Springboard

Capstone Project I: Seeds

Final Report

E. Erik Larsen

Data Science Career Track

January 6th Cohort

I. Introduction

The identification of seeds from geometric properties alone could prove to be a great boon for the agriculture industry, leading to better quality in crop yields as well as retail and urban gardening. Business enterprises involved in the sorting, selling and distribution of seeds are concerned with the quality of their product, whether it be grain or flower seeds. Being able to classify seeds with machine learning algorithms will reduce costs in the quality control department while at the same time increasing the quality of the product. This extends from the sorting and packaging room floor all the way to bulk crop production. Human error will be reduced, saving costs in man-hours and increasing the uniformity of agricultural plants in large scale production.

II. Overview

This project's goal is to investigate the performance of several machine learning algorithms in classifying seeds as one of three species of wheat: Kama, Rosa, or Canadian. Data is explored with graphical representations and analyzed using common statistical techniques. The results of these analyses are used to identify correlations and ultimately aid in feature selection. Methods used included k-nearest neighbors, k-means, logistic regression, naive Bayes, and dense neural networks. While most models show good performance, the best algorithms trained were logistic regression and a neural network.

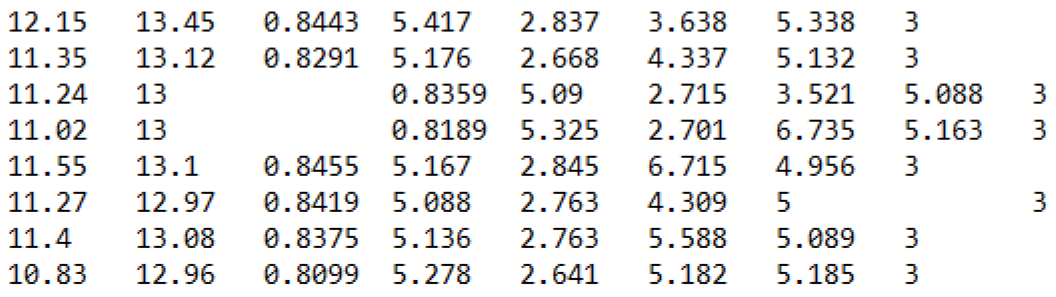
III. Authentics

All analyses are done using Jupyter Notebooks with Python version 3.7.6. These notebooks can be found in the author's gitbhub.com repository at https://github.com/nanodoc2020/Springboard/tree/master/Capstone-Project1. Packages used include the latest versions of pandas, keras, numpy, seaborn, scikit.learn, statsmodels, scipy.stats, pymc3, and matplotlib.pyplot.

IV. Raw Data

The Seeds data set can be found on the UCI ML Repository at https://archive.ics.uci.edu/ml/datasets/seeds. The information is saved in .txt format. It is simple to download and has a description of the features. Upon inspection, it is clear that compared to most data sets in use today, this one is very small; it uses approximately 10 kilobytes of memory. The miniscule size of the dataset is one of the challenges to overcome in this project. A number of ways to address this have been considered including cross-validation, bootstrapping, and creating synthetic data. Synthetic data is not used in model training, but a method to create more "data points" is described in a later section. Bootstrapping is used to determine the statistical significance of feature differences, but could be used to create more points and different sample populations.

There are 210 examples and 8 variables. Seven of these variables are numeric and essentially continuous; they are geometric properties of the seeds taken from precise measurement using X-ray photo plates. Their attributes are area, perimeter, compactness, length, width, asymmetry coefficient, and groove length. The final variable is categorical, and gives the species of the seed with three distinct values: 1="Kama", 2="Rosa", 3="Canadian". The 210 examples are divided evenly among the species with 70 entries for each, eliminating the need to stratify when separating into training and test sets. Feature names are not included in the original data, and so must be added.



```
12.15   13.45   0.8443  5.417   2.837   3.638   5.338   3
11.35   13.12   0.8291  5.176   2.668   4.337   5.132   3
11.24   13              0.8359  5.09    2.715   3.521   5.088   3
11.02   13              0.8189  5.325   2.701   6.735   5.163   3
11.55   13.1    0.8455  5.167   2.845   6.715   4.956   3
11.27   12.97   0.8419  5.088   2.763   4.309   5               3
11.4    13.08   0.8375  5.136   2.763   5.588   5.089   3
10.83   12.96   0.8099  5.278   2.641   5.182   5.185   3
```

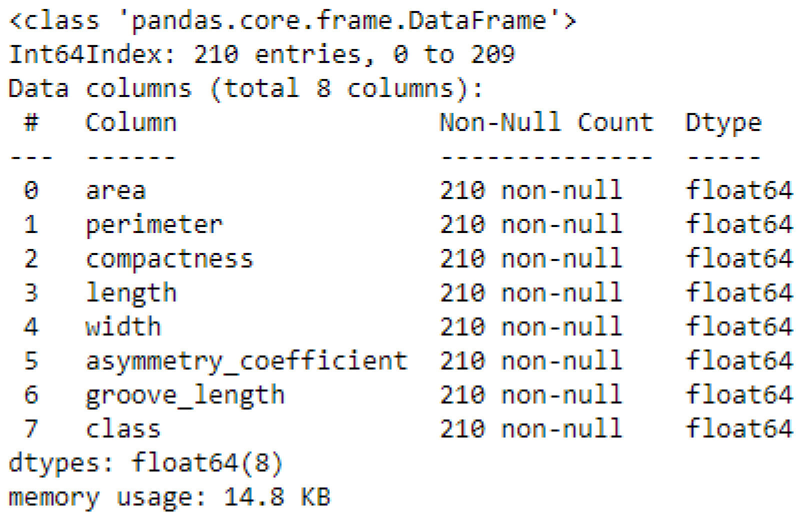
Raw data as seen on the UCI ML website

V. Data Wrangling

There are no missing values in the raw dataset. There is, however, a mismatch of data in correct columns. After inspecting the original text file, the mismatched columns were easy to see. Some of the entries had, in random places, been shifted over to the right a column or two, leaving blank spaces where the information should be. This misalignment is present in about 5% of the row entries, and causes an error when attempting to load the file directly into a pandas dataframe. Skipping these rows, the file can be read into a dataframe with 11 missing entries.

An alert giving the rows seen allows us to open the file and write in an initial row that is beyond the greatest column of misalignment. Doing this allows all of the data to be read into the file without skipping any rows. The consequence is that there are now extra columns that are full of NaN values in the dataframe. The actual corrupted data can be identified, stored for cleaning, and removed from the rest of the properly aligned dataframe.

To align the wayward data, it is simple to read the correct values into a list in the correct order, leaving out the NaN entries causing the misalignment. A function in the final data wrangling notebook does this and returns a new dataframe with all columns and rows properly aligned and indexed. This is then appended to the uncorrupted dataframe, and sorted back in the proper order. The result is a tidy dataframe with no missing values. This is saved into an Excel .csv format for exporting and statistical analysis.

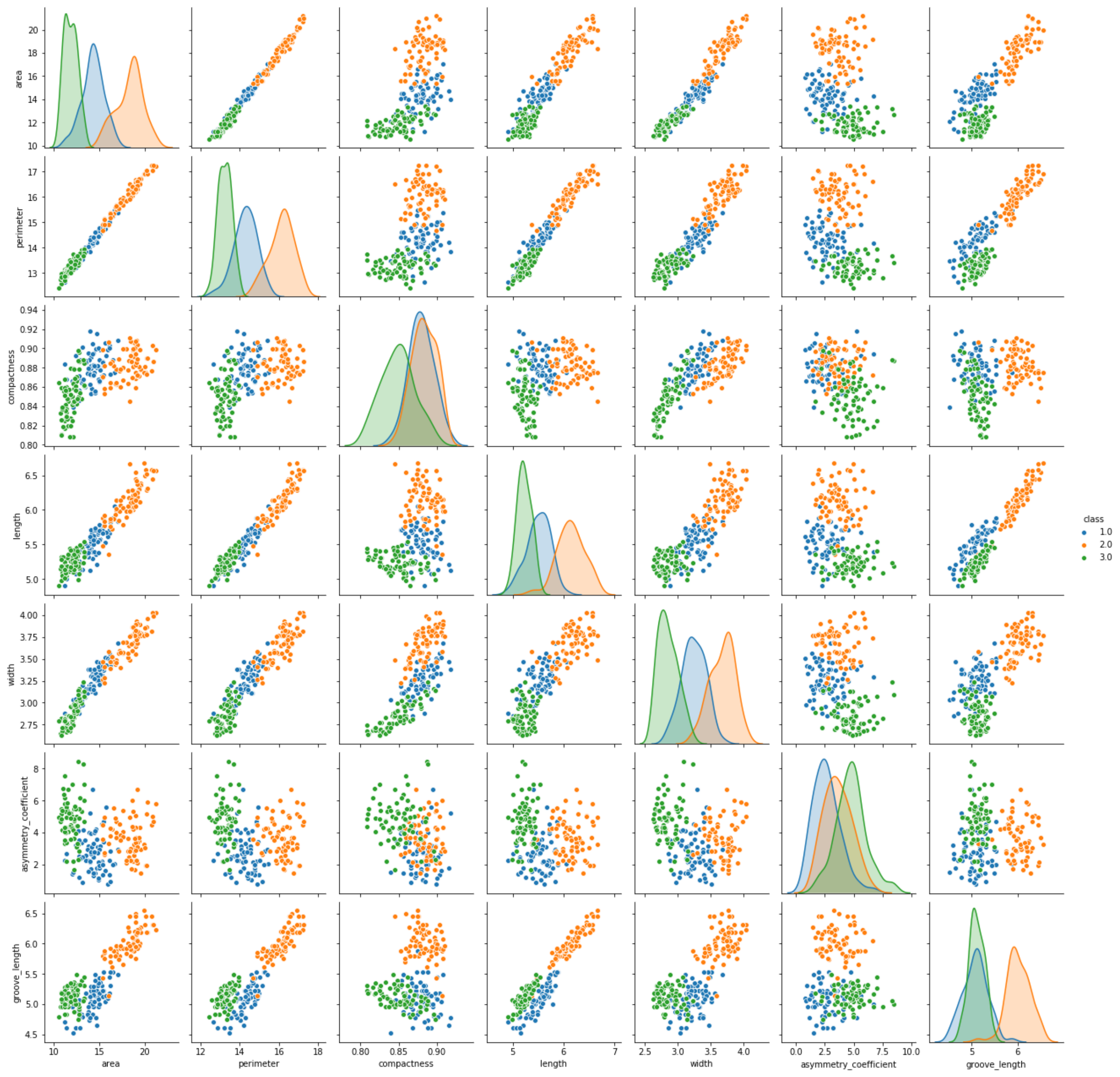


```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 210 entries, 0 to 209
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   area                   210 non-null    float64
 1   perimeter              210 non-null    float64
 2   compactness            210 non-null    float64
 3   length                 210 non-null    float64
 4   width                  210 non-null    float64
 5   asymmetry_coefficient  210 non-null    float64
 6   groove_length          210 non-null    float64
 7   class                  210 non-null    float64
dtypes: float64(8)
memory usage: 14.8 KB
```

Calling seeds.info() shows no missing values and uniform data type

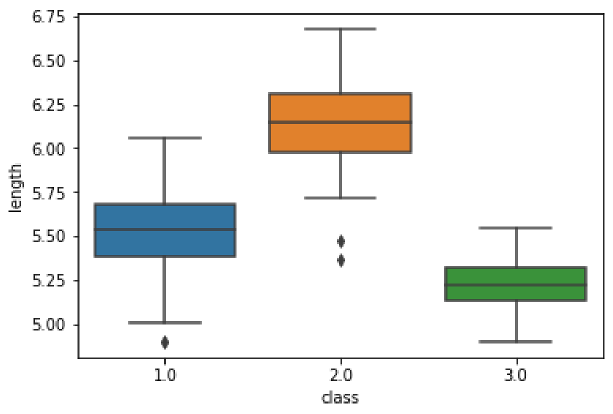
VI. Exploratory Data Analysis

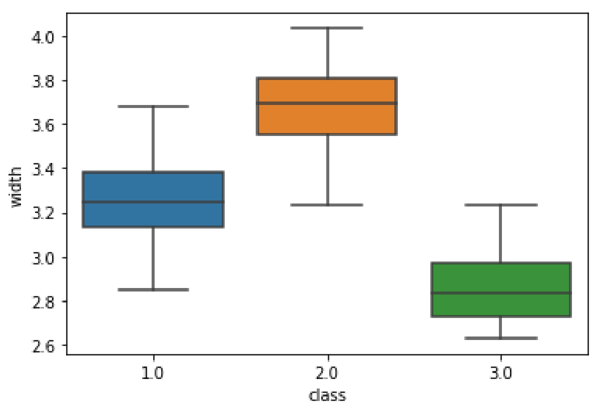
The exploratory data analysis shows great insight into the relationships between the seeds' geometric properties. The full extent of graphics produced can be found here: https://github.com/nanodoc2020/Springboard/blob/master/Capstone-Project1/Data_Story/Data_Story_Notebook.ipynb. Some of the graphs can be seen below. Using seaborn's pairplot function produces a wonderful visual display of their differences.

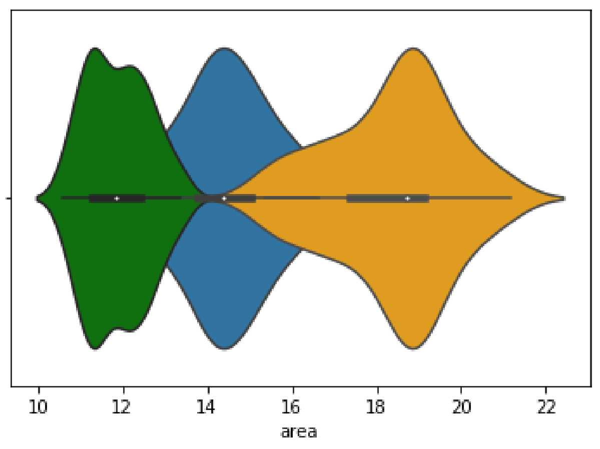


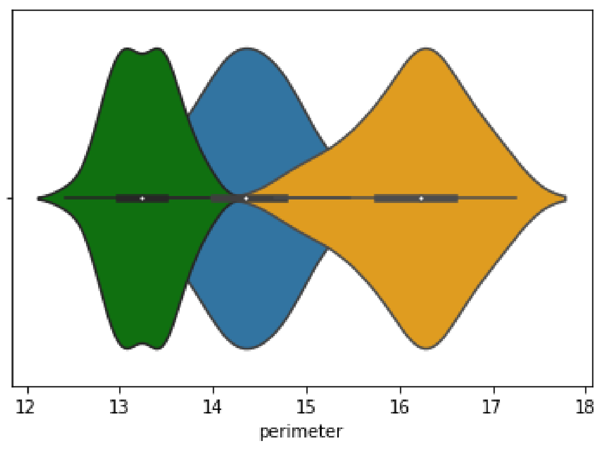
Examining the scatter plots above, there appears to be recognizable clustering. The clustering is more prominent in certain feature comparisons, such as length vs. groove length. The distributions on the diagonal show some skew, but are mostly smooth and not quite normally distributed. The groove length distribution shows separation for one class, and complete overlap of the other two. This separation leads to identifiable clusters in plots where groove length is involved. The area distributions show decent separation, and when combined with groove length could lead to even better clustering. In all cases there is some overlap of the distributions, meaning that it will always be possible to misclassify a certain seed because it will have properties that fall under the distribution of the incorrect classification.

Many of the scatter plots show excellent linear relationships. The length vs. groove length plot shows good cluster separation, as well as a linear trend in each class. The derived features, i.e. asymmetry coefficient and compactness, show no clear trend. They tend to appear more nebulous in relation to the other variables, which may not be of surprise because they are derived from them.

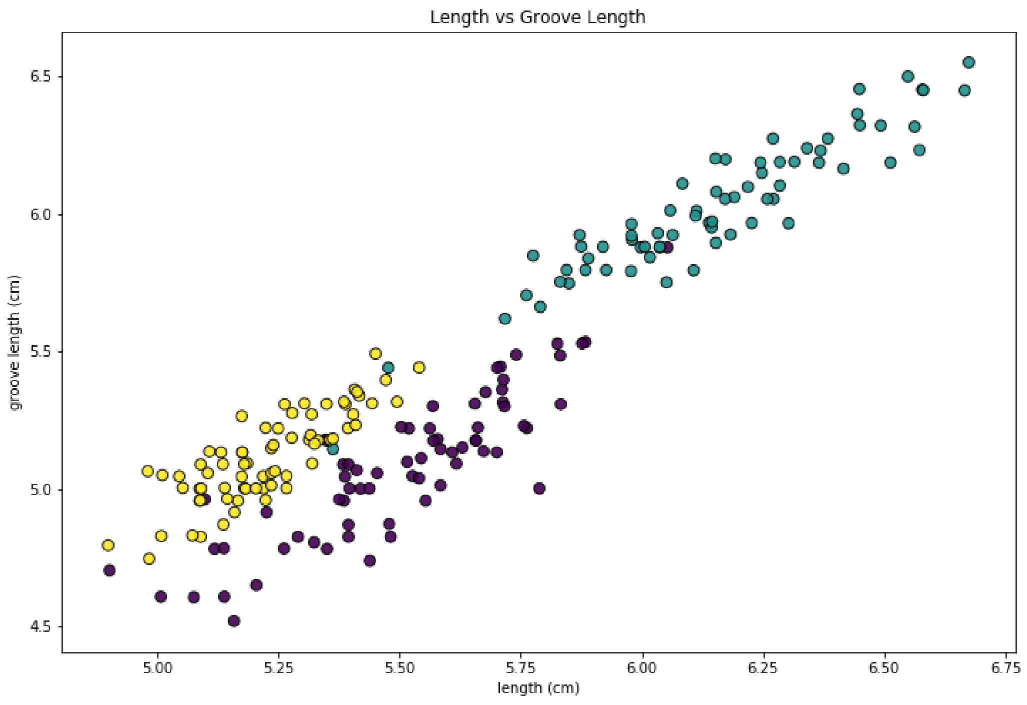








The above plots give us a good understanding of how the class features are individually distributed, as well as how they relate to one another. In the accompanying data story notebook mentioned previously, there are comparisons for every feature. These have been omitted here for brevity. There are a few outliers that can be seen in each class, but class 1 (Kama) seems to have the most. Several of the outliers are found in the derived features, as well, and pose the question if they should be considered in the physical picture of the seeds.



Clustering and linear relationships are evident

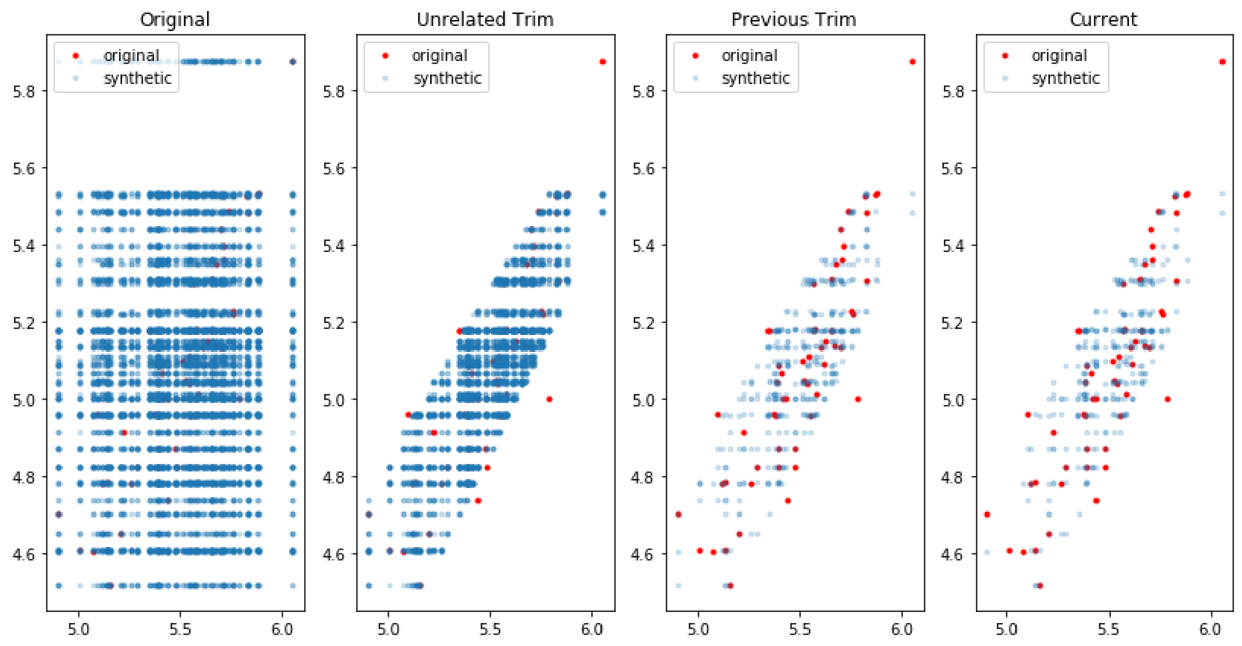
Now we can clearly see the class clusters. The Kama seed seems to be the most separated. Also apparent is the linear relationship between these two variables in each class. There are a few overlaps, which will make classification more difficult for these extreme cases. A new question arises from them: are there any misclassifications? That question is impossible to answer without input from the dataset's creators. Thus, we will proceed with the assumption that there are no misclassifications.

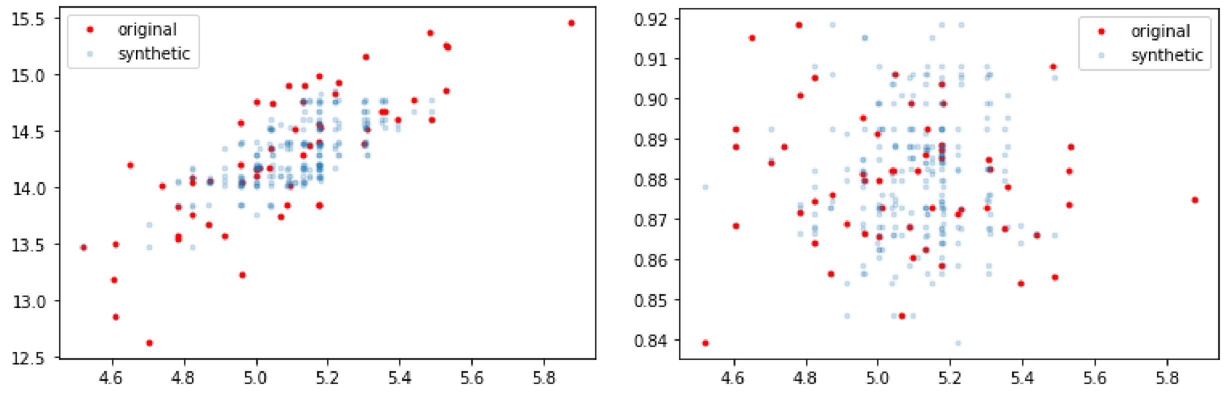
Though the seeds dataset is small, it is rich with correlations between features. Clustering is seen in the scatterplots, with the most separation seen in the length vs. groove_length comparison. This can be used to make a rather accurate k-nearest neighbors algorithm. Each class has different summary statistics for its features. Consequently, each feature is distributed differently for each class. These distributions are in general not normal, but some of them come close to fitting a normal curve.

The underwhelming size of the data can be addressed in three different ways, two of which have been developed here. The third is cross validation (CV), which will be performed during algorithm training. First, we can create as many larger bootstrap samples as we need, sampling the training set with replacement. These different samples can give us a confidence interval for the population, a mean of means, etc., that does approach a normal distribution in the large limit, and describes the entire population of seeds more accurately. However, there are no new data points with which to train and gain new information in the bootstrap method.

Synthetic Data

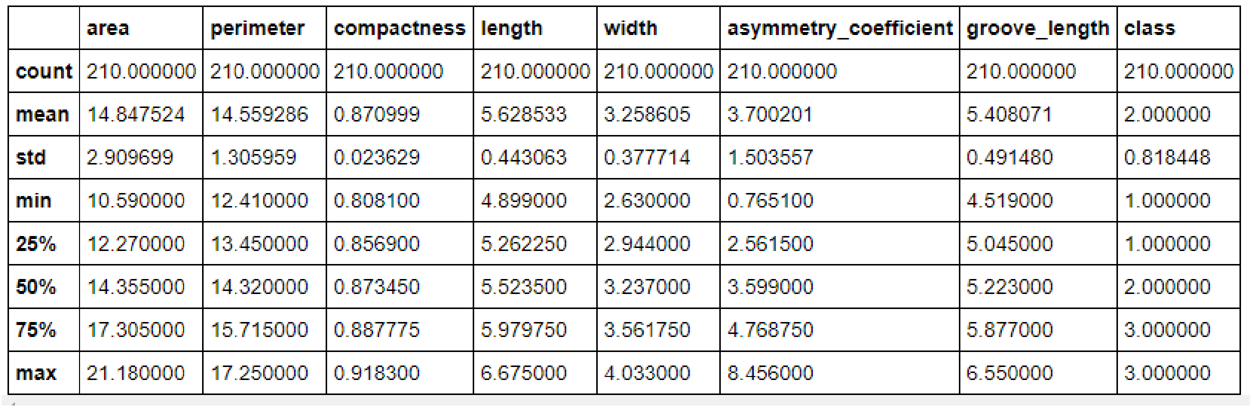
One way to introduce new data is to create a large range of sample points by sampling every distribution across the columns and making new points with these uncoordinated sample values. Due to the randomness, many of the points are unacceptable because one or more features has a value that lies well outside an acceptable range for that feature. In order to exclude the erroneous rows of new synthetic data, we find best fit lines for highly correlated pairs of features in the original set, and trim off the synthetic points that lie outside an arbitrary range close to the best fit line. In this way, no outliers for any feature are introduced, and clustering around the mean tends to be left. We then append the original data to the synthetic, creating a larger dataset that contains all of the original points with their natural outliers, as well as the new data that tends to be clustered around the mean. The efficacy of this bonsai method must be tested, comparing an algorithm's performance on the expanded set versus the original data alone. Below are the results of adding random points and trimming as described.





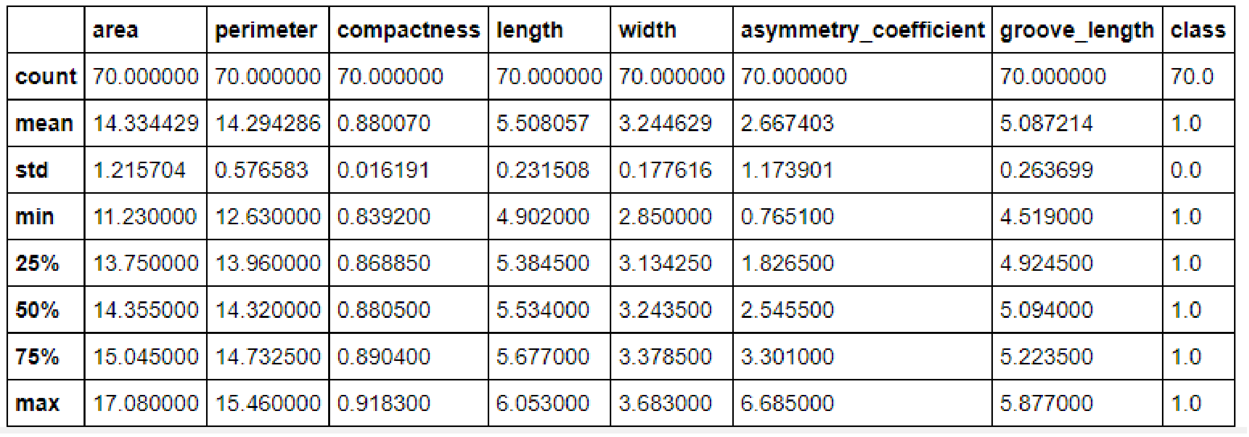
VII. Statistical Analysis

Pandas has a .describe() method which calculates all of the summary statistics for a data set. The tables below show such values for each class as the mean, standard deviation, and quartiles. A clear difference can be seen in the values for each feature between each species. This should help a machine learning algorithm to distinguish between the three. Next we will use bootstrapping to determine confidence that these differences are significant across more than just the small sample.



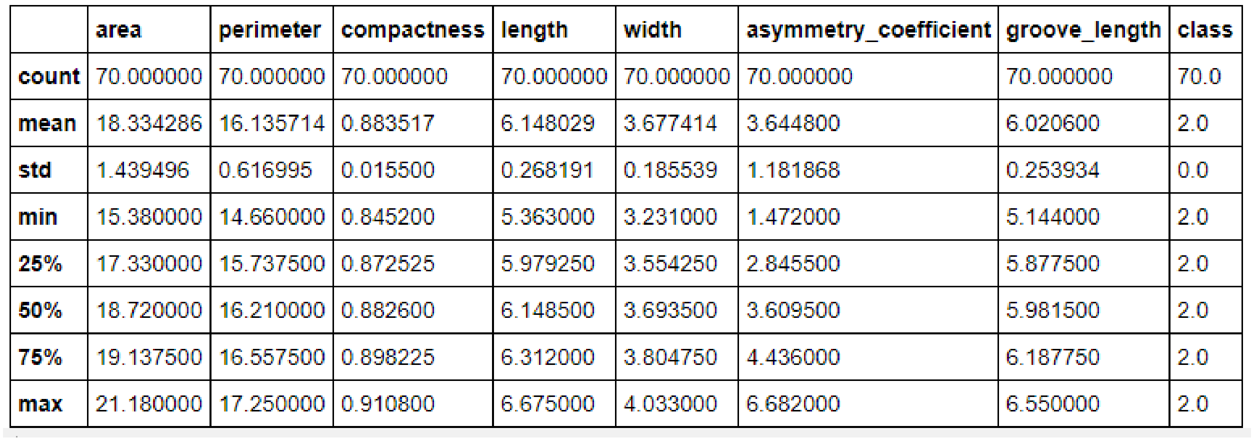
	area	perimeter	compactness	length	width	asymmetry_coefficient	groove_length	class
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000
mean	14.847524	14.559286	0.870999	5.628533	3.258605	3.700201	5.408071	2.000000
std	2.909699	1.305959	0.023629	0.443063	0.377714	1.503557	0.491480	0.818448
min	10.590000	12.410000	0.808100	4.899000	2.630000	0.765100	4.519000	1.000000
25%	12.270000	13.450000	0.856900	5.262250	2.944000	2.561500	5.045000	1.000000
50%	14.355000	14.320000	0.873450	5.523500	3.237000	3.599000	5.223000	2.000000
75%	17.305000	15.715000	0.887775	5.979750	3.561750	4.768750	5.877000	3.000000
max	21.180000	17.250000	0.918300	6.675000	4.033000	8.456000	6.550000	3.000000

Summary Statistics: All Seeds



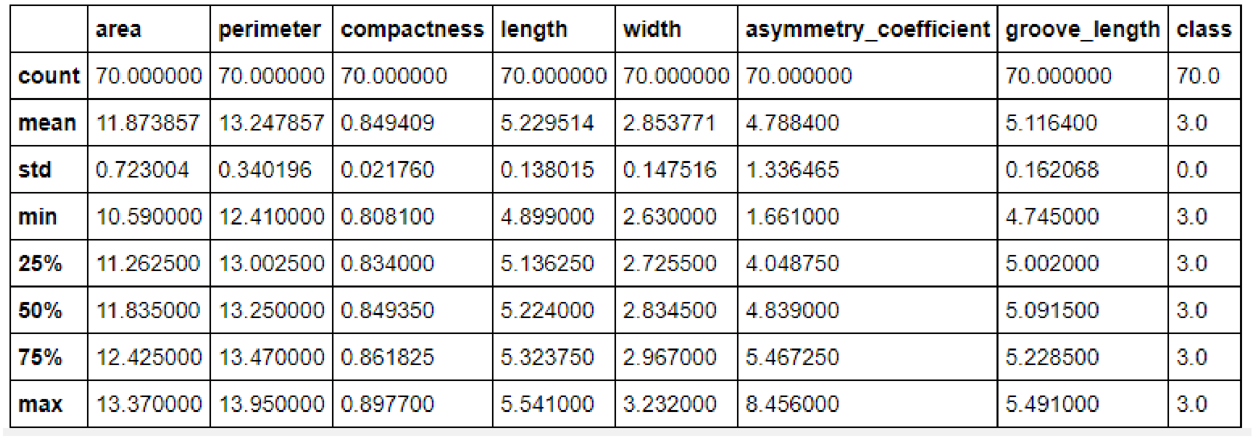
	area	perimeter	compactness	length	width	asymmetry_coefficient	groove_length	class
count	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.0
mean	14.334429	14.294286	0.880070	5.508057	3.244629	2.667403	5.087214	1.0
std	1.215704	0.576583	0.016191	0.231508	0.177616	1.173901	0.263699	0.0
min	11.230000	12.630000	0.839200	4.902000	2.850000	0.765100	4.519000	1.0
25%	13.750000	13.960000	0.868850	5.384500	3.134250	1.826500	4.924500	1.0
50%	14.355000	14.320000	0.880500	5.534000	3.243500	2.545500	5.094000	1.0
75%	15.045000	14.732500	0.890400	5.677000	3.378500	3.301000	5.223500	1.0
max	17.080000	15.460000	0.918300	6.053000	3.683000	6.685000	5.877000	1.0

Summary Statistics: Class 1 (Kama)



	area	perimeter	compactness	length	width	asymmetry_coefficient	groove_length	class
count	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.0
mean	18.334286	16.135714	0.883517	6.148029	3.677414	3.644800	6.020600	2.0
std	1.439496	0.616995	0.015500	0.268191	0.185539	1.181868	0.253934	0.0
min	15.380000	14.660000	0.845200	5.363000	3.231000	1.472000	5.144000	2.0
25%	17.330000	15.737500	0.872525	5.979250	3.554250	2.845500	5.877500	2.0
50%	18.720000	16.210000	0.882600	6.148500	3.693500	3.609500	5.981500	2.0
75%	19.137500	16.557500	0.898225	6.312000	3.804750	4.436000	6.187750	2.0
max	21.180000	17.250000	0.910800	6.675000	4.033000	6.682000	6.550000	2.0

Summary Statistics: Class 2 (Rosa)



	area	perimeter	compactness	length	width	asymmetry_coefficient	groove_length	class
count	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.0
mean	11.873857	13.247857	0.849409	5.229514	2.853771	4.788400	5.116400	3.0
std	0.723004	0.340196	0.021760	0.138015	0.147516	1.336465	0.162068	0.0
min	10.590000	12.410000	0.808100	4.899000	2.630000	1.661000	4.745000	3.0
25%	11.262500	13.002500	0.834000	5.136250	2.725500	4.048750	5.002000	3.0
50%	11.835000	13.250000	0.849350	5.224000	2.834500	4.839000	5.091500	3.0
75%	12.425000	13.470000	0.861825	5.323750	2.967000	5.467250	5.228500	3.0
max	13.370000	13.950000	0.897700	5.541000	3.232000	8.456000	5.491000	3.0

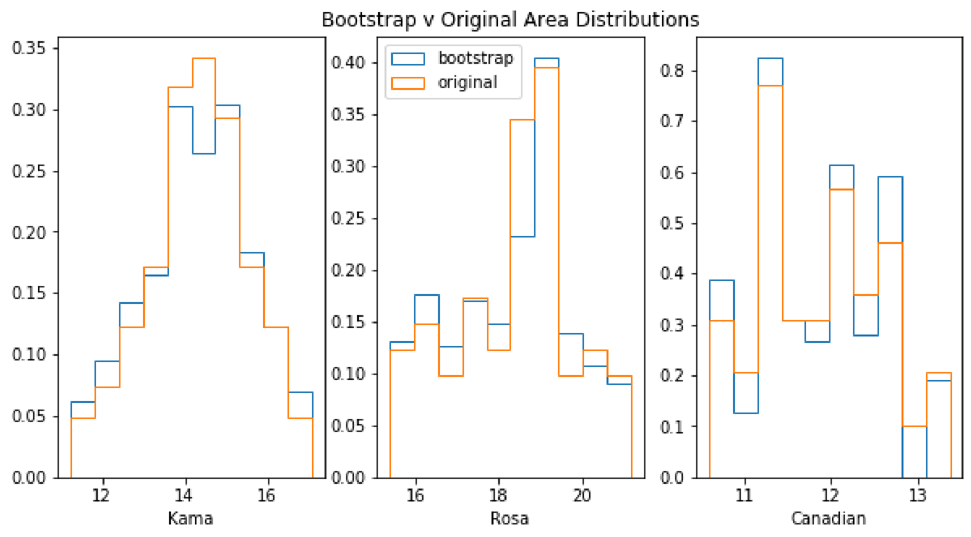
Summary Statistics: Class 3 (Canadian)

Bootstrapping

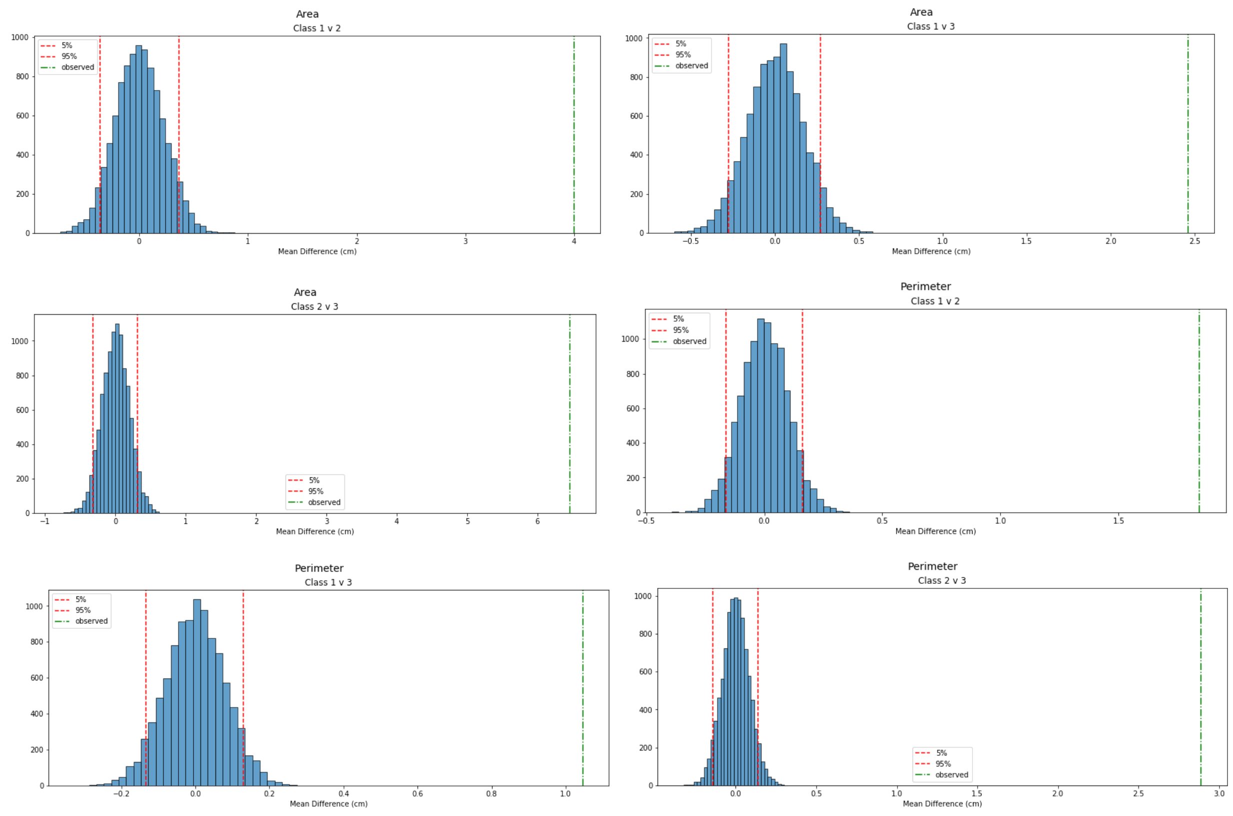
In this section we randomly sample the data for each class using pandas' .sample() method, creating 10,000 samples of the training data with replacement, all of equal size. Results are compared using histograms, then the confidence interval for the simulated sample determined using scipy.stats .percentileofscore() and the built in pandas .quantile () methods. Finally, the percentile score for each observed difference between feature classes is calculated. This value will help us determine if the geometric properties used to distinguish the seed species are truly likely to be different in scope on the scale of the full population, or if the observed difference could be a product of chance due to the tiny sample size.

The null hypothesis is that each class has the same mean for each feature (meaning they are not likely distinguishable by geometric properties on the scale of the whole population). The alternative hypothesis is that each class has a different mean for each feature (meaning they are likely distinguishable by geometric properties on the scale of the whole population).

Springboard

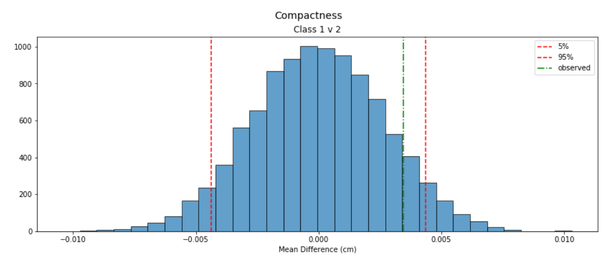


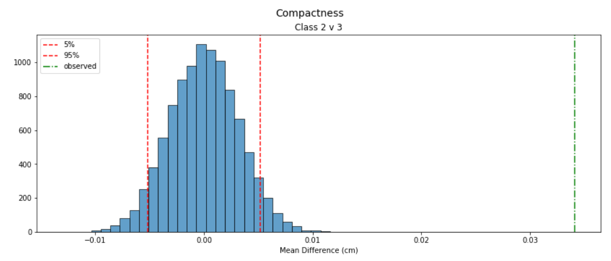
Bootstrap samples closely match original distributions

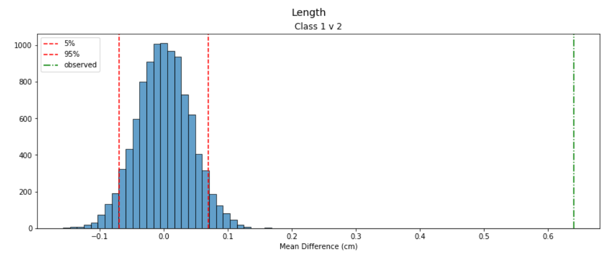


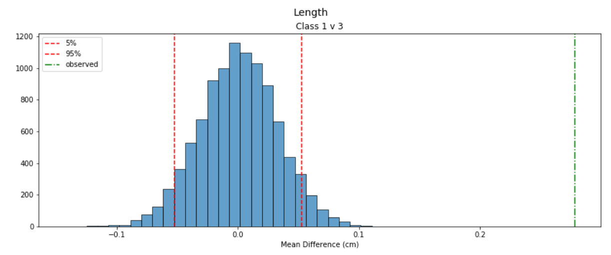
Distributions showing the 90% confidence interval and observed difference

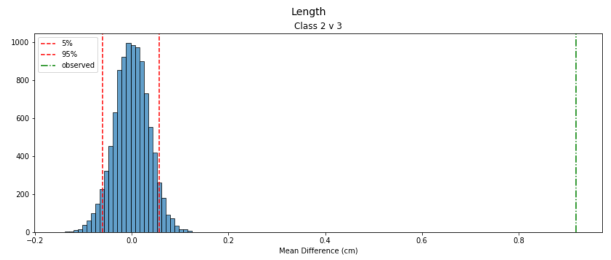
Springboard

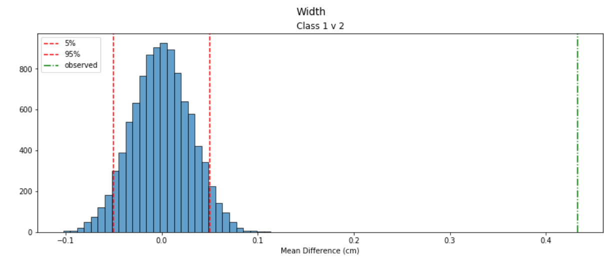


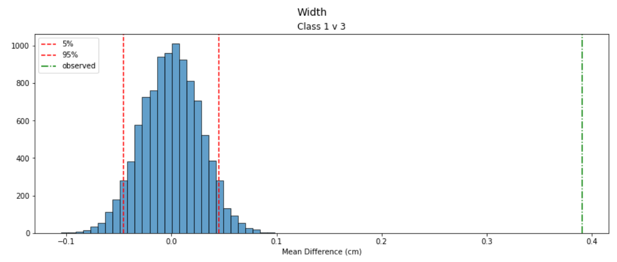


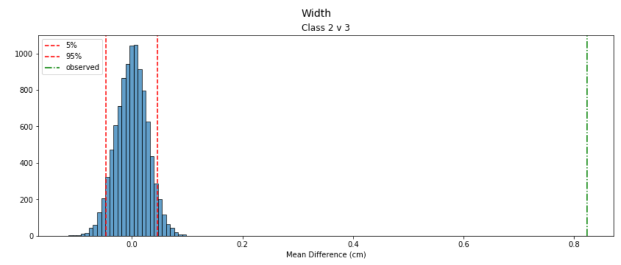


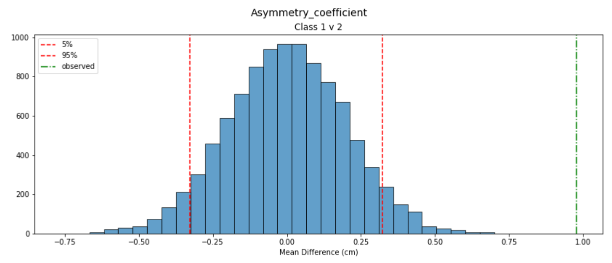


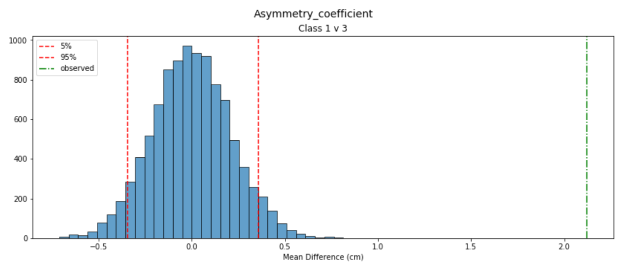




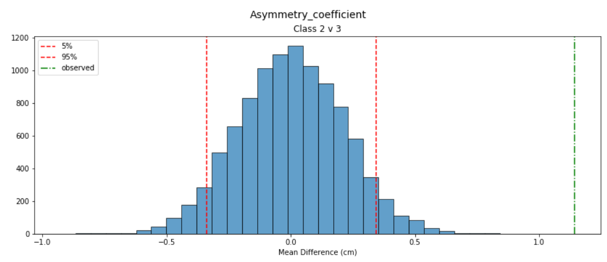


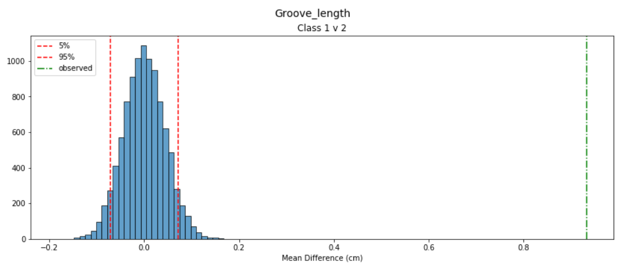


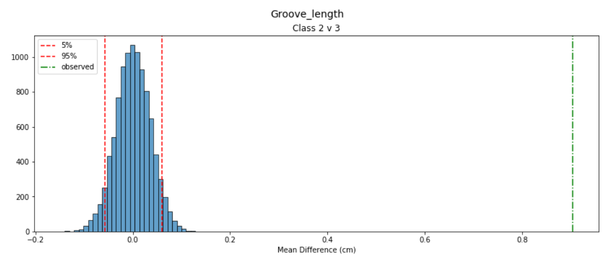


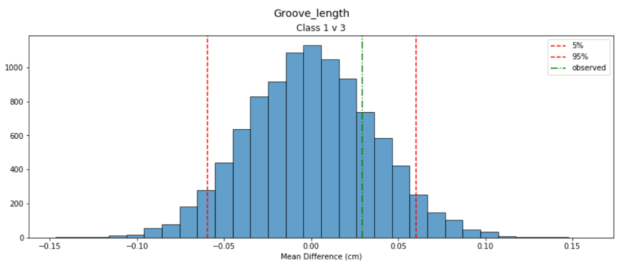


Distributions showing the 90% confidence interval and observed difference continued









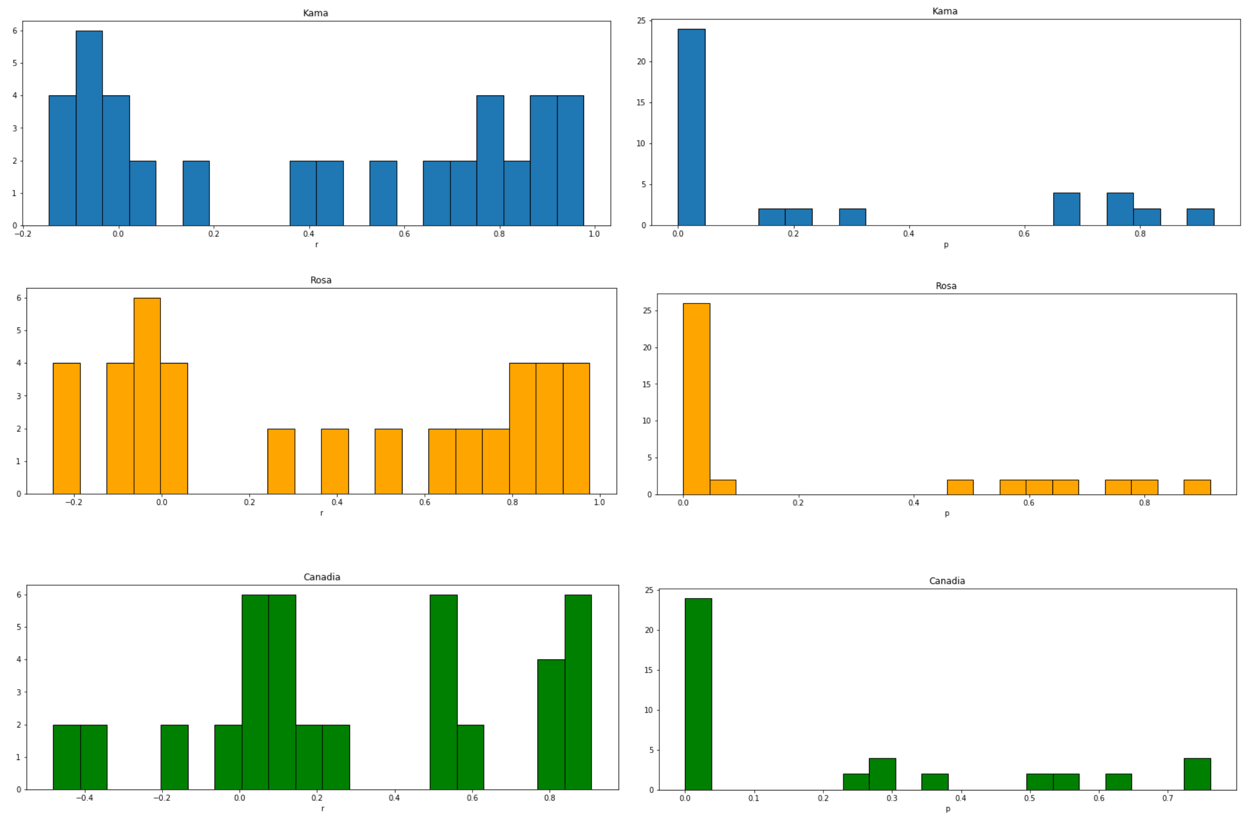
Distributions showing the 90% confidence interval and observed difference continued

The bootstrapped histograms above clearly show that most of the observed differences in seed class are significant, and not just a matter of chance. Though the number of data points for each class is small, the seeds seem to be distinct enough from one another to distinguish them by comparing features most of the time.

The minimum percentile score is the observed difference in mean groove length between classes Kama and Canadian, which was about 79%. The only other score that was not maximum is the difference in mean compactness between classes 1 and 2, which was 90%. This is not surprising, because compactness is derived from other properties by the equation $4\pi\frac{A}{P^2}$, where A is the area and P is the perimeter. Hence, we can be confident that these findings are, overall, statistically significant. We reject the null hypothesis and conclude that the differences seen in the small sample are quite likely to be representative of the populations.

Pearson Correlation Coefficient

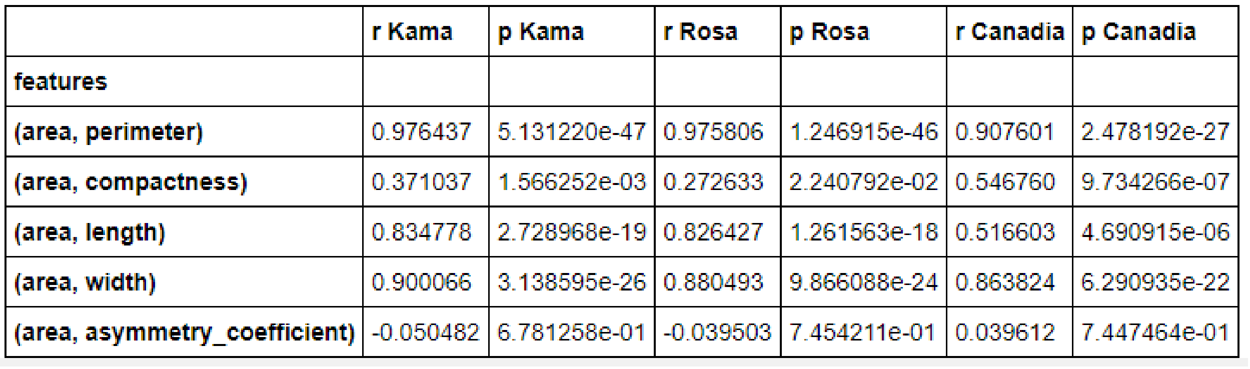
In this section we look for correlation between the features by calculating the Pearson Correlation Coefficient for each feature using scipy.stats pearsonr function. Each class was separated and histograms of the values created to quickly observe the distribution of *r* and *p* values. The histograms show noticeable spikes near zero for the p-values in each class. This means there are likely many statistically significant feature pairs that can be used for training the algorithm. A dataframe shows the values for each comparison for easy reference on the following page.



Pearson correlation coefficient and statistical significance frequencies

Composite Heat Map

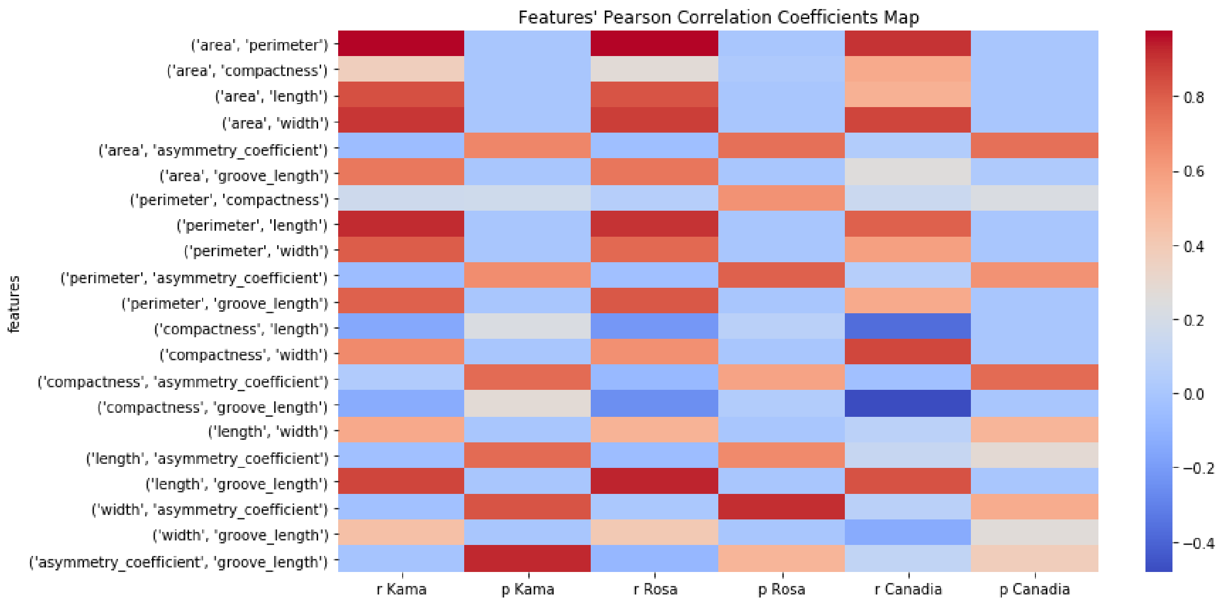
The comparison of features within each seed class can be rendered into a heat map of both r and p values for each class. This leads to a total of six heat maps, two for each class. Instead of showing each heat map individually, a composite can be constructed. The two dimensional array that constitutes the heat map of correlation coefficients or p-values for any one class can be expressed as a (21,1) dimensional vector.



	r Kama	p Kama	r Rosa	p Rosa	r Canadia	p Canadia
features						
(area, perimeter)	0.976437	5.131220e-47	0.975806	1.246915e-46	0.907601	2.478192e-27
(area, compactness)	0.371037	1.566252e-03	0.272633	2.240792e-02	0.546760	9.734266e-07
(area, length)	0.834778	2.728968e-19	0.826427	1.261563e-18	0.516603	4.690915e-06
(area, width)	0.900066	3.138595e-26	0.880493	9.866088e-24	0.863824	6.290935e-22
(area, asymmetry_coefficient)	-0.050482	6.781258e-01	-0.039503	7.454211e-01	0.039612	7.447464e-01

Pearson correlation coefficients, r, and their significance, p, values

By labeling the rows with the features being compared, and the columns with the class names, the data is reduced to a column vector. Each entry is the respective value of r or p for the features being compared within that class. Doing this for all six comparisons and then combining the column vectors into a 2-D array yields the heat map below.



Composite heat map

This gives a side by side comparison for each species of *r* and *p*. Now we can use this information to find which pairings of columns are statistically significant. A large value of Pearson's Correlation Coefficient indicates a highly linear relationship between the features. This is not desirable when training a neural network; the fixed relationship offers no new information. Thus, we can shrewdly eliminate certain pairs of columns that are highly correlated. This will save in computation cost, as well as reduce the variance in the model.

The *p*-values can be used to determine the statistical significance of the correlation coefficients. The statistically insignificant pairings can be later removed from the model for training in order to improve performance.

Feature Selection

To begin, a threshold of α = .5 is chosen; those pairs with a *p*-value below α will be deemed statistically significant. Ideally, the algorithm will be tested with varying values of alpha for optimization. The same should be done with the features that show up as statistically significant for some pairings but not for others. This overlap of significance makes it unclear which feature should be removed. To make the determination, a trial run should be performed with each possibility. The iteration with the best performance indicates the proper choice.

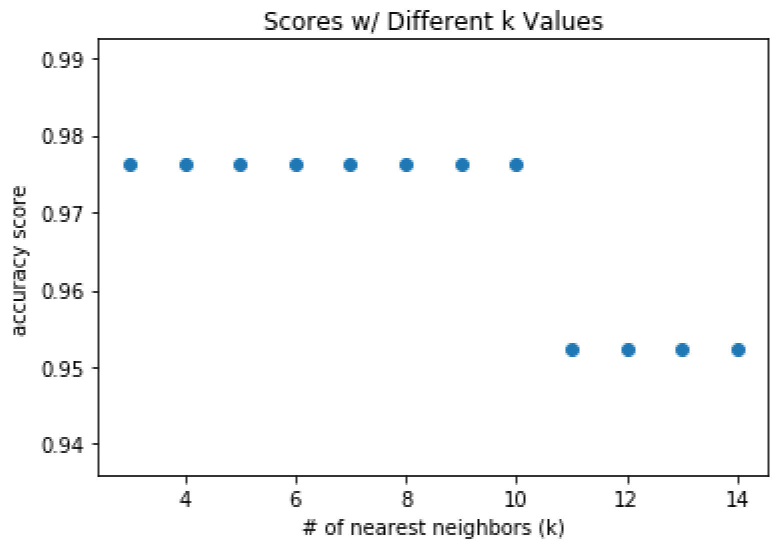
Asymmetry coefficient shows up in 5 out of 7 of the comparisons deemed insignificant. This is a strong indication that this feature can be removed from the model for training purposes. We have seen via bootstrapping that the differences in seed measurement are not flukes of random chance from the collected samples. The Pearson R and P values determined which feature pairs are statistically significant by choosing a threshold of 0.5. We can now make more informed decisions about how to proceed in constructing machine learning models to identify each seed class.

VIII. Machine Learning Models

Several machine learning algorithms were examined to determine the best suited for classification of the seeds. Performance metrics examined include precision, recall, the R.O.C. curve, and overall accuracy. The specific code used for each model can be found in the github repository https://github.com/nanodoc2020/Springboard/tree/master/Capstone-Project1/Code.

K-Nearest Neighbors

The first model tested is a simple KNN algorithm. The KNeighborsClassifier is imported from sklearn.neighbors, while the split into training and test sets is handled by sklearn.model_selection's train_test_split function with test size of 20% and random state of 11. This approach is not explored in-depth because of the overlapping nature of the distributions and the consequential possibility of misclassification. Hence, the only hyperparameter tuned was the number of nearest neighbors. Values from 3 to 15 are examined, and only the accuracy score considered. The best performance is achieved using only 5 neighbors, yielding 97.6% accuracy. The misclassified seed in the test is from Class 1 (Kama)

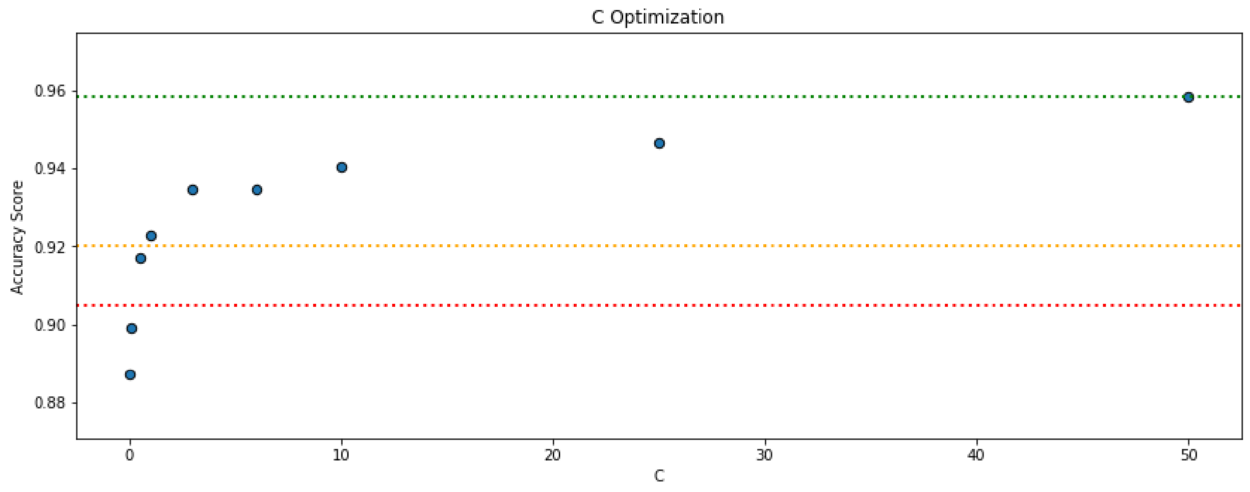


Logistic Regression

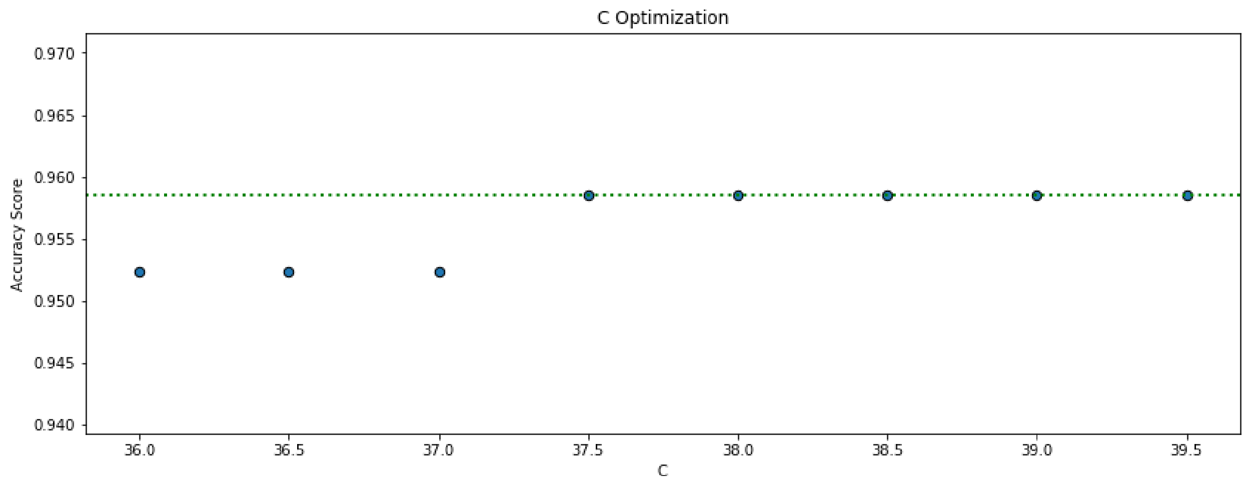
The LogisticRegression classifier from sklearn.linear_model is used here. After model initialization with 5000 iterations, the hyperparameter tuning, cross validation, and penalties are explored in pursuit of greater model performance as defined by model accuracy. In addition to accuracy, which is not always the best measure of a model's performance, the precision, recall, F1 score, and support are all calculated. Training and test sets are created using the train_test_split function. A test size of 20% is used with a random state of 42. The accuracy_score and classification_report functions from sklearn.metrics are used to determine performance. The initial test without hyperparameter tuning returns a decent result of 90.5%

Next, cross validation is used to see the effect on accuracy. Five-fold cross validation using sklearn.model_selection's KFold function is performed. Without any other tuning, just using cross validation improves the score to 92%. A modest improvement, this is still close to the original training score. At first glance it seems that the model might perform well when it encounters new data.

Regularization is applied to the model in order to decrease overfitting. This will result in better classification of new data. To find the best value of the regularization parameter C, we search over a grid by creating new classifiers with test values, score them, and store the value of C corresponding to the highest score in a list for plotting. The optimized C value is 37.5, yielding an accuracy score of 95.9%, another modest improvement. This result is again verified using the GridSearchCV function from sklearn.model_selection.



Regularization accuracy scores with previous values in red and yellow for comparison

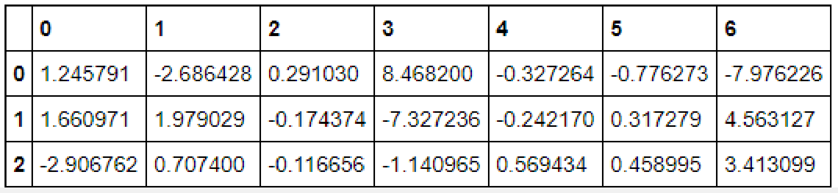


Fine tuning of the regularization parameter

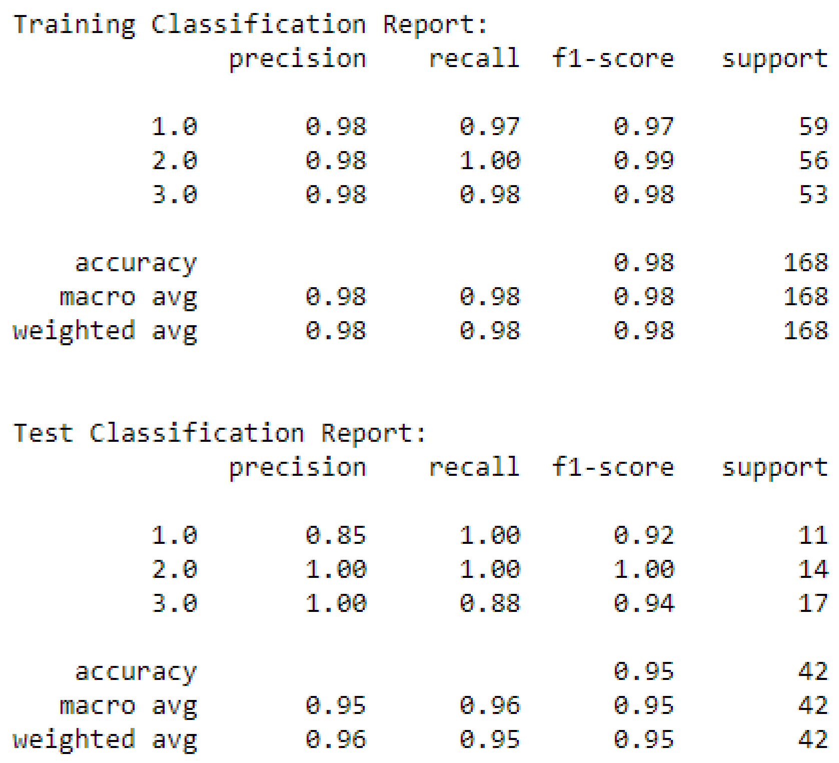
Armed with an optimized regularization parameter, different solver and penalty combinations are explored. The solver 'lbfgs' supports only L2 penalties in the LogisticRegression classifier. Other solver and penalty pairings tested include ‘saga’ and ‘elasticnet’. This combination only showed slight improvement on the test data with an l1_ratio of .3 and 7000 iterations. There is also evidence of possible overfitting with much better performance on the training data. The ‘lbfgs’ solver proved to be best suited for this task after all.

The final optimized logistic regression model using ‘lbfgs’ with L2 and regularization hyperparameter C set to 37.5 is now trained for 5000 iterations and tested. The results are promising with 98.2% and 95.2% accuracy on the training and test sets, respectively. Applying the same five fold cross validation increases the test accuracy to 96%. This is closer to the training score, indicating that variance, and thus overfitting, is reduced. Optimizing the hyperparameters and using cross validation increased the test score accuracy of the logistic regression model by 4%. A more detailed report of performance can be found using scikit-learn's classification report function, seen on the following page.

The test classification report shows very good results for both precision and recall in this model. The F1 and support scores are also quite high, prompting the suspicion of data ‘leakage’, skewing the results. This leakage can come from using highly correlated features, and thus indicates that more detailed feature selection should be explored. The final coefficients of the optimized model are shown below.



	0	1	2	3	4	5	6
0	1.245791	-2.686428	0.291030	8.468200	-0.327264	-0.776273	-7.976226
1	1.660971	1.979029	-0.174374	-7.327236	-0.242170	0.317279	4.563127
2	-2.906762	0.707400	-0.116656	-1.140965	0.569434	0.458995	3.413099



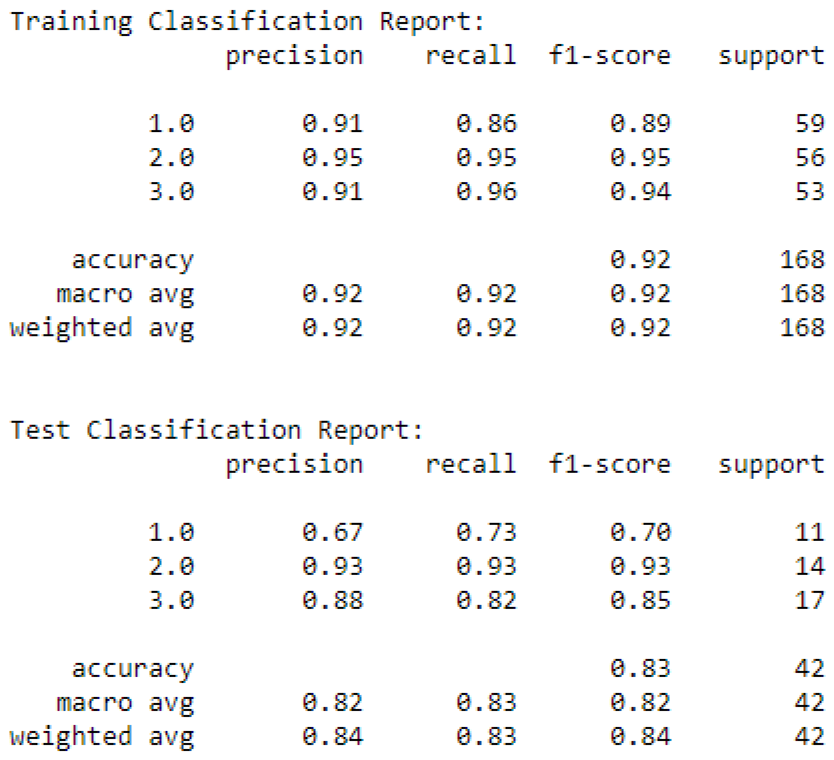
```
Training Classification Report:
              precision    recall  f1-score   support

         1.0       0.98      0.97      0.97        59
         2.0       0.98      1.00      0.99        56
         3.0       0.98      0.98      0.98        53

    accuracy                           0.98       168
   macro avg       0.98      0.98      0.98       168
weighted avg       0.98      0.98      0.98       168


Test Classification Report:
              precision    recall  f1-score   support

         1.0       0.85      1.00      0.92        11
         2.0       1.00      1.00      1.00        14
         3.0       1.00      0.88      0.94        17

    accuracy                           0.95        42
   macro avg       0.95      0.96      0.95        42
weighted avg       0.96      0.95      0.95        42
```

Logistic regression training and test classification reports

Naive Bayes

The data is continuous in each feature, prompting the choice of the GaussianNB classifier from slkearn.naive_bayes. This model is not tuned in depth, and is instantiated with default hyperparameters. Again the data is split into train and test sets with 20% test size and random state 42. The initial scores show clear overfitting, with a training score of 93.4% and test score of only 85.7%. Using five fold cross validation substantially increases the score to 92%. However, the naive Bayes approach still underperforms compared to KNN and logistic regression.

The GaussianNB classification report is very different from that of the previous model. It is interesting to note that the Bayes model has much higher precision and recall for Class 2 (Rosa). Overall it has poorer performance in every area. This classifier is certainly not the best choice for enterprise level projects.



```
Training Classification Report:
              precision    recall  f1-score   support

         1.0       0.91      0.86      0.89        59
         2.0       0.95      0.95      0.95        56
         3.0       0.91      0.96      0.94        53

    accuracy                           0.92       168
   macro avg       0.92      0.92      0.92       168
weighted avg       0.92      0.92      0.92       168


Test Classification Report:
              precision    recall  f1-score   support

         1.0       0.67      0.73      0.70        11
         2.0       0.93      0.93      0.93        14
         3.0       0.88      0.82      0.85        17

    accuracy                           0.83        42
   macro avg       0.82      0.83      0.82        42
weighted avg       0.84      0.83      0.84        42
```

GaussianNB training and test classification reports

Feature Selection

Logistic regression is revisited in this section. The very high precision and recall scores seen in the LogisticRegression report suggest that some leakage could be occurring that gives data about the class to the algorithm, causing such good results.To test this, feature pairs that have been identified as being highly correlated are first checked with one of the features removed from the dataframe. The score is then evaluated to compare against the score with the other feature removed.

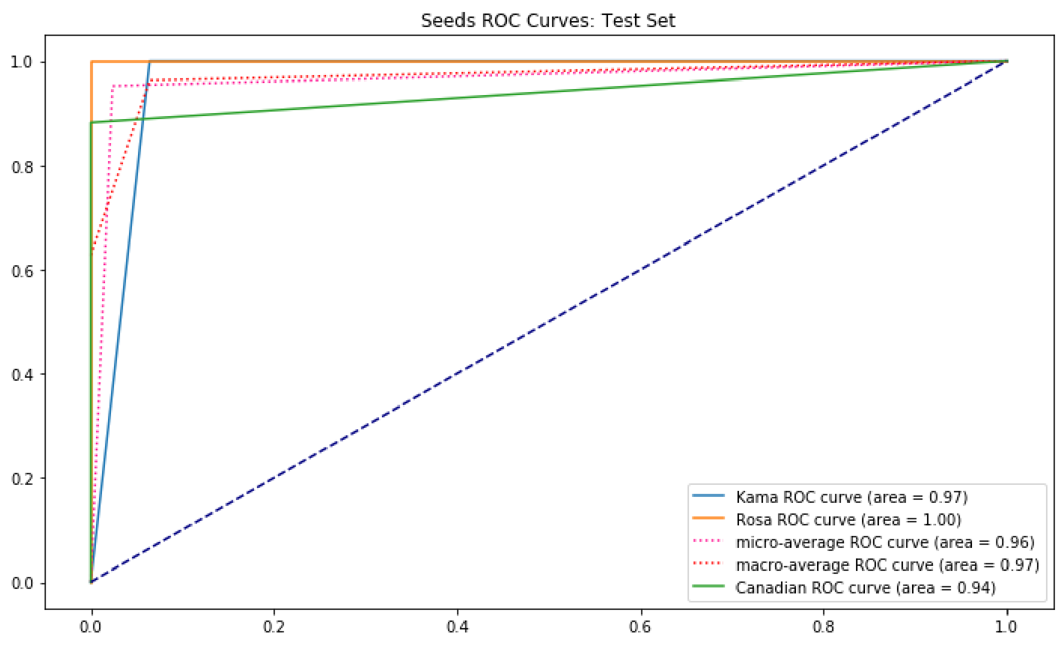
Four new classifiers are instantiated and evaluated against separate training and test sets. With asymmetry coefficient removed the test score drops steeply to 85.7% This feature clearly helps distinguish the seeds. Removing compactness had the most profound impact. The test score rose to 95.2% without cross validation. After performing five fold cv, only a slight gain to 96% is achieved.

The classification report seen on the next page, however, holds less suspicious values for the precision and recall scores. This leads to the conclusion that removing compactness from the data will decrease the variance to avoid overfitting, and help counter leakage that is inflating those scores. Next the unsupervised learning K-means algorithm is tested.

```
Training Classification Report with 'compactness' removed:

              precision    recall  f1-score   support

         1.0       0.98      0.97      0.97        59
         2.0       0.98      1.00      0.99        56
         3.0       0.98      0.98      0.98        53

    accuracy                           0.98       168
   macro avg       0.98      0.98      0.98       168
weighted avg       0.98      0.98      0.98       168

Test Classification Report with 'compactness' removed:

              precision    recall  f1-score   support

         1.0       0.85      1.00      0.92        11
         2.0       1.00      1.00      1.00        14
         3.0       1.00      0.88      0.94        17

    accuracy                           0.95        42
   macro avg       0.95      0.96      0.95        42
weighted avg       0.96      0.95      0.95        42
```

Removing compactness decreases variance and data leakage

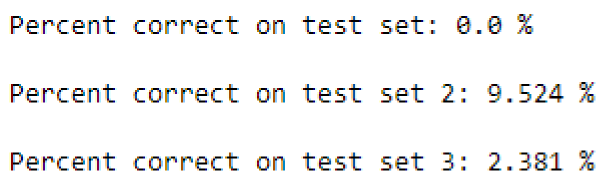
ROC Curve and AUC



K-Means

Three separate models are trained using sklearn.cluster's KMeans classifier on the datawith compactness dropped from the variables. Two extra training sets are created, one with perimeter dropped as well, and the other with area dropped. A train_test_split is performed with a test size of .2 and random state of 42 on each. After fitting and prediction, the accuracy is examined.

The K-means classifier does not perform as well as logistic regression or even naive Bayes models. Even with highly correlated features removed, the model performs poorly. This is likely due to the limited data size paired with a train_test_split that may include the outliers in the test set. It would be very difficult to correctly classify seeds whose geometric profiles overlap. The supervised k-nearest-neighbors model provides better results.



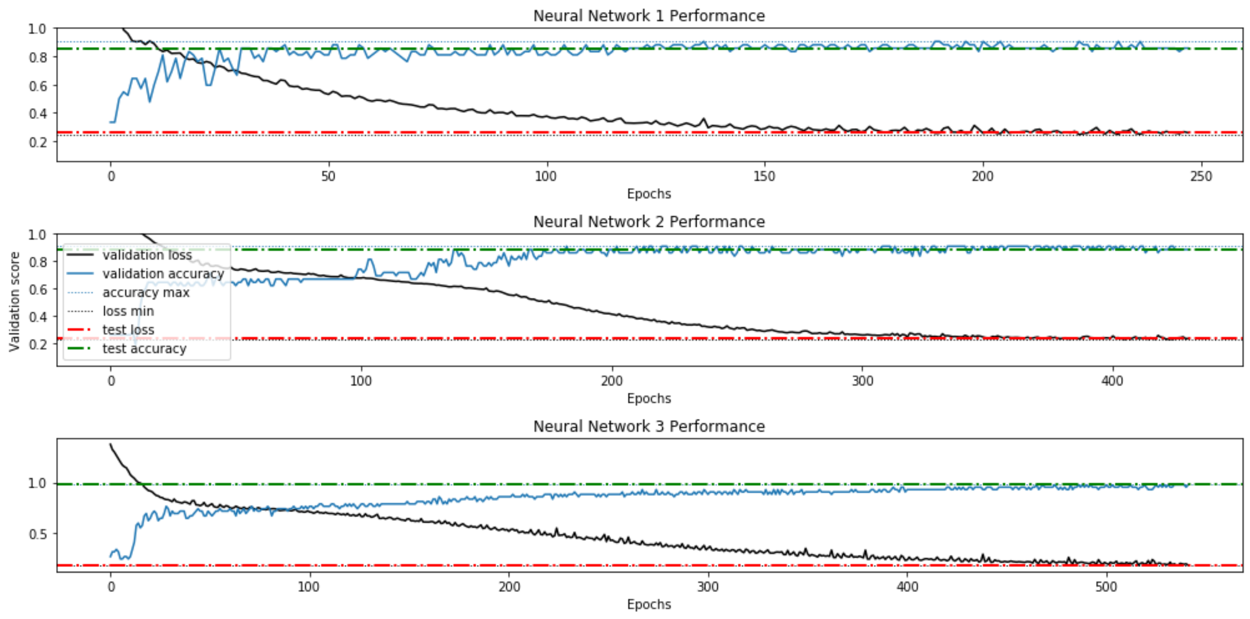
```
Percent correct on test set: 0.0 %

Percent correct on test set 2: 9.524 %

Percent correct on test set 3: 2.381 %
```

Dense Neural Network

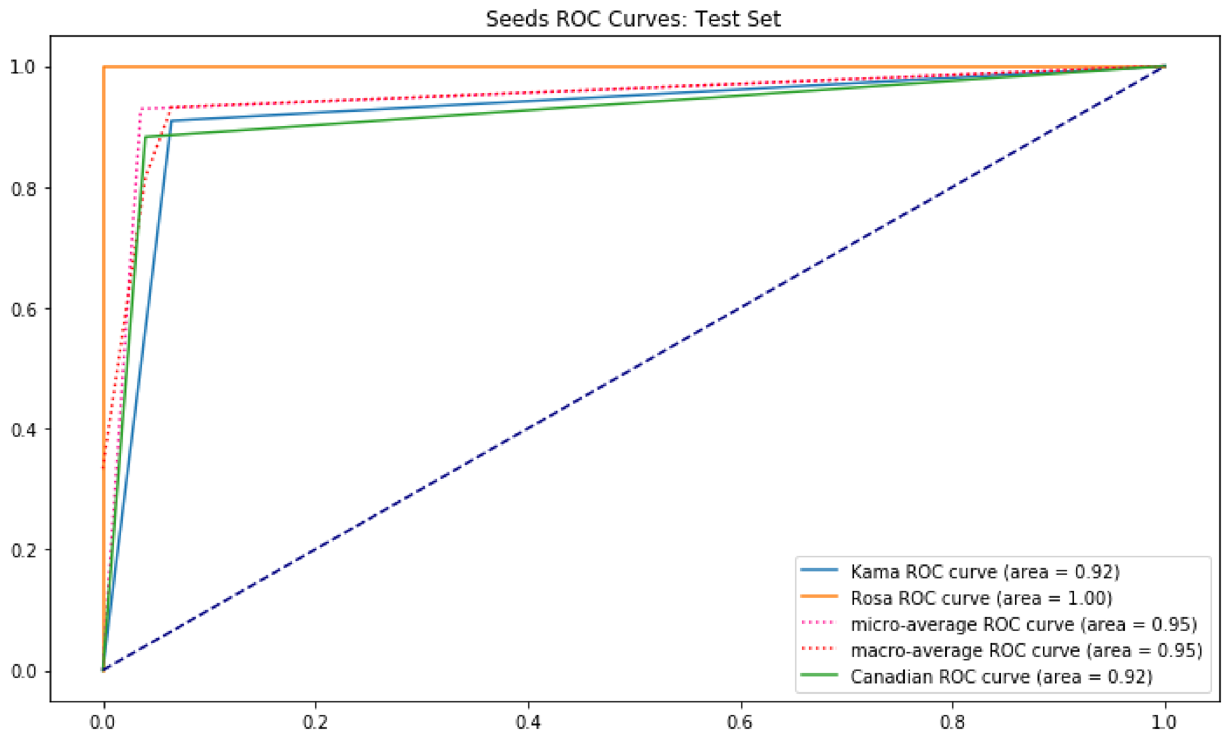
Keras, a high level interface that uses TensorFlow on the back end, is used to construct three dense neural networks. The Sequential classifier is imported from keras.models. Dense and Activation are imported from keras.layers, and from keras.callbacks the EarlyStopping function. The seeds data is split in the same manner as above with the K-Means classifier, eliminating the unnecessary variables.

There are two densely connected hidden layers in the models, which gives a total of four layers including the input and output layers. A stopping patience of 25 iterations is used, with a total of 1000 maximum training epochs available. The number of nodes in each layer is chosen as the square of the number of variables remaining after feature selection in each model. Activation in the front and hidden layers is 'relu', while a 'softmax' activation in the output layer greatly improves the results. For compiling, the 'adam' optimizer tends to work best, with 'accuracy' chosen for a metric; sparse categorical cross entropy is used for the loss function.

Of the three models, number three showed the best performance with 95.2% accuracy. This model was trained on a set with both compactness and area removed. Simplifying the model helps to reduce the variance and make it easier to predict the proper classification.

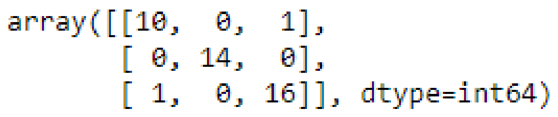


Performance of each network with different features selected



Neural network ROC curve and AUC values

The confusion matrix shows where the errors are occurring. Again it is seen that class Rosa seems to be the easiest to identify, with no misclassifications. It’s also interesting to note that a class 1 was misclassified as a 3, and the only class 3 that was incorrect was classified as class1.



```
array([[10,  0,  1],
       [ 0, 14,  0],
       [ 1,  0, 16]], dtype=int64)
```

Confusion matrix

Conclusion

The seeds dataset, though small, is rich in information. For the purposes of classification, the logistic regression model performs best, with a neural network showing similar results using feature selection to remove highly correlated variables. The four layer dense neural network constructed here is simple in comparison to many commercial architectures in use. Due to the precursory nature of this project and having tested so many different models previously, more complicated hidden layer possibilities are not explored. However, the performance of such a simple model bodes well for experimentally finding an optimum number of layers and nodes, combined with tuned activation and loss hyperparameters.The next steps in improving the model would be to use each network in a larger ensemble structure. Thus, machine learning can be used to help identify seeds through measurement alone.

This practice could help improve quality control for farmers in making sure their crops are pure, with no incorrect seed contamination. That increase in quality control extends all the way to local packaging companies selling flower seeds. Better yields with less need for weeding and chemical herbicides, saving money and man hours for the entire agriculture industry.