



## **PG Project**

**Project Topic :**

**Multilabel Coding Questions Classification**

**Team Members :**

Sarvat Ali (2018201009)  
Prabha Pandey (2018201053)  
C. Sai Sukrutha (2018201054)  
Shafiya Naaz (2018201062)

**Submitted to :**

Dr. Radhika Mamidi

# **REPORT**

## **Introduction**

### **Problem Statement**

Multilabel classification of questions from different coding platforms like codechef, codeforces etc and labelling them based on their respective paradigms.

### **Motivation**

- Create an application that helps the user to identify the category of question he/she is trying to solve.
- To facilitate the user to be able to access multiple websites at one place and get all the questions related to the programming paradigm of interest

## Multi-label Classification

Multi-label classification is a generalization of multi-class classification which is the single-label problem of categorizing instances into precisely one of more than two classes, in the multi-label problem there is no constraint on how many of the classes the instance can be assigned to i.e there could be one, two or many labels in the output data used for training. In Multi-Label, the classes are not mutually exclusive and can occur 'more than one' for the problem in hand whereas Multi-Class have classes that are mutually exclusive which means that at a time 'only single class' can be assigned to the concerned problem.

Formally, multi-label classification is the problem of finding a model that maps inputs  $\mathbf{x}$  to binary vectors  $\mathbf{y}$  (assigning a value of 0 or 1 for each element (label) in  $\mathbf{y}$ ).

The problem we have taken is multi-label classification because a programming problem may belong to different labels. For example, a problem can be related to 'graphs' and can be solved using the technique 'dp (dynamic programming)'. This problem will have both 'graphs', 'dp' as labels.

## Data Scrapping

A separate python script was written for scrapping the data each from codechef and codeforces. The scraped data is stored in a csv file, which is further used in data preprocessing to make it usable.

## Data Preprocessing

Following are the steps performed for cleaning up of data set :

1. Removing Stop Words
2. Removing Special Characters
3. Applying Stemming ( as per English Dictionary )
4. The above approaches are not applied on solution.

Also there were some tags scrapped out which were gibberish, we had to remove that set of unused tags.

The list of used tags are as follows:

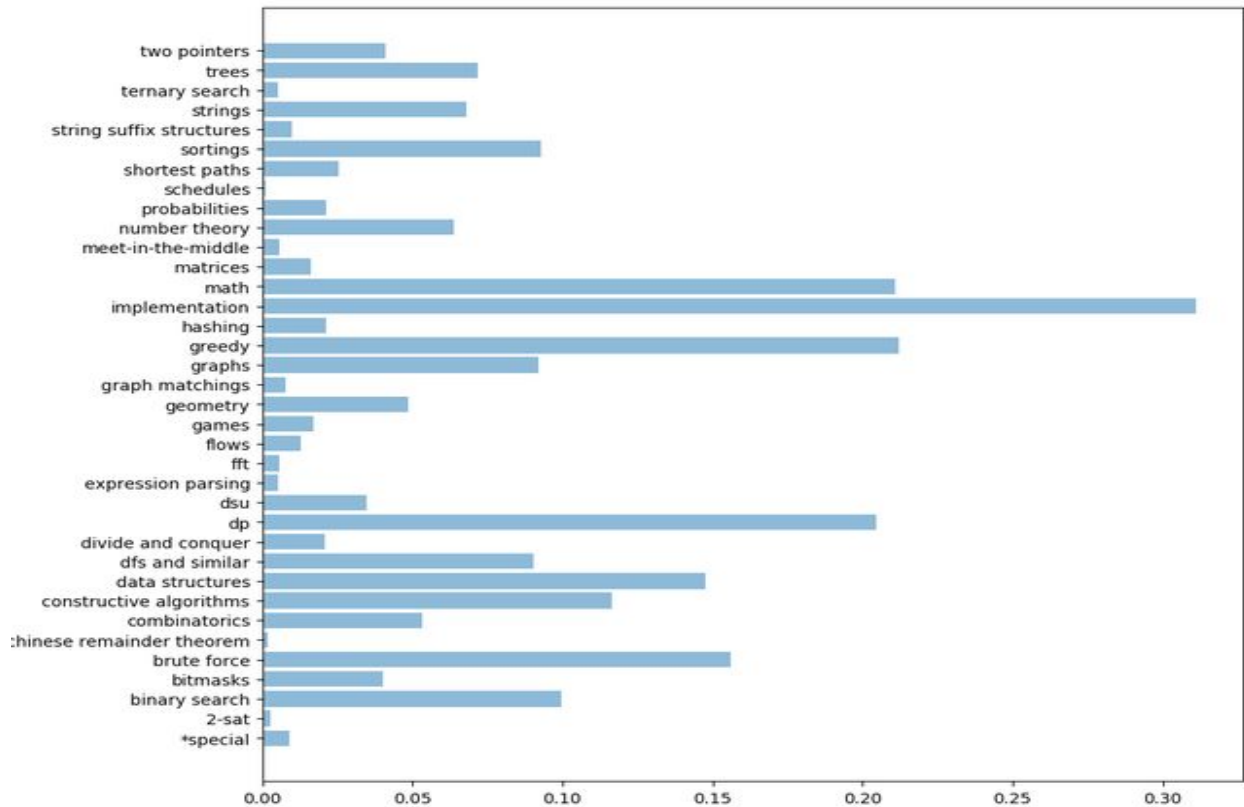
```
global tags_list_codeforces , tags_list_codechef

tags_list_codeforces = ['dsu', 'trees', 'chinese remainder theorem', 'sortings', 'games', 'implementation', 'bitmask',
                        '*special', 'hashing', 'geometry', 'two pointers', 'combinatorics', 'flows', 'strings',
                        'probabilities', 'data structures', 'ternary search', 'greedy', 'math', 'matrices',
                        'divide and conquer', 'dfs and similar', 'constructive algorithms', 'brute force', 'dp',
                        '2-sat', 'graph matchings', 'binary search', 'number theory', 'graphs', 'fft', 'shortest paths',
                        'schedules', 'meet-in-the-middle', 'string suffix structures', 'expression parsing']

tags_list_codechef = ['tree', 'binarysearch', 'combinatorial', 'gcd', 'dijkstra', 'memoization', 'bipartite', 'fibonacci',
                      'strings', 'suffix', 'geometry', 'knapsack', 'sorting', 'recursion', 'pointers', 'maxflow', 'binary', 'feasibility',
                      'constructive', 'expo', 'graph', 'simulation', 'fft', 'algorithm', 'dfs', 'heap', 'bitmasking', 'hashing',
                      'combinatorics', 'graphs', 'greedy', 'interactive', 'bfs', 'implementation', 'advanced', 'number', 'parity',
                      'prime', 'dynamic', 'deque', 'sets', 'disjoint', 'bitwise', 'digraph', 'theory', 'backtracking', 'probability',
                      'series', 'matrix', 'divide', 'kruskal', 'pattern', 'bruteforce', 'easy', 'hard', 'trees', 'maths', 'enumeration',
                      'regex', 'tries', 'algebra', 'matching', 'multiset', 'euler', 'inversions', 'array', 'segment', 'permutation',
                      'recurrence', 'dp', 'ad hoc']
```

## Managing Imbalanced Classes

The set of tags used had imbalance in frequency of their occurrence as shown below:



As it is pretty evident that the number of occurrence of some classes were more than the others, hence introducing imbalance.

Imbalanced classes put “accuracy” out of business. Standard accuracy no longer reliably measures performance.

There are two methods to deal with it:

- Up-sample Minority Class
- Down-sample Majority Class

Up-sampling: It is the process of **randomly duplicating observations from the minority class** in order to reinforce its signal.

Down-sampling: It involves **randomly removing observations from the majority class** to prevent its signal from dominating the learning algorithm.

As we didn't have a ton of data to work with, our best bet was up-sampling, which we used to remove class imbalance from the data. For which we found out the tags which had frequency of  $< 0.02$ , considered them as minority class and duplicated their observations.

The code snippet is:

```
In [20]: 1 def oversample_train_data(X_train,Y_train):
2         global X, Y, mlb, distinct_tags
3         X_train = pd.DataFrame(X_train).values
4         XY = np.column_stack((X_train,Y_train))
5         labels = []
6         data_to_repeat_index = [1]*Y_train.shape[0]
7
8         for index in range(Y_train.shape[1]):
9             if np.sum(Y_train[:,index])/Y_train.shape[0] < 0.02:
10                 labels.append(index)
11
12         for index in range(Y_train.shape[0]):
13             for label in labels:
14                 if Y_train[index][label] == 1:
15                     data_to_repeat_index[index] = 200
16
17         XY = np.repeat(XY, repeats = data_to_repeat_index, axis=0)
18         X_train = XY[:,0]
19         Y_train = XY[:,1:]
20         X_train = pd.Series(X_train)
21         # X_train = X_train.astype(str)
22         # Y_train = Y_train.astype('int')
23         return X_train,Y_train, XY
```

## Baseline Models

In baseline models, we have implemented linear models like SVM, Logistic Regression and neural network based models like LSTM and CNN.

For baseline linear models , OneVsRest strategy and Pipeline are used in both the linear methods.Lets understand them first.

OneVsRest multi-label strategy : Multi-label algorithm accepts a binary mask over multiple labels. The result for each prediction will be an array of 0s and 1s marking which class labels apply to each row input sample. So, we employ OneVsRest scheme to get binary masks to multiple labels.

One-vs-rest (also called OneVsAll) strategy involves training a single classifier per class, with the samples of that class as positive samples and all rest other samples as negatives. This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample.

One-Vs-Rest treats the problem as mutually exclusive on classes and trains N different classifiers (N=no. of labels) corresponding to each class and converting this problem into a binary classification problem.

Advantages:

- Provides probabilities for each label.
- It is more robust: the independent variables don't have to be normally distributed.

Disadvantages:

- When the number of classes are huge, this technique results in the increased time for training and prediction.
- We assume the class independence.

Pipeline : Scikit-learn provides a pipeline utility to help automate machine learning workflows. Pipeline means to sequentially apply a list of transforms and a final estimator. Intermediate steps of pipeline must implement fit and transform methods and the final estimator only needs to implement fit. Pipelines are very common in Machine Learning systems, since there is a lot of data to manipulate and many data transformations to apply.

## **Linear SVM**

Support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum-margin classifier.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

### **Advantages of SVM :**

1. Works well with even unstructured and semi structured data like text.
2. It scales relatively well to high dimensional data.

### **Disadvantages of SVM :**

1. Choosing a “good” kernel function is not easy.



2. It is not that easy to fine-tune the SVM hyper-parameters. It is hard to visualize their impact

## Metrics Used :

Precision: It calculates the proportion of positive predictions, those are actually correct.

Recall: It calculates the proportion of actual positives that were identified correctly.

F1 Score : F1 score is calculated using the harmonic mean of precision and recall.

$$\text{F1 Score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

## Modelling using SVM:

### 1. Defining the model

```
1 classifier = make_pipeline(  
2     CountVectorizer(ngram_range = (1,3),binary = True,lowercase=False),  
3     TfidfTransformer(norm = 'l2',sublinear_tf = True),  
4     OneVsRestClassifier(LinearSVC(penalty="l2",loss="squared_hinge",tol=1,random_state=0,max_iter=1000,C = 1)))  
5  
6 classifier.fit(X_train, Y_train)
```

### 2. Making predictions and get score

```
1 def get_scores( _X , _Y ):  
2     predicted = classifier.predict(_X)  
3     _Y[int(_Y.shape[0]/2),:] = 1  
4     y_labels_predicted = mlb.inverse_transform(predicted)  
5     y_labels_actual = mlb.inverse_transform(_Y)  
6  
7     print("precision_score: ",precision_score(_Y,predicted,average = 'weighted'))  
8     print("recall_score: ",recall_score(_Y,predicted,average = 'weighted'))  
9     print("f1_score: ",f1_score(_Y,predicted,average = 'weighted'))  
10    print()
```

Obtained scores for Linear SVM are:

Precision : 0.76

Recall : 0.82

F1-score : 0.78

## **Logistic Regression**

Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. Logistic Regression is used when the dependent variable(target) is categorical.

In logistic model, the log-odds (the logarithm of the odds) or sigmoid function is used. The defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at a constant rate, with each independent variable having its own parameter; for a binary dependent variable this generalizes the odds ratio.

### **Advantages of Logistic Regression :**

1. Logistic regression is easier to implement, interpret and very efficient to train.
2. Logistic regression is less prone to overfitting but it can overfit in high dimensional datasets.

### **Disadvantages of Logistic Regression :**

1. Main limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.
2. Logistic Regression can only be used to predict discrete functions.

## Metrics Used :

Precision, Recall, F1-score ( same as in SVM)

## Modelling using Logistic Regression:

### 1. Defining the model

```
1 classifier = make_pipeline(  
2     CountVectorizer(ngram_range = (1,3),binary = True,lowercase=False),  
3     TfidfTransformer(norm = 'l2',sublinear_tf = True),  
4     OneVsRestClassifier(LogisticRegression(penalty='l2',solver='lbfgs', max_iter=100)))  
5  
6 for tag in distinct_tags:  
7     print('Processing {} problems...'.format(tag))  
8  
9     classifier.fit(X_train, Y_train)  
10  
11     prediction = classifier.predict(X_validation)  
12     print('Test accuracy is {}'.format(accuracy_score(Y_validation, prediction)))  
13     print("\n")
```

### 2. Making predictions and get score

```
1 def get_scores( _X , _Y ):  
2     predicted = classifier.predict(_X)  
3     _Y[int(_Y.shape[0]/2),:] = 1  
4     y_labels_predicted = mlb.inverse_transform(predicted)  
5     y_labels_actual = mlb.inverse_transform(_Y)  
6  
7     print("precision_score: ",precision_score(_Y,predicted,average = 'weighted'))  
8     print("recall_score: ",recall_score(_Y,predicted,average = 'weighted'))  
9     print("f1_score: ",f1_score(_Y,predicted,average = 'weighted'))  
10    print()
```

Obtained scores for Logistic Regression are:

Precision : 0.72

Recall : 0.87

F1-score : 0.78

## **LSTM**

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

### **Advantages of LSTM :**

1. LSTM can process inputs of any length.
2. An LSTM model is modeled to remember each information throughout the time which is very helpful in any time series predictor.
3. Even if the input size is larger, the model size does not increase.
4. The weights can be shared across the time steps.
5. LSTM can use their internal memory for processing the arbitrary series of inputs which is not the case with feedforward neural networks.

## Disadvantages of LSTM :

1. Due to its recurrent nature, the computation is slow.
2. Training of LSTM models can be difficult.
3. If we are using relu or tanh as activation functions, it becomes very difficult to process sequences that are very long.
4. Prone to problems such as exploding and gradient vanishing.

## Metric Used :

F1 Score : F1 score is calculated using the harmonic mean of precision and recall.

$$\text{F1 Score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

This F1 score is micro averaged to use it as a metric for multi-label classification. It is calculated by counting the value of true positives, false positives, true negatives, and false negatives. All the predicted outputs, in this case, are column indices and are used in sorted order by default.

## Modeling Using LSTMs :

### 1. Padding and making all input sequences of the same length and preparing input sequences :

```
n_most_common_words = 2000
max_len = 500
tokenizer = Tokenizer(num_words=n_most_common_words, filters='; / @ ? ` ~', lower=False)
tokenizer.fit_on_texts(X.values)
sequences = tokenizer.texts_to_sequences(X.values)
word_index = tokenizer.word_index
_X = pad_sequences(sequences, maxlen=max_len)
```

## 2. Defining Model :

For this , we are using the embedding layer as the first layer and a 78 (Total number of unique tags) dimension dense layer as the output layer.

```
model = Sequential()  
model.add(Embedding(n_most_common_words, emb_dim))  
model.add(LSTM(128, dropout=0.1))  
model.add(Dense(78, activation='sigmoid'))  
print(model.summary())
```

Why Sigmoid and not Softmax in the final dense layer?

In the final layer of the above architecture, sigmoid function has been used instead of softmax. The advantage of using sigmoid over Softmax lies in the fact that one question may have many possible labels. Using the Softmax function would imply that the probability of occurrence of one label depends on the occurrence of other labels. But for this application, we need a function that would give scores for the occurrence of labels, which would be independent of occurrences of any other label.

## 3. Training using 'adam' as an optimizer and binary cross-entropy as the loss function :

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])  
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2,  
                    callbacks=[EarlyStopping(monitor='val_loss', patience=7, min_delta=0.0001)])
```

## 4. Analysis of Model and calculating the f1 micro score :

The final dense layer in the model has 78 (Total number of unique tags) dimensions. Each dimension in the output has a score between 0 and 1, 0 being the least probable score for any genre and 1 being the best score.

A threshold matrix has been defined, with values in range 0.1 to 0.9. Then, we run a loop over the predicted output and compare it with the threshold value and choose tags only if the corresponding value of a tag is more than the threshold value.

This helps in two ways:

1. Choosing the best threshold value, and using it to predict tags.
2. Calculating the micro averaged F1 score, by comparing the tags predicted in each iteration and the original tags in the test dataset.

```
y_pred_prob = model.predict_proba(X_validation)
thresholds=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
```

```
for val in thresholds:
    y_pred_new = (y_pred_prob >= val).astype(int)
    precision = precision_score(Y_validation, y_pred_new, average='micro')
    recall = recall_score(Y_validation, y_pred_new, average='micro')
    f1 = f1_score(Y_validation, y_pred_new, average='micro')

    print("Micro-average quality numbers")
    print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
```

```
For threshold: 0.1
Micro-average quality numbers
Precision: 0.2298, Recall: 0.4838, F1-measure: 0.3116
For threshold: 0.2
Micro-average quality numbers
Precision: 0.3091, Recall: 0.3638, F1-measure: 0.3342
For threshold: 0.3
Micro-average quality numbers
Precision: 0.3663, Recall: 0.2902, F1-measure: 0.3238
For threshold: 0.4
Micro-average quality numbers
Precision: 0.4177, Recall: 0.2345, F1-measure: 0.3004
For threshold: 0.5
Micro-average quality numbers
Precision: 0.4738, Recall: 0.1933, F1-measure: 0.2746
For threshold: 0.6
Micro-average quality numbers
Precision: 0.5327, Recall: 0.1572, F1-measure: 0.2427
For threshold: 0.7
Micro-average quality numbers
Precision: 0.5785, Recall: 0.1246, F1-measure: 0.2050
For threshold: 0.8
Micro-average quality numbers
Precision: 0.6372, Recall: 0.0927, F1-measure: 0.1618
For threshold: 0.9
Micro-average quality numbers
Precision: 0.6852, Recall: 0.0601, F1-measure: 0.1105
```

Final scores for LSTM are :

Precision : 0.3091



Recall : 0.3638

F1 - score : 0.3342

The F1 score for different thresholds signifies how the F1 metric score changes with different threshold values. It goes as per what could have been expected out of it - A very large or a very small value of threshold gives a lower value of F1 metric score because when tags are chosen based on a lower threshold value, too many tags get chosen which reduce the F1 metric score, while when the threshold value gets very large, almost no tags get chosen and thus reducing the performance metric.

## **CNN**

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.



## **Advantages of CNN :**

1. Minimize computation compared to a regular neural network.
2. Convolution simplifies computation to a great extent without losing the essence of the data .
3. They are great at handling image classification.
4. They use the same knowledge across all image locations
5. CNNs eliminate the need for manual feature extraction.
6. CNNs can be retrained for new recognition tasks, enabling us to build on pre-existing networks.

## **Disadvantages of CNN :**

1. They need a lot of training data.
2. If the GPU is not good, they are quite slow to train (for complex tasks).

## **Metric Used :**

F1 - score ( same as in LSTM )

## **Modeling Using CNNs :**

The first step remains the same as that of what we did in the above model for LSTMs. The first layer is also the same here and we have used an embedding layer followed by fully connected layers. One can use other variations and depth of layers and also try out different values of Dropouts.

```

model = Sequential()
model.add(Embedding(n_most_common_words, 36, input_length=500))
model.add(Conv1D(64, 3, activation='sigmoid'))
model.add(Conv1D(100, 3, activation='sigmoid'))
model.add(Conv1D(100, 3, activation='sigmoid'))
# model.add(Dropout(0.70))
model.add(Conv1D(48, 3, activation='sigmoid'))
model.add(Flatten())
model.add(Dense(36))

model.summary()

```

Training using adam optimizer and binary cross-entropy.

```

model.compile(optimizer='adam', loss='binary_crossentropy')
model.fit(X_train, Y_train,
          epochs=10,
          verbose=False,
          validation_data=(X_validation, Y_validation),
          batch_size=16)

```

A threshold matrix has been defined, with values in range 0.1 to 0.9. ( same as done in LSTM )

```

y_pred_prob = model.predict_proba(X_validation)
thresholds=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

```

```

for val in thresholds:
    y_pred_new = (y_pred_prob >= val).astype(int)
    precision = precision_score(Y_validation, y_pred_new, average='micro')
    recall = recall_score(Y_validation, y_pred_new, average='micro')
    f1 = f1_score(Y_validation, y_pred_new, average='micro')
    print("Micro-average quality numbers")
    print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

```

```
For threshold: 0.1
Micro-average quality numbers
Precision: 0.2299, Recall: 0.5640, F1-measure: 0.3266
For threshold: 0.2
Micro-average quality numbers
Precision: 0.3369, Recall: 0.3674, F1-measure: 0.3515
For threshold: 0.3
Micro-average quality numbers
Precision: 0.4548, Recall: 0.2483, F1-measure: 0.3212
For threshold: 0.4
Micro-average quality numbers
Precision: 0.5458, Recall: 0.1745, F1-measure: 0.2644
For threshold: 0.5
Micro-average quality numbers
Precision: 0.6115, Recall: 0.1343, F1-measure: 0.2203
For threshold: 0.6
Micro-average quality numbers
Precision: 0.6531, Recall: 0.0927, F1-measure: 0.1623
For threshold: 0.7
Micro-average quality numbers
Precision: 0.7085, Recall: 0.0488, F1-measure: 0.0913
For threshold: 0.8
Micro-average quality numbers
Precision: 0.7125, Recall: 0.0126, F1-measure: 0.0248
For threshold: 0.9
Micro-average quality numbers
Precision: 0.0000, Recall: 0.0000, F1-measure: 0.0000
```

Final scores for CNN are :

Precision : 0.3369

Recall : 0.3674

F1 - score : 0.3515

The same trend of a lower metric score for the very higher or very low value of a threshold.

We have tried out two deep learning architectures namely LSTMs and CNNs respectively, and then make it work for multi-label classification problems. We started with data preprocessing, followed by defining the models. Once the model has been trained, we use different thresholds and then choose tags based on the threshold score which gave the best F1 micro score on the test dataset.

## Advanced Models

### **Evaluations**

We considered following metrics for evaluation in our Advanced models.

**Label ranking average precision (LRAP)** averages over the samples, answers to the question: for each ground truth label, what fraction of higher-ranked labels were true labels? This performance measure will be higher if you are able to give better rank to labels associated with each sample. This metric is used in multilabel ranking problem, where the goal is to give better rank to labels associated to each sample. The obtained score is always strictly greater than 0 and the best value is 1.

The **label\_ranking\_loss(LRL)** function computes ranking loss which averages over the samples the number of label pairs that are incorrectly ordered, i.e. true labels have a lower score than false labels

### **BERT**

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Bert is a transformer based model that has achieved state-of-the-art results on various NLP tasks. Any general transformer uses an encoder and a decoder network, however, as BERT is a pre-training model, it only uses the encoder to learn a latent representation of the input text. Two things are considered in this Masked Language Model (MLM) and Next Sentence Prediction (NSP). MLM makes it possible to perform bidirectional learning from the text, i.e. it allows the model to learn the context of each word from the words appearing both before and after it. The MLM pre-training task converts the text into tokens and uses the token representation as an input and output for the training. A random subset of

the tokens (15%) are masked, i.e. hidden during the training, and the objective function is to predict the correct identities of the tokens. The NSP task allows BERT to learn relationships between sentences by predicting if the next sentence in a pair is the true next or not. For this 50% correct pairs are supplemented with 50% random pairs and the model trained. BERT trains both MLM and NSP objectives simultaneously. BERT was trained using 3.3 Billion words total with 2.5B from Wikipedia and 0.8B from BooksCorpus. The training was done using TPU. This allows us to use a pre-trained BERT model by fine-tuning it for our multilabel classification task.

In our task we fine tuned this model on segment of data using following parameters Learning rate=  $5e-5$  , Training epochs = 3 and obtained metrics as

LRAP = 0.98

LRL= 0.013

### **RoBERTa**

An optimized BERT approach RoBERTa, is a retraining of BERT with improved training methodology, 1000% more data and compute power. To improve the training procedure, RoBERTa removes the Next Sentence Prediction (NSP) task from BERT's pre-training and introduces dynamic masking so that the masked token changes during the training epochs. RoBERTa uses 160 GB of text for pre-training including data used to train BERT and training with much larger mini-batches and learning rates. Hence for many tasks it gives better results than BERT.

In our task we trained fine tuned this model for a segment of data using following parameters Learning rate=  $5e-5$  , Training epochs = 3 and obtained metrics as

LRAP = 0.99

LRL= 0.008

## **XLNET**

It is a large bidirectional transformer that uses improved training methodology, larger data and more computational power to achieve better than BERT prediction metrics on 20 language tasks. It introduces permutation language modeling, where all tokens are predicted but in random order. This is in contrast to BERT's masked language model where only the masked (15%) tokens are predicted. This is also in contrast to the traditional language models, where all tokens were predicted in sequential order instead of random order. This helps the model to learn bidirectional relationships and therefore better handles dependencies and relations between words. XLNet was trained with over 130 GB of textual data and 512 TPU chips running for 2.5 days, both of which are much larger than BERT.

In our task we trained fine tuned this model for a segment of data using following parameters Learning rate=  $5e-5$  , Training epochs = 1 and obtained metrics as

LRAP = 0.99

LRL= 0.018.

We can see difference between three models as xlnet is more resource intensive and takes longer time to train as compared to the other two. There is not much difference in performance however Roberta performs best.

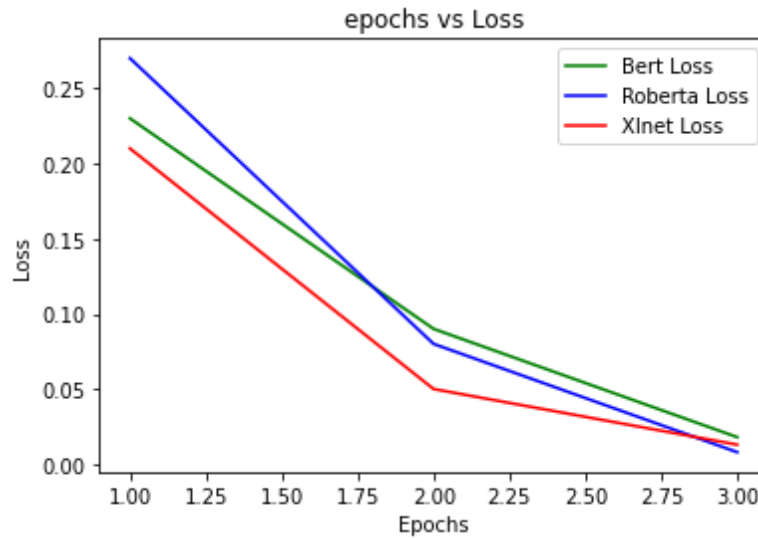


Figure showing convergence of loss of all the three models

## Web Application

Web application is developed to provide an interface for the users to interact with the developed application.

Web application provides the following functionalities:

- To obtain tags/labels for the question of interest.
- To provide questions belonging to a paradigm from different coding platforms at one place.
- Login
- Register

Web application is developed using the following technologies:

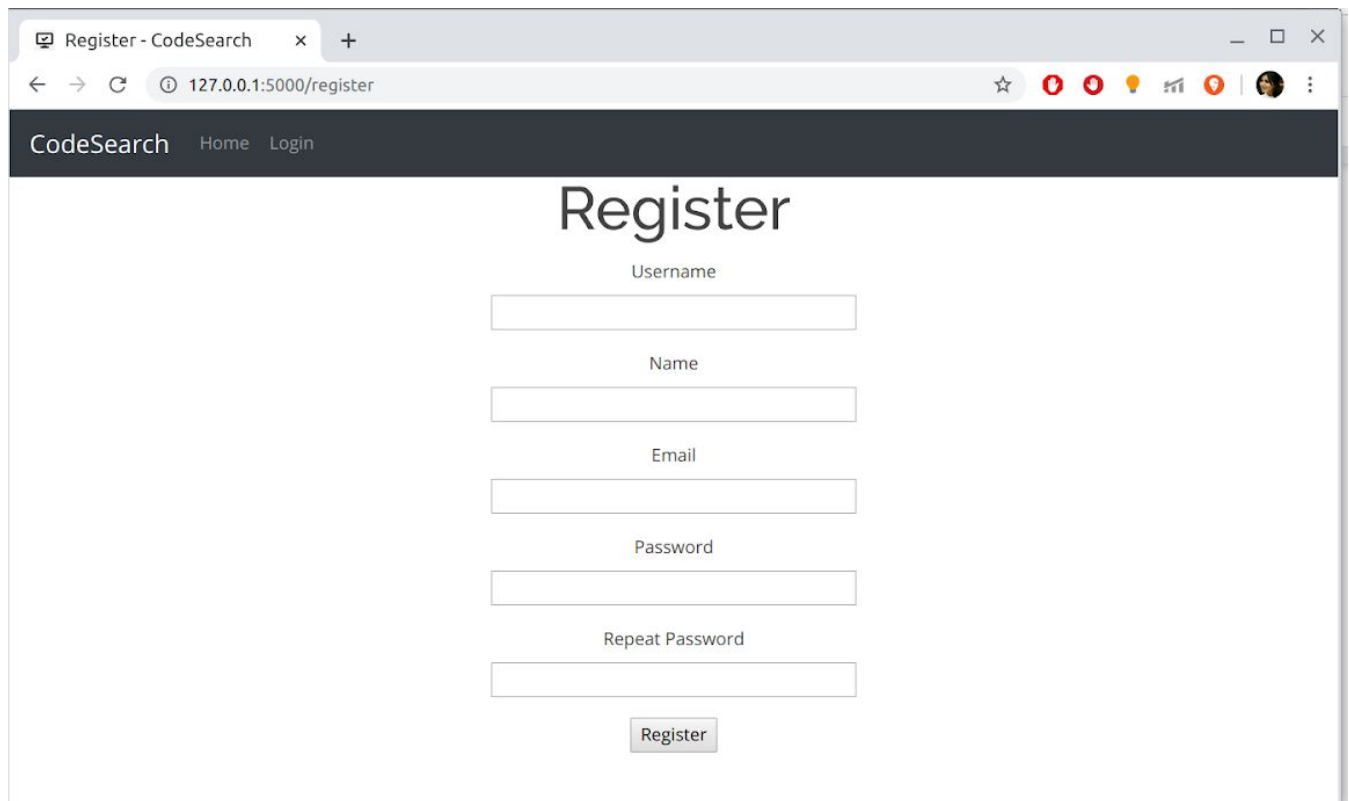
UI - html, css, javascript, bootstrap

Server - Flask framework.

## Interfaces in web application:

### 1. Register:

User has to register to use the application by providing details like username, password, name, email etc.,



The screenshot shows a web browser window with a single tab titled 'Register - CodeSearch'. The address bar displays '127.0.0.1:5000/register'. The browser's toolbar includes a star icon for bookmarks, several red circular icons, a lightbulb icon, a signal strength icon, a location pin icon, and a user profile icon. The website's header is dark grey with the text 'CodeSearch' and navigation links 'Home' and 'Login'. The main content area is white and features a large 'Register' heading. Below the heading are five input fields, each with a label above it: 'Username', 'Name', 'Email', 'Password', and 'Repeat Password'. At the bottom of the form is a 'Register' button.

Register - CodeSearch x +

127.0.0.1:5000/register

CodeSearch Home Login

# Register

Username

Name

Email

Password

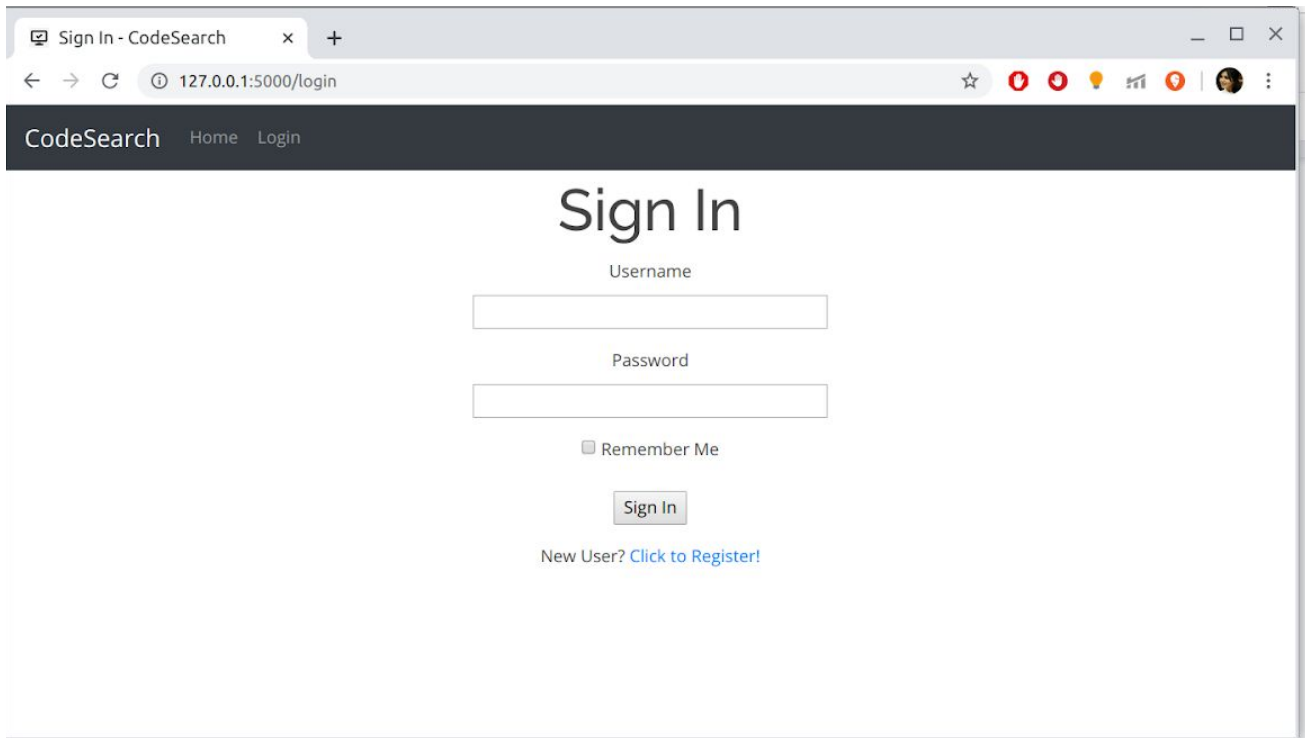
Repeat Password

Register



## 2. Login:

After registration, user can Login by providing the correct username and password. User will be able to access 'Home' only after login.



The screenshot shows a web browser window with the title 'Sign In - CodeSearch'. The address bar displays '127.0.0.1:5000/login'. The browser's navigation bar includes 'CodeSearch', 'Home', and 'Login' links. The main content area features a 'Sign In' heading, followed by 'Username' and 'Password' labels above their respective input fields. Below these fields is a 'Remember Me' checkbox and a 'Sign In' button. At the bottom, there is a link for 'New User? Click to Register!'.

Sign In - CodeSearch

127.0.0.1:5000/login

CodeSearch Home Login

# Sign In

Username

Password

☐ Remember Me

Sign In

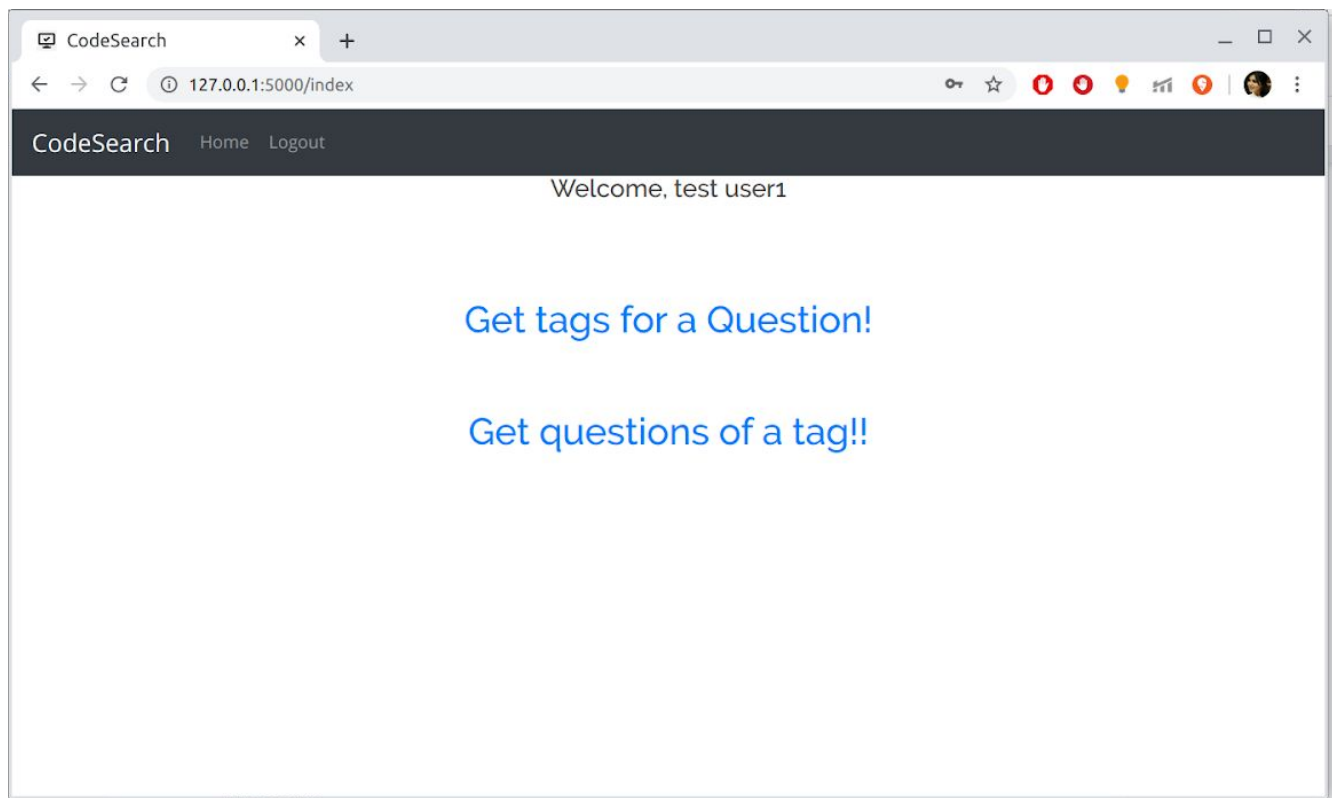
New User? [Click to Register!](#)

### 3. Home:

User will be directed to his/her home page with the name displayed after successful login. In home, user will be able to see the two applications provided

1. Get tags for a Question.
2. Get questions of a tag.

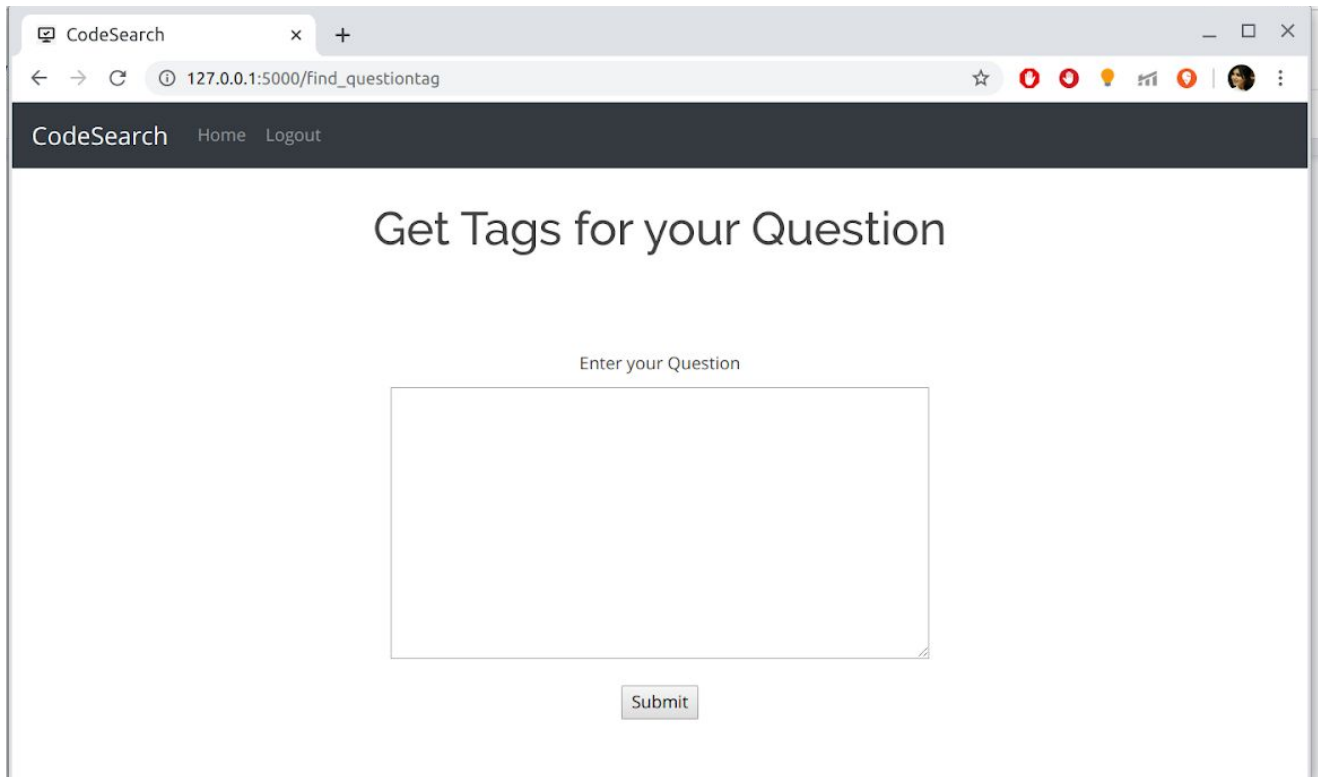
The logout option is visible on the top menu after login to enable user to logout.



## 4. Get labels for a Question:

### 4.1 Enter Question:

If the option 'Get tags for a Question' is selected on the home page, user will be redirected to a page to enter the question.

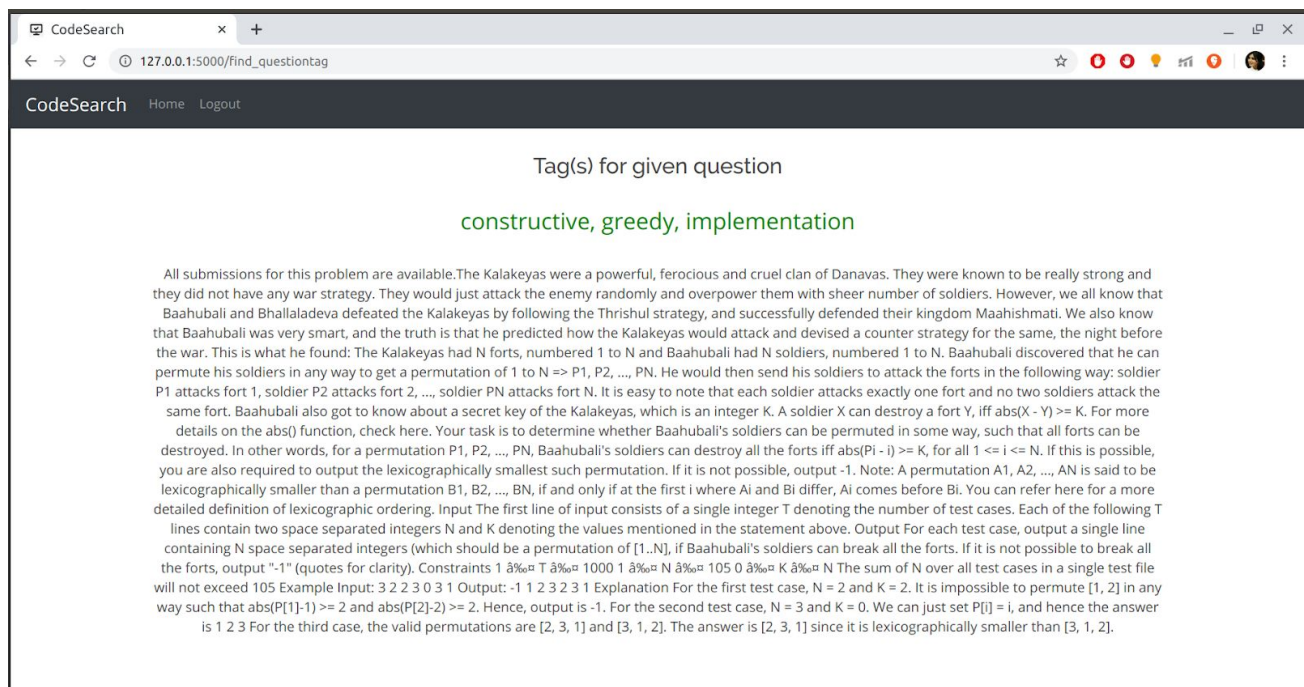


The screenshot shows a web browser window with the title 'CodeSearch'. The address bar displays '127.0.0.1:5000/find\_questiontag'. The browser's navigation bar includes a star icon, several extension icons, and a user profile icon. The page's header is dark grey with the text 'CodeSearch' and links for 'Home' and 'Logout'. The main content area has a large heading 'Get Tags for your Question'. Below this heading is a text input field with the placeholder text 'Enter your Question'. At the bottom of the input field is a small 'Submit' button.

## 4.2 Display tag for the question:

After entering the question, the model is run and the user is shown the label(s) of the entered question on screen. The question entered is also shown below the label(s).

To make use of the machine learning model in the web application, the trained model is packaged to pickle using the joblib library. In the web application, the model is fetched back using the joblib so that we can make predictions for the questions entered and display them on screen.

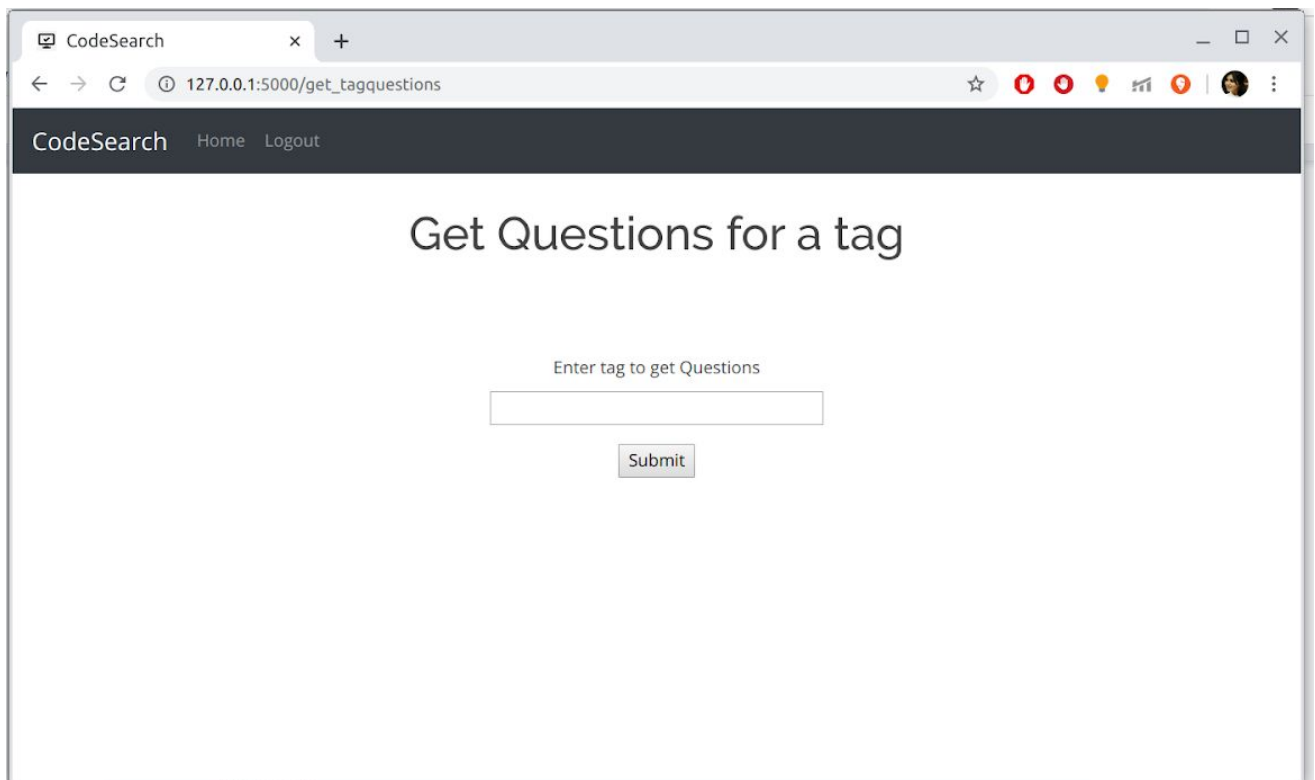


The image shows the labels for the given questions in green as 'constructive, greedy, implementation'. The question entered is also shown below.

## 5. Get questions of a tag:

### 5.1 Enter tag:

If the option 'Get questions of a tag' is selected on the home page, user will be redirected to a page to enter the tag.



The screenshot shows a web browser window with the title 'CodeSearch'. The address bar displays the URL '127.0.0.1:5000/get\_tagquestions'. The browser's toolbar includes navigation buttons (back, forward, refresh), a star icon for bookmarks, and several extension icons. The website's header is dark grey with the text 'CodeSearch' and links for 'Home' and 'Logout'. The main content area has a large heading 'Get Questions for a tag'. Below this heading is a prompt 'Enter tag to get Questions' followed by a text input field and a 'Submit' button.

CodeSearch Home Logout

# Get Questions for a tag

Enter tag to get Questions

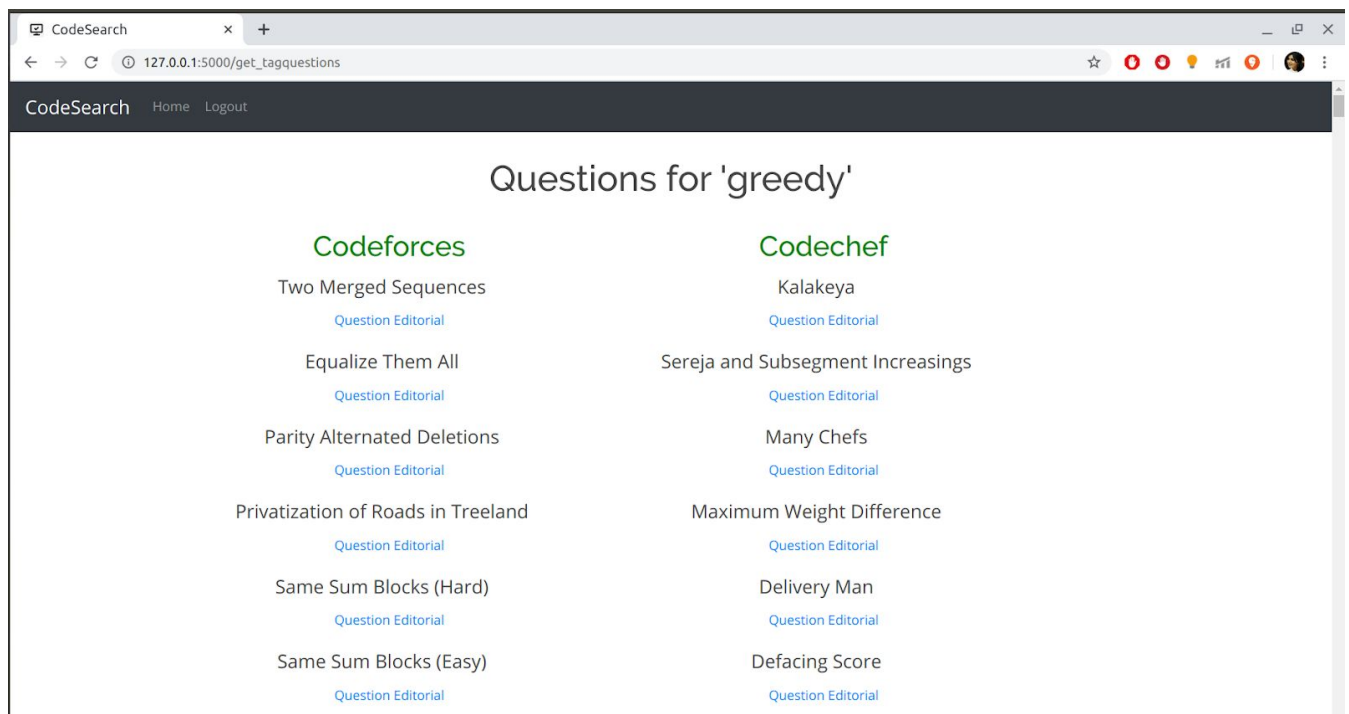
Submit

## 5.2 Display all questions:

After entering the tag, all questions belonging to the tag are fetched from the database and displayed.

Each question has a name which is displayed and it has two options: Question, Editorial which are links to the problem statement and solution of the question respectively. On clicking the links the webpage having the question or editorial is opened in the corresponding websites.

Questions from the coding platforms codechef, codeforces are stored in the database as we have scraped data from these websites previously.



This image shows the questions belonging to the paradigm 'greedy' from codeforces and codechef, along with links to question and editorial for each question.

**Github link:** <https://github.com/sai-sukrutha/Coding-Questions-Classifier>

**References:**

1. <https://towardsdatascience.com/multi-label-text-classification-with-sci-kit-learn-30714b7819c5>
2. <https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18>
3. <https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8>
4. <https://towardsdatascience.com/bert-technology-introduced-in-3-minutes-2c2f9968268c>
5. <https://ai.facebook.com/blog/roberta-an-optimized-method-for-pretraining-self-supervised-nlp-systems>