Recitation 3
Classification and Regression

10-301/10-601: Introduction to Machine Learning

02/07/2025

1 Decision Trees and Beyond

1. **Decision Tree Classification with Continuous Attributes**
Given the dataset $\mathcal{D}_1 = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^2, y^{(i)} \in \{\text{Yellow}, \text{Purple}, \text{Green}\}$ as shown in Fig. 1, we wish to learn a decision tree for classifying such points. Provided with a possible tree structure in Fig. 1, what values of α, β and leaf node predictions could we use to perfectly classify the points? Now, draw the associated decision boundaries on the scatter plot.

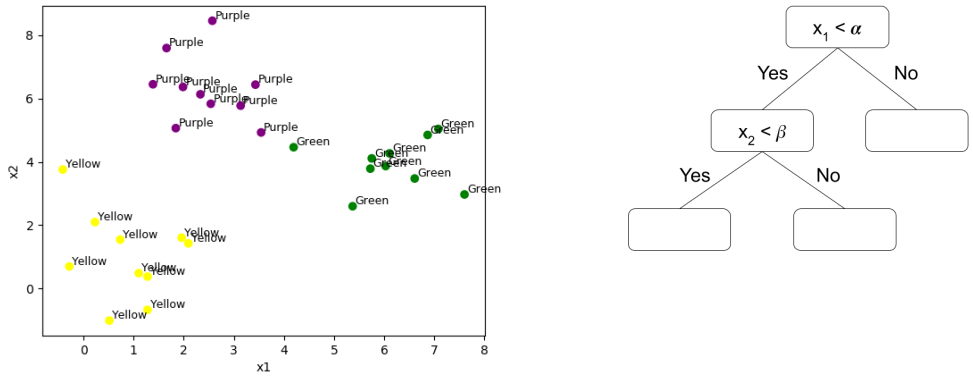
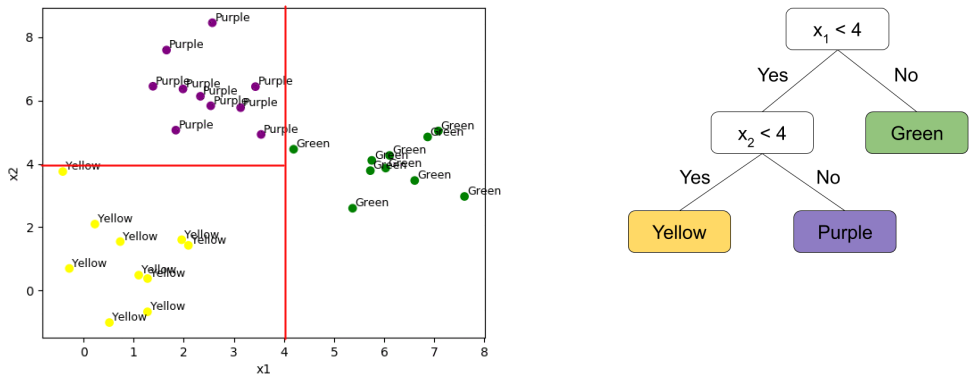


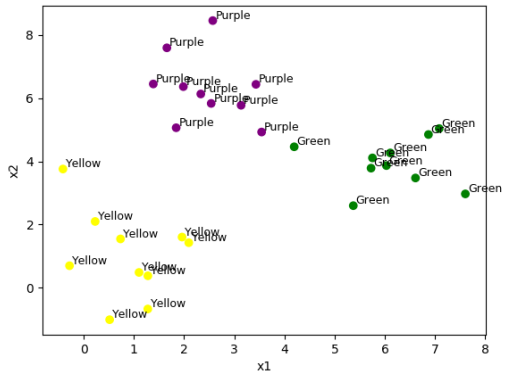
Figure 1: Classification of 2D points, with Decision Tree to fill in

Solution:



Note how our decision tree actually creates partitions in the 2D space of points, and each partition is associated with one predicted class. If we had trees of larger maximum depth, we gain the ability to create even more fine-grained partitions of the feature space, resulting in greater flexibility of predictions.

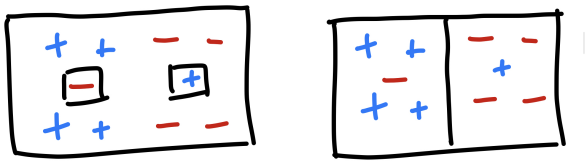
2. **Choosing a Tree:** What might happen if we increased the max-depth of the tree? When predicting on unseen data, would we prefer the depth-2 tree above or a very deep tree?



We would overfit to the training data by learning a complex decision boundary, and would rather prefer the depth-2 tree during inference.

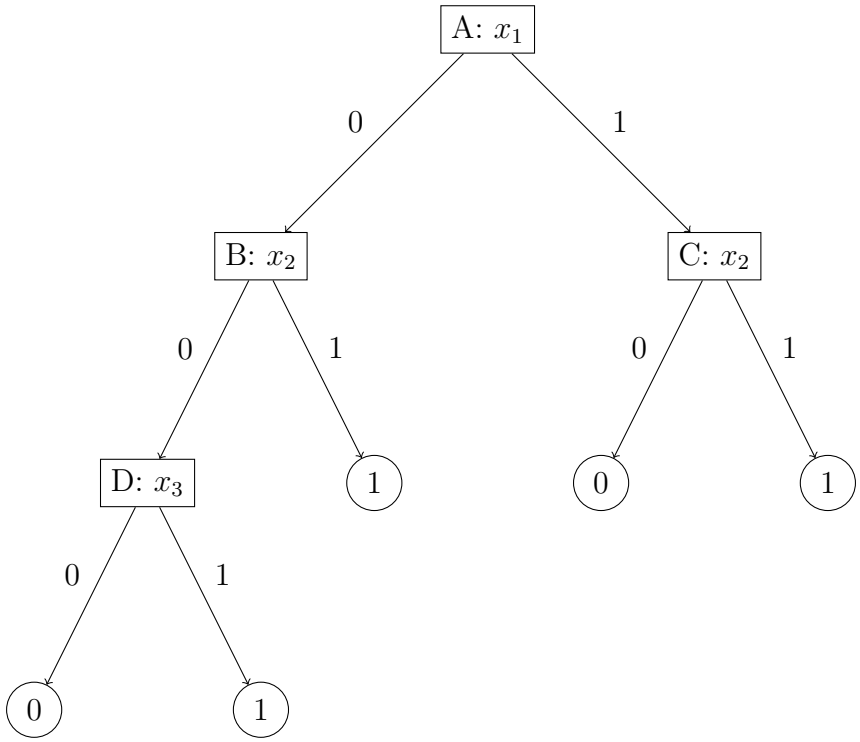
The smaller the depth of the tree, the fewer splits we make, which simplifies the decision boundary.

This question is getting at the inductive bias of a decision tree wherein we prefer trees with a smaller depth that work.



Consider the dataset above. The complex decision boundary on the left overfits the training data, while the simpler boundary on the right will probably generalize to test data better.

3. **Pruning a Tree:** Which node would be the first to be pruned in the following decision tree? In the case of a tie, break the tie in favor of the alphabetically earlier node (eg. prune node B before D)



The following is the validation set, along with additional columns for us to use while solving. For simplicity, suppose that a pruned node is replaced by a prediction of 1.

x_1	x_2	x_3	Label	No prune	Prune A	Prune B	Prune C	Prune D
0	0	0	1					
0	0	1	0					
0	1	0	1					
1	0	1	0					
1	1	0	1					

x_1	x_2	x_3	Label	No prune	Prune A	Prune B	Prune C	Prune D
0	0	0	1	0	1	1	0	1
0	0	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1
1	0	1	0	0	1	0	1	0
1	1	0	1	1	1	1	1	1

Resulting error when pruning:
No prune: 40%
Prune A: 40%
Prune B: 20%
Prune C: 60%

Prune D: 20%

Since B is tied with D for the best node, we will prune node B.

Follow-up question: How do we know when we are done pruning a decision tree?

We know we are done when pruning any of the remaining nodes does not improve our validation error.

2 Perceptron

2.1 Perceptron Mistake Bound Guarantee

If a dataset has margin γ and all points inside a ball of radius R, then the perceptron makes less than or equal to $(R/\gamma)^2$ mistakes.

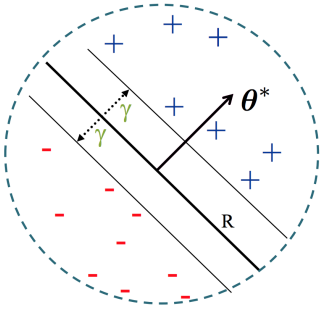


Figure 2: Perceptron Mistake Bound Setup

2.2 Definitions

Margin:

- The margin of example x wrt a linear separator w is the (absolute) distance from x to the plane $w \cdot x = 0$.
- The margin γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.
- The margin γ of a set of examples S is the maximum γ_w over all linear separators w.

Linear Separability: For a binary classification problem, a set of examples S is linearly separable if there exists a linear decision boundary that can separate the points.

Update Rule: When the k-th mistake is made on data point $\mathbf{x}^{(i)}$, the parameter update is

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \mathbf{y}^{(i)}\mathbf{x}^{(i)}$$

We say the (batch) perceptron algorithm has *converged* when it stops making mistakes on the training data.

2.3 Perceptron Mistake Bound: Example

Given dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, suppose:

1. Finite size inputs: $||x^{(i)}|| \leq R$
2. Linearly separable data: $\exists \boldsymbol{\theta}^*$ and $\boldsymbol{\gamma} > 0$ s.t. $||\boldsymbol{\theta}^*|| = 1$ and $y^{(i)}(\boldsymbol{\theta}^* \cdot x^{(i)}) \geq \boldsymbol{\gamma}, \forall i$

Then, the number of mistakes k made by the perceptron algorithm on $\mathcal{D}$ is bounded by $(R/\boldsymbol{\gamma})^2$.

The following table shows a dataset of linearly separable datapoints.

x1	x2	y
1	-1	1
0	2	-1
4	0	1

Assuming that the linear separator with the largest margin is given by:

$$\theta^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0, \text{ where } \theta = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Calculate the theoretical mistake bound for the perceptron.

The radius will be the distance of the point furthest from the origin i.e (4, 0). So, the radius, r will be $\sqrt{16} = 4$

Since the linear separator is already provided, the margin, γ, will the distance of the point closest to the separator which is (1,-1). So,$\gamma = min_{x^{(i)}} \frac{|\theta^T x^{(i)}|}{||\theta||} = \frac{|(-1)*1+(1)*(-1)|}{\sqrt{(-1)*(-1)+1*1}} = 2/\sqrt{2} = \sqrt{2}$

The mistake bound $= (\frac{4}{\sqrt{2}})^2 = 8$

2.4 Theorem: Block, Novikoff

Given dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, suppose:

1. Finite size inputs: $||x^{(i)}|| \leq R$
2. Linearly separable data: $\exists \boldsymbol{\theta}^*$ and $\boldsymbol{\gamma} > 0$ s.t. $||\boldsymbol{\theta}^*|| = 1$ and $y^{(i)}(\boldsymbol{\theta}^* \cdot x^{(i)}) \geq \boldsymbol{\gamma}, \forall i$

Then, the number of mistakes k made by the perceptron algorithm on $\mathcal{D}$ is bounded by $(R/\boldsymbol{\gamma})^2$.

Proof:
Part 1: For some A, $Ak \leq ||\boldsymbol{\theta}^{(k+1)}||$

$$\begin{aligned}
\boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* &= (\boldsymbol{\theta}^{(k)} + y^{(i)}x^{(i)}) \cdot \boldsymbol{\theta}^*, \text{ Perceptron algorithm update} \\
&= \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + y^{(i)}(\boldsymbol{\theta}^* \cdot x^{(i)})) \\
&\geq \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + \boldsymbol{\gamma}, \text{ by assumption} \\
\implies \boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* &\geq k\boldsymbol{\gamma}, \text{ by induction on k since } \boldsymbol{\theta}^{(1)} = 0 \\
\implies ||\boldsymbol{\theta}^{(k+1)}|| &\geq k\boldsymbol{\gamma}, \text{ since } ||w|| \times ||u|| \geq w \cdot u \text{ and } ||\boldsymbol{\theta}^*|| = 1
\end{aligned}$$

Part 2: For some B, $||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$

$$\begin{aligned}
||\boldsymbol{\theta}^{(k+1)}||^2 &= ||\boldsymbol{\theta}^{(k)} + y^{(i)}x^{(i)}||^2, \text{ Perceptron algorithm update} \\
&= ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||x^{(i)}||^2 + 2y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot x^{(i)}) \\
&\leq ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||x^{(i)}||^2, \text{ since } k^{th} \text{ mistake } \implies y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot x^{(i)}) \leq 0 \\
&= ||\boldsymbol{\theta}^{(k)}||^2 + R^2, \text{ since } (y^{(i)})^2||x^{(i)}||^2 = ||x^{(i)}||^2 \leq R^2, \text{ by assumption and } (y^{(i)})^2 = 1 \\
\implies ||\boldsymbol{\theta}^{(k+1)}||^2 &\leq kR^2, \text{ by induction on k since } (\theta^{(i)})^2 = 0 \\
\implies ||\boldsymbol{\theta}^{(k+1)}|| &\leq \sqrt{k}R
\end{aligned}$$

Part 3: Combine the bounds

$$\begin{aligned}
k\boldsymbol{\gamma} \leq ||\boldsymbol{\theta}^{(k+1)}|| &\leq \sqrt{k}R \\
\implies k &\leq (R/\boldsymbol{\gamma})^2
\end{aligned}$$

- Perceptron will not converge.
- However, we can achieve a similar bound on the number of mistakes made in one pass (Freund, Schapire)

Main Takeaway: For linearly separable data, if the perceptron algorithm repeatedly cycles through the data, it will converge in a finite number of steps.

3 k-NN

3.1 A Classification Example

Using the figure below, what would you categorize the green circle as with $k = 3$? $k = 5$? $k = 4$?

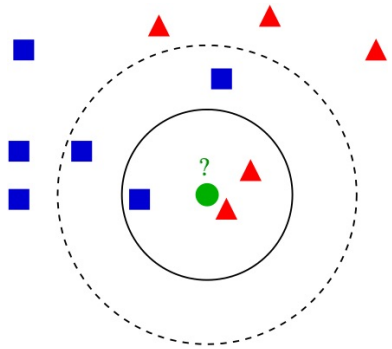


Figure 3: An example of k-NN on a small dataset; image source from Wikipedia

Example of k-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.

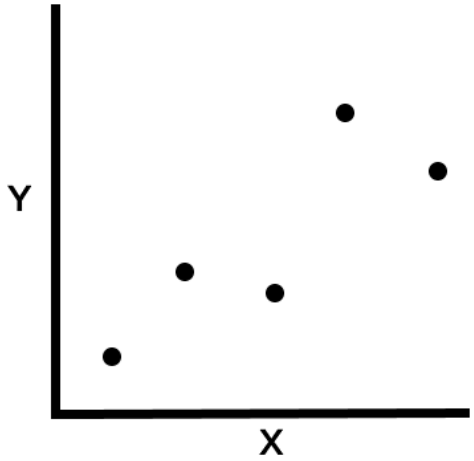
If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.

If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

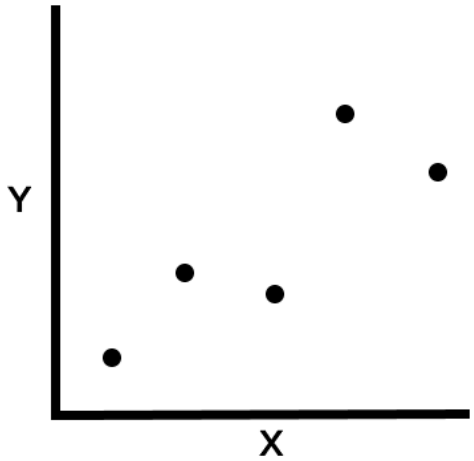
If k = 4, since there is a tie (2 squares vs. 2 triangles), we can arbitrarily break the tie by randomly choosing one of the possible (squares, triangle) classes. Whenever there is a similar tie for k-NNs, other ways to break the include choosing one more nearest sample point, or one less, or taking a weighted vote of the neighbors, etc.

3.2 k-NN for Regression

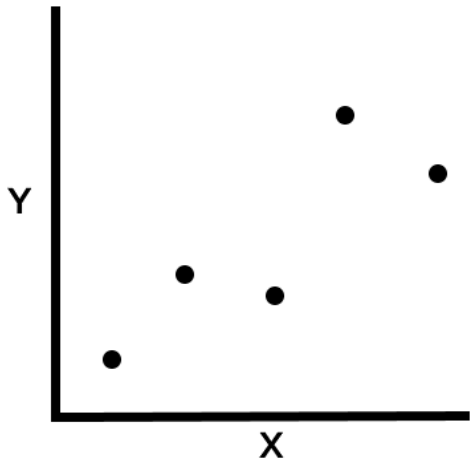
You want to predict a continuous variable Y with a continuous variable X. Having just learned k-NN, you are super eager to try it out for regression. Given the data below, draw the regression lines (what k-NN would predict Y to be for every X value if it was trained for the given data) for k-NN regression with $k = 1$, weighted $k = 2$, and unweighted $k = 2$. For weighted $k = 2$, take the weighted average of the two nearest points. For unweighted $k = 2$, take the unweighted average of the two nearest points. (Note: the points are equidistant along the x-axis)



(a) $k = 1$

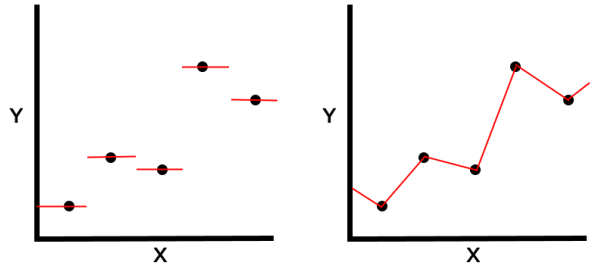


(b) weighted $k = 2$



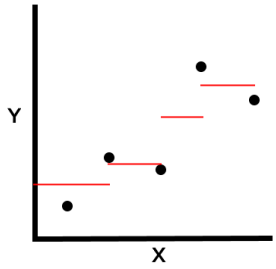
(c) unweighted $k = 2$

SOLUTION:



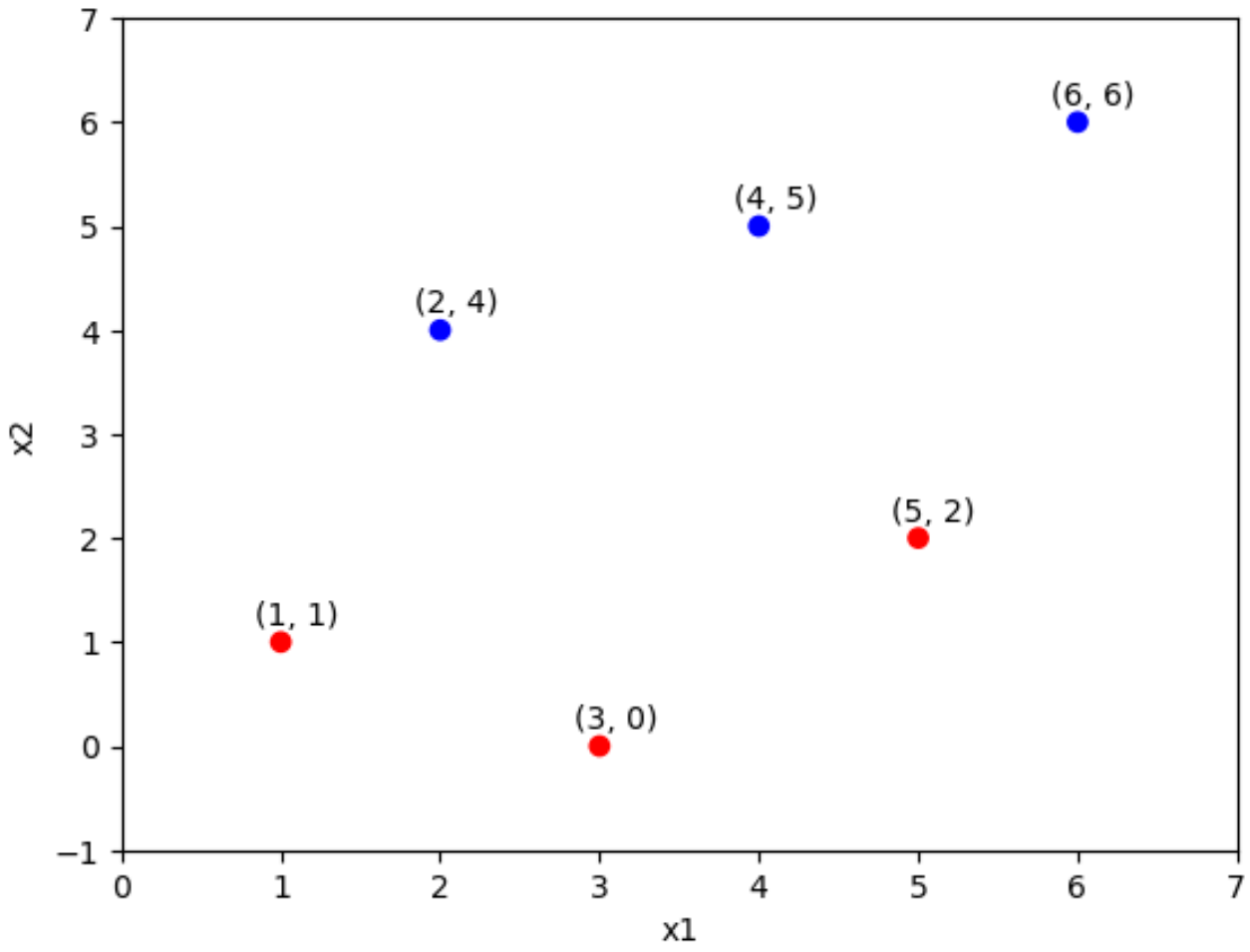
(a) $k = 1$ (b) weighted $k = 2$

For $X < 1$, the regression line tends to the average of the first 2 points.
Similarly, for $X > 5$, the regression line tends to the average of the last 2 points.

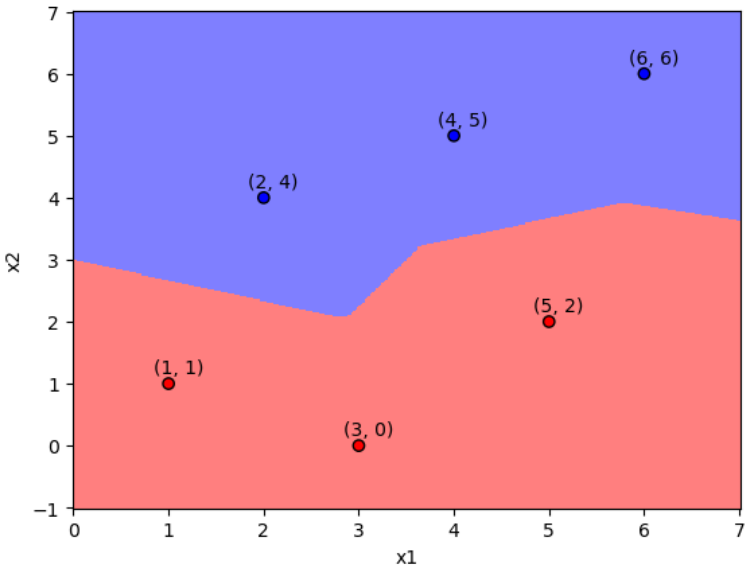


(c) unweighted $k = 2$

3.3 k-NN Decision Boundary and Cross Validation



Draw the decision boundaries for the above training dataset given using kNN algorithm considering *k=1*.

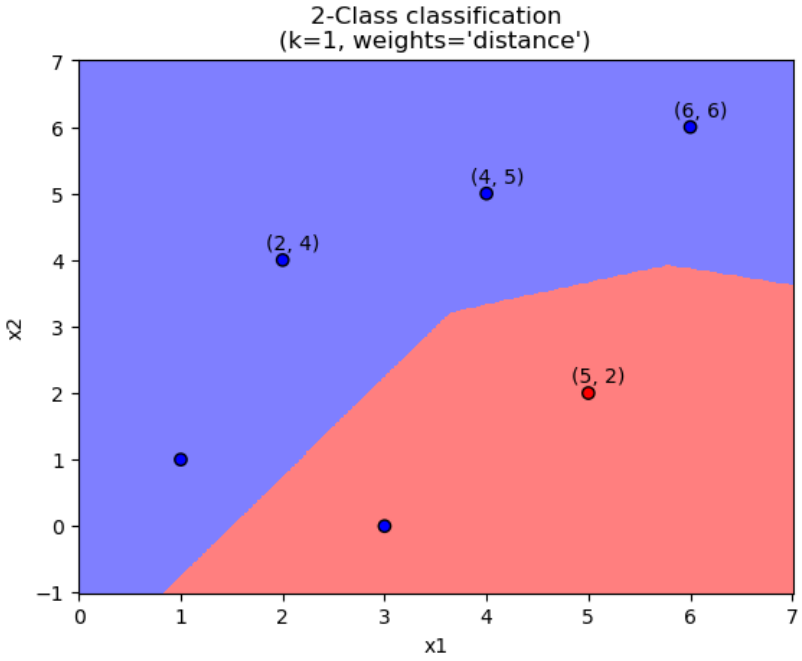


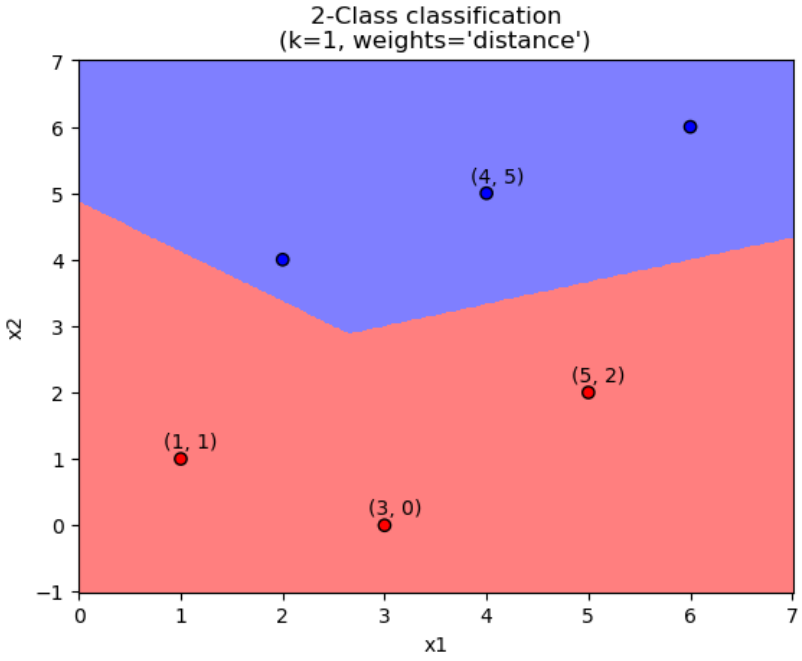
SOLUTION:

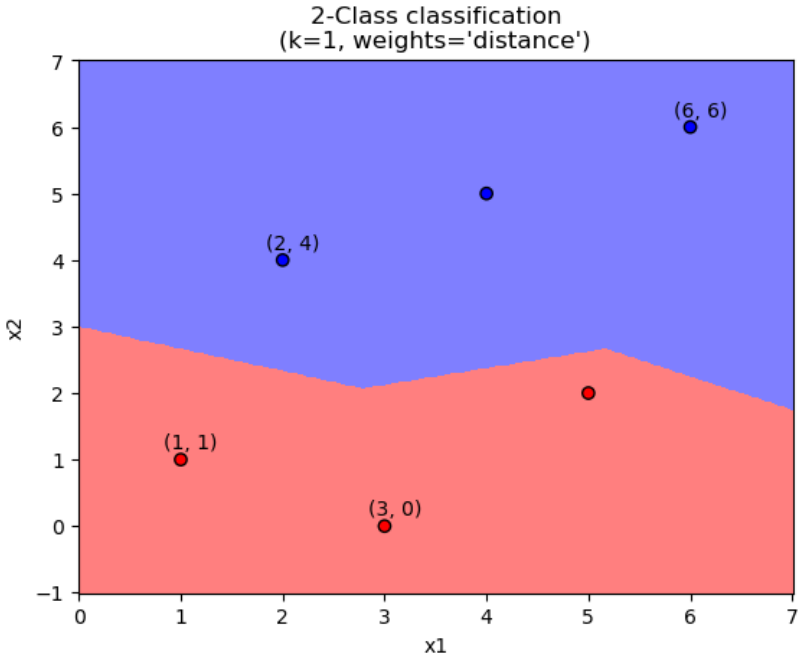
Suppose we use 3-fold Cross Validation for this kNN, with k=1. The folds are [(1,1),(3,0)], [(2,4),(6,6)], and [(4,5),(5,2)] What is the cross-validation error?

SOLUTION: The Cross validation error is $\frac{1}{k}\sum_{i=1}^{k} CV_i$, where CV_i is the error of the i-th fold, and k is the number of folds. Our measure of error is 1 - accuracy, but in general you can use any error metric you want. For each fold, we have to train the model, and test it on the held out fold. Since there are 3-folds, we train it three times and evaluate it. Below are the decision boundaries of the three trained models.

In the first graph, one of the unlabeled points are classified incorrectly, and in the other two graphs, both unlabeled points are classified correctly, so the cross validation error is $\frac{1}{3}(0.5 + 0 + 0) = \frac{1}{6}$ (Note: The color of the unlabeled points should be red not blue, this is a typo)


2-Class classification
(k=1, weights='distance')
(6, 6)
(4, 5)
(2, 4)
(5, 2)
x1
x2


2-Class classification
(k=1, weights='distance')
(4, 5)
(5, 2)
(1, 1)
(3, 0)
x1
x2


2-Class classification
(k=1, weights='distance')
(6, 6)
(2, 4)
(1, 1)
(3, 0)
x1
x2

4 Linear Regression

4.1 Defining the Objective Function

1. What does an objective function $J(\theta)$ do?

   A function to measure how "good" the linear model is

2. What are some examples?

   - Mean Squared Error $\frac{1}{N}\sum_{i=1}^{N} e_i^2$
   - Mean Absolute Error: $\frac{1}{N}\sum_{i=1}^{N} |e_i|$

3. What are some desirable properties of this function?

   - Should be differentiable
   - Preferably convex

4.2 Solving Linear Regression using Gradient Descent

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$
x_1	1.0	2.0	3.0	4.0	5.0
x_2	-2.0	-5.0	-6.0	-8.0	-11.0
y	2.0	4.0	7.0	8.0	11.0

Now, we want to implement the gradient descent method.

Assuming that $\gamma = 0.1$ and $\boldsymbol{\theta}$ has been initialized to $[0, 0, 0]^T$, perform one iteration of gradient descent:

1. What is the gradient of the objective function $J(\theta)$ with respect to θ: $\nabla_\theta J(\theta)$?

$$\frac{dJ(\theta)}{d\theta_k} = \frac{1}{5}\sum_{i=1}^{5} -2x_k^{(i)}(y^{(i)} - \sum_{j=0}^{2}\theta_j x_j^{(i)})$$

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{dJ(\theta)}{d\theta_0} \\ \frac{dJ(\theta)}{d\theta_1} \\ \frac{dJ(\theta)}{d\theta_2} \end{pmatrix} = \begin{pmatrix} \frac{1}{5}\sum_{i=1}^{5} -2x_0^{(i)}(y^{(i)} - \sum_{j=0}^{2}\theta_j x_j^{(i)}) \\ \frac{1}{5}\sum_{i=1}^{5} -2x_1^{(i)}(y^{(i)} - \sum_{j=0}^{2}\theta_j x_j^{(i)}) \\ \frac{1}{5}\sum_{i=1}^{5} -2x_2^{(i)}(y^{(i)} - \sum_{j=0}^{2}\theta_j x_j^{(i)}) \end{pmatrix}$$

2. How do we carry out the update rule?

   We initialize:

$$\theta = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

   Follow the update rule:

$$\theta^{(k+1)} = \theta^{(k)} - \gamma \nabla_{\theta|\theta=\theta^{(k)}} J(\theta)$$

, where $k = 0$ here

$$\frac{1}{5}\sum_{i=1}^{5} -2x_0^{(i)}(y^{(i)} - \sum_{j=0}^{2}\theta_j x_j^{(i)}) = \frac{-2}{5}\cdot(2+4+7+8+11)$$
$$= -12.8$$
$$\frac{1}{5}\sum_{i=1}^{5} -2x_1^{(i)}(y^{(i)} - \sum_{j=0}^{2}\theta_j x_j^{(i)}) = \frac{-2}{5}\cdot(2+8+21+32+55)$$
$$= -47.2$$
$$\frac{1}{5}\sum_{i=1}^{5} -2x_2^{(i)}(y^{(i)} - \sum_{j=0}^{2}\theta_j x_j^{(i)}) = \frac{-2}{5}\cdot(-4-20-42-64-121)$$
$$= 100.4$$

$$\therefore \theta^{(1)} = \theta^{(0)} - \gamma\nabla_{\theta|\theta=\theta^{(0)}}J(\theta)$$
$$= \begin{pmatrix}0\\0\\0\end{pmatrix} - 0.1\begin{pmatrix}-12.8\\-47.2\\100.4\end{pmatrix}$$
$$= \begin{pmatrix}1.28\\4.72\\-10.04\end{pmatrix}$$

3. How could we pick which value of γ to use if we weren't given the step size?

Cross-validation or use a hold-out validation dataset

5 Summary

5.1 Decision Tree

Pros	Cons	Inductive bias	When to use
• Easy to understand and interpret • Very fast for inference	• Tree may grow very large and tend to overfit. • Greedy behaviour may be sub-optimal	• Prefer the smallest tree consistent w/ the training data (i.e. 0 error rate)	• Most cases. Random forests are widely used in industry.

5.2 k-NN

Pros	Cons	Inductive bias	When to use
• No training of parameters • Can apply to multi-class problems and use different metrics	• Slow for large datasets • Must select good k • Imbalanced data and outliers can lead to misleading results	• Similar (i.e. nearby) points should have similar labels • All label dimensions are created equal	• Small dataset • Small dimensionality • Data is clean (no missing data) • Inductive bias is strong for dataset

5.3 Linear regression

Pros	Cons	Inductive bias	When to use
• Easy to understand and train • Closed form solution	• Sensitive to noise (other than zero-mean Gaussian noise)	• The true relationship between the inputs and output is linear.	• Most cases (can be extended by adding non-linear feature transformations)

5.4 Perceptron

Pros	Cons	Inductive bias	When to use
• Easy to understand and works for online learning. • Provable guarantees on mistakes made for linearly separable data.	• No guarantees on finding best (maximum-margin) hyperplane. • Output is sensitive to noise in the training data.	• The binary classes are separable in the feature space by a line.	• Not used much anymore, but variants (kernel perceptron, structured perceptron) may have more success.