Team 39
Predict Closed Questions on StackOverflow

...

November 1, 2016

Rounak Mundra, Sourav Chanduka, Arihant Jain

Problem Statement

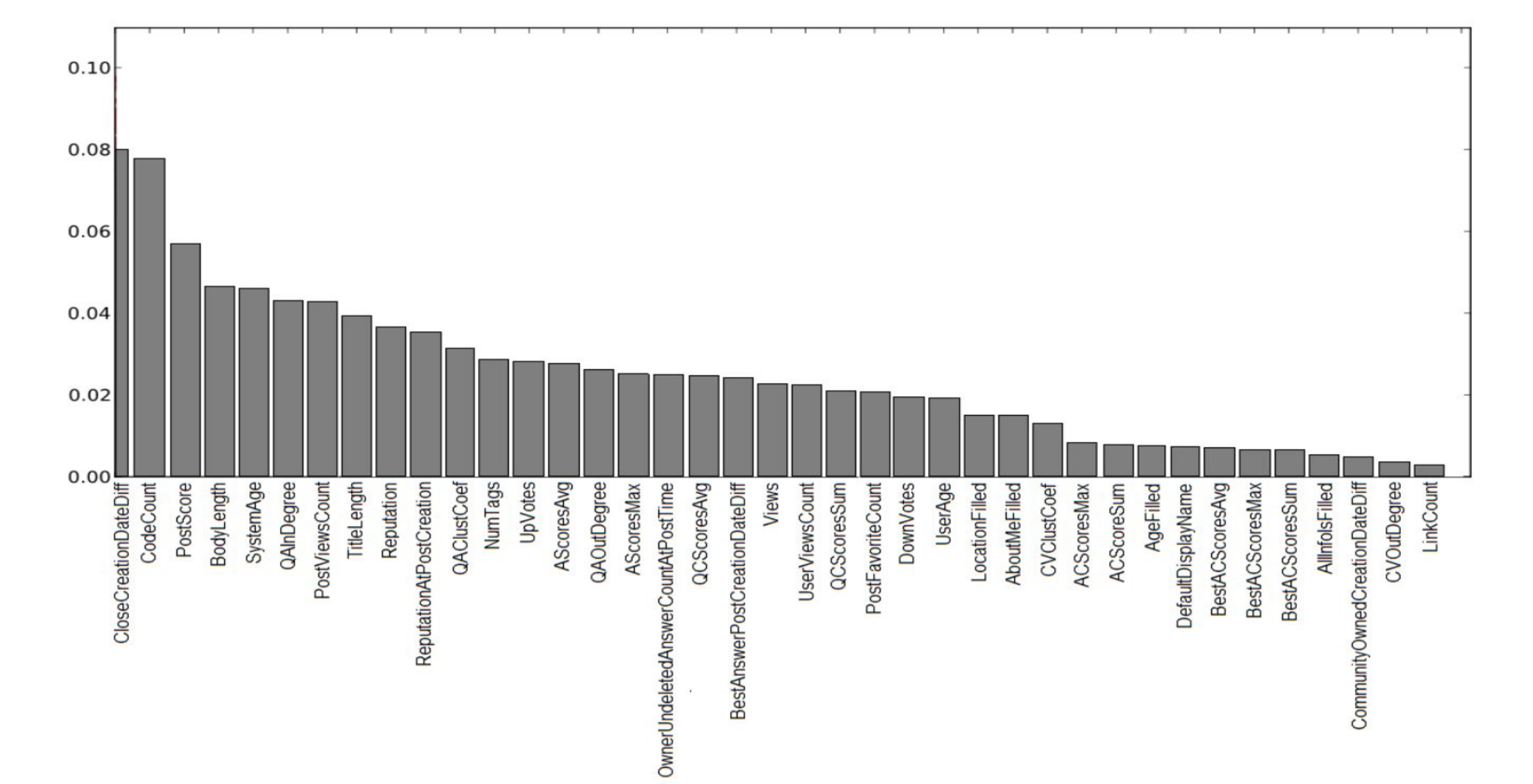
- ”Millions of programmers use StackOverflow to get high quality answers to their programming questions every day.
- More than six thousand new questions is asked on StackOverflow every weekday. Currently about 6% of all new questions end up ”closed”.

Goal:- It is to build a classifier that predicts whether or not a question will be closed given the question is submitted.

Dataset Information

- **Number of Instances** : 0.4 Million (135 MB in CSV Format)
- **Data Available in Dataset** :
  - PostId
  - PostCreationDate
  - OwnerCreationDate
  - ReputationAtPostCreation
  - OwnerUndeletedAnswerCountAtPostTime
  - Title
  - BodyMarkdown
  - 5 Tags
  - PostClosedDate
  - **Status (Open/Closed) (0/1)**
- **Source of Dataset :** Kaggle.com

Feature Selection

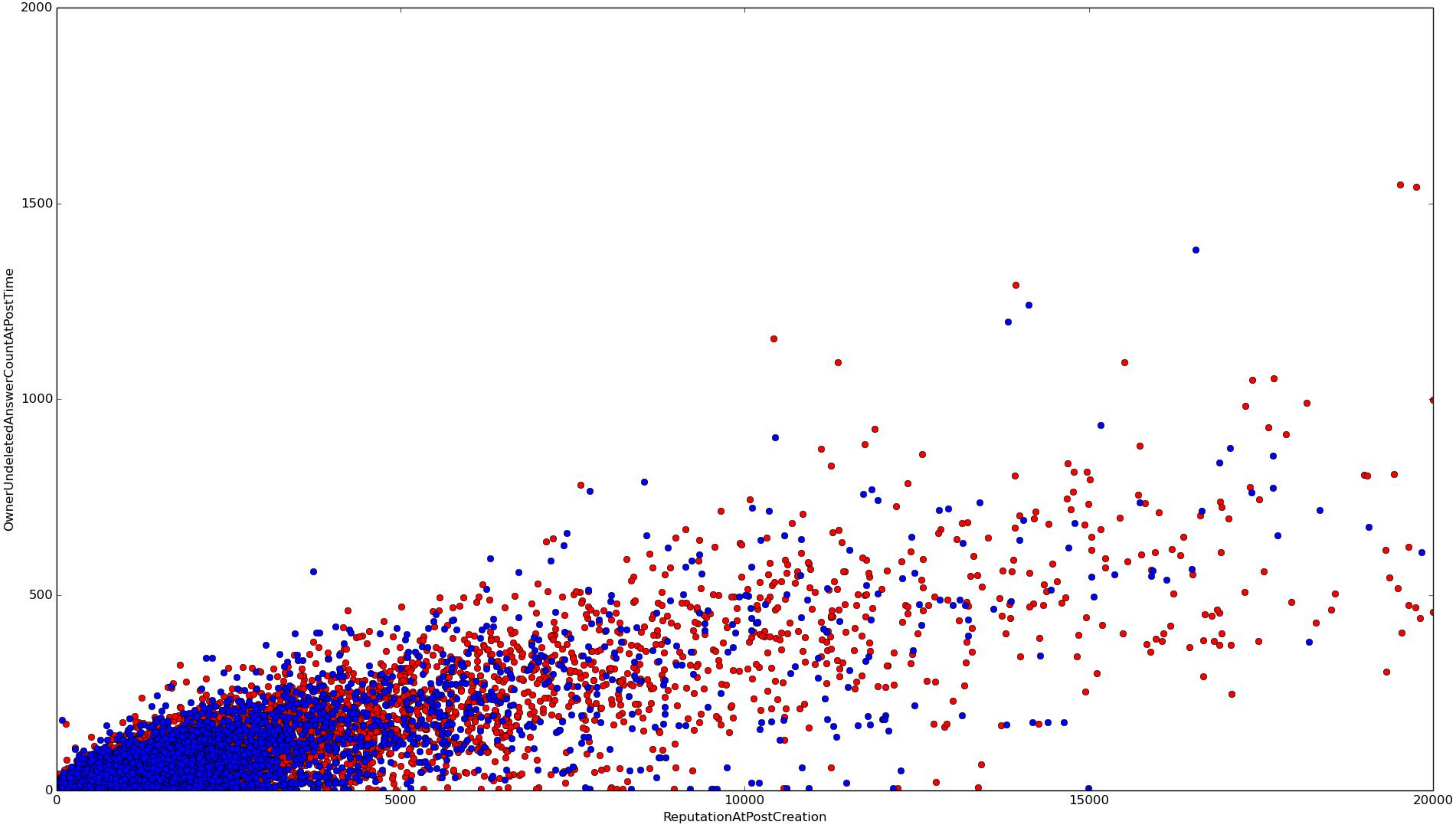


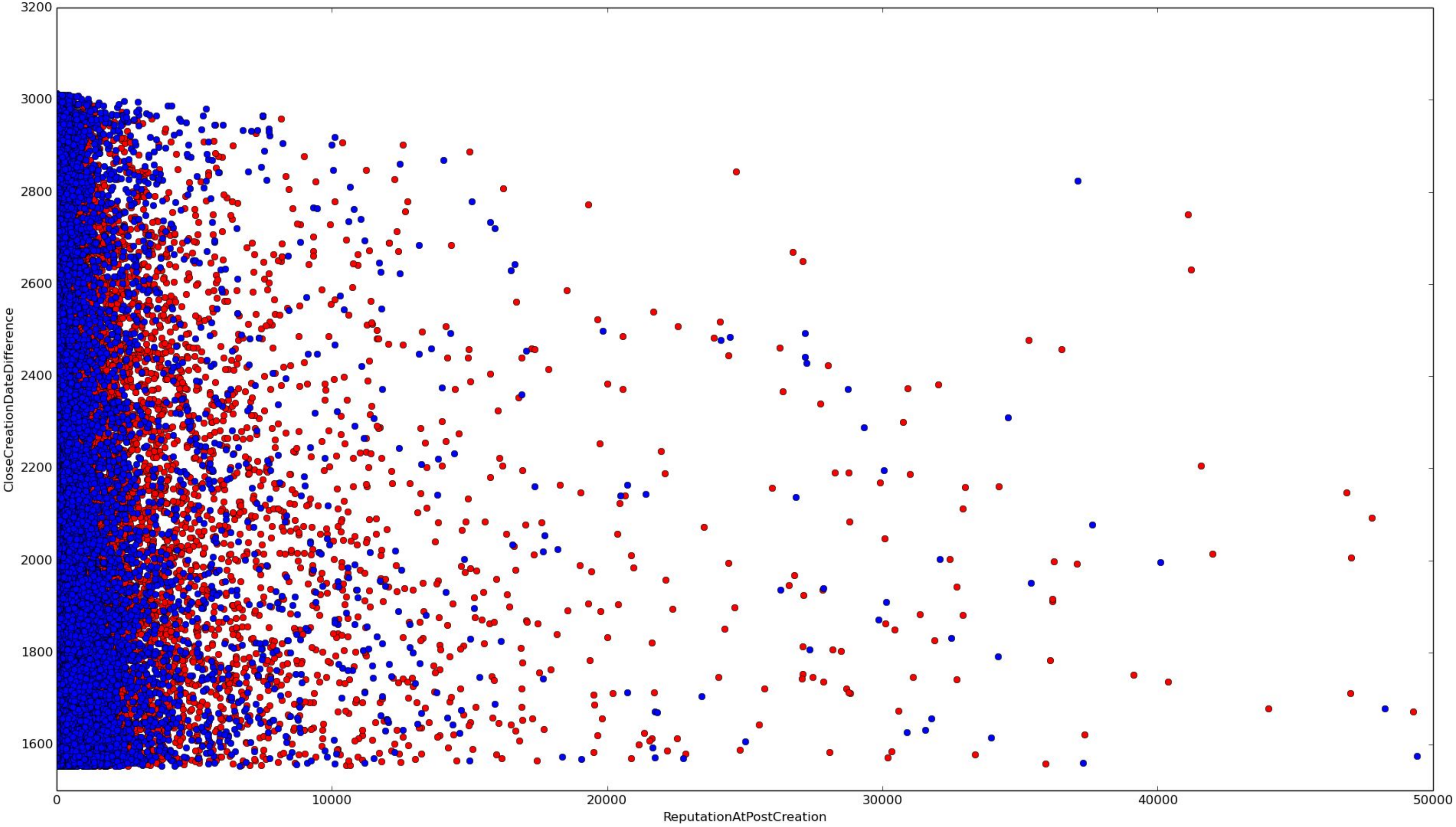
Features Extraction

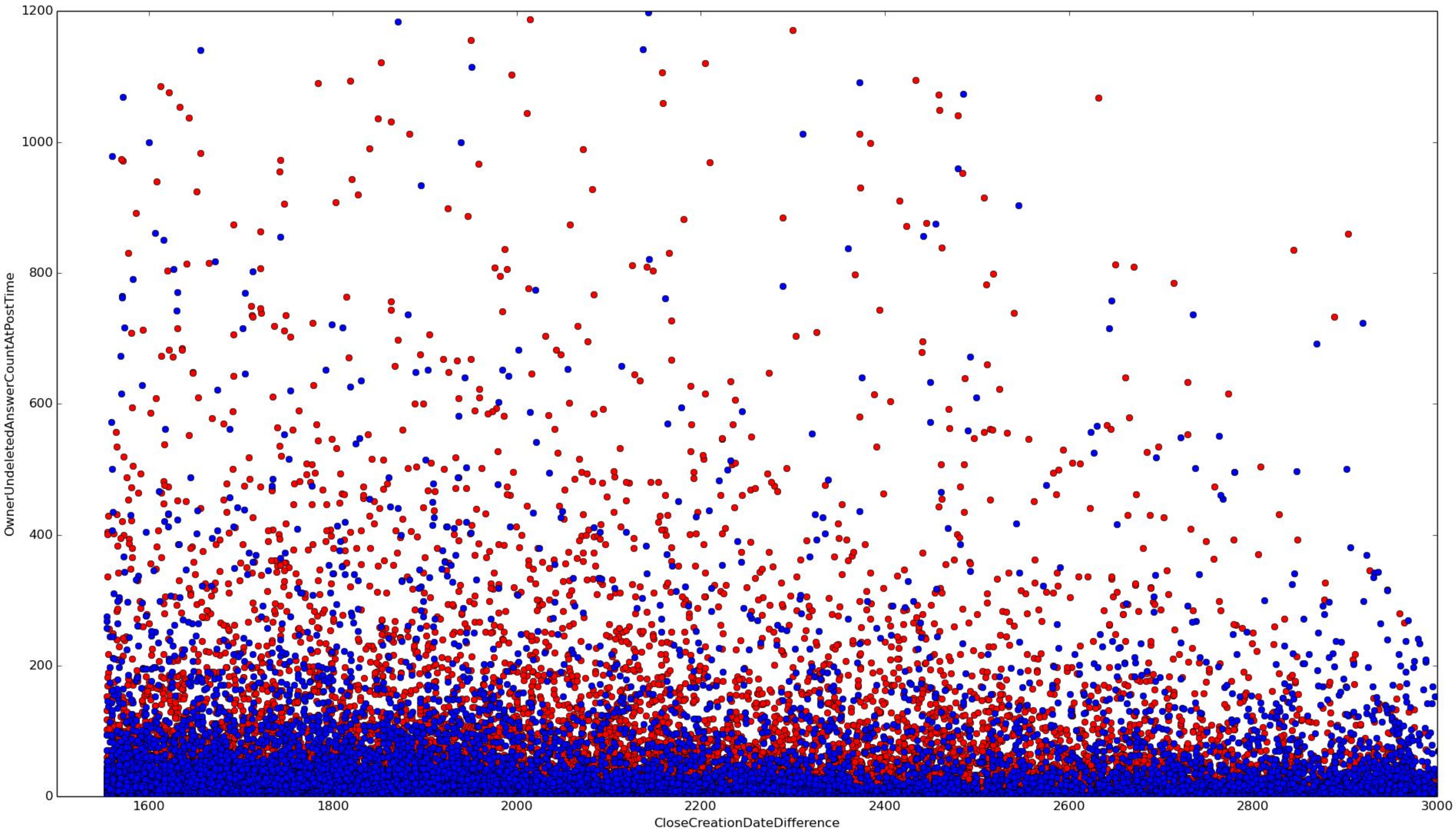
- Status: (0/1) (Open/Closed)
- TitleLength (After text processing)
- BodyLength
- ReputationAtPostCreation: Reputation of User when he posted a question
- NumberOfTags (atleast 1 and upto 5)
- CloseCreationDateDifference: Number of days it took to get deleted.
- UserAge: Profile Age
- OwnerUndeletedAnswerCountAtPostTime
- LinksCount

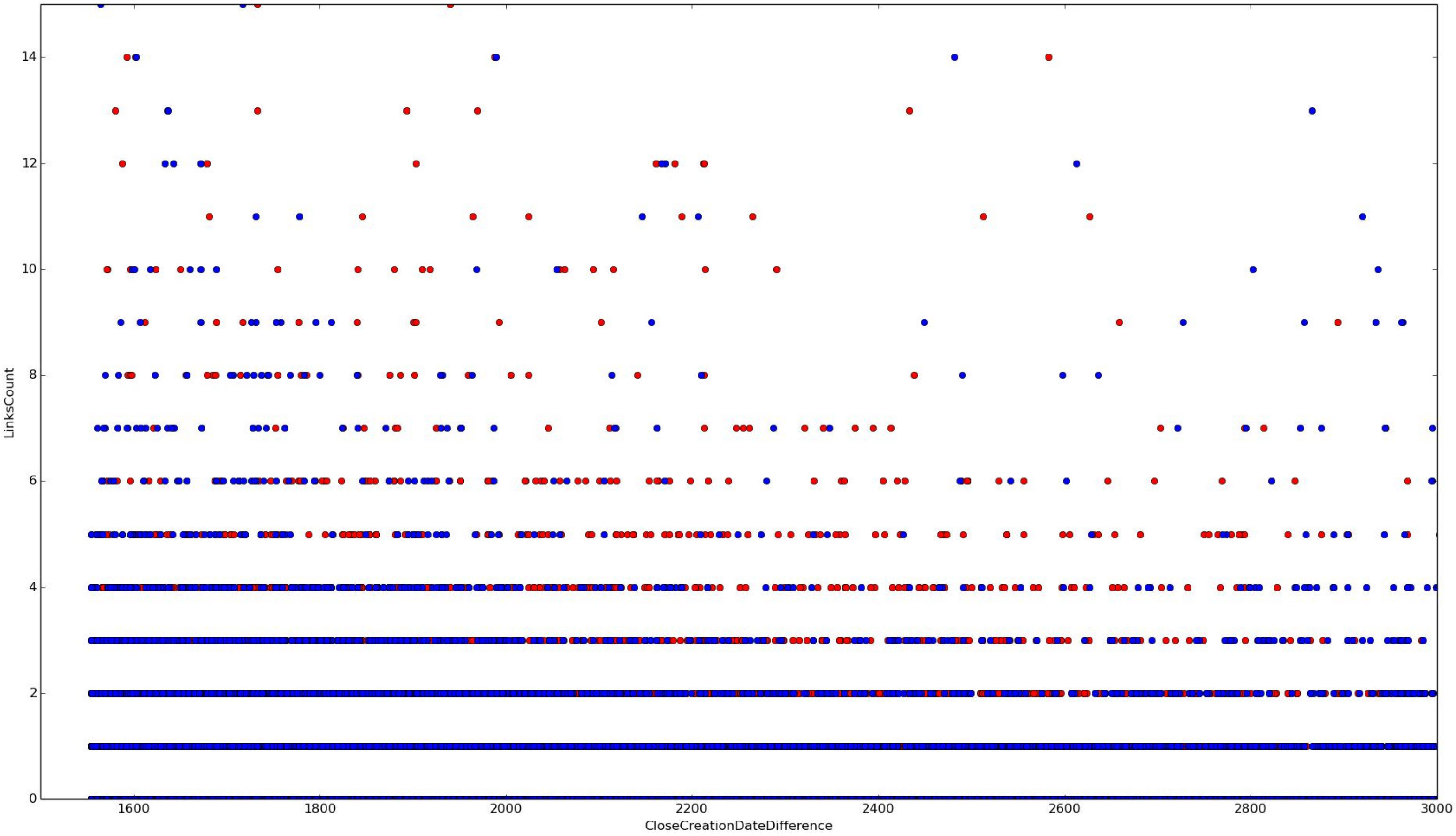
Text Processing (Preprocessing)

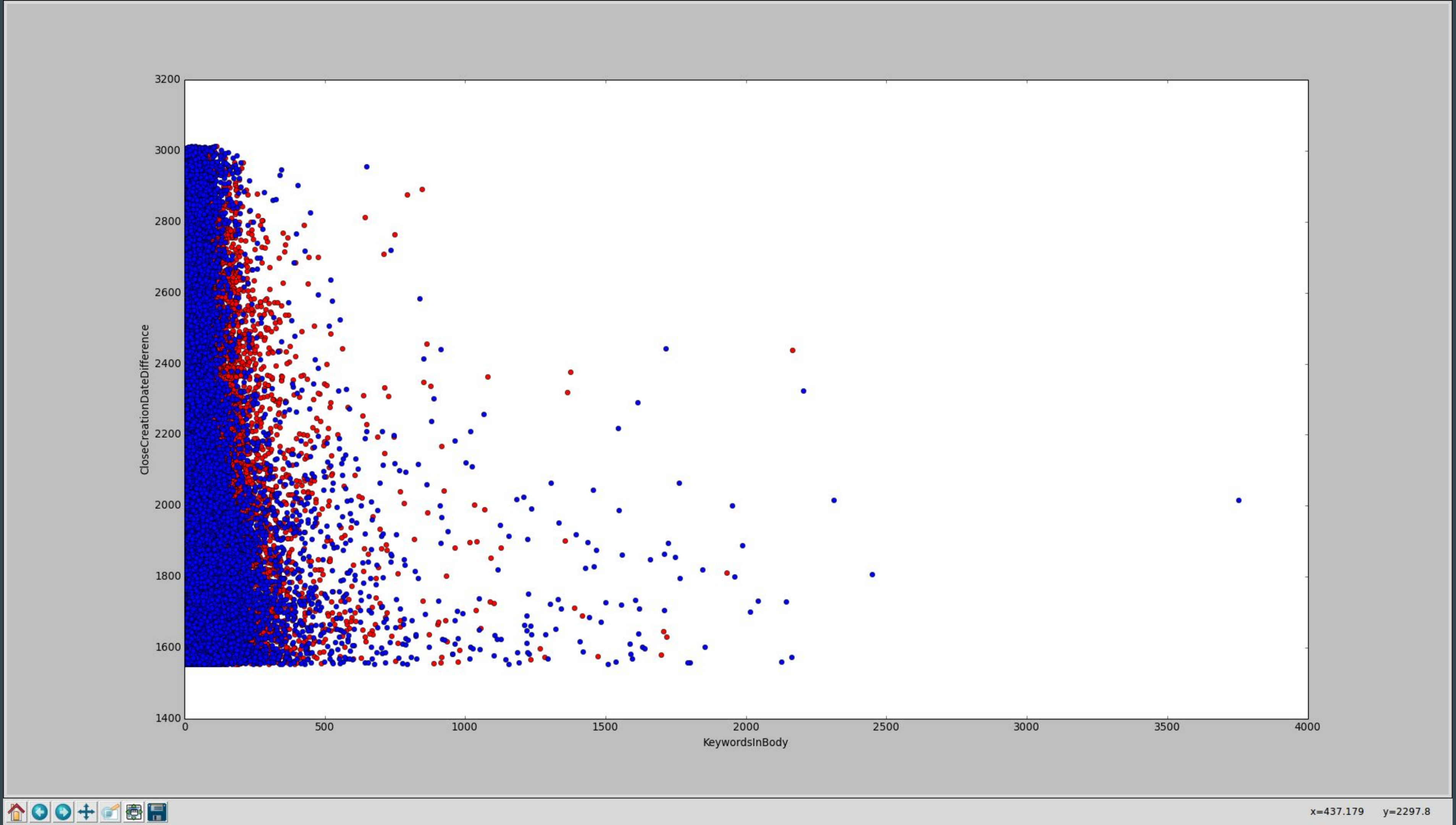
- Both Title and Body contains text.
- We need to re-present the text in numbers without losing the relevance of it.
- Natural Language Process Techniques:
  - **Tokenizing**: converting a document to its atomic elements.
  - **Stopping**: removing meaningless words.
  - **Stemming**: merging words that are equivalent in meaning.
- After these processing we got 10% more accurate prediction than when we took length straight away.

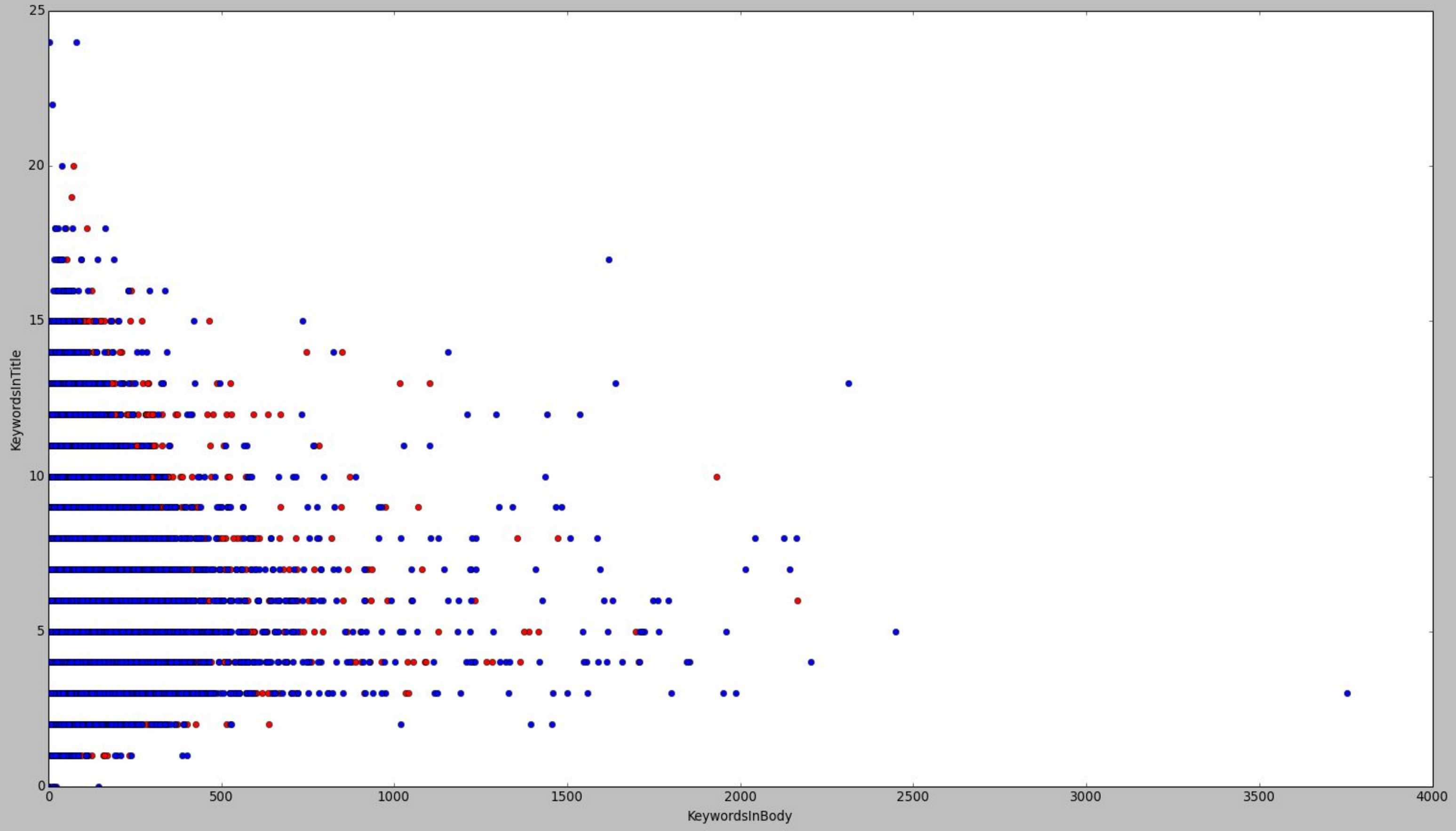

2000
1500
1000
500
0
OwnerUndeletedAnswerCountAtPostTime
0
5000
10000
15000
20000
ReputationAtPostCreation


3200
3000
2800
2600
2400
2200
2000
1800
1600
CloseCreationDateDifference
0
10000
20000
30000
40000
50000
ReputationAtPostCreation


1200
1000
800
600
400
200
0
OwnerUndeletedAnswerCountAtPostTime
1600
1800
2000
2200
2400
2600
2800
3000
CloseCreationDateDifference


LinksCount
0
2
4
6
8
10
12
14
CloseCreationDateDifference
1600
1800
2000
2200
2400
2600
2800
3000


CloseCreationDateDifference
3200
3000
2800
2600
2400
2200
2000
1800
1600
1400
0
500
1000
1500
2000
2500
3000
3500
4000
KeywordsInBody
x=437.179
y=2297.8



25
20
15
10
5
0
KeywordsInTitle
0
500
1000
1500
2000
2500
3000
3500
4000
KeywordsInBody

Algorithms Implemented

Random Forest

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

Support Vector Machine

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships.

Vowpal Wabbit

The Vowpal Wabbit (VW) project is a fast out-of-core learning system sponsored by Microsoft Research and (previously) Yahoo! Research.

Random Forest

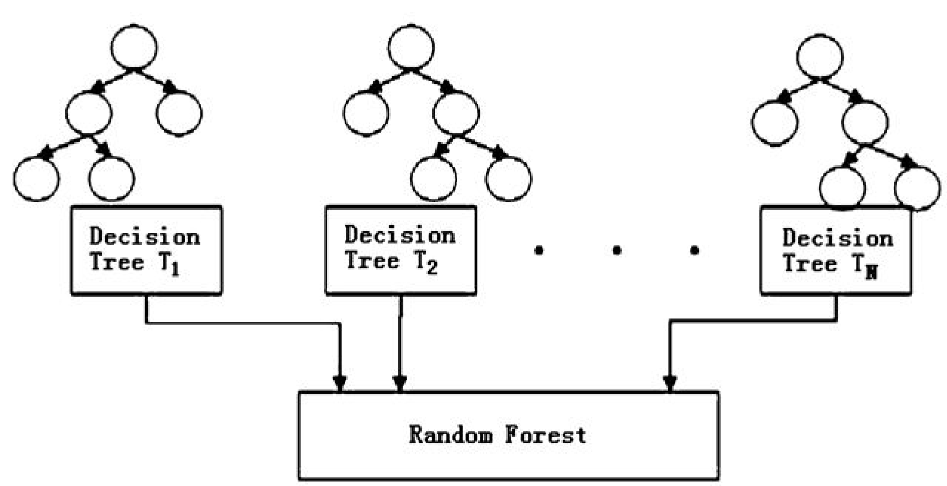
No. of Instances predicted: 520

Accuracy : 65 - 75 %

Precision : 0.65 - 0.70

F1 Score : 0.55 - 0.65

Recall : 0.75 - 0.85



Support Vector Machine (SVM)

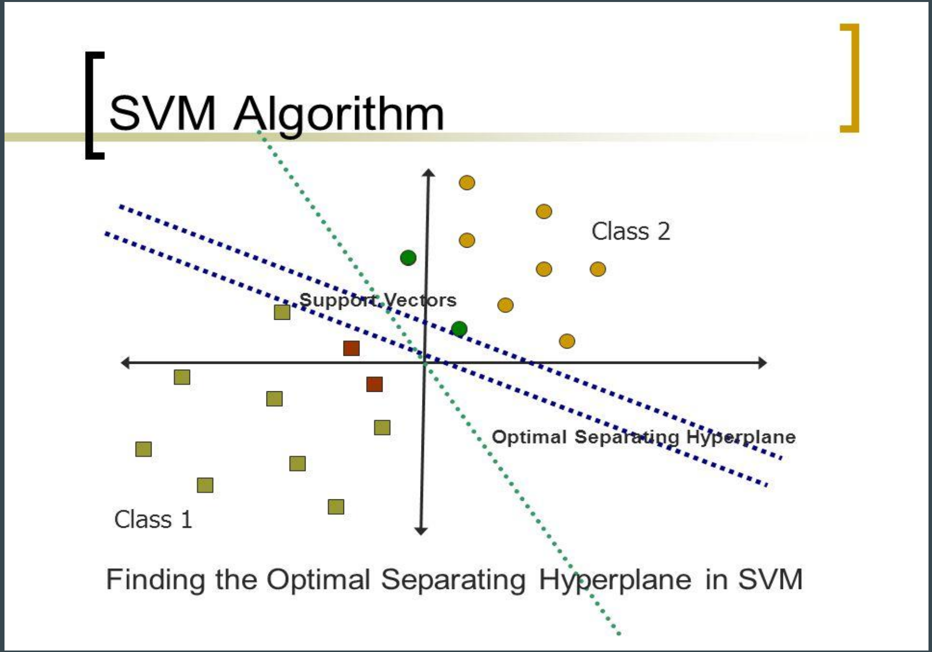
No. of Instances predicted: 520

Accuracy : Around 65 %

Precision : 0.60 - 0.65

F1 Score : 0.50 - 0.60

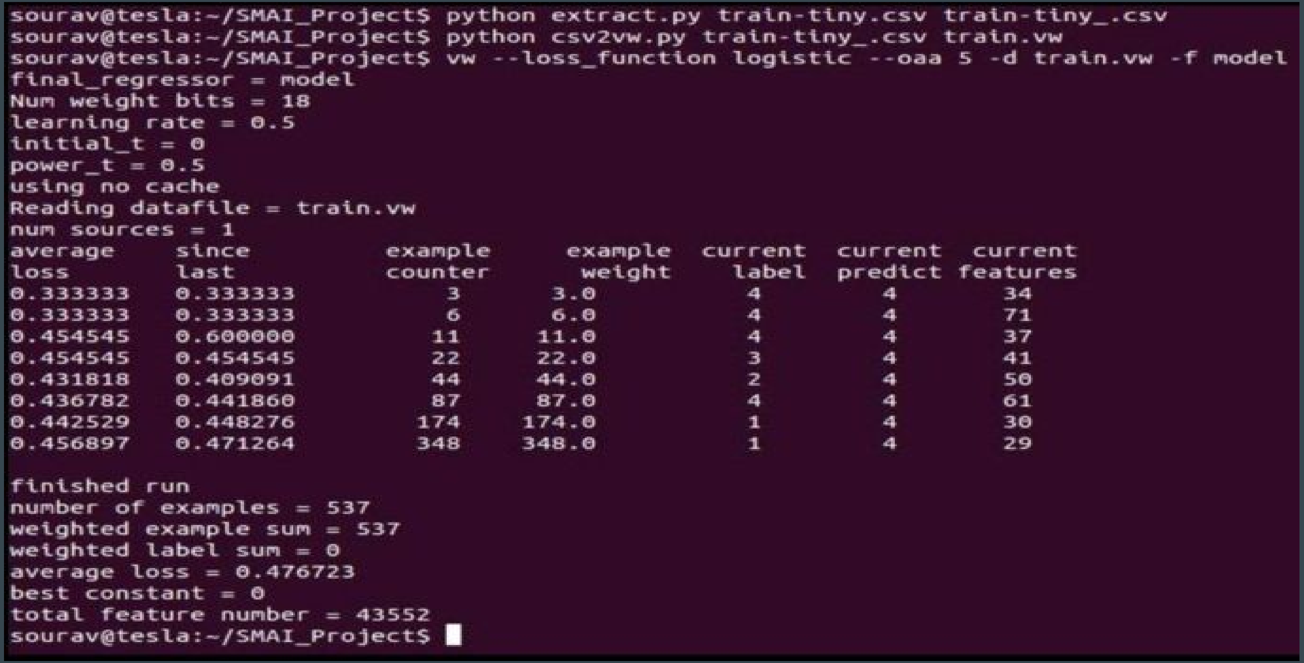
Recall : 0.77 - 0.82


SVM Algorithm
Class 2
Support Vectors
Optimal Separating Hyperplane
Class 1
Finding the Optimal Separating Hyperplane in SVM

Vowpal Wabbit

1. Vowpal Wabbit is a modified stochastic gradient descend algorithm.
2. Pre-processing CSV files to get CSV files slightly more fitting our purpose.
3. VW allows to induce sparsity in learned feature weights
4. For Text processing, it has it's own online Latent Dirichlet Allocation (LDA) implementation.
5. VW focuses on the approach to stream the examples to an online learning algorithm in contrast of parallelization of a batch learning algorithm over many machines.
6. **Average Loss** is the value of loss function as the learner goes along with example counter.
7. The main idea of truncate gradient is that it uses the simple rounding rule of weight to achieve the sparsity

Training Analysis

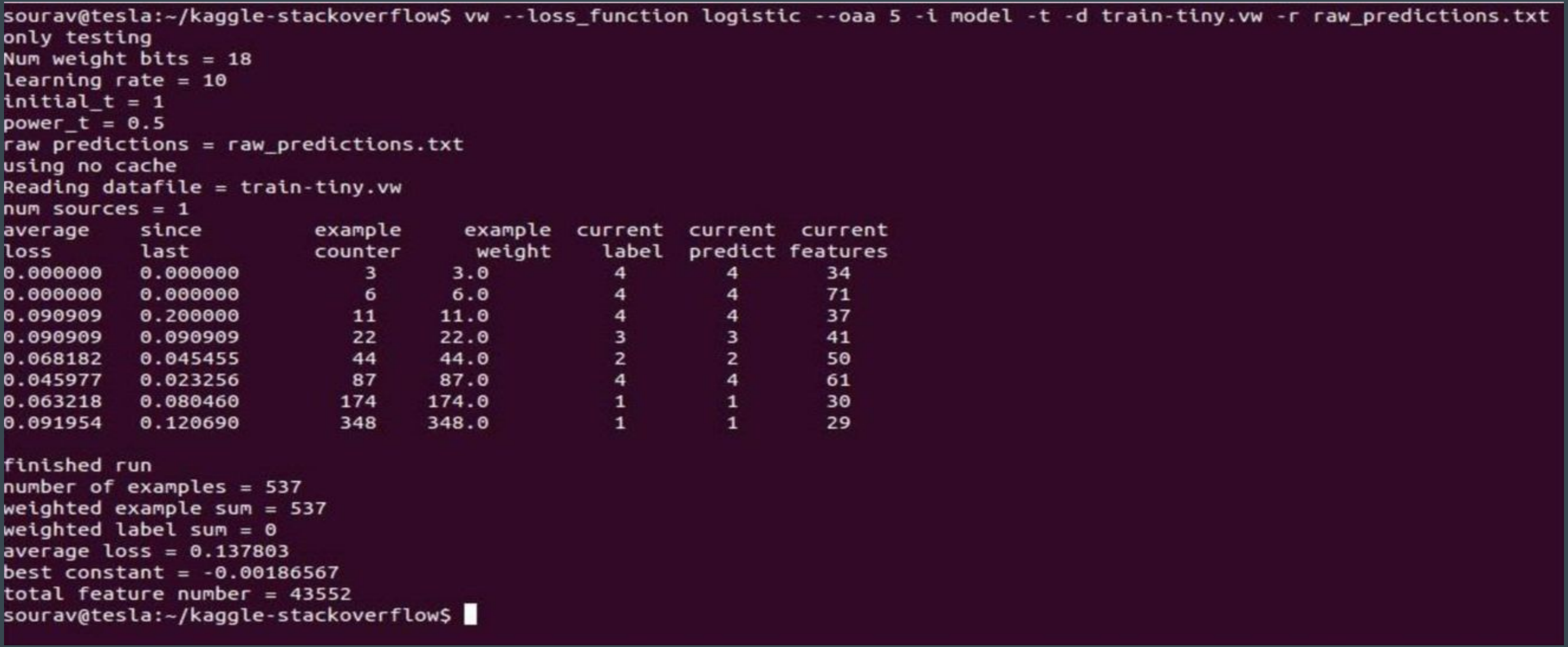


```
sourav@tesla:~/SMAI_Project$ python extract.py train-tiny.csv train-tiny_.csv
sourav@tesla:~/SMAI_Project$ python csv2vw.py train-tiny_.csv train.vw
sourav@tesla:~/SMAI_Project$ vw --loss_function logistic --oaa 5 -d train.vw -f model
final_regressor = model
Num weight bits = 18
learning rate = 0.5
initial_t = 0
power_t = 0.5
using no cache
Reading datafile = train.vw
num sources = 1
average    since          example     example  current  current  current
loss       last           counter      weight    label  predict features
0.333333   0.333333             3      3.0         4        4       34
0.333333   0.333333             6      6.0         4        4       71
0.454545   0.600000            11     11.0         4        4       37
0.454545   0.454545            22     22.0         3        4       41
0.431818   0.409091            44     44.0         2        4       50
0.436782   0.441860            87     87.0         4        4       61
0.442529   0.448276           174    174.0         1        4       30
0.456897   0.471264           348    348.0         1        4       29

finished run
number of examples = 537
weighted example sum = 537
weighted label sum = 0
average loss = 0.476723
best constant = 0
total feature number = 43552
sourav@tesla:~/SMAI_Project$ █
```

Results And Prediction

Efficiency : 1 - Average loss = 1 - 0.137 = 0.863



```
sourav@tesla:~/kaggle-stackoverflow$ vw --loss_function logistic --oaa 5 -i model -t -d train-tiny.vw -r raw_predictions.txt
only testing
Num weight bits = 18
learning rate = 10
initial_t = 1
power_t = 0.5
raw predictions = raw_predictions.txt
using no cache
Reading datafile = train-tiny.vw
num sources = 1
average    since         example     example  current  current  current
loss       last          counter      weight    label  predict features
0.000000   0.000000            3         3.0        4        4       34
0.000000   0.000000            6         6.0        4        4       71
0.090909   0.200000           11        11.0        4        4       37
0.090909   0.090909           22        22.0        3        3       41
0.068182   0.045455           44        44.0        2        2       50
0.045977   0.023256           87        87.0        4        4       61
0.063218   0.080460          174       174.0        1        1       30
0.091954   0.120690          348       348.0        1        1       29

finished run
number of examples = 537
weighted example sum = 537
weighted label sum = 0
average loss = 0.137803
best constant = -0.00186567
total feature number = 43552
sourav@tesla:~/kaggle-stackoverflow$ █
```

Thank You !