

**A Project Report  
on**

**STACKLITE: STACK OVERFLOW TAG PREDICTION**

**Submitted in partial fulfillment of the requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY  
in  
INFORMATION TECHNOLOGY  
by**

**Ch. Ramya (16WH1A1211)**

**G. Shrenika (16WH1A1217)**

**G.V. Samhitha (16WH1A1218)**

**Ch. Urmila (16WH1A1260)**

**Under the esteemed guidance of**

**Mr. A. Rajashekar Reddy**

**Assistant Professor**



**Department of Information Technology  
BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN**

**(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and**

**Affiliated to JNTUH, Hyderabad)**

**Bachupally, Hyderabad – 500090**

**April 2020**

## DECLARATION

We hereby declare that the work presented in this project entitled “**STACKLITE: STACK OVERFLOW TAG PREDICTION**” submitted towards completion of the major project in IV year of B.Tech IT at “BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN”, Hyderabad is an authentic record of our original work carried out under the esteem guidance of **Mr. A. Rajashekar Reddy, Assistant Professor**, IT department.

**Ch. Ramya**  
**(16WH1A1211)**

**G. Shrenika**  
**(16WH1A1217)**

**G.V. Samhitha**  
**(16WH1A1218)**

**Ch. Urmila**  
**(16WH1A1260)**

# **BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN**

(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and Affiliated to JNTUH, Hyderabad)

**Bachupally, Hyderabad – 500090**

## **Department of Information Technology**



## **Certificate**

This is to certify that the Project report on **“Stacklite: Stack Overflow Tag Prediction”** is a bonafide work carried out by **Ch. Ramya (16WH1A1211), G. Shrenika (16WH1A1217), G.V. Samhitha (16WH1A1218), Ch. Urmila (16WH1A1260)** in the partial fulfillment for the award of B.Tech degree in **Information Technology, BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

### **Internal Guide**

**Mr. A. Rajashekar Reddy**

**Assistant Professor**

**Department of IT**

### **Head of the Department**

**Dr. Aruna Rao S L**

**Professor and HOD**

**Department of IT**

### **External Examiner**

## **Acknowledgements**

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal**, BVRIT Hyderabad, for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. Aruna Rao S L, Head**, Dept. of IT, BVRIT Hyderabad for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr. A. Rajashekar Reddy, Assistant Professor, Department of IT**, BVRIT Hyderabad for his constant guidance, encouragement and moral support throughout the project.

Finally, We would also like to thank our Project Coordinator, all the faculty and staff of IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

**Ch. Ramya**  
**(16WH1A1211)**

**G. Shrenika**  
**(16WH1A1217)**

**G.V. Samhitha**  
**(16WH1A1218)**

**Ch. Urmila**  
**(16WH1A1260)**

# **ABSTRACT**

Stack Overflow forum is a widely used platform for people to interact on topics related to Computer Programming languages. With more than three lakh users and ten lakh questions, Stack Overflow is emerging as the biggest question and answers forum for programmers. The questions on Stack Overflow cover a wide range of topics and are categorized using appropriate tags. Currently the tags are entered manually by users depending on the questions they have posted. Since there are a huge number of tags, it is a difficult process to search the correct tag. It may be useful to have an auto-tagging system that suggests tags to users depending on the content of the question.

Large-Scale datasets are available that can be mined and pre-processed and can be used to know users query regarding a particular topic. We propose a system that we will be provided with a bunch of questions. A question in Stack Overflow contains three segments Title, Description and Tags. By using the text in the title and description we suggest the tags related to the subject of the question automatically.

## LIST OF FIGURES

Figure No	Figure Name	Page No
4.1.1	Dataset	12
4.1.2	System Architecture	12
4.3.1	Top 20 tags of complete data	14
4.3.2	Graph on complete data	14
4.3.3	Top 20 tags of 100000 records	15
4.3.4	Graph on 100000 records	15
4.4.1	Splitting Data	16
5.1.1	Data Pre-processing	18
5.1.2	Data Pre-processed	18
5.2.1.1	Binary Relevance	20
5.2.1.2	Classifier Chain	22
5.2.1.3	Label Powerset	24
6.3	Home Page	46
6.4	Example Question	47
6.5	Suggested Tags	47

## LIST OF ABBREVIATIONS

Term/Abbreviation	Definition
TF-IDF	Term Frequency–Inverse Document Frequency
SGDC	Stochastic Gradient Descent Classifier
KNN	K-Nearest Neighbours
LC	Local Cardinality
NLTK	Natural Language Tool Kit

## **TABLE OF CONTENTS**

<b>CHAPTER NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
	<b>ABSTRACT</b>	V
	<b>LIST OF FIGURES</b>	Vi
	<b>LIST OF ABBREVIATIONS</b>	vi
<b>1.</b>	<b>INTRODUCTION</b>	1
	1.1 OBJECTIVE	2
	1.2 PROBLEM IN EXISTING SYSTEM	2
	1.3 SOLUTION	2
	1.4 FEATURES	3
<b>2.</b>	<b>LITERATURE SURVEY</b>	4
	2.1 APPROACH	4
	2.2 INFORMATION GATHERING	5
<b>3.</b>	<b>REQUIREMENT SPECIFICATION</b>	6
	3.1 SOFTWARE REQUIREMENT	6
	3.2 HARDWARE REQUIREMENT	6
	3.2 COMPONENTS	7
<b>4.</b>	<b>DESIGN OF THE SYSTEM</b>	11
	4.1 DOWNLOADING DUMP	11
	4.2 PARSING	13
	4.3 EXTRACTING	13
	4.4 TRAIN DIFFERENT CLASSIFIERS FOR DIFFERENT PARTS OF THE POST	16
<b>5.</b>	<b>IMPLEMENTATION</b>	17
	5.1 DATA PRE-PROCESSING	17
	5.2 MODEL BUILDING	19
	5.3 TECHNOLOGIES	29
	5.4 CODE	32
<b>6.</b>	<b>TESTING</b>	45
<b>7.</b>	<b>CONCLUSION AND FUTURE SCOPE</b>	48
	7.1 CONCLUSION	48
	7.2 FUTURE SCOPE	48
<b>8.</b>	<b>REFERENCES</b>	49





## 1. INTRODUCTION

A stack overflow is an undesirable condition in which a particular computer program tries to use more memory space than the call stack has available. In programming, the call stack is a buffer that stores requests that need to be handled. With the increasing amount of information available on the learning based websites it is very essential to maintain the adequacy. For that purpose we have taken the already existing data into account and with the help of that data we will make a system that is useful with the quality of the information available. For example if we have a question answering website like stack overflow we analyse that huge data is available in form of question and answers posted and answered by user.

To enhance users experience for a website it is required that the information that the user is seeking for must be available the way user expected. If the information related to one particular topic is available at one place and with one click then it enhances the users experience with the particular website. And this fact introduces the concept of tagging.

Importance of the tagging is very much when it comes to grouping the common topics together and it should be done with the system that automatically infer the tag based on the content that the user input in order to get the information that is relevant to him. So we require a system for the purpose of tagging because manually tagging is a burden to the user and it reduces the overall experience of the website to the user. Modelling tag creation accurately could allow a system to predict the hash tag that should be used by the user, given the contextual clues available.

## **1.1 Objective**

The Objective is to predict the tags depending upon the questions provided. Depending upon the question given by the user, the tag or tags which fall under the right category will be displayed.

## **1.2 Problem In Existing System**

Forums like stack overflow rely on the tags of questions to match them to users who can provide answers. However, new users in Stack Overflow or novice developers may not tag their posts correctly. For example, if a user provides a tag to his question then the other users who have answered many questions related to that topic will get notified. So, if a user does not provide a correct tag it would lead to the confusion and also this leads to posts being down voted and flagged by moderators even though the question may be relevant and adds value to the community. In some cases, stack overflow questions that are related to programming languages may lack a programming language tag. This could create confusion among developers who may be new to a programming language and might not be familiar with all of its popular libraries. The problem of missing language tags could be addressed if posts are automatically tagged with their associated programming languages.

## **1.3 Solution**

In Stack overflow people can provide the tags related to the question on their own. This is extremely business critical. The more accurately stack overflow can predict these tags the better it can create an Ecosystem to send the right question to the right set of people. So, developing an auto tagging system would increase the users experience by predicting the correct tag to the given question. In our work we are using existing data to extract relevant information from the data as the data is vast. For this method text classification, the process of classifying text documents into different classes on the basis of their topic is used.

Text classification is a common application of machine learning and natural language processing. Text classification is a multi class and multi-label classification problem, means that there are multiple classes to choose from and when labeling the input and each input might belong to more than one class available. There are two approaches available for implementing multi-label classification.

The first one is using one-vs-all, in this approach we decompose the classification problem into  $k$  independent binary classification problem, an alternative approach for multi-label classification is training a separate classifier for each of the  $k$  possible output labels. This is one of the approach from adaptive algorithms. Examples of adaptive algorithms include boosting and random forests. In our classification system, we use the one-vs-all approach in which we will take use of Support Vector Machines with the linear kernel.

## **1.4 Features**

Importance of the tagging is very much when it comes to grouping the common topics together and it should be done with the system that automatically infer the tag based on the content that the user input in order to get the information that is relevant to him. So we require a system for the purpose of tagging because manually tagging is a burden to the user and it reduces the overall experience of the website to the user. Modeling tag creation accurately could allow a system to predict the hash tag that should be used by the user, given the contextual clues available. There are various benefits to this type of tag prediction. For example, users could be introduced to tags that are of interest to them. We built a tag prediction system for the Stack Overflow dataset that is anticipated to solve these sorts of problems.

## 2. LITERATURE SURVEY

### 2.1 Approach

By using the existing data to extract relevant information from the data as the data is vast, text classification method is used which is the process of classifying text documents into different classes on the basis of their topic is used. Text classification is a common application of machine learning. Text classification is a multi class and multi-label classification problem, means that there are multiple classes to choose from and when labeling the input and each input might belong to more than one class available. There are two approaches available for implementing multi-label classification. The First one is using one vs all, in this approach we decompose the classification problem into k independent binary classification problem, an alternative approach for multi-label classification is training a separate classifier for each of the k possible output labels. Here, logistic algorithm is used.

### 2.2 Information Gathering

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's built and run by you as part of the stack exchange network of question and answer sites. This is all about getting answers. It's not a discussion forum. Focus on questions about an actual problem you have faced. Include details about what you have tried and exactly what you are trying to do. If the information related to one particular topic is available at one place and with one click then it enhances the users experience with the particular website. A question contains three segments Title, Description and Tags. By using the text in the title and description we should suggest the tags related to the subject of the question automatically. Multi-class classification means a classification task with more than two classes; each label are mutually exclusive. The classification makes the assumption that each sample is assigned to one and only one label. On the other hand, Multi-label classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as Tim Horton are often categorized as both

bakery and coffee shop. Multi-label text classification has many real world applications such as categorizing businesses on Yelp or classifying movies into one or more genre.

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous. Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. Sometimes logistic regressions are difficult to interpret; the Intellectual Statistics tool easily allows you to conduct the analysis, then in plain English interprets the output.

### 3. REQUIREMENT SPECIFICATION

#### 3.1 Software Requirement

Software used for predicting the tags in Stack overflow is by using python platform. Python offers all the skill sets that are required for a machine learning project. Python consists of large no. of tools and packages. For multi label and multiclass classification we used sklearn library in python. We also use matplotlib library for generating graphs and visualizations, nltk library for tokenizations and to remove stop words, and also numpy and pandas libraries to build data frames.

#### 3.2 Hardware Requirement

The hardware requirement specifies each interface of the software elements and the hardware elements of the system. These hardware requirements include configuration characteristics.

##### Recommended System Requirements

- **Processors :** Intel Core i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM Intel Xeon processor E5-2698 v3 at 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64 GB of DRAM Intel Xeon Phi processor 7210 at 1.30 GHz (1 socket, 64 cores, 4 threads per core), 32 GB of DRAM, 16 GB of MCDRAM (flat mode enabled)
- **Disk space :** 2 to 3 GB
- **Operating systems :** Windows 10, macOS and Linux

##### Minimum System Requirements

- **Processors :** Intel Atom processor or Intel Core™ i3 processor
- **Disk space :** 1 GB
- **Operating systems :** Windows 7 or later, macOS, and Linux
- **Python\* versions :** 2.7.X, 3.6.X

### 3.3 Components

#### Jupyter notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text.

Jupyter Notebook is maintained by the people at Project Jupyter.

#### Python 3.6

- Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.
- The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.
- The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

#### Sklearn

Scikit-learn is an open source Python library that implements a range of machine learning, pre-processing, cross-validation and visualization algorithms using a unified interface.

##### Important features of scikit-learn:

- Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Accessible to everybody and reusable in various contexts.
- Built on the top of NumPy, SciPy, and matplotlib.

## NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with the arrays. It is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

## PANDAS

- It's an open source data analysis library for providing easy-to-use data structures and data analysis tools. Pandas is Python Data Analysis Library, pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools.
- Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data merging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.
- Primary object types:
- **DataFrame:** rows and columns (like a spreadsheet)
- **Series:** a single column



## Matplotlib

- The matplotlib.pyplot is a plotting library used for two dimensional graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits.
- There are several toolkits which are available that extend python matplotlib functionality. Some of them are separate downloads, others can be shipped with the matplotlib source code but have external dependencies.
- There are various plots which can be created using python matplotlib. Some of them are bar graph, histogram, scatter plot, area plot, pie plot
- Using matplotlib the visualization of data can be done in a better way. Using this library publication quality plots can be developed with just a few lines of code and also create interactive figures that can zoom, pan, update.
- Customization by taking full control of line styles, font properties, axes properties is possible and also export and embed to a number of file formats and interactive environments. Different tailored functionality provided by third party packages is possible.

## Snowball Stemming

In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root.

## FLASK

Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more frequently than the core Flask program.

To install flask firstly we need to do some certain steps:

- Make a Virtual Environment
- Pip install virtualenv
- Connect our project with our environment
- mkdir myproject
- Set project directory
- cd myproject
- Install flask

### Pip install flask

After installing the flask then we need to install certain libraries.

## SGDC Classifier

SGD Classifier is a Linear classifiers with SGD training. Stochastic Gradient Descent is a solver. It is a simple and efficient approach for discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

## 4. DESIGN OF THE SYSTEM

### 4.1 Downloading Dump

The dataset is downloaded which has been already provided by stack overflow. This dataset consists of two files train and test.

- Train.csv contains 4 columns they are Id, Title, Body and Tags.
- Test.csv contains the same columns but without the Tags, which we have to predict.
- Size of Train.csv is 6.75GB.
- Size of Test.csv: 2GB.
- Number of rows in Train.csv is equal to 6034195

The questions are randomized and contains a mix of verbose text sites as well as sites related to math and programming. The number of questions from each site may vary, and no filtering has been performed on the questions.

The Id column consists of the unique identifier for each question. The Title column consists of the title of the question. The whole description of the question is provided in the body column and the tags column consists of the tags associated with the question in a space separated format.

```

Id                                     Title \
0 1 How to check if an uploaded file is an image w...
1 2 How can I prevent firefox from closing when I ...
2 3 R Error Invalid type (list) for variable
3 4 How do I replace special characters in a URL?
4 5 How to modify whois contact details?

Body \
0 <p>I'd like to check if an uploaded file is an...
1 <p>In my favorite editor (vim), I regularly us...
2 <p>I am import matlab file and construct a dat...
3 <p>This is probably very simple, but I simply ...
4 <pre><code>function modify(.....)\n{\n $mco...

Tags
0 php image-processing file-upload upload mime-t...
1 firefox
2 r matlab machine-learning
3 c# url encoding
4 php api file-get-contents
Index(['Id', 'Title', 'Body', 'Tags'], dtype='object')
0:01:54.503500

```

Figure 4.1.1: Dataset

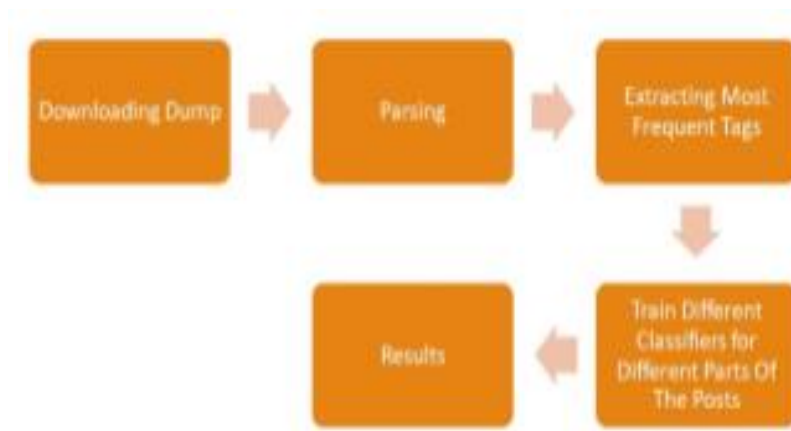


Figure 4.1.2 System Architecture

## 4.2 Parsing

The data pre-processing is done here where all the stop words from the title and the body columns are removed. The code blocks present in the dataset are also removed. And the body column and the title columns are merged into one column which is named as question. Now this question column consists of the data present in both body and title columns. This pre-processed data consists of two columns questions and tags.

## 4.3 Extracting

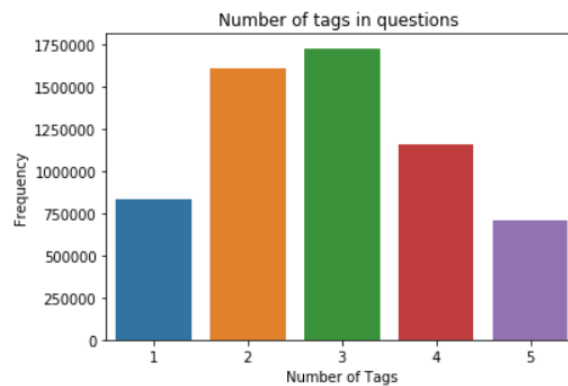
In our system we needed dataset to perform our experiment for that we downloaded our data set from the Stack overflow website. This dataset contains approximately 6 million questions posted by the user on the website which belongs to approx 50,000 unique tags present on the Stack Overflow website. Each post is the combination of:

- Unique identifier
- The title of the question
- The body part of the question containing the content of the question of users query.

The tag also might be associated with the question. Memory could be the big problem with such huge dataset because of which computing gets very hard and it is practical that developing a model for such a large dataset is not computationally feasible for a system. So we need to subsample the data, maintaining it to feasible for at most 1000 most frequent tags that are seen entirely in the data. We also need to ensure that our sub sampled or resultant data contains at least 1000 training examples for each label. With this it will be sure that we have enough diverse training data for us to measure and learn statistics for the relevant tags.

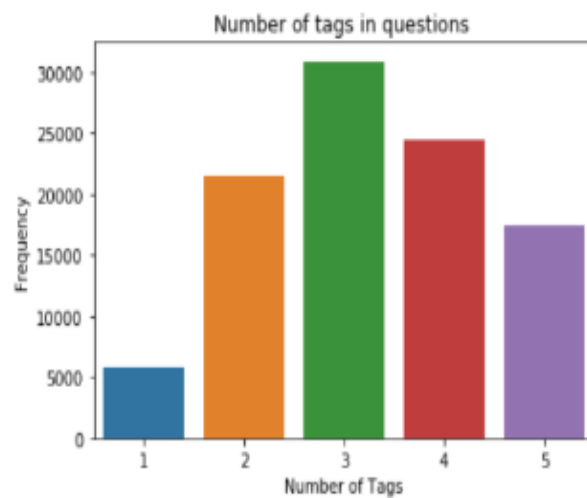
**Analysis on complete data:**

	Tags	Counts
0	net	547537
1	java	439651
2	android	422602
3	php	400694
4	jquery	391867
5	javascript	379876
6	asp	310249
7	windows	294365
8	sql	289843
9	ruby	245569
10	iphone	202919
11	python	201653
12	server	194955
13	html	178496
14	mysql	177797
15	rails	173192
16	on	172750
17	ios	140370
18	linux	139123
19	css	138342

**Figure 4.3.1 Top 20 tags of complete data****Figure 4.3.2 Graph on complete data**

**Analysis on 100000 records:**

	Tags	Counts
0	java	103681
1	android	12360
2	swing	5992
3	spring	5333
4	eclipse	4406
5	hibernate	3183
6	xml	2658
7	web	2361
8	multithreading	2198
9	ee	2110
10	jsp	1936
11	servlets	1814
12	jsf	1799
13	file	1789
14	google	1788
15	maven	1726

**Figure 4.3.3 Top 20 tags of 100000 records****Figure 4.3.4 Graph on 100000 records**

## 4.4 Train Different Classifiers for Different Parts of the Post

The splitting of the data is done. The data is split into test dataset and train data set.

```
[16]: X_train, X_test, y_train, y_test = train_test_split(preprocessed_df, yx_multilabel, test_size = 0.2, random_state = 42)
      print("Number of data points in training data :", X_train.shape[0])
      print("Number of data points in test data :", X_test.shape[0])
```

```
Number of data points in training data : 112943
Number of data points in test data : 28236
```

```
[17]: vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, tokenizer = lambda x: x.split(), ngram_range=(1,3))
      X_train_multilabel = vectorizer.fit_transform(X_train['question'])
      X_test_multilabel = vectorizer.transform(X_test['question'])
```

```
[18]: print("Training data shape X : ", X_train_multilabel.shape, "Y :", y_train.shape)
      print("Test data shape X : ", X_test_multilabel.shape, "Y:", y_test.shape)
```

```
Training data shape X : (112943, 69784) Y : (112943, 10)
Test data shape X : (28236, 69784) Y: (28236, 10)
```

---

**Figure 4.4.1 Splitting the Data**

The number of data points present in the training dataset are 112943. The number of data points present in the test dataset are 28236. After splitting the data logistic regression algorithm is applied to get the accuracy.



## 5. IMPLEMENTATION

### 5.1 Data Pre-processing

The data has been pre-processed here where all the stop words from the title and the body columns are removed. The code blocks present in the dataset are also removed. And the body column and the title columns are merged into one column which is named as question. Now this question column consists of the data present in both body and title columns. This pre-processed data consists of two columns questions and tags. The following is the code implemented for data pre-processing.

```
data["Title"] + data["Body"] = data["question"]
```

Removing stopwords => using Stopwords method from nltk library.

Performing stemming => using SnowballStemmer method from nltk library.

Eg:- “chocolates”, “chocolatey”, “choco” => “chocolate”

**Example :-**

**Data before processing :-**

Data["Title"] => "How to check if an uploaded file is an image without mime type?"

Data["Body"] => "I'd like to check if an uploaded file is an image file (e.g png, jpg, jpeg, gif, bmp) or another file. The problem is that I'm using Uploadify to upload the files, which changes the mime type and gives a 'text/octal' or something as the mime type, no matter which file type you upload. Is there a way to check if the uploaded file is an image apart from checking the file extension using PHP?"

Data["Tags"] => "php image-processing file-upload upload mime-types"

**Data after processing :-**

Data["question"] => "check upload file image mime type check upload file image file png jpg jpeg gif bmp another file problem uploadify upload file change mime type give text octal something mime type file type upload check upload file image check file php".

```
[10]: start = datetime.now()
      qus_list=[]
      qus_with_code = 0
      len_before_preprocessing = 0
      len_after_preprocessing = 0
      for index,row in data.iterrows():
          title, body, tags = row["Title"], row["Body"], row["Tags"]
          if '<code>' in body:
              qus_with_code+=1
          len_before_preprocessing+=len(title) + len(body)
          body=re.sub('<code>(.*?)</code>', '', body, flags=re.MULTILINE|re.DOTALL)
          body = re.sub('<.*?>', ' ', str(body.encode('utf-8')))
          title=title.encode('utf-8')
          question=str(title)+" "+str(body)
          question=re.sub(r'[^A-Za-z]+', ' ', question)
          words=word_tokenize(str(question.lower()))
          question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_words and (len(j)!=1 or j=='c'))
          qus_list.append(question)
          len_after_preprocessing += len(question)
      data["question"] = qus_list
      avg_len_before_preprocessing=(len_before_preprocessing*1.0)/data.shape[0]
      avg_len_after_preprocessing=(len_after_preprocessing*1.0)/data.shape[0]
      print("Avg. length of questions(Title+Body) before preprocessing: ", avg_len_before_preprocessing)
      print("Avg. length of questions(Title+Body) after preprocessing: ", avg_len_after_preprocessing)
      print("% of questions containing code: ", (qus_with_code*100.0)/data.shape[0])
      end = datetime.now()
      print(end-start)
```

Figure 5.1.1: Data Pre-processing

```
Avg. length of questions(Title+Body) before preprocessing: 1052.36693134248
Avg. length of questions(Title+Body) after preprocessing: 237.88458623449662
% of questions containing code: 69.6569603127944
0:05:01.801639
```

```
[11]: preprocessed_df = data[["question", "Tags"]]
      print("Shape of preprocessed data :", preprocessed_df.shape)
```

```
Shape of preprocessed data : (141179, 2)
```

Figure 5.1.2 Pre-processed Data

## 5.2 Model Buliding

### 5.2.1 Problem Transformation Techniques

Problem transformation methods map the multi-label learning task into one or more single-label learning tasks. When the problem is mapped into more than one single-label problem, the multi-label problem is decomposed into several independent binary classification problems, one for each label which participates in the multi-label problem.

In this method, we will try to transform our multi-label problem into single-label problem.

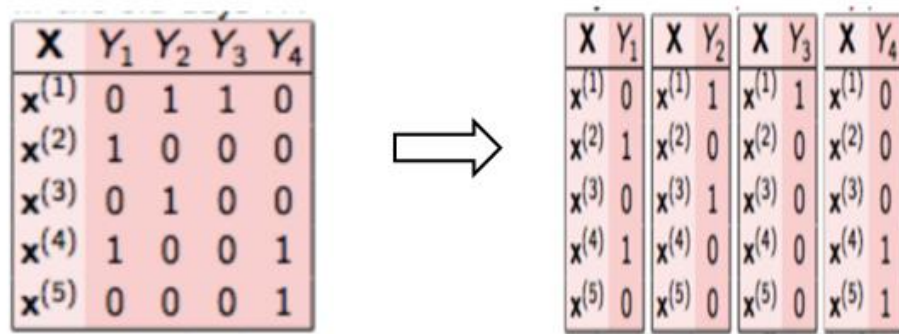
This method can be carried out in three different ways as:

1. Binary Relevance
2. Classifier Chains
3. Label Powerset

#### 1.Binary Relevance

Multi-label text classification has become progressively more important in recent years, where each document can be given multiple labels concurrently. Multi-label text classification is a main challenging task because of the large space of all potential label sets, which is exponential to the number of candidate labels. Among the disadvantages of the earlier multi-label classification methods is that they typically do not scale up with the number of specific labels and the number of training examples. A large amount of computational time for classification is required for a large amount of text documents with high dimensionality, especially, the Arabic language which has a very complex morphology and rich in nature. This generator produces synthetic Machine Learning data allowing the user to select the desired values for some important characteristics, like the number of labels or the level of label dependency.

Binary relevance method, amounts to independently training one binary classifier for each label. Given an unseen sample, the combined model then predicts all labels for this sample for which the respective classifiers predict a positive result. Although this method of dividing the task into multiple binary tasks may resemble superficially the one-vs.-all (OvA) and one-vs.-rest (OvR) methods for multiclass classification, it is essentially different from both, because a single classifier under binary relevance deals with a single label, without any regard to other labels whatsoever.



**Figure 5.2.1.1 Binary Relevance**

**Accuracy :-**

33%

**Drawback :-**

The main drawback of Binary Relevance is that it does not take into account any label dependency and may fail to predict some label combinations if such dependence is present.

## 2. Classifier Chain

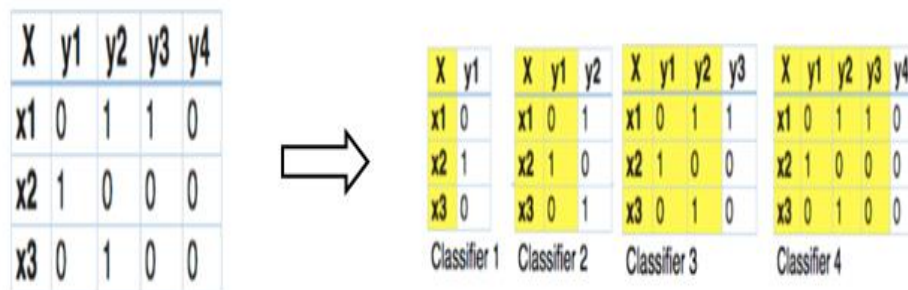
A set of multi-class classifiers can be used to create a multi-label ensemble classifier. For a given example, each classifier outputs a single class (corresponding to a single label in the multi-label problem). These predictions are then combined by an ensemble method, usually a voting scheme where every class that receives a requisite percentage of votes from individual classifiers (often referred to as the discrimination threshold) is predicted as a present label in the multi-label output. However, more complex ensemble methods exist, such as committee machines.

The Classifier Chain model involves  $|L|$  binary classifiers as in Binary Relevance. Classifiers are linked along a chain where each classifier deals with the binary relevance problem associated with label  $l_j \in L$ . The feature space of each link in the chain is extended with the 0/1 label associations of all previous links. This chaining method passes label information between classifiers, allowing Classifier Chain to take into account label correlations and thus overcoming the label independence problem of Binary Relevance. However, Classifier Chain still retains advantages of Binary Relevance including low memory and runtime complexity. Although an average of  $|L|/2$  features is added to each instance, because  $|L|$  is invariably limited in practice, this has negligible consequences on complexity. Classifier Chain's complexity will be worse than Binary relevance's whenever  $|L| > |X|$  holds.

Also, although the chaining procedure implies that Classifier Chain cannot be parallelized, it can be serialized and therefore still only requires a single binary problem in memory at any point in time a clear advantage over other methods.

The order of the chain itself clearly has an effect on accuracy. Although there exist several possible heuristics for selecting a chain order for Classifier Chain, we instead solve the issue by using an ensemble framework with a different random chain ordering for each iteration. In the next section, we present this framework.

In Ensemble of Classifier Chains (ECC) several CC classifiers can be trained with random order of chains on a random subset of data set. Labels of a new instance are predicted by each classifier separately. After that, the total number of predictions or "votes" is counted for each label. The label is accepted if it was predicted by a percentage of classifiers that is bigger than some threshold value.



**Figure 5.2.1.2 Classifier chain**

**Accuracy :-**

42%

**Drawback :-**

In this way, Classifier Model-based methods directly take into account label correlations. A disadvantage of these methods, however, is their worst-case time complexity. The consensus view in the literature is that it is crucial to take into account label correlations during the classification process

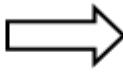
### 3. Label Powerset

Label Powerset method transforms the multi-label problem into one single-label multi-class classification problem, where the possible values for the transformed class attribute is the set of distinct unique subsets of labels present in the original training data. illustrates this idea, where the attribute space was omitted, since the transformation process only modifies the label space. The first table shows a multi-label set of instances in the original form, while the second table shows the same set of instances transformed into the Label Power Set format, where notation  $y_{i,j,\dots,k}$  means that the respective instance is labeled with the conjunction  $y_i \wedge y_j \wedge \dots \wedge y_k$ . Observe that in the first table the set of labels  $L$  is  $\{y_1, y_2, y_3, y_4\}$  while after transformation, the set of multi-class labels is  $\{y_{2,3}, y_{1,3,4}, y_4, y_{2,3}\}$ .

Using this method, learning from multi-label examples corresponds to finding a mapping from the space of features to the space of label sets, i.e., the power set of all labels, resulting in up to  $2^{|L|}$  transformed labels. Thus, although Label Powerset takes into account label dependency, when a large or even moderate number of labels are considered, the task of multi-class learning the label power sets would become rather challenging due to the tremendous (exponential) number of possible label sets. This is a drawback of the LP method. Another rather important issue is the class imbalance problem, which occurs when there are classes in the training set represented by very few examples. As the number of possible label sets increases with  $|L|$ , this is likely to occur.

The most natural approach is the label power set method which generates a new class for every combination of labels and then solves the problem using multiclass classification approaches. The main drawback of this approach is the exponential growth in the number of classes, leading to several generated classes having very few labeled instances leading to overfitting. BR does not consider relationship between labels. This drawback of BR is overcome by LP, also called as LC (Label Cardinality).

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0
x4	0	1	1	0
x5	1	1	1	1
x6	0	1	0	0



X	y1
x1	1
x2	2
x3	3
x4	1
x5	4
x6	3

### 5.2.1.3 Label Powerset

**Accuracy :-**

57%

**Drawback :-**

As the training data increases, number of classes become more. Thus, increasing the model complexity, and would result in a lower accuracy.



### 5.2.2 Adapted Algorithms

Adapted algorithm, as the name suggests, adapting the algorithm to directly perform multi-label classification, rather than transforming the problem into different subsets of problems. Algorithm that changes its behavior at the time it is run, based on information available and on a priori defined reward mechanism. Such information could be the story of recently received data, information on the available computational resources, or other run-time acquired information related to the environment in which it operates.

This method can be carried out in different ways as:

1. K – Nearest Neighbours (kNN)
2. Logistic Regression

#### 1. K - Nearest Neighbours (KNN)

KNN makes predictions using the training dataset directly. Predictions are made for a new instance ( $x$ ) by searching through the entire training set for the  $K$  most similar instances and summarizing the output variable for those  $K$  instances. For regression this might be the mean output variable, in classification this might be the mode class value.

To determine which of the  $K$  instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.

Euclidean distance is calculated as the square root of the sum of the squared differences between a new point ( $x$ ) and an existing point ( $x_i$ ) across all input attributes  $j$ .

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + \dots + (x_n - x'_n)^2}$$

It performs 2 steps :

1. It runs through the whole dataset computing  $d$  between  $x$  and each training observation.
2. It then estimates the conditional probability for each class, that is, the fraction of points in  $A$  with that given class label

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

The computational complexity of KNN increases with the size of the training dataset. For very large training sets, KNN can be made stochastic by taking a sample from the training dataset from which to calculate the K-most similar instances.

KNN has been around for a long time and has been very well studied. As such, different disciplines have different names for it, for example:

**Instance-Based Learning:** The raw training instances are used to make predictions. As such KNN is often referred to as instance-based learning or a case-based learning (where each training instance is a case from the problem domain).

**Lazy Learning:** No learning of the model is required and all of the work happens at the time a prediction is requested. As such, KNN is often referred to as a lazy learning algorithm.

**Non-Parametric:** KNN makes no assumptions about the functional form of the problem being solved. As such KNN is referred to as a non-parametric machine learning algorithm.

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance. For example, in a binary classification problem (class is 0 or 1):

$$p(\text{class}=0) = \text{count}(\text{class}=0) / (\text{count}(\text{class}=0) + \text{count}(\text{class}=1))$$

If you are using K and you have an even number of classes (e.g. 2) it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes.

Ties can be broken consistently by expanding K by 1 and looking at the class of the next most similar instance in the training dataset.

## 2. Logistic regression

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability. A Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the ‘Sigmoid function’ or also known as the ‘logistic function’.

### Logistic function

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It’s an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$1 / (1 + e^{-\text{value}})$$

Where e is the base of the natural logarithms (Euler’s number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform.

Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value. Below is an example logistic regression equation:

$$y = e^{(b_0 + b_1 * x)} / (1 + e^{(b_0 + b_1 * x)})$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

## Types of Logistic Regression

### 1. Binary Logistic Regression

Binary logistic regression is used to predict the odds of being a case based on the values of the independent variables (predictors). The odds are defined as the probability that a particular outcome is a case divided by the probability that it is a non case.

### 2. Multinomial Logistic Regression

Multinomial Logistic Regression is the regression analysis to conduct when the dependent variable is nominal with more than two levels. Similar to multiple linear regression, the multinomial regression is a predictive analysis.

### 3. Ordinal Logistic Regression

Ordinal logistic regression (often just called 'ordinal regression') is used to predict an ordinal dependent variable given one or more independent variables. As with other types of regression, ordinal regression can also use interactions between independent variables to predict the dependent variable.

Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that we can no longer understand the predictions as a linear combination of the inputs as we can with linear regression. Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result.

## 5.3 Technologies

### Count Vectorizer

The Count vectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. Call the `transform()` function on one or more documents as needed to encode each as a vector.

### TF-IDF Vectorizer

In information retrieval, tf-idf or TFIDF, short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

### Term Frequency (TF)

The number of times a word appears in a document divided by the total number of words in the document. Every document has its own term frequency.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

**Inverse Data Frequency (IDF)**

The log of the number of documents divided by the number of documents that contain the word. Inverse data frequency determines the weight of rare words across all documents in the corpus.

$$idf(w) = \log\left(\frac{N}{df_t}\right)$$

**Performance Metrics**

Different classifiers like Logistic regression and SGDC classifier is used to calculate the accuracy.

**Hamming Loss**

In simplest of terms, Hamming-Loss is the fraction of labels that are incorrectly predicted, i.e., the fraction of the wrong labels to the total number of labels.

$$\text{Hamming loss} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \Delta Z_i|}{|L|}$$

**Precision and Recall**

Precision can be seen as a measure of exactness or quality, whereas recall is a measure of completeness or quantity. In simple terms, high precision means that an algorithm returned substantially more relevant results than irrelevant ones, while high recall means that an algorithm returned most of the relevant results.

So for this problem we should get high precision and high recall rates. For example, let's assume that we have a title, description with 4 tags. If we want to predict any of the tags we should have a high precision value i.e, we have to be really sure that the predicted tag belongs to the given question. Also, we want to have a high Recall rate, which means if the tag actually supposed to be present, we want it to be present most number of times.

$$\text{Recall} = \frac{\text{True positives}}{\text{Number of positives}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{True positives}}{\text{Number of predicted positive}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

### Micro Averaged F1 Score

For a standard Binary or Multi-class classification problems, we can use performance metrics like Precision, Recall, F1-Score, Log-loss, AUC Curve etc. But for the present Multi-Label problem the mentioned metrics may not work well. As part of the business requirement we want high precision and recall rates for each and every predicted tag. We can use F1 Score here as it only gives good value if both the Precision and Recall are high. The F1 Score performs really well for Binary classifications.

In Micro-average method, you sum up the individual true positives, false positives, and false negatives of the system for different sets and then apply them to get the statistics. It is a weighted f1 score.

$$\text{Micro - Precision} = \frac{\text{TruePositives1} + \text{TruePositives2}}{\text{TruePositives1} + \text{FalsePositives1} + \text{TruePositives2} + \text{FalsePositives2}}$$

$$\text{Micro - Recall} = \frac{\text{TruePositives1} + \text{TruePositives2}}{\text{TruePositives1} + \text{FalseNegatives1} + \text{TruePositives2} + \text{FalseNegatives2}}$$

$$\text{Micro - F - Score} = 2 \cdot \frac{\text{Micro - Precision} \cdot \text{Micro - Recall}}{\text{Micro - Precision} + \text{Micro - Recall}}$$

## Macro Averaged F1 Score

The Macro-average F-Score will be simply the harmonic mean. Macro-average method can be used when you want to know how the system performs overall across the sets of data. It is non-weighted simple mean average f1 score.

$$\text{Macro - Precision} = \frac{\text{Precision1} + \text{Precision2}}{2}$$

$$\text{Macro - Recall} = \frac{\text{Recall1} + \text{Recall2}}{2}$$

$$\text{Macro - F - Score} = 2 \cdot \frac{\text{Macro - Precision} \cdot \text{Macro - Recall}}{\text{Macro - Precision} + \text{Macro - Recall}}$$

## 5.4 Code

```
import pandas as pd
import numpy as np
import re
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.model_selection import train_test_split
start = datetime.now()
```



```
df = pd.read_csv("/kaggle/input/facebook-recruiting-iii-keyword-extraction/Train.zip")
print(df.head())
print(df.columns)
end = datetime.now()
print(end-start)
duplicate_pairs = df.sort_values('Title', ascending=False).duplicated('Title')
print("Total number of duplicate questions : ", duplicate_pairs.sum())
df = df[~duplicate_pairs]
print("Dataframe shape after duplicate removal : ", df.shape)
df["tag_count"] = df["Tags"].apply(lambda x : "python" and "algorithms" in str(x))
df["tag_count"].value_counts()
start = datetime.now()
i=0
lst1=[]
tags_list = ["android","jquery","php","net","javascript","java","asp","ruby","python","mysql","sql"]
for result in df.Tags:
    if (len(str(result).split())<2 and (str(result) in tags_list)):
        val = [df.iloc[i]["Id"],df.iloc[i]["Title"],df.iloc[i]["Body"],df.iloc[i]["Tags"]]
        lst1.append(val)
        i+=1
print(len(lst1))
print("Done")
end = datetime.now()
print(end-start)
```

```
data = pd.DataFrame(lst1,columns=["Id", "Title", "Body", "Tags"])
                #, columns=["Id", "Title", "Body", "Tags"])
print("Done")
vectorizer = CountVectorizer()
tag_bow = vectorizer.fit_transform(data["Tags"].values.astype('U'))
print("Number of questions :", tag_bow.shape[0])
print("Number of unique tags :", tag_bow.shape[1])
tags = vectorizer.get_feature_names()
freq = tag_bow.sum(axis=0).A1
tag_to_count_map = dict(zip(tags, freq))
lt = []
for key, value in tag_to_count_map.items():
    lt.append([key, value])
l = sorted(lt,key=lambda x:x[1],reverse = True)
print("done")
tag_df = pd.DataFrame(l, columns=['Tags', 'Counts'])
tag_df.head(20)
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
start = datetime.now()
qus_list=[]
qus_with_code = 0
len_before_preprocessing = 0
len_after_preprocessing = 0
for index,row in data.iterrows():
    title, body, tags = row["Title"], row["Body"], row["Tags"]
    if '<code>' in body:
        qus_with_code+=1
        len_before_preprocessing+=len(title) + len(body)
        body=re.sub('<code>(.*?)</code>', '', body, flags=re.MULTILINE|re.DOTALL)
        body = re.sub('<.*?>', ' ', str(body.encode('utf-8'))))
```

```
title=title.encode('utf-8')
question=str(title)+" "+str(body)
question=re.sub(r'[^A-Za-z]+' ,',',question)
words=word_tokenize(str(question.lower()))
question=' '.join(str(stemmer.stem(j)) for j in words if j not in stop_words and
(len(j)!=1      qus_list.append(question)
len_after_preprocessing += len(question)
data["question"] = qus_list
avg_len_before_preprocessing=(len_before_preprocessing*1.0)/data.shape[0]
avg_len_after_preprocessing=(len_after_preprocessing*1.0)/data.shape[0]
print(  "Avg.  length  of  questions(Title+Body)  before  preprocessing:  ",
avg_len_before_preprocessing)
print(  "Avg.  length  of  questions(Title+Body)  after  preprocessing:  ",
avg_len_after_preprocessing)
print ("% of questions containing code: ", (qus_with_code*100.0)/data.shape[0])
end = datetime.now()
print(end-start)

preprocessed_df = data[["question","Tags"]]
print("Shape of preprocessed data :", preprocessed_df.shape)
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
y_multilabel = vectorizer.fit_transform(preprocessed_df['Tags'])
def tags_to_consider(n):
    tag_i_sum = y_multilabel.sum(axis=0).tolist()[0]
    sorted_tags_i = sorted(range(len(tag_i_sum)), key=lambda i: tag_i_sum[i],
reverse=True)
    yn_multilabel=y_multilabel[:,sorted_tags_i[:n]]
    return yn_multilabel

def questions_covered_fn(num):
    yn_multilabel = tags_to_consider(num)
```

```
x= yn_multilabel.sum(axis=1)
    return (np.count_nonzero(x==0))
questions_covered = []
total_tags=y_multilabel.shape[1]
total_qus=preprocessed_df.shape[0]
print("total_tags : ",total_tags)
print("total_qus : ",total_qus)
for i in range(0, total_tags, 3):
    print("entering for loop : ",i)
    qus_cov = questions_covered_fn(i)
    print(qus_cov)
    questions_covered.append(np.round((((total_qus-qus_cov)/total_qus)*100,3))

plt.plot(np.arange(2,total_tags, 2),questions_covered)
plt.xlabel("Number of tags")
plt.ylabel("Number of questions covered partially")
plt.grid()
plt.show()
print("Number of questions that are not covered by 100 tags : ",
questions_covered_fn(1000),"out of ", total_qus)
yx_multilabel = tags_to_consider(1000)
print("Number of tags in the subset :", y_multilabel.shape[1])
print("Number of tags considered :",
yx_multilabel.shape[1],("(",yx_multilabel.shape[1]/y_multilabel.shape[1])*100,"%)")
X_train, X_test, y_train, y_test = train_test_split(preprocessed_df, yx_multilabel, test_size =
0.2,random_state = 42)
print("Number of data points in training data :", X_train.shape[0])
print("Number of data points in test data :", X_test.shape[0])
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, tokenizer = lambda x:
x.split(), ngram_range=(1,3))
X_train_multilabel = vectorizer.fit_transform(X_train['question'])
```

```
X_test_multilabel = vectorizer.transform(X_test['question'])
print("Training data shape X : ",X_train_multilabel.shape, "Y :",y_train.shape)
print("Test data shape X : ",X_test_multilabel.shape,"Y:",y_test.shape)
clf = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2'))
clf.fit(X_train_multilabel, y_train)
y_pred = clf.predict(X_test_multilabel)
print("Accuracy :",metrics.accuracy_score(y_test,y_pred))
print("Macro f1 score :",metrics.f1_score(y_test, y_pred, average = 'macro'))
print("Micro f1 scoore :",metrics.f1_score(y_test, y_pred, average = 'micro'))
print("Hamming loss :",metrics.hamming_loss(y_test,y_pred))
clf2 = OneVsRestClassifier(LogisticRegression(penalty='l1'))
clf2.fit(X_train_multilabel, y_train)
y_pred2 = clf2.predict(X_test_multilabel)
print("Accuracy :",metrics.accuracy_score(y_test,y_pred2))
print("Macro f1 score :",metrics.f1_score(y_test, y_pred2, average = 'macro'))
print("Micro f1 scoore :",metrics.f1_score(y_test, y_pred2, average = 'micro'))
print("Hamming loss :",metrics.hamming_loss(y_test,y_pred2))
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud

import nltk

from Stopwords import stop_words

from nltk.tokenize import word_tokenize

from nltk.stem.snowball import SnowballStemmer

from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

from sklearn.metrics import f1_score, precision_score, recall_score


from flask import *
from flask import session
app = Flask(__name__)
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=200000, tokenizer=lambda x:
x.split(), ngram_range=(1, 3))
clf = OneVsRestClassifier(LogisticRegression(penalty='l2'))
stemmer = SnowballStemmer("english")
tags_list = tags_lst
@app.route('/')
def main():
    print("Main")
    data_half = pd.read_csv("C:/Pycharm/ProjectTest/MajorProject400000Records.csv")
    data_half["tag_count"] = data_half["Tags"].apply(lambda x: len(str(x).split()))
    print(data_half["tag_count"].value_counts())
    final_list = []
    i = 0
    for result in data_half.Tags:
```

```
split_list = str(result).split()
if "java" in split_list:
    val = [data_half.iloc[i]["Id"], data_half.iloc[i]["Title"], data_half.iloc[i]["Body"],
           data_half.iloc[i]["Tags"]]
    final_list.append(val)
i += 1
if i % 100 == 0:
    print(i)
    l = len(final_list)
    print(l)
    if (l > 1000):
        break
print(len(final_list))

final_list = final_list[0:1000]
data = pd.DataFrame(final_list, columns=["Id", "Title", "Body", "Tags"])
# , columns=["Id", "Title", "Body", "Tags"])
# data = data_half.iloc[:10000, :]
print("Done")
data["tag_count"] = data_half["Tags"].apply(lambda x: len(str(x).split()))
print(data["tag_count"].value_counts())
vectorizerC = CountVectorizer()
tag_bow = vectorizerC.fit_transform(data["Tags"].values.astype('U'))
print("Number of questions :", tag_bow.shape[0])
print("Number of unique tags :", tag_bow.shape[1])
tags = vectorizerC.get_feature_names()
freq = tag_bow.sum(axis=0).A1
#stops = stop_words
stops = set(stopwords.words('english'))
print(len(stops))
qus_list = []
```

```
qus_with_code = 0
len_before_preprocessing = 0
len_after_preprocessing = 0
i = 0
for index, row in data.iterrows():
    title, body, tags = row["Title"], row["Body"], row["Tags"]
    if '<code>' in body:
        qus_with_code += 1
    len_before_preprocessing += len(title) + len(body)
    body = re.sub('<code>(.*?)</code>', '', body, flags=re.MULTILINE | re.DOTALL)
    body = re.sub('<.*?>', '', str(body.encode('utf-8')))
    title = title.encode('utf-8')
    question = str(title) + " " + str(body)
    question = re.sub(r'[^A-Za-z]+', '', question)
    words = word_tokenize(str(question.lower()))
    question = ' '.join(str(stemmer.stem(j)) for j in words if j not in stops and (len(j) != 1 or j
== 'c'))
    qus_list.append(question)
    len_after_preprocessing += len(question)
    if i % 10 == 0:
        print(i)
    i += 1
data["question"] = qus_list

avg_len_before_preprocessing = (len_before_preprocessing * 1.0) / data.shape[0]
avg_len_after_preprocessing = (len_after_preprocessing * 1.0) / data.shape[0]
print("Avg. length of questions(Title+Body) before preprocessing: ",
avg_len_before_preprocessing)
print("Avg. length of questions(Title+Body) after preprocessing: ",
avg_len_after_preprocessing)
print("% of questions containing code: ", (qus_with_code * 100.0) / data.shape[0])
```



```
preprocessed_df = data[["question", "Tags"]]
print("Shape of preprocessed data :", preprocessed_df.shape)
vectorizerC = CountVectorizer(tokenizer=lambda x: x.split(), binary='true')
y_multilabel = vectorizerC.fit_transform(preprocessed_df['Tags'])

    def tags_to_consider(n):
        tag_i_sum = y_multilabel.sum(axis=0).tolist()[0]
        sorted_tags_i = sorted(range(len(tag_i_sum)), key=lambda i: tag_i_sum[i],
reverse=True)
        yn_multilabel = y_multilabel[:, sorted_tags_i[:n]]
        return yn_multilabel

def questions_covered_fn(num):
    yn_multilabel = tags_to_consider(num)
    x = yn_multilabel.sum(axis=1)
    return (np.count_nonzero(x == 0))
questions_covered = []
total_tags = y_multilabel.shape[1]
total_qus = preprocessed_df.shape[0]
print("total_tags : ", total_tags)
print("total_qus : ", total_qus)
for i in range(100, total_tags, 100):
    # print("entering for loop : ",i)
    qus_cov = questions_covered_fn(i)
    # print(qus_cov)
    questions_covered.append(np.round((((total_qus - qus_cov) / total_qus) * 100, 3)
        yx_multilabel = tags_to_consider(5)
print("Number of tags in the subset :", y_multilabel.shape[1])
print("Number of tags considered :", yx_multilabel.shape[1], "(",
    (yx_multilabel.shape[1] / y_multilabel.shape[1]) * 100, "%)")
```

```
X_train, X_test, y_train, y_test = train_test_split(preprocessed_df, yx_multilabel,
test_size=0.2, random_state=42)

print("Number of data points in training data :", X_train.shape[0])
print("Number of data points in test data :", X_test.shape[0])
X_train_multilabel = vectorizer.fit_transform(X_train['question'])
X_test_multilabel = vectorizer.transform(X_test['question'])
print("DONE")

clf.fit(X_train_multilabel, y_train)
y_pred = clf.predict(X_test_multilabel)
print("Done")
print("Accuracy :", metrics.accuracy_score(y_test, y_pred))
print("Macro f1 score :", metrics.f1_score(y_test, y_pred, average='macro'))
print("Micro f1 scoore :", metrics.f1_score(y_test, y_pred, average='micro'))
print("Hamming loss :", metrics.hamming_loss(y_test, y_pred))
return render_template('home.html')

@app.route("/inputdata", methods = ['POST', 'GET'])
def inputdata():
    #qus_list = []
    qus_with_code = 0
    len_before_preprocessing = 0
    len_after_preprocessing = 0
    if request.method == 'POST':
        title = request.form['title']
        body = request.form['body']
        print("Title: ",title)
        print("body:",body)
        if '<code>' in body:
            qus_with_code += 1
        len_before_preprocessing += len(title) + len(body)
```

```
body = re.sub('<code>(.*?)</code>', '', body, flags=re.MULTILINE | re.DOTALL)
body = re.sub('<.*?>', '', str(body.encode('utf-8')))
title = title.encode('utf-8')
question = str(title) + " " + str(body)
question = re.sub(r'[^A-Za-z]+', '', question)
words = word_tokenize(str(question.lower()))
question = ' '.join(str(stemmer.stem(j)) for j in words if j not in stop_words and (len(j) != 1
or j == 'c'))
#qus_list.append(question)
msg = []
msg.append(question)
print(msg)
val = vectorizer.transform(msg)
info = pd.DataFrame(val.todense(), columns=vectorizer.get_feature_names())
output = clf.predict(val)
headings = list(info.columns)
lst = list(info.loc[0])
ln = len(headings)
cmplt_lst = []
for i in range(ln):
    tp = []
    tp.append(headings[i])
    tp.append(lst[i])
    cmplt_lst.append(tp)
print("Cmplt_list : ",len(cmplt_lst))
cmp_sort_lst = sorted(cmplt_lst, key=lambda x: x[1], reverse=True)
test_list = cmp_sort_lst[0:60]
print("Tags_list",len(tags_list))
final_list = []
for i in range(60):
    if test_list[i][0] in tags_list:
```

```
final_list.append(test_list[i])  
print(final_list)  
return render_template('home.html', suggested_tags = final_list[:10])  
if __name__ == "__main__":  
    app.run(debug=True)  
main()
```

## 6. TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of software as a system element and attendant costs associated with a software failure are motivating factors for we planned, through testing. Testing is the process of executing a program with the intent of finding an error. The design of tests for software and other engineered products can be as challenging as the initial design of the product itself.

There of basically two types of testing approaches. One is Black-Box testing – the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operated.

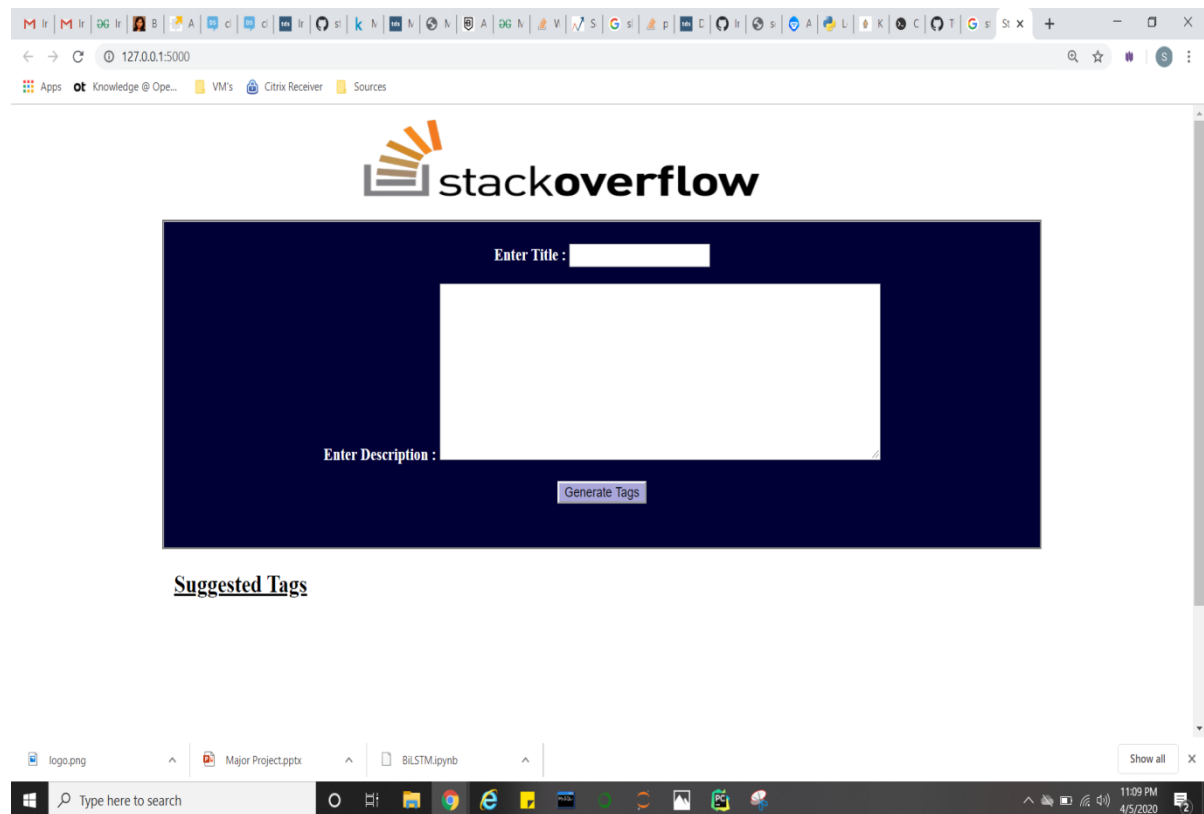
The other is White-Box testing – knowing the internal workings of the product, tests can be conducted to ensure that the internal operation of the product performs according to specifications and all internal components have been adequately exercised.

White box and Black box testing methods have been used to test this package. The entire loop constructs have been tested for their boundary and intermediate conditions. The test data was designed with a view to check for all the conditions and logical decisions. Error handling has been taken care of by the use of exception handlers.

Testing is a set of activities that can be planned in advanced and conducted systematically. A strategy for software testing must accommodation low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

## Results

After data pre-processing, the data has been split into test dataset and train dataset. Once the data has been split, different algorithms are applied. Here logistic regression



**Figure 6.1 Home Page**

The screenshot shows the Stack Overflow question creation interface. At the top, the browser address bar shows '127.0.0.1:5000/inputdata'. Below the browser, the Stack Overflow logo is visible. The main form has a dark blue background. The 'Enter Title' field contains the text 'How to check if an upload'. The 'Enter Description' field contains a text box with the following content: 'I'd like to check if an uploaded file is an image file (e.g. png, jpg, jpeg, gif, bmp) or another file. The problem is that I'm using Uploadify to upload the files, which changes the mime type and gives a 'text/octal' or something as the mime type, no matter which file type you upload. Is there a way to check if the uploaded file is an image apart from checking the file extension using PHP?'. Below the description field is a 'Generate Tags' button.

Figure 6.2 Example Question

The screenshot shows the 'Suggested Tags' section of the Stack Overflow question creation interface. The 'Enter Description' field is empty. Below the description field is a 'Generate Tags' button. The 'Suggested Tags' section lists the following tags: upload, file, check, type, bmp, jpeg, png, php, text, and problem. The Windows taskbar is visible at the bottom of the screen, showing the search bar and various application icons.

Figure 6.3 Suggested tags

## 7. CONCLUSION AND FUTURE SCOPE

### 7.1 Conclusion

We implemented different classifiers that will be able to predict the tags for the questions present on the online forums. Our classification techniques that are based on SGDC classifier and logistic regression approach using tf-idf count of the posts carefully selected the features and tokens from the posts and tried to predict the topic relevant to that post. To achieve the better performance of the system we have taken 1000 most frequent tags in to account using them as the classes for the classification.

### 7.2 Future Scope

As the future work we are planning to include the neighbourhood based tag prediction for predicting tags using k nearest neighbour approach. For example we can train a model with the concept of k-nearest neighbour approach that will predict the tag on the basis of its nearest related terms and tags to increase the accuracy. We can use the fact that the co occurrence of the tags in its neighbourhood depends on the frequency of the terms in a post.



## 8. REFERENCES

- <https://ieeexplore.ieee.org/document/8389059>
- <https://scikit-learn.org/stable/modules/sgd.html>
- <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>
- [https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)
- <https://machinelearningmastery.com/logistic-regression-for-machine-learning/>
- [https://en.wikipedia.org/wiki/Multi-label\\_classification](https://en.wikipedia.org/wiki/Multi-label_classification)
- <https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31>
- <https://www.youtube.com/watch?v=yIYKR4sgzI8>