تمرین سوم شبکه عصبی

سوال سوم:

برای ساخت یک شبکه عصبی RBF برای طبقه بندی دیتاست Iris ابتدا کتابخانه های مربوطه را ایمپورت می کنیم.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans

iris_dataset = load_iris()
X = iris_dataset.data
y = iris_dataset.target
```

سپس داده های X و y را جدا می کنیم.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

داده ها را به دو بخش آموزش و تست تقسیم میکنیم. Random_state را 42 قرار می دهیم تا هر سری با یک توزیع رندوم ثابت کار کنیم.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

با از استفاده از کتابخانه StandardScaler داده ها رو استانداردسازی میکنیم.

```
def gaussian_rbf(x, center, gamma):
    return np.exp(-gamma * np.linalg.norm(x - center) ** 2)
```

یک تابع برای محاسبه تابع شعاعی تعریف کردیم که فاصله اقلیدسی بین متغیر ورودی و مرکز را محاسبه می کند و مقدار تابع گوسی را بر می گرداند.

```
class RBFNetwork:
    def __init__(self, centers_count, gamma=1.0):
        self.centers_count = centers_count
        self.gamma = gamma
```

یک آبجکت برای تعریف شبکه عصبی rbf ایجاد می کنیم و تعداد مراکز و گاما را برای ساخت این شی دریافت می کنیم. توابع مورد نیاز این آبجکت را باهم بررسی می کنیم.

```
def select_random_centers(self, X):
    indexes = np.random.choice(X.shape[0], self.centers_count, replace=False)
    self.centers = X[indexes]
```

این تابع برای پیدا کردن مراکز رندوم استفاده می شود. به این صورت کار می کند که به تعداد مراکز شعاع ها عدد رندوم از تعداد نمونه های دیتاست تولید میکند. تا با استفاده از اون اعداد نمونه هایی که مرکز شعاع هستند را پیدا کنیم و درون متغیر self.centers بریزیم.

```
def select_centers_kmeans(self, X):
    kmeans = KMeans(n_clusters=self.centers_count, random_state=42)
    kmeans.fit(X)
    self.centers = kmeans.cluster_centers_
```

این تابع برای پیدا کردن مراکز با استفاده از الگوریتم KMeans است که به تعداد مراکزی که ست کردیم خوشه ایجاد می کند و پس از اجرای الگوریتم مراکز خوشه ها در متغیر self.centers قرار داده میشود تا در ادامه بتوانیم از این مراکز برای طبقه بندی استفاده کنیم.

```
def compute_interpolation_matrix(self, X):
    G = np.zeros((X.shape[0], self.centers_count))
    for i, x in enumerate(X):
        for j, c in enumerate(self.centers):
            G[i, j] = gaussian_rbf(x, c, self.gamma)
    return G
```

در این تابع کلاس ما ماتریس با ابعاد (تعداد نمونه ها در تعداد مراکز) ایجاد می کنیم تا با استفاده از تابع شعاعی که بالا ایجاد کردیم تا ارتباط همه داده ها با همه مرکز ها را محاسبه کنیم. پس از آن وقتی ماتریس کاملا پر شد میتوانیم برای پیدا کردن وزن شبکه از این ماتریس استفاده کنیم.

```
def fit(self, X, y):
    G = self.compute_interpolation_matrix(X)
    self.weights = np.linalg.pinv(G).dot(y)
```

نوبت به تابع فیت میرسد. اول ماتریس میانی رو پر می کنیم تا ارتباط هر داده با مراکز مشخص شود. سپس برای محاسبه وزن ابتدا ماتریس G را معکوس می کنیم و در y ضرب داخلی می کنیم تا به بهترین خروجی برای y برسیم.

```
def predict(self, X):
    G = self.compute_interpolation_matrix(X)
    predictions = G.dot(self.weights)
    return np.round(predictions).astype(int)
```

در نهایت در تابع پیش بینی دوباره ماتریس محاسبه میشود. سپس برای محاسبه پیش بینی از ضرب داخلی ماتریس با مقادیر وزن استفاده میکنیم و در نهایت آن را گرد می کنیم و به عدد صحیح تبدیل می کنیم تا به مقادیر گسسته برسیم و بفهمیم نمونه مورد نظر از کدام کلاس است.

```
centers_count = 15
gamma = 1.2

rbf_net_random = RBFNetwork(centers_count = centers_count, gamma = gamma)
rbf_net_random.select_random_centers(X_train)
rbf_net_random.fit(X_train, y_train)
```

ما برای آموزش مدل از 15 تا مرکز شعاعی با گاما 1.2 استفاده کردیم. حال یک مدل rbf با مراکز رندوم ایجاد می کنیم و داده های آموزشی را روی مدل فیت می کنیم.

```
y_pred_train_random = rbf_net_random.predict(X_train)
y_pred_test_random = rbf_net_random.predict(X_test)

train_accuracy_random = accuracy_score(y_train, y_pred_train_random)
test_accuracy_random = accuracy_score(y_test, y_pred_test_random)

print('RBF with random centers:')
print('Train accuracy:',train_accuracy_random)
print('Test accuracy:',test_accuracy_random)
```

پس از آن برای داده های تست پیش بینی میکنیم تا بفهمیم مدل ما چقد دقت دارد. در ادامه میزان دقت برای داده های آموزشی و تست را چاپ می کنیم که به صورت زیر است.

```
RBF with random centers:
Train accuracy: 0.8083333333333333
Test accuracy: 0.8333333333333334
```

```
rbf_net_kmeans = RBFNetwork(centers_count=centers_count, gamma=gamma)
rbf_net_kmeans.select_centers_kmeans(X_train)
rbf_net_kmeans.fit(X_train, y_train)
```

پس از این مدل یک مدل دیگر با استفاده از مراکزی که توسط Kmeans بدست آمد ایجاد می کنیم و داده ها را فیت می کنیم تا بتوانیم برای داده های تست پیش بینی کنیم.

```
y_pred_train_kmeans = rbf_net_kmeans.predict(X_train)
y_pred_test_kmeans = rbf_net_kmeans.predict(X_test)

train_accuracy_kmeans = accuracy_score(y_train, y_pred_train_kmeans)
test_accuracy_kmeans = accuracy_score(y_test, y_pred_test_kmeans)

print()
print('RBF with KMean centers:')
print('Train accuracy:',train_accuracy_kmeans)
print('Test accuracy:',test_accuracy_kmeans)
```

میزان دقت این مدل نیز به صورت زیر است.

```
RBF with KMean centers:
Train accuracy: 0.9166666666666666
Test accuracy: 0.8666666666666667
```

مقایسه این دو روش:

به صورت کلی میزان دقت مدل KMeans از مدل random در هر دو داده های آموزشی و تست بیشتر است.

در روش random مراکز به صورت تصادفی انتخاب می شوند و این امر باعث میشود به ساختار داده ها توجه نکند و مراکز در مکان های نامناسب انتخاب شود. مزایای این روش سرعت بالا است.

اما در روش KMeans مراکز به صورت معنادار انتخاب میشوند و هر مرکز نماینده یک توزیع خاص میتواند باشد. اما این روش محاسبات بیشتری دارد.

در مدل rbf انتخاب مراکز از اهمیت بالایی برخودار است و تاثیر مستقیمی بر روی نتیجه نهایی دارد.

در کل انتخاب مراکز با استفاده از الگوریتم KMeans برای مدل rbf بهتر است.

محمد حقیقت - 403722042