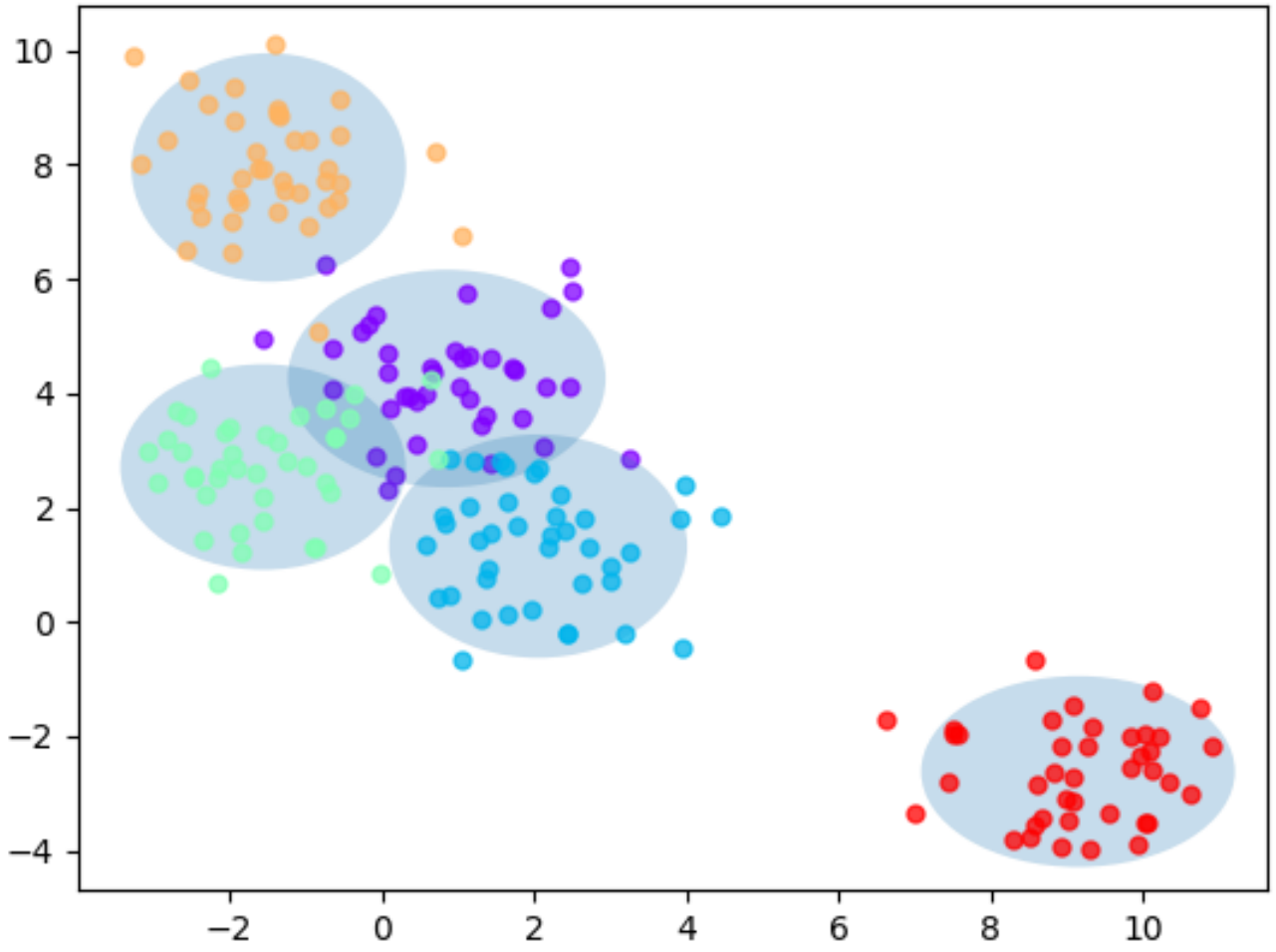
Lab 3:
Bayesian Learning and Boosting

Assignment 1

- Write a function, *mlParams(X,labels)*, that computes the ML- estimates of μ_k and Σ_k for the different classes in the dataset. X here is a set of row vectors, and labels are the class labels for each of the data points (again, ignore the W argument for now). The function should return a C × d-array *mu* that contains the class means, a C × d × d-array *sigma* that contains the class covariances. The covariance should be implemented using your own code and not by applying a library function.
- Use the provided function, *genBlobs()*, that returns Gaussian distributed data points together with class labels, to generate some test data. Compute the ML-estimates for the data and plot the 95%-confidence interval using the function *plotGaussians*.

- In this assignment equations 8 and 10 were implemented in Python.
- Then thanks to the provided functions, the datapoints and the confidence interval were plotted

Assignment 1



Assignment 2

1. Write a function *computePrior(labels)* that estimates and returns the class prior in X (ignore the W argument).
2. Write a function *classifyBayes(X,prior,mu,sigma)* that computes the discriminant function values for all classes and data points, and classifies each point to belong to the max discriminant value. The function should return a length N vector containing the predicted class value for each point.

- Equation 11 (classifyBayes) and 12 (computePrior) were implemented in Python.

Assignment 3

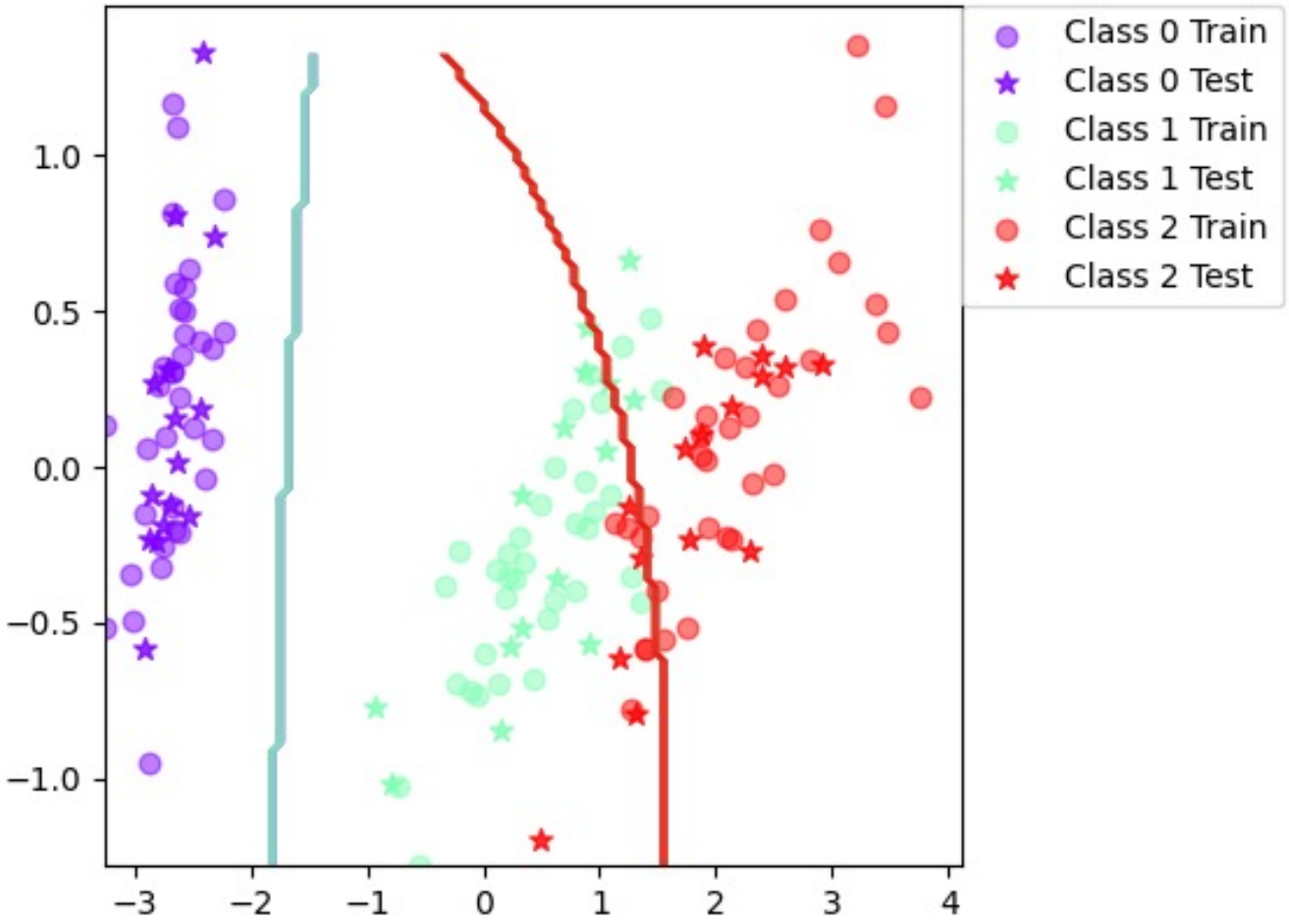
- We now have all functions we need for doing the training and classification. Use the provided function *testClassifier* to test the accuracy for the vowels and iris datasets. *testClassifier* runs a loop that does the following things:
    1. Uses the provided random partitioning function to split the dataset into a training and test dataset.
    2. Trains your classifier on the training partition.
    3. Evaluate the performance of the classifier on the test partition.
- Run *testClassifier* for the datasets and take note of the accuracies. Use *plotBoundary* to plot the decision boundary of the 2D *iris* dataset.

Assignment 3

- Bayes classifier – Iris
- Final mean classification accuracy 89 with standard deviation 4.16

Trial: 0 Accuracy 84.4
Trial: 10 Accuracy 95.6
Trial: 20 Accuracy 93.3
Trial: 30 Accuracy 86.7
Trial: 40 Accuracy 88.9
Trial: 50 Accuracy 91.1
Trial: 60 Accuracy 86.7
Trial: 70 Accuracy 91.1
Trial: 80 Accuracy 86.7
Trial: 90 Accuracy 91.1

Assignment 3



Assignment 3

- Bayes classifier – Vowel
- Final mean classification accuracy 64.7 with standard deviation 4.03

Trial: 0 Accuracy 61
Trial: 10 Accuracy 66.2
Trial: 20 Accuracy 74
Trial: 30 Accuracy 66.9
Trial: 40 Accuracy 59.7
Trial: 50 Accuracy 64.3
Trial: 60 Accuracy 66.9
Trial: 70 Accuracy 63.6
Trial: 80 Accuracy 62.3
Trial: 90 Accuracy 70.8

Assignment 3

Answer the following questions:

1. When can a feature independence assumption be reasonable and when not?
2. How does the decision boundary look for the Iris dataset? How could one improve the classification results for this scenario by changing classifier or, alternatively, manipulating the data?

Assignment 3

1. It is reasonable when features are conditionally independend or mostly independent. When it is applied on data with strongly dependent features a lot of information will be lost. Therefore, it would be wiser to choose another model in such an instance instead
2. The decision boundary seems pretty rough. There seems to be an high bias and low variance due to the boudary difficulty yo follow the datapoints. The decision boundary could be improved by switching classifier, by implementing one of the ensemble learning methods (bagging or boosting) or by manipulating the data

Assignment 4

- Extend the old *mlParams* function to *mlParams(X, labels, W)* that handles weighted instances. Again X is N × d matrix of feature vectors, labels a length N vector containing the corresponding labels and W is a N × 1 matrix of weights. The signature should look like
- *def mlParams(X, labels, W)*

  ...

  return mu, sigma
- Here, the types of return parameters mu and sigma are identical to the old *mlParams*. The function computes the maximum posterior parameters μk and Σk for a dataset D according to Equations 13-14. Assume the usual data format for the first two parameters. Test your function *mlParams(X, labels, W)*, for a uniform weight vector with ω = 1/N. The MAP parameters should be identical to those obtained with the previous version of *mlParams*.

- Equations 13 and 14 were implemented in Python

Assignment 5

1) Modify *computePrior* to have the signature *computePrior(labels, W)*, taking the boosting weights ω into account. We can look at the weights as taking a particular training point xi into account ωi times. So if previously there was Nk points in a particular class, we should now think about "how many times we count" each point. Note that the prior probabilities should still sum to one.

2) Implement the Adaboost algorithm and apply it to the Bayes classifier. Design a function trainBoost(base classifier, X, labels, T) that generates a set of boosted hypotheses, where the parameter T determines the number of hypotheses. Use the modified computePrior(labels, W). The signature in Python should look like

```
def trainBoost(base_classifier, X, labels, T):
    ...
    return classifiers, alphas
```

Note that a new classifier of type base classifier can be trained by calling *new classifier = base classifier.trainClassifier(X, y, W)*, which can then be used for classification by calling new *classifier.classify(X).*

3) Design a function

```
def classifyBoost(X, classifiers, alphas, Nclasses):
    ...
    return yPred
```

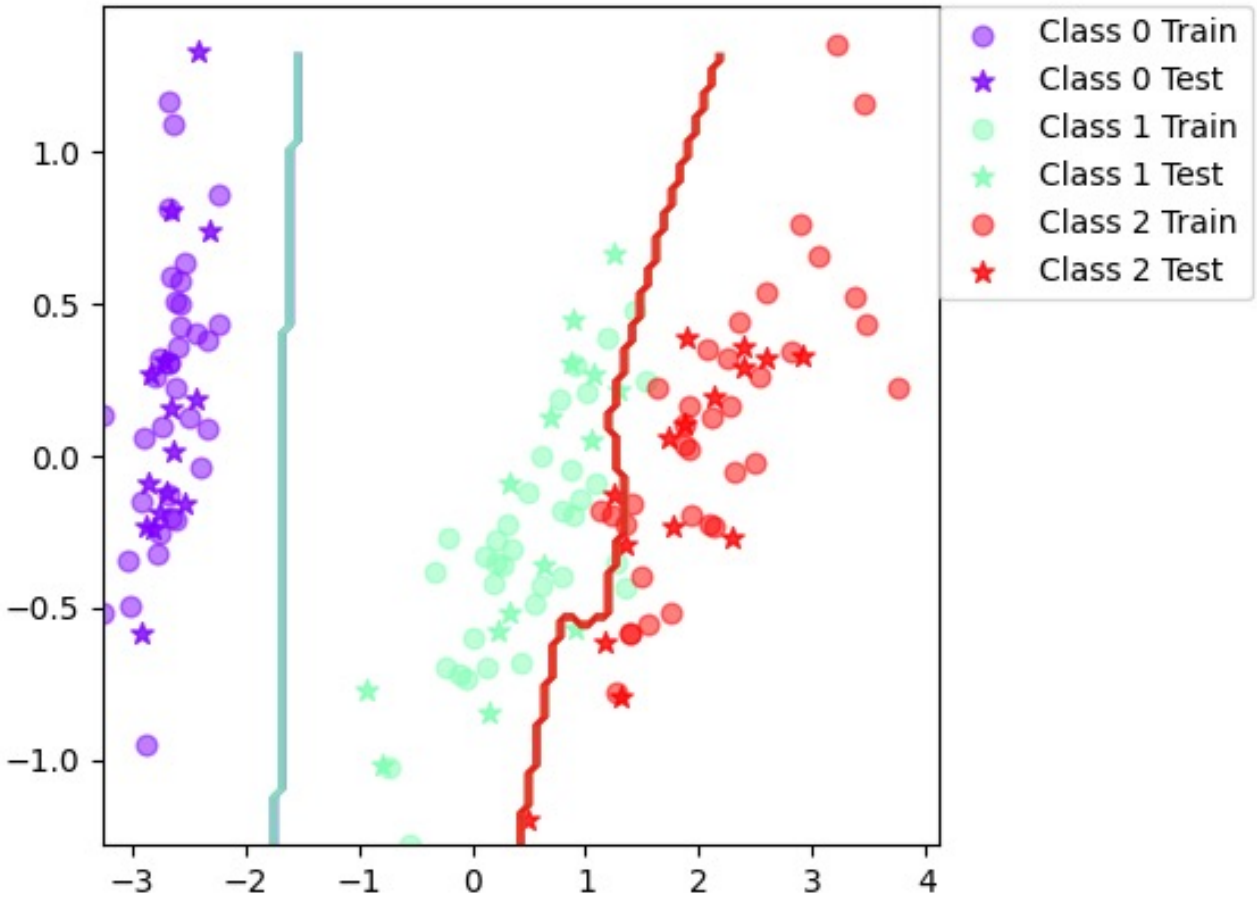
that classifies the instances in data by means of the aggregated boosted classifier according to Equation 15. The resulting classifications are returned in the vector *yPred*.

Assignment 5

- Bayes classifier – Iris Boost
- Final mean classification accuracy 94.7 with standard deviation 2.82
- One of the individual classifiers predicted everything correctly, therefore *epsilon == 0* which caused an error when *np.log(epsilon)* was called. *Alpha* was set to 2 as that is roughly the value that the best classifiers obtained. Occurred due to the small and simple dataset of iris.

Trial: 0 Accuracy 95.6
Trial: 10 Accuracy 100
Trial: 20 Accuracy 93.3
Trial: 30 Accuracy 91.1
Trial: 40 Accuracy 97.8
Trial: 50 Accuracy 93.3
Trial: 60 Accuracy 93.3
Trial: 70 Accuracy 97.8
Trial: 80 Accuracy 95.6
Trial: 90 Accuracy 93.3

Assignment 5



Assignment 5

- Bayes classifier – Vowel Boost
- Final mean classification accuracy 80.2 with standard deviation 3.52

Trial: 0 Accuracy 76.6
Trial: 10 Accuracy 86.4
Trial: 20 Accuracy 83.1
Trial: 30 Accuracy 80.5
Trial: 40 Accuracy 72.7
Trial: 50 Accuracy 76
Trial: 60 Accuracy 81.8
Trial: 70 Accuracy 82.5
Trial: 80 Accuracy 79.9
Trial: 90 Accuracy 83.1

Assignment 5

Answer the following questions:

1. Is there any improvement in classification accuracy? Why / Why not?
2. Plot the decision boundary on *iris* and compare it with that od the basic. What differences do you notice? Is the boundary of the boosted version more complex?
3. Can we make up for not using more advanced model in the basic classifier by using boosting?

Assignment 5

- Iris dataset:
  Final mean classification accuracy **89** with standard deviation **4.16**
  Final mean classification accuracy **94.7** with standard deviation **2.82** (boost)

- Vowel dataset:
  Final mean classification accuracy **64.7** with standard deviation **4.03**
  Final mean classification accuracy **80.2** with standard deviation **3.52** (boost)

- In both datasets the mean classification accuracy improved when utilising boosting. This is due to the prediction after boosting is a weighted average of the predictions from the multiple classifiers. This improves the accuracy of basic model and can make up for some of the information lost (e.g. dependent features) when not utilising a more advanced model. Boosting reduces bias. This creates a more complex boundary which can follow the datapoints better and yields an improved result.

Assignment 6

- We have implemented a class DecisionTreeClassifier based upon skLearns decision tree classifier. The skLearn implementation is similar to the one used in the first lab, however, here the values are continuous and we use the default Gini index to compute the split.

- Test the decision tree classifier on the vowels and iris data sets. Repeat but now by passing it as an argument to the BoostClassifier object. Answer questions 1-3 in assignment 5 for the decision tree.

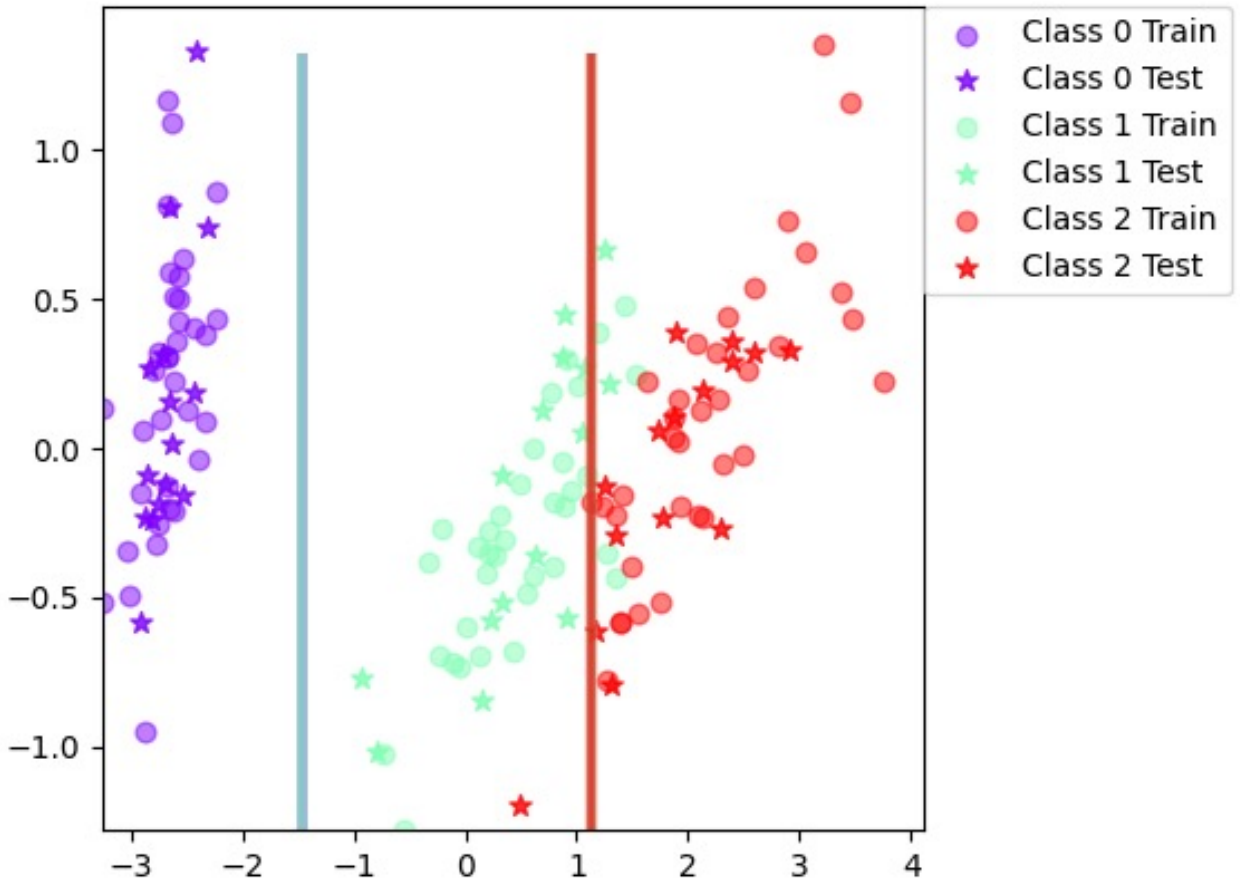
Assignment 6

- Decision Tree – Iris

- Final mean classification accuracy 92.4 with standard deviation 3.71

Trial: 0 Accuracy 95.6
Trial: 10 Accuracy 100
Trial: 20 Accuracy 91.1
Trial: 30 Accuracy 91.1
Trial: 40 Accuracy 93.3
Trial: 50 Accuracy 91.1
Trial: 60 Accuracy 88.9
Trial: 70 Accuracy 88.9
Trial: 80 Accuracy 93.3
Trial: 90 Accuracy 88.9

Assignment 6



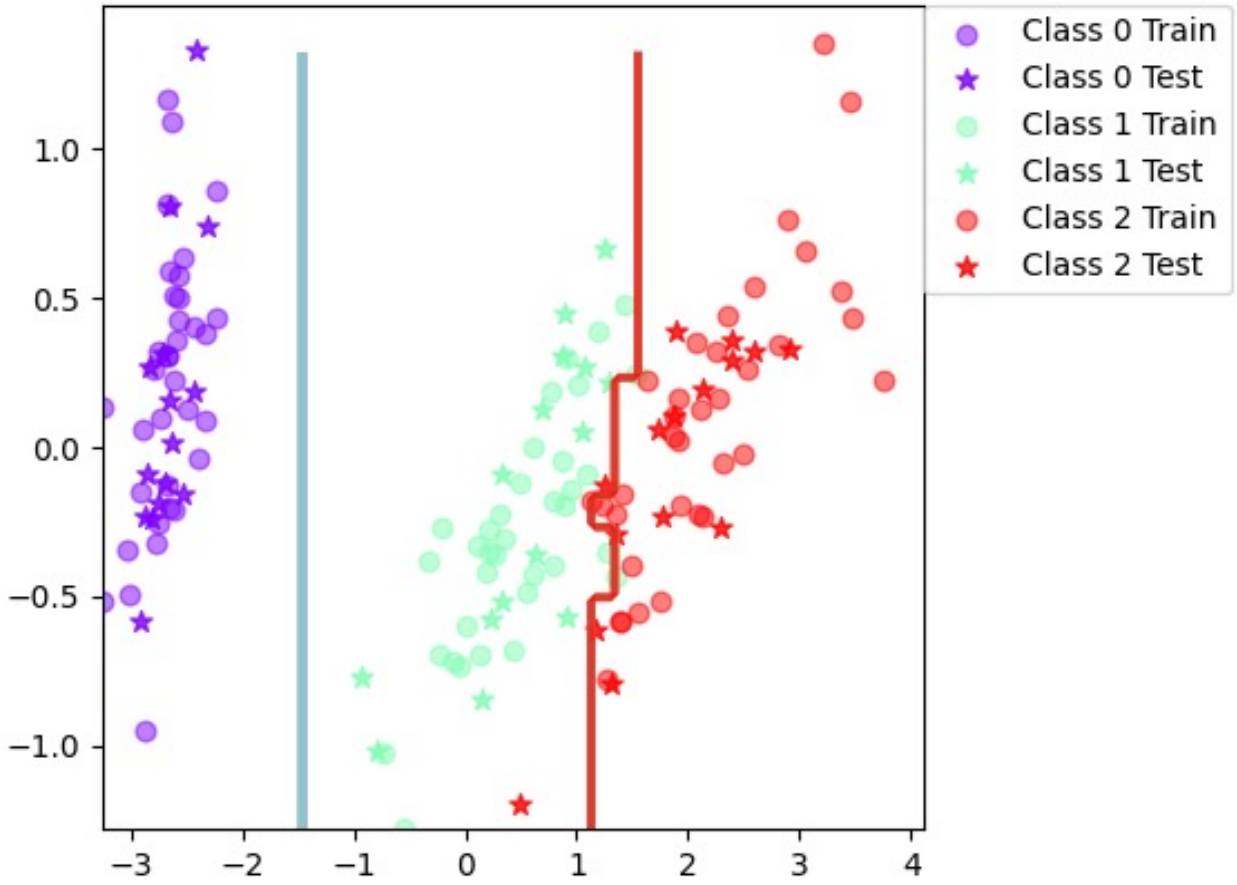
Assignment 6

- Decision Tree – Iris Boost

- Final mean classification accuracy 94.6 with standard deviation 3.65

Trial: 0 Accuracy 95.6
Trial: 10 Accuracy 100
Trial: 20 Accuracy 95.6
Trial: 30 Accuracy 93.3
Trial: 40 Accuracy 93.3
Trial: 50 Accuracy 95.6
Trial: 60 Accuracy 88.9
Trial: 70 Accuracy 93.3
Trial: 80 Accuracy 93.3
Trial: 90 Accuracy 93.3

Assignement 6



Assignment 6

- Decision Tree – Vowel

- Final mean classification accuracy 64.1 with standard deviation 4

Trial: 0 Accuracy 63.6
Trial: 10 Accuracy 68.8
Trial: 20 Accuracy 63.6
Trial: 30 Accuracy 66.9
Trial: 40 Accuracy 59.7
Trial: 50 Accuracy 63
Trial: 60 Accuracy 59.7
Trial: 70 Accuracy 68.8
Trial: 80 Accuracy 59.7
Trial: 90 Accuracy 68.2

Assignment 6

- Decision Tree – Vowel Boost

- Final mean classification accuracy 86.5 with standard deviation 2.85

Trial: 0 Accuracy 86.4
Trial: 10 Accuracy 89.6
Trial: 20 Accuracy 86.4
Trial: 30 Accuracy 90.9
Trial: 40 Accuracy 80.5
Trial: 50 Accuracy 81.8
Trial: 60 Accuracy 86.4
Trial: 70 Accuracy 85.1
Trial: 80 Accuracy 86.4
Trial: 90 Accuracy 89.6

Assignment 7

If you had to pick a classifier, naïve Bayes or a decision tree or the boosted version of these, which on would you pick? Motivate from the following criteria:

- Outliners
- Irrelevant inputs: part of the feature space is irrelevant
- Mixed type of data: binary, categorical or continuous features, etc.
- Scalability: the dimension of the data, D, is large or the number of instances, N, is large, or both.

Assignment 7

- **Outliers:**
  Decision trees are more prone to overfitting than Bayes, ut with good pruning it should handle outliers better.
  Boosting will increase the weight of miscalles samples and generate a more complex boundary, which in this case, would probably lead to overfitting.
- **Irrelevant inputs: part of the future space is irrelevant:**
  Decision trees use information gain when (Gini Inpurity) when constructing the tree and handles irrelevent inputs better.
  Boosting increases the significance of misclassified samples, it would reduce the performance of the classifier

Assignment 7

- **Predictive power:**
  There was a clear advantage in utilizing boosting for both Bayes and decision trees.
  The decision tree seems to have performed slightly better, particularly for the vowel dataset.
- **N mixed types of data: binary, categorical or continuous features:**
  Both decision trees and Bayes can be used for all data types given some pre-processing. Decision tree gives a step boundary, Bayes gives smooth one.

Assignment 7

- **Scalability:**
  Boosting makes the training of a classifier a lot slower which is a problem when there is a large dataset. If computational power is a restrain then it might be preferable to avoid boosting.
  In general decision trees perform better with a large dataset compared to Bayes, and the opposite is true for a small dataset.