EE5907 Pattern Recognition

Lecture 4 Linear Regression

&

(Supplementary) Nonparametric Methods

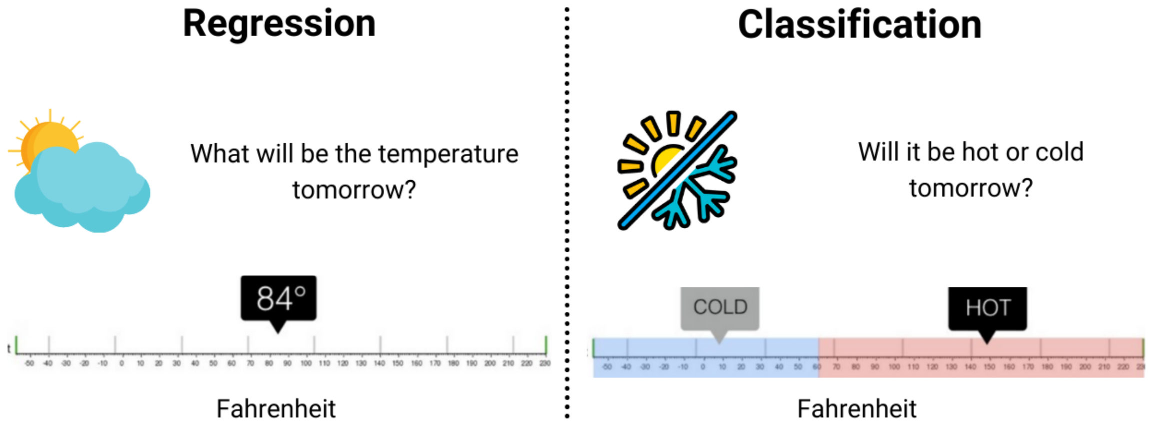
NUS
National University of Singapore
National University of Singapore

Linear Regression

- ✓ Basic Terminology
- ✓ Least Square Estimation for Linear Regression
- ✓ Extension: Ridge Regression and Lasso Regression

Regression? Regression vs. Classification?

- A set of processes for estimating the relationships between a **dependent variable** (a real-valued output) and one or more **independent variables** (real-valued inputs).
- What is the difference between regression and classification?
  - Regression: used to determine **continuous values** (e.g., price, income, temperature).
  - Classification algorithms: used to forecast or **classify** the **distinct values** (e.g., Real or False, Male or Female, Cold or Hot).


Regression
What will be the temperature tomorrow?
84°
Fahrenheit
Classification
Will it be hot or cold tomorrow?
COLD
HOT
Fahrenheit

Define Linear Regression

- In short, predicting the dependent variable $y \in \mathbb{R}$ given a vector of independent variables $x \in \mathbb{R}^D$.
- **Linear** regression: the expected value of the output assumed to be a linear function of the input: $\mathbb{E}[y|x] = \boldsymbol{w}^T x$.
- Mathematically, it is of the form:

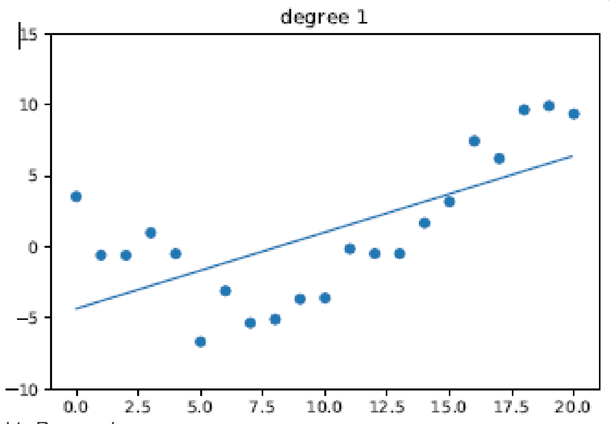
$$p(y|x, \theta) = \mathcal{N}(y|w_0 + \boldsymbol{w}^T x, \sigma^2)$$

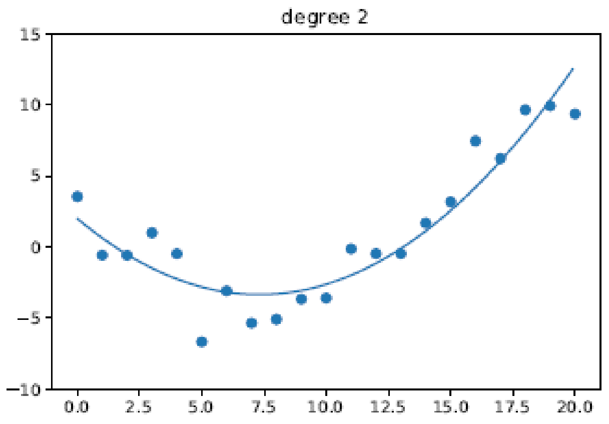
- The parameters of the model are $\theta = (w_0, \boldsymbol{w}, \sigma^2)$.
  - Vector of params $\boldsymbol{w}_{1:D}$: the weights or regression coefficients; w_0: the offset or bias term.
  - For simplicity, we can assume $x = [1, x_1, \dots, x_D]$ to absorb offset term w_0 into the weight vector $\boldsymbol{w}$.

Linear Regression: More Terminologies

- If the input is a 1-D input ($D = 1$), the model has the form $f(x; \boldsymbol{w}) = ax + b$, $b = w_0$ as the intercept and $a = w_1$ as the slope – simple linear regression.
- If input $x \in \mathbb{R}^D$ is multi-dimensional ($D > 1$) – multiple linear regression.
- Further, if the output $y \in \mathbb{R}^J$ is also multi-dimensional ($J > 1$) – multivariate linear regression. The model satisfies the fact that:

$$p(y|x, \boldsymbol{w}) = \prod_{j=1}^{J} \mathcal{N}\left(y_j | w_j^T x, \sigma_j^2\right)$$





Linear Regression for General Cases

- For linear regression: assume linear correlation between input and output – is this possible in real-world cases?
- Unfortunately, in general, a straight line will not provide a good fit to most datasets!
  - How to leverage linear regression then?
- Apply a nonlinear transformation to the input features:

$$p(y|x,\theta) = \mathcal{N}(y|\boldsymbol{w}^T\phi(x), \sigma^2)$$

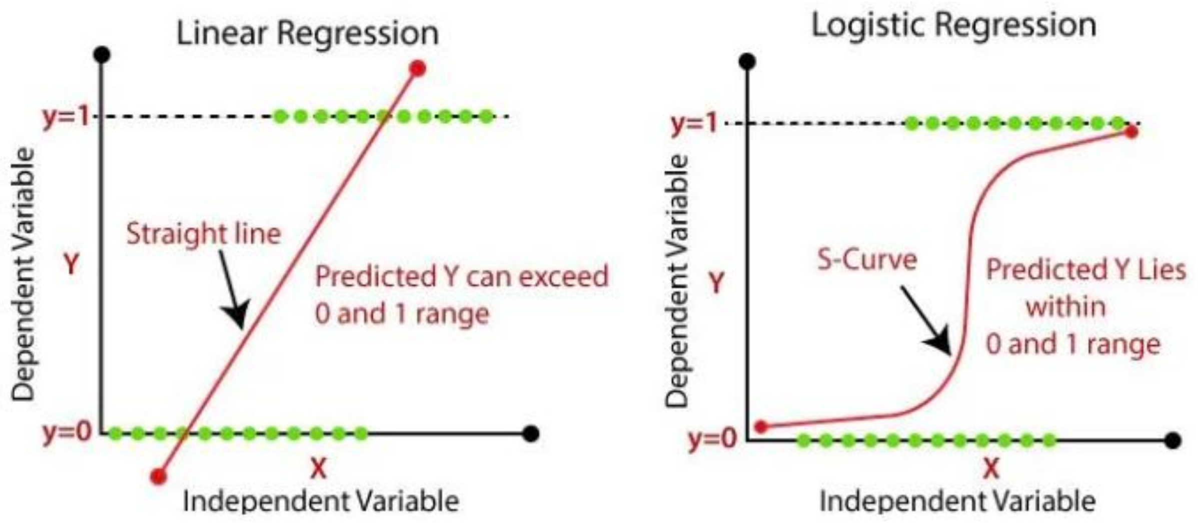
- As long as the parameters of the **feature extractor** ϕ are fixed, the model remains ***linear** in the parameters*.

Logistic Regression and Linear Regression?

- Another “type” of regression that we may encounter in literature is the logistic regression. What is logistic regression?
- Logistic regression is generally **NOT** used for regression task!!
  - Instead, it is more related to the **Classification** task.
  - The output of logistic regression is usually a **class label** $y \in \{1, \dots, C\}$.
- Then why is it called “Logistic Regression”?
  - A related question: can linear regression be used for **classification**?

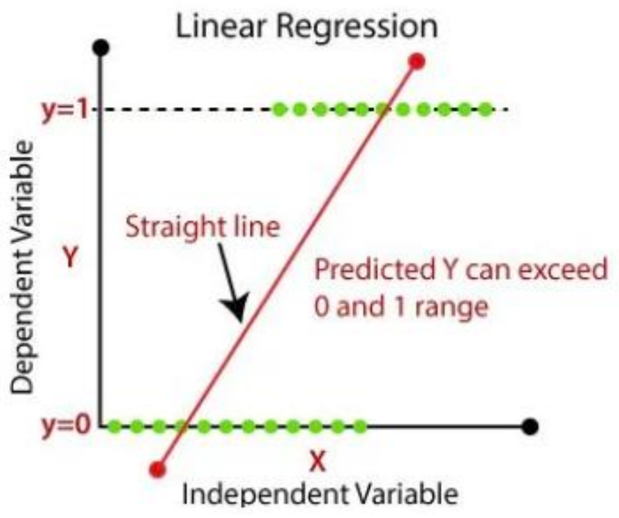
Logistic Regression

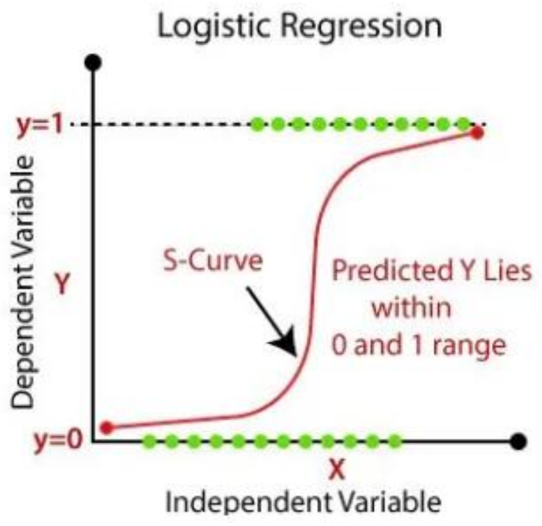
- Yes, but not effective:
  - Classification – prediction of the probability above or below a threshold.
  - Problem with range: linear regression yield predictions below 0 or above 1, does not align with the probability interpretation.
  - Problem with threshold: linear regression predictions may not easily map to a clear classification threshold (e.g., 0.5), and this threshold can shift when new data points are added, causing instability in predictions.
- Logistic regression is designed for classification building on features obtained from linear regression!
- How is it achieved?



Logistic Regression

- The two key components of logistic regression:
- Sigmoid (logistic) function: logistic regression incorporates the logistic / sigmoid function that squeeze output into range $[0,1]$.
- Linear regression: logistic regression builds upon linear regression, feeding its output to the logistic / sigmoid function.





Fitting Linear Regression Models

- How can we fit the linear regression model to given data?
- Recall what we did for the linear classifiers?
  - Minimize the **cost function** that is formulated as the mean square error (MSE) or the least squares error (LSE).
  - E.g., for LSE, we want to minimize: $J(\boldsymbol{w}) = \sum_i^N \left(y_i - x_i^T \boldsymbol{w}\right)^2$.
- Can we apply similar techniques for fitting the linear regression models to given data?
  - Yes, the only difference may be the choice of error (cost function).

Fitting LR Models with NLL

- To fit a LR model, it is common to minimize the negative log likelihood on the training set.
  - Recall NLL: $\mathrm{NLL}(\boldsymbol{\theta}) \triangleq -\log p(\mathcal{D}|\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n|x_n, \boldsymbol{\theta})$.
  - Note that we assume a **normal distribution** of the data.
  - Thus we can obtain the objective function of the params as:

$$\mathrm{NLL}(\boldsymbol{w}, \sigma^2) = -\sum_{n=1}^{N} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-x_n^T\boldsymbol{w})^2}\right]$$

$$= \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2 + \frac{N}{2}\log(2\pi\sigma^2)$$

Fitting LR Models with NLL

- To fit a LR model, it is common to minimize the negative log likelihood on the training set.

$$\mathrm{NLL}(\boldsymbol{w},\sigma^2) = -\sum_{n=1}^{N}\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2\sigma^2}\left(y-x_n^T\boldsymbol{w}\right)^2}\right] = \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n-\hat{y}_n)^2+\frac{N}{2}\log(2\pi\sigma^2)$$

  - The predicted response is $\hat{y}_n \triangleq x_n^T\boldsymbol{w}$.
  - The Maximum Likelihood Estimation (MLE) point is where $\nabla_{w,\sigma}\mathrm{NLL}=0$.
  - Here, we have two parameters to optimize: $\boldsymbol{w}$ and σ.Usually, we first optimize w.r.t $\boldsymbol{w}$ then solve for the optimal σ.
  - We first focus on estimating the weights w.
- In such case, the NLL is **equivalent to the LSE estimation**!

Recap: LSE Estimation for Linear Classifiers

- Minimizing the LSE formulation $\mathrm{LSE}(\boldsymbol{w}) = J(\boldsymbol{w}) = \sum_i^N \left(y_i - x_i^T \boldsymbol{w}\right)^2$ w.r.t. $\boldsymbol{w}$:

$$\sum_i^N x_i\left(y_i - x_i^T \widehat{\boldsymbol{w}}\right) = 0 \Rightarrow \left(\sum_i^N x_i x_i^T\right)\widehat{\boldsymbol{w}} = \sum_i^N x_i y_i$$

- Mathematically, we can define two matrices:

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

- I.e., X is an $N \times D$ matrix whose rows are the available training feature vectors, and Y is a vector consisting of the corresponding desired outputs.

Recap: LSE Estimation for Linear Classifiers

- With the matrices, we have $\sum_i^N x_i x_i^T = X^T X$ and $\sum_i^N x_i y_i = X^T Y$. Hence the equation for optimal weight vector is formulated as:
- $(X^T X)\widehat{w} = X^T Y \Rightarrow \widehat{w} = (X^T X)^{-1} X^T Y$
  - Assuming an **over-determined** system.
  - $X^T X$ is the sample correlation matrix, $X^+ \equiv (X^T X)^{-1} X^T$ is the *pseudoinverse* of X, which is only valid if the matrix $(X^T X)$ is invertible.
  - When $N = D$, the optimal weight vector satisfies $X\widehat{w} = Y$.
  - In most cases where $N > D$, the solution obtained from $\widehat{w} = X^+ Y$ is the vector that minimizes the sum of squared errors.

LSE Estimation for Linear Regression

- The overall formulation of LSE estimation for the optimal weight w in linear regression is exactly the same as that for linear classifier!
- The difference: elements in Y (i.e., $y_1, \dots, y_n$) are now continuous values instead of discrete class label values.
- The solution $\widehat{w} = (X^T X)^{-1} X^T Y$ is defined as the ordinary least squares (OLS) solution. Is this solution unique?

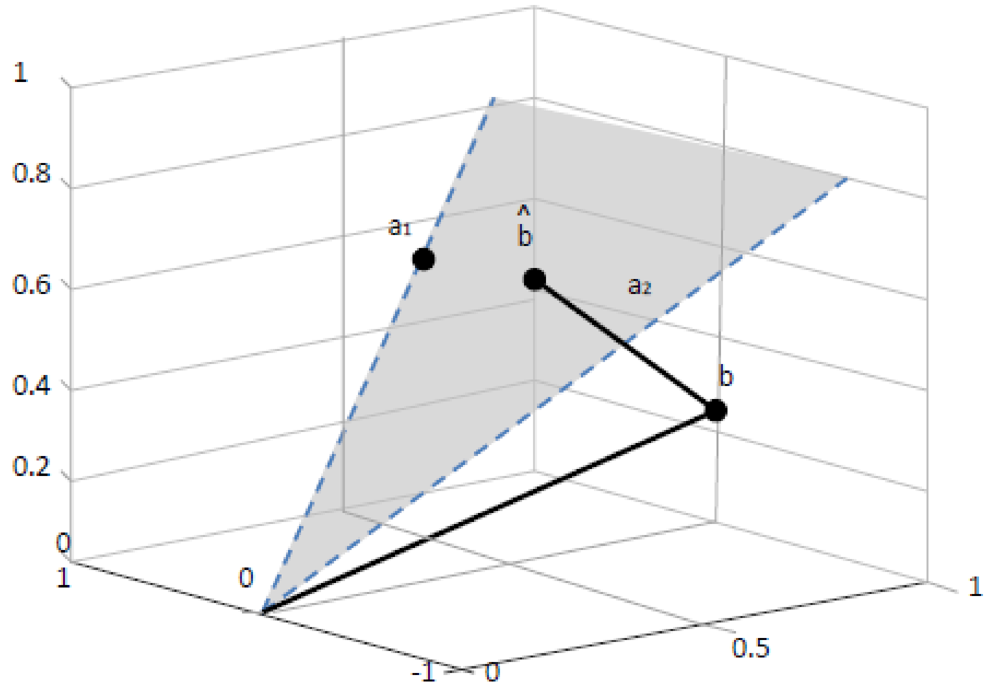
- To check if solution is unique, show that the Hessian is positive definite:
- $H(w) = \frac{\partial^2}{\partial w^2} \mathrm{LSE}(w) = 2X^T X$, which is positive definite if X is full rank (i.e., the columns of X are linearly independent).

Geometric Interpretation of LSE

- Safely assume that $N \gg D$ (or at least $N > D$), thus there are more observations than unknowns – an over-determined system.
- Our goal: seek a vector $\hat{Y} \in \mathbb{R}^N$ that lies in the linear subspace spanned by X and is as close to ground-truth Y: $\arg \min_{\hat{Y} \in \text{span}(X)} \| Y - \hat{Y} \|_2$.
- Since $\hat{Y} \in \text{span}(X)$, there would exists some weight vector w such that: $\hat{Y} = w_1 x_{:,1} + \cdots + w_D x_{:,D} = Xw$.
- To minimize the norm of the residual $Y - \hat{Y}$, we want the residual vector to be orthogonal to every column of X, thus:
  - $x_{:,d}^T (Y - \hat{Y}) = 0 \Rightarrow X^T (Y - Xw) = 0 \Rightarrow w = (X^T X)^{-1} X^T Y$.
  - Hence the predicted value of y is given by $\hat{Y} = Xw = X(X^T X)^{-1} X^T Y$.
- How to interpret this geometrically?

Geometric Interpretation of LSE

- $\hat{Y} = Xw = X(X^TX)^{-1}X^TY$
- This corresponds to an **orthogonal projection** of Y onto the column space of X.
- Suppose $D = 3$ equations and $N = 2$ unknowns for the system $Y = Xw$. a_1 and a_2 are the columns of X, defines a 2-d linear subspace embedded in $\mathbb{R}^3$.
- The target vector b is a vector in $\mathbb{R}^3$; its orthogonal projection onto the linear subspace is denoted $\hat{b}$. The line from b to $\hat{b}$ is the vector of residual errors, whose norm we want to minimize.



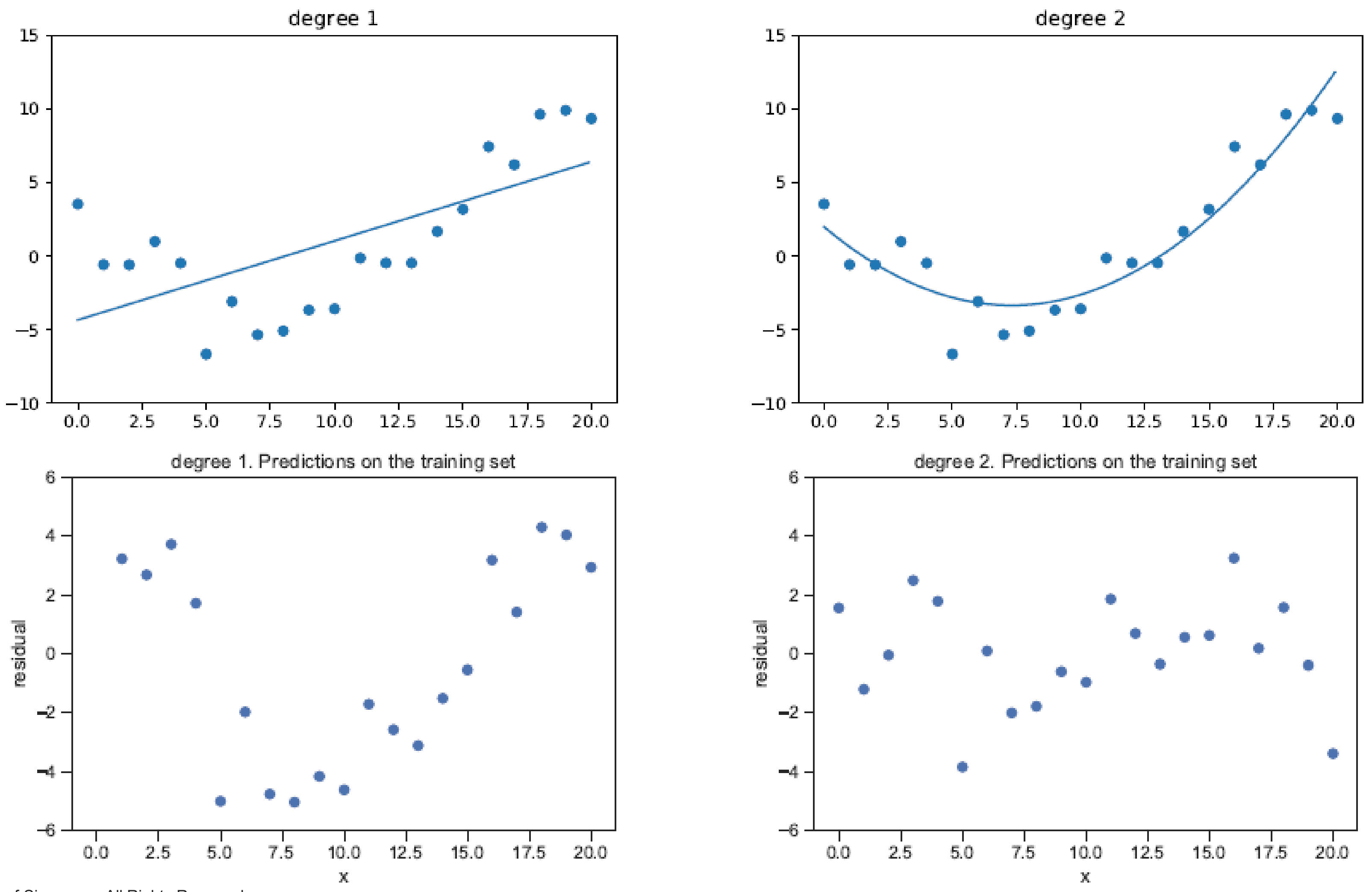
Weighted Least Squares

- In some cases, a weight should be associated with each different sample.
  - Previously, we have $p(y|x;\theta) = \mathcal{N}(y|w^T x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y - x_n^T \boldsymbol{w})^2}$
  - In heteroskedastic regression, the variance depends on the input: $p(y|x;\theta) = \mathcal{N}(y|w^T x, \sigma(x)^2) = \frac{1}{\sqrt{2\pi\sigma(x)^2}} e^{-\frac{1}{2\sigma^2}(y - x_n^T \boldsymbol{w})^2}$;
  - In such regression, we reformulate $p(y|x;\theta) = \mathcal{N}(y|Xw, \Lambda^{-1})$ where $\Lambda = \mathrm{diag}(1/\sigma(x_n)^2)$.
  - This is known as a weighted linear regression. The OLS solution is given as: $\hat{w} = (X^T \Lambda X)^{-1} X^T \Lambda Y$ – weighted least squares estimate.

Measuring Goodness of Fit

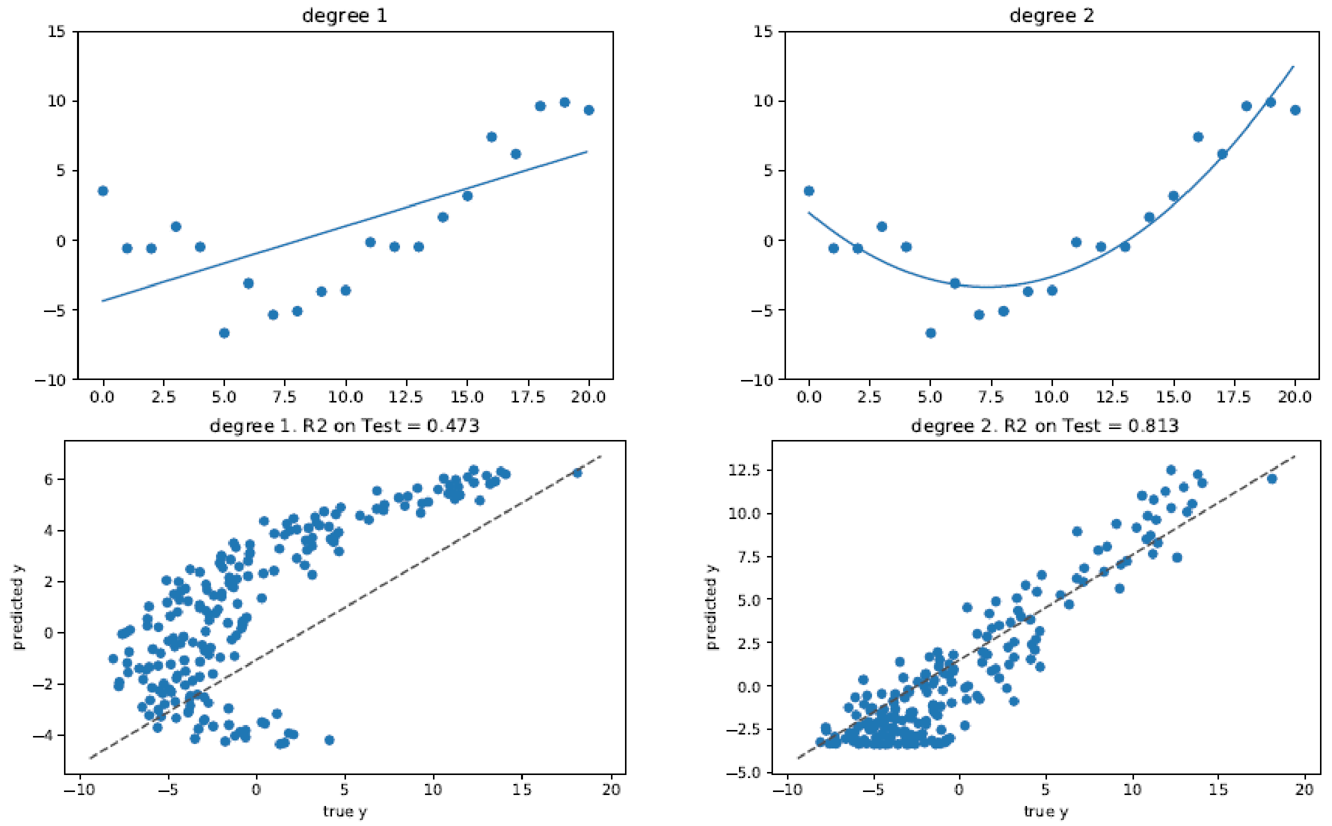
- How to determine if the LR model obtain really fits the data?
  - I.e., how to assess the goodness of fit?
- For the simplest case, let us assume 1-d input, and 1-d output:
  - One intuitive method – Residual plots:
  - Residual plots: plotting the residuals $r_n = y_n - \hat{y}_n$ vs. the input x_n.
  - What should we observe if the fit is good?
  - The model assumes that the residuals have a $\mathcal{N}(0, \sigma^2)$ distribution, so the residual plot should be a cloud of points more or less equally above and below the horizontal line at 0, without any obvious trends.

Goodness of Fit: Visualize the Residual Plot



Measuring Goodness of Fit

- For a multi-dimensional input, we can plot $\hat{y}_n$ vs. y_n instead of r_n vs. x_n. A good model will have points lie on a diagonal line.



Measuring Goodness of Fit

- The residual plot is considered a relatively **qualitative** analysis.
  - How can we assess the goodness quantitatively – and definitively?
  - Computing the least squares error value (also known as the **Residual Sum of Squares** RSS value) can asses the fit quantitatively.

$$\mathrm{LSE}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) = \sum_{i}^{N} \left(y_i - x_i^T \boldsymbol{w}\right)^2$$

  - Another measure often used is the Root Mean Squared Error (RMSE):

$$\mathrm{RMSE}(w) \triangleq \sqrt{\frac{1}{N}\mathrm{RSS}(w)}$$

  - Both quantities: the smaller the better!

Measuring Goodness of Fit

- Further, a more interpretable measure can be computed using the coefficient of determination, formulated as:

$$R^2 \triangleq 1 - \frac{\sum_{n=1}^{N}(\hat{y}_n - y_n)^2}{\sum_{i}^{N}(\bar{y} - y_n)^2} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}}$$

- $\bar{y} = \frac{1}{N}\sum_{n=1}^{N} y_n$ is the empirical mean of the responses, RSS the residual sum of squares, TSS is the total sum of squares $\mathrm{TSS} = \sum_{n=1}^{N}(y_n - \bar{y})^2$.
- R^2 measures the variance in the predictions relative to the simple constant prediction of $\hat{y}_n = \bar{y}$.
  - If a model does no better at predicting than using the mean of the output, $R^2 = 0$.
  - If the model perfectly fits the data, then the RSS will be 0, so $R^2 = 1$.
- In general, **larger values** imply greater reduction in variance (**better fit**).

Limitations of LSE and Ridge Regression

- Fitting the LR model with LSE provides an optimal solution given a set of input data.
  - Disadvantage of fitting through LSE?
  - Prone to overfitting – the solution is optimal ONLY w.r.t. the input data.
- Any possible solutions?
  - To encounter overfitting – add in regularization.
  - E.g., Including a zero-mean Gaussian prior on the weights, formulated as: $p(w) = \mathcal{N}(w|0, \lambda^{-1}I)$. Optimal weight obtained via MAP estimation.
  - This is called **Ridge Regression**.

Ridge Regression

- In detail, the optimal weight is computed via MAP estimation:

$$\begin{aligned}\widehat{w}_{map} &= \arg\min \frac{1}{2\sigma^2}(Y - Xw)^T(Y - Xw) + \frac{1}{2\tau^2} w^T w \\ &= \arg\min \mathrm{RSS}(w) + \lambda \parallel w \parallel_2^2\end{aligned}$$

  - $\lambda \triangleq \sigma^2/\tau^2$ is proportional to the strength of the prior; $\parallel w \parallel_2$ is defined as the ℓ_2 norm of the vector w formulated as:

$$\parallel w \parallel_2 \triangleq \sqrt{\sum_{d=1}^{D} |w_d|^2} = \sqrt{w^T w}$$

  - Penalizing weights that become too large in magnitude. In general, this technique is called ℓ_2 **regularization** or **weight decay**.

Optimal Weight for Ridge Regression

- The MAP estimate corresponds to minimizing the penalized objective:

$$J(w) = (Y - Xw)^T (Y - Xw) + \lambda \parallel w \parallel_2^2$$

  - λ is a **hyperparameter** that controls the strength of the regularizer.
  - The derivative of $J(w)$ w.r.t. w is given by:

$$\nabla_w J(w) = 2(X^T Xw - X^T Y + \lambda w)$$

  - Hence the optimal weight $\hat{w}_{map}$ is thus formulated as:

$$\hat{w}_{map} = (X^T X + \lambda I_D)^{-1} X^T Y = \left(\sum_n x_n x_n^T + \lambda I_D \right)^{-1} \left(\sum_n y_n x_n \right)$$

Recap: Under-Determined Systems

$$(X^T X)\hat{w} = X^T Y \Rightarrow \hat{w} = (X^T X)^{-1} X^T Y$$

- Assume $N > D$, i.e., we have more samples than the number of features per sample – denoted as a over-determined system.

- Question: what if $N < D$, i.e., we DON'T have enough samples?
  - There will be more than one unique solution!
  - We then need to constrain the search. Assume (XX^T) is invertible, denote $w = X^T a$, then we have $XX^T a = y$ from $Xw = y$.
  - Thus $\hat{a} = (XX^T)^{-1} y$, thus $\hat{w} = X^T (XX^T)^{-1} y$
  - Here we denote $X^\dagger = X^T (XX^T)^{-1}$ as the right inverse of X.

Ridge Regression for Less Samples

- Now, what if $N < D$, which is often the case where we would require extra regularizations for regression.
  - Note that without regularization, we have $\hat{w} = X^T(XX^T)^{-1}y$.

- For $N < D$, the optimal weight is formulated as:

$$\hat{w}_{map} = X^T(XX^T + \lambda I_N)^{-1}Y$$

  - This is referred to as the dual form of the ridge regression result, where the form of $\hat{w}_{map} = (X^TX + \lambda I_D)^{-1}X^TY$ is regarded as the primal form.
  - Leave a question, how to obtain the $\hat{w}$ in its dual form?

Optimal Weight Under Less Samples

  - Dual form solution: $\hat{w} = X^T(XX^T + \lambda I_N)^{-1}Y$
- Is it still possible to re-use the primal form of ridge regression for solving ridge regression?
  - Yes, by using SVD on the X matrix
- How it works (normally, write a set of code for SVD)
  - Let $X = USV^T$ be the SVD of X, where $V^TV = I_D$, $UU^T = U^TU = I_N$, and S is a diagonal $N \times N$ matrix.
  - Let $R = US$ an $N \times D$ matrix, thus:
$$\hat{w}_{map} = V(R^TR + \lambda I_N)^{-1}R^TY$$
  - Replace the D-dim vectors x_i with the N-dim vectors r_i and perform our penalized fit as before.
  - The overall time is now $O(DN^2)$ operations, which is less than $O(D^3)$ if $D > N$.

```python
# Imports

from skimage.color import rgb2gray
from skimage import data
import matplotlib.pyplot as plt
import numpy as np
from scipy.linalg import svd

"""
Singular Value Decomposition
"""
# define a matrix
X = np.array([[3, 3, 2], [2, 3, -2]])
print(X)
# perform SVD
U, singular, V_transpose = svd(X)
# print different components
print("U: ", U)
print("Singular array", singular)
print("V^{T}", V_transpose)
```

A More Extreme Regularization

- Recap: for ridge regression, we assumed a Gaussian prior for the regression coefficients when fitting the LR models: $p(w) = \mathcal{N}(w|0, \lambda^{-1}I_D)$.
- Why is it a good choice?
  - Encourage small parameters – prevents overfitting.
  - Is this enough for ALL cases – obviously NOT.
- Sometimes we want the parameters to not **just be** small, but to be exactly zero, i.e., we want $\hat{w}$ to be sparse.
  - Why sparse representation? (Or sparse model?) – think first from model point of view:
  - Leads to simpler model by learning what parameters **can be dropped**, lowering their total number.
  - Produce models that are **smaller** in terms of **memory** usage.
  - How to achieve such a goal?

A More Extreme Regularization

- Goal: parameters that are exactly zero: minimize the $L0$-norm.

$$\| w \|_0 = \sum_{d=1}^{D} \mathbb{I}(|w_d| > 0)$$

- From the feature point of view, sparse models are also useful as it can be used to perform **feature selection**.
  - Note that the prediction has the form $f(x; w) = \sum_{d=1}^{D} w_d x_d$.
  - If any $w_d = 0$, ignore the corresponding feature x_d.
  - This idea can be applied to nonlinear models, such as MLP!

- The next question: how to compute such sparse estimate?

Lasso Regression: LR with Laplace Prior

- To achieve a sparse estimate, replace the Gaussian prior on the weights.
- One feasible solution: apply the Laplace prior over the weights – perform MAP estimation using the Laplace distribution:

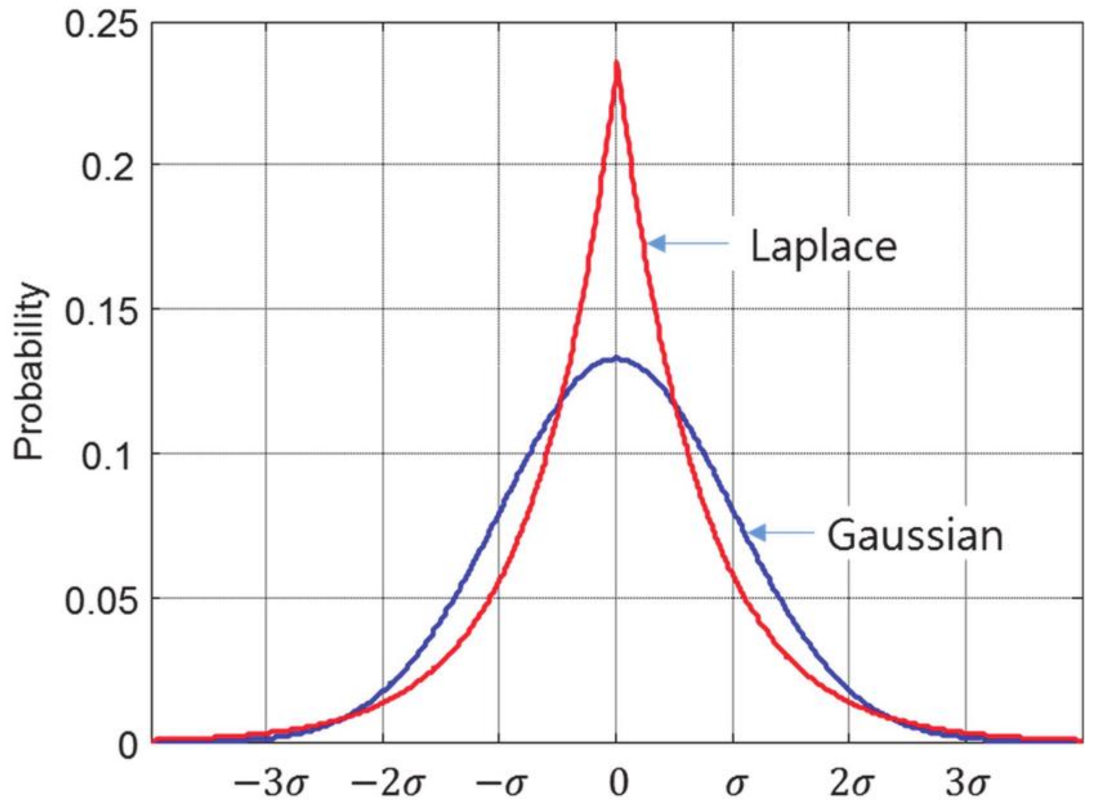
$$p(w|\lambda) = \prod_{d=1}^{D} \mathrm{Laplace}(w_d|0,1/\lambda) \propto \prod_{d=1}^{D} \exp(-\lambda|w_d|)$$

- λ: sparsity param;

$$\mathrm{Laplace}(w|\mu,b) \triangleq \frac{1}{2b}\exp\left(-\frac{|w-\mu|}{b}\right)$$

- Here μ and b are the location and scale parameters.

Lasso Regression: LR with Laplace Prior



- Compared to Gaussian, Laplace distribution puts more density on 0 even when we fix the variance to be the same.

Lasso Regression: LR with Laplace Prior

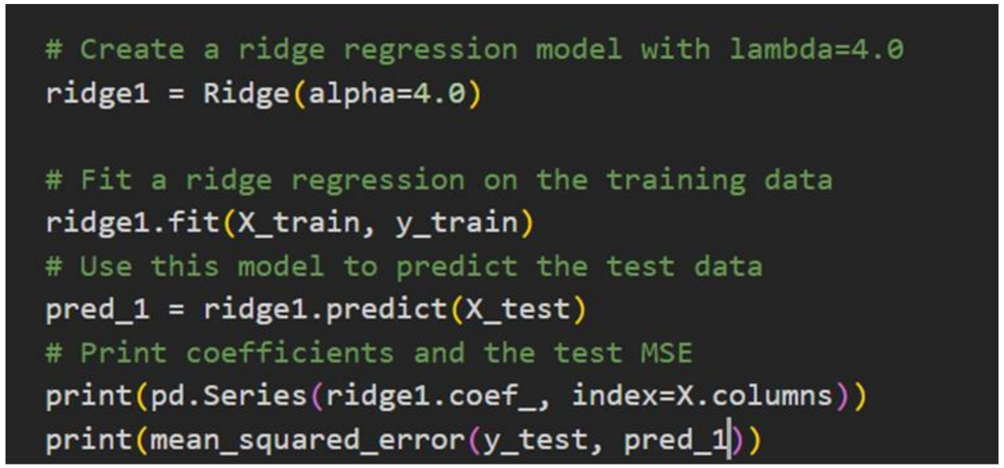
- To perform MAP estimation of a LR with the Laplace prior, minimize the penalized objective of:

$$PNLL(w) = -\log p(\mathcal{D}|w) - \log p(w|\lambda) = \| Xw - Y \|_2^2 + \lambda \| w \|_1$$

- This is often referred as the ℓ_1-regularization, as $\| w \|_1 \triangleq \sum_{d=1}^{D} |w_d|$ is the ℓ_1 norm of w. (Recall, ridge regression leverage ℓ_2-regularization)
  - Method is referred to as **lasso**, “Least Absolute Shrinkage and Selection Operator”.

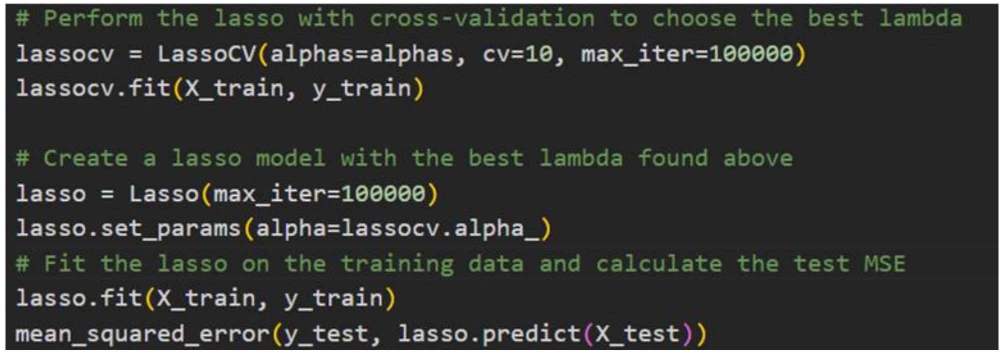
- Note that other norms of the weight vector can also be leveraged. In general, the q-norm is defined simply as: $\| w \|_q \triangleq \left(\sum_{d=1}^{D} |w_d|^q\right)^{1/q}$.
- For $q < 1$, we can get even sparser solutions, but $q \geq 0$.
- Note that for any $q < 1$, the problem becomes non-convex. Thus ℓ_1-norm is the tightest **convex relation** of the ℓ_0-norm.

Regression with Ridge and Lasso



```python
# Create a ridge regression model with lambda=4.0
ridge1 = Ridge(alpha=4.0)

# Fit a ridge regression on the training data
ridge1.fit(X_train, y_train)
# Use this model to predict the test data
pred_1 = ridge1.predict(X_test)
# Print coefficients and the test MSE
print(pd.Series(ridge1.coef_, index=X.columns))
print(mean_squared_error(y_test, pred_1))
```



```python
# Perform the lasso with cross-validation to choose the best lambda
lassocv = LassoCV(alphas=alphas, cv=10, max_iter=100000)
lassocv.fit(X_train, y_train)

# Create a lasso model with the best lambda found above
lasso = Lasso(max_iter=100000)
lasso.set_params(alpha=lassocv.alpha_)
# Fit the lasso on the training data and calculate the test MSE
lasso.fit(X_train, y_train)
mean_squared_error(y_test, lasso.predict(X_test))
```

NUS
National University of Singapore
National University of Singapore

EE5907 Pattern Recognition

Thank you for your listening!

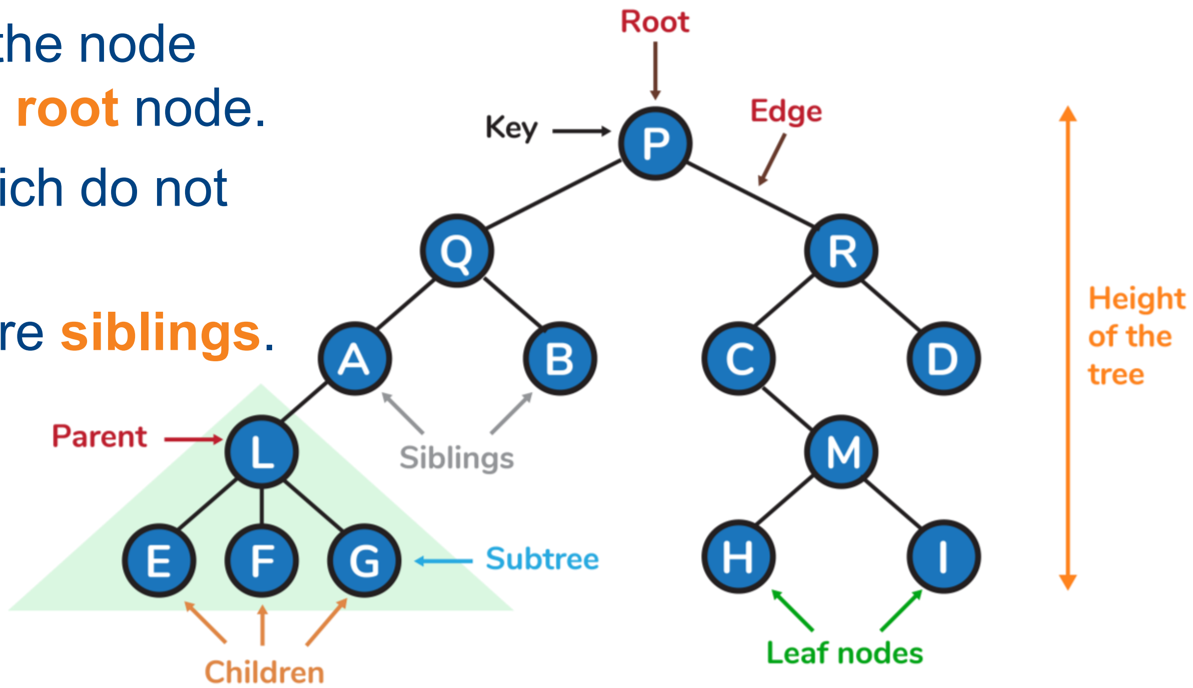
Supplementary Nonparametric PR Methods

- ✓ The Classification and Regression Trees / Decision Trees
- ✓ Bagging / Bootstrap Aggregating
- ✓ Random Forests

A Quick Review: Trees

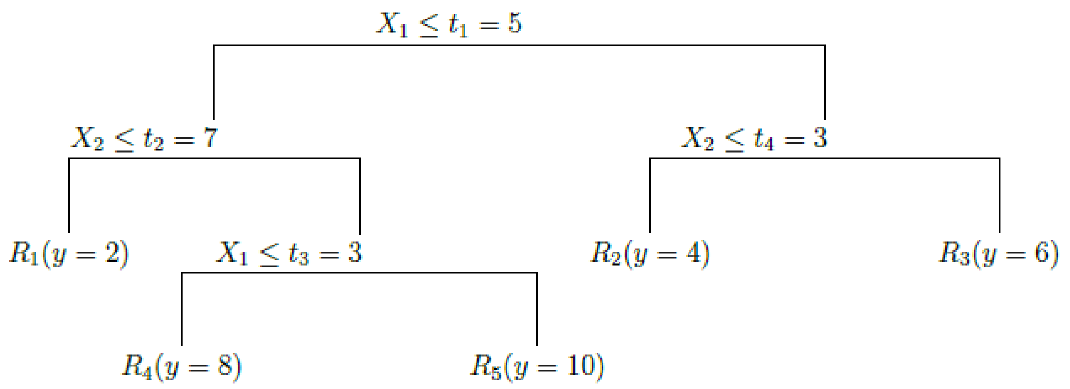
Question: How can we leverage Trees for Classification or Regression?

- Tree data structure is a data structure to store data in hierarchical manner.
  - Tree data structure has roots, branches, and leaves connected.
  - Some basic terminologies:
  - Parent Node: the predecessor of a node is called the **parent** node of that node.
  - Child Node: the immediate successor of a node is called the **child** node of that node.
  - Root Node: The top-most node of a tree or the node which does not have any parent node is the **root** node.
  - Leaf Node or External Node: The nodes which do not have any child nodes are called **leaf** nodes.
  - Sibling: Children of the same parent node are **siblings**.
  - Level of a node: The count of edges on the path from the root node to that node. The root node has level **0**.



Regression Trees

- In general, we leverage the Classification and Regression Tree (CART) for classification and regression tasks.
  - More familiar terminology: Decistion Tree (DT)
- First consider regression trees:
  - Inputs are real-valued.
  - Tree consists of a set of **nested decision rules**.
  - How does it work in general?
  - At each node i, the feature dimension d_i of input x is compared to threshold t_i, and the input then passed down to the left or right branch depending on the comparison.
  - At the leaves of the tree, the model specifics the predicted output for any input that falls into that part of the input space.

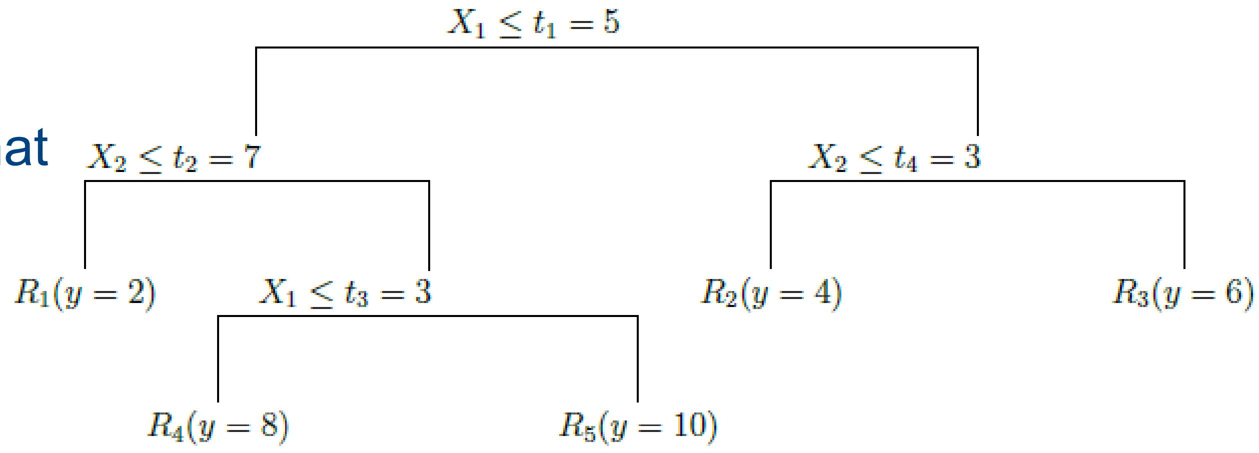


Regression Trees: Example

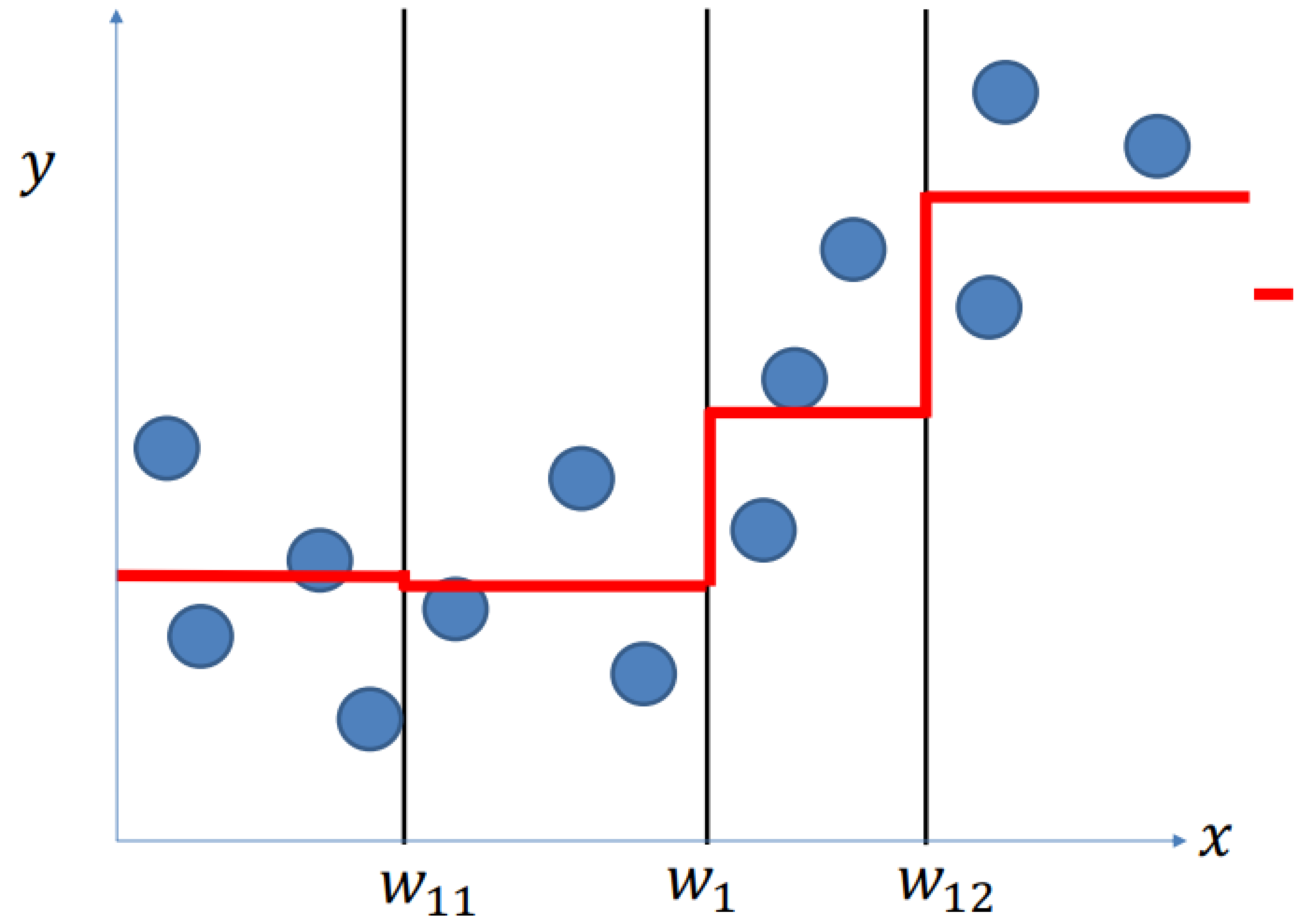
- Example, consider the following regression tree:
  - The first node: Compare if x_1 is less than threshold t_1;
  - If yes: compare if x_2 is less than threshold t_2 – if yes: bottom left leaf node R_1.
  - The space is defined by: $R_1 = \{x: x_1 \leq t_1, x_2 \leq t_2\}$.
  - The entire input space is further partitioned into 5 regions using axis parallel splits.
  - Formally, a regression tree is defined by:

$$f(x;\theta) = \sum_{j=1}^{J} w_j \mathbb{I}(x \in R_j)$$

  - R_j is the region specified by the j-th leaf node and w_j is the predicted output for that leaf node.
  - $\theta = \{(R_j, w_j): j = 1:J\}$, J is number of nodes.

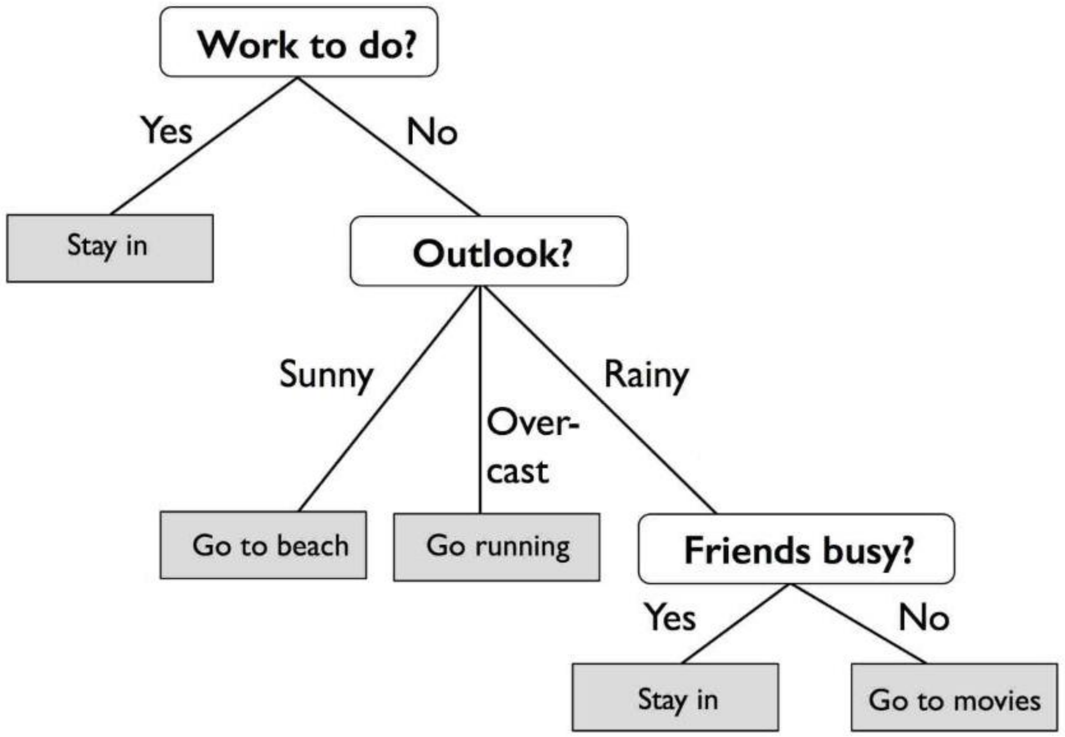


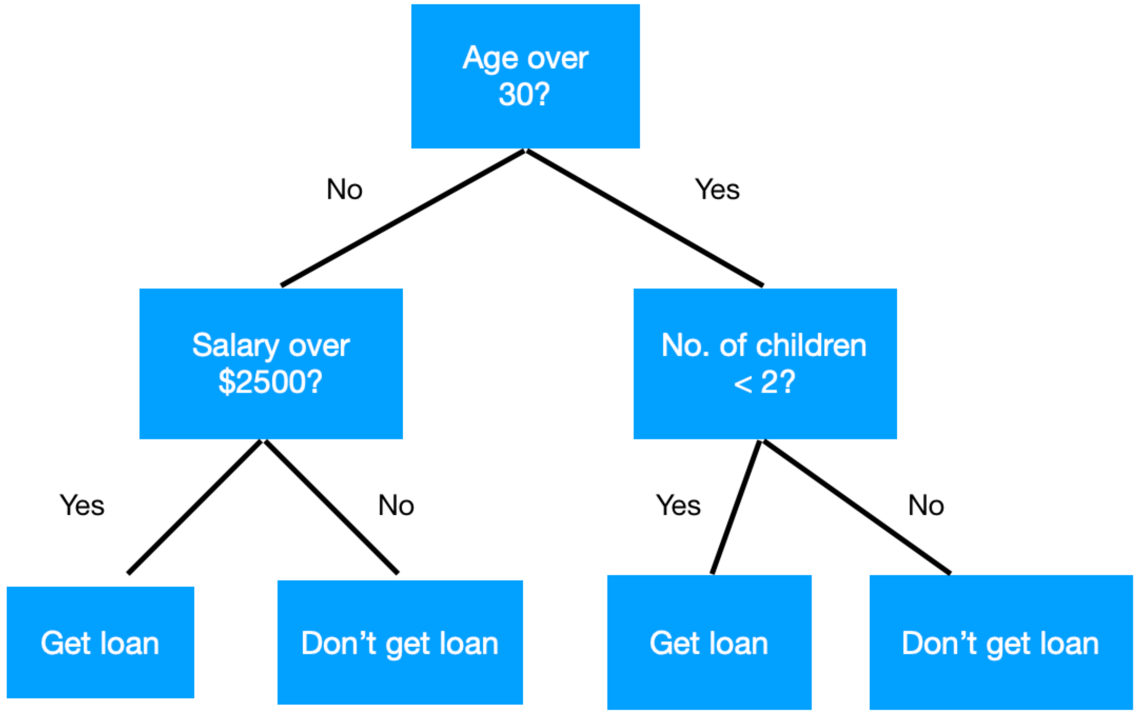
Regression Trees: Another Example (Vis)



Similarly: The Classification Trees

- For classification: the leaves contain a distribution over the class **labels**, rather than just the mean response.





Model Fitting for CART

- Model fitting is essentially – loss minimization.
  - What is the Loss?
  - How to Minimize the Loss?
- To fit the CART model, we need to minimize the loss:

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} \ell\big(y_n, f(x_n;\theta)\big) = \sum_{j=1}^{J} \sum_{x_n \in R_j} \ell\big(y_n, w_j\big)$$

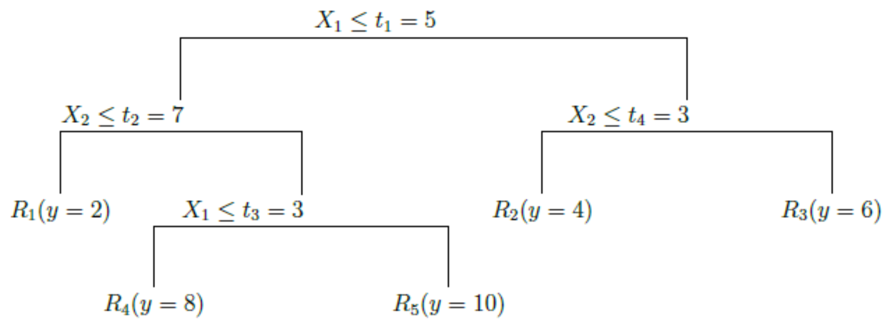
  - Any problem with this loss (or specifically, in minimizing this loss)?
  - It is not differentiable, due to the need to learn discrete tree structure!
  - Note: finding the optimal partitioning of the data is NP-complete. How?

Model Fitting for CART

- Loss to optimize: $\mathcal{L}(\theta) = \sum_{n=1}^{N} \ell\big(y_n, f(x_n;\theta)\big) = \sum_{j=1}^{J} \sum_{x_n \in R_j} \ell\big(y_n, w_j\big)$.
- The idea for model fitting for CART: using a greedy procedure, grow the tree iteratively, one node at a time.
  - Suppose we are at node i;
  - Let $\mathcal{D}_i = \{(x_n, y_n) \in N_i\}$ be the set of examples that reach this node.
  - Subsequent question: how to split this node into left and right branches to minimize the error in EACH child subtree?
  - Case 1: the j-th feature is a real-valued scalar → partition the data at node i by comparing to threshold t. The set of *possible thresholds* $\mathcal{T}_j$ for feature j obtained by sorting unique values.
  - E.g.,, feature 1 has values $\{4, -10, 70, -10\}$, set $\mathcal{T}_1 = \{-10, 4, 70\}$. For each possible threshold, left and right splits defined as $\mathcal{D}_i^L(j,t) = \{(x_n, y_n) \in N_i : x_{n,j} \le t\}$ and $\mathcal{D}_i^R(j,t) = \{(x_n, y_n) \in N_i : x_{n,j} > t\}$.

Model Fitting for CART: Node Splitting

- Loss to optimize: $\mathcal{L}(\theta) = \sum_{n=1}^{N} \ell\big(y_n, f(x_n;\theta)\big) = \sum_{j=1}^{J} \sum_{x_n \in R_j} \ell(y_n, w_j)$
- Greed model fitting via iterative node split:
  - Case 2: the j-th feature is categorical, with K_j possible values → partition the data at node i by checking if the feature is equal to each of those possible values or not.
  - This defines a set of K_j possible binary splits: $\mathcal{D}_i^L(j,t) = \{(x_n, y_n) \in N_i: x_{n,j} = t\}$ and $\mathcal{D}_i^R(j,t) = \{(x_n, y_n) \in N_i: x_{n,j} \neq t\}$.
  - Alternatively we could allow for a multi-way split.
  - This may cause **data fragmentation** – too little data might "fall" into each subtree. What kind of danger would data fragmentation lead to?
  - Overfitting!



Model Fitting for CART: the Optimal Model

- Greed model fitting via iterative node split:
  - Once we have computed $\mathcal{D}_i^L(j,t)$ and $\mathcal{D}_i^R(j,t)$ for each feature j and threshold t at node i, choose the best feature j_i to split on and the best threshold t_i as:

$$(j_i, t_i) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \frac{|\mathcal{D}_i^L(j,t)|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^L(j,t)\right) + \frac{|\mathcal{D}_i^R(j,t)|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^R(j,t)\right)$$

  - Can be viewed as choosing based on **averaged cost function** over split nodes.
  - $c(\mathcal{D}_i) = \frac{|\mathcal{D}_i^L(j,t)|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^L(j,t)\right) + \frac{|\mathcal{D}_i^R(j,t)|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^R(j,t)\right)$
  - So, what have we missed? The cost function $c(\mathcal{D}_i)$ to evaluate the cost of node i:
  - For regression: one typical example is the MSE

$$c(\mathcal{D}_i^*) = \frac{1}{|\mathcal{D}_i^*|} \sum_{n \in \mathcal{D}_i^*} (y_n - \bar{y})^2$$

  - Here $\bar{y}$ is the mean of the response varible for examples reaching node i.

Model Fitting for CART: the Overview

```
Algorithm: Regression Tree Learning
  Input: parameter $max\_depth$ & training set
  Output: Tree
1 root ← all training samples
2 for $d \leftarrow 1$ to $max\_depth$ do
3     for each leaf node $m$ at depth $d-1$ do
4         Find best feature & best threshold, so splitting
            node $m$ into two reduces MSE the most
5         Use decision rule to distribute training samples
            from node $m$ across two new leaf nodes
6 return tree
```

Model Fitting for CART: the Optimal Model

$$(j_i, t_i) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \frac{\left|\mathcal{D}_i^L(j,t)\right|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^L(j,t)\right) + \frac{\left|\mathcal{D}_i^R(j,t)\right|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^R(j,t)\right)$$

- For classification: compute the empirical distribution over class labels for node i: $\hat{\pi}_{ic} = \frac{1}{|\mathcal{D}_i|} \sum_{n \in \mathcal{D}_i} \mathbb{I}(y_n = c)$, which then lead us to the **Gini index** as:

$$G_i = \sum_{c=1}^{C} \hat{\pi}_{ic}(1 - \hat{\pi}_{ic}) = \sum_c \hat{\pi}_{ic} - \sum_c \hat{\pi}_{ic}^2 = 1 - \sum_c \hat{\pi}_{ic}^2$$

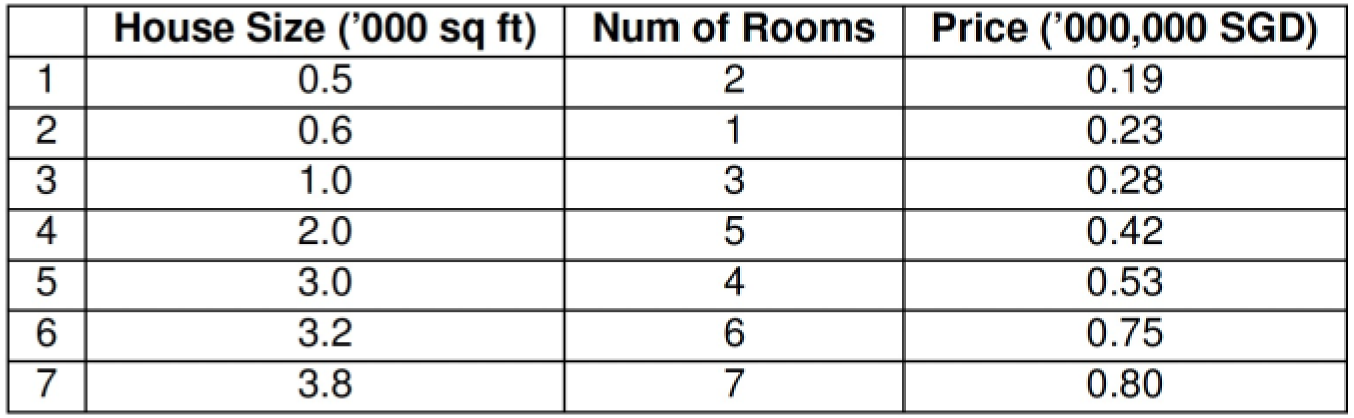
- The Gini index shows the expected error rate. $\hat{\pi}_{ic}$ is the probability of a **random** entry in the leaf belongs to class c and $1 - \hat{\pi}_{ic}$ is the probability it would be misclassified.
- Alternatively, define cost as the **entropy** or **deviance** of the node:

$$H_i = \mathbb{H}(\hat{\boldsymbol{\pi}}_i) = -\sum_{c=1}^{C} \hat{\pi}_{ic} \log \hat{\pi}_{ic}$$

- A node that is **pure** (only example of 1 class) will have ZERO enropy

CART: A Numerical Example

- Consider house prices in SG. Our target is to predict the variable price P.
  - The attributes (input) are house size S and number of rooms R, data as follows:



	House Size ('000 sq ft)	Num of Rooms	Price ('000,000 SGD)
1	0.5	2	0.19
2	0.6	1	0.23
3	1.0	3	0.28
4	2.0	5	0.42
5	3.0	4	0.53
6	3.2	6	0.75
7	3.8	7	0.80

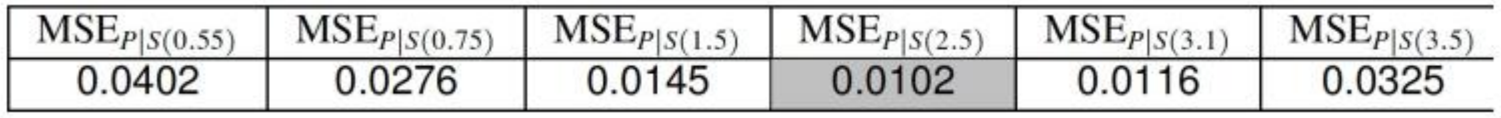
  - Now, how to leverage CART to predict the price of a new house given its house size s and its number of rooms r? Suppose the CART is simple that it only contains one root and all the other nodes are leave nodes.
  - I.e., we only split once for the tree.
  - Question: under which and what criterion should we split the root?

CART: A Numerical Example

- First possibility: split via S. Suppose we set the threshold as $\tau = 0.75$.
  - The targets of the two nodes are $\{0.19, 0.23\}$ and $\{0.28,0.42,0.53,0.75,0.80\}$.
  - We leverage the MSE as the cost function, and we compute the cost at each node as:

$$c(\mathcal{D}_i^*) = MSE = \frac{1}{|\mathcal{D}_i^*|} \sum_{n \in \mathcal{D}_i^*} (y_n - \bar{y})^2$$

  - Therefore: $c(0.75^L) = 4 \times 10^{-4}$, $c(0.75^R) = 0.0385$
  - $c(0.75) = \frac{2}{7} c(0.75^L) + \frac{5}{7} c(0.75^R) = 0.0276$
  - We can sweep through some possible thresholds and their corresponding costs are:

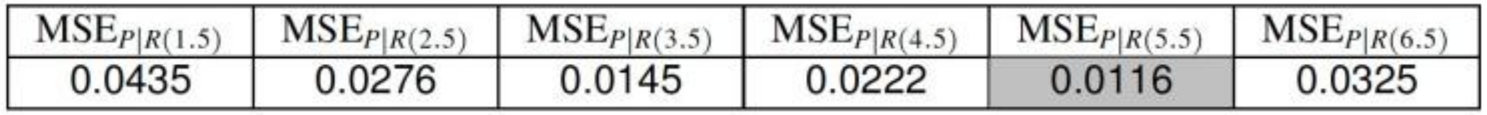


$\text{MSE}_{P\|S(0.55)}$	$\text{MSE}_{P\|S(0.75)}$	$\text{MSE}_{P\|S(1.5)}$	$\text{MSE}_{P\|S(2.5)}$	$\text{MSE}_{P\|S(3.1)}$	$\text{MSE}_{P\|S(3.5)}$
0.0402	0.0276	0.0145	0.0102	0.0116	0.0325

  - The best possible threshold τ for attribute S would be $\tau = 2.5$.

CART: A Numerical Example

- Second possibility: split via R. Similar to how we analyze the split via S, we can obtain a set of cost function values w.r.t. different thresholds τ.



$\text{MSE}_{P\|R(1.5)}$	$\text{MSE}_{P\|R(2.5)}$	$\text{MSE}_{P\|R(3.5)}$	$\text{MSE}_{P\|R(4.5)}$	$\text{MSE}_{P\|R(5.5)}$	$\text{MSE}_{P\|R(6.5)}$
0.0435	0.0276	0.0145	0.0222	0.0116	0.0325

  - We can view that the least cost function is obtained by splitting via S (space) with a threshold of $\tau = 2.5$. The left branch is where $S < 2.5$ and the right branch is where $S \geq 2.5$.
  - Therefore, if a house size is of $s = 3.3$ and $r = 4$, the output (price of this house) with this regression tree is obtained by computing the average of all the house with $S \geq 2.5$:

$$p = \frac{(0.53 + 0.75 + 0.80)}{3} = 0.6933$$

CART: Handling the Special Cases

- What is the **extreme case** for trees?
  - If we let the tree become **deep** enough, it can achieve 0 error on the training set by partioning the input space into sufficiently small regions.
  - Result in overfitting!
- To prevent this, there are two main approaches.
  - 1. **Stop** the tree growing process according to some heuristic, such as having too few examples at a node, or reaching a maximum depth. (Early stop)
  - 2. Grow the tree to its maximum depth, where no more splits are possible, and then to **prune** it back, by merging split subtrees back into their parent. (Tree pruning)

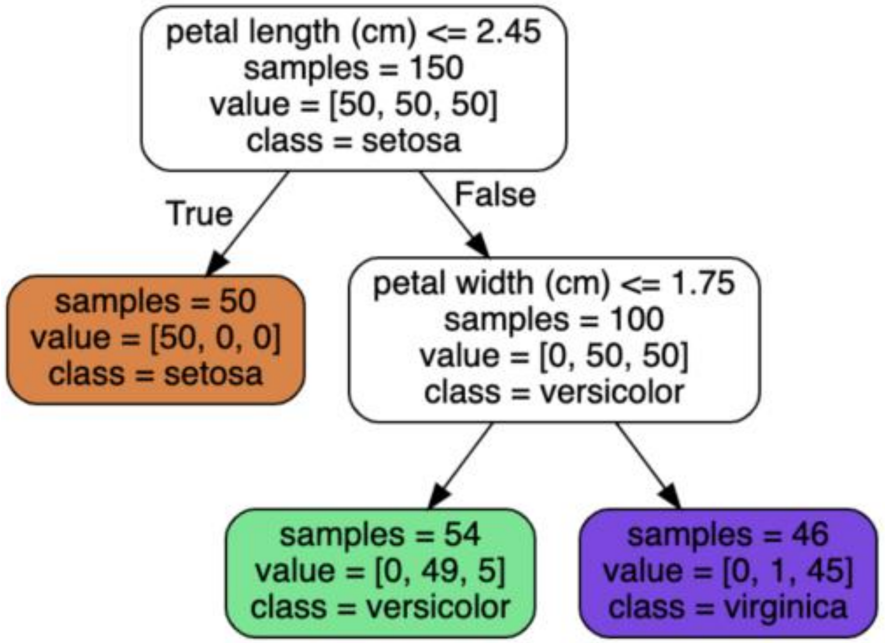
The Pros and Cons of CART / DT

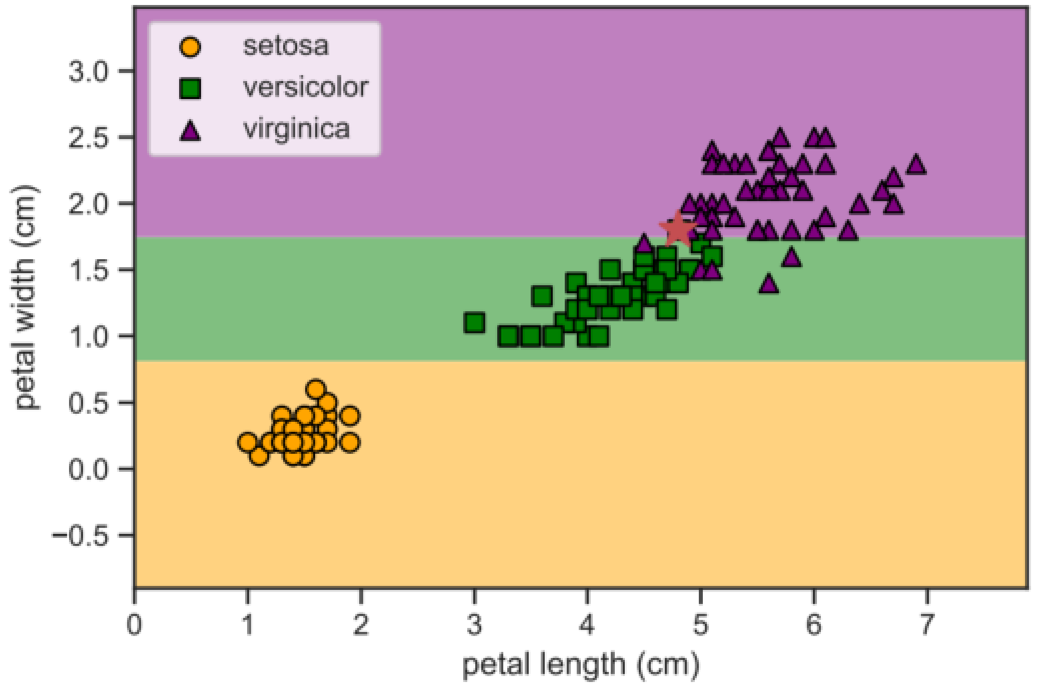
- Trees are very popular, especially in industrial applications!
  - Why? What are the characteristic that allows them to be applied in the industrial scenarios?
  - Think of what characteristics are more required in these scenarios.
  - Easy to interpret, thus fast to fit, and scale well to large data sets;
  - Easily handle mixed discrete and continuous inputs;
  - Insensitive to monotone transformations of the inputs, so there is no need to standardize the data.
  - Ability to perform automatic variable selection.
  - Relatively robust to outliers.

The Pros and Cons of CART / DT

- But, you may not see trees often in research or in SOTA:
- The primary disadvantage of trees is that they do not predict very **accurately** compared to other kinds of model.
  - In part due to the greedy nature of the tree construction.
  - Another side effect of greediness: trees are unstable: small changes to the input data can have large effects on the structure of the tree.
  - This is also related to the hierarchical nature of the tree-growing process, causing errors at the top to affect the rest of the tree.
  - Omitting even a single data point from the training set can result in a dramatically different decision surface
  - An example to illustrate the unstable tree.

Unstable DT: Example with the IRIS dataset





Can We Stabilize the DT?

- Decision trees (DTs) can be quite unstable, due to their predictions may vary a lot if the training data is perturbed.
  - I.e., decision trees are a high variance estimator.
  - Question: how to reduce the variance? (Note that we only care about the variance first, we do not care about the bias)
  - A simple way to reduce variance is to **combine multiple** models.
- This is called **ensemble learning**.
  - The ensemble will have **similar bias** to the base models, but lower variance, generally resulting in improved overall performance

Ensemble Learning: Combining Models

- When discuss about “combining” models:
  - Such discussion focus on the combination of the results of the models.
  - The combination for regressors and classifiers could be different, thanks to the different type of output (continuous vs. discrete)
  - For regression models, a sensible combination method is averaging:

$$f(y|x) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(y|x)$$

  - For classifiers, it can sometimes be better to take a majority vote of the outputs (committee method or voting method).
  - How does the voting works and why does voting improve accuracy?

Ensemble Learning: Voting

- Suppose each base model is a binary classifier with the same accuracy θ.
  - Suppose class 1 is ALWAYS the correct answer. $Y_m \in \{0,1\}$ be the prediction for the m-th model, and let $S = \sum_{m=1}^{M} Y_m$ be the number of votes for class 1.
  - For majority voting: the **final prediction would** be 1 if $S > M/2$. Thus the probability that ensemble will predict class 1 is computed:

$$p = P\left(S > \frac{M}{2}\right) = 1 - B(\frac{M}{2}, M, \theta)$$

  - Here $B(x, M, \theta)$ is the cdf of the binomial distribution with parameters M and θ evaluated at x. For $\theta = 0.51$ and $M = 1000$, the result is $p = 0.73$. By increasing $M = 10000$ the result is further improved to $p = 0.97$.
  - The performance of the voting approach is dramatically improved, because we assumed each predictor made independent errors. In practice, their mistakes may be correlated, but as long as we ensemble sufficiently diverse models, we can still come out ahead.

Bagging: an Introduction

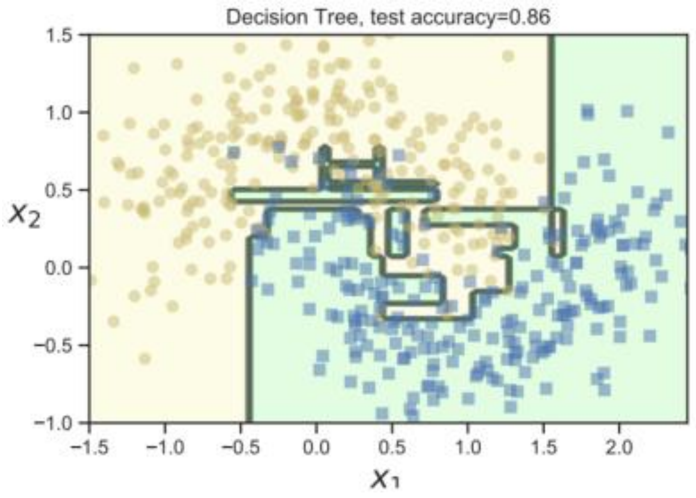
- The original ensemble learning (or called the vanilla ensemble learning) with averaging or voting is somehow **too intuitive**.
- It also does not make any significant limitations and/or guidance towards each individual model for ensembling.
- A “better” form of ensemble learning: bagging (bootstrap aggregating):
  - Fit M different base models to different randomly sampled versions of the data;
  - Encourages the different models to make diverse predictions;
  - The datasets are sampled with replacement (**bootstrap sampling**)
  - A given example may appear **multiple** times, until we have a total of N examples per model (N is the number of original data points).

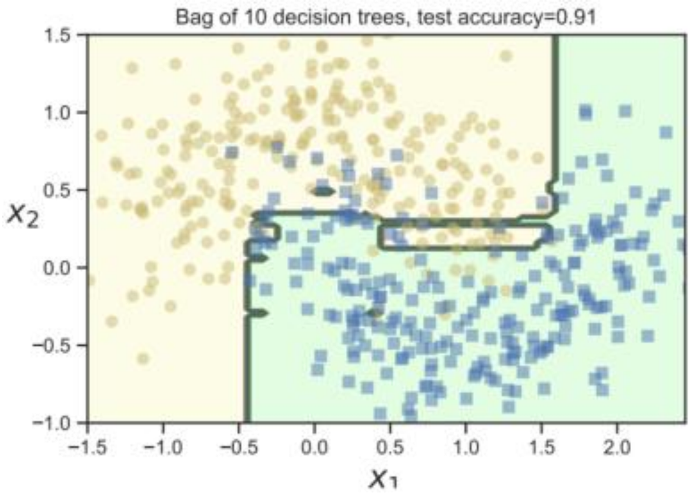
The Cons and Pros of Bagging

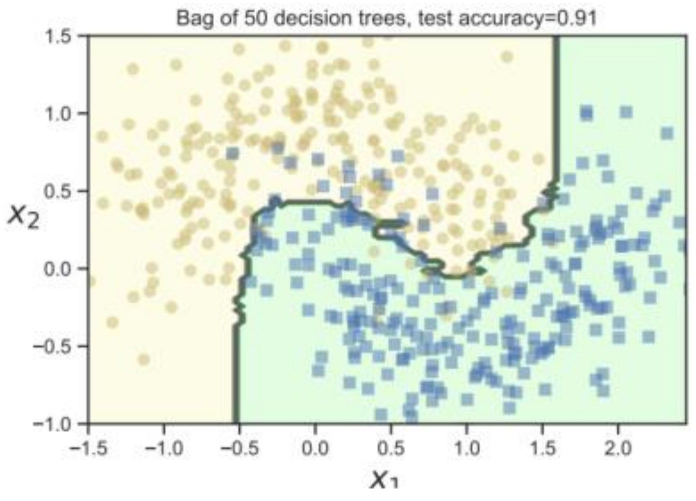
- Now that we know the general process of bagging – what could be the (dis-)advantages of bagging?
  - The key of bagging – each model sees sampled data;
  - Each base model only sees on average 63% of the unique input examples! (Why?)
  - The chance that a single term will **NOT** be selected from a set of size N in any of N draws is $\left(1-\frac{1}{N}\right)^{N}$. With large $N \rightarrow \infty$, this becomes $e^{-1} \approx 0.37$.
- This 37% of the training instances that are NOT used by a given base model are called **out-of-bag instances** (oob).
  - Can use the predicted performance of the base model on these oob instances as an estimate of **test set performance**.
  - This provides a useful alternative to cross validation

The Cons and Pros of Bagging

- Main advantage of bagging: prevents the ensemble from relying too much on any individual training data, enhances robustness and generalization.
  - BUT, bagging does not always improve performance. In particular, it relies on the base models being unstable estimators, so that omitting some of the data significantly changes the resulting model fit (for trees)
  - Quite mixed for others, e.g., bagged DNNs do not usually work well as deep networks will underperform if they only see 63% of the data.







Decorrelate Base Models (more): Random Forests

- Bagging relies on the assumption that re-running the same learning algorithm on **different subsets** of the data will result in sufficiently **diverse base models**.
  - The key is: diverse base models. In other words, the models should not correlate with each other.
- Another technique that further decorrelate the base learners is known as **random forests**.
  - Learning trees based on a **randomly chosen** subset of input variables (at each node of the tree), as well as a **randomly chosen** subset of data cases.

Random Forests: Process and Example

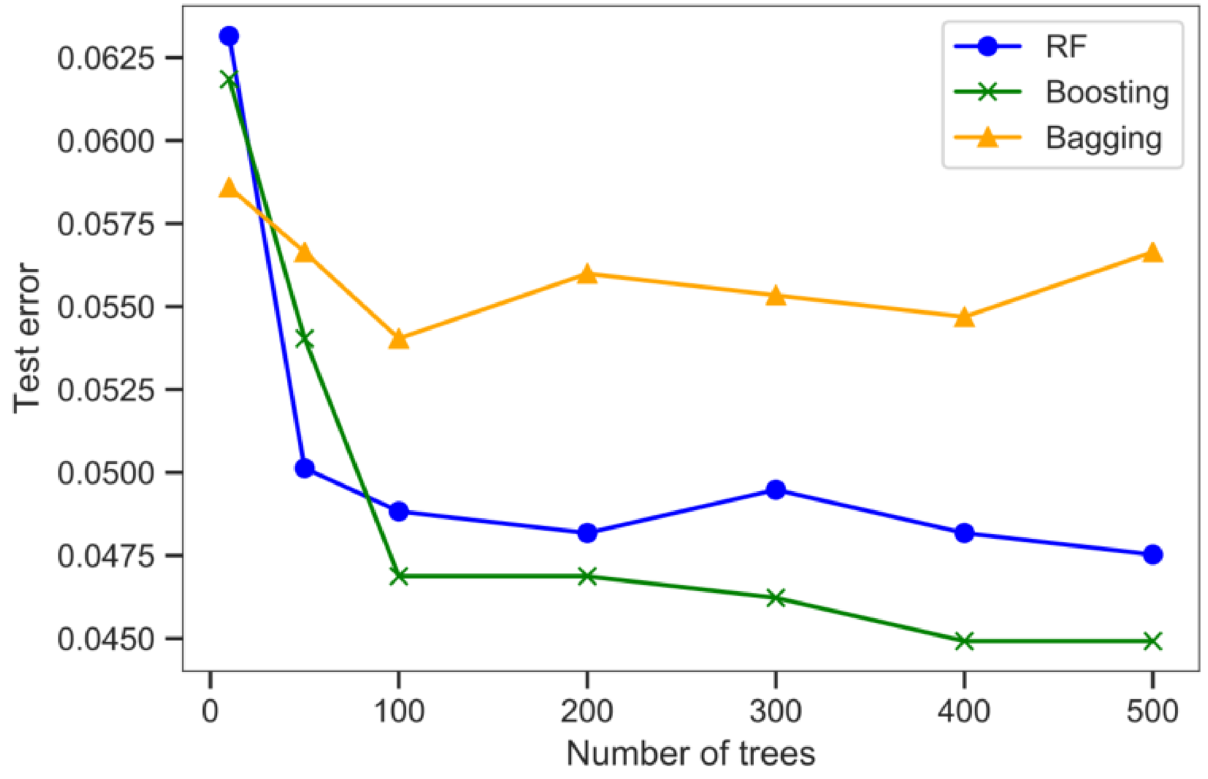
- The general process: modify the tree splitting equation so that the feature split dimension j is optimized over a random subset of the features.
- A simple recap of the tree splitting equation:

$$(j_i, t_i) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \frac{\left|\mathcal{D}_i^L(j,t)\right|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^L(j,t)\right) + \frac{\left|\mathcal{D}_i^R(j,t)\right|}{|\mathcal{D}_i|} c\left(\mathcal{D}_i^R(j,t)\right)$$

- E.g., for a email spam dataset, there are different types of features (57):
  - 48 features corresponding to the percentage of words in the email that match a given word, such as “remove” or “labs”;
  - 6 features corresponding to the percentage of characters in the email that match a given character;
  - 3 features corresponding to the average length, max length, and sum of lengths of uninterrupted sequences of capital letters.

Visualize the Performance of RF vs. Bagging

- RFs work much better than bagged decision trees, because many input features are irrelevant.


RF
Boosting
Bagging
Test error
Number of trees