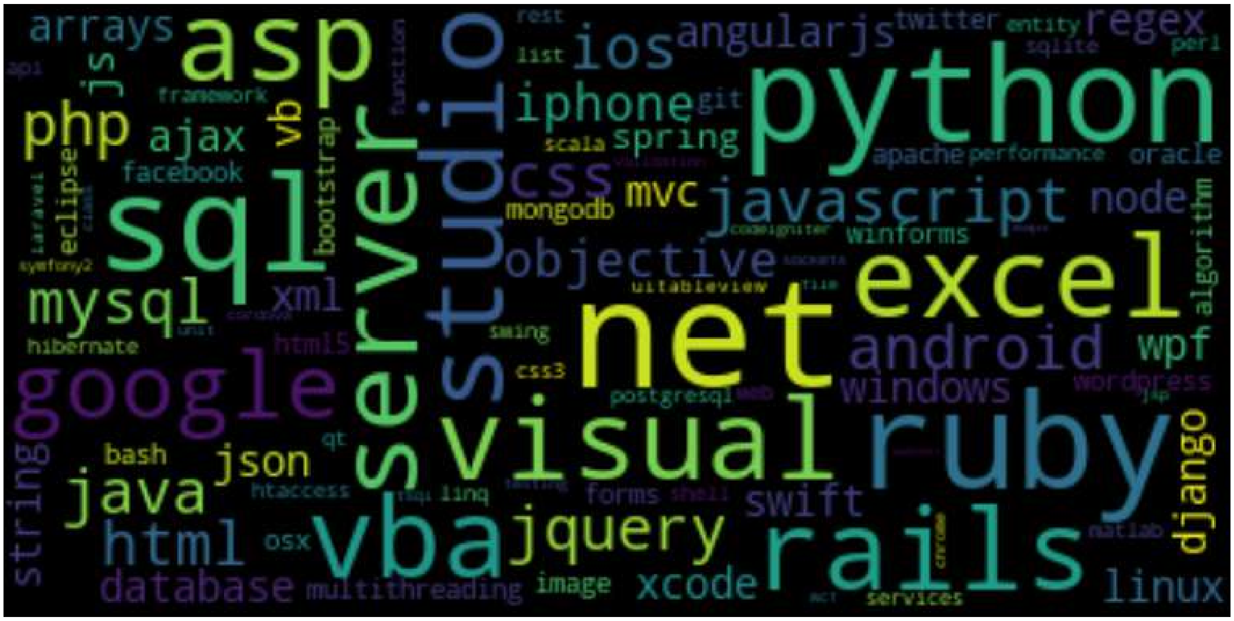


Fig 4: word cloud

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)
```

LSTM

It is a solution for short term memory loss in recurrent neural networks. LSTM is similar to the control flow of a recurrent neural network but only the differences are the operations within the cells of LSTM. LSTM has operations which help in deciding to keep or forget the information which is actually very important to get rid of the short term memory loss problem. It has three different gates to process the information. Those are forget gates,input gate, output gate.

Forget gate:

This gate decides what information should be kept and what information should be forgotten. Based on the sigmoid function output value the forget gate decides to keep or forget the information. If the output value of sigmoid is near to 0 then it forgets the value or else it keeps the value**.**

LSTM

```
from keras.layers import SpatialDropout1D

EMBEDDING_DIM = 100

print('Build model...')

model2 = Sequential()
model2.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model2.add(SpatialDropout1D(0.2))
model2.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model2.add(Dense(5102, activation='relu'))

model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model2.summary())
```

There are four layers added, one is embedding layer, spatial dropout layer, lstm layer and finally an output layer.

```
Build model...
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU
Summary of the built model...
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 4686, 100)         4135100

 spatial_dropout1d (SpatialD  (None, 4686, 100)        0
 ropout1D)

 lstm (LSTM)                 (None, 100)               80400

 dense_7 (Dense)             (None, 5102)              515302

=================================================================
Total params: 4,730,802
Trainable params: 4,730,802
Non-trainable params: 0
_________________________________________________________________
```

For this model I have added only two epochs so the accuracy and loss values are very poor. I have done so because the computing power is not enough but if the epochs and the hyper parameters are changed then the model performance would be very good.

```
num_epochs = 2
batch_size = 100
history = model2.fit(X_train_pad, y_train,
                     batch_size=batch_size,
                     epochs=num_epochs,
                     verbose=2,
                     validation_split=0.2)
```

```
Epoch 1/2
91/91 - 1751s - loss: 0.1634 - accuracy: 0.0308 - val_loss: 0.0114 - val_accuracy: 0.0288 - 1751s/epoch - 19s/step
Epoch 2/2
91/91 - 1779s - loss: 0.0085 - accuracy: 0.0466 - val_loss: 0.0068 - val_accuracy: 0.0288 - 1779s/epoch - 20s/step
```

GRU (Gated Recurrent Unit)

GRU is almost similar to an LSTM. GRU has two gates, one is the reset gate and the other is the update gate.

Update Gate

The update gate is similar to the forget and input gate of an LSTM. It will decide what information to be discarded and what information to be kept.

Reset Gate

The reset gate is another gate which is used to decide how much previous information to be forgotten.

GRU has only fewer tensor operations, so they are a little faster to train compared to LSTM.

```python
#another approach using GRU model, takes longer time
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 100

print('Build model...')

model1 = Sequential()
model1.add(Embedding(vocab_size Loading... IG_DIM, input_length=max_length))
model1.add(GRU(units=32,  dropout=0.2, recurrent_dropout=0.2))
model1.add(Dense(5102, activation='relu'))

# try using different optimizers and different optimizer configs
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model1.summary())
```

As the figure below shows, the model has only three layers, one is embedding, GRU layer, and output layer.

```
Build model...
Summary of the built model...
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 4686, 100)         4135100

 gru (GRU)                   (None, 32)                12864

 dense (Dense)               (None, 5102)              168366

=================================================================
Total params: 4,316,330
Trainable params: 4,316,330
Non-trainable params: 0
_________________________________________________________________
None
```

Similar to the LSTM model, I ran the model with only two epochs for the same reason. But the model performance can be increased with the proper training of the model.

```
num_epochs = 2
batch_size = 100
history = model1.fit(X_train_pad, y_train,
                     batch_size=batch_size,
                     epochs=num_epochs,
                     verbose=2,
                     validation_split=0.2)
```

```
Epoch 1/2
91/91 - 843s - loss: 0.0069 - accuracy: 0.0371 - val_loss: 0.0073 - val_accuracy: 0.0274 - 843s/epoch - 9s/step
Epoch 2/2
91/91 - 910s - loss: 0.0069 - accuracy: 0.0261 - val_loss: 0.0073 - val_accuracy: 0.0274 - 910s/epoch - 10s/step
```

One Vs Rest

In this, we fit one classifier for every class and it is the most generally involved system for multiclass/multi-label classification and is a fair default decision. For every classifier, the class is fitted against a wide range of various classes. In this, I have used SGD classifier and logistic regression.

SGD classifier

In scikit-learn there is a model called SGD Classifier which is a linear classifier optimized by the stochastic gradient descent.

```
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
sgdc = SGDClassifier()
logistic = LogisticRegression()


for clfs in [sgdc, logistic]:
    clf = OneVsRestClassifier(clfs)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    get_score(y_pred, clfs)
```

BERT

In Bert, I have used two models one is bert-base-uncased using hugging transformers and another one is bert-base-cased using hugging transformers and pytorch lightning with different modifications in input data to train each model.

BERT is a transformers model which is a pretrained model on a large corpus of English data. It was pre trained on the raw texts without any labeling, an automatic process to generate inputs and labels from those texts. it was pre trained with two major objectives:

Masked language modeling

It takes a sentence and the model randomly masks 15% of the words in the input and will run the masked sentence through the model then it has to predict the masked words.

Next sentence prediction

The models concatenate two masked sentences as inputs.Then model has to predict whether two sentences are following each other or not.

Bert-base-uncased:

Uncased is nothing but it does not make any difference between the word english and English.

```
# Sections of config

# Defining some key variables that will be used later on in the training
MAX_LEN = 16
TRAIN_BATCH_SIZE = 32
VALID_BATCH_SIZE = 32
EPOCHS = 4
LEARNING_RATE = 1e-05
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
Downloading: 100%  226k/226k [00:00<00:00, 661kB/s]
Downloading: 100%  28.0/28.0 [00:00<00:00, 478B/s]
Downloading: 100%  570/570 [00:00<00:00, 4.48kB/s]
```

The data is transformed into the following manner where all the tags are added as columns and for every question those columns are encoded as 1 or 0 based on the tags associated with the questions.

```
df2
```

	level_0	index	question	target_list
0	4	4	add script functionality .net applications lit...	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
1	5	5	use nest class case ? work collection class us...	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
2	7	8	automatically update version number would like...	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
3	8	10	connect database loop recordset c# ? simplest ...	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
4	9	12	delete file lock another process c# ? look way...	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
...	...	...	...	...
7748	14108	1233496	num++ atomic ' int num ' ? general , int num ,...	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
7749	14109	1235494	strange java behaviour static final qualifiers...	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
7750	14110	1238365	smtp configuration work production try send em...	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
7751	14111	1240566	since xcode 8 ios10 , view size properly viewd...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
7752	14113	1248820	possible write function template return whethe...	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

7753 rows × 4 columns

Only the top 10 tags are considered and questions which have at least one of those top 10 tags are considered for training. So you can see in the above picture, every question has at least one value in the list as 1. The reason behind considering the questions in above fashion is for the proper training. The model performance will be better if only the questions which have at least one of the top ten tags are considered.

```
train_size = 0.8
train_dataset = df2.sample(frac=train_size,random_state=200)
valid_dataset = df2.drop(train_dataset.index).reset_index(drop=True)
train_dataset = train_dataset.reset_index(drop=True)


print("FULL Dataset: {}".format(df2.shape))
print("TRAIN Dataset: {}".format(train_dataset.shape))
print("TEST Dataset: {}".format(valid_dataset.shape))

training_set = CustomDataset(train_dataset, tokenizer, MAX_LEN)
validation_set = CustomDataset(valid_dataset, tokenizer, MAX_LEN)
```

```
FULL Dataset: (7753, 4)
TRAIN Dataset: (6202, 4)
TEST Dataset: (1551, 4)
```

The data after dropping unwanted rows are of the size 7753 and it is splitted into 80% of training data and 20% of test data.

```python
class CustomDataset(Dataset):

    def __init__(self, dataframe, tokenizer, max_len):
        self.tokenizer = tokenizer
        self.data = dataframe
        self.title = dataframe['question']
        self.targets = self.data.target_list
        self.max_len = max_len

    def __len__(self):
        return len(self.title)

    def __getitem__(self, index):
        title = str(self.title[index])
        title = " ".join(title.split())

        inputs = self.tokenizer.encode_plus(
            title,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            return_token_type_ids=True,
            truncation=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']
        token_type_ids = inputs["token_type_ids"]


        return {
            'ids': torch.tensor(ids, dtype=torch.long),
            'mask': torch.tensor(mask, dtype=torch.long),
            'token_type_ids': torch.tensor(token_type_ids, dtype=torch.long),
            'targets': torch.tensor(self.targets[index], dtype=torch.float)
        }
```

The above class is used to encode the questions using bert tokenizer.

```python
class BERTClass(torch.nn.Module):
    def __init__(self):
        super(BERTClass, self).__init__()
        self.l1 = transformers.BertModel.from_pretrained('bert-base-uncased')
        self.l2 = torch.nn.Dropout(0.3)
        self.l3 = torch.nn.Linear(768, 10)

    def forward(self, ids, mask, token_type_ids):
        _, output_1= self.l1(ids, attention_mask = mask, token_type_ids = token_type_ids,return_dict=False)
        output_2 = self.l2(output_1)
        output = self.l3(output_2)
        return output

model = BERTClass()
model.to(device)
```

```
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (10): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
```

Bert class is defined to load the pretrained model from transformers and as the output is 10 the argument to torch.nn.linear is passed as 10 and if the tags are increased then we have to change the number accordingly.

```python
def train_model(start_epochs,  n_epochs, valid_loss_min_input,
                training_loader, validation_loader, model,
                optimizer):

  # initialize tracker for minimum validation loss
  valid_loss_min = valid_loss_min_input


  for epoch in range(start_epochs, n_epochs+1):
    train_loss = 0
    valid_loss = 0

    model.train()
    print('############# Epoch {}: Training Start   #############'.format(epoch))
    for batch_idx, data in enumerate(training_loader):
        #print('yyy epoch', batch_idx)
        ids = data['ids'].to(device, dtype = torch.long)
        mask = data['mask'].to(device, dtype = torch.long)
        token_type_ids = data['token_type_ids'].to(device, dtype = torch.long)
        targets = data['targets'].to(device, dtype = torch.float)

        outputs = model(ids, mask, token_type_ids)

        optimizer.zero_grad()
        loss = loss_fn(outputs, targets)
        #if batch_idx%5000==0:
         #   print(f'Epoch: {epoch}, Training Loss:  {loss.item()}')

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        #print('before loss data in training', loss.item(), train_loss)
        train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.item() - train_loss))
        #print('after loss data in training', loss.item(), train_loss)

    print('############# Epoch {}: Training End     #############'.format(epoch))

    print('############# Epoch {}: Validation Start   #############'.format(epoch))
```

Train model function is used to instantiate and run the model, print the outputs.

In the below picture, you can see the model running and there are only 4 epochs taken.

```
trained_model = train_model(1, 4, np.Inf, training_loader, validation_loader, model,
                      optimizer)

############# Epoch 1: Training Start   #############
############# Epoch 1: Training End     #############
############# Epoch 1: Validation Start   #############
############# Epoch 1: Validation End     #############
Epoch: 1        Avgerage Training Loss: 0.000712        Average Validation Loss: 0.002882
Validation loss decreased (inf --> 0.002882).  Saving model ...
############# Epoch 1  Done   #############

############# Epoch 2: Training Start   #############
############# Epoch 2: Training End     #############
############# Epoch 2: Validation Start   #############
############# Epoch 2: Validation End     #############
Epoch: 2        Avgerage Training Loss: 0.000581        Average Validation Loss: 0.002768
Validation loss decreased (0.002882 --> 0.002768).  Saving model ...
############# Epoch 2  Done   #############

############# Epoch 3: Training Start   #############
############# Epoch 3: Training End     #############
############# Epoch 3: Validation Start   #############
############# Epoch 3: Validation End     #############
Epoch: 3        Avgerage Training Loss: 0.000478        Average Validation Loss: 0.002722
Validation loss decreased (0.002768 --> 0.002722).  Saving model ...
############# Epoch 3  Done   #############

############# Epoch 4: Training Start   #############
############# Epoch 4: Training End     #############
############# Epoch 4: Validation Start   #############
############# Epoch 4: Validation End     #############
Epoch: 4        Avgerage Training Loss: 0.000400        Average Validation Loss: 0.002646
Validation loss decreased (0.002722 --> 0.002646).  Saving model ...
############# Epoch 4  Done   #############
```

Bert-base-cased:

This model is case-sensitive, it finds the word english and English as different.

For this model, we are considering only the questions which have a score greater than 25. So the data count is 10,746.

```
df_questions
```

	Id	OwnerUserId	CreationDate	ClosedDate	Score	Title	Body
0	80	26.0	2008-08-01T13:57:07Z	NaN	26	SQLStatement.execute() - multiple queries in o...	<p>I've written a database generation script i...
1	90	58.0	2008-08-01T14:41:24Z	2012-12-26T03:45:49Z	144	Good branching and merging tutorials for Torto...	<p>Are there any really good tutorials explain...
3	180	2089740.0	2008-08-01T18:42:19Z	NaN	53	Function for creating color wheels	<p>This is something I've pseudo-solved many t...
4	260	91.0	2008-08-01T23:22:08Z	NaN	49	Adding scripting functionality to .NET applica...	<p>I have a little game written in C#. It uses...
5	330	63.0	2008-08-02T02:51:36Z	NaN	29	Should I use nested classes in this case?	<p>I am working on a collection of classes use...
...	...	...	...	...	...	...	...
1233496	39393850	4973224.0	2016-09-08T14:39:51Z	NaN	110	Can num++ be atomic for 'int num'?	<p>In general, for <code>int num</code>, <code...
1235494	39448160	1764889.0	2016-09-12T10:22:48Z	2016-09-13T18:00:35Z	75	Strange Java behaviour with static and final q...	<p>In our team we found some strange behaviour...
1240566	39578530	127493.0	2016-09-19T17:09:18Z	NaN	50	Since Xcode 8 and iOS10, views are not sized p...	<p>It seems that with Xcode 8, on <code>viewDi...
1243077	39640620	1934349.0	2016-09-22T13:35:53Z	NaN	27	What precautions should I take to make a memor...	<p>My initial problem is that I have, on a pro...
1248820	39782090	3187068.0	2016-09-30T00:52:04Z	NaN	44	Is it possible to write a function template wh...	<p>I've been learning about variadic templates...

10746 rows × 7 columns

We are going to consider only the questions associated with top ten tags as below,

```
[ ] # Filter out records ( values in clean_body and tags) that have atleast one of the top tags

    x=[] # To store the filtered clean_body values
    y=[] # to store the corresponding tags
    # Convert to list data type
    lst_top_tags = list(top_tags)

    for i in range(len(df['tags'])):
        temp=[]
        for tag in df['tags'][i]:
            if tag in lst_top_tags:
                temp.append(tag)

        if(len(temp)>0):
            x.append(df['Clean_Body'][i])
            y.append(temp)
```

Then, split the data into train, test and validation sets.

```
[ ] from sklearn.model_selection import train_test_split
    # First Split for Train and Test
    x_train,x_test,y_train,y_test = train_test_split(x, yt, test_size=0.1, random_state=RANDOM_SEED,shuffle=True)
    # Next split Train in to training and validation
    x_tr,x_val,y_tr,y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=RANDOM_SEED,shuffle=True)
```

Then, defined a class to load the dataset and encode it in similar fashion as in the bert-base-uncased model.

```
[ ] class QTagDataset (Dataset):
        def __init__(self,quest,tags, tokenizer, max_len):
            self.tokenizer = tokenizer
            self.text = quest
            self.labels = tags
            self.max_len = max_len

        def __len__(self):
            return len(self.text)

        def __getitem__(self, item_idx):
            text = self.text[item_idx]
            inputs = self.tokenizer.encode_plus(
                text,
                None,
                add_special_tokens=True, # Add [CLS] [SEP]
                max_length= self.max_len,
                padding = 'max_length',
                return_token_type_ids= False,
                return_attention_mask= True, # Differentiates padded vs normal token
                truncation=True, # Truncate data beyond max length
                return_tensors = 'pt' # PyTorch Tensor format
              )

            input_ids = inputs['input_ids'].flatten()
            attn_mask = inputs['attention_mask'].flatten()
            #token_type_ids = inputs["token_type_ids"]

            return {
                'input_ids': input_ids ,
                'attention_mask': attn_mask,
                'label': torch.tensor(self.labels[item_idx], dtype=torch.float)

            }
```

Then initialize the model and encode the questions as below using bert tokenizer,

```python
# Initialize the Bert tokenizer
BERT_MODEL_NAME = "bert-base-cased" # we will use the BERT base model(the smaller one)
Bert_tokenizer = BertTokenizer.from_pretrained(BERT_MODEL_NAME)
```

```python
max_word_cnt = 100
quest_cnt = 0

# For every sentence...
for question in questions:

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    input_ids = Bert_tokenizer.encode(question, add_special_tokens=True)

    # Update the maximum sentence length.
    if len(input_ids) > max_word_cnt:
        quest_cnt +=1

print(f'# Question having word count > {max_word_cnt}: is  {quest_cnt}')
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (629 > 512). Running this sequence through the model will result in indexing errors
# Question having word count > 100: is  3034
```

Setting up the classifier and outputs are directly defined as 10 as we are considering only 10 tags.

```python
class QTagClassifier(pl.LightningModule):
    # Set up the classifier
    def __init__(self, n_classes=10, steps_per_epoch=None, n_epochs=3, lr=2e-5 ):
        super().__init__()

        self.bert = BertModel.from_pretrained(BERT_MODEL_NAME, return_dict=True)
        self.classifier = nn.Linear(self.bert.config.hidden_size,n_classes) # outputs = number of labels
        self.steps_per_epoch = steps_per_epoch
        self.n_epochs = n_epochs
        self.lr = lr
        self.criterion = nn.BCEWithLogitsLoss()

    def forward(self,input_ids, attn_mask):
        output = self.bert(input_ids = input_ids ,attention_mask = attn_mask)
        output = self.classifier(output.pooler_output)

        return output


    def training_step(self,batch,batch_idx):
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        labels = batch['label']

        outputs = self(input_ids,attention_mask)
        loss = self.criterion(outputs,labels)
        self.log('train_loss',loss , prog_bar=True,logger=True)

        return {"loss" :loss, "predictions":outputs, "labels": labels }


    def validation_step(self,batch,batch_idx):
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        labels = batch['label']

        outputs = self(input_ids,attention_mask)
        loss = self.criterion(outputs,labels)
```

Instantiate the classifier defined,

```
# Instantiate the classifier model
steps_per_epoch = len(x_tr)/BATCH_SIZE
model = QTagClassifier(n_classes=10, steps_per_epoch=steps_per_epoch,n_epochs=N_EPOCHS,lr=LR)
#
```

```
Some weights of the model checkpoint at bert-base-cased were not used when initializing BertModel: ['cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.weight', 'cls.predictions.bias',
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequen
```

Training the model,

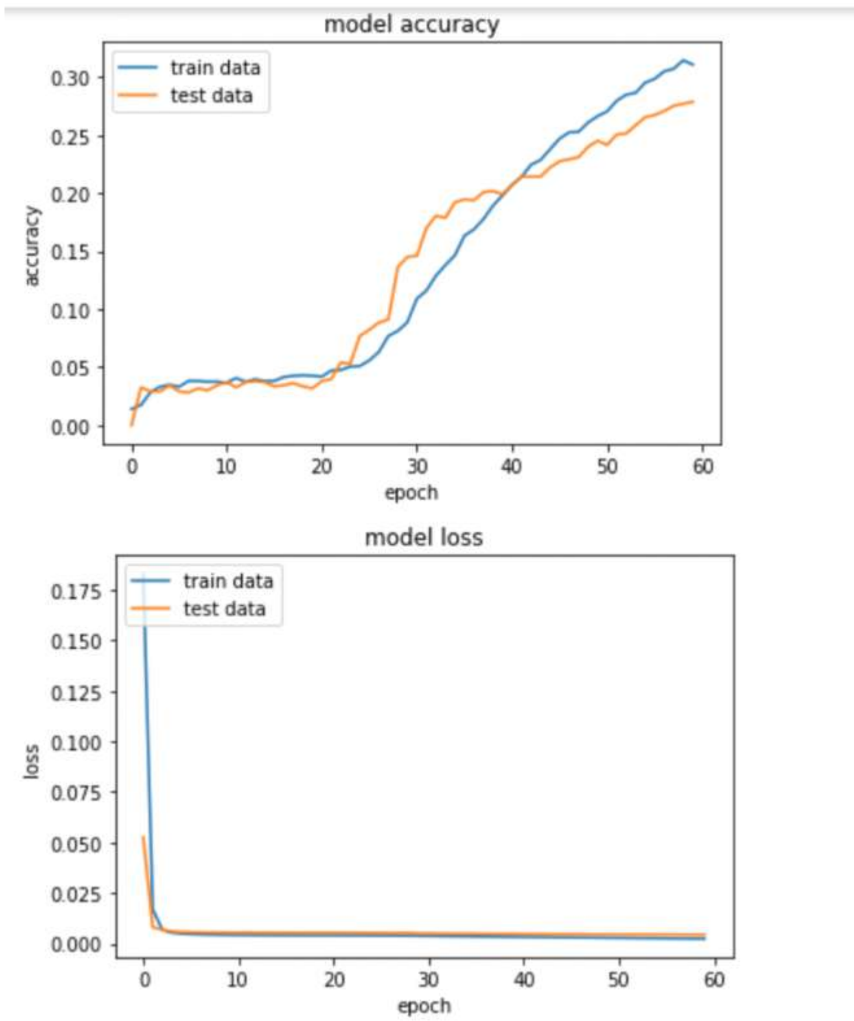
```
trainer.fit(model, QTdata_module)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:310: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementati
  FutureWarning,

  | Name       | Type              | Params
-------------------------------------------------
0 | bert       | BertModel         | 108 M
1 | classifier | Linear            | 7.7 K
2 | criterion  | BCEWithLogitsLoss | 0
-------------------------------------------------
108 M     Trainable params
0         Non-trainable params
108 M     Total params
433.272   Total estimated model params size (MB)
Sanity Checking DataLoader 0: 100%  2/2 [00:00<00:00, 24.95it/s]
Epoch 4: 100%  201/201 [01:28<00:00, 2.28it/s, loss=0.344, v_num=2, train_loss=0.348, val_loss=0.340]
Validation DataLoader 0: 100%  67/67 [00:05<00:00, 11.21it/s]
Validation DataLoader 0: 100%  67/67 [00:06<00:00, 10.83it/s]
Validation DataLoader 0: 100%  67/67 [00:05<00:00, 11.11it/s]
Validation DataLoader 0: 100%  67/67 [00:05<00:00, 11.18it/s]
Validation DataLoader 0: 100%  67/67 [00:05<00:00, 11.14it/s]
```

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:

```
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.

LSTM

```
[ ] def predict(str):
      test =  tokenizer_obj.texts_to_sequences([str])
      test1 = pad_sequences(test, maxlen=max_length, padding='post')

      # test = tokenizer.texts_to_matrix([str], mode='tfidf')
      prediction = model2.predict(np.array(test1))
      print("text: "+ str)

      for i in range(2):
        j = np.argmax(prediction[0])
        predicted_label = target_labels[j]
        prediction[0][j] = 0;
        print("Predicted label: " + predicted_label)
```

```
predict("Why does it throw an IndexOutOfBoundsException")
```

```
text: Why does it throw an IndexOutOfBoundsException
Predicted label: c#
Predicted label: java
```

The results of LSTM are not as good as the results of RNN. but if the model is trained properly by increasing and changing the hyper parameters then the performance could be increased.

GRU

```
def predict(str):
  test =  tokenizer_obj.texts_to_sequences([str])
  test1 = pad_sequences(test, maxlen=max_length, padding='post')

  # test = tokenizer.texts_to_matrix([str], mode='tfidf')
  prediction = model1.predict(np.array(test1))
  print("text: "+ str)

  for i in range(2):
    j = np.argmax(prediction[0])
    predicted_label = target_labels[j]
    prediction[0][j] = 0;
    print("Predicted label: " + predicted_label)
```

```
[ ] predict("Why does it throw an IndexOutOfBoundsException")
```

```
text: Why does it throw an IndexOutOfBoundsException
Predicted label: java
Predicted label: c#
```

As LSTM, The results of GRU are also not as good as the results of RNN. As you can see the accuracy and loss are very poor, but if the model is trained properly by increasing and changing the hyper parameters then the performance could be increased.

One Vs Rest

```
Clf:  SGDClassifier
Jacard score: 0.223922619047619
Hamming loss: 0.1605295191722459
Clf:  LogisticRegression
Jacard score: 0.03270833333333333
Hamming loss: 0.1757455873402313
```

As the figure shows, the hamming loss of SGD is little better than the logistic regression.

BERT (bert-base-uncased)

The accuracy of the model is quite better, as it is 55.2%

```
accuracy = metrics.accuracy_score(val_targets, val_preds)
f1_score_micro = metrics.f1_score(val_targets, val_preds, average='micro')
f1_score_macro = metrics.f1_score(val_targets, val_preds, average='macro')
print(f"Accuracy Score = {accuracy}")
print(f"F1 Score (Micro) = {f1_score_micro}")
print(f"F1 Score (Macro) = {f1_score_macro}")
```

```
Accuracy Score = 0.5527079303675049
F1 Score (Micro) = 0.6907317073170732
F1 Score (Macro) = 0.6753075044430766
```

The classification report for the model,

```
print(classification_report(val_targets, val_preds))
```

	precision	recall	f1-score	support
0	0.66	0.39	0.49	1012
1	0.85	0.40	0.54	1112
2	0.82	0.31	0.45	1040
3	0.94	0.42	0.58	436
4	0.91	0.70	0.79	896
5	0.93	0.51	0.66	536
6	0.92	0.62	0.74	668
7	0.64	0.07	0.12	364
8	0.89	0.41	0.56	564
9	0.69	0.20	0.31	364
micro avg	0.84	0.43	0.57	6992
macro avg	0.83	0.40	0.53	6992
weighted avg	0.83	0.43	0.55	6992
samples avg	0.25	0.25	0.25	6992

Bert-base-cased:

```
# Evaluate the model performance on the test dataset
trainer.test(model,datamodule=QTdata_module)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
Testing DataLoader 0: 100% 38/38 [00:03<00:00, 11.86it/s]
```

Test metric	DataLoader 0
test_loss	0.3384649455547333

```
[{'test_loss': 0.3384649455547333}]
```

According to the threshold, the values which are greater than the threshold are considered and predicted tags are being displayed. If we identify the threshold correctly, then the results would be much more accurate.

The prediction is done on the test set and the results are shown in the below figure,

```
[ ] y_pred = mlb.inverse_transform(np.array(y_pred_labels))
    y_act = mlb.inverse_transform(flat_true_labels)

    df = pd.DataFrame({'Body':x_test,'Actual Tags':y_act,'Predicted Tags':y_pred})
    df
```

	Body	Actual Tags	Predicted Tags
0	simple one to start the day given a dictionary...	(c#,)	(c#,)
1	what are the difference and connection between...	(c#,)	()
2	os windows bit java jdk i have a jnlp file whe...	(java,)	(java,)
3	i am running ntlm using spring security i am g...	(java,)	(c#,)
4	right now i m doing for char c a c z c alphabe...	(java,)	(c#, java)
...	...	...	...
588	in multithreaded net programming what are the ...	(c#,)	()
589	i m at a bit of a loss i ve used the layer pro...	(ios,)	()
590	i got one problem and i have described it belo...	(ios,)	(c#, javascript)
591	original problem in building our projects i wa...	(c#,)	()
592	i am trying to figure out a way to be able to ...	(android,)	()

593 rows × 3 columns

```
[ ] print(metrics.classification_report(y_true,y_pred))

              precision    recall  f1-score   support

           0       0.90      0.96      0.93      5290
           1       0.26      0.12      0.16       640

    accuracy                           0.87      5930
   macro avg       0.58      0.54      0.55      5930
weighted avg       0.83      0.87      0.85      5930
```

Project Management

Work Completed,

Team Member	Responsibility
Alekhya Vachakarla	1) Brainstorming topics on classification. 2) Research on Project Idea, collection of dataset from kaggle. 3) Added different visualizations of data. 4) Data preparation/ cleaning. 5) Implemented a basic sequential model with deep learning layers. 6) Implemented Random Forest Classifier. 7) Implemented LSTM model. 8) Implement GRU model. 9) Implemented Bert-based-uncased using hugging transformers. 10) Implemented Bert-base-cased using hugging transformers and pytorch

	lightning. 11) Implemented one Vs Rest classifier model. 12) Added Evaluation metrics for models. 13) Project documentation.

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

- Micheal Phi, Illustrated Guide to LSTM and GRU: A step by step explanation https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

- Kaw khaung, Multi-label Text Classification with BERT using Pytorch https://kyawkhaung.medium.com/multi-label-text-classification-with-bert-using-pytorch-47011a7313b9

- Prathik Nabriya, Topic Modeling: Predicting Multiple Tags of Research Articles using OneVsRest strategy https://www.analyticsvidhya.com/blog/2021/09/onevsrest-classifier-for-predicting-multiple-tags-of-research-articles/

- Prasad Nageshkar, Multi-label Text Classification using Transformers(BERT) https://medium.com/analytics-vidhya/multi-label-text-classification-using-transformers-bert-93460838e62b