Jacob Meyers
Assignment 1 - Supervised Learning
CS 4641
10 February 2019

The two classification problems I decided to try to solve were classification of red wine quality and classification of car quality. When I sought out datasets for this assignments, I had a few criteria in mind. First of all, is the data interesting to me, and would classifying the data be interesting also? My motivation for this one is simple - why would I try to solve a problem I have no real interest in solving? The second quality I looked for was a non-trivial amount of data. Supervised learning algorithms require a fair amount of data to get interesting results. For the purposes of this assignment, I felt as though data with less than 1000 instances would not be enough data to produce meaningful, interesting results. My last two criteria were not as important as the first, but played a role in my decision making nonetheless - little to no missing values and relatively few number of attributes. I wanted to avoid as much preprocessing as possible, partly out of not wanting to sacrifice instances of data and partly out of laziness. I wanted a relatively small number of attributes to mitigate the curse of dimensionality as much as possible. As you add features to a dataset, the amount of data you have to add to ensure you have a reasonable number of instances of all the combinations of features grows exponentially. I wanted to try to get a good ratio of data size to dimensionality of the feature space to ensure there would be good representation of all the combinations of features. Let's see how each dataset met my criteria.

The first dataset I selected was a red wine quality dataset. The format of the dataset is described in detail in Table 1. There are 11 continuous floating point attributes, and the target is a discrete value from 3 to 8 which describes the quality of the wine, with 3 being the worst and 8 being the best. This data was interesting to me because the features of the dataset are chemical properties of the wine such as pH and total sulfur dioxide, and I was interested to investigate the correlation between non-subjective properties of the wine and a subjective rating assigned to it by a human. There are 1599 instances in this dataset, which I felt were enough to be non-trivial for the investigative purposes of this assignment. The vast majority of the qualities represented are either 5 or 6. Intuitively, this makes sense - most wines will be average, and average wines will dominate over outliers on either end in the data. This can lead to problems when it comes time to classify the wines with our algorithms, however, as we still want to be able to recognize the outliers, and we may not have enough data to do so reliably. I don't love the number of features (11) with this amount of data, but I was convinced enough with everything else about this data to move forward with it.

The second dataset I selected was a car quality dataset. The format of the dataset is described in detail in Table 2. I was interested in this data for similar reasons to the wine dataset; that is, assigning quality scores to cars based on impartial attributes of the car is interesting to me. There are 1728 instances in this dataset, enough for me to consider the amount of data to be non-trivial. There are no missing values in this data, but it did require some preprocessing regardless. The data is presented as categorical data (i.e. descriptors like "acceptable", "unacceptable", "high", "low", etc), but what we really want for our classification algorithms is integral data. It was simple enough, however, to encode the categorical data into integers by mapping the possible values into an integer value. I felt much more comfortable with the dimensionality of the feature set vs. the amount of data for this dataset. There are 6 features in the car dataset, compared to 11 for a similar amount of data in the wine dataset. Overall, this dataset felt solid to me and I felt more than comfortable moving forward with it.

I selected 5 classification algorithms to test on my two datasets: boosted decision trees, a neural network, AdaBoosting with a decision tree classifier, support vector machines, and k-nearest neighbors. My procedure for testing each classifier was the same. For each dataset, I withheld the last 300 instances or so to be used later for final testing of the classifier. Then, with the remaining data, 3-fold cross validation was performed iteratively over 55%, 65%, 75%, 85%, and finally 95% of the data to generate the data for the learning curves. For each iteration, the train and test accuracy was averaged across the 3 folds. These learning curves are what I used to tune my hyperparameters of the algorithms. Once I was done tuning the hyperparameters, I trained on all of the cross validation data, and tested the final model on the withheld data to get a final confusion matrix and accuracy score. For all of the algorithms, I used the classifiers available in sklearn. Figures and tables begin on page 5, and contain both of the tables which describe the datasets and all the learning curves and testing results.

First up were decision trees. Sklearn doesn't currently support tree pruning, but a similar effect could be reached by changing the min_samples_leaf parameter, which dictates the minimum number of samples required to be a leaf node, and max_depth, which dictates the maximum depth of the tree. All the other hyperparameters I left alone because they seemed reasonable at their default values. Let's look at the wine data first.

When left at default values, the decision tree classifier unsurprisingly overfitted very strongly. The learning curve showed 100% training accuracy, but less than 45% cross-validation accuracy. This doesn't surprise me with the wine data. There are a lot of features, and all of them are floating point values. Aggressive pruning (min_samples_leaf=7, max_depth=2) seemed to work the best. This makes sense to me, as with the large number of attributes in this dataset it forced the classifier to single out what the most important attributes were, which will generalize better to unseen data. When tested on the withheld data, the classifier performed at around 58% accuracy. Something interesting to note in the confusion matrix is the fact that tree misclassified every instance of a 3 rating, 4 rating, and 8 rating, preferring to rate something closer to the more common ratings of 5 and 6. This is likely because of the lack of data on these ratings as mentioned earlier. The learning curve can be seen in Figure 1 and the accuracy score and confusion matrix in Figure 2.

The default valued decision tree performed much better on the car dataset, achieving around a 70% cross validation score and a 64% accuracy score on the test data. I attribute this to the fewer number of features and the much smaller range of values the features can take on in the car data. The car classifier still benefited from some pruning, albeit not as aggressively as with the wine data (min_samples_leaf = 7, max_depth = 4). On the test data, this classifier performed at 66.6% accuracy. Another interesting note is that the classifier misclassified all of the "vgood" cars as "unacc". This means that while it was relatively accurate, its misclassifications were not very close to the true value for a significant portion of the test data. Decision trees were also by far the fastest algorithm I used, taking 0.095 seconds for the wine dataset and 0.024 seconds for the car dataset to complete 5 iterations of 3-fold cross validation.

For boosting, I used variants of the previously used decision tree classifiers, but experimented with more aggressive pruning. I also set n_estimators to 1000, allowing sklearn to train 1000 decision tree models before halting.The first thing I noticed is the training time for AdaBoost was significantly longer than for decision trees. The cross validation time for the wine dataset was 27.16 seconds, and for the car dataset it was 23.24 seconds. These are roughly 30,000% and 110,000% increases in runtime, respectively. However, this is influenced strongly by the high value of n_estimators - reducing the number of estimators

to 50 brings the cross validation time down to around 1 second for both algorithms. Shockingly to me, the AdaBoost decision trees did not perform better on the wine dataset - in fact, it performed 2% worse on the final test data! The base decision tree classifier was almost the same as described above, only with max_depth set to 1, rather than 2. The learning curves are very similar in regards to cross-validation score, but the training score is much worse for the boosted trees. Why could this be? If we recall how boosting works, the algorithm attempts to iteratively train a weak learner, eliminating all but the "hardest" problems (the hardest data for the classifier to correctly predict the class of) with each iteration, with the idea being that we eventually arrive to a classifier which can perform well both on hard problems and easy ones. The fact that the boosted trees did worse on the wine set tells me that not only did the algorithm **not** learn the hard cases any better than the regular, pruned decision tree (as can be indicated by the similarities of the confusion matrices at the outliers), but the algorithm slightly overfitted towards the outliers in the process of trying to learn them. If we look at the confusion matrix for boosting on the wine data, we can see that it performed worse at classifying wines with a score of 5, which is one of the two most common classes. This is a very interesting result.

The car data had much more success with boosting. The accuracy score with the exact same base decision tree classifier used earlier tested at 88.6% accuracy, a 24% accuracy increase from just using decision trees. Looking at the confusion matrix, we can see that AdaBoosting did a much better job learning along the edges than the plain decision tree - the boosted trees got almost 100% accuracy for "unacc" labels where the decision tree missed around 40%, and correctly classified around 66% for "vgood" labels where the decision tree had 0% accuracy. This tells me the "hard" problems were much less difficult to learn than in the wine dataset. The smaller dimensionality of the feature set compared to the wine dataset almost certainly played a significant role in this result as well.

After boosting, I moved on to k-nearest neighbors. The only parameter I changed was k when testing. To find an optimal k, I started with the nearest odd number to sqrt(N), where N is the size of the training data, and iterated through every odd numbered value in between 3 and the initial value. I considered only odd numbered values to try to decrease the chance a tie may occur when the "voting" occurs for a particular data point. For the wine dataset, the N I settled on was 33, the second value I tested. Interestingly, there wasn't a significant drop-off in performance as K decreased until roughly K=5 or less. This suggests to me that the data in this dataset is clustered in such a matter that the nearest 7 data points are a decent indicator of what the point in question is, and looking at more neighbors beyond that doesn't gain us much information. Anything around 5 or smaller made it easier for noise to have an influence on the class predicted, and thus less accurate. Out of curiosity, I also decided to test a few values of K greater than sqrt(N). In theory, the more neighbors you poll, the more accurate the final result will be. However, once I got to values much higher than sqrt(N), KNN's performance dropped sharply. This is because once you reach large values of K, the algorithm is more likely to poll outliers when classifying a new point. This, of course, leads to overfitting and poor generalization to previously unseen data. KNN performed at around 60% test accuracy, with K = 33 for wine, a 2% improvement from decision trees and a 4% improvement from boosted decision trees. Cross validation took 0.38 seconds for the wine dataset, which is a bit slower than decision trees but much faster than boosting. KNN performed similarly for the car dataset, with 60.5% accuracy on test data and a 0.27 second cross-validation time.

For support vector machines, I used two kernels - linear and rbf. I chose these two kernels because I felt that if one of the datasets performed with, say, linear and not as well with rbf (or vice versa) it would indicate fairly well the shape of the data and how linearly separable it is. The hyper parameters I chose to tune are gamma and C, with gamma being a kernel parameter for rbf which determines how far the

influence of a single training example reaches, and C being the penalty for misclassification. Having a high gamma will allow the SVM to capture better the complexity of the data at the risk of being more likely to overfit. A low gamma will help the model be less prone to overfitting, at the risk of potentially missing a decision boundary which represents the data well. Similarly, a high C value will punish overfitting and lend itself to a low bias model, and vice versa. SVMs had a similar cross validation time to KNN, ranging from 0.19 seconds to 0.84 seconds across both kernels and datasets. The best linear kernel for the wine dataset had a C value of 1, the default. The C value had little to no effect on the accuracy of the model during my testing of the wine dataset, for either kernel. Since the C value essentially influences the size of the margin of the hyperplane, this tells me there isn't a lot of data concentrated around this decision boundary. To me, this means that this isn't the best boundary. The final accuracy of 58% supports this. There was a slight improvement when switching to an rbf kernel to 61.5% accuracy. The best gamma value I tried for the wine dataset was a very low value - 0.015. This makes sense to me, as there are a lot of features in this data and it would be very easy to overfit with a high gamma, leading to poor generalization to novel data. For the car dataset, the linear SVM performed rather poorly in comparison to the other algorithms on this data with an accuracy of about 57%. If we look at the confusion matrix, we see that the model classified every example as "good". This indicates that the data is probably clustered into shapes that are not linearly separable. However, switching to the rbf kernel with a gamma of 0.6 saw a massive accuracy improvement to 85%. This big difference between linear and rbf shows that the data is much more Gaussian than it is linear. A higher gamma was more tolerable for this data than the wine data. This is likely due to the fewer number of features and overall lower difficulty of the classification of this data. The gamma is quite a bit higher than the sklearn default of 1/(number of features), which in this case would be 0.16. This means that the model could create more complex curve around the training data without losing generality to the test data. This indicates to me the training data is a pretty good representation of the test data. I also was able to use a higher C value to get more accurate results, eventually settling on a value of 5.

Lastly were neural nets. For both datasets, I used a single hidden layer neural network, and varied the number of nodes in the hidden layer and the number of iterations allowed to run during training. I stuck with relu activation and the adam weight solver because these seemed like reasonable and likely effective defaults, and I chose to stay with one hidden layer because this is going to be enough for most problems, and I didn't feel that these classification problems were not overly difficult or complex to the point of requiring multiple hidden layers. Predictably, the neural nets ended up being one of the slowest algorithms, along with boosting. The most effective size of the hidden layer for the wine dataset ended up being what is evidently a common rule of thumb for hidden layer size - the mean of the input size and output size. It's unclear to me where this rule of thumb is from, as the sources I used simply referred to it as a rule of thumb with no further elaboration. Regardless, it was the most effective for wine, with a 59.7% accuracy. The most effective number of iterations ended up being 500. Interestingly, this was not enough iterations to allow the neural network's weights to converge. As a matter of fact, this model performed much better in testing than models which were allowed more maximum iterations and allowed to converge. I attribute this to the propensity of neural networks to severely overfit to data, if allowed. Stopping the training early put a check on the overfitting. I decided to try a larger number of nodes in the hidden layer for the car dataset, 100. The same number of max iterations, 500, was also the most effective. I presume that is for the same reasons as listed above, although the neural nets converged much faster with this dataset and, as you can see from the near 100% training accuracy, still likely overfitted to the training data a tad. I believe the large number of hidden units worked well in combination with the small number of iterations to prevent the weights from getting stuck in local minima due to the short training time.

Tables and Figures

Table 1. Wine Dataset Attributes (target in bold)

Attribute	Number of Instances	Data Type
Fixed Acidity	1599	64 bit float
Volatile Acidity	1599	64 bit float
Citric Acid	1599	64 bit float
Residual Sugar	1599	64 bit float
Chlorides	1599	64 bit float
Free Sulfur Dioxide	1599	64 bit float
Total Sulfur Dioxide	1599	64 bit float
Density	1599	64 bit float
pH	1599	64 bit float
Sulphates	1599	64 bit float
Alcohol	1599	64 bit float
Quality	**1599**	**64 bit integer**

Table 2. Car Dataset Attributes (target in bold)

Attribute	Possible Values	Number of Instances	Data Type
buying	“vhigh”, “high”, “med”, “low”	1728	Categorical
maint	“vhigh”, “high”, “med”, “low”	1728	Categorical
doors	“2”, “3”, “4”, “5more”	1728	Categorical

persons	“2”, “3”, “4”, “5more”	1728	Categorical
lug_boot	“small”, “med” “big”	1728	Categorical
safety	“low”, “med”, “high”	1728	Categorical
acceptability	**“unacc”, “acc”, “good”, “vgood”**	**1728**	**Categorical**

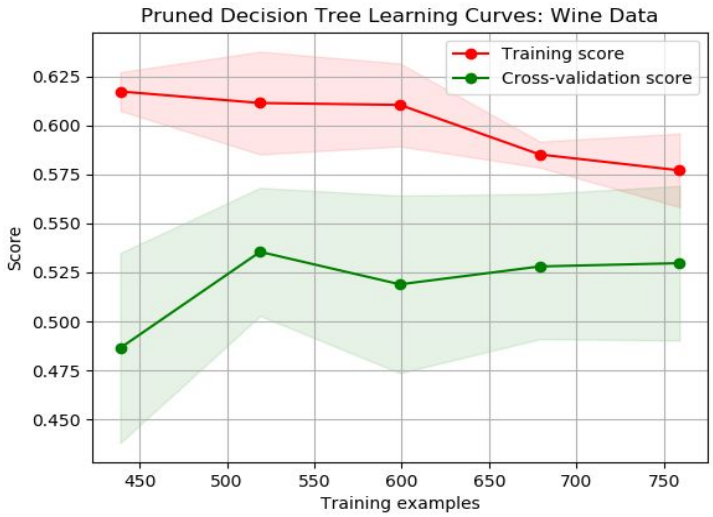

Figure 1. Learning curves for pruned decision tree on wine dataset

```
Accuracy: 0.5804020100502513
Confusion Matrix:
 [[  0   0   4   1   0   0]
 [  0   0  10   7   1   0]
 [  0   0 137  26   5   0]
 [  0   0  75  77  21   0]
 [  0   0   6   6  17   0]
 [  0   0   1   3   1   0]]
```

Figure 2. Final test data for pruned decision tree on wine dataset

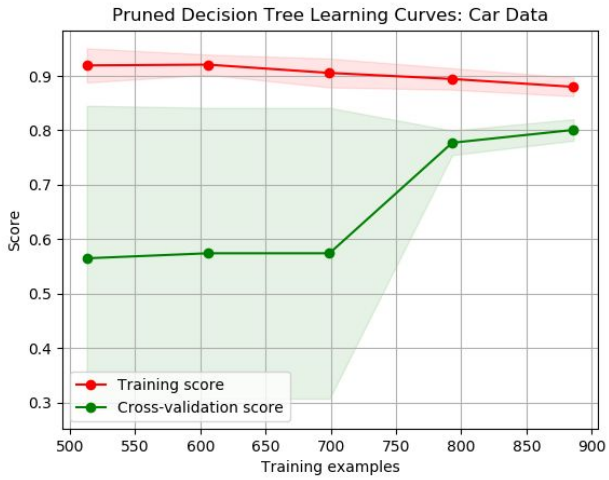

Figure 3. Learning curves for pruned decision tree on car dataset

```
Accuracy: 0.66666666
Confusion Matrix:
 [[ 34   0  21   0]
 [ 46   0   0   0]
 [  3   0 184   0]
 [ 39   0   0   0]]
```

Figure 4. Final test data for pruned decision tree on car dataset

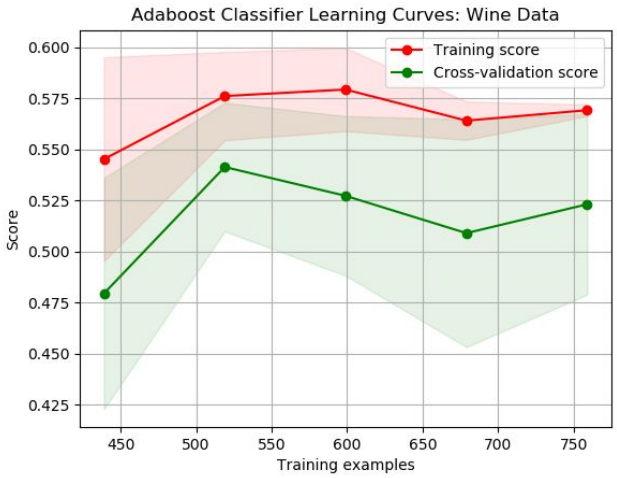

Figure 5. Learning curves for boosted decision trees on wine dataset

```
Accuracy: 0.5628140703517588
Confusion Matrix:
 [[  0   0   4   1   0   0]
 [  0   0   9   8   1   0]
 [  0   0 122  41   5   0]
 [  0   0  61  85  23   4]
 [  0   0   5   6  17   1]
 [  0   0   0   3   2   0]]
```

Figure 6. Final test data for boosted decision trees on wine dataset

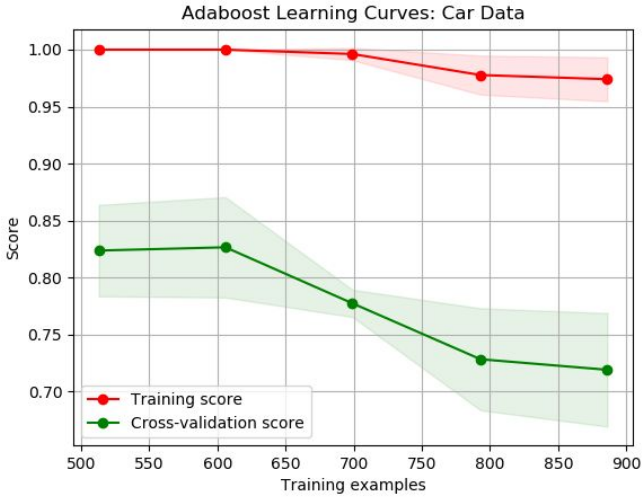

Figure 7. Learning curves for boosted decision trees on car dataset

```
Accuracy: 0.8868501
Confusion Matrix:
 [[ 54   0   1   0]
 [ 23  23   0   0]
 [  0   0 187   0]
 [ 13   0   0  26]]
```

Figure 8. Final test data for boosted decision trees on car dataset

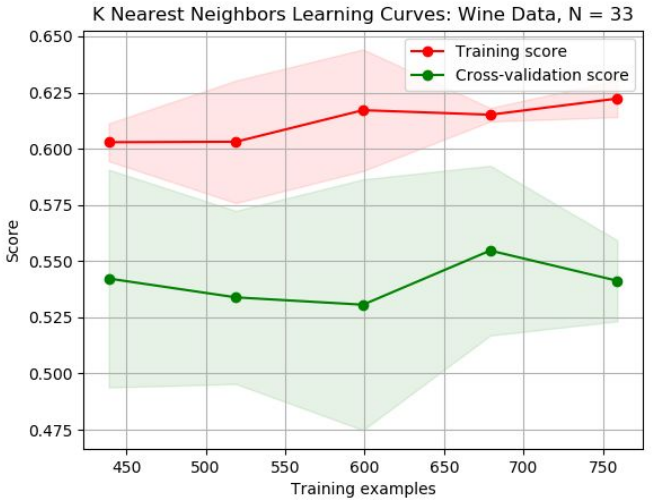

Figure 9. Learning curves for KNN on wine dataset

```
Accuracy: 0.6030150753768844
Confusion Matrix:
 [[  0   0   4   1   0   0]
 [  0   0  15   2   1   0]
 [  0   0 113  53   2   0]
 [  0   0  46 115  12   0]
 [  0   0   2  15  12   0]
_[  0   0   0   2   3   0]]
```

Figure 10. Final test data for KNN on wine dataset

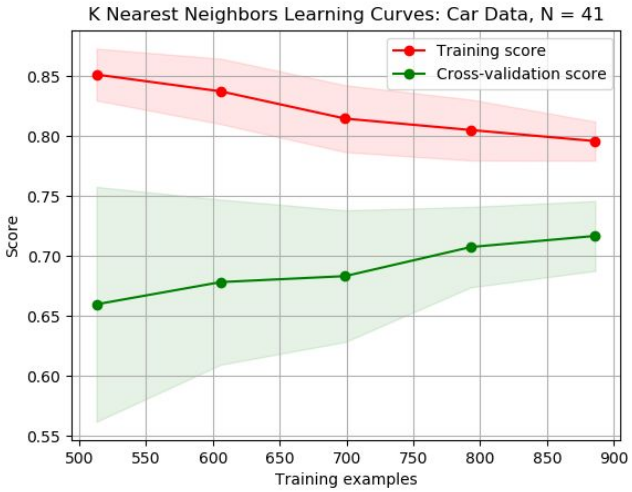

Figure 11. Learning curves for KNN on car dataset

```
Accuracy: 0.60550458
Confusion Matrix:
 [[ 13   0  42   0]
 [ 33   0  13   0]
 [  2   0 185   0]
_[ 35   0   4   0]]
```

Figure 12. Final test data for KNN on car dataset

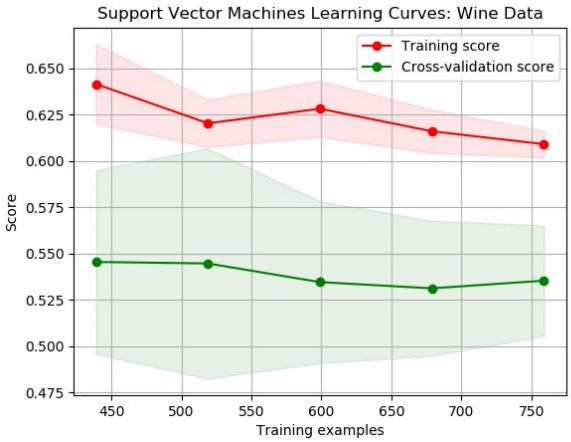

Figure 13. Learning curves for linear kernel SVM on wine dataset

```
Accuracy: 0.585427135678392
Confusion Matrix:
 [[  0   0   5   0   0   0]
 [  0   0  13   4   1   0]
 [  0   0 120  44   4   0]
 [  2   0  45 106  20   0]
 [  0   0   3  19   7   0]
 [  0   0   0   3   2   0]]
```

Figure 14. Final test data for linear kernel SVM on wine dataset

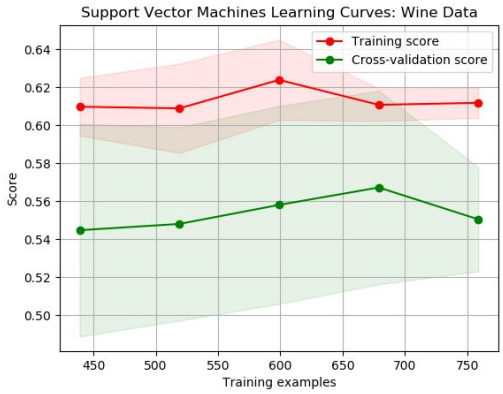

Figure 15. Learning curves for rbf kernel SVM on wine dataset

```
Accuracy: 0.6155778894472361
Confusion Matrix:
 [[  0   0   5   0   0   0]
 [  0   0  14   3   1   0]
 [  0   0 117  51   0   0]
 [  0   0  47 126   0   0]
 [  0   0   3  24   2   0]
 [  0   0   0   5   0   0]]
```

Figure 16. Final test data for rbf kernel SVM on wine dataset

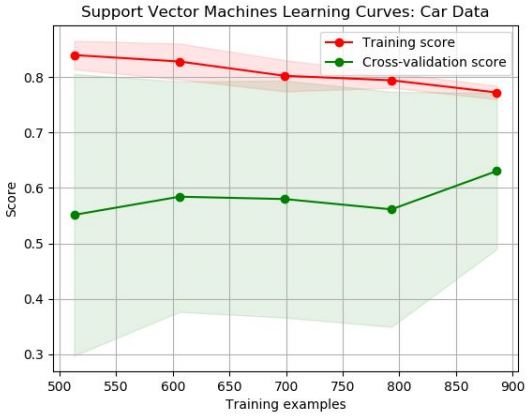

Figure 17. Learning curves for linear kernel SVM on car dataset

```
Accuracy: 0.57186544
Confusion Matrix:
 [[  0   0  55   0]
 [  0   0  46   0]
 [  0   0 187   0]
 [  0   0  39   0]]
```

Figure 18. Final test data for linear kernel SVM on car dataset

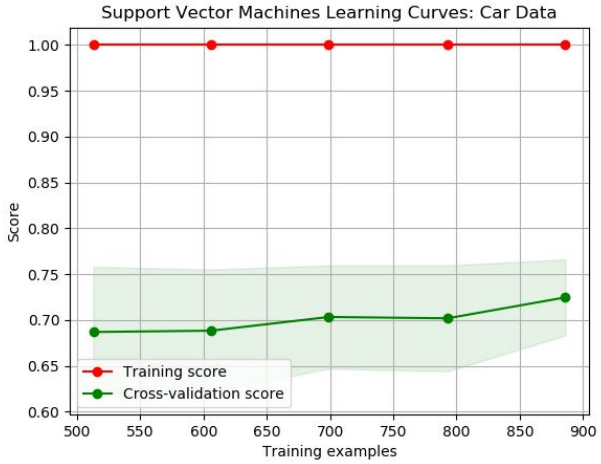

Figure 19. Learning curves for rbf kernel SVM on car dataset

```
Accuracy: 0.8501529
Confusion Matrix:
 [[ 42   0  13   0]
 [ 23  23   0   0]
 [  0   0 187   0]
 [ 13   0   0  26]]
```

Figure 20. Final test data for rbf kernel SVM on car dataset

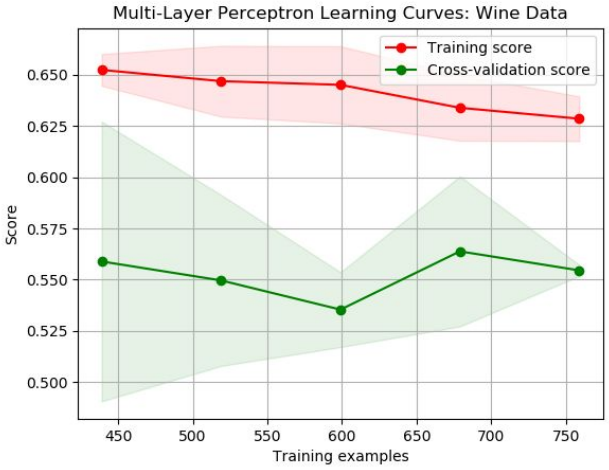

Figure 21. Learning curves for multilayer perceptron on wine dataset.

```
Accuracy: 0.5979899497487438
Confusion Matrix:
 [[  0   0   5   0   0   0]
 [  0   0  13   3   2   0]
 [  0   0 124  43   1   0]
 [  0   0  46 110  17   0]
 [  0   0   3  22   4   0]
 [  0   0   0   3   2   0]]
```

Figure 22. Final test data for multilayer perceptron on wine dataset

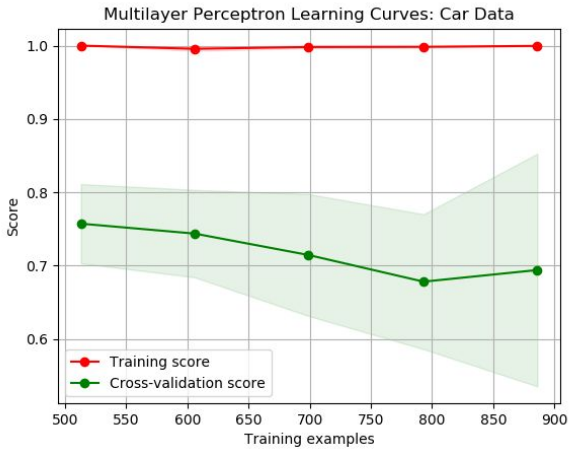

Figure 23. Learning curves for multilayer perceptron on car dataset.

```
Accuracy: 0.69724770
Confusion Matrix:
 [[ 43   0  12   0]
 [ 46   0   0   0]
 [  3   0 184   0]
 [ 38   0   0   1]]
```

Figure 24. Final test data for multilayer perceptron on car dataset.