Daniel González Muela

Comments:
Data is always scaled
CV is always 10 folds
Use of train-validation sets and test set
More Figures Available in code

Q1: Fish Dataset

a) Training multiple classifiers

In the first question we are dealing with a dataset of size 1492 x 7, where each point corresponds to a type of fish. I began splitting the dataset in an 80% training 20% test sets, where the train set represents the available data, and the test set represents new data obtained after training the model. There is a high imbalance among the 7 classes, which I approached using stratified sampling and selecting Kappa score as the main evaluation metric. Moreover, there is a correlation of >90% between Weight, L1, L2 and L3. This suggests that PCA could help, but on the other hand there are very few features. After the initial data exploration process, I applied dimensional reduction techniques and plotted the 3PCs using TSNE, Kernel PCA and PCA. [Figure 1]

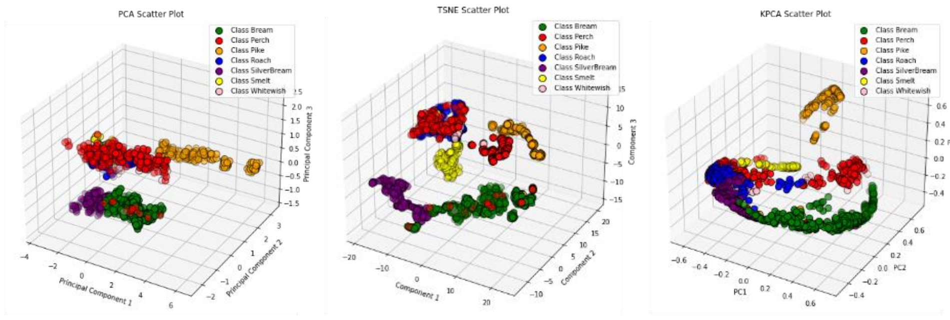

Figure 1: 3PCs PCA/TSNE/KPCA

In linear PCA, 3 components explain 97% of the data, however, it is hard to distinguish all the classes, which suggests that the data might not be linearly separable and/or that we are losing some important information using PCA. TSNE and Kernel PCA (rbf) seem to capture better the separation of some classes (like Smelt), verifying that non-linear methods should be considered. However, there are still some classes that look hard to separate even with 3 nonlinear PCs (Perch, Roach and Whitewish).

In the next step, I chose 7 classifiers that implement a wide variety of approaches: KNN (data-based), QDA (gaussian distribution of classes), SVM (Linear + Non-Linear), Kernel Logistic Regression (Linear + Non-Linear) and RF (feature focused). In my approach I splitted the training data in train-validation sets (size 0.8) and defined a Pipeline consisting on Scaling + Dimension Reduction + Classifier. Then I used GridSearchCV to tune the hyperparameters that optimize kappa score through cross-validation. The hyperparameters are chosen taking the mean and standard deviation of 10 iterations of train-validation splits. All the methods agree in not using PCA (as they chose 5 principal components) and that non-linear kernels perform better. Also, the standard deviation of the parameters is very low, meaning that we can be sure in our parameter selection. The most varying hyperparameters are the RF forest parameters, with a standard deviation of 6 for the depth and 22 for the number of estimators. After defining best model parameters, I created the same Pipelines of before (without dimension reduction) and trained them in a loop of 10 iterations of train-validation split. I present the average kappa score validation set results in a boxplot [Figure 2] and finally compute the confusion matrix of each classifier on the test set defined at the beginning [Figure 3]

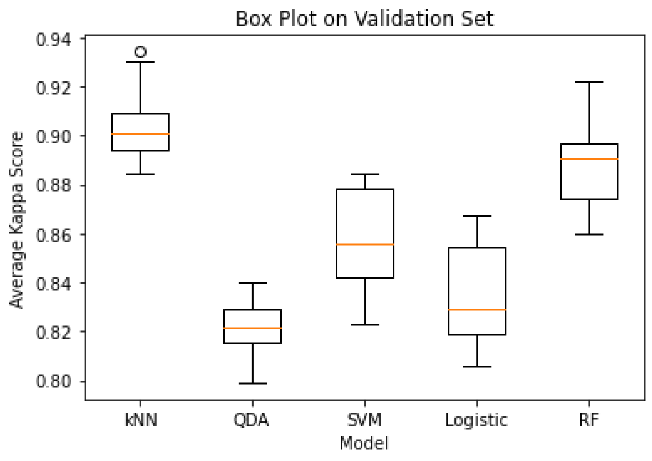

Figure 2: Box Plot of Validation Set 10 Iterations

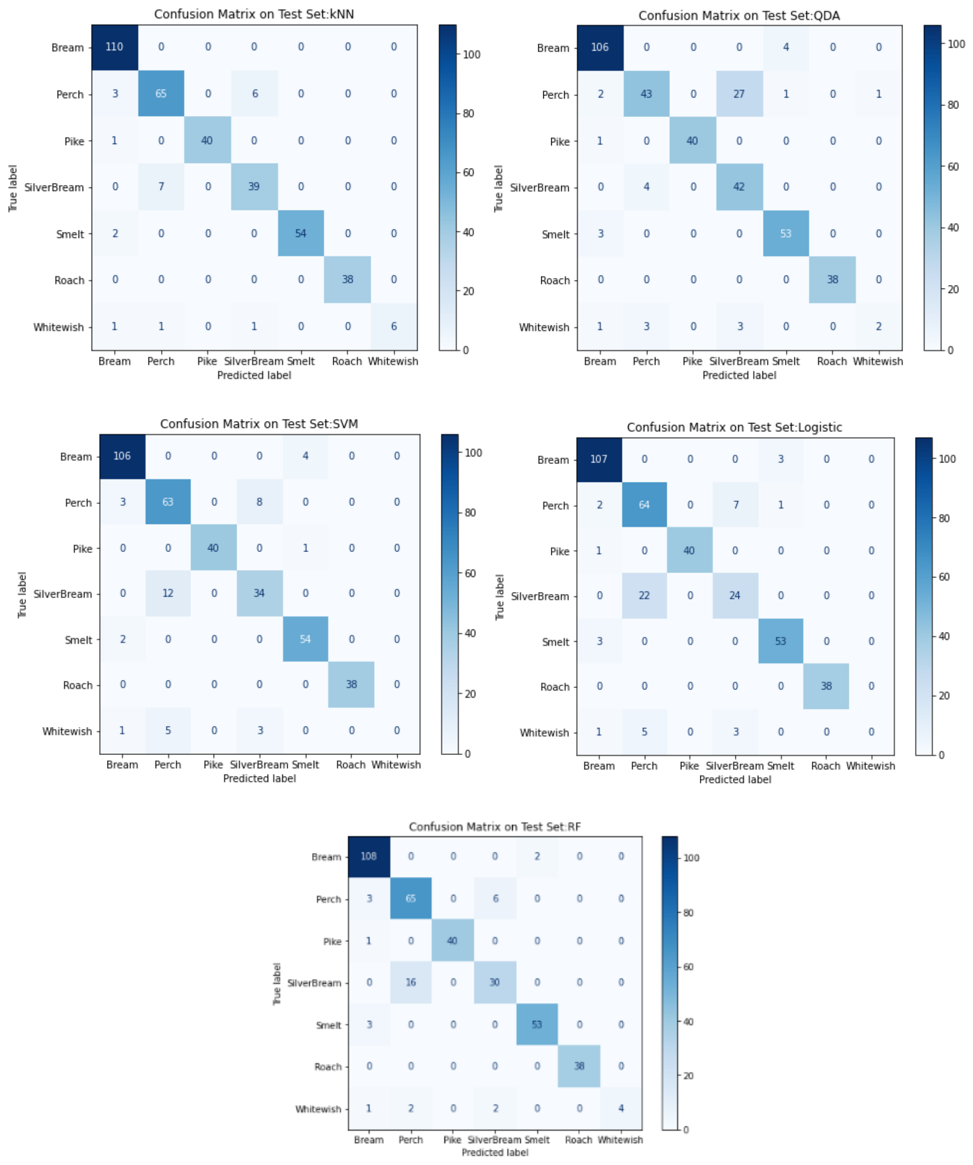

Figure 7: Confusion Matrices of Test Set

Overall, KNN, SVM(rbf) and RF are the best performing models with kappa scores of 0.93, 0.87 and 0.88 respectively. The variation of the classifiers is quite similar, being

KNN slightly more consistent than the others, and therefore coming up as the best classifier for now. Finally, when checking the confusion matrices on the test set, KNN, SVM and RF aare still the best models with scores of 0.92, 0.87 and 0.88 respectively. Note that there is a tendency across all the models to produce some missclassifitcations for predicting Breams when there are not, which makes sense because is the most common class in the dataset and some algorithms may overfit. Also, there is a constant missclasfication of “Perches that are actually SilverBreams” and “Silver Breams that are actually Perches”

b) Feature Importance + Feature Selection

Now we want to explore which features had a greater impact on the classification performance and if these features are consistent across different models. This question can be interpreted as “which features have a bigger importance when predicting data?” (Feature Importance) or “which subset of features should I select to get a classification performance?” (Feature Selection)”

Feature Importance

Given a trained model (or Pipeline) we want some score measure on its features. One of these methods is Permutation Importance, which measures the change in model’s performance when the values of a column are randomly shuffled. Given the previous pipelines, I computed 10 permutation importance scores taking the average of 10 train-validation split iterations [Figure 4]. Additionally, I included a boxplot of the 10 scores, which is not a good display to get important insights, but it shows a good stability across models and features: none of the features has a high variability, and in all the computed scores the other of feature importance is preserved [Figure 5].

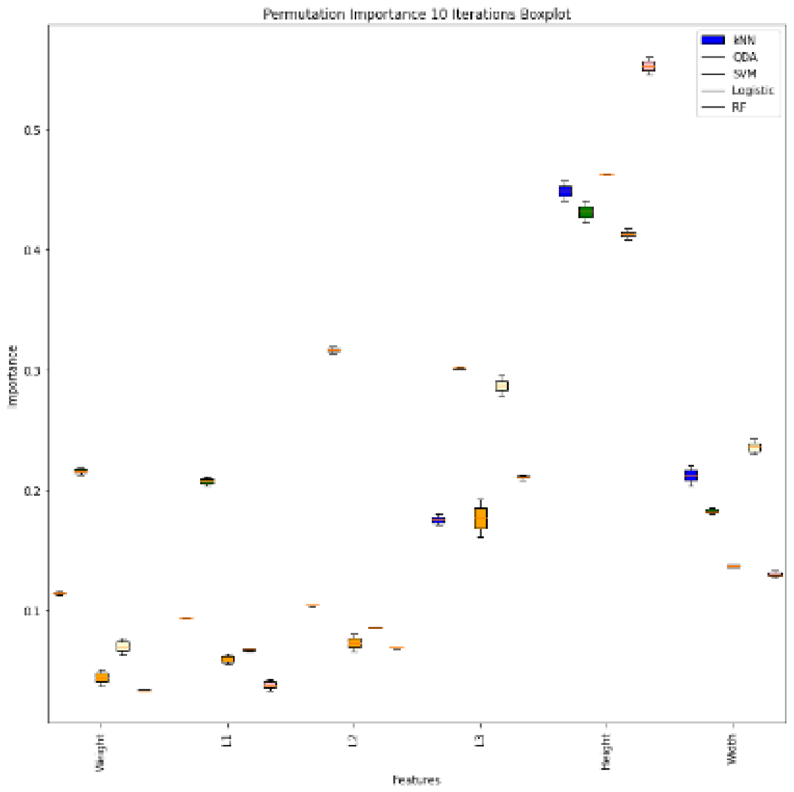

Figure 5: Box Plot of Permutation Importance 10 Iterations

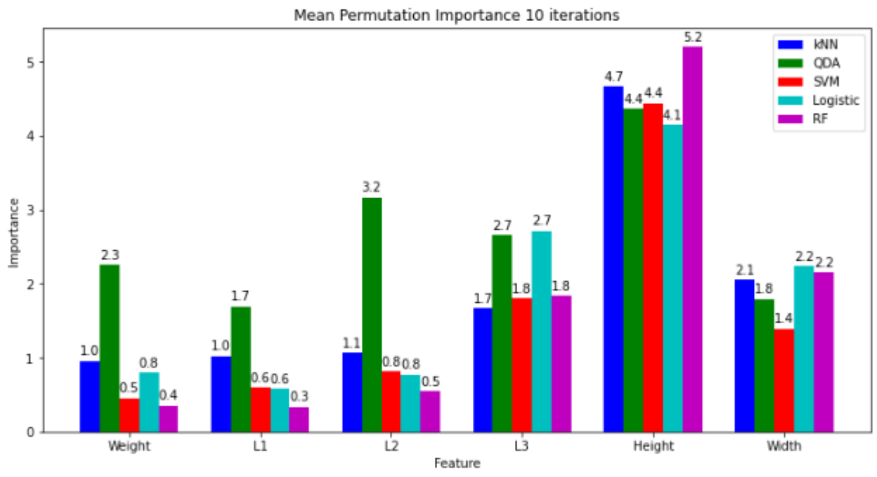

Figure 4: Mean Permutation Importance 10 Iterations

All the models agree on Height as the most important feature, followed by Width and L3, except for QDA, which prioritizes L2, but it was the worse performing model, so I would not take it very seriously. Weight, L1 and L2 are considered not important overall.

Feature Importance

Before we concluded that PCA did not have any improvement at all, potentially due to the reduced number of features. Now, given a pipeline, we want to assess if selecting a subset of features could be beneficial. In this section I only took the 3 best performing algorithms (KNN, RF and SVM). The procedure is as follows: define a pipeline of Feature Selection + Scaling + Classifier and then use GridSearchCV in the original training set to select the optimal features for each classifier. The feature selection methods that I used are Variance, F-score and Best Subset.

- Variance

Choosing features based on variance is not very useful. Despite being a big difference between the variance of some features [Figure 6], the three models agreed on keeping all the features.

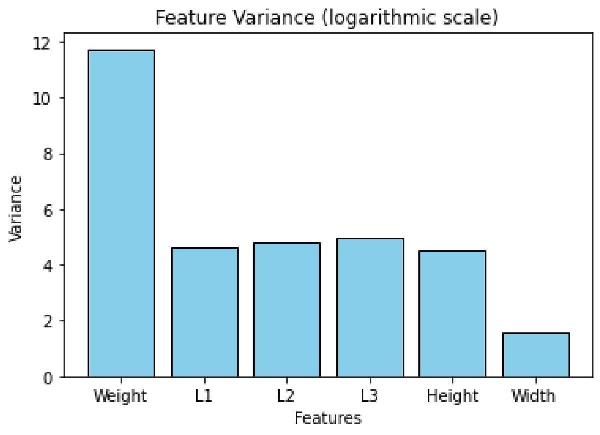

Figure 6: Variance of Features

- F-Score

F-score selects the features that correlate most with response. Using this approach did not seem to get any improvement and the three models kept selecting all the features.

- Best Subset

As there are only 5 features and the pipelines did not take a lot of time to get trained, I decided to try brute force and fit all the potential subsets of the 5 features. This resulted in:

KNN best subset: ['L1', 'L2', 'L3', 'Height', 'Width'] with a score of 0.88

SVM best subset: ['L1', 'L3', 'Height', 'Width'] with a score of 0.86

RF best subset: ['Weight', 'L1', 'L2', 'L3', 'Height', 'Width'] with a score of 0.89

As a result, when choosing the optimal subset, the three models agreed on selecting height and width. Regarding L1, L2, L3 and Weight which were the highly correlated features, each of the algorithms selected a different subset of classes, being SVM the one choosing less features (L1, L3), and RF the one choosing more (all of them), which makes sense as RF is based on the optimal partition of features. Overall, none of the optimal subsets produces a big increase with respect to the average kappa score obtained in [Figure 2], meaning that there exists an optimal set of features for each of the models, but feeding the algorithms with all the features would not be a naïve approach,

c) Adding Noisy Features

Uncorrelated features

When adding uncorrelated features, my approach was to define a number of columns to add [0, 10, 20, 30, 40, 50], and given the number of noisy columns to add, generate each column “i” as a normal distribution with mean “i” and variance “i”. Then, define the model pipeline as Scale + Classifier, compute the kappa score as the mean across 10 train-validation split iterations. I computed it on both training and validation sets. Also, I only did it for KNN, SVM and RF which were the three best methods. [Figure 7].

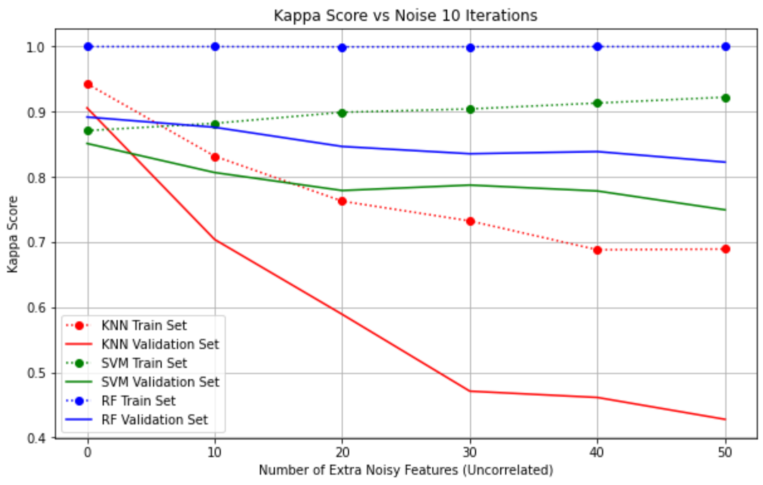

Figure 7: Kappa Score Uncorrelated Features

As a results one can see that there is always a gap between training and validation score, which makes sense, but also, the decrease of the accuracy seems to be only dependent on the model and not on if its train/validation set. Therefore, RF is the strongest model with respect to this type of noise, as the accuracy remains almost constant all the time. SVM is consistent but there is a small decrease during the first noisy features that does not make it as strong as RF. On the other hand, KNN completely breaks down as more noise is add, which can be explained by the model assumptions: distance between samples measures similarity, but in this case is not true, as the uncorrelated data can be very closely related due to randomness, specially in higher dimensions.

Correlated features

For adding correlated features, my approach consisted on: for each of the 50 generated columns "i", chose a random feature (of the original features) and generate and artificial column with a correlation of "i/50", meaning that the latest columns would have the highest correlations. The method to study the accuracies is the same than for uncorrelated features and is summarized in [Figure 8]

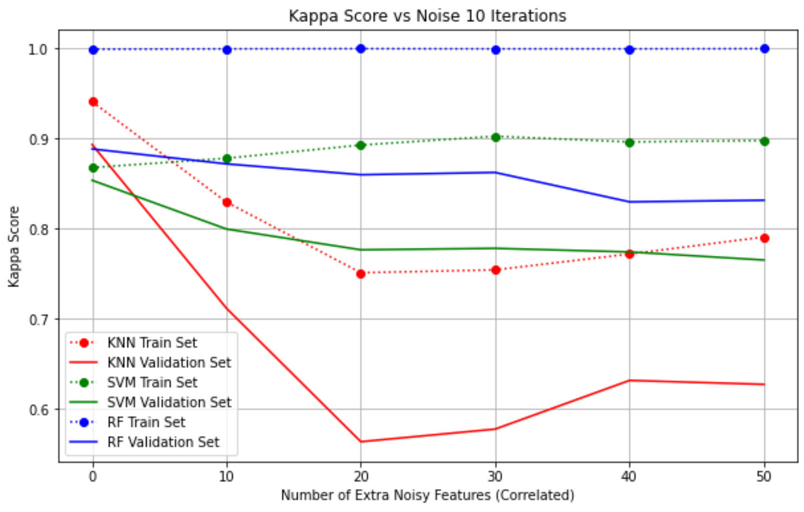

Figure 8: Kappa Score Correlated Features

The results are very similar to the ones obtained in the previous section. The only observation is that in the validation set the three methods decay slightly faster than for uncorrelated features, resulting in a slightly lower accuracy, but nothing significant as one would expect in this type of noisy data. Probably is because KNN would break down no matter what, whereas RF would stay consistent as its inner mechanics defined the best feature partitions. In the case of SVM, it is implemented with rbk kernel, which helps to reduce the potential spurious relations predicted by linear methods.

d) Didn't have time to do it ☹