Pouya Khani 99210283

Importing essential Libraries

```
import numpy as np
import pandas as pd
import plotly.express as pl
```

Importing Data

```
train_data = pd.read_csv("./train.csv")
test_data = pd.read_csv("./test.csv")
```

Preprocessing

```
del train_data['Id']
del test_data['Id']
train_data = train_data.dropna(axis=1, how='any')
test_data = test_data.dropna(axis=1, how='any')
for col in train_data.select_dtypes(include=['object']):
    train_data[col] = train_data[col].astype('category')
    train_data[col] = train_data[col].cat.codes
for col in test_data.select_dtypes(include=['object']):
    test_data[col] = test_data[col].astype('category')
    test_data[col] = test_data[col].cat.codes
trainX = train_data.iloc[:,0:60]
trainY = train_data.iloc[:,60]
testX = test_data.iloc[:,:]
```

Normalization

```
for column in trainX:
    trainX[column] = (trainX[column] - trainX[column].min()) / (trainX[column].max() - trainX[column].min())

for column in testX:
    testX[column] = (testX[column] - testX[column].min()) / (testX[column].max() - testX[column].min())
```

Split train data for validation data

```
cut_index_x = int(0.85 * len(trainX))
cut_index_y = int(0.85 * len(trainY))

validX , validY = trainX[cut_index_x:] , trainY[cut_index_y:]
trainX , trainY = trainX[:cut_index_x] , trainY[:cut_index_y]
validY = validY.to_numpy()
```

box and gaussian kernel function - weight calculation function - prediction function using weight vector

```
def kernel_function(h, x, xi,kernel_type):
  if kernel_type == 2:
    output = (1/( np.sqrt(np.pi * 2))*h) * np.exp(-1*((0.5*np.power(np.linalg.norm(xi-x),2))/h))
    return output
  else:
    if np.linalg.norm(xi-x) <= h:
      return 1
    else:
      return 0

def weights( h, input_vector, test_vector,kernel_type):
    w = np.zeros(shape=(len(input_vector),1))
    k = np.zeros(shape=(len(input_vector),1))
    k_sum = 0
    i=0
    while i < len(input_vector):
        k[i] = kernel_function(h, input_vector.iloc[i,:] , test_vector,kernel_type)
        k_sum = np.add(k_sum , k[i])
        i = np.add(i,1)
    j=0
    while j < len(input_vector):
      if k_sum == 0:
        w[j] = 0
      else:
        w[j] = np.divide(k[j],k_sum)
      j = np.add(j,1)
    return w

def y_pred(h, trainX, trainY , x_i,kernel_type):
    w = weights(h, trainX, x_i,kernel_type)
    y = np.sum(np.dot(trainY,w))
    return y
```

checking predict value for preventing NaN outputs

```
def prediction(h,test,kernel_type):
    y = np.zeros(shape=(len(test),1))
    counter = 0
    while counter < len(test):
        temp = y_pred(h,trainX,trainY, test.iloc[counter,:],kernel_type)
        if temp == 0:
          y[counter] = np.mean(trainY)
        else:
          y[counter] = temp
        counter = np.add(counter,1)
    return np.array(y)
```

RMSE calculation function based on predict labels and real labels

```
def mseCalculator(predict_labels,validY):
  rows,cols = predict_labels.shape
  temp = np.empty((cols,rows))
  i=0
  while i < cols:
    temp[i,:] = np.square(np.subtract(predict_labels[:,i],validY))
    i = np.add(i,1)
  mse = np.sqrt(np.divide(np.sum(temp,axis=1),rows))
  return mse
```

predict labels using gaussian kernel and plot result

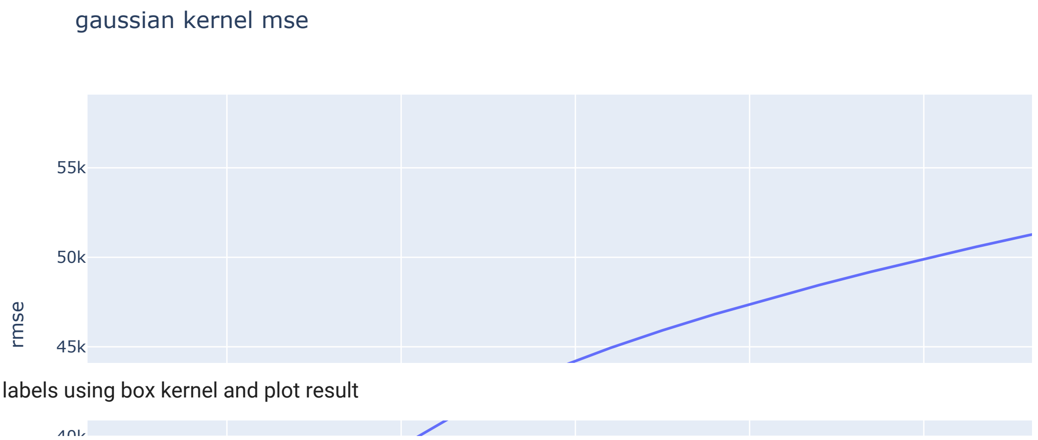
```
predict_labels_gaussian = np.empty((len(validX),0))
for i in np.arange(0.02,1,0.03):
  predict_labels_gaussian = np.append(predict_labels_gaussian,prediction(i,validX,2),axis=1)

rows,cols = predict_labels_gaussian.shape
mse_gauss = np.empty((cols,1))
mse_gauss = mseCalculator(predict_labels_gaussian,validY)

bandwitch1 = np.arange(0.02,1,0.03)
temp = np.vstack((bandwitch1,mse_gauss))

temp2 = pd.DataFrame({'parameter':np.array(temp)[0,:],'rmse':np.array(temp)[1,:]})

fig = pl.line(temp2,x='parameter',y='rmse',title='gaussian kernel mse')
fig.show()
```



predict labels using box kernel and plot result

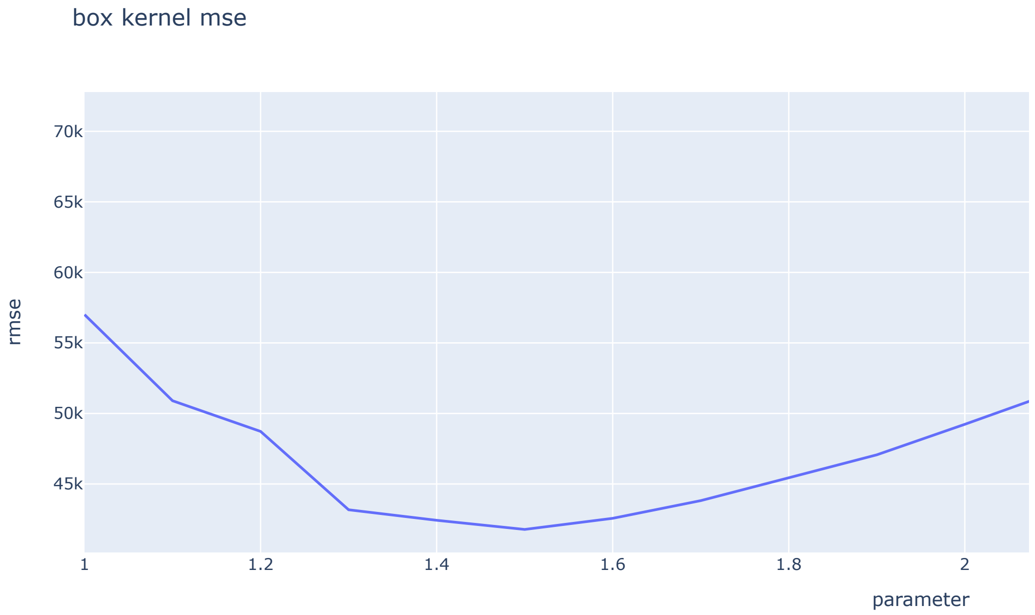
```
predict_labels_box = np.empty((len(validX),0))
for i in np.arange(1,3,0.1):
  predict_labels_box = np.append(predict_labels_box,prediction(i,validX,1),axis=1)

rows,cols = predict_labels_box.shape
mse_box = np.empty((cols,1))
mse_box = mseCalculator(predict_labels_box,validY)

bandwitch2 = np.arange(1,3,0.1)
temp = np.vstack((bandwitch2,mse_box))

temp2 = pd.DataFrame({'parameter':np.array(temp)[0,:],'rmse':np.array(temp)[1,:]})

fig = pl.line(temp2,x='parameter',y='rmse',title='box kernel mse')
fig.show()
```



● ×