Predicting Tags for Stack Overflow Questions

A Project by

Team : Astars

Utsav Chokshi (201505581)
Aditya Gaykar (201505540)
Ayush Khandelwal (201505512)

Under the guidance of
Shailesh Jain

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 predictor that is able to assign tags based on the content of a question.

Motivation:

Manual tagging by users generally works well for experienced users, it can be challenging for inexperienced users to find appropriate tags for their question. Also letting users add new tags it is likely that 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

Initial Plan to solve this problem

1 Selecting and cleaning dump

- Selecting dump out of various dumps available
- Subsampling dump and dividing dump into training, validation and test datasets
- Parsing and Extracting title from post and code snippets from question body
- Tokenizing and maintain separate tokens for title and body
- Pass tokens to the decided classifier with higher priority to title tokens.
- Send code snippets to programming language identification classifier

2 Deciding features and classifier

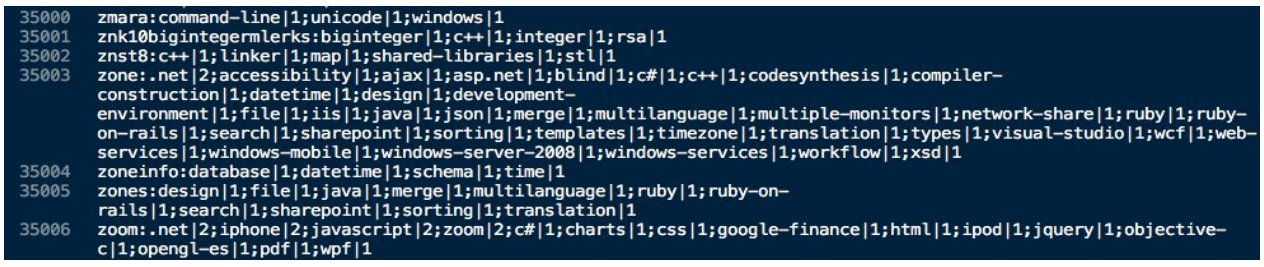
- Decide effective feature deciding and classifier scheme such that it helps us maximise the F1-Score.
- We will look forward for known classifiers as follows :
    - Neural Network - Multilayer perceptron classifier
    - SVM(Support Vector Machines)
    - Bayes Analysis
- Idea is to compare and contrast these algorithms after implementing it over the dataset to gauge its efficiency.

Task Breakdown

Sr no.	Task	Assigned to
1	Gather initial data dumps	Ayush
2	Design approach to implement parsing	All
3	Perform parsing on data dumps	Utsav
4	Tokenizing for title and body separately	Aditya
5	Build classifier for programming language detection	Ayush
6	Try classifier 1 for rest of the data tokens	Utsav
7	Try classifier 2 for rest of the data tokens	Aditya
8	Evaluate and plot results	All

Dump parsing

- Stackoverflow.com dump was downloaded from below link
    - https://archive.org/details/**stack**exchange
- Two separate dump files were downloaded namely **Posts.xml** and **Tags.xml**
- Posts.xml has following tags. Relevant tags are highlighted :
    - Id
    - **PostTypeId**
        - **1: Question**
        - 2: Answer
    - ParentID (only present if PostTypeId is 2)
    - AcceptedAnswerId (only present if PostTypeId is 1)
    - CreationDate
    - Score
    - ViewCount
    - **Body**
    - OwnerUserId
    - LastEditorUserId
    - LastEditorDisplayName="Jeff Atwood"
    - LastEditDate="2009-03-05T22:28:34.823"
    - LastActivityDate="2009-03-11T12:51:01.480"
    - CommunityOwnedDate="2009-03-11T12:51:01.480"
    - ClosedDate="2009-03-11T12:51:01.480"
    - **Title=**
    - **Tags=**
    - AnswerCount
    - CommentCount
    - FavoriteCount
- First step after extracting the dump xml file was to parse it
- Parsing application was written in Java with SAX parser library to parse XML documents
- As we parsed, we created an inverted index of the entire dump. Format is as follows:
    - <key-word>:<tag-1>|<count>;<tag-2>|<count>



Extracting most frequent tags

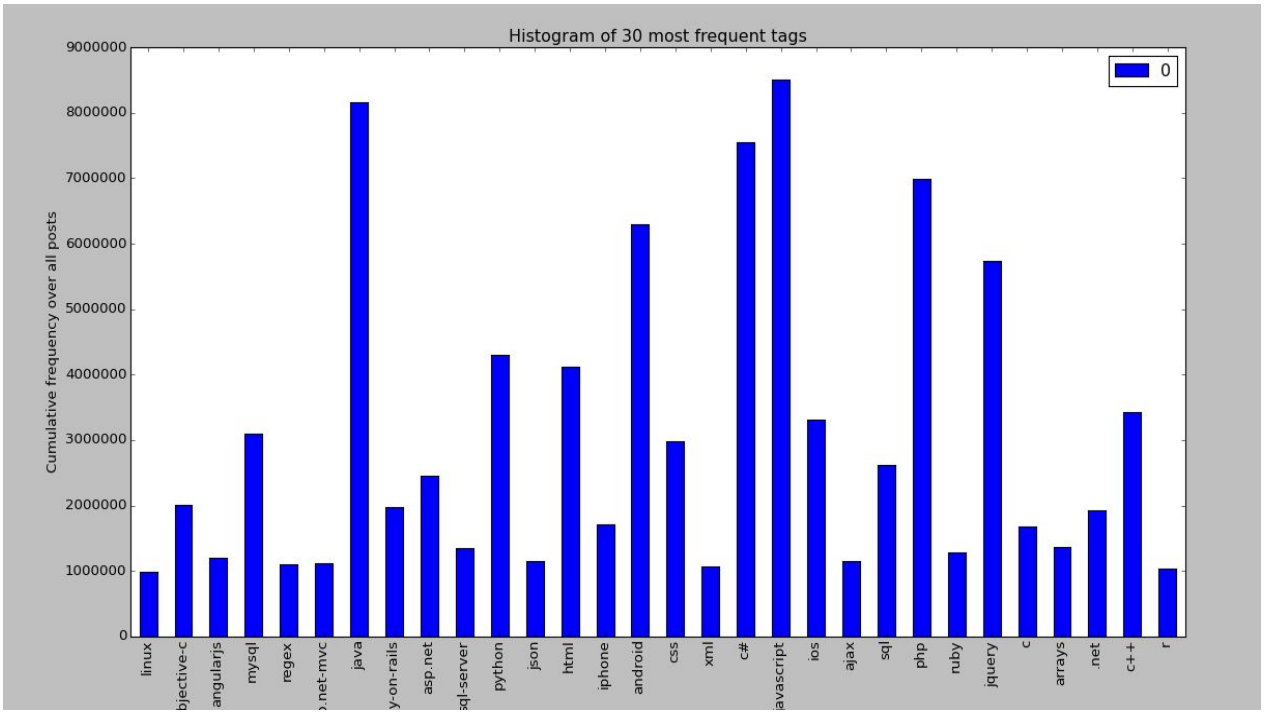


Fig: Graph plot for most frequent tags

- When we analysed the indexed dump, we could generate a list of most frequently occurring tags
- Top 5 tags
    a. Javascript
    b. Java
    c. C#
    d. Android
    e. jQuery
- Now following this approach we decided to predict tags using top 1000 frequent tags, as mentioned in kaggle article[1]. They had suggested that working with just 1000 frequent tags would yield close to good results

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]

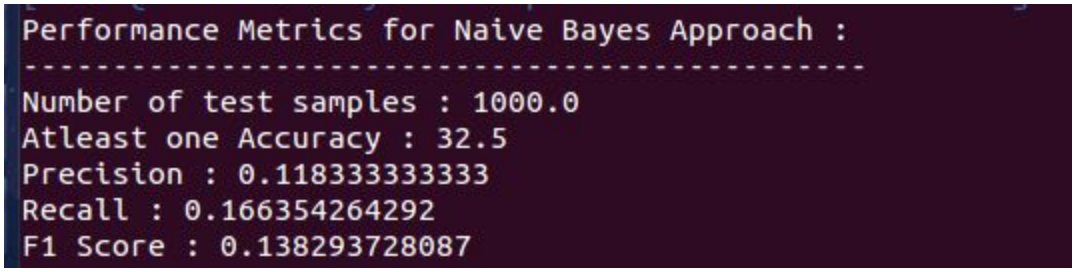
- ***1000_TagFreq.in*** **:**

  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.

We have applied simple *naive bayes approach,* over 100 posts, that tries to predict tag solely based on title text. Following is result :



Observation :

This approach gives considerable accuracy and creates baseline for other approaches.

It requires lot of pre-computing. But once every file is in memory, it performs classification very fast.

SVM Analysis of list of titles

We used sklearn library for multi label classification. For this purpose we have used OneVsRestClassifier, LinearSVC(). This classification technique uses a pipeline structure. Steps in the pipline are as follows:

1. Vectorization : CountVectorizer() was used with stopwords removal
2. TfidfTransformation : TfidfTransformer() was used
3. Classifier : OneVsRestClassifier, LinearSVC()

Results obtained were as follows for a test set of 1000 stackoverflow titles:

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

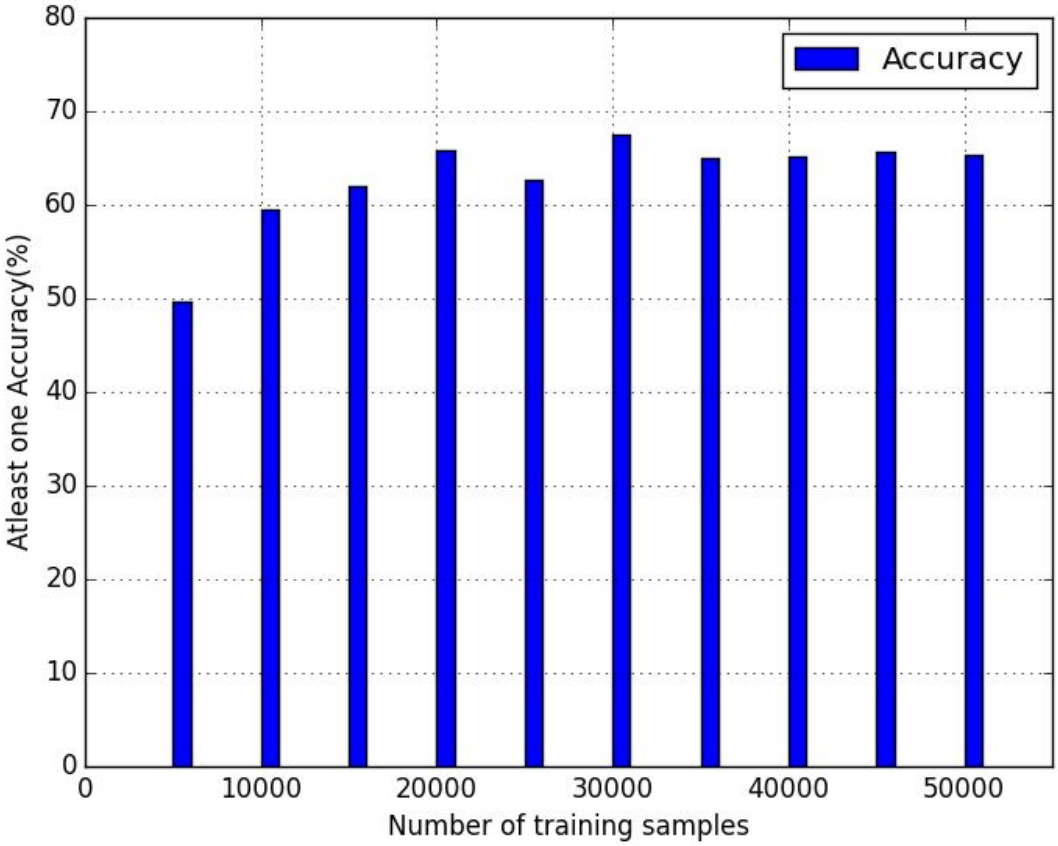


Fig: Accuracy (Atleast one tag present)

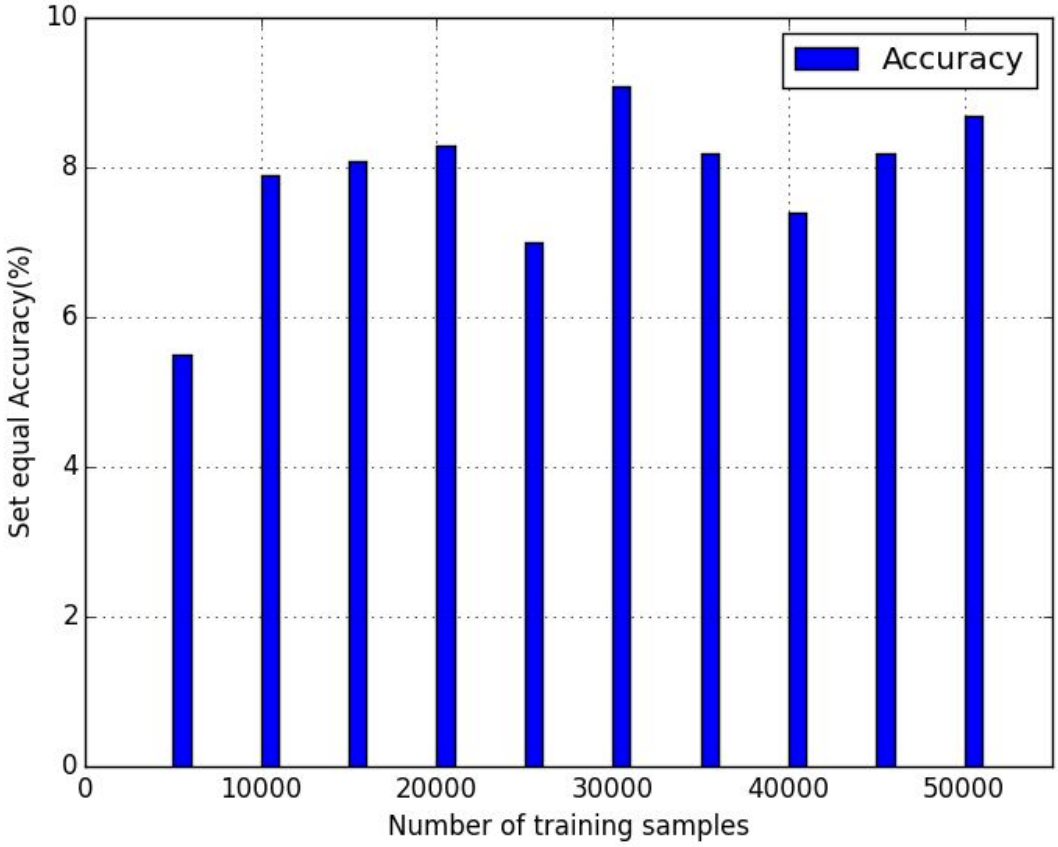


Fig: Accuray considering all tags(Exact)

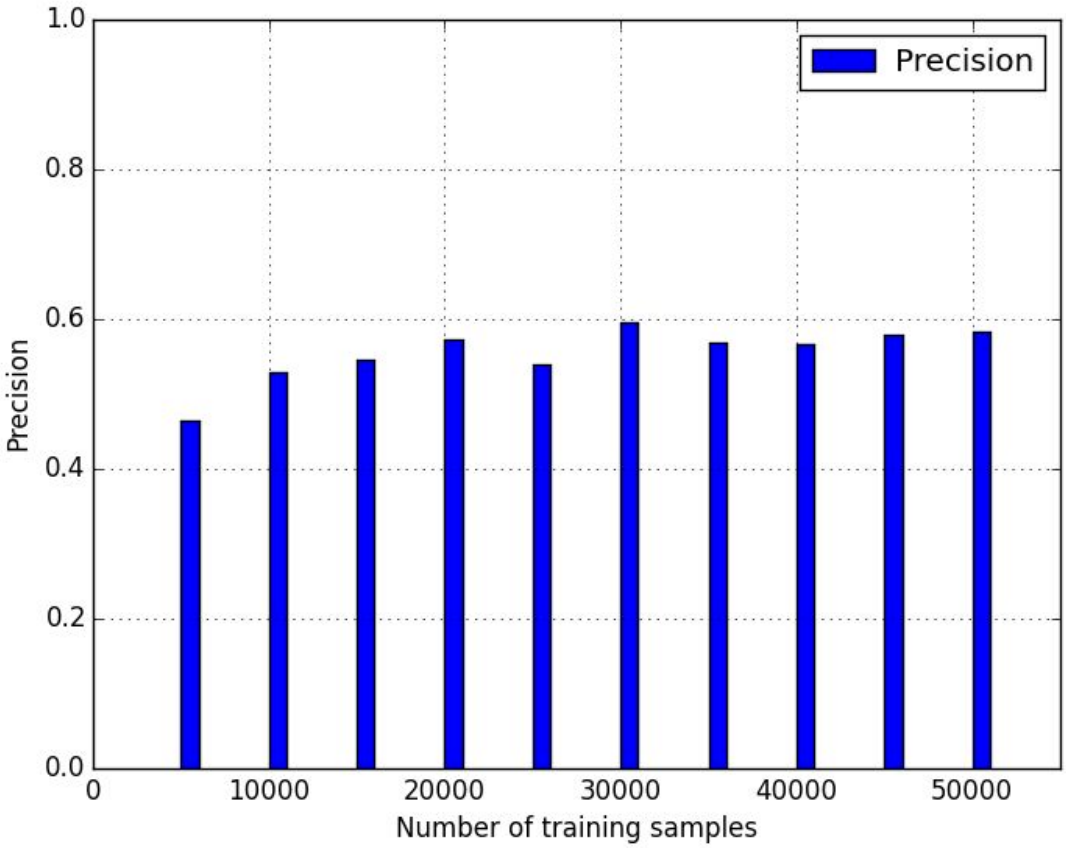


Fig: Precision score

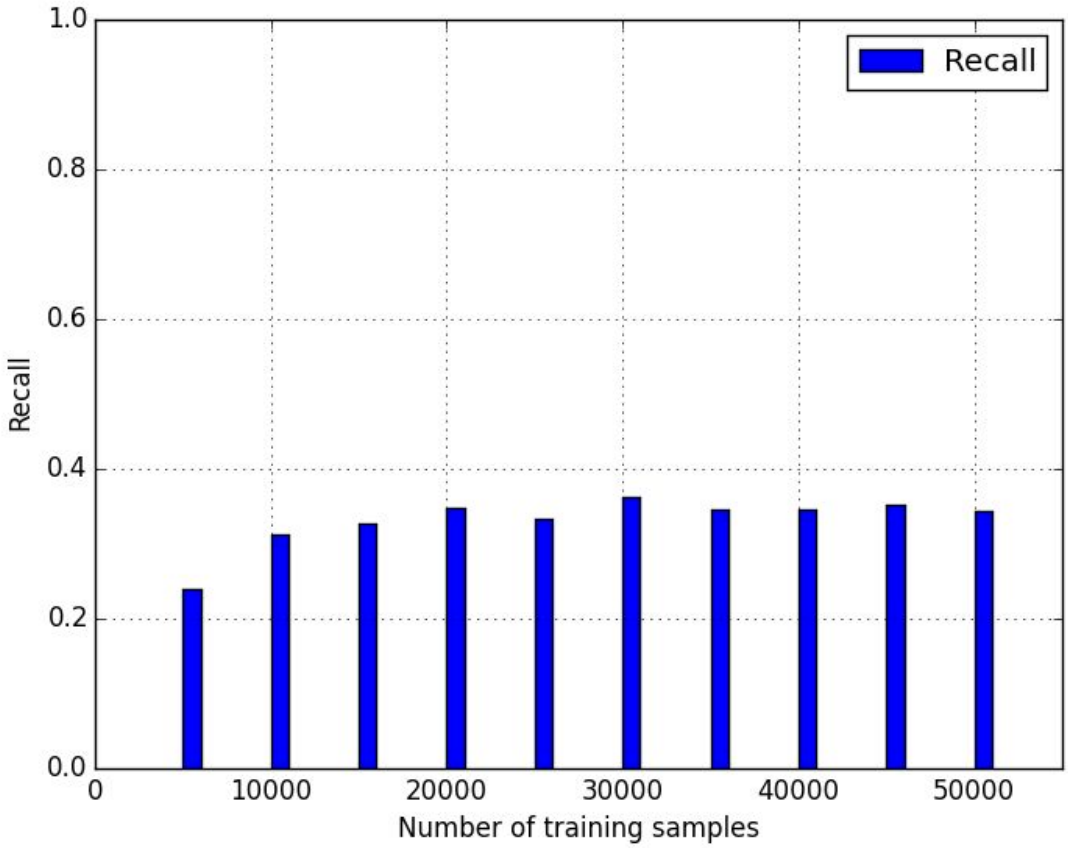


Fig. Recall score

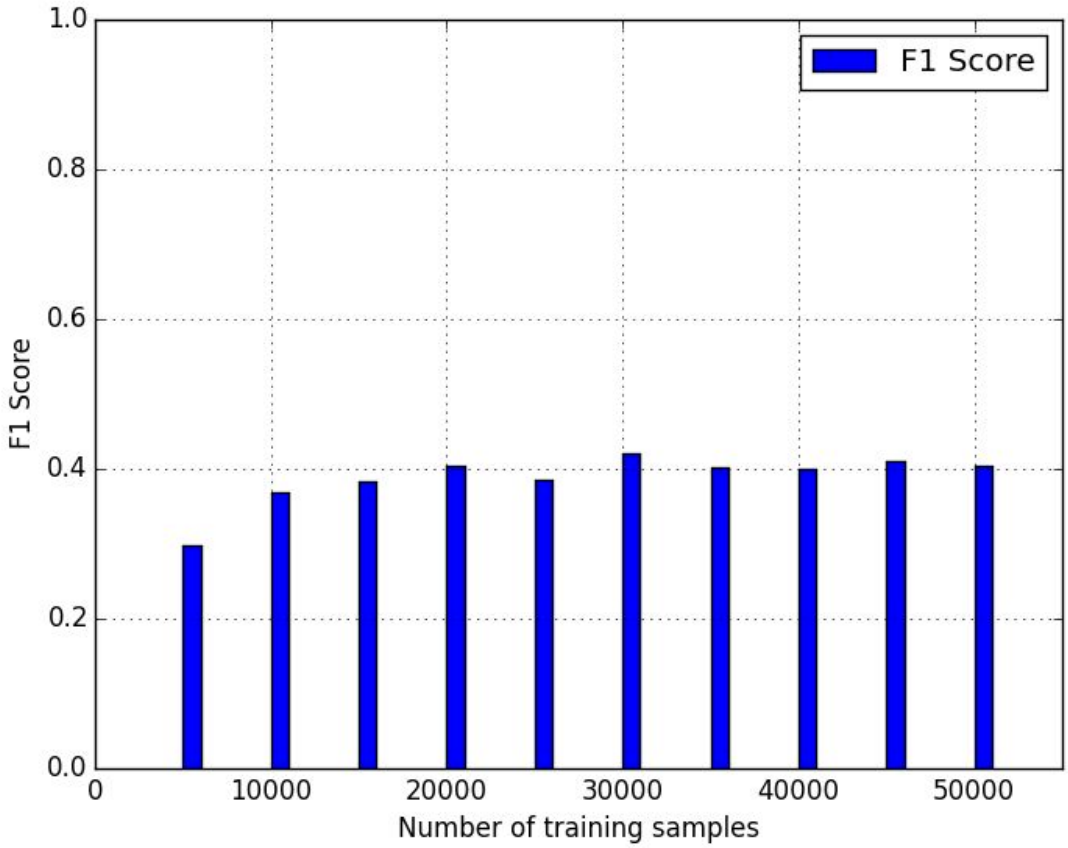

F1 Score
1.0
0.8
0.6
0.4
0.2
0.0
F1 Score
0
10000
20000
30000
40000
50000
Number of training samples

Fig: F1 Score

Unique Feature Extraction Approach

We have considered 1000 most frequent tags as classes in this approach. Each new test sample will be assigned tag from one of the 1000 tags.

We have followed these steps for training and testing :

Training :

1) 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)
2) We have built 1000 binary linear SVM based classifiers [one for each tag] and provided corresponding feature vector as training sample.
3) 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:

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

Six features used :

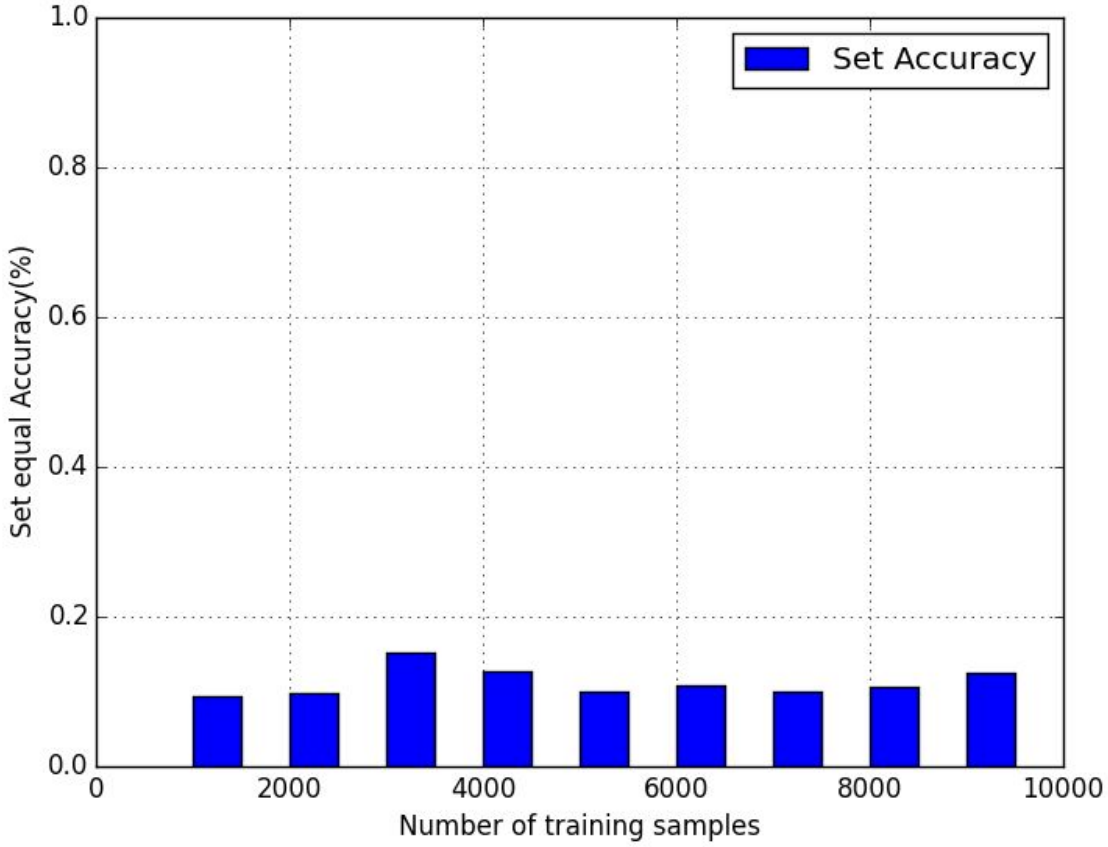
1) Exact Tag occurrence in title
   - Assign 1 if tag occurs as it is in title. Otherwise 0.
2) Exact Tag occurrence in body
   - Assign 1 if tag occurs as it is in body. Otherwise 0.
3) Relaxed Tag occurrence in title
   - Assign 1 if all keywords of tag occurs in title. Otherwise 0.
4) Relaxed Tag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

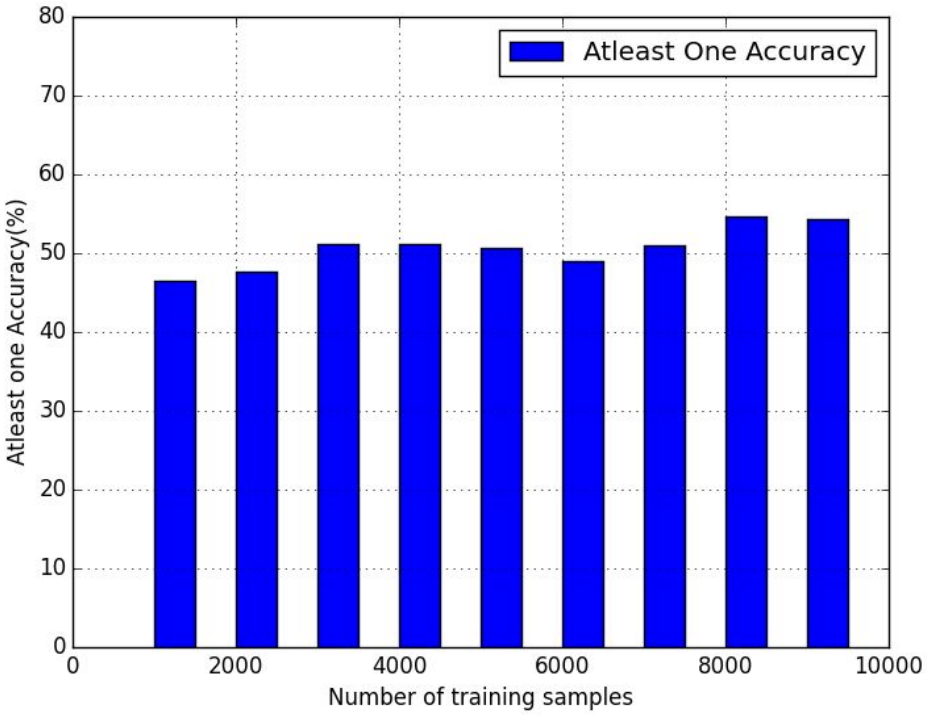
Observation :

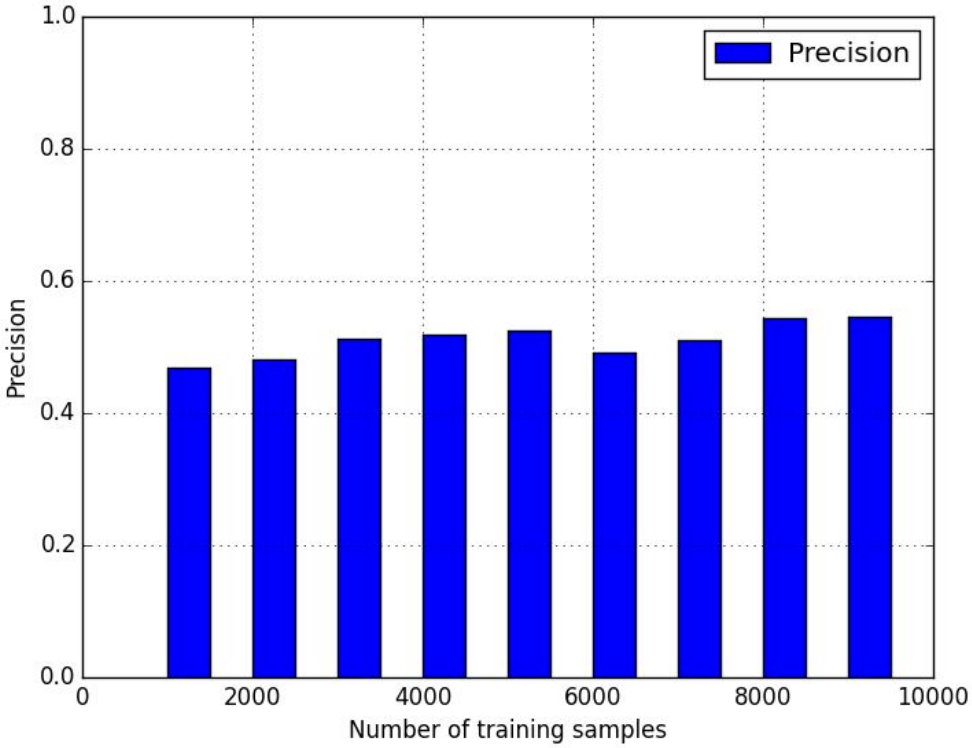
This approach gives good accuracy with small number of training samples compared to other approach. But this approach is computationally very expensive.

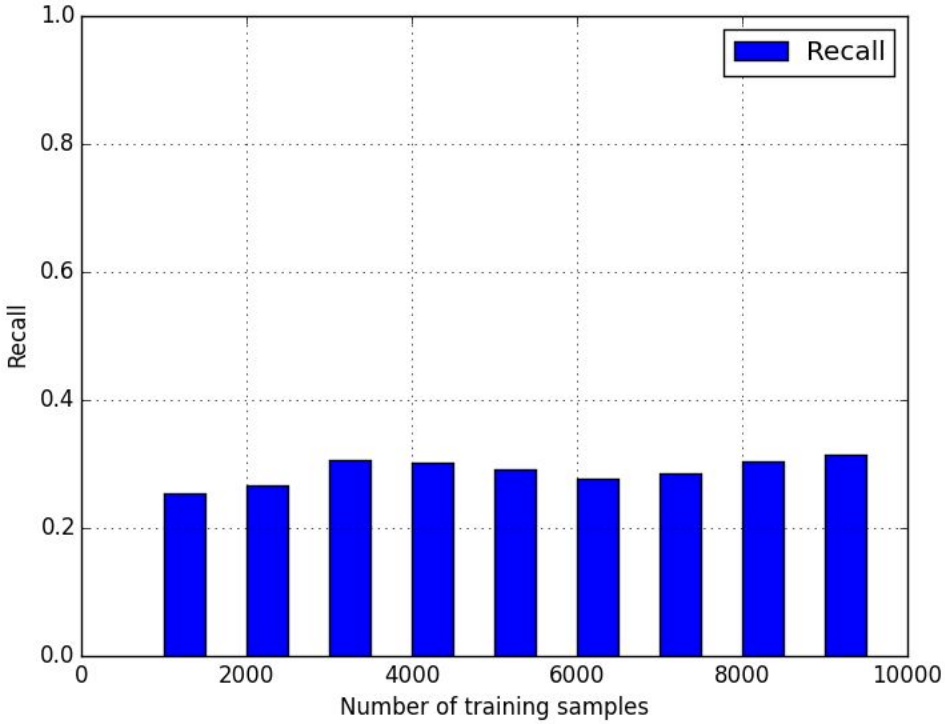
Analysis :

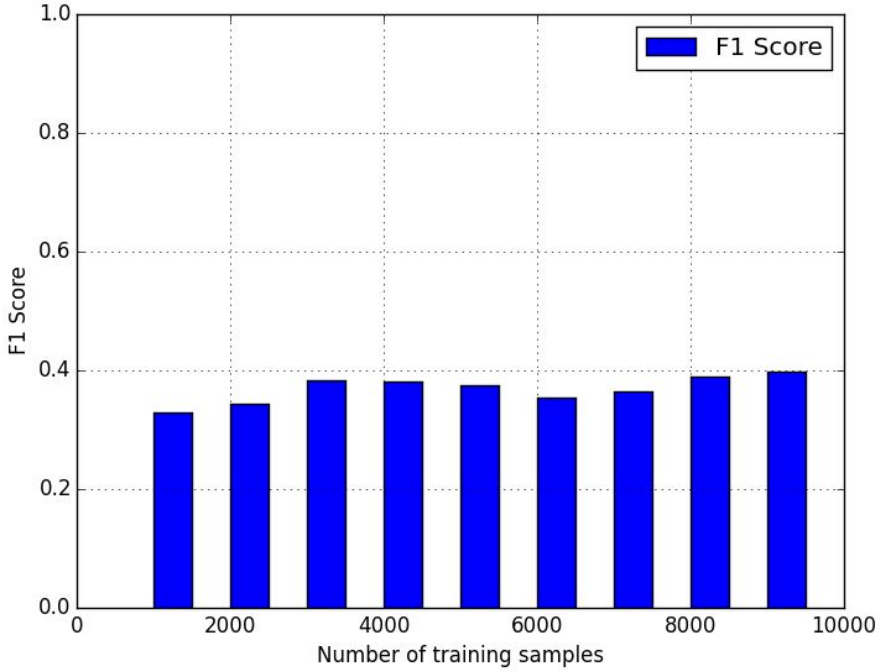
Training samples	At-least-one Accuracy	Set-equal Accuracy(%)	Precision	Recall	F1 Score
1000	0.094	46.5	0.4678362573	0.2540834846	0.3293149074
2000	0.098	47.6	0.4807370184	0.2672253259	0.3435068821
3000	0.152	51.2	0.5118110236	0.3053076562	0.382465431
4000	0.127	51.1	0.5194908512	0.3013382557	0.3814252336
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




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

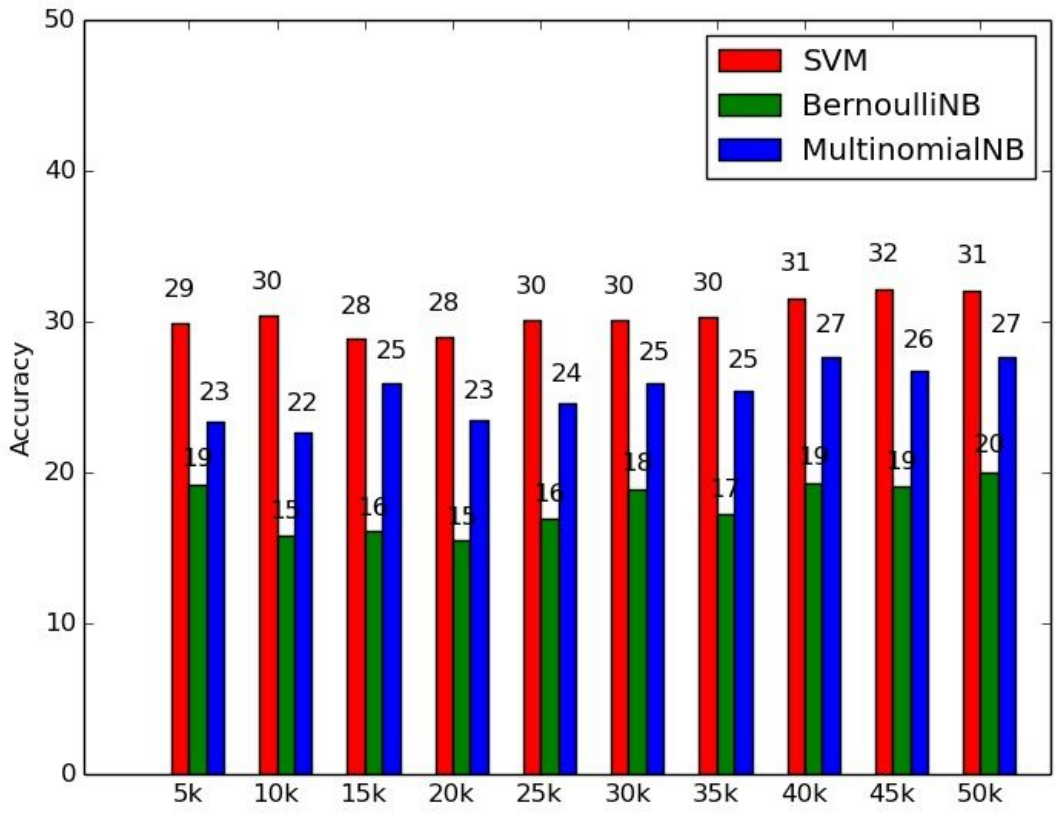

Recall
1.0
0.8
0.6
0.4
0.2
0.0
Recall
0
2000
4000
6000
8000
10000
Number of training samples


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

Analysis on code snippets

From the top 1000 tags we observe that most of them 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. For this we only consider the posts with code in it and who has at least one tag from the given list of programming languages tags.

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



Results:

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

Precision, Recall and F1-Score for SVM Classifier:

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

Negative Results

We tried BernoulliNB classifier of the sklearn library on the titles of the posts. It did not give us good accuracy.

Results obtained were as follows:

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

References

1. Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features.
2. Sanjay Sood, Sara Owsley, Kristian J Hammond, and Larry Birnbaum. 2007. Tagassist: Automatic tag suggestion for blog posts. In ICWSM.
3. Clayton Stanley and Michael D Byrne. 2013. Predicting tags for stackoverflow posts. In Proceedings of ICCM 2013
4. Kaggle (2013). Facebook recruiting III - keyword extraction. https://www.kaggle.com/c/ facebook-recruiting-iii-keyword-extraction.
5. Klassen, M. and Paturi, N. (2010). Web document classification by keywords using random forests. In Networked Digital Technologies, volume 88 of Communications in Computer and Information Science, pages 256–261. Springer Berlin Heidelberg.
6. Loper, E. and Bird, S. (2002). Nltk: The natural language toolkit. In Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, ETMTNLP ’02, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.
7. Manning, C. D., Raghavan, P., and Sch¨utze, H. (2008). Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA. [6] McCallum, A. K. (1999). Multi-label text classification with a mixture model trained by em. In AAAI 99 Workshop on Text Learning.