**Name: SUN RUI**

**Student ID: 18083229g**

**Answer 1:**

**a.**

I(Lifestyle, Street, Polarized) = \* + \* + \* ≈ 1.5656

Entropy for TPR (Tear Production Rate):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TPR | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Reduced | 2 | 3 | 2 | 1.5567 |
| Normal | 4 | 1 | 3 | 1.4056 |

E(TPR) = \* 1.5567 + \* 1.4056 ≈ 1.4761

Information\_Gain (TPR) = 1.5656 – 1.4761 = 0.0895

Entropy for Sex:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sex | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| M | 4 | 1 | 3 | 1.4056 |
| F | 2 | 3 | 2 | 1.5567 |

E(Sex) = \* 1.4056 + \* 1.5567 ≈ 1.4761

Information\_Gain (Sex) = 1.5656 – 1.4761 = 0.0895

Entropy for Age:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Age | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Young | 3 | 0 | 2 | 0.971 |
| Old | 2 | 2 | 2 | 1.585 |
| Middle | 1 | 2 | 1 | 1.5 |

E(Age) = \* 0.971 + \* 1.585 + \* 1.5 ≈ 1.3577

Information\_Gain (Age) = 1.5656 – 1.3577 = 0.2079

Entropy for SP (Spectacle Prescription):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SP | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Myope | 2 | 4 | 1 | 1.3788 |
| Hypermetrope | 4 | 0 | 4 | 1 |

E(SP) = \* 1.3788 + \* 1 ≈ 1.1768

Information\_Gain (SP) = 1.5656 – 1.1768 = 0.3888

Entropy for Astigmatism:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Astigmatism | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Yes | 5 | 0 | 3 | 0.9544 |
| No | 1 | 4 | 2 | 1.3788 |

E(Astigmatism) = \* 0.9544 + \* 1.3788 ≈ 1.1525

Information\_Gain (Astigmatism) = 1.5656 – 1.1525 = 0.4131

According to above tables, we can find that the Information Gain of Astigmatism is larger than others, so we can set the **Astigmatism as root node**.

Next, we continue choosing its child nodes when Astigmatism is Yes and No respectively:  
**Yes:** I(Lifestyle, Street, Polarized) = \* + \* + \* ≈ 0.9544

Entropy for TPR (Tear Production Rate):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TPR | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Reduced | 2 | 0 | 2 | 1 |
| Normal | 3 | 0 | 1 | 0.8113 |

E(TPR) = \* 1 + \* 0.8113 ≈ 0.9057

Information\_Gain (TPR) = 0.9544 – 0.9057 = 0.0487

Entropy for Sex:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sex | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| M | 3 | 0 | 1 | 0.8113 |
| F | 2 | 0 | 2 | 1 |

E(Sex) = \* 0.8113 + \* 1 ≈ 0.9057

Information\_Gain (Sex) = 0.9544 – 0.9057 = 0.0487

Entropy for Age:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Age | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Young | 2 | 0 | 0 | 0 |
| Old | 2 | 0 | 2 | 1 |
| Middle | 1 | 0 | 1 | 1 |

E(Age) = \* 0 + \* 1 + \* 1 ≈ 0.75

Information\_Gain (Age) = 0.9544 – 0.75 = 0.2044

Entropy for SP (Spectacle Prescription):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SP | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Myope | 1 | 0 | 1 | 1 |
| Hypermetrope | 4 | 0 | 2 | 0.9183 |

E(SP) = \* 1 + \* 0. 9183 ≈ 0.9387

Information\_Gain (SP) = 0. 9544 – 0.9387 = 0.0157

According to above four tables, we can find the Information Gain of Age is the largest one, so we choose **Age as one of child nodes** of Astigmatism when Astigmatism is Yes.

**No:** I(Lifestyle, Street, Polarized) = \* + \* + \* ≈ 1.3788

Entropy for TPR (Tear Production Rate):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TPR | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Reduced | 0 | 3 | 0 | 0 |
| Normal | 1 | 1 | 2 | 1.5 |

E(TPR) = \* 0 + \* 1.5 ≈ 0.8571

Information\_Gain (TPR) = 1.3788 – 0.8571 = 0.5217

Entropy for Sex:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sex | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| M | 1 | 1 | 2 | 1.5 |
| F | 0 | 3 | 0 | 0 |

E(Sex) = \* 1.5 + \* 0 ≈ 0.8571

Information\_Gain (Sex) = 1.3788 – 0.8571 = 0.5217

Entropy for Age:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Age | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Young | 1 | 0 | 2 | 0.9183 |
| Old | 0 | 2 | 0 | 0 |
| Middle | 0 | 2 | 0 | 0 |

E(Age) = \* 0.9183 + \* 0 + \* 0 ≈ 0.3936

Information\_Gain (Age) = 1.3788 – 0.3936 = 0.9852

Entropy for SP (Spectacle Prescription):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SP | Lifestyle | Street | Polarized | I(Lifestyle, Street, Polarized) |
| Myope | 1 | 4 | 0 | 0.7219 |
| Hypermetrope | 0 | 0 | 2 | 0 |

E(SP) = \* 0.7219 + \* 0 ≈ 0.5156

Information\_Gain (SP) = 1.3788 – 0.5156 = 0.8632

According to above four tables, we can find the Information Gain of Age is the largest one, so we choose **Age as another child nodes** of Astigmatism when Astigmatism is No.

Now, analyze present situation, we can conclude that:

**IF Astigmatism = Yes AND Age = Young THEN Recommendation = Lifestyle**

**IF Astigmatism = No AND Age = Middle THEN Recommendation = Street**

**IF Astigmatism = No AND Age = Old THEN Recommendation = Street**

Because of this, we can ensure three leaf nodes. In additional, when Astigmatism is No, the Age only have one attribute Young which is not ensured, however the attribute SP can help us to divide Recommendations in this branch if Age is Young, so we can draw a conclusion:

**IF Astigmatism = No AND Age = Young AND SP = Myope THEN Recommendation = Lifestyle**

**IF Astigmatism = No AND Age = Young AND SP = Hypermetrope THEN Recommendation = Polarized**

Until now, there is no more child branches when Astigmatism is No, we start consider that Astigmatism is Yes:

Firstly, when Astigmatism is Yes and Age is Old, we can find attribute TPR (Tear Production Rate) can divide this child branch perfectly:

**IF Astigmatism = Yes AND Age = Old AND TPR = Normal THEN Recommendation = Lifestyle**

**IF Astigmatism = Yes AND Age = Old AND TPR = Reduced THEN Recommendation = Polarized**

Secondly, when Astigmatism is Yes and Age is Middle, we can find attribute Sex can divide this child branch perfectly:

**IF Astigmatism = Yes AND Age = Middle AND SEX = F THEN Recommendation = Lifestyle**

**IF Astigmatism = Yes AND Age = Middle AND SEX = M THEN Recommendation = Polarized**

Finally, the decision tree:

Astigmatism

No

Yes

Age

Age

Middle

Old

Young

Young

Old

Middle

TPR

Sex

Lifestyle

SP

Street

Street

Hypermetrope

Myope

F

M

Reduced

Normal

Polarized

Lifestyle

Polarized

Polarized

Lifestyle

Lifestyle

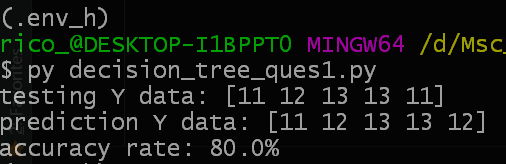
**Note:** In order to use these data in program, we need to transform these data from strings to integers:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tear Production Rate | Sex | Age | Spectacle Prescription | Astigmatism | Recommendation |
| Reduced (4) | M (2) | Young (6) | Myope (9) | Yes (1) | Lifestyle (11) |
| Normal (5) | F (3) | Old (7) | Hypermetrope (10) | No (0) | Street (12) |
|  |  | Middle (8) |  |  | Polarized (13) |

**b.** The testing accuracy rate is **80%**, the data in last row of Testing Data Set is not matching in my decision tree. The result from my decision tree is Street(12), actually the true result is Lifestyle(11).

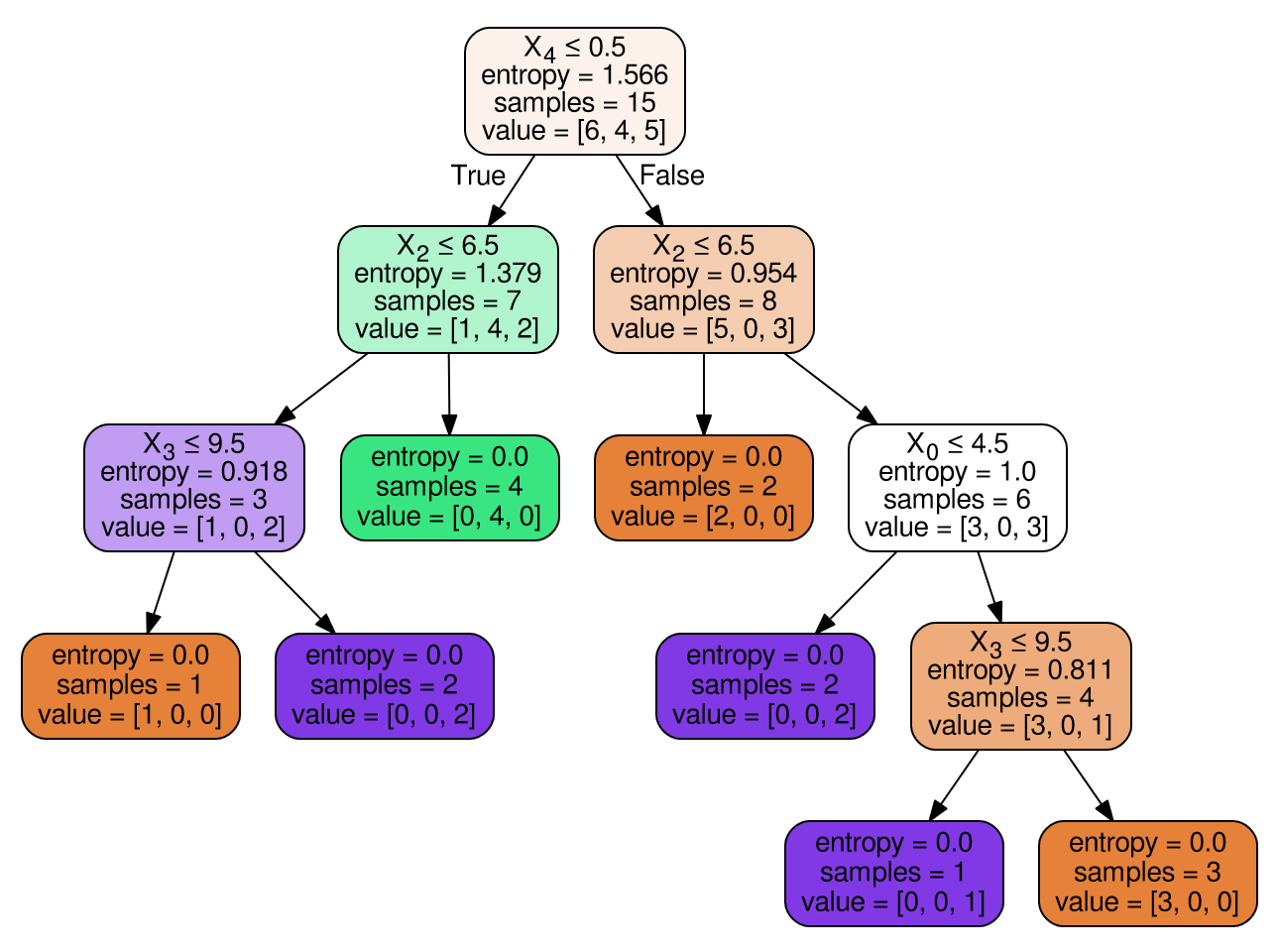
**c.**

**The result by Scikit-Learn:**



We can find that the accuracy rate is 80% by Scikit-Learn, the fault result is also Street(12), actually the result should be Lifestyle, which is same to my conclusion. In conclusion, Scikit-Learn has the similar prediction to my decision tree.

**The decision tree by scikit-learn:**



But, we can find the structure of this tree is a little different from my tree, the first layer and second layer are same in both this tree and my tree, and the count of samples are same, too. But other layers are different.

The reference code:

*# -\*- encoding:utf-8 -\*-*import pandas as pd  
from sklearn import tree  
from sklearn.externals.six import StringIO  
from IPython.display import Image  
from sklearn.tree import export\_graphviz  
import pydotplus

df\_train = pd.read\_csv("training\_data\_set.csv")  
df\_train.columns = ["Tear Production Rate", "Sex", "Age", "Spectacle Prescription", "Astigmatism", "Recommendation"]  
train\_X = df\_train.values[:, 0:5]  
train\_Y = df\_train.values[:, 5]  
  
df\_test = pd.read\_csv("test\_data\_set.csv")  
df\_test.columns = ["Tear Production Rate", "Sex", "Age", "Spectacle Prescription", "Astigmatism", "Recommendation"]  
test\_X = df\_test.values[:, 0:5]  
test\_Y = df\_test.values[:, 5]  
  
dtree = tree.DecisionTreeClassifier(criterion="entropy")  
dtree.fit(train\_X, train\_Y)  
  
Y\_pred = dtree.predict(test\_X)  
print("testing Y data: {0}".format(test\_Y))  
print("prediction Y data: {0}".format(Y\_pred))  
print("accuracy rate: {0}%".format(dtree.score(test\_X, test\_Y)\*100))  
  
dot\_data = StringIO()  
export\_graphviz(dtree, out\_file=dot\_data, filled=True, rounded=True, special\_characters=True)  
graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())  
graph.write\_pdf("decision\_tree.pdf")

**Answer 2:**

**a).**

step1, normalize data ((x-min)/(max-min)):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Customer No. | Sex | Average No. of Transactions | Average Monthly Payment | Average No. of months in Silver |
| 1 | 1 | 0.2632 | 0.0718 | 0.0 |
| 2 | 0 | 0.7895 | 0.4628 | 0.3636 |
| 3 | 1 | 0.1053 | 0.0824 | 0.4545 |
| 4 | 0 | 0.0 | 0.0931 | 0.1818 |
| 5 | 1 | 0.4211 | 0.6596 | 0.5455 |
| 6 | 0 | 0.0 | 1.0 | 0.8182 |
| 7 | 1 | 0.3158 | 0.5745 | 0.0909 |
| 8 | 0 | 0.3684 | 0.0691 | 0.2727 |
| 9 | 1 | 0.2105 | 0.0 | 0.7273 |
| 10 | 1 | 0.8947 | 0.8005 | 1.0 |
| 11 | 0 | 1.0 | 0.6809 | 0.4545 |
| 12 | 1 | 0.5789 | 0.2367 | 0.1818 |
| 13 | 0 | 0.3684 | 0.359 | 0.3636 |
| 14 | 0 | 0.6316 | 0.5452 | 0.2727 |
| 15 | 0 | 0.5263 | 0.4548 | 0.6364 |

step2, normalize sample data:

|  |  |  |  |
| --- | --- | --- | --- |
| Sex | Average No. of Transactions | Average Monthly Payment | Average No. of months in Silver |
| 0 | 0.3158 | 0.3617 | 0.0909 |

step3, calculate euclidean distances among the sample and all 15 training data:

|  |  |
| --- | --- |
| Customer No. | distance |
| 1 | 1.0465 |
| 2 | 0.5559 |
| 3 | 1.12 |
| 4 | 0.4244 |
| 5 | 1.143 |
| 6 | 1.0179 |
| 7 | 1.0224 |
|  |  |

|  |  |
| --- | --- |
| Customer No. | distance |
| 8 | 0.3486 |
| 9 | 1.2437 |
| 10 | 1.5343 |
| 11 | 0.838 |
| 12 | 1.0455 |
| 13 | 0.2778 |
| 14 | 0.408 |
| 15 | 0.5921 |

step4, select 5 smallest distances and their Customer No., further more to find their Decisions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| distance | 0.2778 | 0.3486 | 0.408 | 0.4244 | 0.5559 |
| Customer No. | 13 | 8 | 14 | 4 | 2 |
| Decision | Remain | Upgrade | Remain | Downgrade | Downgrade |

step5, select Decision with largest count number:

According to above table in step4, we find that the count of both Remain and Downgrade is 2, so we can get these two results. If we must get one result, I think that we can choose Remain as final result, because it has smaller distances than Downgrade.

**b)**

The method is similar to question a), the difference is that the target is to find Average No. of Transactions rather than Decision, the detail steps:

step1, normalize data ((x-min)/(max-min)):

|  |  |  |  |
| --- | --- | --- | --- |
| Customer No. | Sex | Average Monthly Payment | Average No. of months in Silver |
| 1 | 1 | 0.0718 | 0.0 |
| 2 | 0 | 0.4628 | 0.3636 |
| 3 | 1 | 0.0824 | 0.4545 |
| 4 | 0 | 0.0931 | 0.1818 |
| 5 | 1 | 0.6596 | 0.5455 |
| 6 | 0 | 1.0 | 0.8182 |
| 7 | 1 | 0.5745 | 0.0909 |
| 8 | 0 | 0.0691 | 0.2727 |
| 9 | 1 | 0.0 | 0.7273 |
| 10 | 1 | 0.8005 | 1.0 |
| 11 | 0 | 0.6809 | 0.4545 |
| 12 | 1 | 0.2367 | 0.1818 |
| 13 | 0 | 0.359 | 0.3636 |
| 14 | 0 | 0.5452 | 0.2727 |
| 15 | 0 | 0.4548 | 0.6364 |

step2, normalize sample data:

|  |  |  |
| --- | --- | --- |
| Sex | Average Monthly Payment | Average No. of months in Silver |
| 1 | 0.2821 | 0.4655 |

step3, calculate euclidean distances among the sample and all 15 training data:

|  |  |
| --- | --- |
| Customer No. | distance |
| 1 | 0.5108 |
| 2 | 1.0213 |
| 3 | 0.2 |
| 4 | 1.0565 |
| 5 | 0.3859 |
| 6 | 1.2805 |
| 7 | 0.4752 |
|  |  |

|  |  |
| --- | --- |
| Customer No. | distance |
| 8 | 1.0405 |
| 9 | 0.3848 |
| 10 | 0.7446 |
| 11 | 1.0766 |
| 12 | 0.2874 |
| 13 | 1.0081 |
| 14 | 1.0519 |
| 15 | 1.0291 |

step4, select 5 smallest distances and their Customer No., further more to find their Average No. of Transactions and to calculate weights:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| distance | 0.2 | 0.2874 | 0.3848 | 0.3859 | 0.4752 |
| Customer No. | 3 | 12 | 9 | 5 | 7 |
| Average No. of Transactions | 5 | 14 | 7 | 11 | 9 |
| Weights | 0.317 | 0.2206 | 0.1647 | 0.1643 | 0.1334 |

**Note:** method of calculating weights:

(distance/sum(distances))^-1 / sum((distance/sum(distances))^-1)

step5, calculate the final Average No. of Transactions through multiply these 5 Average No. of Transactions with their weights respectively in step4, then plus these five results together:

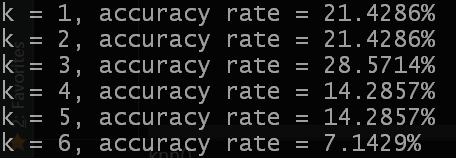
5\*0.317 + 14 \* 0.2206 + 7\*0.1647 + 11\*0.1643 + 9\*0.1334 = 8.8342

In conclusion, we expect the average no. of transactions of the customer is **8.8342**

**c)**

For part a), I will choose k=3. If k = 3, we can get only decision of Remain. And the RMSE is 0.089 if k=3, which is less than 0.206 if k=5. As a conclusion, I think k=3 is better than k=5 in part a). Then we can also test the accuracy rate, each time we use every sample of 15 data as target sample to calculate the final accuracy rate, which actually is cross validation. Let us look at the performance when k equals 1, 2, 3, 4, 5 and 6 respectively:

(**Note: the program is coded by myself, because I use it to calculate results of every step in part a) and b), it also can be used in part c), I will put the code in the last page, ref1**)

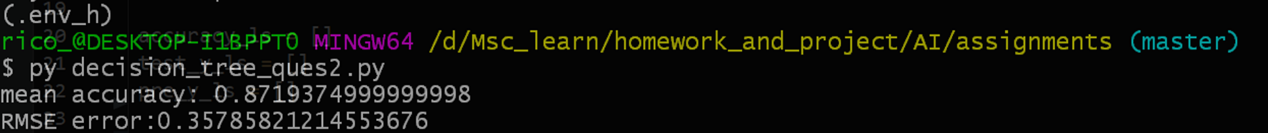


We can find that when k =3, the accuracy rate is the largest one among all results.

**Answer 3:**

**a)**

Without pre-processing data, the result:



I make the training step loop for **100000** times, which gets mean accuracy about **87.19%** and global RMSE error about **0.3579**. Reference code:

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn import tree  
from sklearn import metrics  
  
file\_name = "cardiac.csv" df\_train = pd.read\_csv(file\_name)  
df\_train.columns = ["bhr", "basebp", "basedp", "pkhr", "sbp", "dp", "dose", "maxhr", "%mphr(b)", "mbp", "dpmaxdo",  
 "dobdose", "age", "gender", "baseEF", "dobEF", "chestpain", "posECG", "equivecg", "restwma",  
 "posSE", "newMI", "newPTCA", "newCABG", "hxofHT", "hxofdm", "hxofcig", "hxofMI",  
 "hxofPTCA", "hxofCABG", "death"]  
X = df\_train.values[:, 0:-1]  
Y = df\_train.values[:, -1]  
  
accuracy\_ls = []  
test\_y\_ls = []  
pre\_y\_ls = []  
  
times = 100000  
for i in range(times):  
 print("finish {0}%\r".format(int(i / times \* 100)), end='')  
 trainX, testX, trainY, testY = train\_test\_split(X, Y, test\_size=0.3)  
 test\_y\_ls.append(testY)  
 dtree = tree.DecisionTreeClassifier(criterion="entropy")  
 dtree.fit(trainX, trainY)  
 Y\_pred = dtree.predict(testX)  
 pre\_y\_ls.append(Y\_pred)  
 accuracy\_ls.append(dtree.score(testX, testY))  
  
print("mean accuracy: {0}".format(np.mean(accuracy\_ls)))  
print("RMSE error:{0}".format(np.sqrt(metrics.mean\_squared\_error(test\_y\_ls, pre\_y\_ls))))

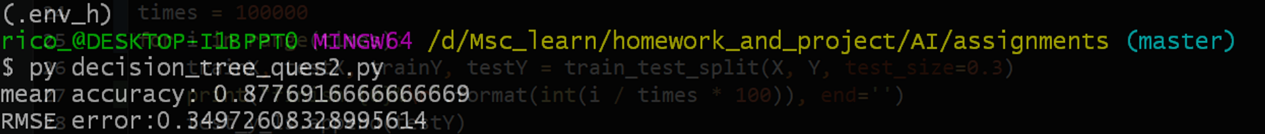
**b)**

After observing data set, I found some data is much dispersive in a large range, so I decide to divide them into some intervals, the way:

|  |  |
| --- | --- |
| max<=50 | Interval is 5: x0~x2 → x0, x3~x7 → x5, x8~y0 → y0 (y=x+1) |
| 50<max<=100 | Interval is 10: x0~x4 → x0, x5~x9 → y0 (y=x+1) |
| 100<max | only left first two higher bits, other lower bits are set 0, if exist <100,  refer to “50<max<=100” |

I will put the code in the last page, (**ref2**).

Through the dispose in terms of above methods, we can get a better result by the same way to part a):



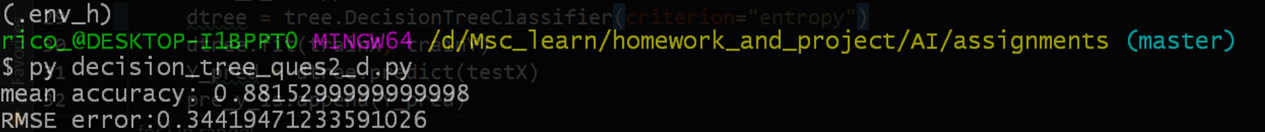
We can get mean accuracy about **87.77%** and global RMSE error about **0.3497**. We can find that the mean accuracy is higher than part a) about **0.58%**, and RMSE error is lower than a) about **0.0082**, which are better than part a).

Then I found the kinds of data of several attributes are too much single (by a python script to count data, I will put the code in the last page, **ref3**), so after delete these attributes (refer to the table below), we see the performance:

delete these attributes:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| hxofPTCA | | hxofcig | | newCABG | | newPTCA | | posECG | |
| data | count | data | count | data | count | data | count | data | count |
| 0 | 74 | 0.0 | 66 | 0 | 71 | 0 | 73 | 0 | 72 |
| 1 | 3 | 1.5 | 6 | 1 | 6 | 1 | 4 | 1 | 5 |
|  |  | 1.0 | 5 |  |  |  |  |  |  |

performance:



We can get mean accuracy about **88.15%** and global RMSE error about **0.3442**. We can find that the mean accuracy is higher than part a) about **0.96%**, and RMSE error is lower than a) about **0.0137**, which are better than part a).

Draw a conclusion, through above two steps, we can improve accuracy of the classification process to some extent, but the performance is limited.

ref1:

*# -\*- encoding:utf-8 -\*-*import numpy as np  
import heapq  
from decimal import Decimal  
import copy  
from collections import Counter  
  
def knn(k, target\_sample, sex\_train, avg\_trans\_train, avg\_payment\_train, avg\_silver\_train):  
 *# ############# find max and min ###############* max\_avg\_trans = max(avg\_trans\_train)  
 min\_avg\_trans = min(avg\_trans\_train)  
 max\_avg\_payment = max(avg\_payment\_train)  
 min\_avg\_payment = min(avg\_payment\_train)  
 max\_avg\_silver = max(avg\_silver\_train)  
 min\_avg\_silver = min(avg\_silver\_train)  
  
 round\_fun = lambda x: float('{:.4f}'.format(Decimal(x)))  
 *# ############# normalize training data ###############* normalization\_fun = lambda input\_ls, max\_val, min\_val: [round\_fun((each\_data - min\_val)/(max\_val - min\_val)) for each\_data in input\_ls]  
 normal\_avg\_trans = normalization\_fun(avg\_trans\_train, max\_avg\_trans, min\_avg\_trans)  
 normal\_avg\_payment = normalization\_fun(avg\_payment\_train, max\_avg\_payment, min\_avg\_payment)  
 normal\_avg\_silver = normalization\_fun(avg\_silver\_train, max\_avg\_silver, min\_avg\_silver)  
 print("normalization avg\_trans: {0}".format(normal\_avg\_trans))  
 print("normalization avg\_payment: {0}".format(normal\_avg\_payment))  
 print("normalization avg\_silver: {0}".format(normal\_avg\_silver))  
 print("..................................................................")  
  
 *# ############# normalize target sample ###############* sample\_normal\_avg\_trans = round\_fun((target\_sample["avg\_trans"] - min\_avg\_trans)/(max\_avg\_trans - min\_avg\_trans))  
 sample\_normal\_avg\_payment = round\_fun((target\_sample["avg\_payment"] - min\_avg\_payment)/(max\_avg\_payment - min\_avg\_payment))  
 sample\_normal\_avg\_silver = round\_fun((target\_sample["avg\_silver"] - min\_avg\_silver)/(max\_avg\_silver - min\_avg\_silver))  
 sample\_sex = target\_sample["sex"]  
 print("sample attributes normalization: avg\_trans({0}), avg\_payment({1}), avg\_silver({2}), "  
 "sex({3})".format(sample\_normal\_avg\_trans,sample\_normal\_avg\_payment, sample\_normal\_avg\_silver, sample\_sex))  
 print("..................................................................")  
  
 *# ############# calculate euclidean distance ###############  
 # euclidean distance power 2 in every attribute:* distance\_fun = lambda normal\_ls, sample\_data: [round\_fun(np.square(sample\_data - each\_data, dtype=np.float64)) for each\_data in normal\_ls]  
 distance\_in\_avg\_trans = distance\_fun(normal\_avg\_trans, sample\_normal\_avg\_trans)  
 distance\_in\_avg\_payment = distance\_fun(normal\_avg\_payment, sample\_normal\_avg\_payment)  
 distance\_in\_avg\_silver = distance\_fun(normal\_avg\_silver, sample\_normal\_avg\_silver)  
 distance\_in\_sex = distance\_fun(sex\_train, sample\_sex)  
  
 *# final euclidean distance:* overall\_distance = []  
 for i in range(len(distance\_in\_avg\_trans)):  
 overall\_distance.append(round\_fun(np.sqrt(distance\_in\_avg\_trans[i] + distance\_in\_avg\_payment[i] + distance\_in\_avg\_silver[i] + distance\_in\_sex[i], dtype=np.float64)))  
 print("every euclidean distance: {0}".format(overall\_distance))  
 print("..................................................................")

ref1(2):

*# ############# find min k samples ###############* min\_k\_ls = heapq.nsmallest(k, overall\_distance)  
 index\_smallest\_distance = [overall\_distance.index(each\_distance) for each\_distance in min\_k\_ls]  
 print("smallest K distances: {0}".format(min\_k\_ls))  
 print("smallest K index (first 0): {0}".format(index\_smallest\_distance))  
 print("smallest K items:")  
 final\_des\_ls = []  
 for index in index\_smallest\_distance:  
 final\_des\_ls.append(decision\_dic[decision\_ls[index]])  
 print("\t" + decision\_dic[decision\_ls[index]])  
 word\_counts = Counter(final\_des\_ls)  
 top\_one = word\_counts.most\_common(1)  
 return top\_one[0][0]  
  
*# target\_sample = {"sex": 0, "avg\_trans": 9, "avg\_payment": 410, "avg\_silver": 5}  
  
# ############# data ###############*sex = [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0] *# F: 1, M:0*avg\_trans = [8, 18, 5, 3, 11, 3, 9, 10, 7, 20, 22, 14, 10, 15, 13]  
avg\_payment = [301, 448, 305, 309, 522, 650, 490, 300, 274, 575, 530, 363, 409, 479, 445]  
avg\_silver = [4, 8, 9, 6, 10, 13, 5, 7, 12, 15, 9, 6, 8, 7, 11]  
decision\_dic = {1: "Remain", 2: "Downgrade", 3: "Upgrade"}  
decision\_ls = [1, 2, 1, 2, 1, 2, 3, 3, 2, 3, 2, 3, 1, 1, 1]  
  
  
def cal\_accuracy\_rate(k):  
 count1 = 0  
 for i in range(len(sex)):  
 target\_sample = {"sex": sex[i], "avg\_trans": avg\_trans[i], "avg\_payment": avg\_payment[i], "avg\_silver": avg\_silver[i]}  
 sex\_train = copy.deepcopy(sex)  
 avg\_trans\_train = copy.deepcopy(avg\_trans)  
 avg\_payment\_train = copy.deepcopy(avg\_payment)  
 avg\_silver\_train = copy.deepcopy(avg\_silver)  
 sex\_train.pop(i)  
 avg\_trans\_train.pop(i)  
 avg\_payment\_train.pop(i)  
 avg\_silver\_train.pop(i)  
 prediction = knn(k, target\_sample, sex\_train, avg\_trans\_train, avg\_payment\_train, avg\_silver\_train)  
 real\_value = decision\_dic[decision\_ls[i]]  
 if real\_value == prediction:  
 count1 += 1  
 round\_fun = lambda x: float('{:.4f}'.format(Decimal(x)))  
 accuracy\_rate = round\_fun(count1/(len(sex)-1)\*100)  
 return "{}%".format(accuracy\_rate)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 k\_ls = [1, 2, 3, 4, 5, 6]  
 accuracy\_dict = {}  
 for k in k\_ls:  
 print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* k = {0} \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*".format(k))  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 accuracy\_rate = cal\_accuracy\_rate(k)  
 accuracy\_dict[k] = accuracy\_rate  
 for (k, v) in accuracy\_dict.items():  
 print("k = {0}, accuracy rate = {1}".format(k, v))

ref2:

*# -\*- encoding:utf-8 -\*-*import pandas as pd  
import numpy as np  
  
df\_data = pd.read\_csv("cardiac.csv")  
df\_data.columns = ["bhr", "basebp", "basedp", "pkhr", "sbp", "dp", "dose", "maxhr", "%mphr(b)", "mbp", "dpmaxdo",  
 "dobdose", "age", "gender", "baseEF", "dobEF", "chestpain", "posECG", "equivecg", "restwma",  
 "posSE", "newMI", "newPTCA", "newCABG", "hxofHT", "hxofdm", "hxofcig", "hxofMI",  
 "hxofPTCA", "hxofCABG", "death"]  
  
  
dic\_clean\_data = {}  
*# bhr\_series = df\_data["bhr"]*dic\_clean\_data["bhr"] = df\_data["bhr"]  
*# basebp\_series = df\_data["basebp"]*dic\_clean\_data["basebp"] = df\_data["basebp"]  
*# basedp\_series = df\_data["basedp"]*dic\_clean\_data["basedp"] = df\_data["basedp"]  
*# pkhr\_series = df\_data["pkhr"]*dic\_clean\_data["pkhr"] = df\_data["pkhr"]  
*# sbp\_series = df\_data["sbp"]*dic\_clean\_data["sbp"] = df\_data["sbp"]  
*# dp\_series = df\_data["dp"]*dic\_clean\_data["dp"] = df\_data["dp"]  
*# maxhr\_series = df\_data["maxhr"]*dic\_clean\_data["maxhr"] = df\_data["maxhr"]  
*# mphr\_b\_series = df\_data["%mphr(b)"]*dic\_clean\_data["%mphr(b)"] = df\_data["%mphr(b)"]  
*# mbp\_series = df\_data["mbp"]*dic\_clean\_data["mbp"] = df\_data["mbp"]  
*# dpmaxdo\_series = df\_data["dpmaxdo"]*dic\_clean\_data["dpmaxdo"] = df\_data["dpmaxdo"]  
*# age\_series = df\_data["age"]*dic\_clean\_data["age"] = df\_data["age"]  
*# baseEF\_series = df\_data["baseEF"]*dic\_clean\_data["baseEF"] = df\_data["baseEF"]  
*# dobEF\_series = df\_data["dobEF"]*dic\_clean\_data["dobEF"] = df\_data["dobEF"]

for (k, v) in dic\_clean\_data.items():  
 *# max\_item = max(v)  
 # min\_item = min(v)* temp\_ls = []  
 for each\_data in v:  
 if max(v) <= 50:  
 if 3 <= each\_data % 10 <= 7:  
 new\_data = each\_data//10 \* 10 + 5  
 elif 7 < each\_data % 10 <= 9:  
 new\_data = each\_data//10 \* 10 + 10  
 elif 1 <= each\_data % 10 < 3:  
 new\_data = each\_data // 10 \* 10  
 else:  
 new\_data = each\_data  
 temp\_ls.append(new\_data)

ref2(2):

elif 50 < max(v) <= 100:  
 if 1 <= each\_data % 10 <= 4:  
 new\_data = each\_data//10 \* 10  
 elif 5 <= each\_data % 10 <= 9:  
 new\_data = each\_data//10 \* 10 + 10  
 else:  
 new\_data = each\_data  
 temp\_ls.append(new\_data)  
 else:  
 if each\_data < 100:  
 if 1 <= each\_data % 10 <= 4:  
 new\_data = each\_data // 10 \* 10  
 elif 5 <= each\_data % 10 <= 9:  
 new\_data = each\_data // 10 \* 10 + 10  
 else:  
 new\_data = each\_data  
 temp\_ls.append(new\_data)  
 else:  
 num\_len = len(str(each\_data))  
 new\_data = (each\_data//np.power(10, num\_len-2)) \* np.power(10, num\_len-2)  
 temp\_ls.append(new\_data)  
 df\_data[k] = temp\_ls  
  
df\_data.to\_csv("nor\_data1.csv") *# normalization data*

*# -\*- encoding:utf-8 -\*-*import pandas as pd  
  
df\_data = pd.read\_csv("nor\_data1.csv")  
df\_data.columns = ["bhr", "basebp", "basedp", "pkhr", "sbp", "dp", "dose", "maxhr", "%mphr(b)", "mbp", "dpmaxdo",  
 "dobdose", "age", "gender", "baseEF", "dobEF", "chestpain", "posECG", "equivecg", "restwma",  
 "posSE", "newMI", "newPTCA", "newCABG", "hxofHT", "hxofdm", "hxofcig", "hxofMI",  
 "hxofPTCA", "hxofCABG", "death"]  
count = 0  
for col in df\_data.columns:  
 count += 1  
 print(df\_data[col].value\_counts())  
print(count)

ref3: