18-10-21

CS6301 Machine Learning Week 7 Exercise

Nanda Kumar R
2019103545

Dataset - Iris dataset
150 input instances
3 target classes

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

```
iris = pd.read_csv("iris.csv")
iris= iris.sample(frac=1).reset_index(drop=True)

X=np.array(iris)[:,:-1]
X.shape

one_hot_encoder = OneHotEncoder(sparse=False)
Y = iris.species
Y = one_hot_encoder.fit_transform(np.array(Y).reshape(-1, 1))
Y.shape
```

```
(150, 3)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1)
X_train, X_valid, Y_train, Y_valid = train_test_split(X_train, Y_train, test_size=0.1)
```

K-Means algorithm

```python
def compute_euclidean_distance(point, centroid):
    return np.sqrt(np.sum((point - centroid)**2))

def assign_label_cluster(distance, data_point, centroids):
    index_of_minimum = min(distance, key=distance.get)
    return [index_of_minimum, data_point, centroids[index_of_minimum]]

def compute_new_centroids(cluster_label, centroids):
    return np.array(cluster_label + centroids)/2

def iterate_k_means(data_points, centroids, total_iteration):
    label = []
    cluster_label = []
    total_points = len(data_points)
    k = len(centroids)
    for iteration in range(0, total_iteration):
        for index_point in range(0, total_points):
            distance = {}
            for index_centroid in range(0, k):
                distance[index_centroid] = compute_euclidean_distance(data_points[index_point], centroids[index_centroid])
            label = assign_label_cluster(distance, data_points[index_point],centroids)
            centroids[label[0]] = compute_new_centroids(label[1], centroids[label[0]])
            if iteration == (total_iteration - 1):
                cluster_label.append(label)
    return [cluster_label, centroids]

def print_label_data(result):
    print("Result of k-Means Clustering: \n")
    for data in result[0]:
        print("data point: {}".format(data[1]))
        print("cluster number: {} \n".format(data[0]))
    print("Last centroids position: \n {}".format(result[1]))

def create_centroids(X,k):
    centroids = []
    indexs=np.random.choice(len(X),k)
    for i in range(len(indexs)):
        centroids.append(X[indexs[i]])
    return np.array(centroids)
```

RBF Network -

```python
class RBFNet(object):
    def __init__(self, k=2, lr=0.01, epochs=100, rbf=rbf):
        self.k = k
        self.lr = lr
        self.epochs = epochs
        self.rbf = rbf
        self.w = np.random.randn(k,3)#weights
        self.b = np.random.randn(3)#bias
        #centers
        centroids=create_centroids(X,k)
        [cluster_label, new_centroids]=iterate_k_means(X,centroids,10)
        dMax = max([compute_euclidean_distance(c1,c2) for c1 in new_centroids for c2 in new_centroids])
        #radius
        stds = np.repeat(dMax / np.sqrt(2*k), k)
        self.centers=new_centroids
        self.stds=stds
```

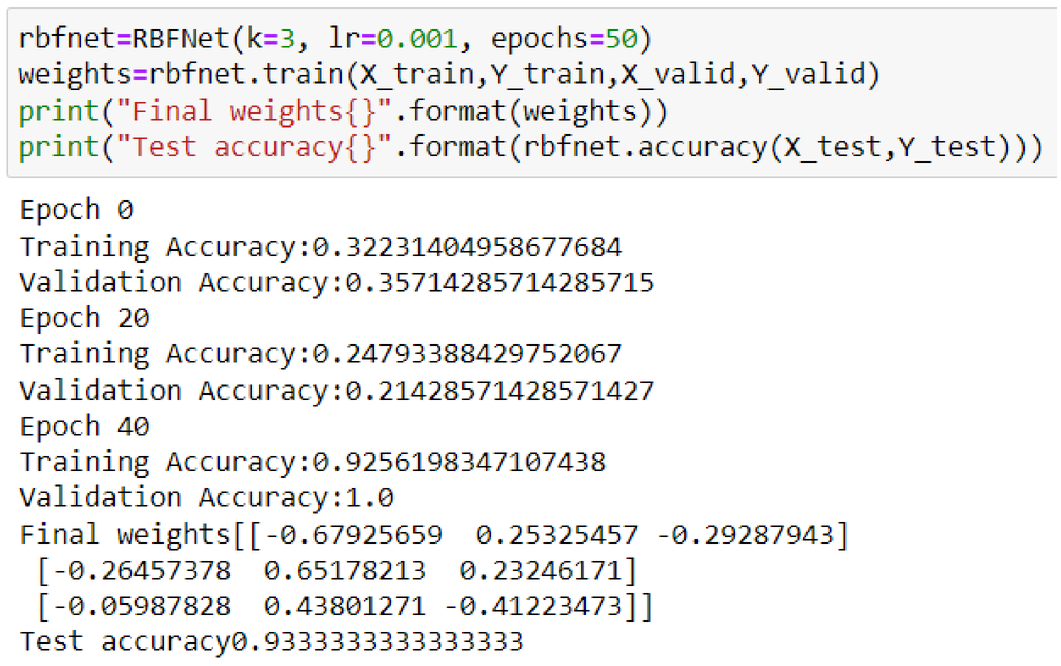
```
def train(self,X,Y,X_val,Y_val):
    for epoch in range(self.epochs):
        for i in range(X.shape[0]):
            # forward pass
            a = np.array([self.rbf(X[i], c, s) for c, s, in zip(self.centers, self.stds)])
            F = a.T.dot(self.w) + self.b
            loss = (Y[i] - F).flatten() ** 2
            # backward pass
            error = -(Y[i] - F).flatten()
            # online update
            a=np.expand_dims(a,axis=1)
            error=np.expand_dims(error,axis=0)
            self.w = self.w - self.lr*a*error
            self.b = self.b - self.lr * error
        if(epoch % 20 == 0):
            print("Epoch {}".format(epoch))
            print("Training Accuracy:{}".format(self.accuracy(X, Y)))
        if X_val.any():
            print("Validation Accuracy:{}".format(self.accuracy(X_val, Y_val)))
    return self.w
```

```
def predict(self, X):
    y_pred = []
    for i in range(X.shape[0]):
        a = np.array([self.rbf(X[i], c, s) for c, s, in zip(self.centers, self.stds)])
        F = a.T.dot(self.w) + self.b
        y_pred.append(F)
    return np.array(y_pred)

def accuracy(self,X,Y):
    y_pred=self.predict(X)
    y_pred=y_pred.reshape(len(X),3)
    y_pred_final=[]

    for i in range(len(y_pred)):
        ind=np.argmax(y_pred[i])
        y_pred2=np.zeros(3)
        y_pred2[ind]=1
        y_pred_final.append(y_pred2)

    y_pred_final=np.array(y_pred_final)
    cnt=0
    for i in range(len(Y)):
        if(Y[i]==y_pred_final[i]).all():

            cnt=cnt+1
    return cnt/len(Y)
```

Output

1.
K= 3
Learning rate = 0.001

Epochs = 50



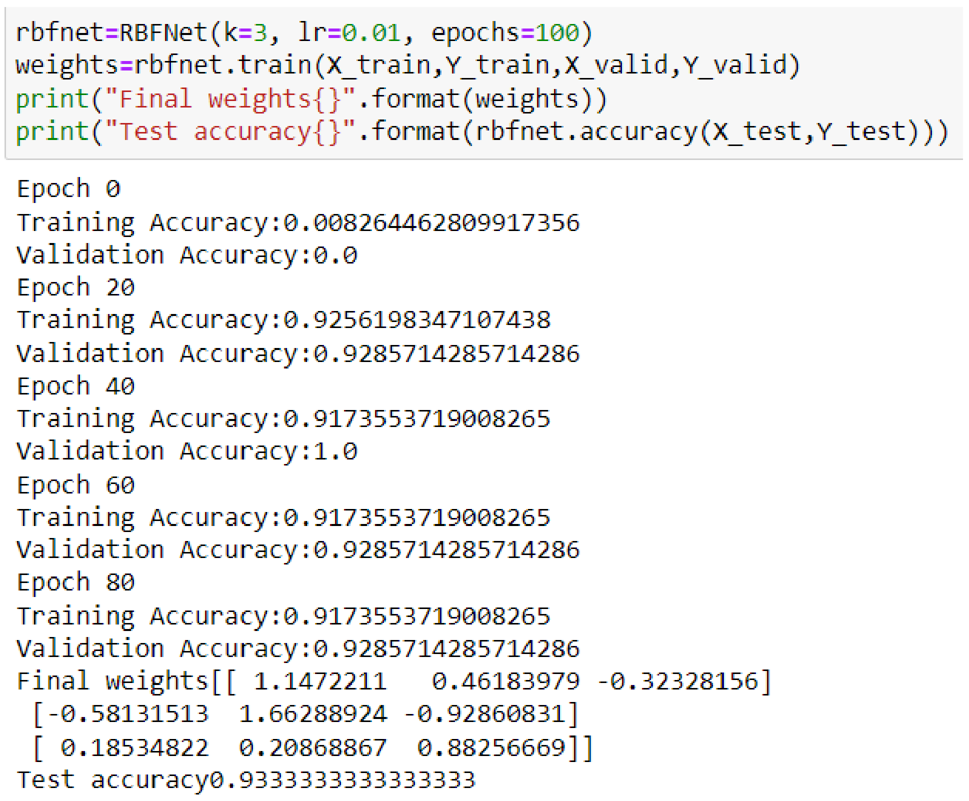
```
rbfnet=RBFNet(k=3, lr=0.001, epochs=50)
weights=rbfnet.train(X_train,Y_train,X_valid,Y_valid)
print("Final weights{}".format(weights))
print("Test accuracy{}".format(rbfnet.accuracy(X_test,Y_test)))
```

```
Epoch 0
Training Accuracy:0.32231404958677684
Validation Accuracy:0.35714285714285715
Epoch 20
Training Accuracy:0.24793388429752067
Validation Accuracy:0.21428571428571427
Epoch 40
Training Accuracy:0.9256198347107438
Validation Accuracy:1.0
Final weights[[-0.67925659  0.25325457 -0.29287943]
 [-0.26457378  0.65178213  0.23246171]
 [-0.05987828  0.43801271 -0.41223473]]
Test accuracy0.9333333333333333
```

K= 3

Learning rate = 0.01

Epochs = 100



```
rbfnet=RBFNet(k=3, lr=0.01, epochs=100)
weights=rbfnet.train(X_train,Y_train,X_valid,Y_valid)
print("Final weights{}".format(weights))
print("Test accuracy{}".format(rbfnet.accuracy(X_test,Y_test)))
```

```
Epoch 0
Training Accuracy:0.008264462809917356
Validation Accuracy:0.0
Epoch 20
Training Accuracy:0.9256198347107438
Validation Accuracy:0.9285714285714286
Epoch 40
Training Accuracy:0.9173553719008265
Validation Accuracy:1.0
Epoch 60
Training Accuracy:0.9173553719008265
Validation Accuracy:0.9285714285714286
Epoch 80
Training Accuracy:0.9173553719008265
Validation Accuracy:0.9285714285714286
Final weights[[ 1.1472211   0.46183979 -0.32328156]
 [-0.58131513  1.66288924 -0.92860831]
 [ 0.18534822  0.20868867  0.88256669]]
Test accuracy0.9333333333333333
```

K= 5

Learning rate = 0.001

Epochs = 150

```
rbfnet=RBFNet(k=5, lr=0.001, epochs=150)
weights=rbfnet.train(X_train,Y_train,X_valid,Y_valid)
print("Final weights{}".format(weights))
print("Test accuracy{}".format(rbfnet.accuracy(X_test,Y_test)))
```

```
Epoch 0
Training Accuracy:0.024793388429752067
Validation Accuracy:0.0
Epoch 20
Training Accuracy:0.4049586776859504
Validation Accuracy:0.35714285714285715
Epoch 40
Training Accuracy:0.5454545454545454
Validation Accuracy:0.35714285714285715
Epoch 60
Training Accuracy:0.7520661157024794
Validation Accuracy:0.7142857142857143
Epoch 80
Training Accuracy:0.8264462809917356
Validation Accuracy:0.7857142857142857
Epoch 100
Training Accuracy:0.9008264462809917
Validation Accuracy:0.8571428571428571
Epoch 120
Training Accuracy:0.9173553719008265
Validation Accuracy:0.8571428571428571
Epoch 140
Training Accuracy:0.9173553719008265
Validation Accuracy:0.8571428571428571
Final weights[[ 1.93804724 -2.50165667  0.88719522]
 [-1.2513661   0.97894542 -0.15460445]
 [ 0.05199205  0.65176996 -0.77326996]
 [-0.43691645  0.66666809 -0.04349527]
 [-0.05636537 -1.02992781  0.84639128]]
Test accuracy0.8666666666666667
```