Phan Anh Nguyen
Comp135

Project 2 Report

Problem 1a – BOW Pipeline:

The provided CSV files were first read in using the *read_csv()* function from the **Pandas** library. Individual sentences were then stored into a text list using the *values.tolist()* function on the '*text*' column of the csv file. Each sentence was then fed into a tokenizer function (seen below) to remove special symbols and numbers, converted into lower case and added to a processed list.

```
def tokenizer(str_input):
    words = re.sub(r"[^A-Za-z0-9\-]", " ", str_input).lower()
    return words
```

This processed list had to then be vectorized. For this problem, I chose to use the *TfidfVectorizer* instead of the *CountVectorizer*. The *TfidfVectorizer* reweighs the values associated with the counts for each word. This allows less weight to be assigned to common or stop words typically used such as "not", "a", etc. These words do not contribute to the sentence meaning and if used with a *CountVectorizer*, may overshadow words that have significant meaning to the overall sentiment.

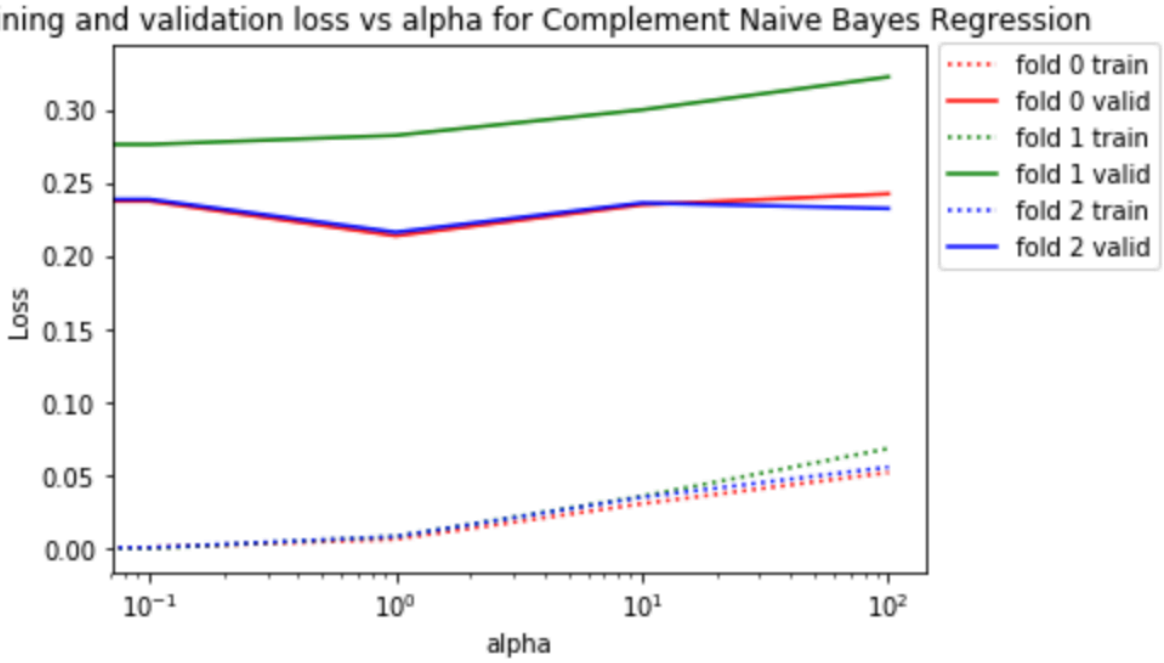
For this problem, I chose to include all words as the *tfidfVectorizer* would most likely handle sparse words (occurs less than 10 times) by reweighing accordingly. I also only used monograms as the size of the vocabulary (21239 features) is already large and increasing the number of features would cause more performance time constraints.

Next, I used the *GridSearchCV* pipeline from sklearn to perform classification as well as cross validation. This pipeline allows me to feed in a dictionary containing the different hyperparameters to be tested as well as the number of folds for cross validation can be specified at the start which makes testing convenient. Based on eyeballing the accuracy of different number of folds, I determined that cross validation using 3 folds gave the highest accuracy. As there is only 2400 data, anything more than 3 folds will result in too little data being used for training each fold which decreases performance due to overfitting.

Error rates for training and validation across different folds could be accessed via the *cv_results_* attribute from *GridSearchCV.* These were then stored into a csv file for plotting performance. A list of False Positives and False Negatives was also identified for the best performing classifier.

Phan Anh Nguyen
Comp135

Problem 1b – Naïve Bayes Classifier:



The first classifier used was the Naïve Bayes Classifier. The advantage of this classifier is its simplicity and speed of training as it uses properties of Bayes Theorem. In the BOW representation, there is rarely a clear cut relationship between features, so Naïve Bayes may perform well as the conditional independence assumption holds.

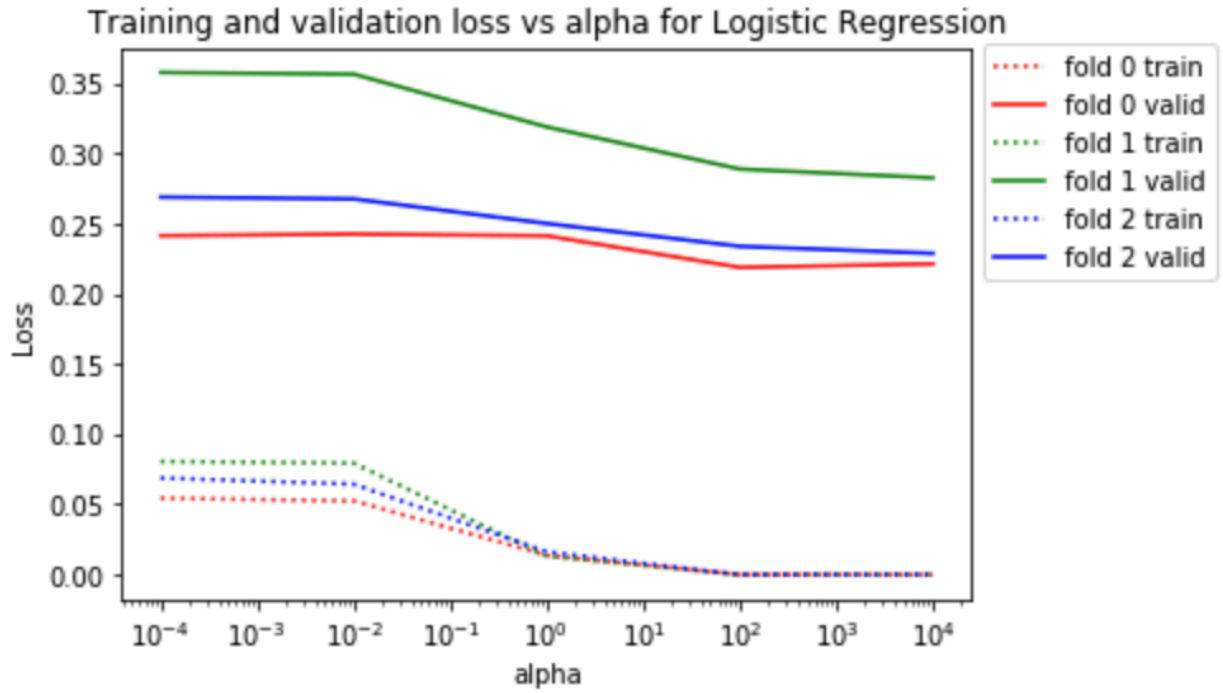
I used the Complement Naïve Bayes version that is built into sklearn as it provides better performance than the Gaussian Naïve Bayes classifier. Complement Naïve Bayes is a classifier that is designed to work well with discrete features with integer data which is the case for sentiment analysis.

I explored a range of smoothing values for this classifier. The tuning consisted of smoothing variables from 0 to 100. As the smoothing variable effectively changes the weight of features, giving more weight to less prominent features, it is possible to overfit or underfit. Testing showed that a **smoothing parameter of 1** resulted in the lowest error on validation set errors across folds as seen in the graph.

Cross validation was handled by the default *GridSearchCV* method which is StratisfiedKfold. The obtained results were decisive.

Phan Anh Nguyen
Comp135

Problem 1c – Logistic Regression Classifier:



The second classifier used was the Logistic Regression Classifier. The advantage of this classifier it is more versatile and can be used with many different types of data. The model can also be easily be updated should more data be available.

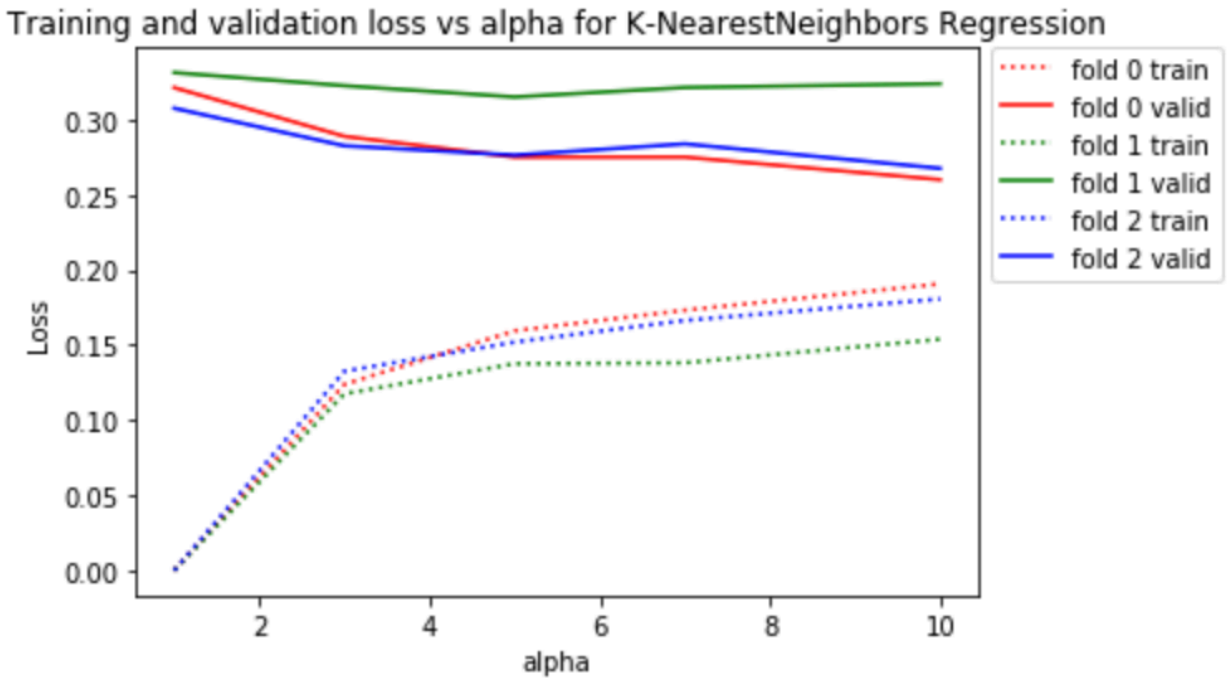
For this implementation, I specified max number of iterations at 10000 and used 'newton-cg' as the solver. This combination of iterations was used as it allowed me to prevent from running into any convergence issues during training.

I explored a range of alpha values for this classifier. The tuning consisted of smoothing variables from 0.0001 to 10000. As the alpha value is a penalty on the weights, a value that is too large will cause underfitting and too small a penalty will result in overfitting. Testing showed that **alpha value of 100** resulted in the lowest error on validation set errors across folds as seen in the graph. Even though the loss at alpha = 10000 is only slightly higher, this signals that we are starting to underfit the data. At alpha values lower than 100, there is overfitting which results in higher loss as seen in the graph.

Cross validation was handled by the default *GridSearchCV* method which is StratisfiedKfold. The obtained results were decisive.

Phan Anh Nguyen
Comp135

Problem 1d – k-NearestNeighbors Classifier:



The third classifier used was the K-NearestNeighbor Classifier. The advantage of this classifier it is quick. There is little or no training as the values passed into fit are used as a 'lookup' table for predicting new data. I did not run into any convergence issues during training.

I explored a range of neighbor to be used for classification. The tuning consisted of varying the number of neighbors from 1 to 10. With K-NN, having 2 or more neighbors provides more predictive power as averaging the feature value based on neigbors might yield a more accurate representation as a simple 1-neighbor lookup might cause underfitting. Too many neighbors, on the other hand, might cause overfitting as K-NN tends towards median guessing as the number of neighbors increases to the number of samples. Testing showed that both **5 neighbors** and **10 neighbors** yielded equally good results, although validation error increased for 7 neighbors.

Given more time, I would have liked to tune for up to 15 neighbors to confirm that overfitting is taking place for neighbors > 5 .However, as K-NN is already performing worse compared to the other methods, I stopped tuning early.

Cross validation was handled by the default *GridSearchCV* method which is StratisfiedKfold. The obtained results were indecisive.

Phan Anh Nguyen
Comp135

Problem 1e – Analysis of Best Classifier:

The table below shows the performance of the three classifiers based on best hyperparameter chosen:

Method	Error on validation
Complement Naïve Bayes	0.21979167
Logistic Regression	0.24416667
K-NearestNeighbors	0.28876667

Based on the performances above, the best performing classifier is the Complement Naïve Bayes. This is perhaps due to the Complement/Multinomial variant of Naïve Bayes being tuned especially for text analysis, whereas for the logistic regression model, only basic hyperparameters such as alpha and solver type were tuned. Another factor that could contribute to the performance of CNB is the size of the feature vector. Due to the sparseness of the feature vector and the large size of the vocabulary, it is unlikely that there would be interactions between classes which makes the fundamental assumptions used by Naïve Bayes analyses true to a large extent.

Below are some False Positive sentiments by the Complement Naïve Bayes:

```
The loudspeaker option is great, the bumpers with the lights is very ... appealing.
Graphics is far from the best part of the game.
Not easy to watch.
The tables outside are also dirty a lot of the time and the workers are not always
friendly and helpful with the menu.
```

In the first sentence, we see that the Classifier fails to identify more subtle, higher order properties of languages such as sarcasm. Based on the words alone, that sentence seems positive. However, based on the structure and tone, we know that it is the opposite. The fourth sentence also contains a low of positive words. We only know that the overall sentiment is negative as there is a ‘not’ before all the adjectives.

Below are some False Negative sentiments by the Complement Naïve Bayes:

```
I did not have any problem with this item and would order it again if needed.
This is the kind of money that is wasted properly.
Predictable, but not a bad watch.
```

Similar to the False Positive examples, the first sentence is a an example where a lot of negative words were used but was preceded by a ‘not’. Sentence 2 is also an example of sarcasm that was mis-identified.

Phan Anh Nguyen
Comp135

Problem 1f – Leaderboard Performance:

Error rate: 0.165

AUROC: 0.9242

Performance on heldout data was worse than performance on validation data (error rate = 0.165 vs 0.2375). This goes against expectations that heldout error is higher. Perhaps this might be due random variations in the test data as well as the best Cross-Validation fold. The weights for the chosen fold might have performed better for a particular type of sentence (i.e: does better on Amazon ones than Yelp) and it is possible that the test data contains more of the type of sentence that the fold performs well on.

Phan Anh Nguyen
Comp135

Problem 2a – Word Embedding Pipeline:

Similar to problem 1a, the provided CSV files were first read in using the *read_csv()* function from the **Pandas** library. Individual sentences were then stored into a text list using the *values.tolist()* function on the '*text'* column of the csv file. Each sentence was then fed into a tokenizer function to remove special symbols and numbers, converted into lower case and added to a processed list. The *split()* function is used to divide the sentence into individual words.

GLOVE embedding was then used to assign values to words in each sentence. For this implementation, the average value across the sentence was calculated by summing up the found values for each word then dividing by the number of words in the sentences. Although this removes the possible distinction between short and long sentences, I used it as it reduces the need for scaling each feature value later on during classification and is easier to preprocess. The final feature vector for each sentence was then added to a processed list.

For this problem, I conducted several additional preprocessing procedures. The first is addition of **min, max,** and **standard deviation** for features in the word embedding. I used the numpy functions *np.amin()* and *np.amax()* to retrieve the word with the minimum value across features and maximum value across features for each sentence. My hypothesis behind this was that words with extreme feature values as well as the spread of feature values are often meaningful as they can help contribute to the overall sentiment and should be emphasized.

After this step, the total number of features in my training data was around 200. To increase dimensionality, I used the built in **polynomial feature transform** to increase the number of features to around 20,000 to help increase performance.

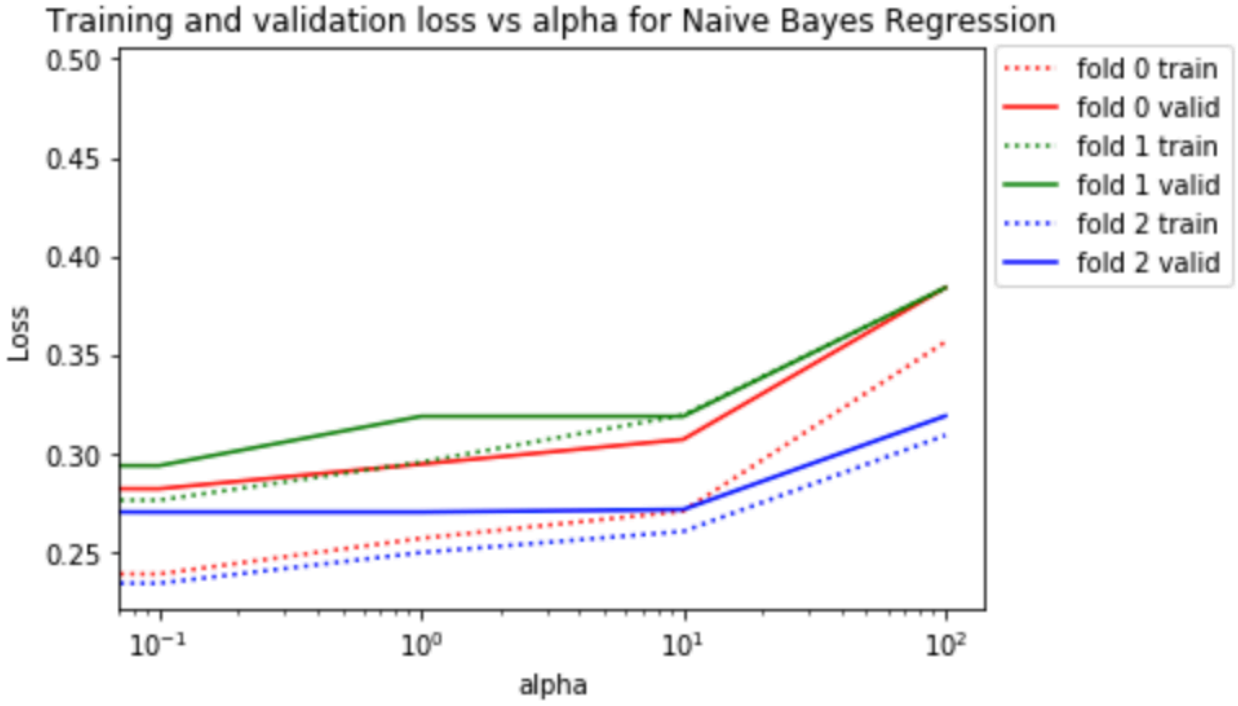
I also implemented **Bootstrap Aggregation**. I implemented the bootstrap sampling function shown in class and ran the chosen classifier for each sample for a total of 20 samples which were averaged at the end to produce the final predicted list.

Next, I used the *GridSearchCV* pipeline from sklearn to perform classification as well as cross validation. This pipeline allows me to feed in a dictionary containing the different hyperparameters to be tested as well as the number of folds for cross validation can be specified at the start which makes testing convenient. Similar to Problem 1, 3 folds yielded the best results.

Error rates for training and validation across different folds could be accessed via the *cv_results_* attribute from *GridSearchCV.* These were then stored into a csv file for plotting performance. A list of False Positives and False Negatives was also identified for the best performing classifier.

Phan Anh Nguyen
Comp135

Problem 2b – Naïve Bayes Classifier:



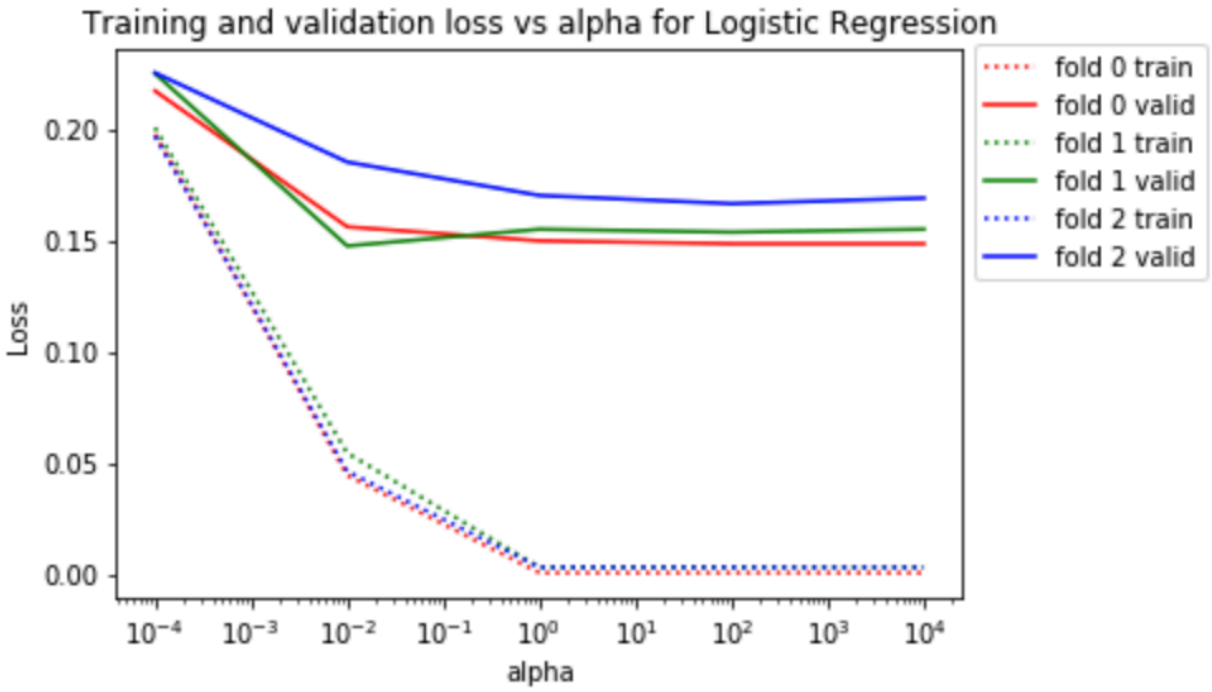
The first classifier used was the Naïve Bayes Classifier, similar to Problem 1. The advantage of this classifier is its simplicity and speed of training as it uses properties of Bayes Theorem. As I could not use Complement Naïve Bayes or Multinomial Naïve Bayes due to the feature value range, I used the standard Gaussian Naïve Bayes Classifier.

I explored a range of smoothing values for this classifier. The tuning consisted of smoothing variables from 0 to 100. As the smoothing variable effectively changes the weight of features, giving more weight to less prominent features, it is possible to overfit or underfit. Testing showed that a **smoothing parameter of 0.1** resulted in the lowest error on validation set errors across folds as seen in the graph. A possible improvement is to use more data to obtain more accurate representations across folds.

Cross validation was handled by the default *GridSearchCV* method which is StratisfiedKfold. The obtained results were decisive.

Phan Anh Nguyen
Comp135

Problem 2c – Logistic Regression Classifier:



The second classifier used was the Logistic Regression Classifier. The advantage of this classifier it is more versatile and can be used with many different types of data. The model can also be easily be updated should more data be available.

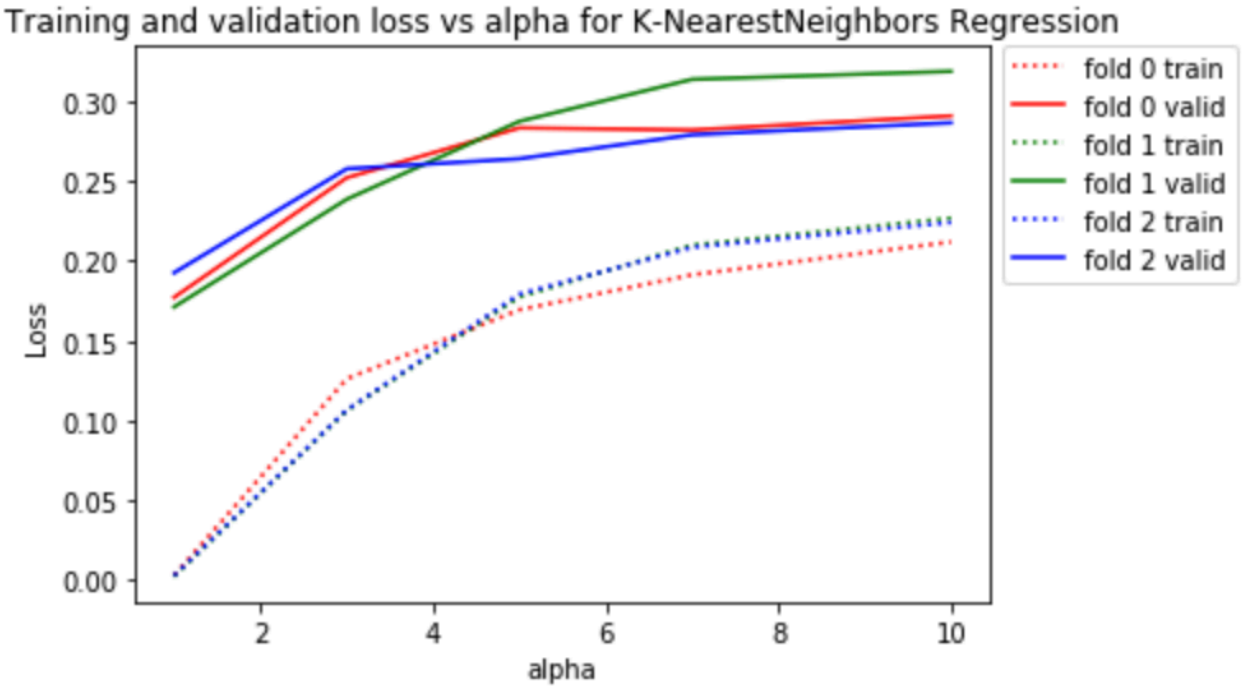
For this implementation, I specified max number of iterations at 10000 and used 'newton-cg' as the solver. This combination of iterations was used as it allowed me to prevent from running into any convergence issues during training.

I explored a range of alpha values for this classifier. The tuning consisted of smoothing variables from 0.0001 to 10000. As the alpha value is a penalty on the weights, a value that is too large will cause underfitting and too small a penalty will result in overfitting. Testing showed that **alpha value of 100** resulted in the lowest error on validation set errors across folds as seen in the graph. As alpha value of 10000, there was underfitting. At alpha values lower than 100, there is overfitting which results in higher loss as seen in the graph. Fold 2 best represents the relationship across alpha values. However, more data would allow any underfit/overfit trend to be better represented across folds.

Cross validation was handled by the default *GridSearchCV* method which is StratisfiedKfold. The obtained results were decisive.

Phan Anh Nguyen
Comp135

Problem 2d – k-NearestNeighbors Classifier:



The third classifier used was the K-NearestNeighbor Classifier. The advantage of this classifier it is quick. There is little or no training as the values passed into fit are used as a 'lookup' table for predicting new data. I did not run into any convergence issues during training.

I explored a range of neighbor to be used for classification. The tuning consisted of varying the number of neighbors from 1 to 10. With K-NN, having 2 or more neighbors provides more predictive power as averaging the feature value based on neigbors might yield a more accurate representation as a simple 1-neighbor lookup might cause underfitting. Too many neighbors, on the other hand, might cause overfitting as K-NN tends towards median guessing as the number of neighbors increases to the number of samples.

Testing showed that using **1 neighbor** yielded the best solution. This could potentially be due to the mechanics of GLOVE embedding. In the dictionary, up to 400000 words are represented in a narrow range of 50 features. This means that the boundary between words for each feature is small. As a result, averaging over neighbors might change the word meaning completely and cause higher loss overall.

Cross validation was handled by the default *GridSearchCV* method which is StratisfiedKfold. The obtained results were decisive.

Phan Anh Nguyen
Comp135

Problem 2e – Analysis of Best Classifier:

The table below shows the performance of the three classifiers based on best hyperparameter chosen:

Method	Error on validation
Gaussian Naïve Bayes	0.28206667
Logistic Regression	0.15623333
K-NearestNeighbors	0.18043333

Based on the performances above, the best performing classifier is Logistic Regression. This is perhaps due to the versatility of Logistic Regression in response to sample preprocessing which allowed it to obtain better performance improvements compared to the other two methods. Many additional features were added (min, max, sd), as well as bootstrap sample also influenced the spread of data for each run. While this increased the accuracy for Logistic Regression, GNB did not seem to gain much improvement. This is most likely because the assumptions underlying Naïve Bayes are no longer true unlike problem 1.

Below are some False Positive sentiments by the Logistic Regression:

```
Very wind-resistant.
Will be back again!
GO AND SEE IT!
Would recommend this item.
```

In general, it seems that the classifier struggles with short sentences. Perhaps averaging the feature vector over 3 or 4 words is not enough to accurately portray the sentence sentiment. One possible suggestion would be to sum the values instead.

Below are some False Negative sentiments by the Logistic Regression:

```
I would not recommend this item to anyone.
It is not good.
Not good enough for the price.
Would not recommend to others.
```

Once again, it seems like after removal of stop-words, these sentences all had few words left. This may have prevented the classifier from accurately determining the overall sentiment. It seems to struggle with the words “recommend” and “good”.

Phan Anh Nguyen
Comp135

Problem 2f – Leaderboard Performance:

Error rate: 0.209999

AUROC: 0.84739

Performance on heldout data was worse than performance on validation data (error rate = 0.210 vs 0.156). This is to be expected as new test data could contain some information that has not been accounted for in the weights during training.

Phan Anh Nguyen
Comp135

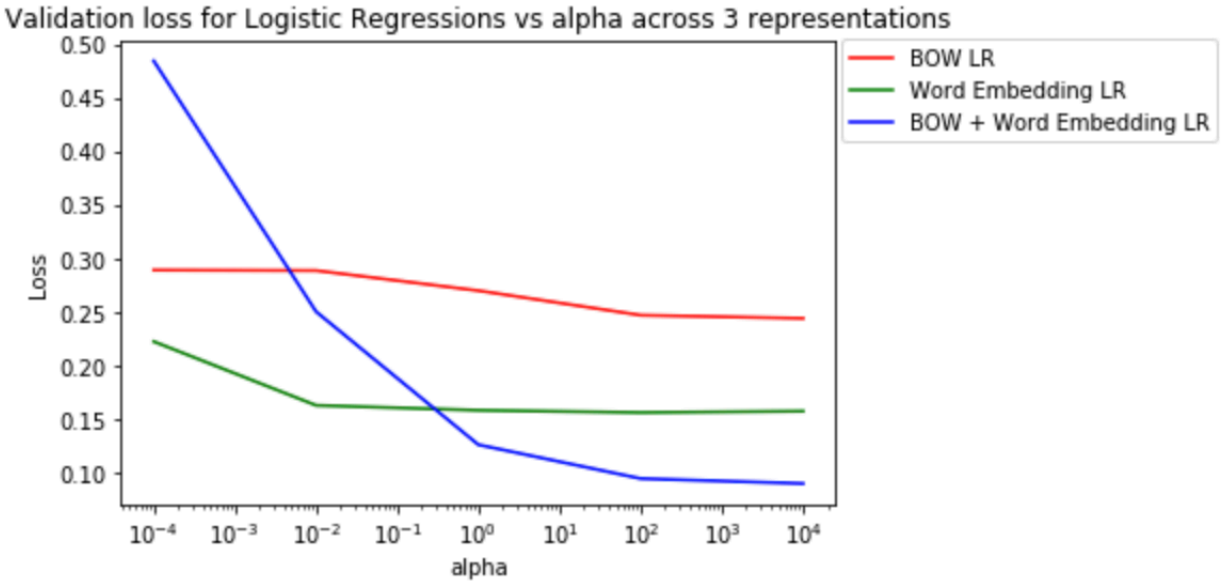
Problem 3 - Custom

For Problem 3, I have decided to combine both representations from Problem 1 and 2 such that the feature vector contains both BOW representations as well as Word Embeddings.

When performing the preprocessing, I first noticed that the number of features for BOW (>20000) representations severely outweighs those for Word Embeddings (50). If I simply added them, then the BOW representation would far outweigh Word Embeddings. Therefore, I had to increase the number of features for Word Embeddings.

I first added the min, max and standard deviation features for each sentence to the Word Embedding representation. This increased the total number of features to 200 (50 for each addition). Following that, I used polynomial feature transform of degree 2 without interaction to increase the total number of features to >21000. This is roughly equal to that of the BOW representation and would ensure that both representations have roughly equal weightage when it comes to fitting.

I chose Logistic Regression as the classifier as it performed close to CNB in problem 1 and was the best classifier for problem 2. Below is a figure showing validation loss across alpha values averaged over 3 folds:



As seen in the figure, from alpha = 1 onwards, BOW + Word Embedding significantly outperforms both BOW and Word Embedding representations. The observed error rate on average is **0.09 for the best alpha** and can even go down to 0.06 for some folds. In contract, Word Embedding LR loss never goes below 0.15. Thus, we see that this implementation has been successful.