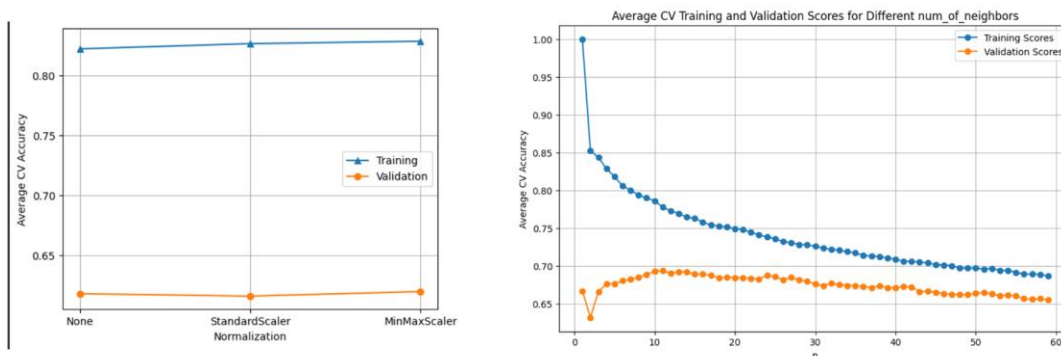


Brady Gehrman
CSE 546
Final Project Report

Completed Experiments:

KNN Classifier:

The first classifier I chose to implement is the KNN classifier because it is the most basic one. I first used cross validation to find the average cross validation scores for the KNN model when it was given different types of normalization data, the results can be seen below in plot 1. I ended up deciding to utilize the MinMaxScaler to normalize my data for the remaining trials because of its improved validation accuracy compared to the other preprocessing methods. I then decided to find the optimal n parameter and distance metric for the KNN classifier using cross validation and pipelined grid search. I chose the distance metric to be cosine and the n parameter to be 15, yielding me an average cross validation accuracy of 66.26%. Since the accuracy value seemed low and cases of underfitting were found, I then chose to experiment with a linear model.



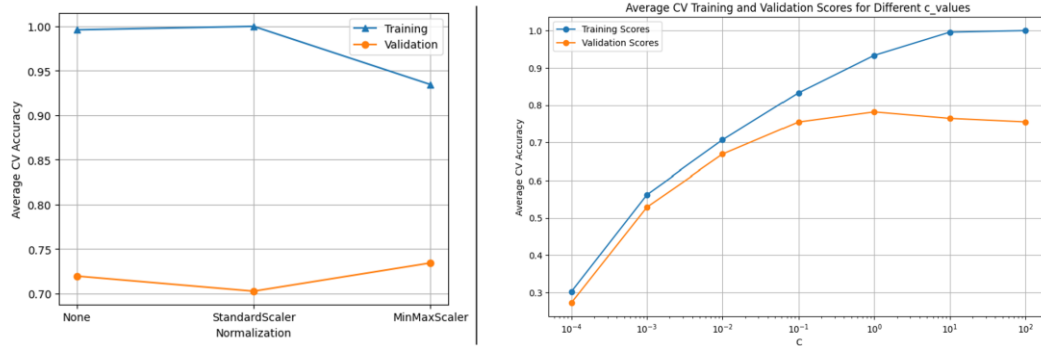
Plot 1: Plot of average cross validation scores versus data normalization and n parameter.

```
Best metric for KNN: cosine
Best number of neighbors for KNN: 15
Best cross-validation accuracy for KNN Classifier: 0.6626
```

Image 1: Screenshot of the pipelined gridsearch results for the KNN classifier.

Logistic Regression Classifier Model:

The next model that I chose to investigate was the logistic regression classifier. I first used cross validation to find the average cross validation scores for the LR model when it was given different types of normalization data, the results can be seen below in plot 2. I ended up deciding to utilize the StandardScaler to normalize my data for the remaining trials because of its improved validation accuracy compared to the other preprocessing methods. I then decided to find the optimal C parameter for the LR classifier using cross validation and pipelined grid search. My optimal C value was found to be 1, yielding me an average cross validation accuracy of 73.42%. This was an improvement from the KNN model with regards to accuracy, but I also wanted to experiment with more advanced models, so I chose to shift my focus to some non-linear models.



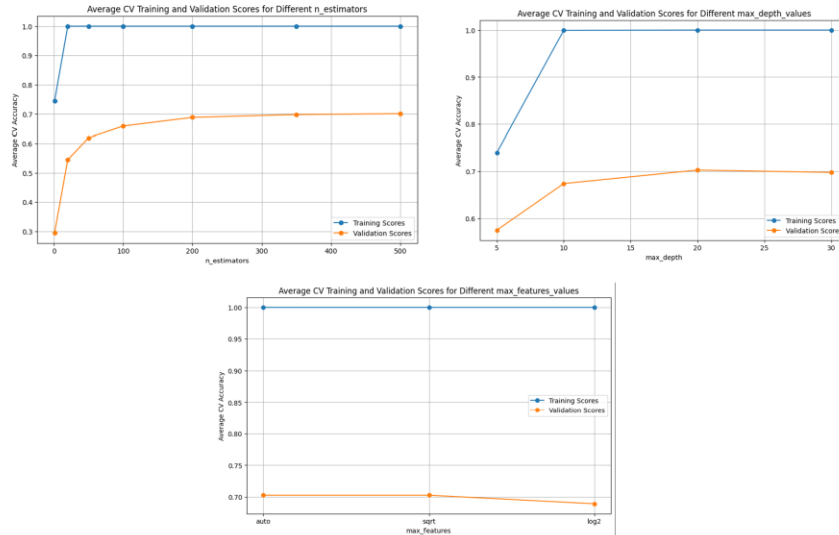
Plot 2: Plot of average cross validation scores versus data normalization techniques and the C parameter.

```
Best regularization strength (C) for Logistic Regression: 1
Best cross-validation accuracy for Logistic Regression: 0.7342
```

Image 2: Screenshot of the pipelined gridsearch results for the LR classifier.

Random Forest Classifier:

The first non-linear classifier that I experimented with was the random forest classifier. The RF classifier doesn't see any improvements when adding data normalization, so I skipped that and went straight to finding the optimal parameters for the model. The optimal parameters in question are the number of estimators, max depth, and max features. I optimized these parameters for the RF classifier using cross validation and pipelined grid search. I chose the number of estimators to be 500, max depth to be 25, and the max features to be 'sqrt,' yielding me an average cross validation accuracy of 73%. This accuracy value was a bit lower than the one received using the logistic regression classifier, but I still wanted to explore a few other non-linear methods.



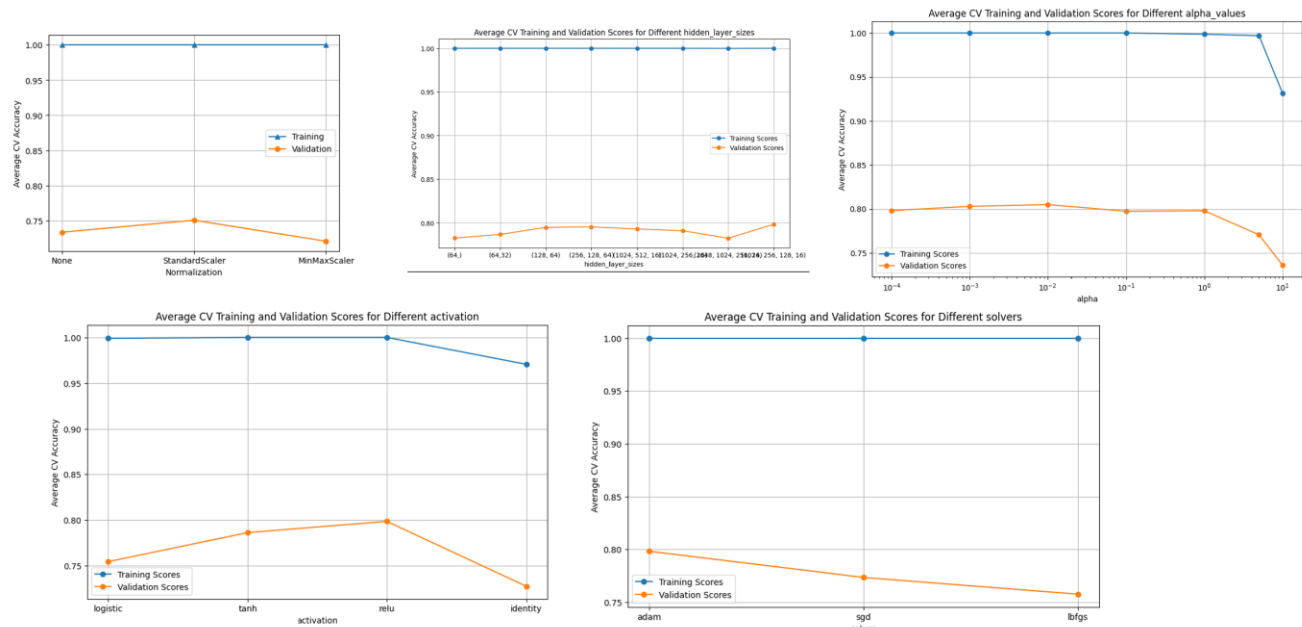
Plot 3: Plot of average cross validation scores versus number of estimators, max depth, and max features.

```
{'max_depth': 25, 'max_features': 'sqrt', 'n_estimators': 500}
Best n_estimators parameter for Random Forest Classifier: 500
Best accuracy on test set: 0.73
Best AUC of ROC on test set: 0.9483905555555556
Best F1-measure (macro) on test set: 0.7310130257369403
```

Image 3: Screenshot of the pipelined gridsearch results for the RF classifier.

MLP Classifier:

The next classifier that I wanted to experiment with is the MLP classifier. I first used cross validation to find the average cross validation scores for the MLP model when it was given different types of normalization data, the results can be seen below in plot 4. I ended up deciding to utilize the StandardScaler to normalize my data for the remaining trials because of its improved validation accuracy compared to the other preprocessing methods. I then decided to find the optimal hidden layer sizes, activation method, solver, learning rate, and alpha for the MLP classifier using cross validation and pipelined grid search. My optimal hidden layer sizes were found to be value was found to be (128, 64), optimal activation method found to be 'relu', optimal solver to be 'adam', optimal learning rate to be constant, and optimal alpha value found to be 0.1, yielding me an average cross validation accuracy of 76.16%. This was an improvement from all of the previous models with regards to accuracy, but I also wanted to experiment with the Kernel SVM model to see what it had to offer because I noticed signs of overfitting within my optimal MLP classifier.



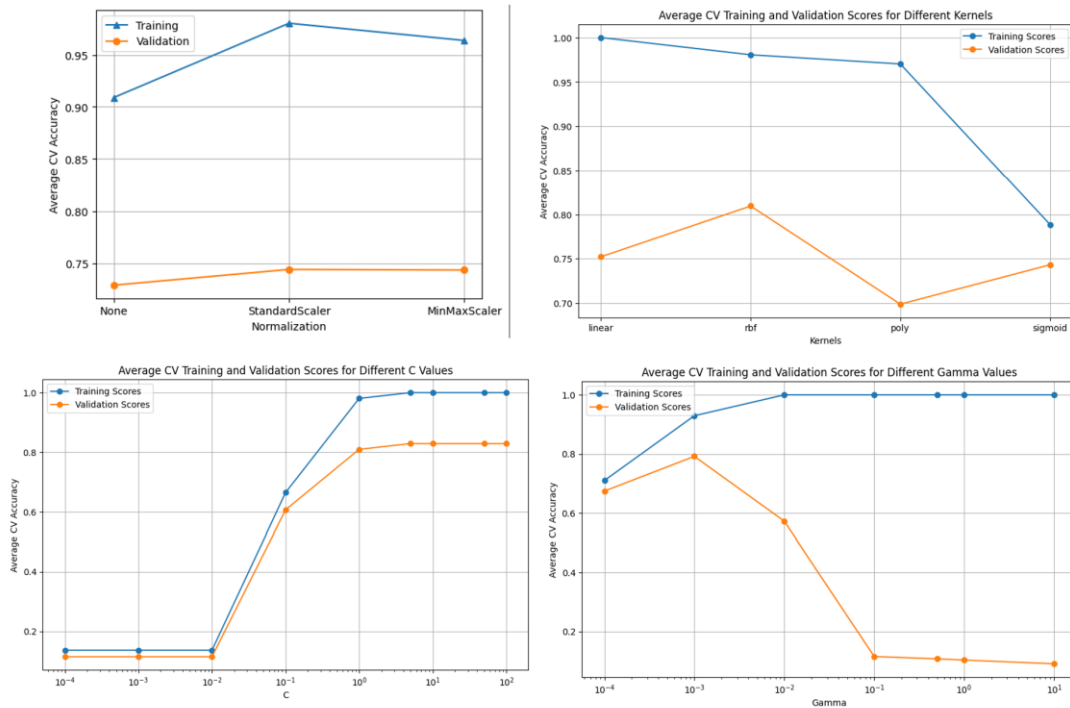
Plot 4: Plot of average cross validation scores versus data normalization techniques, hidden layer sizes, activation, solver, and alpha.

```
Best hidden_layer_sizes value for MLPClassifier: (128, 64)
Best activation value for MLPClassifier: relu
Best solver value for MLPClassifier: adam
Best learning_rate value for MLPClassifier: constant
Best alpha value for MLPClassifier: 0.1
Best cross-validation accuracy for MLPClassifier: 0.7615999999999999
```

Image 4: Screenshot of the pipelined gridsearch results for the MLP classifier.

Kernel SVM Classifier:

The next classifier that I wanted to experiment with is the Kernel SVM classifier. I first used cross validation to find the average cross validation scores for the Kernel SVM model when it was given different types of normalization data, the results can be seen below in plot 5. I ended up deciding to utilize the StandardScaler to normalize my data for the remaining trials because of its improved validation accuracy compared to the other preprocessing methods. I then decided to find the optimal kernel, C, and gamma values for the Kernel SVM classifier using cross validation and pipelined grid search. The optimal kernel was found to be 'rbf', optimal C value to be 5, and optimal gamma value to be 0.001, yielding me an average cross validation accuracy of 76.24%. This was an improvement on all the previous models with regards to accuracy and yielded the best results for a single model!



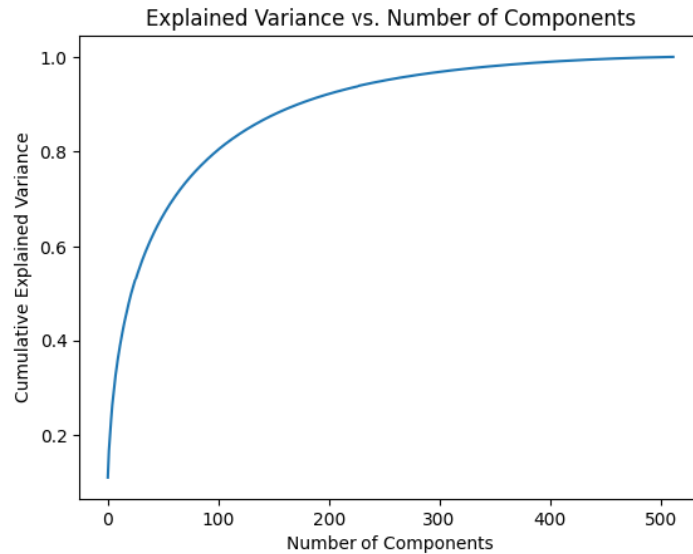
Plot 5: Plot of average cross validation scores versus data normalization technique, kernel, C, and gamma.

```
Best C value for Kernel SVM: 5
Best gamma value for Kernel SVM: 0.001
Best cross-validation accuracy for Kernel SVM: 0.7624
```

Image 5: Screenshot of the pipelined gridsearch results for the Kernel SVM classifier.

PCA Analysis:

I also decided to investigate how I could utilize PCA to reduce the dimensionality of my dataset and hopefully improve the performance of my models. In plot 6, below, you can see the graph of the cumulative explained variance versus the number of components used within the PCA. I ended up choosing to investigate using 125 components and 250 components for PCA experiments.



Plot 6: Plot of number of PCA components versus explained variance.

Logistic Regression:

```
Logistic regression (PCA=125) accuracy for selection
Best Parameters: {'classifier__C': 0.01}
Mean Test Scores for Each Fold: [0.5742 0.6684 0.6924 0.676 0.6668 0.6684 0.6704]
Best cross-validation accuracy for Logistic Regression without PCA: 0.7342
```

```
Logistic regression (PCA=250) accuracy for selection
Best Parameters: {'classifier__C': 0.01}
Mean Test Scores for Each Fold: [0.5856 0.6882 0.722 0.707 0.6768 0.6584 0.6432]
Best cross-validation accuracy for Logistic Regression without PCA: 0.7342
```

Image 6: Screenshot of the pipelined gridsearch results for the LR classifier using PCA.

MLP Classifier:

```
MLP classifier (PCA=125) accuracy for selection
Best Parameters: {'classifier__activation': 'relu', 'classifier__alpha': 0.1, 'classifier__hidden_layer_sizes': (128, 64),
Mean Test Scores for Each Fold: [0.7098 0.7146 0.7032 0.7088 0.7236 0.7082 0.7096 0.7174 0.7108 0.7366
0.7344 0.723 ]
Best cross-validation accuracy for MLPClassifier without PCA: 0.7616
```

```
MLP classifier (PCA=250) accuracy for selection
Best Parameters: {'classifier__activation': 'relu', 'classifier__alpha': 0.1, 'classifier__hidden_layer_sizes': (256, 128, 64),
Mean Test Scores for Each Fold: [0.7264 0.7342 0.7246 0.7242 0.732 0.721 0.7266 0.7396 0.7326 0.7482
0.7482 0.7228]
Best cross-validation accuracy for MLPClassifier without PCA: 0.7616
```

Image 7: Screenshot of the pipelined gridsearch results for the MLP classifier using PCA.

Kernel SVM:

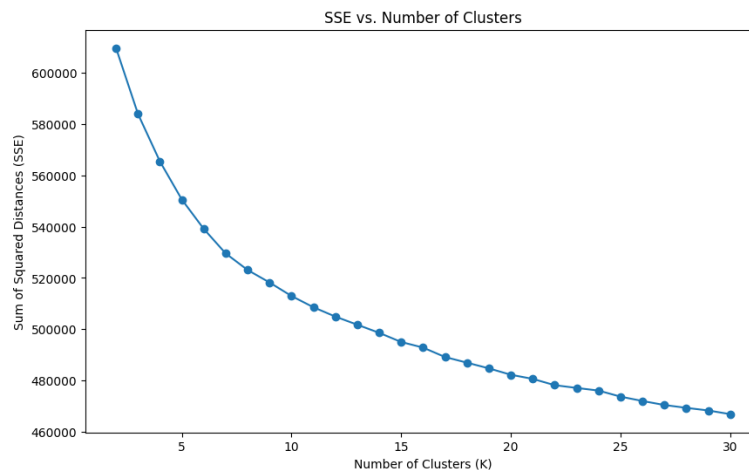
```
SVM classifier (PCA=125) accuracy for selection
Best Parameters: {'classifier__C': 100, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf', 'pca__n_components': 125}
Mean Test Scores for Each Fold: [0.4964 0.3452 0.4972 0.3466 0.5864 0.3474 0.7354 0.654 0.7552 0.6682
0.761 0.6694]
Best cross-validation accuracy for Kernel SVM without PCA: 0.7624
```

```
SVM classifier (PCA=250) accuracy for selection
Best Parameters: {'classifier__C': 10, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf', 'pca__n_components': 250}
Mean Test Scores for Each Fold: [0.4914 0.4258 0.491 0.425 0.5762 0.4712 0.7436 0.7248 0.7668 0.7406
0.7666 0.74 ]
Best cross-validation accuracy for Kernel SVM without PCA: 0.7624
```

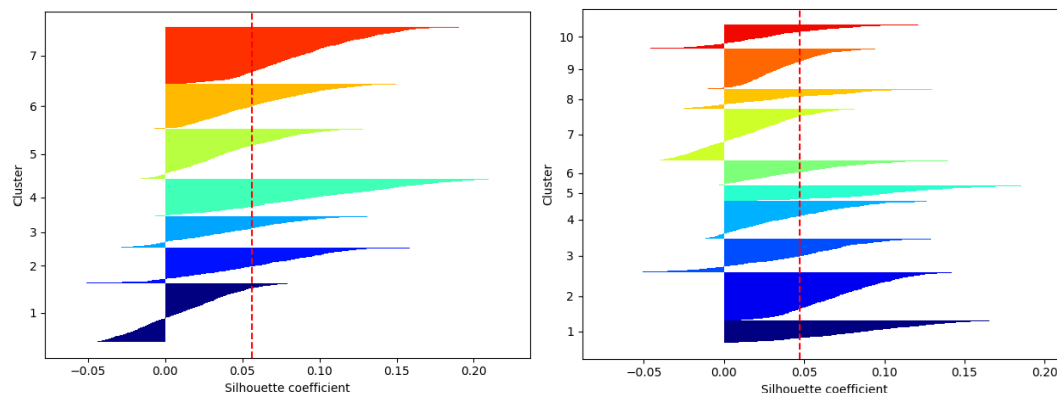
Image 8: Screenshot of the pipelined gridsearch results for the Kernel SVM classifier using PCA.

After looking at my cross-validation results for the three models that were trained using the PCA, I then concluded that utilizing PCA within my final model would not be a good choice because of the smaller cross validation accuracies I was receiving for those models. These models were each trained using a pipeline that allowed for data normalization, PCA, and a selected classifier. Those models were then tested using cross validation.

Data Mapping Using Clustering:



Plot 7: plot of K-means clustering number of clusters (k) versus sum of squared distances.



Plot 8: Silhouette plots for 7 clusters and 10 clusters.

The unsupervised learning approach of data clustering using the K-means clustering algorithm was explored in the two plots above. Through careful evaluation plot 7, I decided to investigate how 7 and 10 data clusters could be used for data mapping purposes. In conclusion, through the analysis of the Silhouette plots, adjusted rand index calculations, and several experimental models I concluded that data mapping using clustering didn't offer positive accuracy benefits for our dataset.

Ensemble Methods:

After creating the most optimal models for a couple single classifiers, I then decided to investigate how Scikit Learn's ensemble methods could be used to produce a higher accuracy. I was unsure which models to combine and or enhance, so for each ensemble method I ran a couple trials for different classifiers to try to find any pattern that I could take advantage of.

Bagging:

Bagging is an ensemble method that trains a specific base classifier multiple times on different subsets of the training data, and then combines their predictions to reach a conclusion. I investigated the KNN classifier, decision tree classifier, and kernel SVM classifier when bagging was applied. Overall, I found a couple of decent boosted classifiers, but nothing that stood out to me with regards to accuracy.

KNN:

```
Best Parameters: {'estimator__metric': 'cosine', 'estimator__n_neighbors': 15, 'n_estimators': 15}
Best Cross-validated Accuracy: 0.66
```

Image 9: GridSearch results for KNN base classifier.

Decision Tree:

```
Best Parameters: {'estimator__max_depth': 20, 'estimator__min_samples_split': 5, 'n_estimators': 20}
Best Cross-validated Accuracy: 0.51
```

Image 10: GridSearch results for decision tree base classifier.

Kernel SVM:

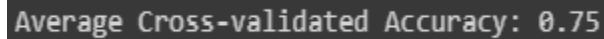
```
Best Parameters: {'estimator__C': 10, 'estimator__kernel': 'rbf', 'n_estimators': 15}
Best Cross-validated Accuracy: 0.75
```

Image 11: GridSearch results for Kernel SVM base classifier.

Ada-Boosting:

Ada-boosting is an ensemble method that looks to combine multiple weak classifiers to create a boosted classifier. The boosted classifier trains each weak classifier on weighted versions of the training data. The training data is weighted based on the previously misclassified samples. I investigated the decision tree classifier and the logistic regression classifier. Both base classifiers

gave an accuracy of around 75%, which was interesting to see, but didn't offer any advantage with regards to accuracy compared to my previously optimized models.



Average Cross-validated Accuracy: 0.75

Image 12: Accuracy results for DT base classifier.



Average Cross-validated Accuracy: 0.75

Image 13: Accuracy results for LR base classifier.

Stacking:

The stacking ensemble method combines several base models and then passes their predictions to a meta learner which then uses that information to form a prediction. I used a random forest classifier and a gradient boosted classifier as my base classifier and a logistic regression classifier as my meta learner. I only performed one experiment for this ensemble method because it produced such a low accuracy and took a very long time to execute.

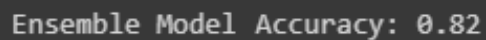


Average Cross-validated Accuracy: 0.63

Image 14: GridSearch results for (RF + GB) -> LR stacked classifier.

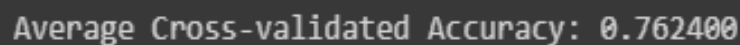
Voting:

The voting ensemble method was one that I wanted to investigate from the start because I knew it would allow me to combine all my best classifiers together in a simple and explainable way. This ensemble method excels when it is given many “different” input models; therefore, I used my optimal logistic regression, random forest, MLP, and kernel SVM classifiers as the base classifiers. By combining these classifiers together in the most optimal way, I received an accuracy that was around 1-2 percentage points higher with regards to cross validation than the original models themselves. This ended up giving me a testing accuracy of around 82%, which was the best that I had received so far, so I ended up choosing this voting classifier as my optimal classifier for our given data set.



Ensemble Model Accuracy: 0.82

Image 15: Testing accuracy for my optimal voting classifier.



Average Cross-validated Accuracy: 0.762400

Image 16: Average CV accuracy for my optimal voting classifier.

Image Classification:

After selecting my optimal classifier, I then decided to investigate which images it had an easy time classifying and which ones it had a tendency to misclassify. I gathered those image sets in image 17 and 18 below. My optimal model seemed to have an easy time classifying pictures of vehicles where the back view was present, but tended to misclassify the vehicle images where just the front view was available or the images themselves were outliers (headlights were on or the doors were open).



Image 17: set of images that were easy for my optimal model to classify.

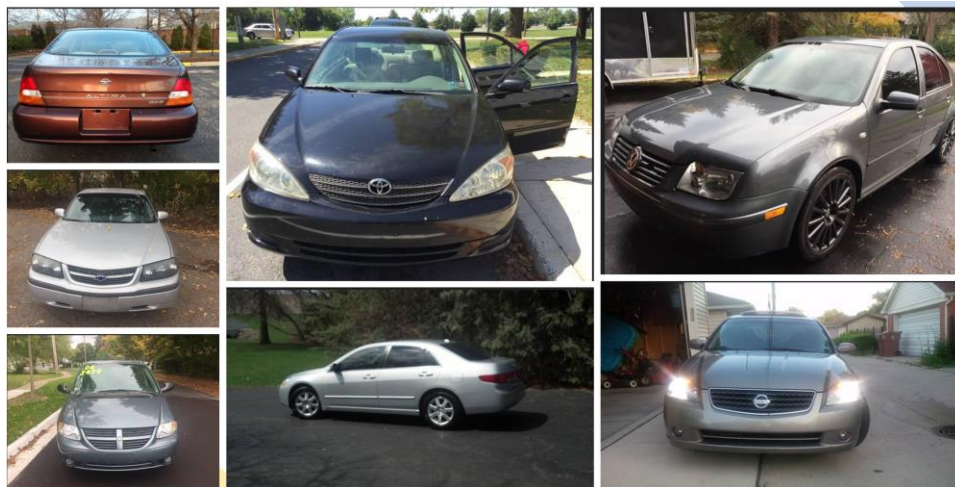


Image 18: set of images that were misclassified by my optimal model.