Project 4

Boling, Kenneth S.[1]
(1) Earth and Planetary Sciences, University of Tennessee, Knoxville, TN 37996
kboling4@vols.utk.edu
ECE 571

Abstract

Cross-validation is a method for testing the performance of a classifier using a labeled dataset. The classifier is trained using a subset of the dataset, and a separate sub-set is then classified using the trained function. The results of this classification are then compared to the actual known values of the target feature. In this project the performance of k-nearest neighbor, decision tree, and multi-layer perceptron neural network classifiers were determined using m-fold cross-validation. The results of these performance evaluations were used to tune the parameters of the classifiers to minimize the error of the results.

Contents

Introduction

Cross-validation can be used to ensure a classifier is not over fitted to the training data. This process works by dividing the original dataset into a number (m) of partitions (folds). The data from all but one of these folds is treated as the training data set for the classifier, which is then run on the final remaining fold. The performance of the classifier is then evaluated and the process is repeated using a different fold.

The data used in this project is from the *fglass* dataset, which contains elemental content and refractive index (RI) of 214 fragments of glass from various sources (B.D. Ripley, 1996). The glass data include six types of glass labeled 1 through 7 (type 4 is not used). This dataset was originally produced for the purpose of using glass fragments for forensic investigations. A pairplot of all features in the data set is displayed in Figure 1.

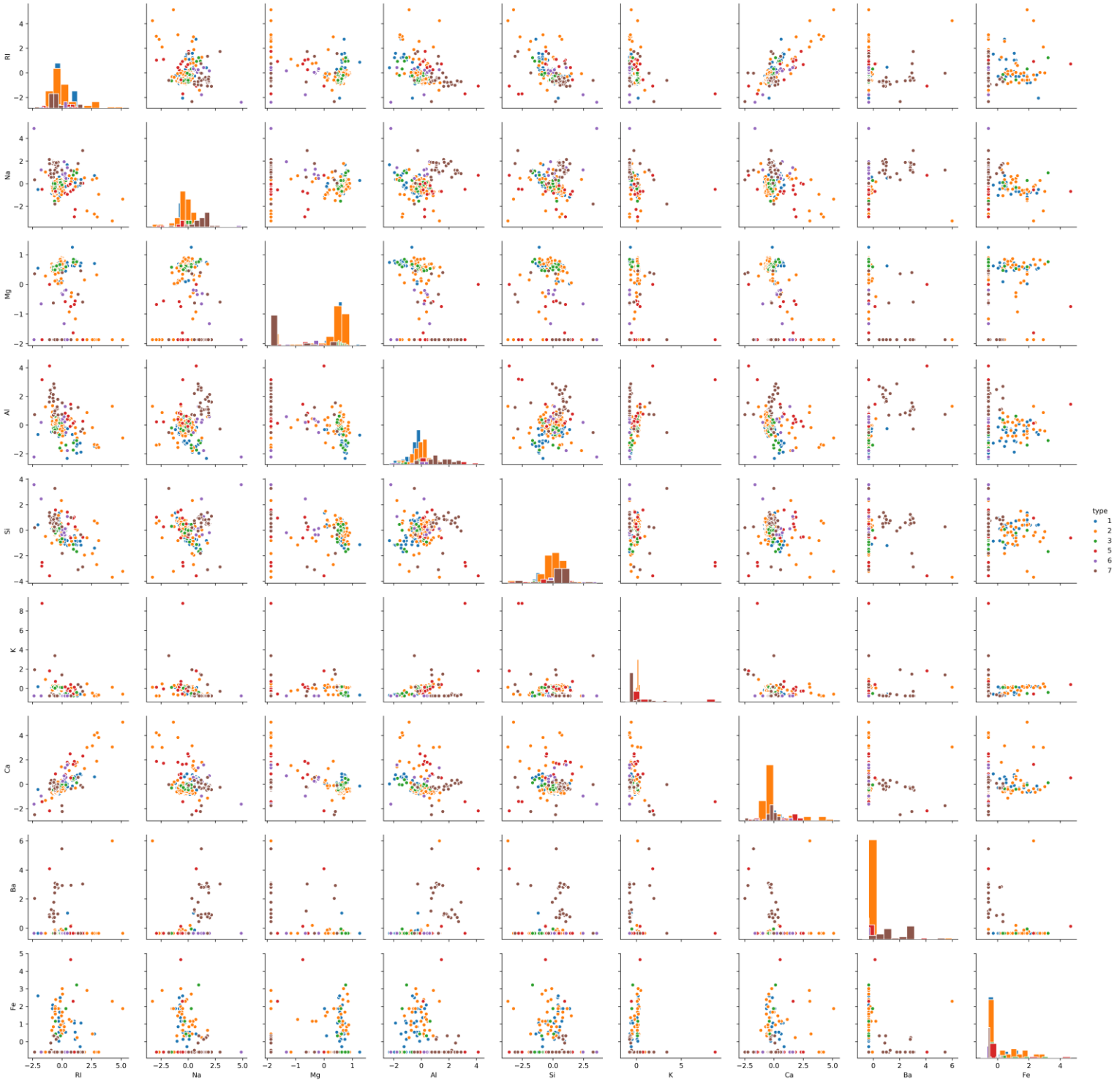

Figure 1: Pair-plots of the *fglass* dataset (after normalization) showing the relationships in the data. Few classes appear to be easily linearly separable.

Methods and Technical Approach

Several classifiers were used to classify each fold, however these classifiers first require the data to be normalized. The data were normalized using standard score or z-score normalization:

Equation 1: "Z-Score" or "Standard Score"

$$z = \frac{x - \mu}{\sigma}$$

m-Fold Cross-Validation

Normally m-fold cross-validation uses a number of randomly selected samples from the original data and puts them into (m) number of folds, but for this project the subsets have been pre-defined. The dataset provided is broken down into 10 predefined sub-sets of the original dataset. Each of the classifiers used for this project were trained using 9 of these folds and the remaining fold was used as a testing dataset.

k-Nearest Neighbor

A kNN classifier which was constructed during project two was trained on each fold using different values for k. The performance of this classifier was then evaluated for each value of k using the averaged accuracy for each of the ten cross-validation fold iterations. Simple euclidean distance was used as the distance metric.

Decision Tree

A decision tree classifier is a non-parametric supervised learning method. The classifier develops a set of decision rules based on the training data set. Each of these rules are usually binary decisions and group data into two output nodes based on the value of a single feature. The *DecisionTreeClassifier()* function from the *scikit-learn* python library (Pedregosa et al., 2011)was used as the classifier. Each node attempts to find a value of a feature that best separates the data, such that the total Gini Impurity (measurement of the heterogeneity of each class within the node, where if all samples in the node are from the same class the Gini Impurity = 0, and if all samples are from different

classes and in equal proportions the Gini Impurity = 1.0) of the classes is reduced as much as possible. This process continues until all data points from each class are separated into terminal nodes which make up the 'leaves' of the tree.

Multi-Layer Perceptron (Neural Network)

A multilayer perceptron is a neural network with more than one layer. This project used a neural network consisting of three layers and a variable number of nodes to classify the testing data set for each m-fold iteration. The *MLPClassifier()* function from the *scikit-learn* python library was used for this project. Different numbers of hidden nodes were used and the performance was determined for each number of hidden nodes.

Experiments and Results

The *fglass* data was classified using each of the classifiers. The predefined m-fold cross-validation sets were used for each. All classifiers were implemented with varying parameters to find the optimal parameters for this dataset. One parameter was varied for each classifier using a single variable. A total of twenty (20) iterations were performed across each of the classifiers for comparison purposes.

kNN

The averaged 10-fold cross-validation accuracy of the kNN classifier was notably higher for k-values less than 4. The optimal values for k seems to be 2 or 3 with an accuracy of ~0.70. The maximum value of k was initially set at k = 15 based on the sqrt(n) ~ 15 (where n is the total number of samples in the data set). The maximum was changed to k=20 for comparison with other classifiers.

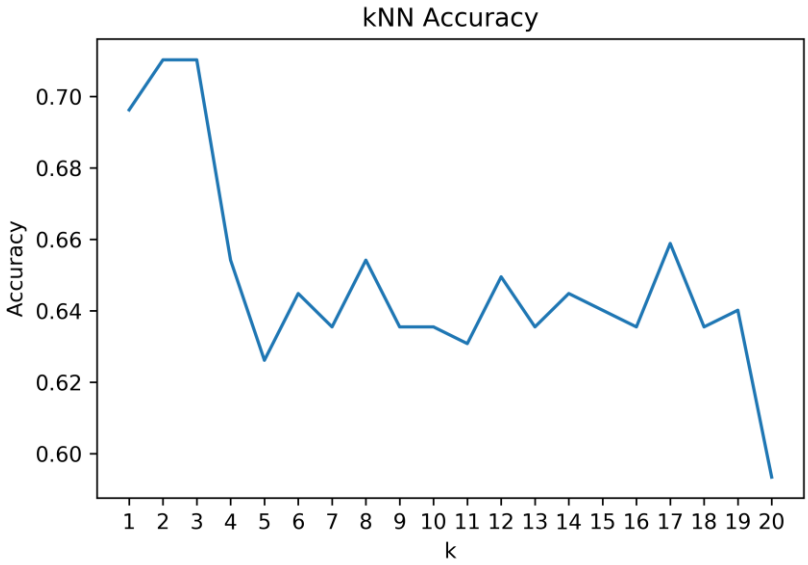

Figure 2: Accuracy of kNN for values of k using the defined m-fold cross-validation averages

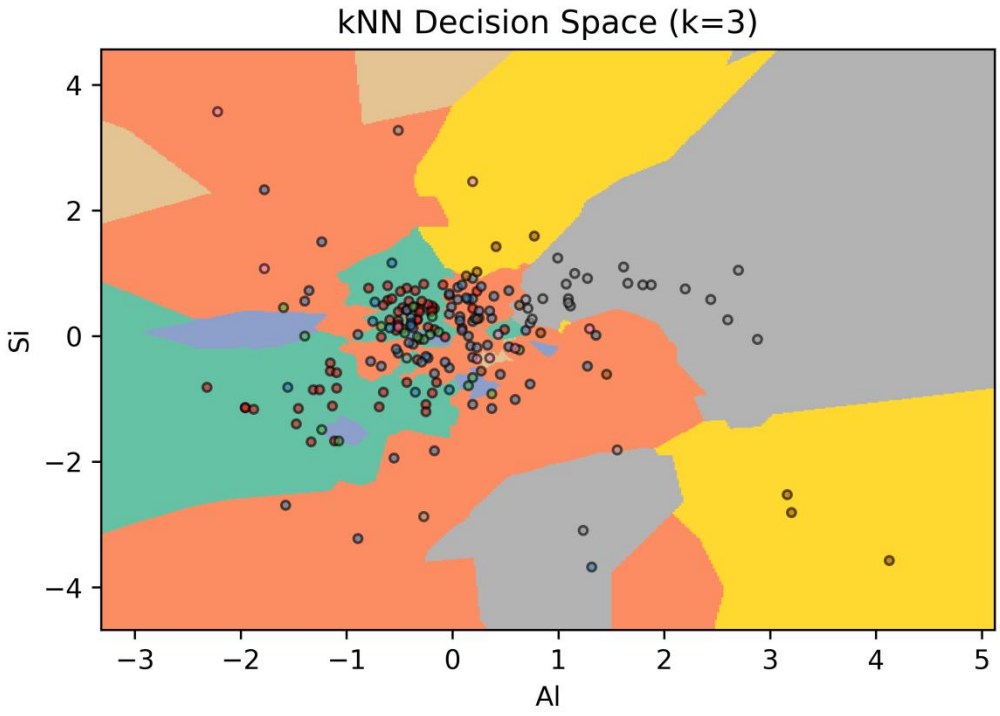

Figure 3 : k- Nearest Neighbor decision regions for the six classes with aluminum and silicon using the optimal k = 3 value. Points shown are training data points. (Please note that the decision regions for this plot are only valid for the 2-d space of two features and are only meant to show the complexity and general structure of the decision regions for comparison purposes).

Decision tree results

The results of the m-fold cross validation appear to vary slightly for each run even when running the same algorithm multiple times over the same subset of the dataset. This is due to the random starting state of the decision tree (though this can be set to a specified seed to get the same result each time). The decision trees generated using the default settings seem to be over-fitted and result in a large number of terminal leaf nodes (Figure 6). Better results were obtained by limiting the minimum impurity reduction required to split a node, or limiting the maximum number of leaf-nodes.

One of the advantages to decision trees is they can be easily graphically represented once trained, in contrast to Neural Networks. A few results of a single iteration were represented visually using the *GraphViz* library using the output from *scikit-learn*.

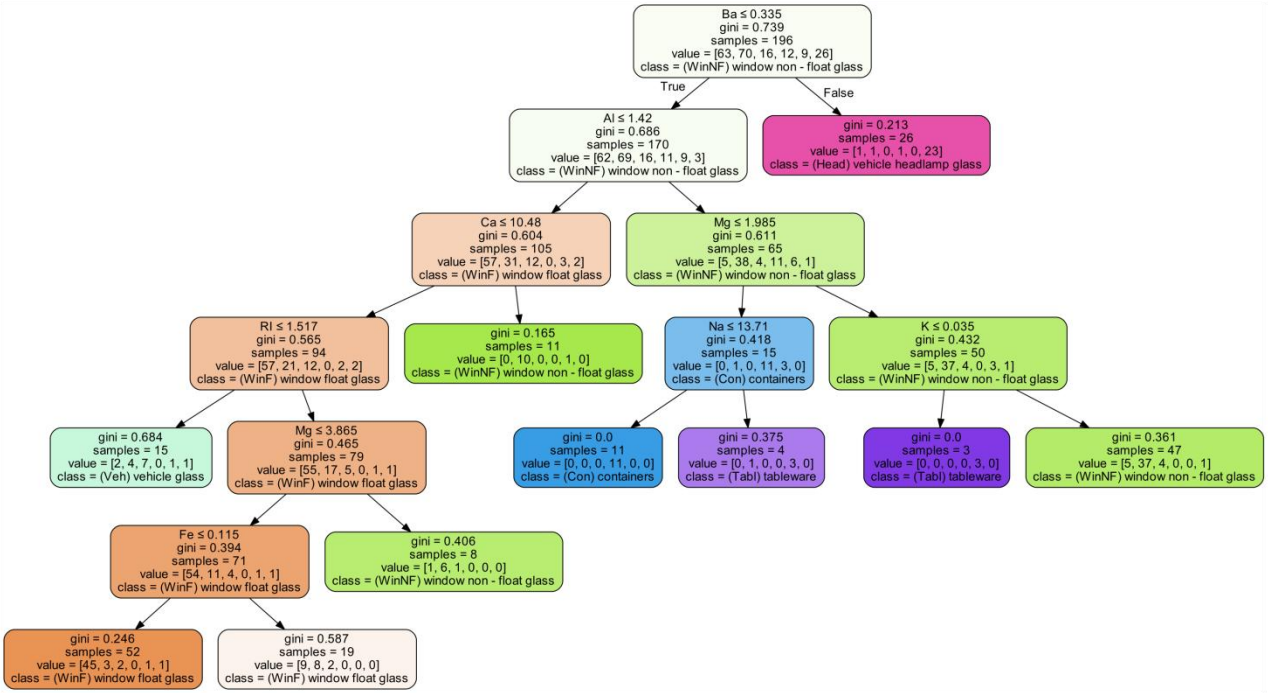

Figure 4: Optimal results from setting the minimum required impurity decrease to split a node = 0.02

Several parameters were varied for the decision tree classifier to find the optimal parameters for the dataset using the m-fold cross-validation accuracy. These parameters include: The Maximum number of leaf nodes, the maximum "depth" of the tree, the minimum number of samples per leaf, and the minimum impurity decrease required to create a new split.

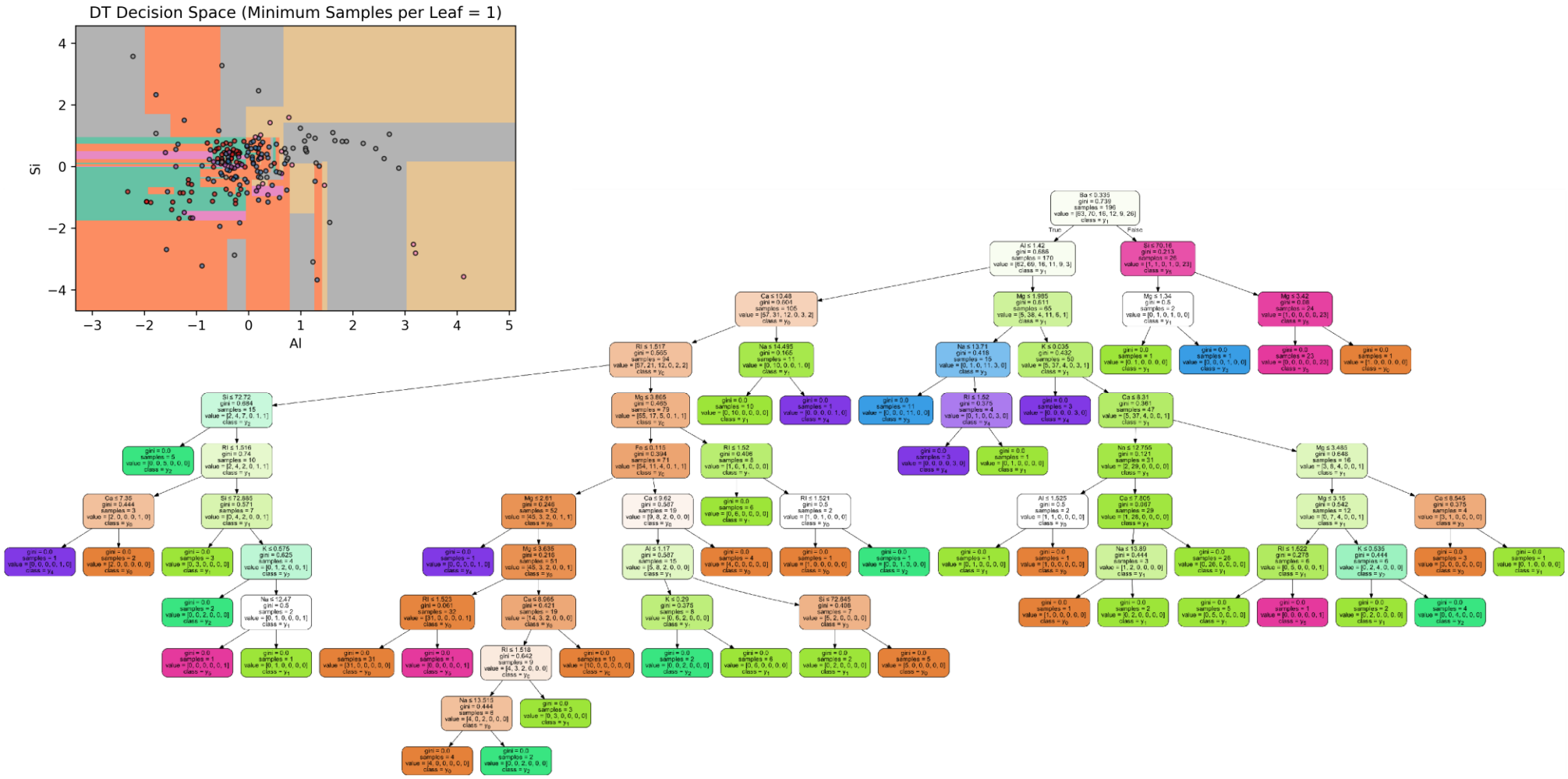

Figure 5: Over-fitted decision tree. No restrictions on growth, the decision tree is allowed to continue to grow until each leaf has completely separated all classes (minimum samples per leaf =1). The decision space for the over-fitted decision tree is shown in the top left.

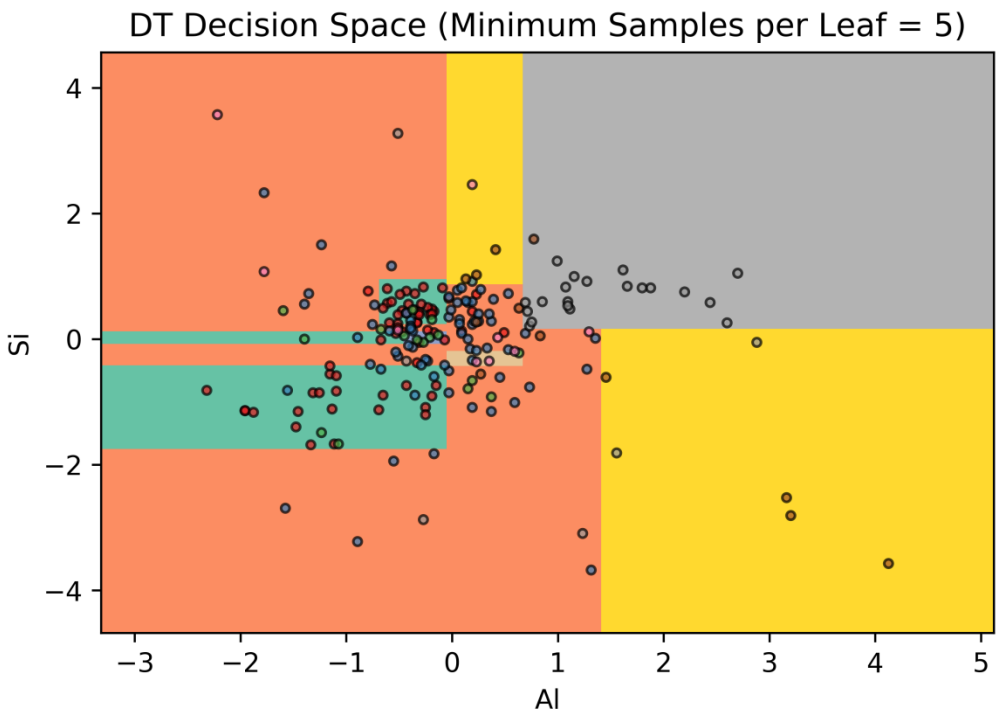

Figure 6: Decision Tree algorithm results showing the decision regions for the six classes across two features (aluminum and silicon) using the optimal parameters

The results from setting either the Maximum number of leaf-nodes or the maximum depth did not greatly affect the accuracy once the parameter was over a threshold value. The minimum number of samples per leaf parameter had the greatest impact on the results. Setting this value can control the final number of leaf nodes generated by the decision tree algorithm, and can prevent over fitting to the training dataset. The accuracy plot of this parameter indicates that setting this parameter to a value of between 1-4 results in lower accuracy due to over-fitting (because the resulting number of leaf-nodes is so high), and setting the parameter to a number greater than 5 results in under-fitting, because there are too few nodes to split apart the different classes.

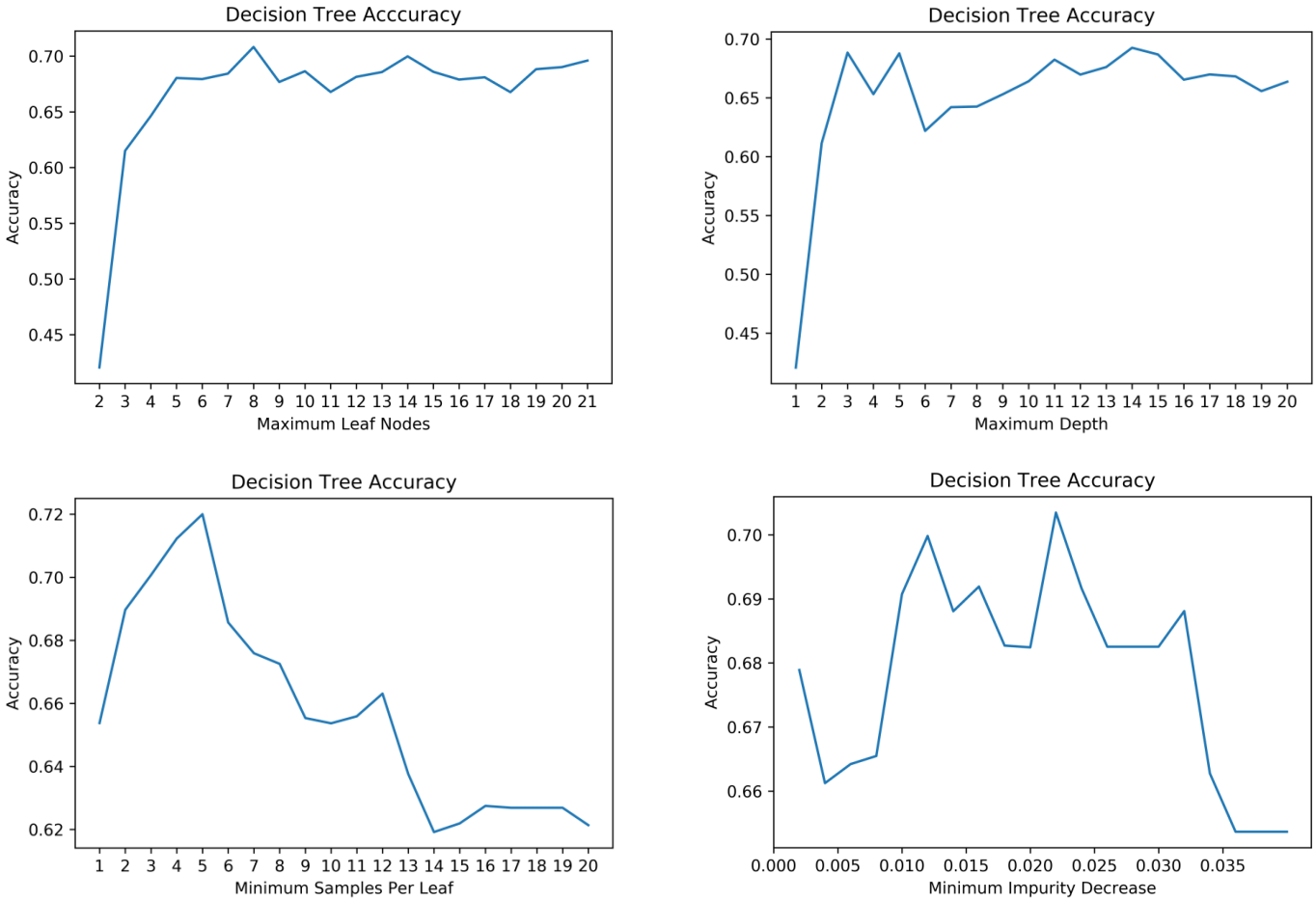

Figure 7: Visualizations of the accuracy resulting from different decision tree parameters.

Neural Network (Multi-Level Perceptron)

Neural Networks attempt to simulate the biological funtions peformed by neurons in the brain. This is done by assigning weights to the inputs of the perceptron, and determining the weight values that minimise the loss function using gradient decent. The (Single Layer) perceptron can be used for classifcation only if a linear decision boundary exisits. This limitation was able to be overcome by the development of the Multi-Layer Perceptron. The multi-layer perceptron consits of an input layer, an output layer, and a "hidden layer" in between. This hidden layer can consitst of multiple layers, each with a number of hidden nodes. For this project, only one hidden layer was used, but a variable number of hidden nodes were used.

The scikit-learn library multi-level perceptron includes an option for obtaing the training cross-validation score using the training dataset which was used to build the model. This value will always be greater than any testing data accuracy. This is shown in Figure 8, where the accuracy is shown to increase slightly with an increasing number of nodes.

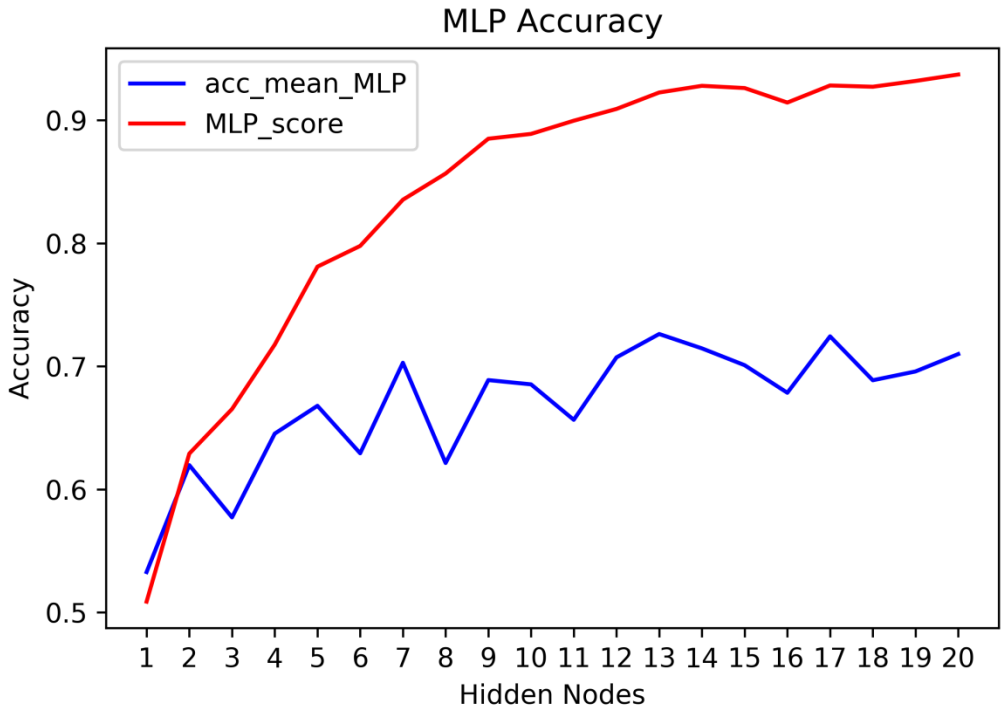

Figure 8: Plot showing the accuracy of the model using the training data (red) and testing data (blue). The model performs much better for the training set as would be expected, since this is the data which was used to construct the model.

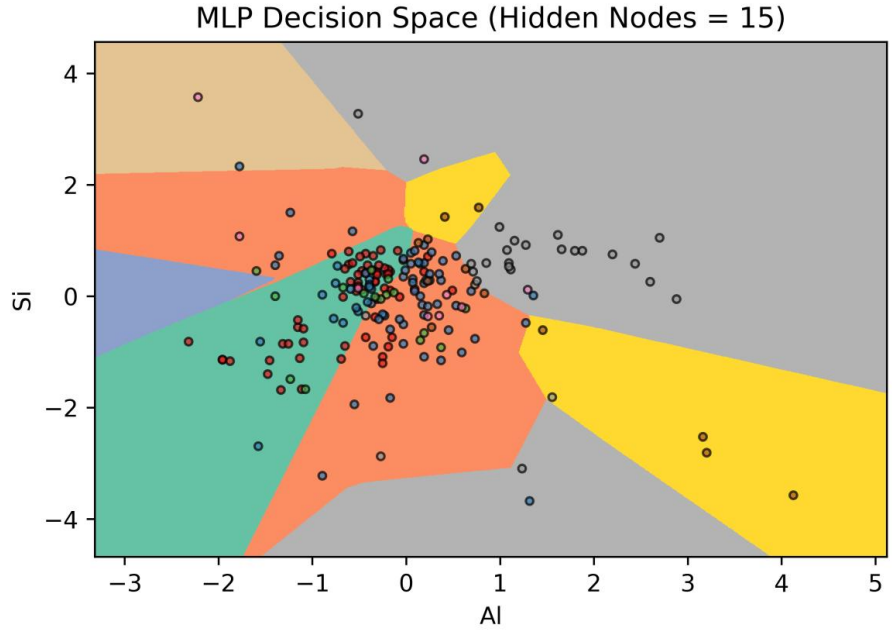

Figure 9: Multi-Level Perceptron Neural Network algorithm results showing the decision regions for the six classes across two features (aluminum and silicon) using the optimal parameters

Classifier performance

Averaged accuracy for each classifier using different parameters. The parameter value (k) for the kNN classifier is as it is typically defined; the number of training points used to classify an unknown sample. The parameter value was also used to change the parameters of the other classifiers. The decision tree classifier was constrained by the parameter that controls the minimum number of samples required for an exit node (leaf). The multi-layer perceptron neural network was constrained by the parameter value that controls the number of hidden nodes in the hidden layer (between the input and output nodes).

Table 1: Accuracy of each classifier

MLP(NN)	kNN	DT	"parameter"
0.289	0.692	0.657	1
0.448	0.701	0.687	2
0.506	0.692	0.691	3
0.472	0.645	0.718	4
0.543	0.614	0.713	5
0.611	0.635	0.686	6
0.540	0.629	0.676	7
0.624	0.651	0.673	8
0.631	0.625	0.655	9
0.652	0.633	0.654	10
0.617	0.624	0.656	11
0.666	0.643	0.668	12
0.679	0.630	0.638	13
0.651	0.641	0.619	14
0.654	0.641	0.616	15
0.661	0.632	0.627	16
0.652	0.655	0.627	17
0.696	0.630	0.627	18
0.685	0.640	0.627	19
0.684	0.588	0.621	20

Discussion

The use of m-fold cross-validation is useful for tuning parameters of classifiers. The best performing classifiers for the fglass data set appears to be the decision tree algorithm with a minimum of 4 or 5 samples per leaf node. The second best performing classifier was the k-nearest neighbor algorithm with k=2. The multi-layer perceptron was the worst performing classifier, though the number of nodes did seem to increase classifier performance slightly with higher numbers of nodes.

The k-nearest neighbor algorithm saw the best results at k=2 or 3, most likely because this dataset is highly variable and seems to have very small clusters of similar points. The parameter that resulted in the optimal performance for the decision tree was the minimum number of samples per leaf. In this case, a minimum number of samples of less than 4 resulted in over fitting, and greater than 5 resulted in under fitting. The other parameters tested for the decision tree algorithm did not seem to affect the performance so long as it was greater than a certain value.

Finally, the multi-layer perceptron neural network seemed to increase in accuracy with increasing hidden nodes, however it was still out performed by the other classifiers. It may be that the optimal number of nodes was not reached in this range of values however. Increasing the number of hidden layers also seemed to increase this performance.

References

B.D. Ripley, 1996, Datasets used in the book Pattern Recognition and Neural Networks by B.D. Ripley (1996):, http://www.stats.ox.ac.uk/pub/PRNN/ (accessed January 2019).

Pedregosa, F. et al., 2011, Scikit-learn: Machine Learning in Python: Journal of Machine Learning Research, v. 12, p. 2825−2830.

Appendix

Decision tree limited leaf nodes to 15