Amy Pitts
Assignment Number 2
Instructor: Dr. Pablo Rivas

Math 440L: Machine Learning
Due: February 30, 2019

Exercise List

Chapter 2,3

Exercise 1. *Use sklearn's implementation of k-Nearest Neighbors for regression purposes, which is found in sklearn.neighbors.KNeighborsRegressor. You will find the best value of k using 10 fold CrossValidation (CV), which is found in sklearn.model_selection.KFold.*

a) *You will modify the python code below to generate 1000 data points, or alternatively you could use part of your semester project dataset if it is related to regression.*

```
import numpy as np
from matplotlib import pyplot as plt

def genDataSet (N) :
    x = np . random . normal (0 , 1 , N)
    ytrue = (np.cos ( x ) + 2) / (np.cos ( x * . 4 ) + 2)
    noise = np . random . normal (0 , 0 . 2 , N)
    y = ytrue + noise
    return x , y , ytrue

x , y , ytrue = genDataSet(100)
plt.plot(x , y , '.')
plt.plot(x , ytrue , 'rx')
plt.show( )
```

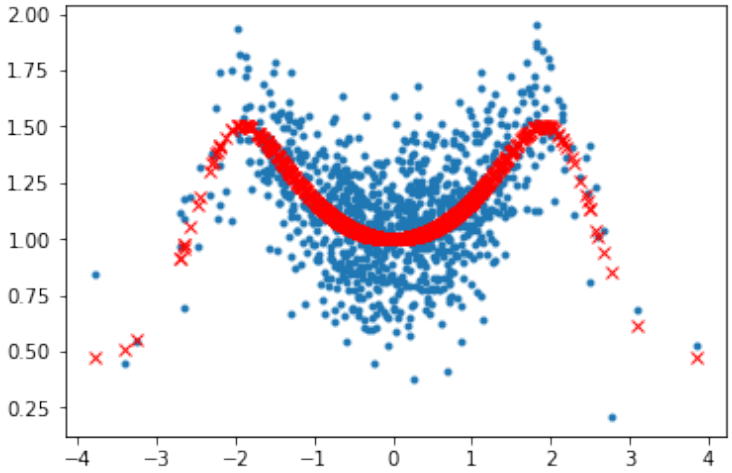
b) *Using 10-fold CV, you will report the three best values of k-neighbors that yield the best CV E_{out}. You will vary the values of k in the following range: $k = 1, 3, 5, \ldots 2[\frac{N+1}{2}] - 1$.*

c) *You will report the best CV E_{out}.*

Solution. **a)**

```
import numpy as np

from matplotlib import pyplot as plt



def genDataSet (N) :
```

```
    x = np . random . normal (0 , 1 , N)
    ytrue = (np.cos ( x ) + 2) / (np.cos ( x * . 4 ) + 2)
    noise = np . random . normal (0 , 0 . 2 , N)
    y = ytrue + noise
    return x , y , ytrue


x , y , ytrue = genDataSet(1000) #changed this number to 1000
plt.plot(x , y , '.')
plt.plot(x , ytrue , 'rx')
plt.show( )
```



b) and c)

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
kf = KFold(n_splits=10)
```

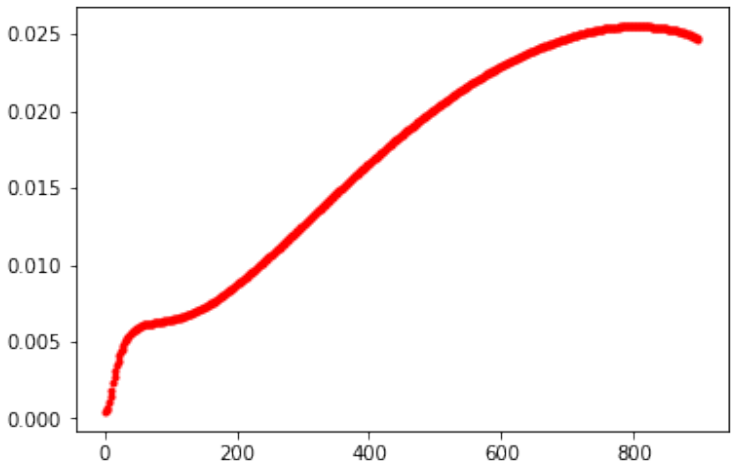
```
X = np.array([x,y]).T

kf.get_n_splits(X)


msek = {}

bestk= 0

bestmse = 100000

for k in np.arange(1,2*np.floor(((len(ytrue)*0.9)+1)/2)-1,2):

  print(k)

  mse = []

  for train_index, test_index in kf.split(X):

    X_train, X_test =     X[train_index],     X[test_index]

    y_train, y_test = ytrue[train_index], ytrue[test_index]

    neigh = KNeighborsRegressor(n_neighbors = int(k))

    neigh.fit(X_train, y_train)

    predict = neigh.predict(X_test)

    mse.append(mean_squared_error(y_test, predict))

  print(np.mean(mse))

  msek[k] = np.mean(mse)

  #part c

  if bestmse > msek[k]:

      bestmse = msek[k]

      bestk = k
```

```
        print(k)

        print(bestmse)

    plt.plot(k,msek[k],'r.') #mse


plt.show()

print(bestk)
```

The best three best k using mse as an evaluation metric are:

k	1.0	3.0	5.0
MSE	0.00045241177636707794	0.000512687006146458	0.0006638327509173002

This can also be seen by plotting the MSE by k.



The smallest MSE is the best, making smaller ks better then larger ks.

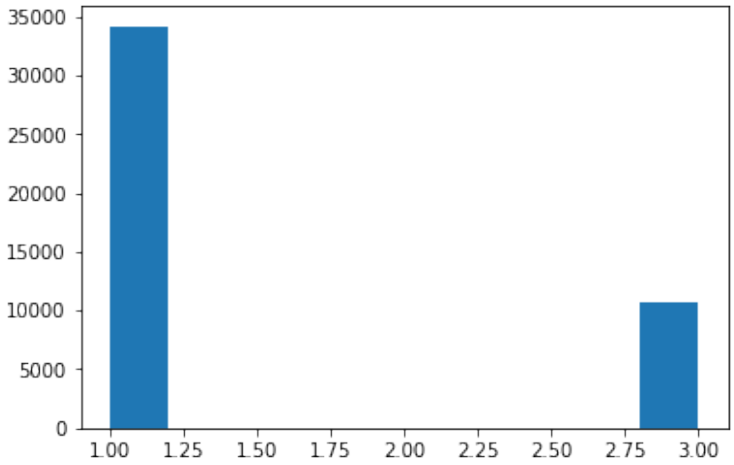
Exercise 2. *Using the same dataset you just tried in the previous problem, repeat the experiment 100 times storing the best three k number of neighbors in every single trial, and at the end of all trials plot a histogram of all the values of k that you saved.*

Solution.

```
from sklearn.neighbors import KNeighborsRegressor
```

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold


kf = KFold(n_splits=10)
bestkall = []
for i in range(100):
  x, y, ytrue = genDataSet(1000)
  X = np.array([x,y]).T
  kf.get_n_splits(X)


  msek = {}
  bestk= 0
  bestmse = 100000
  for k in np.arange(1,2*np.floor(((len(ytrue)*0.9)+1)/2)-1,2):
    mse = []
    for train_index, test_index in kf.split(X):
      X_train, X_test =     X[train_index],     X[test_index]
      y_train, y_test = ytrue[train_index], ytrue[test_index]


      neigh = KNeighborsRegressor(n_neighbors = int(k))
      neigh.fit(X_train, y_train)
      predict = neigh.predict(X_test)
```

```
        mse.append(mean_squared_error(y_test, predict))
    msek[k] = np.mean(mse)
    if bestmse > msek[k]:
        bestmse = msek[k]
        bestk = k
        print(k)
        print(bestmse)
    bestkall.append(bestk)
plt.hist(bestkall)
```



Exercise 3. ***Experiment with k-means for color quantization.***
Using sklearn's implementation of k-means, find the best color clustering for an image of your choice. A good portrait of yourself could be fun (just sayin'). Color quantization is the science behind compression of images. The idea is to represent an image with fewer colors than the original. The experiment consists of the following steps:

a) *Download to your computer the file hw3.kmeans.img.py*

b) *In the same folder where you downloaded the program, save a copy of your picture for experimentation.*

c) *Go to line 23 and set n_colors with your choice of a number of colors between 2 and 64. This number is actually the number of clusters (or k) we are searching for in an*

unsupervised manner using kmeans.

d) Then go to line 25 and replace the image file name with the name of your image file. This is where your image is read into a numpy array.

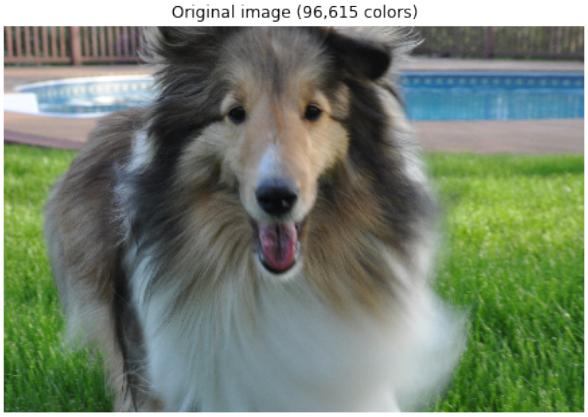
e) Run the program. Observe the result. If the result does not look funny to you, repeat from 3.(c) until it does. Then, report your result image along with your answers to the following questions:

i) Explain what happens when you increase or decrease the value of n_colors?

ii) Explain in what other possible applications do you think this can be useful?

iii) Why do you think the resulting picture was funny at the end?

Solution. This is my puppy Penny



i) When I increase the n_color there are more colors in the picture, when I decrease it there are less colors. When there is only 2 n_colors then the picture has only two colors which end up being the averages of two groups of colors related to eachother

ii) This would be helpful in displaying images on devices that do not have the ability to represent a lot of different colors. It decreases the complexity and it only takes away some of the detail. One place were this is helpful is on old video game hand held devices.

iii) The picture looks cool to me. It is more cartoon-ish but less bright with color. With less group the color turns more grayish green and more washed out.

Exercise 4. ***Neural networks: The MLP***

a) Download the python program hw3.MLP.sol.py which implements a 10 fold cross validation approach to find the best number of neurons and the best learning parameter η (eta) in an MLP for regression. For learning purposes you could download hw3.MLP.py first, which is a simple implementation of the MLP.

b) Download the python program hw3_4_a_gendata.py which generates random data.

c) Run the program in 4.(a) for 1,000 samples, and then take note of the best number of neurons and η. Go here https://goo.gl/forms/QFmaNWYaFLcPWdim2 and report your results. You can do it as many times as you want, but at least one is required.

d) Explain your results. What do you think is happening? What is your interpretation of the number of neurons with respect to the performance of the network?

e) (extra credit) Repeat 4.(c)-(d) but for 10,000 samples.

Solution. **a)**

```
import hw3_4_a_gendata
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import KFold
# generates data & split it into X (training input) and y (target output)
X, y = hw3_4_a_gendata.genDataSet(100)
neurons = 1000  # <- number of neurons in the hidden layer
```

```
eta = 0.01        # <- the learning rate parameter
# here we create the MLP regressor
mlp =  MLPRegressor(hidden_layer_sizes=(neurons,), verbose=True, learning_rat
# here we train the MLP
mlp.fit(X, y)
# E_out in training
print("Training set score: %f" % mlp.score(X, y))
# now we generate new data as testing set and get E_out for testing set
X, y = hw3_4_a_gendata.genDataSet(100)
print("Testing set score: %f" % mlp.score(X, y))
ypred = mlp.predict(X)


plt.plot(X[:, 0], X[:, 1], '-')
plt.plot(X[:, 0], y, '-r')
plt.plot(X[:, 0], ypred, '-k')
plt.show()
```

b)

```
import hw3_4_a_gendata
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import KFold
```

```
# number of samples
N = 1000

# generate data & split it into X (training input) and y (target output)
X, y = hw3_4_a_gendata.genDataSet(N)

# linear regression solution
w=np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(y)



#neurons <- number of neurons in the hidden layer
#eta <- the learning rate parameter

bestNeurons=0
bestEta=0
bestScore=float('-inf')
score=0
for neurons in range(1,101,1):
  for eta in range(1,11,1):
    eta=eta/10.0
    kf = KFold(n_splits=10)
```

```
    cvscore=[]
    for train, validation in kf.split(X):
      X_train, X_validation, y_train, y_validation = X[train, :], X[validation
      # here we create the MLP regressor
      mlp =  MLPRegressor(hidden_layer_sizes=(neurons,), verbose=False, learn
      # here we train the MLP
      mlp.fit(X_train, y_train)
      # now we get E_out for validation set
      score=mlp.score(X_validation, y_validation)
      cvscore.append(score)

    # average CV score
    score=sum(cvscore)/len(cvscore)
    if (score > bestScore):
      bestScore=score
      bestNeurons=neurons
      bestEta=eta
      print("Neurons " + str(neurons) + ", eta " + str(eta) + ". Testing set

# here we get a new training dataset
X, y = hw3_4_a_gendata.genDataSet(N)
# here we create the final MLP regressor
```

```
mlp = MLPRegressor(hidden_layer_sizes=(bestNeurons,), verbose=True, learning
# here we train the final MLP
mlp.fit(X, y)
# E_out in training
print("Training set score: %f" % mlp.score(X, y))
# here we get a new testing dataset
X, y = hw3_4_a_gendata.genDataSet(N)
# here test the final MLP regressor and get E_out for testing set
ypred=mlp.predict(X)
score=mlp.score(X, y)
print("Testing set score: %f" % score)
plt.plot(X[:, 0], X[:, 1], '.')
plt.plot(X[:, 0], y, 'rx')
plt.plot(X[:, 0], ypred, '-k')
ypredLR=X.dot(w)
plt.plot(X[:, 0], ypredLR, '--g')
plt.show()
```

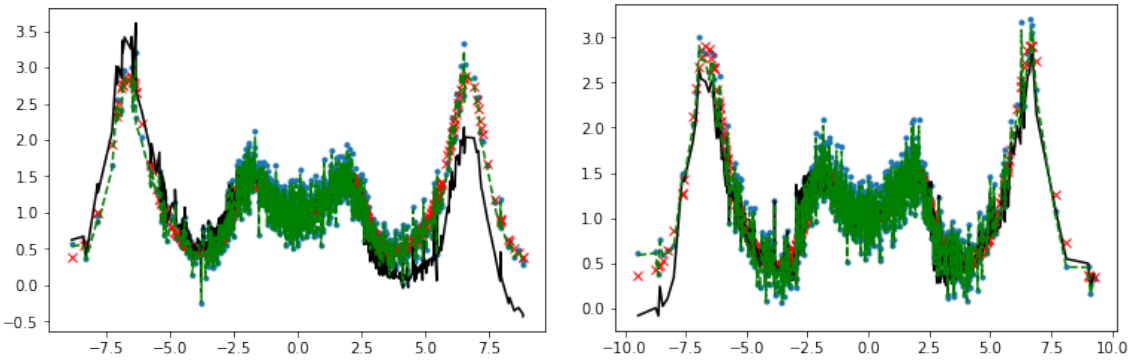
c) First Run: Neurons 34, eta 0.2. Testing set CV score: -1.055029

Second Run: Neurons 36, eta 0.2. Testing set CV score: -1.182477

d) This is a relatively low number of neurons compared to other amounts classmates were getting. The testing CV score for 34 neurons is still negative 1.06 which still relatively small. As the number of neurons goes up the performance also goes up until it reaches a

plato. The second run had 36 Neurons and at eta of 0.2.

Training set score: 0.900475, Testing set score: 0.869687



e) Neurons 20, eta 0.1. Testing set CV score: -1.505540.

This is also a relatively small number of Neurons comparatively. Even though it seems small it looks to do a okay job looking at the graph below. The green line seems to represent the data to be able to use to predict it. It is just completely green in the center which means all the data values are accounted for.

Training set score: 0.935958

Testing set score: 0.933730