Lecture 2: Linear Regression and Decision Tree

Instructor: Prof. Shuai Huang

Industrial and Systems Engineering

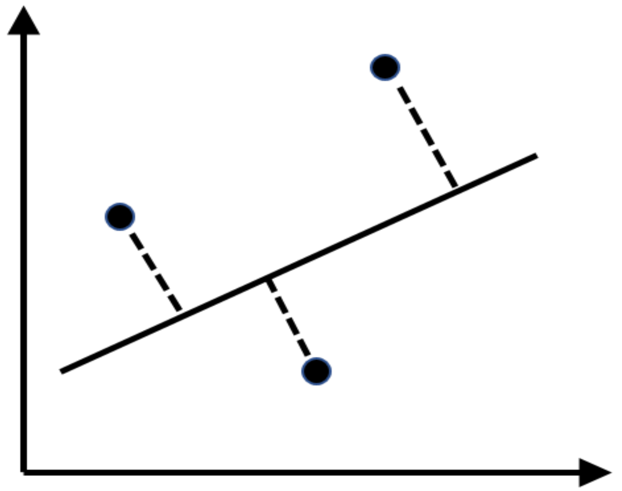
University of Washington

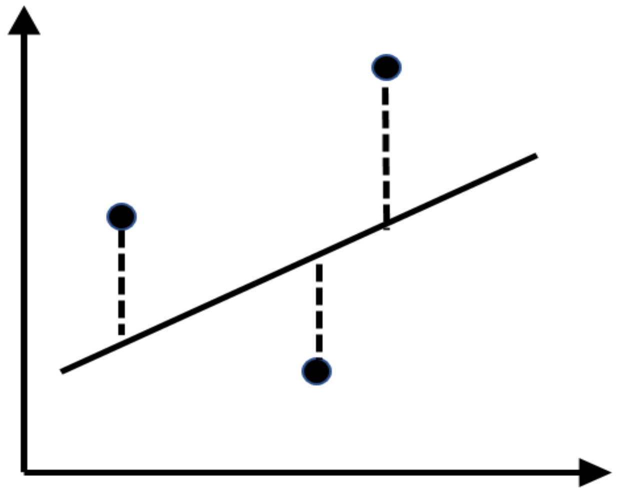
The linear regression model

A simple example: $f(x) = \beta_0 + \beta_1 x, \epsilon \sim N\left(0, \sigma_\varepsilon^2\right)$

- The linear relationship remains the same for all the values of x (a global pattern).
- The model suggests a fundamental unpredictability of y, if y is generated by a combination of the signal (the $f(x)$) and the noise (ϵ).
- ***R-squared:*** $\frac{\sigma_y^2 - \sigma_\varepsilon^2}{\sigma_y^2}$.
- The noise is usually modeled as gaussian distribution, but this assumption could be relaxed.

Parameter estimation





Two principles to fit a linear regression model: (left) perpendicular offsets; (right) vertical offsets.

In the historic development of linear regression, the paradigm of vertical offsets gained popularity which led to the least-squares estimation

Derivation of the least-squares estimation

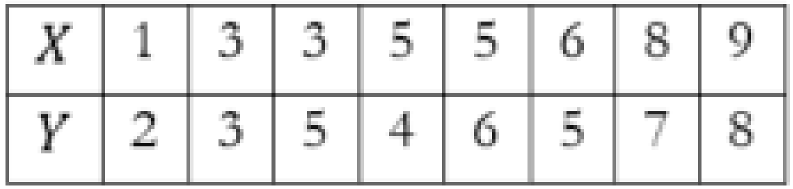
- Suppose that we have collected N data points, denoted as, (x_n, y_n) for $n = 1, 2, \dots, N$.
- The sum of the squared of the vertical derivations of the observed data points from the line is:

$$l(\beta_0, \beta_1) = \sum_{n=1}^{N} [y_n - (\beta_0 + \beta_1 x_n)]^2.$$

- To estimate β_0 and β_1 is to minimize this least-square loss function $l(\beta_0, \beta_1)$.

A simple example

Table 2.2: An exemplary dataset



X	1	3	3	5	5	6	8	9
Y	2	3	5	4	6	5	7	8

The R-code to verify your calculation:

```
## Simple example of regression with one predictor
data = data.frame(rbind(c(1,2),c(3,3),c(3,5),c(5,4),c(
5,6),c(6,5),c(8,7),c(9,8)))
colnames(data) = c("Y","X")
str(data)

lm.YX <- lm(Y ~ X, data = data)
summary(lm.YX)
```

Extension to multiple linear regression

- There are more than one predictor: $y = \beta_0 + \sum_{i=1}^{p} \beta_i x_i + \varepsilon$
- In matrix form: $\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$,

$$\text{where } \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{p1} \\ 1 & x_{12} & x_{22} & \cdots & x_{p2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1N} & x_{2N} & \cdots & x_{pN} \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix},$$

$$\text{and } \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{bmatrix}.$$

Derivation of the least-squares estimation

To estimate $\boldsymbol{\beta}$, we can derive the optimization formulation in matrix form as:

$$\min_{\boldsymbol{\beta}} (\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})^T (\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta}).$$

Take the gradient of the objective function and set it to be zero:

$$\frac{\partial (\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})^T (\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0.$$

This leads to the least square estimator of $\boldsymbol{\beta}$ as

$$\widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \boldsymbol{Y}.$$

Notice the resemblance between $\widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \boldsymbol{Y}$ with $\beta_1 = \frac{cov(x,y)}{var(x)}$!

Hypothesis testing of regression parameters

- It is important to recognize that, since $\boldsymbol{y}$ is a random vector and induce uncertainty, $\widehat{\boldsymbol{\beta}}$ is a random vector as well.
- The mean of $\widehat{\boldsymbol{\beta}}$ is $\boldsymbol{\beta}$, as

$$E\left(\widehat{\boldsymbol{\beta}}\right) = E\left[\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\boldsymbol{y}\right] = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T E[\boldsymbol{y}] = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}.$$

- The covariance matrix of $\widehat{\boldsymbol{\beta}}$ can be readily derived as

$$cov\left(\widehat{\boldsymbol{\beta}}\right) = \sigma_{\varepsilon}^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}.$$

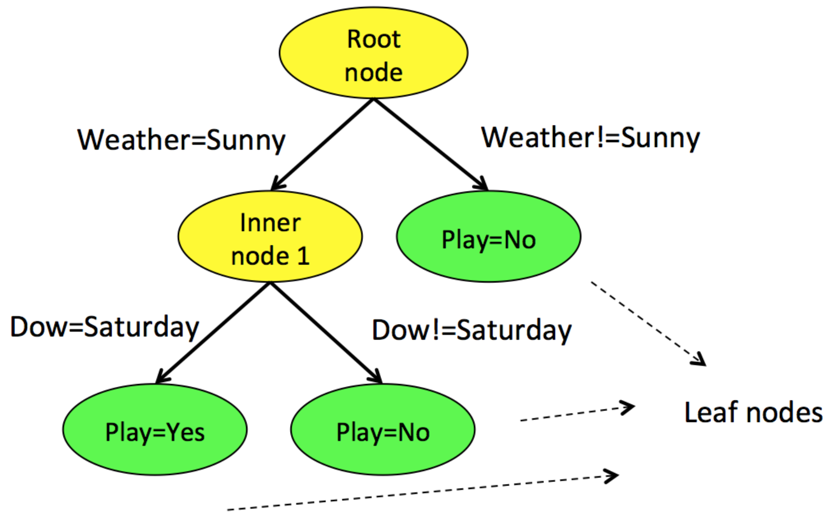
- Thus, it is readily available to derive the hypothesis testing procedure for any regression parameter.

R lab

- Download the markdown code from course website
- Conduct the experiments
- Interpret the results
- Repeat the analysis on other datasets

Decision tree

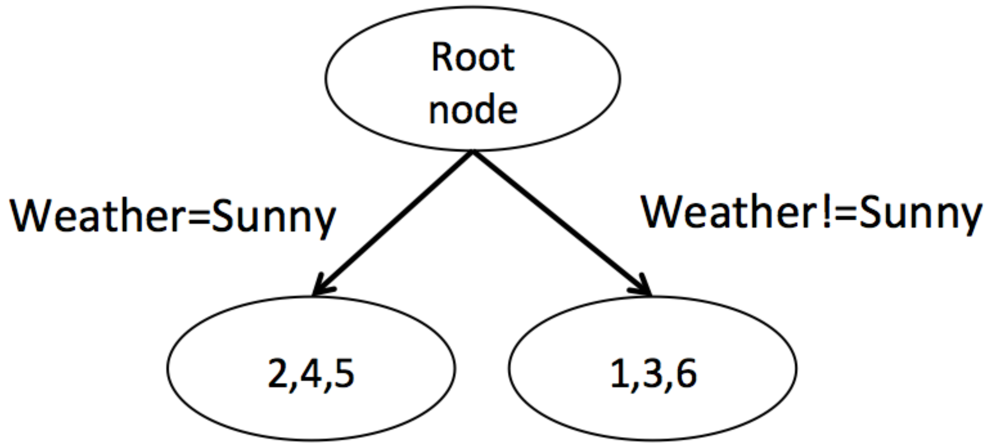
- The linear regression model formalizes the data generating mechanism, which emphasizes an understanding of the underlying system. In contrast, the decision tree mimics heuristics in human reasoning.

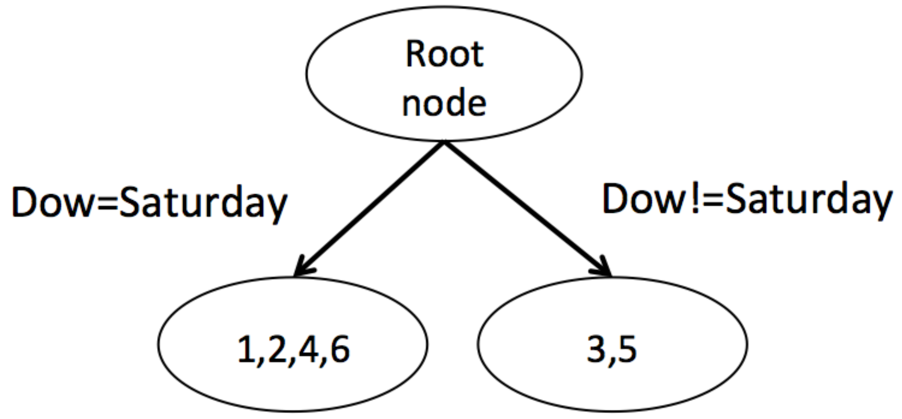


ID	Weather	Dow (day-of-week)	Play
1	Rainy	Saturday	No
2	Sunny	Saturday	Yes
3	Windy	Tuesday	No
4	Sunny	Saturday	Yes
5	Sunny	Monday	No
6	Windy	Saturday	No

A recursive splitting procedure

- A key element of a decision tree is the splitting rules that can lead us through the inner nodes to the final decision node to reach a decision
- E.g., for the exemplary dataset, possible splitting rules are {Weather = Rainy, Weather = Sunny, Dow = Saturday, Dow = Monday, Dow = Tuesday}





Entropy as a measure of Impurity

- For classification problem (the outcome variable is categorical), the impurity of data points in a given node can be measured by **entropy**:

$$e = \sum_{i=1,..,K} -P_i log_2 P_i.$$

- K represents the number of classes and P_i is the proportion of data points in the node that belong to the class i. The entropy e is defined as 0 when the data points in the node all belong to one single class.

Information gain (IG)

- A node having a large impurity is not ready to be a decision node yet.
- Thus, if we want to further split the node and create two more child nodes under it, we look for the best splitting rules that can minimize the impurity of the children nodes.
- This reduction of impurity can then be measured by **information gain** (**IG**), which is the difference of the entropy of the splitting node and the average entropy of the two children nodes weighted by their number of data points.
- The IG is defined as:

$$IG = e_s - \sum_{i=1,..,n} w_i * e_i.$$

- Here, e_s is the entropy at the splitting node, e_i is the entropy at the child node i, and w_i is the number of data points in the children node i divided by the number of data points at the splitting node.

Apply IG on the exemplary dataset

Let’s look at the decision tree model in the left figure. The entropy of the root node is calculated as

$$-\frac{4}{6}log_2\frac{4}{6}-\frac{2}{6}log_2\frac{2}{6}=0.92.$$

The entropy of the left child node (“Weather = Sunny”) is

$$-\frac{2}{3}log_2\frac{2}{3}-\frac{1}{3}log_2\frac{1}{3}=0.92.$$

The entropy of the right child node (“Weather != Sunny”) is 0 since all the three data points (ID = 1,3,6) belong to the same class.

The information gain for the splitting rule “Weather = Sunny” is then

$$IG=0.92-\ \frac{3}{6}*0.92-\frac{3}{6}*0=0.46.$$

Apply IG on the exemplary dataset – cont’d

For the decision tree in the right figure, the entropy of the left child node (“Dow = Saturday”) is

$$-\frac{2}{4}log_2\frac{2}{4}-\frac{2}{4}log_2\frac{2}{4}=1.$$

The entropy of the right child node (“Dow != Saturday”) is 0 since the two data points (ID = 3,5) belong to the same class.

Thus, the information gain for the splitting rule “Dow = Saturday” is then

$$IG=0.92-\ \frac{3}{6}*1-\frac{3}{6}*0=0.42.$$

As the information gain of “Weather = Sunny” is higher, the splitting rule “Weather = Sunny” is preferred over its competitor “Dow = Saturday”.

Greedy nature of recursive splitting

- Based on the concept of impurity and IG, we could develop a greedy strategy that recursively split the data points until there is no further IG, e.g., when there is only one data point, or only one class in the node.
- Most tree-building methods use this approach, with difference in the definitions of the impurity and IG.
- However, this will inevitably lead to a very large decision tree with many inner nodes and unstable decision nodes (i.e., since the decisions assigned to these nodes are inferred empirically based on very few data points).
- Thus, such a decision tree can be sensitive to noisy data, leading to worse accuracy and interpretability.

Pruning of a tree

- To mitigate this problem, the pre-pruning or post-pruning methods can be used to control the complexity of a decision trees.
- Pre-pruning stops growing a tree when a pre-defined criterion is met. One can define the maximum depth of a tree, or minimum number of data points at each node.
- Since we can never anticipate the future growth of a tree, it is commonly perceived that pre-pruning can cause over-simplified and under-fitted tree models. In other words, it is too cautious.
- In contrast, post-pruning prunes a tree after it is fully-grown and has less risk of under-fit as a full-grown tree contains sufficient information captured from the data.

Post-pruning of a tree

- Post-pruning starts from the bottom of the tree. If removing the sub-tree of an inner node does not increase the error, then the sub-tree under the inner node can be pruned.
- Note that, the error here is not the error calculated based on the **training data** that is used to train the tree. This error calculated based on training data is called as **empirical error**.
- Rather, here, we should use the **generalization error**, that is, the error when applied to unseen data.
- Thus, in the famous decision tree algorithm, C4.5, the **pessimistic error estimation** approach is used.

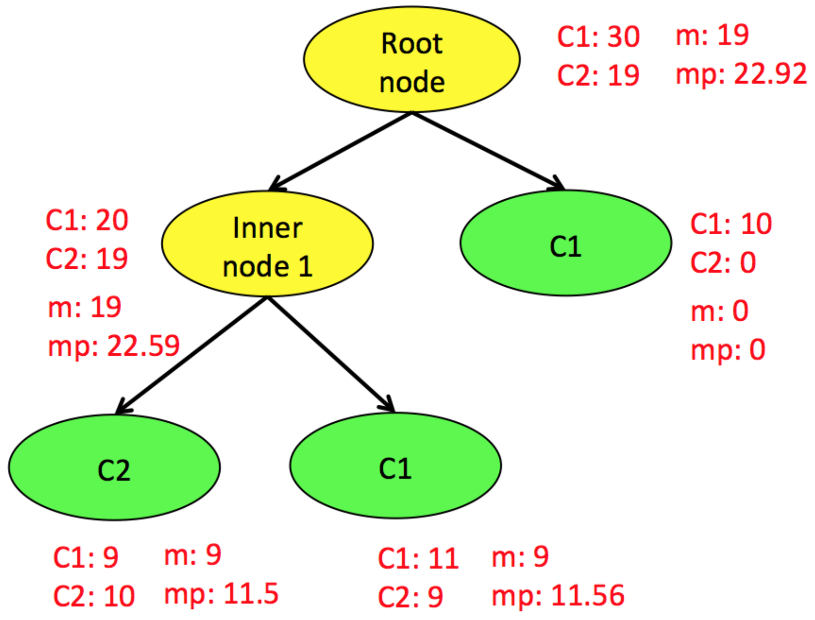
Pessimistic error estimation

- Denote the empirical error rate on the training data as $\hat{e}$, which is only an estimation of the generalization error e.
- Since each data point can be either correctly or wrongly classified, we can view the probability of being correctly classified as a binomial distribution with probability e.
- With this insight, the normal distribution approximation can be applied here to derive the confidence interval of the generalization error e as:

$$\hat{e} - z_{\alpha/2}\sqrt{\frac{\hat{e}(1-\hat{e})}{n}} \leq e \leq \hat{e} + z_{\alpha/2}\sqrt{\frac{\hat{e}(1-\hat{e})}{n}}.$$

- The upper bound of the interval, $\hat{e} + z_{\alpha/2}\sqrt{\frac{\hat{e}(1-\hat{e})}{n}}$, is used as the estimate of e. As this is an upper bound, it is named as pessimistic error estimation.
- Note that, the estimate of the error depends on three values, α, which is often set to be 0.25 so that $z_{\alpha/2}$=1.15; $\hat{e}$, which is the training error; and n, which is the number of data points at the node. Therefore, the estimated error is larger with a smaller n, accounting for the sample size as well.

An example



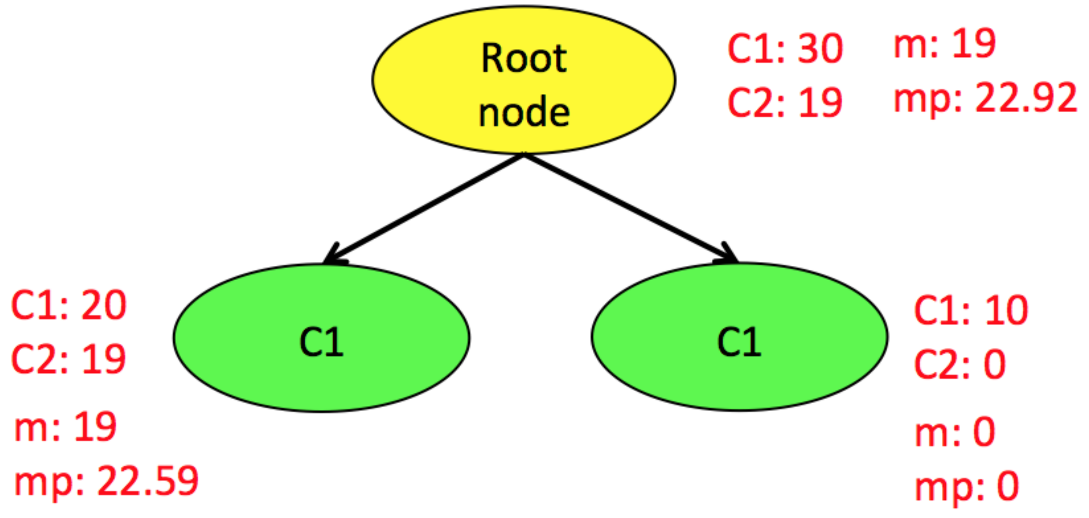
- Consider the inner node 1. We can see that, if we prune the subtree below inner node 1, we will label it with class C1, as 20 of the included data points are labeled as C1 while 19 are labeled as C2. And we can get that the total misclassified instances m is 19. Thus, $\hat{e} = \frac{19}{39} = 0.4871$.
- For the pessimistic error estimation, we can get that

$$\hat{e} + z_{\alpha/2}\sqrt{\frac{\hat{e}(1-\hat{e})}{n}} = 0.4871 + 1.15\sqrt{\frac{0.4871(1-0.4871)}{39}} = 0.579.$$

- Thus, we can get that $mp = 0.579 \times 39 = 22.59$.

An example – cont’d

- Now, let’s see if the splitting of this inner node into its two child nodes improves the pessimistic error. It can be seen that, with this subtree of the two child nodes, the total misclassified instances m is 9+9=18. And by pessimistic error estimation, $mp = 11.5 + 11.56 = 23.06$ for the subtree. Therefore, based on the pessimistic error, we should prune the subtree.



The parameter “cp” in the R package “rpart”

- Both pre-pruning and post-pruning are useful
- Pre-pruning tends to lead to oversimplified decision tree models. In this aspect, post-pruning seems to be more preferable, but not always.
- E.g, another popular strategy, as used in the “**rpart**” R package, the complexity parameter (cp) is used. Using the parameter, all splits need to improve the impurity score, e.g., information gain, by at least a factor of cp, that is, splits do not decrease the impurity score by cp will not be pursued. This strategy also works well in many applications

R lab

- Download the markdown code from course website
- Conduct the experiments
- Interpret the results
- Repeat the analysis on other datasets