

Capstone Project

Machine Learning Engineer Nano degree

Pradeep Pujari

August 11, 2016

Definition

Project Overview:

This project work is from yelp data challenge. Many businesses collect reviews of their services or products. Along with reviews text, reviewer is also asked to enter rating in the range of one to five, five being excellent. Sometimes, there is discrepancy between the numeric value of the rating and reviews text. In fact many start-ups evolved to understand and interpret this data and provide good insight to the business. Yelp provides data set for this challenge. In their data challenge page, they also put some ideas to consider. I have chosen the idea of predicting numeric rating from the reviews text alone. They also provided example code to use for this datasets.

Problem Statement

Predict numeric rating from reviews text.

We are given labeled training data, which has reviews text, and it's rating. Rating is considered as label or class that is to be predicted. Each review was given a rating on a discrete label in the scale 1 to 5. Data set is provided by Yelp. The details can be found [here](#).

The problem which needs to be solved is clearly defined. A strategy for solving the problem, including discussion of the expected solution, has been made.

This project is from Yelp data challenge. How well we can guess a Yelp review's rating from its reviews text alone? Is there a correlation exists between rating and review text? What kind of interesting features were found when rating is high? Although the rating is for that particular product or service, here I have not considered that relationship. Reviews text is tokenized first. Count tokens, filter tokens that have high occurrences and stop words. Occurrence count is a good start but there is an issue: longer documents will have higher average count values than shorter documents, even though they

might talk about the same topics. To avoid these potential discrepancies it suffices to divide the number of occurrences of each word in a document by the total number of words in the document: these new features are called `tf` for Term Frequencies.

Another refinement on top of `tf` is to downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus.

This downscaling is called `tf-idf` for “Term Frequency times Inverse Document Frequency”, with TF-IDF Vectorizer. Then Logistic Regression Classifier is used to train the model. I persist the model with pickle. The predict API is used to predict for unseen reviews text.

Metrics

F1 score which is defined as harmonic mean of precision and recall used to measure the performance of the model, where best value reaches at 1 and the worst score at 0. This project is not a binary classifier. This is a multi-class and multi-label case. By default, parameter search uses the `score` function of the estimator to evaluate a parameter setting. But in this case, alternate scoring function

F1_MACRO used to address class imbalance. This is specified as scoring parameter to GridSearchCV constructor.

Analysis

Data Exploration

If a dataset is present, features and calculated statistics relevant to the problem have been reported and discussed, along with a sampling of the data. In lieu of a dataset, a thorough description of the input space or input data has been made. Abnormalities or characteristics about the data or input that need to be addressed have been identified.

Yelp business review data set consists five data files. For this project, I considered only reviews file.

The features are Reviews Text, Vote (Useful, Funny or Cool) and length of reviews text (new feature not part of data set) is considered for this project.

Found relationship between each of the vote types (cool/useful/funny) and the number of stars as follows.

	Cool	useful	funny
Stars			
1	0.268	1.289	0.547
2	0.404	1.192	0.567
3	0.578	1.055	0.521

4	0.714	1.064	0.481
5	0.545	0.869	0.333

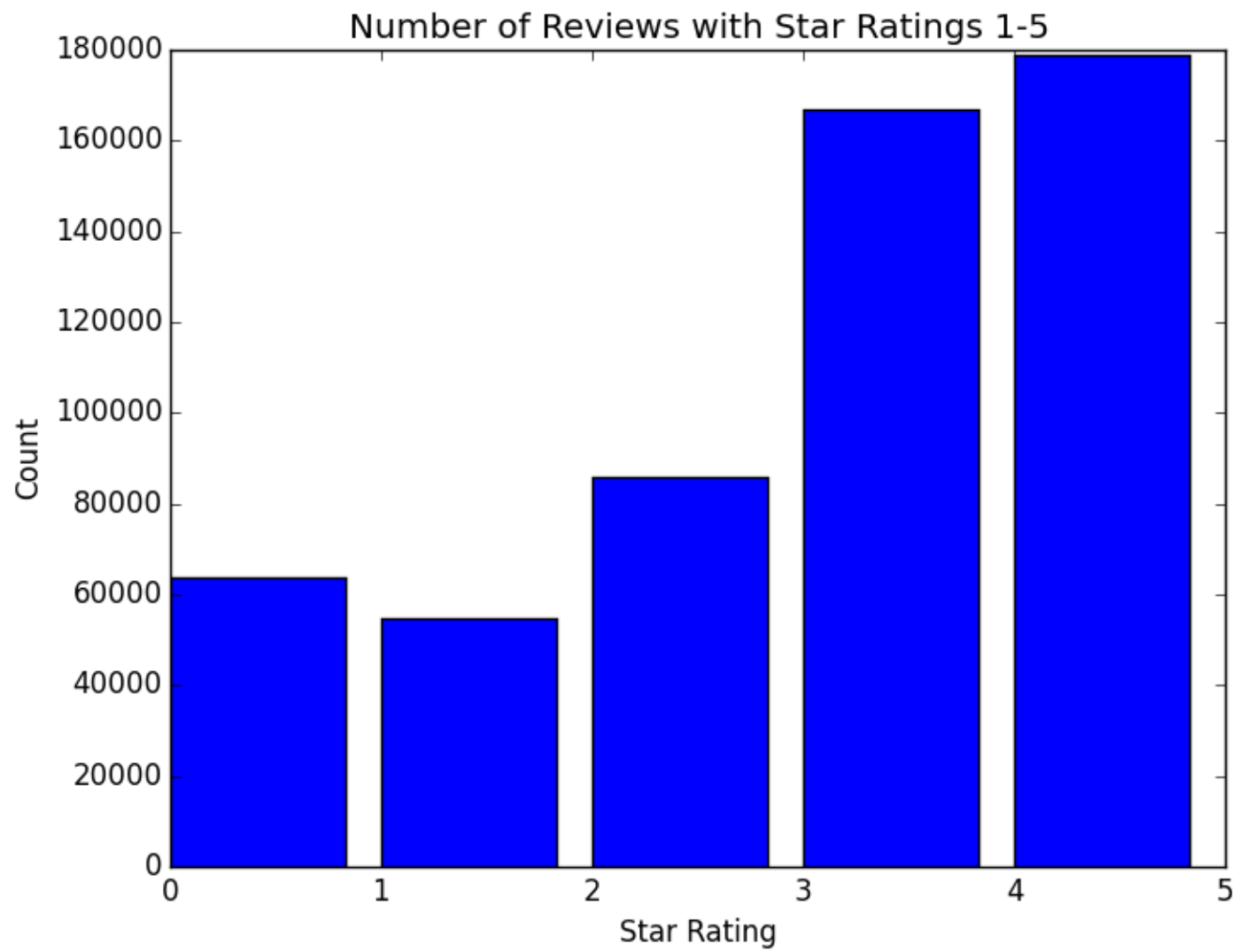
Mostly see that where people vote for Useful, Funny or Cool, star rating is above 3.
Below are few examples:

Review Text	Star Rating	Vote (Useful/Funny/Cool)
Decent range somewhat close to the city. The mats are pretty solid; however, the grass range needs to be tended too. It's like hitting out of US Open type rough...not very amenable to practicing. Which kind of defeats the purpose of going to a golf range...Still gets 3 stars because the range is lit up at night which is excellent	3	Useful = 1

for those of us who are addicted to this amazing game, but are somewhat short on time (having a job kinda sucks sometimes, no?)."}}		
Yummy Yum, Let me get me some. This place rocks!! Went here with the wife and had a great dinner. Portions are very generous and the food is delicious. Be prepared to pay top dollar though for the entrees, but they are worth every penny!	5	Useful = 1
Starbucks is a chain coffee and tea shop you will find most everywhere. They offer free WiFi. They sell their coffees in bags if you like. They also serve sandwiches and pastries. Some of the Starbucks I've seen offer drive through services. Prices are reasonable. I stopped in here since I've been hunting all over for a gingerbread man cookie. They told me they are getting one in and ugly sweater design but they don't have it yet. I plan to keep checking since it appears that	3	Cool = 1

no place I have been to has my holiday favorite so I'll certainly be back to one of their locations.		
This is smaller store than ones i've seen inside malls, but they seem to have all of the necessities. They have a great collection of candles! My favorites were in the \"relax\" section. Turquoise Sky & Home Sweet Home were great! I'm not a huge fan of food scented candles, but they actually had very nice ones! Apple Pumpkin was great and surprisingly i lovedddd Christmas Cookie! The employees were all helpful.	4	Funny = 1

Looking at below rating (stars) distribution. Distribution is obviously skewed. People tend to write positive reviews more. Also, reviews with 4 star and 5 star very close to each other. Model might predict 4 star, a 5 star and vice versa. Similarly, 1 star and 2 Star ratings are very close to each other.



Identified correlation between review length and star rating if any, and plotted as histogram for each rating as below:

Exploratory Visualization

Visualization has been provided that summarizes or extracts a relevant characteristic or feature about the dataset or input data with thorough discussion. Visual cues are clearly defined.

There are 2225213 reviews. Reviews are for a business or any properties of business. Business types range from restaurants to home improvement to automotive services. I do see new line character \n in most of the place and were removed.

Star Rating	Mean	Standard Deviation
1	6.257	0.914
3	6.22	0.861

5	5.93	0.88
---	------	------

Please see explore_data ipython notebook for code details.

Total documents in the data set: 2225213

1 rated reviews = 11.71%

2 rated reviews = 8.54%

3 rated reviews = 12.68%

4 rated reviews = 26.59%

5 rated reviews = 40.49%

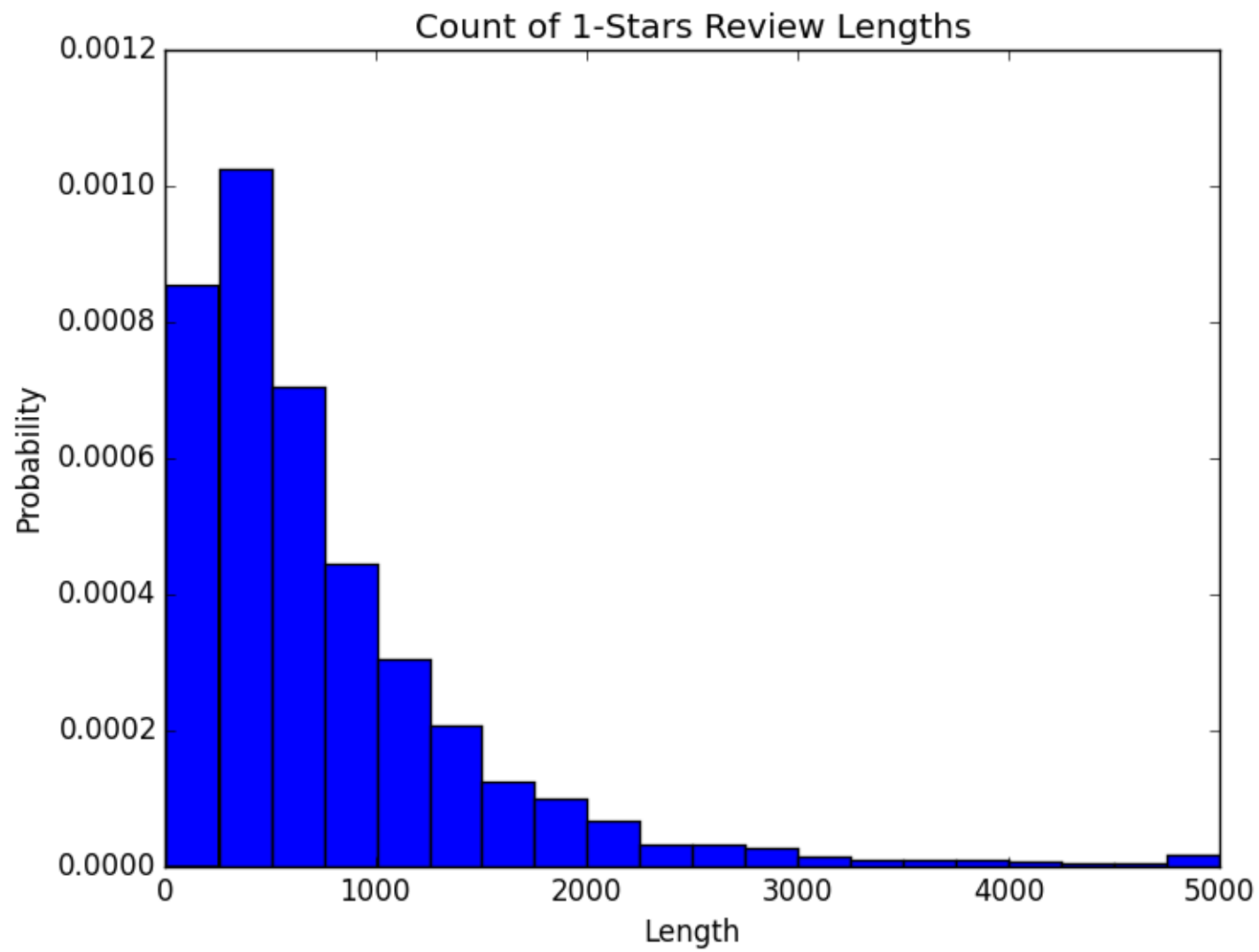
Data Format

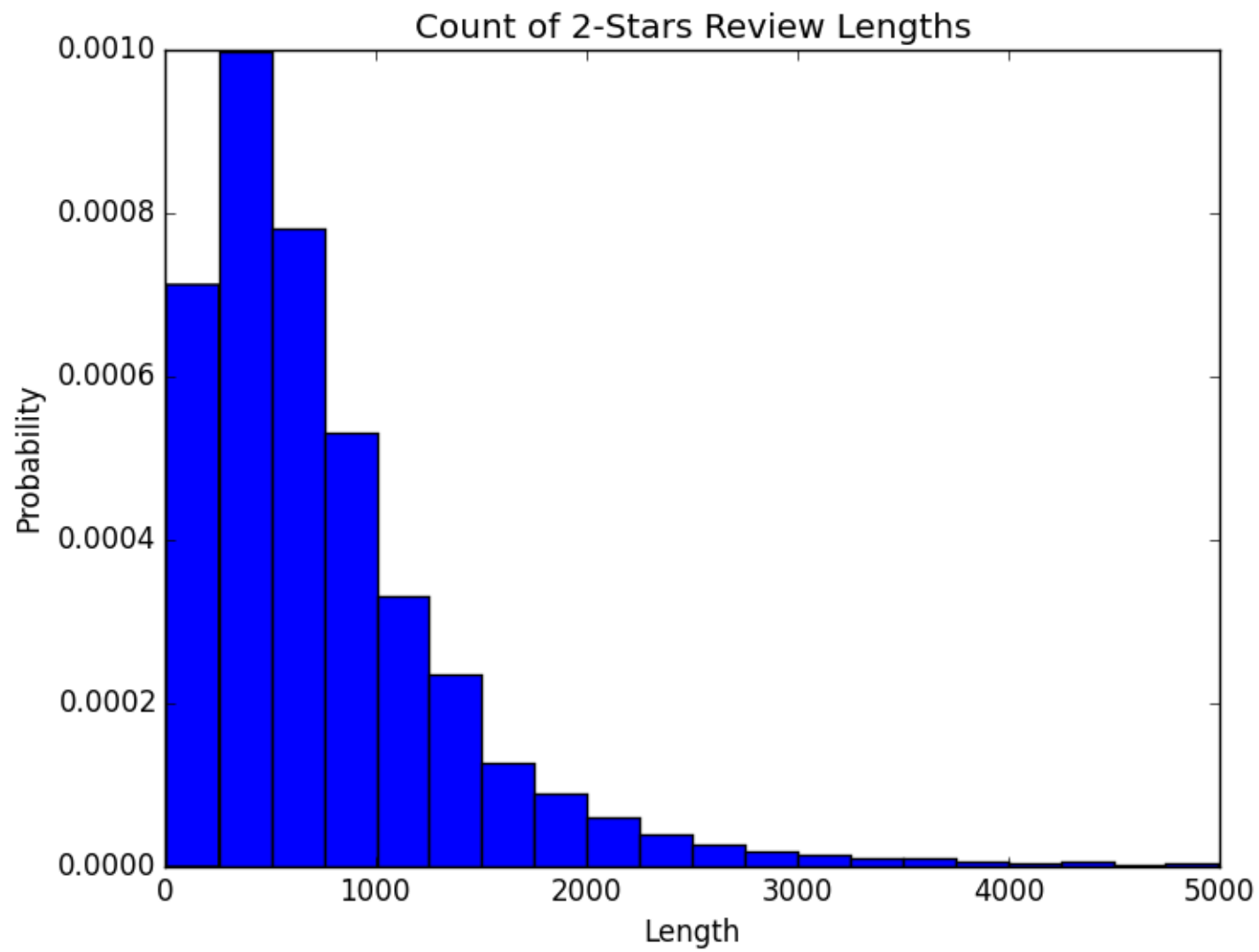
review

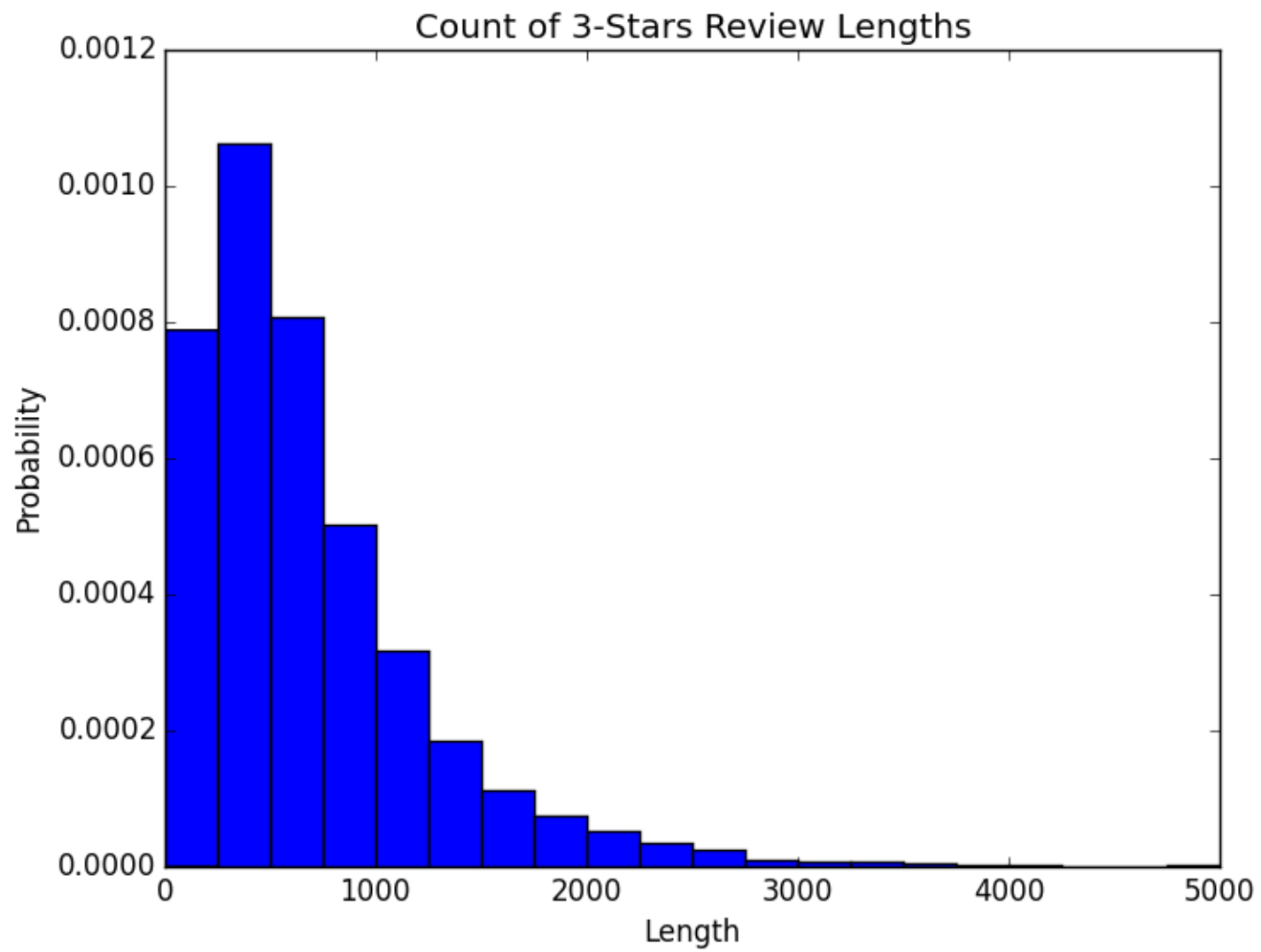
```
{  
  'type': 'review',  
  'business_id': (encrypted business id),  
  'user_id': (encrypted user id),  
  'stars': (star rating, rounded to half-stars),  
  'text': (review text),
```

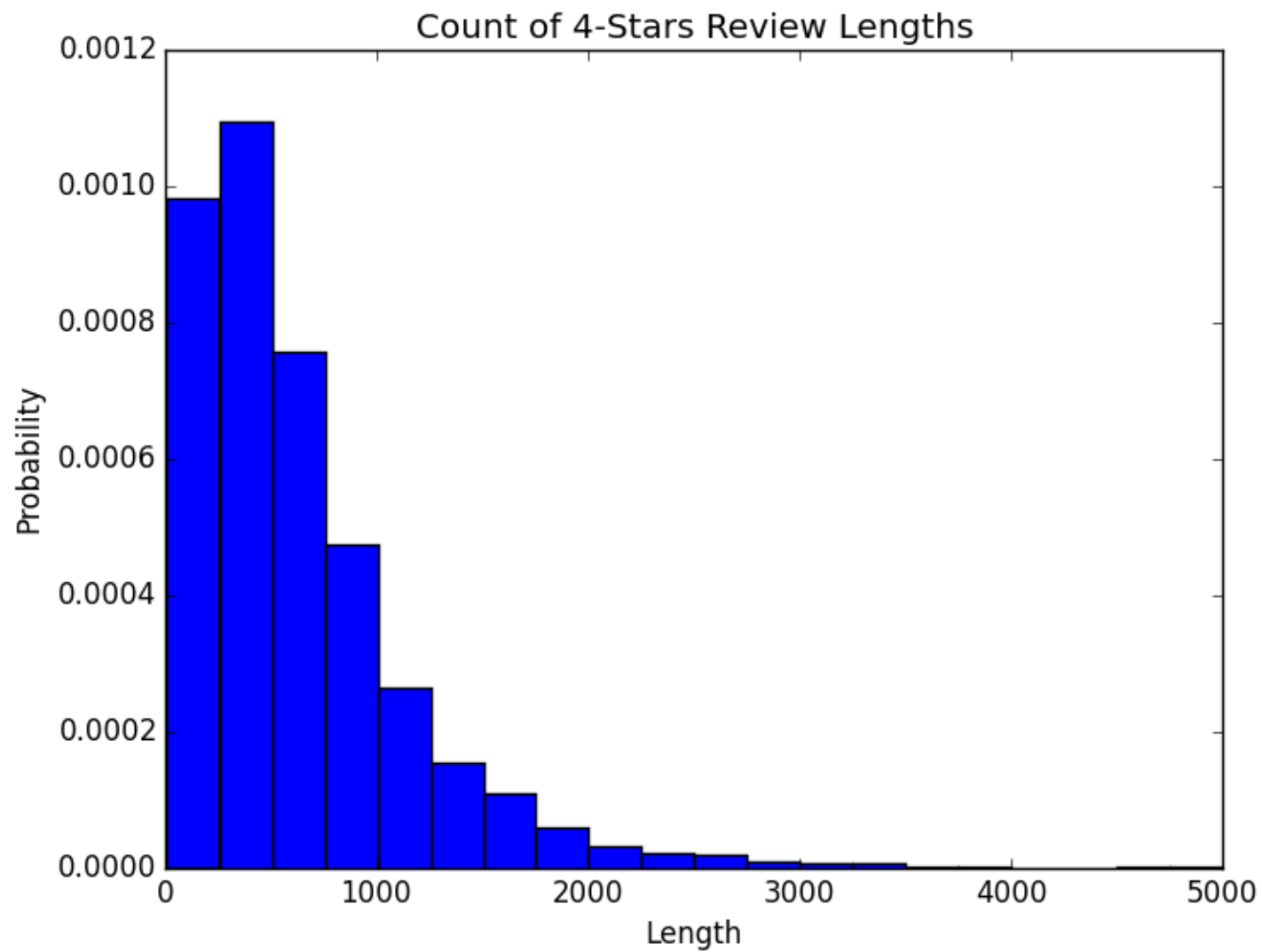
```
'date': (date, formatted like '2012-03-14'),  
'votes': {(vote type): (count)},  
}
```

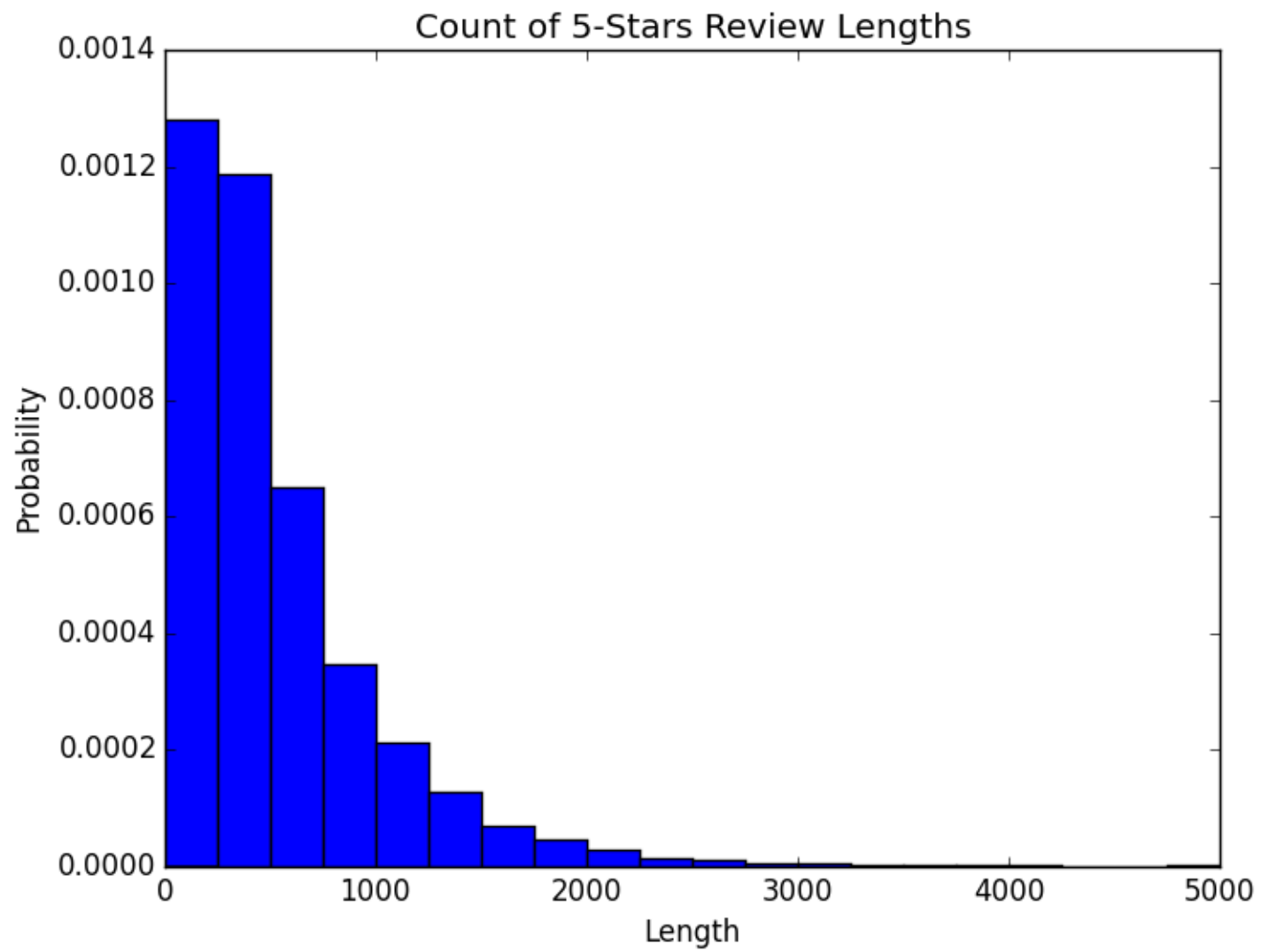
Below are few more analysis of 1 star rating versus reviews length.











Algorithms and Techniques

We have an array of algorithms like Naive Bayes, decision trees, Random Forests, Support Vector Machines, and many others. Logistic regression classifier classifies text documents by using a bag-of-words approach, efficiently handles sparse features. I have chosen Logistic Regression, as it is fast, easy to regularize and it outputs well-calibrated predicted probabilities. It also works with categorical features.

Raw reviews data cannot be directly feed to Logistic Regression algorithm as it expects numerical feature vector with a fixed size rather than the raw text documents of variable length. In order to address this, sci-kit learn provides utilities to extract numerical features from text content, namely

- **tokenizing** strings and giving an integer id for each possible token, for instance by using white spaces and punctuation as token separators.
- **counting** the occurrences of tokens in each document.
- **normalizing** and weighting with diminishing importance tokens that occur in the majority of samples / documents.

I used tf-idf vectorizer to generate features from reviews text. I set the parameter to remove English stop words.

Tuned Parameters

min_df : float in range [0.0, 1.0] or int, default=1 (Tuned to 0.1)

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

Stop word English is used.

Simple Logistic Regression Classifier with GridSearchCV to train the model.

GridsearchCV searches over parameter values for logistic regression.

Parameters that are not directly learnt, also called hyper parameters can be set by searching a parameter space for the best score. A search consists of:

- a classifier
- a parameter space;
- a method for searching or sampling candidates;
- a cross-validation scheme; and
- a score function.

By default, parameter search uses the `score` function of the estimator to evaluate a parameter setting.

Following parameters were tuned with GridsearchCV

```
grid = {'C': 10.0 ** np.arange(-2, 3),  
        'penalty': ['l1', 'l2'],  
        'class_weight': [None, 'auto']}
```

C Inverse of regularization strength

Penalty whether L1 or L2 regularization

class_weight Weights associated with classes

Naive Bayes classifier for multinomial models

Multinomial Naive Bayes implements the naive Bayes algorithm for multinomially distributed data. The multinomial Naive Bayes classifier is suitable for classification with discrete features like text classification. The multinomial distribution normally requires integer feature counts unlike logistic regression.

Logistic regression

From train.py

```
clf_base = MultinomialNB()  
grid = {'alpha': 0.1 * np.arange(1, 11, 2),  
        'fit_prior': [True, False]}
```

alpha : float, optional (default=1.0)

Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

fit_prior : boolean

Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

One interesting finding is that NB can perform better when there's a low amount of training data. But LR should always outperform given enough data according to <http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf>

Benchmark

Student clearly defines a benchmark result or threshold for comparing performances of solutions obtained.

This challenge has not given any predefined threshold or benchmark. Here, I experiment with 2 algorithms Multinomial NB and Logistic Regression. Logistic Regression performed better than Multinomial NB, so in the final project I choose Logistic Regression. Below is performance of both the models.

Multinomial Naïve Bayes performance result:

Best estimator MultinomialNB(alpha=0.10000000000000001, class_prior=None, fit_prior=False)

Precision on training data = 0.98121875

Precision on test data = 0.534125

Accuracy on training data = 0.98121875

Accuracy on test data = 0.534125

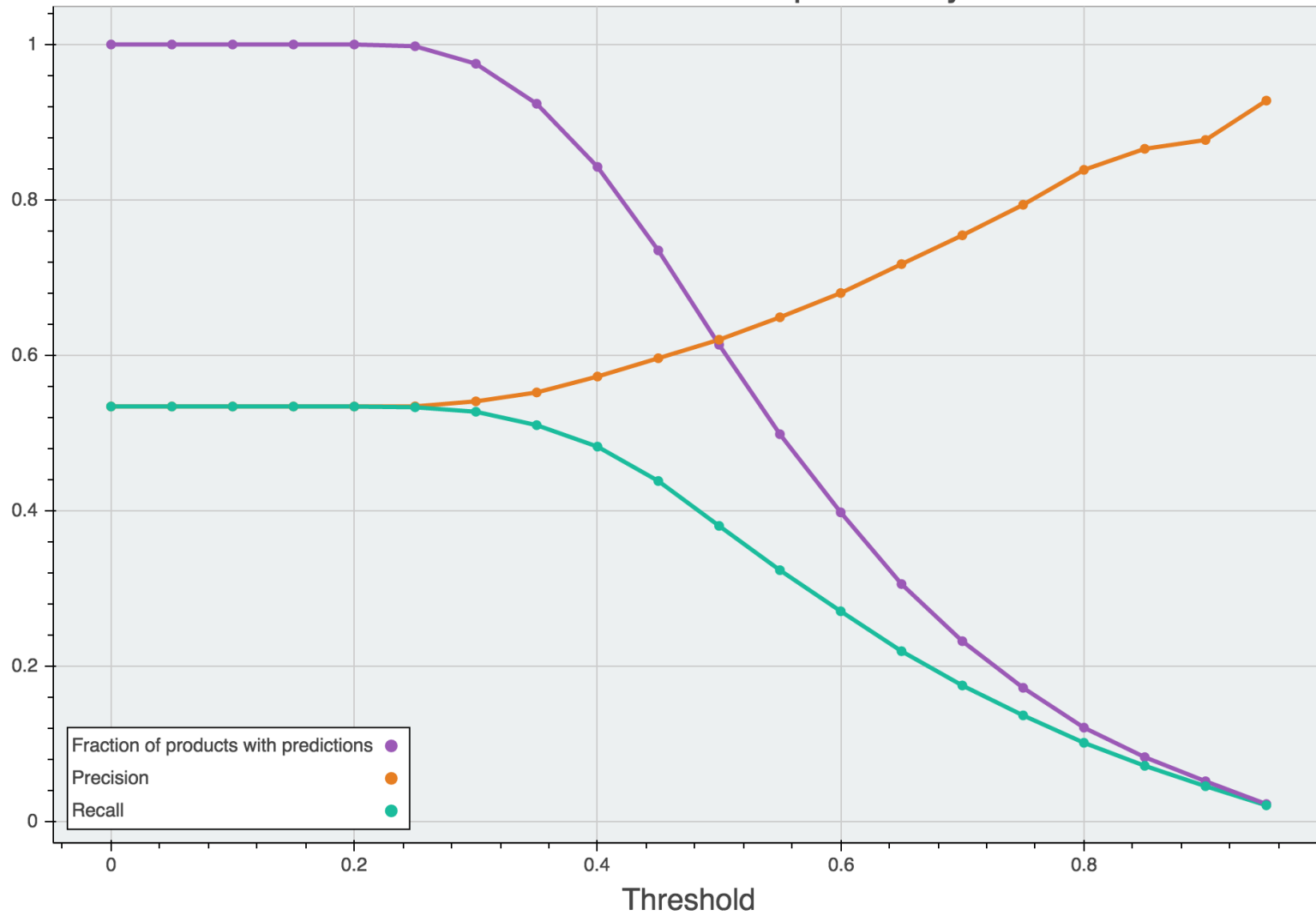
Classification Report

Star	precision	recall	f1-score	support
1	0.60	0.71	0.65	948
2	0.45	0.22	0.30	810
3	0.42	0.28	0.33	1306
4	0.45	0.62	0.52	2348
5	0.66	0.62	0.64	2588
avg / total	0.53	0.53	0.52	8000

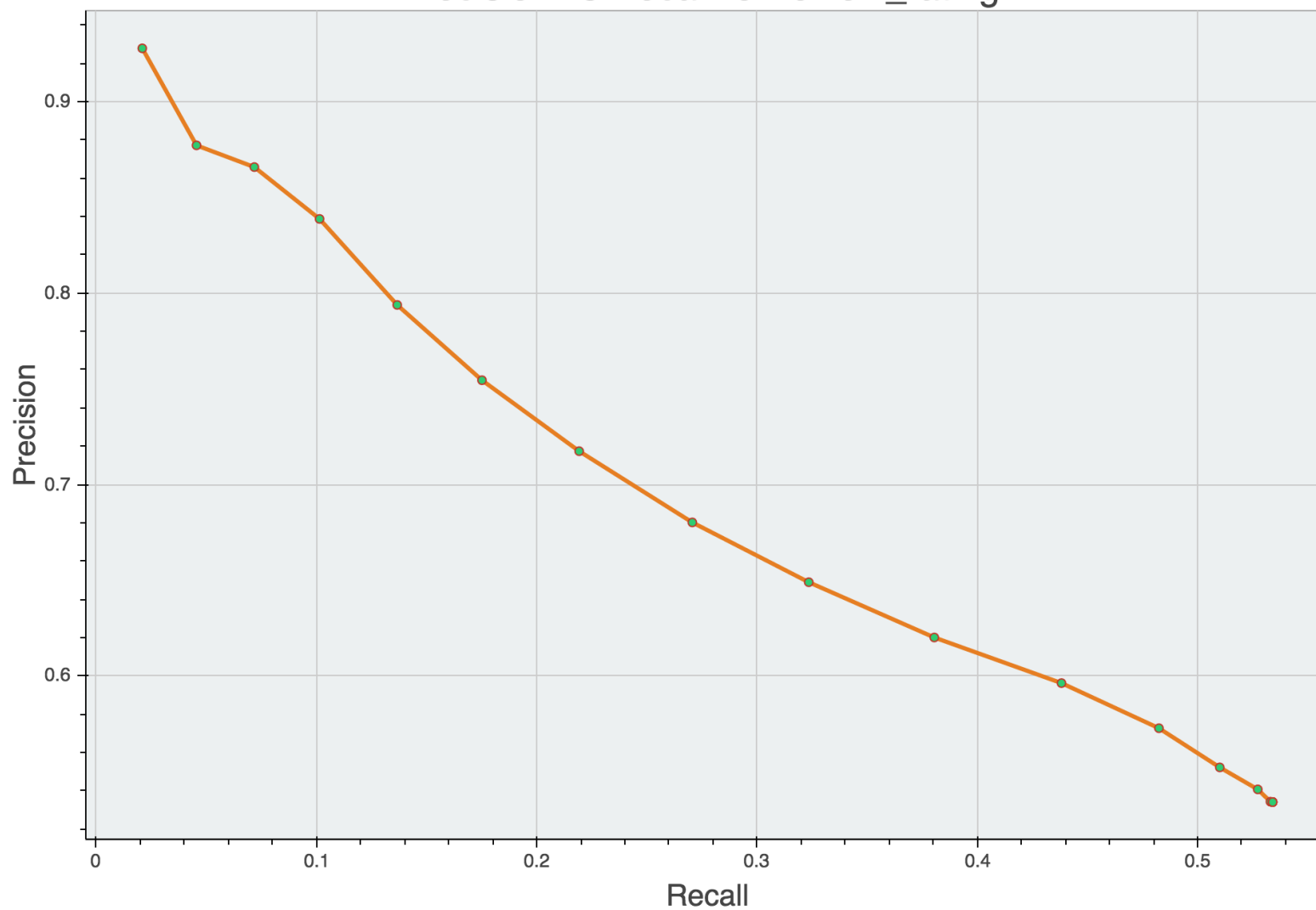
Printing model stats

Overall precision = 0.534125

Precision/Recall as a function of probability threshold



Precision vs Recall for review_rating



Logistic Regression:

```
Best estimator LogisticRegression(C=1.0, class_weight='auto', dual=False,
fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr',
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0)
```

Precision on training data = 0.82665625

Precision on test data = 0.577

Accuracy on training data = 0.82665625

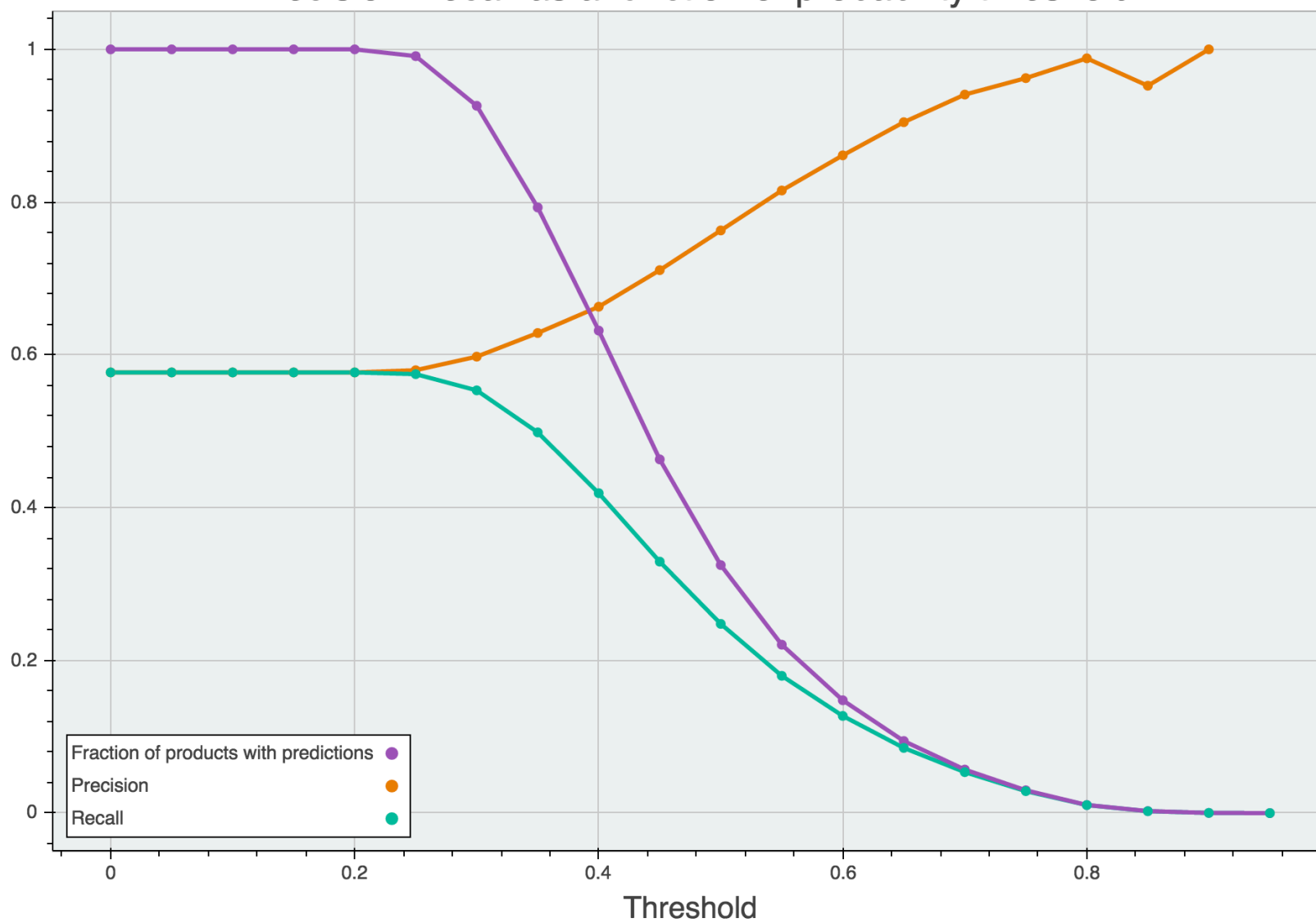
Accuracy on test data = 0.577

Classification Report

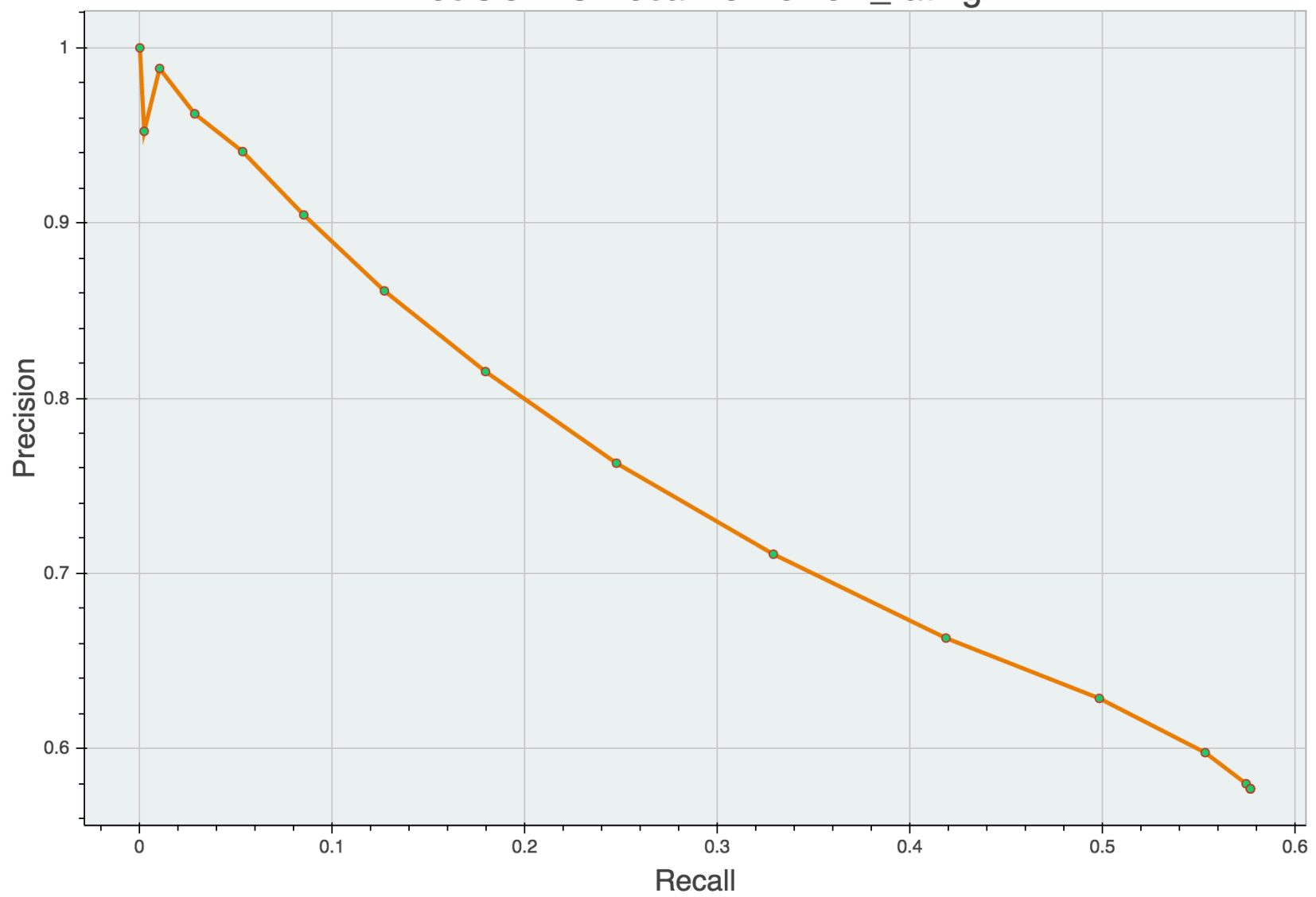
Stars	precision	recall	f1-score	support
1	0.60	0.77	0.67	948
2	0.43	0.38	0.40	810
3	0.50	0.40	0.44	1306
4	0.54	0.52	0.53	2348
5	0.67	0.71	0.69	2588
avg / total	0.57	0.58	0.57	8000

Printing model stats:
Overall precision = 0.577

Precision/Recall as a function of probability threshold



Precision vs Recall for review_rating



Methodology

Data Preprocessing

During preprocessing, I extracted only relevant fields required for the algorithm.
They are

Review_id

Review_text

Votes (funny, useful and cool)

Stars (i.e. numeric rating) this is the class label.

Each observation in this dataset is a review of a particular business by a user.

Added a new feature, reviews text length. Instead of absolute length, I made it a range, because adding absolute length in fact reduced precision and recall.

Explored the relationship between each of the vote types (cool/useful/funny) and the number of stars.

“stars” field is a categorical variable. Looking the difference between each vote types.

Implementation

The process for which metrics, algorithms, and techniques were implemented with the given datasets or input data has been thoroughly documented. Complications that occurred during the coding process are discussed.

Finally, based on the model evaluation as explained above, selected logistic regression classifier.

It is common practice when performing supervised machine learning experiment to hold out part of the available data as a **test set**. In scikit-learn a random split into training and test sets can be quickly computed with the `train_test_split` helper function.

I have taken a training data set size = 50K. Training/test split is configured to be 80:20. That means 80% of 50K is used for training the model and 20% of 50K used for prediction and to estimate model performance.

Original Yelp data set size is over 2.2 million. I could not able to take all the data due to my memory limitation of my laptop. However, I have taken approximately equal number data point from each of star ratings. This helped to get rid of class/label imbalance issue. I have used GridSearchCV which is computationally expensive when dealing with large different hyper parameter and bigger data sets.

New additional features 1) length of reviews and 2) Votes were added to training data set. Precision, Recall and F1 Score with these additional new features added were as follows.

Classification Report

Star Rating	Precision	Recall	f1-score	support
1	0.58	0.76	0.66	948
2	0.37	0.37	0.37	810
3	0.45	0.40	0.43	1306
4	0.53	0.45	0.49	2348
5	0.65	0.69	0.67	2588

avg / total	0.54	0.55	0.54	8000
-------------	------	------	------	------

Precision, Recall and F1 Score with simple reviews text without any additional features:

Classification Report

Stars	precision	recall	f1-score	support
1	0.60	0.77	0.67	948
2	0.43	0.38	0.40	810
3	0.50	0.40	0.44	1306
4	0.54	0.52	0.53	2348
5	0.67	0.71	0.69	2588
avg / total	0.57	0.58	0.57	8000

Preprocessed input data set as explained above in the data preprocessing section. After preprocessing, an output file, yelp_reviews_train.txt is generated.

Four modules:

Extract.py : This is responsible for reading parameter file and input data.

Model.py : which holds model object for persistence using pickle

Train.py: Has all algorithms that is used for model evaluation. All possible algorithms can be specified in the config file, without changing code. This will create a persistence model using training data set. I persist using pickle.

Predict.py: This has interface predict which takes input reviews text and returns numeric rating.

In the config file, I specify evaluation metrics to be used, training data folder, algorithm used to create model, threshold value below that value, do not predict any rating. Performance metrics used are F1-Matric (Precision, Recall and F1 Score)

Refinement

The process of improving upon the algorithms and techniques used is clearly documented. Both the initial and final solutions are reported, along with intermediate solutions, if necessary.

Added review length feature, below are the result of experiment:

Evaluated the model by splitting it into training and testing sets 80:20 and K-fold validation.
CV tunes hyper parameters and chooses best estimates.

```
cv = KFold(self.x_train.shape[0], n_folds=5, shuffle=True, random_state=0)
```

random_state: When shuffle=True, pseudo-random number generator state used for shuffling.
If None, use default numpy RNG for shuffling.

shuffle : Whether to shuffle the data before splitting into batches.

```
Best estimator LogisticRegression(C=1.0, class_weight='auto', dual=False,  
fit_intercept=True,  
    intercept_scaling=1, max_iter=100, multi_class='ovr',  
    penalty='l1', random_state=None, solver='liblinear', tol=0.0001,  
    verbose=0)
```

Precision on training data = 0.63115625

Precision on test data = 0.549625

Accuracy on training data = 0.63115625

Accuracy on test data = 0.549625

Classification Report

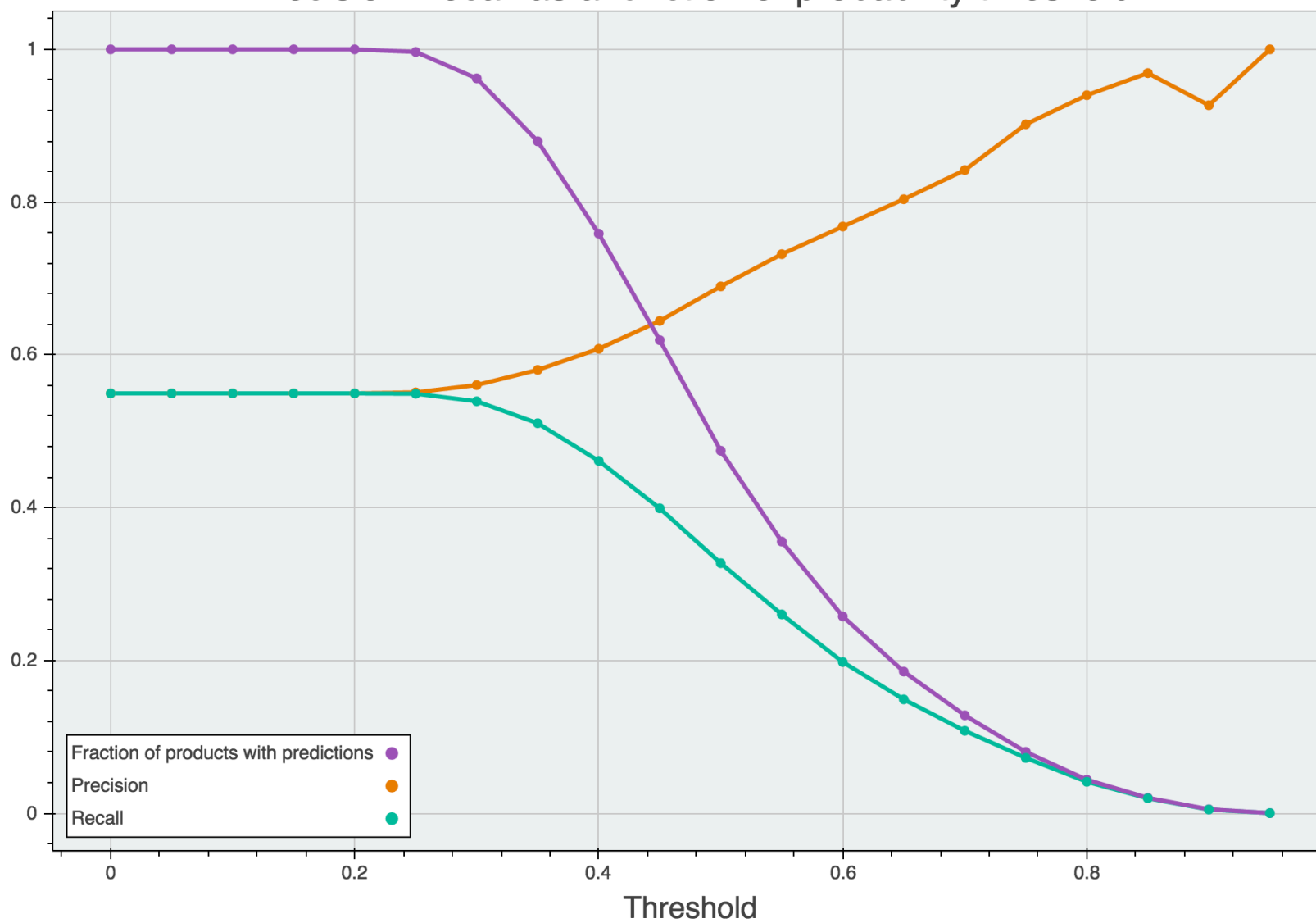
Stars	precision	recall	f1-score	support
1	0.58	0.76	0.66	948
2	0.37	0.37	0.37	810
3	0.45	0.40	0.43	1306
4	0.53	0.45	0.49	2348
5	0.65	0.69	0.67	2588
avg / total	0.54	0.55	0.54	8000

Printing model stats

Overall precision = 0.549625

Actually, after adding text length and votes feature, model performance did not increase.

Precision/Recall as a function of probability threshold



Results

Model Evaluation and Validation

The final model's qualities — such as parameters — are evaluated in detail. Some type of analysis is used to validate the robustness of the model's solution.

Used GridSearchCV to tune hyper parameters.

Tuned input parameters were:

```
tuned_parameters = {'C': 10.0 ** np.arange(-2, 3),  
                    'penalty': ['l1', 'l2'],  
                    'class_weight': [None, 'auto']}
```

After running GridSearchCV, I found that the following values were found for the parameters.

```
Best estimator LogisticRegression(C=1.0, class_weight='auto', dual=False,
fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr',
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0)
```

Output of GridSearchCV as follows:

```
[CV] .. penalty=l2, C=10.0, class_weight=auto, score=0.507743 - 21.4s
[CV] penalty=l2, C=100.0, class_weight=None .....
[CV] .. penalty=l2, C=10.0, class_weight=auto, score=0.500575 - 20.5s
[CV] penalty=l2, C=100.0, class_weight=None .....
[CV] .. penalty=l2, C=10.0, class_weight=auto, score=0.482799 - 19.5s
[CV] penalty=l2, C=100.0, class_weight=None .....
[CV] . penalty=l1, C=100.0, class_weight=None, score=0.497103 - 25.0s
[CV] penalty=l2, C=100.0, class_weight=None .....
[CV] . penalty=l1, C=100.0, class_weight=None, score=0.486825 - 25.4s
[CV] penalty=l2, C=100.0, class_weight=None .....
[CV] . penalty=l1, C=100.0, class_weight=None, score=0.486045 - 26.1s
[CV] penalty=l1, C=100.0, class_weight=auto .....
[CV] . penalty=l1, C=100.0, class_weight=None, score=0.488467 - 27.6s
[CV] penalty=l1, C=100.0, class_weight=auto .....
```

```

[CV] . penalty=l2, C=100.0, class_weight=None, score=0.494077 - 24.3s
[CV] penalty=l1, C=100.0, class_weight=auto .....
[CV] . penalty=l1, C=100.0, class_weight=None, score=0.478051 - 27.6s
[CV] penalty=l1, C=100.0, class_weight=auto .....
[CV] . penalty=l2, C=100.0, class_weight=None, score=0.484346 - 25.1s
[CV] penalty=l1, C=100.0, class_weight=auto .....
[CV] . penalty=l2, C=100.0, class_weight=None, score=0.487463 - 24.7s
[CV] penalty=l2, C=100.0, class_weight=auto .....
[CV] . penalty=l2, C=100.0, class_weight=None, score=0.484251 - 25.2s
[CV] penalty=l2, C=100.0, class_weight=auto .....
[CV] . penalty=l2, C=100.0, class_weight=None, score=0.472903 - 25.0s
[CV] penalty=l2, C=100.0, class_weight=auto .....
[CV] . penalty=l1, C=100.0, class_weight=auto, score=0.494368 - 26.3s
[CV] penalty=l2, C=100.0, class_weight=auto .....
[CV] . penalty=l1, C=100.0, class_weight=auto, score=0.488527 - 24.6s
[CV] penalty=l2, C=100.0, class_weight=auto .....
[CV] . penalty=l1, C=100.0, class_weight=auto, score=0.487787 - 24.5s
[CV] . penalty=l1, C=100.0, class_weight=auto, score=0.487218 - 26.6s
[CV] . penalty=l1, C=100.0, class_weight=auto, score=0.478690 - 25.8s
[CV] . penalty=l2, C=100.0, class_weight=auto, score=0.500404 - 23.0s
[CV] . penalty=l2, C=100.0, class_weight=auto, score=0.492690 - 19.8s

```

Precision on training data = 0.82665625

Precision on test data = 0.577

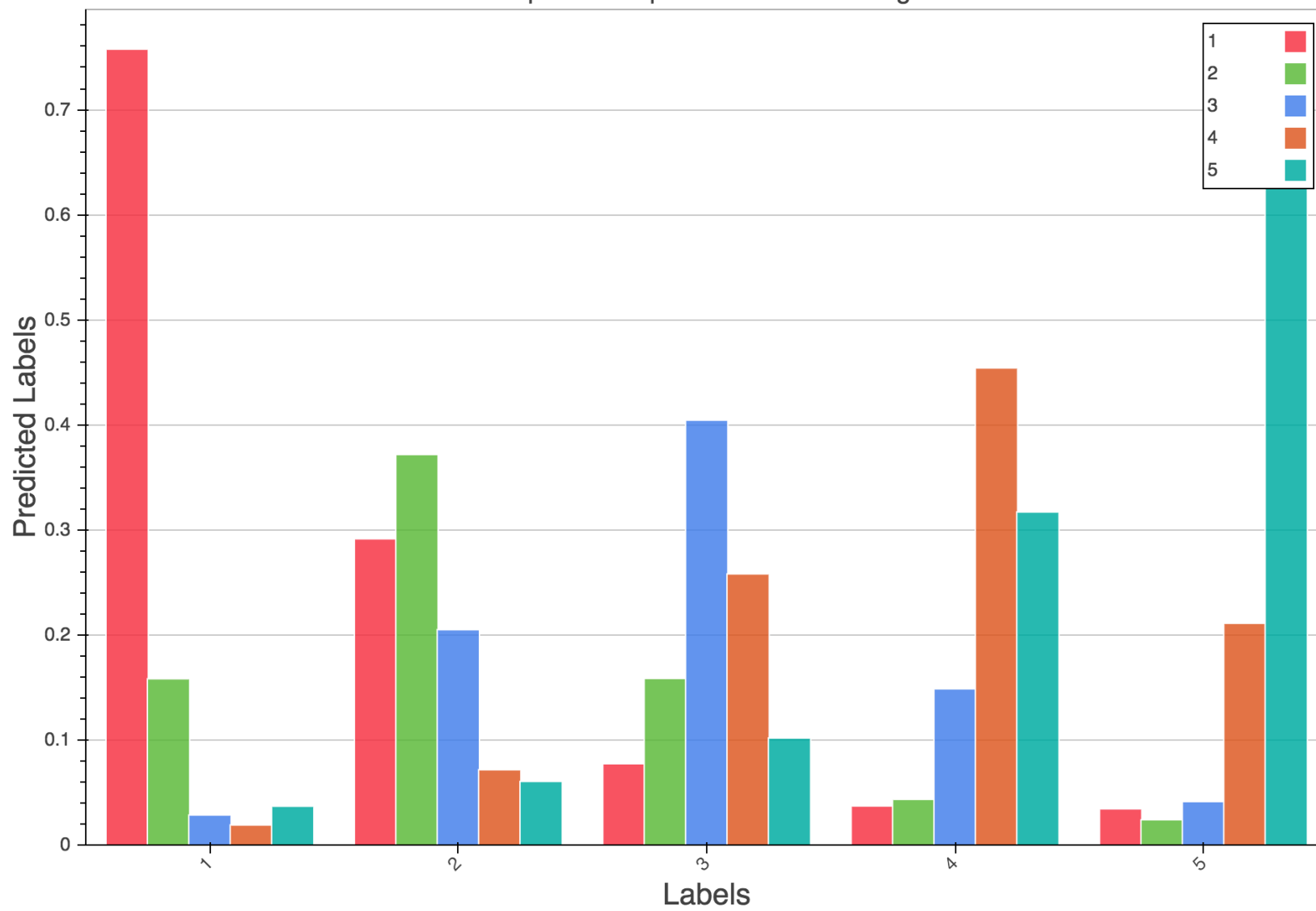
Accuracy on training data = 0.82665625

Accuracy on test data = 0.577

Classification Report					
	Stars	precision	recall	f1-score	support
	1	0.60	0.77	0.67	948
	2	0.43	0.38	0.40	810
	3	0.50	0.40	0.44	1306
	4	0.54	0.52	0.53	2348
	5	0.67	0.71	0.69	2588
avg / total		0.57	0.58	0.57	8000

Confusion matrix plot:

Comparison of predicted vs actual tags



Unnormalized confusion Matrix:

		Predicted labels				
		1	2	3	4	5
A	1	[[718 236 101 87 89]				
c	2	[150 301 207 102 62]				
t	3	[27 166 528 349 107]				
u	4	[18 58 337 1066 546]				
a	5	[35 49 133 744 1784]]				
l						

Justification

The final results are compared to the benchmark result or threshold with some type of statistical analysis. Justification is made as to whether the final model and solution is significant enough to have adequately solved the problem.

I have analyzed two algorithms a) Multinomial Naïve Bayes and b) Logistic Regression. For training data set size 10K, 20K and 40K are tabulated below:

	Precision on Training data	Precision on Test data	Average precision (all labels)	Average recall (all labels)	F1-Score	Overall Precision
10K	0.61925	0. 5265	0.52	0.53	0.52	0.5496
20K	0.6315625	0.5455	0.54	0.55	0.54	0.5455
40K	0.63115625	0.549625	0.54	0.55	0.54	0.5496
Above 40K	Out of CPU limit					

CPU usage exceeds, could not run for training data more than 40K. Logistic Regression performs better as the data points increase.

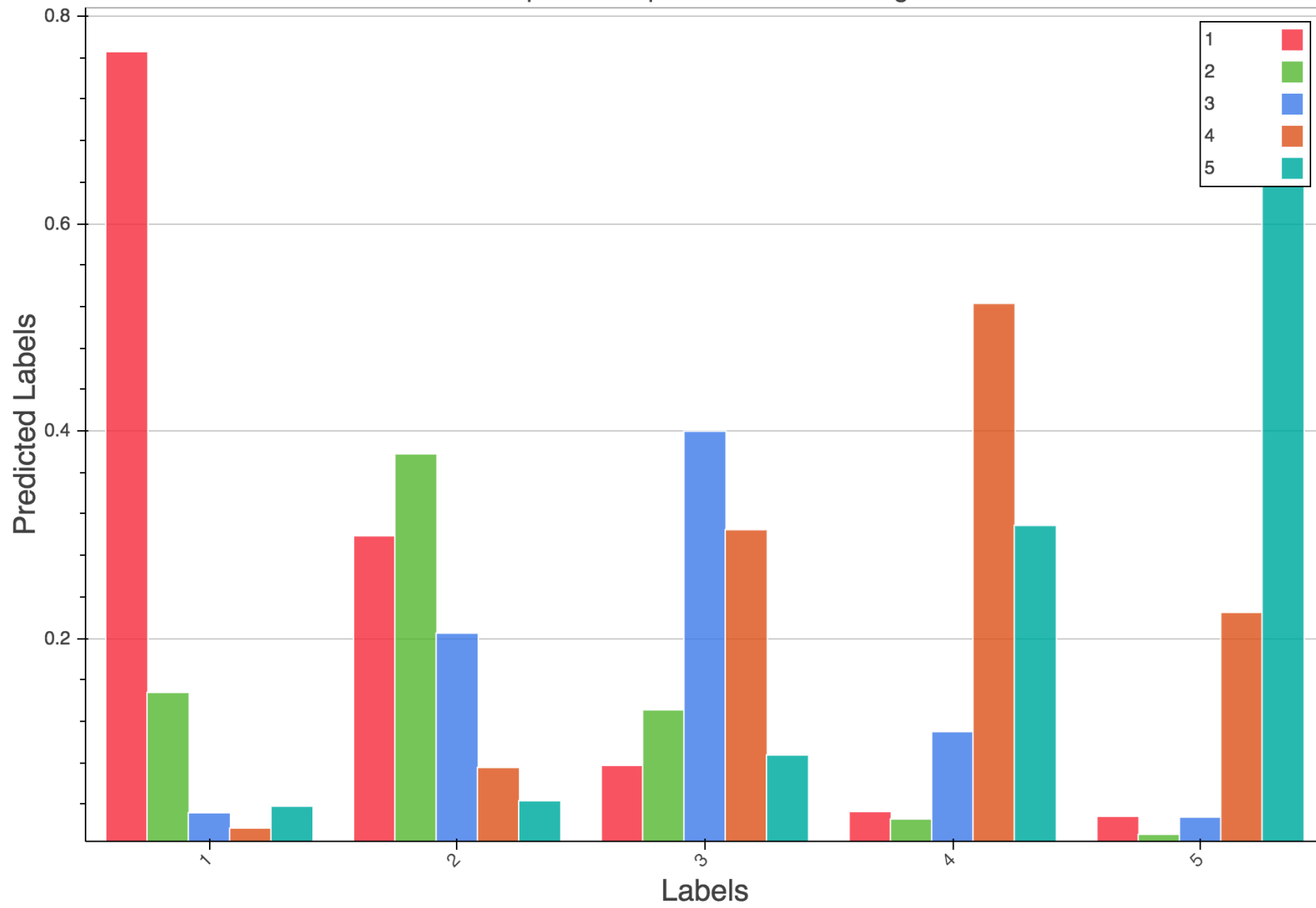
Conclusion

Free-Form Visualization

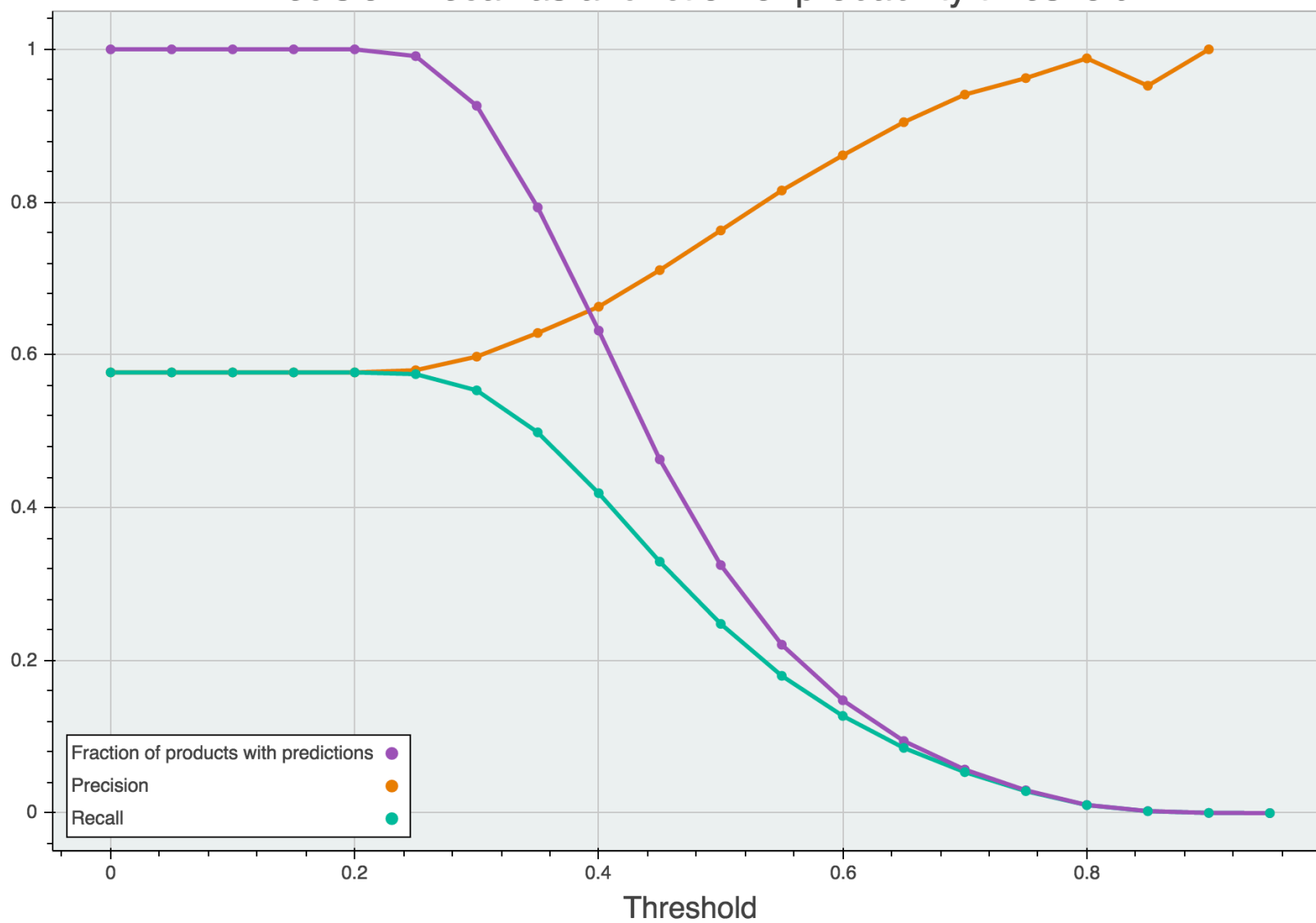
Visualization has been provided that emphasizes an important quality about the project with thorough discussion. Visual cues are clearly defined.

Below is a graph of final confusion matrix, comparison of predicted rating with actual rating. Interestingly, many actual 5 ratings misclassified as 1. Similarly, many actual “4” ratings misclassified as “5”.

Comparison of predicted vs actual tags



Precision/Recall as a function of probability threshold



In the precision recall curve above, I determine at what point prediction probability is optimal that means precision and recall both have optimal value. From the above plot, precision increases to a certain point. Recall graph is constant after certain value. At 0.8 both precision and recall has maximum optimal value beyond this point precision decreases and recall remains constant.

Reflection

Student adequately summarizes the end-to-end problem solution and discusses one or two particular aspects of the project they found interesting or difficult.

As mentioned above, I have taken this project from Yelp Data Challenge. I challenge is to predict a numerical value for a reviews text. Data analysis gave me idea about how many people gives 1 star rating and what are the more popular words in a bad review and vice versa. I have taken approximately same number of data points from each star rating in order to avoid class imbalances. This problem modeled as a classification problem. I compared performance data of two classification models namely Multinomial Naïve Bayes and Logistic Regression. Logistic Regression estimator gave best result. In the process of model performance improvement, added new features a) reviews length and b) Votes. Distribution of reviews_length with numeric rating was plotted. Using GridSearchCV, tuned hyper parameters.

In conclusion, adding new features did not improve accuracy. It might work to find out popular words in one rating reviews and five review ratings and use them as feature, might improve model performance. Another problem I faced was adding more training data above 50K, training process dies. How to improve the precision to at least 90%?

Improvement

Discussion is made as to how one aspect of the implementation could be improved. Potential solutions resulting from these improvements are considered and compared/contrasted to the current solution

Supervised classifiers, generally improve adding more training data. So, if I all of Yelp data, might help to improve precision, recall and F1 score and model accuracy. However, there is a memory limitation. I will study and understand Neural Network in the next course, which might help too

Reference:

Yelp Data challenge

https://www.yelp.com/dataset_challenge

Get start code

<https://github.com/Yelp/dataset-examples>