**I. Pen-and-paper**

1. Not accounting y0 in the transformation function:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | y0 | y1 | y2 | y3 | Output=z |
| x1 | 1 |  | 2 |  | 1 |
| x2 | 1 |  | 27 |  | 3 |
| x3 | 1 | 0 | 20 | 0 | 2 |
| x4 | 1 |  | 14 |  | 0 |
| x5 | 1 |  | 53 |  | 6 |
| x6 | 1 |  | 3 |  | 4 |
| x7 | 1 |  | 8 | 8 | 5 |
| x8 | 1 |  | 85 |  | 7 |
| x9 | 1 |  | 4 |  | 2 |
| x10 | 1 |  | 6 |  | 4 |

Ex (x1 transformation):

||**x1**||2=

ᶲ(**x1**) = [1 ]T = [1 ]T

**X**= **XT** =

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  |  | 0 |  |  |  |  |  |
| 2 | 27 | 20 | 14 | 53 | 3 | 8 | 85 |
|  |  | 0 |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  | 2 |  |
| 1 |  | 27 |  |
| 1 | 0 | 20 | 0 |
| 1 |  | 14 |  |
| 1 |  | 53 |  |
| 1 |  | 3 |  |
| 1 |  | 8 |  |
| 1 |  | 85 |  |

**XTX**= **(XTX)-1 =**

|  |  |  |  |
| --- | --- | --- | --- |
| 8,196 | -6,231 | 1,305 | -0,079 |
| -6,231 | 5,078 | -1,104 | 0,069 |
| 1,305 | -1,104 | 0,247 | -0,016 |
| -0,079 | 0,069 | -0,016 | 0,001 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 35,884 | 212 |  |
| 35,884 |  | 1482,281 |  |
| 212 | 1482,281 | 11436 |  |
| 1482,281 |  | 93573,52 |  |

**XTz=** **W=(XTX)-1XTz=** f(**x,w**) = **Wx**

|  |
| --- |
|  |
| 155,235 |
| 1088 |
| 8537,228 |

|  |
| --- |
| 4,584 |
| -1,687 |
| 0,338 |
| -0,013 |

f(**x9,w**) = **Wx9** =

f(**x10,w**) = **Wx10** =

1. Considering equal depth binarization of y3 and class targets defined as requested:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | y1 | y2 | y3 | output = z |
| x1 | 1 | 1 | 0 | N |
| x2 | 1 | 1 | 1 | N |
| x3 | 0 | 2 | 1 | N |
| x4 | 1 | 2 | 0 | N |
| x5 | 2 | 0 | 1 | P |
| x6 | 1 | 1 | 0 | P |
| x7 | 2 | 0 | 0 | P |
| x8 | 0 | 2 | 1 | P |
| x9 | 2 | 0 | 0 | N |
| x10 | 1 | 2 | 0 | P |

* p(z) = ½, z=P; ½, z=N.
* p(y1) = ¼, y1=0; ½, y1=1; ¼, y1=2.
* p(z|y1=0)= ½, z=N; ½, z=P. p(z|y1=1)= ¾, z=N; ¼, z=P. p(z|y1=2)= 0, z=N; 1, z=P.
* = 0,156
* p(y2) = ¼, y2=0; 3/8, y2=1; 3/8, y2=2.
* p(z|y2=0)= 0, z=N; 1, z=P. p(z|y2=1)= 2/3, z=N; 1/3, z=P. p(z|y2=2)= 2/3, z=N; 1/3, z=P.
* p(y3) = ½, y2=0; ½, y2=1.
* p(z|y3=0)= ½, z=N; ½, z=P. p(z|y3=1)= ½, z=N; ½, z=P.
* H(z|y1) < H(z|y2) < H(z|y1) => IG(z|y1) = H(z) - H(z|y1) > IG(z|y2) = H(z) - H(z|y2) > IG(z|y3) = H(z) - H(z|y3) => We choose y1 for the root

y1

0

2

1

? ? z=P

* The values of each y2 and each y3 for y1=0 are equal, but the two values of z are different, so p(z|y1=0) = 0,5 whether z=P or z=N.

y1

0

2

1

p(z=P) = p(z=N)=0,5 ? z=P

* p(y2|y1=1) = 0, y2=0; ¾, y2=1; ¼, y2=2
* p(z|y1=1, y2=1) = 2/3, z=N; 1/3, z= P. p(z|y1=1, y2=2) = 1, z=N; 0, z=P.
* p(y3|y1=1) = ¾, y3=0; ¼, y3=1.
* p(z|y1=1, y3=0) = 2/3, z=N; 1/3, z=P. p(z|y1=1, y3=1) = 1, z=N; 0, z=P.
* Choosing y3 for the (y1=1) branch, since there is no information gain difference between (z|y3|y1=1) and (z|y2|y1=1):

y1

0

2

1

p(z=P) = p(z=N)=0,5 y3 z=P

0

1

y2 z=N

2

1

undefined z=N

* There is no data about the (z|y2=0|y3=0|y1=1) branch, so it remains undefined; otherwise, there is no more uncertainty and the ID3 is complete.

1. x9: y1= 2 => z=P (false positive);

x10: y1=1 => (Branch y1=1) y3=0 =>(Branch y1=1, y3=0) y2= 2 => Z=N (false negative)

Accuracy = 0/2 = 0%

**II. Programming and critical analysis**

Training accuracy and testing accuracy difference increases as depth and selected features increase, due to overfitting.

As the number of selected features increases, the decision tree has more data to fit closest to its training observations output and features with less relevance have equal influence on the results, neglecting to generalize for the whole population.

The same applies to the maximum depth, although it does not overfit as much on smaller maximum depths because it prioritizes the most important features.

1. A smaller max depth in the decision tree implies a smaller number of features to choose from, which is why the training accuracies are so close to each other.

However, a bigger max depth, while implying a bigger number of features to choose from, is not using the mutual information parameter, and as such gives priority to the most relevant features, unlike the “number of selected features” decision tree, which gives all nine features equal priority and suffers more from overfitting.

1. False Positive Rate per max depth (training): 1: 0.0790; 3: 0.0232; 5: 0.0153; 9: 0.0005

False Positive Rate per max depth (testing): 1: 0.1053; 3: 0.0418; 5: 0.0839; 9: 0.0920

We chose depth = 3.

While not the one with the highest training accuracy, it is the one that offers the better balance between high testing accuracy and lack of overfitting (least difference between testing and training accuracy).

It is also one with relatively low false negative rate (considering negative = benign cancer): it is more of an issue to be given a benign cancer diagnosis when you really do have malign cancer than the inverse.

**III. APPENDIX**

import numpy as np; from scipy.io import arff; import pandas as pd;

from sklearn import metrics; from sklearn.model\_selection import StratifiedKFold;

from sklearn.tree import DecisionTreeClassifier; from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_selection import SelectKBest, mutual\_info\_classif

tableVar = pd.DataFrame(arff.loadarff("breast.w.arff")[0]).dropna()

tableVar["Class"] = tableVar["Class"].apply(lambda y : y.decode("utf-8"))

x=[]

for i in range(len(tableVar)):x += [list(tableVar.iloc[i][0:9])]

x= np.array(x);y = list(tableVar["Class"]);y= np.array(y)

for el in range(len(y)):

    if y[el] == "benign":y[el] = 0

    else:y[el] = 1

#========================================== 5 =============================

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=78) # 70% training and 30% test

d\_acc\_i\_train = {1:[], 3:[], 5:[], 9:[]}; d\_acc\_i\_test = {1:[], 3:[], 5:[], 9:[]}

d\_acc\_ii\_test ={1:[], 3:[], 5:[], 9:[]}; d\_acc\_ii\_train ={1:[], 3:[], 5:[], 9:[]}

d\_p\_n = {1:[], 3:[], 5:[], 9:[]}; d\_f\_n = {1:[], 3:[], 5:[], 9:[]}

def check\_stats(\_y\_test, \_y\_pred\_test, \_n):

    cnf\_matrix = confusion\_matrix(\_y\_test, \_y\_pred\_test)

    FP = cnf\_matrix[0][1];FN = cnf\_matrix[1][0]; TP = cnf\_matrix[1][1];

    TN = cnf\_matrix[0][0]; FP = FP.astype(float);FN = FN.astype(float);

    TP = TP.astype(float); TN = TN.astype(float); FPR = FP/(FP+TN);FNR = FN/(TP+FN);

    d\_p\_n[\_n].append(FPR); d\_f\_n[\_n].append(FNR)

    return [d\_p\_n,d\_f\_n]

def compute\_average(d\_to\_average):

    average\_dict = {}

    for e in d\_to\_average:

        average = np.average(d\_to\_average[e]); average\_dict[e] = average

    return average\_dict

skf = StratifiedKFold(n\_splits=10,random\_state=78,shuffle=True)

for train\_index, test\_index in skf.split(x, y):

    X\_train, X\_test = x[train\_index], x[test\_index]; y\_train, y\_test = y[train\_index], y[test\_index]

    for e in [1,3,5,9]:

        selector = SelectKBest(score\_func= mutual\_info\_classif ,k=e)

        x\_new = selector.fit\_transform(X\_train, y\_train)

        x\_train\_new\_transf= x\_new; x\_test\_new\_transf= selector.transform(X\_test)

        clf = DecisionTreeClassifier(max\_features= e,max\_depth=None)

        clf = clf.fit(x\_train\_new\_transf, y\_train)

        y\_pred\_test = clf.predict(x\_test\_new\_transf)

        d\_acc\_i\_test[e].append(metrics.accuracy\_score(y\_test, y\_pred\_test))

        y\_pred\_train= clf.predict(x\_train\_new\_transf)

        d\_acc\_i\_train[e].append(metrics.accuracy\_score(y\_train, y\_pred\_train))

        full\_dict\_ii\_train = check\_stats(y\_train, y\_pred\_train, e)

        full\_dict\_ii\_test = check\_stats(y\_test, y\_pred\_test, e)

        clf\_ii = DecisionTreeClassifier(max\_depth=e)

        clf\_ii = clf\_ii.fit(X\_train,y\_train)

        y\_pred\_ii\_test = clf\_ii.predict(X\_test)

        d\_acc\_ii\_test[e].append(metrics.accuracy\_score(y\_test, y\_pred\_ii\_test))

        y\_pred\_ii\_train= clf\_ii.predict(X\_train)

        d\_acc\_ii\_train[e].append(metrics.accuracy\_score(y\_train, y\_pred\_ii\_train))

"""print("False Positives: ", full\_dict\_ii\_train[0], "False Negatives: ", full\_dict\_ii\_train[1])

print( "d\_acc\_i\_train=", d\_acc\_i\_train, "d\_acc\_i\_test=", d\_acc\_i\_test,)

print("d\_acc\_ii\_train=", d\_acc\_ii\_train, "d\_acc\_ii\_test=", d\_acc\_ii\_test)

print("d\_acc\_i\_train\_average= ",compute\_average(d\_acc\_i\_train),"d\_acc\_i\_test\_average= ",compute\_average(d\_acc\_i\_test))

print("d\_acc\_ii\_train\_average= ",compute\_average(d\_acc\_ii\_train),"d\_acc\_ii\_test\_average= ", compute\_average(d\_acc\_ii\_test))

print("d\_acc\_ii\_train\_average\_FPR= ",compute\_average(full\_dict\_ii\_train[0]))

print("d\_acc\_ii\_train\_average\_TPR= ",compute\_average(full\_dict\_ii\_train[1]))

print("d\_acc\_ii\_test\_average\_FPR= ",compute\_average(full\_dict\_ii\_test[0]))

print("d\_acc\_ii\_test\_average\_TPR= ",compute\_average(full\_dict\_ii\_test[1]))

"""