Note (copied from Assignment 2)*: At the outset, I had a go at tweaking certain parameters/hyperparameters for multiple classifier models in an effort to improve accuracy outcomes. For instance, following the instructions of the kNN lecture slides, I found the optimal k value for our Iris dataset to be k=13, yielding 98% accuracy. I actually happened upon a neat package that purportedly tweaks these parameters automatically, though I did not test it: https://optuna.readthedocs.io/en/stable/. However, there are multiple hyperparameters for which the python documentation was not helpful in understanding, e.g., reg_param for QuadraticDiscriminantAnalysis. I therefore submit all accuracies and confusion matrices according to the default parameter values with the exception of p=2 for kNN clustering, since p=2 corresponds with the Euclidean distance argument most suitable for petal and sepal measurements. With this in mind…*

a. The classifier that performs most optimally on the Iris dataset continues to be LDA, with an accuracy of 98% for default parameters.

b. Brief descriptions are listed below according to the classifier type.

QDA

What was striking was that LDA_accuracy > QDA_accuracy, since QDA introduces an additional flexibility of potentially different covariance matrices for different classes. In fact, the histogram plots of each feature column display different variances about the mean, and a few are certainly not Gaussian. (See **Appendix****.) If I were to tweak the QDA parameters some, then I may generate an outcome either comparable to or more improved than LDA, as the default might not have positioned boundaries most effectively. I did find 98% accuracy with reg_param=0.05, all else constant; however, as described above, I am only considering default parameter values here.

Naive Bayesian

NB has the disadvantage of assuming independence between features, which is problematic for such likely related features as petal and sepal length/width. Indeed, a dinky sepal length paired with an absurd sepal width evokes an image of a biscuit rather than the tips of an iris. Embellishment aside, NB nevertheless performs well, with an accuracy of 96%.

kNN Classifier

As mentioned above, this model is about as robust as LDA for the Iris dataset if I tweak the n_neighbor (k) parameter. I noticed that, when raising k by increments of 1 beginning at k=5, the peak of the accuracy distribution occurred at k=13, yielding an accuracy of 98%. I would argue, then, that kNN similarly did not perform as well as LDA due to k=5 not being the most appropriate n_neighbor parameter.

Linear Regression

This classifier gives an accuracy of 96.7%, rendering it the second most viable model for this dataset. Particularly compared with the higher degree polynomial regression models, this accuracy suggests that the data are more linearly-oriented. But one complication, which may contribute to the slightly lower accuracy, is that there may be outliers in the training data that affect the positioning of the linear fit, and which do not reflect the arrangement of the test data.

Polynomial Regression (2-degree, 3-degree)***

I think the most apparent cause of lesser-than accuracy scores here is that the data could simply be more linear than not. The leap from a low 2-degree score to an even lower

3-degree score, (and an even lower still 10-degree score) could be attributable to over-fitting. As the number of degrees comprising a polynomial rises, the better tuned the function becomes to deviant points. This is suggestive of the accuracy score being high; however, also bear in mind that we are fitting a regression curve to the *training data*, so the function will be most suited for that set of training data. If we transition to the test data and apply that precise function as a fit, the quirks of the training data that tailored the function will likely not translate well to the quirks of the test data. From degree-2 to degree-3 to degree-10, this is precisely the pattern that emerges.

Linear SVM

The linear nature of this classifier in particular could contribute to its inferior performance. Indeed, the sklearn documentation claims that the selected kernel function heavily dictates the accuracy score, so if the scattering of class clusters is nonlinear then the linear division line will not predict test point classes as robustly. LinearSVC in particular has trouble discriminating between Iris-virginica and Iris-Versicolor, so it could be that the distribution of these data are not divided in a linear way.

Decision Tree

This classifier fails largely due to its instability, that is, the lack of a consistent accuracy score each iteration. All else constant (including default parameters), the output predictions yielded an accuracy score of 0.953 for the first iteration, 0.927 for the second, 0.92 for the third, and 0.953 for the fourth. I then looped through 100 iterations and plotted the results.**** Thus, if the accuracy score ever exceeded the LDA score of 0.98, there can be no guarantee that the next iteration will output this exact decision tree -- or if the 0.98 decision tree was even the globally optimal decision tree.

Random Forest

While generating systematically higher accuracy scores****, this classifier likewise suffers from the weakness of instability. There is a more favorable performance, however, if I manipulate the n_estimators parameter, which encodes the number of decision trees per iteration. For instance, if I let n_estimators=1000 rather than the default value of 100, then the score is more frequently 0.96, which is the highest score that the model generated during my own tests.

Extra Trees

From the histogram plot****, Extra Trees is a marked improvement from the previous two decision tree classifiers, with the second highest score lying at 0.96. As with the case of Random Forest, it could be that the highest achievable score could be attainable if I tweak args even further. I am only utilizing defaults, alas.

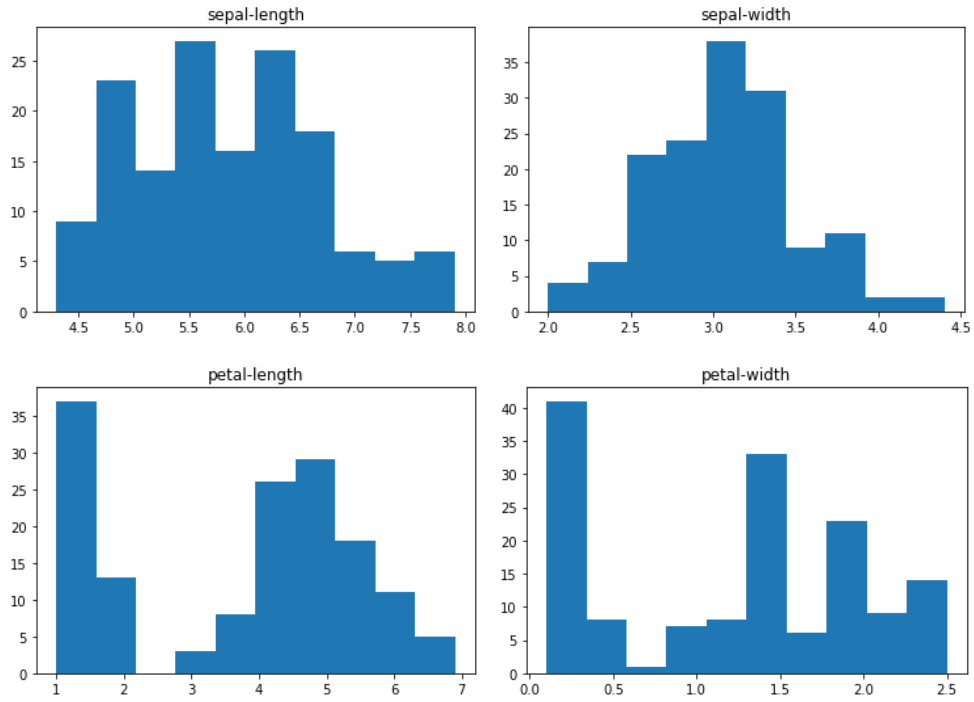
Simple Neural Network

Again, I suspect the use of ‘default’ args translated to a relatively poor performance of this classifier. In addition to inconsistency between runs*****, this inconsistency ranges from about 0.80 to 0.97 accuracy. As the class notes clarify, one of the shortcomings of this model is indeed the need to tune “a number of hyperparameters, such as the number of hidden neurons, layers, and iterations.” All I tweaked here is the maximum number of iterations to find ‘optimal’ weights and biases per hidden layer, as I occasionally received errors that the model failed to converge after arriving at the default of max_iter=200.

==========

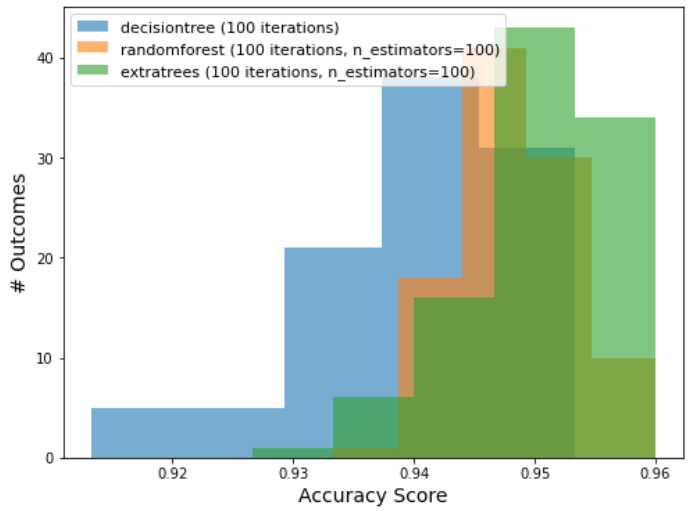
Appendix

**Histogram plots



***For both the 2-degree and 3-degree regression fits, a few model predictions did give class floats somewhat dramatically above (3.2) and below (-0.4) the specified class integers of 0, 1, and 2. I kept these for the accuracy score calculation but applied an 'if' statement to convert <0 to 0 and >2 to 2 for the confusion matrices (Dr. Johnson's recommendation).

****Histogram plot, decision tree classifiers



*****Histogram plot, simple neural network (500 outcomes)