assignment2

October 13, 2024

[15]:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

n_centers: Number of RBF neurons in the hidden layer (i.e., the number of centers).
sigma: The spread or width of the Gaussian function, which can be set manually or calculated automatically.
centers: The RBF centers, which will be selected from the input data.
weights: The weights connecting the RBF layer to the output.

[16]:

```
class RBFNetwork:
    def __init__(self, n_centers, sigma=None):
        self.n_centers = n_centers
        self.centers = None
        self.sigma = sigma
        self.weights = None
        self.bias = None

    def _gaussian(self, x, c, s):
        return np.exp(-1 / (2 * s**2) * np.linalg.norm(x - c)**2)

    def _select_centers(self, X):
        kmeans = KMeans(n_clusters=self.n_centers, random_state=42)
        kmeans.fit(X)
        self.centers = kmeans.cluster_centers_

    def _calculate_sigma(self):
        if self.sigma is None:
            distances = np.linalg.norm(self.centers[:, np.newaxis] - self.
↪centers, axis=2)
            self.sigma = np.mean(distances)

    def fit(self, X, y):
        self._select_centers(X)
        self._calculate_sigma()
        G = np.zeros((X.shape[0], self.n_centers))
        for i in range(X.shape[0]):
            for j in range(self.n_centers):
                G[i, j] = self._gaussian(X[i], self.centers[j], self.sigma)

        G = np.hstack([G, np.ones((G.shape[0], 1))])
        self.weights = np.linalg.pinv(G).dot(y)

    def predict(self, X):
        G = np.zeros((X.shape[0], self.n_centers))
        for i in range(X.shape[0]):
            for j in range(self.n_centers):
                G[i, j] = self._gaussian(X[i], self.centers[j], self.sigma)

        G = np.hstack([G, np.ones((G.shape[0], 1))])

        return np.dot(G, self.weights)
```

Generate synthetic data

[17]:
```
X, y = make_classification(n_samples=1000, n_features=3, n_informative=2,␣
 ↪n_redundant=0, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[18]:
```
X.shape, y.shape
```

[18]:
```
((1000, 3), (1000,))
```

Sample X and y

[19]:
```
print(X[:5])
```

```
[[ 0.96422927  1.99566749  0.24414337]
 [-1.35806186 -0.25495579  0.50289028]
 [ 1.73205679  0.26125053 -2.21417748]
 [-1.51987766  1.02370955 -0.26269143]
 [ 4.02026158  1.38145408 -1.58214341]]
```

[25]:
```
print(y[:5])
```

```
[1 0 1 0 1]
```

[20]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[21]:
```
n_centers = 10
rbf = RBFNetwork(n_centers=n_centers)
```

```
rbf.fit(X_train, y_train)
```

Test Accuracy

[22]:

```
y_pred = rbf.predict(X_test)

from sklearn.metrics import accuracy_score
y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred_binary)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.85
```

Train Accuracy

[24]:

```
y_pred = rbf.predict(X_train)

y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_train, y_pred_binary)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.87
```