Course: Machine Learning by Dr. Seyyed Salehi

Homework: HW4

Name: Mohammad Mohammadi

Student ID: 402208592

Question 1

Suppose we want to construct a decision tree that predicts in which games AI is better than human players. The table below shows the dataset, with x_1, x_2, and x_3 as features and y as the target output. We display the dataset with x and y in the table below.

x_1 (Team or Individual)	x_2 (Mental or Physical)	x_3 (Skill or Chance)	y (Win or Lose)
T	M	S	W
I	M	S	W
T	P	S	W
I	M	C	W
T	P	S	L
I	M	C	L
T	P	C	L
T	P	C	L
T	P	C	L
I	P	S	W

a) Calculate the entropy (H(y)) of class (y).

b) According to the information gain criterion, construct the best tree with a depth of 1 and perform the calculations in full detail. Show which feature at the root of the tree will split.

Answer

Part (a)

The entropy (H(y)) is calculated using the formula: $H(y) = -\sum_{i=1}^{k} p_i \log_2(p_i)$ where (p_i) is the probability of class (i).

In our dataset, (y) (Win or Lose) has 10 samples:

- 5 samples are Win (W)
- 5 samples are Lose (L)

The probabilities are:

$$p(\text{Win}) = \frac{5}{10} = 0.5$$

$$p(\text{Lose}) = \frac{5}{10} = 0.5$$

Now, calculate the entropy:

$$H(y) = -(0.5 \log_2(0.5) + 0.5 \log_2(0.5))$$

$$H(y) = -(0.5 \cdot (-1) + 0.5 \cdot (-1))$$

$$H(y) = -(-0.5 - 0.5)$$

$$H(y) = 1$$

So, the entropy (H(y)) is 1.

Part (b)

To find the best feature to split on, we need to calculate the information gain for each feature. As the information gain is calculated as follows:

$$\text{Information Gain}(X) = H(y) - H(y|X)$$

Where $H(y|X)$ is the conditional entropy of y given X.

1. Calculating $H(y|x_1)$

Feature x_1 (Team or Individual):

- For $x_1 = T$:
  - 5 samples: $\{W, W, L, L, L\}$
  - $p(\text{Win}) = \frac{2}{5}$, $p(\text{Lose}) = \frac{3}{5}$
  - Entropy $H(y|x_1 = T) = -\left(\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5}\right)$

$$H(y|x_1 = T) = -\left(\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$= -\left(0.4\log_2 0.4 + 0.6\log_2 0.6\right)$$

$$= -\left(0.4 \cdot -1.322 + 0.6 \cdot -0.737\right)$$

$$= -\left(-0.529 - 0.442\right)$$

$$= 0.971$$

- For $x_1 = I$:
  - 5 samples: $\{W, W, W, L, W\}$
  - $p(\text{Win}) = \frac{4}{5}$, $p(\text{Lose}) = \frac{1}{5}$
  - Entropy $H(y|x_1 = I) = -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$

$$H(y|x_1 = I) = -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$$

$$= -\left(0.8\log_2 0.8 + 0.2\log_2 0.2\right)$$

$$= -\left(0.8 \cdot -0.322 + 0.2 \cdot -2.322\right)$$

$$= -\left(-0.258 - 0.464\right)$$

$$= 0.722$$

Now, we calculate the conditional entropy $H(y|x_1)$:

$$H(y|x_1) = \frac{5}{10}H(y|x_1 = T) + \frac{5}{10}H(y|x_1 = I)$$

$$= 0.5 \cdot 0.971 + 0.5 \cdot 0.722$$

$$= 0.4855 + 0.361$$

$$= 0.8465$$

Information Gain for x_1:

$$\text{IG}(x_1) = H(y) - H(y|x_1)$$

$$= 1 - 0.8465$$

$$= 0.1535$$

2. Calculating $H(y|x_2)$

Feature x_2 (Mental or Physical):

- For $x_2 = M$:
  - 5 samples: $\{W, W, W, W, L\}$
  - $p(\text{Win}) = \frac{4}{5}$, $p(\text{Lose}) = \frac{1}{5}$
  - Entropy $H(y|x_2 = M) = -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$
  - We have already calculated this as 0.722.
- For $x_2 = P$:
  - 5 samples: $\{W, L, L, L, L\}$
  - $p(\text{Win}) = \frac{1}{5}$, $p(\text{Lose}) = \frac{4}{5}$
  - Entropy $H(y|x_2 = P) = -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right)$

$$H(y|x_2 = P) = -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right)$$

$$= -\left(0.2\log_2 0.2 + 0.8\log_2 0.8\right)$$

$$= -\left(0.2 \cdot -2.322 + 0.8 \cdot -0.322\right)$$

$$= -\left(-0.464 - 0.258\right)$$

$$= 0.722$$

Now, we calculate the conditional entropy $H(y|x_2)$:

$$H(y|x_2) = \frac{5}{10}H(y|x_2 = M) + \frac{5}{10}H(y|x_2 = P)$$

$$= 0.5 \cdot 0.722 + 0.5 \cdot 0.722$$

$$= 0.361 + 0.361$$

$$= 0.722$$

Information Gain for x_2:

$$\text{IG}(x_2) = H(y) - H(y|x_2)$$

$$= 1 - 0.722$$

$$= 0.278$$

3. Calculating $H(y|x_3)$

Feature x_3 (Skill or Chance):

- For $x_3 = S$:
  - 5 samples: $\{W, W, W, L, W\}$
  - $p(\text{Win}) = \frac{4}{5}$, $p(\text{Lose}) = \frac{1}{5}$
  - Entropy $H(y|x_3 = S) = -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$
  - We have already calculated this as 0.722.
- For $x_3 = C$:
  - 5 samples: $\{W, L, L, L, L\}$
  - $p(\text{Win}) = \frac{1}{5}$, $p(\text{Lose}) = \frac{4}{5}$

- Entropy $H(y|x_3 = C) = -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right)$
- We have already calculated this as 0.722.

Now, we calculate the conditional entropy $H(y|x_3)$:

$$H(y|x_3) = \frac{5}{10}H(y|x_3 = S) + \frac{5}{10}H(y|x_3 = C)$$
$$= 0.5 \cdot 0.722 + 0.5 \cdot 0.722$$
$$= 0.361 + 0.361$$
$$= 0.722$$

Information Gain for x_3:

$$\text{IG}(x_3) = H(y) - H(y|x_3)$$
$$= 1 - 0.722$$
$$= 0.278$$

Conclusion

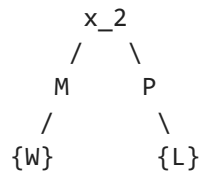
Comparing the information gains for each feature:

- $\text{IG}(x_1) = 0.1535$
- $\text{IG}(x_2) = 0.278$
- $\text{IG}(x_3) = 0.278$

The features x_2 (Mental or Physical) and x_3 (Skill or Chance) have the highest information gain of 0.278. We can choose either one as the root of the tree. We will select x_2 as the root.

Final Decision Tree of Depth 1

The decision tree with depth 1 and the feature x_2 at the root will look like this:



Where:

- If $x_2 = M$, the outcome is Win (W).
- If $x_2 = P$, the outcome is Lose (L).

Question 2

Consider the AdaBoost algorithm. In this algorithm, T classifiers with names $G_1, \ldots, G_T$ are repeatedly trained on the given data, and in each step, the weight assigned to the training data is adjusted. Suppose z is a test sample from the dataset. The output of the metalearner(main model) is a linear combination of the outputs of the classifiers (mainly):

$$M_T(z) = \sum_{t=1}^{T} \beta_t G_t(z)$$

where $M_T(z)$ is the output of the meta-learner, and $G_t(z)$ is the output of the t-th classifier for sample z.

The AdaBoost model, trains metalearner M_T with exponential training cost function. In this question, the boosting algorithm named Gradient Boosting is reviewed. Suppose $\rho = M_T(z)$ is the output metalearner in this algorithm, and $\ell \in \{+1, -1\}$ is the actual label for sample z. The Gradient Boosting algorithm replaces the exponential cost function with the squared error loss function:

$$L(\rho, \ell) = (\rho - \ell)^2$$

Suppose the classification is done with 2 classes and we have n training data points $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ and vector $y \in \mathbb{R}^n$ of labels as $y_i \in \{\pm 1\}$.

a) Write an expression for the cost function of this model. Pay attention to the fact that based on explanations the cost function is the mean squared error (MSE). Answer based on β_ts and y_is and G_ts.

b) Suppose we are in step T of the Gradient Boosting algorithm, and we have trained weak classifier G_T but we have not yet determined β_t. Also, suppose that the coefficients $\beta_1, \beta_2, \ldots, \beta_{t-1}$ in step T remain fixed and do not change. Determine β_T in a way that the training cost of the meta-learner is minimized. YOur expression should be as simplified as possible.

Answer

Part (a)

Given the model:

$$M_T(z) = \sum_{t=1}^{T} \beta_t G_t(z),$$

we aim to minimize the squared error between the model's prediction $M_T(z)$ and the actual label ℓ.

For the n training data points (X_i, y_i), the cost function can be written as:

$$L = \frac{1}{n} \sum_{i=1}^{n} (M_T(X_i) - y_i)^2.$$

Substituting $M_T(X_i) = \sum_{t=1}^{T} \beta_t G_t(X_i)$, we get:

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{t=1}^{T} \beta_t G_t(X_i) - y_i \right)^2.$$

This is the expression for the cost function based on β_t, y_i, and $G_t(X_i)$.

Part (b)

We are at step T of the Gradient Boosting algorithm, where we have trained the weak classifier G_T but have not yet determined β_T. The coefficients $\beta_1, \beta_2, \ldots, \beta_{T-1}$ are fixed. We aim to find β_T such that the training cost is minimized.

The cost function at step T is:

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{t=1}^{T} \beta_t G_t(X_i) - y_i \right)^2$$

Let's denote the output of the model up to step $T-1$ as:

$$M_{T-1}(X_i) = \sum_{t=1}^{T-1} \beta_t G_t(X_i)$$

Then, the cost function can be rewritten as:

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(M_{T-1}(X_i) + \beta_T G_T(X_i) - y_i \right)^2$$

To minimize L with respect to β_T, we take the derivative of L with respect to β_T and set it to zero:

$$\frac{\partial L}{\partial \beta_T} = \frac{\partial}{\partial \beta_T} \left(\frac{1}{n} \sum_{i=1}^{n} \left(M_{T-1}(X_i) + \beta_T G_T(X_i) - y_i \right)^2 \right) = 0$$

Carrying out the differentiation, we have:

$$\frac{\partial L}{\partial \beta_T} = \frac{1}{n} \sum_{i=1}^{n} 2 \left(M_{T-1}(X_i) + \beta_T G_T(X_i) - y_i \right) G_T(X_i) = 0$$

Simplifying, we get:

$$\frac{1}{n} \sum_{i=1}^{n} \left(M_{T-1}(X_i) + \beta_T G_T(X_i) - y_i \right) G_T(X_i) = 0$$

Rearranging to solve for β_T:

$$\sum_{i=1}^{n} \left(M_{T-1}(X_i) G_T(X_i) + \beta_T G_T(X_i)^2 - y_i G_T(X_i) \right) = 0$$

$$\sum_{i=1}^{n} M_{T-1}(X_i) G_T(X_i) + \beta_T \sum_{i=1}^{n} G_T(X_i)^2 = \sum_{i=1}^{n} y_i G_T(X_i)$$

$$\beta_T \sum_{i=1}^{n} G_T(X_i)^2 = \sum_{i=1}^{n} y_i G_T(X_i) - \sum_{i=1}^{n} M_{T-1}(X_i) G_T(X_i)$$

$$\beta_T = \frac{\sum_{i=1}^{n} y_i G_T(X_i) - \sum_{i=1}^{n} M_{T-1}(X_i) G_T(X_i)}{\sum_{i=1}^{n} G_T(X_i)^2}$$

Thus, the coefficient β_T that minimizes the training cost at step T is:

$$\beta_T = \frac{\sum_{i=1}^{n} y_i G_T(X_i) - \sum_{i=1}^{n} M_{T-1}(X_i) G_T(X_i)}{\sum_{i=1}^{n} G_T(X_i)^2}$$

Question 3

Answer the following questions.

(a) If we use bootstrapping from the data of size N to choose $N' = pN$ samples, show that approximately $e^{-p}N$ data points will not be selected in the entire sampling process. (Assume N is very large.)

(b) Consider a random forest G consisting of three binary decision trees $\{g_k\}_{k=1}^{3}$. Assume that each of these decision trees has the following errors on test data:

$$E_{\text{out}}(g_1) = 0.15, \quad E_{\text{out}}(g_2) = 0.25, \quad E_{\text{out}}(g_3) = 0.35$$

Find the error bound $E_{\text{out}}(G)$ with reasoning.

Answer

Part (a)

In bootstrapping, each sample is drawn with replacement. For each data point x_i, the probability that it is not selected in a single draw is:

$$1 - \frac{1}{N}$$

When drawing $N' = pN$ samples, the probability that x_i is not selected at all is:

$$\left(1 - \frac{1}{N}\right)^{pN}$$

For large N, we can use the approximation:

$$\left(1 - \frac{1}{N}\right)^{pN} \approx e^{-p}$$

Thus, the expected number of data points that are not selected is:

$$N \cdot e^{-p}$$

Hence, approximately $e^{-p}N$ data points will not be selected in the entire sampling process.

Part (b)

The random forest G makes its decision by majority vote of the three trees. We need to find the error bound $E_{\text{out}}(G)$.

First, let's denote the errors as:

$$e_1 = 0.15, \quad e_2 = 0.25, \quad e_3 = 0.35$$

Assume the errors of the individual trees are independent. The error of the majority vote can be analyzed by considering the probability that at least two out of three trees make an incorrect prediction.

The probability that each tree makes a correct prediction is:

$$1 - e_1 = 0.85, \quad 1 - e_2 = 0.75, \quad 1 - e_3 = 0.65$$

The probability that the majority vote is incorrect (at least two trees make an incorrect prediction) can be calculated using the binomial distribution. There are three cases to consider for at least two errors:

1. Exactly two trees are wrong.
2. All three trees are wrong.

Let's calculate these probabilities:

1. Probability that exactly two trees are wrong:

$$\text{P}(2 \text{ errors}) = \binom{3}{2} \cdot e_i e_j (1 - e_k)$$

where (i, j, k) are all permutations of (1, 2, 3).

Summing over all permutations:

$$\text{P}(2 \text{ errors}) = 3 \cdot e_1 e_2 (1 - e_3) + 3 \cdot e_1 e_3 (1 - e_2) + 3 \cdot e_2 e_3 (1 - e_1)$$

2. Probability that all three trees are wrong:

$$\text{P}(3 \text{ errors}) = e_1 e_2 e_3$$

Now, substituting the given error values:

$$\text{P}(2 \text{ errors}) = 3 \cdot 0.15 \cdot 0.25 \cdot 0.65 + 3 \cdot 0.15 \cdot 0.35 \cdot 0.75 + 3 \cdot 0.25 \cdot 0.35 \cdot 0.85$$

$$= 3 \cdot 0.024375 + 3 \cdot 0.039375 + 3 \cdot 0.074375$$

$$= 3 \cdot (0.024375 + 0.039375 + 0.074375)$$

$$= 3 \cdot 0.138125$$

$$= 0.414375$$

And for all three errors:

$$\mathrm{P}(3 \text{ errors}) = 0.15 \cdot 0.25 \cdot 0.35$$

$$= 0.013125$$

Adding these probabilities together:

$$E_{\text{out}}(G) = \mathrm{P}(2 \text{ errors}) + \mathrm{P}(3 \text{ errors})$$

$$= 0.414375 + 0.013125$$

$$= 0.4275$$

Therefore, the error bound $E_{\text{out}}(G)$ for the random forest G consisting of the three decision trees is:

$$E_{\text{out}}(G) \approx 0.4275.$$

Question 4

Which of the following statements about bagging and boosting is correct? (Multiple choices may be correct)

a) Different learners in bagging can be trained in parallel.

b) Different learners in boosting can be trained in parallel.

c) Each of the learners in bagging is trained on the entire training data.

d) Each of the learners in boosting is trained on the entire training data.

Answer

a) Correct.

b) Incorrect.

c) Incorrect.

d) Correct.

a) In bagging (Bootstrap Aggregating), each learner (model) is trained independently on a different bootstrap sample of the data. Therefore, the training of different learners can be done in parallel since there are no dependencies between them during the training phase.

b) In boosting, each learner is trained sequentially, with each one attempting to correct the errors of its predecessor. The training process is dependent on the previous models because the weights of the training data points are adjusted based on the errors of the previous model. Thus, different learners in boosting cannot be trained in parallel.

c) In bagging, each learner is trained on a bootstrap sample of the data, which means a random subset of the training data with replacement. Therefore, each learner is not trained on the entire training data but on a subset of it.

d) In boosting, each learner is trained on the entire training data set. However, the weights of the training data points are adjusted after each learner is trained to focus more on the misclassified points. So, while each learner sees the entire training data, the emphasis on different data points changes with each iteration.

Question 5

In this question, consider a KNN classifier with the L_2 distance metric. Assume that the classes are completely binary. Answer the following questions based on the dataset shown below.

a) What is the error rate of this classifier for a specific K value? How much is this error rate?

b) Why can using very large or very small K values on this dataset be erroneous?

c) Suppose we use the leave-one-out cross-validation method. What is the error rate for this value of K?

d) Show the decision boundary of this 1NN classifier for this dataset in the image.

Answer

Part (a)

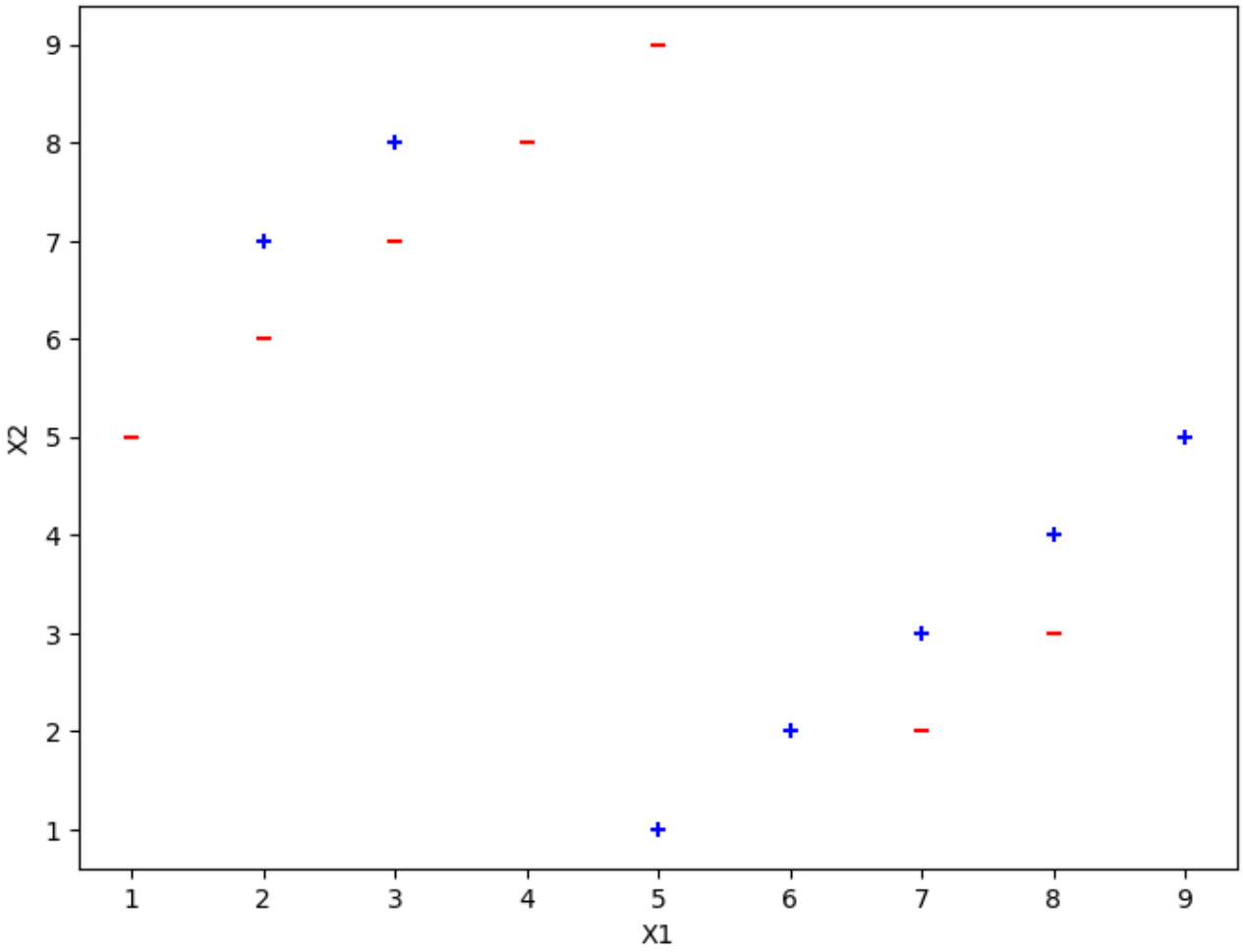
Considering the point itself it's own nearest neighbour, based on TA's explanations, the least error is 0 and it happens in k=1, 2 and 4 as in k=1 the point will always be classified as it's label and for k=2 and 4 there is equal votes and we consider the the ecludian distance and weight based on the order we observe neighbours which we see the points label first.

I used a python script for this matter as below in which I plot the points and calculate based on value of k.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter


def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def predict_label(point, points, labels, k):
    distances = []
    for i in range(len(points)):
        distance = euclidean_distance(point, points[i])
        distances.append((distance, labels[i]))

    distances.sort(key=lambda x: x[0])
    k_nearest_labels = [label for _, label in distances[:k]]
    # print(k_nearest_labels, k, point)

    most_common = Counter(k_nearest_labels).most_common(1)
    return most_common[0][0]

def knn_errors(points, labels, k_values):
    errors = {}

    numeric_labels = np.array([1 if label == '+' else 0 for label in labels])

    for k in k_values:
        predictions = []
        for i in range(len(points)):
            predicted_label = predict_label(points[i], points, numeric_labels, k)
            predictions.append(predicted_label)

        errors[k] = np.mean(np.array(predictions) != numeric_labels)
```

```
    return errors

def plot_data(points, labels):
    plt.figure(figsize=(8, 6))
    for point, label in zip(points, labels):
        if label == '+':
            plt.scatter(point[0], point[1], c='blue', marker='+')
        else:
            plt.scatter(point[0], point[1], c='red', marker='_')

    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.grid(False)
    plt.show()

# Data
points = np.array([
    [1, 5], [2, 6], [2, 7], [3, 7], [3, 8], [4, 8],
    [5, 1], [5, 9], [6, 2], [7, 2], [7, 3], [8, 3], [8, 4], [9, 5]
])
labels = ['-', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '+']
k_values = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# k_values = [3]

errors = knn_errors(points, labels, k_values)
for err in errors:
    print("K= " + str(err) + ", error= " + str(errors[err]))

plot_data(points, labels)
```

```
K= 1, error= 0.0
K= 2, error= 0.0
K= 3, error= 0.42857142857142855
K= 4, error= 0.0
K= 5, error= 0.2857142857142857
K= 6, error= 0.2857142857142857
K= 7, error= 0.2857142857142857
K= 8, error= 0.2857142857142857
K= 9, error= 0.2857142857142857
```



Part (b)

- **With (K = 1):** The classifier will be highly sensitive to each point's label and each point's nearest neighbor for k = 2. In the dataset, points close to the boundary between classes ('+' and '-') can be easily misclassified if their immediate neighbor belongs to the opposite class. This sensitivity can lead to high error rates in regions where the classes are intermixed or close together.
- **With Very Large (K):** The classifier will consider many neighbors, possibly including points from distant parts of the feature space. For example, using a large (K) will average over points from different clusters, leading to misclassification of points in minority clusters. In the image, '+' and '-' points are spread across different regions. A large (K) would cause the classifier to ignore these local regions, leading to a model that cannot accurately represent the true decision boundaries.

Using very small K values can lead to high variance and sensitivity to noise, resulting in overfitting. Conversely, using very large K values can lead to high bias and underfitting, resulting in a model that fails to capture the data's structure. Both scenarios can significantly degrade the classifier's performance on this dataset. Therefore, choosing an appropriate K value is crucial to balancing bias and variance, ensuring good generalization performance.

Part (c)

This is similar to part (a) just not considering the point itself as it's own nearest neighbour.

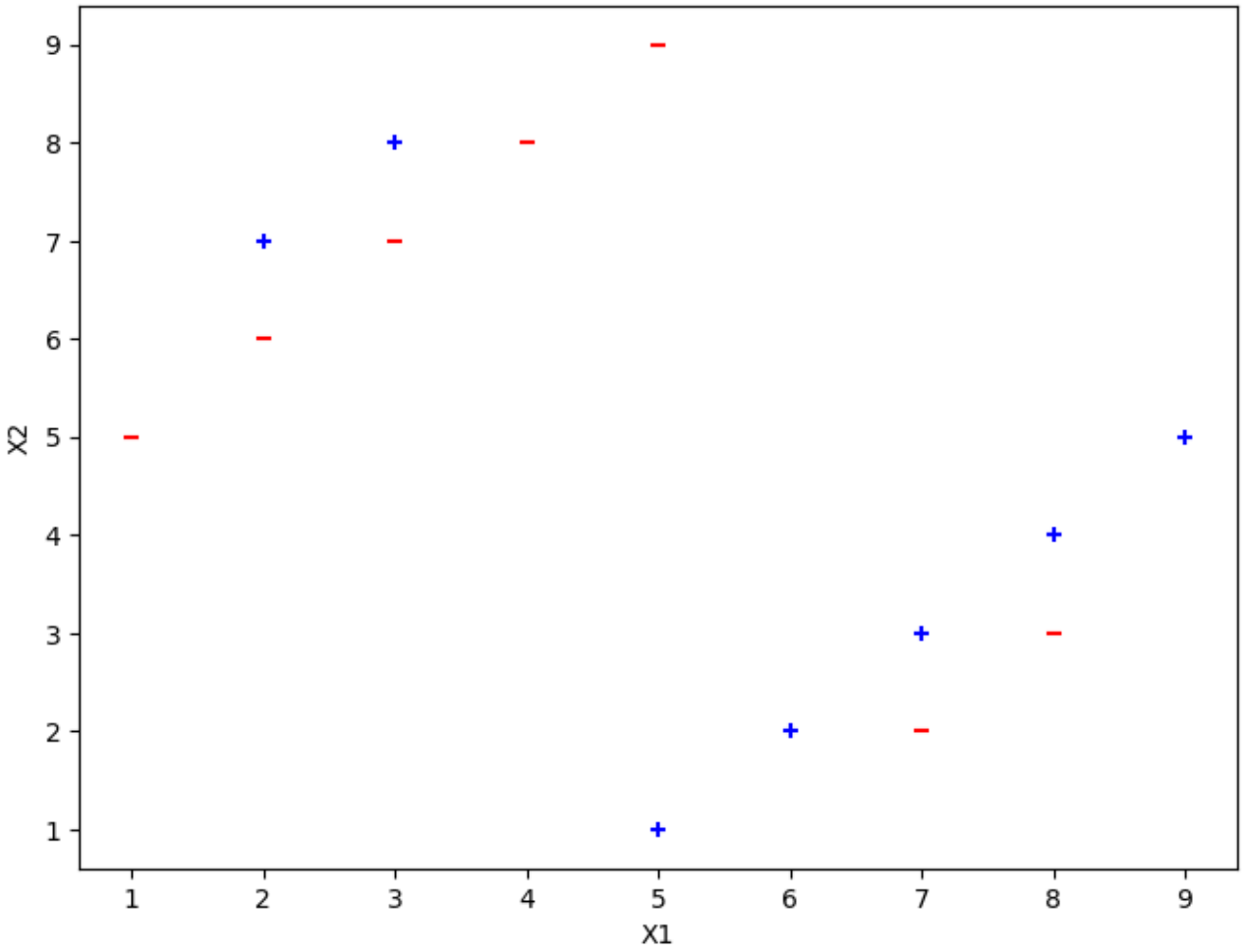
We just use the code we used in part (a) just shift the neighbour finding process by one unit.

The least error is 0.28 and it happens in k=5.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
```

```python
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def predict_label(point, points, labels, k):
    distances = []
    for i in range(len(points)):
        distance = euclidean_distance(point, points[i])
        distances.append((distance, labels[i]))

    distances.sort(key=lambda x: x[0])
    k_nearest_labels = [label for _, label in distances[1:k+1]]
    # print(k_nearest_labels, k, point)

    most_common = Counter(k_nearest_labels).most_common(1)
    return most_common[0][0]

def knn_errors(points, labels, k_values):
    errors = {}

    numeric_labels = np.array([1 if label == '+' else 0 for label in labels])

    for k in k_values:
        predictions = []
        for i in range(len(points)):
            predicted_label = predict_label(points[i], points, numeric_labels, k)
            predictions.append(predicted_label)

        errors[k] = np.mean(np.array(predictions) != numeric_labels)

    return errors

def plot_data(points, labels):
    plt.figure(figsize=(8, 6))
    for point, label in zip(points, labels):
        if label == '+':
            plt.scatter(point[0], point[1], c='blue', marker='+')
        else:
            plt.scatter(point[0], point[1], c='red', marker='_')

    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.grid(False)
    plt.show()

# Data
points = np.array([
    [1, 5], [2, 6], [2, 7], [3, 7], [3, 8], [4, 8],
    [5, 1], [5, 9], [6, 2], [7, 2], [7, 3], [8, 3], [8, 4], [9, 5]
])
labels = ['-', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '+']
k_values = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# k_values = [3]

errors = knn_errors(points, labels, k_values)
for err in errors:
    print("K= " + str(err) + ", error= " + str(errors[err]))

plot_data(points, labels)
```

```
K= 1, error= 0.7142857142857143
K= 2, error= 0.7142857142857143
K= 3, error= 0.42857142857142855
K= 4, error= 0.7142857142857143
K= 5, error= 0.2857142857142857
K= 6, error= 0.2857142857142857
K= 7, error= 0.2857142857142857
K= 8, error= 0.7142857142857143
K= 9, error= 1.0
```



Part (d)

We do it in 2 ways.

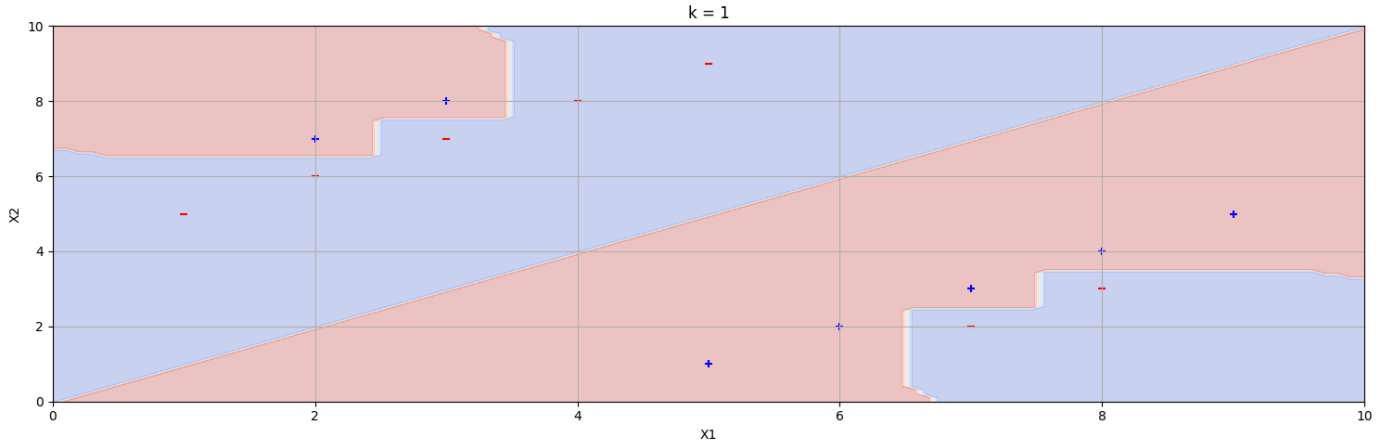
First we do it in the way we solved part (a) and consider each point as it's own nearest neighbour.

It is as below:

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def predict_label(point, points, labels, k):
    distances = []
    for i in range(len(points)):
        distance = euclidean_distance(point, points[i])
        distances.append((distance, labels[i]))

    distances.sort(key=lambda x: x[0])
    k_nearest_labels = [label for _, label in distances[:k]]

    most_common = Counter(k_nearest_labels).most_common(1)
    return most_common[0][0]

def knn_errors(points, labels, k_values):
    errors = {}

    numeric_labels = np.array([1 if label == '+' else 0 for label in labels])

    for k in k_values:
        predictions = []
        for i in range(len(points)):
            predicted_label = predict_label(points[i], points, numeric_labels, k)
```

```
            predictions.append(predicted_label)

        errors[k] = np.mean(np.array(predictions) != numeric_labels)

    return errors

def plot_data_and_boundaries(points, labels, k_values):
    x_min, x_max = points[:, 0].min() - 1, points[:, 0].max() + 1
    y_min, y_max = points[:, 1].min() - 1, points[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))

    fig, axes = plt.subplots(1, len(k_values), figsize=(15, 5))
    if len(k_values) == 1:
        axes = [axes]

    numeric_labels = np.array([1 if label == '+' else 0 for label in labels])

    for ax, k in zip(axes, k_values):
        Z = np.array([predict_label(np.array([x, y]), points, numeric_labels, k) for x, y in zip(xx
        Z = Z.reshape(xx.shape)

        ax.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')

        for point, label in zip(points, labels):
            if label == '+':
                ax.scatter(point[0], point[1], c='blue', marker='+')
            else:
                ax.scatter(point[0], point[1], c='red', marker='_')

        ax.set_title(f'k = {k}')
        ax.set_xlabel("X1")
        ax.set_ylabel("X2")
        ax.grid(True)

    plt.tight_layout()
    plt.show()

points = np.array([
    [1, 5], [2, 6], [2, 7], [3, 7], [3, 8], [4, 8],
    [5, 1], [5, 9], [6, 2], [7, 2], [7, 3], [8, 3], [8, 4], [9, 5]
])
labels = ['-', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '+']
k_values = [1]

errors = knn_errors(points, labels, k_values)
print("Error:", errors)

plot_data_and_boundaries(points, labels, k_values)
```

```
Errors for different k values: {1: 0.0}
```



Now we do it in the way we solved part (c) and leaving each point out of it's own nearest neighbours.

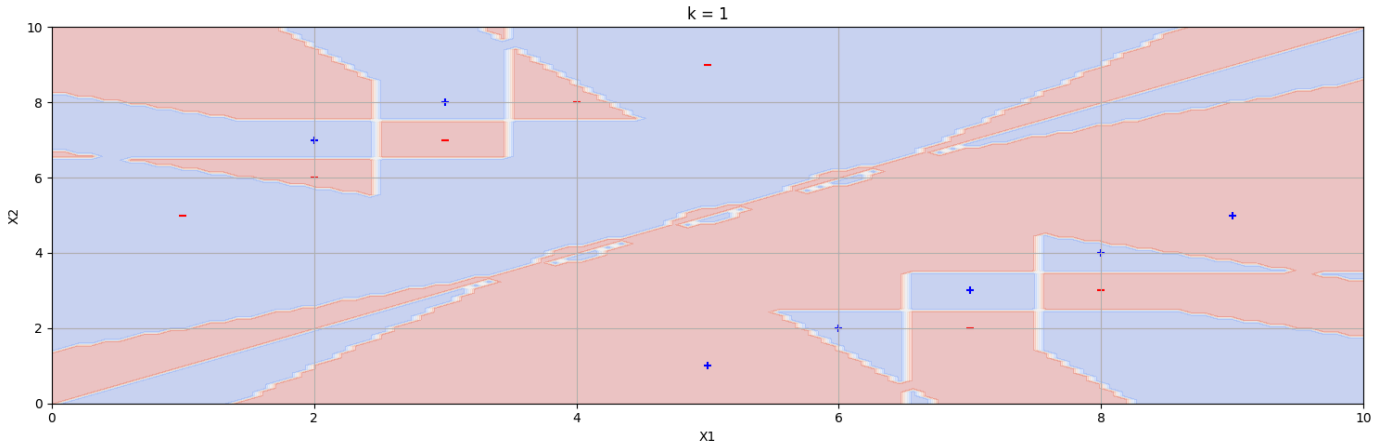
It is as below:

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def predict_label(point, points, labels, k):
    distances = []
    for i in range(len(points)):
        distance = euclidean_distance(point, points[i])
        distances.append((distance, labels[i]))

    distances.sort(key=lambda x: x[0])
    k_nearest_labels = [label for _, label in distances[1:k+1]]

    most_common = Counter(k_nearest_labels).most_common(1)
    return most_common[0][0]

def knn_errors(points, labels, k_values):
    errors = {}

    numeric_labels = np.array([1 if label == '+' else 0 for label in labels])

    for k in k_values:
        predictions = []
        for i in range(len(points)):
            predicted_label = predict_label(points[i], points, numeric_labels, k)
            predictions.append(predicted_label)

        errors[k] = np.mean(np.array(predictions) != numeric_labels)

    return errors

def plot_data_and_boundaries(points, labels, k_values):
    x_min, x_max = points[:, 0].min() - 1, points[:, 0].max() + 1
    y_min, y_max = points[:, 1].min() - 1, points[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))

    fig, axes = plt.subplots(1, len(k_values), figsize=(15, 5))
    if len(k_values) == 1:
        axes = [axes]

    numeric_labels = np.array([1 if label == '+' else 0 for label in labels])

    for ax, k in zip(axes, k_values):
        Z = np.array([predict_label(np.array([x, y]), points, numeric_labels, k) for x, y in zip(xx
        Z = Z.reshape(xx.shape)

        ax.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')

        for point, label in zip(points, labels):
            if label == '+':
                ax.scatter(point[0], point[1], c='blue', marker='+')
            else:
                ax.scatter(point[0], point[1], c='red', marker='_')

        ax.set_title(f'k = {k}')
        ax.set_xlabel("X1")
        ax.set_ylabel("X2")
        ax.grid(True)

    plt.tight_layout()
    plt.show()

points = np.array([
    [1, 5], [2, 6], [2, 7], [3, 7], [3, 8], [4, 8],
    [5, 1], [5, 9], [6, 2], [7, 2], [7, 3], [8, 3], [8, 4], [9, 5]
```

```
])
labels = ['-', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '+']
k_values = [1]

errors = knn_errors(points, labels, k_values)
print("Error:", errors)

plot_data_and_boundaries(points, labels, k_values)
```

```
Error: {1: 0.7142857142857143}
```



Question 6

Suppose T is a set of n training data with features $A_1, \ldots, A_a$ and k classes c_1 to c_k. Consider a particular feature A. Assume the set T is split according to this feature and a fully disjoint and complete partitioning is done on T. This partitioning is as $\{T_i^A | i \in \{1, \ldots, m_A\}\}$ where m_A is number of different values for feature A. Also, assume that the distribution of classes T is uniform and independent of the feature A. Calculate the following terms in terms of the parameter k (number of classes) and interpret your answers:

$$H(T), H(T|A), IG(T, A)$$

Answer

$H(T)$

The entropy $H(T)$ represents the uncertainty in the classification of the training data set T. Since the distribution of classes in T is uniform, each class c_i (for $i \in \{1, \ldots, k\}$) has an equal probability of occurring.

If the classes are uniformly distributed, the probability of each class c_i is:

$$P(c_i) = \frac{1}{k}$$

The entropy $H(T)$ is calculated using the formula for entropy:

$$H(T) = -\sum_{i=1}^{k} P(c_i) \log_2 P(c_i)$$

Substituting $P(c_i) = \frac{1}{k}$:

$$H(T) = -\sum_{i=1}^{k} \frac{1}{k} \log_2 \frac{1}{k}$$

$$= -\sum_{i=1}^{k} \frac{1}{k} (-\log_2 k)$$

$$= \log_2 k \sum_{i=1}^{k} \frac{1}{k}$$

$$= \log_2 k \cdot 1$$

$$= \log_2 k$$

So, the entropy $H(T)$ is:

$$H(T) = \log_2 k$$

$H(T|A)$

The conditional entropy $H(T|A)$ measures the average uncertainty remaining about the class labels after knowing the value of the feature A. Given that the set T is partitioned into subsets $\{T_i^A \mid i \in \{1, \ldots, m_A\}\}$ based on the values of the feature A, and assuming that the distribution of classes in T is uniform and independent of the feature A:

Since the class distribution is uniform and independent of A, the conditional entropy $H(T|A)$ is the same as the entropy $H(T)$ because knowing the feature A does not reduce any uncertainty about the classes. Therefore:

$$H(T|A) = H(T) = \log_2 k$$

$IG(T, A)$

The information gain $IG(T, A)$ measures the reduction in entropy of T due to the knowledge of the feature A . It is calculated as the difference between the entropy of T and the conditional entropy of T given A:

$$IG(T, A) = H(T) - H(T|A)$$

Substituting the values we calculated:

$$IG(T, A) = \log_2 k - \log_2 k$$

$$= 0$$

Interpretation of my answers

1. **Entropy $H(T)$:** The entropy $H(T) = \log_2 k$ indicates the amount of uncertainty or randomness in the class distribution of the training set T. With k classes uniformly distributed, the entropy represents the maximum uncertainty.

2. **Conditional Entropy $H(T|A)$:** The conditional entropy $H(T|A) = \log_2 k$ shows that knowing the feature A does not reduce the uncertainty about the classes. This is because the classes are uniformly distributed and independent of the feature A.

3. **Information Gain $IG(T, A)$:** The information gain $IG(T, A) = 0$ means that the feature A does not provide any useful information for predicting the class labels. This is expected since the class distribution is independent of the feature A, implying that A does not help in reducing the uncertainty about the class labels.

In summary, when the class distribution is uniform and independent of the feature, the feature does not contribute to reducing the uncertainty in class prediction, resulting in zero information gain.