CS 7641 – Machine Learning

Spring 2018

Submitted by- Yashovardhan Jallan (GT ID # 90320 6747)

1. Summary

This assignment applies different supervised learning algorithms to analyze two distinct datasets: Pima Indians Diabetes Data Set and Wine Quality Data Set, both taken from UCI Machine Learning Repository. In particular, Decision tree, K-nearest Neighbors, Boosting, Support vector machines and Neural networks are the learning algorithms applied. Performance for different models are also compared in the analysis. The whole analysis is done using Scikit-Learn Machine Learning library in Python 3.

2. Datasets Description and why are they interesting?

- **Pima Indians Diabetes Data Set**

Pima Indians are a group of Indigenous Americans living in Arizona that have a high incidence rate of diabetes. The Pima Indian diabetes dataset is a collection of medical diagnostic reports from 768 records of female patients at least 21 years old. There are eight attributes and one class variable with binary values 0 for tested positive or 1 for tested negative. The eight attributes are number of times pregnant, glucose concentration, diastolic blood pressure, triceps skin fold thickness, insulin levels, body-mass-index, diabetes pedigree function and age. There are more negative diagnosis (500) than positive diagnosis (268). The interesting aspect of this dataset is its real-world usefulness, converging medical diagnosis and forecast the onset of diabetes in high-risk population.

From machine learning perspective, this dataset is significant. I looked at its existing results and found that for most classifiers, it turns out this dataset is not easily classifiable and performance with most classifiers is generally low. I am greatly interested in the interconnections among the variables in the dataset as that will be a good test against all the learners I'll try. Although there are no "missing" values, some attributes have a zero value indicating that there might be some error in the dataset.

- **Wine Quality Data Set**

Wine industry shows a recent growth spurt as social drinking is on the rise. The price of wine depends on a rather abstract concept of wine appreciation by wine tasters, opinion among whom may have a high degree of variability. Pricing of wine depends on such a volatile factor to some extent. Another key factor in wine

certification and quality assessment is physicochemical tests which are laboratory-based and takes into account factors like acidity, pH level, presence of sugar and other chemical properties. For the wine market, it would be of interest if human quality of tasting can be related to the chemical properties of wine so that certification and quality assessment and assurance process is more controlled.

The UCI Machine Learning library has two sets of datasets for Wine Quality. One dataset is on red wine and have 1599 different varieties and the other is on white wine and have 4898 varieties. I have only analyzed the red wine data. All wines are produced in a particular area of Portugal. Data is collected on 12 different properties of the wines one of which is Quality, based on sensory data, and the rest are on chemical properties of the wines including density, acidity, alcohol content etc. All chemical properties of wines are continuous variables. Quality is an ordinal variable with possible ranking from 1 (worst) to 10 (best).

On preliminary analysis of the Red wine quality dataset, all the sample points were classified into 6 quality scores (3, 4, 5, 6, 7, 8). Out of 1599 records, there were only a small handful of records having quality score as either 3 or 8. This presented difficulty in running some of the algorithms as I received an error for procedures which required more data points. Hence I dropped the samples which were classified as either 3 or 8 and went ahead with 1571 samples for the rest of the quality scores.

- **Bank Marketing Data Set**

I had initially chosen the Bank Marketing Data Set from UCI Machine Learning Repository instead of Wine Quality Dataset. However, on running few tests with the various algorithms, I received almost perfect results of close to 90-100% accuracies with the vanilla (un-tuned) versions of the supervised algorithms required for this project. I felt that this dataset wasn't interesting enough as it probably had a simple dependency on a couple of attributes which made it a trivial problem. Hence, I chose not to go ahead with it.

3. Tools Used and Methodology

I have used Python 3 as my primary tool for this project. I have heavily utilized the Scikit Learn Machine Learning library. I have used Pandas library to organize the data into data frames for easy handling. I have used NumPy for basic functionalities and Matplotlib for making the required plots for this project.

I have chosen a ***70-30 train-test split*** for experiments with all the algorithms and both datasets. This has been done by using the train_test_split tool of SkLearn. This method quickly splits the entire dataset into random train and test subsets. For learning curve, I have divided the 70 portion of the training set into smaller parts by using the same tool and the test set remained same for all the different training set sizes.

Also, for each algorithm, at first, I have implemented the vanilla version of the Scikit Learn Classifier without changing/tuning any hyperparameters. I have used the following classification tools: DecisionTreeClassifier, Support Vector Classifier, AdaBoostClassifier, Multi-layer Perceptron Classifier and KNeighborsClassifier. I recorded the performance of the vanilla version of each algorithm.

Next, to check for improvements in the performance and accuracy, I used the GridSearchCV functionality in Scikit Learn. This allows the user to check for the best hyper-parameters of any algorithm which they are implementing. It also incorporates Cross-Validation in its hyper parameter tuning. For the purposes of this project, ***I have used a 10-Fold Cross Validation*** technique in determining the best fit model.

Having found the best fit model using the GridSearchCV technique and its internal cross-validation functionality, I then applied the model to my test data and recorded any improvements in accuracy of performance for each algorithm. I then created plots for the change in performance by varying some of the hyper-parameters. I have also created learning curves for the algorithms and noted down the time required to run the algorithms.

4. Decision Trees

For the decision tree, I've used the Gini impurity measure to assess the quality of any potential split, i.e. the feature and threshold which gives the best Gini impurity score is selected for the split. Although SkLearn does not implement pruning, there are several ways for "pruning like" procedures. Specifically, I restricted the 'maximum depth' and 'maximum leaf nodes' of the decision tree, and thus prevent the decision tree from growing too large and possibly overfitting based on unimportant features.

- **Diabetes dataset**

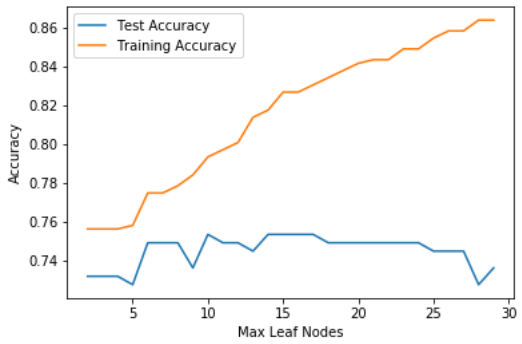

Fig 1.

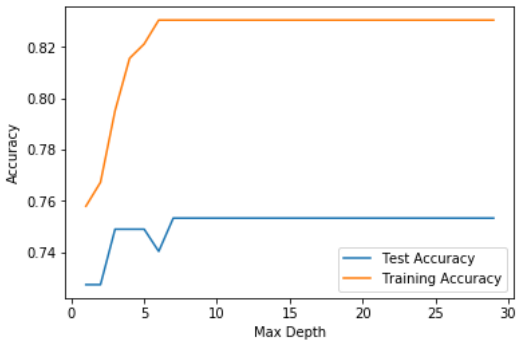

Fig 2.

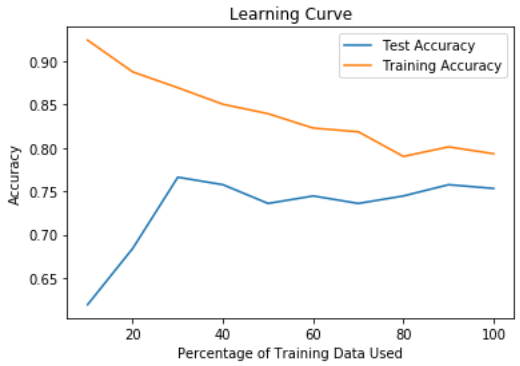

Fig 3.

Fig 1. and Fig 2. show us the performance of the Decision tree classifier for the Diabetes dataset with varying hyperparameters. We can see that the test/train accuracies do not change for max_depth > (7-8). The dataset itself has 8 attributes so this is expected. Also once the max leaf nodes goes past roughly ~10, the train accuracy increases without similar change in test accuracy.

This is an example of overfitting. Looking at Fig. 3, which is the Learning Curve, we see that as the percentage of training data increases, the performance of the algorithm improves and overfitting reduces. Based on 10-fold cross validation, the best set of hyper-parameters are a max_depth of 6 and a max_leaf_nodes of 17.

- **Wine Quality dataset**

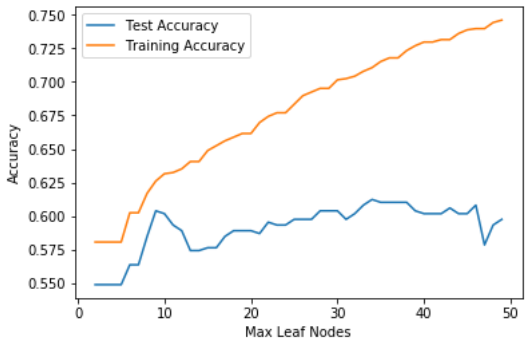

Fig 4.

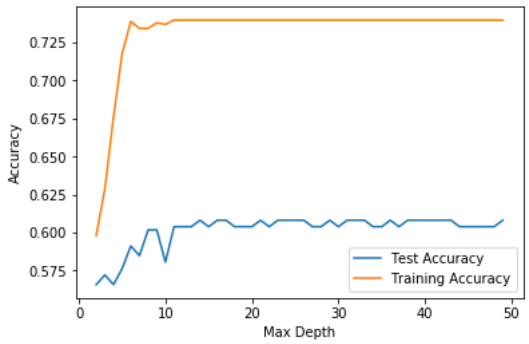

Fig 5.

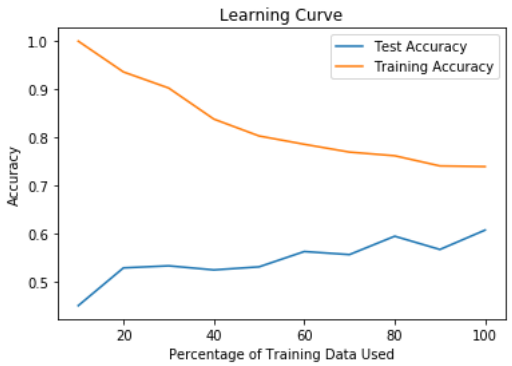

Fig 6.

Fig 4. and Fig 5. show us the performance of the Decision tree classifier for the Wine Quality dataset with varying hyperparameters. The max leaf node curve and the max depth curve have similar behavior as the Diabetes dataset. After a max depth of ~8-9, the performance is constant. This is because the dataset has a total of 11 attributes. Similarly, for varying max leaf node, the algorithm starts overfitting after a max leaf node of ~10. Looking at Fig. 6, which is the Learning Curve, we see that as the percentage of training data increases, the performance of the algorithm improves and

overfitting reduces. Based on 10-fold cross validation, the best set of hyper-parameters are a max_depth of 19 and a max_leaf_nodes of 46.

5. Support Vector Machines

SVM is very good with high dimensional data and it can work well with small datasets. It is an eager learner. One of the challenges of using SVMs is that picking best parameter can be computationally intensive.

The C Parameter in SVM helps us in determining how much we want to penalize misclassified points. It tells the SVM optimization how much we want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

The Kernel type can be thought of as adding an extra dimension to the data. So if the data cannot be linearly separated, we can experiment with other kernel type to find best fit model.

- **Diabetes dataset**

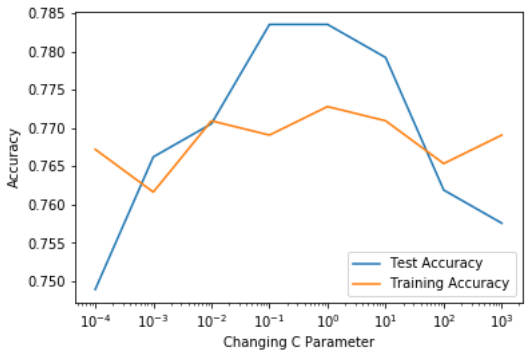

Fig 7.

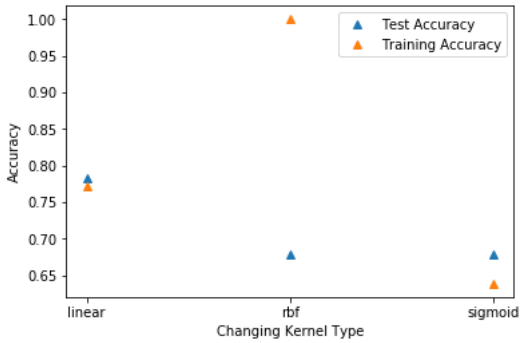

Fig 8.

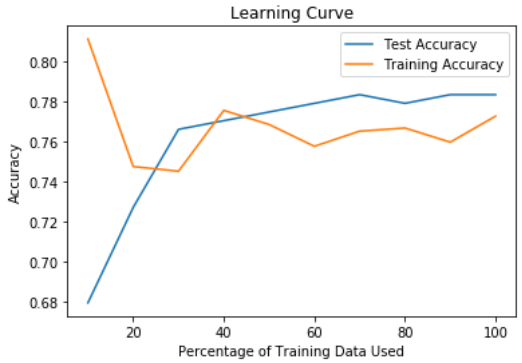

Fig 9.

Fig 7. shows the performance of the Diabetes dataset with changing C parameter. Fig. 8. shows the training and test accuracies for different kernel types and finally Fig 9. represents the learning curve. Based on 10-fold cross validation, the best set of hyper-parameters are C equal to 0.1 and kernel type as Linear. An important observation was that as the C parameter increases, the observed computation time significantly increases. Especially for C values greater than 10 raised to second power.

- **Wine Quality dataset**

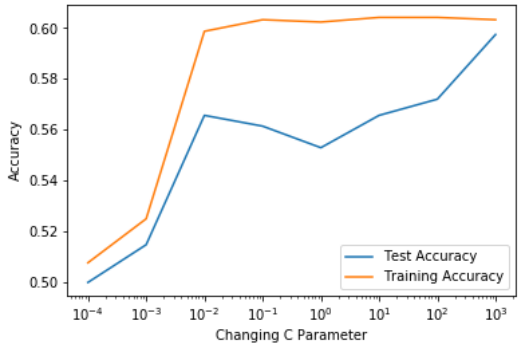

Fig 10.

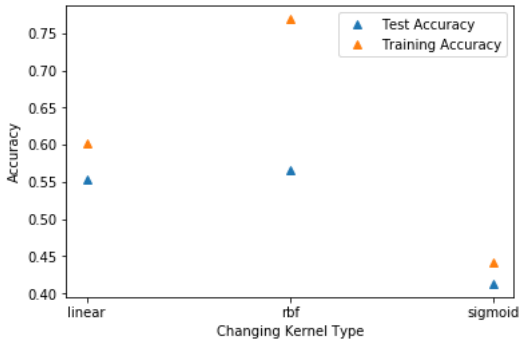

Fig 11.

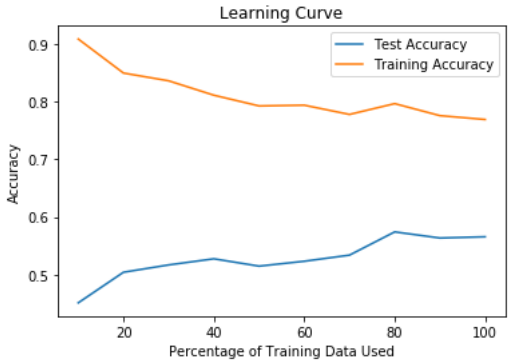

Fig 12.

Fig 10. shows the performance of the Diabetes dataset with changing C parameter. It is observed that as the C parameter is increased, the performance of both train and test accuracies were improved. This means that a smaller-margin hyperplane works better for this dataset. Fig. 11. shows the training and test accuracies for different kernel types and finally Fig 12. represents the SVM learning curve. Based on 10-fold cross validation, the best set of hyper-parameters are C equal to 1000 and kernel type as Linear.

6. Boosting

Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers. AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem.

SkLearn's AdaBoostClassifier use DecisionTreeClassifier as its base estimator by default. The parameters available for tweaking are maximum number of estimators at which boosting is terminated, Learning rate, which shrinks the contribution of each classifier; and a choice between two algorithm 'SAMME' and 'SAMME.R'.

- **Diabetes dataset**

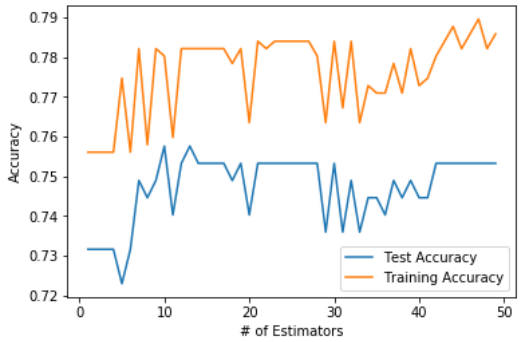

Fig 13.

Fig 14.

Fig 13. And Fig 14. Show the performance of the Diabetes dataset with changing hyper-parameters. We can see that varying the maximum number of estimators and learning rate produce varying accuracies. From the graphs, it is seen that the value of the number of estimators should be greater than ~8-9 and the learning rate be greater than ~0.15-0.2. Other than this, it is difficult to interpret a significant pattern.

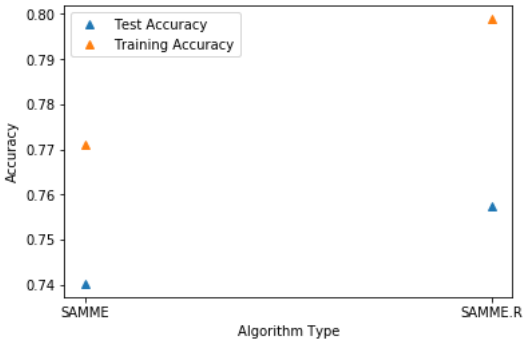

Fig 15.

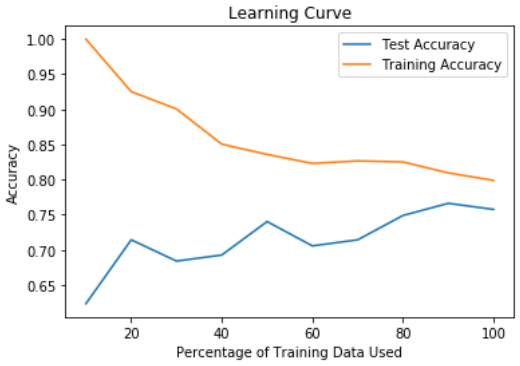

Fig 16.

Fig 15. Shows the performance of the training and test datasets for the two types of algorithm available to choose from. And Fig. 16 shows us the learning rate. Based on 10-fold cross validation, the best set of hyper-parameters are algorithm = SAMME.R, learning_rate =0.55, and n_estimators = 36.

- **Wine Quality dataset**

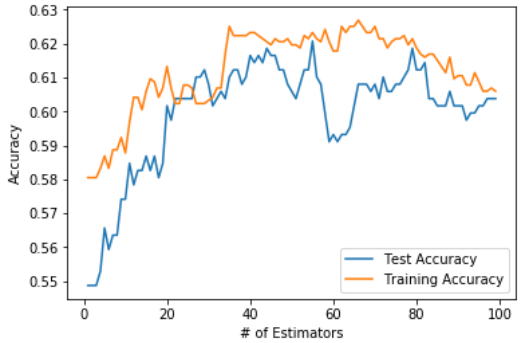

Fig 17.

Fig 18.

Fig 17. And Fig 18. show the performance of the Wine Quality dataset with changing hyper-parameters. We can see that varying the maximum number of estimators and learning rate produce varying accuracies. From the graphs, it is seen as the number of estimators increase, the accuracy increases up until n=50 and then starts decreasing. The learning rate also sees a similar trend with improvement in performance till values of about 0.3-0.4, and then decline.

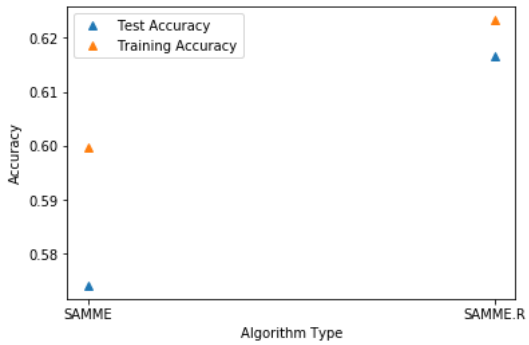

Fig 19.

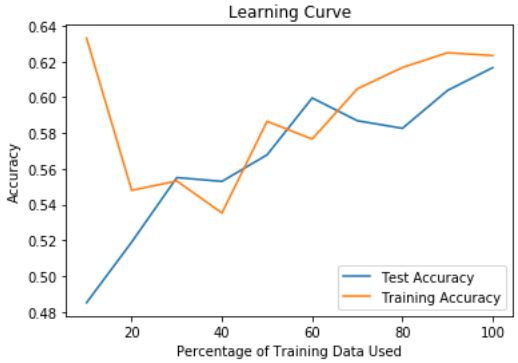

Fig 20.

Fig 19. shows the performance of the training and test datasets for the two types of algorithm available to choose from. And Fig. 20 shows us the learning rate. Based on 10-fold cross validation, the best set of hyper-parameters are algorithm = SAMME.R, learning_rate =0.24, and n_estimators = 39.

7. Neural Networks

Multilayer Perceptron classifier in SkLearn is used to evaluate the datasets. A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation. It can distinguish data that is not linearly separable.

The hyper-parameters that I have tuned are size of the hidden layer, type of solver and the type of learning rate. A simplistic model for choosing the number of hidden layers in MLP is given as:

$$perceptron\_units = (attributes + classes) / 2$$

- **Diabetes dataset**

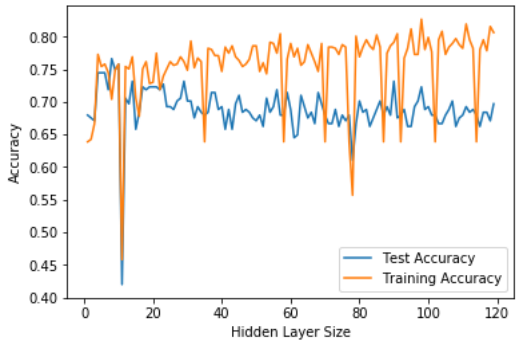

Fig 21.

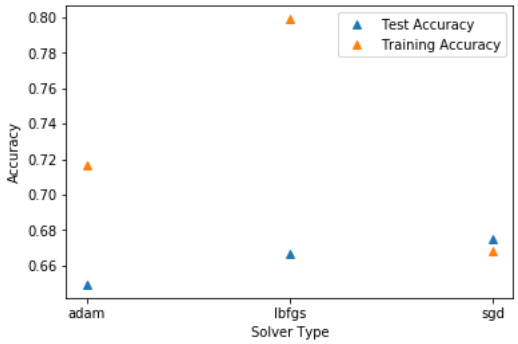

Fig 22.

Fig 21. And Fig 22. show the performance of the Diabetes dataset with changing hyper-parameters. For the diabetes dataset, the perceptron units by the above formula is found to be 5. We can see from Fig 21. That after a hidden layer size of ~5-6, the model starts overfitting. Looking at the solver type, we see that the 'lbfgs' solver performs the best.

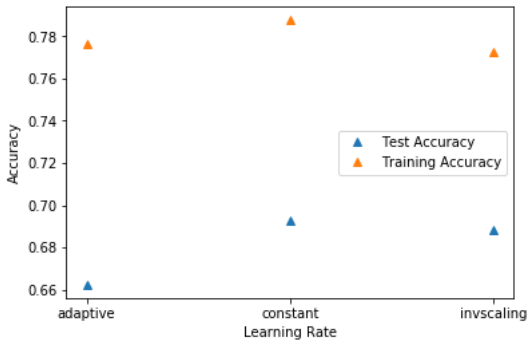

Fig 23.

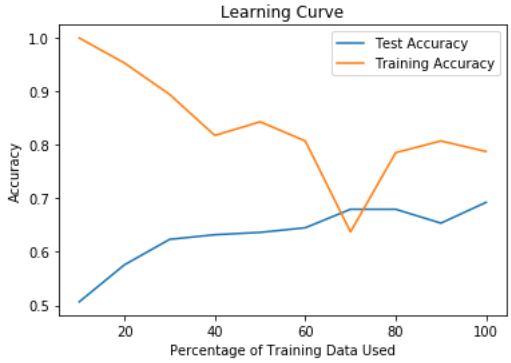

Fig 24.

Fig 23. shows the performance with different learning rate types. And Fig. 24 shows the learning curve for the model. Based on 10-fold cross validation, the best set of hyper-parameters are 'hidden_layer_sizes': 11, 'learning_rate': 'constant', 'solver': 'lbfgs'.

- **Wine Quality dataset**

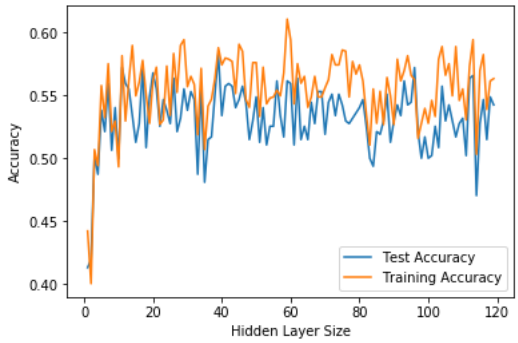

Fig 25.

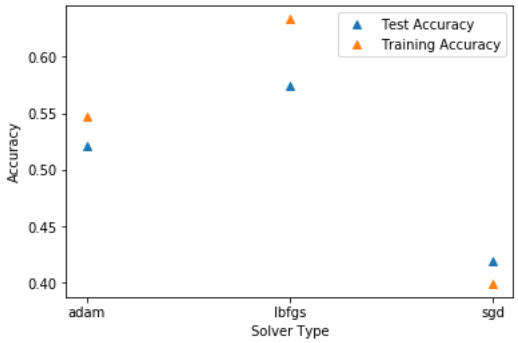

Fig 26.

Fig 25. And Fig 26. show the performance of the Wine quality dataset with changing hyper-parameters. For the wine quality dataset, the perceptron units by the given formula is found to be 8. We can see from Fig 25. that after a hidden layer size of ~10-11, the model shows no major change in performance. Looking at the solver type, we see that the 'lbfgs' solver performs the best once again.

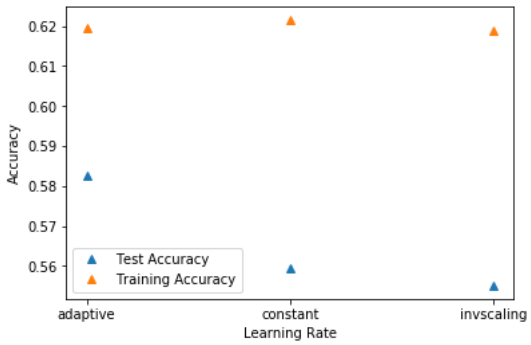

Fig 27.

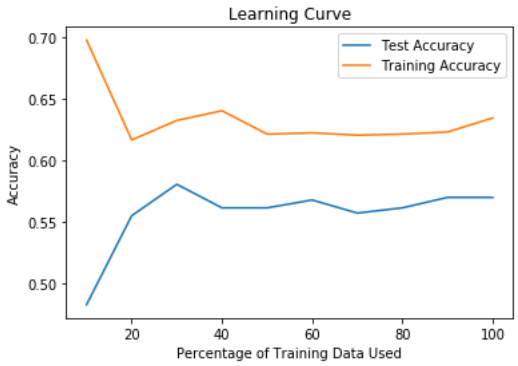

Fig 28.

Fig 27. shows the performance with different learning rate types. And Fig. 28 shows the learning curve for the model. Based on 10-fold cross validation, the best set of hyper-parameters are 'hidden_layer_sizes': 77, 'learning_rate': 'adaptive', 'solver': 'lbfgs'.

8. kNN

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

- **Diabetes dataset**

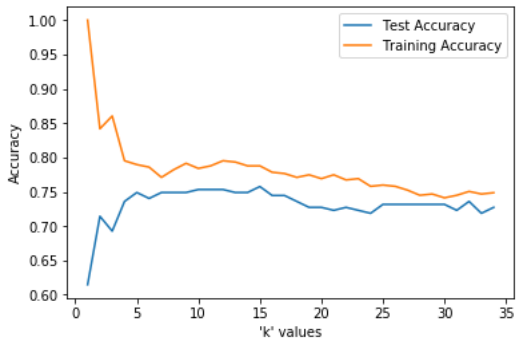

Fig 29.

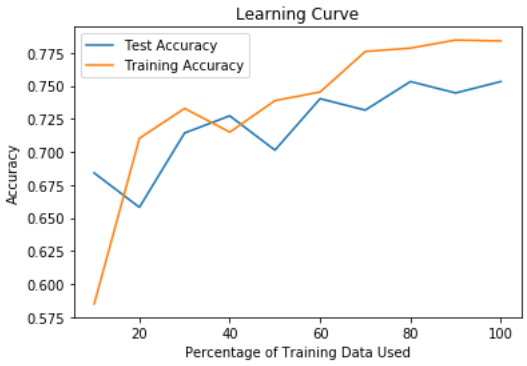

Fig 30.

Fig 29. shows the performance of the Diabetes dataset with changing hyper-parameter ‘k’. For the diabetes dataset, ‘k’ values greater than 5 perform almost the same. Fig. 30 represents the learning curve associated with kNN. Based on 10-fold cross validation, the best performance was found at k=11.

- **Wine Quality dataset**

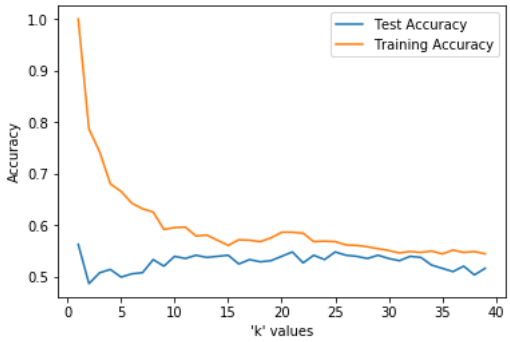

Fig 31.

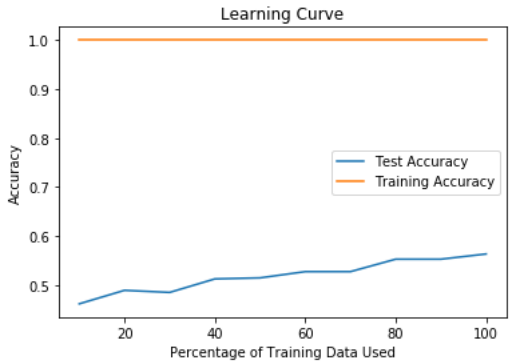

Fig 32.

Fig 31. shows the performance of the Wine Quality dataset with changing hyper-parameter 'k'. Fig. 32 represents the learning curve associated with kNN. Based on 10-fold cross validation, the best performance was found at k=1.

9. Summary and Conclusions

- **Diabetes dataset**

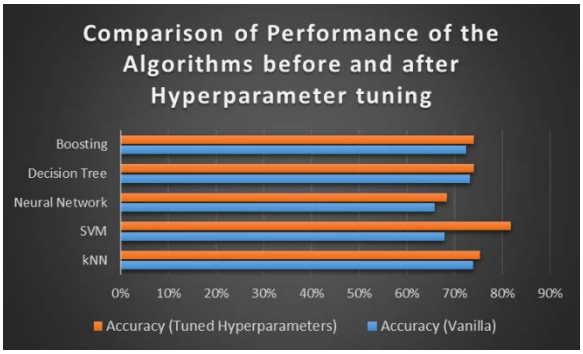

Fig 33.

Fig 34.

Fig 33. Shows us the comparison of performance of different algorithms before and after the hyper-parameter tuning. We see that the SVM algorithm performs the best among all and a significant improvement is seen over the vanilla version of SVM provided by the SkLearn library.

Coming to Fig. 34, we compare the running times of the different algorithms for the Diabetes dataset. It should be noted that these run times are for running the algorithm with the tuned set of hyper-parameters. To find the best hyper-parameters, I had to run the GridSearchCV method which was way more computationally time consuming. We see that Neural networks, Boosting and SVM have relatively higher computational times.

- **Wine Quality dataset**

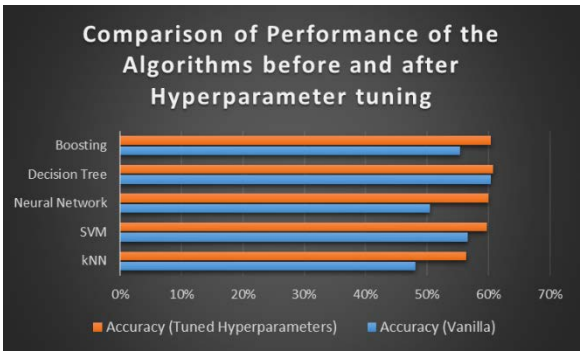

Fig 35.

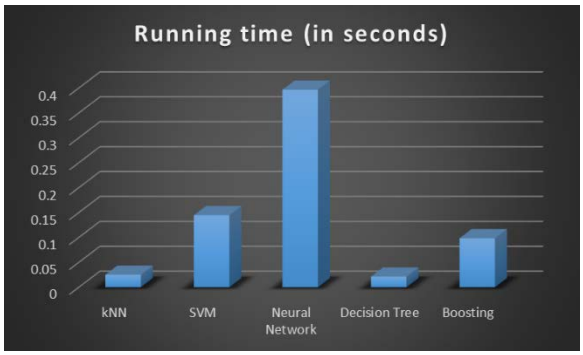

Fig 36.

Fig 35. shows us the comparison of performance of different algorithms before and after the hyper-parameter tuning. In this case of the wine-quality dataset, almost all the algorithms perform equally. Considering that my output class variable consisted of 4 classes, I think that an accuracy of close to 60% is definitely statistically significant over a random chance of 25%. However, I even after several tries, I could not achieve accuracies of anything greater than 62%. I am curious to know if we will use certain methods later in this course which can help me improve the performance for this dataset.

Coming to Fig. 36, we compare the running times of the different algorithms for the Wine quality dataset. It should be noted that these run times are for running the algorithm with the tuned set of hyper-parameters. To find the best hyper-parameters, I had to run the GridSearchCV method which was way more computationally time consuming. We see that Neural networks, SVM and Boosting have relatively higher computational times.

10. References

- PIMA Indians Diabetes Data Set - https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes
- Wine Quality Dataset - https://archive.ics.uci.edu/ml/datasets/wine+quality
- Scikit Learn Machine Learning Library - http://scikit-learn.org/stable/
- Pandas Library – https://pandas.pydata.org/
- NumPy Library – http://www.numpy.org/
- Matplotlib Library - https://matplotlib.org/