In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.style as style
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

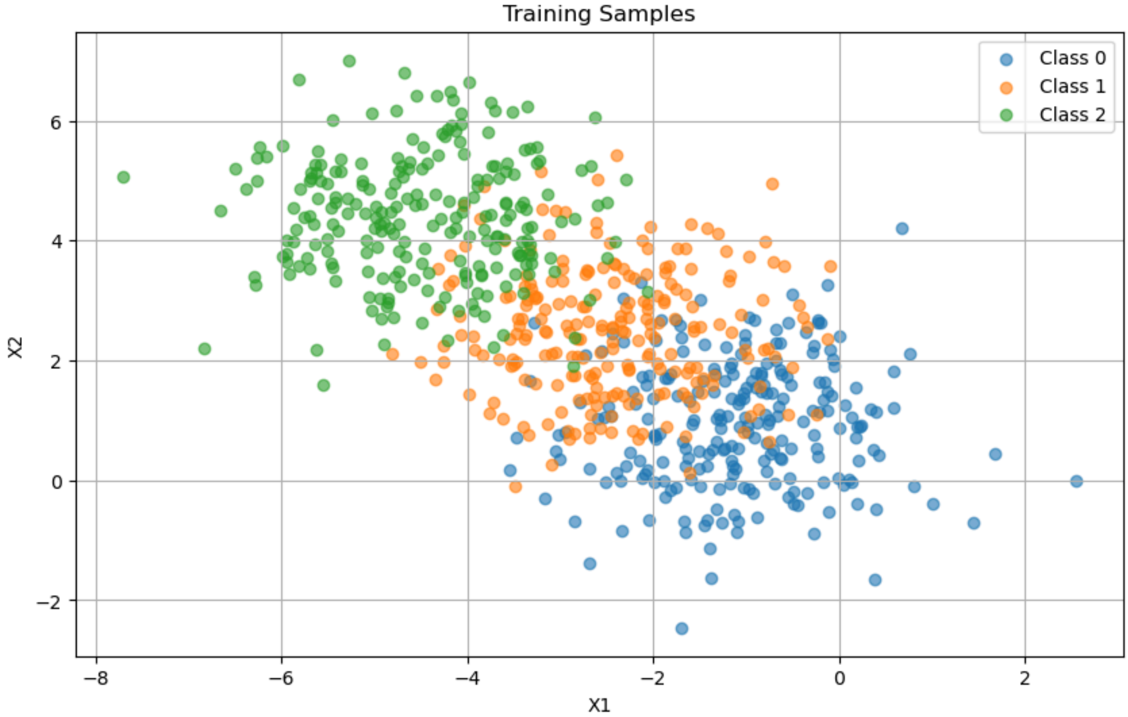
In [2]:
```
# Define the means and covariance matrices
mu1 = np.array([-1, 1])
mu2 = np.array([-2.5, 2.5])
mu3 = np.array([-4.5, 4.5])
sigma = np.eye(2)
```

Question 1.1

In [3]:
```
# Number of samples
n_samples = 700
n_classes = 3
samples_per_class = n_samples // n_classes

# Generate samples
np.random.seed(859)
samples1 = np.random.multivariate_normal(mu1, sigma,
                                         samples_per_class)
samples2 = np.random.multivariate_normal(mu2, sigma, samples_per_class)
samples3 = np.random.multivariate_normal(mu3, sigma, samples_per_class)

# Combine the samples
X_train = np.vstack([samples1, samples2, samples3])
y_train = np.hstack([[0]*samples_per_class,
                     [1]*samples_per_class, [2]*samples_per_class])
```

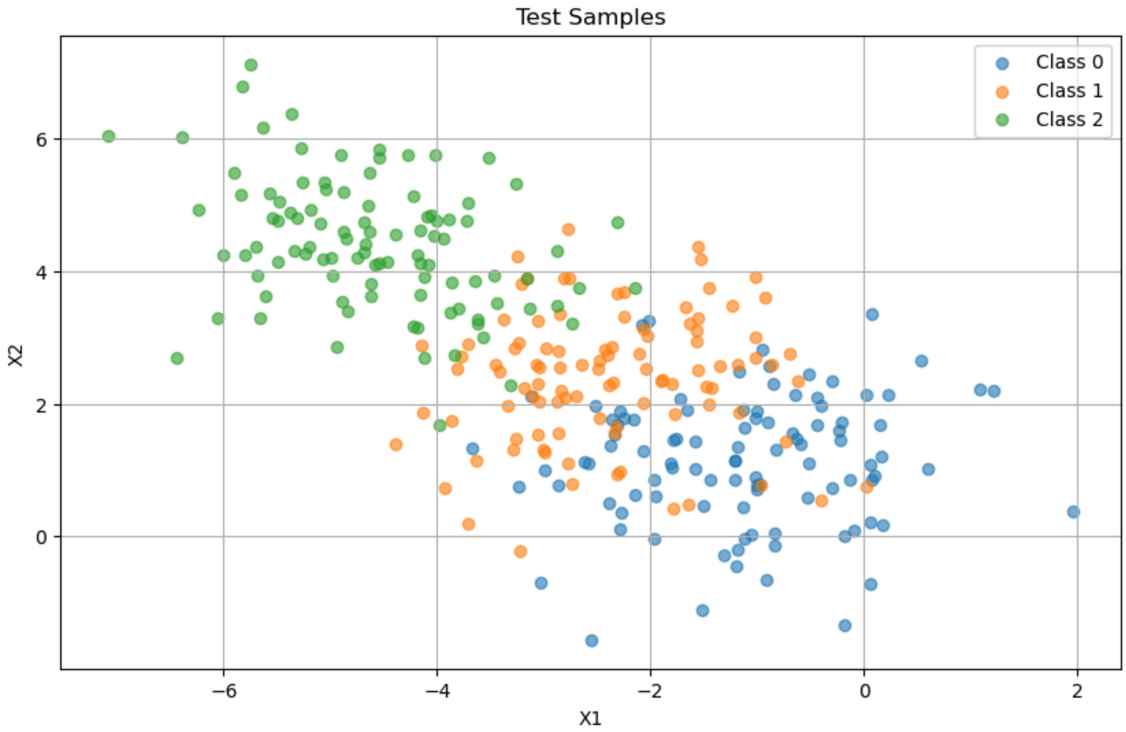
Question 1.2

In [4]:
```
# Plotting the training samples
plt.figure(figsize=(10, 6))
plt.scatter(samples1[:, 0], samples1[:, 1], label='Class 0',
            alpha=0.6)
plt.scatter(samples2[:, 0], samples2[:, 1], label='Class 1', alpha=0.6)
plt.scatter(samples3[:, 0], samples3[:, 1], label='Class 2', alpha=0.6)
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Training Samples')
plt.grid(True)
plt.show()
```



Question 1.3

In [5]:
```
# Generate test samples
n_test_samples = 300
test_samples_per_class = n_test_samples // n_classes

test_samples1 = np.random.multivariate_normal(mu1, sigma,
                                              test_samples_per_class)
test_samples2 = np.random.multivariate_normal(mu2, sigma, test_samples_per_
test_samples3 = np.random.multivariate_normal(mu3, sigma, test_samples_per_

# Combine the test samples
X_test = np.vstack([test_samples1, test_samples2, test_samples3])
y_test = np.hstack([[0]*test_samples_per_class,
                    [1]*test_samples_per_class, [2]*test_samples_per_class]

# Plotting the test samples
plt.figure(figsize=(10, 6))
plt.scatter(test_samples1[:, 0], test_samples1[:, 1], label='Class 0',
            alpha=0.6)
plt.scatter(test_samples2[:, 0], test_samples2[:, 1], label='Class 1', alph
plt.scatter(test_samples3[:, 0], test_samples3[:, 1], label='Class 2', alph
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Test Samples')
plt.grid(True)
plt.show()
```



Question 1.4

In [6]:
```
# Train k-NN classifier
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
knn.fit(X_train, y_train)

# Predict on the training set
y_train_pred = knn.predict(X_train)

# Predict on the test set
y_test_pred = knn.predict(X_test)

# Calculate classification error rates
train_error_rate = np.mean(y_train_pred != y_train)
test_error_rate = np.mean(y_test_pred != y_test)

print(f'Classification Error Rate on the training set:
      {train_error_rate * 100:.2f}%')
print(f'Classification Error Rate on the test set:
      {test_error_rate * 100:.2f}%')
```

```
Classification Error Rate on the training set: 0.00%
Classification Error Rate on the test set: 21.33%
```

Train set error is low (0%) as expected because of the algorithm structure - each training point is its nearest neighbor, resulting in perfect classification on the training set. However in the Test set, the error rate is much higher with approx ~20% because the test set contains new data. Using k=1, the classifier may not classify all test points correctly. We've seen in the lecture, for 1-NN there is a case of Overfitting, where the model fits “too well” on the training data, yet its performance on the “true” world is very poor.

Question 1.5

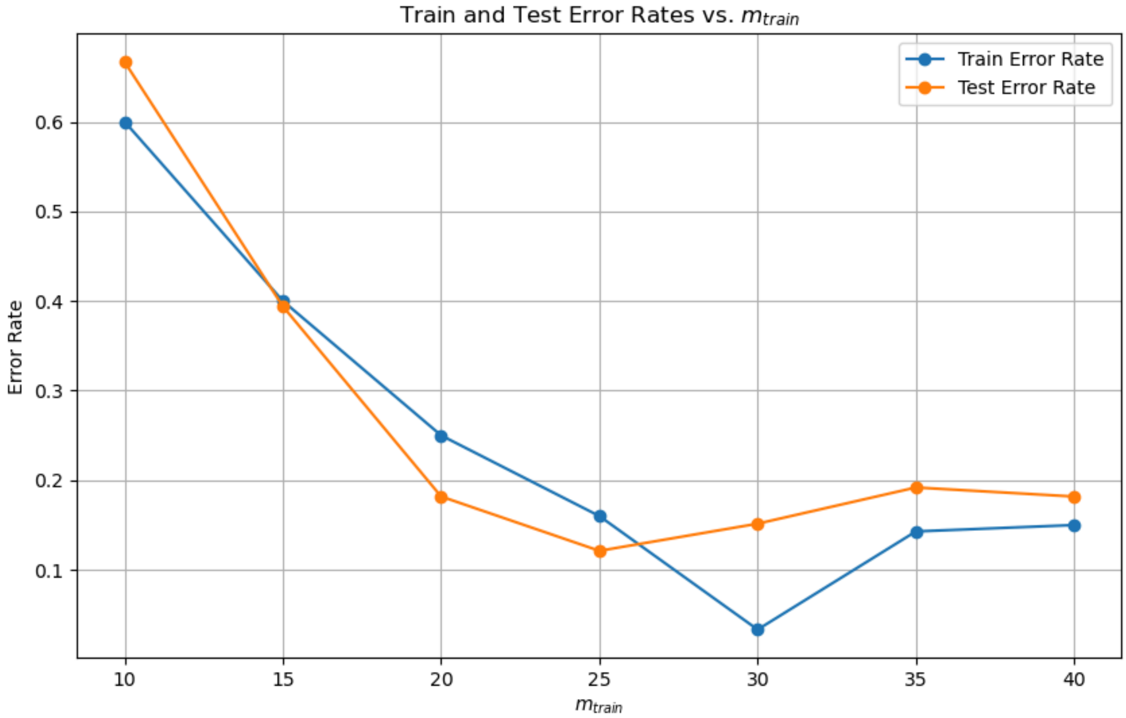
In [7]:
```
# Initialize lists to store error rates
train_error_rates = []
test_error_rates = []

# Iterate over k values from 1 to 20
for k in range(1, 21):
    # Train k-NN classifier
    knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
    knn.fit(X_train, y_train)

    # Predict on the training set
    y_train_pred = knn.predict(X_train)

    # Predict on the test set
    y_test_pred = knn.predict(X_test)

    # Calculate classification error rates
    train_error_rate = np.mean(y_train_pred != y_train)
    test_error_rate = np.mean(y_test_pred != y_test)

    # Store the error rates
    train_error_rates.append(train_error_rate)
    test_error_rates.append(test_error_rate)

# Plotting the error rates
plt.figure(figsize=(10, 6))
plt.plot(range(1, 21), train_error_rates, label='Train Error Rate', marker=
plt.plot(range(1, 21), test_error_rates, label='Test Error Rate', marker='o
plt.xlabel('k')
plt.ylabel('Error Rate')
plt.title('Train and Test Error Rates vs. k')
plt.legend()
plt.grid(True)
plt.show()
```

The test error does not always decrease with k. We can see that in our data, te optimal k is 10, where the error rate is mininal for the test set. The optimal k balances bias and variance, minimizing the test error. The behavior of error rates as a function of k reflects the trade-off between overfitting and underfitting. For a k>10 we can see that the error rate increases as the model becomes less sensitive causing it to have trouble to properly classify the data, leading it to become underfit.

Question 1.6

In [8]:
```
# Generate test samples
n_test_samples = 100
test_samples_per_class = n_test_samples // 3

test_samples1 = np.random.multivariate_normal(mu1, sigma, test_samples_per_
test_samples2 = np.random.multivariate_normal(mu2, sigma, test_samples_per_
test_samples3 = np.random.multivariate_normal(mu3, sigma, test_samples_per_

X_test = np.vstack([test_samples1, test_samples2, test_samples3])
y_test = np.hstack([[0]*test_samples_per_class,
                    [1]*test_samples_per_class, [2]*test_samples_per_class]

# Initialize lists to store error rates
train_error_rates = []
test_error_rates = []

# Values of m_train to test
m_train_values = list(range(10, 41, 5))

# MUST CHANGE TO RANDOM
# Iterate over m_train values
for m_train in m_train_values:
    # Allocate samples to each class
    base_samples_per_class = m_train // 3
    remainder = m_train % 3

    samples_per_class = [base_samples_per_class] * 3
    for i in range(remainder):
        samples_per_class[i] += 1

    # Generate training samples
    samples1 = np.random.multivariate_normal(mu1, sigma, samples_per_class[
    samples2 = np.random.multivariate_normal(mu2, sigma, samples_per_class[
    samples3 = np.random.multivariate_normal(mu3, sigma, samples_per_class[

    X_train = np.vstack([samples1, samples2, samples3])
    y_train = np.hstack([[0]*samples_per_class[0],
                         [1]*samples_per_class[1], [2]*samples_per_class[2]

    # Ensure k does not exceed the number of training samples
    current_k = min(10, len(y_train))

    # Train k-NN classifier with k=10
    knn = KNeighborsClassifier(n_neighbors=current_k, metric='euclidean')
    knn.fit(X_train, y_train)

    # Predict on the training set
    y_train_pred = knn.predict(X_train)

    # Predict on the test set
    y_test_pred = knn.predict(X_test)

    # Calculate classification error rates
    train_error_rate = np.mean(y_train_pred != y_train)
    test_error_rate = np.mean(y_test_pred != y_test)

    # Store the error rates
    train_error_rates.append(train_error_rate)
    test_error_rates.append(test_error_rate)

# Plotting the error rates
```

```
plt.figure(figsize=(10, 6))
plt.plot(m_train_values, train_error_rates, label='Train Error Rate', marke
plt.plot(m_train_values, test_error_rates, label='Test Error Rate', marker=
plt.xlabel('$m_{train}$')
plt.ylabel('Error Rate')
plt.title('Train and Test Error Rates vs. $m_{train}$')
plt.legend()
plt.grid(True)
plt.show()
```



As expected, we see that when the number of samples in the training set increases - the error rate will decrease as we have more data to train our algorithm on, resulting in higher accuracy and decrease of the error rate in both the train and test set. As we can see for the plot, the error rate decrease is not linear.

Question 1.7

The plot changes between trials, we can assume that the cause of it is that the train set is of a small size which might affect the process of training each time and contributes to the changes of each run. Moreover, we can see that each iteration the error rate decreases.

Question 1.8

The variation of a k-NN classifier where the distances of the neighbors are taken into account in the prediction is Distance-weighted k-NN regression rule as we've seen in tutorial 1.

☆ שאלה 3 סעיף 1:

צ"ל:

$$P_w(y_i=1|x_i)=\frac{e^{w^T x_i}}{1+e^{w^T x_i}} =: p$$

מקרה פרטי של:

$$P_w(y_i=k|x_i) = \frac{e^{w_k^T x_i}}{\sum_{j=1}^{k} e^{w_j^T x_i}} =: p_k$$

הוכחה:

עבור $k=2$ מתקיים: $w_1, w_2 \in \mathbb{R}^d$.

נסמן:

$$p_1 = \frac{e^{w_1^T x_i}}{e^{w_1^T x_i}+e^{w_2^T x_i}} = \frac{e^{w^T x_i}}{1+e^{w^T x_i}} = p$$

$$e^{w_1^T x_i}(1+e^{w^T x_i}) = e^{w^T x_i}(e^{w_1^T x_i}+e^{w_2^T x_i})$$

$$e^{w_1^T x_i} + \cancel{e^{w_1^T x_i} e^{w^T x_i}} = \cancel{e^{w^T x_i} e^{w_1^T x_i}} + e^{w^T x_i} e^{w_2^T x_i}$$

$$e^{w_1^T x_i} = e^{w^T x_i + w_2^T x_i}$$

$$w_1^T x_i = w^T x_i + w_2^T x_i$$

$$w^T = w_1^T - w_2^T = (w_1-w_2)^T$$

$$\Rightarrow w = w_1 - w_2 \quad \Rightarrow w_2 = w_1 - w$$

$$p_2 = \frac{e^{w_2^T x_i}}{e^{w_1^T x_i}+e^{w_2^T x_i}} =$$

$$= \frac{e^{(w_1-w)^T x_i}}{e^{w_1^T x_i}+e^{w_2^T x_i}} = \frac{e^{w_1^T x_i}}{e^{w^T x_i}(e^{w_1^T x_i}+e^{w_2^T x_i})} =$$

$$= \frac{e^{w_1^T x_i}}{e^{(w+w_1)^T x_i}+e^{(w+w_2)^T x_i}} = \frac{e^{w_1^T x_i}}{e^{w_1^T x_i}(e^{w^T x_i}+1)} =$$

$$= \frac{1}{1+e^{w^T x_i}} = \frac{1+e^{w^T x_i}-e^{w^T x_i}}{1+e^{w^T x_i}} = 1-\frac{e^{w^T x_i}}{1+e^{w^T x_i}} =$$

$$= 1-p = p_2$$

✧ שאלה 3 סעיף 2:

$$P(y_i = k \mid x_i) = P_{ik} = \frac{e^{w_k^T x_i}}{\sum_{i=1}^{k} e^{w_k^T x_i}} \quad , w_1, \dots, w_k \in \mathbb{R}^d$$

$$\log(P(x_i)) = \log\left(\frac{e^{w_k^T x_i}}{\sum_{i=1}^{k} e^{w_k^T x_i}}\right) =$$

$$= w_k^T x_i - \log\left(\sum_{i=1}^{k} e^{w_k^T x_i}\right)$$

$$l_s(w) = \sum_{i=1}^{m} \log(P(x_i)) = \sum_{i=1}^{m}\left(w_k^T x_i - \log\left(\sum_{i=1}^{k} e^{w_k^T x_i}\right)\right)$$

$$\frac{d l_s(w)}{d(w_k)} = \sum_{i=1}^{m} I(y_i = k) \cdot x_i - \sum_{i=1}^{m} \frac{e^{w_k^T x_i} \cdot x_i}{\sum_{i=1}^{k} e^{w_k^T x_i}} = 0$$

$$\sum_{i=1}^{m} I(y_i = k) \cdot x_i = \sum_{i=1}^{m} \frac{e^{w_k^T x_i} \cdot x_i}{\sum_{i=1}^{k} e^{w_k^T x_i}}$$

נשים לב כי w_i כך ש-$i=1,\dots,k$ הם k משתנים שונים,
בהתאם ללייבל. כפי שראינו בהרצאה, ניתן לפתור
את בעיית אופטימיזציה זו ע"י אלגוריתמים כמו:
Stochastic Gradient Descent.

✧ שאלה 3 סעיף 3 :

א) כמות הוקטורים w_i , $i=1,\dots,k$, שווה לכמות הלייבלים".
לכל לייבל יש w משלו, כלומר לכל מחלקה על-מישור משלה.
לפי נתוני השאלה, קיימות 3 לייבלים אפשריים ל-x_i לכן
קיימים 3 וקטורים w_1,w_2,w_3.

ב) כפי שראינו בהרצאה, אחת הכניסות בוקטור w_i
תכיל את ה- bias . לכן אם $x_i \in \mathbb{R}^k \Rightarrow w_i \in \mathbb{R}^{k+1}$.

ג) בהמשך לסעיף ב, נניח שהכניסה הראשונה של וקטור w_i
תתאים ל- bias , כלומר $x_{i0}=1$. מכאן:

$$\sum_{i=1}^{k} e^{w_k^T x_1} =$$

$$= e^{8\cdot1+-2.5\cdot1+2\cdot8}+e^{2\cdot1+0.5\cdot1-1.5\cdot8}+e^{-10\cdot1+2\cdot1-0.5\cdot8}=$$

$$= e^{21.5}+e^{-9.5}+e^{-12}$$

$$\left.\begin{array}{l}
p_{11} = \dfrac{e^{w_1^T x_1}}{\sum_{i=1}^{k} e^{w_k^T x_1}} = \dfrac{e^{21.5}}{e^{21.5}+e^{-9.5}+e^{-12}} \approx 1 \\[2ex]
p_{12} = \dfrac{e^{w_2^T x_1}}{\sum_{i=1}^{k} e^{w_k^T x_1}} = \dfrac{e^{-9.5}}{e^{21.5}+e^{-9.5}+e^{-12}} \approx 0 \\[2ex]
p_{13} = \dfrac{e^{w_3^T x_1}}{\sum_{i=1}^{k} e^{w_k^T x_1}} = \dfrac{e^{-12}}{e^{21.5}+e^{-9.5}+e^{-12}} \approx 0
\end{array}\right\} \Rightarrow \hat{y}_1 = 0$$

$$\sum_{i=1}^{K} e^{w_K^T x_2} =$$

$$= e^{8\cdot 1 - 2.5\cdot 6 + 2\cdot -2} + e^{2\cdot 1 + 0.5\cdot 6 - 1.5\cdot -2} + e^{-10\cdot 1 + 2\cdot 6 - 0.5\cdot -2} =$$

$$= e^{-11} + e^{8} + e^{3}$$

$$P_{21} = \frac{e^{w_1^T x_2}}{\sum_{i=1}^{K} e^{w_K^T x_2}} = \frac{e^{-11}}{e^{-11} + e^{8} + e^{3}} \approx 5.56\cdot 10^{-9}$$

$$P_{22} = \frac{e^{w_2^T x_2}}{\sum_{i=1}^{K} e^{w_K^T x_2}} = \frac{e^{8}}{e^{-11} + e^{8} + e^{3}} \approx 0.99 \qquad \Rightarrow \hat{y}_2 = 1$$

$$P_{23} = \frac{e^{w_3^T x_2}}{\sum_{i=1}^{K} e^{w_K^T x_2}} = \frac{e^{3}}{e^{-11} + e^{8} + e^{3}} \approx 6.69\cdot 10^{-3}$$

$$\sum_{i=1}^{K} e^{w_K^T x_3} =$$

$$= e^{8\cdot 1 - 2.5\cdot 12 + 2\cdot 4} + e^{2\cdot 1 + 0.5\cdot 12 - 1.5\cdot 4} + e^{-10\cdot 1 + 2\cdot 12 - 0.5\cdot 4} =$$

$$= e^{-14} + e^{2} + e^{12}$$

$$P_{31} = \frac{e^{w_1^T x_3}}{\sum_{i=1}^{K} e^{w_K^T x_3}} = \frac{e^{-14}}{e^{-14} + e^{2} + e^{12}} \approx 5.1\cdot 10^{-12}$$

$$P_{32} = \frac{e^{w_2^T x_3}}{\sum_{i=1}^{K} e^{w_K^T x_3}} = \frac{e^{2}}{e^{-14} + e^{2} + e^{12}} \approx 4.53\cdot 10^{-5} \qquad \Rightarrow \hat{y}_2 = 2$$

$$P_{33} = \frac{e^{w_3^T x_3}}{\sum_{i=1}^{K} e^{w_K^T x_3}} = \frac{e^{12}}{e^{-14} + e^{2} + e^{12}} \approx 0.99$$