Abhiroop Goel
3067979
EECS 658 Assignment 3

4a. Based on accuracy which model is the best one?

Based on my results, the best model is Polynomial Regression Degree 2 (Linear Regression as classifier) because it produced the highest accuracy = 0.973.

4b For each of the 11 other models, explain why you think it does not perform as well as the best one.

Below is why the other 11 models did not perform as well as the best one.

1. Naive Bayesian (GaussianNB) (accuracy = 0.947)
   Naive Bayes works by assuming the features are conditionally independent once the class is known. With the iris dataset, some measurements are related to each other, so that assumption is not perfect. Because of that, the probability estimates are slightly off for borderline samples, and I see most mistakes happening between class 1 and class 2.

2. Linear Regression (LinearRegression as classifier) (accuracy = 0.960)
   Linear regression is not a true classifier. It predicts continuous values and then I have to round them into class labels 0, 1, and 2. That rounding step can easily flip a prediction when a point is close to the boundary, which is why I still see a few class 1 and class 2 mix-ups.

3. Polynomial Regression Degree 3 (accuracy = 0.907)
   Degree 3 adds a lot of flexibility, and in my case it clearly hurt performance. The higher-degree model is more likely to fit the training fold too closely and then generalize worse to the test fold. That shows up in the confusion matrix with many more errors, especially between class 1 and class 2.

4. kNN (KNeighborsClassifier, k = 9) (accuracy = 0.953)
   kNN makes decisions based on distance. That works well when classes are clearly separated, but it struggles when two classes overlap. In iris, the hardest overlap is between class 1 and class 2, so when nearby points from different classes surround a sample, the majority vote can go the wrong way and create those errors.

5. LDA (LinearDiscriminantAnalysis) (accuracy = 0.967)
   LDA performs well, but it assumes the classes follow a Gaussian pattern and share a common covariance structure. That creates a simpler, more constrained boundary. A few samples that are close to the class 1 and class 2 dividing line still get pushed into the wrong class, which keeps it slightly below the top accuracy.

6. QDA (QuadraticDiscriminantAnalysis) (accuracy = 0.967)
   QDA is more flexible than LDA because it models a separate covariance for each class. The downside is that with a limited amount of training data per fold, those covariance estimates can be a bit unstable. That can create a few extra boundary mistakes, again mostly between class 1 and class 2.

7. SVM (LinearSVC) (accuracy = 0.940)
   This SVM setup uses a linear decision boundary. A linear separator is not always the best fit for the overlap between class 1 and class 2 in iris. Because the boundary is forced to stay linear, it cannot bend to capture some tricky points, so more class 1 and class 2 errors show up.

8. Decision Tree (DecisionTreeClassifier) (accuracy = 0.947)
   A decision tree splits the data using a sequence of if-then rules. With a small training fold, it can choose splits that look good for the training data but do not transfer as well to the test fold. That can lead to a few wrong paths for borderline cases, especially between class 1 and class 2.

9. Random Forest (RandomForestClassifier) (accuracy = 0.940)
   Random forests are usually stronger than a single tree because they average many trees, but with default settings and this simple dataset, the improvement is not guaranteed. In my run, it still confused a handful of class 2 samples as class 1, which pulled the accuracy down.

10. ExtraTrees (ExtraTreesClassifier) (accuracy = 0.940)
    ExtraTrees adds additional randomness in how splits are chosen. That randomness can help in some cases, but it can also slightly weaken the decision boundary when the dataset is small and already fairly clean. In my case, it ended up making a few extra class 1 and class 2 mistakes.

11. Neural Network (MLPClassifier) (accuracy = 0.967)
    The neural network performs well, but it depends heavily on training settings like iteration limits and internal optimization. With mostly default settings, it may not land on the best possible boundary every time. In my results, it still has a couple class 1 and class 2 mix-ups, so it ends up just below the best model.

7a) Does the program use k-fold cross-validation?
No. It uses a single train/test split (train_test_split), not k-fold cross-validation.

7b) What percentage of the dataset was used to train the DBN model?
80 percent (because test_size = 0.2, so training gets 1 - 0.2 = 0.8).

7c) How many samples are in the test set?
360 samples.

7d) How many samples are in the training set?
1437 samples.

7e) How many features are in the test set?
64 features per sample.

7f) How many features are in the training set?
64 features per sample.

7g) How many classes are there?
10 classes.

7h) List the classes
0, 1, 2, 3, 4, 5, 6, 7, 8, 9.