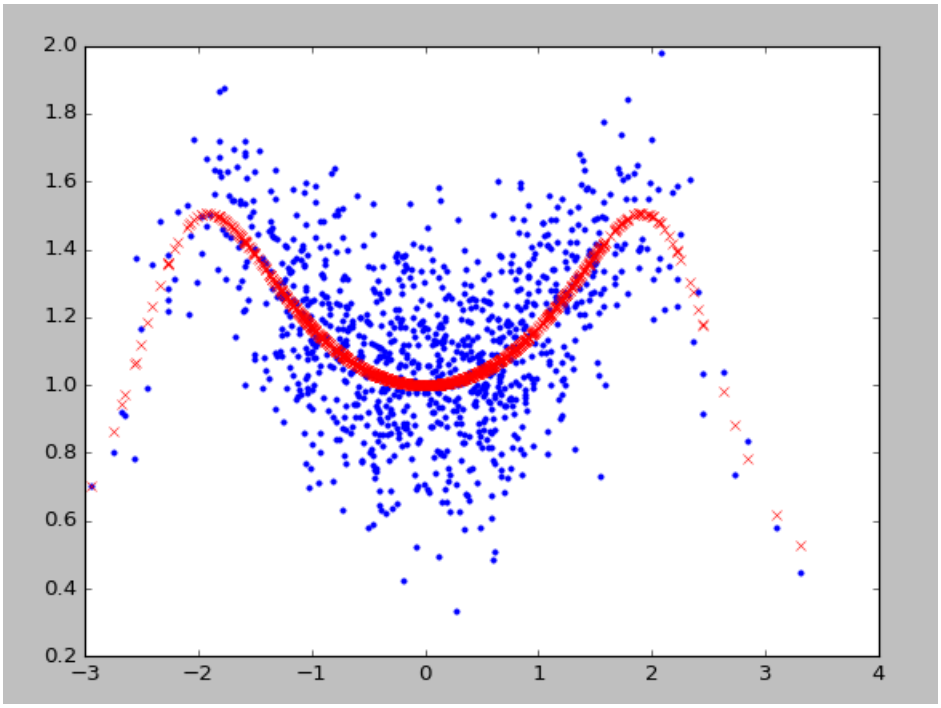
Jack Barry

AI_HW4

Professor Rivas



1NN:

kscore:

[-0.15064080536556812, -0.14663701805299967, -0.54214813595635247, -0.47039076934481683, -0.11627676122758857, -0.45036607101984605, -0.35340169266436544, -0.29309734191289083, -0.4259817975074276, -0.69820179690302098]

scores:

[-0.15064081 -0.14663702 -0.54214814 -0.47039077 -0.11627676 -0.45036607

-0.35340169 -0.29309734 -0.4259817 -0.6982018]

getindex

[-0.14863891170928389]

bestk:

[-0.15064080536556812, -0.14863891170928389, -0.27980865312497344, -0.32745418217993427, -0.28521869798946514, -0.3127432601611953, -0.3185516076616482, -0.3153698244430535, -0.32766003256131732, -0.36471420899548768]

Process finished with exit code 0

3NN:

scores:

[-0.71863482 -0.19327347 -0.55116568 -0.06492079 0.04557421 -0.73319311

 -0.06876029 -0.07291749 -0.11590666 -0.21957435]

kscore:

[-0.2794291475360879 4, 0.086249660252917137, -0.1664806594048498, 0.20192848372008876, 0.07325452425732526, -0.47174071675057699, 0.10262830953520541, 0.077368496650865315, 0.0613430377737465, 0.062352664263966025]

scores:

[-0.27942915 0.08624966 -0.16648066 0.20192848 0.07325452 -0.47174072

  0.10262831 0.0773685   0.06134304 0.06235266]

getindex

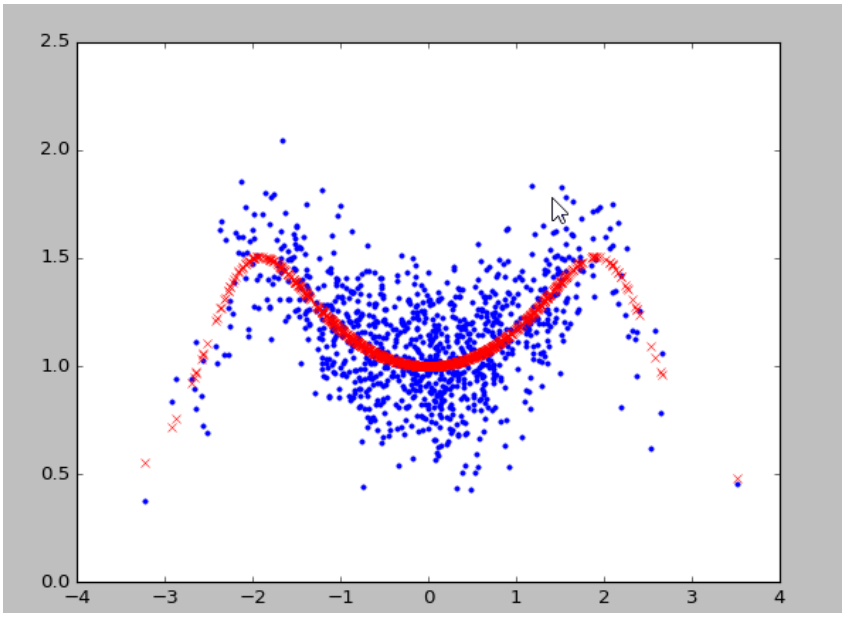
[-0.016895427742121317]

bestk:

[-0.71863482465560558, -0.45595414869243001, -0.48769132714107327, -0.38199869291709171, -0.29648411184246865, -0.36926894455437509, -0.32633913696645456, -0.29466143138243117, -0.27479979037516172, -0.26927724612103715, -0.27942914753608794, -0.096589743641585402, -0.11988671556267354, -0.03943291574198296, -0.016895427742121317, -0.092702975910197272, -0.064798506560854036, -0.047027631159389113, -0.034986445722374047, -0.02525253472374004]

Process finished with exit code 0

5NN:



kscore:

[-0.57302954225624902, -0.45828113968198991, -0.47438944141340755, -0.04679391096734653, -0.48645504121608435, -0.10272232962016138, 0.18824695817725345, -0.35575602231834647, -0.39718973177667322, -0.55402581729026523]

scores:

[-0.57302954 -0.45828114 -0.47438944 -0.04679391 -0.48645504 -0.10272233

 0.18824696 -0.35575602 -0.39718973 -0.55402582]

kscore:

[-0.21885783646913759, -0.099745592758274659, -0.11694343527384299, 0.10867758531006311, -0.19302338112492956, 0.095658570604665427, 0.27158949823381018, -0.0067138521826517472, -0.10089879513857848, -0.26704094834933167]

scores:

[-0.21885784 -0.09974559 -0.11694344 0.10867759 -0.19302338 0.09565857

 0.2715895 -0.00671385 -0.1008988 -0.26704095]

kscore:

[-0.093594469068774, -0.089038753717980201, -0.022630627304968787, 0.13388265211574157, -0.057381230794172344, 0.14901810378070335, 0.33590420548829281, 0.079932110383807542, -0.068464406889322094, -0.17542260879873608]

scores:

[-0.09359447 -0.08903875 -0.02263063 0.13388265 -0.05738123 0.1490181

  0.33590421 0.07993211 -0.06846441 -0.17542261]

kscore:

[-0.033326488277719557, -0.044912047745245154, 0.050292334364093527, 0.15358335274396129, -0.0063256038019126057, 0.24034885043556298, 0.33743526505516752, 0.10900675409656868, -0.009763640245892713, -0.14063879389384182]

scores:

[-0.03332649 -0.04491205 0.05029233 0.15358335 -0.0063256   0.24034885

  0.33743527 0.10900675 -0.00976364 -0.14063879]

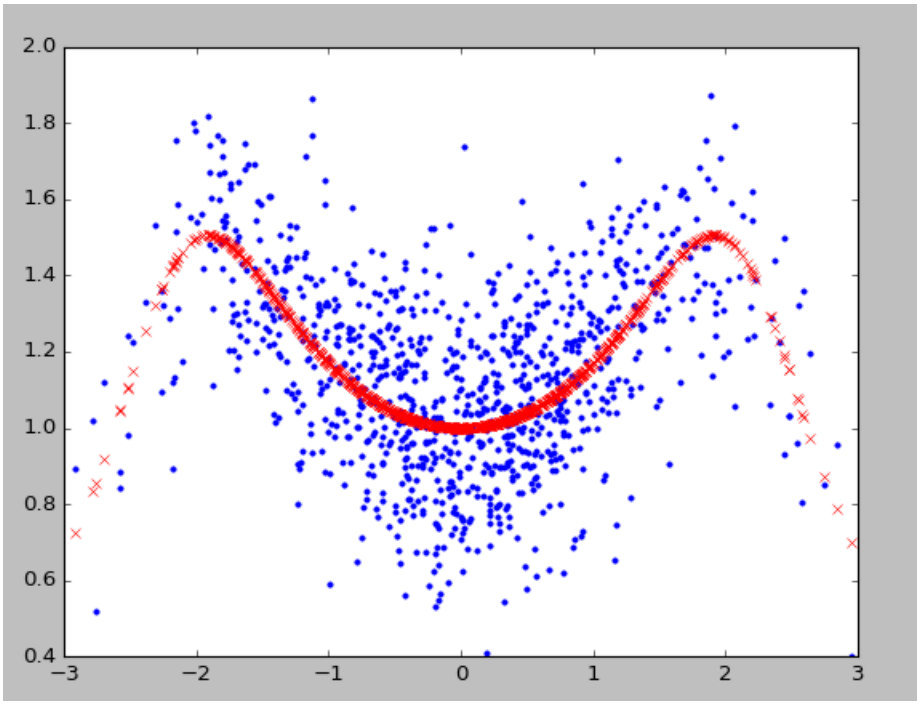
getindex

[0.10076280210880958]

bestk:

[-0.57302954225624902, -0.51565534096911947, -0.50190004111721553, -0.38812350857974826, -0.40778981510701551, -0.35694523419253982, -0.27906063528256936, -0.2886475586620415, -0.30070780011922277, -0.32603960183632702, -0.21885783646913759, -0.15930171461370612, -0.14518228816708509, -0.081717319797798033, -0.10397853206322434, -0.070705681618576044, -0.021806370211092298, -0.019919805457537229, -0.028917470977652921, -0.052729818714820798, -0.093594469068774, -0.091316611393377101, -0.068421283363907667, -0.017845299493995354, -0.025752485754030753, 0.0033759458350915983, 0.050879982928406058, 0.054511498860331242, 0.040847509332591986, 0.019220497519459177, -0.033326488277719557, -0.039119268011482355, -0.0093154005529570618, 0.031409287771272526, 0.023862309456635498, 0.059943399619790082, 0.099585094681986855, 0.10076280210880958, 0.088482086291620438, 0.06556999827307422]

999NN



kscore:

[0.29327588598532739, 0.21689469319524016, 0.34434116555146177, 0.14747839216428626, 0.39053103462244898, 0.31033260644262373, 0.26360154451910833, 0.27928430431262785, 0.29162980740116839, 0.064944937053684759]

scores:

[0.29327589 0.21689469 0.34434117 0.14747839 0.39053103 0.31033261

  0.26360154 0.2792843   0.29162981 0.06494494]

Traceback (most recent call last):

  File "C:/Users/Jack/PycharmProjects/AI_HW4/hw4.py", line 35, in <module>

    kscore.append(clf.score(X_test, y_test))

  File "C:\Users\Jack\Miniconda2\lib\site-packages\sklearn\base.py", line 386, in score

```
    return r2_score(y, self.predict(X), sample_weight=sample_weight,
  File "C:\Users\Jack\Miniconda2\lib\site-packages\sklearn\neighbors\regression.py", line 144, in predict
    neigh_dist, neigh_ind = self.kneighbors(X)
  File "C:\Users\Jack\Miniconda2\lib\site-packages\sklearn\neighbors\base.py", line 343, in kneighbors
    (train_size, n_neighbors)
ValueError: Expected n_neighbors <= n_samples,  but n_samples = 900, n_neighbors = 901
```

The code I used to complete this homework is found below

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from numpy import genfromtxt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn import neighbors

def genDataSet(N):
    X= np.random.normal(0,1,N)
    ytrue = (np.cos(X) + 2)/(np.cos(X*1.4)+2)
    noise = np.random.normal(0,0.2,N)
    y = ytrue+noise
    return X, y, ytrue

X, y, ytrue = genDataSet(1000)
plt.plot(X, y, '.')
plt.plot(X, ytrue, 'rx')
plt.show()

X = X.reshape(len(X), 1)
bestk = []
kc = 0

for n_neighbors in range(1,999):
    kf = KFold(n_splits=10)
    kscore = []
    k = 0
    for train, test in kf.split(X):
        X_train, X_test, y_train, y_test = X[train], X[test], y[train], y[test]

        #we create an instance of Neighbors regressor and fit the data.
        clf = neighbors.KNeighborsRegressor(n_neighbors, weights='distance')
```

```
        clf.fit(X_train, y_train)
        kscore.append(clf.score(X_test, y_test))
        k = k + 1
        #print n_neighbors
        bestk.append(sum(kscore) / len(kscore))
        kc += 1
# to do here: given this array of E_outs in CV, find the max, its
# corresponding index, and its corresponding value of n_neighbors
    #this method allows me to control cv
    scores = cross_val_score(clf, X, y, cv=10)
    print 'kscore:'
    print kscore
    print 'scores:'
    print scores


nbk = sorted(bestk,reverse = True)
getindex = [nbk[0], nbk[1], nbk[2]]
print 'getindex'
print getindex
#eout = clf.score(clf,X,y)
#print 'eout:'
#print eout

#idx = sorted(range(len(bestk), key = bestk.__getiten__))
#print (idx[0] + 2)+1
#print (idx[1] + 2)+1
#print (idx[2] + 2)+1
print 'bestk:'
print bestk
```