Introduction
After searching for a while for a reasonably-sized non-linear interesting dataset to use I stumbled upon a wine quality dataset. As a fan of wines and in particular white wines I thought it would be funny to have a AI that is a white wine snob. The dataset was split into two different subsets of white wine and red wine. The white wine set was what I was more interested in and it had plenty of data (4000+). The data was also all numerical so I did not have to do much preprocessing other than removing the header. The data was broken up in the following way:

Input variables (based on physicochemical tests):
1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
Output variable (based on sensory data):
12 - quality (score between 0 and 10)

To measure performance, I will be using how accurately each algorithm can correctly score each wine. This will be done using the score method provided by ski-learn on each of the classifiers.

Description
In order to show the dataset is not linearly separable, I used a Support Vector Machine with a linear kernel. This will try and fit linear hyperplanes to separate the data. The hyperparameters that are used for all the SVM Kernels are the C-regularization value, and the maximum number of iterations. The non-linear kernels also trained on the dataset were a polynomial kernel and a radial basis function kernel. These kernels transform the space in a way to try and make the data more separable. For these we have a gamma value that is also used and for the polynomial kernel we have the degree of the polynomial. AdaBoost combines a bunch of weak-learners, in this case Decision Trees, to form a stronger learner. The hyperparameters that are mainly tweaked for AdaBoost are the number of weak learners and in this case the maximum depth of the decision trees used for the weak learners. Random Forest also uses Decision trees but in a slightly different way. Random Forest uses bagging to train deeper decision trees on random samples of the training data and then averages those trees in a majority vote to classify. The two main hyperparameters I focused on tweaking was the number

of trees and the minimum number of samples needed to split a node on the trees. Lastly, is the Neural Net. A Neural Net takes the training data and feeds it through multiple layers of perceptrons applying an activation function to it as it goes to arrive at an output and then uses back propagation to update the layers and repeat the process. The hyper parameters that I mainly used were the learning rate and the structure of the Neural Net itself.

Hyperparameters
The main consideration I had when setting up the cross-validation for each algorithm was the size of the search space. For some of the algorithms the amount of hyperparameters and the ranges were not that bad in the amount of combinations, however for instance the Neural Net can have many different combinations in just the sizes of the hidden layers not to mention also having an alpha parameter to also tweak. When you also consider the run-time complexity of some of these it became quite a balancing act. For this reason, the Neural Net was limited to three hidden layers with the same amount of nodes as input features. I also limited most of the algorithms to a maximum of 1000 iterations to keep computation time to a reasonable time. All cross-validation was also done with 3-fold cross-validation.

For cross validation for Random Forest the parameters search space was {'n_estimators':[10,100,1000],'min_samples_split':[2,3,4]}. Running the best estimator on the test data gave a score of 0.6846938775510204.

Params: {min_samples_split, n_estimators}	**Mean Score**	**Score Standard Deviation**
{2, 1000}	0.65364982	0.00095499
{3, 1000}	0.6510975	0.00072191
{4, 1000}	0.6510975	0.00477496
{2, 100}	0.64420623	0.00072191
{3, 100}	0.6431853	0.00472007
{4, 100}	0.64037774	0.00477496
{3, 10}	0.61817254	0.00969211
{4, 10}	0.61332312	0.00760575
{2, 10}	0.60158244	0.00851114

For cross validation for AdaBoost the parameters search space was {'n_estimators':[10,100,1000],'base_estimator':[DecisionTreeClassifier(max_depth=1),DecisionT reeClassifier(max_depth=2)]}. Running the best estimator on the test data gave a score of 0.5153061224489796.

Params:{max_depth, n_estimators}	**Mean Score**	**Score Standard Deviation**
{2, 10}	0.45865237	0.04133817
{1, 10}	0.44078612	0.02439305
{1, 100}	0.43823379	0.02493189
{1, 1000}	0.4374681	0.02510375
{2, 1000}	0.31623277	0.02662013
{2, 100}	0.29989791	0.06069593

For cross validation for Linear SVM the parameters search space was {'C': [.5, 1, 10]}. Running the best estimator on the test data gave a score of 0.11122448979591837. The low score indicates that the dataset is not linearly separable and is thus non-trivial.

Params: {C}	**Mean Score**	**Score Standard Deviation**
.5	0.30423686	0.09392894
1	0.22970904	0.03418016
10	0.21745789	0.02537219

For cross validation for Non-Linear SVM the parameters search space was
[{'C': [.5, 1, 10],'gamma':10.0**-np.arange(1,4)},
{'C': [.5, 1, 10],'kernel':['poly'], 'degree':2+np.arange(3), 'gamma':10.0**-np.arange(1,4)}].

Running the best estimator on the test data gave a score of 0.5683673469387756. For this one I only put the best 12 as the search space was rather large.

Params: rbf:{C, gamma}, poly:{C, degree, gamma}	**Mean Score**	**Score Standard Deviation**
{10, 0.1}	0.5467075	0.0084574
{1, 0.1}	0.54645227	0.00406773
{10, 0.01}	0.52348137	0.00629343
{0.5, 0.1}	0.50331802	0.00190998
{10, 0.001}	0.4920878	0.00641644
{1, 0.01}	0.48111281	0.00095499
{0.5, 0.01}	0.47090352	0.00710078
{1, 0.001}	0.46988259	0.00563826
{0.5, 0.001}	0.45865237	0.00496228
{1, 2, .001}	0.38871873	0.05445008
{0.5, 2, .001}	0.38871873	0.02565814
{10, 2, .001}	0.32695253	0.04631463

For cross validation for Neural Network the parameters search space was {'hidden_layer_sizes':(11,11,11),'alpha':10.0 ** -np.arange(1, 5)}. Running the best estimator on the test data gave a score of 0.5408163265306123.

Params: {alpha}	**Mean Score**	**Score Standard Deviation**

{0.001}	0.5553854	0.01534368
{0.01}	0.55385401	0.00260287
{0.01}	0.55385401	0.00688654
{0.1}	0.55308831	0.01311904
{0.001}	0.55308831	0.00853408
{0.001}	0.55283308	0.00859493
{0.0001}	0.55104645	0.01170178
{0.01}	0.55079122	0.01146558
{0.0001}	0.55002552	0.00904545
{0.0001}	0.54874936	0.00563826
{0.1}	0.54772843	0.00969211
{0.1}	0.54083716	0.01378727

Performance Comparison
To compare the performance of the algorithms, I used the best performing hyperparameters from the previous experiments, and ran them on the data reshuffled. I got the original dataset and made 5 different permutations of it each with a training/test split and fit the algorithm on each and then scored them. From there I took the mean and standard deviation of each of the runs to compare performance.

Algorithm	**Mean**	**Standard Deviation**
Random Forest	0.6881632653061225	0.008076064872411984
AdaBoost	0.4057142857142857	0.11577132093277943
Linear SVM	0.25999999999999995	0.05431179364355108
Non-Linear SVM	0.5779591836734694	0.006938775510204079
Neural Network	0.562857142857143	0.006375714021148289

Conclusion
As can be seen from the performance table, Random Forest was able to model the data far better and far more consistently than the other algorithms. I believe this might stem from how the data was labeled. It was a human rating a wine so two different people might rate the same wine differently. Because of this a majority vote would probably result in the best classification. Random Forest employs a majority vote style system with its bagging which might line up well while decision boundary style models like AdaBoost and SVM might struggle. I feel one thing that hurt the Neural Networks performance was the fact that it was difficult to experiment too much with the structure of the hidden layers as it would quickly increase the runtime unmanageable high. So in conclusion I would say a model like Random Forest that can more effectively handle an input having multiple different class labels is best suited for inferring a rating a human might give and has the best taste in white wine.

Acknowledgements
UCI - Used for finding the dataset (https://archive.ics.uci.edu/ml/datasets/Wine+Quality)

Skikit-Learn - Used for the implementations of the algorithms (https://scikit-learn.org/stable/index.html)

A combination of Lecture material and Wikipedia articles - Used to make sure I had some of the finer details correct