# Recitation 3
# Classification and Regression

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

02/09/2022

## 1 Decision Trees and Beyond

1. **Decision Tree Classification with Continuous Attributes**
Given the dataset $\mathcal{D}_1 = \{\mathbf{x}^{(i)}, y\}_{i=1}^{N}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^2, y \in \{\text{Yellow}, \text{Purple}, \text{Green}\}$ 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 $\alpha, \beta$ and leaf node predictions could we use to perfectly classify the points? Now, draw the associated decision boundaries on the scatter plot.

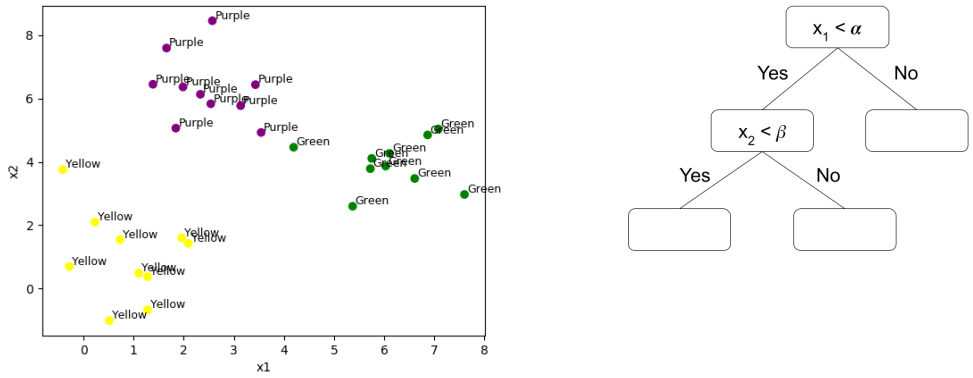
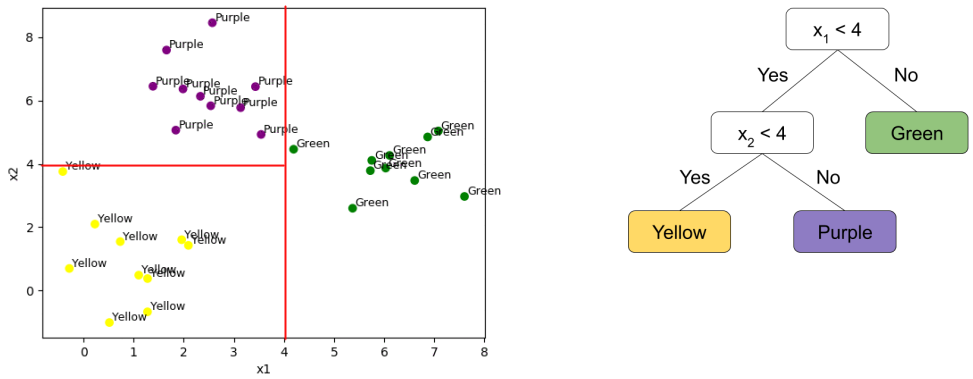



Figure 1: Classification of 2D points, with Decision Tree to fill in

**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.

**Decision Tree Regression with Continuous Attributes**

Now instead if we had dataset $\mathcal{D}_2 = \{\mathbf{x}^{(i)}, y\}_{i=1}^{N}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^2, y \in \mathbb{R}$ as shown in Fig. 2, we wish to learn a decision tree for regression on such points. Using the same tree structure and values of $\alpha, \beta$ as before, what values should each leaf node predict to minimize the training Mean Squared Error (MSE) of our regression? Assume each leaf node just predicts a constant.

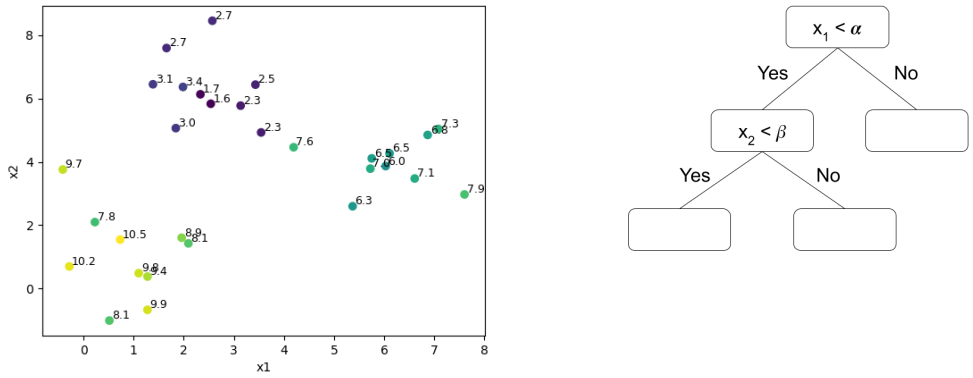
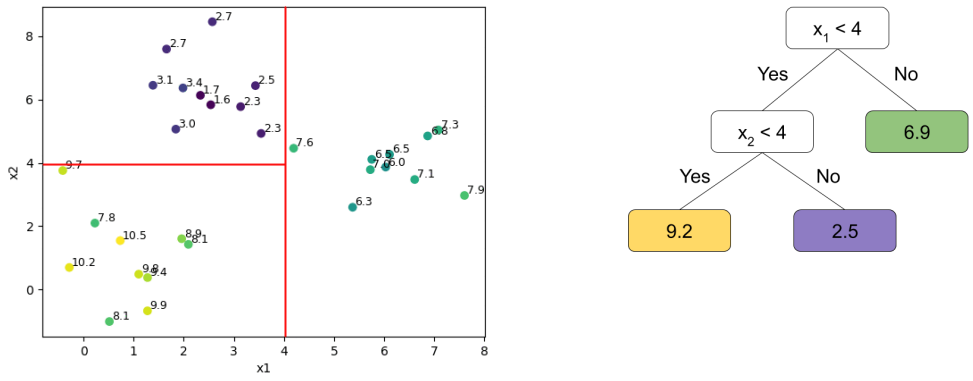



Figure 2: Regression on 2D points, with Decision Tree to fill in

**Solution:**




In this example we see how decision trees can be used for regressions too. Since we already know the splits, we partition up the feature space in the same way as before where each partition yields a single constant as a prediction. Instead of predicting a class, we want to predict a real number for each partition that will minimize our metric, MSE. The mean value of $y$ in each partition will be the prediction that minimizes MSE.

# 2 $k$-NN

## 2.1 A Classification Example

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

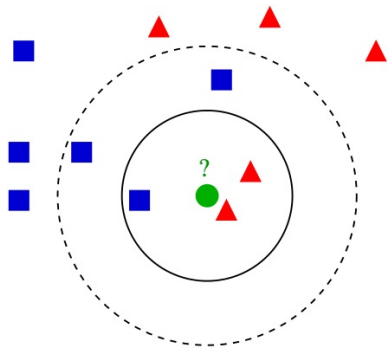



Figure 3: From wiki

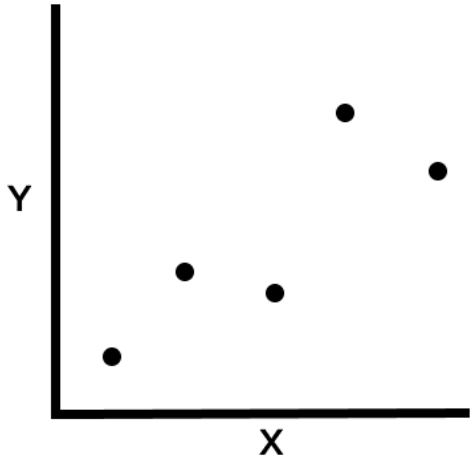
Example of $k$-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.

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.

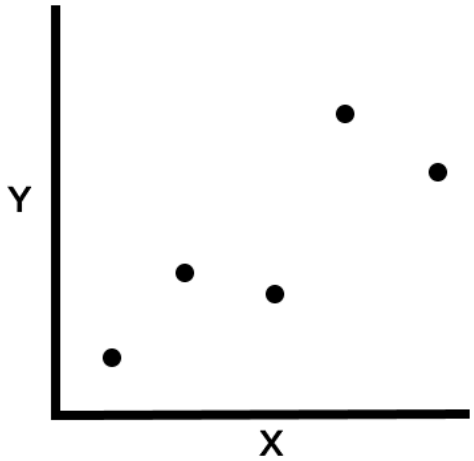
If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

## 2.2 $k$-NN for Regression

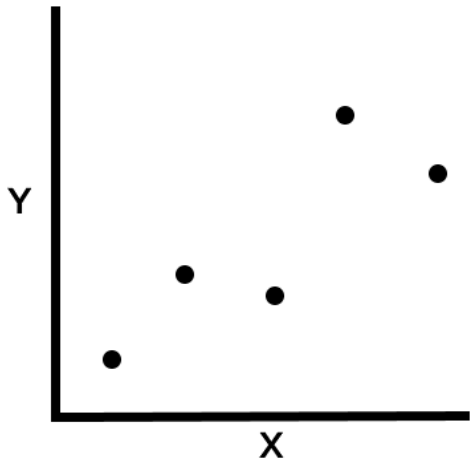
You want to predict a continuous variable $Y$ with a continuous variable $X$. Having just learned $k$-NN, you are super eager to try it out for regression. Given the data below, draw the regression lines (what $k$-NN would predict $Y$ to be for every $X$ value if it was trained for the given data) for $k$-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) $k = 1$

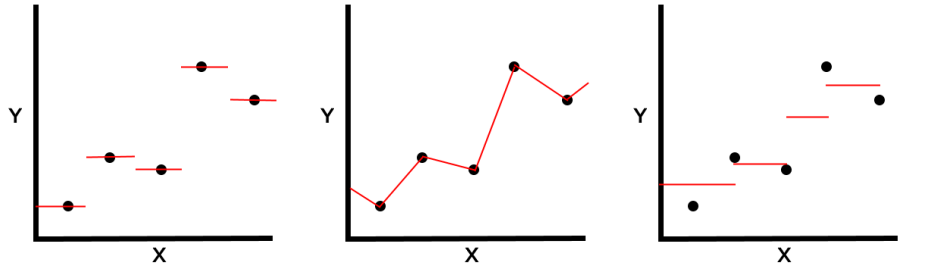



(b) weighted $k = 2$




(c) unweighted $k = 2$

SOLUTION:




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

# 3 Linear Regression

## 3.1 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 properties of this function?
   - Should be differentiable
   - Preferably convex

3. What are some examples?
   - Mean Squared Error $\frac{1}{N}\sum_{i=1}^{N} e_i^2$
   - Mean Absolute Error: $\frac{1}{N}\sum_{i=1}^{N} |e_i|$

## 3.2 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 |
| $x_3$ | 3.0 | 8.0 | 9.0 | 12.0 | 14.0 |
| $y$ | 2.0 | 4.0 | 7.0 | 8.0 | 11.0 |

Now, we want to implement the gradient descent method.

**Assuming that $\alpha = 0.1$ and $\mathbf{w}$ has been initialized to $[0, 0, 0]^T$, perform one iteration of gradient descent:**

1. What is the gradient of the objective function , $J(\theta)$, w.r.t $\theta$: $\nabla_\theta J(\theta)$

$$\frac{dJ(\theta)}{d\theta_k} = \frac{1}{5}\sum_{i=1}^{5} -2x_k^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)})$$

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{dJ(\theta)}{d\theta_0} \\ \frac{dJ(\theta)}{d\theta_1} \\ \frac{dJ(\theta)}{d\theta_2} \\ \frac{dJ(\theta)}{d\theta_3} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{5}\sum_{i=1}^{5} -2x_0^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) \\ \frac{1}{5}\sum_{i=1}^{5} -2x_1^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) \\ \frac{1}{5}\sum_{i=1}^{5} -2x_2^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) \\ \frac{1}{5}\sum_{i=1}^{5} -2x_3^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) \end{pmatrix}$$

2. How do we carry out the update rule?

   We initialize:

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

   Follow the update rule:

$$\theta^{(k+1)} = \theta^{(k)} - \underbrace{\alpha}_{\text{“Cross-validated”}} \nabla_{\theta|\theta=\theta^{(k)}} J(\theta)$$

   , where $k = 0$ here

$$\frac{1}{5}\sum_{i=1}^{5} -2x_0^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) = \frac{-2}{5}\cdot(2+4+7+8+11)$$
$$= -12.8$$
$$\frac{1}{5}\sum_{i=1}^{5} -2x_1^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) = \frac{-2}{5}\cdot(2+8+21+32+55)$$
$$= -47.2$$
$$\frac{1}{5}\sum_{i=1}^{5} -2x_2^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) = \frac{-2}{5}\cdot(-4-20-42-64-121)$$
$$= 100.4$$
$$\frac{1}{5}\sum_{i=1}^{5} -2x_3^{(i)}(y^{(i)} - \sum_{j=0}^{3}\theta_j x_j^{(i)}) = \frac{-2}{5}\cdot(6+32+63+96+154)$$
$$= -140.4$$

$$\therefore \theta^{(1)} = \theta^{(0)} - \alpha\nabla_{\theta|\theta=\theta^{(0)}}J(\theta)$$
$$= \begin{pmatrix}0\\0\\0\\0\end{pmatrix} - 0.1\begin{pmatrix}-12.8\\-47.2\\100.4\\-140.4\end{pmatrix}$$
$$= \begin{pmatrix}1.28\\4.72\\-10.4\\14.4\end{pmatrix}$$

***Convexity of objective function ensures that the local min(max) of the function is the global min(max).**

# 4 Perceptron

## 4.1 Perceptron Mistake Bound Guarantee

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

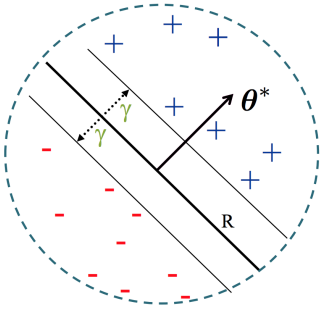



Figure 6: Perceptron Mistake Bound Setup

## 4.2 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 $\gamma_w$ of a set of examples $S$ wrt a linear separator $w$ is the smallest margin over points $x \in S$.
- The margin $\gamma$ of a set of examples $S$ is the maximum $\gamma_w$ 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.

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

## 4.3 Theorem: Block, Novikoff

Given dataset $D = (x^{(i)}, y^{(i)})_{i=1}^{N}$. Suppose:

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

Then, the number of mistakes made by the Perceptron algorithm on this dataset is $k \leq (R/\gamma)^2$

**Proof:**
Part 1: For some $A$, $Ak \leq ||\boldsymbol{\theta}^*||$

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

Part 2: For some $B$, $||\boldsymbol{\theta}^*|| \leq B\sqrt{k}$

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

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.

# 5 Summary

## 5.1 $k$-NN

| Pros | Cons | Inductive bias | When to use |
|---|---|---|---|
| • Simple, minimal assumptions made about data distribution<br>• No training of parameters<br>• Can apply to multi-class problems and use different metrics | • Becomes slow as dataset grows<br>• Requires homogeneous features<br>• Selection of $k$ is tricky<br>• Imbalanced data can lead to misleading results<br>• Sensitive to outliers | • Similar (i.e. nearby) points should have similar labels<br>• All label dimensions are created equal | • Small dataset<br>• Small dimensionality<br>• Data is clean (no missing data)<br>• Inductive bias is strong for dataset |

## 5.2 Linear regression

| Pros | Cons | Inductive bias | When to use |
|---|---|---|---|
| • Easy to understand and train<br>• Closed form solution | • Sensitive to noise (other than zero-mean Gaussian noise) | • The relationship between the inputs x and output y is linear. i.e. hypothesis space is Linear Functions | • Most cases (can be extended by adding non-linear feature transformations) |

## 5.3 Decision Tree

| Pros | Cons | Inductive bias | When to use |
|---|---|---|---|
| • Easy to understand and interpret<br>• Very fast for inference | • Tree may grow very large and tend to overfit.<br>• 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. |

## 5.4 Perceptron

| Pros | Cons | Inductive bias | When to use |
|---|---|---|---|
| • Easy to understand and works in an online learning setting.<br>• Provable guarantees on mistakes made if the data is known to be linearly separable (perceptron mistake-bound). | • No guarantees on finding maximum-margin hyperplane (like in SVM), only that you will find **a** separating hyperplane.<br>• Output is sensitive to noise in the training data. | • The binary classes are separable in the feature space by a line. | • The basic perceptron algorithm is not used much anymore, but other variants mentioned in class such as kernel perceptron or structured perceptron may have more success. |