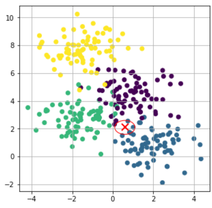
Modify the KNN scratch code in our lecture such that:

- 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)`

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [7]:
```
#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?T
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)
```

Out[7]:
```
<matplotlib.patches.Circle at 0x20a19b96fa0>
```



Prepare data

In [53]:
```
#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)
```

Pairwise Distance

In [15]:
```
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
```

Argsort pairwise distance matrix

In [16]:
```
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
```

Get majority class

In [17]:
```
def get_most_common(y):
    return np.bincount(y).argmax()
#if you don't understand what is this function, see below
```

Define class

In [182]:
```
class KNN():

    def __init__(self, k = 3):
        self.k = k

    def find_best_k(self, X_train, y_train, div, k_range):
        batch_size = int(X_train.shape[0] / div)
        y_hat_arr = np.zeros((len(k_range), div))
        y_hat_probs = np.zeros((len(k_range), div))

        for k_idx, kneighbor in enumerate(k_range):
            self.k = kneighbor
            for idx, i in enumerate(range(0, X_train.shape[0], batch_size)):
                X_test2 = X_train[i:i+batch_size]
                y_test2 = y_train[i:i+batch_size]
                X_train2 = np.concatenate((X_train[:i], X_train[i+batch_size:]))
                y_train2= np.concatenate((y_train[:i], y_train[i+batch_size:]))
                yhat, yhat_prob = self.predict(X_train2, X_test2, y_train2)
#                 print(np.sum(yhat == y_test2) / len(y_test2))
                acc_correct = yhat == y_test2
                accuracy = np.sum(acc_correct) / len(y_test2)
                y_hat_arr[k_idx, idx] = accuracy
                y_hat_probs[k_idx, idx] = yhat_prob.mean()
        return y_hat_arr, y_hat_probs

    def find_distance(self, 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


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


    def get_most_common(self, y, k):
        y = y[0:k]
        count = np.bincount(y)
        largest_first = count.argmax()
        largest_second = count.argsort()[-2:][0]
        if count[largest_first] == count[largest_second]:
            y = y[0: k + 1]
            return np.bincount(y).argmax(), count[largest_first] /count.sum()

        return np.bincount(y).argmax(), count[largest_first] / count.sum()


    def predict(self, X_train, X_test, y_train):
        neighbors_ix = self.find_neighbors(X_train, X_test, self.k)
        self.pred = np.zeros(X_test.shape[0])
        self.probs = np.zeros(X_test.shape[0])
        for ix, y in enumerate(y_train[neighbors_ix]):
            self.pred[ix], self.probs[ix] = self.get_most_common(y, self.k)
        return self.pred, self.probs
```

In [183]:
```
knn = KNN()
k_range = np.arange(2, 8)
accuracies, prob_scores = knn.find_best_k(X_train, y_train, 10, k_range)

acc_mean = accuracies.mean(axis=1)
prob_scores_mean = prob_scores.mean(axis=1)

n_classes = len(np.unique(y_test))
```

In [184]:
```
# Display accuracy for each k
for k_idx, k in enumerate(k_range):
    print(f"Score with k={k}: {acc_mean[k_idx]} and probability: {prob_scores_mean[k_idx]}")
```

```
Score with k=2: 0.9142857142857144 and probability: 0.9452380952380952
Score with k=3: 0.9095238095238095 and probability: 0.9460317460317746
Score with k=4: 0.9190476190476619 and probability: 0.9345238095238095
Score with k=5: 0.9190476190476619 and probability: 0.9323809523809524
Score with k=6: 0.9285714285714286 and probability: 0.9261904761904762
Score with k=7: 0.9333333333333333 and probability: 0.9210884353741499
```

Highest score is k = 7 but higher prob is k = 2

In [188]:
```
knn = KNN(k = 2)
yhat, yhat_probs = knn.predict(X_train, X_test, y_train)

print("Accuracy: ", np.sum(yhat == y_test)/len(y_test))
print("================================ Classification report ================================")
print("Report: ", classification_report(y_test, yhat))
print("=======================================  Probability score  ========================================")
print("Probability: ", yhat_probs.mean())
```

```
Accuracy:  0.9333333333333333
================================ Classification report ================================
Report:                precision    recall  f1-score   support

           0       0.87      0.87      0.87        23
           1       0.95      0.88      0.91        24
           2       0.95      1.00      0.97        18
           3       0.96      1.00      0.98        25

    accuracy                           0.93        90
   macro avg       0.93      0.94      0.93        90
weighted avg       0.93      0.93      0.93        90

=======================================  Probability score  ========================================
Probability:  0.9722222222222222
```