Predicting Tags for Stack Overflow Questions

TEAM: A-STARS

ADITYA GAYKAR (201505540)

AYUSH KHANDELWAL (201505512)

UTSAV CHOWKSHI (201505581)

Problem Statement

The question answering site Stack Overflow allows users to assign tags to questions in order to make them easier for other people to find.

In this project we aim to develop a classifier that is able to assign tags based on the content of a question.

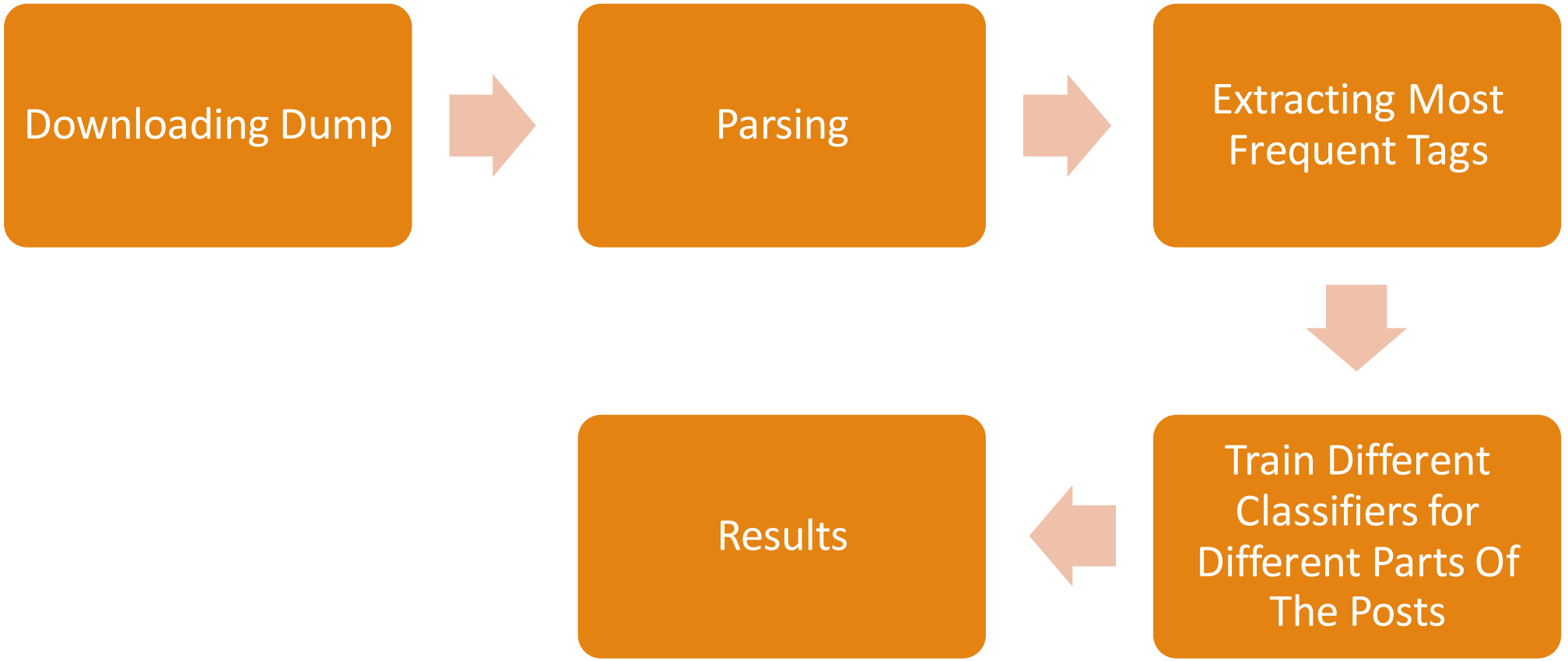
Motivation

Manual tagging by users can be a challenging task for inexperienced users.

Different users use different orthographic versions of tags that mean the same thing such as “php5” and “php- 5”.

For these reasons it is desirable to a have a system that is able to either automatically tag questions or to suggest relevant tags to a user based on the question content

Approach


Downloading Dump
Parsing
Extracting Most Frequent Tags
Train Different Classifiers for Different Parts Of The Posts
Results

Parsing

Parsing was done in Java using the SAX parser library

There were two types of posts in the XML file – Question and Answer

Body, Title and Tags of the 'Question' type posts were extracted

An inverted index was created in the following format:

- <key-word>:<tag-1>|<count>;<tag-2>|<count>

```
35000   zmara:command-line|1;unicode|1;windows|1
35001   znk10bigintegermlerks:biginteger|1;c++|1;integer|1;rsa|1
35002   znst8:c++|1;linker|1;map|1;shared-libraries|1;stl|1
35003   zone:.net|2;accessibility|1;ajax|1;asp.net|1;blind|1;c#|1;c++|1;codesynthesis|1;compiler-
        construction|1;datetime|1;design|1;development-
        environment|1;file|1;iis|1;java|1;json|1;merge|1;multilanguage|1;multiple-monitors|1;network-share|1;ruby|1;ruby-
        on-rails|1;search|1;sharepoint|1;sorting|1;templates|1;timezone|1;translation|1;types|1;visual-studio|1;wcf|1;web-
        services|1;windows-mobile|1;windows-server-2008|1;windows-services|1;workflow|1;xsd|1
35004   zoneinfo:database|1;datetime|1;schema|1;time|1
35005   zones:design|1;file|1;java|1;merge|1;multilanguage|1;ruby|1;ruby-on-
        rails|1;search|1;sharepoint|1;sorting|1;translation|1
35006   zoom:.net|2;iphone|2;javascript|2;zoom|2;c#|1;charts|1;css|1;google-finance|1;html|1;ipod|1;jquery|1;objective-
        c|1;opengl-es|1;pdf|1;wpf|1
```

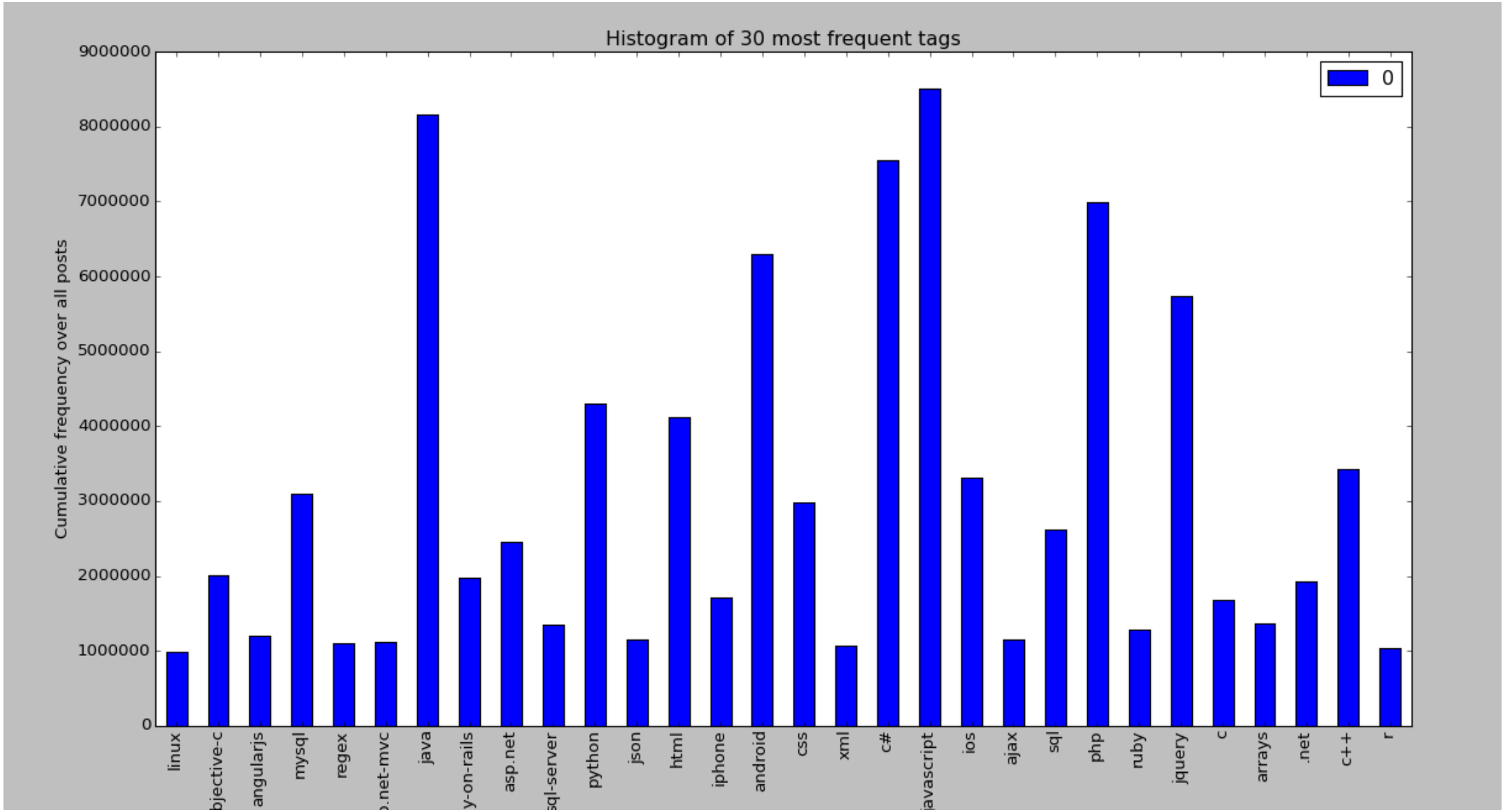
Extracting Most Frequent Tags

From the indexed dump we generated a list of most frequent tags.

Top 5 tags:

- Javascript
- Java
- C#
- Android
- JQuery

For training we used the top 1000 frequent tags.



Graph plot for most frequent tags

Classifiers Used

SVM on titles

SVM on code

Naïve Bayes on code

Naïve Bayes on titles

SVM On Titles

We used sklearn library for multi-label classification

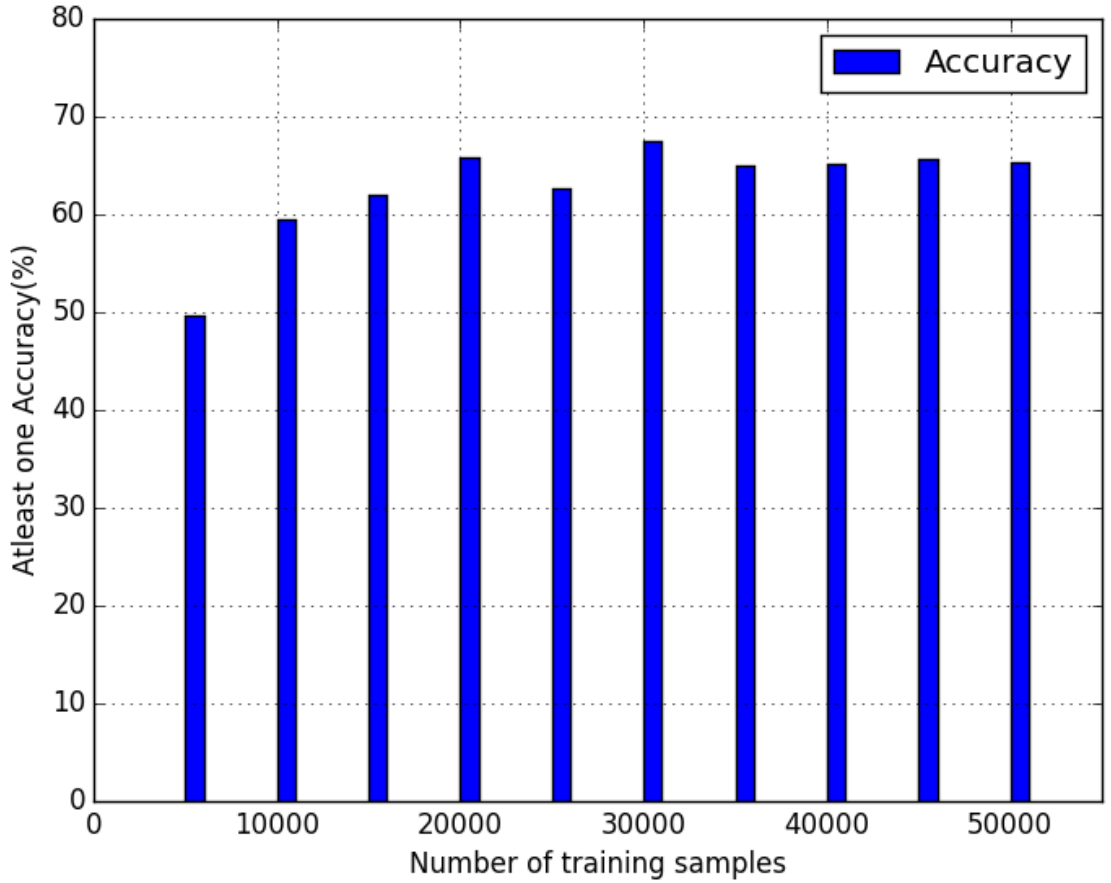
OneVsRestClassifier

LinearSCV

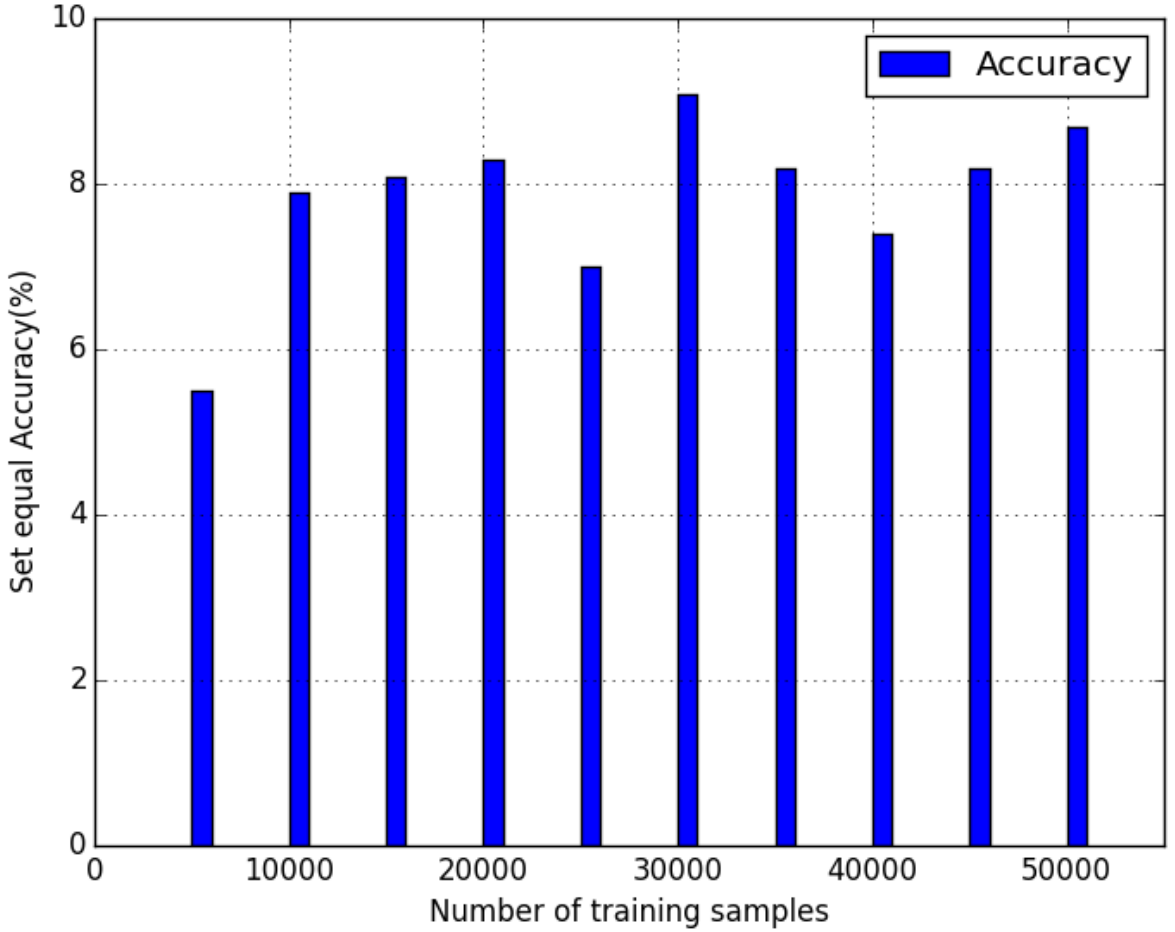
CountVectorization

TfidfTransformation

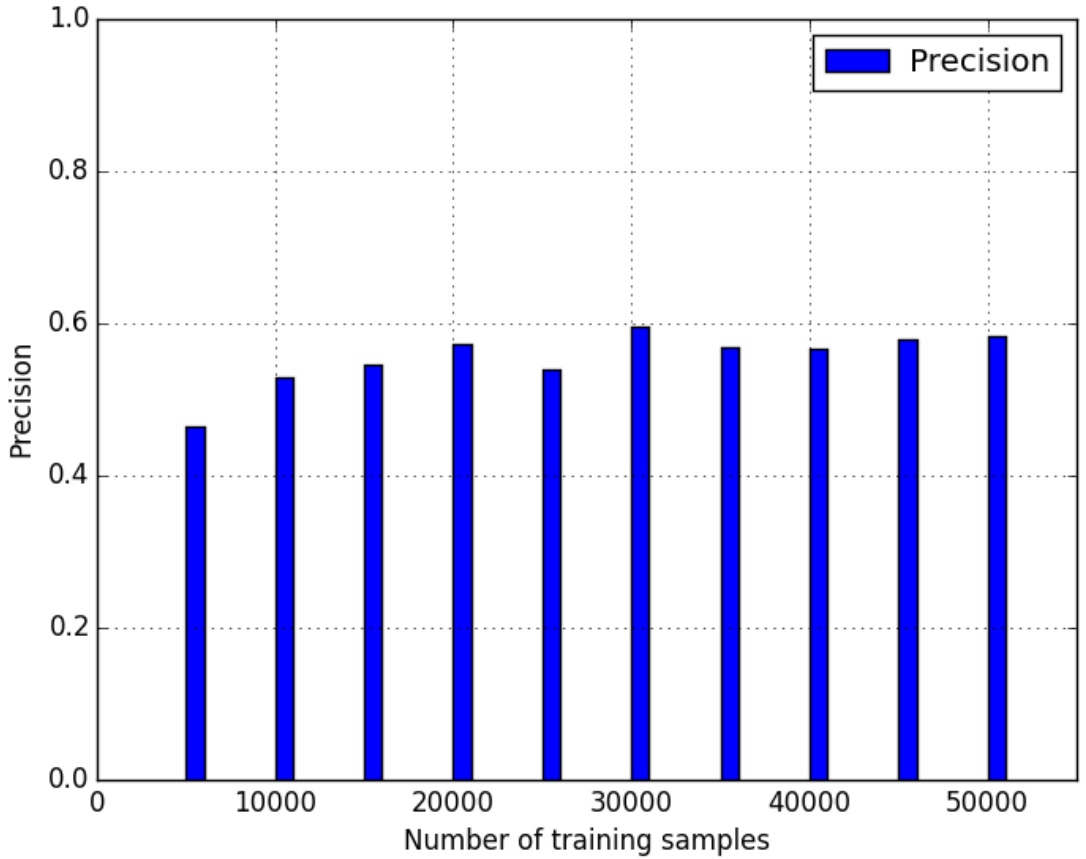
Training samples	Accuracy(Atleast one)	Accuracy(Set equal)	Precision	Recall	F1 Score
5000	49.65034965	5.494505495	0.4637029637	0.23996004	0.2987092273
10000	59.44055944	7.892107892	0.5285880786	0.3121045621	0.3684799328
15000	62.03796204	8.091908092	0.5463036963	0.3276057276	0.383028083
20000	65.83416583	8.291708292	0.5734931735	0.3472693973	0.4038310895
25000	62.63736264	6.993006993	0.5392274392	0.3333166833	0.3862859363
30000	67.43256743	9.090909091	0.5957042957	0.3620712621	0.4214182643
35000	64.93506494	8.191808192	0.5683483183	0.3458208458	0.403002553
40000	65.13486513	7.392607393	0.5676656677	0.3451714952	0.399475128
45000	65.63436563	8.191808192	0.5783882784	0.3526473526	0.4100367886
50000	65.33466533	8.691308691	0.5824009324	0.3436063936	0.4045454545



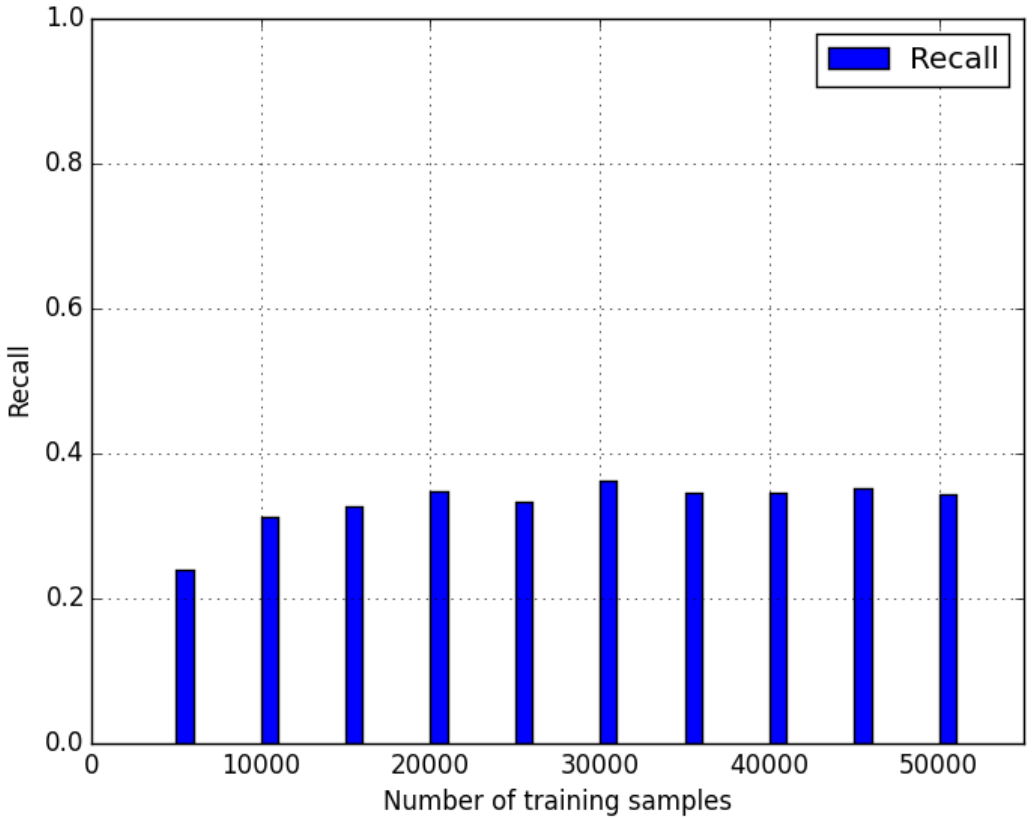
Accuracy (At least one tag present)



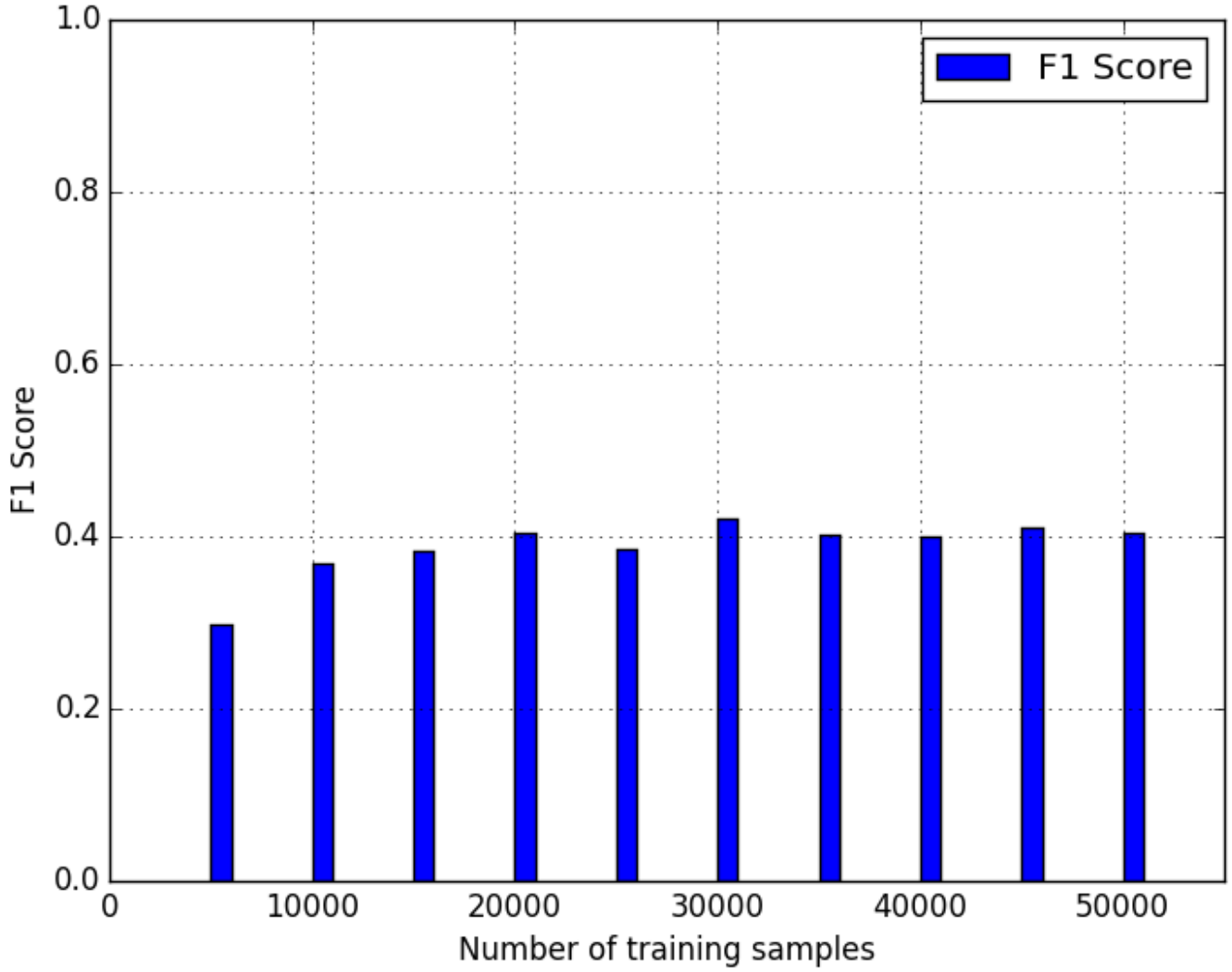
Accuray considering all tags(Exact)



Precision



Recall


F1 Score
1.0
0.8
0.6
0.4
0.2
0.0
F1 Score
0
10000
20000
30000
40000
50000
Number of training samples

F1 Score

Classification On Code

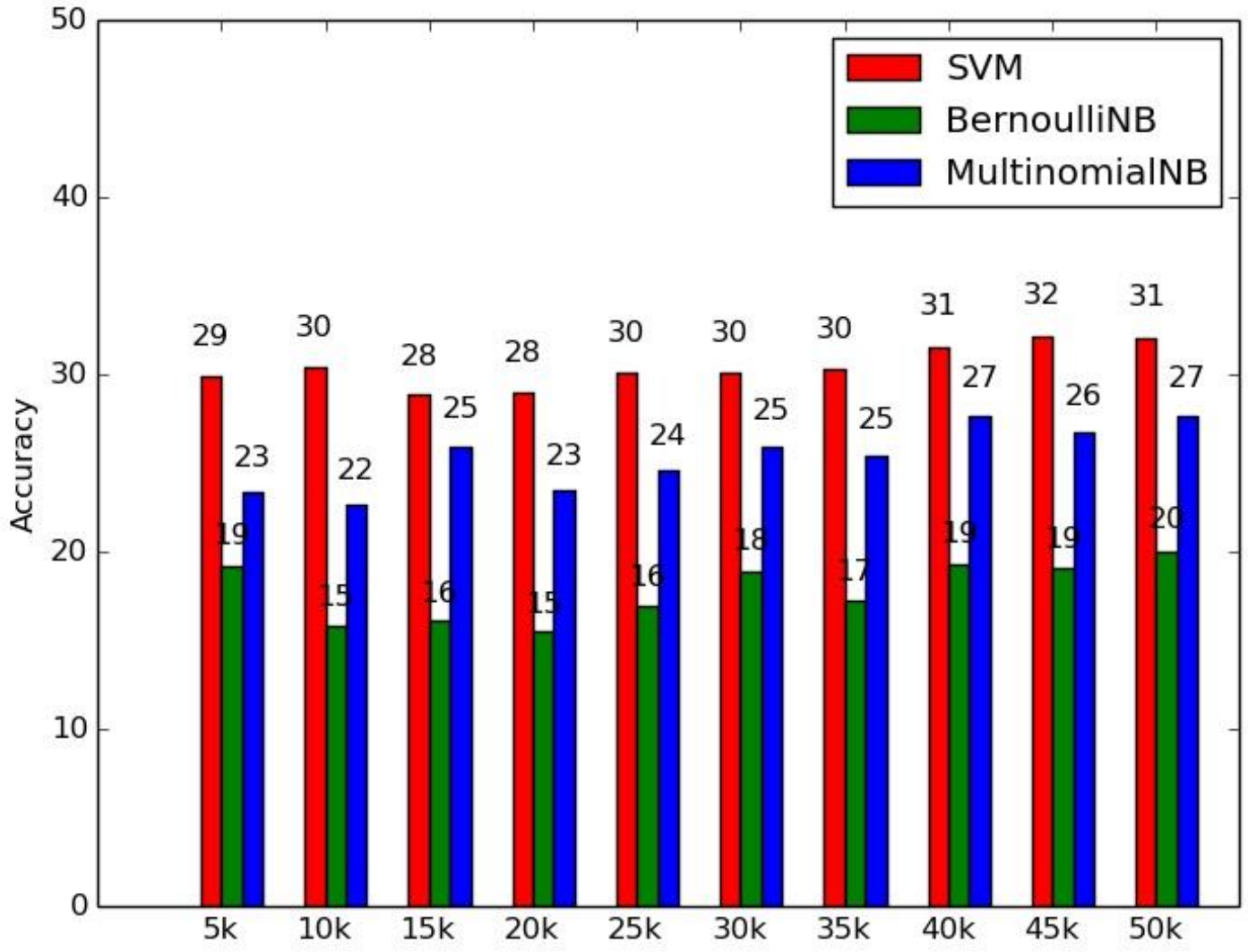
Most of the top 1000 tags are based on programming languages like java, javascript, python etc.

We make a list of such programming language related tags and extract the code portion from the posts and use it as an input to train the classifier and output the desired tag.

We only consider the posts with code in it and who has at least one tag from the given list of programming languages tags.

We use SVM, BernoulliNB and MultinomialNB classifiers of the sklearn library.

No. of Posts	Posts with code and matching tags	SVM (Accuracy%)	BernoulliNB (Accuracy%)	MultinomialNB (Accuracy%)
5000	4336	29.9043062201	19.1387559809	23.3253588517
10000	8644	30.3527980535	15.8150851582	22.6277372263
15000	12911	28.8217107523	16.0769495019	25.9361044315
20000	17355	28.9716840537	15.4992548435	23.4873323398
25000	21581	30.1033230941	16.9177035582	24.557956778
30000	25857	30.0714598221	18.885810121	25.8713723203
35000	30160	30.2941176471	17.181372549	25.4044117647
40000	34586	31.5164220825	19.298858607	27.5797810389
45000	39062	32.1307891075	19.0121248261	26.6944941364
50000	43277	31.9840420356	20.0155687457	27.663715092



Comparison of the three classifiers

We observe that SVM gives us better accuracy than the other classifiers.

No. of Post	Accuracy	Precision	Recall	F1-Score
5000	29.9043062201	0.085200804633	0.041545252976	0.046548672266
10000	30.3527980535	0.106942544984	0.055241729591	0.061653574762
15000	28.8217107523	0.075680276771	0.036323176639	0.041998693675
20000	28.9716840537	0.128198814218	0.061567617916	0.071870415699
25000	30.1033230941	0.104055749116	0.053961406220	0.061846195150
30000	30.0714598221	0.132653542436	0.045476920480	0.056209916837
35000	30.2941176471	0.146084769605	0.052658821629	0.066823337350
40000	31.5164220825	0.153133905202	0.054298982443	0.067770627111
45000	32.1307891075	0.125647779966	0.052821550469	0.064050894893
50000	31.9840420356	0.127782791858	0.052662801862	0.062999973334

Precision, Recall and F1 Score for the SVM classifier

Bayes Analysis on indexed dump

We have preprocessed the complete dump and generated following index files :

- Tersaury.in -> Secondary.in -> Primary.in :
- For storing word vs tag-frequency mapping
- [All three files have been created for title and body separately]
- 1000_TagFreq.in :
  - For storing 1000 most frequent tags along with their frequency
  - Prior.in and Likelihood.in :
  - Probabilities calculated for each tag as well as each tag-title_word combination seen.

```
Performance Metrics for Naive Bayes Approach :
-----------------------------------------------
Number of test samples : 1000.0
Atleast one Accuracy : 32.5
Precision : 0.118333333333
Recall : 0.166354264292
F1 Score : 0.138293728087
```

Unique Feature Extraction Approach

Training :

- We have constructed a feature vector (consisting of six features) for each document-tag pair.(i.e. 1000 feature vectors are constructed for each document)
- We have built 1000 binary linear SVM based classifiers [one for each tag] and provided corresponding feature vector as training sample.
- For all feature vectors of assigned tags , we have kept class-label ‘1’ in corresponding classifier. For all other classifiers, class-label assigned is ‘0’.

Testing :

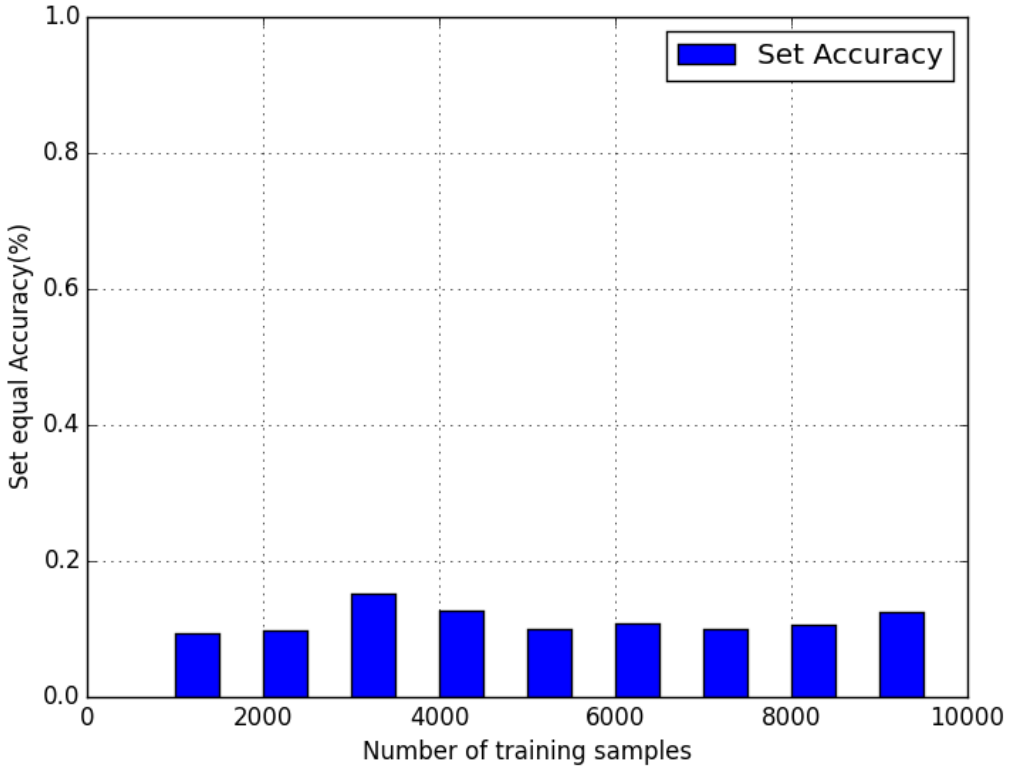
- For each test document, 1000 feature vectors are created.
- Each feature vector is fed to corresponding classifier.
- If classifier emits class-label ‘1’ then that tag is assigned to document.

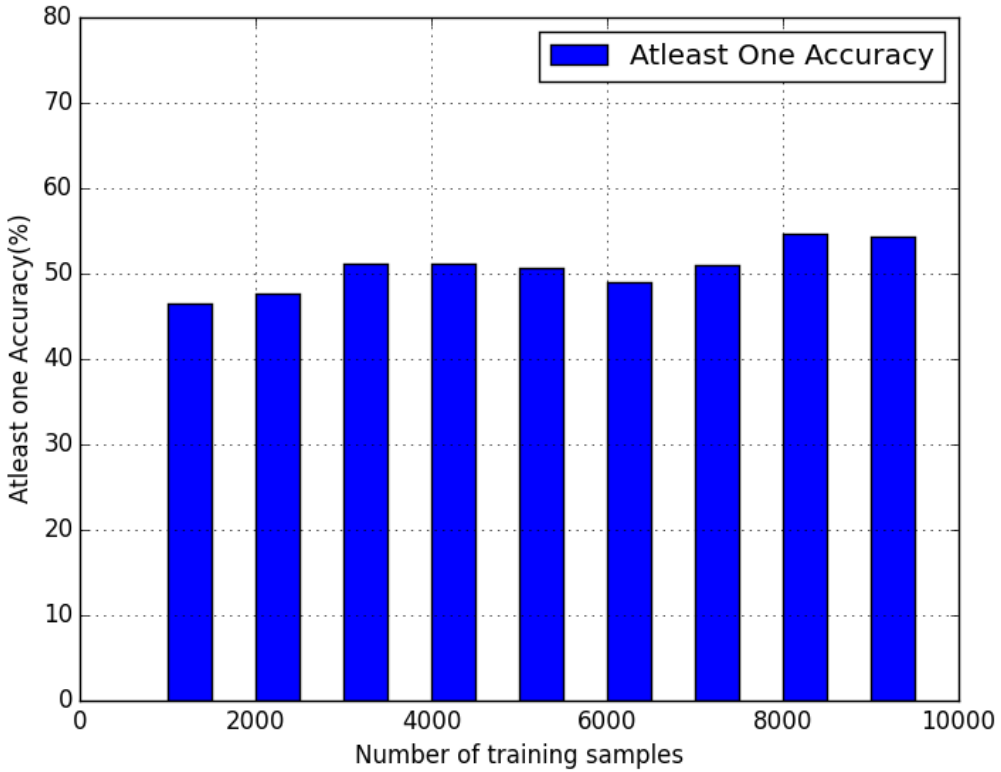
Unique Feature Extraction Approach

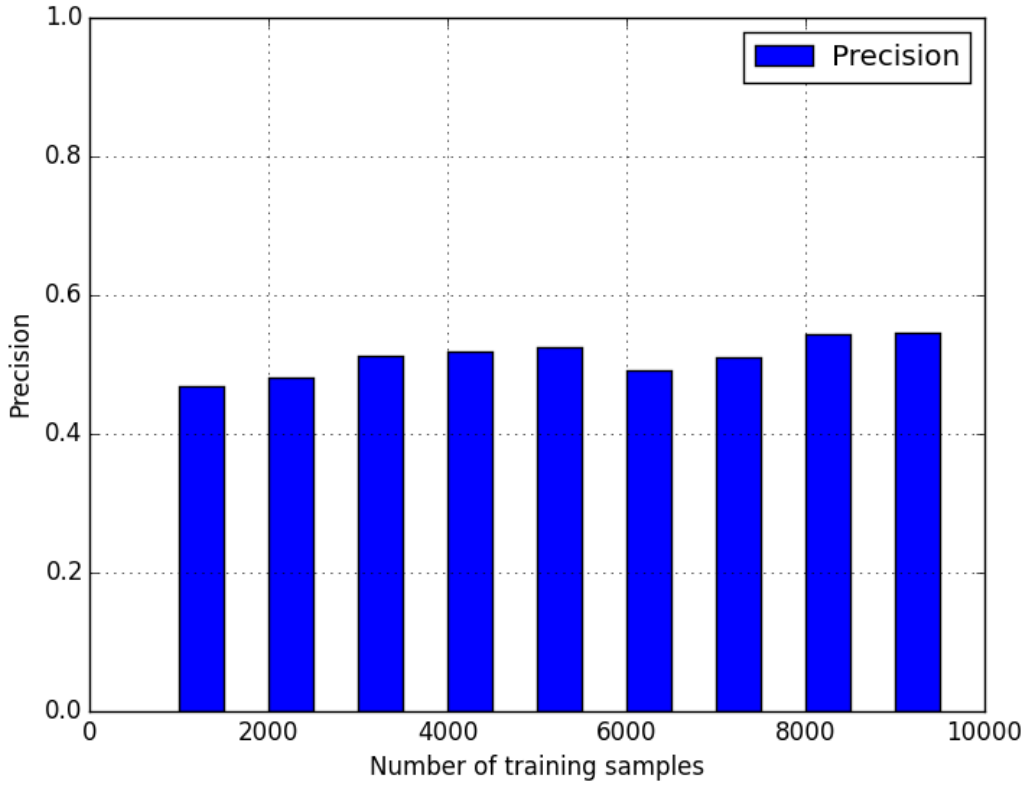
Six features used :

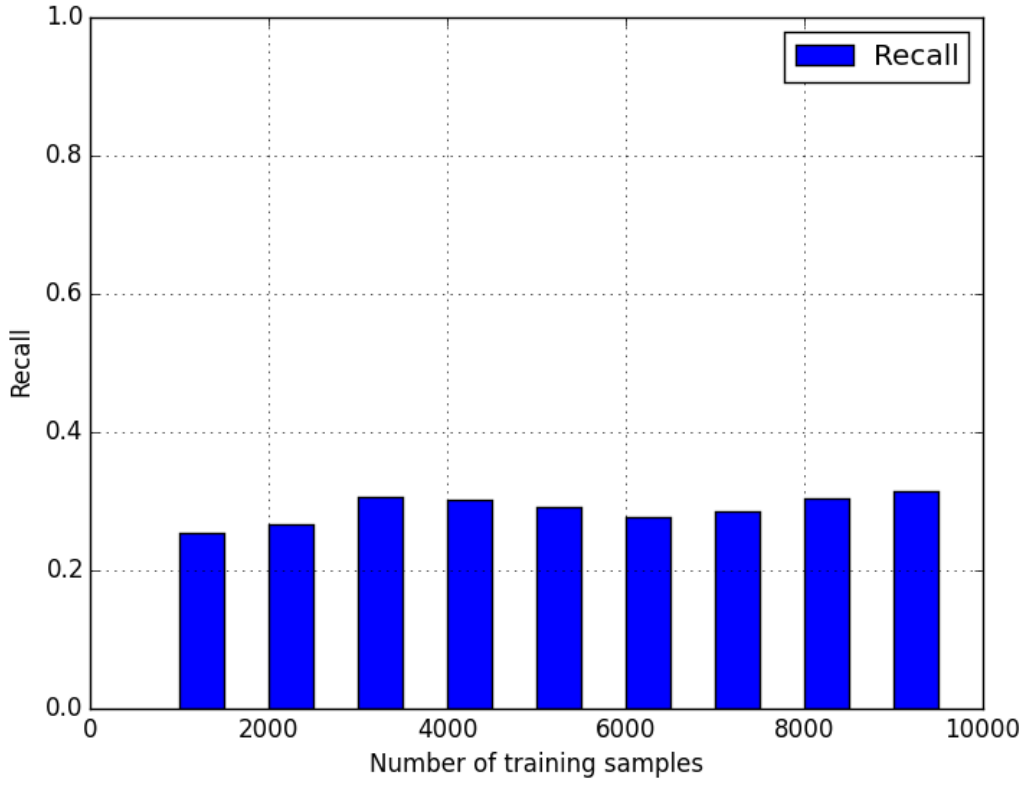
- Exact Tag occurrence in title
- Assign 1 if tag occurs as it is in title. Otherwise 0.
- Exact Tag occurrence in body
- Assign 1 if tag occurs as it is in body. Otherwise 0.
- Relaxed Tag occurrence in title
- Assign 1 if all keywords of tag occurs in title. Otherwise 0.
- Relaxed Tag occurrence in body
- Assign 1 if all keywords of tag occurs in body. Otherwise 0.
- Title PMI (Pointwise Mutual Information)
- Summation over all title-words' PMI. PMI is likelihood of occurring title word and tag occurring together.
- Body PMI (Pointwise Mutual Information)
- Summation over all title-words' PMI. PMI is likelihood of occurring title word and tag occurring together.

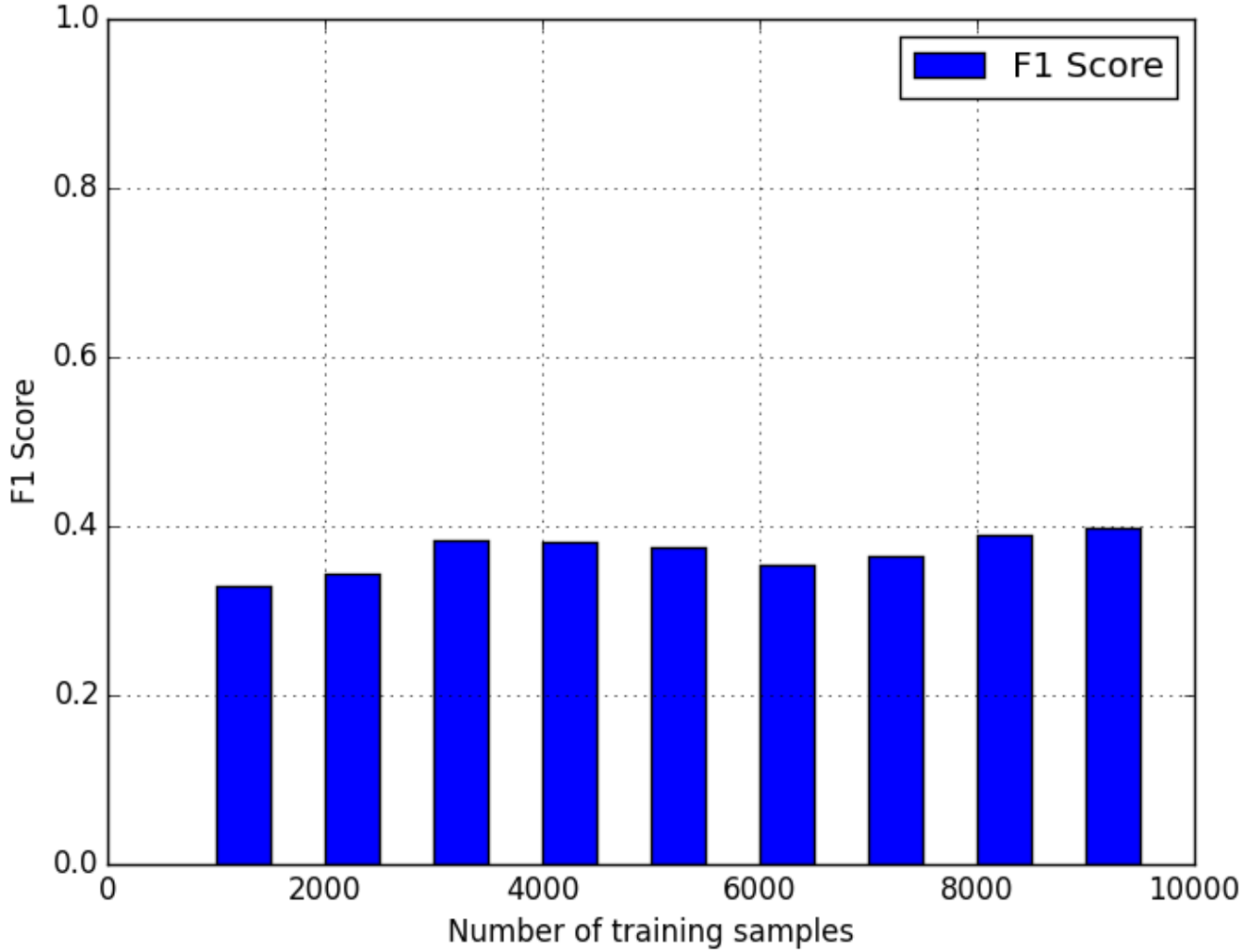
Training samples	At-least-one Accuracy	Set-equal Accuracy(%)	Precision	Recall	F1 Score
1000	0.094	46.5	0.4678362573	0.2540834846	0.3293149074
2000	0.098	47.6	0.4807370184	0.2672253259	0.3435068821
3000	0.152	51.2	0.5118110236	0.3053076562	0.382465431
4000	0.127	51.1	0.5194908512	0.3013382557	0.3814252336
5000	0.101	50.6	0.5252442997	0.2917232022	0.3751090433
6000	0.108	49	0.4925496689	0.2768729642	0.3544831695
7000	0.101	51	0.5098522167	0.2849931161	0.3656167206
8000	0.106	54.6	0.5446708464	0.3044240035	0.3905591458
9000	0.126	54.3	0.5458937198	0.3141797961	0.3988235294


Set Accuracy
Set equal Accuracy(%)
Number of training samples
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000


Atleast One Accuracy
Atleast one Accuracy(%)
Number of training samples
0
10
20
30
40
50
60
70
80
0
2000
4000
6000
8000
10000


Precision
1.0
0.8
0.6
0.4
0.2
0.0
Precision
0
2000
4000
6000
8000
10000
Number of training samples


Recall
1.0
0.8
0.6
0.4
0.2
0.0
Recall
0
2000
4000
6000
8000
10000
Number of training samples


F1 Score
1.0
0.8
0.6
0.4
0.2
0.0
F1 Score
0
2000
4000
6000
8000
10000
Number of training samples

Negative Results

We tried BernoulliNB classifier of the sklearn library on the titles of the posts.

It did not give us good accuracy.

No. of Posts	Accuracy (%)
5000	1.72
10000	4.79
15000	6.73333333333
20000	8.42
25000	9.8999999999
30000	10.58
35000	11.1628571429
40000	12.67
45000	13.4777777778
50000	15.654

Demo …