Book Review Sentiment Analysis

Ray Fregly
Earnest Salgado
CMSC 35300

Abstract

Among many common text categorization tasks is sentiment analysis, the extraction of sentiment, the positive or negative orientation that a writer expresses toward some object (Jurafsky, 2021). We examined this within the context of a bag-of-words binary classification implementation. Using a formatted compilation of book review data from Goodreads.com, we minimized the sum of the squared residuals using X and y, defining wLS as the weight of each word. Words that tend to appear in positive reviews would have positive weights and words that appear more in negative reviews would have negative weights. We also trained our models to make predictions of the total number of stars received in each book review. Finally, we discuss how these results compare with error values we found after employing LASSO regression.

1 Introduction

Our study takes one common text categorization task, sentiment analysis, which involves the mining of text for context that extracts subjective information in source material such as reviews for movies, concerts, or appliances for sale on the web. One can easily see the widespread relevance of this application and how it could be useful in a variety of fields such as finance and politics. A simple version of sentiment analysis is a binary classification task, and the words of reviews provide excellent cues.

Consider the following example of phrases extracted from positive and negative reviews of movies and restaurants (Jurafsky, 2021). Words like great, richly, awesome, and pathetic, and awful and ridiculously are very informative cues:

+ ...zany characters and richly applied satire, and some great plot twists
− It was pathetic. The worst part about it was the boxing scenes...
+ ...awesome caramel sauce and sweet toasty almonds. I love this place!
− ...awful pizza and ridiculously overpriced...

1.1 Multinomial Naïve Bayes Classifier

We explain the intuition of the classifier we chose in a literal bag-of-words example. Suppose all words from a book review are poured into a bag, thus in an unordered set with their position ignored. Sentences and phases entirely are broken up, and only the frequency of the word in the document is noted. We note that the word 'girl' occurred 5 times in the entire excerpt, the word 'they' 6 times, the words 'holiday', 'recommend', and 'powerful' once, and so on.

There are two key assumptions in this case. The first is the bag of words assumption we hinted at above: we assume position doesn't matter, and that the word "they" has the same effect on classification whether it occurs as the 1st, 20th, or last word in the document. Thus we assume that the features f1, f2,..., fn only encode word identity and not position (Jurafsky, 2021). The second is commonly called the naive Bayes assumption: in simple terms it assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. The following equation displays this.

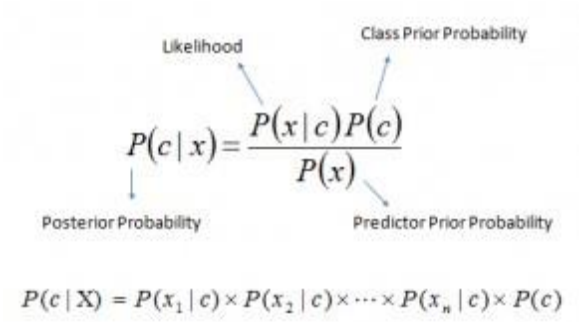

Figure 1: Bayes theorem computes posterior probability. P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes). P(c) is the prior probability of class. P(x|c) is the likelihood which is the probability of predictor given class. P(x) is the prior probability of predictor.

1.2 LASSO Regression

For our dataset we chose lasso, or the 'least absolute shrinkage and selection operator', to run the regression as opposed to ordinary least squares or ridge regression due to the tradeoffs of having a marginal increase in bias for significant decrease in variance for our models. In minimizing the sum of the squared residuals (that is, to fit a line to training data using OLS) the line may not fit the testing data very well. One could then deduce that OLS has low bias but high variance.

If fitting a line to training data using ridge regression, the line may not fit the training data as well as our OLS line. There is more initial bias for less variance. Ridge regression will provide a slightly worse fitting line to training data but better long-term predictions in a given model. Ridge regression components can be essentially understood as least squares, in addition to a penalty for ridge regression. This penalty is expressed as lambda λ (which can be any value from 0 to $+\infty$ and is determined using cross validation) multiplied by the squared slope.

If we again are fitting a line to training data, only this time minimizing the sum of the squared residuals plus λ multiplied by the absolute value of slope, the fitted line is using lasso regression. Similar to ridge regression, we can observe small bias but significantly less variance than OLS regression. Note that the bias in our lasso line is dependent on the value of λ. If $\lambda = 0$, then our lasso regression line will be the same fit on the training data as OLS. As λ increases, the slope gets smaller and can reach 0, making lasso regression ideal when trying to exclude "useless" variables from your equation. This is especially helpful in complicated models with many estimated parameters, such as in predicting book reviews. Ridge regression is a continuous process that shrinks coefficients and hence is more stable: however, it does not set any coefficients to 0 and hence does not give an easily interpretable model (Tibshirani, 1996).

We observe two advantages of lasso regression to OLS. The first is prediction accuracy: the OLS estimates often have low bias but large variance; prediction accuracy can sometimes be improved by shrinking or setting to 0 some coefficients (Tibshirani, 1996). By doing so we sacrifice a little bias to reduce the variance of the predicted values and hence may improve the overall prediction accuracy. The second is interpretation. With a large number of predictors, we often would like to determine a smaller subset that exhibits the strongest effects (Tibshirani, 1996).

2 Methods

2.1 Data Collection and Preparation

The evaluated dataset consists of book reviews scraped from Goodreads with Maria Antoniak and Mealanie Walsh's Goodreads scraper [4]. The first ten pages of reviews were pulled for each of the top ten books on the Most Read Books This Week in the United States list from the week of November 29, 2021. This list was chosen as it contained books with both very positive and somewhat negative overall ratings, providing more variety for our dataset. Each review that was pulled provided an integer rating of 1 to 5 stars and an accompanying text review. The ratings would become our label matrix y and the reviews would become our feature matrix X.

[4] https://github.com/maria-antoniak/goodreads-scraper

In total, 2,511 reviews were pulled to create our dataset. Twenty percent (502 samples) of the reviews were then set aside to create our test set and the remaining eighty percent (2,009 samples) became our training set. Labels were redefined into binary classification such that ratings greater than 3 became 1 and all other ratings became –1.

To prepare our feature matrix, we first had to change the words to vectors. To do this, we appended every review in the training set into one long list that would represent our corpus, with all capital letters changed to lowercase so as to not create false distinctions between capitalized and lowercase versions of the same word. To account for negatives within the data, a NOT_ prefix was prepended to all words following a negative word, such as "not" or "no," until the next phrasal boundary, which we classified as any form of punctuation. As such, a sentence such as "This book was not the best, but I liked it all the same." became "this book was not NOT_the NOT_best , but i liked it all the same . " The words were then tokenized using the nltk package's WordPunctTokenizer and total counts for each word were taken. To create a vocabulary, words with more than two instances within the training data were replaced with an <unk> token to represent values outside the vocabulary. Then, each remaining word was given a unique identification number which would represent its index within a feature vector. In total, there were 11,768 words in the vocabulary of the training data.

Once the vocabulary was created, the text of each review in the training and test sets were converted into feature vectors. Each review was represented as a feature matrix of length 11,768, with each index representing a word in the vocabulary. The count of each word in a given review was taken and recorded at its index within the review's feature matrix. Thus, if the review contained the word "and" twelve times, then the value at the index representative of "and" would be 12 and if the review did not contain the word "excellent", then the value at the index representative of "excellent" would be zero. In this way, the training reviews were represented in a 2,009 by 11,768 matrix with 2,009 samples and 11,768 features.

2.2 LASSO Model

Due to a long runtime when attempting to do LASSO regression from scratch, we used the sklearn LASSO model to train and test our data. After fitting the model to the prepared training data, we had the model predict the labels for the training data, then set the labels to –1 if the value was less than 0 and 1 otherwise. We then took the mean squared error to find the training loss. Similarly, we predicted and refined labels for the test data and took the mean squared error to find the test loss.

As the initial labels were integers, we also trained and tested a LASSO model to predict the star scores of each review. Instead of determining whether each review was positive or negative, the model's goal was to estimate the star rating of each review. We fit the sklearn LASSO model to the data with labels number 1 through 5, then used the model to generate predicted labels for the training and testing data.

2.2 Linear Regression Model

In order to better evaluate our binary classification LASSO model, we also ran a sklearn Linear Regression model on the binary classification label, with similar methods to those described for the LASSO model. We repeated the process of fitting the linear regression model to our prepared training data, and having it set to the same training labels, -1 if the value was negative, and 1 otherwise.

For model comparison we trained and tested the Linear Regression model towards predicting the star scores of each review. We fit model to the data using the sklearn Linear Regression model, and updated our labels to be number 1 through 5.

3 Results

3.1 Comparison of Model Performance

For predictions whether a book review was either positive or negative, the LASSO model training loss was 38.8329756778952 while the Linear Regression model training loss was 2.0. In terms of test loss, the LASSO regression performed significantly better. LASSO test loss was 19.697715603592208, and Linear Regression test loss was 30.659419433511783.

When predicting a star score ranging between 1 and 5 for each book review, the results for LASSO model training loss was 39.94947021280382 and the test loss was 22.013886501596104, while for the Linear Regression model, training loss was 0.7071342341556409 and the test loss was found to be 3115953151.939873.

3.2 Code Findings

We examine our results in the context of our code. In total, there were 11,768 words in the vocabulary of the training data. Processed training reviews were represented in a 2,009 by 11,768 matrix with 2,009 samples and 11,768 features. Additionally, when initializing the Lasso Regression model our code selected an alpha value of 0.01 for both predictions.

4 Discussion

4.1 Comparison of Model Performance

Overall, we found that our LASSO model contained more bias than the Linear Regression model when we minimized the sum of the squared residuals, but also provided much better label predictions in both of our label types.

Evaluating model performance predictions of whether a book review was either positive or negative, there was virtually no observed training loss in the Linear Regression model (2.0), an indication of very small bias. Conversely the higher LASSO model training loss value (38.8329756778952) can be explained by presence of the LASSO penalty term in its loss function equation. The training data is not being penalized in the Linear Regression model, making the model able to fit well onto the training set.

In terms of test loss, the LASSO model performed significantly better. LASSO test loss was 19.697715603592208, and Linear Regression test loss was 30.659419433511783. These values indicate better prediction accuracy from the LASSO model. This can be expected since the regression for LASSO shrinks coefficients or sets them to 0 altogether. This reduces variance in the model. In the case of Linear Regression however, OLS estimates often have low bias but large variance.

In simple terms, the loss function for LASSO regression can be expressed as below:

Loss function = OLS + alpha * Σ (absolute values of the magnitude of the coefficients)

We know that OLS, or the ordinary least squares method, uses parameters of the Linear Regression model as a weighted sum *(e.g. $y = a1x1 + a2x2 + a3x3 + ... + anxn + b$)*. The loss function for LASSO uses this same method, but with the addition of its penalty *(e.g. alpha * summation (absolute values of the magnitude of the coefficients)* (Zou, 2006). Our LASSO model used an alpha value 0.01 while making predictions. The observed differences in training loss values could also be explained by the omission of the y-intercept (e.g. *b*) in the loss function for LASSO Regression. As the number of features was much larger than the number of samples, this penalty addition allowed the LASSO model to generalize to new samples better than the Linear Regresison model

Comparing model performance in predicting a star score ranging between 1 and 5 for each book review, we observed similar performance in one and overfitting in another. The results for LASSO model training loss was 39.94947021280382 and the test loss was 22.013886501596104, while for the Linear Regression model, training loss was 0.7071342341556409 and the test loss was found to be 3115953151.939873.

The LASSO model performed about the same when making label predictions for number of star scores and whether a book review was either positive or negative. This could be attributed to the alpha parameter in both loss function model predictions being equal to 0.01.

Results for the Linear Regression are quite different. Its weight vector was too closely aligned to the training set, and thus when we attempted to make predictions with the model, it negatively impacted the performance of the model on new, unseen test set data. Many of the weights within the weight vector were extremely large, often being of size 10^7 or more in a negative or positive direction. Thus, each word in a given review had a much larger impact on the predicted results than was accurate. These results support LASSO Regression model being a substantially better classifier in the context of language models and especially in book review sentiment.

References

[1] Daniel Jurafsky & James H. Martin. Speech and Language Processing. Draft of September 21, 2021. <https://web.stanford.edu/~jurafsky/slp3/4.pdf>

[2] Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society (Series B), 58, 267-288.
<https://www.ccs.neu.edu/home/eelhami/courses/EE290A/LASSO_Tibshirani.pdf>

[3] The Adaptive Lasso and Its Oracle Properties. Hui Zou. Journal of the American Statistical Association, 2006, vol. 101, 1418-1429 <http://pages.cs.wisc.edu/~shao/stat992/zou2006.pdf>

[4] https://github.com/maria-antoniak/goodreads-scraper

[5] Our group Github link that houses Final Project files, including .csv of words: https://github.com/rfregly/MFML_proj

In [1]:
```
import pandas as pd
import math

#pull reviews
reviews = pd.read_csv("reviews.csv")

#take train data...
num_test = math.floor(len(reviews) * .2)
num_train = len(reviews) - num_test

print(len(reviews))
print(num_test)
print(num_train)
```

```
2511
502
2009
```

For access to the data used in this project (the reviews.csv), please email rfregly@uchicago.edu and the csv will be provided. Thanks!

In [2]:
```
test = reviews.iloc[:num_test]
train = reviews.iloc[num_test:]

print(len(test))
print(len(train))
```

```
502
2009
```

In [52]:
```
test_labels = test['rating'].to_frame()
test_text = test['text'].to_frame()

train_labels = train['rating'].to_frame()
train_text = train['text'].to_frame()
```

In [4]:
```
import nltk
from nltk.tokenize import WordPunctTokenizer

tk = WordPunctTokenizer()
NEG = 'NOT_'
negs = ['not', 'none', 'no', 'impossible', 'n\'t']
punct = [',', '.', '?', '!', ';', '-']

#preps data
def prep(text):
    data_list = []
    for i, data in text.iterrows():
        words = []
        review = data['text']
        listed = review.split()
        neggy = False
        for word in listed:
            w = word.lower()
            if neggy:
                if word in punct:
                    words.append(w)
                    neggy = False
                else:
                    words.append(NEG + w)
            else:
                words.append(w)
                if w in negs:
                    if neggy:
                        neggy = False
                    else:
                        neggy = True
        data_list.append(' '.join(words))
    return data_list

train_list = prep(train_text)
tokens = tk.tokenize(' '.join(train_list))
freq = nltk.FreqDist(tokens)
vocab = {'<unk>':0}
i = 1
for item in freq.keys():
    if item not in vocab:
        if freq[item] > 2:
            vocab[item] = i
            i += 1
print(len(vocab))
```

```
11768
```

In [42]:
```
import numpy as np

def make_matrix(text_list):
    n = len(text_list)
    X = np.zeros((n, len(vocab)))
    for i in range(n):
        review = tk.tokenize(text_list[i])
        for word in review:
            if word in vocab:
                X[i][vocab[word]] += 1
            else:
                X[i][0] += 1
    return X

train_matrix = make_matrix(train_list)

print("train feature matrix shape: ({}, {})".format(len(train_matrix), len(tr
print("train label array shape: ({},)".format(len(train_labels)))
```

```
train feature matrix shape: (2009, 11768)
train label array shape: (2009,)
```

In [43]:
```
#clean test data

test_list = prep(test_text)
test_matrix = make_matrix(test_list)

print("test feature matrix shape: ({}, {})".format(len(test_matrix), len(test
print("test label array shape: ({},)".format(len(test_labels)))
```

```
test feature matrix shape: (502, 11768)
test label array shape: (502,)
```

In [46]:
```
def clean_labels(y):
    for i, val in enumerate(y):
        if val > 3:
            y[i] = 1
        else:
            y[i] = -1
    return y

X_train = train_matrix
y_train = clean_labels(np.ravel(train_labels.to_numpy()))

X_test = test_matrix
y_test = clean_labels(np.ravel(test_labels.to_numpy()))
```

In [31]:
```
y_train.shape
```

Out[31]:
```
(2009,)
```

In [ ]:
```
import time
```

In [48]:
```
#model for pos/negative review

from numpy import arange
from sklearn.linear_model import LassoCV
from sklearn.model_selection import RepeatedKFold

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

model = LassoCV(alphas=arange(0, 1, 0.01), cv=cv, n_jobs=-1)

model.fit(X_train, y_train)

print(model.alpha_)
```

```
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 0.36927251401
766625, tolerance: 0.22715790929203458
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: UserWarning: Coordinate descent with no regulariza
tion may lead to unexpected results and is discouraged.
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.51981859836
9942, tolerance: 0.22920353982300853
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.75228913354
3394, tolerance: 0.22984690265486726
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.53447057450
048, tolerance: 0.23055199115044198
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 1.39768777157
7239, tolerance: 0.23018888274336302
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: UserWarning: Coordinate descent with no regulariza
tion may lead to unexpected results and is discouraged.
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.72640587003
48635, tolerance: 0.22715790929203458
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 0.34926535205
818254, tolerance: 0.2311795907079653
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
```

```
0.01
```

In [49]:
```
y_hat = model.predict(X_train)

def reg_score(y_hat):
    for i, val in enumerate(y_hat):
        if val < 0:
            y_hat[i] = -1
        else:
            y_hat[i] = 1
    return y_hat

y_hat = reg_score(y_hat)
loss = np.linalg.norm(y_hat - y_train)
print(loss)
```

```
38.8329756778952
```

In [50]:
```
y_hat_test = model.predict(X_test)
y_hat_test = reg_score(y_hat_test)
loss = np.linalg.norm(y_hat_test - y_test)
print(loss)
```

```
19.697715603592208
```

In [55]:
```
#model for number of stars review

test_labels = test['rating'].to_frame()
train_labels = train['rating'].to_frame()

X_train = train_matrix
y_train_nums = np.ravel(train_labels.to_numpy())

X_test = test_matrix
y_test_nums = np.ravel(test_labels.to_numpy())

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

nums_model = LassoCV(alphas=arange(0, 1, 0.01), cv=cv, n_jobs=-1)

nums_model.fit(X_train, y_train_nums)

print(nums_model.alpha_)
```

```
[5. 2. 5. 1. 3. 3. 4. 5. 5. 3. 4. 4. 4. 4. 4. 1. 3. 5. 3. 5. 5. 4. 5. 3.
 4. 4. 5. 5. 5. 2. 3. 5. 5. 4. 1. 5. 4. 5. 4. 2. 5. 4. 4. 5. 5. 5. 4. 5.
 3. 4. 2. 4. 4. 4. 5. 4. 4. 5. 5. 2. 5. 5. 5. 5. 4. 4. 5. 4. 3. 1. 2. 3.
 3. 3. 4. 5. 5. 4. 4. 5. 3. 4. 3. 5. 4. 1. 1. 5. 4. 3. 3. 4. 2. 3. 4. 5.
 3. 2. 1. 5. 5. 3. 5. 4. 4. 3. 5. 5. 5. 5. 3. 3. 2. 1. 3. 4. 1. 3. 2. 5.
 2. 2. 3. 4. 4. 5. 5. 3. 5. 4. 5. 3. 5. 5. 4. 2. 4. 4. 5. 4. 4. 2. 2. 3.
 4. 4. 4. 5. 2. 3. 4. 4. 4. 3. 4. 2. 4. 5. 4. 4. 5. 5. 5. 5. 4. 3. 4. 3.
 5. 5. 5. 5. 4. 3. 2. 5. 3. 5. 5. 5. 5. 5. 5. 4. 4. 1. 4. 4. 5. 5. 5. 4.
 4. 2. 4. 5. 5. 4. 4. 5. 5. 4. 4. 5. 2. 4. 5. 4. 2. 5. 5. 5. 4. 4. 2. 4.
 1. 5. 5. 5. 5. 5. 5. 1. 5. 5. 4. 4. 5. 4. 5. 5. 2. 2. 4. 5. 4. 1. 5. 1.
 4. 5. 5. 5. 5. 5. 5. 3. 4. 5. 4. 2. 3. 2. 4. 4. 5. 5. 3. 1. 5. 4. 4. 4.
 5. 3. 5. 4. 4. 4. 4. 4. 5. 5. 5. 5. 5. 5. 5. 5. 5. 4. 5. 5. 4. 4. 4. 5. 5.
 5. 4. 5. 5. 5. 5. 5. 5. 5. 4. 5. 5. 5. 5. 5. 4. 5. 5. 5. 5. 4. 5. 4. 5.
 5. 1. 5. 3. 4. 5. 4. 5. 4. 4. 4. 4. 5. 4. 4. 5. 5. 4. 4. 5. 5. 3. 4. 4.
 4. 4. 5. 5. 4. 5. 5. 3. 4. 5. 5. 4. 4. 5. 4. 4. 5. 4. 4. 4. 4. 3. 5. 4.
 4. 4. 5. 5. 4. 4. 4. 5. 4. 5. 5. 4. 5. 5. 5. 5. 5. 5. 4. 4. 5. 5. 5. 5.
```

```
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 1.37564917623
14096, tolerance: 0.22701852876106152
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 0.62713712177
10566, tolerance: 0.2343289823008853
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 0.47951130401
33856, tolerance: 0.22600398009950282
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.74146823720
2128, tolerance: 0.22448119469026515
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.55621557929
63885, tolerance: 0.23581393805309672
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.49603893353
2033, tolerance: 0.22873185840707966
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 2.50470109867
53477, tolerance: 0.2343289823008853
  model = cd_fast.enet_coordinate_descent(
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 4.74385138399
7208, tolerance: 0.22600398009950282
  model = cd_fast.enet_coordinate_descent(
0.01
/Users/rayfregly/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_
coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. Yo
u might want to increase the number of iterations. Duality gap: 0.30024821027
83688, tolerance: 0.25375311100049797
  model = cd_fast.enet_coordinate_descent(
```

In [57]:

```
y_hat_nums = nums_model.predict(X_train)

loss = np.linalg.norm(y_hat_nums - y_train_nums)
print(loss)

y_hat_test_nums = nums_model.predict(X_test)
loss = np.linalg.norm(y_hat_test_nums - y_test_nums)
print(loss)
```

```
39.94947021280382
22.013886501596104
```

In [1]:
```
import pandas as pd
import math

#pull reviews
reviews = pd.read_csv("reviews.csv")

#take train data...
num_test = math.floor(len(reviews) * .2)
num_train = len(reviews) - num_test

print(len(reviews))
print(num_test)
print(num_train)
```

```
2511
502
2009
```

In [2]:
```
test = reviews.iloc[:num_test]
train = reviews.iloc[num_test:]

print(len(test))
print(len(train))
```

```
502
2009
```

In [3]:
```
test_labels = test['rating'].to_frame()
test_text = test['text'].to_frame()

train_labels = train['rating'].to_frame()
train_text = train['text'].to_frame()
```

In [4]:
```
import nltk
from nltk.tokenize import WordPunctTokenizer

tk = WordPunctTokenizer()
NEG = 'NOT_'
negs = ['not', 'none', 'no', 'impossible', 'n\'t']
punct = [',', '.', '?', '!', ';', '-']

#preps data
def prep(text):
    data_list = []
    for i, data in text.iterrows():
        words = []
        review = data['text']
        listed = review.split()
        neggy = False
        for word in listed:
            w = word.lower()
            if neggy:
                if word in punct:
                    words.append(w)
                    neggy = False
                else:
                    words.append(NEG + w)
            else:
                words.append(w)
                if w in negs:
                    if neggy:
                        neggy = False
                    else:
                        neggy = True
        data_list.append(' '.join(words))
    return data_list

train_list = prep(train_text)
tokens = tk.tokenize(' '.join(train_list))
freq = nltk.FreqDist(tokens)
vocab = {'<unk>':0}
i = 1
for item in freq.keys():
    if item not in vocab:
        if freq[item] > 2:
            vocab[item] = i
            i += 1
print(len(vocab))
```

```
11768
```

In [5]:
```
import numpy as np

def make_matrix(text_list):
    n = len(text_list)
    X = np.zeros((n, len(vocab)))
    for i in range(n):
        review = tk.tokenize(text_list[i])
        for word in review:
            if word in vocab:
                X[i][vocab[word]] += 1
            else:
                X[i][0] += 1
    return X

train_matrix = make_matrix(train_list)

print("train feature matrix shape: ({}, {})".format(len(train_matrix), len(tr
print("train label array shape: ({},)".format(len(train_labels)))
```

```
train feature matrix shape: (2009, 11768)
train label array shape: (2009,)
```

In [6]:
```
#clean test data

test_list = prep(test_text)
test_matrix = make_matrix(test_list)

print("test feature matrix shape: ({}, {})".format(len(test_matrix), len(test
print("test label array shape: ({},)".format(len(test_labels)))
```

```
test feature matrix shape: (502, 11768)
test label array shape: (502,)
```

In [7]:
```
def clean_labels(y):
    for i, val in enumerate(y):
        if val > 3:
            y[i] = 1
        else:
            y[i] = -1
    return y

X_train = train_matrix
y_train = clean_labels(np.ravel(train_labels.to_numpy()))

X_test = test_matrix
y_test = clean_labels(np.ravel(test_labels.to_numpy()))
```

In [8]:
```
# OLS

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import RepeatedKFold

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

lin_regressor=LinearRegression()

mse=cross_val_score(lin_regressor,X_train,y_train,scoring='neg_mean_squared_e
mean_mse=np.mean(mse)
print(mean_mse)
```

```
-3.2562075043509395e+19
```

In [9]:
```
X_train = train_matrix
y_train_LS = np.ravel(train_labels.to_numpy())

X_test = test_matrix
y_test_LS = np.ravel(test_labels.to_numpy())
```

In [10]:
```
lin_regressor.fit(X_train, y_train)
lin_regressor.predict(X_train)
```

Out[10]:
```
array([ 1.00026008,  1.00009486, -0.99994023, ...,  1.00014421,
        1.00006762,  1.00010351])
```

In [11]:
```
# loss = np.linalg.norm(y_hat_test_OLS - y_test_OLS)
```

In [12]:
```
y_hat_LS = lin_regressor.predict(X_train)

def reg_score(y_hat):
    for i, val in enumerate(y_hat):
        if val < 0:
            y_hat[i] = -1
        else:
            y_hat[i] = 1
    return y_hat

y_hat_LS = reg_score(y_hat_LS)
loss = np.linalg.norm(y_hat_LS - y_train)
print(loss)
```

```
2.0
```

In [13]:
```
y_hat_test = lin_regressor.predict(X_test)
y_hat_test = reg_score(y_hat_test)
loss_test = np.linalg.norm(y_hat_test - y_test)
print(loss_test)
```

```
30.659419433511783
```

In [14]:
```
#model for number of stars review

test_labels = test['rating'].to_frame()
train_labels = train['rating'].to_frame()

X_train = train_matrix
y_train_LS_nums = np.ravel(train_labels.to_numpy())

X_test = test_matrix
y_test_LS_nums = np.ravel(test_labels.to_numpy())
```

In [15]:
```
lin_regressor.fit(X_train, y_train_LS_nums)

lin_regressor.predict(X_train)
```

Out[15]:
```
array([4.00013108, 4.0000583 , 1.00003934, ..., 5.00007067, 5.00004654,
       5.00005964])
```

In [16]:
```
y_hat_LS_nums = lin_regressor.predict(X_train)
loss = np.linalg.norm(y_hat_LS_nums - y_train_LS_nums)
print(loss)

y_hat_testLS_nums = lin_regressor.predict(X_test)
loss_nums = np.linalg.norm(y_hat_testLS_nums - y_test_LS_nums)
print(loss_nums)
```

```
0.7071342341556409
3115953151.939873
```

In [18]:
```
y_hat_testLS_nums
```

Out[18]:
```
array([ 8.81191033e+07, -4.29095030e+07, -1.65510144e+08, -6.37107753e+07,
       -2.34130515e+08,  1.68148313e+08, -6.69357742e+07, -1.31499365e+08,
       -3.00946966e+07, -4.48755504e+07, -4.41841789e+08,  5.63314234e+08,
       -5.90015601e+06,  1.73552863e+08,  1.37735323e+08, -1.41683882e+07,
        3.02957867e+08,  8.00071194e+07,  6.25250712e+07,  4.00685101e+07,
        8.05647161e+07, -3.91490366e+08, -1.64211359e+08,  8.20852196e+07,
       -3.38024140e+07,  1.77102380e+07, -5.62557526e+07, -1.63592476e+08,
        1.24730498e+06, -4.21856235e+07, -1.81433287e+08, -1.25236496e+08,
       -1.27240668e+06,  4.70012809e+07,  9.89770402e+07, -9.69699822e+07,
       -1.37200706e+07,  6.27930882e+07,  1.50196159e+08, -2.38523702e+08,
        1.43035691e+08, -4.72908803e+08, -1.01426425e+08,  7.03370424e+07,
        4.49049569e+07,  1.70336895e+08,  2.16654281e+07, -3.48306734e+07,
        1.25163614e+07,  2.11939354e+08, -3.07568236e+07,  2.20114231e+08,
        2.46192129e+08, -2.59417824e+08,  9.36090162e+05, -2.08688552e+08,
        2.91498794e+07,  1.15085317e+08, -6.65020712e+07,  6.12657255e+07,
        3.04719301e+08, -1.43168297e+07,  9.92086814e+06, -2.34822245e+08,
        7.17557183e+07,  6.07934343e+07, -1.62174164e+07, -1.81799009e+08,
        4.41115402e+07,  3.25611056e+07,  1.45279239e+07,  2.58068317e+08,
        3.97722480e+08,  1.89466386e+08,  9.44810579e+07,  7.01700609e+07,
       -3.43571422e+06, -1.60050392e+07,  2.50469595e+08,  1.03562499e+07,
        1.11723737e+08, -5.07729997e+07,  3.05601066e+07,  3.70413523e+07,
       -1.00156829e+07, -1.17917995e+07, -2.59110459e+08, -9.03416055e+07,
        1.77668956e+08, -1.26471734e+08, -4.40339973e+07,  9.51337197e+07,
        5.82398027e+06,  2.75653009e+08, -3.96536605e+07, -7.18433782e+07,
       -2.57045431e+07,  1.32725984e+08, -1.46362147e+08,  3.22723933e+08,
        8.57897468e+07,  1.55134357e+08,  1.56260603e+08, -2.12017572e+08,
        6.65913730e+05,  2.25539129e+06,  7.17890899e+07, -1.30470330e+07,
```