James Corwell

Supervised Learning Algorithms Typified Through Quintessential Datasets

Introduction

In this paper, we work with a dataset depicting the qualities of an Iris flower, and a dataset that typifies the qualities of different wine samples produced by three different producers. I figured it wise to pick datasets that are commonly used to demonstrate Supervised Learning algorithms, because third party literature is widely available for reference.

The Wine dataset is culled from three wineries in the same region of Italy. 13 variables including alcohol content and hue were considered. I approached this analysis using two attributes that do not classically define wine to produce novel conclusions; specifically, 0D280/OD315 of diluted wines and Proline concentrations. I chose these metrics as they are something that producers wouldn't attempt generally manipulate to make their product more appealing for consumer purchase , unlike such variables as the alcohol content, hue, malic acid (sour), or polyphenol/flavonoid content (proposed health benefits). The iris dataset is comprised of less variables. It contains measurements of length and width of different Iris flowers, which we then can use to classify the flowers into three distinct classes.

Mission Objectives

Our objective is to accurately match each wine with its winery by its attributes alone. Additionally, we expect to generate a database by sorting the wines according to the respective winery. Accurate classification may be implemented and accelerated with machine learning. Consider that the global wine market will reach upwards of a 400 Billion market cap by 2022—wine sommeliers could take advantage of a 'wine sorting service' to match wines with customers using metadata about their past purchases. The future of wine sales could be automated the metadata would serve as feedback for the algorithms.

The goal of my analysis to see how we can accurately determine which winery the wine came from based off of the wine's attributes. Our goal is to clearly classify each of the wines into a different class, representing a different producer. If you're able to demonstrate through machine Learning that an algorithm can determine which kinds of wines were grown by which grower, and a larger scale of this project could be used in various ways to influence the global wine market which is expected to reach a 400 billion USD market value by 2022.

The Iris dataset is a collection of measuring's of Iris flowers' sepals and petals. With this information, we attempt to classify each sample as either Setosa, Versicolour, or Virginica. This dataset has been used for classification tasks ever since Ronald Fisher first described it in 1936. It is in a significant amount of literature relating supervised learning and therefore I figured it

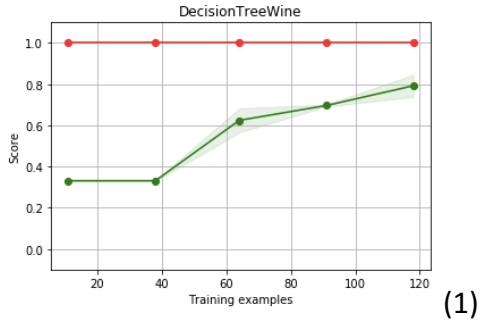
would behoove anyone studying Machine Learning to not to know every nuance of this dataset under different learning algorithms.

Methods

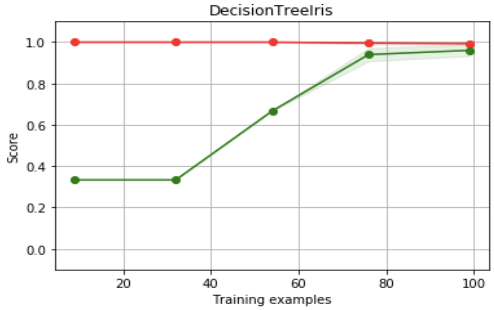
My code is wholly developed in python, with particular dependency on sci-kit learn. My inspiration on how to approach the analysis came initially from "Python Machine Learning" by Dr. Sebastian Raschka. As per the class assignment I will demonstrate the ability and usability of several Supervised Learning algorithms. Particularly Decision Trees, MLPerceptron networks, AdaBoosting, Support Vector Machines, and K-Nearest Neighbors. Unless otherwise noted, all KFold cross-validation runs were done with 3 folds. For several graphs, the data was standardized in order to determine relationships and trends better.

Decision Trees

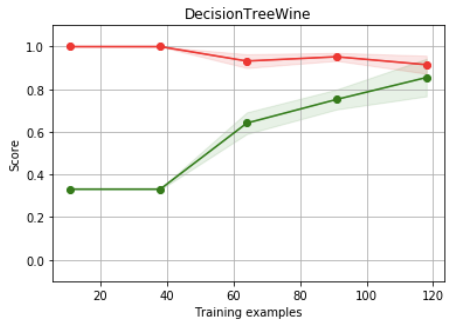
The decision trees were generated under two different circumstances, with an unrestricted amount of nodes, as well as a 'pruned' tree. By comparing the two results for each dataset, you can see how a max_depth=None leads to overfitting of the of the data for wine, but still generalizes well for the Iris dataset. The iris dataset at max_depth = None still shows slightly higher bias with the training curve not moving from 1 despite the training size, and could benefit from having more data.



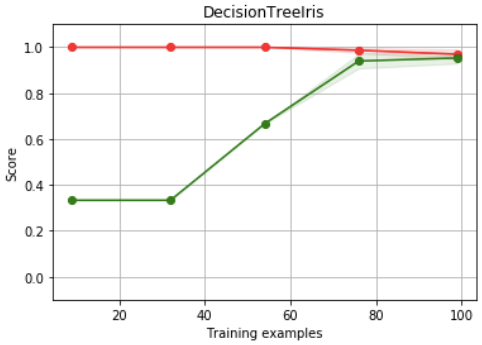
(1)



(3)



(2)

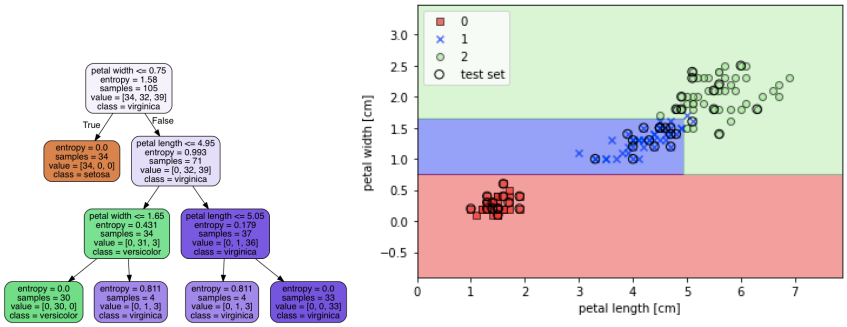


(4)

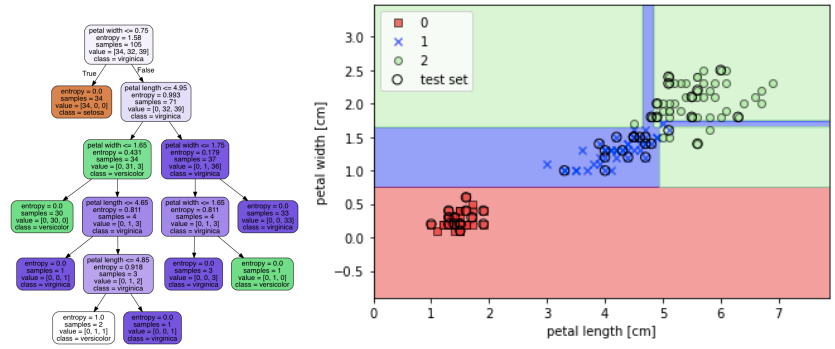
(1) learning curve wine max_depth = None (2) learning curve wine max_depth = 2.
(3) learning curve iris max_depth = None (4) learning curve iris max_depth = 3

The following classification graphs are to better show how a non-pruned decision tree caused overfitting of the data into overly specific classification regions.

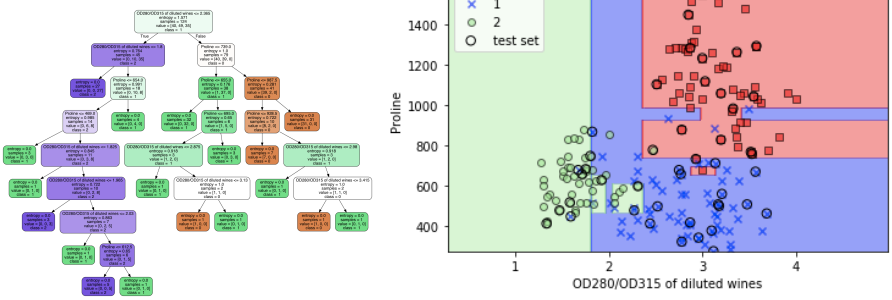
Iris decision tree: max_depth = 3



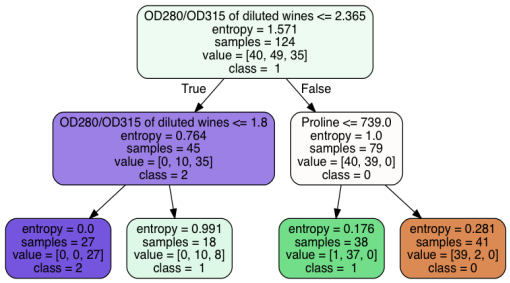
Iris DT max_depth = None:

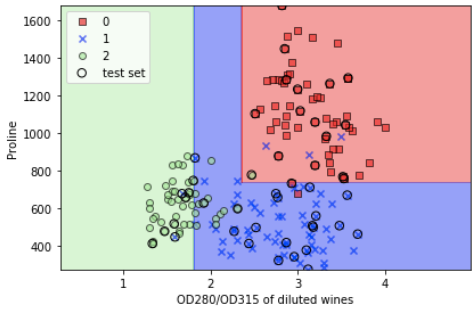


Wine max_depth = None:



Wine max_depth = 2



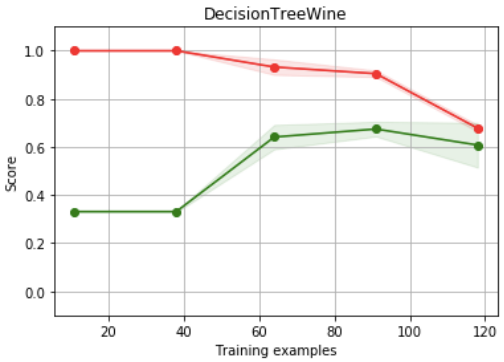


Clearly, the results show that a pruned decision tree prevents overfitting, and creates more intuitive classification regions for both the Iris and Wine dataset.

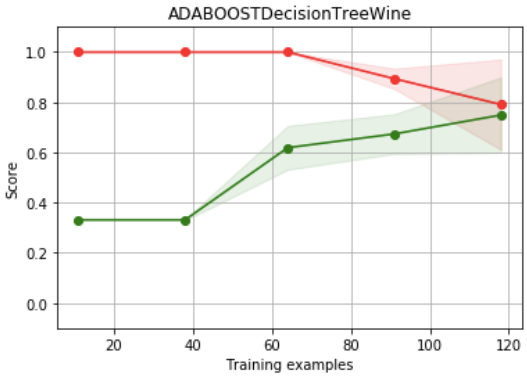
Boosting

I decided to implement AdaBoost on my decision trees that were pruned greatly to generate a decent classification result. AdaBoost is an excellent boosting method, that requires little tweaking of parameters to generate good results. For this experiment, all trials were conducted with n_estimators=500 and learning_rate=.1.

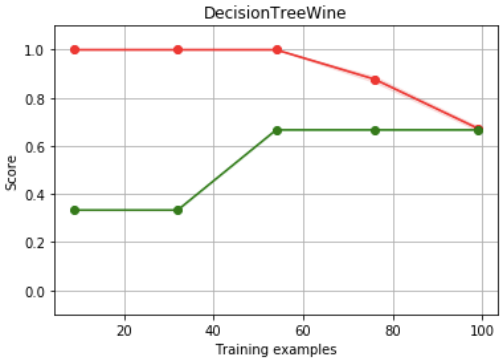
The following are decision trees for my two datasets, pruned to max_depth = 1, and showing symptoms of high bias (underfitting) alongside their AdaBoosted counterparts:

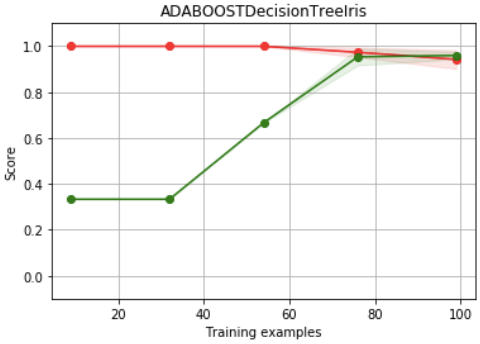


DT Train/test accuracies 0.605/0.574



AdaBoost accuracies Wine: 0.919/0.815





DT Train/test accuracies Iris: .695/.600 Adaboost accuracies Iris: .962/.889

This clearly shows that Adaboost was able to correct for the underfitting caused by the overpruning of the trees by a significant margin, and increase the overall accuracy of the classification, without overt signs of high bias. When testing Adabost on unpruned versions of the trees, the gains to accuracy were minimal, and the results still suffered from the similar high variance.

Support Vector Machines

According to scikit-learn documentation, The polynomial and RBF kernels for SVM are the kernels of particular use when dealing with nonlinearly separable data. I implemented a Grid Search to tune the parameters for the SVM, which produced results as such:

Grid scores on development WINE:

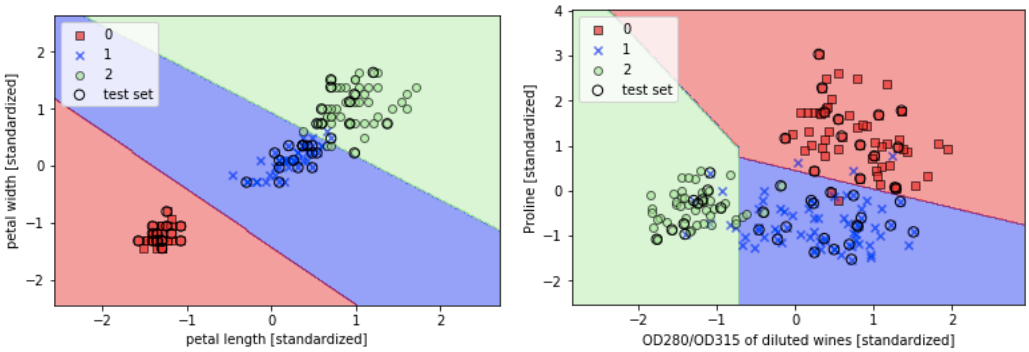
0.777 (+/-0.159) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.741 (+/-0.222) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.734 (+/-0.184) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.762 (+/-0.170) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.749 (+/-0.098) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.745 (+/-0.211) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.780 (+/-0.141) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.857 (+/-0.139) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.938 (+/-0.131) for {'kernel': 'linear', 'C': 1}
0.935 (+/-0.114) for {'kernel': 'linear', 'C': 10}
0.928 (+/-0.114) for {'kernel': 'linear', 'C': 100}
0.915 (+/-0.164) for {'kernel': 'linear', 'C': 1000}

Grid scores on development set IRIS:

0.399 (+/-0.147) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.333 (+/-0.000) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.851 (+/-0.195) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.399 (+/-0.147) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.983 (+/-0.067) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.851 (+/-0.195) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.951 (+/-0.134) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.983 (+/-0.067) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.968 (+/-0.081) for {'kernel': 'linear', 'C': 1}
0.951 (+/-0.134) for {'kernel': 'linear', 'C': 10}
0.954 (+/-0.122) for {'kernel': 'linear', 'C': 100}

0.954 (+/-0.122) for {'kernel': 'linear', 'C': 1000}

For the Iris dataset, ideal parameters were found to be kernel = ‘rbf’, C = 100, and gamma = 0.001. This generated a precision, recall, and f1 score of .95, an excellent result. Using grid-search to tune parameters for the Wine dataset we find kernel = linear, C = 1. These parameters produced the following classification diagrams:

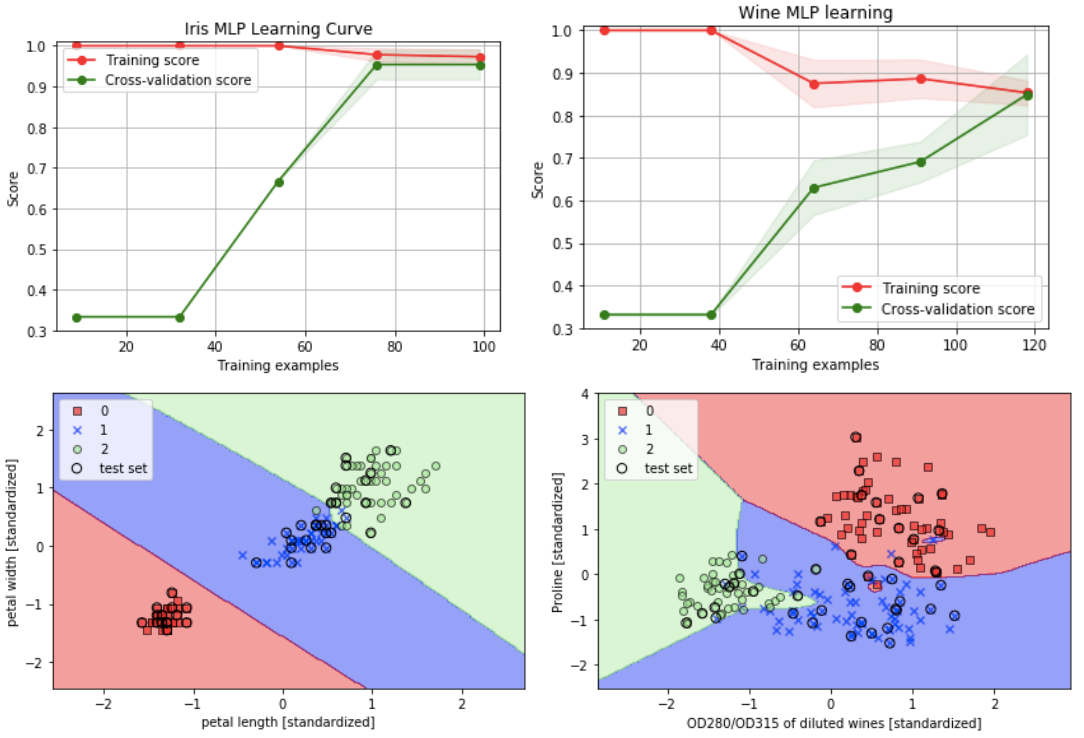


Classification accuracy IRIS: 0.96000 WINE: 0.91011

This data clearly shows that ‘linear’ outperformed ‘rbf’ for the Wine dataset, but ‘rbf’ outperformed ‘linear’ for the Iris dataset. This is perhaps counterintuitive, because the wine dataset hasn’t exhibited significant linear separability in previous tests. While the classification accuracies aren’t the highest for wine, the classification regions are very simple and would generalize extraordinarily well to new data.

Neural Networks

I modeled aa Multilayer Perceptron model of Neural network to classify the data. All trials were run with a constant learning rate, and a L2 penalty of .01. A basic accuracy test was conducted of a few different parameters to determine a reasonable L2 which ended up working well for both of the datasets. In determining the number of neurons and layers, I decided to keep the model simple and only implement a single hidden layer. Initially starting with 100 neurons, the Iris dataset converged well, while still showing a high, nearly perfect training score. This suggests a benefit to introducing more data. The wine curve however did not perform nearly as well with only 100 neurons, and got significantly better results with 1000. As there is no hard rule for how many layers and neurons to use, I found this general testing method to produce reasonable results.

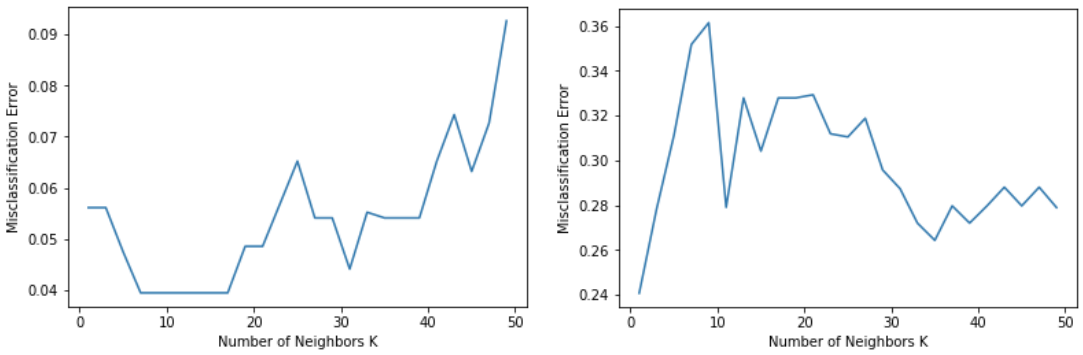


figures depiciting MLP learning curves for Iris and Wine datasets with 100 neurons and 1000 neurons each respectively, and their corresponding classification diagrams

K-Nearest Neighbors

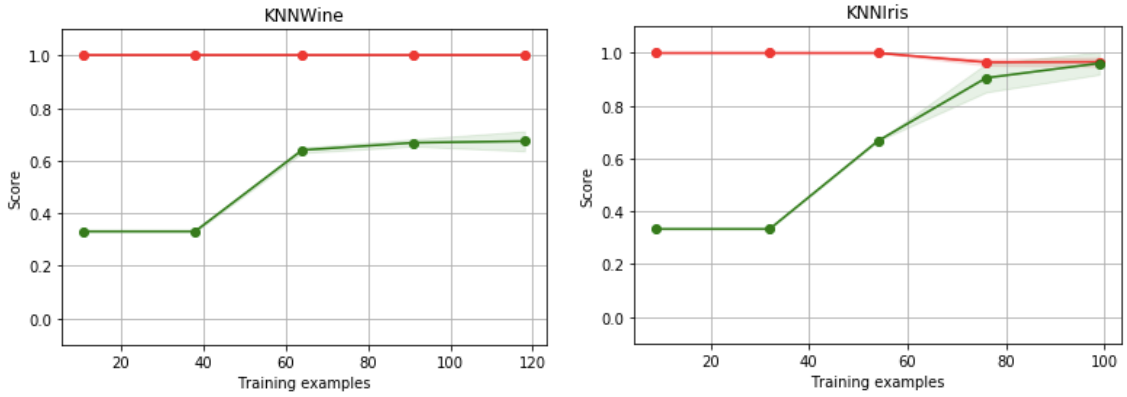
The KNN algorithm operates by studying the 'nearest neighbors' of points around the point in question. It then classifies this data as whatever kind of points are closest to it. Closeness of course being defined by convention of using a distance function. For my analysis I chose to use the minkowski distance metric with p=2, which is equivalent to a Euclidean distance metric.

In order to determine a good number for k, instead of plugging in random numbers I ran a K-fold cross-validation algorithm. Running this for the Iris and Wine datasets generate the following plots:

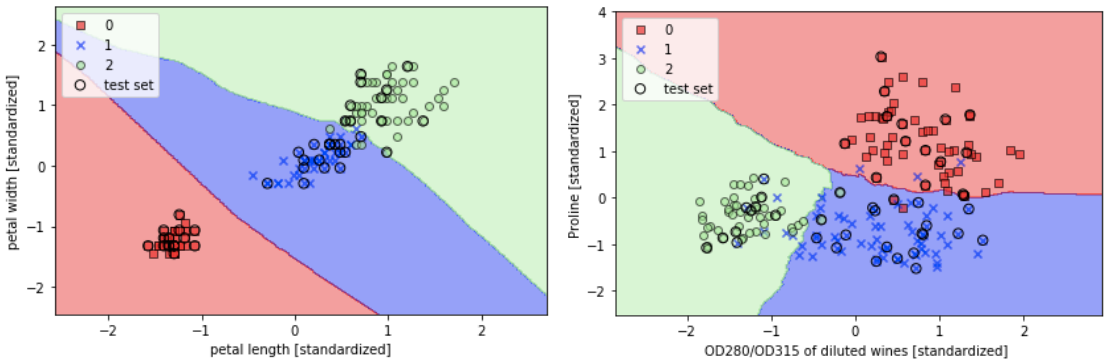


Ideal value K for iris (left) and wine (right).

The iris dataset has a clear ideal K of 7. Wine however shows a preferred K of 1. I found this to be rather low, and causing some serious overfitting, as shown:



Of course, using K-Nearest neighbors with only 1 neighbor is nonsensical. Therefore, I determined a better amount of nearest neighbors for the Wine dataset to be 35. With a value of K = 7 for the Iris data, and K=35 for the wine data, you can clearly see the following classifications of each dataset:



Classification of Iris data(left) and Wine data(right)

Iris(k=7):

Total accuracy: 0.96000
Class0 accuracy: 1.00000
Class1 accuracy: 0.96000
Class2 accuracy: 0.92000
Geometric mean accuracy: 0.95944

Wine(k=35):

Total accuracy: .92315
Class0 accuracy: .96610
Class1 accuracy: .85915
Class2 accuracy: .95833
Geometric mean accuracy: .92655

K-Nearest-Neighbors has thus shown itself to accurately classify the data into regions without significant bias or variance. The training score always being high in the iris dataset does however suggest that this algorithm could benefit from having more data.

Conclusion

A big trend I noticed in the learning curves of my data, is that the training set was usually very close to 1. This shows great accuracy, however perhaps a sign of overfitting caused by not enough data. As these datasets are rather small in the grand scheme of learning algorithms, it suggests more data for improved classification models.

That being said, most of the methods did a good job at classification of the data it seems. A notable exception being the MLP neural network did a poor job on the wine dataset. Possibly the dataset required better parameters, I merely followed the rough customs one might use when defining a neural network around how many layers compared to the amount of variables. AdaBoost did a beautiful job at fixing the mess caused by an overpruned decision tree, while requiring very little tuning to achieve great results. K-nearest neighbors performed well while being the simplest algorithm to implement, and it produced very respectable accuracies without horrible symptoms of overfitting. SVM with grid search parameter tuning by far performed the best on both of my datasets – presumably due to the implantation of parameter tuning.

In terms of speed of the algorithms, I will briefly note that all of the algorithms performance times were small due to the small (<1000) amount of data. When using the MLP neural network, I only used one hidden layer, preventing too long of computation time. The only time computation became notable was when running two layers of 1000 hidden neurons.

Cross validation certainly would help for the MLP, it was not implemented because I couldn't find a streamlined method for parameter tuning, so I was stuck just eyeballing for specific parameters that seemed like they might work decently for each dataset. It would be interesting to attempt other methods for pruning a decision tree, other than simply limiting the maximum_depth of the tree.

The performance of my algorithms was significantly influenced by the data I chose. The IRIS dataset has been analyzed greatly, and thus I was able to quickly refer to other texts / research papers when results were suspect. The wine dataset is also a clean dataset, while still producing

interesting features, and performing ultimately far differently than the Iris dataset. Overall, this project could have been much harder if using hundreds of thousands of data on stock market predictions.

Defining best algorithm performance as accurate classification without overfitting the data, it can be seen that SVM with Grid Search tuning for the parameters was the most accurate of any algorithm for both of the datasets. The classification regions are also fairly intuitive, signaling that the algorithm did an excellent job classifying the data without significant bias or variance.