EN 685.621 HW4 - Noboru Hayashi

Q1:

Implemented RBF NN for iris dataset:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

def multiDiag(X1, X2):
    r1, c1 = X1.shape
    r2, c2 = X2.shape

    X1_tmp = X1.T
    X1_tmp = X1_tmp.flatten('F')
    X2_tmp = X2.flatten('F')

    X = (X1_tmp * X2_tmp).reshape(c1, r1)

    if len(X) > 1:
        xDiag = sum(X).T
    else:
        xDiag = X.T

    return xDiag

class RBF:

    def __init__ (self, X, y, input_spread):
        self.X = X
        self.y = y
        self.input_spread = input_spread

    def train(self):
        # initialize the dataset ~ O(n*d)
        y = self.y
        n, d = self.X.shape
        X = self.X.T
        H = np.zeros([n, n])
        spread = np.sqrt(-np.log(.5))/self.input_spread

        # the loop cost O(n)
        # multiDiag() operates element-wise multiplication for matrix D and D', O(n*d)
        # so in total O(n^2*d)
```

```
        for j in range(n):
            W = X[:, j]
            D = X - np.tile(W, (n, 1)).T
            D = D*spread
            s = multiDiag(D.T, D)
            H[:, j] = np.exp(-s)

        # calculating weights
        # lstsq requires O(max(n, d) * min(n, d)^2) = O(nd^2)
        # ref:
https://stackoverflow.com/questions/11567710/check-how-fast-numpy-linalg-lstsq-is-find
ing-convergence
        H_tmp = np.concatenate((H, np.zeros([1, n])))
        W_tmp = np.linalg.lstsq(H_tmp.T, y)[0].T

        # extract weights and bias from W_hat
        W_hat = W_tmp[:-1]
        bias = W_tmp[-1]

        # classify with W and bias
        yt = (np.dot(H,W_hat.reshape(n, 1))).T[0] + bias
        ypred = np.ones(y.shape)
        ypred[yt < 0] = -1

        # calculate error
        predError = 1 - sum(y == ypred)/y.shape[0]

        self.W_hat = W_hat
        self.W = X
        self.bias = bias
        self.spread = spread
        self.error = predError

    def classify(self, X):
        n1, d1 = X.shape
        X = X.T
        n2, d2 = self.W.T.shape

        H = np.zeros([n1,n2])
        for j in range(n2):
            W = self.W[:, j]
            D = X - np.tile(W, (n1, 1)).T
```

```
            D = D*self.spread
            s = multiDiag(D.T, D)
            H[:, j] = np.exp(-s)

        y = (np.dot(H,self.W_hat)).T + self.bias
        ypred = np.ones(y.shape)
        ypred[y < 0] = -1

        return ypred

if __name__ == '__main__':

   df = pd.read_csv('iris.csv')
   X = df.iloc[:100,:4].to_numpy()
   y = df.iloc[:100,4].to_numpy()

   y[y=='setosa'] = -1
   y[y=='versicolor'] = 1
  # y[y=='virginica'] = 3

   y=y.astype('float')

   X_train, X_test, y_train, y_test = train_test_split(X, y)
   model = RBF(X_train, y_train, 0.21)
   model.train()
   print('Train Error:', model.error)

   y_test_pred = model.classify(X_test)

   error = 1 - sum(y_test == y_test_pred)/y_test.shape[0]
   print('Test error: ', error)
```

Output:
% python p1.py
Train Error: 0.52
Test error: 0.43999999999999995

Since the train & test error is around 50%, it seems the model is still underfitting. Possibly the size of the dataset and cardinality are the cause of the misclassifications.

Q2:
Implemented PNN for iris dataset:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split


if __name__ == '__main__':
   df = pd.read_csv('iris.csv')
   X = df.iloc[:,:4].to_numpy()
   y = df.iloc[:,4].to_numpy()

   y[y=='setosa'] = 0
   y[y=='versicolor'] = 1
   y[y=='virginica'] = 2

   y=y.astype('float')

   x = np.zeros(X.shape)

   # Normalize the data
   # O(n*d)
   for i in range(len(X)):
       x[i, :] = X[i,:]/np.sqrt(np.dot(X[i,:] ,X[i,:].T))

   w = x
   w1 = w[:50, :]
   w2 = w[51:100, :]
   w3 = w[101:, :]

   temp = np.zeros([3, 1])
   sigma = 0.5
   ypred = np.zeros([1, 150])

   m1 = len(w1)
   m2 = len(w2)
   m3 = len(w3)


   # classification with possibilities
   # Outerloop: O(n)
   # Innerloops: O(n/3 * d) *3 = O(nd)
```

```
for i in range(len(X)):

    sum1 = 0
    for j in range(m1):
        z1 = np.dot(w1[j, :],x[i,:].T)
        sum1 = sum1 + np.exp((z1-1)/sigma**2)

    temp[0] = sum1/m1

    sum2 = 0
    for j in range(m2):
        z2 = np.dot(w2[j, :], x[i, :].T)
        sum2 = sum2 + np.exp((z2-1)/(sigma**2))

    temp[1] = sum2/m2

    sum3 =0
    for j in range(m3):
        z3 = np.dot(w3[j,:] , x[i, :].T)
        sum3 = sum3 + np.exp((z3-1)/sigma**2)

    temp[2] = sum3/m3

    ypred[0,i] = np.where(temp ==np.amax(temp))[0]

accuracy = sum((ypred == y)[0])/150

print('Classification accuracy: ', accuracy)

# With the normalization and classification ,total cost will be O(n^2*d)
```

Output:
% python p2.py
Classification accuracy: 0.9733333333333334

The accuracy is around 97% as shown above.