CS584 – MACHINE LEARNING

FALL 2016

Predicting Answer Quality in Stack Overflow

Group Members:

Mayuri Kadam

Chinnu Pittapally

Table of Contents

Task 2

Dataset 2

- Data source 2
- Target variable 2
- Features 2
- Data size 2

Preprocessing 3

Visualization 3

- Target 3
- Features 4

Evaluation 5

- Performance Measure 5
- Classifiers 5
- Evaluation Strategy 11
- Performance Results 11
- Top Features 12
- Discussion 12

Interesting/Unexpected Results 13

Contributions of Each Group Member 13

Conclusion 14

References 14

Predicting Answer Quality in Stack Overflow

Group Members: Mayuri Kadam and ChinnuPittapally

Task

To predict the quality of an answer as good or bad (i.e. if the answer gets any up-votes or not) on stack overflow.

For simplicity we are dealing with questions and answers related to python only. Any answer on stack overflow that gets 1 or more up-votes we classify it as a GOOD (represented as 1) and answers having 0 or negative votes are classified as BAD (represented as 0) answers. Relevance of question to answers is established initially, to go forward with class prediction process. We have used different models to predict the classes and finally consider one which predicts with the highest precision.

Dataset

- We have three csv files which hold data for questions, answers and tags for all the questions related to python.
    1. **Questions.csv:** title, body, creation date, score, and owner ID for each Python question.
    2. **Answers.csv:** body, creation date, score, and owner ID for every answer to each of these questions. The ParentId column links back to the Questions table.
    3. **Tags.csv:** tags on each question.

Data source

- We have collected the dataset from Kaggle for stack overflow questions on python. The dataset contains questions up to 19 October 2016 (UTC) containing ~1M answers.
- https://www.kaggle.com/stackoverflow/pythonquestions
- For analysis we pre-processed the data and limited our scope to 100,000 answers and corresponding questions only.
- We created the target class by going through the score values and added a Label for each answer as 1 for Good and 0 Bad answer based on the up-vote counts

Target variable

The target here is to predict the class and this target variable can take two values – Good and Bad. We marked Good as 1 and Bad as 0 in our dataset. Any answer that has 1 or more up votes should be classified as good answer and others as bad answers.

Features

Input Features: We have usedsklearn'sTf-idfvectorizer for tokenizing the dataset for 100,000 samples and we got 229608 features for the best model for this analysis.

X - Matrix: We usedsklearn'sTf-idfvectorizer and CountVectorizer to create the vocabulary for different models i.e. to generate features for our dataset. We used fit-transform to fit the input data to our vocabulary.Our best model we uses the tf-idfVectorizer.

Feature Processing: For the input matrix we considered the 'question+answers'instance from the merged table and tokenized the training set. The tokenize function flattens the text to lowercase. Also removes stop words and words of length less than 2.

Data size

Raw data: 1M instances

Processed data: 100,000 instances

The raw data contained about 1M instances answer related to python from stack overflow. We have pre-processed the data prior to the analysis and reduced the size to 100,000 question-answer instances which helped to reduce the time complexity.

Preprocessing

Merging Data: From the join of answers and questions table we mapped all the answers to their corresponding question using the "ParentId" from answer table and "Id" from questions table. We stored this data.

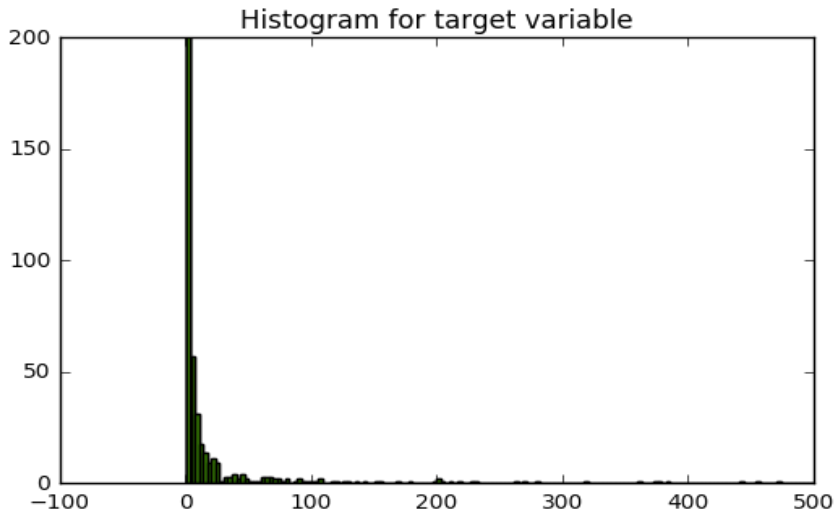
We then picked 100,000 answers from the raw data set at random and stored them in "Question_answer.csv" at shared location available for download from: https://drive.google.com/open?id=0B6hJIF_NL2HsMk80Y0FOM3pWb3c

Visualization

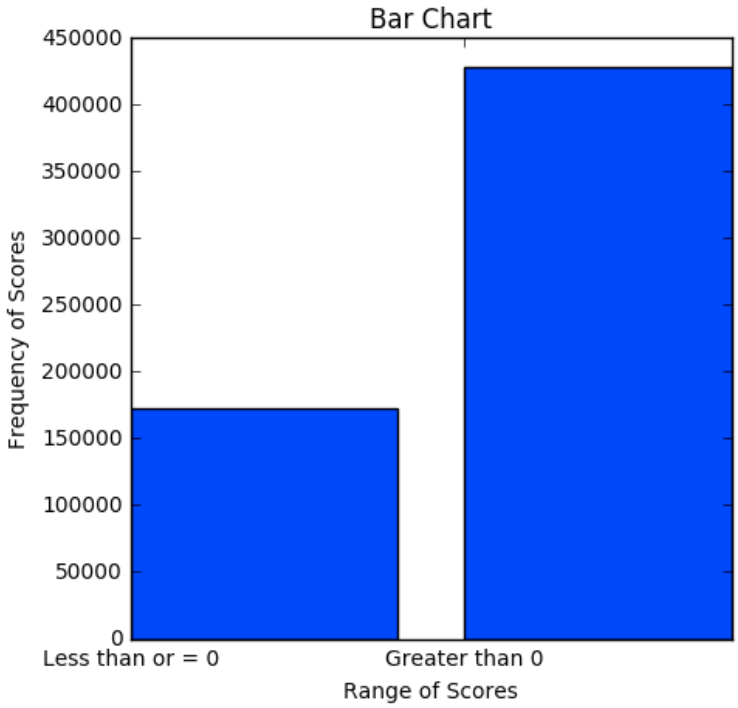
Target

Establishing Target Class : From the merged table we extracted scores for every answer and categorized each of them as 'Good' if they had score greater than 0 and 'Bad' if score was 0 or less. This we call as our target variable/class

- The target class (score/up votes) is visualized using histogram plot. From the histogram figure, we can observe most of our data lies in 0 to 100, while there many score values (higher score/up votes) that are less frequent.



- Bar plot to show the range of values in the score/up votes and their frequency. From the bar plot, we can observe our data is skewed ("Less than or = 0" is Bad answer and "Greater than 0" is Good answer).



Features

10 most frequent words.

1. ('code', 3434581)
2. ('pre', 1335861)
3. ('python', 415808)
4. ('self', 357360)
5. ('http', 354788)
6. ('href', 325781)
7. ('use', 271458)
8. ('import', 247609)
9. ('rel', 239210)
10. ('nofollow', 237969)

Evaluation

Performance Measure

Precision Score

- The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
- We are doing the evaluation of different classification models based on the precision score as the performance measure. Since our data is skewed and has more instances in positive class we ignored the accuracy measure and considered precision to find how precisely an instance can be predicted as positive when it is ctually positive and how often it fails.
- We are also printing the Classification Report for each setting and the Matthews correlation coefficient (MCC).The MCC is used in machine learning as a measure of the quality of binary (two-class) classifications. It takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The MCC is in essence a correlation coefficient value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction. The statistic is also known as the phi coefficient.

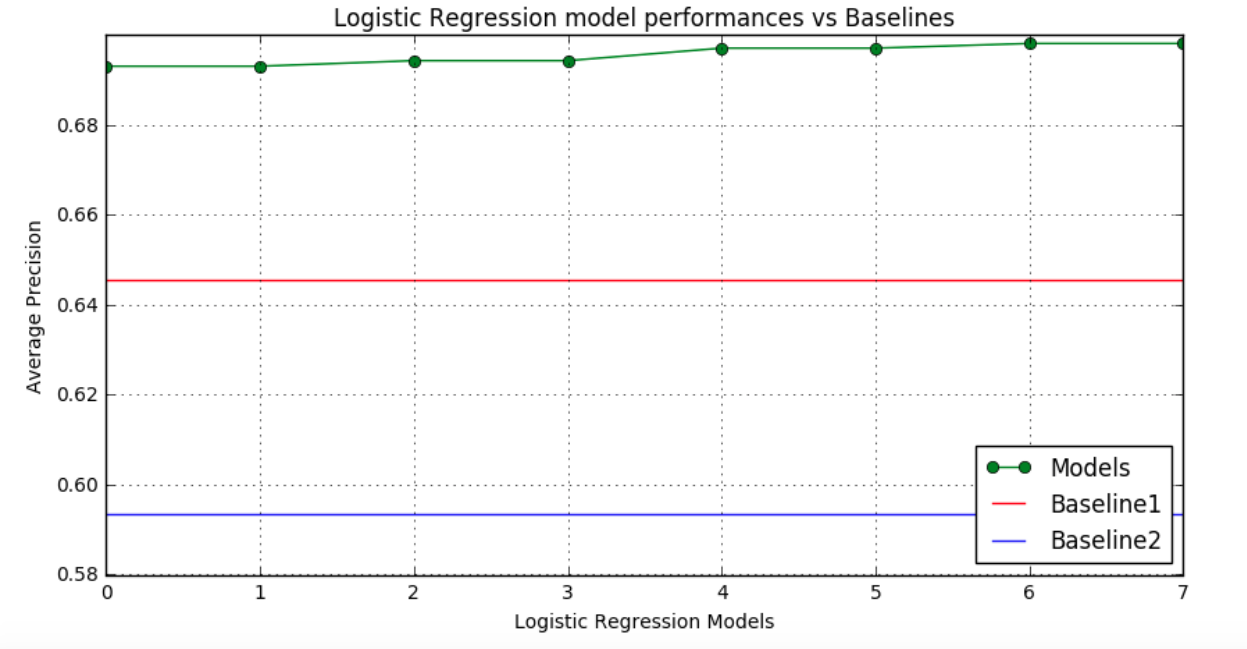
Classifiers

We have tried different settings for the following classifiers for conducting our analysis.

1. Logistic Regression (using CountVectorizer)
2. Logistic Regression (using TF-IDF vectorizer)
3. Naive Bayes (using MultinomialNB)
4. Stochastic Gradient Descent(using SGD)
5. Suppport Vector Machine (using LinearSVC)
6. Neural Network (using MLPClassifier)

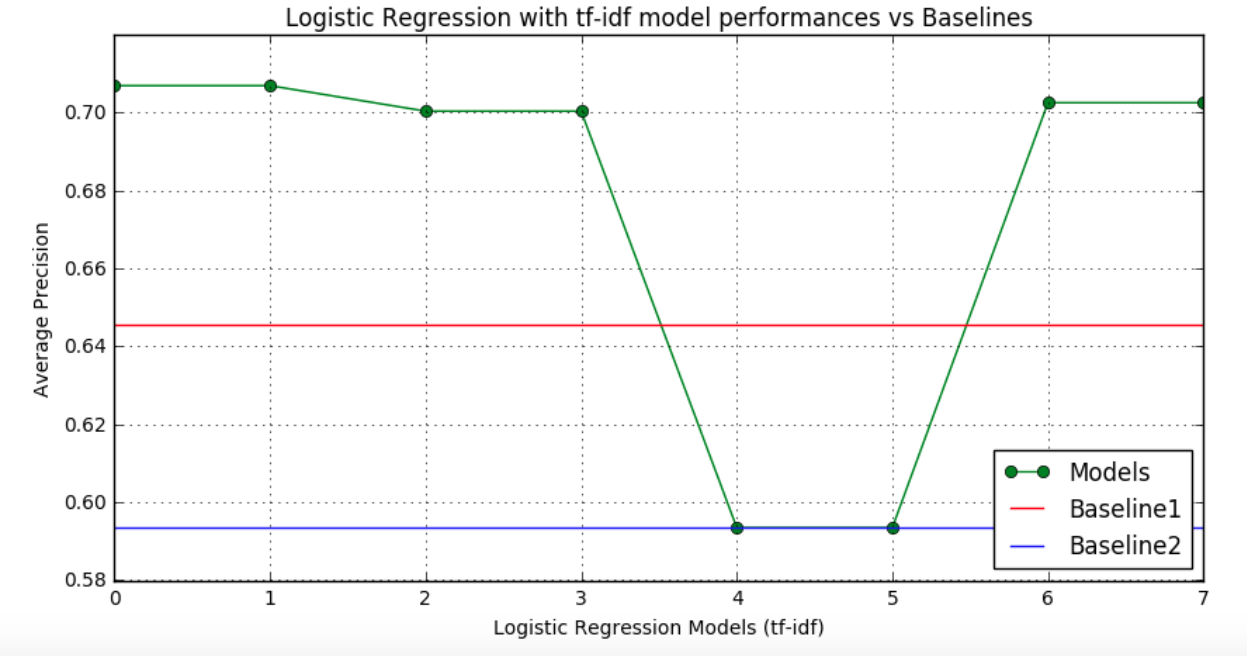
Logistic Regression (using CountVectorizer)

#	Settings	Precision
1	class weight : balanced C : 0.100000 random_state : 20	0.69
2	class weight : balanced C : 0.100000 random_state : 42	0.69
3	class weight : balanced C : 1.000000 random_state : 20	0.69
4	class weight : balanced C : 1.000000 random_state : 42	0.69
5	class weight : None C : 0.100000 random_state : 20	0.70
6	class weight : None C : 0.100000 random_state : 42	0.70
7	class weight : None C : 1.000000 random_state : 42	0.70



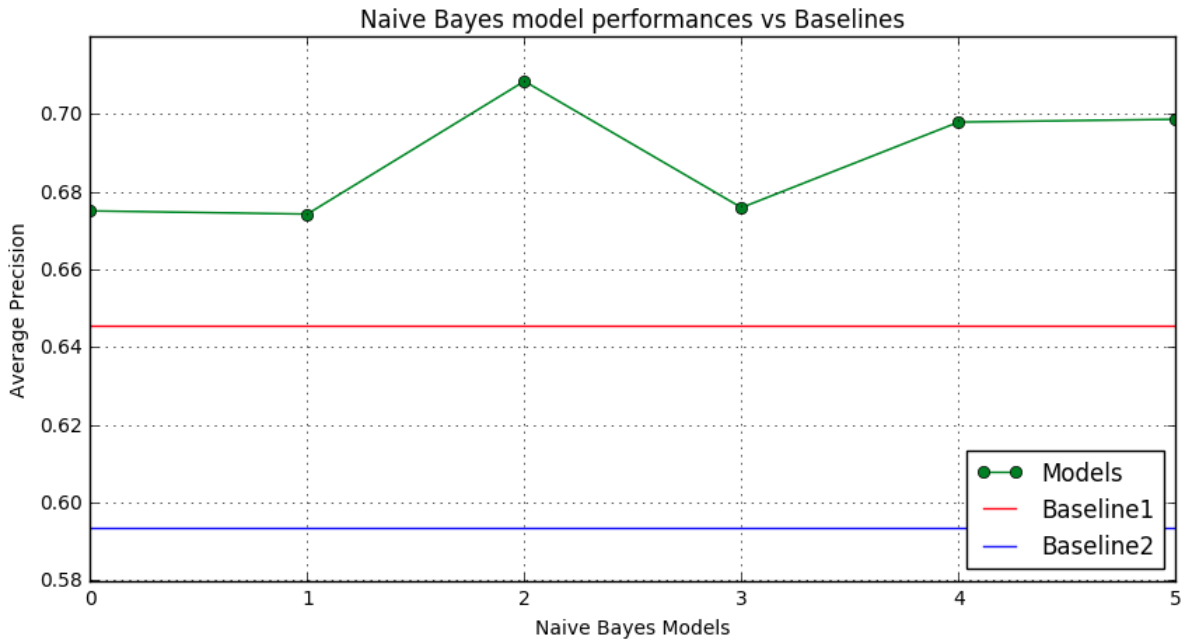
Logistic Regression (using Tf-idfVectorizer)

#	Settings	Precision
1	class weight : balanced C : 0.100000 random_state : 20	0.71
2	class weight : balanced C : 0.100000 random_state : 42	0.71
3	class weight : balanced C : 1.000000 random_state : 20	0.70
4	class weight : balanced C : 1.000000 random_state : 42	0.70
5	class weight : None C : 0.100000 random_state : 20	0.59
6	class weight : None C : 0.100000 random_state : 42	0.59
7	class weight : None C : 1.000000 random_state : 20	0.70
8	class weight : None C : 1.000000 random_state : 42	0.70



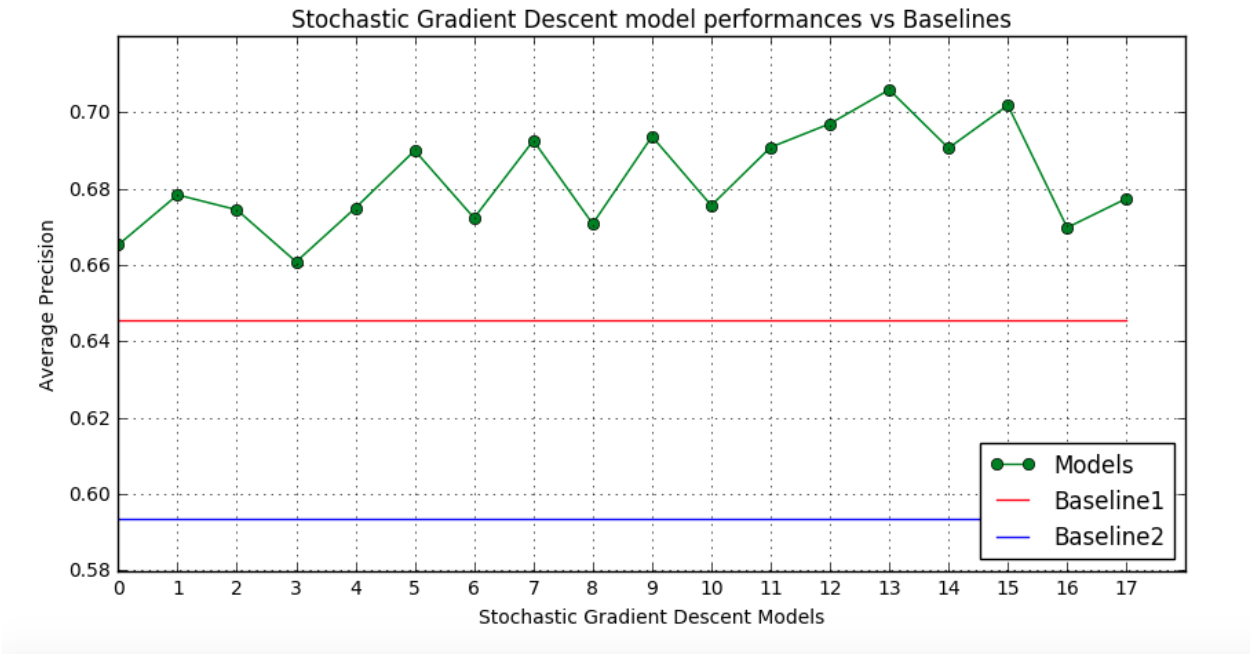
Naive Bayes (using MultinomialNB)

#	Settings	Precision
1	alpha : 0.100 fit_prior : True	0.68
2	alpha : 0.100 fit_prior : False	0.67
3	alpha : 1.000 fit_prior : True	0.71
4	alpha : 1.000 fit_prior : False	0.68
5	alpha : 2.000 fit_prior : True	0.70
6	alpha : 2.000 fit_prior : False	0.70



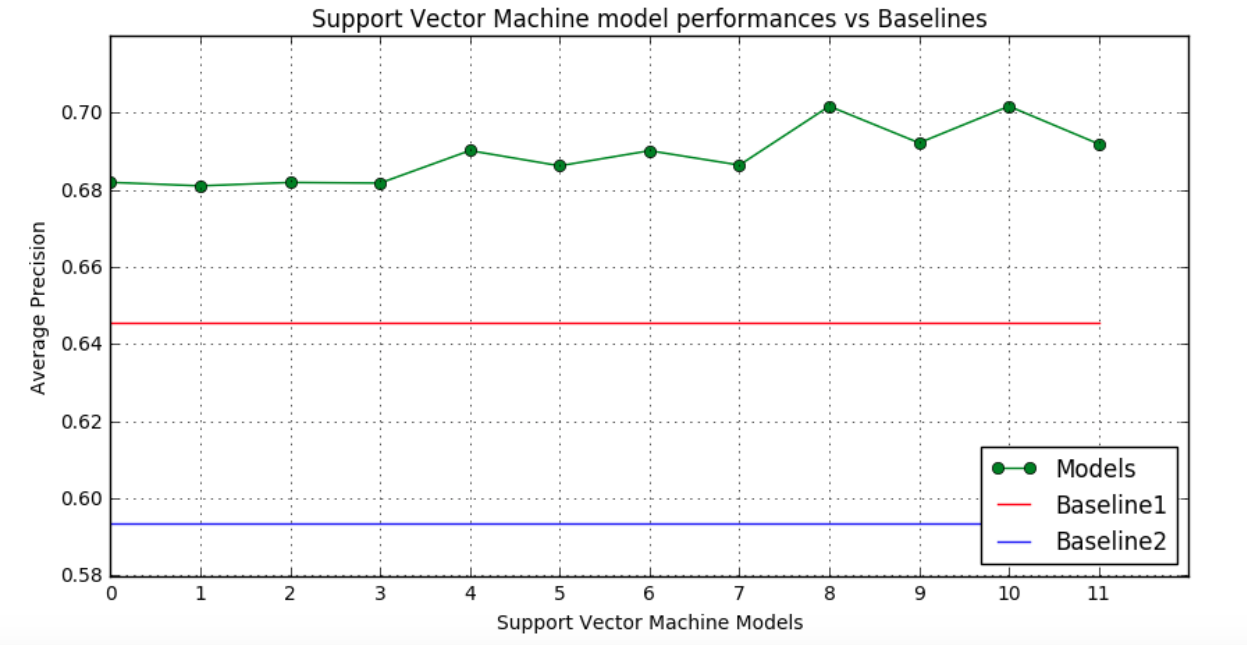
Stochastic Gradient Descent (using SGD)

#	Settings	Precision
1	alpha : 0.000 loss : hinge class_weight : None	0.67
2	alpha : 0.000 loss : hinge class_weight : balanced	0.68
3	alpha : 0.000 loss : modified_huberclass_weight : None	0.67
4	alpha : 0.000 loss : modified_huberclass_weight : balanced	0.66
5	alpha : 0.000 loss : perceptron class_weight : None	0.67
6	alpha : 0.000 loss : perceptron class_weight : balanced	0.69
7	alpha : 0.001 loss : hinge class_weight : None	0.67
8	alpha : 0.001 loss : hinge class_weight : balanced	0.69
9	alpha : 0.001 loss : modified_huberclass_weight : None	0.67
10	alpha : 0.001 loss : modified_huberclass_weight : balanced	0.69
11	alpha : 0.001 loss : perceptron class_weight : None	0.68
12	alpha : 0.001 loss : perceptron class_weight : balanced alpha : 0.001 loss : perceptron class_weight : balanced	0.69
13	alpha : 0.100 loss : hinge class_weight : None	0.70
14	alpha : 0.100 loss : hinge class_weight : balanced	0.71
15	alpha : 0.100 loss : modified_huberclass_weight : None	0.69
16	alpha : 0.100 loss : modified_huberclass_weight : balanced	0.70
17	alpha : 0.100 loss : perceptron class_weight : None	0.67
18	alpha : 0.100 loss : perceptron class_weight : balanced	0.68



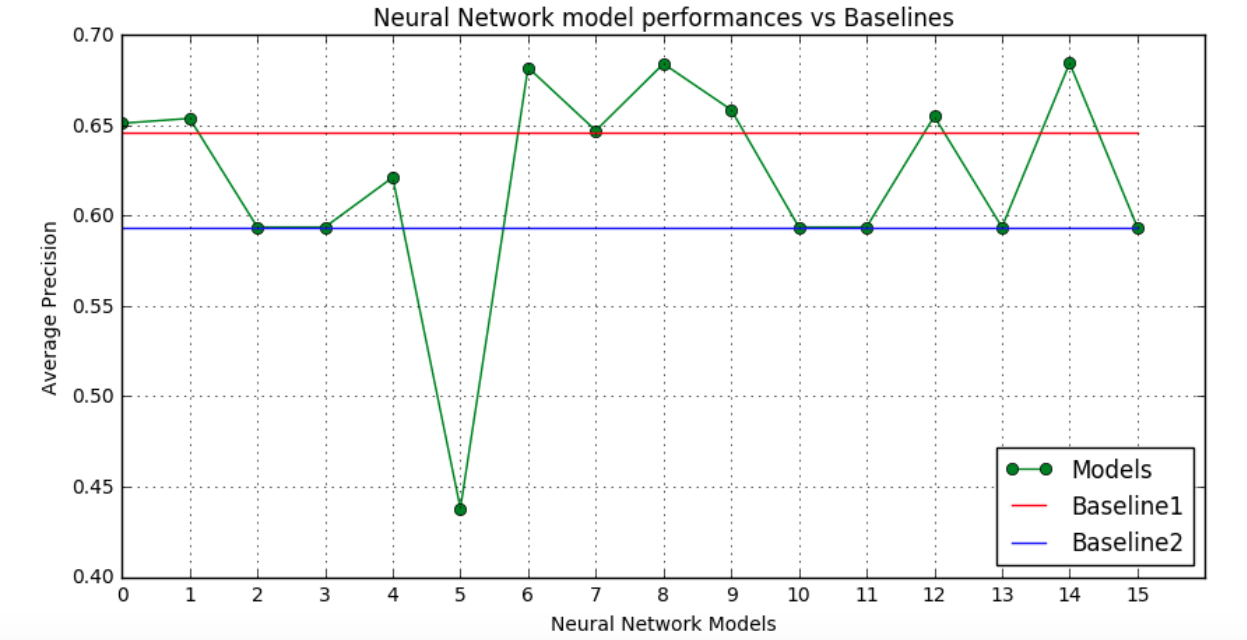
Suppport Vector Machine (using LinearSVC)

#	Settings	Precision
1	C: 1.000000 class_weight: balanced max_iter :1000 loss :hinge	0.68
2	C: 1.000000 class_weight: balanced max_iter :1000 loss :squared_hinge	0.68
3	C: 1.000000 class_weight: balanced max_iter :10000 loss :hinge	0.68
4	C: 1.000000 class_weight: balanced max_iter :10000 loss :squared_hinge	0.68
5	C: 0.100000 class_weight: balanced max_iter :1000 loss :hinge	0.69
6	C: 0.100000 class_weight: balanced max_iter :1000 loss :squared_hinge	0.69
7	C: 0.100000 class_weight: balanced max_iter :10000 loss :hinge	0.69
8	C: 0.100000 class_weight: balanced max_iter :10000 loss :squared_hinge	0.69
9	C: 0.010000 class_weight: balanced max_iter :1000 loss :hinge	0.70
10	C: 0.010000 class_weight: balanced max_iter :1000 loss :squared_hinge	0.69
11	C: 0.010000 class_weight: balanced max_iter :10000 loss :hinge	0.70
12	C: 0.010000 class_weight: balanced max_iter :10000 loss :squared_hinge	0.69



Neural Network(using MLPClassifier)

#	Settings	Precision
1	hidden_layer : (100,)activation_fn : identity learning_rate : 0.001000	0.65
2	hidden_layer : (100,)activation_fn : identity learning_rate : 0.100000	0.65
3	hidden_layer : (100,)activation_fn : logistic learning_rate : 0.001000	0.59
4	hidden_layer : (100,)activation_fn : logistic learning_rate : 0.100000	0.59
5	hidden_layer : (100,)activation_fn : tanhlearning_rate : 0.001000	0.62
6	hidden_layer : (100,)activation_fn : tanhlearning_rate : 0.100000	0.44
7	hidden_layer : (100,)activation_fn : relulearning_rate : 0.001000	0.68
8	hidden_layer : (100,)activation_fn : relulearning_rate : 0.100000	0.65
9	hidden_layer : (2,)activation_fn : identity learning_rate : 0.001000	0.68
10	hidden_layer : (2,)activation_fn : identity learning_rate : 0.100000	0.66
11	hidden_layer : (2,)activation_fn : logistic learning_rate : 0.001000	0.59
12	hidden_layer : (2,)activation_fn : logistic learning_rate : 0.100000	0.59
13	hidden_layer : (2,)activation_fn : tanhlearning_rate : 0.001000	0.65
14	hidden_layer : (2,)activation_fn : tanhlearning_rate : 0.100000	0.59
15	hidden_layer : (2,)activation_fn : relulearning_rate : 0.001000	0.68
16	hidden_layer : (2,)activation_fn : relulearning_rate : 0.100000	0.59



For each of these model type we have tried different settings and picked the best one for each.

After finding the best from each model type we compared them with each other and picked the final model the one which consistently gave the highest precision score over the test data.

Evaluation Strategy

Dividing Train and Test sets: We have divided 60% of our dataset as train and the remaining as test set. We did this train test split since we had a huge dataset which had the score values for each instances which we could use to verify the performance of the model. The score values helped us to get the insight about the truth values for the test set instances which we used for verifying the precision score of the predicted labels.

Performance Results

Below are the performance result of best parameter setting for each model:

Model	Parameter	Performance
Baseline1	Random class	0.65
Baseline2	Majority class	0.59
Logistic regression(using CountVectorizer)	C=1.0, random_state=None	0.70
Logistic Regression (using TF-IDF vectorizer)	C=0.1, class_weight="balanced"	0.71 (Best Model)
Naive Bayes (using MultinomialNB)	alpha=1.0, class_prior=None, fit_prior=True	0.71
Stochastic Gradient Descent(using SGD)	alpha=0.100, loss="modified_huber"	0.68

Suppport Vector Machine (using LinearSVC)	C=0.01, class_weight="balanced", max_iter=10000, loss="hinge"	0.70
Neural Network (using MLPClassifier)	hidden_layer_sizes=(2,), activation="identity", learning_rate_init=0.001	0.69

The logistic regression model with tf-idfvectorizer gave the best performance.

Top Features

Positive Class	Negative Class
href (2.7344)	rel (-2.10935)
code (2.1003)	nofollow (-2.08903)
http (2.02101)	tried (-1.13362)
org (1.4948)	problem (-1.09181)
docs (1.46882)	error (-0.789313)
python (0.873201)	works (-0.742989)
numpy (0.83672)	thanks (-0.718121)
library (0.790319)	script (-0.685481)
timeit (0.757402)	maybe (-0.680525)
course (0.723749)	could (-0.652846)

Discussion.

The best classifier did perform better than the baselines which was the minimum set expectation for this analysis. Though it did not give impressive performance it is doing better than other models we tried. We expected our classifier to perform with higher precision such as in the range of 0.80-0.85 but given the dataset it is difficult for this classifier to learn better on the keywords and understand the context for each instance with respect to the class that instance belongs to. As we can see from the input matrix shape(100,000, 229608) we had a humungous amount of columns (features) and this might be the case where, the not very important featuremight be contributing towards the weird predictions sometimes. It is easy to get confused for tokens and their classes in such a huge matrix where tokens that might be appearing less number of times are still getting considered highly for one or the other classes.

Interesting/Unexpected Results

- **Question Title: How do you create a daemon in Python?**

  80% of the time, when folks say "daemon", they only want a server. Since the question is perfectly unclear on this point, it's hard to say what the possible domain of answers could be. Since a server is adequate, start there. If an actual "daemon" is actually needed (this is rare), read up on as a way to daemonize a server. Until such time as an actual daemon is actually required, just write a simple server. Also look at the implementation

  http://www.python.org/doc/2.5.2/lib/module-wsgiref.html

  http://www.python.org/doc/2.5.2/lib/module-SimpleHTTPServer.html

  "Are there any additional things that need to be considered? " Yes. About a million things. What protocol? How many requests? How long to service each request? How frequently will they arrive? Will you use a dedicated process? Threads? Subprocesses? Writing a daemon is a big job.

- **Answer:**

  Searching on Google reveals x2 code snippets. The first result is to this code recipe which has a lot of documentation and explanation, along with some useful discussion underneath.

  However, another code sample, whilst not containing so much documentation, includes sample code for passing commands such as start, stop and restart. It also creates a PID file which can be handy for checking if the daemon is already running etc.

- **True label :** 0 (Negative/Bad Answer)
- **Predicted** : 1 (Positive/Good Answer)

  When we checked the true labels this answer has the second most negative score value(-23) in dataset but still the classifier predicted it as the positive.

- **Reasons:**

  1. Quality/Relevance
  2. Context establishment

- **Quality/Relevance**

  The answer talks about the code snippets found over google and does not provide exhaustive information sought by the user who posted the question.

  Also the data shows that about 23 people (second most negative score) have down-voted this answer. Since people can judge and decide to down-vote based on the code quality and requirement satisfaction, they think it’s a bad answer

- **Context Establishment**

  As the classifier establishes the context based on the tokens and the target label for each sample. It might be the case there are a few tokens that the classifier is confused on.

  Since there are majority of the answers in the positive class and almost everyone has a code snippet. Predicting them into one class always and establish the context of the code to the question is difficult task for this classifier

Contributions of Each Group Member

For every phase of the project both of us have worked together. We had scheduled 3 hours meeting every day for this project work. We discussed the tasks for all the phases in detail and planned them well. We discussed the approach and strategies for each task and coded together. All the decisions are made with mutual consent and both of us are aware of every single code snippet submitted for this project.

Conclusion

We found that the logistic regression classifier with CountVectorizer performs well on an average and does not fluctuate much with change of parameter settings. Neural networks on the other hand behave different for each setting. They are prone to overfitting and drastically change the behavior with changed parameter setting.

Other models' performance does show a fluctuation in a given range butdid not deviate drastically when the settings were altered. We observed the best precision with Naïve Bayes and Logistic Regression with tf-idfvectorizer. Since the Naïve Bayes did not give the same result on repeated runs we considered the Logistic Regression with tf-idfvectorizer as our best model with precision of 0.71.

References

1. http://scikit-learn.org/stable/supervised_learning.html#supervised-learning
2. https://www.kaggle.com/datasets
3. http://stackoverflow.com/
4. CS584 Lecture Slides