Udacity Machine Learning Nanodegree

Predicting Tags for the Questions in Stack Overflow

Debojyoti Sarkar

Jun 12nd, 2019

I. Definition

Project Overview

In Stack Overflow's current system, it allows users to manually assign between one and five tags to a posting. Users are encouraged to use existing tags but they are also allowed to create new ones, so the set of possible tags is infinite. Manual tagging can be challenging for users because 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 have a system that is able to either automatically tag questions or to suggest relevant tags to a user based on the question content.

Problem Statement

Suggest the tags based on the content that was there in the question posted on Stackoverflow.

The tasks involved are the following :

1. Download and loading the stack overflow data.
2. Analysis of Tags(Do statistical analysis of the data).
3. Cleaning and preprocessing of questions.
4. Designing machine learning model by converting tags to multilabel problem.
5. Split the data into train and test dataset.
6. Featurizing the data.
7. Run different machine learning model.

The final model is expected to predict as many tags as possible with high precision and recall.

Datasets and Inputs

Source: https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data

All of the data is in 2 files, Train and Test.

i. Train.csv contains 4 columns: Id, Title, Body, Tags.
ii. Test.csv contains the same columns but without the Tags, which we have to predict.
iii. Size of Train.csv: 6.75GB
iv. Size of Test.csv: 2GB
v. Number of rows in Train.csv = 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 (such as closed questions).

Data set contains 6,034,195 rows. The columns in the table are:
Id : Unique identifier for each question
Title: The question's title
Body: The body of the question
Tags: The tags associated with the question in a space separated format (all lowercase, should not contain tabs '\t' or ampersands '&')

Metrics

A data set which contains xi, yi and yi is a set in itself! Because our yi's represents tags in the questions and xi's represents questions and a question in the Stack Overflow can have multiple tags Q1 = {t1,t2,t3,t4,…etc}.
There could be questions like
Q1 = {t1,t2,t3,t4}
Q2 = {t1,t2,t3}
Q3 = {t1,t2}
So yi is a set of classes.
So it is a multi-label classification problem
Multi-label Classification: Multilabel 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 topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these.
Credit: http://scikit-learn.org/stable/modules/multiclass.html

For a standard Binary of 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. **I 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. So for Multi Label Setting we can modify it into two types

i. Micro Averaged F1 Score
Imagine we have t1,t2,t3,…,tn labels and 'A' is basically set of all labels. We know the general formula of precision and recall.

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

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

Formulas of f1 score and micro f1 score

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

F1 score Formula

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

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

$$Micro-F-Score = 2.\frac{Micro-Precision\,.Micro-Recall}{Micro-Precision + Micro-Recall}$$

Micro averaged f1 score

- Microaveraging Precision $Prc^{micro}(D) = \frac{\sum_{c_i \in C} TPs(c_i)}{\sum_{c_i \in C} TPs(c_i) + FPs(c_i)}$

- Microaveraging Recall $Rcl^{micro}(D) = \frac{\sum_{c_i \in C} TPs(c_i)}{\sum_{c_i \in C} TPs(c_i) + FNs(c_i)}$

Micro Precision and Micro Recall scores
Here ci belongs to capital 'C' and 'C' = set of class labels/tags.

Explanation:
For every label/Tag class 'ci' we can compute the True Positive, False Negative and False Positive rates and 'ci' could be any of the 'n' tags [t1,t2,t3,…,tn]. So here micro precision and micro recall are summing all the true positive rates for each of our label 'ci' and also summing each of the true positive and false positive rates. There is one advantage with this, Imagine we have three tags t1, t2 & t3 and assume t1 occurs in 90% , t3 occurs in 80% and t2 occurs only in 1% of our data points. Let's consider xi as one data point with having two tags t1 & t2. Because the occurrence rate of t2 is low, it's TP and FP will be low and therefore the contribution will be very low. So in Micro averaged f1-score we are giving weightages based on how frequently the label/tag occurs, we are taking that into considerations as we were using actual TP, FP and FN rates. The true positive value of t2 will be much smaller than t1 as the occurrence of t1 is very high when compare to t2. so here we're getting weighted averages based on the occurrence of the label. The good thing with micro averaged f1-score is, it is taking the tag/label frequency of occurrence into consideration when it is computing the micro precision and recall. Just want to add one more point here, let's say we have two tags t1,t2 adn t1 has high precision and recall rates when compare to t2 then also the score would not be affected as the occurrence rate, precision and recall are high compared to t2.

ii. Macro Averaged F1 Score

$$Macro - Precision = \frac{Pecision1 + Precision2}{2}$$

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

$$Macro - F - Score = 2.\frac{Macro - Precision\,.\,Macro - Recall}{Macro - Precision + Macro - Recall}$$

Explanation:
Let's say we have a set of tags like t1,t2,t3,…,tk. we compute precision and recall for each tag and get micro averaged f1 scores, then we sum up all the f1 scores and divide them by the number of labels/tags we have. So macro is the simple average of each f1 score for each of the label 'k' and hence it doen't take the frequency of occurrence of a tag/label into consideration. Simply, micro averaged f1 score is the weighted f1 where macro is non weighted simple mean average f1 score. It is not preferred when some tags occur lot of times and some occur few times.
Ref::
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
https://medium.com/@klintcho/explaining-precision-and-recall-c770eb9c69e9
https://sebastianraschka.com/faq/docs/multiclass-metric.html

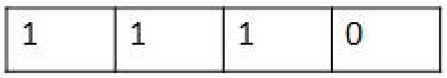
iii. Hamming Loss

It is an interesting metric that can be used in Multi-Label classification problems.

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

Hamming Loss Formula
|D| = number of samples,
|L| =number of labels,
yi = ground truth,

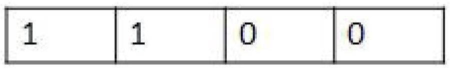
xi = prediction.
Note: xi is not a data point but a predicted one
Let’s say we have a set called ‘L’ with 4 labels
L = {t1,t2,t3,t4} and |L| = 4
We have a question ‘Q1’ with 3 original labels
Q —> t1,t2,t3 [yi]
If we represent the above question as binary vector representation then,



1	1	1	0

Vector Representation

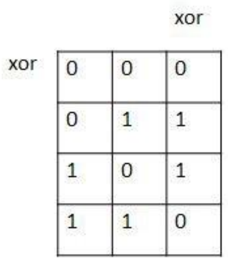
here t1,t2,t3 are present so we’ve set it to 1 and t4 is not present so we’ve set it to 0. If our model predicted t1,t2 and forget to predict t3 then the predicted yi binary vector would be



1	1	0	0

Binary Vector Representation of predicted yi

By formula we are summating all the data points and dividing with the number of points and having an xor function of (xi,yi)

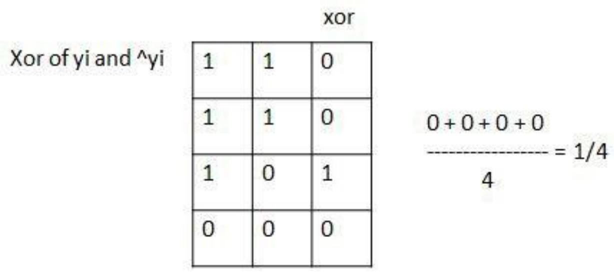


xor

xor

0	0	0
0	1	1
1	0	1
1	1	0

xor funtion of (xi,yi)
The xor function will return ‘1’ if both the values are different and otherwise returns ‘0’. Since we have two binary vectors, we can compare the xor function of both actual and predicted results (yi,^yi)



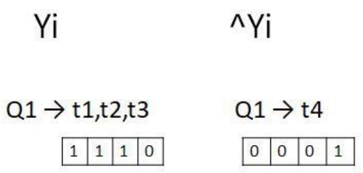
xor

Xor of yi and ^yi

1	1	0
1	1	0
1	0	1
0	0	0

$$\frac{0+0+0+0}{4} = 1/4$$

xor function of (yi,^yi)
The more dis agree of yi and ^yi tends to larger value of xor



Yi

Q1 → t1,t2,t3

1	1	1	0

^Yi

Q1 → t4

0	0	0	1

Second Example
The contribution to the error is more if the predicted and actual labels differs more. That’s why this is a loss. It gives good sense on how bad things are.

II. Analysis

Data Exploration

Due to hardware limitations, I am considering only 1M records.

Data contains duplicate records -

```
duplicate rows details:

 1    888456
2     53832
3      1293
Name: RecordCount, dtype: int64
```

After dropping duplicate records, checked for total number of unique tags -

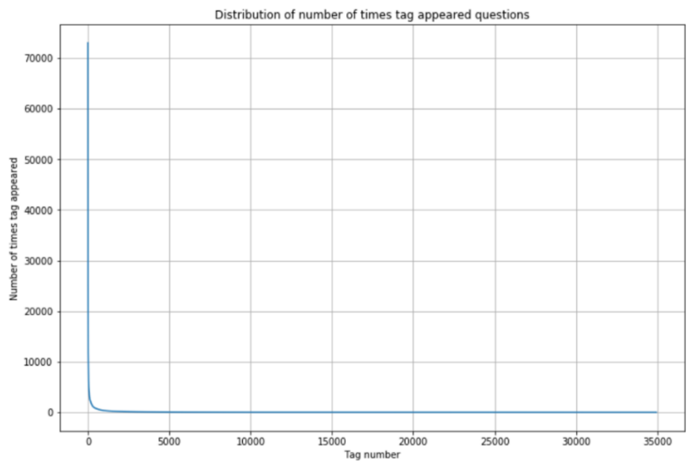
```
Number of data points : 943582
Number of unique tags : 34945
```

Calculated number of times a tag appeared according descending order –
Checked top 5 tags –

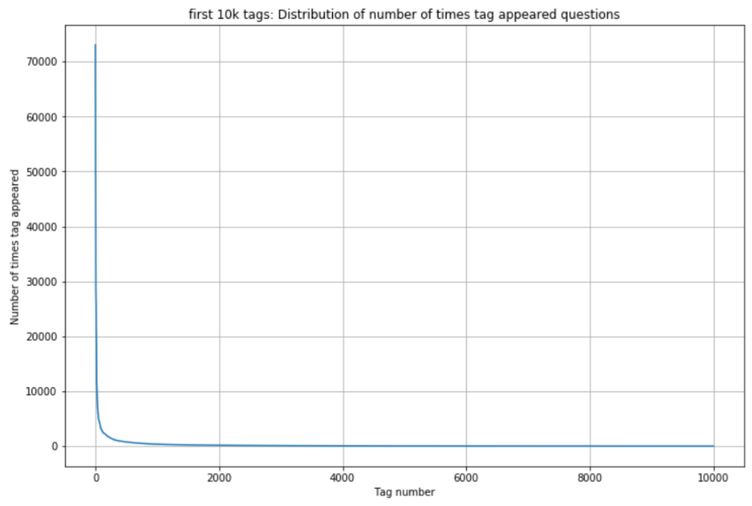
	Tags	Counts
3606	c#	73028
15026	java	64770
22632	php	61712
15100	javascript	57722
1020	android	50630

Exploratory Visualization

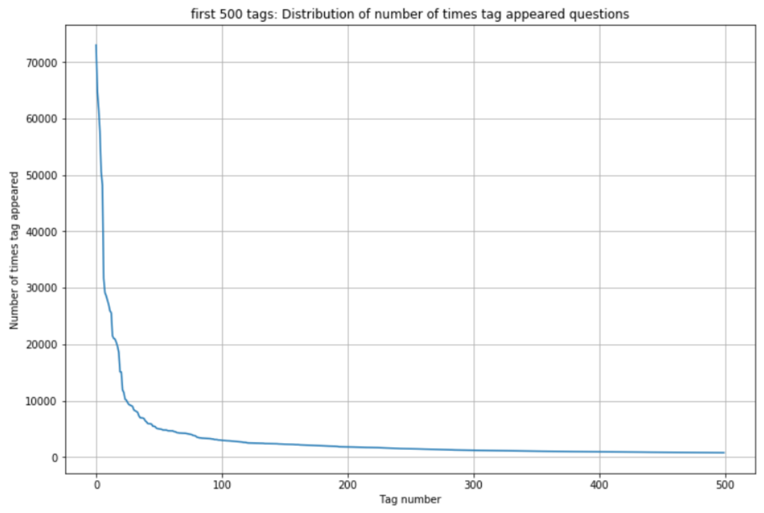
Plotted 'Tags' Vs 'Frequency of their appearances'



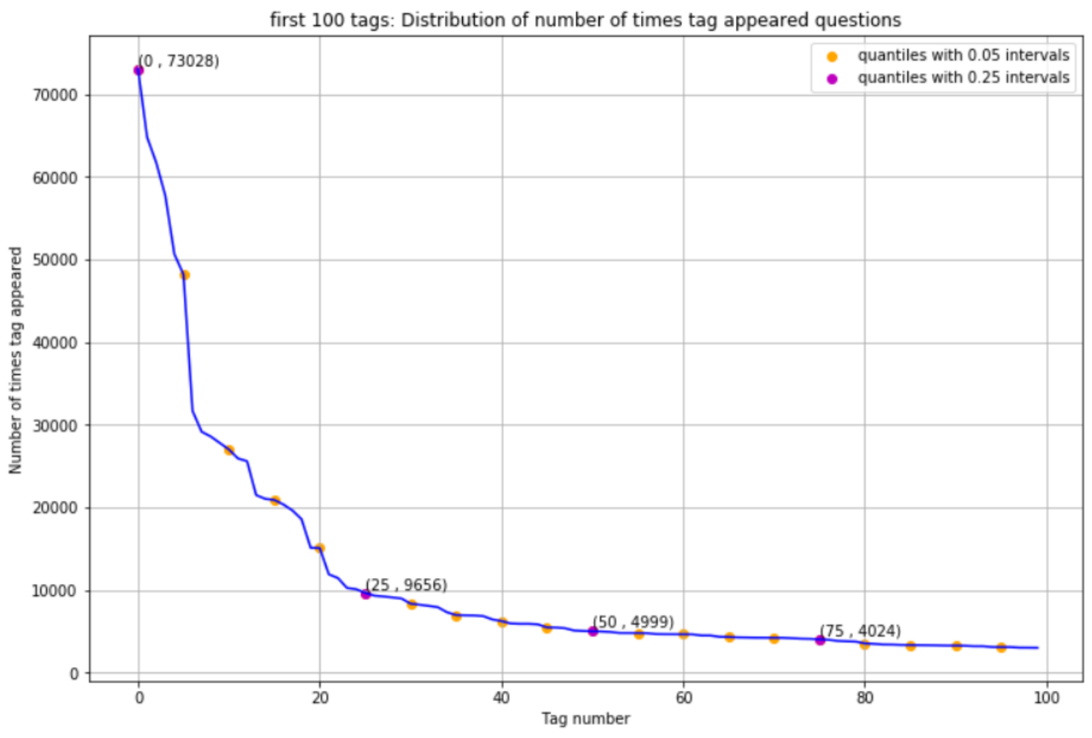
If we observe, the first tag occurs more than 70000 times and last tag occurs only one time. By looking at the diagram we can say that this is a **highly positively skewed distribution**. To understand better, lets **zoom the plot.**



Again zoom it.



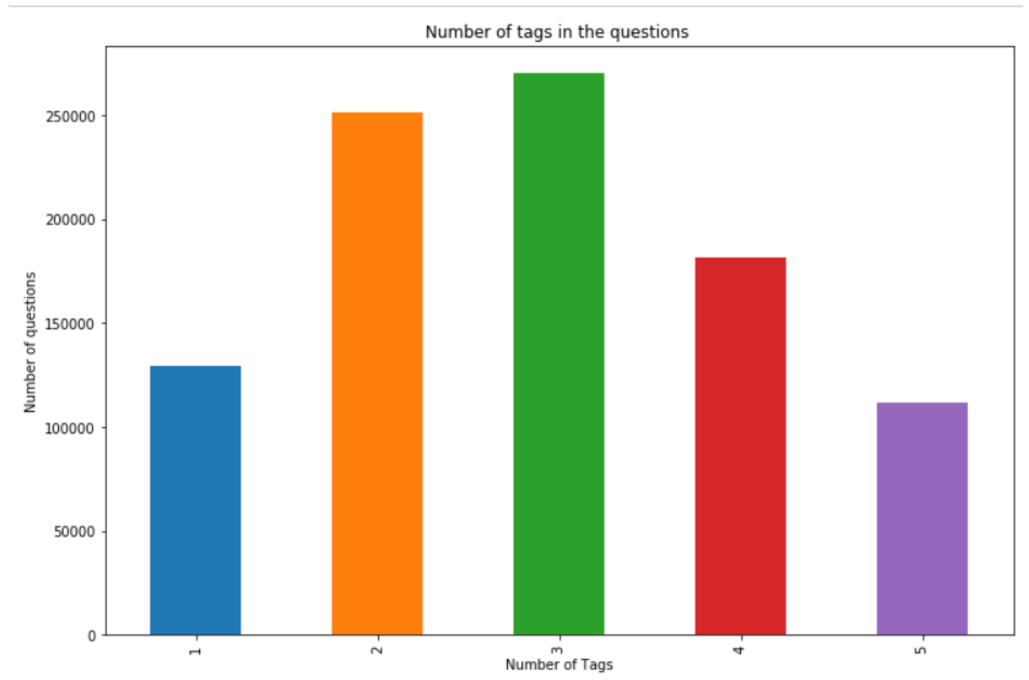
Zoom it again to find more details.



Observations:

1. There are total 25 tags which are used more than 10000 times.
2. 0 tags are used more than 100000 times.
3. Most frequent tag (i.e. c#) is used 73028 times.
4. Since some tags occur much more frequenctly than others, Micro-averaged F1-score is the appropriate metric for this probelm.

Tags per question:

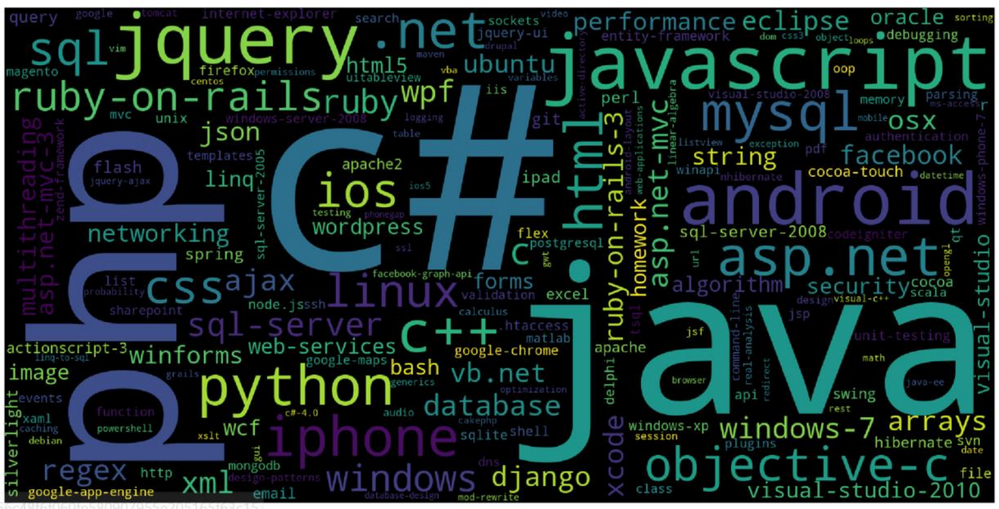


Observations:

1. Maximum number of tags per question: 5
2. Minimum number of tags per question: 1
3. Avg. number of tags per question: 2.887779
4. Most of the questions are having 2 or 3 tags

Most Frequent Records:

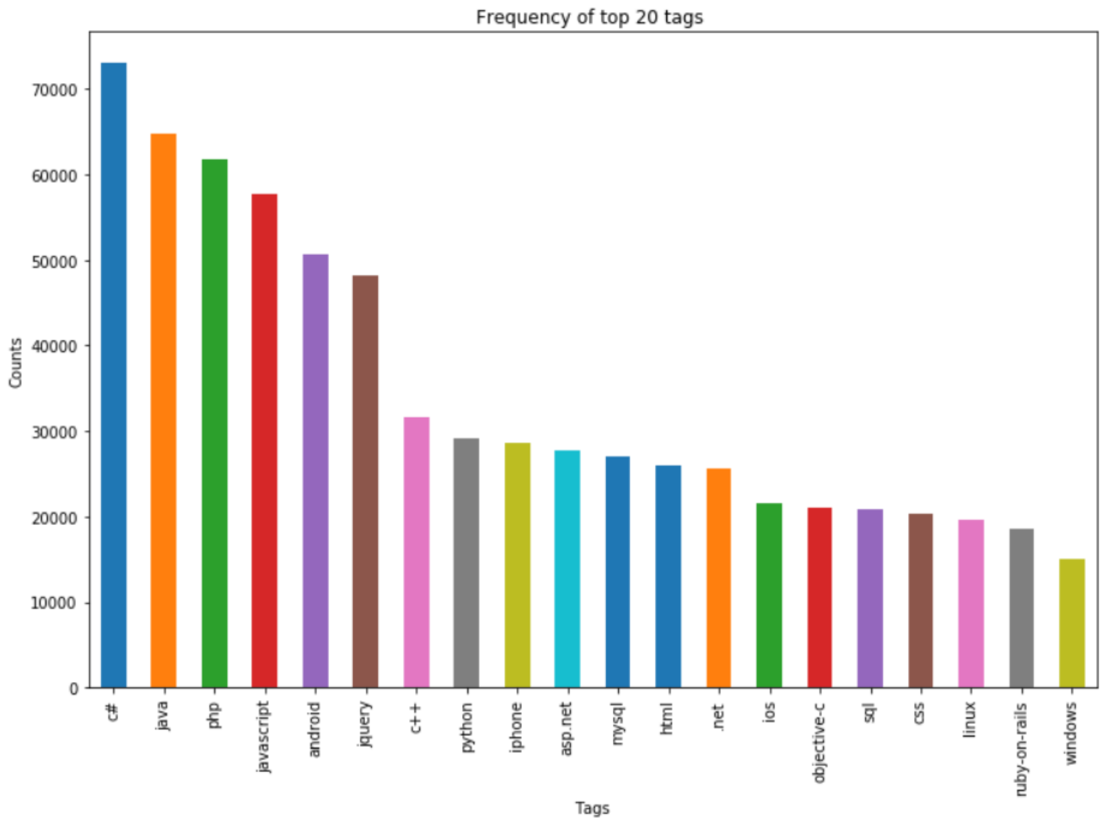
Plot the the data using WordCloud api



Observations:

A look at the word cloud shows that "c#", "java", "php", "asp.net", "javascript", "c++" are some of the most frequent tags.

The top 20 tags



Observations:

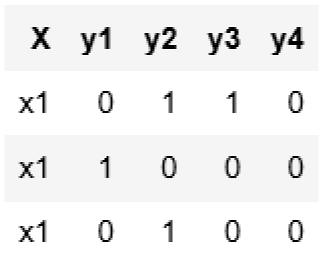
1. Majority of the most frequent tags are programming language.
2. C# is the top most frequent programming language.
3. Android, IOS, Linux and windows are among the top most frequent operating systems.

Algorithms and Techniques

We can solve Binary and Multi-Class classification problems with the help of algorithms like Logistic Regression, Support Vector Machines, Random Forest etc. So to solve our present Multi-Label classification we need to convert it into a binary or a multi class classification.
Imagine we have three questions with tags,
Q1-->{t1,t2}
Q2-->{t2,t3}
Q3-->{t3,t4}

The total set of labels are 4 (t1,t2,t3,t4). So now we can convert each question into a vector of size 4.
Q1-->{t1,t2} [1|1|0|0]
Q2-->{t2,t3} [0|1|1|0]
Q3-->{t3,t4} [0|0|1|1]
Binary Vector Representation

We can use CountVectorizer to convert all our tags into a binary vector.



X	y1	y2	y3	y4
x1	0	1	1	0
x1	1	0	0	0
x1	0	1	0	0

Here we are going to use **Problem Transformation(Binary Relevance)** method to solve the problem.

Binary Relevance:¶

Here I am going to convert multi-label classification problem into multiple single class classification problems.For example if I am having 5 multi-label classification problem, then I need to train 5 single class classification models. Basically in this method, I treat each label (in our case its tag) as a separate single class classification problem.

Ref :: https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/

I shall sample the number of tags instead considering all of them (due to limitation of computing power) Downscaling of data

Coming back to stackoverflow predictor problem, I need to train 30645 models literally!!! It is really huge. So I shall sample the number of tags instead considering all of them. Need to **down scale** the data.

But how many tags to be sampled with the minimal information loss ?

Plotting 'percentage of questions covered' Vs 'Number of tags' would help to solve this.

I shall sample the number of tags instead considering all of them (due to limitation of computing power)
Let's take **partial coverage** technique -

Q1 -->{t1,t2,t3} Q2 -->{t4,t5,t3}

Total tags C ={t1,t2,t3,t4,t5} Lets take subset of tags/labels to process - (Instead of taking 30645 lables we need to take subset)
C' is strict subset of C
I want C' to be as much small as possible.
If I take C'={t1,t2} , here t1 and t2 labels are present in Q1 but t1 and t2 are absent in Q2
So I can represent Q1 with respect to C' as Q1 -->t1,t2 and Q2--> Null Set. I can remove Q2 also in that case. Here Q1 is not completely taken care of as t3 is absent in C' but I am still ok as long as partial subset is present, I can count it. This is known as partial coverage

To find out smallest C'is to take those tags because for every tag I have count of tag i.e frequency that how many times tag occurs.
Let's take every tag has frequency(Number of time each tag is coming) like -
t1 = f1 t2=f2 t3=f3
If we take top tags by frequency they will cover most set of questions. If t1 and t2 are most frequent tags then using that we will cover most set of questions.

I am going to use the below list of algorithms on the given dataset and will check which performs better among them

I will reduce the size of the dataset while preserving the variance between rows

Algorithms to evaluate-------------
● Logistic Regression (LR)
● OneVsRest
● SGD Classifier

Below approaches(changing parameters) I will take to find best results from above algos -

1. Use bag of words upto 4 grams and compute the micro f1 score with Logistic regression(OvR)
2. Perform hyperparam tuning on alpha (or lambda) for Logistic regression to improve the performance using GridSearch
3. Try OneVsRestClassifier with Linear-SVM (SGDClassifier with loss-hinge)

Benchmark Model

i. Predict as many tags as possible with high precision and recall.
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 Recal rate, which means If the tag actually supposed to be present, we want it to be present most number of times.

ii. Incorrect tags could impact customer experience on Stack Overflow
For example, we have 4 actual tags t1,t2,t3 and t4. (i) Suppose if we predict t5, which turns out to be an incorrect tag or (ii) The given t4 is an inappropriate tag or (iii) If we didn't predicted an important tag.
So from the above mentioned cases, If we predict an incorrect tag then Precision will decrease and If we missed out an important tag then Recall will decrease. So both are impacting customer experience very badly. To understand it in a better way, lets say If we predict 'C#' tag to a question which is related to C language, People who get the message or notification from Stack Overflow may not be an expert in C language or If the question is actually related to 'C#' and If we missed out stating the tag then we could send the question to a wrong person, this will create a mess and impacts the business of Stack Overflow.

iii. No Strict Latency Constraints
We don't have a strict latency constraints. For example, let's say somebody posts a question with title and description. We don't have to provide all the tags or predict all the tags in milliseconds. We can have a couple of seconds to provide/predict right tags. Even 5 minutes should be fine. The most important thing is to increase the Rates of Precision and Recall.

III. Methodology

Data Preprocessing

The preprocessing done in the "Preprocessing" section in the notebook consists of the following steps –

1. Again, reducing the data size to 500k due to H/W limitation and it was taking more than 3 hours to clean up the data.

2. Check if questions containing HTML tags and what is the percentage of that.

3. Remove HTML tags based on below rule -
   3.1 Remove Spcial characters from Question title and Body (not in code)

   3.2 Remove stop words (Except 'C')

   3.3 Remove HTML Tags
    eg. <br> <b> abcd </b></br>
   Here br and b are html tags which does not says about actual class label that we need to predict.

   3.4 Convert all the characters into small letters

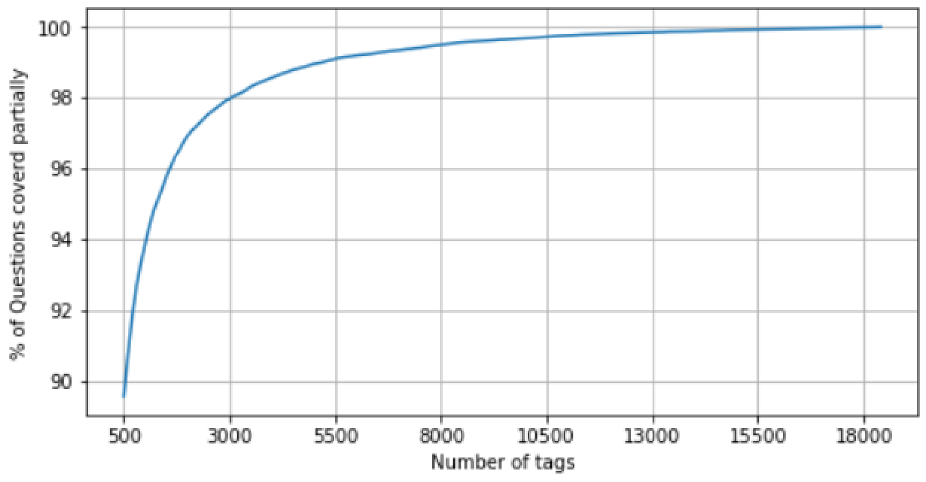
3.5 Use SnowballStemmer to stem the words.
Stemming is the process of reducing a word to its word stem.
eg: "python" is the stem word for the words ["python" "pythoner", "pythoning","pythoned"]

3.6 Give more weightage to title: Add title three times to the question. Title contains the information which is more specific to the question and also only after seeing the question title, a user decides whether to look into the question in detail. At least most of the users do this if not all.
That is repeat the title 3 times in the text because tf-idf simply counts.

Implementation

The implementation process can be split into multiple steps –

1. Converting the tags for multi label problem
2. Use Binary Relevance approach
   Sample the number of tags instead considering all of them (due to limitation of computing power) .
   But how many tags to be sampled with the minimal information loss ?
   Plotting 'percentage of questions covered' Vs 'Number of tags' would help to solve this.



3. Calculate how many tags covering how many questions -
   With 500 tags we are covering 89.566 % of questions
   With 600 tags we are covering 90.775 % of questions
   With 5500 tags we are covering 99.088 % of questions

By using only 600 tags (2% approximately) of the total 18487 tags I am loosing only 1.845% of the questions & also training 600 models is reasonable . So I shall choose 600 tags.

4. Split the data in training(80%) and testing(20%) data sets.

5. Featurizing question data with TfIdf vectorizer upto 1,2 and3 grams.

6. Featurizing question data with Count Vectorizer(Bag of Words).

7. Run Logistic Regression with OneVsRestClassifier using Tfidf vectorizer

8. Run SGDClassifier with OneVsRestClassifier using Count vectorizer(Bag Of Words)

Refinement

As mentioned in the Benchmark section we require high precision and recall, we need tweak the parameters.

I need to run Logistic and SGD classifier with using Log loss and Hinge Loss with different alpha values .

alphas = [10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3]

IV. Results

Model Evaluation and Validation

SGDClassifier with log loss is Logistic Regression-

accuracy : 0.22165

macro f1 score : 0.3278048410455833

micro f1 scoore : 0.46904172911287273

hamming loss : 0.0024504166666666667

Note ::
Micro f1 score(0.469..) is important even though we are calculating macro f1 score,hamming loss.

Micro-average quality numbers

Precision: 0.7162, Recall: 0.3487, F1-measure: 0.4690

Macro-average quality numbers

Precision: 0.5201, Recall: 0.2588, F1-measure: 0.3278

Note ::
We have printed precision and recall also. For class 0 precision(0.62) is very good but recall is slightly low. You can see precision is pretty high but recalls are not that high compared to precision except few cases like class/tag 4 0.96 (precision) and 0.76(recall). But ideally we want high precision and high recall value.

OneVsRestClassifier with proper LogisticRegression with penalty(regularization) l1 with parallelization –

Accuracy : 0.22335

Hamming loss 0.0024585833333333335

macro f1 score : 0.33893978485149223

micro f1 scoore : 0.4711112704587419

hamming loss : 0.0024585833333333335

Note ::

micro f1 scoore(0.471...) is better compared to previous one which is 0.469..

Micro-average quality numbers

Precision: 0.7089, Recall: 0.3528, F1-measure: 0.4711

Macro-average quality numbers

Precision: 0.5248, Recall: 0.2706, F1-measure: 0.3389

SGDClassifier with OneVsRestClassifier using Count vectorizer(Bag Of Words) –

alphas = [10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3]

Using Log loss

Accuracy : 0.158

Hamming loss 0.00300775

Micro-average quality numbers

Precision: 0.5224, Recall: 0.3605, F1-measure: 0.4266

Time taken to run this cell : 0:08:17.024258

Using Hinge loss

Accuracy : 0.15085

Hamming loss 0.003052416666666667

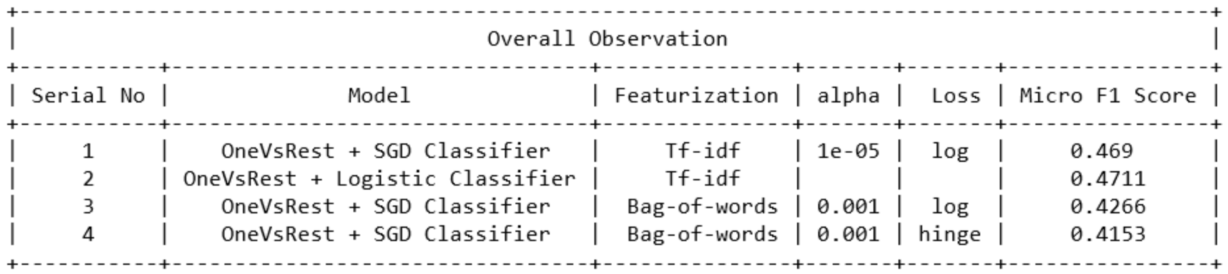
Micro-average quality numbers

Precision: 0.5121, Recall: 0.3492, F1-measure: 0.4153

Time taken to run this cell : 0:05:32.599905

Justification

After running different models, I can see the below results –



```
+-------------------------------------------------------------------------------------------+
|                                    Overall Observation                                    |
+-----------+--------------------------------+--------------+-------+-------+----------------+
| Serial No |             Model              | Featurization | alpha |  Loss | Micro F1 Score |
+-----------+--------------------------------+--------------+-------+-------+----------------+
|     1     |    OneVsRest + SGD Classifier   |    Tf-idf    | 1e-05 |  log  |     0.469      |
|     2     | OneVsRest + Logistic Classifier |    Tf-idf    |       |       |     0.4711     |
|     3     |    OneVsRest + SGD Classifier   | Bag-of-words | 0.001 |  log  |     0.4266     |
|     4     |    OneVsRest + SGD Classifier   | Bag-of-words | 0.001 | hinge |     0.4153     |
+-----------+--------------------------------+--------------+-------+-------+----------------+
```

V. Conclusion

- We have choosen 'f1_micro' scoring metric because of the stated business statement.
- Used bag of words upto 2 grams instead of higher number due to memory issue and Tfidf upto 3 grams.

- I have taken simplest model like Logistic Regression and SGDClassifier eventhough OneVsRest can take any model like RF,SVM and GBDT mostly because ¶

  High dimentional data: Since we are converting text to TfIdf or BOW vectors, the dimensions we get are very large in size. And when the dimensions are large linear model works very well but typically RF, SVM & GBDT won't work well. In linear model we can consider Logistic Regression and Linear SVM. But Linerar SVM has problem with time complexity as mentioned in the 2nd point below.

  Too many models to train: We have literally 600 models to train (after downscaling of data). And Logistic Regression is the simplest model one can use & it is comparitively faster.We can try with Linear SVM with SGD Classifier and hinge loss but performance will be similar Logistic Regression. If we start using other models like GBDT or RF, it will take too much time to train the model.

- We can see in the performance table that Logistic Regression with Tfidf vectorizer works better than SGDClassifier.

Improvement

1. To try with more data points (on a system more than 8GB RAM & highend processor)

2. Featurizing Text Data with Word2Vec: When you try Word2Vec, the dimentionality of data reduces & hence complex models like Random Forests or GBDT might work well

3. Try using scikit-multilearn library. Please note that this library doesn't take sparse matrix as input, you need to give dense matrix as input.So obviously you need to have more RAM to use this library