Project A Report

Matthew Carey and Alex Lemieux

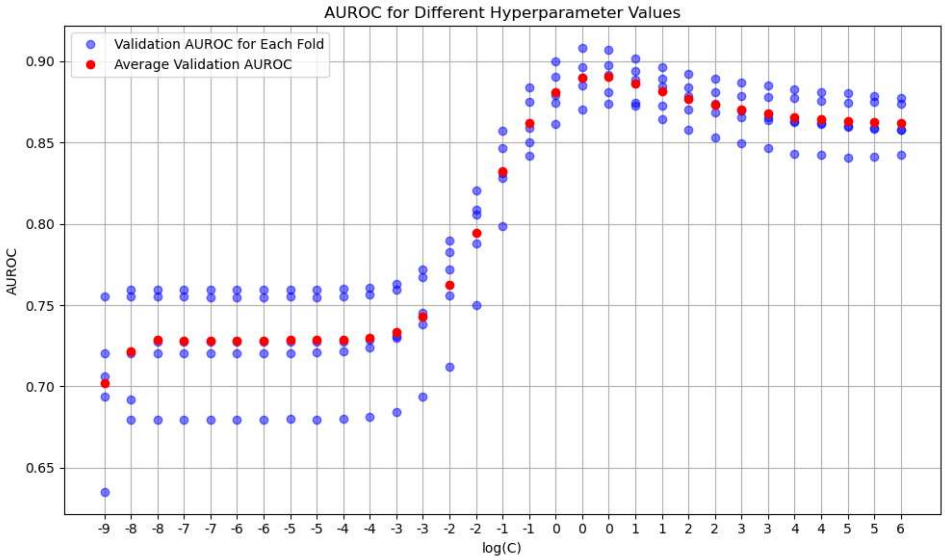
Problem 1:

a) For our first classifier, we used a **count vectorizer** object from Sklearn to transform our data of sentences into a bag of words (BoW) representation. We cleaned our data using built in parameters of the countVectorizer object to turn all **upper-case letters to lower case**, as well as to **filter out stop words** chosen by a built-in list of stop words. Additionally, we filtered out any **words that are found in more than 80% of the documents**, to remove any words that may be irrelevant in the context of reviews but are not general enough to have been included in the built-in stop words list. The final vocabulary set that our model uses to build BoW representations is solely based on the training data that we are presented, and **new values that it sees in unseen data will be ignored**. This is because the function that is being called to extract BoW features from test data directly references the vocabulary of the vectorizer used on the training set, rather than creating its own vocabulary. This vocabulary used by our model was roughly **4,000 elements**. The vector for each word was built using **counts of words**, as opposed to binary values.

b) To perform cross validation (CV), we used two off-the-shelf objects from the sklearn model_selection package: **Kfold** and **GridSearchCV**. We used these objects to execute CV by creating a Kfold object with **5 even split folds** that **shuffled** data entries, as well as creating a grid of hyperparameters and ranges to choose from. We then created a GridSearchCV object that took our logistic regression object, our parameter grid, our Kfold object, and a scoring metric to use, which we selected to be **AUROC**, based on the fact our model was being evaluated on AUROC. With all this information, we were then able to use the GridSearchCV object's fit function to fit it to our training data, and in this fit it used the K-folds validation we specified to search each configuration of hyperparameters for the model that scored the best AUROC score. This best model was then able to be **extracted from the GridSearchCV object**.

c) The following graph shows our search for optimal hyperparameters using Cross validation. On the x-axis is $\log_{10}$ of the **value for C**, the penalty term applied to penalize overfitting. We searched for this hyperparameter over a scale from. $\mathbf{10^{-9}}$ **to** $\mathbf{10^{6}}$. This wide range of C values to search over gives us the ability to avoid both under and over fitting to the data. We can see from the graph that there is a sharp increase in the performance in the range from -0.01 to 10, then

gradually decreases. We can see that our best model came from a **C value of roughly 3.1623**



d) Below are some telling examples of our model's false positives and false negatives:

False Positives:

['amazon','I really wanted the Plantronics 510 to be the right one, but it has too many issues for me.The nice']

['amazon', 'Excellent starter wireless headset.']

['amazon', 'Not nice enough for the price.']

['imdb', 'Graphics is far from the best part of the game. ']

['imdb', 'The only place nice for this film is in the garbage. ']

False Negatives:

['imdb', 'Predictable, but not a badly watch. ']

['imdb', 'The last 15 minutes of movie are also not badly as well. ']

['imdb', 'Waste your money on this game. ']

By looking at these examples, false positives are susceptible to **litotes**, where the unigrams may seem positive, but looking at larger contexts of words, a human reader can see they are negative. Additionally, it is susceptible to **typos** that have nothing to do with the review. From the false negatives, we see our model seems to be susceptible to **ironic or contradictory statements**, like in the last example. Additionally, it seems specifically susceptible to the word 'badly'. Using these to improve performance, in part 2 we can allow our model to look at bigrams or more to try and catch litotes. We can also remove words that show up in equal numbers of positive and negative reviews, so that words like badly that may be assigned to be negative do not mess up the many positive reviews they are in.

e) Upon submission to the autograder, our model received an AUROC score of **0.88559** on the test data. This score is very **consistent with our predictions** of AUROC from validation data, which

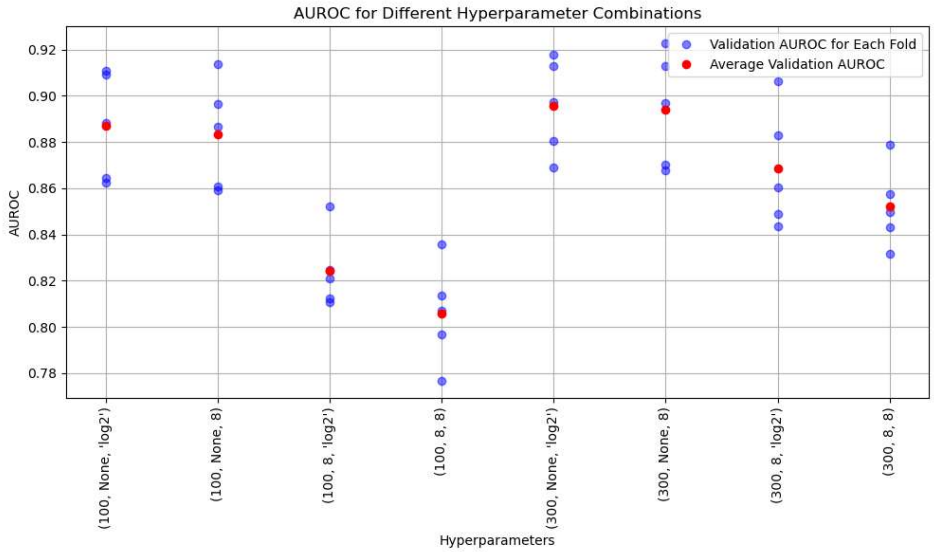
predicted the highest score of 0.8903. The close alignment between our estimated performance from cross-validation and the actual test performance indicates that our model generalizes well to unseen data. Additionally, this performance reaffirms the robustness of our model's hyperparameter selection, as the chosen parameters yielded competitive results across different datasets. Moving forward, we can further explore strategies to enhance the model's performance, such as feature engineering or ensemble methods, to potentially achieve even higher AUROC scores.

Problem 2:

a) For Problem 2, we elected to use SKLearn's **TFIDF Vectorizer** as opposed to a Count Vectorizer. Like in part 1, we cleaned the data by converting all text to **lowercase** and filtering out words occurring with a **frequency of greater than 0.8** (appearing in 80% of documents). We also allowed our vectorizer to construct features of **bigrams** as well as unigrams so that we could detect word pairings that may be more indicative of sentiment than standalone words. By applying the TFIDF Vectorizer with this specified cleaning, we expect that the resulting classifier would be able to better discern sentiment due to the nature of how the value of features is adjusted based on the relative frequency of words occurring within a document and the entire vocabulary.

b) Similar to part 1, we used the SKLearn **KFold** and **GridSearchCV** objects to perform cross validation. The KFold object was initialized to the **same parameters as in part 1**. Our cross-validation procedure differed from Problem 1 substantially with regards to the hyperparameters passed to GridSearchCV. We configured our code to iterate over various classifier objects and then use GridSearchCV to optimize hyperparameters associated with each one. After the optimal hyperparameters were generated for each model scored on the AUROC, we compared its score to the best score of the other models and saved the one with the best score.
   - `Logistic Regression: (solver = 'lbfgs', max_iter = 10000, penalty = 'L2'), {C: np.logspace(-9, 8, 50)}`
   - `Random Forest: (n_jobs = -1) ,{max_features : 2, 4, 8, max_depth: 2, 4, 8, n_estimators: 50 100 150 250 300}`
   - `Gradient Boosting Classifier: {learning_rate: [0.01, 0.05, 0.1], n_estimators: [50, 100, 150], max_depth: [3, 4, 5]}`

c) After performing cross validation across various classifiers, it appeared that the **Random Forest** would give the best performance. As such we proceeded with training by performing further hyperparameter selection on the Random Forest. Here we used the same **GridSearchCV** procedure as in part 1 to further optimize the random forest over a greater array of potential hyperparameter combinations. The graph below shows its performance over a subset of the values we used (n_estimators, max_depth, max_features), that gives a good example of different performances over a wide range of hyperparameters:



d) Below are some informative examples of false positives and false negatives of our second model in the validation set:

False Positive: 'If you are looking for a nice quality Motorola Headset keep looking, this isnt it.'

False Negative: ‘Much less than the jawbone I was going to replace it with.’

The inclusion of bigrams seemed to protect the model from misclassification of simpler litotes, however as we can see by the false positive example, the model still struggles with sentences that contain **negation** at the end of a more complex sentence structure. Understanding the relationship between different sections of a sentence like this may require consideration of larger numbers of combinations of words than just bigrams. Also, as can be seen in the false negative example, it struggles in sentences with **implied context**. The sentence ‘Much less than the jawbone I was going to replace it with.’ does not explicitly mention price, in which less is better, so the classifier is unable to determine that less than something is a good thing.

e) Ultimate Test Performance: **0.9077**. Our performance here was **better than our problem 1** performance of 0.88559. The performance improved due to the change of vectorizer and from using a different regression model. The **TFIDF** vectorizer appeared to do better than the count vectorizer when testing on the same regression model due to its improved capacity to determine the **relative importance of words**. The inclusion of **bigrams** as features also boosted performance, as word pairings are sometimes more meaningful than standalone words. Also, using a **random forest classifier** would lead to performance in this classification task as it is better suited for **nonlinear relationships** in sentence structure. They are better suited for capturing the **relationships between features**, which is key for analyzing sentences where the **context** and interplay of words is key.

Resources:

Feature Extraction: https://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-representation

Tuning Hyperparameters of an Estimator: https://scikit-learn.org/stable/modules/grid_search.html#grid-search

TFIDF Vectorizer: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer

Count Vectorizer: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer

Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html