**I. Pen-and-paper**

1. First, we transformed the design matrix using the basis function

Then, we will calculate the vector W that minimizes the square-error loss function, this vector is given by the following expression

So, for the training dataset we obtain that

1. To measure the differences between the observed values and the predictions given by the model, in order to test it, we’ll use the RMSE

For that we’ll need to calculate the predictions of our model:

Each prediction is given by the following polynomial regression model

Therefore, the RMSE is:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 0 | N |
|  | 1 | N |
|  | 1 | N |
|  | 0 | N |
|  | 1 | P |
|  | 0 | P |
|  | 0 | P |
|  | 1 | P |

|  |  |
| --- | --- |
|  | 0.25 |
|  | 0.5 |
|  | 0.25 |
|  | 0.5 |
|  | 0.75 |
|  | 0 |
|  | 0.5 |
|  | 0.25 |
|  | 1 |

|  |  |
| --- | --- |
|  | 0.25 |
|  | 0.375 |
|  | 0.375 |
|  | 1 |
|  | 0.666667 |
|  | 0.666667 |
|  | 0 |
|  | 0.333333 |
|  | 0.333333 |

|  |  |
| --- | --- |
|  | 0.5 |
|  | 0.5 |
|  | 0.5 |
|  | 0.5 |
|  | 0.5 |
|  | 0.5 |

1. Firstly, we computed the equal depth binarization of , in which we used the median of its train values as the criteria for the binarization, , and the class targets , so we obtained the following table:  
    To learn a decision tree using ID3, we need to calculate the information gain (IG) of each variable, which is given by , and the entropy.  
   Probabilities of the training dataset:  
     
     
     
     
     
     
   -  
   -  
   -Information Gain: -

Diagram

Description automatically generated  
We can conclude that the variable that will be used as root for the decision tree is as it is the variable with the highest IG, and always leads to , and to uncertain (?). So, we will need to study now only the cases that , which are on the following table:

|  |  |  |  |
| --- | --- | --- | --- |
|  | y2 | y3 | t |
| x1 | 1 | 0 | N |
| x2 | 1 | 1 | N |
| x4 | 2 | 0 | N |
| x6 | 1 | 0 | P |

From this, we can conclude that is uncertain, and that leads to . Since the information gain of and is the same (using analogous calculus from above), we choose the add to the decision tree, which leaves us with the final table:

|  |  |  |
| --- | --- | --- |
|  | y3 | t |
| x1 | 0 | 0 |
| x2 | 1 | 0 |
| x6 | 0 | 1 |

Here, we can conclude that when , then , and with analogous methods, it is clear that when the information gain will be 0, so that will lead to uncertainty. With all this information, we are now able to create the decision tree:

1. Using the “test” dataset on the decision tree obtained earlier ( and ) we get a prediction , (by following the branch), opposed to the true values of , , so, the accuracy will be:

**II. Programming and critical analysis**

1. Using 10-fold cross validation for splitting the data into training and testing data, we trained our decision tree model in two different ways:   
    i)

ii)

After fitting the model, the score is calculated for both training and testing data. The score of each model can be compared in the given plots:

Chart, line chart

Description automatically generated

1. In both graphs we can observe that the values of the training and testing accuracy are somewhat correlated, this observation may be explained by:

* Increasing the depth/number of features leads us to an increase of the test data accuracy (these variables are correlated with the test accuracy). By choosing a lower value for these parameters we a restraining how much information/detail the model has to learn the problem, therefore by lowering the restrictions leads to an increase in learning ability.
* Increasing the depth/number of features leads us to an increase of the training data accuracy (these variables are correlated with the training accuracy). By increasing the parameters, we are giving the model more room, to learning all dataset irregularities instead of learning the problem that the dataset represents. Therefore, the increase of these variables leads to a gather overfitting risk, that represents a gather score in training data.

1. We can observe that as we increase the depth, the overfitting risk grows, since the model starts to learn quite well the training set (the accuracy of the training data increases). To some extent the accuracy of the test data follows the increase in training data, but when the maximum tree depth is above 5, it slightly drops, which leads us to conclude that the model isn’t really leaning the problem at hand, but it’s actually learning the dataset noise of the training data (overfitting). With this in mind, we believe that the best choice for the value of depth is 5 (best fit).

**III. APPENDIX**

**5)**

dataset = pandas.DataFrame(arff.loadarff(<path>)[0])

dataset["Class"] = dataset["Class"].str.decode('utf-8')

inputs = dataset.drop(columns=["Class"]).values; outputs = dataset["Class"].values

k\_fold = KFold(n\_splits=10, random\_state=13, shuffle=True)

def calculateDictMean(dictionary):

return [sum(dictionary[key])/len(dictionary[key]) for key in dictionary]

trainingAccuracy\_i, testingAccuracy = {"1" : [], "3" : [], "5": [], "9": []}

for maxFeatures in [1, 3, 5, 9]:

inputsNew = SelectKBest(mutual\_info\_classif, k=maxFeatures).fit\_transform(inputs ,outputs)

for train, test in k\_fold.split(dataset):

treeClassifier = tree.DecisionTreeClassifier(criterion="entropy", max\_features=maxFeatures)

treeClassifier.fit(inputsNew[train], outputs[train])

trainingAccuracy\_i[str(maxFeatures)].append(treeClassifier.score(inputsNew[train], outputs[train]))

testingAccuracy\_i[str(maxFeatures)].append(treeClassifier.score(inputsNew[test], outputs[test]))

trainingAccuracyMean\_i, testingAccuracyMean\_i = calculateDictMean(trainingAccuracy\_i), calculateDictMean(testingAccuracy\_i)

trainingAccuracy\_ii, testingAccuracy\_ii = {"1" : [], "3" : [], "5": [], "9": []}

for depth in [1, 3, 5, 9]:

for train, test in k\_fold.split(dataset):

treeClassifier = tree.DecisionTreeClassifier(criterion="entropy", max\_depth=depth)

treeClassifier.fit(inputs[train], outputs[train])

trainingAccuracy\_ii[str(depth)].append(treeClassifier.score(inputs[train], outputs[train]))

testingAccuracy\_ii[str(depth)].append(treeClassifier.score(inputs[test], outputs[test]))

trainingAccuracyMean\_ii, testingAccuracyMean\_ii = calculateDictMean(trainingAccuracy\_ii), calculateDictMean(testingAccuracy\_ii)

plt.plot([1, 3, 5, 9], trainingAccuracyMean\_i); plt.plot([1, 3, 5, 9], testingAccuracyMean\_i)

plt.show(); plt.clf()

plt.plot([1, 3, 5, 9], trainingAccuracyMean\_ii); plt.plot([1, 3, 5, 9], testingAccuracyMean\_ii)

plt.show()