Machine Learning Project 02

Cecelia Crumlish

Question 1: Pipeline

My tokenizing function

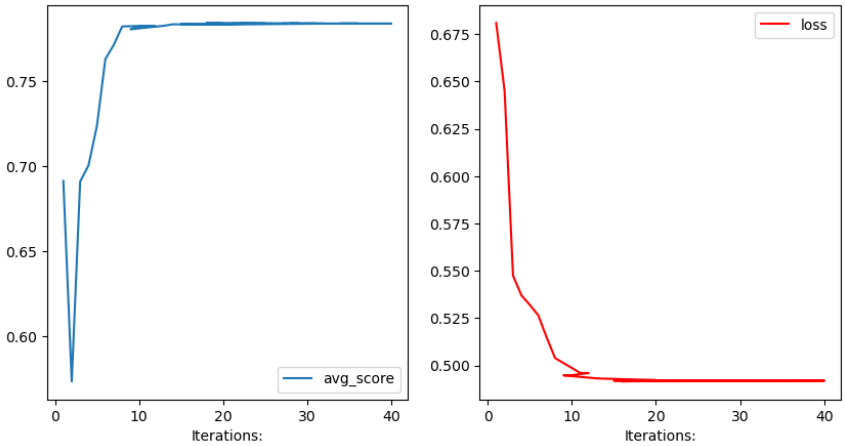
I used a regular expression to convert to a universal case and remove non-English characters. And then I used the SnowBallStemmer(“english”) set to remove word stems. I chose to use the snowball stemmer over the Porter stemmer, as it can handle a larger number of inflectional forms. In addition to this, it had a ~.03 percent performance boost in comparison to Lemmatization. This is probably because lemmatization groups inflected forms together into the same feature, which may have caused the Frequency-inverse document Vectorizer to ignore important words that have a similar meaning. For my Vectorizer I chose to use the SKlearn Frequency-inverse document Frequency Vectorizer (TfidVectorizer()). I chose this vectorizer because initially, I had started with the simple CountVectorizer, and I was overfitting on very common words, and I also had a very large feature set. One of the benefits of using this vectorizer is that it reduces the importance of common words such as “the" or “and”, allowing less-frequent words to have more importance. One of the drawbacks of this particular method however is that it can become quite overfit, given a vector with a lot of common words and maybe a very very uncommon word, it will weigh the uncommon word very highly even if that word isn’t strongly related to the given input.

Other things I tried:

In the last part of the project I tried to experiment with NLTK’s POS tagger and some spaCy pre-processing but It was just a lot of content and I wasn’t able to get very promising results. In fact when I implemented basic POS tagging my testing score went down to .42%

Question 02: Logistic Regression Model

The Hyperparameters that I chose for testing were inverse penalty C and solver. I systematically tested my hyperparameters by using the GridSearchCV function, with the parameter space above. I specified the GridSearch to include a 5-fold cross-validation, and score based on 'f1' recall scoring.

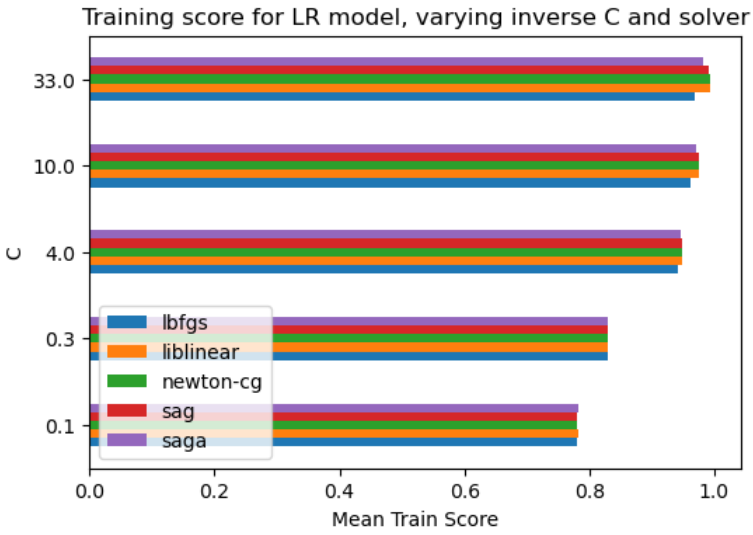


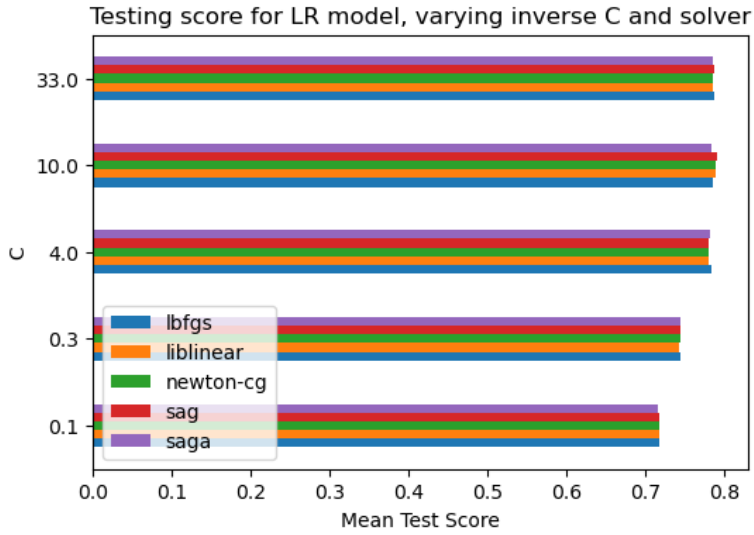
Before varying other hyperparameters I decided to narrow down the best Max_Iter by looking at the Lbfgs model with MAX_ITER's: 1 - 40. As shown by the graph below the areas I want to test look to be around MAX_ITER = 5 - 25, because that is when the loss is being reduced and the avg_score plateau's out, any further and you can see overfitting start to happen around the 25 - 40 range.

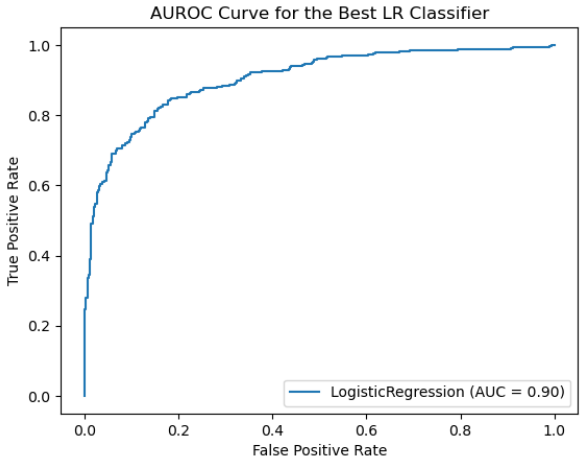
Overall my testing parameter space was
penalty = 'l2'
Solver = ['lbfgs', 'liblinear', 'newton-cg', 'saga', 'sag'],
C= [.1, .3, 4, 5, 10],
Cross validated: with 5-fold cross validation
Max-iter: [16]

Uncertainty:
Across the model folds, there was an average standard deviation of .043

Graph Analysis







Analysis:

It seems that the best scoring is with a higher inverse C value. Models with less regularization, and a higher inverse C value performed better on test and train than models with higher regularization values, the best performing inverse regularization strength was 10, and higher strengths than 10 ended up performing worse on the testing. The model with C = 33 performed ~ .2 worse on the testing than the model with C = 10, in all solvers. This is because after a certain point the model becomes overfit, as it becomes less and less regularized. In terms of the best solver, for C = 10, both 'liblinear' and 'newton-cg', achieve the best results of .79. The reason for this may be that both of these solvers are best suited for binary classification problems, the 'liblinear' solver can handle both l1 and l2 regularization which is useful in controlling model complexity in binary classification problems. In terms of the 'newton-cg' solver, this solver may be performing well because it finds the minimum cost function which can be effective for finding the best decision boundary in a binary classification problem.

The overall best parameters for this model were, C = 10, and solver = liblinear / or newton-cg. This model ended up scoring a mean testing score (over five folds) of .79 testing and a mean training score of .97 .

Question 3, (MLP)

I chose to construct my MLP model using the SKLEARN MLP() constructor. Again I varied the hyperparameters using GridSearchCV, with 5-fold cross-validation, and 'f1' scoring. I decided to choose the 'tanh' activation function and the 'adam' solver because they are commonly used for a larger range of tasks. And they play well with increasing layers and alpha values. For the hyperparameter values, I chose to specify the hidden layer sizes with sizes 50 and 50, 100, 50, and 100 to cover a wide range of the possible values available. I chose Alpha the regularization strength as well to vary regularization strength between little to none and a lot of regularization.

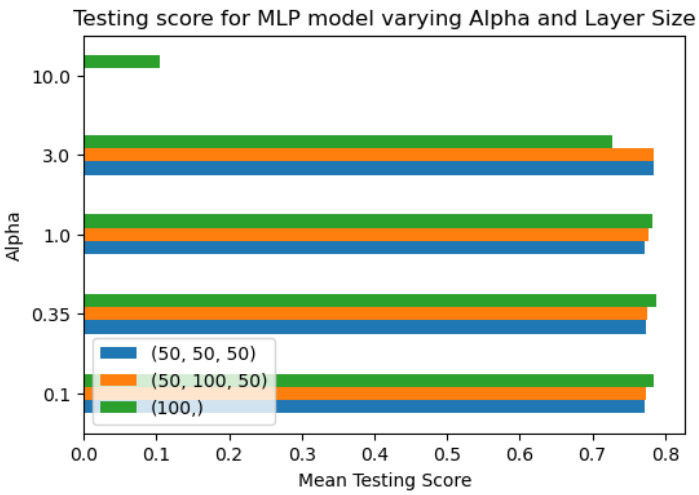
Overall my testing parameter space was

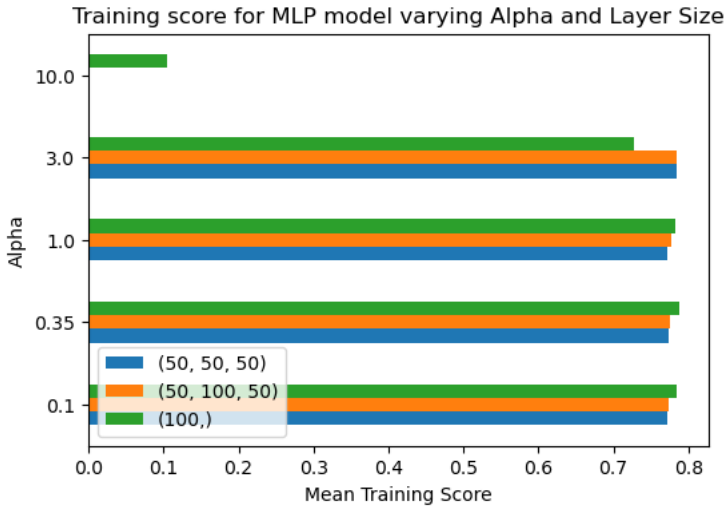
Hidden layer sizes: [(50,50,50), (50, 100, 50), (100)]
Activation: 'tanh'
Solver: 'adam'
Alpha: [.1, .35, 1, 3.0, 10]
Learning_rate: ['constant']

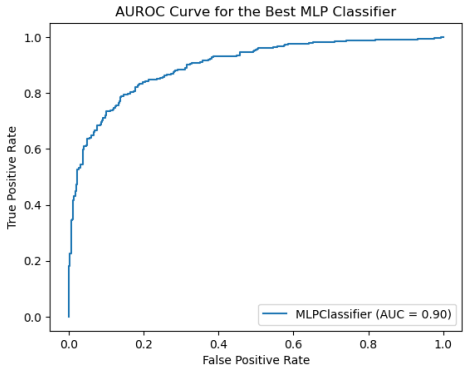
Uncertainty:

Across the model folds, there was an average standard deviation of .034

Graphs







Analysis:

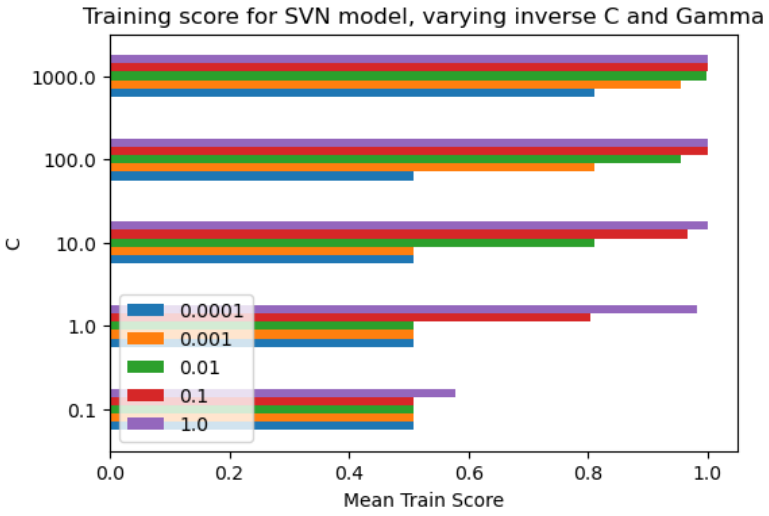
The above graphs demonstrate the model's tendency to underfit as the value of alpha increases with the highest alpha value of 10, scoring the lowest score of all the models (.15). The best layer size was found to be around 100, although as regularization strength increased the performance of the hidden-layer 100 models diminished before the other layers. This may be occurring because models with smaller layer sizes like the (50, 50, 50) model are less affected by regularization. After all, the neurons are already less-complex and less able to overfit. After the Grid analysis, It appears that the MLP performs the best with the best train/test score at alpha = .35, and hidden_layer_size: (100), and a score of .788 on the test and 1 on the train. Given the relatively high training score, it is more likely that this MLP was slightly overfitted.

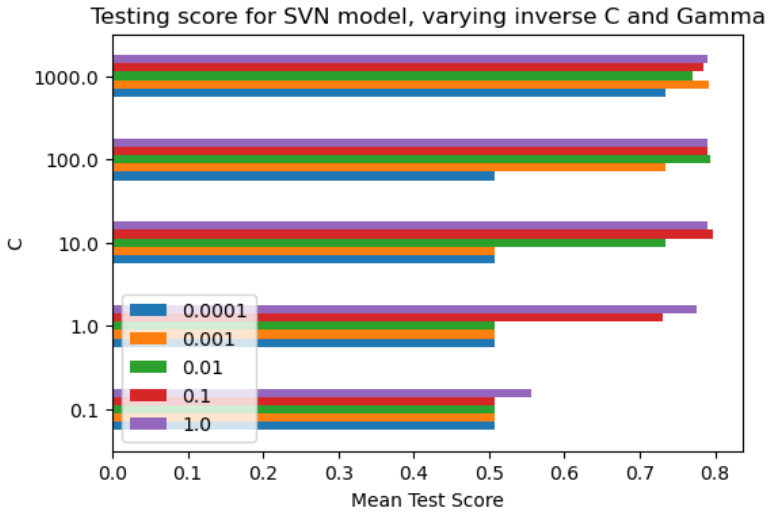
Question 4, SVM

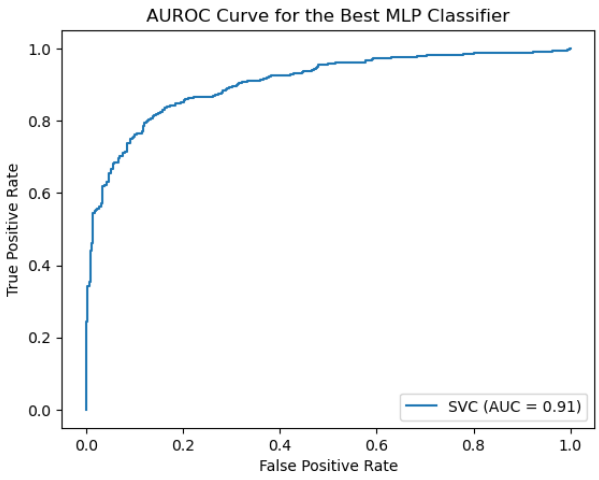
For the model of my own choice, I choose to try the SVM model that Sklearn provides. Like the rest of the models up to this point, I varied parameters by using GridSearchCV and specifying a parameter space that included 5- fold cross-validation, and 'f1' scoring. The hyperparameters I chose to investigate were the inverse C penalty and gamma. I decided to use the 'rbf' kernel as the default kernel function because of its flexibility to parameter changes, other kernel functions were harder to vary in the parameter space, I experimented with linear, polynomial, and sigmoid functions but they did not play nice with a lot of varying C and Gamma along a range of values.

Overall Testing Parameter Space

C : [.1 , 1, 10, 100, 1000],
Kernel :['rbf'],
Gamma: [1, .1, 0.01, 0.0001],







Analysis

The higher values of gamma, tended to perform much better in the train scoring, this is because the grammar parameter determines how sensitive the model is to training examples, this can explain why the model with the best testing score was overfitting, with a testing score of .797 and a training score of 1. Above models with higher gamma values have more complex decision boundaries which are why the highest gamma value of 1(pictured in purple) tended to score well even with a very low inverse regularization (see above test and train graphs). The best settings found for the SVM were C = 10, and gamma = .1, at this point the model performance. AUROC curve for the best SVM classifier, slightly outperformed the other models by an AUC score of .91. However this model may be more overfit than the MLP and LR models, given that it has a Train score of 1.

Question 5, Summary, Best Classifier and Common Mistakes

The classifier that performed best on the labeled testing data was the SVN, which was followed closely by the Logistic Regression model (LR), and then the MLP model. The SVN may perform better on this particular given testing set, however, this model does have a training score of 1 which leads me to believe it may be overfitting.
Stats for the svn:

	precision	recall	f1-score	support
0	.81	.85	.83	348
1	.85	.82	.84	372
Macro avg	.83	.83	.83	720
Weighted avg	.83	.83	.83	720

I decided to find and group common false-positive Mistakes, looking at a set of 6 false-positive features, the words that came up from those mistakes were: of, ever, and not. The most common false-positive mistakes happened to be: ride, much, back, very, convenient, clear, and out. My hypothesis for the common false negatives is that sentences with combined double negatives like not bad, and the best, versus the worst ever, or nothing has ever been that good. If I had more time to pre-process my data, I might look into POS tagging, which takes in word positions or use bi-grams with the most common double negatives. In terms of false positives, it appears as though the words clear, out, and back were commonly identified to be correlated with positive reviews. My working hypothesis for these identifiers is that they are modifiers to words that may be positive and negative such as very bad, very good, and much bad/good. One thing that may be useful to do after TFID is to consolidate words, which is to combine like words into their parts of speech such as very, much, and a lot, etc. this would help with overweighting similar words and also in making the feature-space smaller.

Question 6, (application)

In the report, my SVN classifier has an AUROC of **.845**, and an error rate of **.155.** The SVN's performance on gradescope is **.015%** better than the model's weighted average. Scoring about as well as the model's precision during training cross-validation. An increase in performance could be because I used all of the labeled data as training data before getting the SVN's prediction for the unlabeled test data. Given more examples of data, it would make sense that a marginal improvement is seen in the model testing accuracy and precision.