Classifying Beans into 7 Groups of Dry Beans

EE 559 Course Project
Data Set: Dry Beans

Willis Ong, willisyo@usc.edu

July 26, 2024

1 Abstract

The problem is to classify 10889 grains of drybeans into 7 different classes. The dataset I used is the dry beans dataset. The machine learning methods I used are perceptron, KNN, Random Forest Classifier, and Support Vector Classifier (SVC). I compared them using the classification accuracy, along with the macro averaged F1 score and weighted averaged F1 score. The best result was the SVC with parameters, $C = 10$ and gamma $= 0.1$. Its testing classifcation accuracy was 0.9328, macro averaged F1 score is 0.9454, and weighted averaged F1 score is 0.933. I obtained it through model selection.

2 Introduction: Problem Assessment and Goals

This dataset consists of 10889 beans from 7 different classes of dry beans. It includes the following features: Area, Perimeter, Major Axis Length, Minor Axis Length, Aspect Ration, Eccentricity, Convex Area, Equiv Diameter, Extent, Solidity, Roundness, Compactness, and Shape Factors 1-4. The classes of dry beans are Barbunya, Bombay, Cali, Dermason, Horoz, Seker, and Sira.

The goal I would like to achieve is to create four ML models that could accurately classify the data into 7 different groups of dry beans.

3 Approach and Implementation

3.1 Dataset Usage

Describe the procedure you followed in the use of your dataset. You should clearly state how many data points were used for training, validation set(s), and testing. For cross validation, also specify the number of folds and number of runs.

For model selection, roughly 8711 points were used for training and 2178 points were used for validation. For cross validation I used 5 folds and 1 run.

Validation sets were used during model selection, where I could assess if the given parameters give me a high validation accuracy. They were used after training the model using the remaining training sets. Cross validation was used in nested loops when I determined multiple parameters and sequential loops when determining a single parameter.

After determining the best parameters to use during model selection, I would first train the model with the whole dataset using the best parameters found during model selection. Then, I would use the test data to estimate the final accuracy of each classifer. I would use the test dataset to determine the testing classification accuracy, and macro and weighted averaged F1 scores for each ML model.

3.2 Preprocessing

The preprocessing method I did was standardizing each feature. I standardized each feature because some features had extremely large values while other features only had small values. For example, the feature, area, had values in the 10000's while the feature, extent, had values less than 1. Since classification is very sensitive to different scale sizes, I standardized each feature, so that there would not be any sensitivity to the extremity of the values.

To standardize a feature, I relied solely on the values in the training set because I assumed that the test set is not known beforehand. Let's denote m^f_{train} and and σ^f_{train} the mean and standard deviation of a feature f computed for the training set. For a sample S, the value of feature f after the standardization is equal to:

$$v^S_f = \frac{v^S_i - m^f_{train}}{\sigma^f_{train}},$$

where v^S_i is the value of that feature f of the sample S before standardization and v^S_f is the value of that feature f of the sample S after standardization.

This transformation was also applied to the test set with the same values computed from the training set. This is because the processing that performed on the training set should be applied as is to the test set.

It is problematic to compute the mean and standard deviation of the test set and use them because it will create a transformation that different from the training set. In addition, as I mentioned previously, I assumed that the test set and its distribution are not known beforehand.

Nonetheless, as I will show later in the confusion matrices of the different classifier results, the class with the fewest number of samples (i.e., the class "BOMBAY") had a perfect precision and recall (i.e., 100% for both metrics). This led me to believe that augmenting that class with more samples yields no particular benefit as I will not improve the performance of classification of that particular class. On the other hand, I might cause problems for the other classes by introducing such samples. In brief, throughout experiments, I have note that the most misclassified class is the one with the highest number of samples. Adding more samples for that class is not beneficial, and removing some samples leads to an even worse accuracy. As such, I decided not to perform the data augmentation.

3.3 Feature engineering (if applicable)

Feature engineering was done on the fly while we were training our models. The idea behind feature engineering is to keep only features that are relevant to the classification. This is because many features are either redundant or lead to overfitting as they are very specific. As such, feature selection was opted for to identify relevant feature.

That said, due to the implementation of feature selection in Scikit learn taking a very long time to run, I relied on optimization of the hyperparameters of the different classifiers to avoid problems of overfitting; rather than discarding features, we select the hyperparameters of the classifier to be not so specific so that the classifier does not learn very specific patterns. For example, when training the random forest classifier, we made sure that the random trees are not very deep, because very deep trees lead to learning very specific combinations of features which, in return, leads to overfitting.

In section 3.5, we will explain in detail our hyperparameter tuning.

3.4 Feature dimensionality adjustment (if applicable)

Feature dimensionality adjustment was not done because when doing model selection, the highest validation accuracy is greater than 0.9, so it was unnecessary to do dimensionality adjustment.

3.5 Training, Classification or Regression, and Model Selection

3.5.1 Perceptron

The first model I tried is the perceptron. The perceptron is a non-probabilistic model. It is a linear classifier. I tried the perceptron in order to see if the dataset is linearly separable or not. I coded it using the perceptron function in Scikit learn (Version 1.4.2)[2].

The parameters I used in the model are alpha and eta. Alpha is the regularization term while eta is the learning rate. I included a regularization term, so that the perceptron won't overfit my data. To determine the best alpha and eta, I did model selection. What I did is that I picked a range of values for alpha, starting from 0.01 to 10 in powers of 10, and for eta, starting at 0.01 to 10 in powers of 10. I then used the validation set and the combinations of alpha and eta (there are 16 total combinations) to see which would give the highest validation accuracy.

I picked those ranges for alpha because if I picked a very small alpha, then the model could overfit, and if I picked a very large alpha, then the model could underfit. For eta, I chose those ranges because if eta was very small, then the model would take long to converge, and it eta was large, then the model could fail to converge.

Here are the results I got for each combination of alpha and eta.

```
Alpha = 0.01 , Eta = 0.01 , Average Validation Accuracy = 0.8703294691602202
Alpha = 0.01 , Eta = 0.1 , Average Validation Accuracy = 0.8046557359623714
Alpha = 0.01 , Eta = 1 , Average Validation Accuracy = 0.6042837866281304
Alpha = 0.01 , Eta = 10 , Average Validation Accuracy = 0.3848723591196553
Alpha = 0.1 , Eta = 0.01 , Average Validation Accuracy = 0.8211043495463256
Alpha = 0.1 , Eta = 0.1 , Average Validation Accuracy = 0.6147451041926342
Alpha = 0.1 , Eta = 1 , Average Validation Accuracy = 0.38891883717957965
Alpha = 0.1 , Eta = 10 , Average Validation Accuracy = 0.10258529673905295
Alpha = 1 , Eta = 0.01 , Average Validation Accuracy = 0.7176968245953923
Alpha = 1 , Eta = 0.1 , Average Validation Accuracy = 0.3874501055149988
Alpha = 1 , Eta = 1 , Average Validation Accuracy = 0.10258529673905295
Alpha = 1 , Eta = 10 , Average Validation Accuracy = 0.10258529673905295
Alpha = 10 , Eta = 0.01 , Average Validation Accuracy = 0.5033467425750384
Alpha = 10 , Eta = 0.1 , Average Validation Accuracy = 0.12591611188512677
Alpha = 10 , Eta = 1 , Average Validation Accuracy = 0.10258529673905295
Alpha = 10 , Eta = 10 , Average Validation Accuracy = 0.10258529673905295
Best alpha: 0.01
Best eta: 0.01
Best Average Validation Accuracy 0.8703294691602202
```

Figure 1: Perceptron Model Selection.

Based on the average validation accuracies, the best value of alpha is 0.01, and the best value of eta is 0.01.

The degrees of freedom is 16, since we have 16 features and did not do any polynomial transformation. The number of constraints is roughly 8711.

After doing model selection, we trained our model with the best parameters. The classification testing accuracy was about 0.8894. The Macro F1 score is 0.9034, and the weighted F1 score is 0.8889. Here is the confusion matrix of the model.

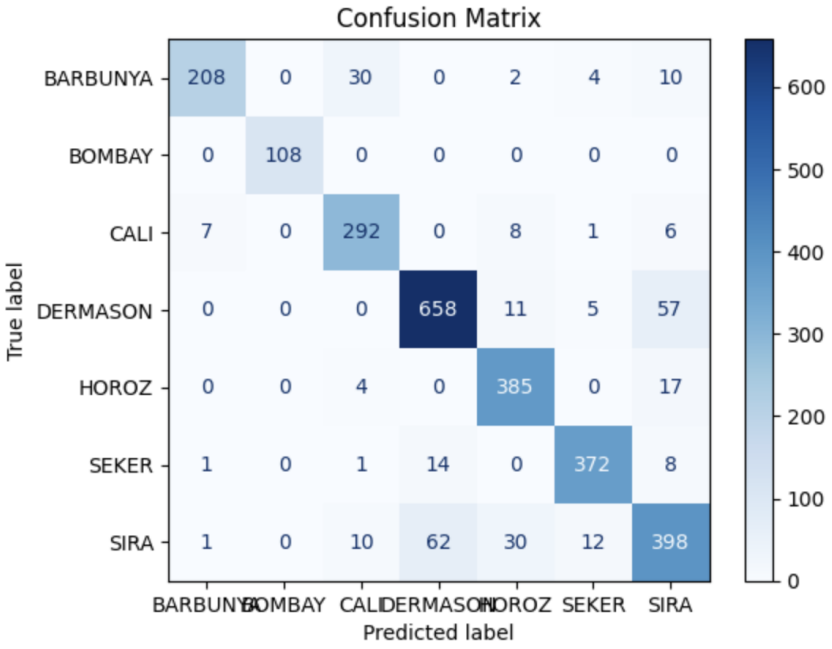

Figure 2: Confusion Matrix of Perceptron Classification.

Also, as alpha increased, the average validation accuracy decreased. This is because our model was non linearly separable, if we increased alpha, then we are underfitting the model. Here is a result of alpha vs average validation accuracies.

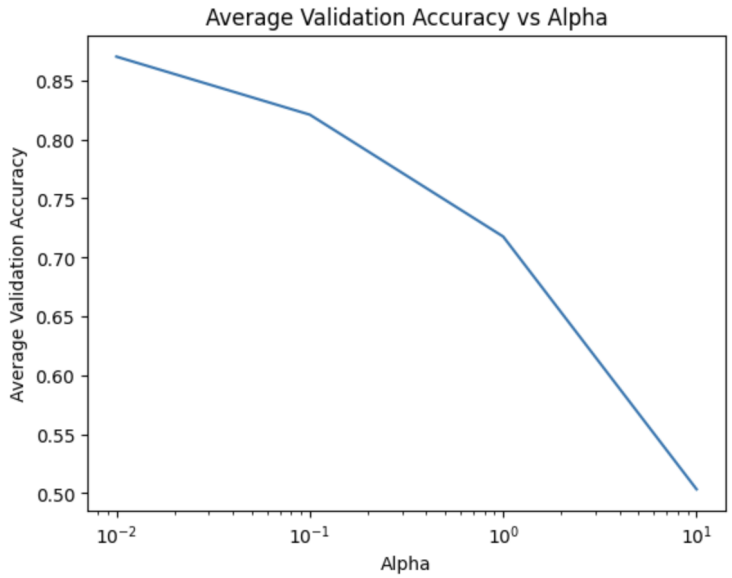

Figure 3: Validation Accuracies vs. Alpha.

Moreover, as eta increased, the validation accuracies decreased. This is because increasing eta, the learning rate by a lot can cause the model to fail to converge. Here is a result of eta vs average validation accuracies.

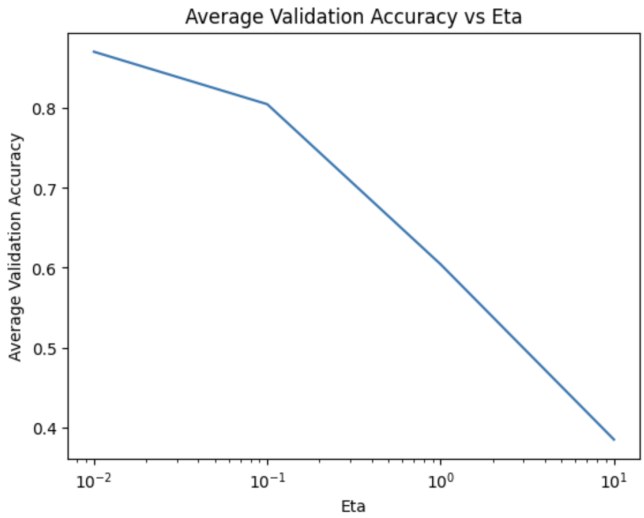

Figure 4: Validation Accuracies vs. Eta.

3.5.2 Support Vector Classifier

The next model we did was support vector classifier. A support vector classifier is a support vector machine model. I used a support vector classifier because it can give a non-linear boundary, and since we found out from the perceptron that the dataset is non linear, it would be best for us to try a non linear kernel. I coded it using the SVC function in SciKit learn (Version 1.4.2).

The parameters I used in the model are C and gamma. C is the slack variable parameter while gamma is the coefficient for the rbf kernel. To determine the best C and gamma, I did model selection. What I did is that I picked a range of values for C, starting from 0.01 to 100 in powers of 10, and for eta, starting at 0.01 to 100 in powers of 10. I then used the validation set and the combinations of alpha and eta (there are 16 total combinations) to see which would give the highest validation accuracy.

I picked those ranges for C because if I picked a very small C, then the model could underfit, and if I picked a very large C, then the model could overfit. For gamma, I chose those ranges because if gamma was very small, then the model could underfit, and it gamma was large, then the model could overfit.

Here are the results I got for each combination of C and gamma.

```
C = 0.1 , Gamma = 0.1 , Average Validation Accuracy = 0.924327587057783
C = 0.1 , Gamma = 1 , Average Validation Accuracy = 0.879878882363536
C = 0.1 , Gamma = 10 , Average Validation Accuracy = 0.2855185251268268
C = 0.1 , Gamma = 100 , Average Validation Accuracy = 0.25851891782906106
C = 1 , Gamma = 0.1 , Average Validation Accuracy = 0.9284602402696528
C = 1 , Gamma = 1 , Average Validation Accuracy = 0.9197360922879776
C = 1 , Gamma = 10 , Average Validation Accuracy = 0.6546067009089518
C = 1 , Gamma = 100 , Average Validation Accuracy = 0.2669679211626011
C = 10 , Gamma = 0.1 , Average Validation Accuracy = 0.9308479626515288
C = 10 , Gamma = 1 , Average Validation Accuracy = 0.9145006249069387
C = 10 , Gamma = 10 , Average Validation Accuracy = 0.6739833293472579
C = 10 , Gamma = 100 , Average Validation Accuracy = 0.26880463717645825
C = 100 , Gamma = 0.1 , Average Validation Accuracy = 0.9288274653664891
C = 100 , Gamma = 1 , Average Validation Accuracy = 0.8996229889828253
C = 100 , Gamma = 10 , Average Validation Accuracy = 0.673983329347258
C = 100 , Gamma = 100 , Average Validation Accuracy = 0.26880463717645825
Best gamma: 0.1
Best C: 10
Best Average Validation Accuracy: 0.9308479626515288
```

Figure 5: Model Selection using C and gamma.

Based on the average validation accuracies, the best value of C is 0.1, and the best value of gamma is 10.

The degrees of freedom is 16, since we have 16 features and did not do any polynomial transformation. The number of constraints is roughly 8711.

After doing model selection, we trained our model with the best parameters. The classification testing accuracy was about 0.9328. The Macro F1 score is 0.9454, and the weighted F1 score is 0.9330. Here is the confusion matrix of the model.

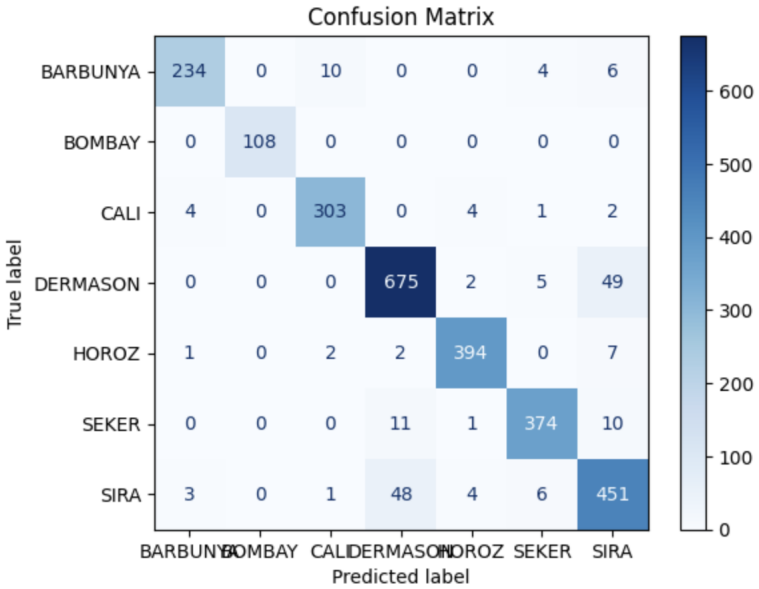

Figure 6: Confusion Matrix of SVC Classification.

Also, as C increased, the average validation accuracy increased but decreased when C = 100. This is because when we increase C, we are trying to stop overfitting from happening. However, as we increase C to about 100, the model then underfits, causing the mean validation accuracy to decrease. Here is a result of C vs average validation accuracies.

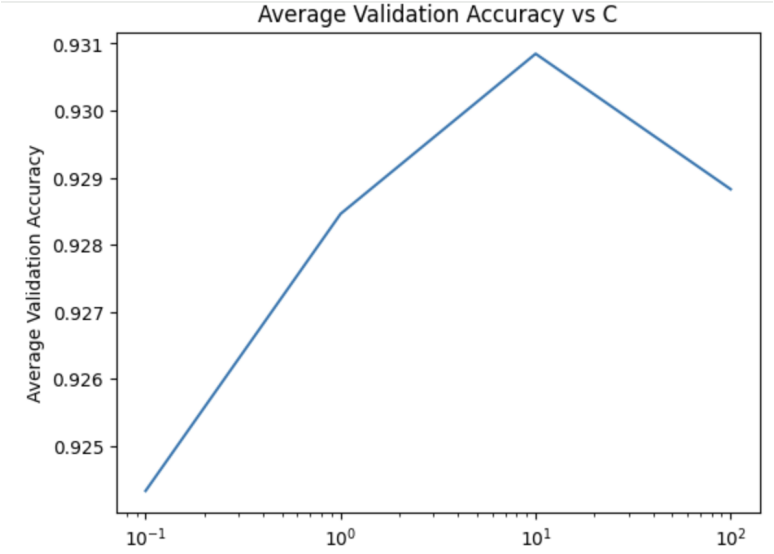

Figure 7: Validation Accuracies vs. C.

Moreover, as gamma increased, the validation accuracies decreased. This is because increasing gamma, would cause the model to overfit. Here is the result of gamma vs average validation accuracies.

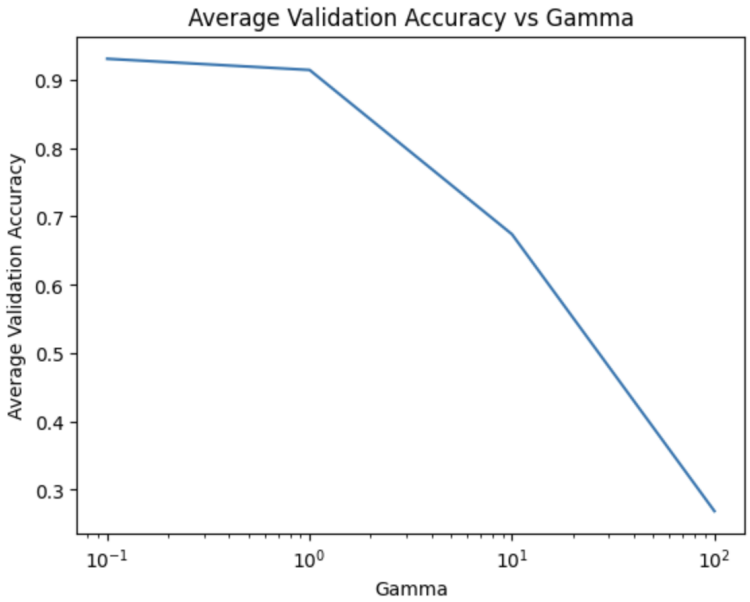

Figure 8: Validation Accuracies vs. Gamma.

3.5.3 K-Nearest Neighbors (KNN)

The third model I tried is the K Nearest Neighbors (KNN) Model. KNN model is a probablisitic model. I used KNN because I wanted to see whether the number of neighbors has any effect on the validation accuracy. I coded it using the KNN function in SciKit learn (Version 1.4.2).

The parameter I used in this model is the number of neighbors. Number of nearest neighbors is how many nearest neighbors are in a given region. To determine the best number of neighbors, I used model selection. I picked the number of neighbors from 1 to 1000 in powers to 10 to see whether or not the number of nearest neighbors had any effect on validation accuracy. I did not include 10000 in my range for number of nearest neighbors because I only used 8711 training datapoints, and the number of nearest neighbors cannot exceed the total number of training datapoints. I then used my validation sets to see which number of neighbors would give me the highest validation accuracy. Here are the results for each value of the number of nearest neighbors.

```
Number of Neighbors = 1 , Average Validation RMSE = 0.9034802444624134
Number of Neighbors = 10 , Average Validation RMSE = 0.9251543286036125
Number of Neighbors = 100 , Average Validation RMSE = 0.9176237254576922
Number of Neighbors = 1000 , Average Validation RMSE = 0.8501239057801466
Best number of neighbors: 10
Best average validation accuracy: 0.9251543286036125
```

Figure 9: Model Selection for KNN.

Based on the average validation accuracies, the best number of neighbors is 10.

The degrees of freedom is 7^{16}, since we have 16 features, 7 classes, and we do not know the distribution of the dataset. The number of constraints is roughly 8711.

After doing model selection, we trained our model with the best parameters. The classification testing accuracy was about 0.9273. The Macro F1 score is 0.9393, and the weighted F1 score is 0.9273. Here is the confusion matrix of the model.

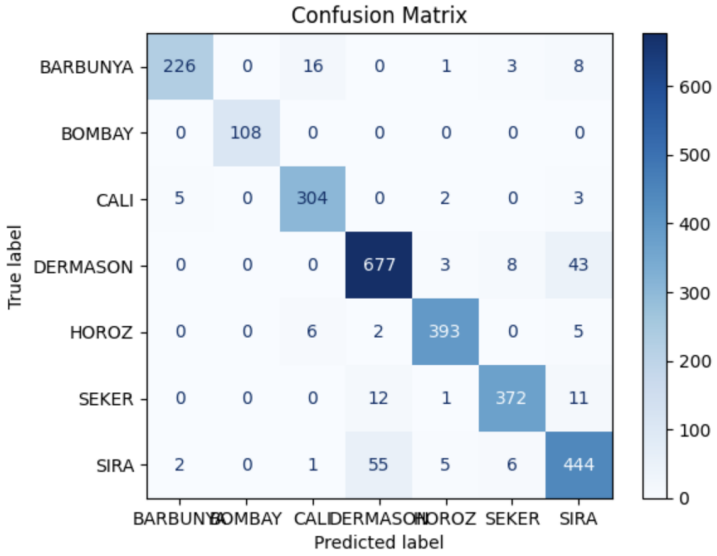

Figure 10: Confusion Matrix of KNN Classification.

As the number of neighbors increases, the model will easily underfit. This is because we are getting a poor estimate of the probability distribution of the dataset. However, as the number of neighbors decreases, the model will overfit because we are overestimating the probability of the region. Here is the result of average validation accuracy vs number of nearest neighbors.

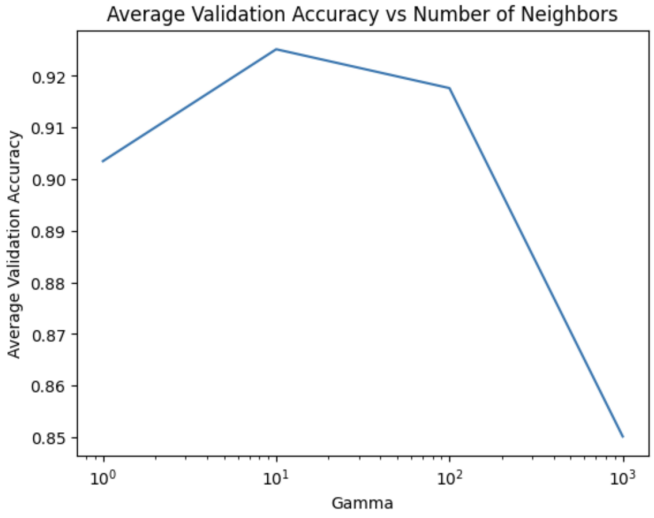

Figure 11: Validation Accuracies vs. Number of Neighbors.

3.5.4 Random Forest Classifier

The last model we experimented is random forest classifier [1]. The random forest classifier model is a non probabilistic model. Specifically, it is an ensemble of decision trees. I chose to try a random

forest because it would be interesting to see how a decision tree could have an effect on validation and testing accuracy.

The parameters I used in this model is the number of trees and maximum depth. The number of trees represents the number of decision trees. The maximum depth represents the number of splits each decision tree is allowed to make. To determine the best number of trees and maximum depth, I did model selection. What I did is that I picked a range of values for the number of trees, starting from 1 to 100 in powers of 10, and for maximum depth, starting at 1 to 100 in powers of 10. I then used the validation set and the combinations of number of trees and maximum depth (there are 9 total combinations) to see which would give the highest validation accuracy.

I picked those ranges for number of trees because if the number of trees was small, then the model could underfit, and if the number of trees was too many, then the model could overfit. For the maximum depth, I chose those ranges so that I could find the best value without overfitting or underfitting the model.

Here are the results I got for each combination of number of trees and maximum depth.

```
Tree = 1 , Max Depth = 1 , Average Validation Accuracy = 0.4055469717849139
Tree = 1 , Max Depth = 10 , Average Validation Accuracy = 0.894757066636634
Tree = 1 , Max Depth = 100 , Average Validation Accuracy = 0.8841031098558139
Tree = 10 , Max Depth = 1 , Average Validation Accuracy = 0.4992138362790219
Tree = 10 , Max Depth = 10 , Average Validation Accuracy = 0.921113418394915
Tree = 10 , Max Depth = 100 , Average Validation Accuracy = 0.9156035656181812
Tree = 100 , Max Depth = 1 , Average Validation Accuracy = 0.437136302263458
Tree = 100 , Max Depth = 10 , Average Validation Accuracy = 0.9220317764018435
Tree = 100 , Max Depth = 100 , Average Validation Accuracy = 0.9235932634061836
Best Number of Trees: 100
Best Max Depth: 100
Best Average Validation Accuracy 0.9235932634061836
```

Figure 12: Random Forest Model Selection.

Based on the average validation accuracies, the best value of the number of trees is 100, and the best value of maximum depth is 100.

The degrees of freedom is 16, since we have 16 features and did not do any polynomial transformation. The number of constraints is roughly 8711.

After doing model selection, we trained our model with the best parameters. The classification testing accuracy was about 0.9199. The Macro F1 score is 0.9321, and the weighted F1 score is 0.92. Here is the confusion matrix of the model.

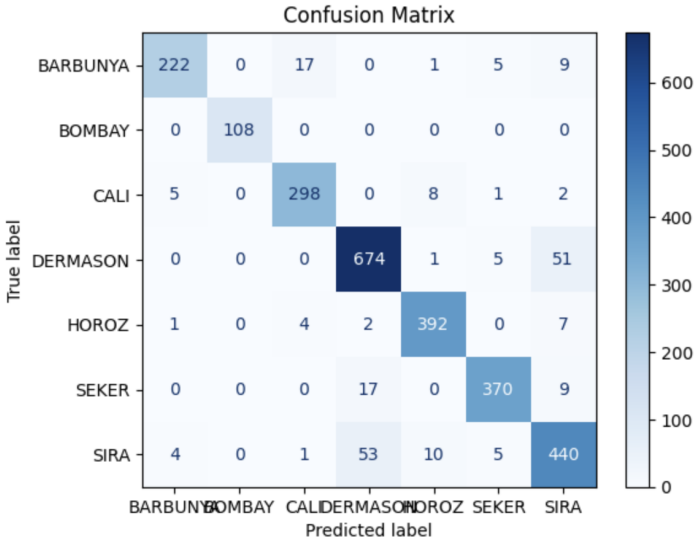

Figure 13: Confusion Matrix of Random Forest Classifier Classification.

Also, as the number of trees increased, the average validation accuracy increased. This is because the more trees we have in our model, the better the model could make predictions. Here is the result of average validation accuracy vs number of trees.

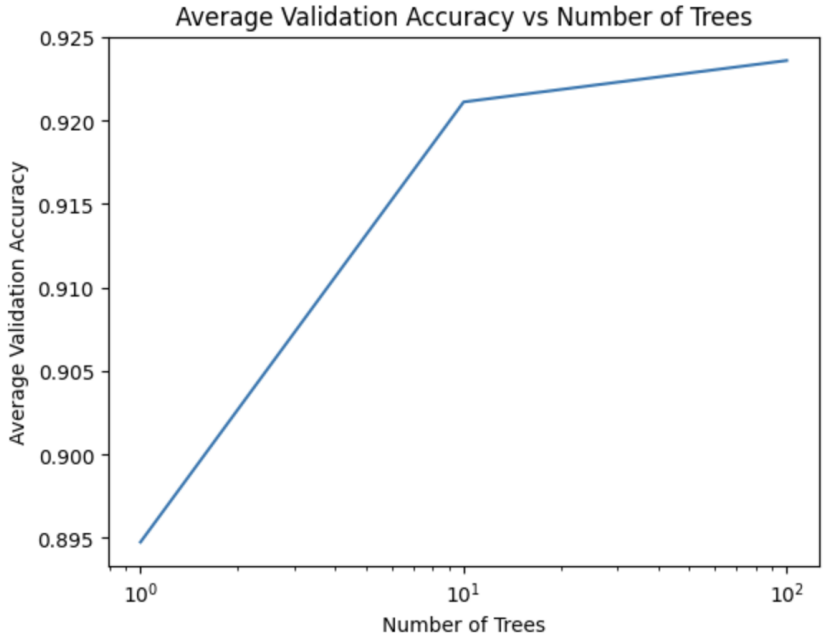

Figure 14: Validation Accuracies vs. Number of Trees.

Moreover, as gamma increased, the validation accuracies increased. This is because increasing the maximum depth increases the number of splits each decision tree could make, which could make the model predict better. However, the validation accuracies plateaued, which means that potential overfitting could happen. Here is the result of average validation accuracy vs maximum depth.

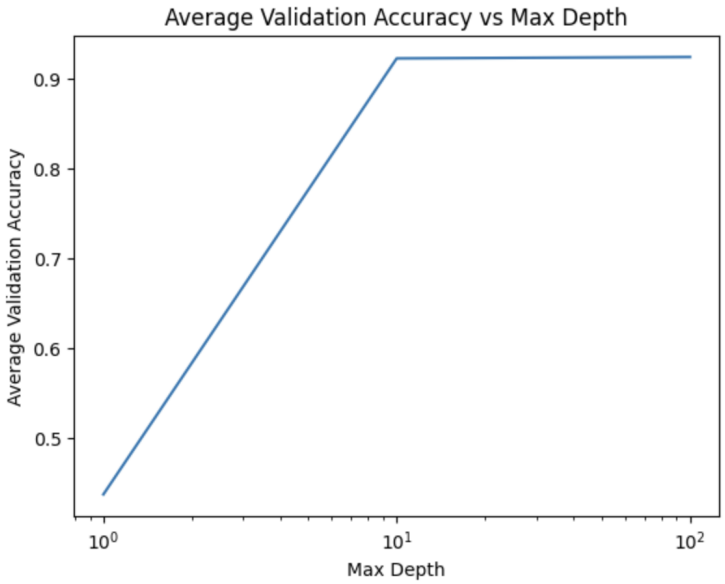

Figure 15: Validation Accuracies vs. Maximum Depth.

4 Results and Analysis: Comparison and Interpretation

For the trivial system, I did two methods: predicting based on probabilities and giving all the samples with the same label with the highest probability. What I noticed that for the former, the testing accuracy was around 0.1752 while for the latter, 0.2686. The testing accuracies in the two methods are different because in the former, we are inputting a label based on the probability of the distribution of the training datset while in the latter, we are hard coding the labels.

For the baseline classification system, I created a nearest means classifer with both unstandardizing and standardizing the data. For the nearest means classifier with unstandardized data, it performed with a testing accuracy of 0.6187 with a macro averaged F1 score of 0.6348 and weighted averaged F1 score of 0.6262. Here is the confusion matrix for the unstandardized data.

For the nearest means classifier with standardized data it performed with a testing accuracy of 0.888 with a macro averaged F1 score of 0.9047 and weighted averaged F1 score of 0.8886. Here is the confusion matrix for the standardized data.

From this, we can see that standardization plays a huge role in improving classification accuracy.

For the four models I tried, the best performing model is the SVC with parameters $C = 10$ and gamma = 0.1 with testing accuracy of 0.9328, along with the macro averaged F1 score of 0.9454 and the weighted averaged F1 score of 0.933. The worst performing model is the perceptron with parameters alpha = 0.01 and eta = 0.01 with testing accuracy of 0.8894, along with the macro averaged F1 score of 0.9034 and the weighted averaged F1 score of 0.8889. The middle two performing models are the random forest classifier with parameters number of estimators = 100 and max depth = 100 with testing accuracy of 0.9199, along with the macro averaged F1 score of 0.9321 and the weighted averaged F1 score of 0.92, and KNN with parameter, number of neighbors = 10 with testing accuracy of 0.9273, along with the macro averaged F1 score of 0.9393 and the weighted averaged F1 score of 0.9273.

For the worst performing model, because the dataset is not linearly separable, it is obvious that the accuracy did not fare as much compared to the other models done.

For the best performing model, because we know that the dataset is non linearly separable, and we used an rbf kernel, we managed to improved the testing accuracy by a lot.

The best performing system is the SVC with parameter C = 10 and gamma = 0.1. The only preprocessing step done is standardization.

Although I did not oversample the undersampled class (as according to the confusion matrix, the samples from that class were classified correctly), one thing we could do to improve the model is to oversample the undersampled class.

The d.o.f was the same, and the number of constraints were the same. However, one thing we noticed is that for a linear model, the accuracy was much lower compared to the other models that did not have a linear decision boundary. One conjecture I can make is that the dataset is non linearly separable. There is nothing I observed that is unexpected.

5 Libraries used and what you coded yourself

The libraries I used are Pandas, Matplotlib, Sci-Kit Learn, and Numpy. I coded the model selection process, finding the best parameter, plotting accuracy vs the parameters used for each ML model, a function to standardize training and testing dataset, the nearest means classifier, and the trivial solution myself. For the rest of the ML algorithms, I used Sci-Kit Learn.

6 Contributions of each team member

I did this project independently.

7 Summary and conclusions

In conclusion, I used four methods to see whether they could classify dry beans correctly: Perceptron, KNN, SVC, and Random Forest. The best performing model is the SVC with parameter $C = 10$ and gamma $= 0.1$. The worst performing model is the perceptron with parameter alpha $= 0.01$ and eta $= 0.01$.

One thing that would be interesting do to follow-on work is to see if a neural network could classify the drybeans data with almost perfect accuracy. One thing I learned while doing the project is that the dry beans dataset is non linearly separable, and classification is very sensitive to differing scale sizes of various features, so normalization is important.

8 References

[1] L. Breiman, "Random Forests," Machine Learning, Vol. 45, pp. 5–32, 2001.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.