KNN - Lab

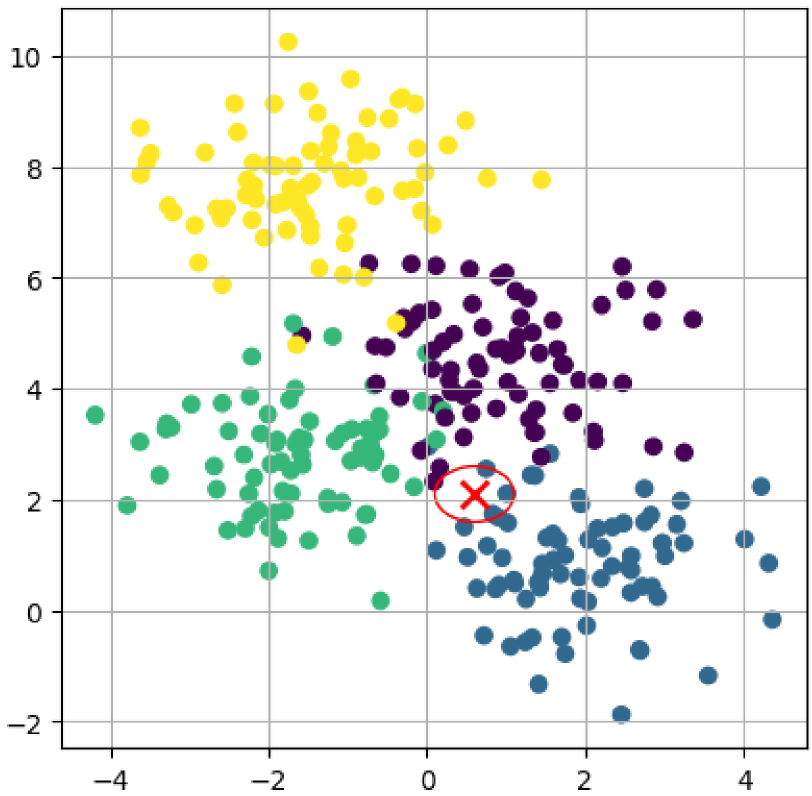
September 9, 2021

Modify the KNN scratch code in our lecture such that: - If the majority class of the first place is equal to the second place, then ask the algorithm to pick the next nearest neighbors as the decider - Modify the code so it outputs the probability of the decision, where the probability is simply the class probability based on all the nearest neighbors - Write a function which allows the program to receive a range of k, and output the cross validation score. Last, it shall inform us which k is the best to use from a predefined range - Put everything into a class KNN(k=3). It should have at least one method, predict(X_train, X_test, y_train)

1 Prepare the data

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
#let's consider the following 2D data with 4 classes
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X, y = make_blobs(n_samples=300, centers=4,
                  random_state=0, cluster_std=1.0)

xfit = np.linspace(-1, 3.5)

figure = plt.figure(figsize=(5, 5))
ax = plt.axes()  #get the instance of axes from plt

ax.grid()
ax.scatter(X[:, 0], X[:, 1], c=y)

#where should this value be classified as?
ax.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

#let's say roughly 5 neighbors
circle = plt.Circle((0.6, 2.1), 0.5, color='red', fill=False)
ax.add_artist(circle)
plt.show()
```




```
[63]: #standardize
scaler = StandardScaler()
X = scaler.fit_transform(X)

#do train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

2 Define functions

```
[64]: def find_distance(X_train, X_test):
    #create newaxis simply so that broadcast to all values
    dist = X_test[:, np.newaxis, :] - X_train[np.newaxis, :, :]
    sq_dist = dist ** 2

    #sum across feature dimension, thus axis = 2
    summed_dist = sq_dist.sum(axis=2)
    sq_dist = np.sqrt(summed_dist)
    return sq_dist
```

[65]:
```python
def find_neighbors(X_train, X_test, k=3):
    dist = find_distance(X_train, X_test)
    #return the first k neighbors
    neighbors_ix = np.argsort(dist)[:, 0:k]
    return neighbors_ix
```

[66]:
```python
def get_most_common(y, y_train, X_train, X_test, n_classes, ix, k):
    bin_count = np.bincount(y, minlength=n_classes)
    most_common_idx = bin_count.argsort()[-1]
    second_most_common_idx = bin_count.argsort()[-2]

    if bin_count[most_common_idx] == bin_count[second_most_common_idx]:
        y = y_train[find_neighbors(X_train, X_test, k=k+1)[ix]]
        bin_count = np.bincount(y, minlength=n_classes)
        most_common_idx = bin_count.argsort()[-1]

    prediction = np.bincount(y).argmax()
    probabiliy = bin_count[most_common_idx] / sum(np.bincount(y))
    return prediction, probabiliy
```

3 Define the class

[124]:
```python
class KNN:
    def __init__(self):
        pass

    def predict(self, X_train, X_test, y_train, k=3):
        n_classes = len(np.unique(y_train))
        neighbors_ix = find_neighbors(X_train, X_test, k)
        pred = np.zeros(X_test.shape[0])
        prob = np.zeros(X_test.shape[0])
        for ix, y in enumerate(y_train[neighbors_ix]):
            pred[ix], prob[ix] = get_most_common(y, y_train, X_train, X_test,␣
↪n_classes, ix, k=k)
        return pred, prob

    def grid_search_cv(self,X, y, num_fold, k_candidates):

        indices = list(range(len(X)))
        np.random.shuffle(indices)

        sample_per_fold = math.floor(len(indices) / num_fold)

        best_acc = 0
        best_k = None
```

```
        mean_acc_in_each_k = []
        mean_prob_in_each_k = []

        for k in k_candidates:
            current_acc = []
            current_prob = []
            for i in range(num_fold):
                test_indices = indices[i * sample_per_fold: i * sample_per_fold␣
 ↪+ sample_per_fold]
                train_indices = [index for index in indices if index not in␣
 ↪test_indices]

                X_train = X[train_indices]
                X_test = X[test_indices]

                y_train = y[train_indices]
                y_test = y[test_indices]


                yhat, yprob = cls.predict(X_train, X_test, y_train, k=k)

                acc =  np.sum(yhat == y_test)/len(y_test)
                current_acc.append(acc)
                current_prob.append(yprob.mean())

            current_k_acc = np.mean(current_acc)
            current_k_prob = np.mean(current_prob)

            mean_acc_in_each_k.append(current_k_acc)
            mean_prob_in_each_k.append(current_k_prob)

            if current_k_acc > best_acc:
                best_acc = current_k_acc
                best_k = k

        return {'k_candidates':k_candidates,
                'mean_acc_in_each_k':mean_acc_in_each_k,
                'mean_prob_in_each_k':mean_prob_in_each_k,
                'best_k':best_k}
```

[125]:
```
cls = KNN()
```

[126]:
```
yhat, yprob = cls.predict(X_train, X_test, y_train, k=3)

n_classes = len(np.unique(y_test))

print("Accuracy:", np.sum(yhat == y_test)/len(y_test))
```

```
print(f"Probably: {yprob.mean()}")
```

```
Accuracy: 0.9222222222222223
Probably: 0.9611111111111111
```

[127]:
```
cls = KNN()

k_candidates = [2,3,4,5,6,7,8,9,10,50,100,300,500]

num_fold = 5

result = cls.grid_search_cv(X, y, num_fold, k_candidates)
```

[128]:
```
for k, mean_acc_in_each_k, mean_prob_in_each_k in zip(result['k_candidates'],␣
↪result['mean_acc_in_each_k'], result['mean_prob_in_each_k']):
    print(f'k = {k:03}, Average Accuracy: {mean_acc_in_each_k:.2f}, Average␣
↪Probability: {mean_prob_in_each_k:.2f}')

print(f"\nBest k is {result['best_k']}")
```

```
k = 002, Average Accuracy: 0.93, Average Probability: 0.96
k = 003, Average Accuracy: 0.93, Average Probability: 0.95
k = 004, Average Accuracy: 0.93, Average Probability: 0.94
k = 005, Average Accuracy: 0.93, Average Probability: 0.94
k = 006, Average Accuracy: 0.93, Average Probability: 0.94
k = 007, Average Accuracy: 0.93, Average Probability: 0.93
k = 008, Average Accuracy: 0.93, Average Probability: 0.93
k = 009, Average Accuracy: 0.93, Average Probability: 0.93
k = 010, Average Accuracy: 0.93, Average Probability: 0.93
k = 050, Average Accuracy: 0.93, Average Probability: 0.83
k = 100, Average Accuracy: 0.93, Average Probability: 0.56
k = 300, Average Accuracy: 0.20, Average Probability: 0.26
k = 500, Average Accuracy: 0.20, Average Probability: 0.26

Best k is 2
```