ASSIGNMENT 2

October 2, 2019

Bar Kadosh (bk497), Ben Kadosh (bk499)
CS5785: Applied Machine Learning
Instructor: Nathan Kallus, Teaching Assistants: Xiaojie Mao, Yichun Hu

Contents

0.1 Summary 2
0.2 Problem 1: Regularized Linear Regression 4
0.3 Problem 2: Sentiment Analysis of Online Reviews 13
0.4 Problem 3: Written Exercises 29

0.1 Summary

Our general approach to the coding sections of this assignment was to cleanse and transform the data as needed to make it easy to use and provide the greatest insights. Below we describe the approach in more detail for each problem.

For the House Pricing exercise, our approach was first to review the dataset and determine if there were any features that could be dropped due to a large amount of NaN values. We also looked for features that were administrative to the dataset but weren't relevant for predicting House Prices. In doing so, we found five columns with numerous NaN values, which we dropped, along with the Id feature. We then encoded all categorial features so that we could run the different regression models on the data. We then filled in NaN values with median values from each respective feature where applicable. In addition, we used standard scaler to transform the data for the regression where applicable.

Furthermore, we analyzed the correlation between all features with SalePrice to plot a scatter matrix, and split the training data to run the different regression models with cross validation where applicable. We present the results below. We also added quadratic features to the dataset to see how they would impact different metric scores, and analyzed which variables were retained by certain models. Finally, we submitted our results to Kaggle for our top performing classifier, Ridge, which resulted in a RSMLE score of **0.158**.

For the Sentiment Analysis of Online reviews exercise, our first step was to transform the data. To do so, we chose to convert all words to lowercase as we saw that there wasn't a significant amount of reviews with all uppercase words, and that even when they were, their sentiment was still likely to be captured by the word itself. We then stripped punctuation, much for the same reason as lowercase, as we felt that "awful" and "awful!" should be captured in the same fashion and not as two separate words. We then removed stop words under the assumption that filler words wouldn't be the words driving sentiment in reviews and that certain stronger words were the ones we were looking for. Lastly, once we had made all those changes, only then did we lemmatize the words to group together similar words. Once the data was in the desired format, we split the dataset.

For building bag of words, we identified all of the unique words in our training set of reviews and added them to a dictionary. Similarly, for 2-Gram we identified all of the successive 2-word pairs in our training set of reviews and added them to a separate dictionary. We then iterated over our test and train reviews to convert the text into feature vectors with the same length of the respective dictionary and added counts of how many times that feature's associated word appeared in that train or test review text. We then log normalized those counts in both dataframes of feature vectors (one for Bag of Words, one for 2-Gram) and built and ran Naive Bayes Classification on the test sets to predict sentiment. The method we used for Naive Bayes Classification will be discussed in much more detail in section 3.

Finally we analyzed the accuracy of our Bag of Words and N-Gram Naive Bayes predictions and of our Bag of Words and N-Gram Logistic Regression predictions. We were able to achieve the following accuracy values:

1. Naive Bayes Bag of Words Accuracy = **0.7364**
2. Naive Bayes 2-Gram Accuracy =**0.6091**
3. Logistic Regression (L2) Bag of Words Accuracy = **0.7930**
4. Logistic Regression (L1) 2-Gram Accuracy = **0.6479**

An analysis of which words and 2-grams are better predictors of whether a review is positive or negative will be discussed in more detail in section 3. However, to assess this, we used the coefficients from logistic regression for each feature (word or 2-gram) and isolated the ones with the highest magnitude, as they were influencing the sentiment the most.

For the written exercises, our approach, reasoning, and calculations are all explained in detail in section 4.

0.2 Problem 1: Regularized Linear Regression

(a) Join the House Prices: Advanced Regression Techniques competition on Kaggle. Download the training and test data. The competition page describes how these files are formatted.

(b) Tell us about the data. How many samples are there in the training set? How many features? Which features are categorical?

There are 1,460 samples in the training set, as well as 81 features (including Id and SalePrice).

There were 47 categorical features in the data set, which are listed below:

MSSubClass, MSZoning, Street, Alley, LotShape, LandContour, Utilities, LotConfig, LandSlope, Neighborhood, Condition1, Condition2, BldgType, HouseStyle, OverallQual, OverallCond, RoofStyle, RoofMatl, Exterior1st, Exterior2nd, MasVnrType, ExterQual, ExterCond, Foundation, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, Heating, HeatingQC, CentralAir, Electrical, KitchenQual, Functional, FireplaceQu, GarageType, GarageFinish, GarageQual, GarageCond, PavedDrive, PoolQC, Fence, MiscFeature, MoSold, SaleType, SaleCondition

(c) What variables seem to be important? Which seem to correlate with the sale price? Plot the relationship between sale price and year of sale, garage area, lot area, and other variables of your choice. Choose 7 variables and, along with the response variable, make a scatterplot matrix (hint: look at pandas.plotting.scatter_matrix or seaborn.pairplot). Explain what you see.

As part of our data cleanse, we dropped the features "Alley", "FireplaceQu", "PoolQC", "Fence", and "MiscFeature" because each had between 690 - 1453 NaN values, and therefore with such a large portion of the samples having no value, we felt these features would be poor or misleading indicators if used. Additionally, we dropped Id as it was a unique Id for each sample but was not relevant for predicting SalePrice.

In terms of correlation to SalePrice, we calculated the correlation between each feature and SalePrice (see Jupyter Notebook attached for all values). Some of the features that had the largest magnitudes of correlation were "ExterQual", "BsmtQual", "GrLivArea", and "OverallQual".

For the Scatter Matrix, we plotted the features with large positive/negative correlation along with the features noted in the question. The features are listed below:

["SalePrice", "YrSold", "GarageArea", "LotArea","ExterQual", "BsmtQual", "GrLivArea",and "OverallQual"]

As can be seen in the Scatter Matrix, YrSold appears to not be correlated with SalePrice, which is understandable as housing prices may go up slightly year over year, barring recessions. However, in general, year sold will not tell how much a house sold for. Garage Area, GrLivArea, and OverallQual have positive correlations, which makes sense, while ExterQual and BsmtQual have negative correlations. LotArea appears to have a slightly positive correlation, but is less defined.

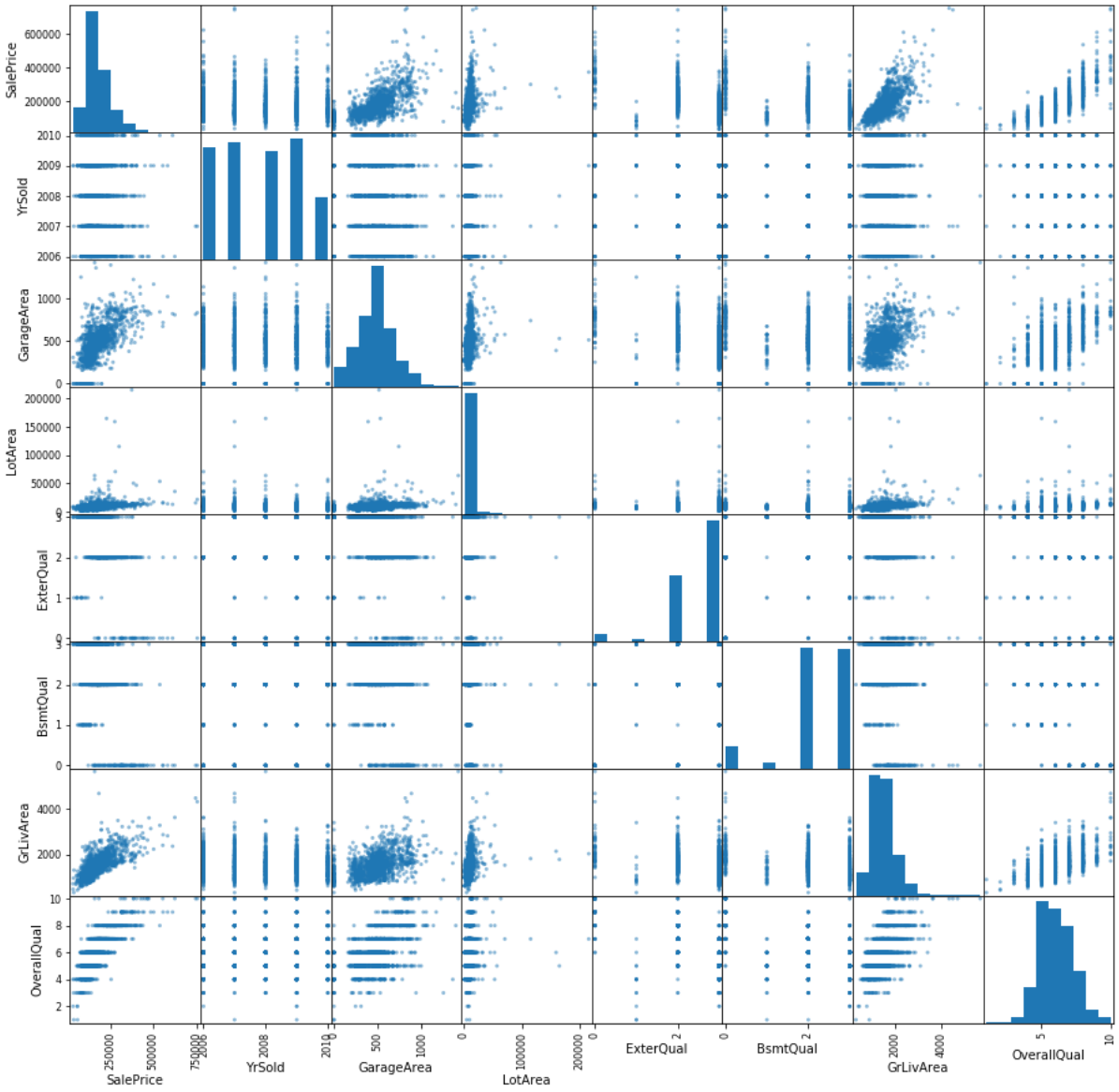

Figure 1: Scatterplot Matrix

(d) (optional) Using statsmodels, run ordinary least squares on all the features and report which features have a 95 percent confidence interval that contains 0 and which do not. Comment on what this means.

(e) Split the training data into a training (80 percent) and test set (20 percent). Try to run a variety of regression methods using sklearn methods:

- Ordinary least squares
- k-Nearest Neighbors with 10-fold cross validation to choose k
- Ridge regression with 10-fold cross validation to chose λ
- LASSO with 10-fold cross validation to chose λ
- Backward stepwise (linear) regression with 10-fold cross validation to choose k (number of features)
- Forward stepwise (linear) regression with 10-fold cross validation to choose k (number of features)

Since each feature in this dataset is measured in completely different units and dimensions (e.g., LotArea vs LotFrontage), make sure that for regularized models (ridge and LASSO) you standardize your data first so that coefficients of smaller units are not unfairly penalized relative to coefficients of bigger units (hint: use sklearn.preprocessing.StandardScaler). (Extra hint: sklearn also has methods called RidgeCV and LassoCV.) For each, give a brief description of how well it works and why that might be and report the test accuracy (note: this refers to split off test set above; not the Kaggle test set).

> Given that the exercise here is to predict SalePrice (a regression exercise), and not a classification exercise, our measure of effectiveness/accuracy can't be the number of correct predictions, because the likelihood that we predict the exact price is extremely low. Therefore, we used the mean squared error (MSE) and Root Mean Squared Log Error (RMSLE), amongst others, as metrics for accuracy (where we want to find the models that have the smallest MSE/RMSLE). To better understand the returned scores for each model, we also explored root mean square error (RMSE), by taking the square root of the mean squared error value, to understand on a more intuitive level the average difference in our predicted SalePrice and the actual SalePrice.

Two further items to note:

1. Sklearn does not have a direct mean squared error scoring method; rather, it has neg mean squared error. To determine the best models, we wanted to maximize the neg mean squared error (find the least negative error).
2. For the backward and forward stepwise regressions, we utilized the Sequential Feature Selector from the mlxtend library, as suggested on the course slack channel.

Lastly, our RMSLE for each of the models (best to worst) is listed below (more detail can be found in the attached jupyter notebook):

- Ridge Regression = 0.164
- Lasso Regression = 0.172
- Forward Stepwise = 0.186
- OLS = 0.191
- Backward Stepwise = 0.198
- kNN = 0.287

For Ridge and Lasso, we expected better performance than the other methods for several reasons. The first reason is that these two methods use regularization, which works to prevent overfitting and find a good balance in the bias-variance tradeoff. The second reason is that similarly to forward and backward stepwise, Ridge and Lasso work to reduce the magnitude of certain coefficients, with Lasso eventually setting certain coefficients to 0 and Ridge making them very small.

We believe Ridge performed a bit better than Lasso because it didn't fully remove features by setting their coefficients to 0, but rather by reducing their value enough to not have a major influence, but still have some where applicable.

Forward stepwise performs quite well, which we expected, as it works to iteratively find the features that contribute the most to a better fit. We

believe this is a better approach for the dataset in hand, as we want to find those features that have most influence on predicting an accurate SalePrice.

OLS performed fairly well, which makes sense given the performance of Ridge. In Ridge we are using lambda to tune the coefficients of OLS, which gives us better performance. OLS has no penalties through lambda to change the coefficients; thus, it still performs fairly well, as the linear regression fits the data fairly well, and when tuned, even better as we see with Ridge.

Backward Stepwise performs fairly well, similarly to some of the other methods listed above, and noticeably better than kNN. Backward stepwise is more effective than kNN because by definition, it looks for the optimal combination of features to best predict SalePrice. However, by design, this method works backward and removes those features which help the least. This is different, and in some ways can be argued, less effective in this case than forward stepwise, which seeks to add those features which improve the prediction rather than remove those that help the least.

kNN performs the worst of all of the different methods used. It makes sense that kNN performs poorly in comparison becuase where other methods have ways to eliminate or reduce less relevant features, kNN does not. Therefore, by keeping more features, the k nearest neighbors may not actually be very close and effective in predicting SalePrice.

(f) Repeat the above, except for OLS, after adding all the quadratic features: X_{ij}, X_{ik} for all j,k = 1,...,p (this includes X_{ij}^2). (Backward/forward stepwise regression are optional).

When we added quadratic features (using PolynomialFeatures) to kNN, Ridge, and Lasso, our RMSLE and R squared scores (respectively) were as follows:

- kNN = 0.266, 0.622
- Ridge = 0.221, 0.859
- Lasso = 0.228, 0.892

The takeaways from the scores above are that Ridge and Lasso performed worse based on RMSLE scores and kNN was still poor, while R squared scores went up for each category. That means that adding the quadratic features did help to better explain the proportion of variance for SalePrice in the model. Furthermore, while the R squared score went up, we believe the RMSLE did not improve, in part, because we did not have enough samples to accomodate the number of features added in an effective manner.

(g) Which variables are being retained by LASSO and the stepwise regression models and which are regularized away? Do these variables match your intuitions about which variables are important and which are not? Compare this to (d).

Retained features for forward stepwise:

('MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'MasVnrArea', 'GrLivArea', 'BsmtFullBath', 'Fireplaces', 'GarageCars', 'WoodDeckSF', '3SsnPorch', 'ScreenPorch', 'LandSlope', 'Neighborhood', 'RoofStyle', 'Exterior1st', 'MasVnrType', 'ExterQual', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'CentralAir', 'KitchenQual', 'Functional', 'GarageType', 'GarageFinish', 'PavedDrive')

Retained features for backward stepwise:

('MSSubClass', 'LotFrontage', 'OverallQual', 'OverallCond', 'MasVnrArea', 'GrLivArea', 'BsmtFullBath', 'Fireplaces', 'GarageCars', '3SsnPorch', 'LotShape', 'LandContour', 'LandSlope', 'Neighborhood', 'RoofStyle', 'MasVnrType', 'ExterQual', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'KitchenQual', 'Functional', 'GarageType', 'GarageFinish', 'PavedDrive')

Retained features for lasso:

['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'TotalBsmtSF', '1stFlrSF', 'GrLivArea', 'BsmtFullBath', 'FullBath', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'WoodDeckSF', '3SsnPorch', 'ScreenPorch', 'MSZoning', 'LotShape', 'LandContour', 'Utilities', 'LandSlope', 'Neighborhood', 'Condition2', 'BldgType', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'MasVnrType', 'ExterQual', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'HeatingQC', 'CentralAir', 'KitchenQual', 'Functional', 'GarageType', 'GarageFinish', 'GarageCond', 'PavedDrive', 'SaleCondition']

Most of the retained features across these three sets do match out intuition that features such as overall quality, condition, the year built, number of bathrooms, kitchen quality, central air, garage type, and sale condition would be important data points for users in determining how much they are willing to pay when purchasing a home.

(h) Train your best-performing classifier with all of the training data, and generate test labels on the Kaggle test set. Submit your results to Kaggle and report the accuracy. Is it higher or lower than the cross validation accuracy? Why might that happen? Is it higher or lower than the held-out validation accuracy? Why might that happen?

Our Kaggle results for the Ridge Classifier produced a score of 0.158, in comparison to 0.164 for the held-out validation score, which took the optimal alpha or k based on the best cross validation accuracy. Because the held-out validation score was only fit on 80 percent of the data, we believe that when we fit the Ridge classifier on the entire data set it resulted in a better classifier, which resulted in a better score on Kaggle than the held-out validation score.

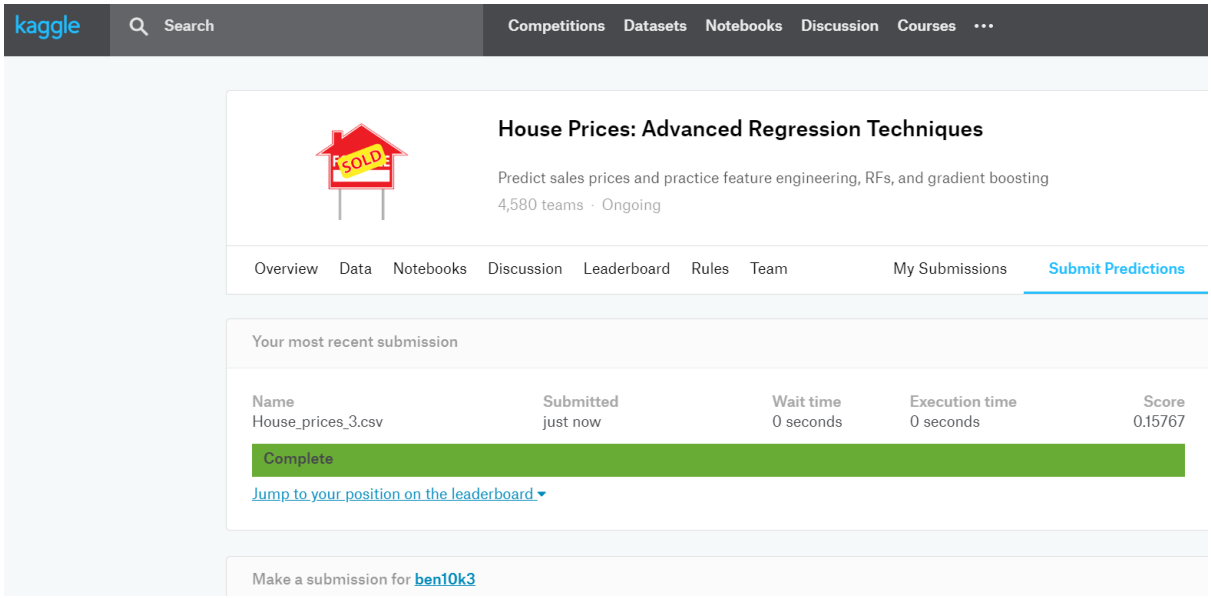

Figure 2: Kaggle Submission Results

0.3 Problem 2: Sentiment Analysis of Online Reviews

In this assignment you will use several machine learning techniques from the class to identify and extract subjective information in online reviews. Specifically, the task for this assignment is sentiment analysis. According to Wikipedia, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. It has been shown that people's attitudes are largely manifested in the language they adopt. This assignment will walk you through the mystery and help you better understand our posts online! Important: Use your own implementations for this question. Any existing bag-of-words or naive Bayes implementations are NOT allowed.

(a) Download Sentiment Labelled Sentences Data Set. There are three data files under the root folder. yelp_labelled.txt, amazon_cells_labelled.txt and imdb_labelled.txt. Parse each file with the specifications in readme.txt. Are the labels balanced? If not, what's the ratio between the two labels? Explain how you process these files.

> For this problem, we read each of the .txt files into pandas dataframes using read_csv(). However, since the reviews and labels were not separated by commas, we specified within the method that they should be separated when tabs appear with the code segment: sep = '\t'. We then called the value_counts() method to identify the counts of each label. In both the yelp and amazon sets, the reviews are balanced (1:1 ratio), as each set has 500 positive reviews and 500 negative reviews. The imdb set, however, is not balanced, as there are 386 positive reviews and 362 negative reviews (193:181 ratio).

(b) Pick your preprocessing strategy. Since these sentences are online reviews, they may contain significant amounts of noise and garbage. You may or may not want to do one or all of the following. Explain the reasons for each of your decision (why or why not).

- Lowercase all of the words.
- Lemmatization of all the words (i.e., convert every word to its root so that all of "running," "run," and "runs" are converted to "run" and and all of "good," "well," "better," and "best" are converted to "good"; this is easily done using nltk.stem).

- Strip punctuation.
- Strip the stop words, e.g., “the”, “and”, “or”.
- Something else? Tell us about it

In order to pre-process our reviews, we made several strategic choices regarding our data. First, we used the lower() method to make all words in our reviews lowercase. We thought about this choice primarily from the view point that we didn't want 'best' and 'Best' to be counted as separate words – indeed, they are the same word, they might just represent capitalization errors or reflect that the word is at a different point in a sentence (starting a sentence vs. not starting a sentence). To avoid such a scenario, we made all the words lowercase so that any words that have the same semantic meaning would be treated as such. One aspect we considered was words that might be written in all capital letters such as 'HATE' or 'LOVE.' We hypothesized that such words might add extra emotion/sentiment to such a word, but we felt that the general sentiment of the word would still be captured. Therefore, we resolved to just lowercase everything.

We then chose to strip all of the punctuation. Again, one aspect we considered is that punctuation such as '!' or '...' might convey additional meaning (excitement or anger, for example). However, such punctuation could be positive or negative – ultimately it really depends on what words it is being used with. Therefore, we decided that the words would be the driving force behind the sentiment, and that we could therefore simplify our data by getting rid of the punctuation. We did this by writing a simple function strip_punctuation().

Next, we decided to strip the stop words – words, such as 'the', 'and', 'or', etc. Such words don't particularly hold much meaning or sentiment – they are just common words in the english language that are needed to properly structure a sentence. Therefore, we decided that we can remove them from our review text, as they most likely hold little to no sentiment value. Again, we did this by writing a simple function remove_stops(), making use of nltk

to download the stop words.

Finally, we decided to use lemmatization to convert the words to their root form. We decided that words with similar roots usually convey similar meaning. This is pretty obvious for words like 'cook' and 'cooking', but also still holds for less obvious examples such as 'better' and 'best.' Since our main goal is to understand sentiment, we felt it made sense to group such words into one group, as their sentiment was very similar. One aspect we considered was for a sentence such as 'could have been better.' In this case, the sentiment is actually negative, but better will be considered a positive words (best, better, well, etc.). However, we reasoned that the negative aspect would be captured through the word 'could', or in the ngram portion through the bigram 'could have.' Therefore, we felt confident using lemmatization. We wrote a function stem_strings() to achieve this, and made use of nltk to stem the words. We waited to do this step last so that we only had to perform lemmatization on the review text once irrelevant words and punctuation had been removed.

(c) Split training and testing set. In this assignment, for each file, please use the first 400 instances for each label as the training set and the remaining 100 instances as testing set. In total, there are 2400 reviews for training and 600 reviews for testing.

Used train_test_split() on our reviews and labels to split our data into 80% training data and 20% testing data.

(d) Bag of Words model. Extract features and then represent each review using bag of words model, i.e., every word in the review becomes its own element in a feature vector. In order to do this, first, make one pass through all the reviews in the training set (Explain why we can't use testing set at this point) and build a dictionary of unique words. Then, make another pass through the review in both the training set and testing set and count up the occurrences of each word in your dictionary. The ith element of a review's feature vector is the number of occurrences of the ith dictionary word in the review. Implement the bag of words model and report feature vectors of any two reviews in the training set.

To begin, I created an empty dictionary to represent the vocabulary we will gather from our training set (all unique words in the training set). By looping through the training reviews, I split each review into its individual words and assigned a value of 0 to that word as a key in the dictionary (no repeat keys are created in dictionaries, so this method will automatically only create entries for unique words).

The reason why we only consider the training set when building our vocabulary is because we want our possible 'vocabulary' to be built on the training data. When we test on our testing data, we want to ignore words that aren't present in the training data. However, if we were to include test words in our vocabulary, those words would no longer be ignored. Additionally, when we train on our training data, any columns representing test words will be a column of 0's, because the word will never be found in the train data. This is extremely problematic, because when we find that test word on a test set, it will calculate that feature probability to be 0 (number of times it will appear in the train data will be 0, and since we no longer ignore it, it will return a probability of 0). This will make the entire probability equal to 0, which will make it very hard for us to properly calculate our probabilities.

Once our dictionary is set up, we convert each review into a feature vector (length of the dictionary/vocabulary) and assign counts for each word (column) based on how many times that word appears. We do this seperately for both the train and test reviews, creating a dataframe for each scenario.

Within our code, two examples of training feature vectors (each representing a text review) can be found printed out. Please see attached jupyter notebook.

(e) Since the vast majority of English words will not appear in most of the reviews, most of the feature vector elements will be 0. This suggests that we need a postprocessing or normalization strategy that combats the huge variance of the elements in the feature vector. You may want to use one of the following strategies. Whatever choices you make, explain why you made the decision.

- log-normalization. For each element of the feature vector x, transform it into $f(x) = \log(x+1)$.
- l1 normalization. Normalize the l1 norm of the feature vector, $\hat{x} = \frac{x}{|x|}$
- l2 normalization. Normalize the l2 norm of the feature vector, $\hat{x} = \frac{x}{||x||}$
- Standardize the data by subtracting the mean and dividing by the variance.

For postprocessing, we selected the log normalization method. We ruled out standardizing the data because we were concerned with the variance between features in the feature vector, and not only at standardizing each feature. Standardizing still wouldn't address variance between features. To do so, we needed to use a normalization method, and we chose log normalization because it used a consistent methodology that wasn't dependent on a feature's range. Furthermore, we felt that log best captured the reduction in magnitude of certain features(words) that would have otherwise driven the huge variance of the elements in the feature vector.

(f) Sentiment prediction. Train a naive Bayes model on the training set and test on the testing set. Report the classification accuracy and confusion matrix.

The basic idea of the naive Bayes model is to take one of our test reviews, and calculate what the probability that it is a positive review given the words the review has and does not have, and the probability that it is a negative review given the words the review has and does not have. Whichever probability is larger determines what we classify the test review as.

To start, we first split our training data into two separate sets of data – one for the positive reviews (label = 1) and one for the negative reviews (label = 0). We then began iterating through each test row/vector, counting how many times each value for that word in the test row appears in the train set for positive reviews and for negative reviews (two separate values). For example, if the value for a word in the test set is 0, we count how many other rows in the train set also have a value of 0 for that given word. For the

counts gathered for positive reviews, each value is divided by the number of positive reviews. Similarly for the counts gathered for negative reviews, each value is divided by the number of negative reviews. One value therefore represents the probability of that value for that given word/column appearing in the positive reviews in the training set, while the other represents the probability of that value appearing in the negative reviews in the training set. For each column in the test set, a probability value is added to a list for positive review comparisons and another probability value is added to a list for negative review comparisons.

To calculate the probability that a test review is a positive review given the words it has and does not have, we then multiply all of the 'positive' probabilities we stored in the step previously discussed for each column and then multiply that product by the probability of being a positive review in the training set (number of positive training reviews divided by the total number of training reviews). We repeat the same to calculate the probability that a test review is a negative review given the words it has and does not have, except we use the 'negative' probabilities stored for each column and multiply by the product of being a negative review in the training set (number of negative training reviews divided by the total number of training reviews).

Lastly, since we do this for each test review, we then compare the two probabilities: if the probability that a test review is a positive review given the words it has and does not have is greater than the probability that a test review is a negative review given the words it has and does not have, then we classify it as a positive (1) review. Otherwise, we classify it as a negative review (0). In the event of a tie, we arbitrarily classify it as a negative (0) review.

After completing this entire process for the whole test set, we had successfully classified each review as either positive or negative. We then compared these predicted labels with the actual labels associated with those reviews and produced an accuracy score of **0.7364.**

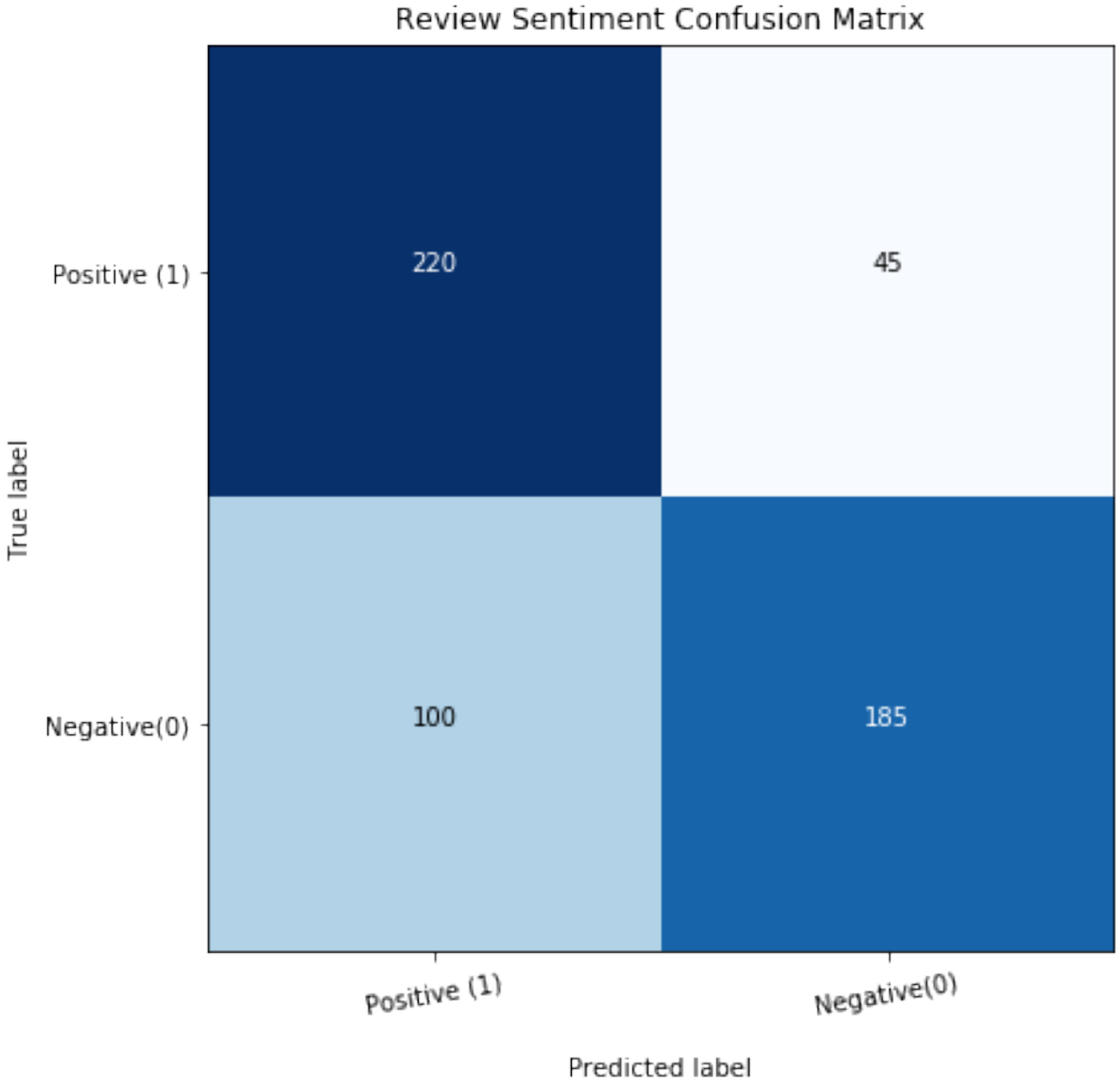

Figure 3: Confusion Matrix For Bag Of Words Model

As can be seen in the confusion matrix, our naive bayes predictor on the bag of words model correctly identifies 220 positive reviews and 185 negative reviews. However, it predicts 100 reviews that are negative to be positive and 45 reviews that are positive to be negative. Therefore, it seems to struggle a bit more with correctly identifying reviews that are negative.

(g) Logistic regression. Now repeat using logistic regression classification, and compare performance (you can use existing packages here). Try using both L2 (ridge) regularization and L1 (lasso) regularization and report how these affect the classification accuracy and the coefficient vectors (hint: sklearn has a method called LogisticRegressionCV; also note that sklearn doesn't actually have an implementation of unregularized logistic regression). Inspecting the coefficient vectors, what are the words that play the most important roles in deciding the sentiment of the reviews?

We used both L1 (lasso) and L2 (ridge) penalties with a logistic regression classification model to also train and test our bag of words data. Aside from just trying both the L1 and L2 penalties, we also played around with different values for C and solver in the logistic regression model to improve our accuracy score. After fitting and scoring our logistic regression model on our bag of words data, the following accuracy scores were achieved for the following two penalties:

- L1 Accuracy = **0.7735**
- L2 Accuracy = **0.7930**

For each of these models, we then used a dictionary to pair each model coefficient with its corresponding column/word and sorted the dictionary so that the most negative values are at the front and the most positive values are at the back. Therefore, the values closest to the front of the dictionary most strongly indicate that a review is negative, while the values closest to the back of the dictionary most strongly indicate that a review is positive. The results that we gathered from our coefficents are:

L1 Top 5 Most Positive Words

1. ('great', 7.369656738251705)
2. ('love', 4.4582370133357285)
3. ('good', 3.8336812829158755)
4. ('nice', 3.464797384325938)
5. ('excel', 3.1903570837336397)

L1 Top 5 Most Negative Words

1. (‘bad’, -4.4111940331587585)
2. (‘disappoint’, -3.2753269639108873)
3. (‘wast’, -2.8690037347973227)
4. (‘worst’, -2.6156568177686097)
5. (‘poor’, -2.5564632673959675)

L2 Top 5 Most Positive Words

1. (‘great’, 5.240750665294592)
2. (‘love’, 3.816390518341043)
3. (‘nice’, 3.53812564040485)
4. (‘delici’, 3.5183454268008028)
5. (‘amaz’, 3.4558254838350924)

L2 Top 5 Most Negative Words

1. (‘bad’, -3.578999012751685)
2. (‘poor’, -3.2642507308388757)
3. (‘worst’, -3.004636783819291)
4. (‘wast’, -2.8630096280696975)
5. (‘disappoint’, -2.5988454220144597)

As can be seen, the coefficient values also vary with whether we use L1 or L2. With L1, the coefficients seem to have a larger magnitude and are more spread (at least for the 5 most positive and 5 most negative reviews), whereas for L2, the coefficients seem to have smaller magnitudes and have less spread.

By inspecting the words associated with the most negative and most positive coefficient values, we can gather a better idea of what words play a big role in influencing whether a review is negative or positive. For instance,

the top 5 negative words for both L2 and L1 are ‘bad’, ‘poor’, ‘worst’, ‘wast’, and ‘disappoint’ (though not both in that same order), where ‘wast’ is most likely representing the word ‘waste’ (might have been altered using lemmatization). Notably, most of these words clearly hold negative sentiment, such as ‘bad’, ‘worst’, and ‘poor’, or convey that something did not live up to expectations, such as ‘wast’ and ‘disappoint.’

Similarly, we can see a similar trend in the positive words, as the top L2 and L1 coefficients are associated with words such as ‘great’, ‘love’, ‘nice’, ‘’delici’, ‘amaz’, and ‘excel’. Again, certain words clearly hold positive sentiment on their own, such as ‘amaz’ (which most likely represents amazing), ‘excel’, ‘nice’, ‘love’, and ‘delici’ (which most likely represents delicious).

(h) N-gram model. Similar to the bag of words model, but now you build up a dictionary of ngrams, which are contiguous sequences of words. For example, “Alice fell down the rabbit hole" would then map to the 2-grams sequence: [“Alice fell", “fell down", “down the", “the rabbit", “rabbit hole"], and all five of those symbols would be members of the n-gram dictionary. Try n = 2, repeat (d)-(g) and report your results.

To build our n-gram data set, we started by building a dictionary that took every unique successive two word pair in the training reviews (for example, ‘Sally Ran Fast’ would be split up into ‘Sally Ran’ and ‘Ran Fast’). To do this, we used the same procedure we used for building up our bag of words vocabulary, except we searched successive two-word pairs instead of just each word. One edge case we accounted for was reviews that were just one word. In such a case, we would append an empty string to the word so that the bigram was just the word and an empty string.

As we did with bag of words, we then used our vocabulary to create feature vectors for each text review that represented the counts of each possible word in the dictionary based on how many times it appeared in the training or test text review. We also again log normalized these feature vectors. Similarly, we also again split our training feature vectors into one set for those with label 0 and another set for those with label 1.

For running the Naive Bayes algorithm, we essentially ran the exact same code/process with the main difference that we were now treating our data as bigrams instead of unigrams. Another key point to note is that we had to make one change to allow our run time to be feasible (as initially it was taking several hours and not outputting anything). To simplify the process, we added a line of code to ignore values in the feature vector that did not have a value of greater than 0. This significantly reduces our run time as we are only looking at words in the test set that appear, and not those that don't. While we know this is not exactly what Naive Bayes is meant to do, we felt this change was justified for two reasons:

1. The run time if we did not do this was simply not feasible. Our best guess was that it would take on the order of over 4-6 hours.
2. In the bag of words model, we actually tested both scenarios – one where we included all values, and one where we only included values greater than 0 (only present words, not those that are not present). Both cases returned almost the exact same accuracy, which makes sense. If we assume that most words are sparse in the english language and will not be used in most reviews, then those feature probabilities will be calculated to be close to 1 if most of the values will be 0. Therefore, there shouldn't be too much of a mathematical difference if we choose to ignore the words that do not appear in the test review, which is why we felt comfortable simplifying the model in this way.

After completing this entire process for the whole test set, we had successfully classified each review as either positive or negative. We then compared these predicted labels with the actual labels associated with those reviews and produced an accuracy score of **0.6091.**

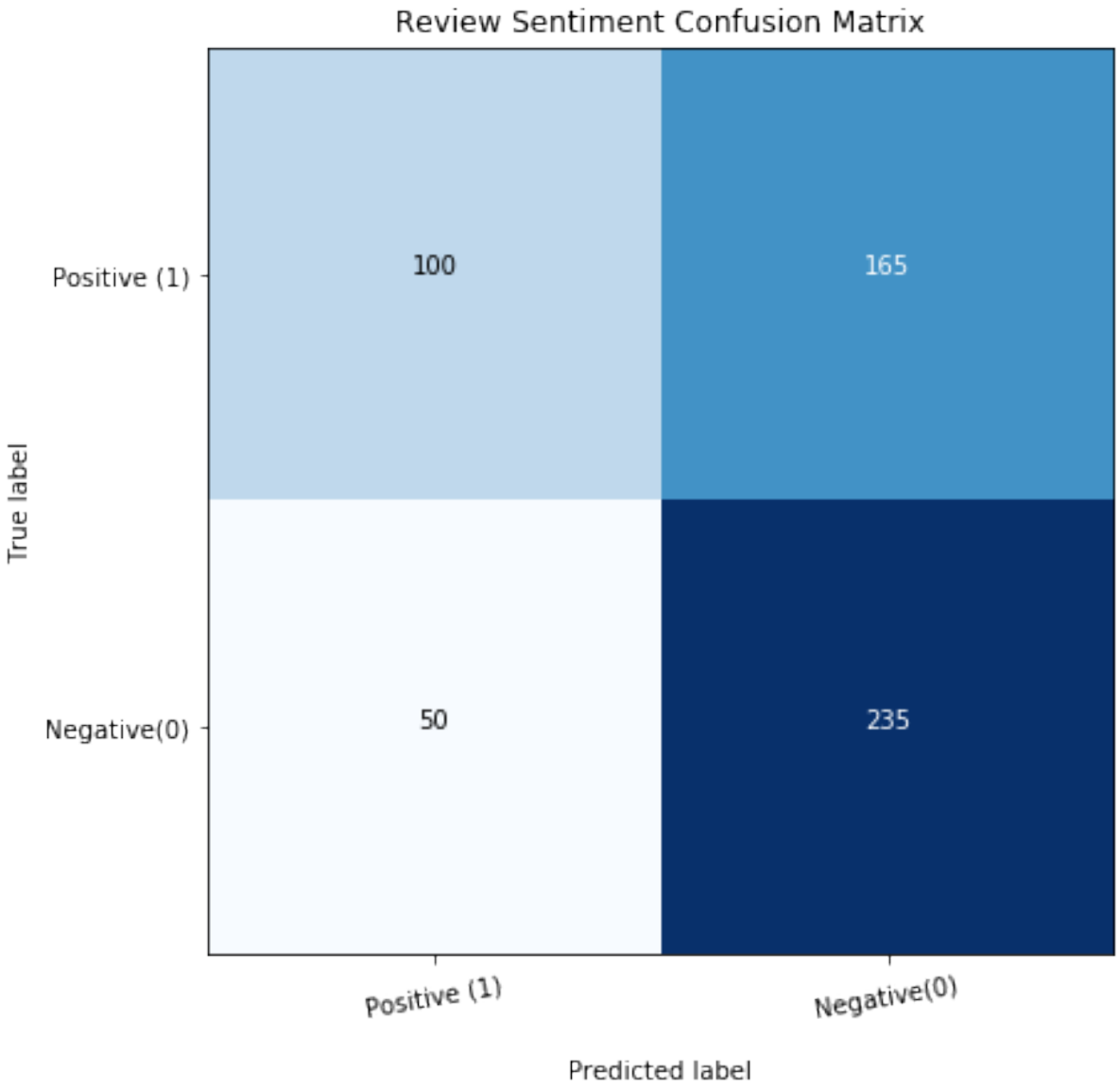

Figure 4: Confusion Matrix For N-Gram Model

As can be seen in the confusion matrix, our naive Bayes predictor on the bag of words model correctly identifies 100 positive reviews and 235 negative reviews. However, it predicts 50 reviews that are negative to be positive and 165 reviews that are positive to be negative. Therefore, it seems to struggle a lot more with correctly identifying reviews that are positive. With the n-gram model, one reason that negative classification might be better is that a seemingly neutral word might convey negative meaning when paired with another word. For example, the word 'could' is most likely neutral in a

vaccuum; however, when paired with the word ‘have’, the 2-word phrase might indicate a sense of regret or a desire for more.

As with Bag of Words, we again used both L1 (lasso) and L2 (ridge) penalties with a logistic regression classification model to train and test our n-gram data. Using a similar method as described for Bag of Words, the following accuracy scores were achieved for the following two penalties:

- L1 Accuracy = **0.6479**
- L2 Accuracy = **0.6457**

In this case, L1 and L2 penalties for logistic regression produce almost identical accuracy scores. Using the same approach described for Bag of Words, we were able to produce the following coefficient results:

L1 Top 5 Most Positive Words

1. ((‘work’, ‘great’), 3.0814713123851654)
2. ((‘great’, ‘phone’), 1.9737156088612244)
3. ((‘highli’, ‘recommend’), 1.7882889156164241)
4. ((‘realli’, ‘good’), 1.4999664974075808)
5. ((‘work’, ‘well’), 1.4473092107629733)

L1 Top 5 Most Negative Words

1. ((‘wast’, ‘money’), -2.414566331219471)
2. ((‘disappoint’, ‘’), -2.0396327952647697)
3. ((‘dont’, ‘wast’), -1.9093117824602133)
4. ((‘wast’, ‘time’), -1.8673128414004267)
5. ((‘dont’, ‘think’), -1.7190480230208236)

L2 Top 5 Most Positive Words

1. (('work', 'great'), 2.719656931842833)
2. (('great', 'phone'), 2.3929133323620735)
3. (('highli', 'recommend'), 1.9559973595915314)
4. (('great', 'product'), 1.826655061015979)
5. (('realli', 'good'), 1.7898537560952201)

L2 Top 5 Most Negative Words

1. (('disappoint', ''), -2.4641914330939643)
2. (('wast', 'money'), -2.3294649074775897)
3. (('wast', 'time'), -1.863128666372419)
4. (('dont', 'think'), -1.8182589371052584)
5. (('wont', 'back'), -1.547983579848017)

As can be seen, the coefficient values also vary with whether we use L1 or L2. With L1, the coefficients for the positive reviews seem to have a larger magnitude and are more spread (at least for the 5 most positive and 5 most negative reviews), whereas for L2, the coefficients for the positive reviews seem to have smaller magnitudes and have less spread. The coefficients for negative reviews for L1 and L2 seem to have similar magnitudes and a similar spread.

By inspecting the bigrams associated with the most negative and most positive coefficient values, we can gather a better idea of what bigrams play a big role in influencing whether a review is negative or positive. For instance, the most negative bigrams for L2 and L1 are ('disappoint', ''), ('wast', 'money'), ('wast', 'time'), ('dont', 'think'), ('wont', 'back'), and ('dont', 'wast'). Notably, most of these bigrams clearly hold negative sentiment or convey that something did not live up to expectations.

Similarly, we can see a similar trend in the positive words, as the top L2 and L1 coefficients are associated with bigrams such as ('work', 'great'), ('great',

'phone'), ('highli', 'recommend'), ('great', 'product'), ('realli', 'good'), and ('work', 'well'). Again, certain bigrams clearly hold positive sentiment, words, and expressions.

(i) Algorithms comparison and analysis. According to the above results, compare the performances of naive Bayes, logistic regression, naive Bayes with 2-grams, and logistic regression with 2-grams. Which method performs best in the prediction task and why? What do you learn about the language that people use in online reviews (e.g., expressions that will make the posts positive/negative)? Hint: Inspect the weights learned from logistic regression.

The following accuracy values were found for our 4 prediction methods:

1. Naive Bayes Bag of Words Accuracy = **0.7364**
2. Naive Bayes 2-Gram Accuracy =**0.6091**
3. Logistic Regression (L2) Bag of Words Accuracy = **0.7930**
4. Logistic Regression (L1) 2-Gram Accuracy = **0.6479**

The first notable result is that Bag of Words performs much better than 2-Gram. Additionally, Logistic Regression also performs better than the Naive Bayes model. Therefore, the method that had the highest accuracy was Logistic Regression (L2) Bag of Words.

There are several reasons for why this might be. For one, Naive Bayes assumes that features are independent, even though they might not be. In some cases, this method is aptly named – it is naive to assume this (as language/words are very much not independent) and this can contribute to a lower accuracy. Therefore, Logistic Regression might be a better approach.

Additionally, Bag of Words isolates each word as its own feature. While this might not be ideal, it makes sense why this might work better than 2-gram. While 2-gram can account for context by looking at phrases instead of just words, only looking at 2 words seems like it might cause more confusion than just looking at 1. More words would be needed in each 'gram' to really

gather context, but additionally, all ‘good’ counts are no longer being counted together. Rather if the word ‘good’ appears 100 times but is followed by 100 different words, we will get 100 separate features with only 1 count, which provides us less insight into the fact that ‘good’ might be a good predictor for positive sentiment.

As already discussed in parts g and h, we learn a lot about what words/2-grams make reviews positive or negative, mainly that words such as ‘worst’ and ‘poor’ inherently hold negative sentiment and words such as ‘nice’ and ‘love’ inherently hold positive sentiment. Additionally, 2-grams such as (‘dont’, ‘wast’) and (‘wast’, ‘money’) convey regret, which can indicate negative sentiment, while 2-grams such as (‘work’, ‘great’), (‘great’, ‘phone’), and (‘highli’, ‘recommend’) convey a feeling of satisfaction, which can indicate positive sentiment.

0.4 Problem 3: Written Exercises

1. Ridge Regression. Recall that ridge regression is the solution to:

$$\min_{\beta \in \mathbb{R}^{1+p}} \sum_{i=1}^{n} (Y_i - \beta^T X_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where β_0 is the intercept and is excluded from the penalty term.

In this excercise we will show that the ridge regression solution can be obtained by ordinary least squares on an augmented data set. Let us augment the X matrix and Y vector with p additional rows, so that:

$$\mathbf{X}_{\text{aug}} = \begin{bmatrix} 1 & X_{11} & X_{12} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & \dots & X_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & X_{np} \\ 0 & \sqrt{\lambda} & 0 & \dots & 0 \\ 0 & 0 & \sqrt{\lambda} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sqrt{\lambda} \end{bmatrix} \quad \mathbf{Y}_{\text{aug}} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1}$$

Show that OLS on X_{aug}, Y_{aug} is the same as ridge on X, Y with penalty parameter λ. Comment briefly on how we can think of ridge regression as including more prior information about the value of beta as reflected by adding new data points not present in the observed dataset.

Start by minimizing OLS on $Y_{Aug,i}$ and $X_{Aug,i}$:

$$\sum_{i=1}^{n+p} (Y_{Aug,i} - \beta^T X_{Aug,i})^2$$

Split the summations to 1 to n, and n to n + p:

$$= \sum_{i=1}^{n} (y_{Aug,i} - \beta^T X_{Aug,i})^2 + \sum_{i=n}^{n+p} (y_{Aug,i} - \beta^T X_{Aug,i})^2$$

Note that from 1 to n, $Y_{Aug,i}$ is Y_i and $X_{Aug,i}$ is X_i. Furthermore, from i =n to n + p, $Y_{Aug,i}$ is 0 and $X_{Aug,i}$ is $\sqrt{\lambda}\beta_{n-i+1}$:

$$= \sum_{i=1}^{n} (y_i - \beta^T X_i)^2 + \sum_{i=n}^{n+p} (\sqrt{\lambda}\beta_{n-i+1})^2$$

Note that if we square $\sqrt{\lambda}$ we will get λ:

$$= \sum_{i=1}^{n} (y_i - \beta^T X_i)^2 + \sum_{i=n}^{n+p} \lambda(\beta_{n-i+1})^2$$

Lastly, observe that the indices for $\sum_{i=n}^{n+p} \lambda(\beta_{n-i+1})^2$ can be reset, as below:

$$= \sum_{i=1}^{n} (y_i - \beta^T X_i)^2 + \sum_{i=1}^{p} \lambda(\beta_i)^2$$

2. Naive Bayes classifiers. In a medical study, 100 patients all fell into one of three classes: Pneumonia, Flu, or Healthy. The following database indicates how many patients in each class had fever and headache.

Pneumonia

Fever	Headache	Count
T	T	5
T	F	0
F	T	4
F	F	1
Total:		10

Flu

Fever	Headache	Count
T	T	9
T	F	6
F	T	3
F	F	2
Total:		20

Healthy

Fever	Headache	Count
T	T	2
T	F	3
F	T	7
F	F	58
Total:		70

Consider a patient with fever and no headache.

(a) Assuming that the above counts represent the whole population, what probability would a Bayes optimal classifier assign to each of the three propositions that the patient has Pneumonia, Flu, or neither? Show your work. (For this question, the three values should sum to 1.)

As we know, the Bayes Formula can be represented in the following way:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

For simplicity, I will use the following abbreviations: Pneuom (Pneumonia), Fev (Fever), Health (Healthy) and No Head (No Headache). In general, we can therefore represent our three probability values as:

$$\text{P(Pneum | Fev, No Head)} = \frac{\text{P(Fev, No Head | Pneum) P(Pneum)}}{\text{P(Fev, No Head)}}$$

$$\text{P(Health | Fev, No Head)} = \frac{\text{P(Fev, No Head | Health) P(Health)}}{\text{P(Fev, No Head)}}$$

$$\text{P(Flu | Fev, No Head)} = \frac{\text{P(Fev, No Head | Flu) P(Flu)}}{\text{P(Fev, No Head)}}$$

For optimal Bayes, we can simply use the equations exactly as they currently are (since we do not assume independence between the two

features, we can simply calculate P(A| B, C) directly through the data we have rather than treating features independently and breaking them up into components). We can treat it this way (as optimal) because we are told that our data represents the WHOLE population.

Using our data, we can derive the following values:

$$\text{P(Pneum)} = \frac{\text{\# Pneum Patients}}{\text{\# Total Population}} = \frac{10}{100} = 0.1$$

$$\text{P(Flu)} = \frac{\text{\# Flu Patients}}{\text{\# Total Population}} = \frac{20}{100} = 0.2$$

$$\text{P(Health)} = \frac{\text{\# Health Patients}}{\text{\# Total Population}} = \frac{70}{100} = 0.7$$

$$\text{P(Fev, No Head)} = \frac{\sum\text{(Pneum, Flu, \& Health with Fev = T, Head = F)}}{\text{\# Total Patients}} = \frac{0+6+3}{100} = 0.09$$

$$\text{P(Fev, No Head | Pneum)} = \frac{\text{\# Pneum Patients with Fev = T \& Head = F}}{\text{\# Total Pneumonia Population}} = \frac{0}{10} = 0$$

$$\text{P(Fev, No Head | Flu)} = \frac{\text{\# Flu Patients with Fev = T \& Head = F}}{\text{\# Total Flu Patients}} = \frac{6}{20} = 0.3$$

$$\text{P(Fev, No Head | Health)} = \frac{\text{\# Health Patients with Fev = T \& Head = F}}{\text{\# Total Health Patients}} = \frac{3}{70}$$

Finally, we can plug in our calculations for the probability values above and calculate the probability for optimal Bayes of having pneumonia, having the flu, or being healthy given having a fever and no headache:

$$\text{P(Pneum | Fev, No Head)} = \frac{0 * 0.1}{0.09} = \mathbf{0}$$

$$\text{P(Flu | Fev, No Head)} = \frac{0.3 * 0.2}{0.09} = \mathbf{\frac{2}{3}}$$

$$\text{P(Health | Fev, No Head)} = \frac{3 \ / \ 70 * 0.7}{0.09} = \mathbf{\frac{1}{3}}$$

(b) What probability would a naïve Bayes classifier assign to each of the three propositions that the patient has Pneumonia, Flu, or neither? Show your work. (For this question, the three values should sum to 1.)

This time, we are going to be using a naive Bayes classifier instead, meaning that we now assume the features are independent. This alters our formulas because terms such as P(Fev, No Head | Pneum), P(Fev, No Head | Flu), and P(Fev, No Head | Health) now need to be further broken up into individual components. Our equations therefore become:

$$\text{P(Pneum | Fev, No Head)} = \frac{\text{P(Fev | Pneum) P(No Head | Pneum) P(Pneum)}}{\text{D}}$$

$$\text{P(Flu | Fev, No Head)} = \frac{\text{P(Fev | Flu) P(No Head | Flu) P(Flu)}}{\text{D}}$$

$$\text{P(Health | Fev, No Head)} = \frac{\text{P(Fev | Health) P(No Head | Health) P(Health)}}{\text{D}}$$

In these expressions, we can represent D as:

$$\text{D} = \text{P(Fev | Pneum)} * \text{P(No Head | Pneum)} * \text{P(Pneum)} + \text{P(Fev | Flu)} * \text{P(No Head | Flu)} * \text{P(Flu)} + \text{P(Fev | Health)} * \text{P(No Head | Health)} * \text{P(Health)}$$

Using our data, we can derive the following additional values:

$$\text{P(Fev} \mid \text{Pneum)} = \frac{\text{\# Pneum Patients with Fev = T}}{\text{\# Total Pneumonia Population}} = \frac{5}{10} = 0.5$$

$$\text{P(No Head} \mid \text{Pneum)} = \frac{\text{\# Pneum Patients with Head = F}}{\text{\# Total Pneumonia Population}} = \frac{1}{10} = 0.1$$

$$\text{P(Fev} \mid \text{Flu)} = \frac{\text{\# Flu Patients with Fev = T}}{\text{\# Total Flu Population}} = \frac{15}{20} = 0.75$$

$$\text{P(No Head} \mid \text{Flu)} = \frac{\text{\# Flu Patients with Head = F}}{\text{\# Total Flu Population}} = \frac{8}{20} = 0.4$$

$$\text{P(Fev} \mid \text{Health)} = \frac{\text{\# Health Patients with Fev = T}}{\text{\# Total Health Population}} = \frac{5}{70} = \frac{1}{35}$$

$$\text{P(No Head} \mid \text{Health)} = \frac{\text{\# Health Patients with Head = F}}{\text{\# Total Health Population}} = \frac{61}{70}$$

Using the probability values that we just calculated and some of the values we calculated in part a, we can then calculate our 3 final probability values (using the Naive Bayes assumption):

$$\text{P(Pneum} \mid \text{Fev, No Head)} = \frac{(0.5)(0.1)(.1)}{(0.5)(0.1)(.1) + (0.75)(0.4)(.2) + (\frac{1}{35})(\frac{61}{70})(.7)} = \mathbf{0.0461}$$

$$\text{P(Flu} \mid \text{Fev, No Head)} = \frac{(0.75)(0.4)(.2)}{(0.5)(0.1)(.1) + (0.75)(0.4)(.2) + (\frac{1}{35})(\frac{61}{70})(.7)} = \mathbf{0.5526}$$

$$\text{P(Health} \mid \text{Fev, No Head)} = \frac{(\frac{1}{35})(\frac{61}{70})(.7)}{(0.5)(0.1)(.1) + (0.75)(0.4)(.2) + (\frac{1}{35})(\frac{61}{70})(.7)} = \mathbf{0.4013}$$

3. Naive Bayes for data with nominal attributes. Given the training data in the table below (Buy Computer data), predict the class of the following new example using Naive Bayes classification: age≤30, income=medium, student=yes, credit-rating=fair. Please show your work.

ID	age	income	student	credit-rating	Class: buys-computer
1	≤30	high	no	fair	no
2	≤30	high	no	excellent	no
3	31…40	high	no	fair	yes
4	>40	medium	no	fair	yes
5	>40	low	yes	fair	yes
6	>40	low	yes	excellent	no
7	31…40	low	yes	excellent	yes
8	≤30	medium	no	fair	no
9	≤30	low	yes	fair	yes
10	>40	medium	yes	fair	yes
11	≤30	medium	yes	excellent	yes
12	31…40	medium	no	excellent	yes
13	31…40	high	yes	fair	yes
14	>40	medium	no	excellent	no

For this problem, we will use Naive Bayes to calculate the following two probability values:

P(Buys Computer = Yes | Age≤30, Income=Medium, Student=Yes, Credit-Rating=Fair)

P(Buys Computer = No | Age≤30, Income=Medium, Student=Yes, Credit-Rating=Fair)

We will then compare each of these probabilities – if the probability for Buys Computer = Yes is greater than Buys Computer = No, we will classify this test sample as Buys Computer = Yes (and vice versa). Since we are just comparing the two values, we can drop the denominator in the Naive Bayes formula, since both denominators are equal. After splitting the features into independent probabilites, as is done with Naive Bayes, our two expressions above can be written as:

P(Buy Computer=Yes | Age≤30, Income=Medium, Student=Yes, Credit-Rating=Fair) = P(Age≤30 | Buy Computer=Yes)*P(Income=Medium | Buy Computer=Yes) *P(Student=Yes | Buy Computer=Yes)*P(Credit-Rating=Fair | Buy Computer=Yes) *P(Buy Computer=Yes)

P(Buy Computer=No | Age≤30, Income=Medium, Student=Yes, Credit-Rating=Fair) = P(Age≤30 | Buy Computer=No)*P(Income=Medium | Buy Computer=No) *P(Student=Yes | Buy Computer=No)*P(Credit-Rating=Fair | Buy Computer=No) *P(Buy Computer=No)

We can set up the following calculations for the several independent conditional probabilities we must calculate:

$$\text{P(Age≤30 | Buy Computer=Y)} = \frac{\text{\# (Age≤30 ∩ Buy Computer=Y)}}{\text{\# Buy Computer=Y}} = \frac{2}{9}$$

$$\text{P(Income=Med. | Buy Computer=Y)} = \frac{\text{\# (Income=Med. ∩ Buy Computer=Y)}}{\text{\# Buy Computer=Y}} = \frac{4}{9}$$

$$\text{P(Student=Y | Buy Computer=Y)} = \frac{\text{\# (Student=Y ∩ Buy Computer=Y)}}{\text{\# Buy Computer=Y}} = \frac{6}{9}$$

$$\text{P(Credit=Fair | Buy Computer=Y)} = \frac{\text{\# (Credit=Fair ∩ Buy Computer=Y)}}{\text{\# Buy Computer=Y}} = \frac{6}{9}$$

$$\text{P(Buy Computer=Y)} = \frac{\text{\# Buy Computer=Y}}{\text{\# Buy Computer=Y + \# Buy Computer=N}} = \frac{9}{14}$$

$$\text{P(Age≤30 | Buy Computer=N)} = \frac{\text{\# (Age≤30 ∩ Buy Computer=N)}}{\text{\# Buy Computer=N}} = \frac{3}{5}$$

$$\text{P(Income=Med. | Buy Computer=N)} = \frac{\text{\# (Income=Med. ∩ Buy Computer=N)}}{\text{\# Buy Computer=N}} = \frac{2}{5}$$

$$\text{P(Student=Y | Buy Computer=N)} = \frac{\text{\# (Student=Y ∩ Buy Computer=N)}}{\text{\# Buy Computer=N}} = \frac{1}{5}$$

$$\text{P(Credit=Fair | Buy Computer=N)} = \frac{\text{\# (Credit=Fair} \cap \text{ Buy Computer=N)}}{\text{\# Buy Computer=N}} = \frac{2}{5}$$

$$\text{P(Buy Computer=N)} = \frac{\text{\# Buy Computer=N}}{\text{\# Buy Computer=Y + \# Buy Computer=N}} = \frac{5}{14}$$

With these values, we can calculate the following two probabilities:

P(Buy Computer=Yes | Age≤30, Income=Medium, Student=Yes, Credit-Rating=Fair) = $(\frac{2}{9})\times(\frac{4}{9})\times(\frac{6}{9})\times(\frac{6}{9})\times(\frac{9}{14})$ = **0.028219**

P(Buy Computer=No | Age≤30, Income=Medium, Student=Yes, Credit-Rating=Fair) = $(\frac{3}{5})\times(\frac{2}{5})\times(\frac{1}{5})\times(\frac{2}{5})\times(\frac{5}{14})$ = **0.006857**

Since the first probability is clearly larger than the second, we will classify the sample as **'Buy Computer = Yes'** using the Naive Bayes assumption/method.