Lab7

November 20, 2024

0.0.1 1. Nearest Neighbours with user-defined distances

[1]:

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

X = np.genfromtxt("ionosphere.txt", delimiter=",",
                  usecols=np.arange(34))
y = np.genfromtxt("ionosphere.txt", delimiter=",",
                  usecols=34, dtype='int')
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

[2]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',␣
 ↪metric_params=None, n_jobs=1, n_neighbors=1, p=2, weights='uniform')
```

[2]:

```
KNeighborsClassifier(n_jobs=1, n_neighbors=1)
```

[3]:

```
def my_dist(x, y):
    return np.sum((x-y)**2)

knn = KNeighborsClassifier(n_neighbors=1, metric=my_dist)
knn.fit(X_train, y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric=my_dist,␣
 ↪metric_params=None, n_jobs = 1, n_neighbors=1, p=2, weights='uniform')
```

[3]:

```
KNeighborsClassifier(metric=<function my_dist at 0x7f8caa0eeac0>, n_jobs=1,
                     n_neighbors=1)
```

[4]:

```
np.mean(knn.predict(X_test) == y_test)
```

[4]:

```
np.float64(0.8522727272727273)
```

[5]:

```
knn = KNeighborsClassifier(n_neighbors=1, p=1)
knn.fit(X_train, y_train)
```

```
np.mean(knn.predict(X_test)==y_test)
```

[5]:
```
np.float64(0.9090909090909091)
```

Exercise 1

[6]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(knn, X_train, y_train)

print(np.mean(scores))
```

```
0.897822931785196
```

the best value for P is 1 where the accuracy of Cross_val score is 0.897822...

0.0.2 2. Kernel Methods

[7]:
```
def poly_kernel(x, y, d):
    return (1+np.dot(x, y)) ** d
d = 2
def poly_dist(x, y):
    return poly_kernel(x, x, d) + poly_kernel(y, y, d) - 2*poly_kernel(x, y, d)

knn = KNeighborsClassifier(n_neighbors=1, metric=poly_dist)
knn.fit(X_train, y_train)
np.mean(knn.predict(X_test) == y_test)
```

[7]:
```
np.float64(0.8522727272727273)
```

[8]:
```
def rbf_kernel(x, y, gamma):
    return np.exp(-gamma*np.sum((x-y)**2))
gamma = 1
def rbf_dist(x, y):
    return rbf_kernel(x, x, gamma) + rbf_kernel(y, y, gamma) - 2*rbf_kernel(x,␣
 ↪y, gamma)

knn = KNeighborsClassifier(n_neighbors=1, metric=rbf_dist)
knn.fit(X_train, y_train)
np.mean(knn.predict(X_test) == y_test)
```

[8]:
```
np.float64(0.8522727272727273)
```

[9]:
```
best_score = 0
for gamma in [0.01, 0.1, 1, 10, 100]:
    def rbf_dist(x, y):
        return rbf_kernel(x, x, gamma) + rbf_kernel(y, y, gamma) -␣
 ↪2*rbf_kernel(x, y, gamma)
```

```
    knn = KNeighborsClassifier(n_neighbors=1, metric=rbf_dist)
    scores = cross_val_score(knn, X_train, y_train, cv=5)
    score = np.mean(scores)
    if score > best_score:
        best_score = score
        best_gamma = gamma


def rbf_dist(x, y):
    return rbf_kernel(x, x, best_gamma) + rbf_kernel(y, y, best_gamma) -␣
 ↪2*rbf_kernel(x, y, best_gamma)

knn = KNeighborsClassifier(n_neighbors=1, metric=rbf_dist)
knn.fit(X_train, y_train)
test_score = knn.score(X_test, y_test)
print("Best CV score: ", best_score)
print("Best parameter gamma: ", best_gamma)
print("Test set score with best parameters: ", test_score)
```

```
Best CV score:  0.9317851959361393
Best parameter gamma:  10
Test set score with best parameters:  0.9431818181818182
```

0.0.3 3. Creating your own estimator

[10]:

```
class My_Classifier(KNeighborsClassifier):
    def __init__(self, n_neighbors=1):
        KNeighborsClassifier.__init__(self, n_neighbors=1)

    def fit(self, X, y):
        KNeighborsClassifier.fit(self, X, y)
        return self

    def predict(self, X, y=None):
        return KNeighborsClassifier.predict(self, X)

    def score(self, X, y):
        return KNeighborsClassifier.score(self, X, y)
```

[11]:

```
knn = My_Classifier()
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

[11]:

```
0.8522727272727273
```

[12]:

```
class rbfClassifier(KNeighborsClassifier):
    def __init__(self, n_neighbors=1, gamma=1):
        def rbf_dist(x, y): # squared distance
            return rbf_kernel(x,x,gamma) + rbf_kernel(y,y,gamma) -␣
 ↪2*rbf_kernel(x,y,gamma)
        KNeighborsClassifier.__init__(self, n_neighbors=n_neighbors,␣
 ↪metric=rbf_dist)
        self.gamma = gamma
        self.n_neighbors=n_neighbors
    def fit(self, X, y):
        KNeighborsClassifier.fit(self, X, y)
        return self
    def predict(self, X, y=None):
        return KNeighborsClassifier.predict(self, X)
    def score(self, X, y):
        return KNeighborsClassifier.score(self, X, y)
```

```
[13]: knn = rbfClassifier()
      knn.fit(X_train, y_train)
      knn.score(X_test, y_test)
```

```
[13]: 0.8522727272727273
```

0.0.4 4. Uncertainty estimates for the nearest neighbours algorithm

```
[14]: from sklearn.datasets import load_iris
      iris = load_iris()
      X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,␣
       ↪random_state=42)
      knn = KNeighborsClassifier()
      knn.fit(X_train, y_train)
      knn.predict(X_test)
```

```
[14]: array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
             0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0])
```

```
[15]: knn.predict_proba(X_test)
```

```
[15]: array([[0. , 1. , 0. ],
             [1. , 0. , 0. ],
             [0. , 0. , 1. ],
             [0. , 1. , 0. ],
             [0. , 1. , 0. ],
             [1. , 0. , 0. ],
             [0. , 1. , 0. ],
             [0. , 0. , 1. ],
             [0. , 0.6, 0.4],
             [0. , 1. , 0. ],
             [0. , 0.2, 0.8],
             [1. , 0. , 0. ],
             [1. , 0. , 0. ],
             [1. , 0. , 0. ],
             [1. , 0. , 0. ],
             [0. , 0.8, 0.2],
             [0. , 0. , 1. ],
             [0. , 1. , 0. ],
             [0. , 1. , 0. ],
             [0. , 0. , 1. ],
             [1. , 0. , 0. ],
             [0. , 0.2, 0.8],
             [1. , 0. , 0. ],
             [0. , 0. , 1. ],
             [0. , 0. , 1. ],
             [0. , 0. , 1. ],
             [0. , 0. , 1. ],
             [0. , 0. , 1. ],
             [1. , 0. , 0. ],
             [1. , 0. , 0. ],
             [1. , 0. , 0. ],
             [1. , 0. , 0. ],
             [0. , 1. , 0. ],
             [1. , 0. , 0. ],
             [1. , 0. , 0. ],
             [0. , 0.4, 0.6],
             [0. , 1. , 0. ],
             [1. , 0. , 0. ]])
```

0.0.5 5. More Exercises

2. From the array in [14], if you iterate through each list and return the index with the highest float value. You will end up with the same list as in [13].
3.

[16]:

```
class rbfClassifier(KNeighborsClassifier):
    def __init__(self, n_neighbors=1, gamma=1):
        def rbf_dist(x, y): # squared distance
            return rbf_kernel(x,x,gamma) + rbf_kernel(y,y,gamma) -␣
 ↪2*rbf_kernel(x,y,gamma)
        KNeighborsClassifier.__init__(self, n_neighbors=n_neighbors,␣
 ↪metric=rbf_dist)
        self.gamma = gamma
        self.n_neighbors=n_neighbors
    def fit(self, X, y):
        KNeighborsClassifier.fit(self, X, y)
        return self
    def predict(self, X, y=None):
        return KNeighborsClassifier.predict(self, X)
    def score(self, X, y):
        return KNeighborsClassifier.score(self, X, y)
    def predict_proba(self, X, y=None):
        return KNeighborsClassifier.predict_proba(self, X)
```

4.

[17]:
```
knn = rbfClassifier()
knn.fit(X_train, y_train)
knn.predict_proba(X_train)
```

[17]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
[0., 0., 1.],
[0., 0., 1.],
[0., 1., 0.],
[0., 1., 0.],
[0., 0., 1.],
[0., 1., 0.],
[1., 0., 0.],
[0., 1., 0.],
[0., 0., 1.],
[1., 0., 0.],
[1., 0., 0.],
[0., 1., 0.],
[0., 1., 0.],
[1., 0., 0.],
[0., 0., 1.],
[1., 0., 0.],
[1., 0., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 0., 1.],
[0., 1., 0.],
[0., 0., 1.],
[0., 0., 1.],
[0., 1., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 0., 1.],
[0., 0., 1.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 1., 0.],
[0., 0., 1.],
[1., 0., 0.],
[0., 0., 1.],
[0., 0., 1.],
[1., 0., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 0., 1.],
[0., 1., 0.],
[0., 0., 1.],
[1., 0., 0.],
[0., 0., 1.],
[0., 1., 0.],
[0., 0., 1.],
[0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```