D.J. Anderson dra2zp 10/09/2017

Homework 2

1 (b) Annual family income, the number of years a couple stayed married, and the number of children the couple had seem to have little, if any, correlation on the couple's marriage status in five years. Couples can stay married or divorced regardless of how long they've been together, how many kids they have, and how much money they make. Thus, this estimated model does not have practical value, and it would not be appropriate to use predictive learning to estimate a data-analytic model.

(c) A person's cell phone number and gender tell nothing about the person's remaining life expectancy. Although a woman's life expectancy is about five years longer than men's life expectancy, on average, this information along with their cell phone number, which is completely unrelated, would not give any useful information about their remaining life expectancy. Thus, this estimated model does not have practical value, and it would not be appropriate to use predictive learning to estimate a data-analytic model.

(d) Today's closing price of a stock index is generally not a good indicator of the next day's closing price because stock prices can fluctuate up and down throughout the day, so simply knowing today's closing price will not give useful information on tomorrow's closing price. Thus, this estimated model does not have practical value, and it would not be appropriate to use predictive learning to estimate a data-analytic model.

(e) A time series of stock index prices recorded over the past five days would be a good indicator of whether the next day's price change was up or down because if stock prices have been increasing for the past five days, it's likely that the following day's prices will increase and vice versa. However, this is not the best indicator since stock prices can still fluctuate and shift dramatically depending on the stock index. Thus, the dependency can be estimated from data, it has useful practical value, and it is appropriate to use predictive learning to estimate a data-analytic model.

(f) A patient's age, brain scan information, and results of psychological tests would be an excellent indicator of the presence or absence of Alzheimer's disease diagnosis in the next five years because typically older people are diagnosed with Alzheimer's disease as opposed to younger people, and a brain scan along with psychological test data would reveal if the person has or likely will have Alzheimer's disease. Thus, the dependency can be estimated from data, it has useful practical value, and it is appropriate to use predictive learning to estimate a data-analytic model.

2 Source Code:

```
# 1) loading libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from collections import Counter
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
# define column names
names = ['obesity', 'class']
# loading training data
df = pd.read_csv('obesity.data.txt', header=None, names=names)
df.head()
# making our predictions
predictions = []
# create design matrix X and target vector y
X = np.array(df['obesity'])     # end index is exclusive
y = np.array(df['class'])   # another way of indexing a pandas df
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=(1/3), random_state=42)
X_train = X_train.reshape(-1, 1)
# creating odd list of K for KNN
myList = list(range(1,31))
# subsetting just the odd ones
neighbors = list(filter(lambda x: x % 2 != 0, myList))
# empty list that will hold cross validation scores
cv_scores = []
# perform 10-fold cross validation we are already familiar with
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10,
scoring='accuracy')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]
print ("k          Score          MSE")
for i in range(len(neighbors)):
    print ('%d          %.5f          %.5f' % (neighbors[i],
cv_scores[i], MSE[i]))
""" Output
k         Score         MSE
1         0.72667         0.27333
3         0.67333         0.32667
5         0.77333         0.22667
7         0.66000         0.34000
9         0.60667         0.39333
11         0.52000         0.48000
13         0.58667         0.41333
15         0.65333         0.34667
17         0.65333         0.34667
```

```
19         0.65333          0.34667
21         0.65333          0.34667
23         0.65333          0.34667
25         0.65333          0.34667
27         0.65333          0.34667
29         0.65333          0.34667
"""
# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d" % optimal_k)
# plot misclassification error vs k
plt.plot(neighbors, MSE)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
def train(X_train, y_train):
    # do nothing
    return
from collections import Counter
def predict(X_train, y_train, x_test, k):
    # create list for distances and targets
    distances = []
    targets = []
    for i in range(len(X_train)):
        # first we compute the Euclidean distance
        # (use x_test and X_train[i, :]. Also, where appropriate, you
can use np.sqrt, np.square, and np.sum...)
        distance = np.sqrt(np.sum(np.square(x_test - X_train[i, :])))
        # add it to list of distances
        distances.append([distance, i])
    # sort the list
    distances = sorted(distances)
    # make a list of the k neighbors' targets
    for i in range(k):
        # (Hint: index receives particular value in
distances[something][something])
        index = distances[i][1]
        # (Hint: use y_train and index below)
        targets.append(y_train[index])
    # return most common target
    return Counter(targets).most_common(1)[0][0]
from sklearn.metrics import accuracy_score
def kNearestNeighbor(X_train, y_train, X_test, predictions, k):
    # train on the input data
    train(X_train, y_train)
    # loop over all observations
    for i in range(len(X_test)):
        predictions.append(predict(X_train, y_train, X_test[i, :], k))
# making our predictions
# Using the optimal value of K discovered above
predictions = []
try:
    X_test = X_test.reshape(-1, 1)
```

```
    X_test = np.array([[0.259], [0.269], [0.281], [0.250], [0.257],
[0.214], [0.292], [0.261], [0.290], [0.272], [0.218],
                       [0.211], [0.252], [0.245], [0.306], [0.255],
[0.240]])
    optimalK = 5 # Add your answer here (and delete line!)
    kNearestNeighbor(X_train, y_train, X_test, predictions, optimalK)
    predictions = np.asarray(predictions)
    # evaluating accuracy
    accuracy = accuracy_score(y_test, predictions) * 100
    print('\nThe accuracy of OUR classifier is %d%%' % accuracy)
    print ("X_test")
    print (X_test)
    print ("X_train")
    print (X_train)
    print ("y_train")
    print (y_train)
    print ("y_test")
    print (y_test)
    print ("predictions")
    print (predictions)
except ValueError:
    print('Can\'t have more neighbors than training samples!!') # Need
to be careful about value of k
```

Predictions:

State	*Obesity*	*2004*	*Prediction*	*[Ground Truth] 2000*
Alabama	0.301	R	R	R
Alaska	0.273	R	R	R
Arizona	0.233	R	R	R
Arkansas	0.281	R	R	R
California	0.231	D	R	D
Colorado	0.210	R	D	R
Connecticut	0.208	D	D	D
District of Columbia	0.221	D	D	D
Delaware	0.259	D	R	D
Florida	0.233	R	R	R
Georgia	0.275	R	R	R
Hawaii	0.207	D	D	D
Idaho	0.246	R	D	R
Illinois	0.253	D	D	D
Indiana	0.275	R	R	R
Iowa	0.263	R	R	D
Kansas	0.258	R	R	R
Kentucky	0.284	R	R	R
Lousiana	0.295	R	R	R
Maine	0.237	D	R	D
Maryland	0.252	D	D	D
Massachusetts	0.209	D	D	D
Michigan	0.277	D	R	D

Minnesota	0.248	D	D	D
Mississippi	0.344	R	R	R
Missouri	0.274	R	R	R
Montana	0.217	R	D	R
Nebraska	0.265	R	R	R
Nevada	0.236	R	R	R
New Hampshire	0.236	D	R	R
New Jersey	0.229	D	R	D
New Mexico	0.233	R	R	D
New York	0.235	D	R	D
North Carolina	0.271	R	R	R
North Dakota	0.259	R	R	R
Ohio	0.269	R	R	R
Oklahoma	0.281	R	R	R
Oregon	0.250	D	D	D
Pennsylvania	0.257	D	R	D
Rhode Island	0.214	D	R	D
South Carolina	0.292	R	R	R
South Dakota	0.261	R	R	R
Tennessee	0.290	R	R	R
Texas	0.272	R	R	R
Utah	0.218	R	D	R
Vermont	0.211	D	D	D
Virginia	0.252	R	D	R
Washington	0.245	D	D	D
West Virginia	0.306	R	R	R
Wisconsin	0.255	D	D	D
Wyoming	0.240	R	R	R

Accuracy: $\frac{36}{51} = 70.59\%$

3 (a) Source Code:

```
import os
import subprocess
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, export_graphviz
def get_iris_data():
    names = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width', 'Name']
    df = pd.read_csv("iris.data.csv", header=None, names=names)
    df.head()
    return df
df = get_iris_data()
def encode_target(df, target_column):
    df_mod = df.copy()
    targets = df_mod[target_column].unique()
    map_to_int = {name: n for n, name in enumerate(targets)}
    df_mod["Target"] = df_mod[target_column].replace(map_to_int)
    return (df_mod, targets)
df2, targets = encode_target(df, 'Name')
print("* df2.head()", df2[["Target", "Name"]].head(),
      sep="\n", end="\n\n")
print("* df2.tail()", df2[["Target", "Name"]].tail(),
      sep="\n", end="\n\n")
print("* targets", targets, sep="\n", end="\n\n")
features = list(df2.columns[:4])
print("* features:", features, sep="\n")
y = df2["Target"]
X = df2[features]
dt = DecisionTreeClassifier(random_state=42, criterion="entropy")
dt.fit(X, y)
def visualize_tree(tree, feature_names):
    with open("dt.dot", 'w') as f:
        export_graphviz(tree, out_file=f,
                        feature_names=feature_names)
    command = ["dot", "-Tpng", "dt.dot", "-o", "dt.png"]
    try:
        subprocess.check_call(command)
    except:
        exit("Could not run dot, ie graphviz, to "
             "produce visualization")
visualize_tree(dt, features)
""" Output
* df2.head()
   Target         Name
0       0  Iris-setosa
1       0  Iris-setosa
2       0  Iris-setosa
3       0  Iris-setosa
4       0  Iris-setosa
* df2.tail()
     Target            Name
145       2  Iris-virginica
```

```
146       2  Iris-virginica
147       2  Iris-virginica
148       2  Iris-virginica
149       2  Iris-virginica
* targets
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
* features:
['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
"""
```

(b) The attribute that was used as the first decision node of this tree generated by the DT classifier was petal_length.

(c) Tear Production Rate has the highest information gain with value ∞.

$E(Contact\ Lenses) = -\left(\frac{9}{24}\log_2\frac{9}{24}\right) - \left(\frac{15}{24}\log_2\frac{15}{24}\right) = 0.9544.$

Age of the Patient	Contact Lenses	No Contact Lenses	Total
Young	4	4	8
Pre-Presbyopic	3	5	8
Presbyopic	2	6	8
Total			24

$E(Young) = -\left(\frac{4}{8}\log_2\frac{4}{8}\right) - \left(\frac{4}{8}\log_2\frac{4}{8}\right) = 1.$ $E(Pre - Presbyopic) = -\left(\frac{3}{8}\log_2\frac{3}{8}\right) - \left(\frac{5}{8}\log_2\frac{5}{8}\right) = 0.9544.$ $E(Presbyopic) = -\left(\frac{2}{8}\log_2\frac{2}{8}\right) - \left(\frac{6}{8}\log_2\frac{6}{8}\right) = 0.8113.$

$E(Contact\ Lenses, Age\ of\ the\ Patient) = \frac{8}{24} * 1 + \frac{8}{24} * 0.9544 + \frac{8}{24} * 0.8113 = 0.9219.$

$E(Contact\ Lenses) = 0.9544.$ $G(Contact\ Lenses, Age\ of\ the\ Patient) = 0.9544 - 0.9219 = \mathbf{0.0325}.$

Spectacle Prescription	Contact Lenses	No Contact Lenses	Total
Myope	5	7	12
Hypermetrope	4	8	12
Total			24

$E(Myope) = -\left(\frac{5}{12}\log_2\frac{5}{12}\right) - \left(\frac{7}{12}\log_2\frac{7}{12}\right) = 0.9799.$ $E(Hypermetrope) = -\left(\frac{4}{12}\log_2\frac{4}{12}\right) - \left(\frac{8}{12}\log_2\frac{8}{12}\right) = 0.9183.$ $E(Contact\ Lenses, Age\ of\ the\ Patient) = \frac{12}{24} * 0.9799 + \frac{12}{24} * 0.9183 = 0.9491.$ $E(Contact\ Lenses) = 0.9544.$ $G(Contact\ Lenses, Age\ of\ the\ Patient) = 0.9544 - 0.9491 = \mathbf{0.0053}.$

Astigmatic	Contact Lenses	No Contact Lenses	Total
No	5	7	12
Yes	4	8	12
Total			24

$E(No) = -\left(\frac{5}{12}\log_2\frac{5}{12}\right) - \left(\frac{7}{12}\log_2\frac{7}{12}\right) = 0.9799.$ $E(Yes) = -\left(\frac{4}{12}\log_2\frac{4}{12}\right) - \left(\frac{8}{12}\log_2\frac{8}{12}\right) = 0.9183.$

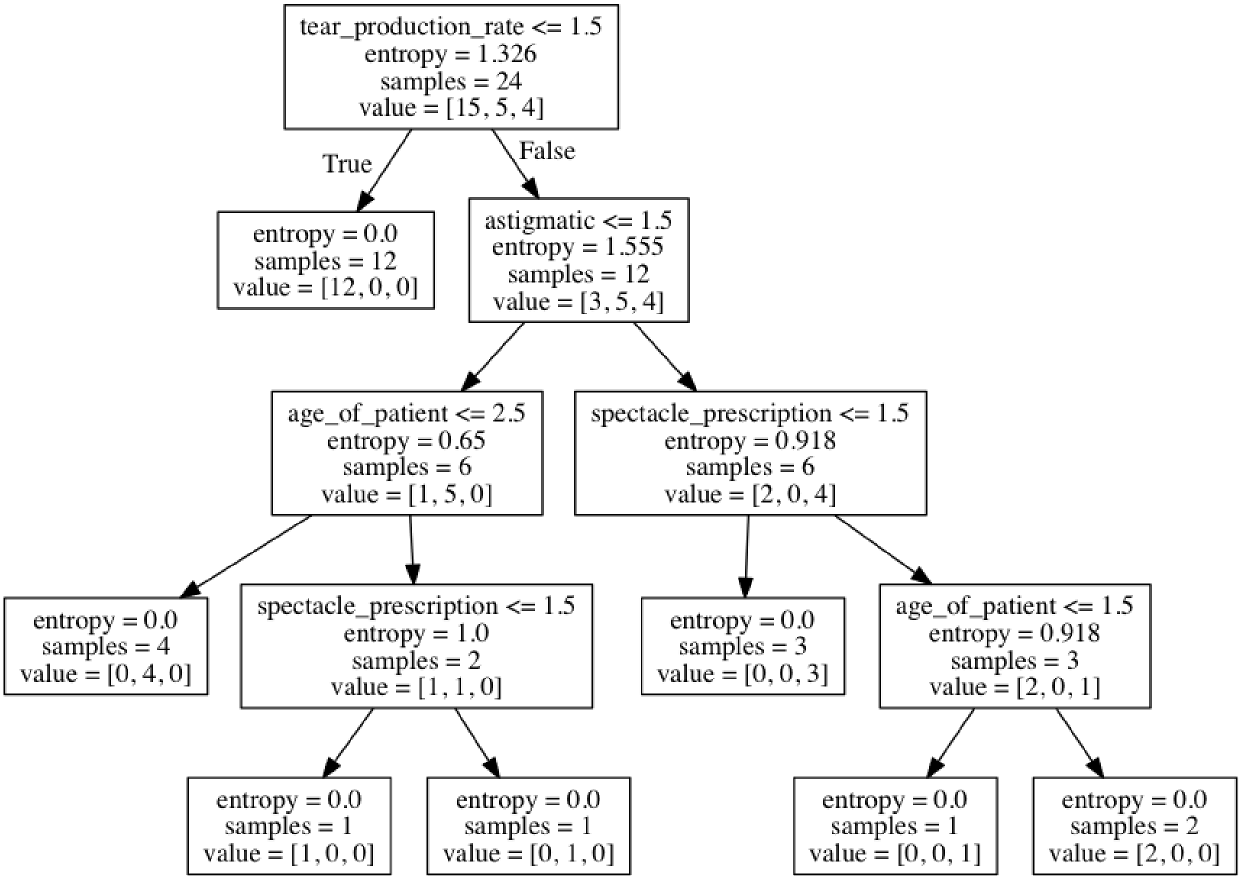
$E(Contact\ Lenses, Age\ of\ the\ Patient) = \frac{12}{24} * 0.9799 + \frac{12}{24} * 0.9183 = 0.9491.$

$E(Contact\ Lenses) = 0.9544.$ $G(Contact\ Lenses, Age\ of\ the\ Patient) = 0.9544 - 0.9491 = \mathbf{0.0053}.$

Tear Production Rate	Contact Lenses	No Contact Lenses	Total
Reduced	0	12	12
Normal	9	3	12
Total			24

$E(Reduced) = -\left(\frac{0}{12}\log_2\frac{0}{12}\right) - \left(\frac{12}{12}\log_2\frac{12}{12}\right) = -\infty$. $E(Normal) = -\left(\frac{9}{12}\log_2\frac{9}{12}\right) - \left(\frac{3}{12}\log_2\frac{3}{12}\right) = 0.8113$. $E(Contact\ Lenses, Age\ of\ the\ Patient) = \frac{12}{24} * -\infty + \frac{12}{24} * 0.8113 = -\infty$. $E(Contact\ Lenses) = 0.9544$. $G(Contact\ Lenses, Age\ of\ the\ Patient) = 0.9544 - -\infty = \infty$.

(a) (With Lenses data set)



(b) (With Lenses data set)
The attribute that was used as the first decision node of this tree generated by the DT classifier was tear_production_rate.

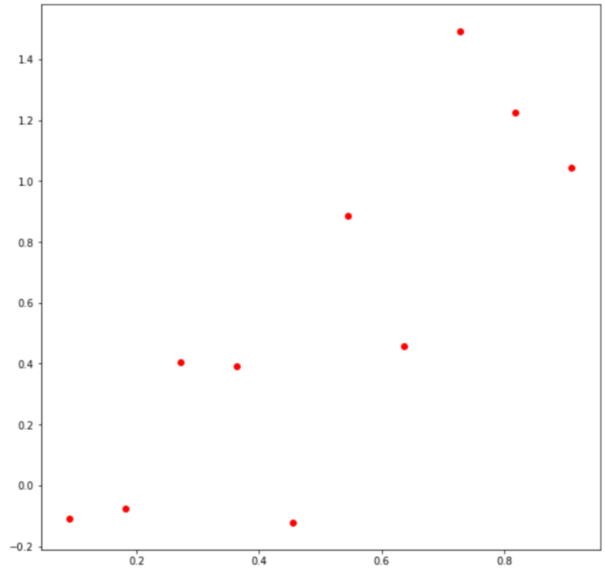
(d) Yes, my choice for the first decision node matched the one by the decision tree classifier.

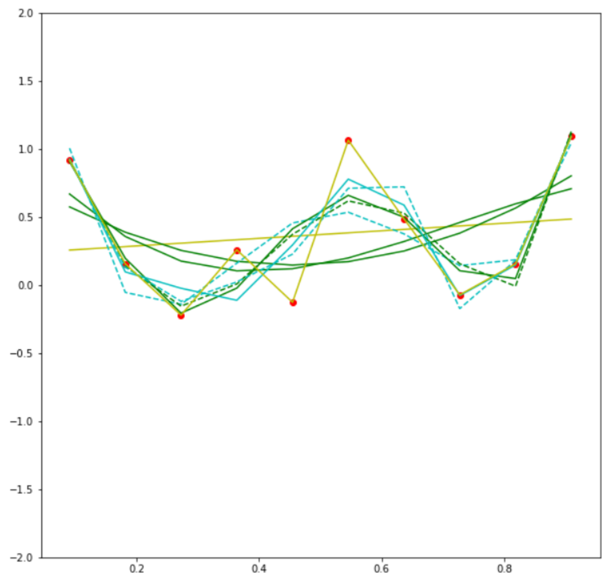
(e) It is possible to carry out parts (c) and (d) on the Iris data set, but there was no clear or obvious way how to partition the data. There were multiple options, including how to partition the four features – sepal length, sepal width, petal length, and petal width, as well as how to combine the three types of Iris flowers to make it a binary classifier. While it could have been possible to arbitrarily choose these partitions (or choose them through analyzing the data more closely), whatever was chosen would have affected the tree that was produced, which cannot reliably be compared to the one that the algorithm produced. Thus, I chose to use the Lenses data set since there was no question about how to partition the data or how to make it a binary classifier.

4 (b) Source Code:

```
import random
import numpy as np
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as pyplt
from decimal import Decimal
x = []
y = []
y_val=[]
noise = np.array([])
noise = np.append(noise,np.random.normal(0,0.5,10))
print (noise)
pos_inf = Decimal('Infinity')
#x = np.append(x,np.random.uniform(0,1,10))
for i in range (len(noise)):
    t = (i+1)/11.0
    val = (t*t) +(0.1*t) + noise[i]
    y.append(val)
    x.append(t)
def plot(x,y):
    n=len(x)
    plt.figure(figsize = (10,10))
    plt.scatter(x, y, color = 'red')
    w,h = n,n
    color=['r','b','g','y','c','m']
    pattern=['--','']
    y_vals = [[0 for m in range(w)] for n in range(h)]
    m=[]
    dof=[]
    remp=[]
    sf=[]
    rpen=[]
    for i in range(1,n):
        y_const=np.polyfit(x,y,i)
        m.append(len(y_const)-1)
        dof.append(len(y_const))
        sum=0
        p=len(y_const)/(n*1.0)
        r=0
        if(p==1):
            r=pos_inf
        else:
            r=1+((p/(1-p))*(np.log(n)))
        sf.append(r)
        for k in range(0,n):
            xi=x[k]
            y_val=0;
```

```
            for j in range(0,len(y_const)):
                temp=y_const[len(y_const)-j-1]*pow(xi,(j))
                y_val=y_val+temp
            y_vals[i][k]=y_val
            sum=sum+((y_val-y[i])*(y_val-y[i]))
        sum=sum/n
        remp.append(sum)
        value=0
        if(r==pos_inf):
            value=pos_inf
        else:
            value=(sum*r)
        rpen.append(value)
        t=random.choice(color)+random.choice(pattern)
        pyplt.plot(x,y_vals[i],t)
    axes = pyplt.gca()
    axes.set_ylim([-2,2])
    #pyplt.show()
    for i in range(0,9):
print(str(m[i])+"\t"+str(dof[i])+"\t"+str(sf[i])+"\t"+str(remp[i])+"\t
"+str(rpen[i])+"\n")
    pyplt.show()
plot(x,y)
""" Output
[ 0.9022396   0.10537983 -0.32139197  0.09139231 -0.37741751  0.71420537
  0.01894241 -0.67258188 -0.59429517  0.17536533]
1       2       1.57564627325    0.0518378508447  0.0816781164967

2       3       1.98682218271    0.392846365001   0.780515872381

3       4       2.53505672866    0.0592264259642  0.150142349655

4       5       3.30258509299    0.394188203619   1.30184008511

5       6       4.45387763949    0.643024407886   2.86395203193

6       7       6.37269855032    0.176495361296   1.12475173307

7       8       10.210340372     0.372895645626   3.80739146507

8       9       21.7232658369    0.229406021838   4.98344799699

9       10      Infinity 0.74026571443    Infinity
```





" " "

m	*DoF (m+1)*	*Remp*	*Penalized factor (r)*	*Rpen = r*Remp*
0	1	0.00190541524007	1.2558	0.0024
1	2	9.73750179882e-32	1.5756	1.5342e-31
2	3	2.00913011798e-31	1.9868	3.9917e-31
3	4	3.37731075048e-31	2.5351	8.5618e-31
4	5	5.83017512765e-31	3.3026	1.9254e-30
5	6	5.43574467504e-31	4.4539	2.4210e-30
6	7	2.34193081237e-32	6.3727	1.4924e-31
7	8	1.23259516441e-31	10.2103	1.2585e-30
8	9	1.04770588975e-31	21.7233	2.2760e-30

5 (c) Source Code:

```
import numpy as np
import matplotlib.pyplot as pyplt
import random
from decimal import Decimal
pos_inf = Decimal('Infinity')
mu, sigma = 0, 0.25
s = np.random.normal(mu, sigma, 10)
y=[]
x=np.array([])
x=np.append(x,np.random.uniform(0,1,10))
for i in range(len(s)):
    t=x[i]
    val=(t*t)+(0.1*t)+s[i]
    y.append(val)
    #x.append(t)
def plot(x,y,flag):
    n=len(x);
    print("Length is : "+str(n))
    toReturn=[]
    rEmp=[]
    for i in range(0,n):
        y_const=np.polyfit(x,y,i)
        sum=0
        for k in range(0,n):
            xi=x[k]
            y_val=0;
            for j in range(0,len(y_const)):
                temp=y_const[len(y_const)-j-1]*pow(xi,(j))
                y_val=y_val+temp
            sum=sum+((y_val-y[i])*(y_val-y[i]))
        sum=sum/n
        rEmp.append(sum)
    for i in range(0,n):
        print(str(i)+"\t"+str(rEmp[i])+"\n")
    return rEmp
#This is part 2
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
import numpy
kf = KFold(n_splits=5)
ct=1
#print("Length of x: "+str(len(x)))
for train in kf.split(x):
    newX=[]
    newY=[]
    for elt in train[0]:
        newX.append(x[elt])
        newY.append(y[elt])
    #doing LOO
```

```
    loo=LeaveOneOut()
    xs=loo.split(newX)
    ct2=1
    mins=[]
    print("fold #"+ str(ct))
    for a in xs :
        newXX=[]
        newYY=[]
        for b in a[0]:
            newXX.append(newX[b])
            newYY.append(newX[b])
        print("LeaveOneOut"+str(ct2))
        val=plot(newXX,newYY,False)
        mins.append(numpy.amin(val))
        ct2=ct2+1
    ct=ct+1
    print(numpy.amin(mins))
""" Output
fold #1
LeaveOneOut1
Length is : 7
0       0.000371843604189

1       0.0334168720015

2       0.0257115417172

3       0.0131463699141

4       0.057830355264

5       0.0127068495543

6       0.0182525008308

LeaveOneOut2
Length is : 7
0       0.0655687166439

1       0.0344031449497

2       0.0290350300137

3       0.0234400363358

4       0.0870732871969

5       0.0264713491735

6       0.0367046601587

LeaveOneOut3
Length is : 7
0       0.0750817346625

1       0.0214370088693

2       0.0302267037184
```

3 0.02146192032

4 0.0764777175167

5 0.0229148213592

6 0.0310163510197

LeaveOneOut4
Length is : 7
0 0.0727813131051

1 0.0223746207815

2 0.0365324305666

3 0.0223343700546

4 0.0793814630826

5 0.0241593324278

6 0.0327633630227

LeaveOneOut5
Length is : 7
0 0.0661333963502

1 0.0236830913264

2 0.0346711113911

3 0.0292378342775

4 0.0865534505657

5 0.0263830552998

6 0.0364856731013

LeaveOneOut6
Length is : 7
0 0.0496692964766

1 0.0159769094268

2 0.0183474758269

3 0.0149454945446

4 0.0152140178048

5 0.0211650966626

6 0.0353418035853

LeaveOneOut7
Length is : 7
0 0.0629417564916

1 0.0235035980822

2 0.0329132063483

3 0.0278519905687

4 0.0231722492178

5 0.0892937233414

6 0.0375081626812

LeaveOneOut8
Length is : 7
0 0.0587564625267

1 0.0223841396191

2 0.0296619665605

3 0.0251032517153

4 0.0219220975711

5 0.0921176608159

6 0.0261553932329

0.000371843604189
fold #2
LeaveOneOut1
Length is : 7
0 0.112202844613

1 0.0395751036731

2 0.0349858251727

3 0.031713522337

4 0.101661286928

5 0.0359014300764

6 0.0476968222712

LeaveOneOut2
Length is : 7
0 0.217606365859

1 0.0535767100661

2 0.0502856006674

3 0.0508847724399

4 0.13135760525

5 0.0570004992897

6 0.0713995775322

LeaveOneOut3
Length is : 7
0 0.186464712714

1 0.138209938718

2 0.0607515578276

3 0.0674738088471

4 0.164592967234

5 0.0766385519976

6 0.0951555898336

LeaveOneOut4
Length is : 7
0 0.182829150951

1 0.136031498846

2 0.0618642273546

3 0.0683462585817

4 0.1674967128

5 0.0778830630662

6 0.0969026018366

LeaveOneOut5
Length is : 7
0 0.172199502234

1 0.128047100424

2 0.0600029081791

3 0.0597626883866

4 0.174668700283

5 0.0801067859381

6 0.100624911915

LeaveOneOut6
Length is : 7
0 0.144910586295

1 0.0950771392828

2 0.0436792726149

3 0.0454703486538

4 0.0612259063319

5 0.0748888273009

6 0.0994810423992

LeaveOneOut7
Length is : 7
0 0.167025140093

```
1       0.12324017908

2       0.0582450031363

3       0.0583768446779

4       0.0691841377449

5       0.177408973059

6       0.101647401495

LeaveOneOut8
Length is : 7
0       0.160162008611

1       0.115870979699

2       0.0549937633485

3       0.0556281058245

4       0.0679339860982

5       0.180232910534

6       0.0798791238713

0.031713522337
fold #3
LeaveOneOut1
Length is : 7
0       0.107037781825

1       0.0930838725043

2       0.039497656623

3       0.0389234064619

4       0.112616825066

5       0.0437973851357

6       0.0565193753241

LeaveOneOut2
Length is : 7
0       0.210389484481

1       0.100482581034

2       0.0590065416261

3       0.0580946565647

4       0.142313143389

5       0.064896454349

6       0.080222130585
```

LeaveOneOut3
Length is : 7

0	0.19849861026
1	0.146996303016
2	0.0674825916553
3	0.066368142566
4	0.15690114312
5	0.0743265352519
6	0.0912142926976

LeaveOneOut4
Length is : 7

0	0.164891535511
1	0.128829261596
2	0.090791589643
3	0.0766618089877
4	0.186144075053
5	0.0880910348711
6	0.109666452025

LeaveOneOut5
Length is : 7

0	0.165786300439
1	0.129631227471
2	0.0913909914908
3	0.0783756599248
4	0.185624238422
5	0.0880027409974
6	0.109447464968

LeaveOneOut6
Length is : 7

0	0.13903247793
1	0.0966612663306
2	0.0647355440799
3	0.0706694780251
4	0.0684357904568
5	0.0827847823602
6	0.108303595452

```
LeaveOneOut7
Length is : 7
0       0.160709948431

1       0.124824306128

2       0.087740665049

3       0.0781961666805

4       0.0763940218697

5       0.188364511197

6       0.110469954548

LeaveOneOut8
Length is : 7
0       0.153979188091

1       0.117455106747

2       0.0819335468773

3       0.0770767082175

4       0.075143870223

5       0.191188448672

6       0.0877750789306

0.0389234064619
fold #4
LeaveOneOut1
Length is : 7
0       0.0693826916289

1       0.0537054064967

2       0.0353460277255

3       0.0247736739399

4       0.0244225822639

5       0.044271383311

6       0.0645671379683

LeaveOneOut2
Length is : 7
0       0.155963377253

1       0.0611041150268

2       0.0548549127285

3       0.0387752803329

4       0.0397223577586
```

```
5       0.0653704525243

6       0.0882698932293

LeaveOneOut3
Length is : 7
0       0.145749464725

1       0.0958791051585

2       0.0633309627578

3       0.0439472390563

4       0.0456731529176

5       0.0748005334272

6       0.0992620553418

LeaveOneOut4
Length is : 7
0       0.117174795277

1       0.077712063738

2       0.0514131236355

3       0.0449335120045

4       0.0489966412141

5       0.0885650330465

6       0.11771421467

LeaveOneOut5
Length is : 7
0       0.129783387119

1       0.088676867908

2       0.0590738364997

3       0.07197794857

4       0.0501883149188

5       0.0850085052322

6       0.112025905531

LeaveOneOut6
Length is : 7
0       0.126753280617

1       0.0864984280363

2       0.0576742330635

3       0.0729155604823

4       0.0470627976214
```

```
5       0.0862530163008

6       0.113772917534

LeaveOneOut7
Length is : 7
0       0.113654012066

1       0.0737071082703

2       0.0483621990415

3       0.074044537783

4       0.0434435734031

5       0.0478136017691

6       0.118517717192

LeaveOneOut8
Length is : 7
0       0.108005218077

1       0.066337908889

2       0.0425550808698

3       0.0729250793199

4       0.0401923336153

5       0.0450648629157

6       0.0882490771059

0.0244225822639
fold #5
LeaveOneOut1
Length is : 7
0       0.076537982005

1       0.0672530828524

2       0.0415737646736

3       0.0343302596307

4       0.0331944957258

5       0.0402214628761

6       0.138169028624

LeaveOneOut2
Length is : 7
0       0.166603591016

1       0.0746517913824

2       0.0610826496766
```

3 0.0483318660237

4 0.0484942712205

5 0.0593927129789

6 0.167865346947

LeaveOneOut3
Length is : 7
0 0.156041215242

1 0.111866024231

2 0.0695586997059

3 0.0535038247471

4 0.0544450663795

5 0.0676661989802

6 0.182453346678

LeaveOneOut4
Length is : 7
0 0.126420840165

1 0.0936989828106

2 0.0649607999911

3 0.0544900976952

4 0.057768554676

5 0.0779598654019

6 0.211696278611

LeaveOneOut5
Length is : 7
0 0.139504981827

1 0.104663786981

2 0.0726215128553

3 0.0782056855182

4 0.0589602283807

5 0.0759817493862

6 0.201100708931

LeaveOneOut6
Length is : 7
0 0.136362779292

1 0.102485347109

2 0.0712219094191

```
3       0.0791432974304

4       0.0566193833122

5       0.0768541991208

6       0.204004454497

LeaveOneOut7
Length is : 7
0       0.127204454859

1       0.0945009486864

2       0.0655602018389

3       0.0804517679753

4       0.0547580641367

5       0.0579713589398

6       0.21117644198

LeaveOneOut8
Length is : 7
0       0.103915510514

1       0.0615309875456

2       0.038904754428

3       0.0727455860757

4       0.0384344285725

5       0.0436790192069

6       0.069733846871

0.0331944957258
"""
```

Fold number	*Optimal m*	*Test error*
1	0	0.000371843604189
2	3	0.031713522337
3	3	0.0389234064619
4	4	0.0244225822639
5	4	0.0331944957258