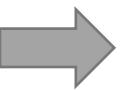


MACHINE LEARNING IN FINANCE



Big picture: Econometrics vs Machine Learning



What are we trying to do as a researcher?



Solve real world problems, right?

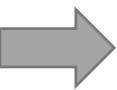


Is there a theory?

Yes → Start Modelling with Theory.
No → Start with Data

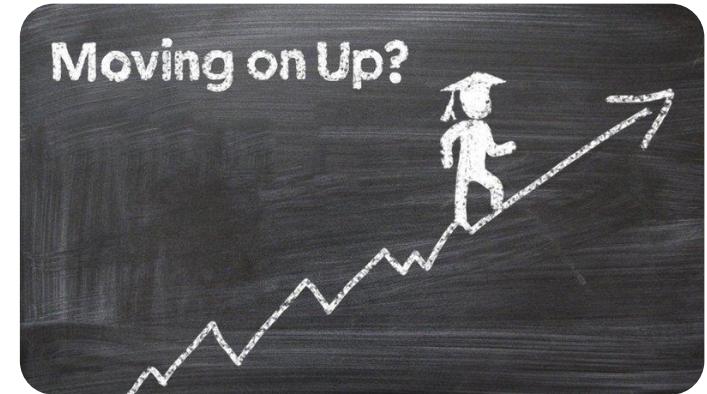
What is the relationship between

- Sales and advertisement / R&D expenditure / seasonality / industry / ... ? *No Theory, start with Data.*
- Quantity demanded and price / income / technology / price of competitors / ... ? *Theory Exists*
- Wage and education/ age/ gender/ experience/ ...?



A simple example

- Quantifying wage components! (is there a theory?) *No*
- What are the drivers:
 - Education, age, experience, IQ, ...
 - Ethnicity, race, gender, ...
 - Industry, location, working hours, ...
- Let's build a model (**assuming** a linear functional form!) (*Econometrics way*)

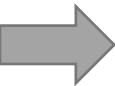


$$wage = \beta_0 + \beta_1 educ + \beta_2 age + \beta_3 exper + \beta_4 IQ + \cdots + \beta_k hours + u$$

Yes. With $\beta_1 = 0.5$, If Education increases by 1 year your hourly wage increases by 50 pence.

- Can you **interpret** this model? Do you care about the interpretability? *Yes*
- Can you make **predictions** using your model? *Yes*
- Can you make this functional form more flexible? What are the caveats?

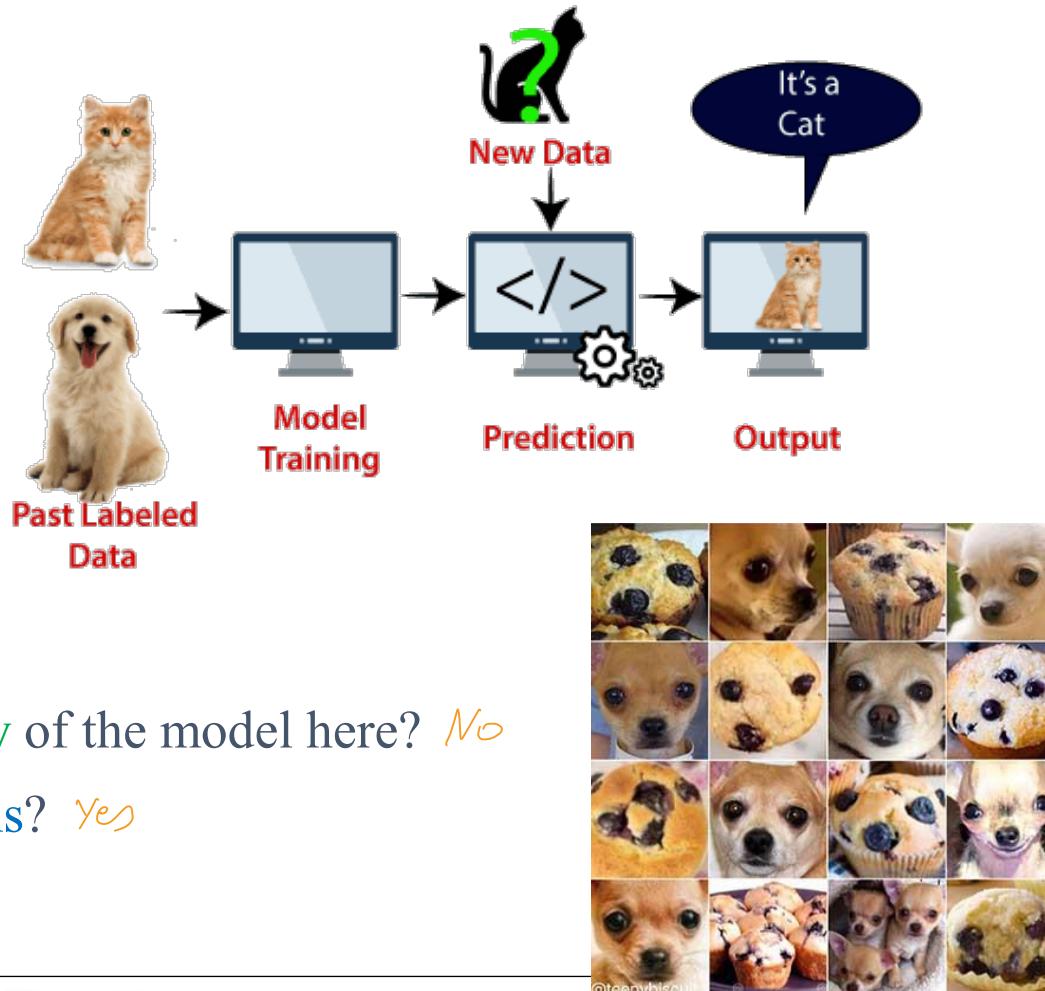
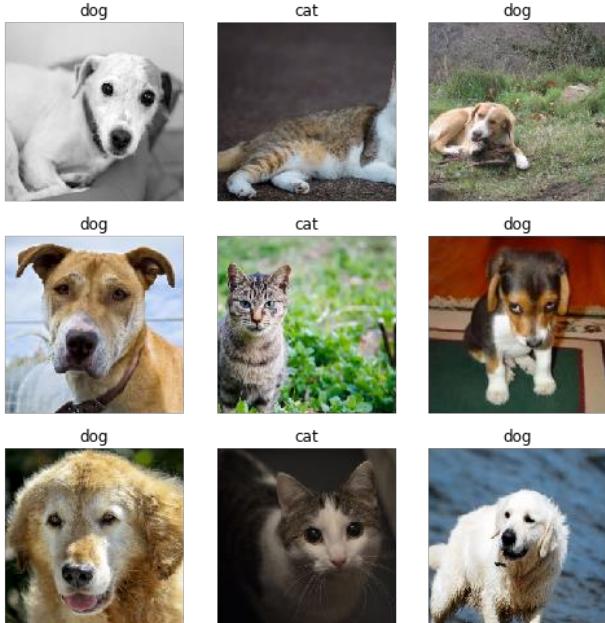
*Adding more flexibility will result in loss of interpretability
So in Econometric we stick with simple models.*



A different example

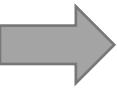
Econometric can only handle **Structural** data.
The image data is **Non-structural** (can't be presented in a table).

- Cat vs dog classification problem (image classification)



- Do you really care about **interpretability** of the model here? *No*
- What about accuracy of your predictions? *Yes*





Artificial intelligence vs Machine learning vs Deep learning

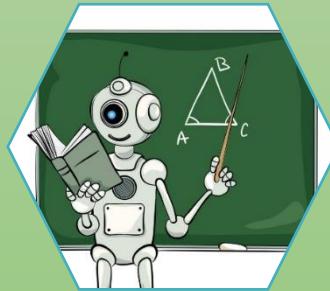
Artificial intelligence: Any technique which enables machines to mimic human behavior

Machine Learning: Subset of AI that enables computers to learn from data. the model is trained with a set of algorithms

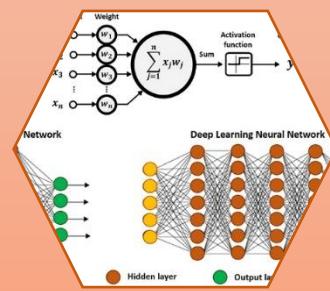
Deep Learning: Subset of ML that extract patterns from data using neural networks.



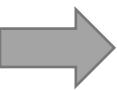
1950's



1980's

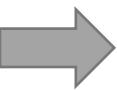


2010's



Statistical learning vs machine learning

	Statistical Learning	Machine Learning / Deep Learning
Focus	Hypothesis testing & interpretability	Predictive accuracy and extracting complex patterns
Driver	Math, theory, hypothesis	Fitting data
Data size	Any reasonable set	Big data
Data type	Structured	Structured, unstructured, semi-structured
Dimensions / scalability	Mostly low dimensional data	High dimensional data
Strength	Understand causal relationship & behavior	Prediction (forecasting and nowcasting)
Interpretability	High	Medium to Low



Limitations of Econometrics/Structured ML

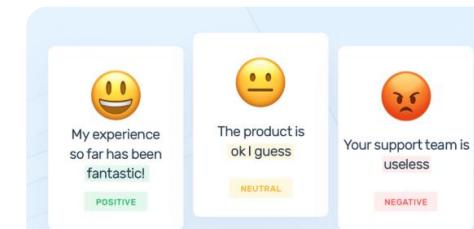
Econometrics/structured ML can only handle structured data (tabular data)!

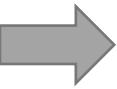
Structured Data

	A	B	C	D
1	Date	Account	Transaction Type	Amount
2	2017-01-12	123	Credit	6089.78
3	2017-01-12	123	Fee	9.99
4	2017-01-12	456	Debit	1997
5	2017-01-12	123	Debit	20996.12
6	2017-01-13	123	Debit	17
7	2017-01-13	123	Debit	914.36
8	2017-01-14	789	Credit	11314
9	2017-01-14	789	Fee	9.99
10	2017-01-14	456	Debit	15247.89
11	2017-01-14	123	Debit	671.28
12	2017-01-15	456	Credit	5072.1
13	2017-01-15	456	Fee	9.99
14	2017-01-16	456	Debit	5109.07
15	2017-01-19	123	Credit	482.01



Unstructured Data
(everything else!!)





A more complex example

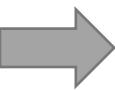
Stock price prediction \$\$\$

- What are the classical drivers:
 - Company's fundamentals (balance sheet, income statement, cash flow statement)
 - Competitors (comparing multiples)
 - Technical analysis!
 - Seasonality (holidays, months, days, ...)



What else?

- Market sentiment (news, tweets, blogger opinions, conference calls, ...)
- Satellite images from parking lots!

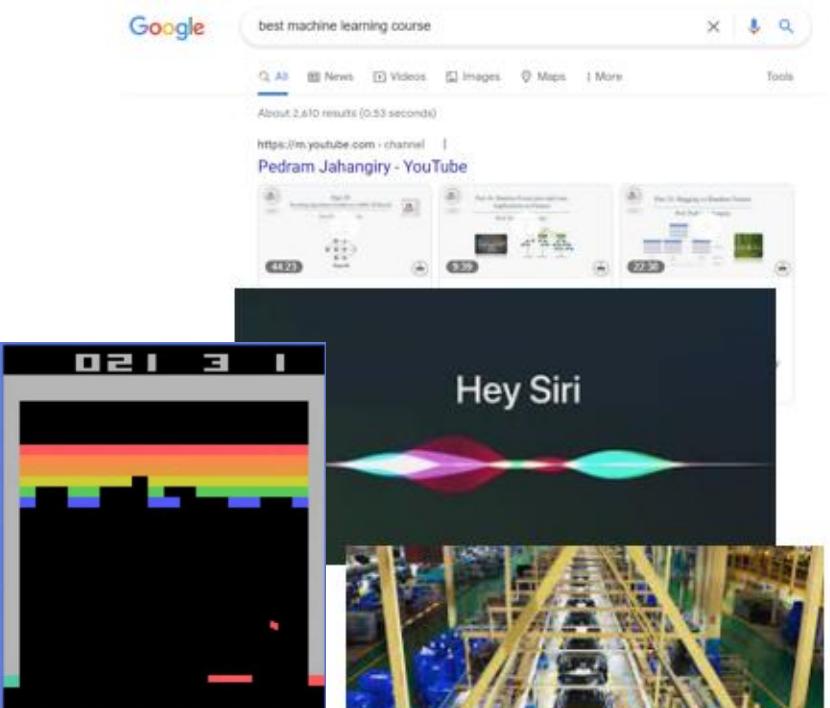


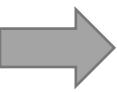
Why should I learn it?

- It's a bid deal, it is **everywhere!**
- Better career opportunities
- Hedge against next recession



 OpenAI





Class Modules

- Module 1- Introduction to Machine Learning
- Module 2- Setting up Machine Learning Environment
- Module 3- Linear Regression (Econometrics approach)
- Module 4- Machine Learning Fundamentals
- Module 5- Linear Regression (Machine Learning approach)
- Module 6- Penalized Regression (Ridge, LASSO, Elastic Net)
- Module 7- Logistic Regression
- Module 8- K-Nearest Neighbors (KNN)
- Module 9- Classification and Regression Trees (CART)
- Module 10- Bagging and Boosting
- Module 11- Dimensionality Reduction (PCA)
- Module 12- Clustering (KMeans – Hierarchical)



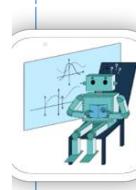
Module 1- Part II

What is Machine Learning?



Supervised

- Regression
- Classification



Unsupervised

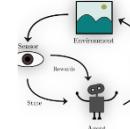
- Clustering
- Anomaly detection
- Dimensionality reduction



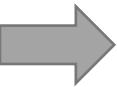
Semi-supervised



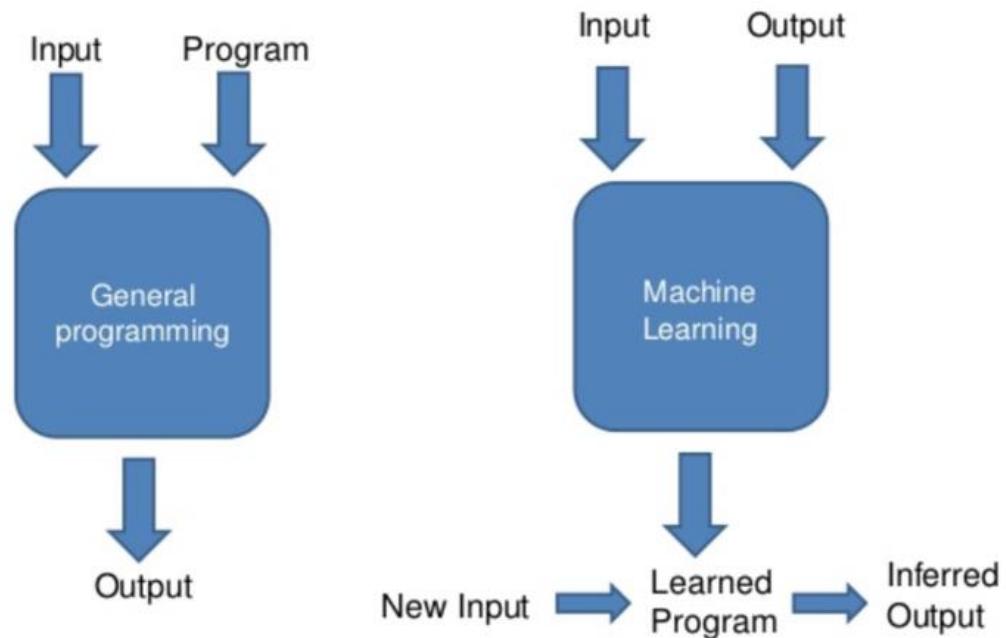
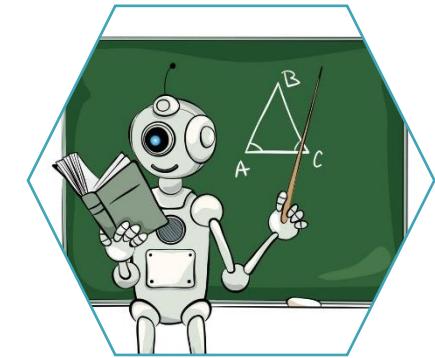
Self-supervised



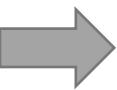
Reinforcement Learning



General programming vs Machine learning

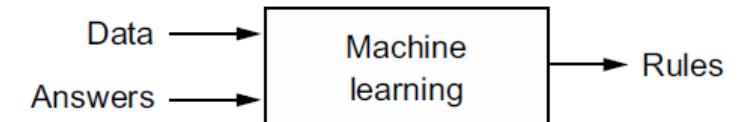
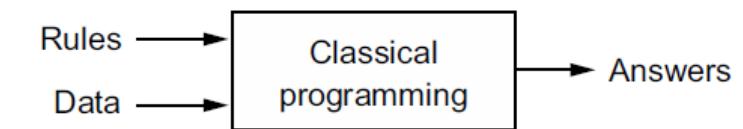


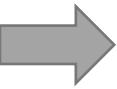
- Machine Learning: Involves **automated detection of meaningful patterns** in data and apply the pattern



What is Machine Learning?

- A machine learning system is **trained** (with algorithms) rather than explicitly **programmed**.
- Machine Learning is a subset of AI that enables computers to **learn** from data.
- ML involves automated detection of meaningful **patterns** in data and apply the pattern to make **predictions** on **unseen data!**
- This is done by **minimizing** the loss on the training data.
- The goal is to **maximize** the performance on the unseen data.
- ML involves splitting the dataset into 3 distinct subsets:
 - (i) Training dataset (70%)
 - (ii) Validation dataset (10-15%)
 - (iii) Test dataset. (10-15%)





Artificial intelligence vs Machine learning vs Deep learning

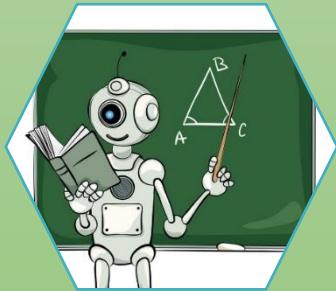
Artificial intelligence: Any technique which enables machines to mimic human behavior

Machine Learning: Subset of AI that enables computers to learn from data. the model is trained with a set of algorithms

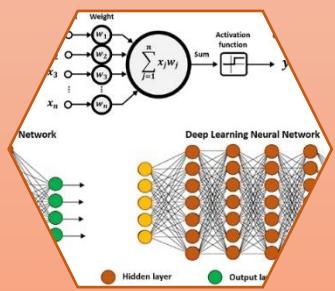
Deep Learning: Subset of ML that extract patterns from data using neural networks.



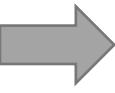
1950's



1980's



2010's

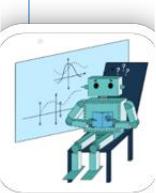


Types of Machine Learning



Supervised

- Regression
- Classification



Unsupervised

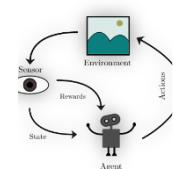
- Clustering
- Anomaly detection
- Dimensionality reduction



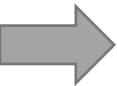
Semi-supervised



Self-supervised



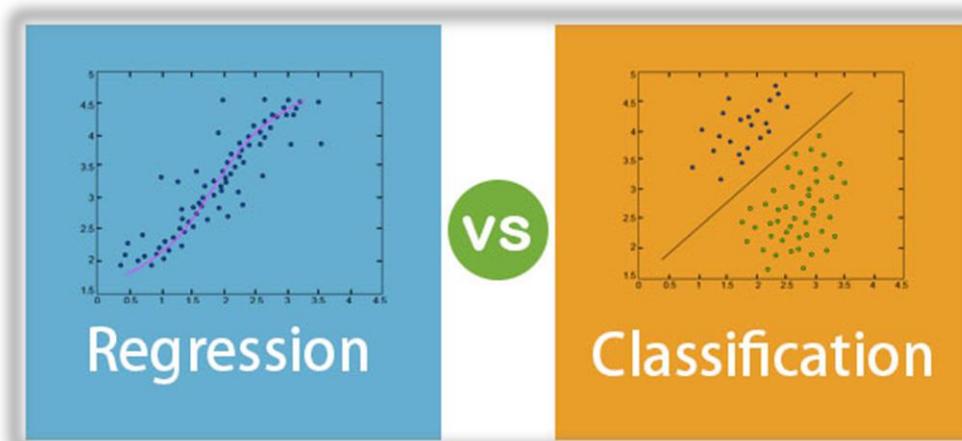
Reinforcement Learning



Supervised Learning

(Data contains both input & output)

- Supervised learning is a type of machine learning where the algorithm is **trained on labeled data**.
 - The data is labeled, meaning that the data has been **tagged with the correct output**.
 - The model is then able to learn **the relationship** between the input data and the corresponding output labels and can make predictions on new data.
-
- **Regression:**
 - Predicting housing price
 - Predicting stock market returns
 - **Classification:**
 - Generating buy, sell, hold signals.
 - Predicting credit default rate.



- If Target Variable (output) is:
 - (i) Continuous Random Variable → Do Regression.
 - (ii) Categorical or Binary Variable or class → Do Classification.

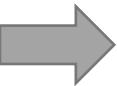
Class exercise

- Can you think of an example of Regression? Classification?

Regression : (1) Predicting Wages.

(2) Predicting Revenue growth.

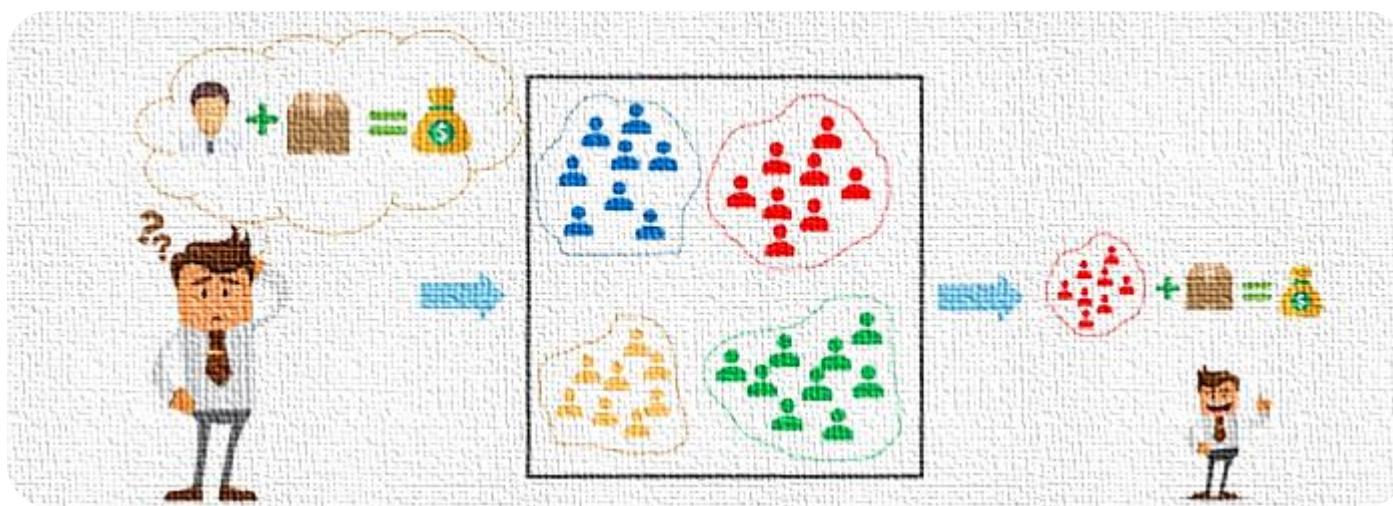
Classification: (1) Likelihood of a successful M&A or IPO/SPAC.
(2) Enhancing Detection of Fraud in financial statements.
(3) Classification on winning & losing funds or ETF's.

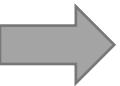


Unsupervised Learning

(Data contains only input)

- Unsupervised learning is a type of machine learning where the algorithm is **not given any labeled** training data.
- The goal is to discover the **underlying patterns** and find groups of samples that behave similarly. **Find something interesting!**
- **Clustering:** group similar data points together
 - ✓ Mall customer segmentation
 - ✓ Client profiling and asset allocation





Unsupervised Learning

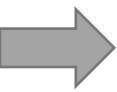
$x = \text{Explanatory Variables} = \text{features}_{\text{space}}$
 $y = \text{Response Variable} = \text{Target variable}$

- Unsupervised learning is a type of machine learning where the algorithm is **not given any labeled** training data.
- **Anomaly detection:** Find anomalies (unusual data points)
 - ✓ Fraud detection
- **Dimensionality Reduction:** compress data in lower dimension
 - ✓ Identify the most predictive factors underlying asset price models.



Reduce feature Space (PCA: Principle Component Analysis)



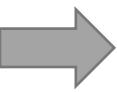


Class exercise

- Can you think of an example of Clustering? Anomaly detection?

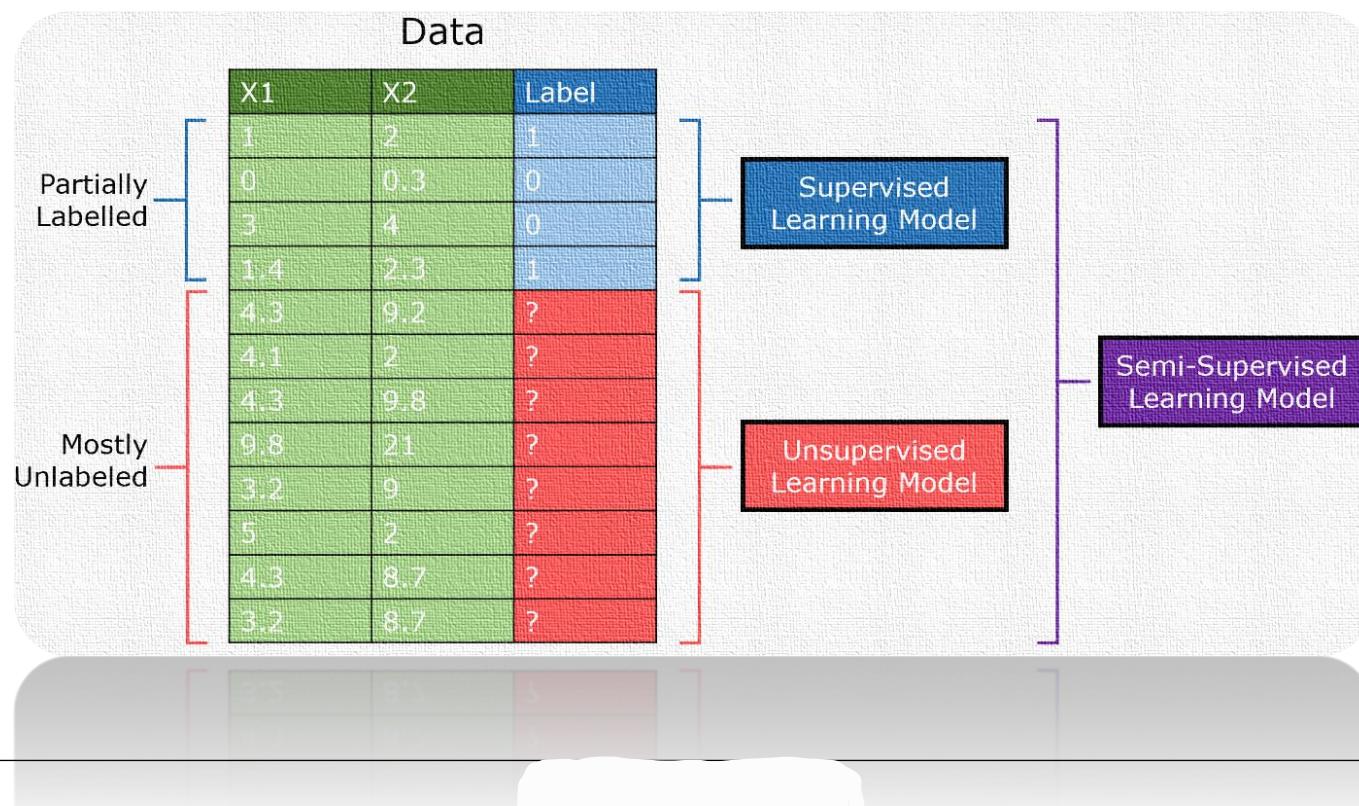
Clustering:

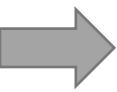
- ① Grouping companies into peer groups based on some non-standard characteristics like financial statement data or corporate characteristics rather than sectors or countries.
- ② Portfolio diversification & stock selection based on co-movements similarities.



Semi-Supervised

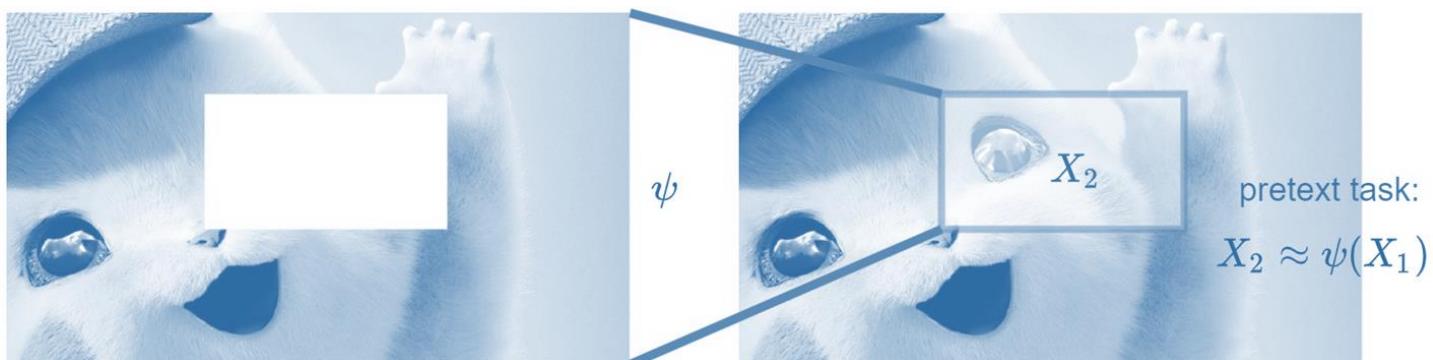
- Semi-supervised learning is a **combination** of supervised and unsupervised learning.
- It's used when you have a large dataset with some **labeled** examples and many **unlabeled** examples.
- The goal is to use the labeled examples to learn a mapping from inputs to outputs, and then use that mapping to make predictions on the unlabeled examples (**pseudo labels**)

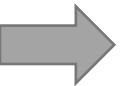




Self-Supervised

- Self-supervised learning is a type of **unsupervised** learning, but it has the property that the learning process is being fed with the data itself (and not a human annotation).
- The **goal** of self-supervised learning is learning useful **representations** from the data (representation learning)
- The model learns a representation of the data by predicting properties of the **input data itself**.
- Example:
 - Predicting missing part of an input (text, image, ...)



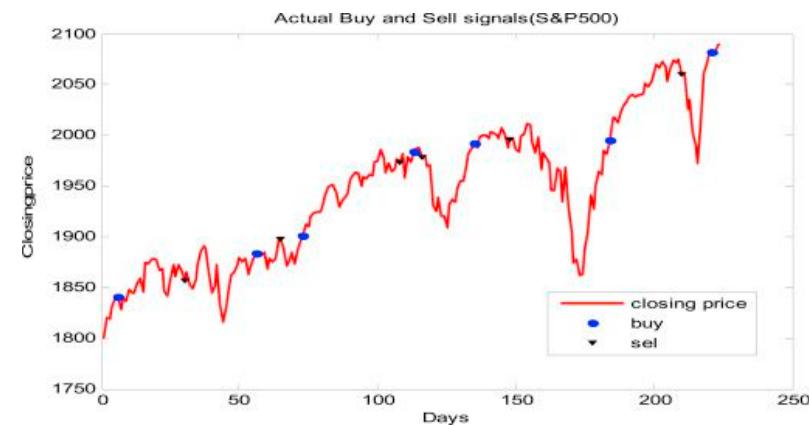
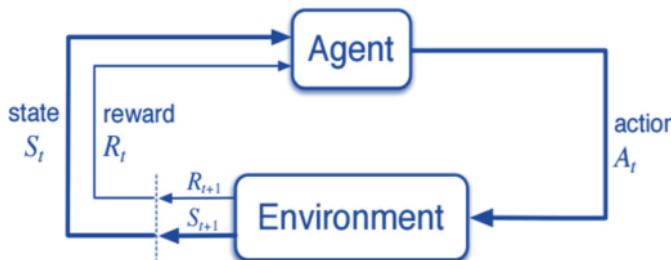


Reinforcement Learning

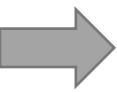
Computers



- Reinforcement learning is a type of machine learning where an **agent** learns to interact with its **environment** in order to maximize a **reward**.
- The agent receives rewards for performing actions that lead to successful outcomes and learns to repeat successful actions and avoid unsuccessful ones. (**Explore** and **Exploit**)

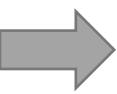


- Example: a virtual trader (**agent**) who follows certain trading rules (**actions**) in a specific market (**environment**) to maximize its profits (**reward**).



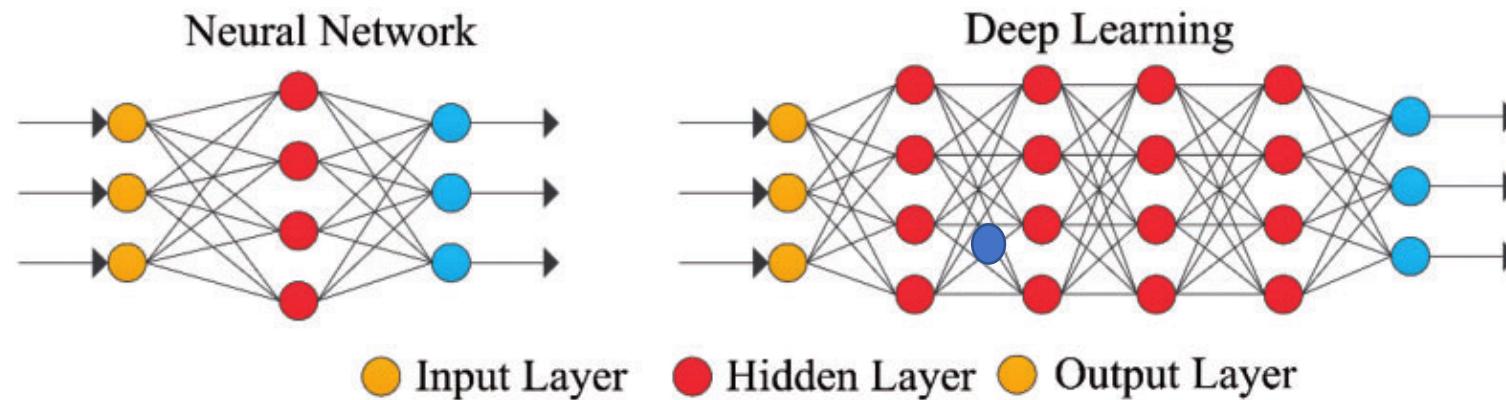
Class exercise

- Can you think of an example of reinforcement learning?
 - AI Games.
 - Self-driving Car.
 - Skill Acquisition.
 - Learning tasks.
 - Robot Navigation.
 - Real-time decisions.
- Reinforcement learning hasn't made promising lead in Finance Sector so far.

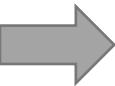


Deep Learning

Deep learning is a subset of machine learning where artificial neural networks, algorithms inspired by the human brain, learn from large amounts of data.



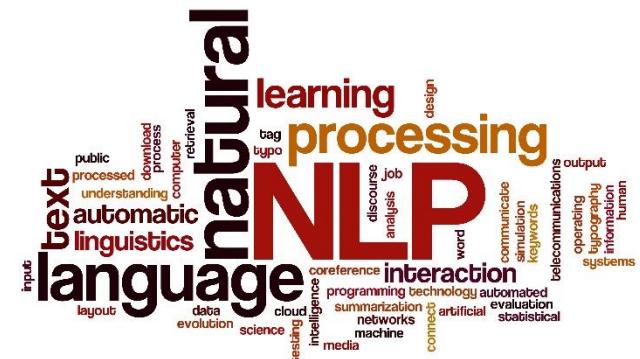
- Examples:
 1. Image recognition algorithms can now analyze data from satellite-imaging systems to provide intelligence on the number of consumers in retail store parking lots,
 2. Shipping activity and manufacturing facilities, and
 3. Yields on agricultural crops

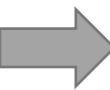


Natural Language Processing (NLP)

NLP is a field of research at the intersection of **computer science**, **artificial intelligence**, and **linguistics** that focuses on developing computer programs to analyze and interpret **human language**.

- Automated tasks using NLP include translation, speech recognition, text mining, sentiment analysis, and topic analysis.
- Examples:
 - Reading millions of pages of annual reports, thousands of hours of earning calls, transcripts, news articles and social media posts to identify trend in shorter timespans!
 - Analyzing communications and transcripts from policymakers (FED, ECB, ...) to provide insights around trending topics like interest rate policy, GDP, inflation expectation and etc
 - Chatbots answer basic retirement savings questions, learning from their interactions with investors.





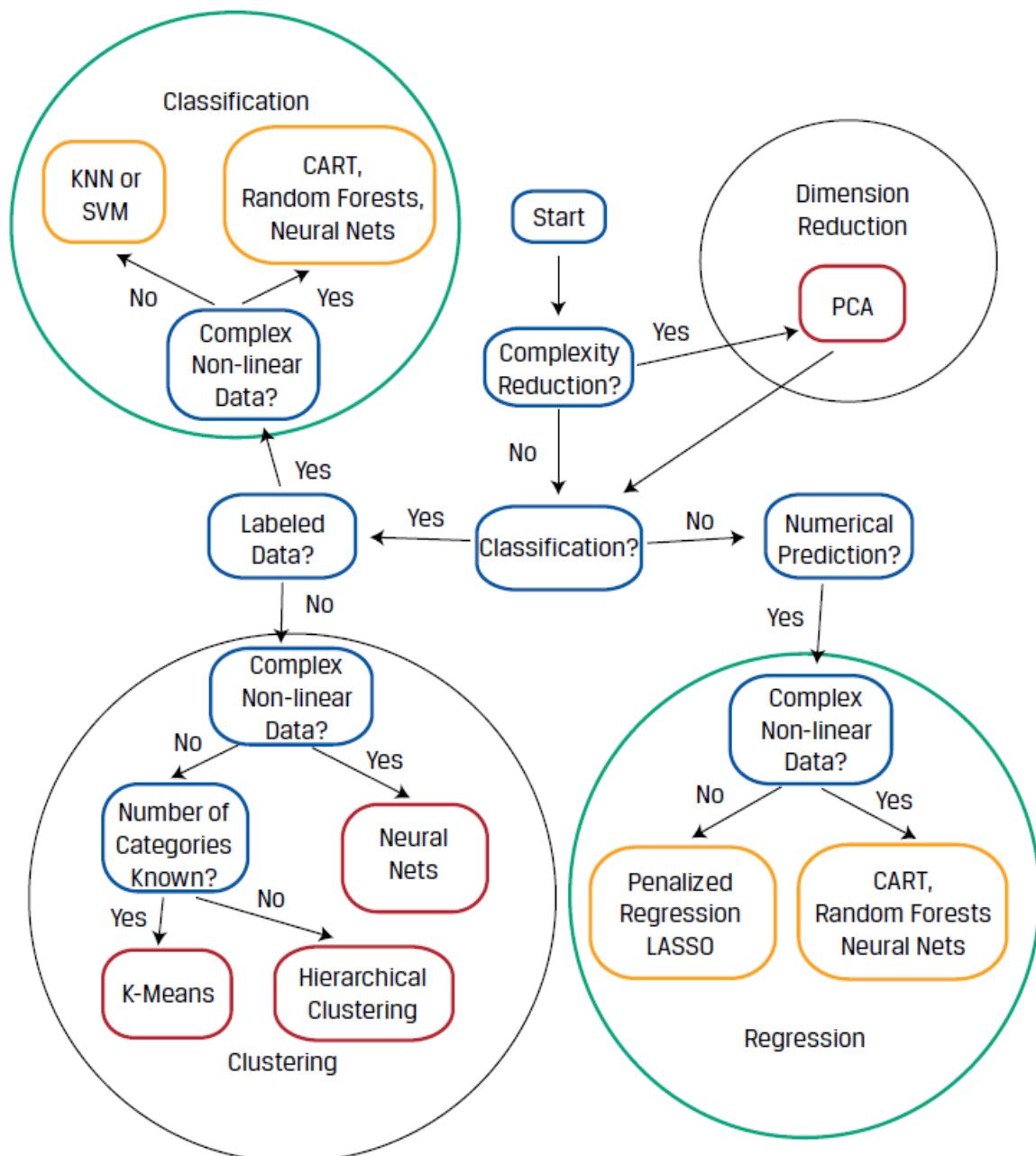
Cognitive computing

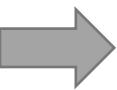
- **Cognitive Computing** focuses on **mimicking** human behavior and reasoning to solve complex problems.
- **Cognitive Computing** is **not responsible for making the decision** for humans. They simply supplement information for humans to make decisions.

- **Algo trading:** Algorithmic trading is the computerized buying and selling of financial instruments, in accordance with pre-specified rules and guidelines.
- **High-frequency trading (HFT)** is a form of algorithmic trading that makes use of vast quantities of granular financial data (tick data, for example) to automatically place trades when certain conditions are met. Trades are executed on ultra-high-speed, low-latency networks in fractions of a second. HFT algorithms decide what to buy or sell and where to execute based on real-time prices and market conditions, seeking to earn a profit from intraday market mispricing.

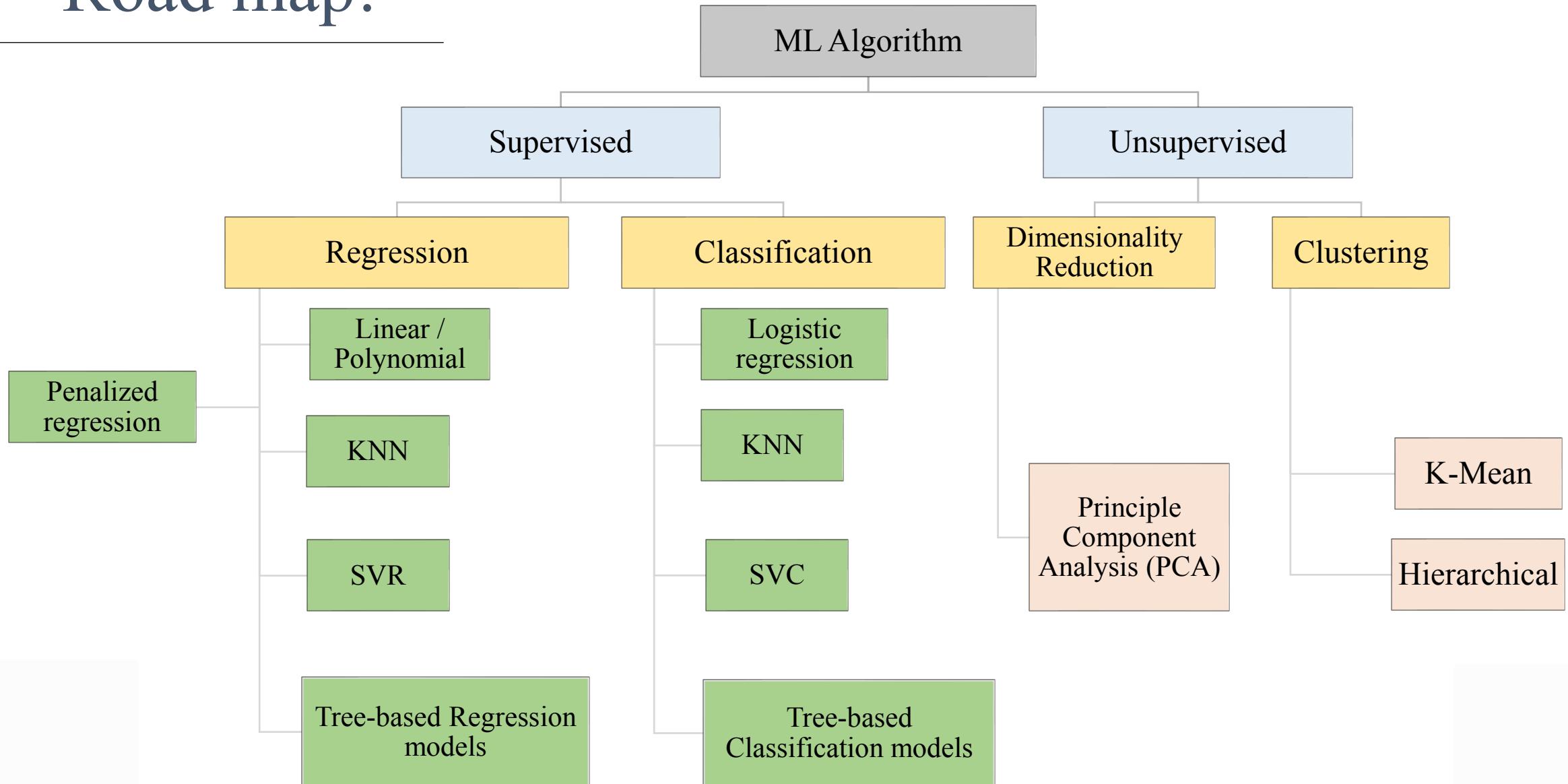


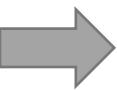
Exhibit 37 Stylized Decision Flowchart for Choosing ML Algorithms





Road map!

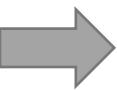




What's on GitHub

- ✓ Module 1- Introduction to Machine Learning
 - Module 2- Setting up Machine Learning Environment
 - Module 3- Linear Regression (Econometrics approach)
 - Module 4- Machine Learning Fundamentals
 - Module 5- Linear Regression (Machine Learning approach)
 - Module 6- Penalized Regression (Ridge, LASSO, Elastic Net)
 - Module 7- Logistic Regression
 - Module 8- K-Nearest Neighbors (KNN)
 - Module 9- Classification and Regression Trees (CART)
 - Module 10- Bagging and Boosting
 - Module 11- Dimensionality Reduction (PCA)
 - Module 12- Clustering (KMeans – Hierarchical)





Having said that...

- **Warning:** A ML algorithm will always find a pattern, even if there is none.



If you torture the data long enough,
it will confess.

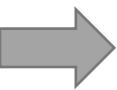
— *Ronald Coase* —

AZ QUOTES

Class 4- Machine Learning concepts

Part I

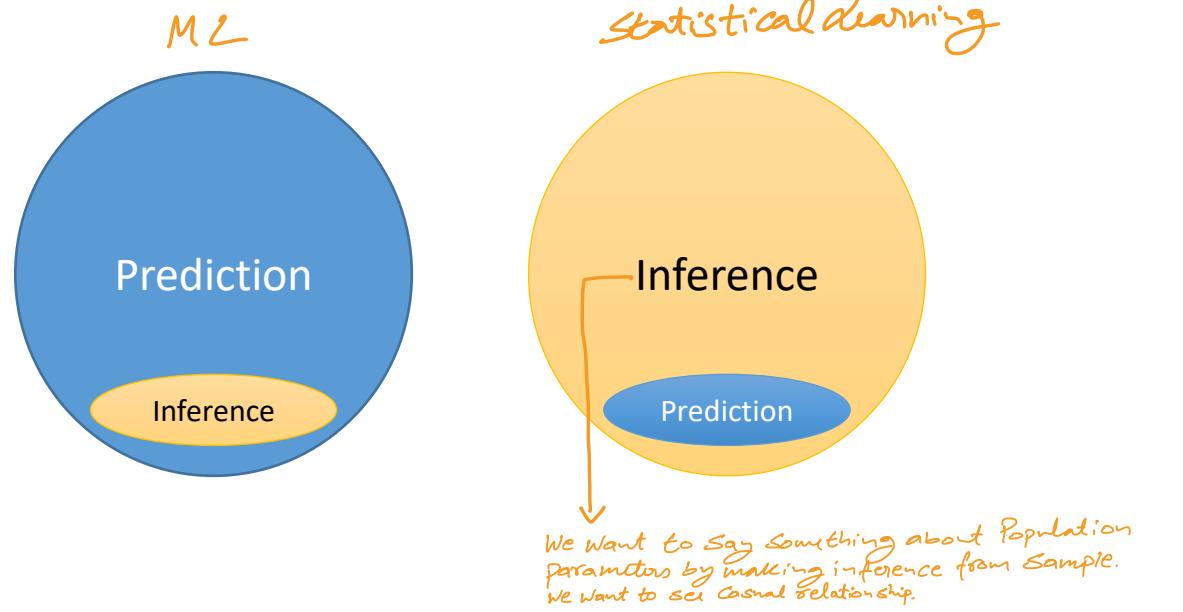




Motivation

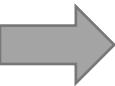
Machine learning fundamental concepts:

- Inference and prediction
- Part I: The Model
 - Parameters and hyperparameters
 - Parametric vs nonparametric ML models
- Part II: Evaluation metrics
- Part III: Bias-Variance tradeoff
- Part IV: Resampling methods
- Part V: How do machines learn?
- Part VI: Solvers/learners (GD, SGD, Adagrad, Adam, ...)



Part I

The Model



The Model

$$y = f(X, \theta) + \epsilon = f(X_1, X_2, \dots, X_m, \theta_1, \theta_2, \dots, \theta_k) + \epsilon$$

Noise added since all Models are wrong.
↑

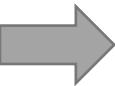
y: response, dependent variables, output, Target Variable

X: predictors, independent variables, input, Features

θ : estimates, specifications, Parameters

✓ It is all about estimating f by \hat{f} for two purposes:

- 1) Inference (interpretable ML)
- 2) Prediction



Parameters and Hyperparameters

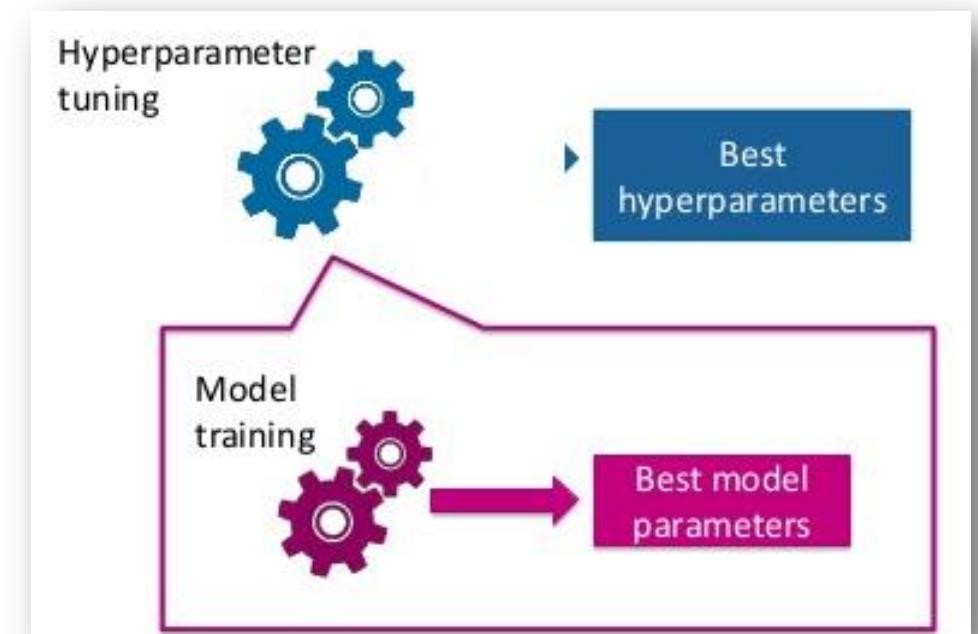
$$y = f(X, \theta) + \epsilon = f(X_1, X_2, \dots, X_m, \theta_1, \theta_2, \dots, \theta_k) + \epsilon$$

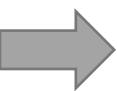
Model **parameters** are estimated from data automatically and model **hyperparameters** are set manually (prior to training the model) and are used in processes to help estimate model parameters.

Example? $\text{Wage} = \beta_0 + \beta_1 \text{Edu} + \beta_2 \text{Edu}^2 + \dots + \beta_K \text{Edu}^K + \epsilon$

Parameters : β_0, \dots, β_K

Hyper Parameters : Degree of Polynomial i.e. K .





Parametric Vs. Nonparametric models

$$y = f(X, \theta) + \epsilon$$

The true relationship, $f(X)$ is **unknown** and the goal is to see which ML algorithm is better at **approximating** it. An algorithm learns/estimates $f(X)$ from training data.

Parametric Model: Functional form

$f(X)$ is **assumed**. Examples:

Linear regression, GLM, ^{Generalises linear Regression},
logistic regression, simple
Neural networks,

Parametric
algorithms

Non-Parametric Model: Functional form

$f(X)$ is NOT assumed. Free
to **learn any functional form**.

Examples: KNN, CART,
Random forest, SVM, ANN,
...

Nonparametric
algorithms

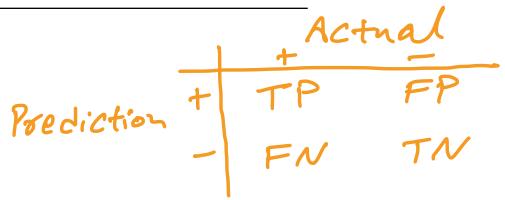
Classification & Regression Trees

	Pros 	Cons 
Parametric algorithms	Simpler Easier to understand and to interpret Faster Very fast to fit your data Less data Require "few" data to yield good perf.	Limited complexity Because of the specified form, parametric algorithms are more suited for "simple" problems where you can guess the structure in the data
Nonparametric algorithms	Flexibility Can fit a large number of functional forms, which doesn't need to be assumed Performance Performance will likely be higher than parametric algorithms as soon as data structures get complex	Slower Computations will be significantly longer More data Require large amount of data to learn Overfitting We'll see in a bit what this is, but it affects model performance

Part II

Evaluation Metrics

Evaluation metrics



In general, we want to compare how close are the predictions to the actual numbers in the **test set**.

This is typically assessed using

- MSE for **quantitative** response
- Misclassification rate for **qualitative** response
- Can't use Evaluation metric for Unsupervised learning as it doesn't have labels.

Evaluation Metrics

Classification

- Confusion Matrix
- Accuracy
- Precision and Recall
- F-score
- AUC-ROC
- Log Loss
- Gini Coefficient

Regression

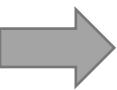
- MAE
(mean abs. error)
- MSE
(mean sq. error)
- RMSE
(Root mean sq.error)
- RMSLE
(Root mean sq.error log error)
- R^2 and Adjusted R^2

Think of Bias-Variance Trade-off in Repeated Sample. Bias is how much the model doesn't fit the data. For eg. higher order polynomial will have low bias as they fit the data perfectly. Variance is how much the model fit varies for different dataset. For eg. higher order polynomial will have low bias but for new dataset it will vary more on average compared to st. line so will have higher variance.

Part III

Bias-Variance Tradeoff

$$IE(\hat{f}(x)) = f(x) \Rightarrow \text{Unbiased}.$$



ML relative to statistical learning algorithms

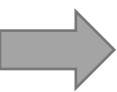
- **Advantages**

- Ability to uncover complex interactions
- Process massive amount of data quickly
- Capture non-linear relationships
- Predict structural changes between features and target

- **Disadvantages**

- Can produce overly complex models
- Difficult to interpret
- Sensitive to noise *(Model Shouldn't memorize data)*
- Can overfit!

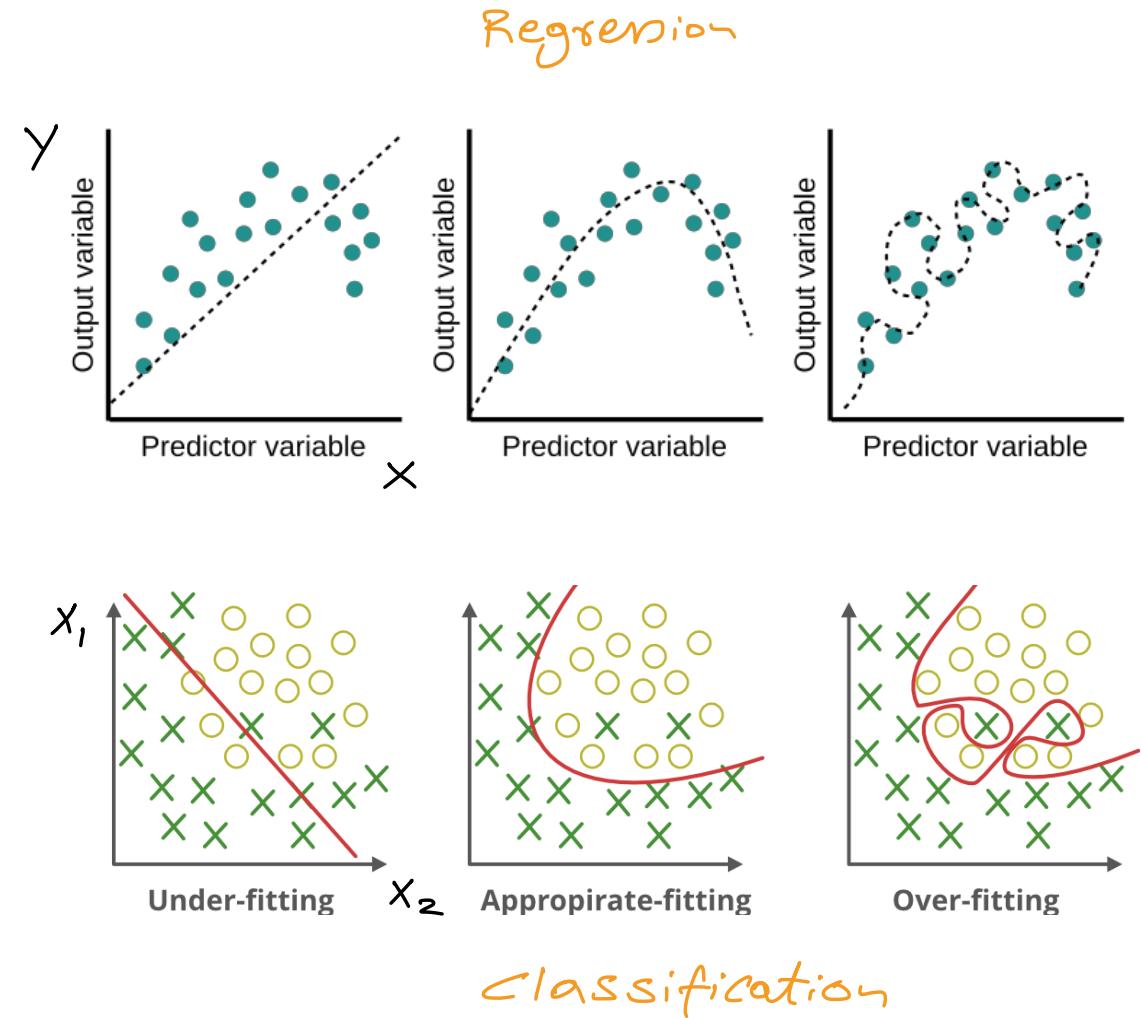
	Statistical Learning	Machine Learning
Focus	Hypothesis testing & interpretability	Predictive accuracy
Driver	Math, theory, hypothesis	Fitting data
Data size	Any reasonable set	Big data
Data type	Structured	Structured, unstructured, semi-structured
Dimensions / scalability	Mostly low dimensional data	High dimensional data
Model choice	Parameter significance & in-sample goodness of fit	Cross-validation of predictive accuracy on partitions of data
Interpretability	High	Low
Strength	Understand causal relationship & behavior	Prediction (forecasting and nowcasting)

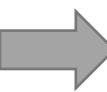


Overfitting

Overfitting happens when the fitted algorithm does **not generalize** well to new data:

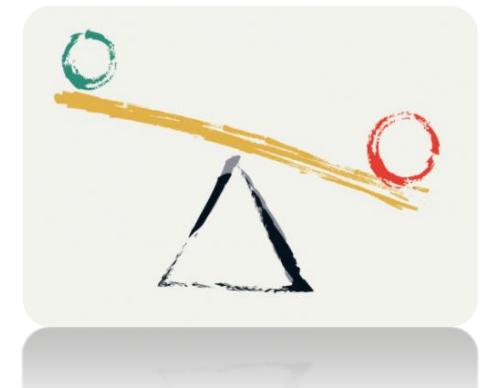
- The model fits the training data **too well** while not predicting well in the new data
- The model **fits the noise (ϵ)** in training data (finds a pattern that does not exist)
- The algorithm has simply **memorized** the data, rather than **learned** from it!
- The model is too **complex**!





MSE decomposition

The **bias-variance** tradeoff is one of the core concepts in supervised learning.



Assume that the data is generated by a simple model!

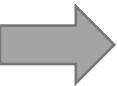
$$y_i = f(\mathbf{x}_i) + \epsilon_i, \quad \mathbb{E}[\epsilon] = 0, \quad \mathbb{V}[\epsilon] = \sigma^2$$

The estimated model yields

$$\hat{y}_i = \hat{f}(X_i)$$

Let us decompose the mean squared error (**MSE**):

$$\begin{aligned} \text{MSE} &= \mathbb{E}[(y - \hat{y})^2] = \mathbb{E}[(\underbrace{y}_{\text{true}} - \underbrace{\hat{f}(\mathbf{x})}_{\text{predicted}})^2] = \mathbb{E}[(f(\mathbf{x}) + \epsilon - \hat{f}(\mathbf{x}))^2] \\ &= \underbrace{\mathbb{E}[(f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2]}_{\text{total quadratic error}} + \underbrace{\mathbb{E}[\epsilon^2]}_{\text{irreducible error}} \quad \dots \quad = \underbrace{\mathbb{V}[\hat{f}(\mathbf{x})]}_{\text{variance of model}} + \underbrace{\mathbb{E}[(f(\mathbf{x}) - \hat{f}(\mathbf{x}))]^2}_{\text{squared bias}} + \sigma^2 \end{aligned}$$

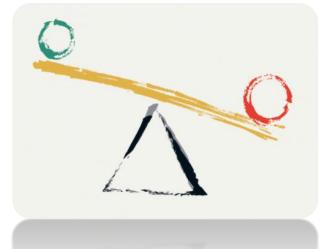


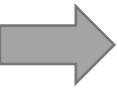
MSE decomposition

- Increasing sample size decreases both Variance & Bias.
- Bias-Variance Tradeoff only applies to Repeated Sampling not increasing sample size.

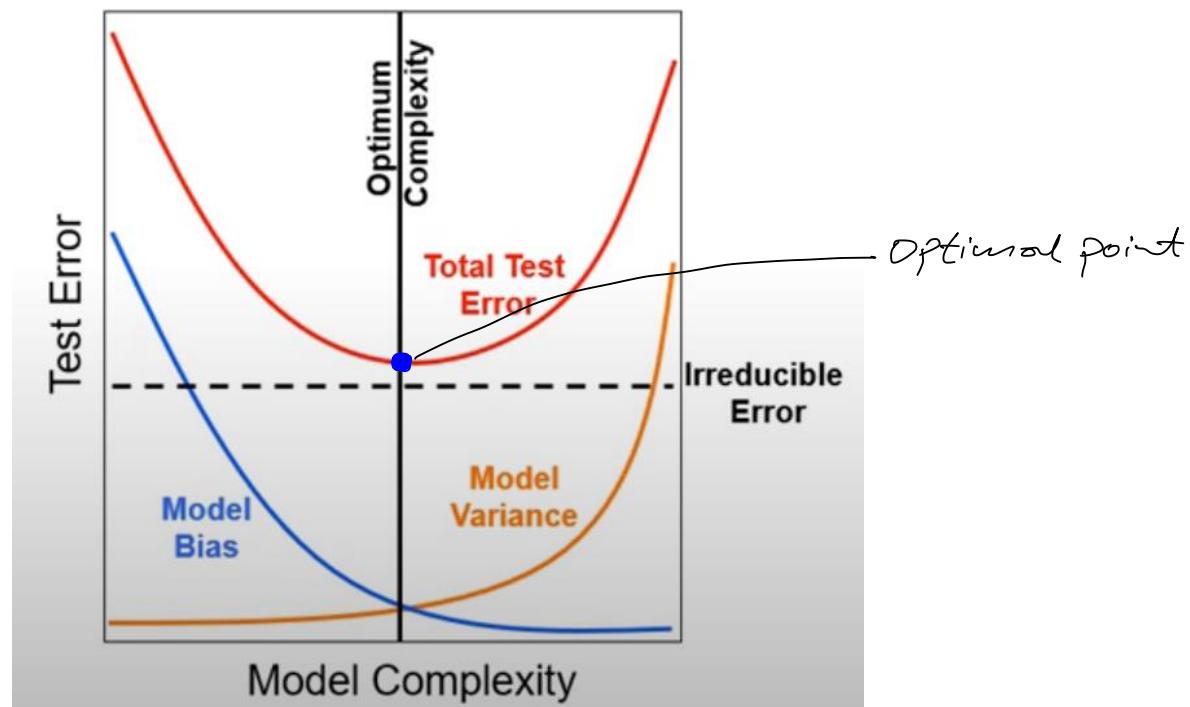
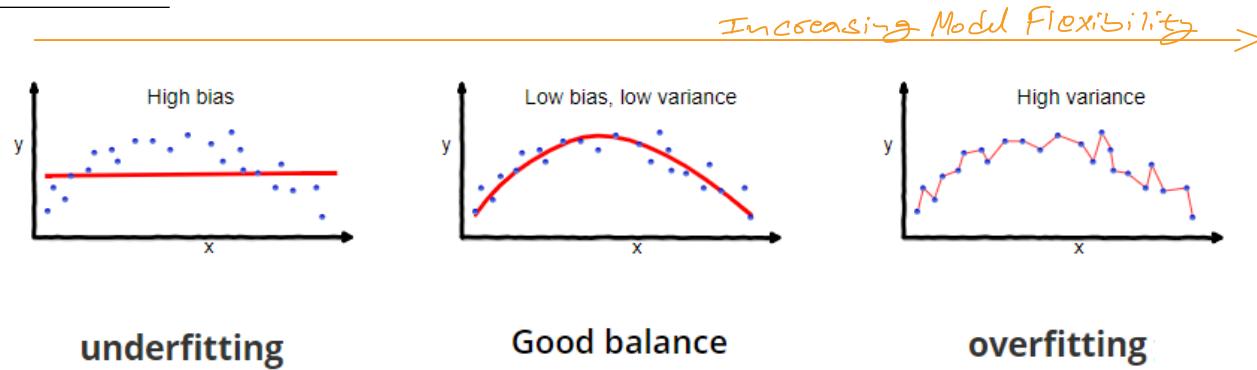
$$MSE = \text{model variance} + \text{model bias} + \text{irreducible error}$$

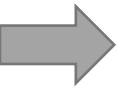
- 1) Model variance is the variance if we had estimated the model with a different **training set**
 - 2) Model bias is the error due to using an approximate model (model is too simple)
 - 3) Irreducible error is due to missing variables and limited samples. Can't be fixed with modeling
-
- The goal is to minimize the sum of **model variance** and **model bias**.
 - This is known as the bias-variance tradeoff because reducing one often leads to increasing the other.
 - Choosing the flexibility (complexity) of $\hat{f}(X)$, will amount to bias-variance tradeoff.



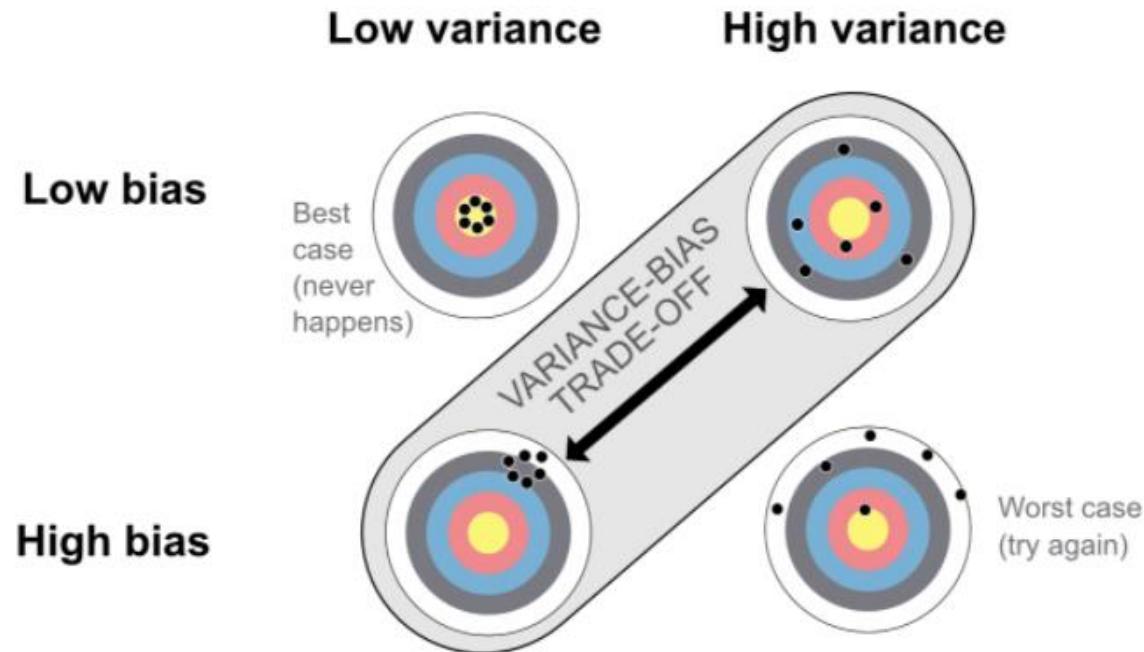


Representations of the bias-variance tradeoff



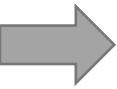


Other representations of the bias-variance tradeoff



Part IV

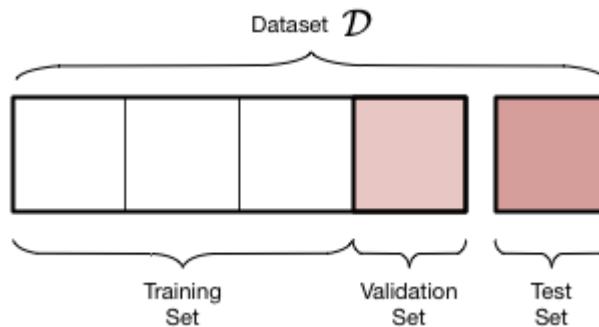
Resampling methods



Partitioning of the dataset

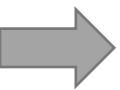
The data set is typically divided into three non-overlapping samples:

- 1) Training set used to train the model
- 2) Validation set for validating and tuning the model (*hyperparameters*)
- 3) Test set (holdout set) for testing the model's ability to predict well on new data



To be valid and useful, any supervised machine learning model **must** generalize well beyond the training data.

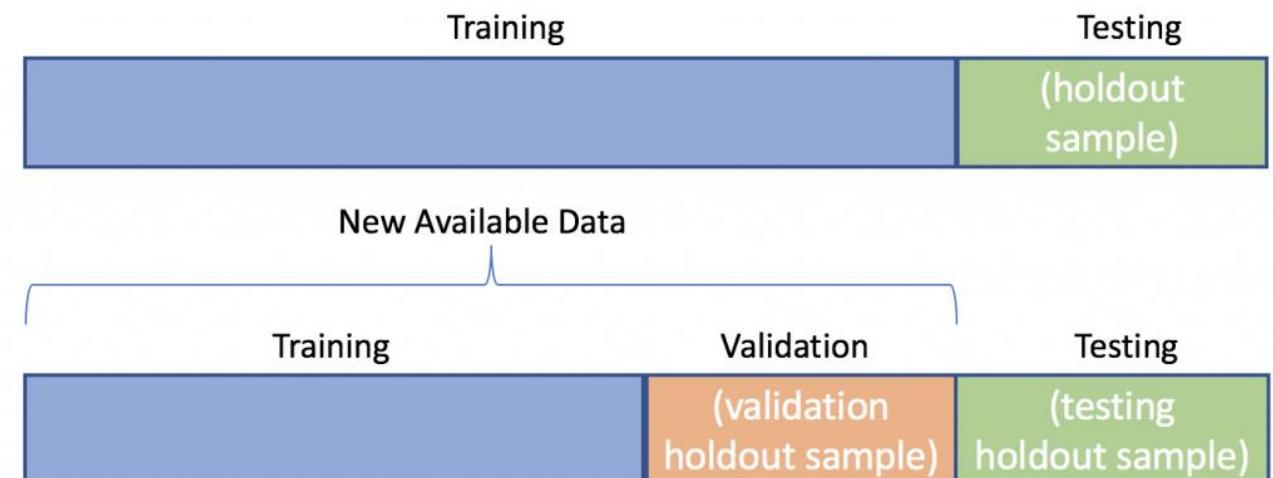
Large dataset is needed! But what if we don't have it? *ReSample DataSet*.



Resampling methods

Cross validation

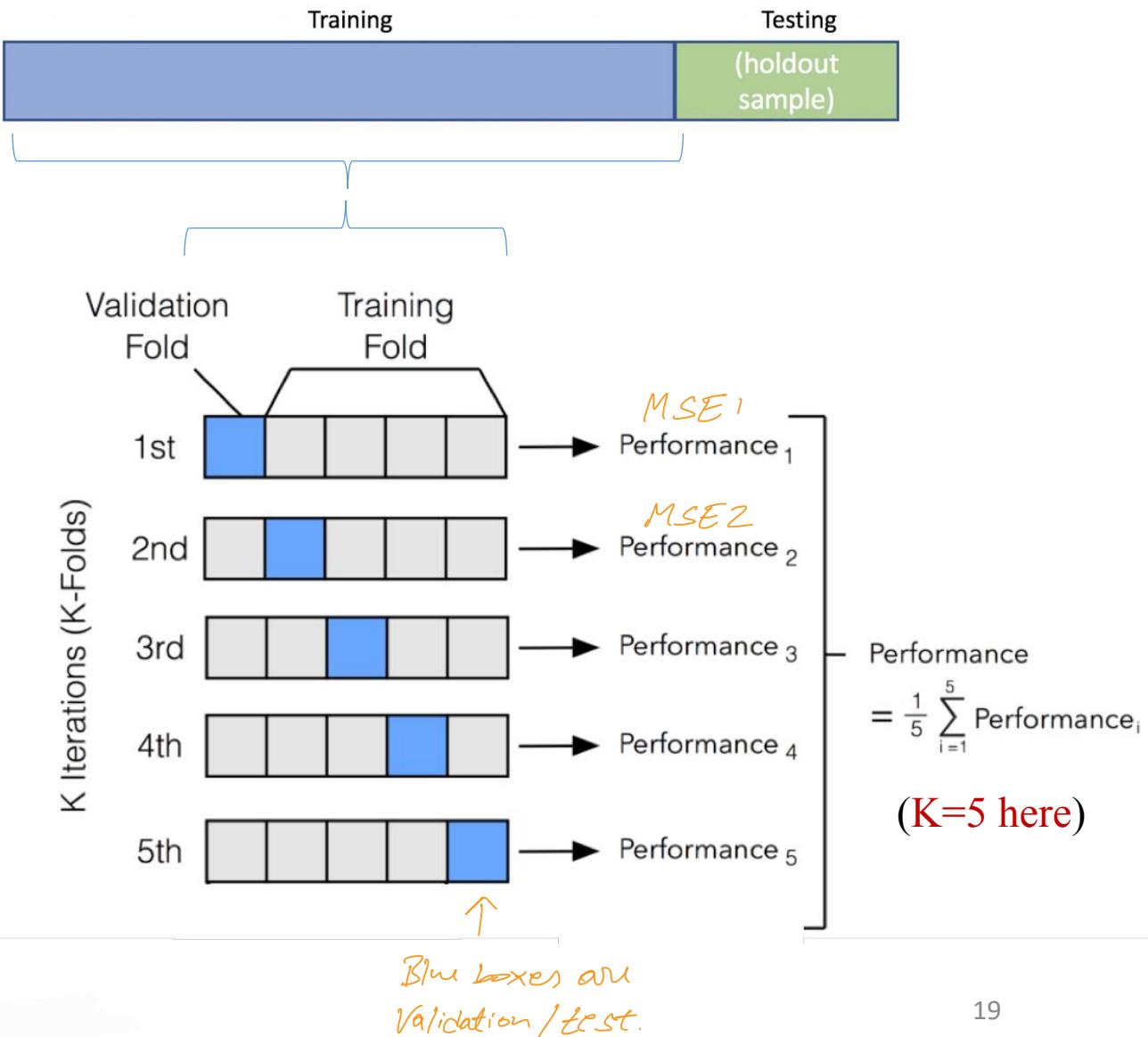
- Sometimes we cannot afford to split the data in three because the algorithm may **not learn** anything from a **small training dataset!**
- **Small validation set** is also problematic because we cannot tune the hyperparameters properly!
- Solution: combining the training and validation sets!
- The goal is to obtain additional information about the fitted model!
For example, to provide **estimates of test set prediction errors.**

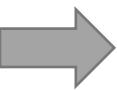


K-fold Cross Validation

Since Test set is always not labelled we need to come up with estimate for performance measure in Test set. This is a fair way of Estimation.

- 1) Divide the training data into K roughly equal-sized non-overlapping groups. Leave out k^{th} fold and fit the model to the other $k - 1$ folds. Finally, obtain predictions for the left-out k^{th} fold.
 - 2) Performance can be any of the evaluation metrics for regression or classification models. For example, MSE, accuracy, ...
 - 3) This is done in turn for each part $k = 1, 2, \dots, K$, and then the results are combined.- Leave one out CV (LOOCV): if there is only 1 observation in each fold.

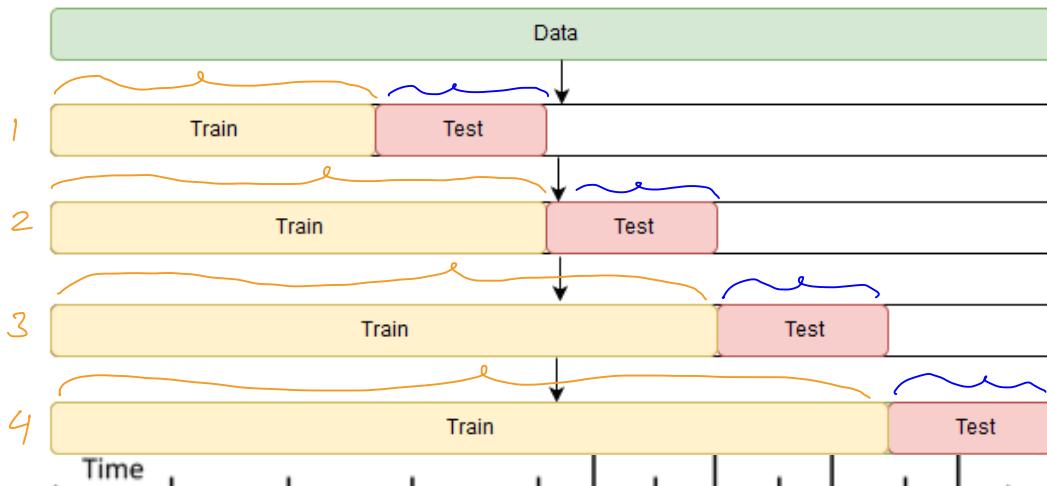




Time Series Cross Validation

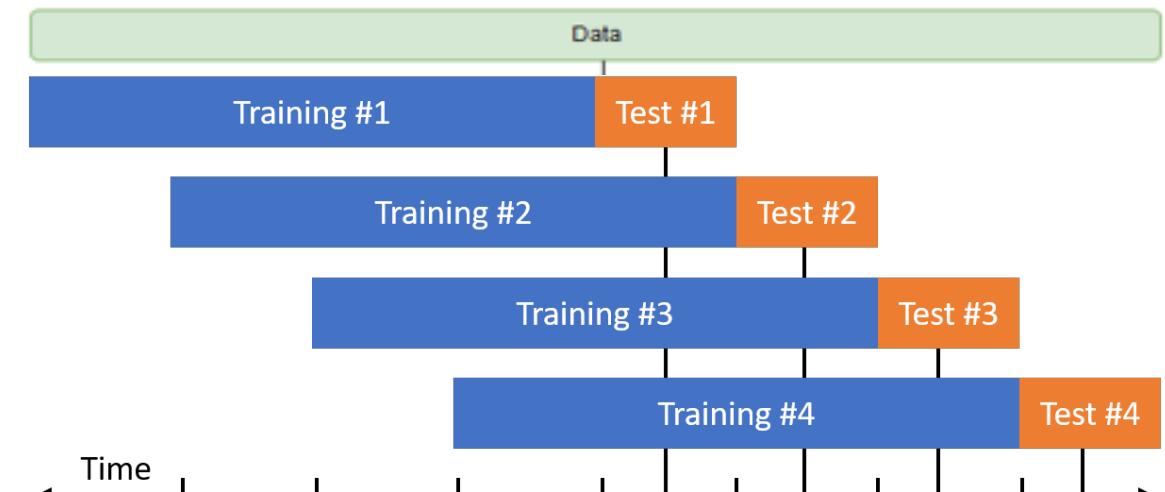
With time series data, we **cannot shuffle** the data! We also need to **avoid look-ahead bias**!

① Walk forward cross validation Expanding windows

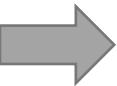


Don't Shuffle data. The test part is always in future
So we avoid look-ahead Bias.

② Walk forward cross validation Rolling windows



Rolling Windows performs better for Time-Series data as it Contains more recent data.

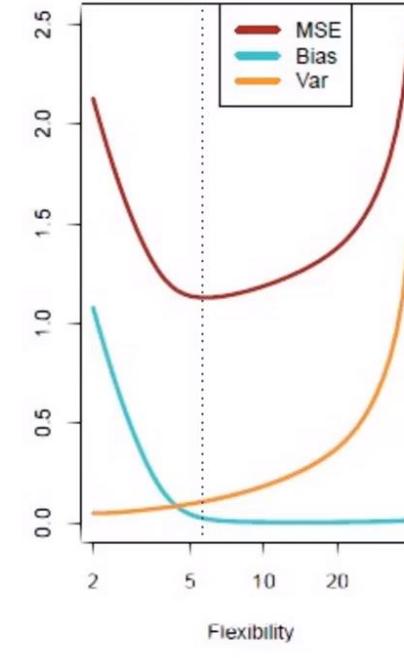
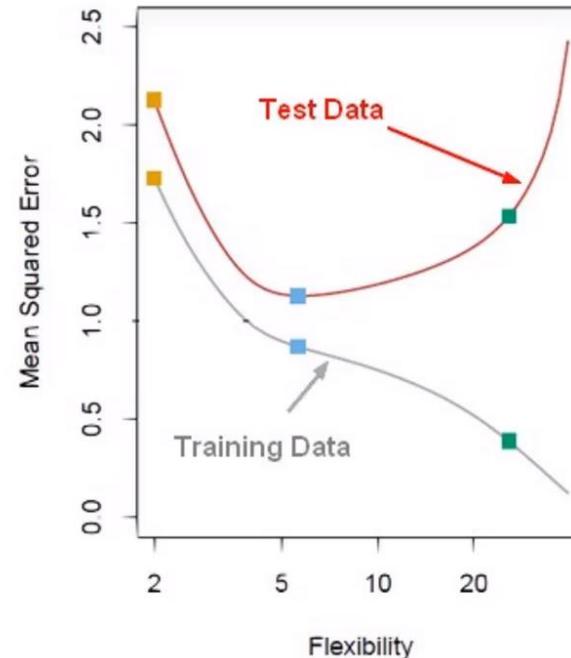
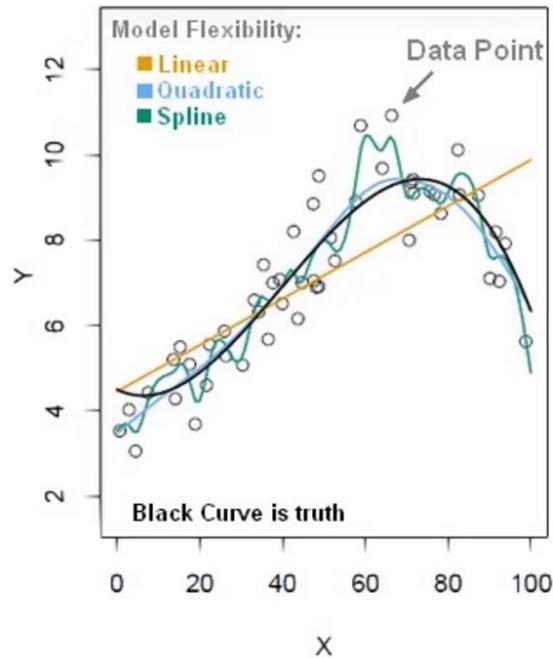


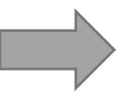
Mitigate overfitting

- We will plot MSE of Cross-Validated data as MSE of Test data is not observable & it is a good estimate.

The main techniques used to mitigate overfitting risk in a model construction are:

- 1) Complexity reduction (regularization) *by Penalization*.
- 2) Cross validation (estimate the test error)





Scaling the features

Let us use x_i for raw input and \tilde{x}_i for the transformed data. Common scaling practices include:

- Standardization:

$$\tilde{x}_i = \left(\frac{x_i - \mu_x}{\sigma_x} \right)$$

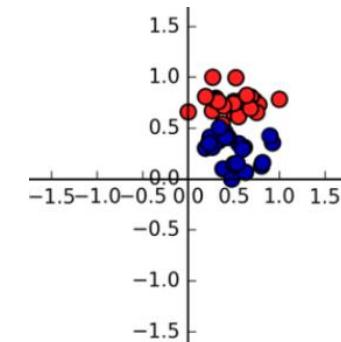
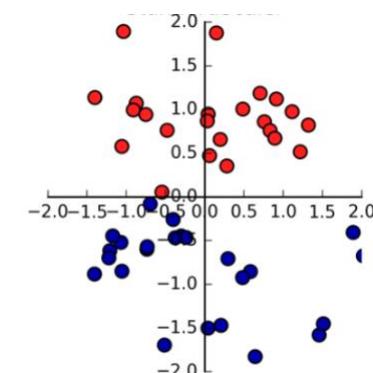
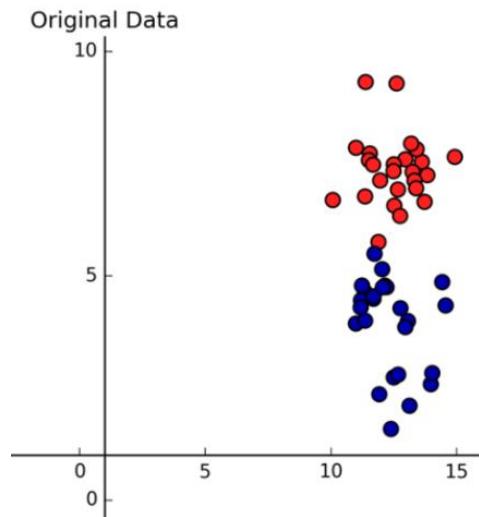
- Normalization:

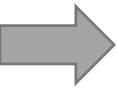
- Min-Max scaler over [0,1]:

$$\tilde{x}_i = \left(\frac{x_i - \text{Min}(X)}{\text{Max}(X) - \text{Min}(X)} \right)$$

- Min-Max scaler over [-1,1]:

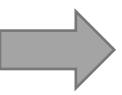
$$\tilde{x}_i = 2 * \left(\frac{x_i - \text{Min}(X)}{\text{Max}(X) - \text{Min}(X)} \right) - 1$$





Scaling the features

- Normalization is good to use when the distribution of the data does not follow a Normal distribution. (ideal for non-parametric algorithms like KNN)
- Standardization, can be helpful in cases where the data follows a Normal distribution. However, this does not have to be necessarily true.
- Unlike normalization, standardization does not have a **bounding range**. So, even if you have **outliers** in your data, they will not be affected by standardization.
- Be careful when scaling the **time series data!** Why?
- To avoid **data leakage**, It is a good practice to fit the scaler on the training data and then use it to transform the testing data.
- The choice of using normalization or standardization will **depend on your problem** and the **machine learning algorithm** you are using



Question of the day!

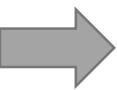


Overfitting.

Class 5- Machine Learning concepts

Part II

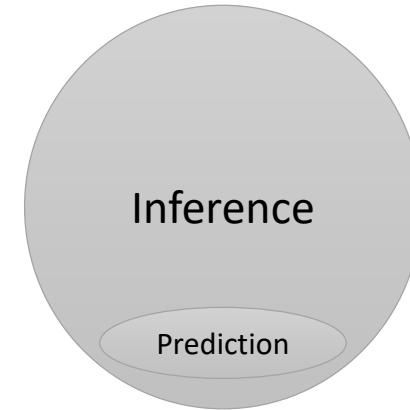




Motivation

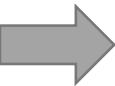
Machine learning fundamental concepts:

- Inference and prediction
- Part I: The Model
 - Parameters and hyperparameters
 - Parametric vs nonparametric ML models
- Part II: Evaluation metrics
- Part III: Bias-Variance tradeoff
- Part IV: Resampling methods
- Part V: How do machines learn?
- Part VI: Solvers/learners (GD, SGD, Adagrad, Adam, ...)



Part V

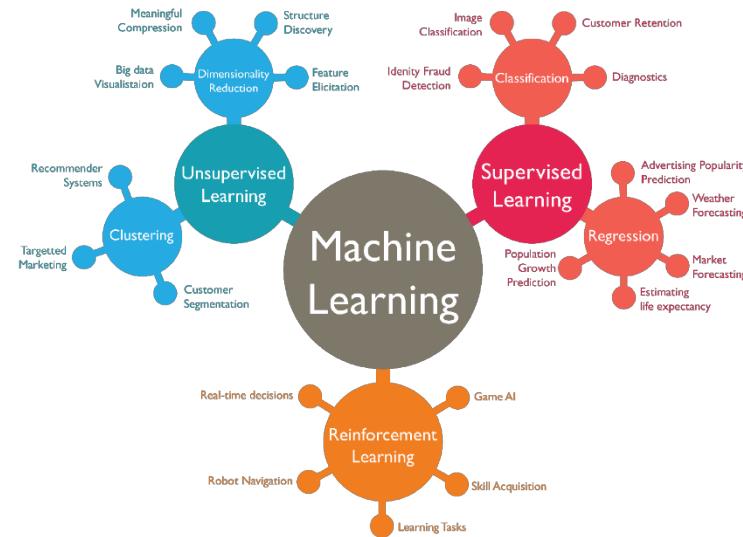
How do machines learn?

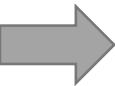


How do machines learn?

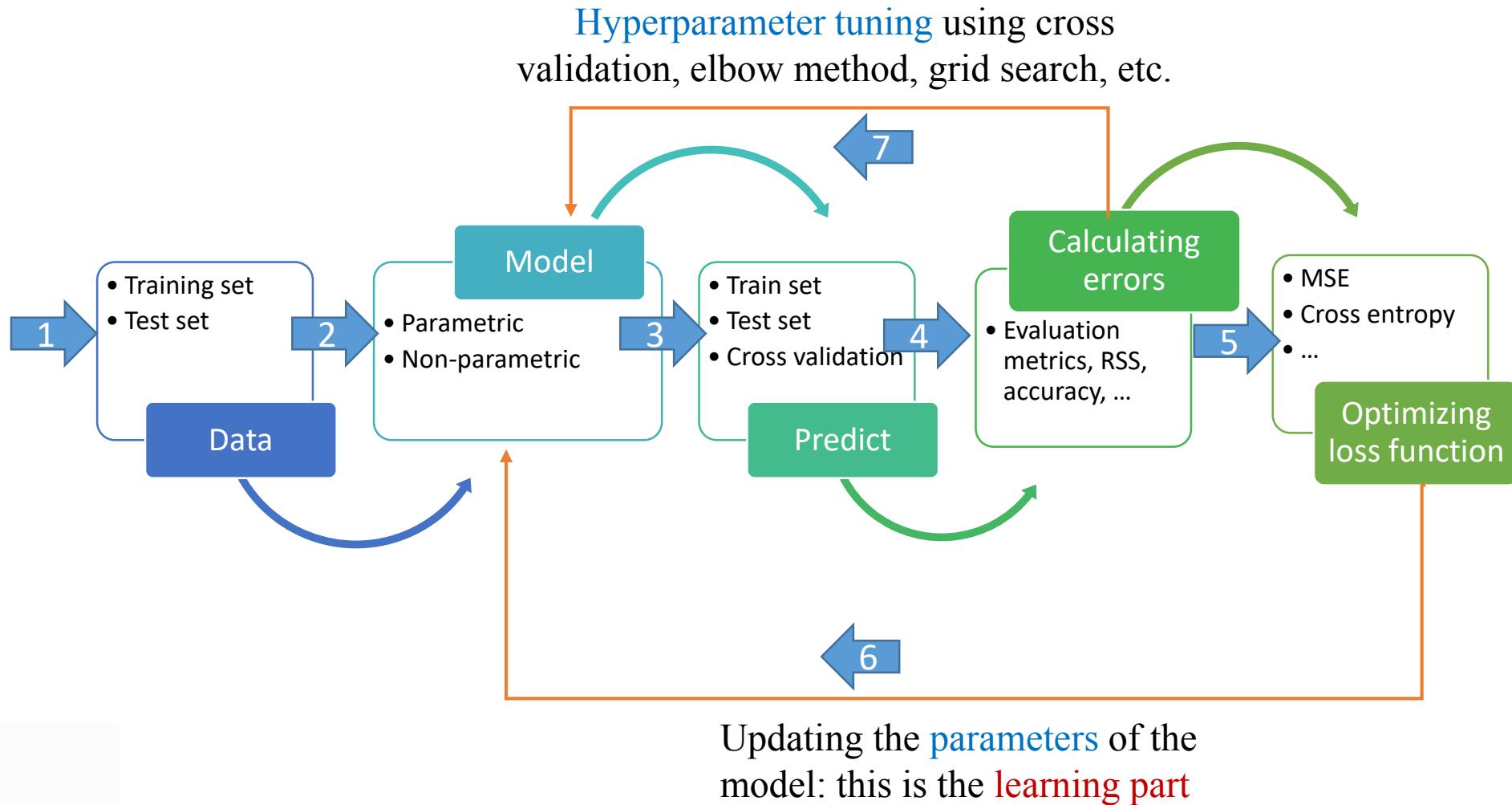
The short answer: **Algorithms!**

- **Algorithm**: a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
- Generally, the more data a machine learning algorithm is provided the more accurate it becomes.
- Different types of algorithms:
 - Supervised
 - Unsupervised
 - Reinforcement



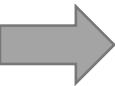


How do machines learn?



Part VI

Solvers (GD, SGD, Adagrad, Adam, ...)



Solvers (learners)!

$$\text{Errors} = \hat{y} - \hat{\hat{y}}$$

$$\text{E.g. of Loss fn: } \text{MSE} = E[(\hat{y} - \hat{\hat{y}})^2]$$

A **Loss Function** tells us “how good” our model is at making predictions for a given set of parameters. The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate.

The two most frequently used optimization algorithms when the **loss function** is differentiable are:

- 1) Gradient Descent (GD)
- 2) Stochastic Gradient Descent (SGD)

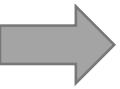
Gradient Descent: is an iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one starts at some random point and takes steps proportional to the negative of the gradient of the function at the current point.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

\downarrow
update by RHS

- θ_j is the model's j^{th} parameter
- α is the learning rate
- $J(\theta)$ is the loss function (which is differentiable)

loss fn is convex function so we always get a minimum not maximum.

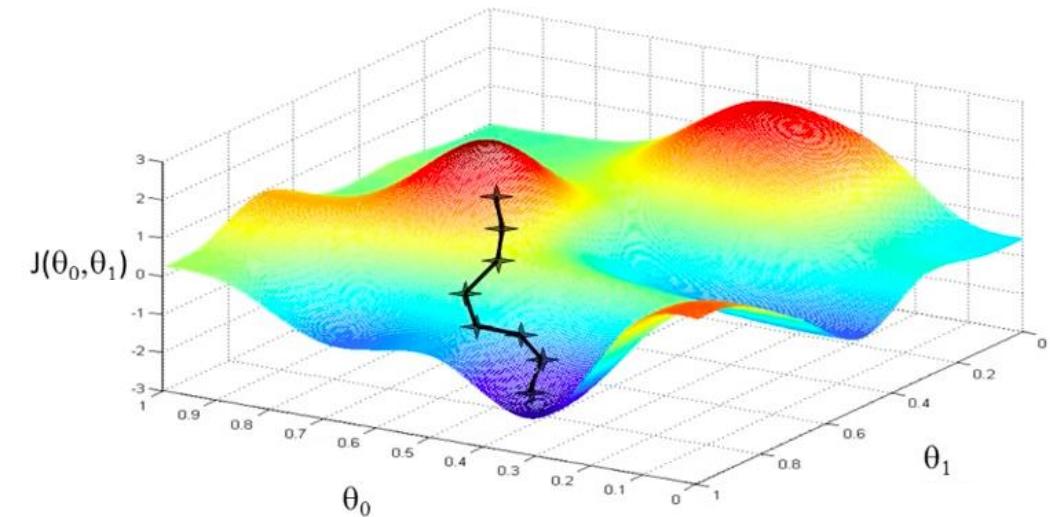
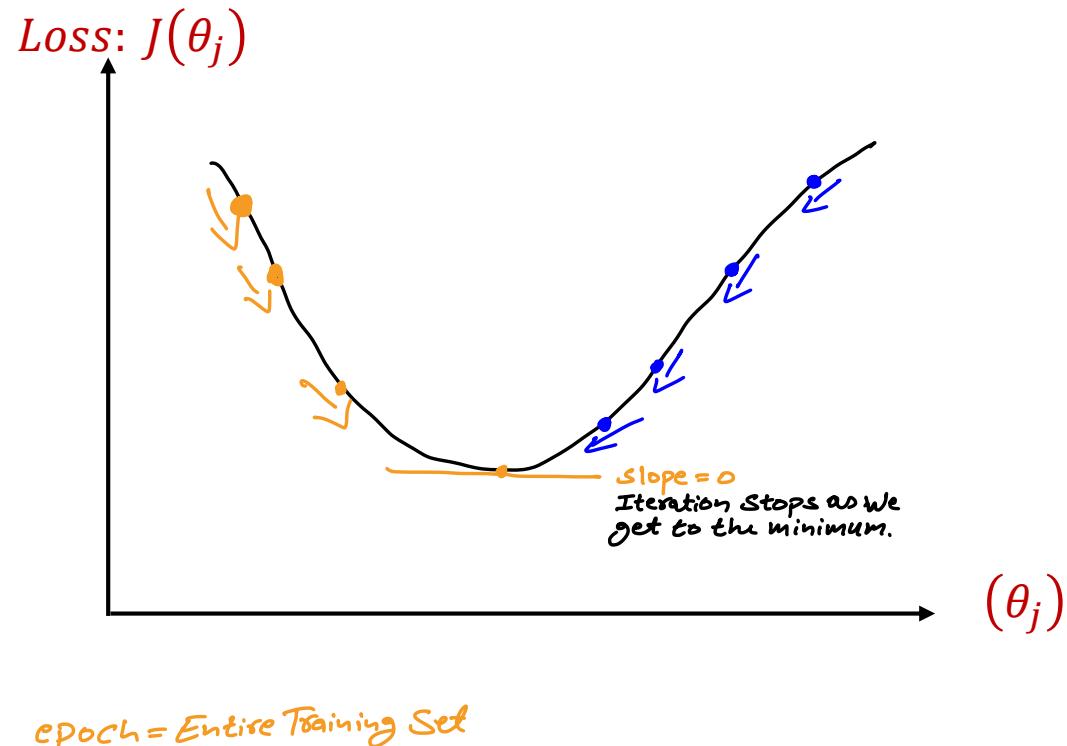


Gradient Descent Visualization

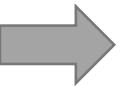
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Gradient descent proceeds in **epochs**. An epoch consists of using the training set entirely to update each parameter. The learning rate α controls the size of an update

eg: $Wage = \theta_0 + \theta_1 Edu + u$



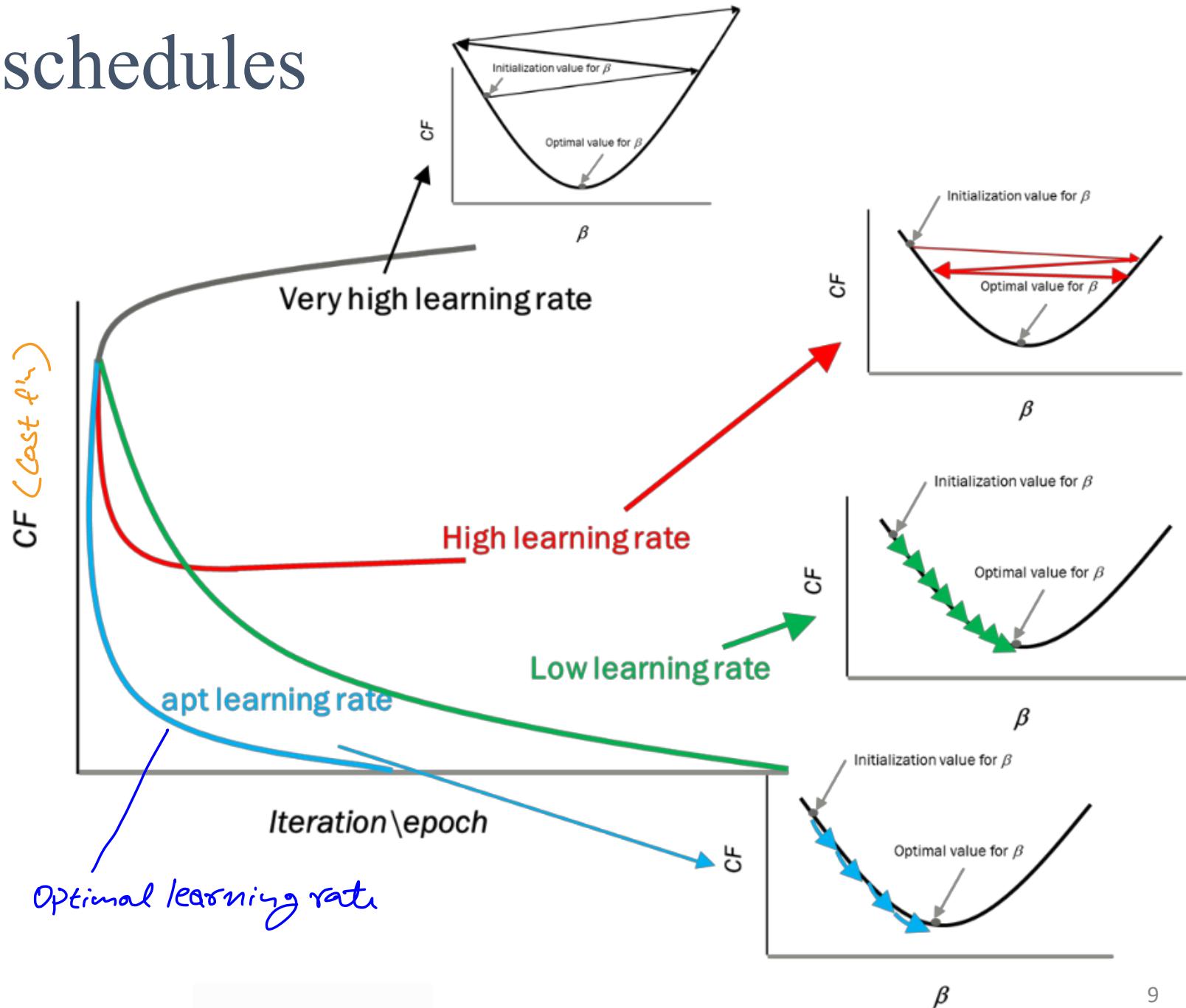
repeat until convergence {
 $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
(for $j = 1$ and $j = 0$)
}

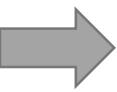


Learning rate schedules

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- If α is **too small**, gradient descent can be **slow**
- If α is **too large**, gradient descent can **overshoot** the minimum. It may fail to converge, or even **diverge**.





Beyond Gradient Descent?

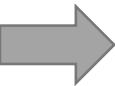
Disadvantages of gradient descent:

- Single batch: use the entire training set to update a parameter!
- Sensitive to the choice of the learning rate
- Slow for large datasets

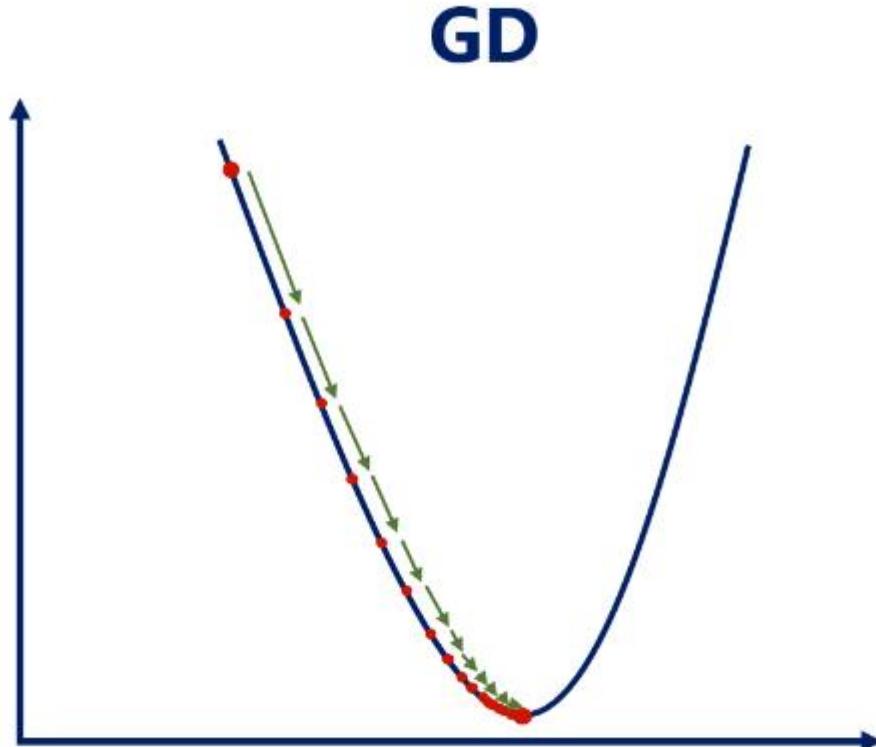
Solution: SGD

(Minibatch) Stochastic Gradient Descent: is a version of the algorithm that speeds up the computation by approximating the gradient using **smaller batches** (subsets) of the training data. SGD itself has various “upgrades”.

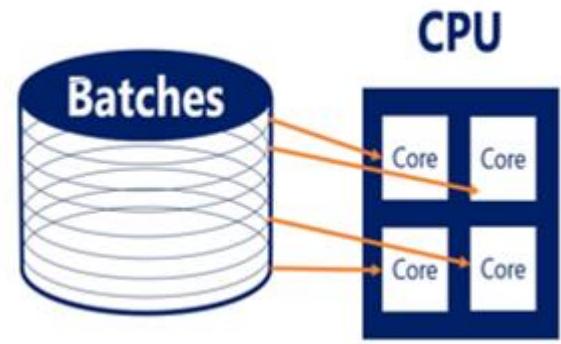
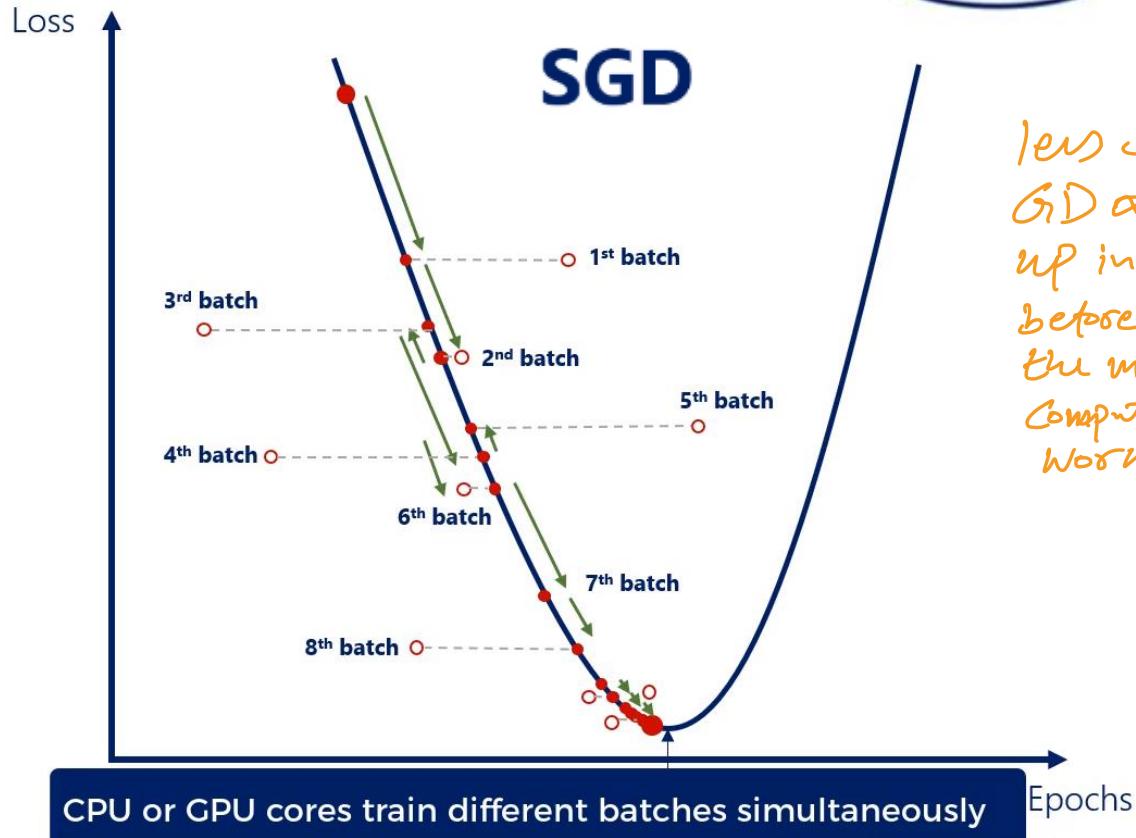
- 1) Adagrad
- 2) Adam



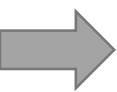
Why SGD?



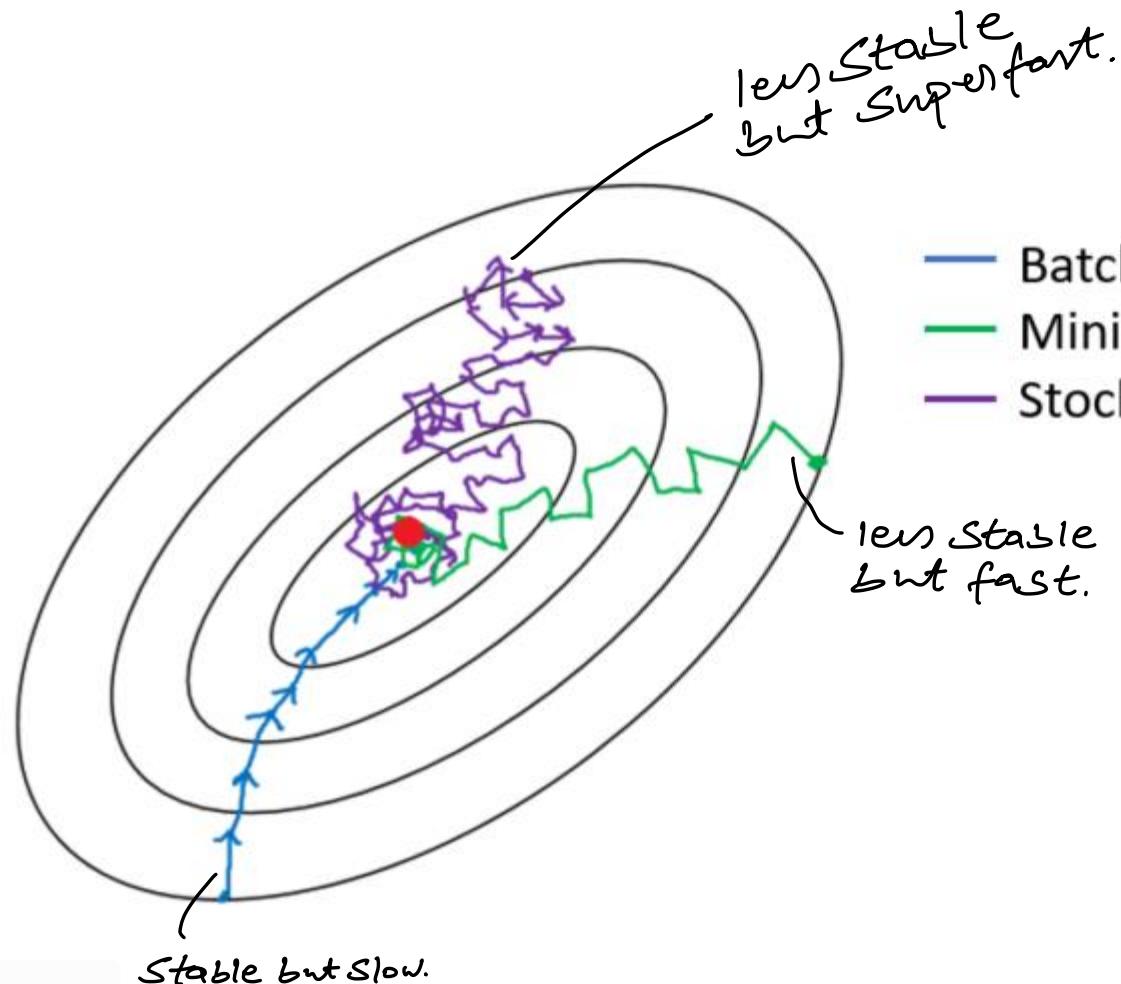
GD approaches minimum smoothly but very slowly.



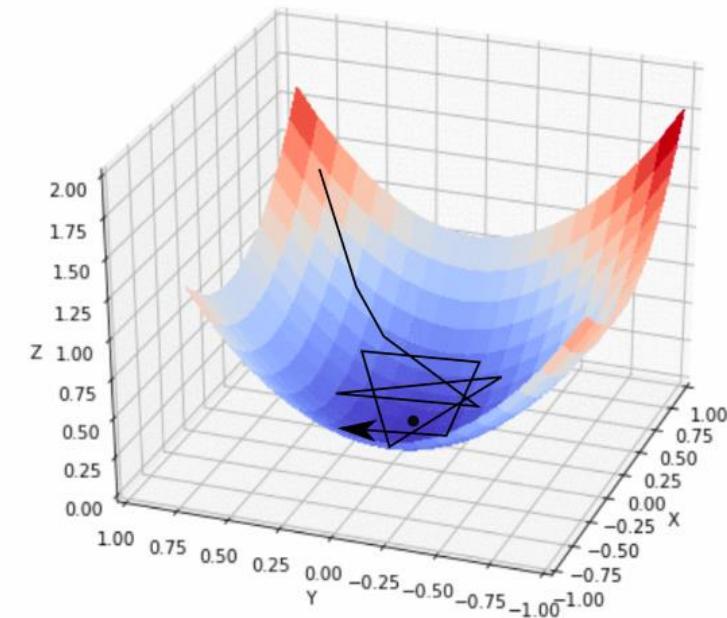
less stable than
GD as it can go up in other direction before approaching the minimum but you can do entire work much faster.

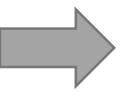


SGD vs GD

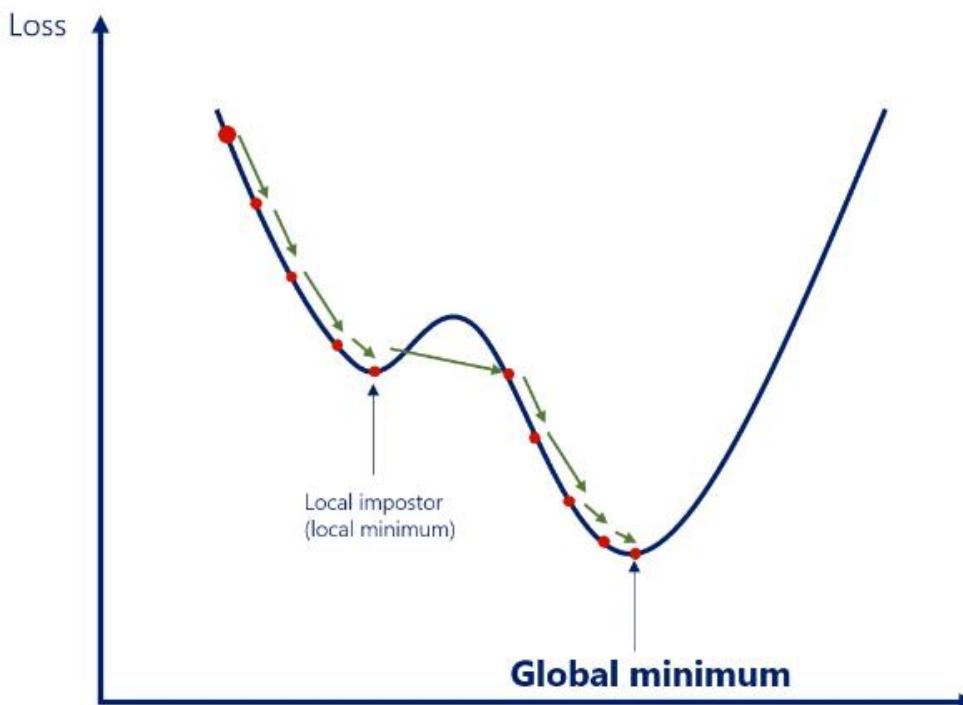
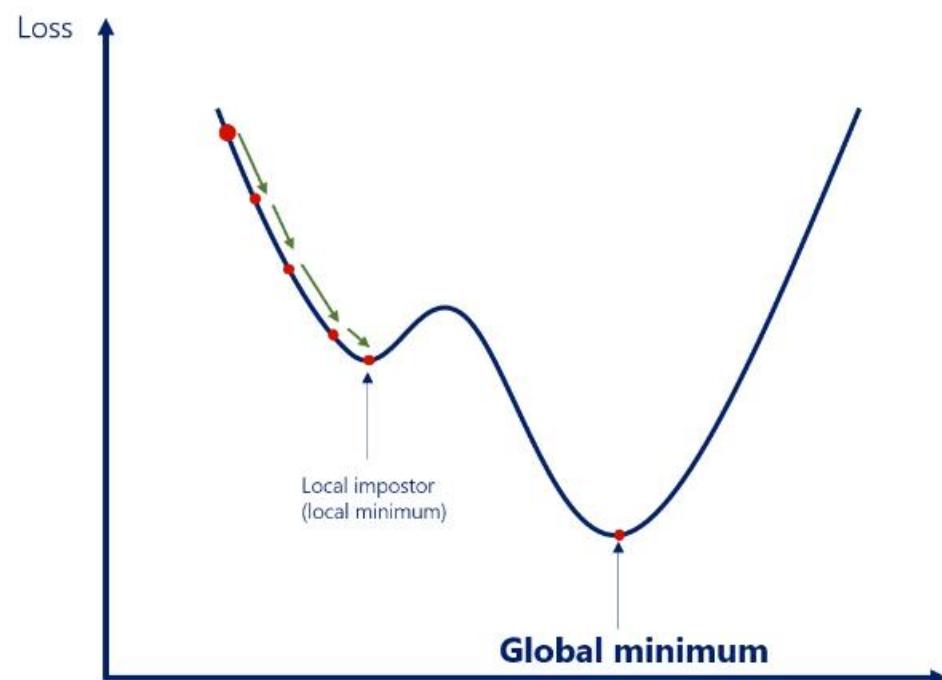


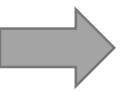
- Batch gradient descent (entire epoch i.e. ∇D)
- Mini-batch gradient Descent (batch = 10 or 20)
- Stochastic gradient descent (batch = 1 i.e. using 1 observation to update parameters)





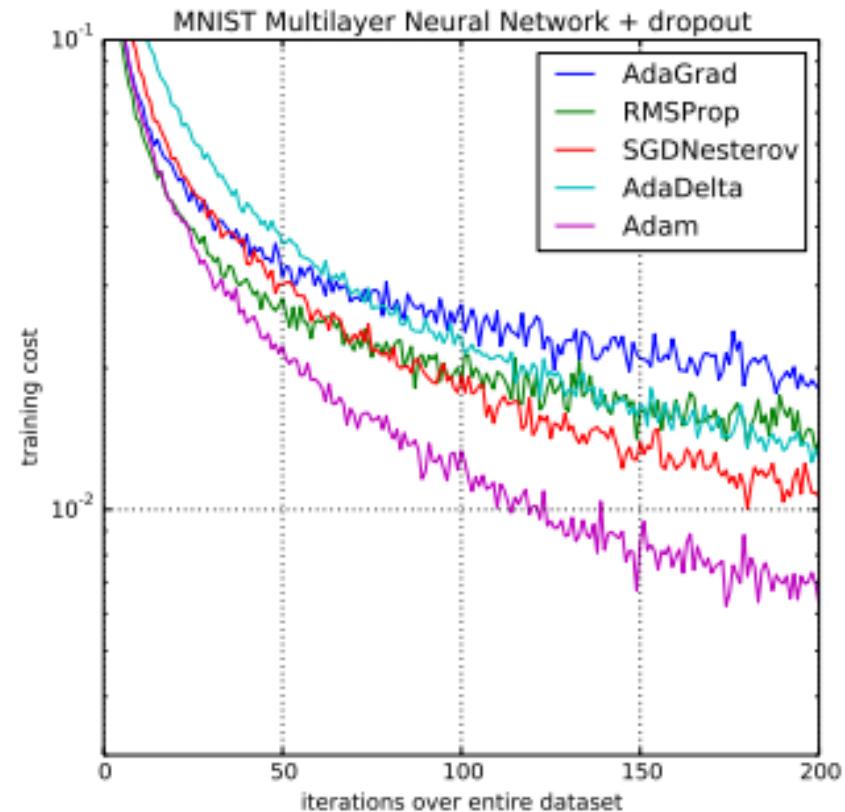
Why upgrade SGD?

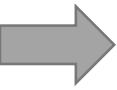




SGD upgrades

- **Adagrad** (Adaptive Gradient Algorithm): is a version of SGD that scales α for each parameter according to the history of gradients. As a result, is reduced for very large gradients and vice-versa.
- **Adam** (Adaptive Moment Estimation): is a method that helps accelerate SGD by orienting the gradient descent in the relevant direction and reducing oscillations.
*Best so far.
Used extensively in Deep learning Models.*





Final message!

Notice that gradient descent and its **variants are not machine learning algorithms**. They are **solvers** of minimization problems in which the function to minimize has a gradient (in most points of its domain).

Question of the day!

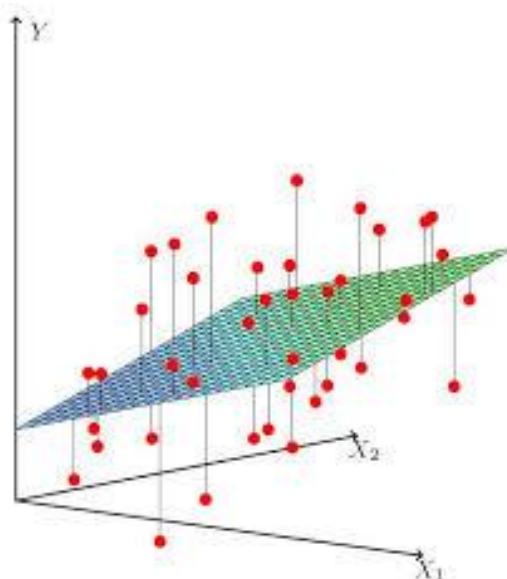
Me optimizing linear regression
using gradient descent

(OLS) Least Squares:



For linear Regression
OLS is Much
faster. For logistic
Regression use MLE
(Max. Likelihood Esti.)

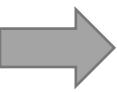
Class 6- Linear Regression



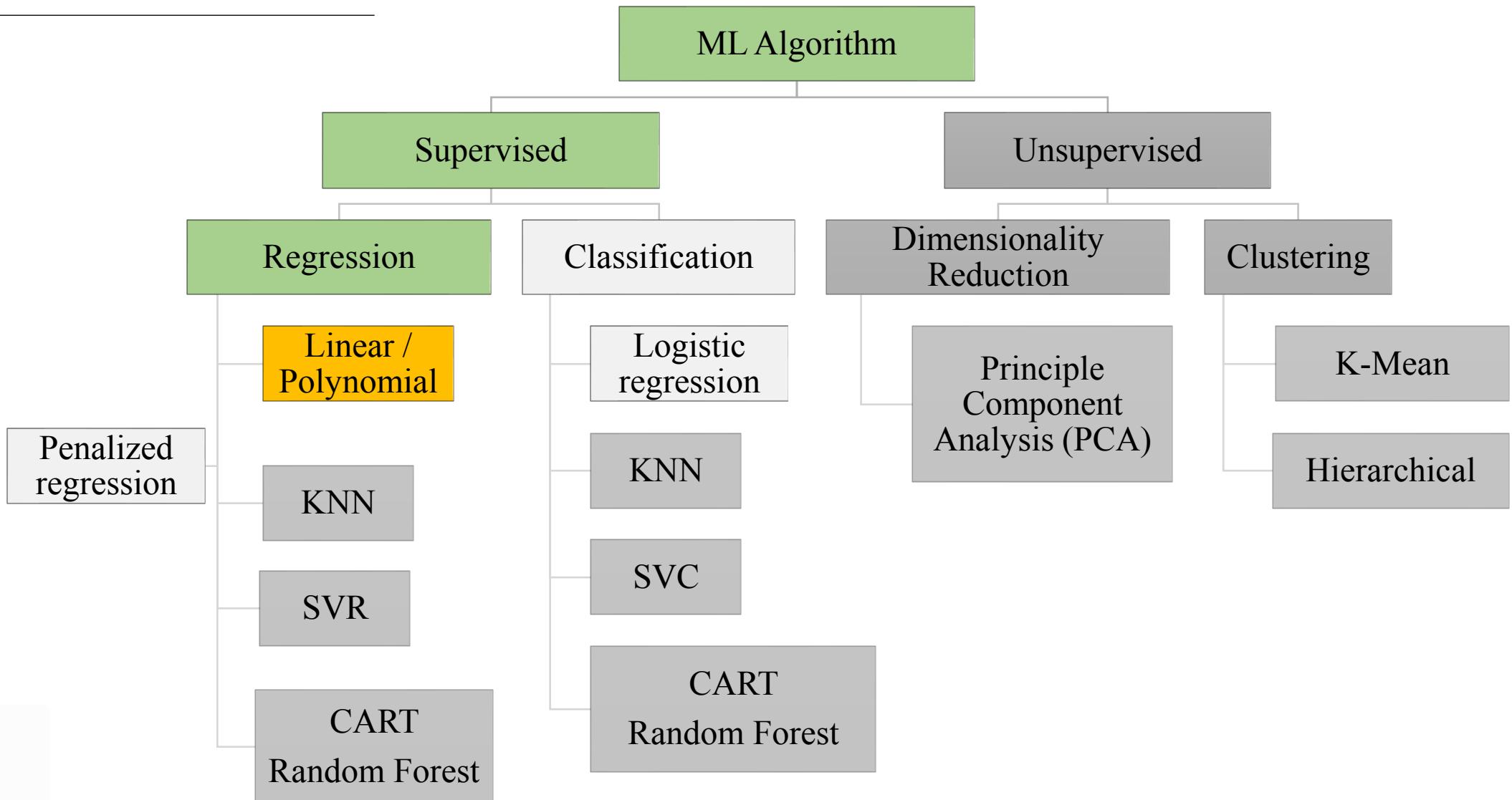
$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + \epsilon$
is called Linear Regression Model
as long as it is linear in parameters.
The x 's can have powers.

In Machine Learning Terminology:

$$\beta_0 = b$$
$$\beta_1, \dots, \beta_k = w_1, \dots, w_k \text{ (weights)}$$

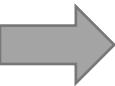


Road map



Part I

Linear regression: Machine Learning approach



Linear Models

The linear model is an important example of a **parametric** model

- We have a collection of labeled examples $\{(X_i, y_i)\}_{i=1}^N$, where
 - N is the size of the collection *sample*
 - X_i is the **D-dimensional** feature vector
 - y_i is a real-valued target

$$\hat{f}(x) = f_{w,b}(X) = \mathbf{W}X + \mathbf{b}$$

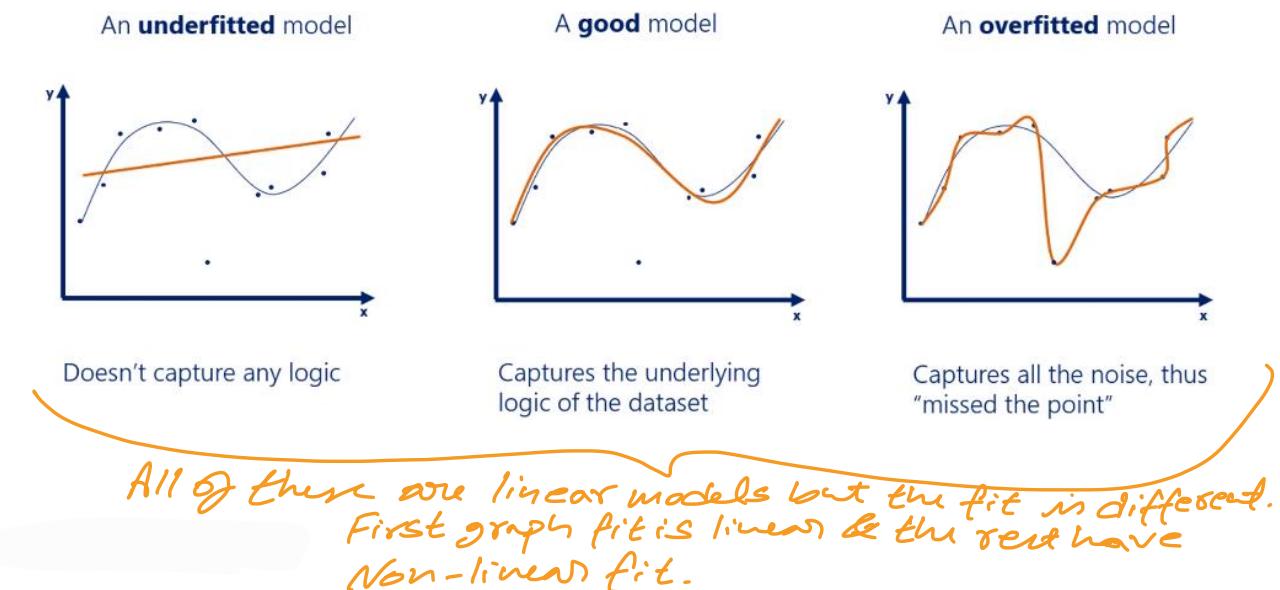
The model $f_{w,b}$ is a linear combination of features and parameterized by \mathbf{W} and \mathbf{b}

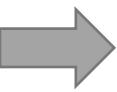
- \mathbf{W} is a D-dimensional vector of parameters
- \mathbf{b} is a real number

→ Linear Models (cont'd)

$$f_{w,b}(X) = \mathbf{W}X + \mathbf{b}$$

- The model is specified with **D+1** parameter.
- We estimate the parameters (W^*, b^*) by fitting the model to training data.
- Although it is almost never correct, a linear model often serves as a **simple** and **interpretable** approximation the unknown true $f(X)$.
- It may seem overly simplistic, but linear regression is extremely useful both conceptually and practically.
- Linear regression models rarely overfit. *(but high Bias)*





The optimization problem

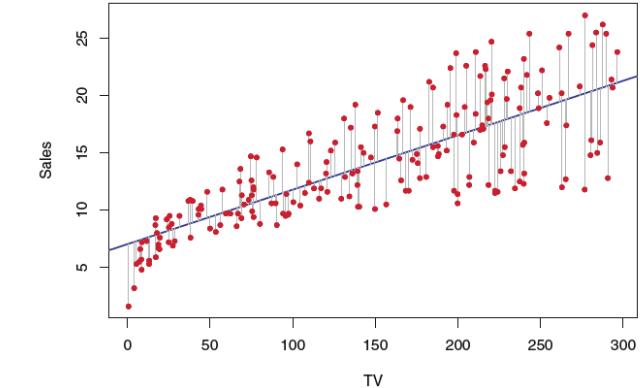
The optimization problem is defined as:

$$\text{Min}_{\mathbf{w}, \mathbf{b}} \text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2$$

$(y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2$ is called the **objective function**, or the **loss function**! Or the **squared error loss**. *or cost fn.*

Why quadratic loss function? Why not using absolute value or cube?

1. More convenient (well-behaved derivative).
2. There exists a closed form solution.



The closed form solving OLS is
 $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$
only if there is no perfect collinearity.
Perfect collinearity implies no sol'n as the system
of equations have more unknowns than eqns.
In ML, we don't care about such assumptions as
we can make use of Gradient descent.

The solution to this optimization problem is \mathbf{w}^* and \mathbf{b}^* . Now we can make predictions!

Linear Regression Evaluation Metrics

$R^2 \rightarrow$ What part of the variation in my Target Variable is explained by the model.

R^2 will increase as we add features to the model. So we need to adjust for adding explanatory variable by penalizing R^2 .
In the Train Set, we have to work with Adjusted R^2 .

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2} = 1 - \frac{SS_{residuals}}{SS_{total}}$$

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}|$$

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2}$$

Adjusted
versions



$$Adjusted R^2 = 1 - (1 - R^2) * \frac{n - 1}{n - k - 1}$$

$$AIC = \frac{2K}{N} - \frac{2 \ln(\hat{L})}{N}$$

$$BIC = \ln(N) K - 2 \ln(\hat{L})$$

- **AIC:** Akaike information criterion
- **BIC:** Bayesian information criterion
- K : number of estimated parameters
- \hat{L} : Maximum value of the likelihood function

Adding more Explanatory Variable means K would increase & So would AIC & BIC. High AIC & BIC is bad. High Adjusted R^2 is better.

Part II

Linear regression: **Econometrics** approach (this part is optional)

GAUSS MARKOV ASSUMPTIONS FOR REGRESSION

THE GAUSS-MARKOV ASSUMPTIONS

The following is a summary of the five Gauss-Markov assumptions that we used in this chapter. Remember, the first four were used to establish unbiasedness of OLS, whereas the fifth was added to derive the usual variance formulas and to conclude that OLS is best linear unbiased.

Assumption MLR.1 (Linear in Parameters)

The model in the population can be written as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + u,$$

where $\beta_0, \beta_1, \dots, \beta_k$ are the unknown parameters (constants) of interest and u is an unobserved random error or disturbance term.

Assumption MLR.2 (Random Sampling)

We have a random sample of n observations, $\{(x_{i1}, x_{i2}, \dots, x_{ik}, y_i): i = 1, 2, \dots, n\}$, following the population model in Assumption MLR.1.

Assumption MLR.3 (No Perfect Collinearity)

In the sample (and therefore in the population), none of the independent variables is constant, and there are no *exact linear* relationships among the independent variables.

Assumption MLR.4 (Zero Conditional Mean)

The error u has an expected value of zero given any values of the independent variables. In other words,

$$E(u|x_1, x_2, \dots, x_k) = 0.$$

Assumption MLR.5 (Homoskedasticity)

The error u has the same variance given any value of the explanatory variables. In other words,

$$\text{Var}(u|x_1, \dots, x_k) = \sigma^2.$$

Standard assumptions for the multiple regression model

Assumption MLR.1

Linear in Parameters

The model in the population can be written as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + u, \quad [3.31]$$

where $\beta_0, \beta_1, \dots, \beta_k$ are the unknown parameters (constants) of interest and u is an unobserved random error or disturbance term.

Assumption MLR.2

Random Sampling

We have a random sample of n observations, $\{(x_{i1}, x_{i2}, \dots, x_{ik}, y_i): i = 1, 2, \dots, n\}$, following the population model in Assumption MLR.1.

Standard assumptions for the multiple regression model

Assumption MLR.3

No Perfect Collinearity

In the sample (and therefore in the population), none of the independent variables is constant, and there are no exact linear relationships among the independent variables.

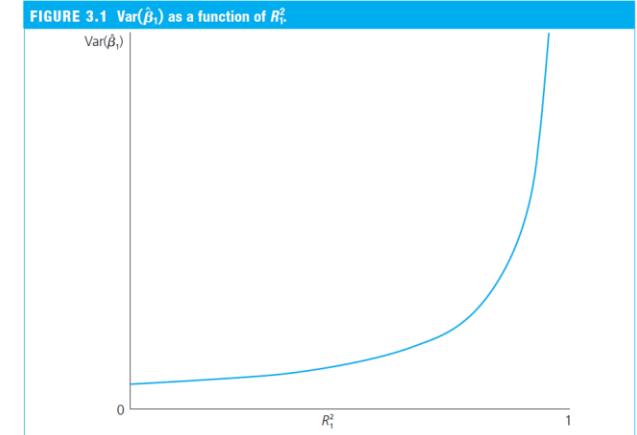
1. The assumption only rules out **perfect collinearity/correlation** between explanatory variables; **imperfect correlation is allowed**
2. If an explanatory variable is a perfect linear combination of other explanatory variables it is superfluous and may be eliminated
3. MLR.3 fails if $n < k + 1$. Intuitively, this makes sense: to estimate $k + 1$ parameters, we need at least $k + 1$ observations.

Detecting multicollinearity

Multicollinearity may be detected through [Variance Inflation Factors](#):

$$VIF_j = 1/(1 - R_j^2)$$

$$\text{Var}(\hat{\beta}_j) = \frac{\sigma^2}{\text{SST}_j(1 - R_j^2)} \longrightarrow \text{Var}(\hat{\beta}_j) = \frac{\sigma^2}{\text{SST}_j} \cdot \text{VIF}_j$$



As an arbitrary rule of thumb, the variance inflation factor should not be larger than 10

Standard assumptions for the multiple regression model (cont.)

Assumption MLR.4

Zero Conditional Mean

The error u has an expected value of zero given any values of the independent variables. In other words,

$$E(u|x_1, x_2, \dots, x_k) = 0. \quad [3.36]$$

- The value of the **explanatory variables** must contain no information about the mean of the unobserved factors
- In a multiple regression model, the **zero conditional mean assumption** is much **more likely to hold** because fewer things end up in the error.

Theorem 3.1 (Unbiasedness of OLS)

**THEOREM
3.1**

UNBIASEDNESS OF OLS

Under Assumptions MLR.1 through MLR.4,

$$E(\hat{\beta}_j) = \beta_j, j = 0, 1, \dots, k, \quad [3.37]$$

for any values of the population parameter β_j . In other words, the OLS estimators are unbiased estimators of the population parameters.



Unbiasedness is an **average property in repeated samples**;
In a given sample, the estimates may still be far away from the true values!

Standard assumptions for the multiple regression model (cont.)

Assumption MLR.5

Homoskedasticity

The error u has the same variance given any value of the explanatory variables. In other words,
 $\text{Var}(u|x_1, \dots, x_k) = \sigma^2$.

- The value of the explanatory variables must contain no information about the **variance** of the unobserved factors
- Example: Wage equation

$$\text{Var}(u_i|educ_i, exper_i, tenure_i) = \sigma^2$$

This assumption may also be hard to justify in many cases

THEOREM
3.2

SAMPLING VARIANCES OF THE OLS SLOPE ESTIMATORS

Under Assumptions MLR.1 through MLR.5, conditional on the sample values of the independent variables,

$$\text{Var}(\hat{\beta}_j) = \frac{\sigma^2}{\text{SST}_j(1 - R_j^2)'} \quad [3.51]$$

for $j = 1, 2, \dots, k$, where $\text{SST}_j = \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2$ is the total sample variation in x_j , and R_j^2 is the R -squared from regressing x_j on all other independent variables (and including an intercept).

The sampling variability of the estimated regression coefficients depends on 4 things:

1. Variability of the unobserved factors (σ^2)
2. Variation in the explanatory variable $\text{var}(X_j)$ or SST_j
3. Number of observations n
4. Linear relationships among the independent variables (R^2)

Testing for Heteroskedasticity

There are many tests for heteroskedasticity; two popular:

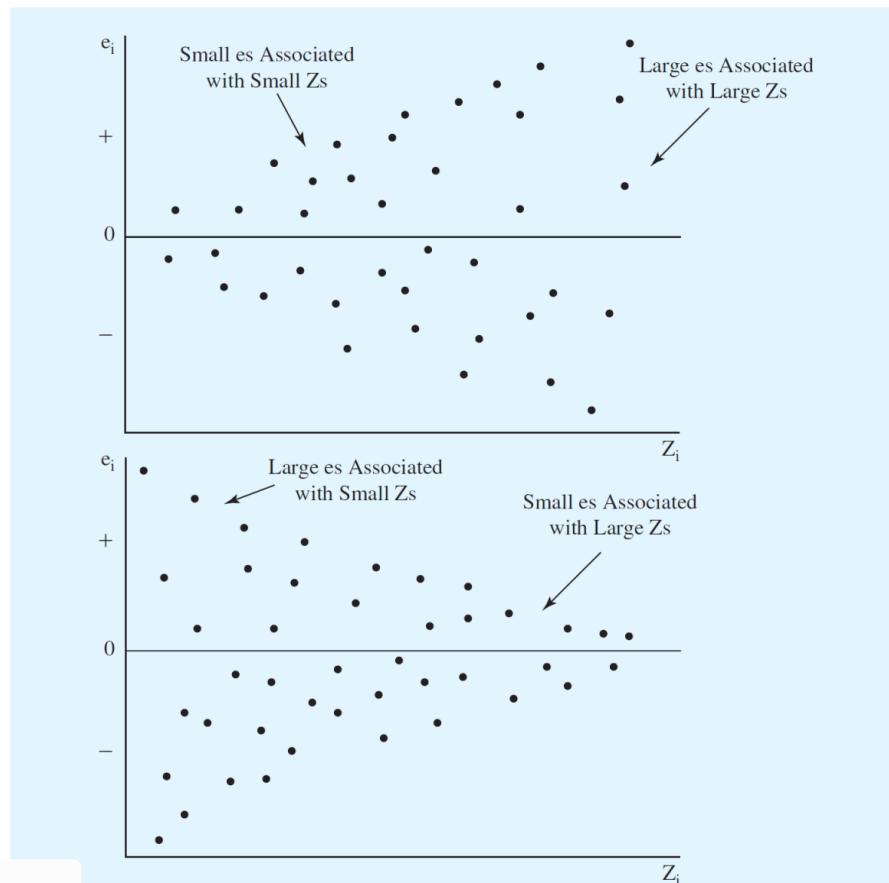
- Breusch-Pagan test
- White test

Before testing for heteroskedasticity, start with asking:

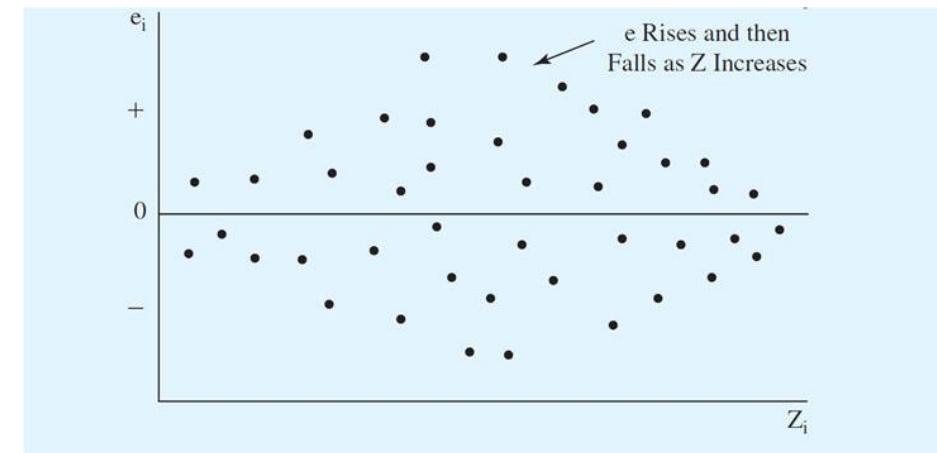
1. Are there any obvious specification errors?
2. Are there any early warning signs of heteroskedasticity?
3. Does a graph of the residuals show any evidence of heteroskedasticity?

Testing for Heteroskedasticity (cont'd)

Eyeballing Residuals for Possible Heteroskedasticity



If you plot the residuals of an equation with respect to a potential explanatory variable Z , a **pattern** in the residuals is an indication of possible **heteroskedasticity**.



The Breusch-Pagan Test for Heteroskedasticity:

Steps:

1. Estimate the model $y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_kx_k + u$ by OLS, as usual. Obtain the squared OLS residuals \hat{u}
2. Run the regression in $\hat{u}^2 = \delta_0 + \delta_1x_1 + \delta_2x_2 + \dots + \delta_kx_k + \text{error}$ Keep the R-squared from this regression $R_{\hat{u}^2}^2$
3. Form either the **F statistic** or the **LM statistic** and compute the p -value. If the p -value is sufficiently small, that is, below the chosen significance level, then we reject the **null hypothesis of homoskedasticity**.

$$H_0 : \text{Var}(u|x_1, x_2, \dots, x_k) = \text{Var}(u|x) = \sigma^2 \longrightarrow H_0 : \delta_1 = \delta_2 = \dots = \delta_k = 0$$

Regress squared residuals on all explanatory variables and test whether this regression has explanatory power.

$$F = \frac{R_{\hat{u}^2}^2/k}{1 - R_{\hat{u}^2}^2/(n - k - 1)}$$
$$LM = n \cdot R_{\hat{u}^2}^2 \sim \chi_k^2$$

A large **F statistic** or a large **Lagrange multiplier** statistic, (LM) lead to rejection of the null hypothesis.

The White Test for Heteroskedasticity

Steps:

1. Estimate the model $y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_kx_k + u$ by OLS, as usual. Obtain the squared OLS residuals \hat{u}
2. Run the regression in $\hat{u}^2 = \delta_0 + \delta_1x_1 + \delta_2x_2 + \delta_3x_3 + \delta_4x_1^2 + \delta_5x_2^2 + \delta_6x_3^2 + \delta_7x_1x_2 + \delta_8x_1x_3 + \delta_9x_2x_3 + error.$ Keep the R-squared from this regression $R_{\hat{u}^2}^2$
3. Form either the **F statistic** or the **LM statistic** and compute the *p*-value. If the *p*-value is sufficiently small, that is, below the chosen significance level, then we reject the **null hypothesis of homoskedasticity**.

$$H_0 : Var(u|x_1, x_2, \dots, x_k) = Var(u|x) = \sigma^2 \longrightarrow H_0 : \delta_1 = \delta_2 = \dots = \delta_9 = 0$$

Regress squared residuals on all explanatory variables, **their squares**, and **interactions**
(here: example for k=3)

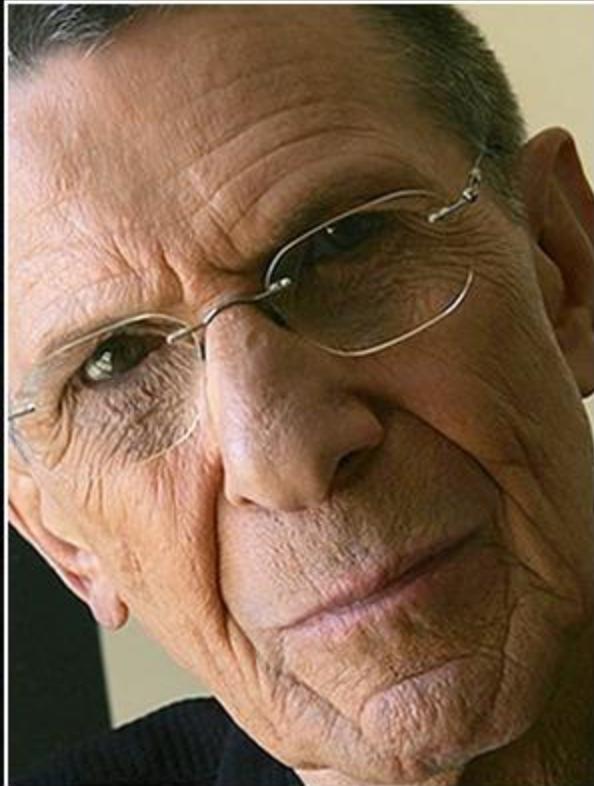
$$F = \frac{R_{\hat{u}^2}^2/k}{1 - R_{\hat{u}^2}^2/(n - k - 1)}$$
$$LM = n \cdot R_{\hat{u}^2}^2 \sim \chi_k^2$$

A large **F statistic** or a large **Lagrange multiplier** statistic, (LM) lead to rejection of the null hypothesis.

Remedies for Heteroskedasticity

- ❑ If heteroskedasticity is found, the first thing to do is examine the equation carefully for specification errors.
- ❑ If there are no obvious specification errors, the heteroskedasticity is probably pure in nature and one of the following remedies should be considered.
 1. Redefining the Variables
 2. Heteroskedasticity-Corrected Standard Errors
 3. Weighted Least Square Estimation!

→ Question of the day!



You proceed from a false
assumption: I have no ego to bruise.

— *Leonard Nimoy* —

AZ QUOTES

A

Performance Metrics:

- ① Feature Selection (How many X ?)

$$\begin{array}{l} \text{ok } R^2 \rightarrow \text{adj } R^2 \\ \text{MSE} \rightarrow \text{BIC, AIC} \end{array}$$

↑ Penalizing as $K \uparrow$

X_1, X_2, \dots, X_K

Statistical learning approach
usually in-sample performance

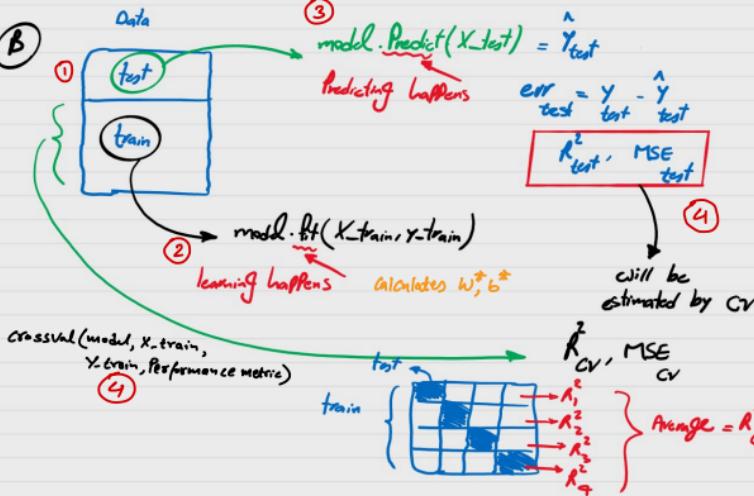
- ② Tuning hyperparameters:



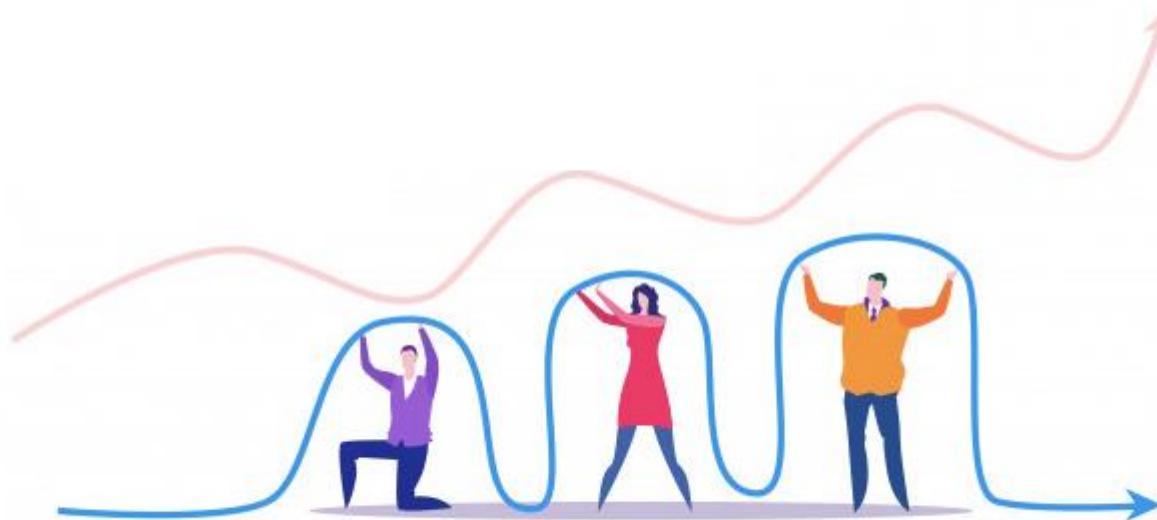
needs to be estimated by Cross validation.

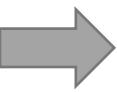
ML approach
Always test set

B

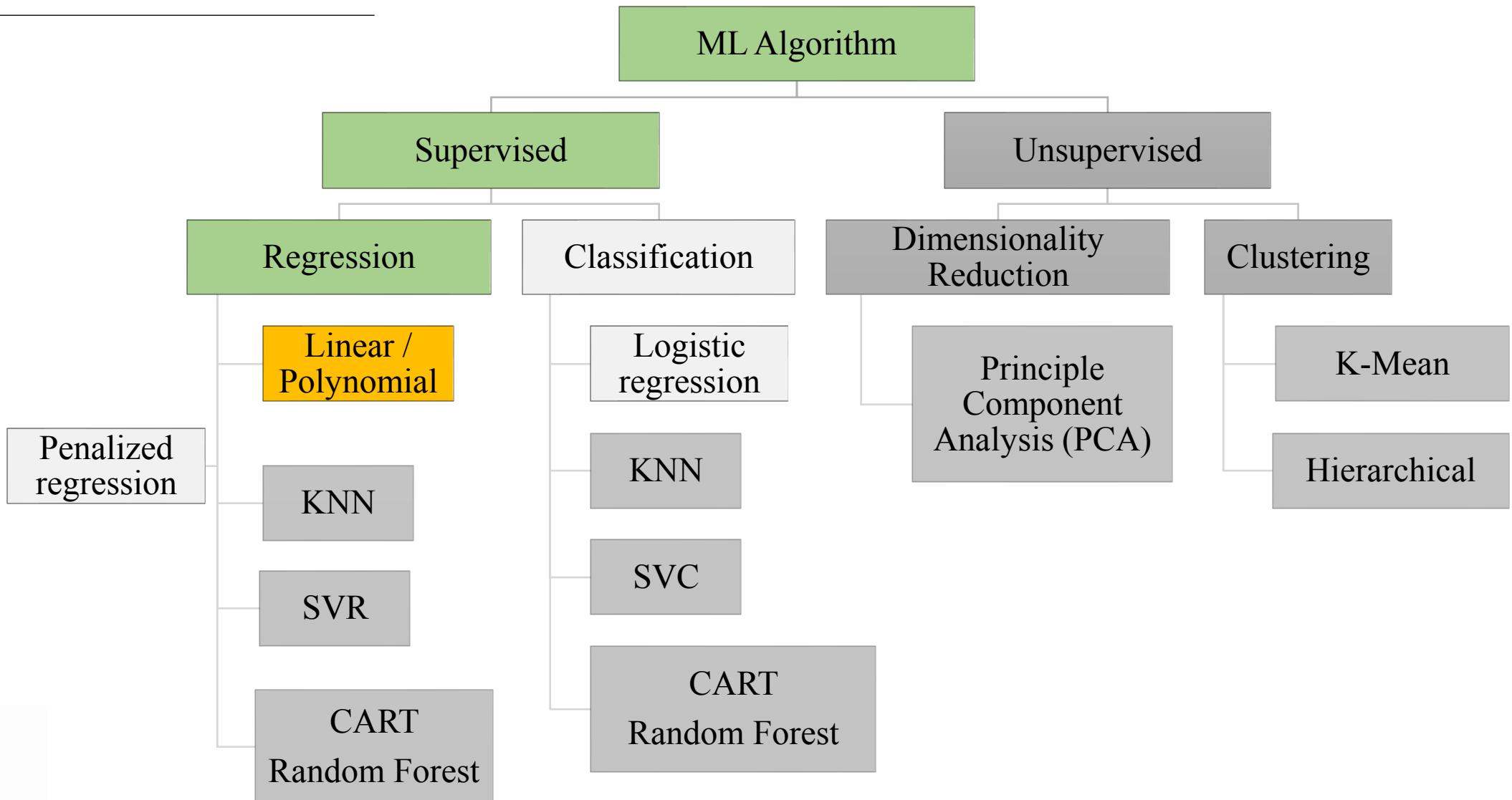


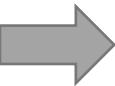
Class 7- Polynomial Regression





Road map





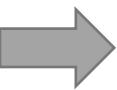
Polynomial regression model

The polynomial regression model is a special case of multiple linear regression models!

- Create new variables $X_1 = X$, $X_2 = X^2$, ... etc and then treat as **multiple linear regression**.
- Not really interested in the coefficients; more interested in the fitted function!

$$\hat{f}(X) = f_{\mathbf{w}, b}(X) = b + w_1 x + w_2 x^2 + \dots + w_d x^d$$

- \mathbf{W} is a d -dimensional vector of parameters
- b is a real number
- d is the polynomial degree of the model (we either fix the d at some reasonably low value, else use **cross-validation** to choose d)

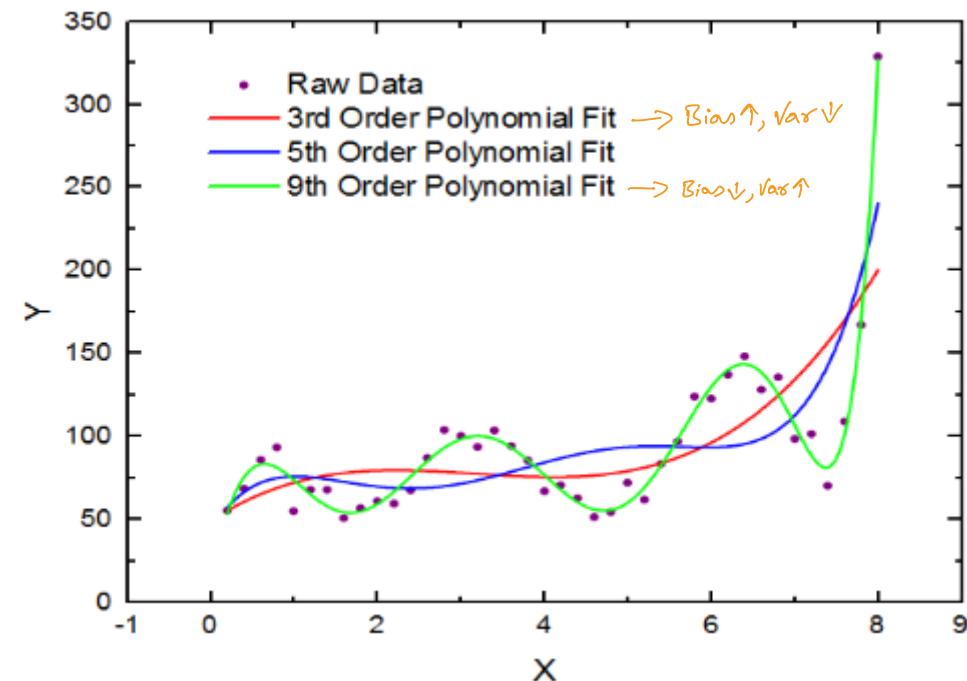


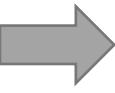
The optimization problem

The optimization problem is defined as:

$$\text{Min}_{\mathbf{w}, \mathbf{b}} \text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2$$

- We use the same **loss function** as in linear regression!
- The solution to this optimization problem is \mathbf{w}^* and \mathbf{b}^*
- Now we can make predictions!



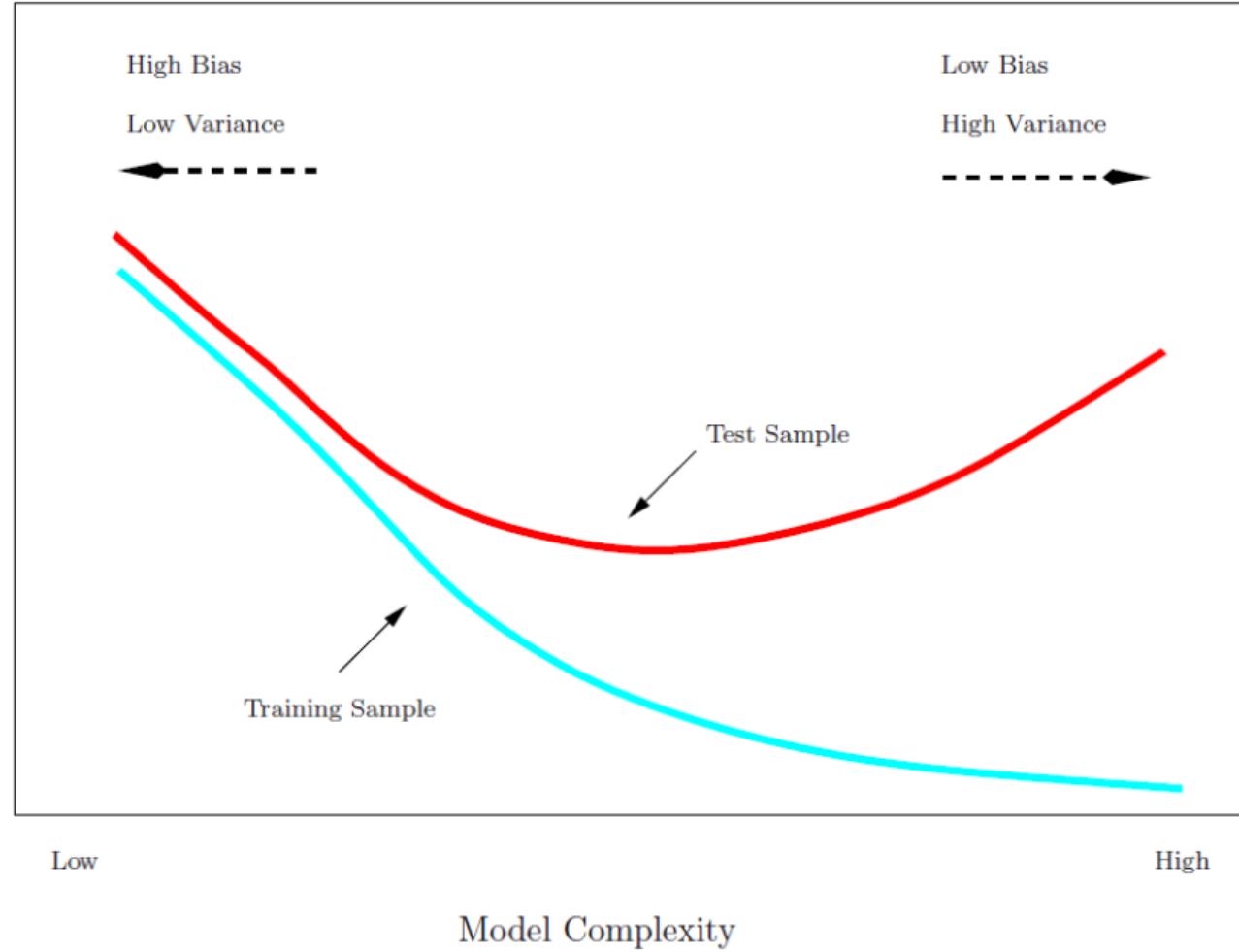


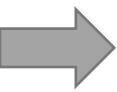
Polynomial Regression Evaluation Metrics

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE}$$

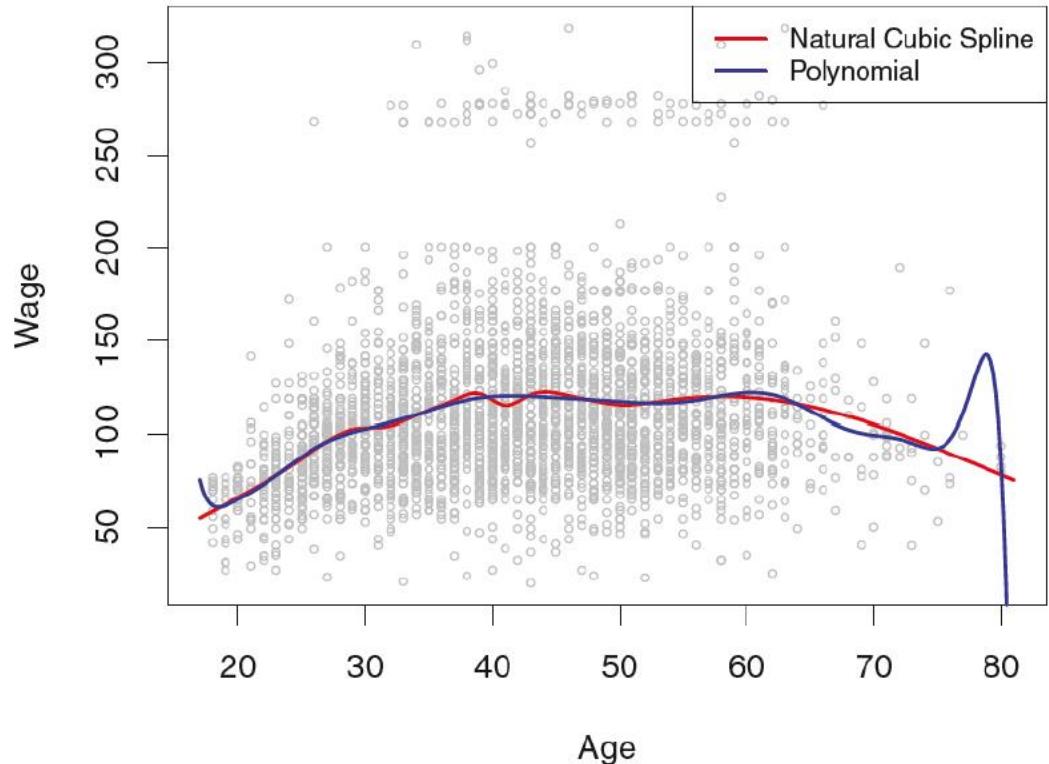
Prediction Error

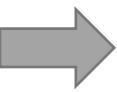




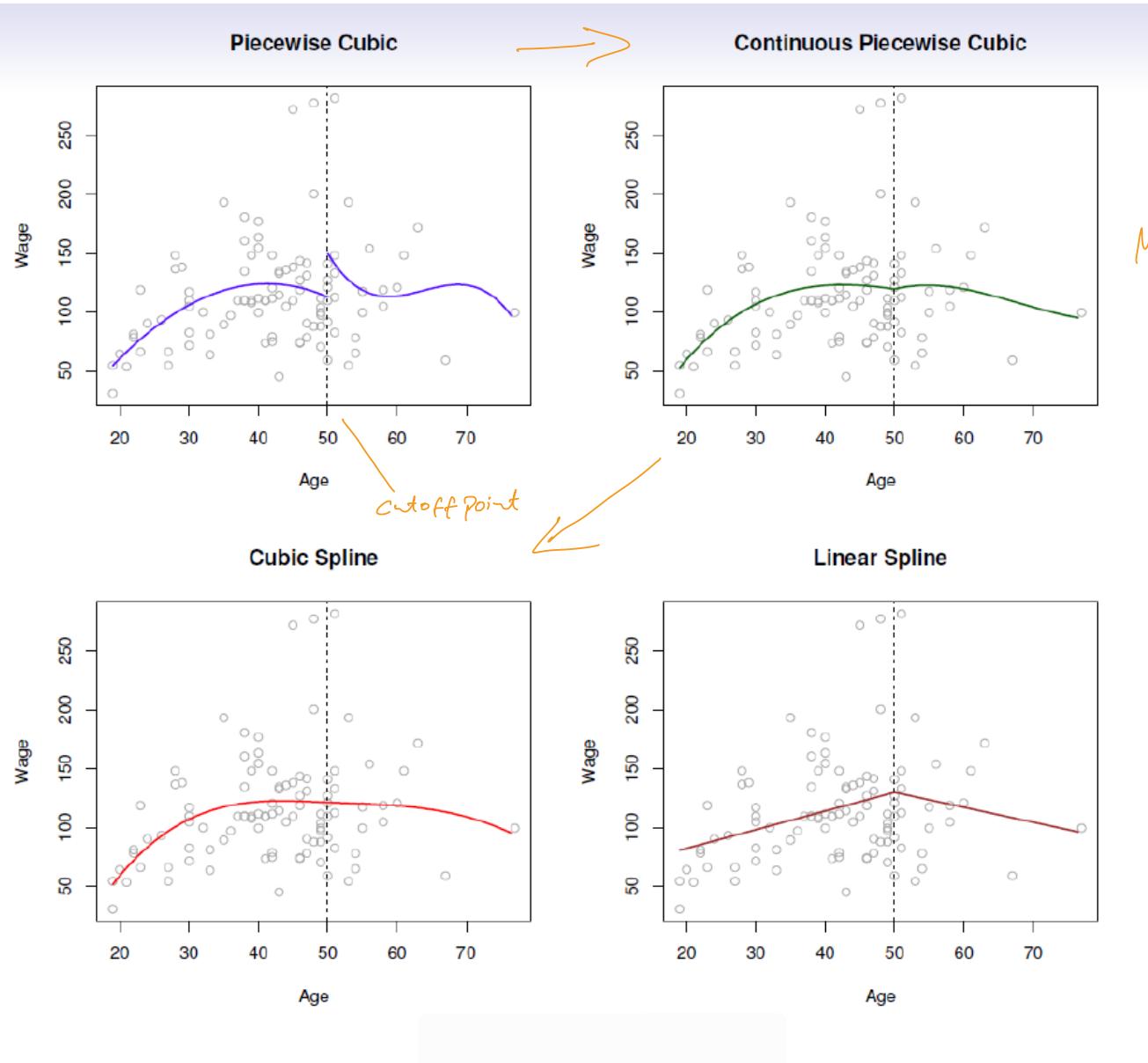
Caveats!

- In general quadratic loss functions are **sensitive to outliers**
- Polynomials have **notorious tail behavior**
- Polynomials are **global fit!**
- **Solution:** piecewise polynomial, splines and local regressions.

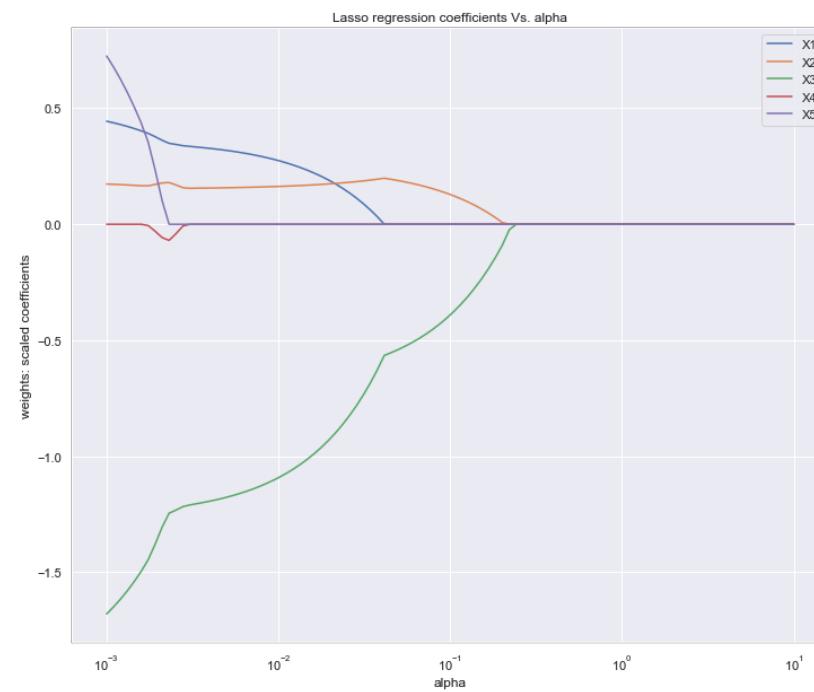
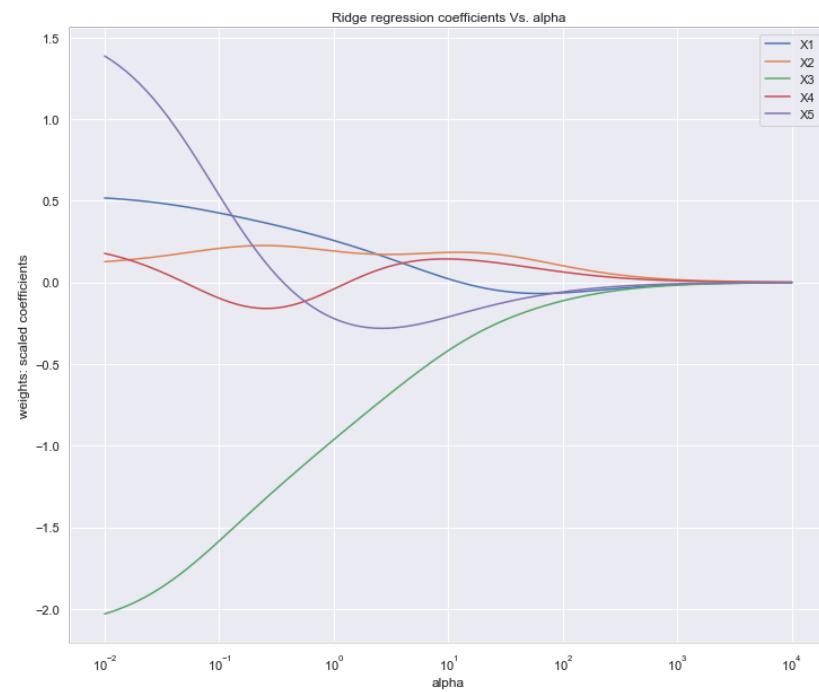


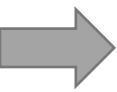


Piecewise polynomials and splines!

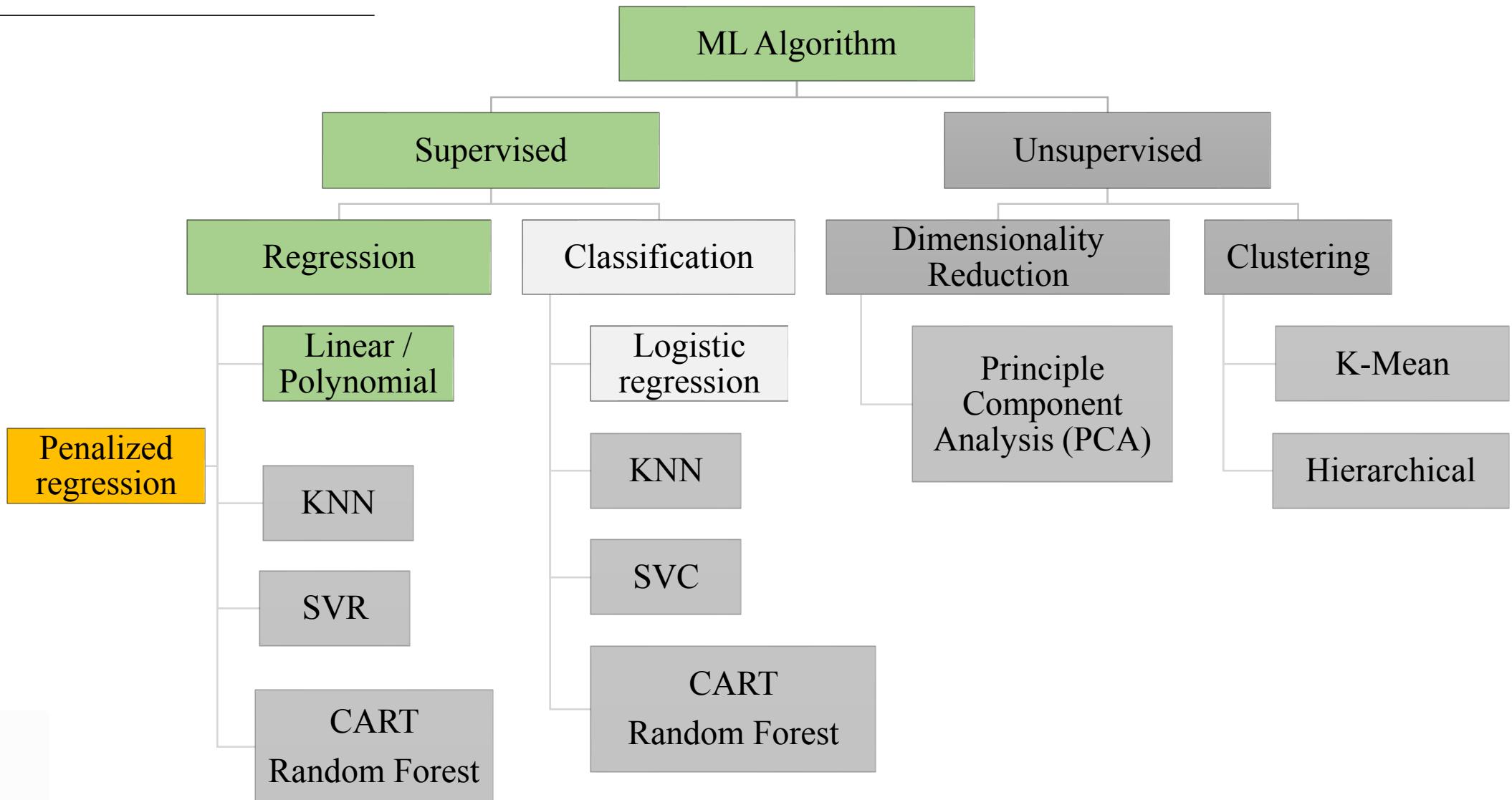


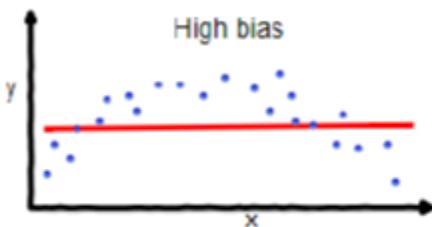
Class 8- Regularization (Penalized Regression) (Ridge, Lasso and Elastic Net)



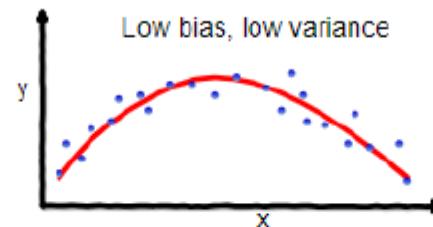


Road map

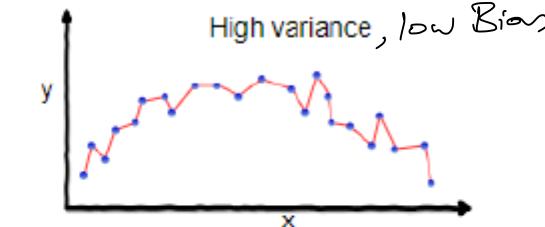




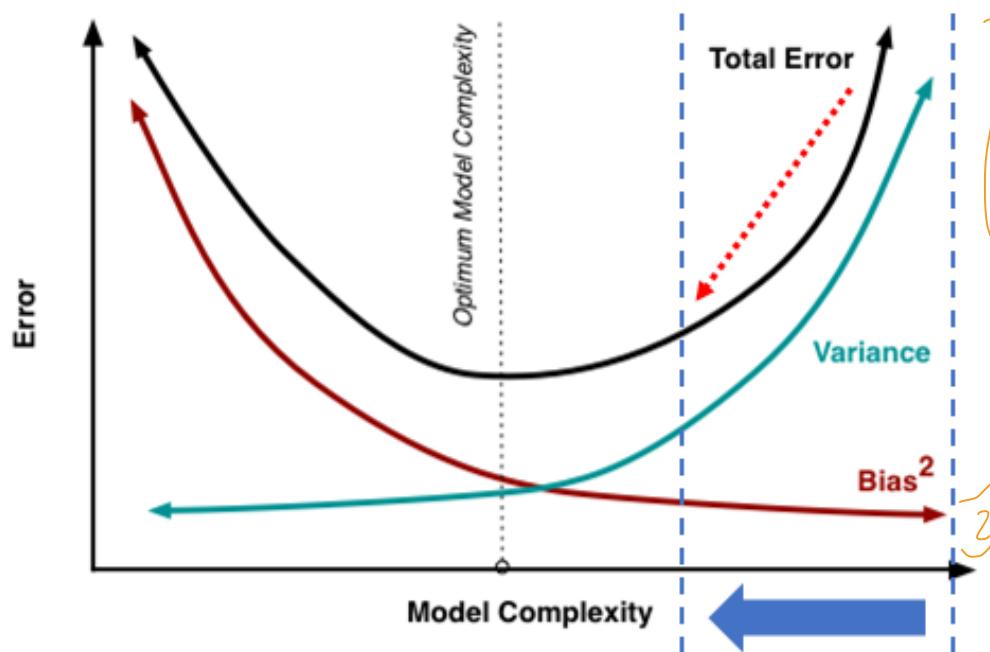
underfitting



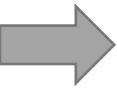
Good balance



overfitting



Regularization / Penalized regression



Norms

- In mathematics, the **norm** of a vector is its **length**.
- In regression analysis, to fit our linear model, we need a measure of **mismatch!**
- Our vector is error at each training data. **We want to measure the length of error!**

- **L1** norm: Least absolute errors

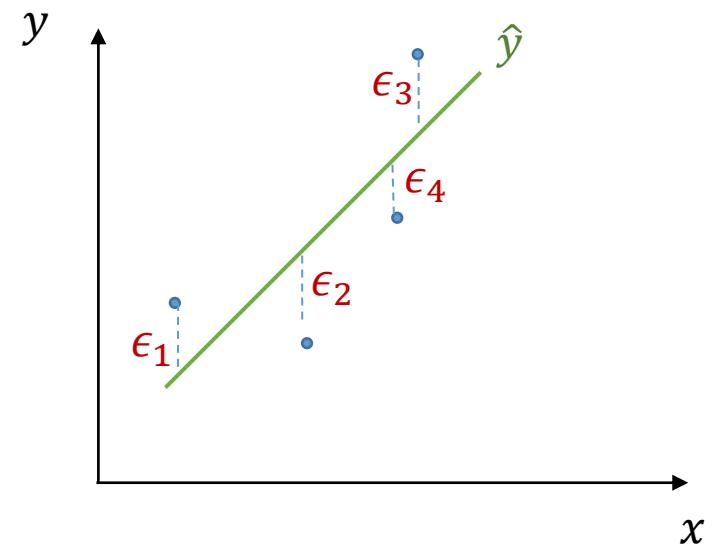
Manhattan norm

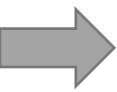
$$L^1 = \sum_i |\epsilon_i|$$

- **L2** norm: Least squares

Euclidean norm

$$L^2 = \sum_i (\epsilon_i)^2$$



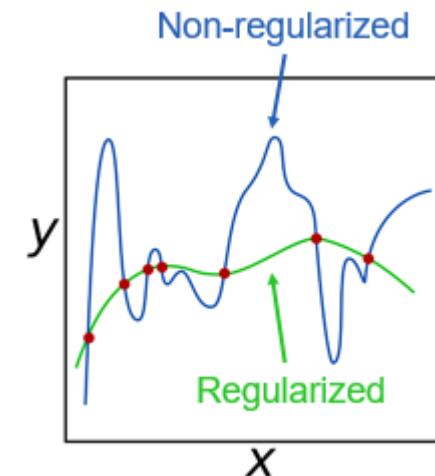


Regularization

- In machine learning there are often **many features** (usually **correlated** with each other). This can lead to **overfitting** and models that are **unnecessarily complex**.
- **Regularization** force the learning algorithm to build a **less complex model**. In practice, that often leads to **slightly higher bias** but **significantly reduces the variance**.

- ✓ The two most widely used types of regularization are called **L1** and **L2** regularization. The idea is quite simple. To create a regularized model, we modify the loss function by adding a penalizing term whose value is higher when the model is more complex.

$$\text{Min}_{\mathbf{w}, \mathbf{b}} (\text{MSE} + \text{penalty}) = \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \text{penalty}(\mathbf{w}) \right]$$



→ Penalized regression

$$\text{Min}_{\mathbf{w}, \mathbf{b}} (\text{MSE} + \text{penalty}) = \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \text{penalty}(\mathbf{w}) \right]$$

- Penalized regression is useful for reducing a large number of features to a manageable set and for making good predictions especially where features are correlated (i.e., when classical linear regression breaks down).
- Penalized regression can be used to avoid overfitting.
- To use the penalized regression, we need to first standardize the features. This will allow us to compare the magnitudes of regression coefficients for the feature variables.

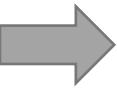
- 1) Ridge regression $\xrightarrow{\text{Penalty } (\ell^2)}$
- 2) LASSO regression $\xrightarrow{\ell^1}$
- 3) Elastic Net regression $\xrightarrow{\ell^1 \& \ell^2}$

The only difference is in the penalty term

Good idea to standardize both X & y .
Simple ones are normalisation $\frac{x-\mu}{\sigma}$.

Part I

Ridge Regression



1) Ridge regression

$$\begin{aligned} \text{Min}_{\mathbf{w}, \mathbf{b}} (\text{MSE} + \text{penalty}) &= \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \underbrace{\text{penalty}(\mathbf{w})}_{\ell^2} \right] \\ &= \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \lambda \sum_{j=1}^D w_j^2 \right] \end{aligned}$$

- Ridge regression uses **L2** norm.
- The shrinkage penalty has the effect of shrinking the estimates of w_j towards zero. *but never equal to zero.*
- The tuning parameter λ serves to control the relative impact of the penalty term on the regression coefficient estimates. *$\lambda \uparrow$ forces to regularize more strongly i.e. scale down on features.*
- Selecting a good value for λ is critical; cross-validation is used for this.
- It is best to apply ridge regression after variable **standardization**.

The true model is:

$$y = f(x) = x + 2x^2 - 3x^3 + \epsilon$$

Imposed functional form:

$$\hat{y} = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + w_5x^5$$

Ridge:

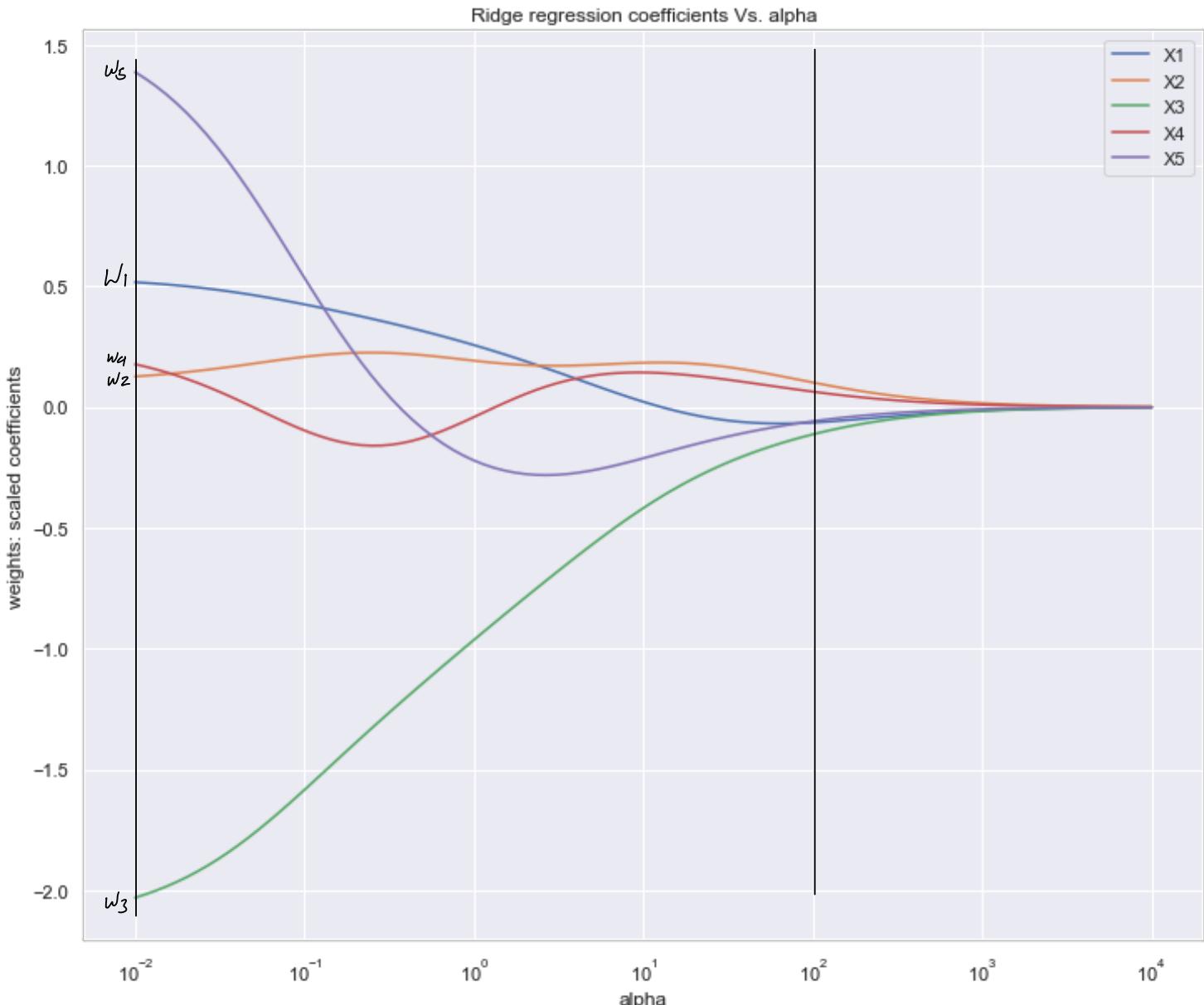
$\lambda=0 \Rightarrow$ linear Regression

$\lambda=0.01$

$$\hat{y} = 0.5x + 0.1x^2 - 2x^3 + 0.2x^4 + 1.4x^5$$

$\lambda=100$

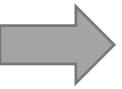
$$\hat{y} = -0.1x + 0.2x^2 - 0.2x^3 + 0.1x^4 - 0.2x^5$$



$\lambda=\text{alpha}$ (in Python)

Part II

LASSO Regression



2) LASSO regression

$$\begin{aligned} \text{Min}_{\mathbf{w}, \mathbf{b}} (\text{MSE} + \text{penalty}) &= \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \text{penalty}(\mathbf{w}) \right] \\ &= \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \lambda \sum_{j=1}^D |w_j| \right] \end{aligned}$$

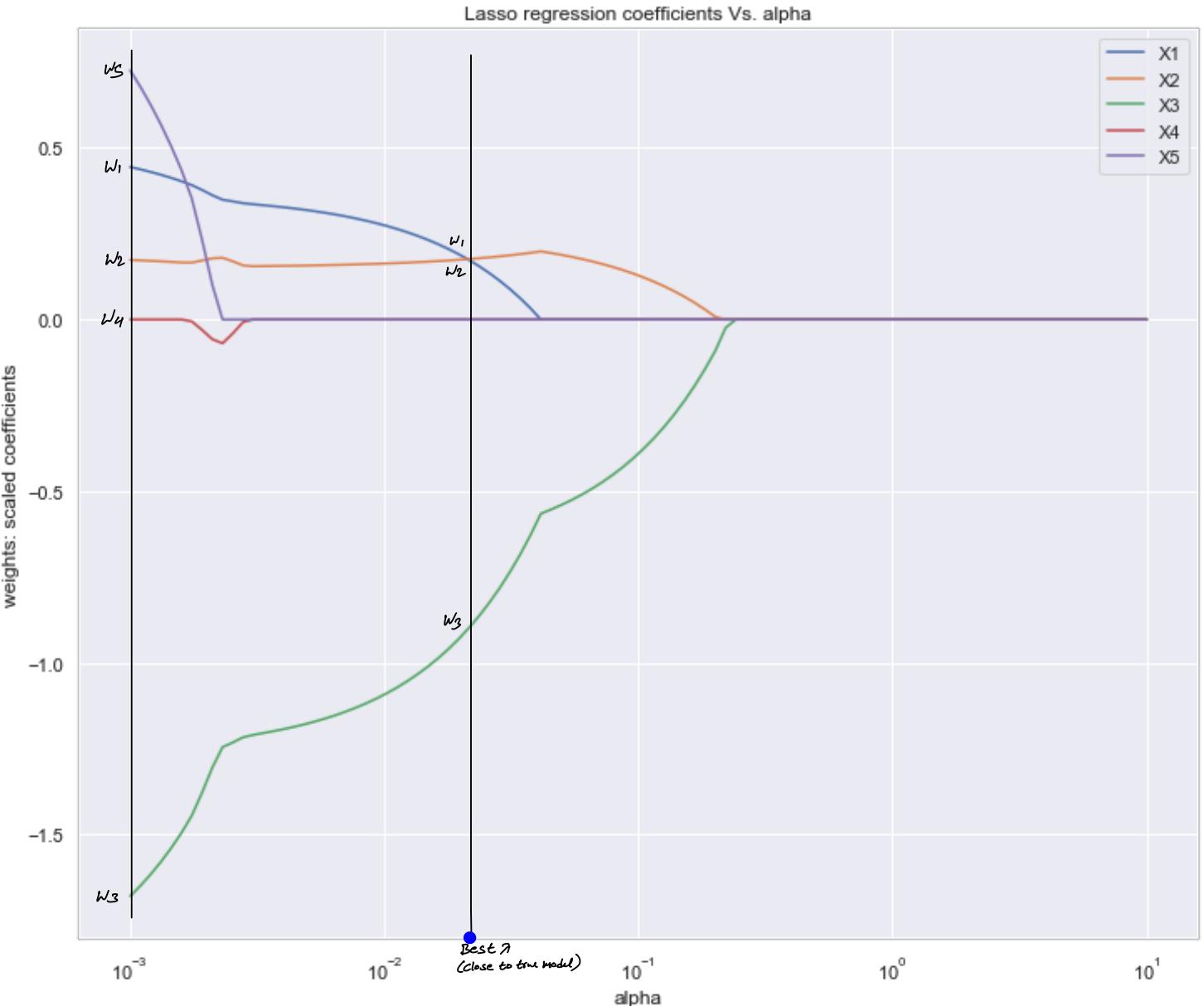
- LASSO stands for “Least Absolute Shrinkage and Selection Operator”
- LASSO regression uses L1 norm.
- LASSO eliminates the least important features from the model, it automatically performs a type of **feature selection**. *w_j can be zero.*
- Selecting a good value for λ is critical; cross-validation is used for this.
- It is best to apply LASSO regression after variable standardization.

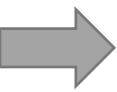
The true model is:

$$y = f(x) = x + 2x^2 - 3x^3 + \epsilon$$

Imposed functional form:

$$\hat{y} = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + w_5x^5$$

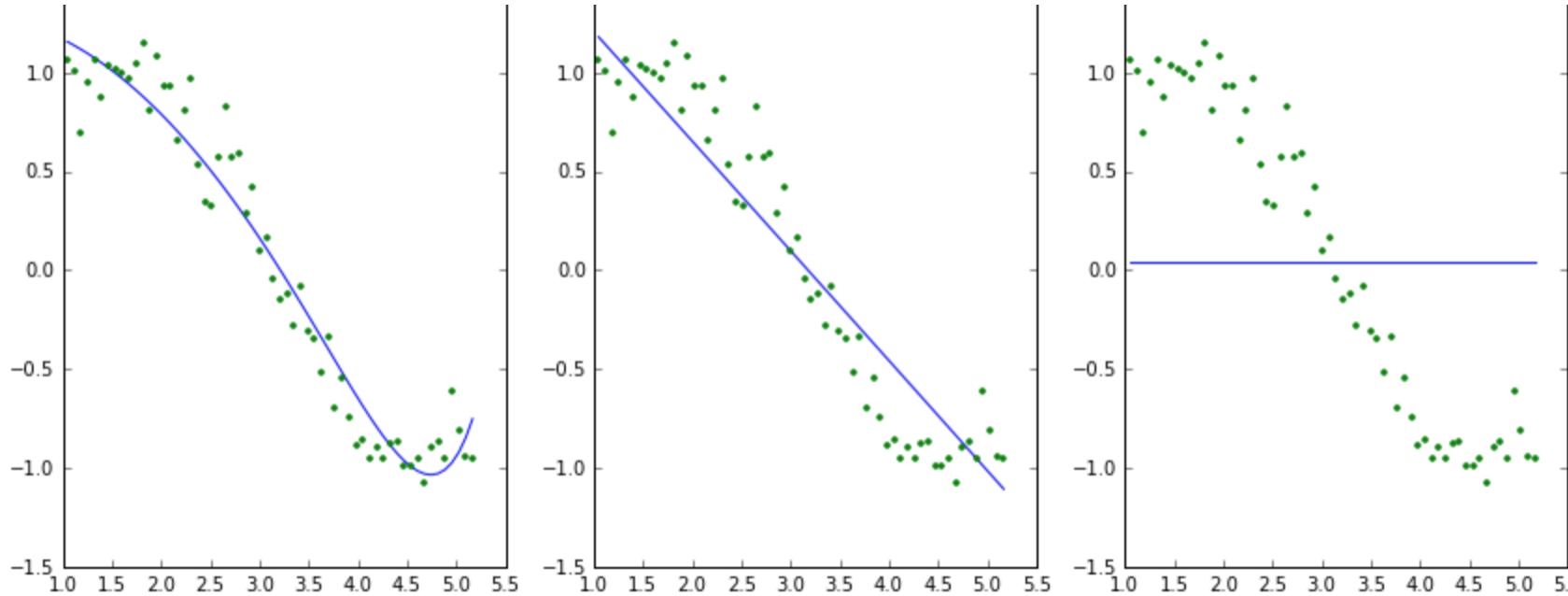


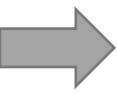


Ridge and LASSO vs Lambda

As λ increases, the model becomes simpler

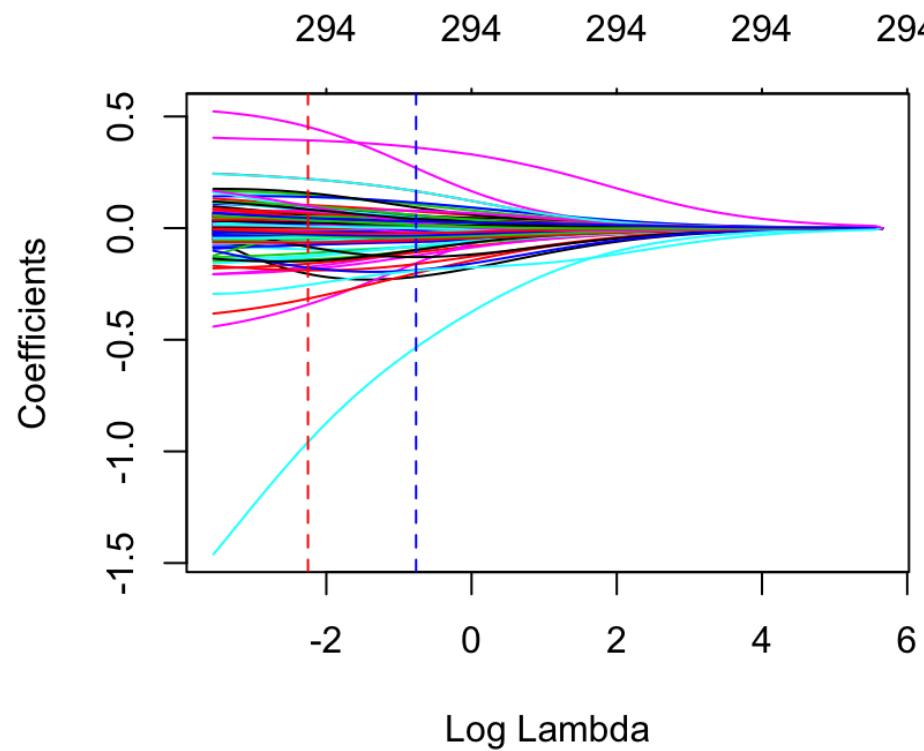
Bias ↑
Variance ↓



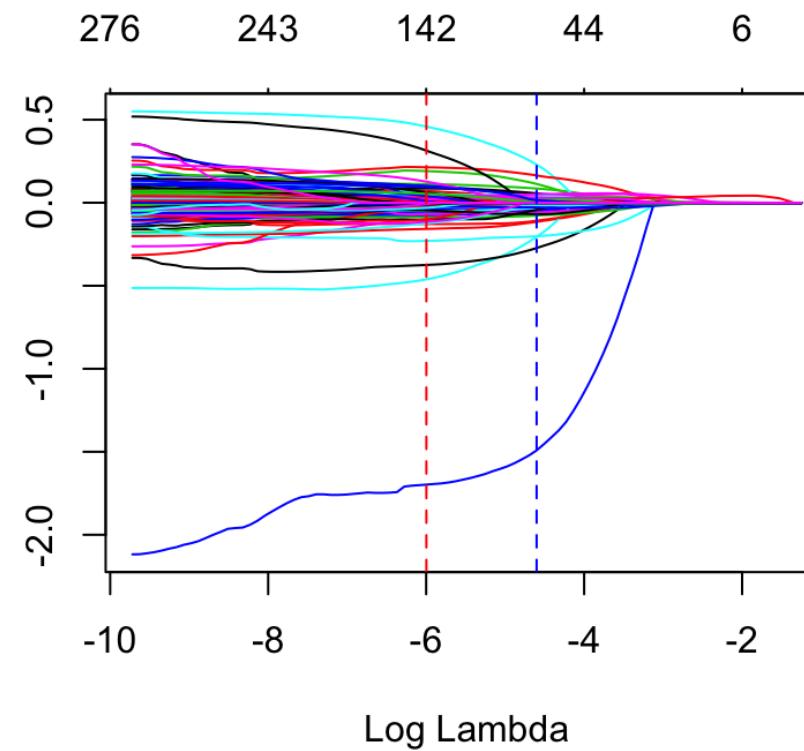


Question of the day: Ridge vs LASSO?

?



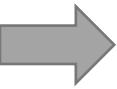
Ridge



LASSO

Part III

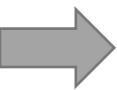
Elastic Net Regression



3) Elastic Net Regression

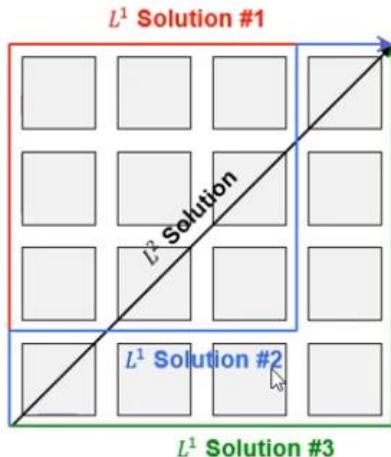
$$\begin{aligned} \text{Min}_{\mathbf{w}, \mathbf{b}} (\text{MSE} + \text{penalty}) &= \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \text{penalty}(w) \right] \\ &= \text{Min} \left[\frac{1}{N} \sum_{i=1}^N (y_i - f_{\mathbf{w}, \mathbf{b}}(X_i))^2 + \lambda_1 \sum_{j=1}^D |w_j| + \lambda_2 \sum_{j=1}^D w_j^2 \right] \end{aligned}$$

- In LASSO some weights are reduced to zero, but others may be quite large. In Ridge, weights are small in magnitude, but they are not reduced to zero.
- In Elastic Net, we may be able to get the **best of both worlds** by making some weights zero while reducing the magnitude of the others.



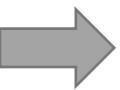
Ridge vs LASSO vs Elastic Net

Property	Ridge	LASSO	Elastic Net
Can shrink the coefficient estimate toward zero?	Yes	Yes	Yes
Can include all the features in the model even with large λ ?	Yes	No	No
Can force some of the coefficient estimates to be exactly = 0? Hence, can be used for <u>feature selection</u> ? Or <u>sparse output</u> ? More explainable?	No	Yes	Yes
Is robust : resistant to <u>outliers</u> ?	No	Yes	Not very
No Analytical solution i.e., requires gradient descent?	No	Yes	Yes
Always unique solution?	Yes	No	Yes



- Ridge has Analytical Sol'n.
- Lasso & Elastic has L' Norm which is NOT differentiable, so No close form Sol'n.

Appendix



LASSO vs Ridge, behind the scene? (optional)

Why is it that the lasso, unlike ridge regression, results in coefficient estimates that are exactly equal to zero?

One can show that the lasso and ridge regression coefficient estimates solve the problems

Min (RSS + Penalty)

Convex
Optimisation

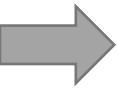
$$\underset{\beta}{\text{minimize}} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^p |\beta_j| \leq s$$

and

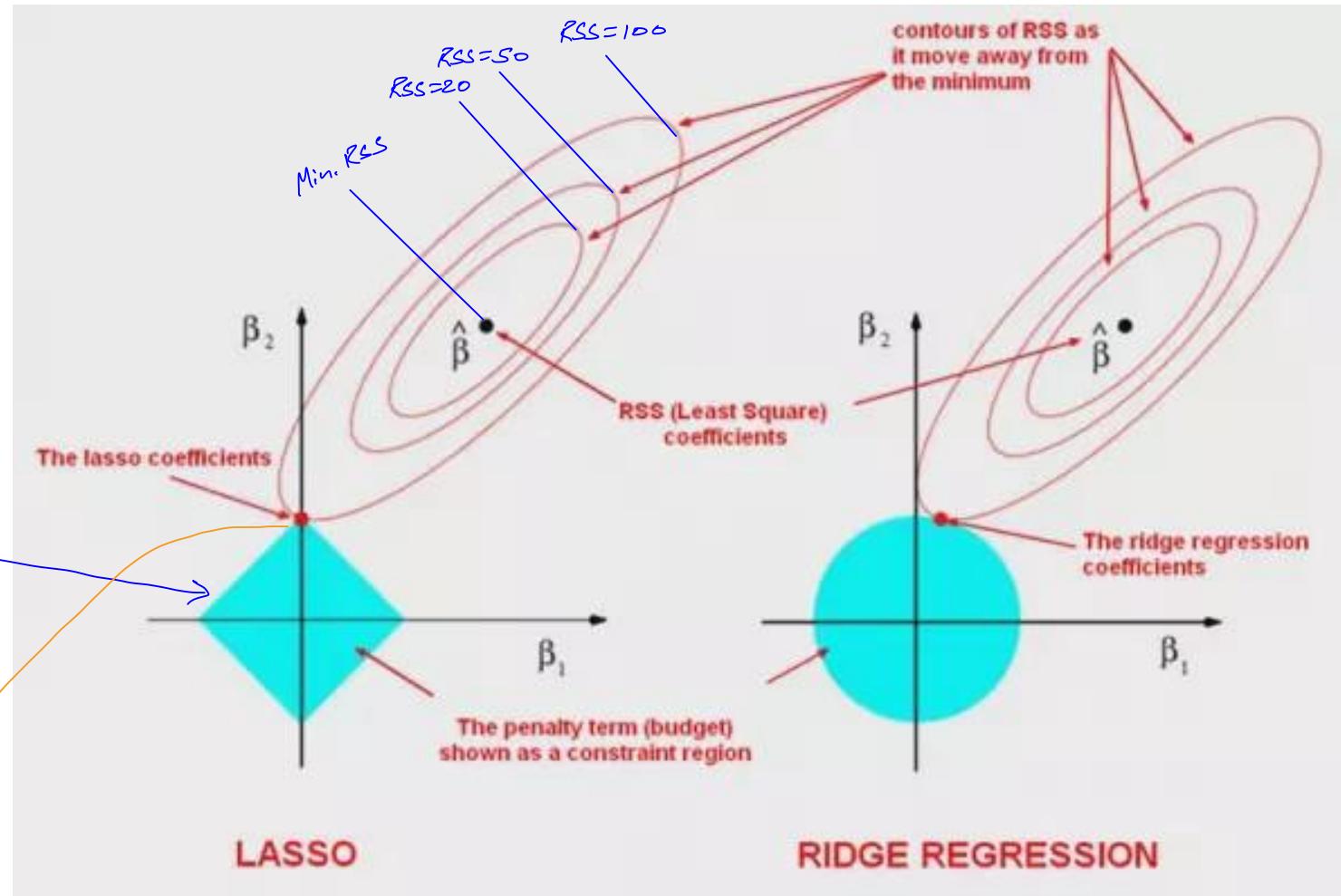
$$\underset{\beta}{\text{minimize}} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^p \beta_j^2 \leq s,$$

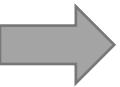
LASSO

Ridge

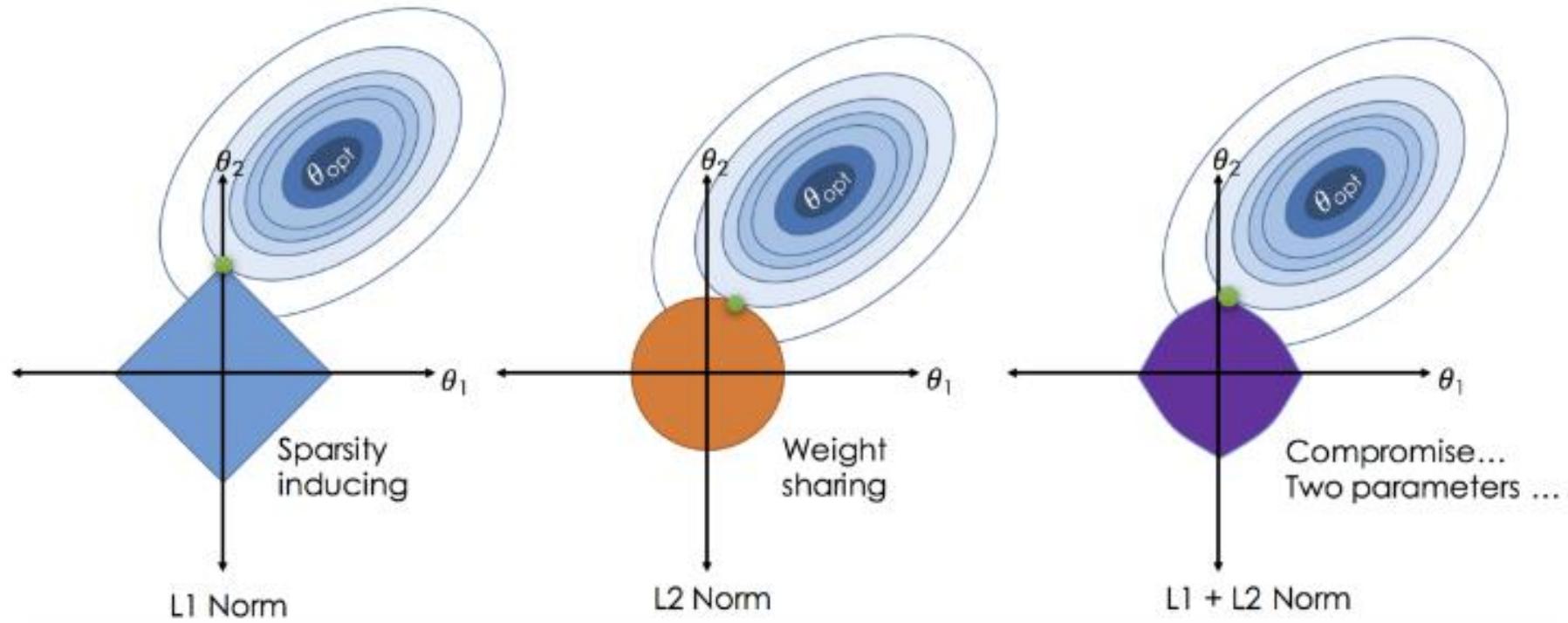


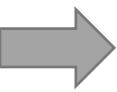
LASSO vs Ridge, behind the scene? (optional)





LASSO vs Ridge vs Elastic Net, behind the scene? (optional)



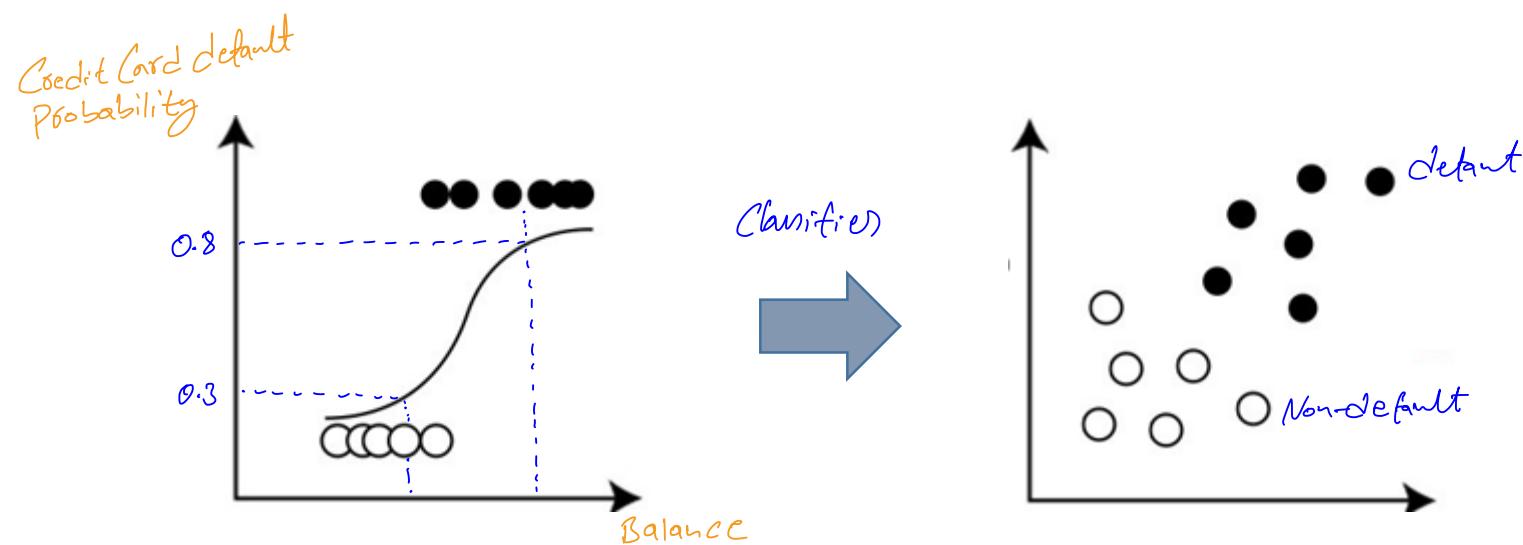


Students' questions

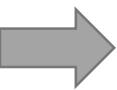
- 1) Where does regularization happen? Entire date set or train set? *Train Set.*
- 2) Why the number of parameters needs to be less than the number of observations? *In System of eq's can't have more unknowns than eq's.*
- 3) Isn't it better to check the multi-collinearity before using penalized regressions? What if Lasso drop them all? *Lasso doesn't drop all, it keeps the most relevant one even in collinearity.*
- 4) Why bother with Ridge and Lasso if ElasticNet is better? *pg 17 Comparison.*
- 5) How to find the optimal lambda? (shrinkage term) *Using cross validation.*
- 6) Is regularization useful for big data or only small sample data? *For Big data*
- 7) What do we put in loss function? What if we had dummy variables?
↳ Penalty Term of weights eg: $\sum |w_j|$
*→ Relevant for lasso:
For two classes its fine.
For multiple classes you have to group them together.*

→ Mostly used when you have 2 classes. For multiple classes use other models.

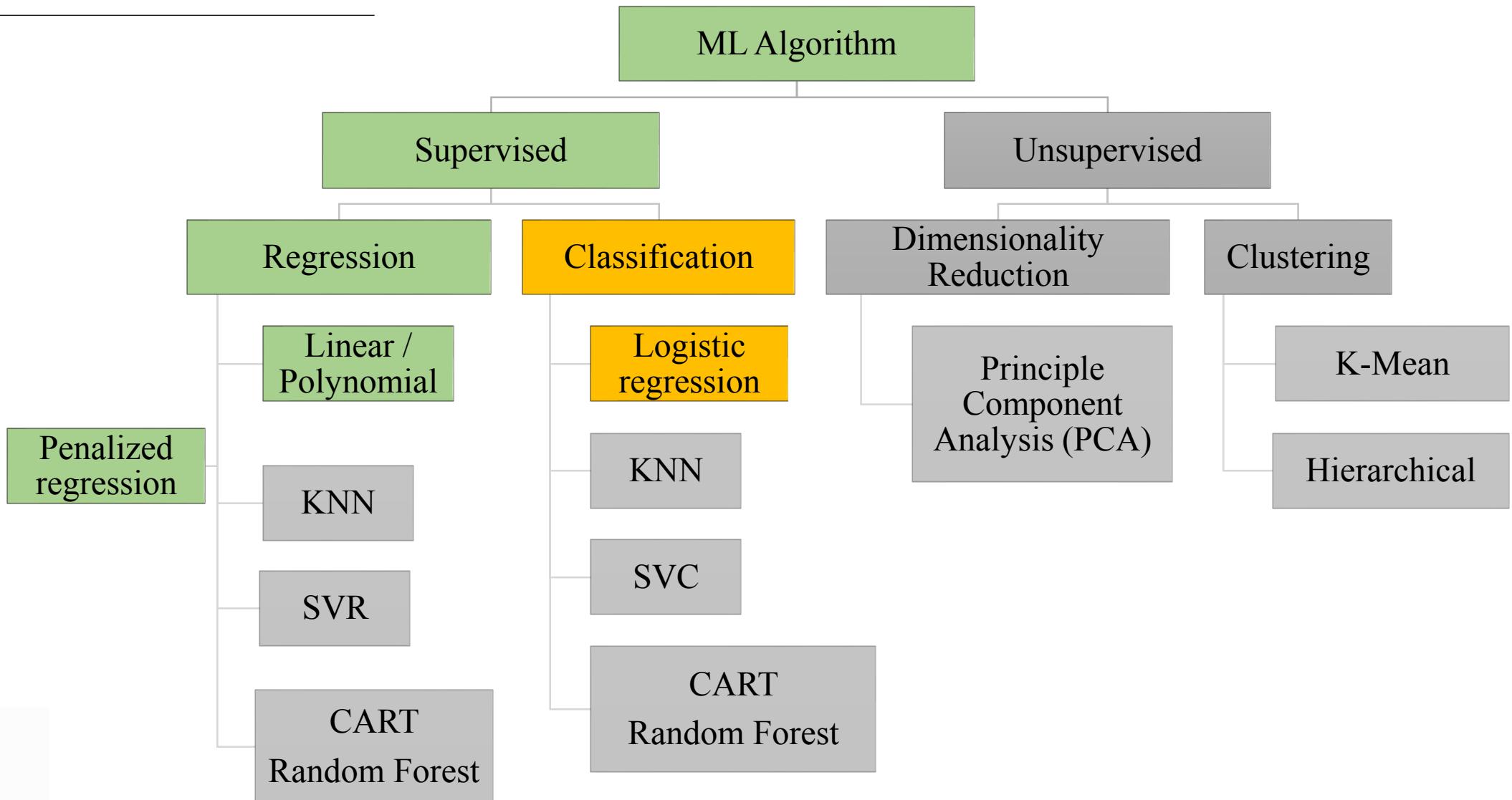
Class 10 – Logistic Regression

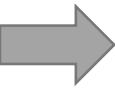


The outcome of Logistic Regression are Probabilities. We use logistic Regression for Classification.



Road map





Topics

Part I

1. Linear probability model (LPM) vs Logistic regression
2. Sigmoid function
3. Logistic regression

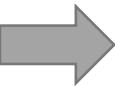
We want to maximize TN & TP, And minimize FN & FP.

Part II

1. Classification performance metrics
 - a) Accuracy,
 - b) Precision,
 - c) Recall,
 - d) F1 score,
 - e) ROC and AUC.

		Predictions	
		0 negative	1 positive
Actual	0 negative	TN	FP
	1 positive	FN	TP

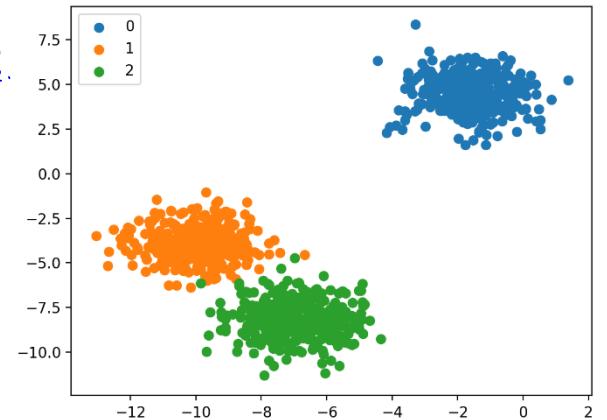
Confusion Matrix for 2 classes

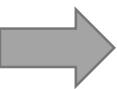


Classification

- Qualitative variables can be either nominal or ordinal.
- Qualitative variables are often referred to as **categorical**.
- **Classification** is the process of predicting categorical variables.
- Classification problems are quite common, perhaps even more than regression problems.
- **Examples:**
 - Financial instrument tranches (investment grade or junk)
 - Online transactions (fraudulent or not)
 - Loan application (approved or denied)
 - Credit card default (default or not)
 - Car insurance customers (high, medium, low risk)

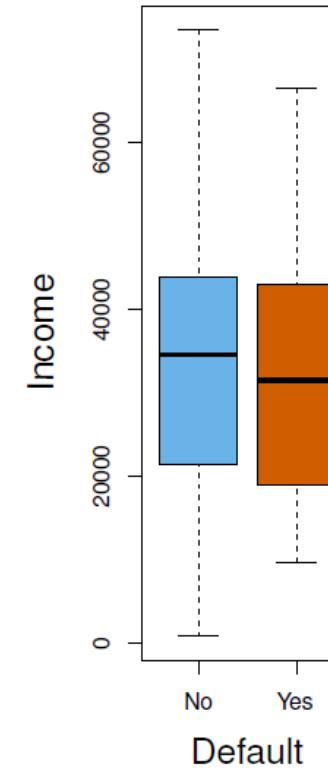
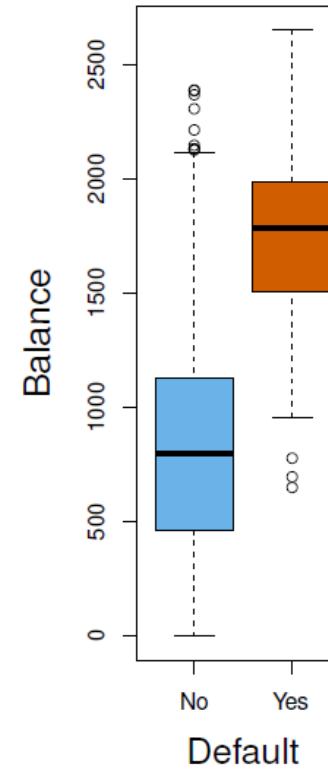
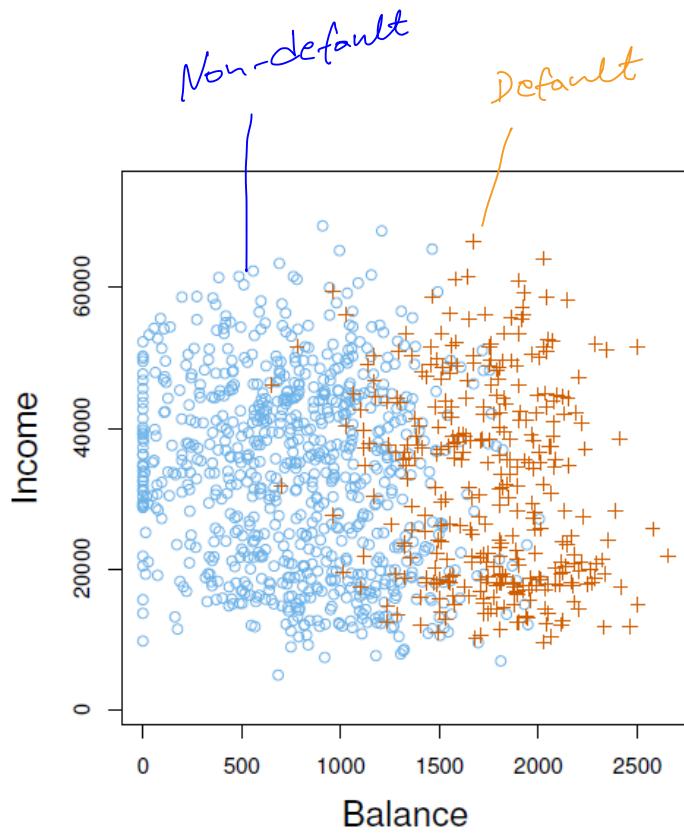
Race, Gender
↑
Order Matters but Magnitude makes no sense.
High, Medium, Low.





Credit card default example *(Very Imbalance data i.e. only 3% in data are Defaulter)*

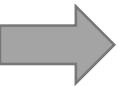
- Goal: Build a **classifier** that performs well in **both** train and test set.



Balance has more classification power than Income.

Part I

Logistic Regression



Linear Probability Model (LPM) vs Logistic Regression

Starting with simple LPM : $y = \beta_0 + \beta_1 bal + \epsilon$ where, $Y = 1$ for default and 0 otherwise.

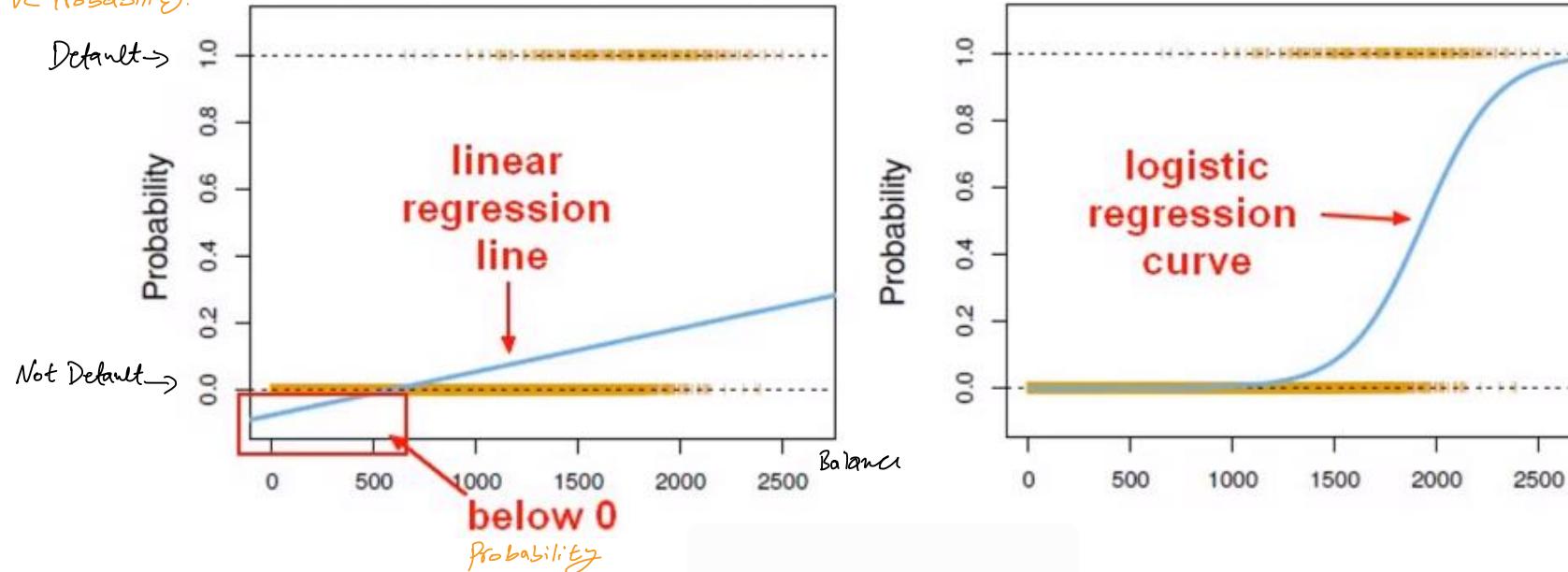
$$E(Y|bal) = \sum_{y_i} y_i P(y_i|bal) \stackrel{?}{=} E(Y = 1|bal) = \Pr(Y = 1|bal)^{\frac{1}{1}} = P(x) = \beta_0 + \beta_1 bal$$

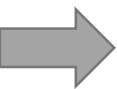
$0 \text{ for } y_i = 0$
 $y_i = 1 \text{ remains}$

\hookrightarrow Probability of default.

- It seems that simple regression is perfect for this task,
- But what are the caveats?

(1) By construction LPM is heteroskedastic.
(2) -ve Probability.

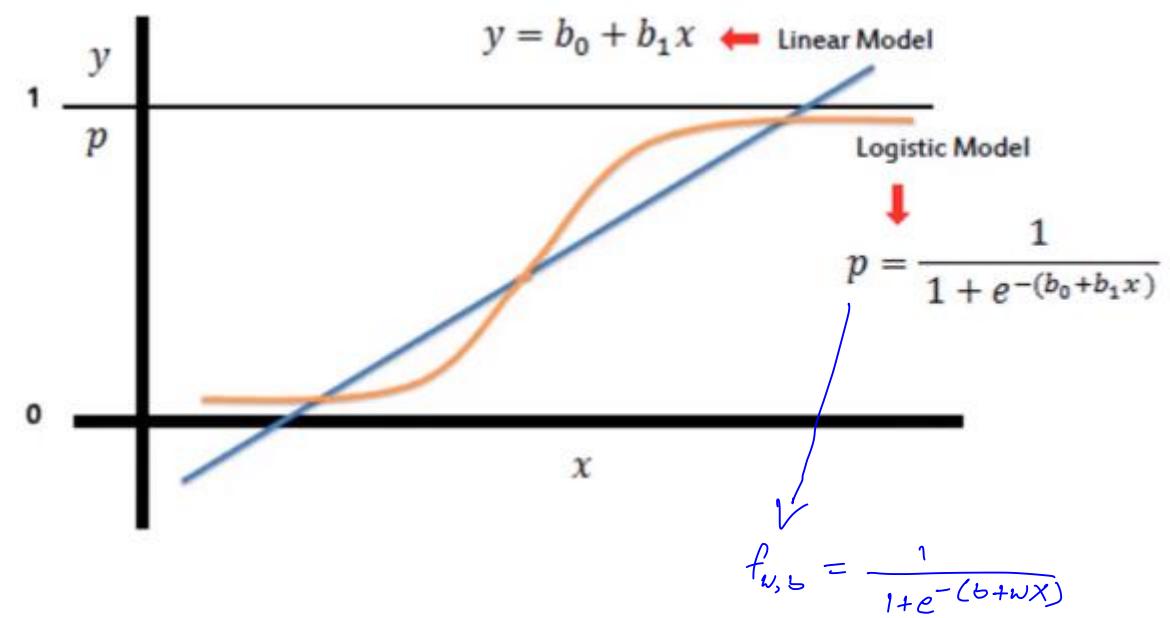
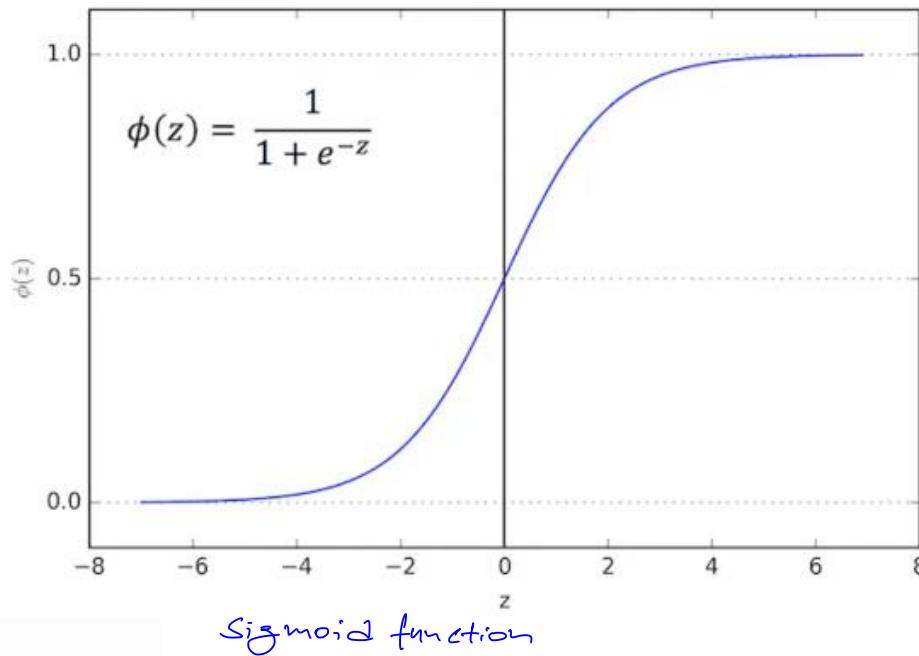


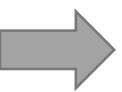


Sigmoid Function

- For Transformation, if we use:
Sigmoid fn \rightarrow we call it logit.
Cdf \rightarrow " " " " Probit.

- We need a **monotone** mapping function that has a **range** of [0,1]
↗ strictly increasing





Logistic Regression (Model)

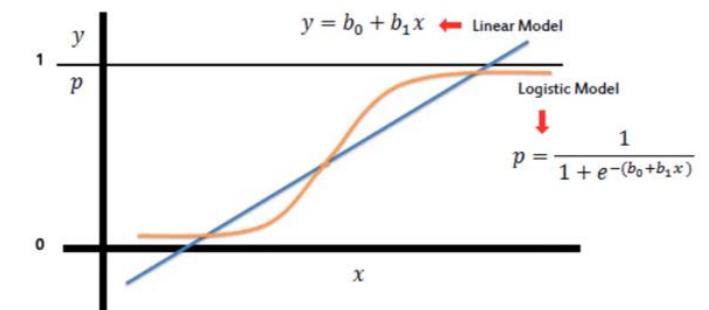
- The model:

$$P(x) = f_{w,b}(X) = \frac{1}{1+e^{-(WX+b)}}$$

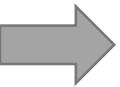
$$\begin{aligned} P(x) + P(x) e^{-z} &= 1 \\ \Rightarrow P e^{-z} &= 1 - P(x) \\ \Rightarrow e^{-z} &= \frac{1-P}{P} \Rightarrow e^z = \frac{P(x)}{1-P(x)} \Rightarrow z = \log\left(\frac{P(x)}{1-P(x)}\right) \end{aligned}$$

- In case of two classes, $f_{w,b}(X) = \Pr(Y = 1|x) = p(x)$.
- A bit of rearrangement gives

$$\log\left(\frac{p(x)}{1-p(x)}\right) = WX + b$$

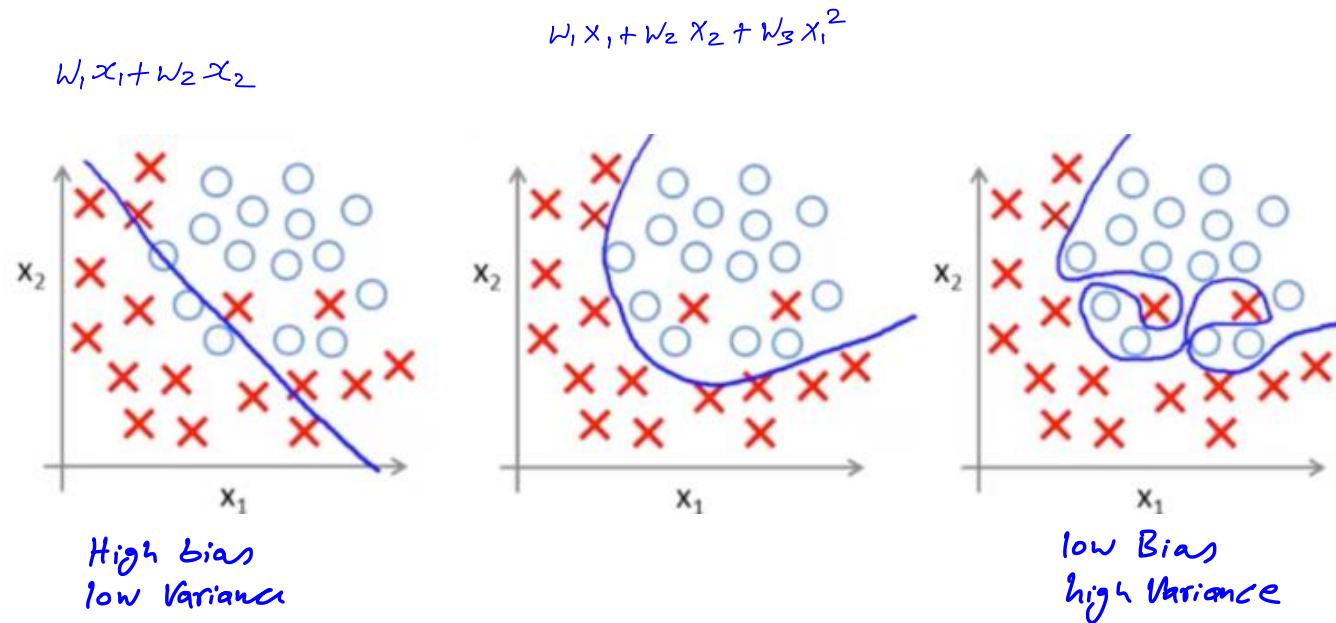


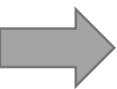
- This monotone transformation is called the **log odds** or **logit** transformation of $p(x)$.
- Logistic regression ensures that our estimates always lie between 0 and 1



Logistic regression fit (Decision boundary)

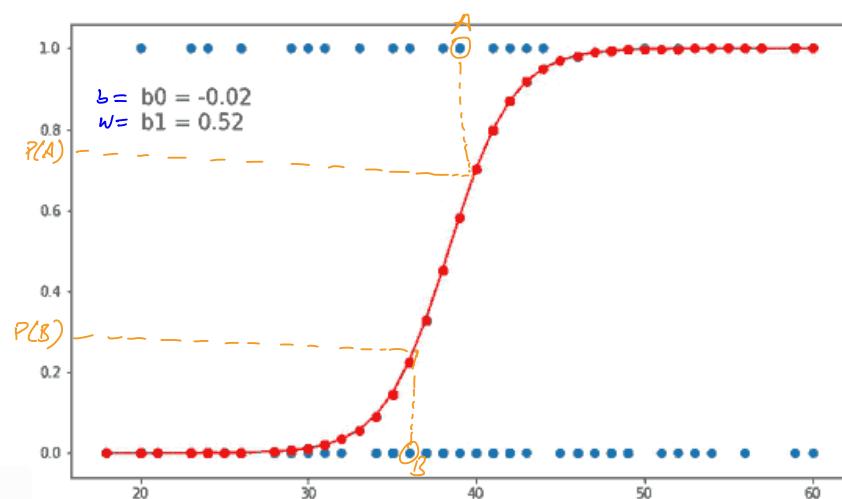
- Depending on how we define $WX + b$, we can get any of the following **fits** from logistic regression **classifier**.





Logistic Regression (Maximum Likelihood)

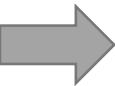
- In logistic regression, instead of minimizing the average loss, we maximize the **likelihood** of the training data according to our model. This is called **maximum likelihood estimation**.
- A fantastic visualization!
- Can you do the same visualization with the S curve?



*A & B are observations.
Since observations are independent
 $P(A \cap B) = P(A) \cdot P(B)$*

$$L_{w,b} = \prod_i f_{w,b}(x_i)^{y_i} (1 - f_{w,b}(x_i))^{1-y_i}$$

joint probabilities



Logistic Regression (Objective function)

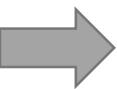
- Maximizing the likelihood function:

$$\text{Max } \{L_{w,b} = \prod_i f_{w,b}(x_i)^{y_i} \left(1 - f_{w,b}(x_i)\right)^{1-y_i}\}$$

- **Solution:** In practice, it is more convenient to maximize the **log-likelihood** function. This log-likelihood maximization, gives us w^* and b^* . There is **no closed form solution** to this optimization problem. We need to use **gradient descent**.
- We are now ready to make **predictions**.

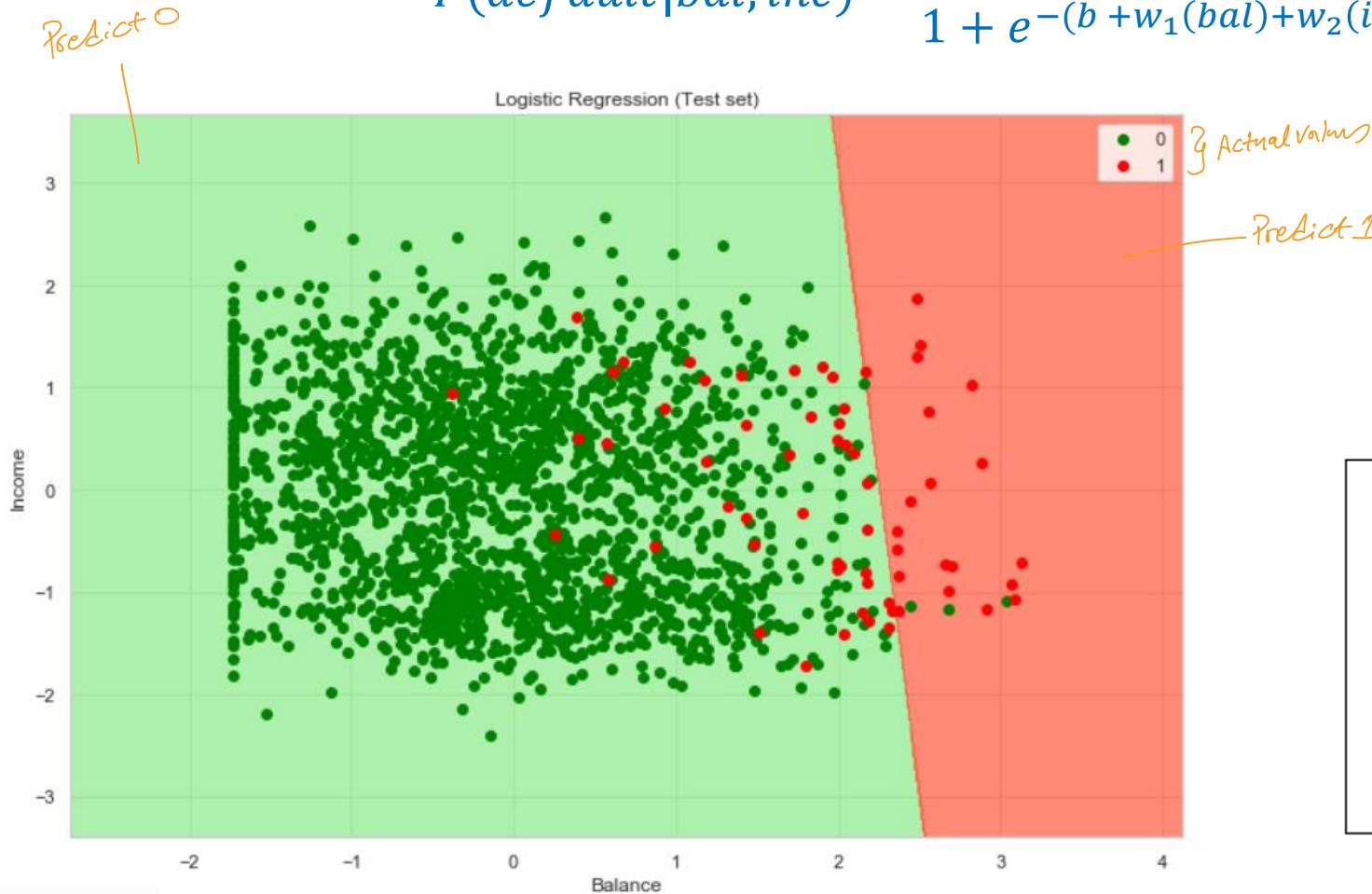
$$f_{w^*,b^*}(X) = \frac{1}{1+e^{-(W^*X+b^*)}}$$

- Depending on how we define the probability threshold, we can classify the observations. In practice, the choice of the threshold could be different depending on the problem.



Logistic regression output for credit card default example

$$P(\text{default}|\text{bal}, \text{inc}) = \frac{1}{1 + e^{-(b + w_1(\text{bal}) + w_2(\text{inc}))}}$$



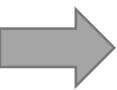
		Predictions (Decision boundary)	
		0 No Default	1 Default
Actual	0 No Default	TN=1933	FP=3
	1 Default	FN=44	TP=20

1936

64

Part II

Classification Performance Metrics



Confusion Matrix

		Predictions	
		0 negative	1 positive
Actual	0 negative	TN	FP*
	1 positive	FN**	TP

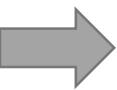
FP* Type I error

FN** Type II error

3-Classes ex:

		predicted class		
		class 1	class 2	class 3
actual class	class 1	True positives	False Class 2	False Class 3
	class 2	False Class 1	True positives	False Class 3
	class 3	False Class 1	False Class 2	True positives

↓
True Class 3



Accuracy, Precision, Recall and F1score

$$\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP}$$

Not good if dataset is imbalanced.

$$\text{Error rate} = 1 - \text{Accuracy}$$

*Good no's
All no's*

		Predictions	
		0 negative	1 positive
Actual	0 negative	TN	FP
	1 positive	FN	TP

While **recall** expresses the ability to find all **relevant** instances in a dataset, **precision** expresses the proportion of the data points our model says was relevant were actually relevant.

$$\text{Recall} = \frac{TP}{TP + FN}$$

→ of the actual details what % were correctly predicted / classified.

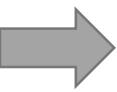
$$\text{Precision} = \frac{TP}{TP + FP}$$

→ Proportion of your prediction were correct.

$$\text{F1 Score} = 2 * \frac{PR}{P + R}$$

F1 uses the **harmonic mean** instead of a simple average because it punishes extreme values. *F1 uses both Precision(P) & Recall(R).*

If $P=1$ & $R=0$, Simple average gives $F1=0.5$, this is NOT a good indicator as Model has Recall of 0. So, by using harmonic mean $F1=0$, so it penalizes for $R=0$.



ROC (Receiver Operating Characteristic)

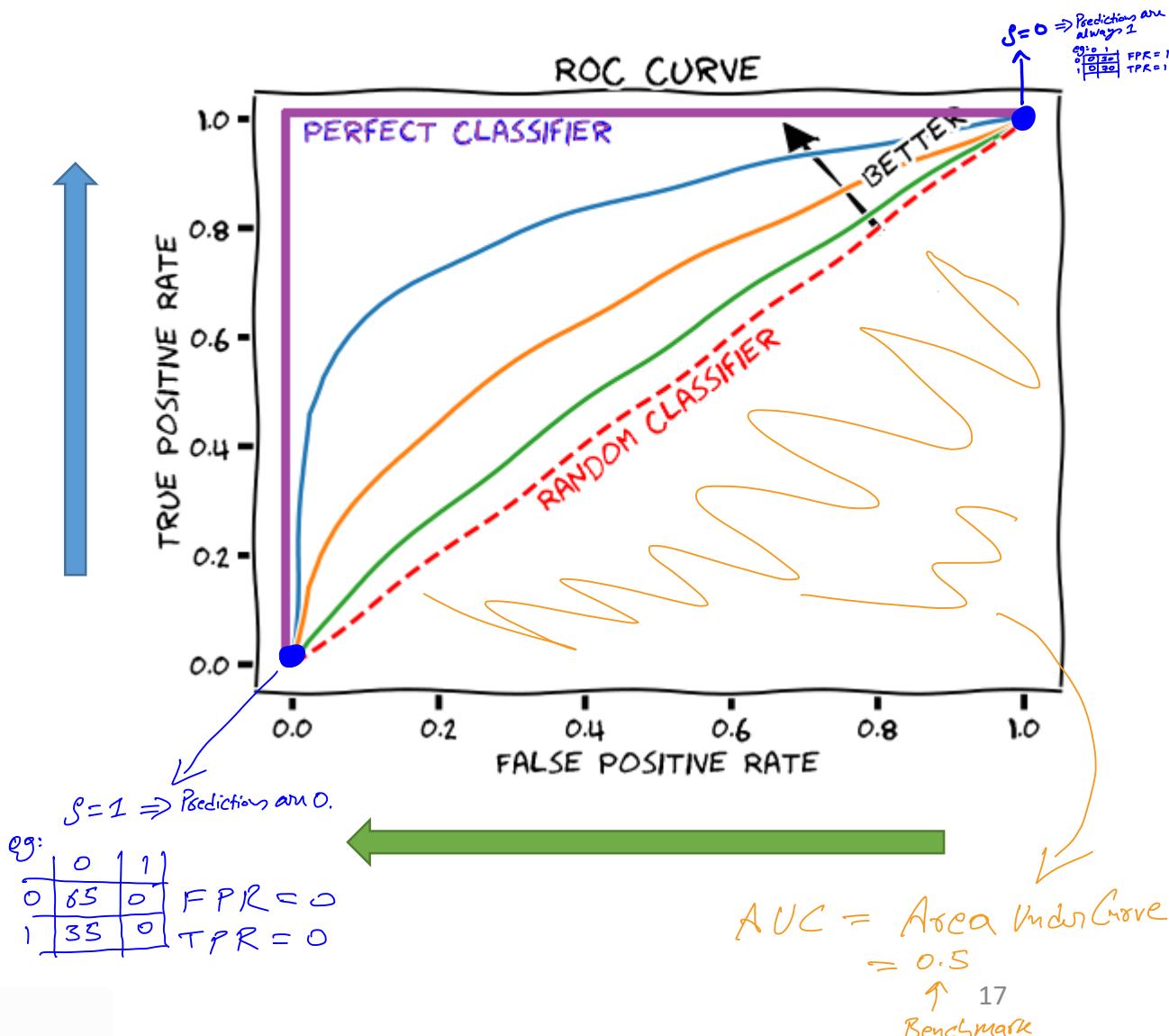
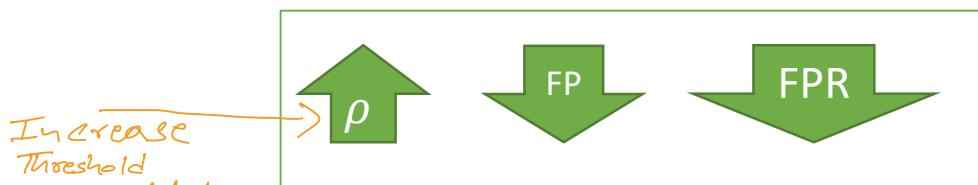
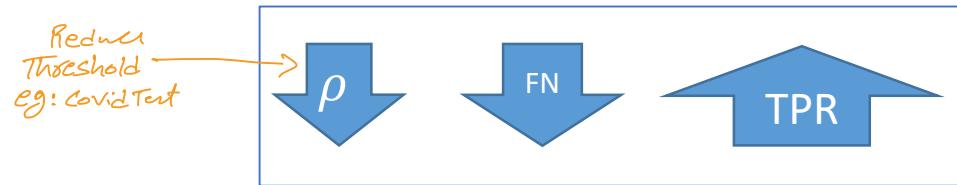
Changing Threshold changes values of Performance Metric.

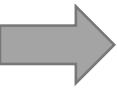
		Predictions	
		0 negative	1 positive
Actual	0 negative	TN	FP
	1 positive	FN	TP

Specificity = $1 - \text{False Positive Rate}$

$\text{False Positive Rate} = \frac{FP}{FP + TN}$

$\text{True Positive Rate} = \frac{TP}{TP + FN}$
OR
Recall or sensitivity

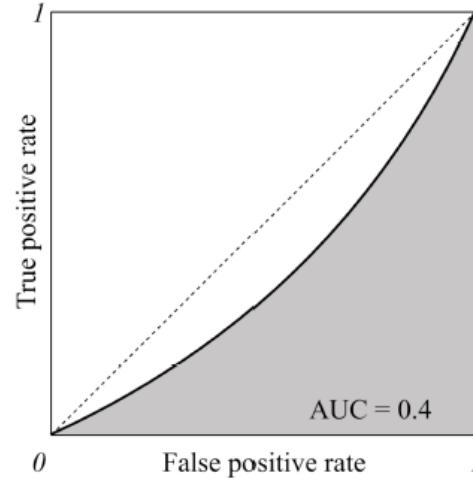




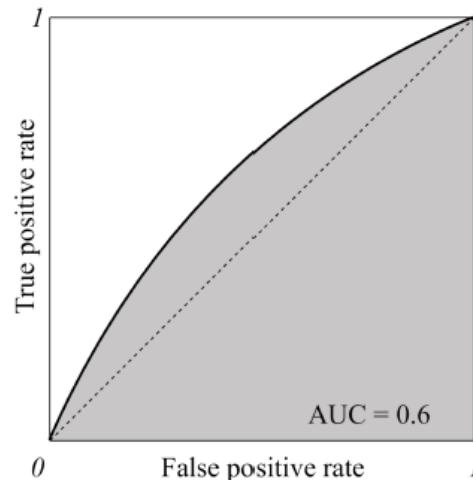
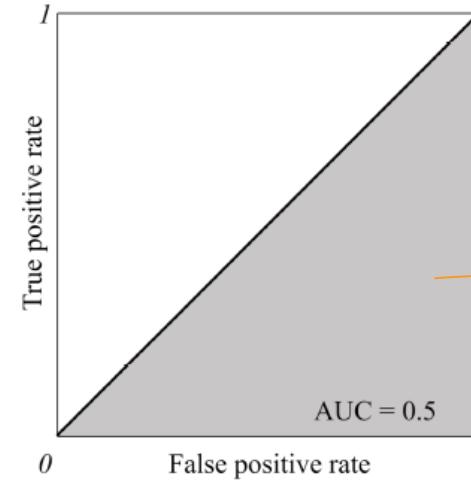
AUC

$AUC > 0.5 \Rightarrow$ Better than Random
 $AUC < 0.5 \Rightarrow$ Model is Poor.

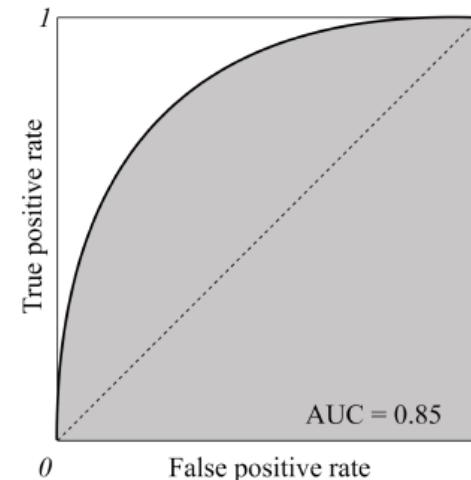
Poor Model



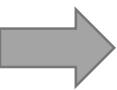
Bench Mark



Good Model

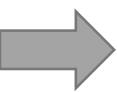


V. Good Model



Some other classification metrics

		True condition		Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Total population	Condition positive	Condition negative			
Predicted condition	Predicted condition positive	True positive	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR-}}$
		False negative rate (FNR), Miss rate = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

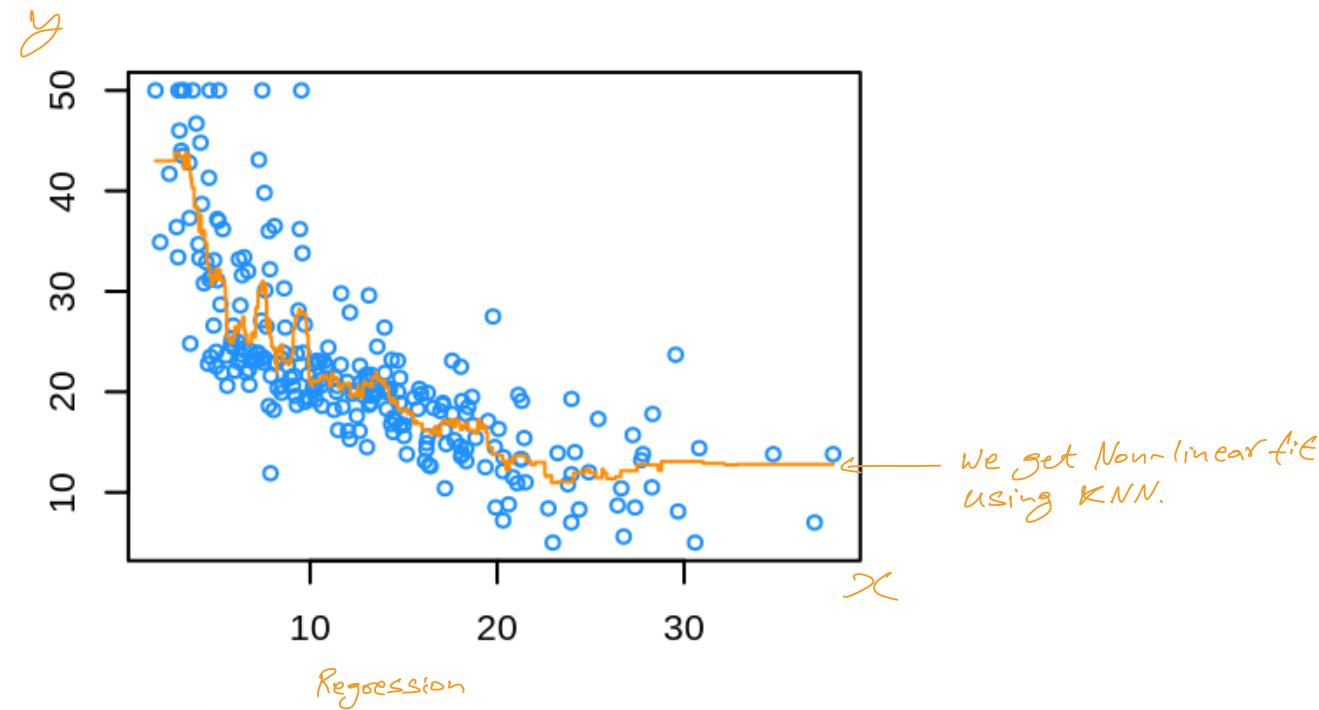
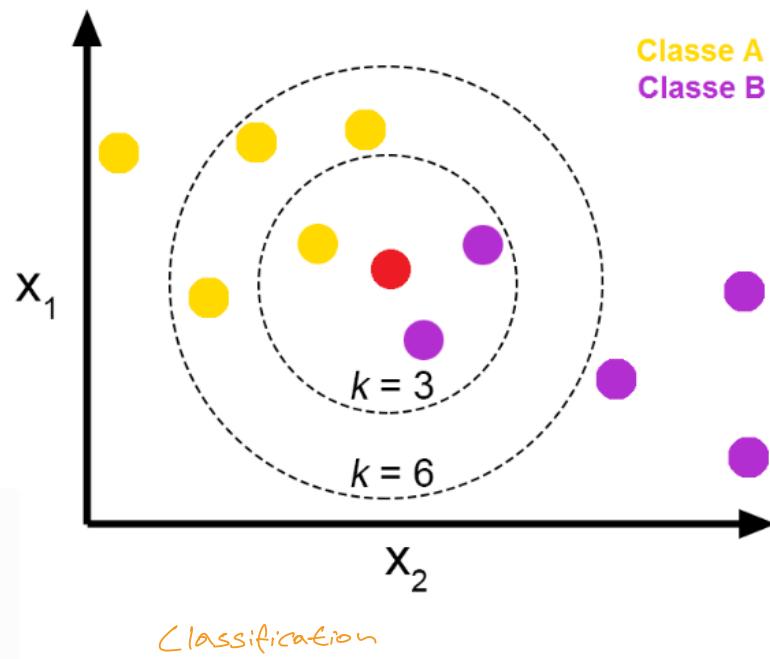


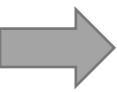
Students' questions

- 1) Are we treating (classifying) $\hat{y} = 0.51$ and $\hat{y} = 0.99$ the same? *If threshold is 50% yes, if it's 70% then No.*
- 2) Does it make sense to have non-linear decision boundaries in logistic regression? *Yes*
- 3) Is logistical regression useful for anything beyond probability prediction? *Yes for classification.*
- 4) What do ROC and AUC tell us about our predictions? *larger AUC the better the Model.*

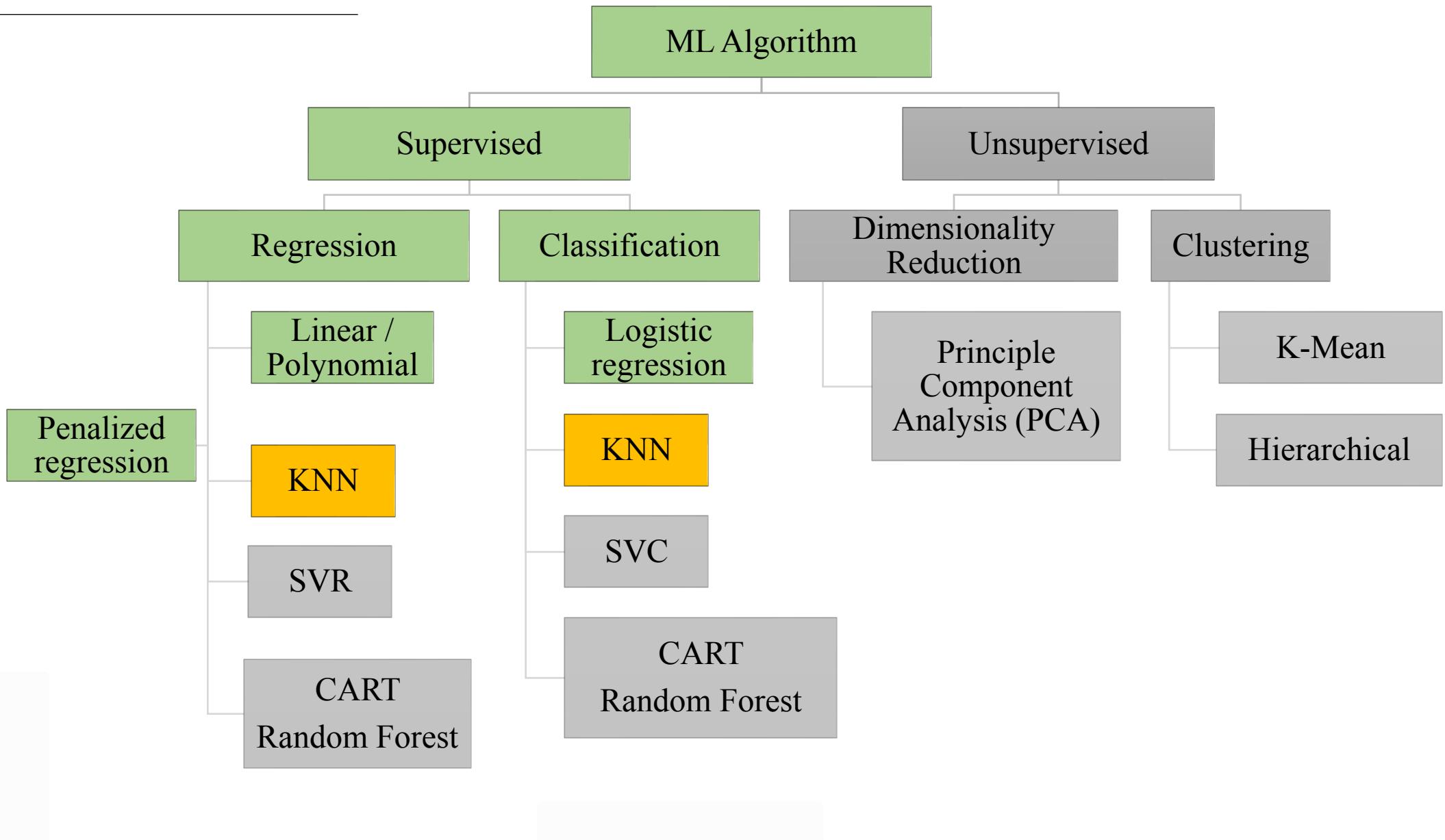
Class 12 – KNN

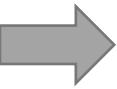
K-Nearest Neighbors





Road map





Topics

Part I

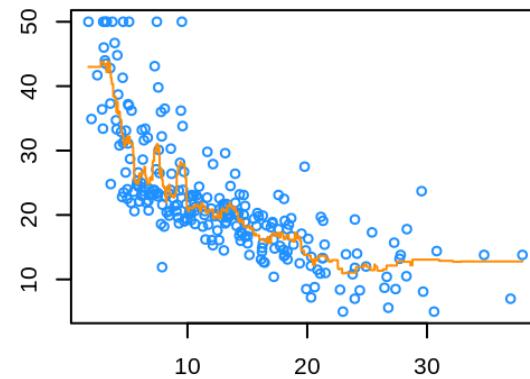
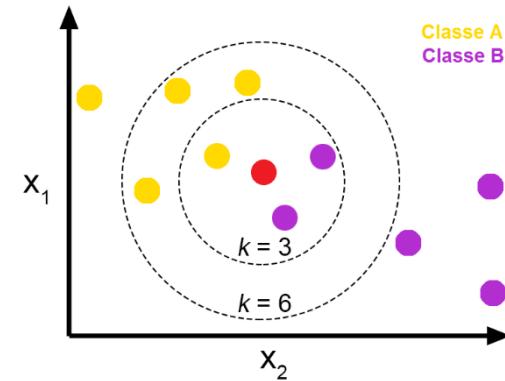
1. KNN Classification
2. Performance metrics and choice of K

Part II

1. KNN Regression
2. KNN vs Linear Regression
3. Performance metrics and choice of K

Part III

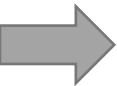
1. Curse of Dimensionality
2. Pros and Cons of KNN
3. KNN applications in Finance



In general, any Machine learning models that deal with distance metric, we should (standardize) scale the features. KNN is a Lazy Algorithm, so for large datasets it may not be the most efficient. The K in KNN is the hyperparameter & we use Cross Validation to find the optimal one. Another hyperparameter is the distance Metric.

Part I

KNN Classification



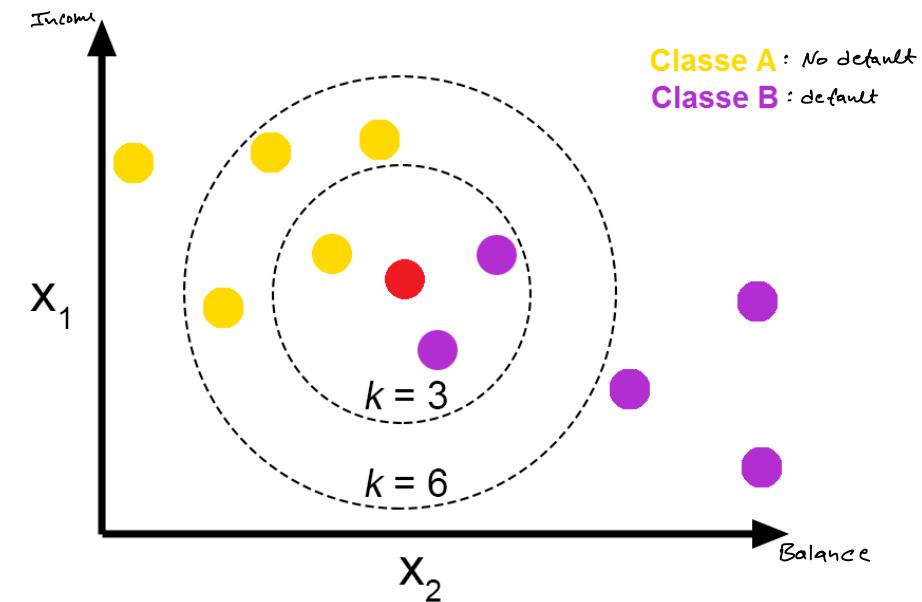
KNN (K-Nearest Neighbors)

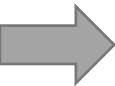
The interpretability of KNN is 0, we can only make predictions.

K-nearest neighbor (KNN) is one of the simplest and best-known **non-parametric supervised** learning technique most often used for **classification**. The idea is to classify a new observation by finding **similarities** (“nearness”) between it and its k -nearest neighbors in the existing dataset.

- Contrary to other learning algorithms that allow discarding the training data after the model is built, KNN **keeps all training examples in memory**. *That's why we call it lazy.*
- The choice of the **distance metric**, as well as the **value for k** , are the choices the analyst makes before running the algorithm. So, these are **hyperparameters**.
- There are no parameters in KNN.

The outcome of KNN classification are probabilities.





KNN steps

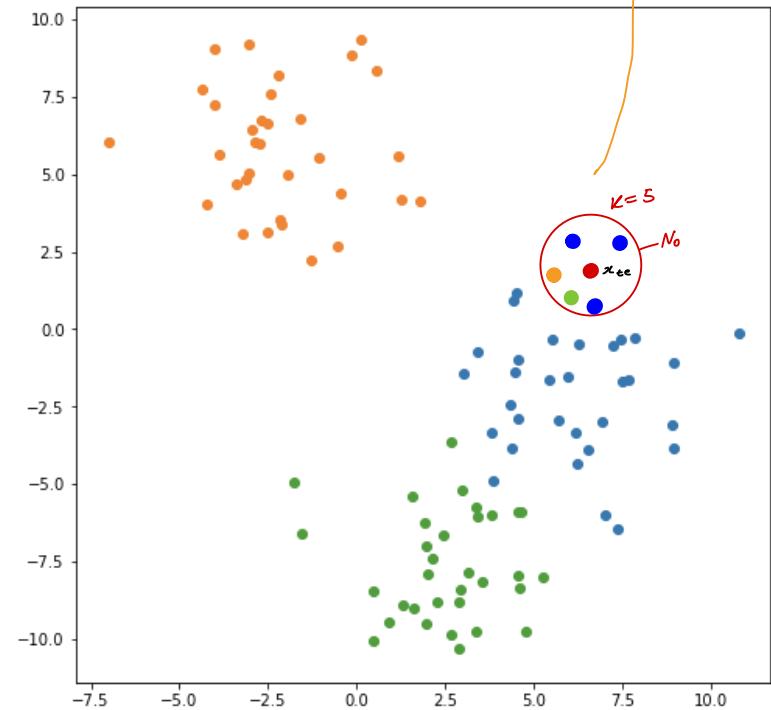
1. Choose number of neighbors **K** (positive integer)
2. Choose the distance metric (Minkowski, Euclidian, Manhattan, etc.)
3. Identify the K points in the training data that are closest to x_{te} in the test set. This **neighborhood** is represented by N_0 .
4. Estimate the **conditional probability** for class j as the fraction of points in N_0 whose response values equal j:

$$\Pr(Y = j | X = x_{te}) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$$

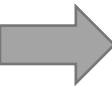
5. Classifies the test observation x_{te} to the class with the largest probability.

$$\begin{aligned} \Pr(\text{orange} | X_{te}) &= 1/5 \\ \Pr(\text{blue} | X_{te}) &= 3/5 \\ \Pr(\text{green} | X_{te}) &= 1/5 \end{aligned}$$

So, $\bullet x_{te}$ is classified as blue.



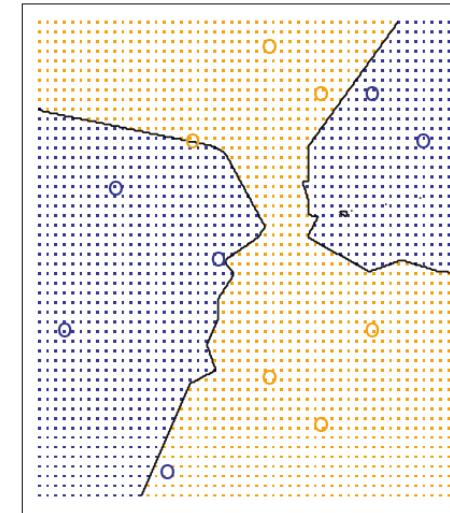
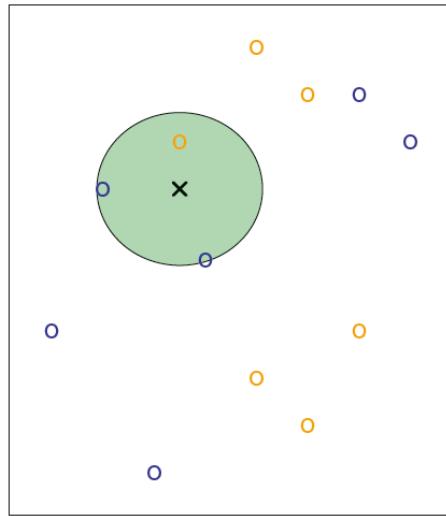
In KNN there is No loss fn. So, locally in the neighbourhood, we are maximizing conditional probability.



KNN Decision Boundary

Increasing K means making the Model less flexible so the decision boundary becomes much smoother.

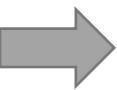
$x \rightarrow$ Test Observation
 $K=3 \rightarrow$ Classification
Blue
Algorithm picks randomly
 $K=2 \rightarrow$
 $K=1 \rightarrow$ orange



Fit is always Non-Linear in KNN.

- Two classes: blue and orange!
- Black cross: a test observation.
- When $K=3$, black cross is classified as:
 Blue
- Repeat this process for every potential element in the feature space!

- KNN **decision boundary** is shown in **black**.
- The blue grid indicates the region in which a test observation will be assigned to the blue class, and
- the orange grid indicates the region in which it will be assigned to the orange class.



Performance metrics

Summarizes no. of incorrect classification i.e. misclassification.



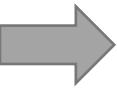
- Error rate = $1 - \text{Accuracy}$ \rightarrow

$$\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP}$$

$$\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$

		Predictions	
		0 negative	1 positive
Actual	0 negative	TN	FP
	1 positive	FN	TP

- A good classifier is one for which the **test error** is smallest.
- Like any other classifier, if the data is highly imbalanced, then we should use f1 score, precision and recall instead of the error rate.

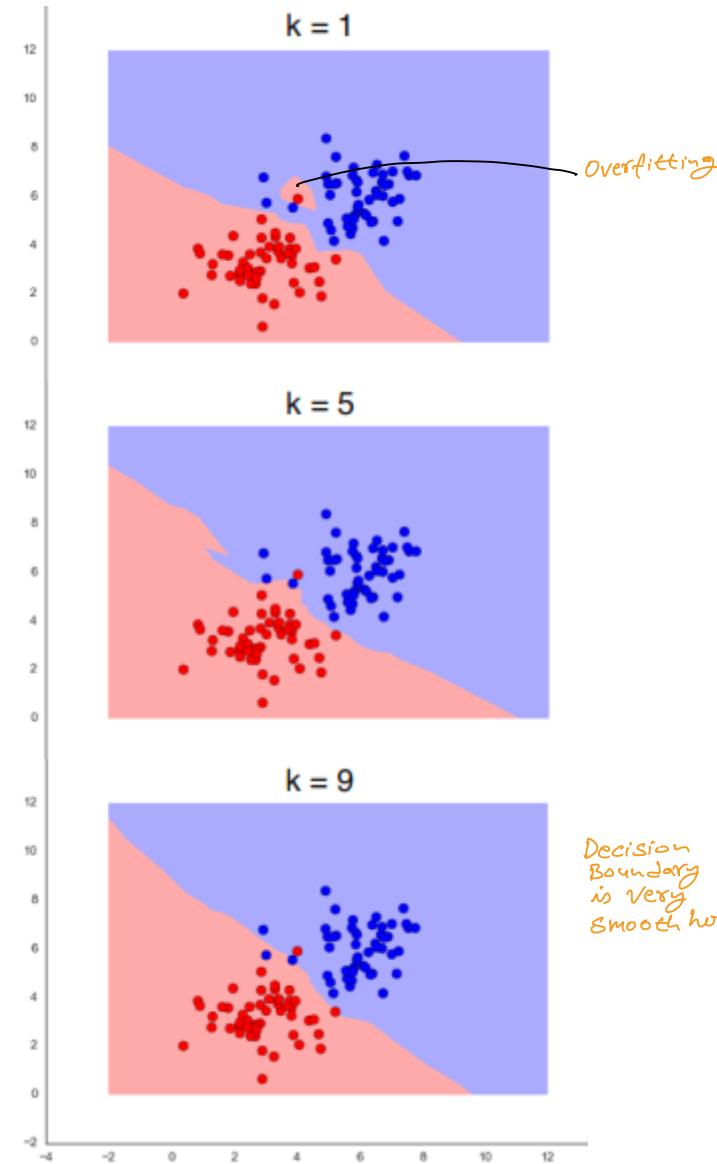
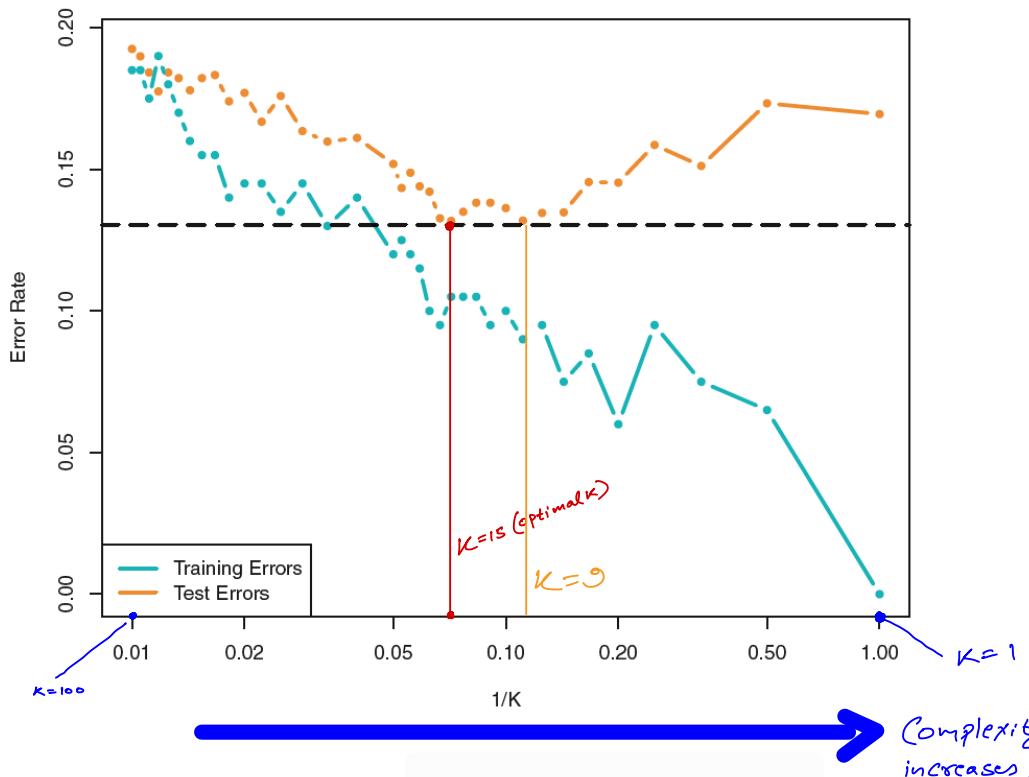


Choice of K (Bias Variance Trade Off)

↑ or SuperComplex

- $K=1$ very flexible model: Low bias but high variance.
- As K grows, less flexible model, decision boundary gets close to linear. This corresponds to a low variance but high bias.
- Optimal value of K :

We have same minimum value for 2 Test errors
So, the optimal choice in this case is pick the simpler model i.e. pick a higher valued K . In this eg, we picked $K=15$ instead of $K=9$.

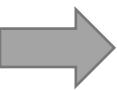


Q: If $K \geq$ No. of points in Sample in Train set, What happens?
 $\Rightarrow KNN$ will always predict one class

Part II

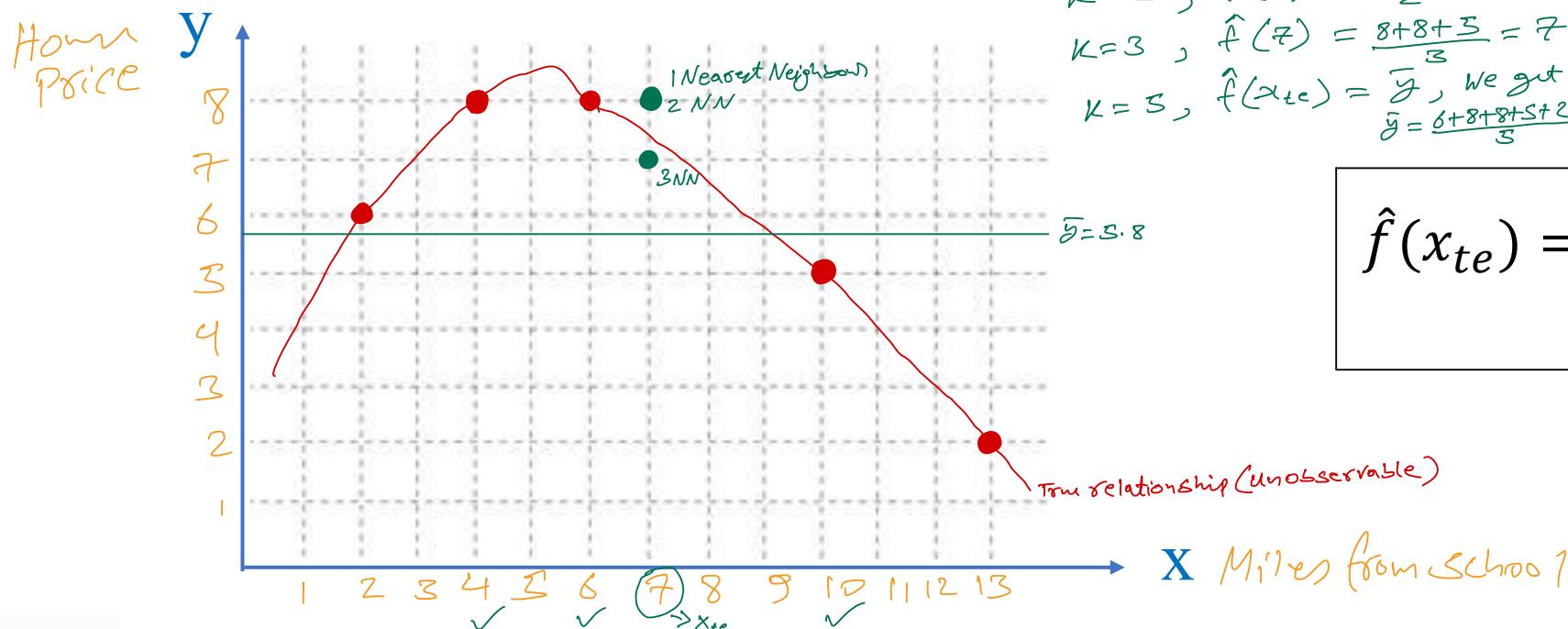
KNN Regression

[In literature, KNN is rarely used for Regression.]



KNN Regression

- The KNN regression method is closely related to the KNN classifier



Here feature Space is 1-Dimension.

$$K=1, \hat{f}(7) = 8$$

$$K=2, \hat{f}(7) = \frac{8+8}{2} = 8$$

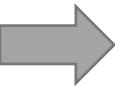
$$K=3, \hat{f}(7) = \frac{8+8+5}{3} = 7$$

$$K=5, \hat{f}(x_{te}) = \bar{y}, \text{ we get a st. line.}$$
$$\bar{y} = \frac{6+8+8+5+2}{5} = 5.8$$

$$\hat{f}(x_{te}) = \frac{1}{K} \sum_{i \in N_0} y_i$$

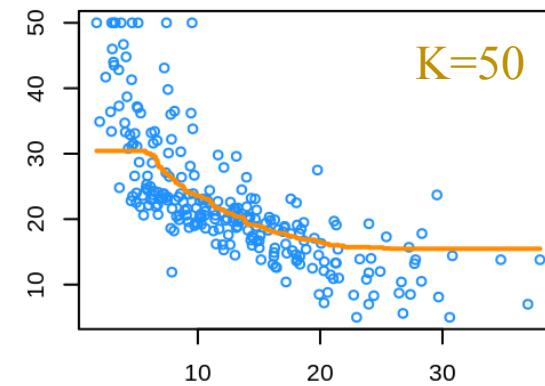
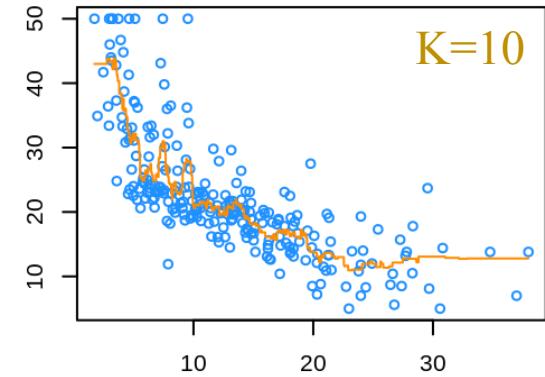
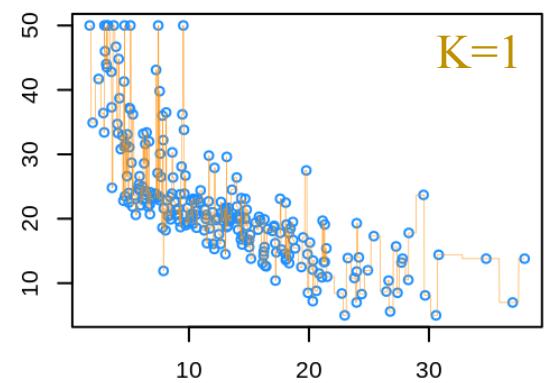
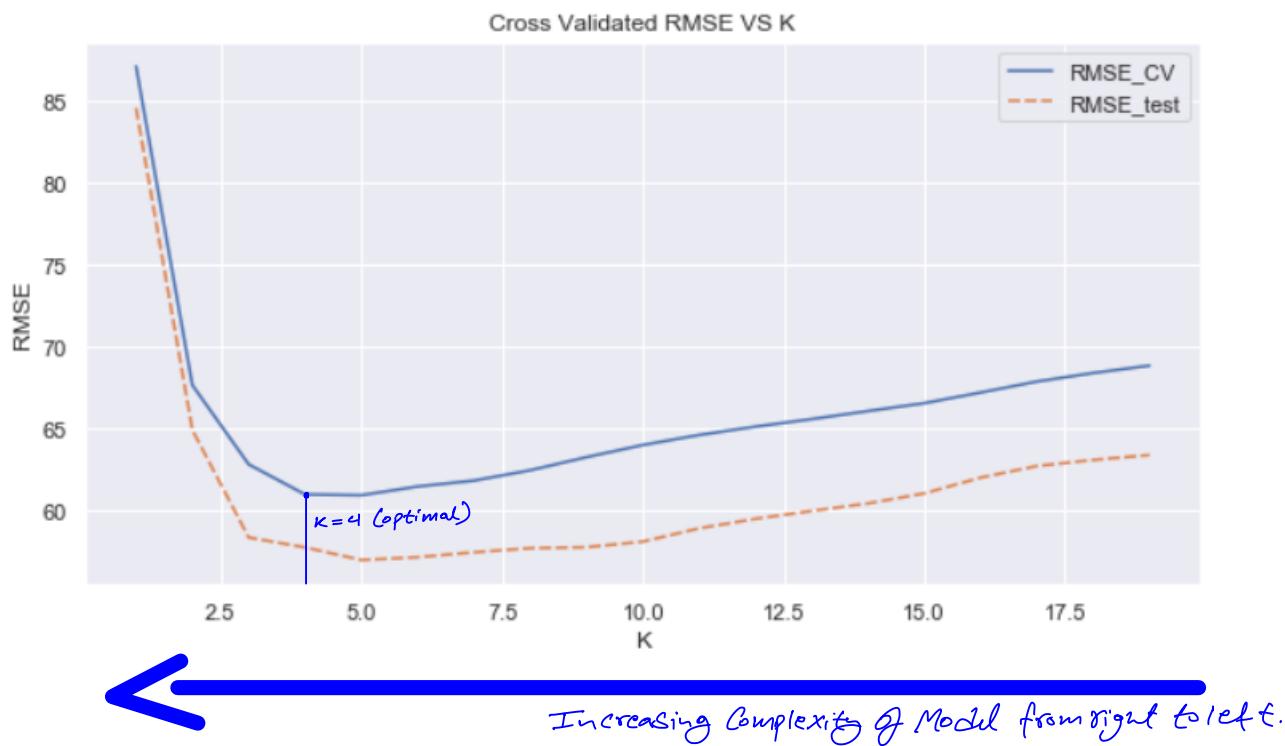
In Regression, we are simply calculating the Average for those neighbours.

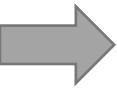
KNN Regression will do good job if the true relationship between Target Variable & features is a Non-linear one.



Choice of K (Bias Variance Trade Off)

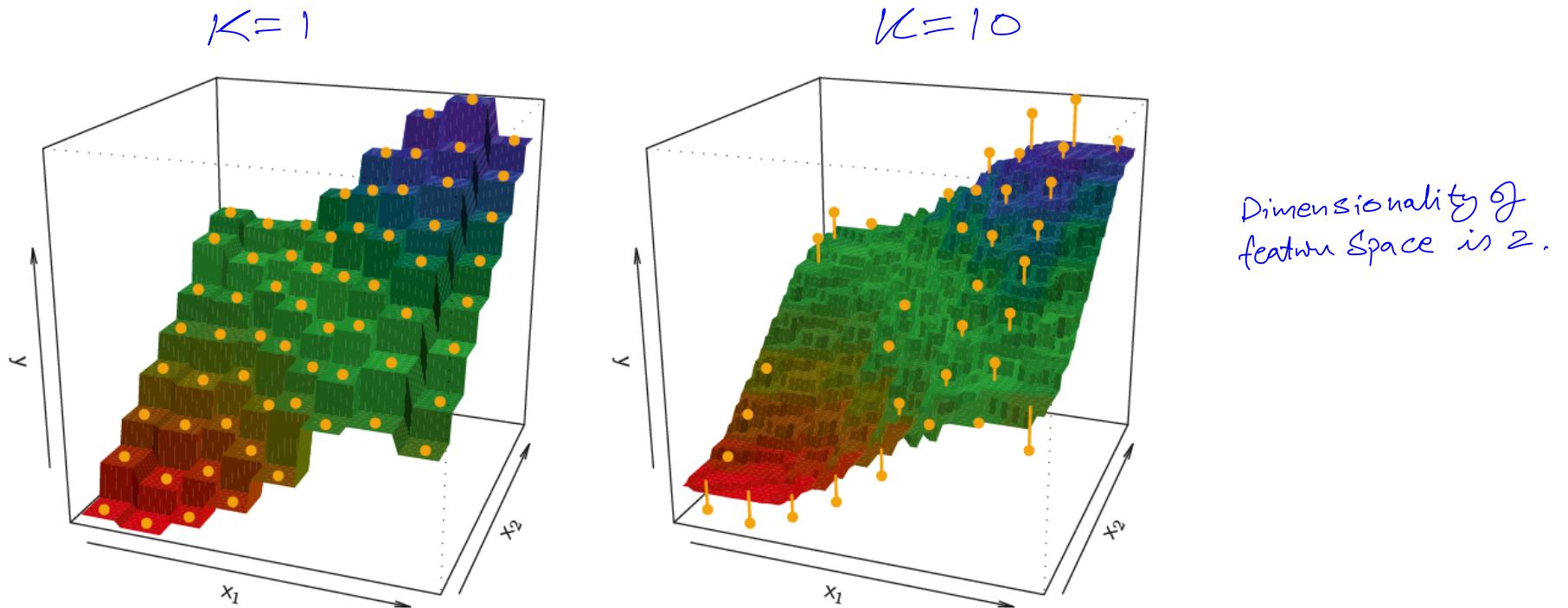
- $K=1$ very flexible model: Low bias but high variance.
- As K grows, less flexible model, regression fit gets smoother and smoother. This corresponds to a low variance but high bias.
- Optimal value of K :

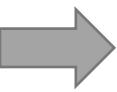




Choice of K (Bias Variance Trade Off)

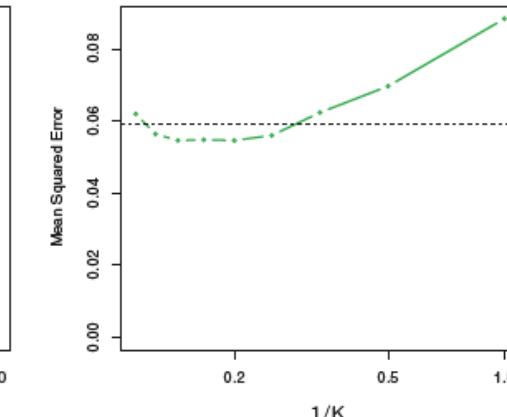
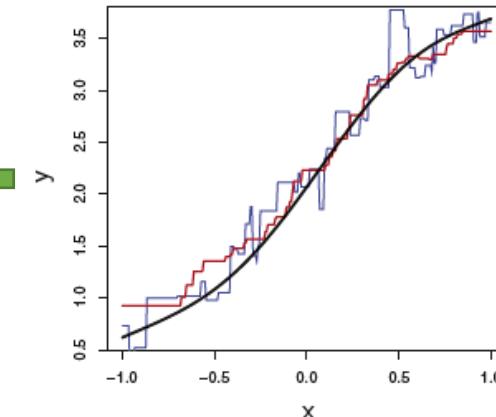
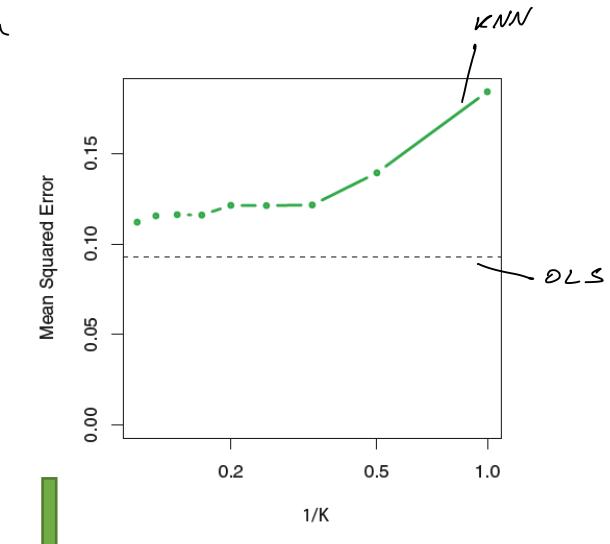
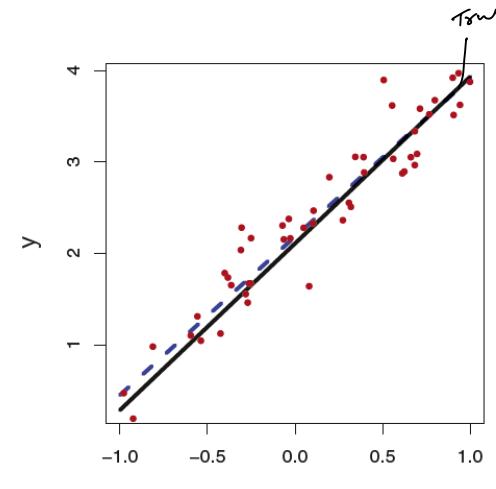
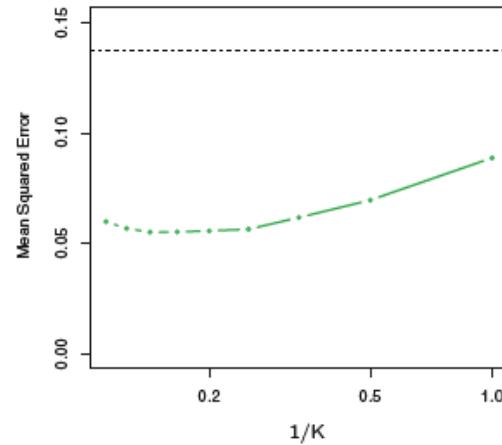
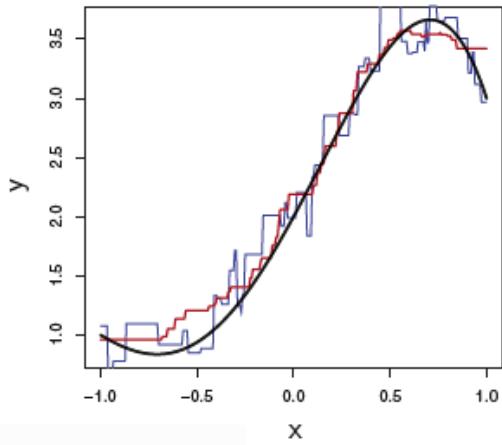
- $K=1$ very flexible model: Low bias but high variance.
- As K grows, less flexible model, regression fit gets smoother and smoother. This corresponds to a low variance but high bias.





Linear regression vs KNN regression

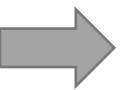
- Black curve is the true relationship between y and X
- Green dashed line: KNN MSE_test
- Black dashed line: OLS MSE_test
- The more non-linear the true relationship, the better performance of KNN compared to OLS.



Part III

Pros and Cons

Applications in Finance



Curse of Dimensionality

- The “Curse of Dimensionality” is a problem with the relationship between **dimensionality** and **volume**.
- Sparsity of data occurs when moving to higher dimensions. the volume of the space represented grows so quickly that the data **cannot keep up** and thus becomes sparse. (*Bellman, 1957*)

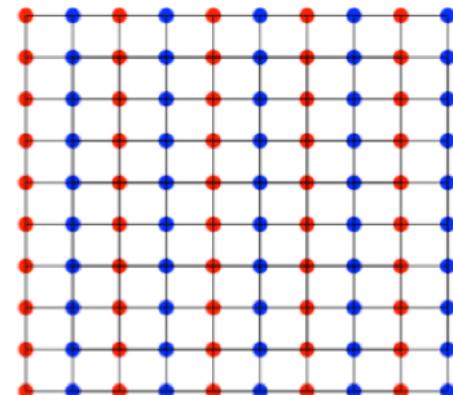
Random Variable X : discrete $[1-10]$

10 observations to span in 1-D.



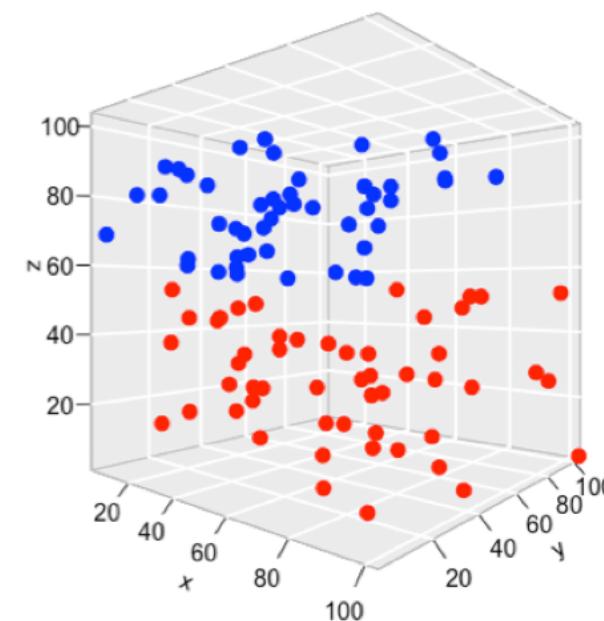
(A) 1-D

$10^2 = 100$ observations needed to span in 2-D.

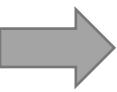


(B) 2-D

$10^3 = 1000$ observations needed in 3-D.

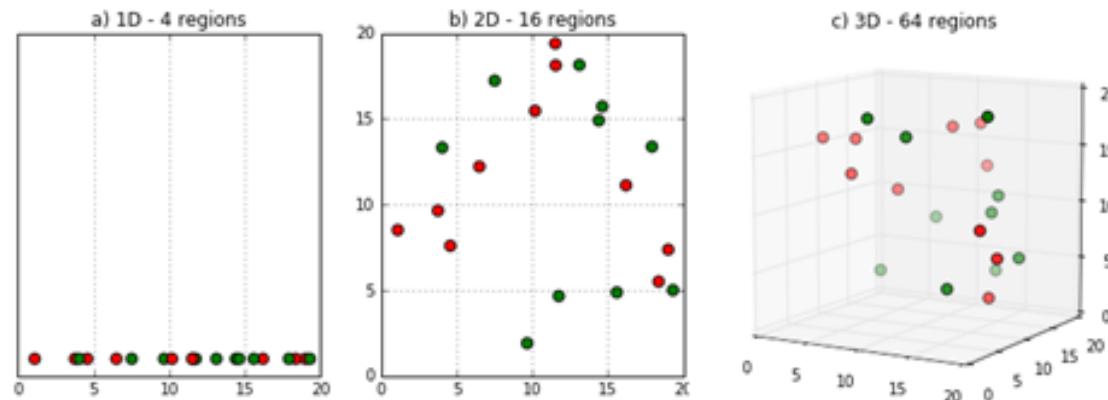


(C) 3-D

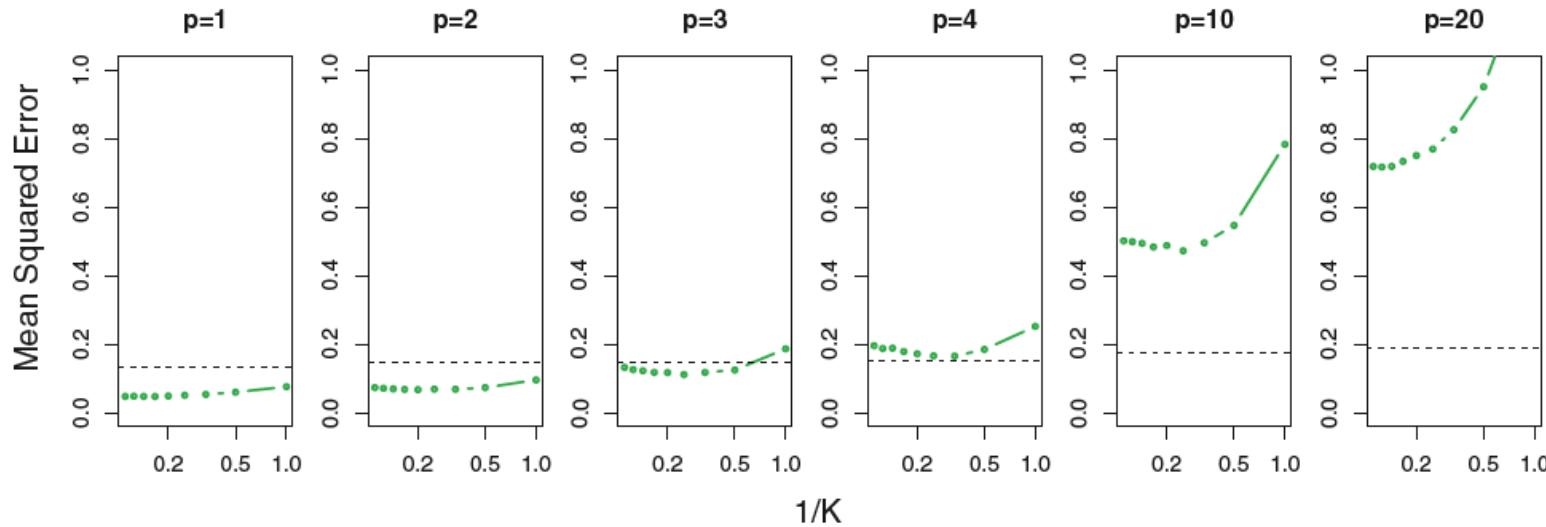


KNN and the Curse of Dimensionality

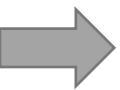
Classification:



Regression:



p = dimension
i.e. no. of features



KNN's Pros and Cons

Pros:

- Intuitive and simple
- No assumption (non-parametric)
- Easy to implement for multi-class problem
- Used both for classification and regression
- Few parameters/hyper-parameters

K
distance metric

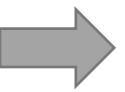
Cons:

- Slow (memory-based approach)
- Curse of dimensionality
- Not good with multiple categorical features
- Choice of K
- No interpretation (None!)



KNN beats Linear Regression (OLS) for optimal value of K when dataset is low dimensional i.e. less features. When dataset contains many features, it suffers from curse of dimensionality.

Hence, we rarely use KNN for Regression as our datasets will contain lots of features in Real World.



KNN Applications in finance

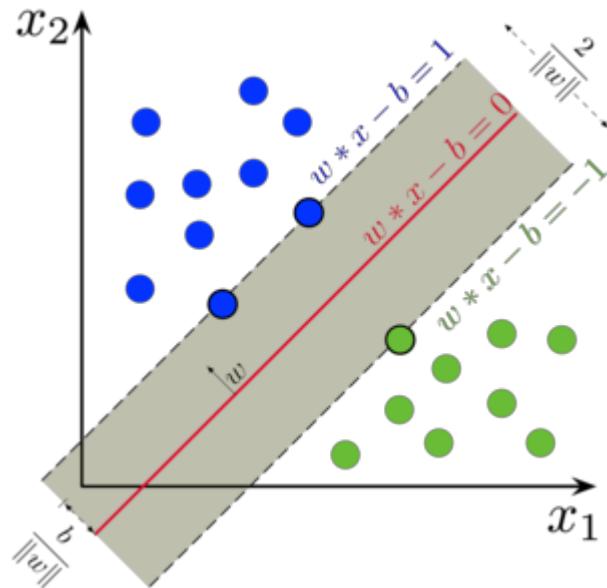
- Bankruptcy prediction
- Stock price prediction (buy/sell/hold)
- Corporate bond credit rating assignments
- Money laundering analysis
- Bank customer profiling
- Loan management
- Customized equity and bond index creation



Class 14 –15 Support Vector Machines SVM

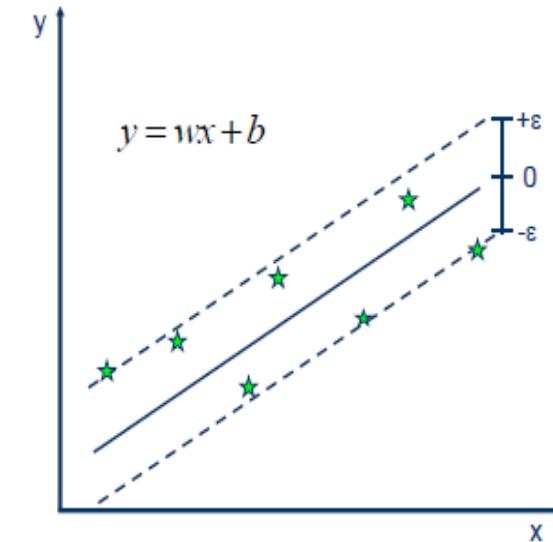


Goal: To find hyperplane that gives max. distance between classes.



Classification

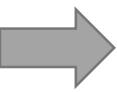
2-D



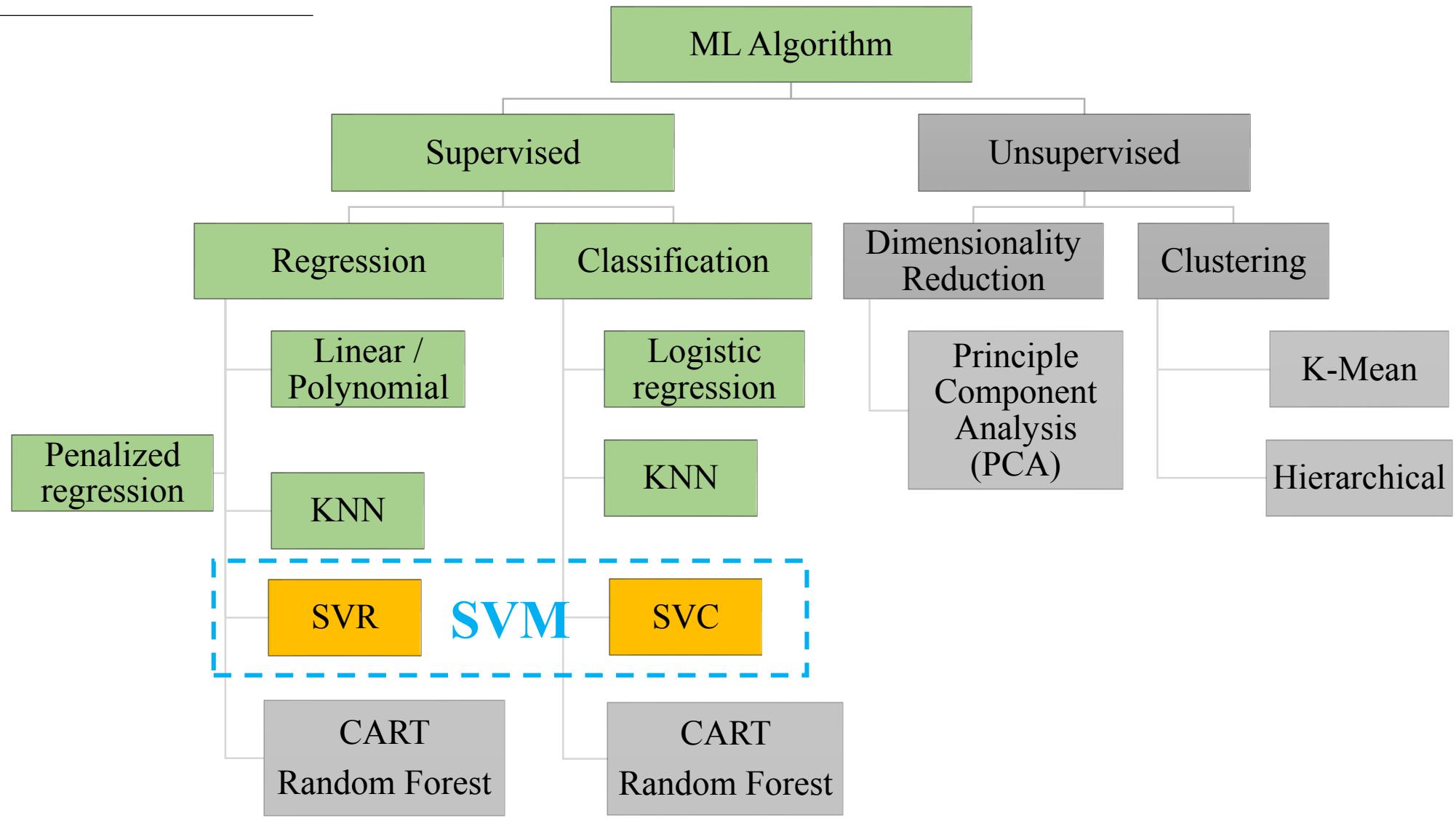
Regression

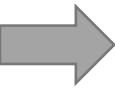
1-D

Goal: To find hyperplane such that majority of observation in train set are within the margin ϵ .



Road map





Topics

Part I

1. SVM Geometry
2. SVM Motivation

Part II

1. Maximum Margin Classifier (MMC)
 2. Support Vector Classifiers (SVC)
 3. Support Vector Machines (SVM)
- 

Part III

1. Support Vector Regressors (SVR)

Part IV

1. Tuning Hyperparameters
2. SVM pros and cons
3. SVM applications in Finance

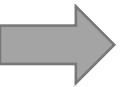
- SVM are super powerful amongst Machine learning models. For small datasets SVM is very good but becomes computationally heavy for large sample size. SVM is being replaced by Deep learning (Neural Networks). SVM can be used for both Classification & Regression.
- Classification: Unlike logistic Regression & KNN where the output was probabilities, in SVM the outputs are 0 or 1 of classes. Hence, it is called Non-probabilistic model.

Part I

Geometry

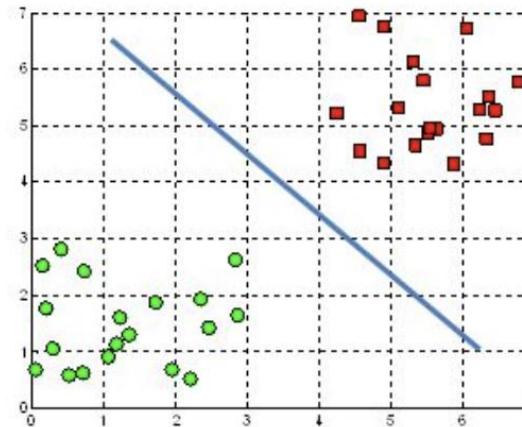
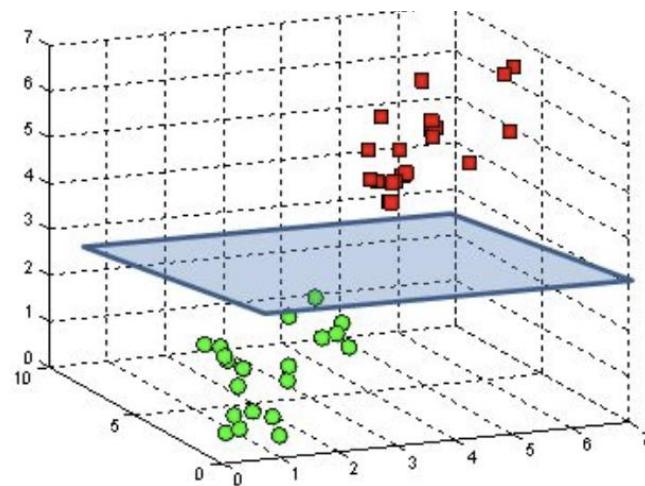
SVM Motivation

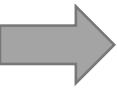
• SVM by def'n is designed for 2-class classification.



SVM Geometry

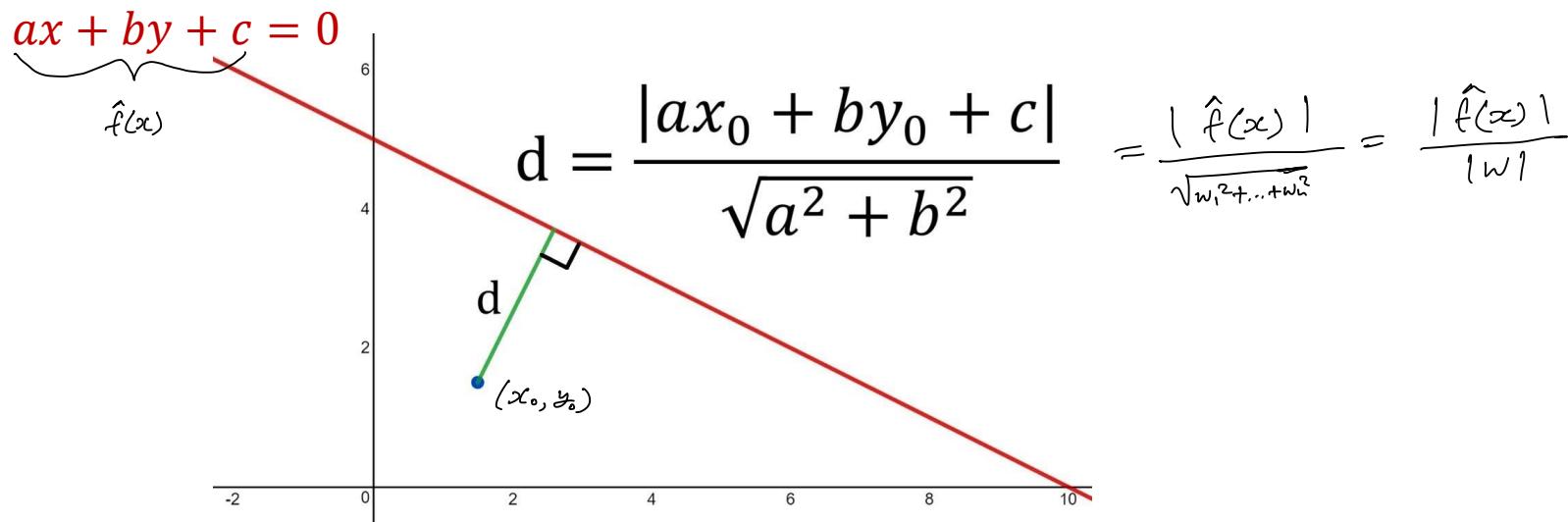
- In geometry, a **hyperplane** is a subspace whose dimension is **one less** than that of its **ambient space**. A hyperplane separates the space into two spaces.
- If a space is 3-dimensional then its hyperplanes are the 2-dimensional **planes**,
- If the space is 2-dimensional, its hyperplanes are the 1-dimensional **lines**.
- If the space is 1-dimensional, its hyperplanes are **single points**.

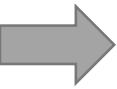




Geometry

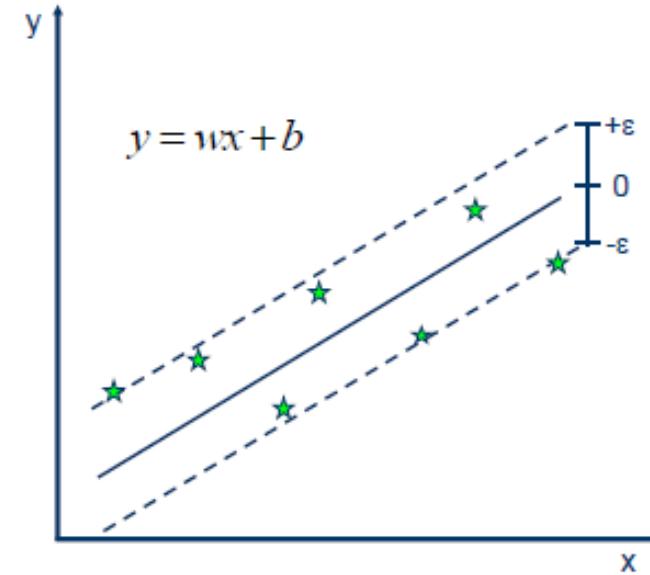
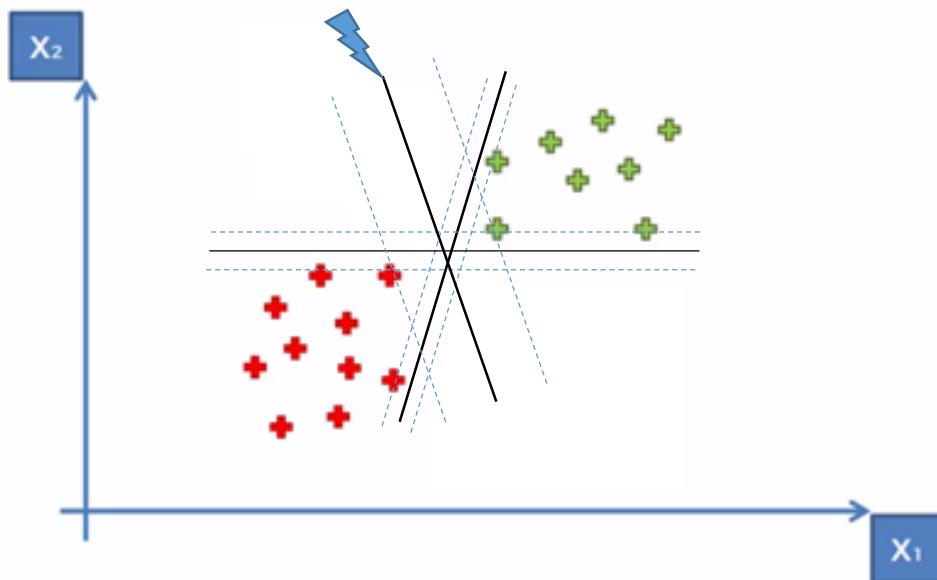
- The **perpendicular distance** between two objects is the distance from one to the other, measured along a line that is perpendicular to one or both.
- The **distance** between a point (x_0, y_0) and a line parameterized by $ax + by + c = 0$ is equal to:





SVM Motivation

Support vector machine (SVM) is one of the most popular algorithms in machine learning. It is a powerful supervised algorithm used for **classification** and **regression**.



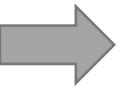
Part II

Maximum Margin Classifier (MMC): Hard Margin.

Support Vector Classifiers (SVC): Soft Margin.

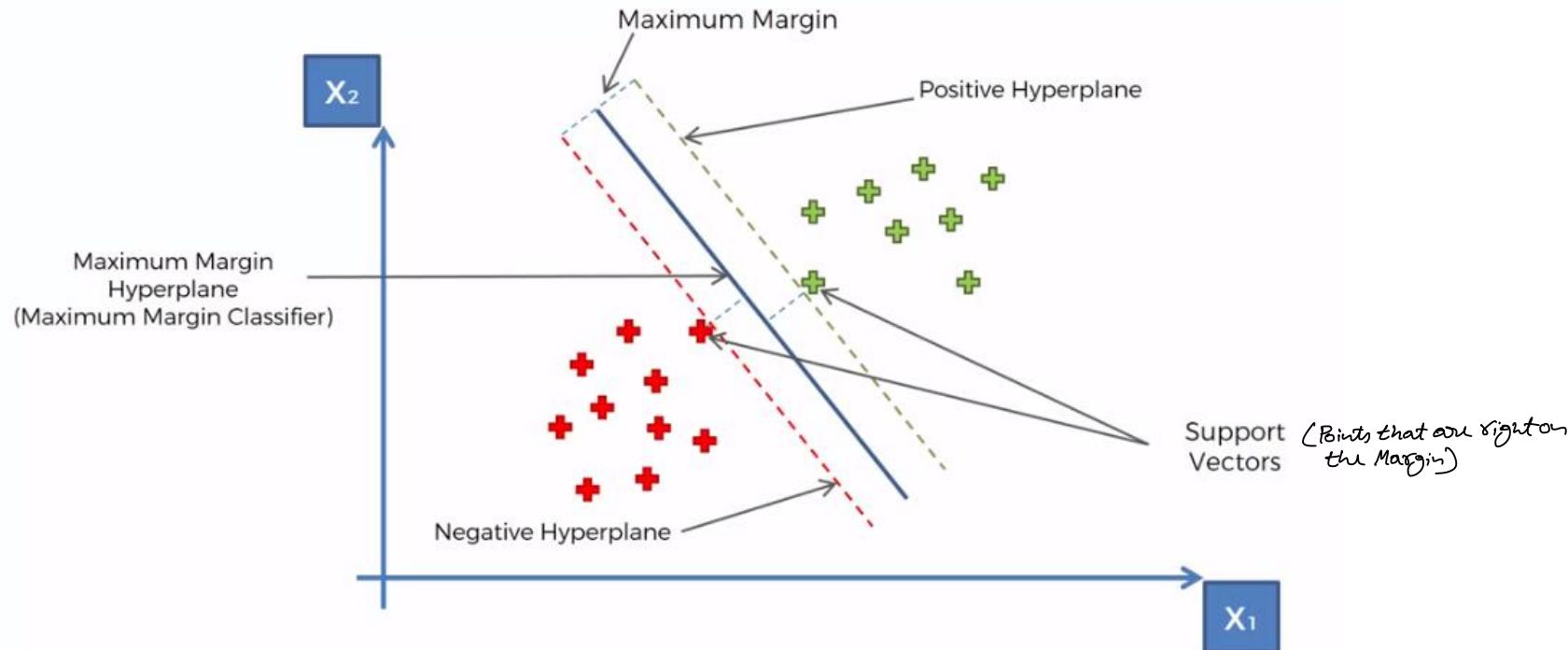
Support Vector Machines (SVM): Soft Margin + Kernel to handle Non-linear data

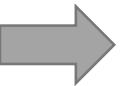
→ Penalizing (Regularization) will adjust the width of margin to balance trade off between Bias & Variance.



Maximum Margin Classifier (MMC) – Hard Margin

MMC is the hyperplane that among all separating hyperplanes, find the one that makes the biggest gap (margin) between two classes.





MMC optimization problem

- The core idea of **hard margin** is to maximize the margin, under the constraint that the classifier does **not** make **any** mistake.
- SVMs try to pick the most **robust** model (by finding the w^* and b^*) among all those that yield a correct classification. If we numerically define **blue circles** as **+1** and **green circles** as **-1**, any **good** linear model is expected to satisfy:

By multiplying both eqⁿ by $y \rightarrow$
we can summarize to only
one eqⁿ, as both eqⁿ yields ≥ 1 .

↳ To merge two constraints into one mathematically.

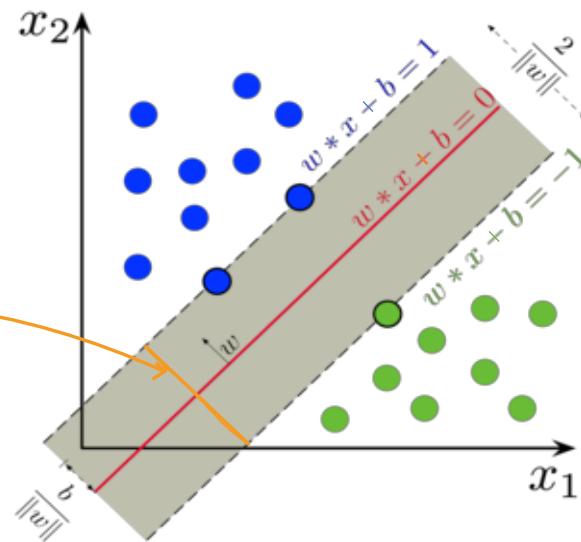
$$\begin{cases} \sum_{k=1}^K w_k x_{i,k} + b \geq +1 & \text{when } y_i = +1 \\ \sum_{k=1}^K w_k x_{i,k} + b \leq -1 & \text{when } y_i = -1 \end{cases}$$

The meaning of constraint
is that you are not
allowing for any kind
of misclassification within
the margin. (Hard Margin)

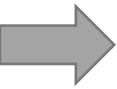
$$\begin{aligned} \text{Min}_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t. } y_i \left(\sum_{k=1}^K w_k x_{i,k} + b \right) & \geq 1 \end{aligned}$$

Instead of Max. margin we minimize $\frac{1}{2} \|w\|^2$ because $\frac{2\|f(w)\|}{\|w\|} = \frac{2\|f(w)\|}{\sqrt{w_1^2+w_2^2}} = \text{length of the Margin}$

We minimize $\frac{1}{2} \|w\|^2$ instead of $\|w\|$ since minimizing a linear fn is same as minimizing quadratic version of that linear fn (its monotone transformation). The choice of $\frac{1}{2}$ is to make things easier when we take derivatives, to come up with closed form solns.

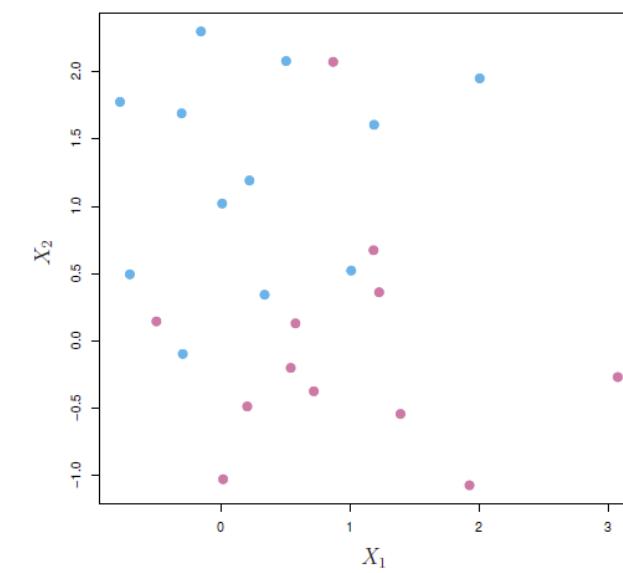
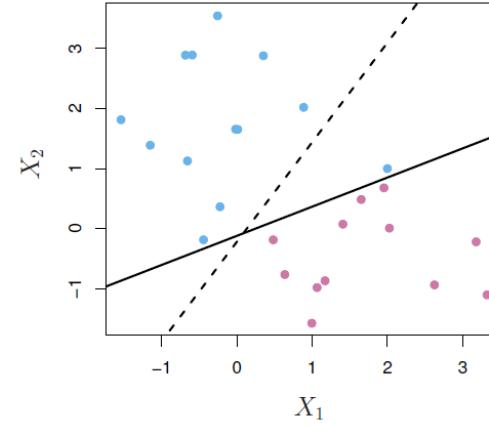
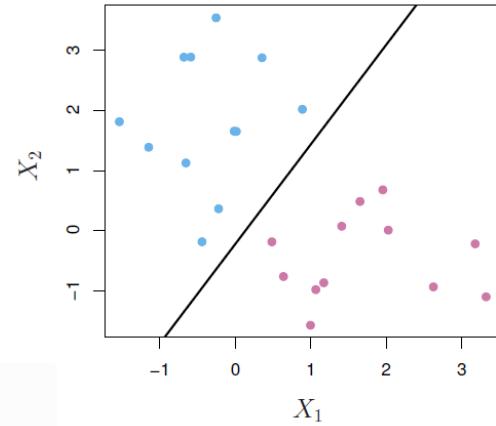


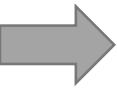
We don't need to do Gradient descent to find w^* & b^* like in Logistic Regression as MMC has close form soln. The only caveat is that sometimes the constraint is not satisfied & we need relax it.



Support Vector Classifier (SVC) – Soft Margin

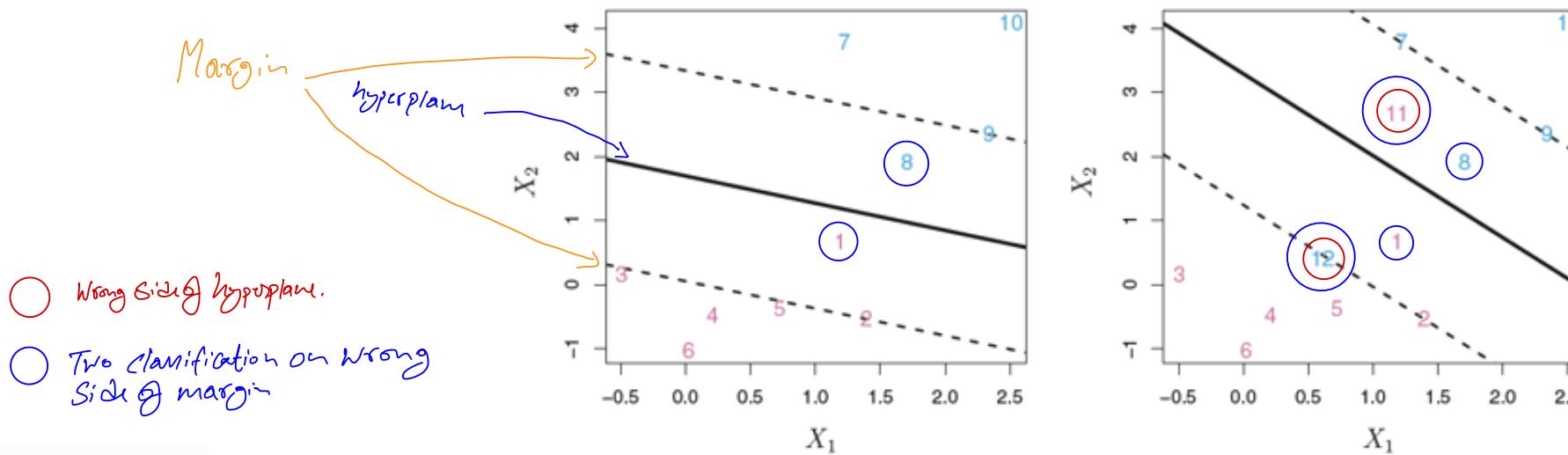
- The MMC optimization problem becomes infeasible whenever the condition cannot be satisfied, that is, when **a simple line cannot perfectly separate the labels**, no matter the choice of coefficients.
- This happens when:





Support Vector Classifier (SVC) – Soft Margin

- **Solution:** we can extend the concept of a separating hyperplane in order to develop a hyperplane that **almost** separates the classes, using a so-called **soft margin**.
- The generalization of the maximal margin classifier using soft margin is known as the **support vector classifier (SVC)**.
- It could be worthwhile to **misclassify a few training observations** in order to do a **better job in classifying the remaining observations**.



SVC optimization problem

- Soft margin classification adds a **penalty (C)** to the objective function for observations in the training set that are misclassified. In essence, the SVM algorithm will choose a decision boundary that optimizes the trade-off between a wider margin and a lower total error penalty.
- **Slack variable ξ_i** allow some observations to fall on the wrong side of the margin, but will penalized them by parameter **C: Cost of misclassification**

Objective fn:

$$\underset{w,b}{\text{Min}} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^I \xi_i$$

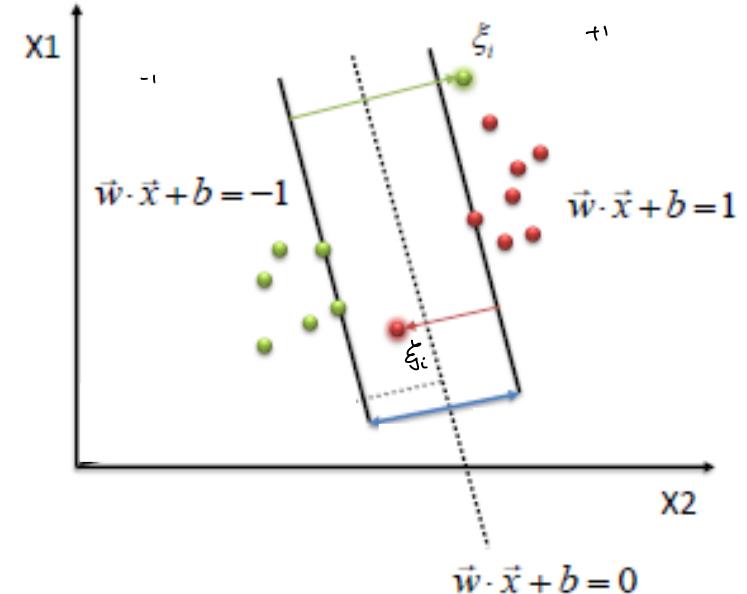
Constraints:

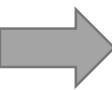
$$s.t. \quad y_i \left(\sum_{k=1}^K w_k x_{i,k} + b \right) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i$$

ξ_i = distance of Points that are wrong side of the margin.

- Increasing Cost C will make Margin narrower.
- C is a hyperparameter that needs tuning to balance trade off between Bias & Variance.

$$\begin{cases} \sum_{k=1}^K w_k x_{i,k} + b \geq +1 - \xi_i & \text{when } y_i = +1 \\ \sum_{k=1}^K w_k x_{i,k} + b \leq -1 + \xi_i & \text{when } y_i = -1 \end{cases}$$





Regularization parameter

C : Cost of misclassification

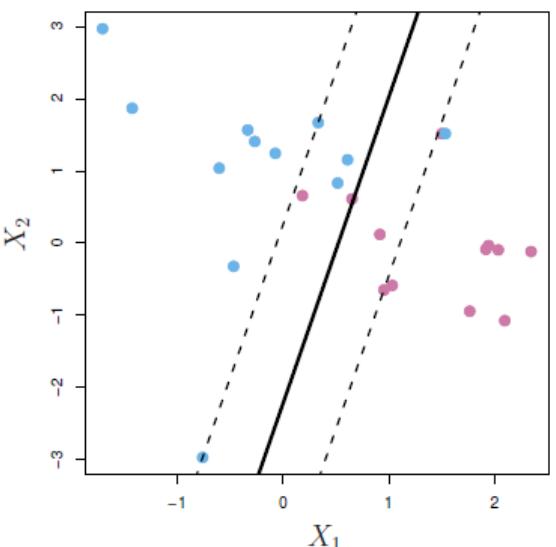
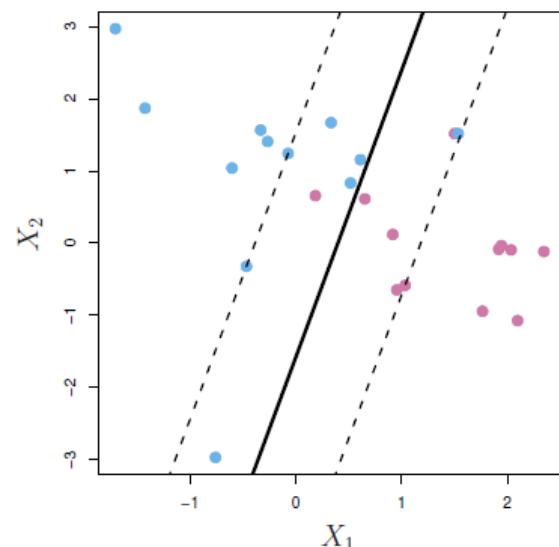
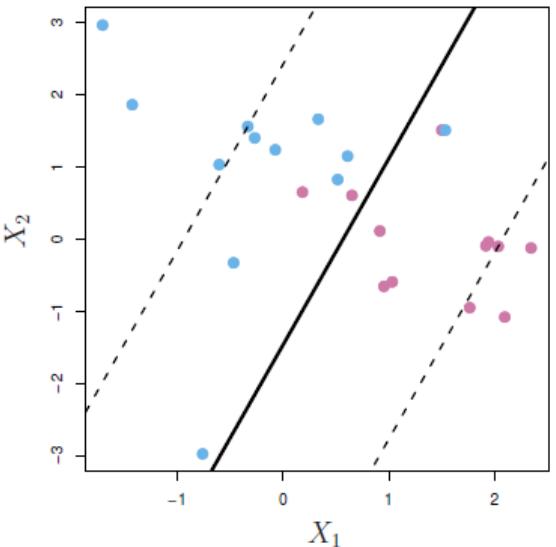
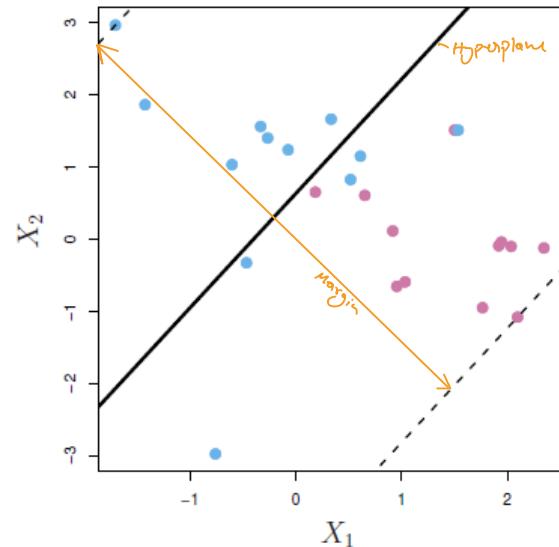


Small C :
wide margin : high bias : low variance

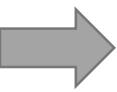
Large C :
narrow margin : low bias : High variance

Oversetting \rightarrow v. Large C

Small C

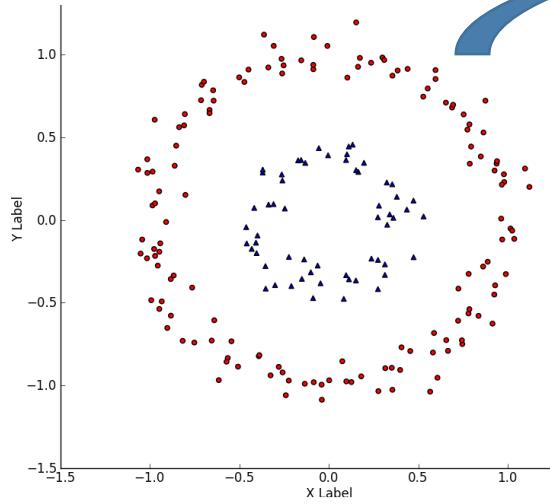


large C

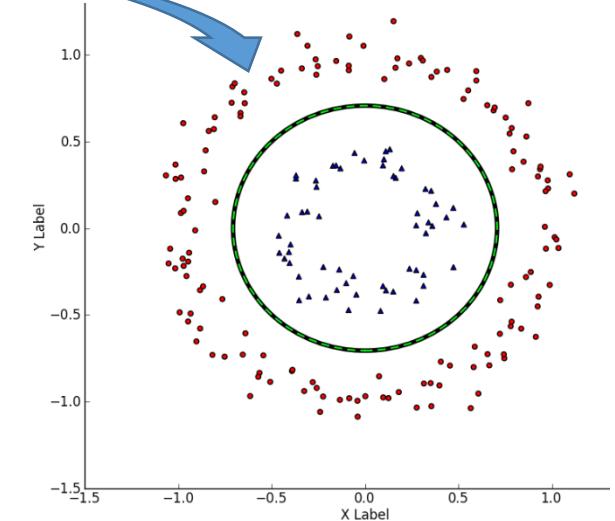
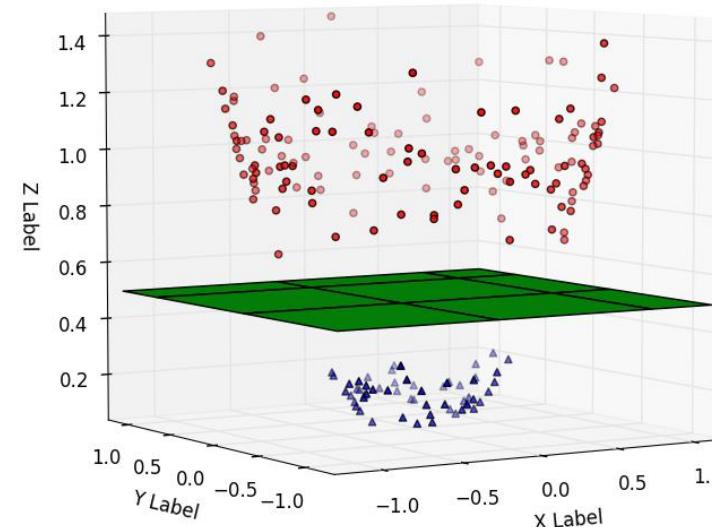


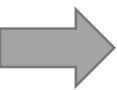
Kernel Trick!

- Non-linearly separable data: sometime a linear boundary simply **won't work**, no matter what value of C.
- We need a non-linear decision boundary!
- Mapping to higher dimensional space, finding the hyper plane and projecting it back to low dimensional space can be **computationally expensive**.
- Solution: **Kernel Trick!**



Kernel Trick Without going to higher dimension





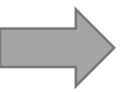
Support Vector Machines (SVM)

• Linear \rightarrow SVC
• Nonlinear \rightarrow SVM

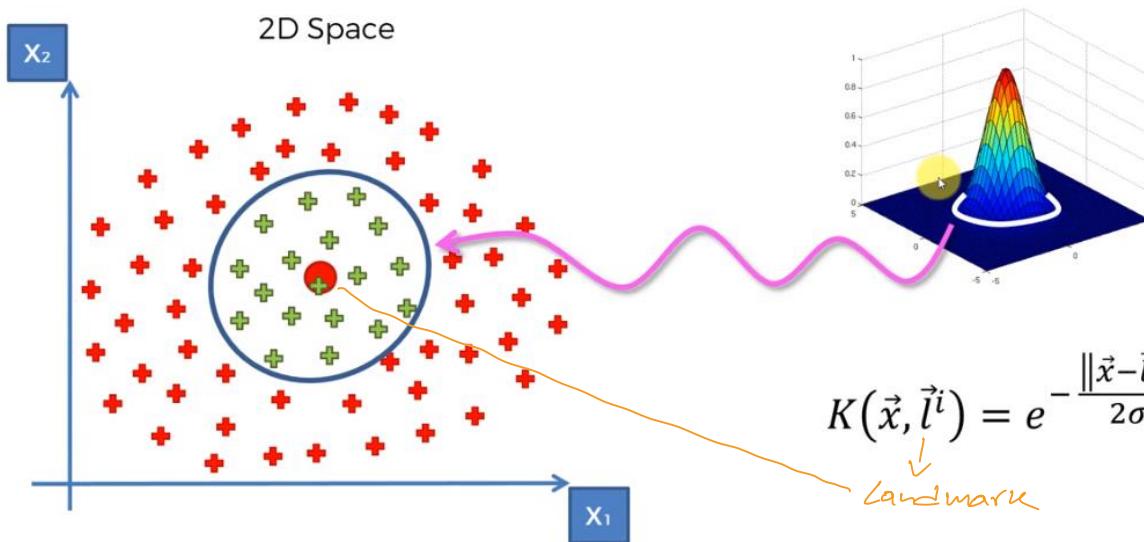
- SVM generalizes the SVC to a nonlinear model, via the **kernel ϕ** which is applied to the input points $x_{i,k}$.
- The Kernel $\phi(x_{i,k})$ is a function that quantifies the **similarities** between observations by summarizes the relationship between every single pairs in the training set.

SVC + Non-linear Kernel = SVM

$$\begin{aligned} \text{Min}_{w,b} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^I \xi_i \\ \text{s. t.} \quad & y_i \left(\sum_{k=1}^K w_k \phi(x_{i,k}) + b \right) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i \end{aligned}$$



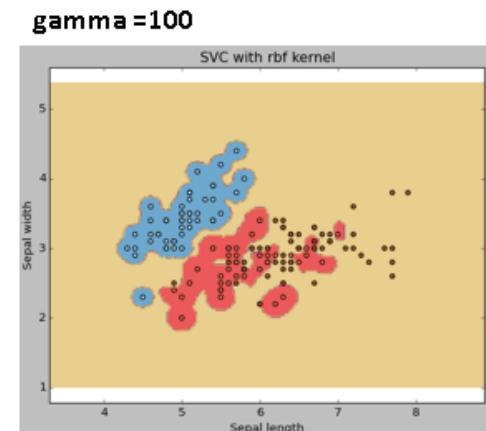
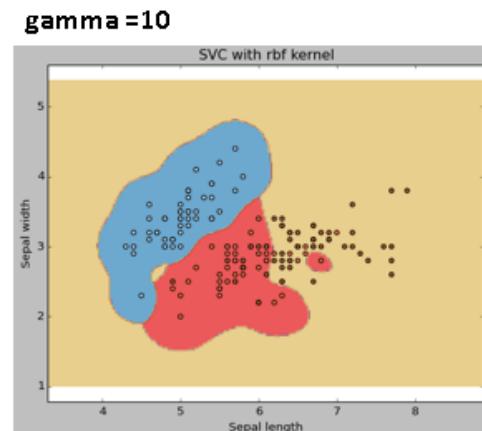
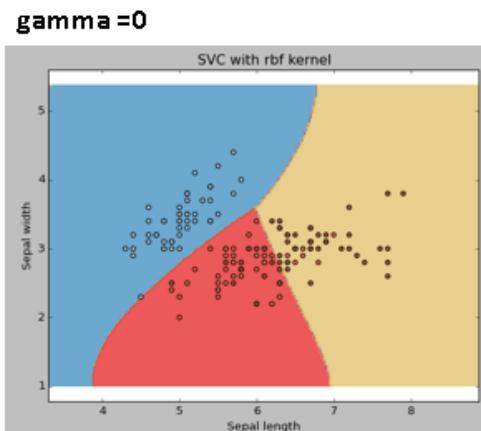
The Gaussian RBF Kernel (Radial Basis Function)

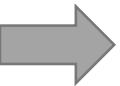


$\gamma \uparrow \Rightarrow \sigma \downarrow \Rightarrow$ close observation

$\gamma \downarrow \Rightarrow \sigma \uparrow \Rightarrow$ further observations

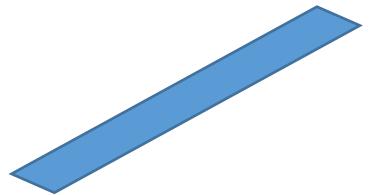
[γ is a free parameter that defines how far the influence of a single training example reaches, with low value meaning far & high meaning close. It is seen as inverse of radius of influence of samples selected by modulus Support Vectors.]



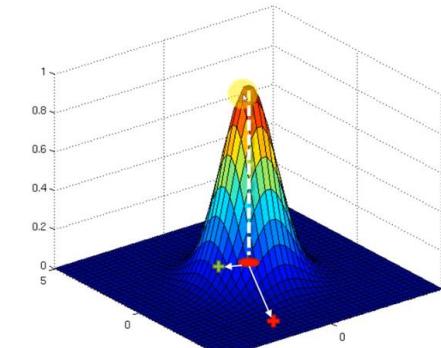


Most common types of Kernel

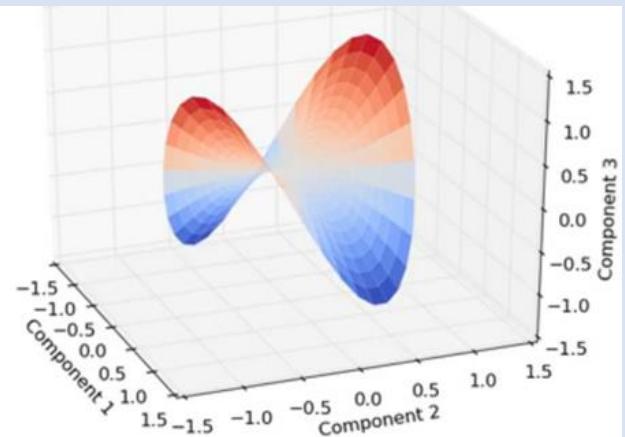
Linear Kernel (SVC)



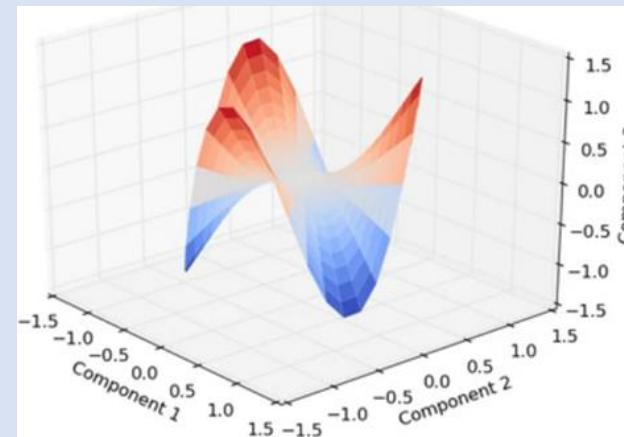
The Gaussian RBF Kernel

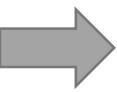


Polynomial Kernel

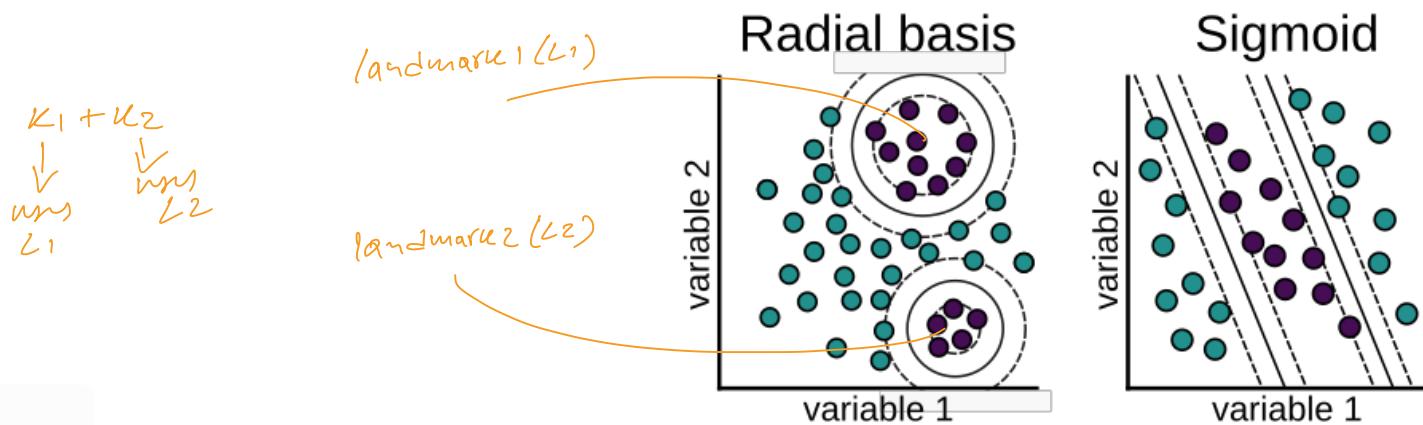
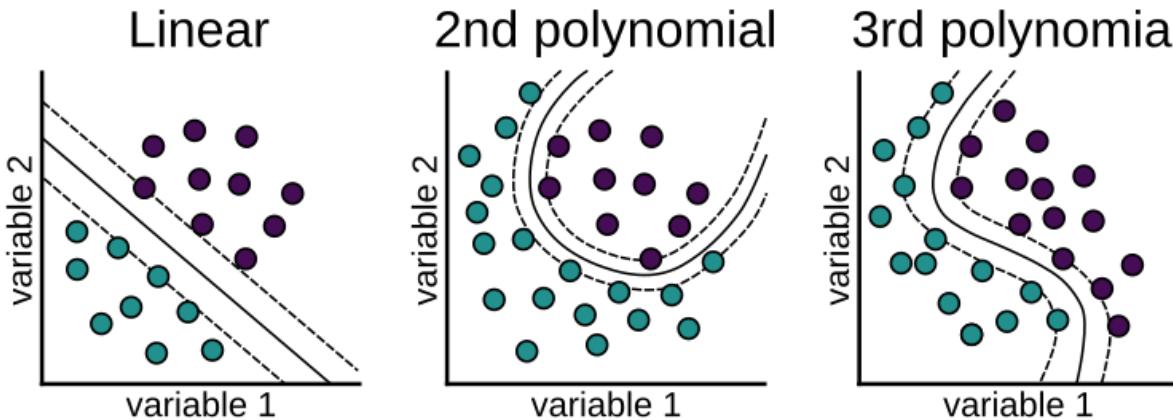


Sigmoid Kernel

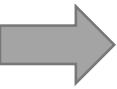




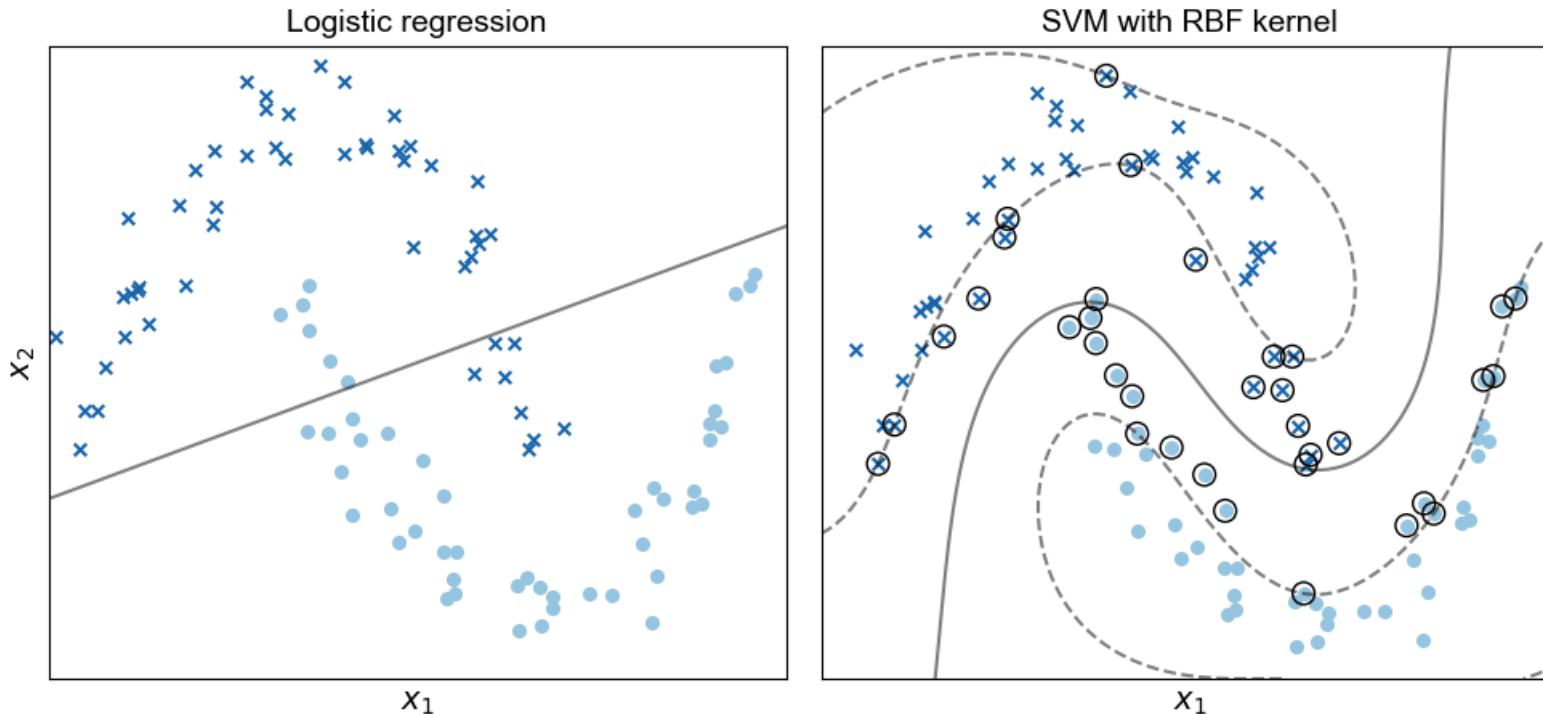
Decision boundaries with different Kernels



Source: [Machine learning with R, tidyverse, and mlr](#)



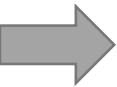
Comparing Logistic Regression and SVM



Source: <http://gregoryundersen.com/blog/2019/12/23/random-fourier-features/>

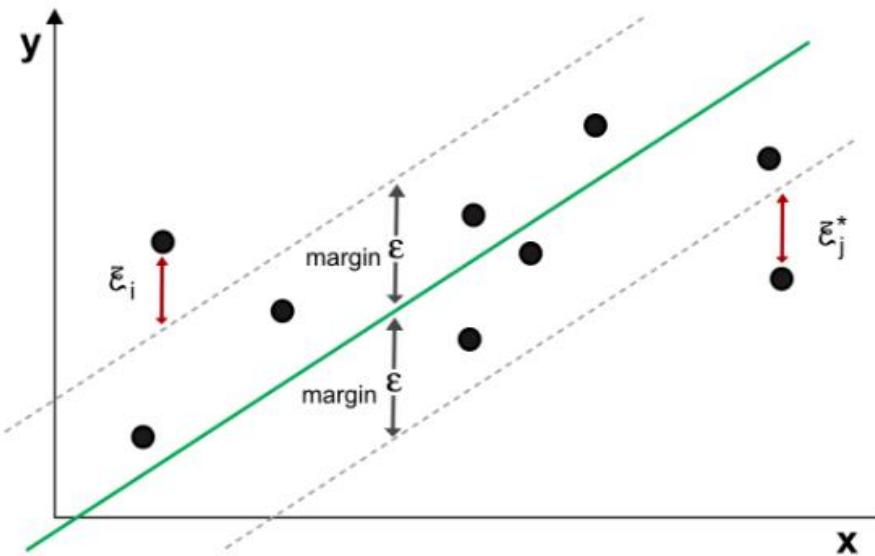
Part III

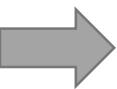
Support Vector Regressors (SVR)



SVM for regression (Support Vector Regressors)

- The idea of SVM classification can be transposed to regression problems.
- However, the **role of the margin** is different. Our objective, is to basically find the hyperplane that holds maximum training observations within the margin ϵ (tolerance level).

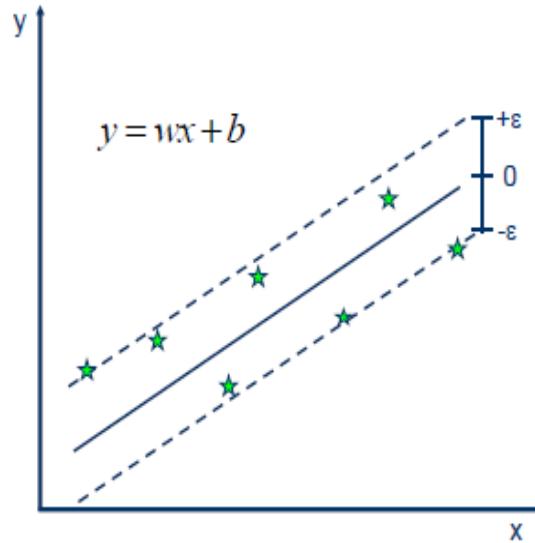




SVR optimization

ξ = Distance of observations outside of Margin

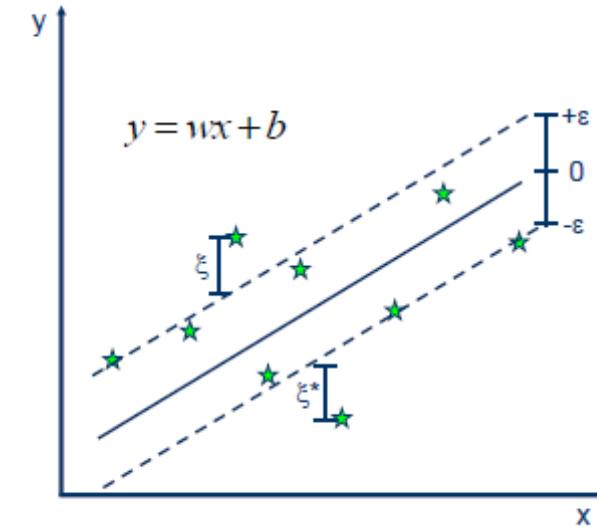
- $C \uparrow \Rightarrow \xi \downarrow \Rightarrow$ Flexible Model \Rightarrow Bias $\downarrow \Rightarrow$ Var \uparrow (Flexible Margin)
If $C=0$, Model will always predict $y=\bar{y}$. (Horizontal line)



- Minimize:
$$\min \frac{1}{2} \|w\|^2$$
- Constraints:
$$y_i - wx_i - b \leq \varepsilon \rightarrow y - \hat{y} \leq \varepsilon$$

$$wx_i + b - y_i \leq \varepsilon \rightarrow \hat{y} - y \leq \varepsilon$$

Similar to Hard Margin just like MMC



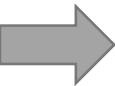
- Minimize:
$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*)$$
- Constraints:
$$y_i - wx_i - b \leq \varepsilon + \xi_i$$

$$wx_i + b - y_i \leq \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

Similar to Soft Margin just like SVC.

Source: https://www.saedsayad.com/support_vector_machine_reg.htm



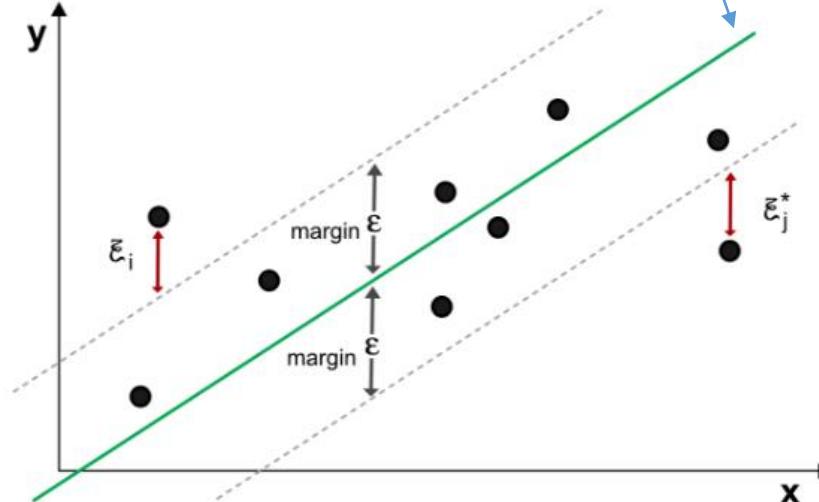
Kernel SVR optimization

(similar to SVM)

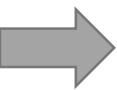
Predictions:

$$\sum_{k=1}^K w_k \phi(x_{i,k}) + b$$

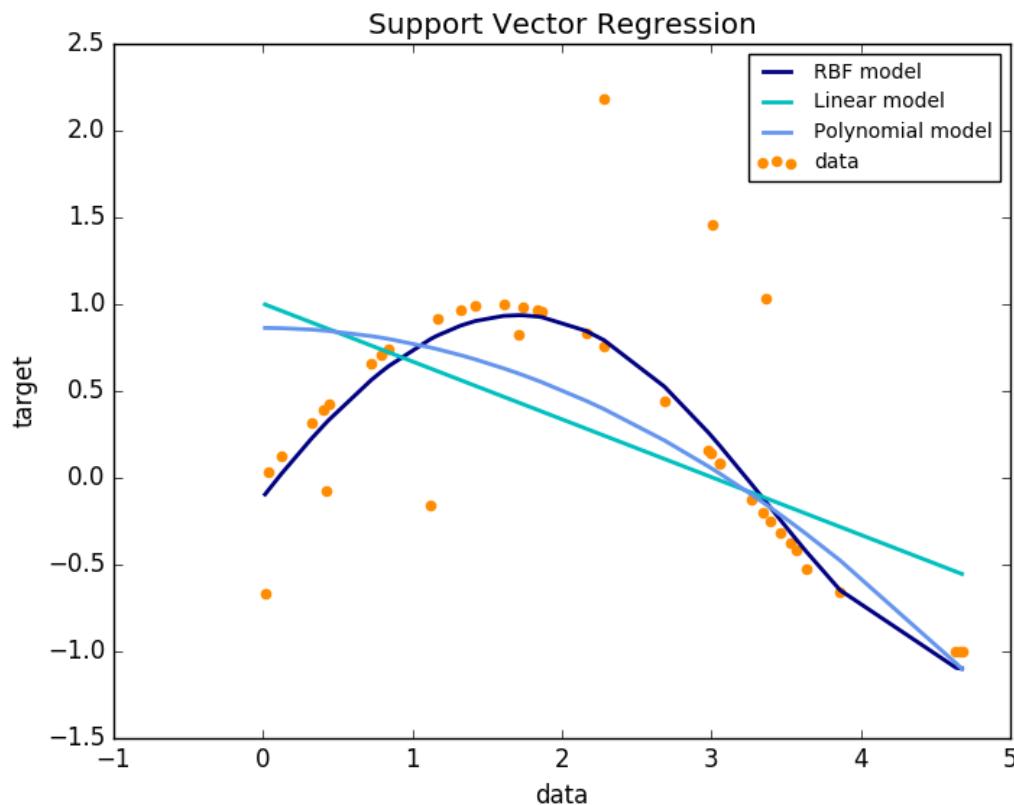
$$\begin{aligned} \text{Min}_{w,b} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^I (\xi_i + \xi_i^*) \\ & \left(\sum_{k=1}^K w_k \phi(x_{i,k}) + b \right) - y_i \leq \epsilon + \xi_i^* \\ & y_i - \left(\sum_{k=1}^K w_k \phi(x_{i,k}) + b \right) \leq \epsilon + \xi_i \\ & \xi_i, \xi_i^* \geq 0 \quad \forall i \end{aligned}$$



- Goal: minimize the sum of squared weights subject to the error being small enough
- This is somewhat the opposite of the penalized linear regressions which seek to minimize the error, subject to the weights being small enough



SVR using Linear and Non-linear Kernels



RBF seems to do the best job in Finance Applications.

Source: Scikit learn documentation

Classification & Regression.

Part IV

Tuning hyperparameters

SVM pros and cons

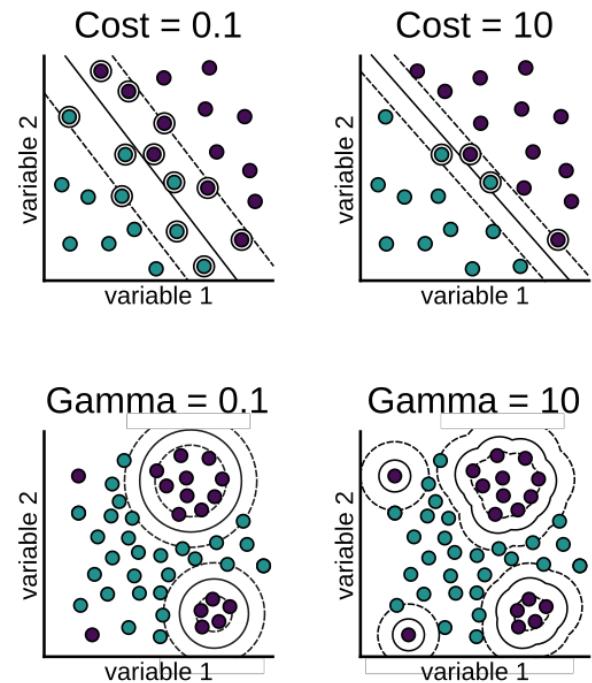
SVM applications in Finance

Tuning hyperparameters

SVM hyperparameters:

- 1) **C**, Cost of misclassification: controls bias variance trade off
- 2) Kernel
- 3) **Gamma**, controls how far the influence of a single training set reaches

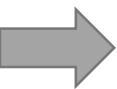
$$\sigma \propto \frac{1}{\sigma^2}$$



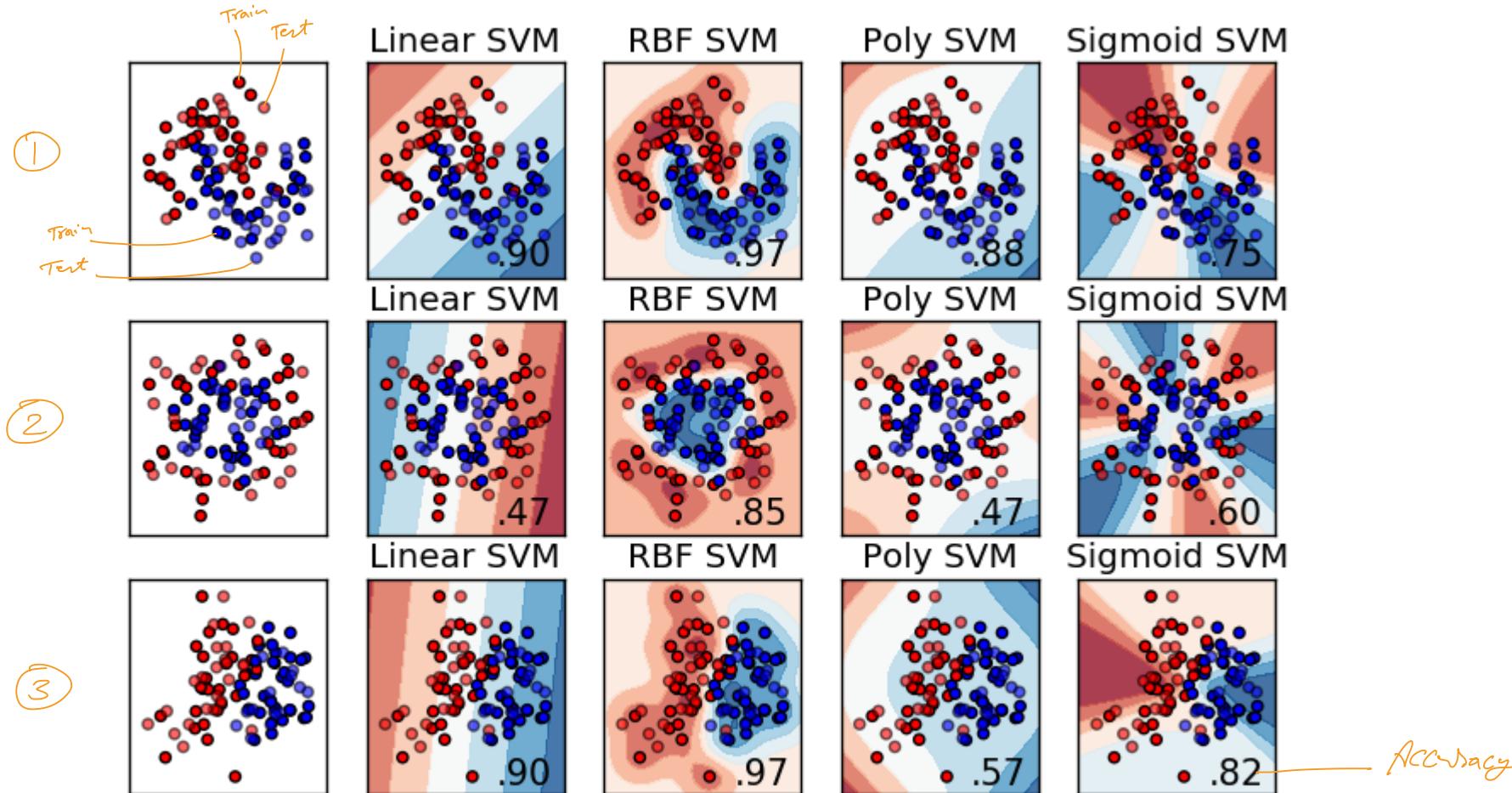
Source: [Machine learning with R, tidyverse, and mlr](#)

Grid search cross validation is used to tune the hyper parameters.

Kernel	C	Gamma	CV
Linear, rbf, poly,	0.1, 1, 10, 100, ...	0.001, 0.01, 0.1, 1, ...	5,10,...



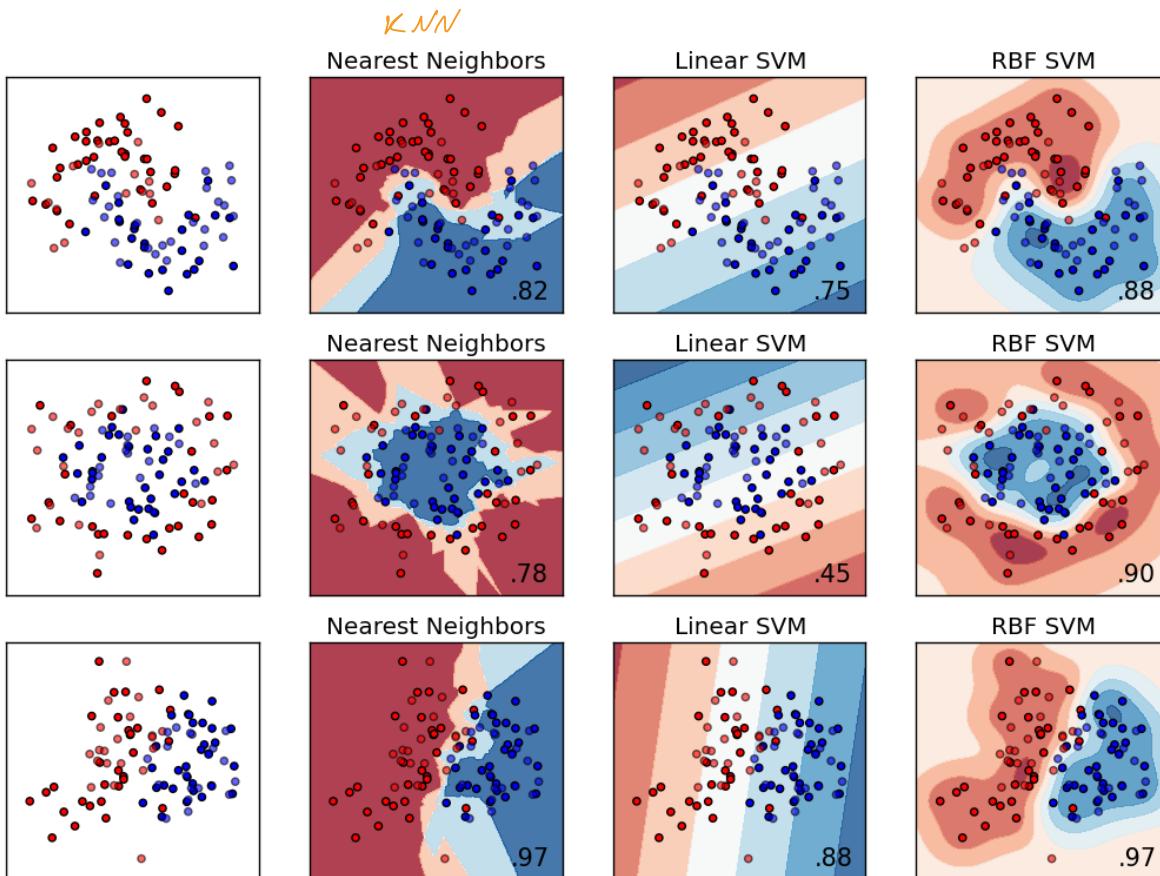
Decision boundaries with different Kernels *(Classification)*



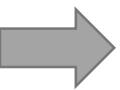
Source <https://www.kaggle.com/residentmario/kernels-and-support-vector-machine-regularization>

Comparing classifiers (so far)

- For large sample size with low dimensions, KNN would outperform SVM. (Also for noisy data sets.)

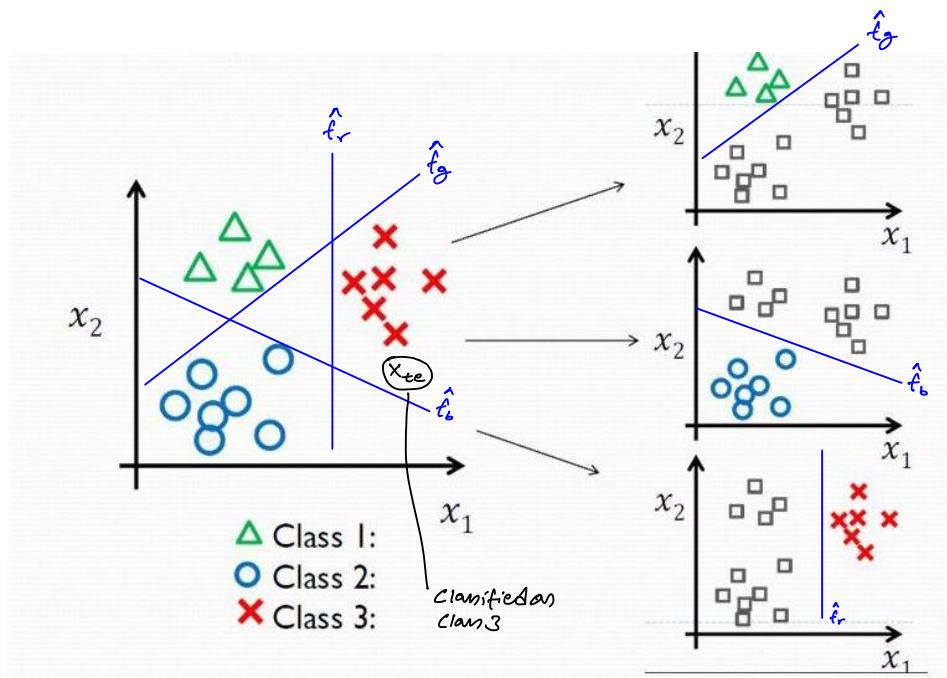


Source: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html



K-Multiple class SVM

- One-VS-All (OVA)
 1. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, each class versus the rest. Here $K=3$.
 2. Classify x_{te} to the class for which $\hat{f}_k(x_{te})$ is largest.



$$\begin{array}{c} \hat{f}_r(x_{te}) \\ \hat{f}_b(x_{te}) \\ \hat{f}_g(x_{te}) \end{array} \quad \begin{array}{c} 3 \\ -1 \\ -7 \end{array}$$

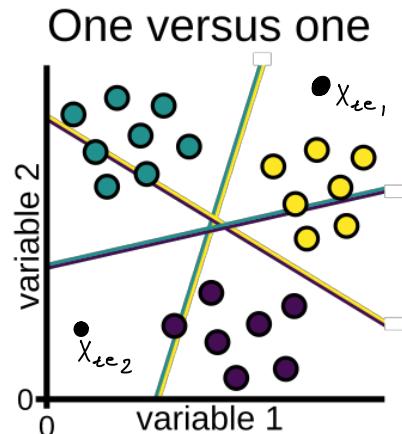
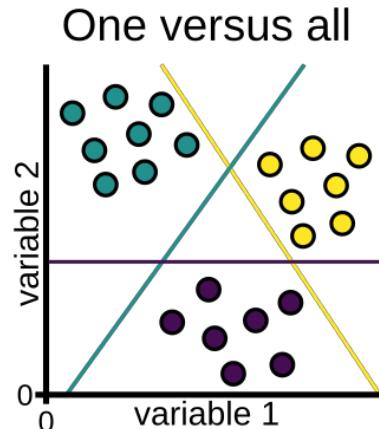
largest so classified as Red (cross) Class 3.

\hat{f} is the distance from hyperplane so, the further you are from hyperplane, more confident you are that the observation belongs to that class.

K-Multiple class SVM

- One-VS-One (OVO)

- Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{kl}(x)$, each class versus the rest
- Classify x_{te} to the class that wins the most pairwise competitions.

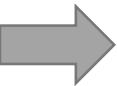


	\hat{f}_{gy}	\hat{f}_{gp}	\hat{f}_{yp}
x_{te_1}	g	g	y
x_{te_2}	y	p	p

$\Rightarrow g \text{ (classified)}$
 $\Rightarrow p \text{ (classified)}$

If $K=10$,
 • OVA $\Rightarrow 10 \hat{f}$
 • OVO $\Rightarrow \binom{10}{2} = 45$ eg: 3 classes
 $R_d \Rightarrow \binom{3}{2} = 3$
 $R_g \quad RL \quad \hookrightarrow$

If no. of classes is small
 OVO will be better comparison
 but as $K \uparrow$ OVO is very costly.



SVM's Pros and Cons

Pros:

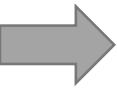
- SVM can be **memory efficient!** uses only a subset of the training data (support vectors)
- Can handle non-linear data sets
- Can handle high dimensional spaces (even when $D > N$) Dimension > Sample Observations
- Used both for classification and regression
- Linear SVM are not very sensitive to overfitting (soft margin; regularization)
- Can have high accuracy (even compared to NN)

→ compare to KNN

Cons:

- No probability outcome!
- Long training time when we have large data sets.
- Limited interpretability (specially for Kernel SVM)
- Does not perform well with noisy data
- Suited for small to medium size data



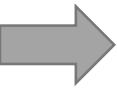


SVM's Applications in finance

- Corporate financial statements and bankruptcy (high dimensional)
- Identifying stressed companies to short sell (using many fundamental and technical features)
- Sentiment analysis (classify text from documents e.g., news articles, company announcements, and company annual reports into useful categories for investors)
- Money laundering analysis and spam detection
- Loan management



Appendix



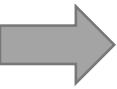
MMC solution

$$\operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } y_i \left(\sum_{k=1}^K w_k x_{i,k} + b \right) \geq 1$$

$$L(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^I \lambda_i \left(y_i \left(\sum_{k=1}^K w_k x_{i,k} + b \right) - 1 \right)$$

$$\frac{\partial L}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\lambda}) = \mathbf{0}, \quad \frac{\partial L}{\partial b} L(\mathbf{w}, b, \boldsymbol{\lambda}) = 0,$$

$$\mathbf{w}^* = \sum_{i=1}^I \lambda_i u_i \mathbf{x}_i.$$



Students' questions

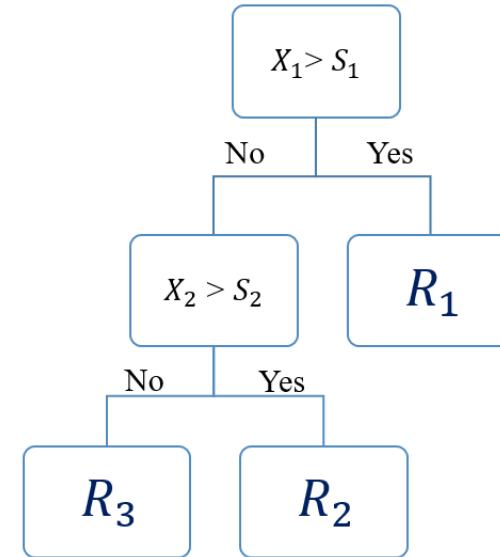
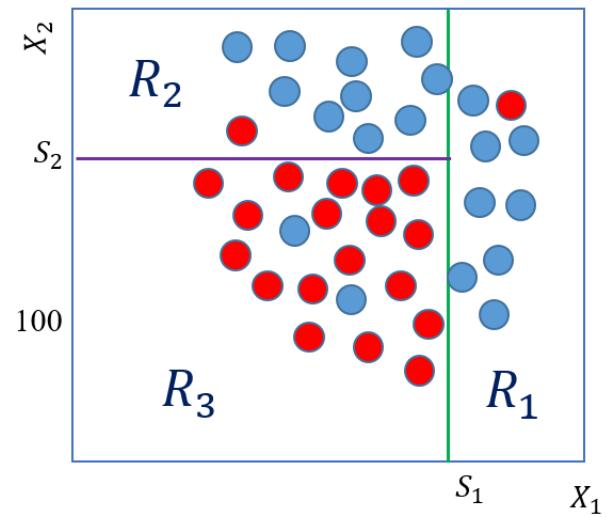
- 1) What does non-separable data mean?
- 2) Does the Kernel function act the same way as standardization conceptually, whereby the data is scaled up but the distribution of data does not change? *No*
- 3) When do we use MMC vs SVM?
- 4) Even after using the kernel trick, does SVM perform faster than KNN on average?
Depends on size of data

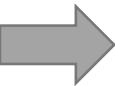
Class -18

Decision Trees (DTs)

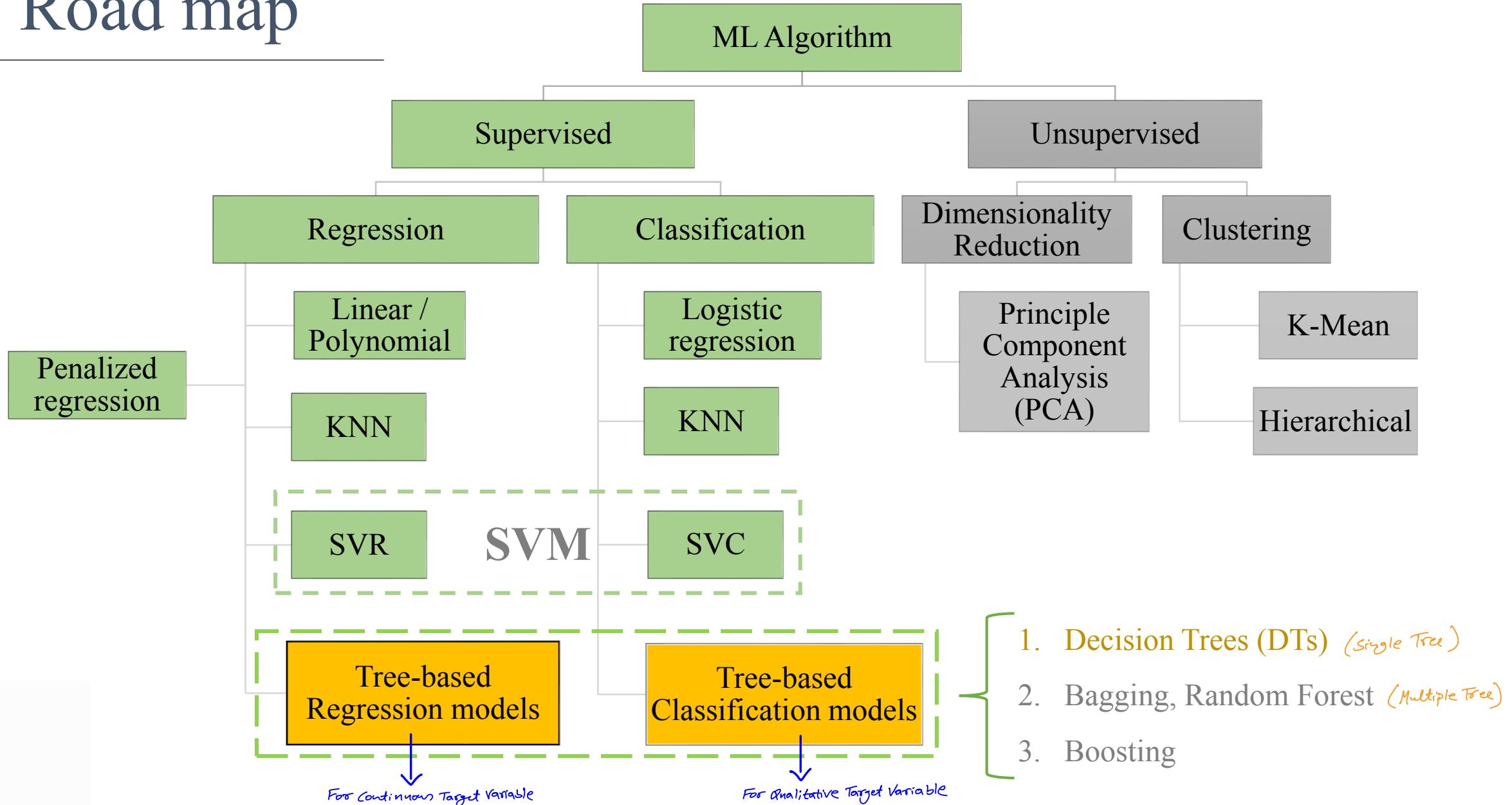
(Non-Parametric)
(Supervised learning)

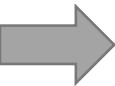
(Highly Interpretable)





Road map





Topics



Part I

1. Decision Trees Definitions
2. Decision Trees criteria
 - MSE
 - Error Rate
 - Gini Index
 - Entropy

Part II

1. Regression Trees *(use MSE)*
2. Classification Trees *(use Gini Index & Entropy)*

Part III

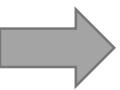
1. Pruning a tree
2. Hyperparameters

Part IV

1. Pros and Cons
2. Applications in Finance

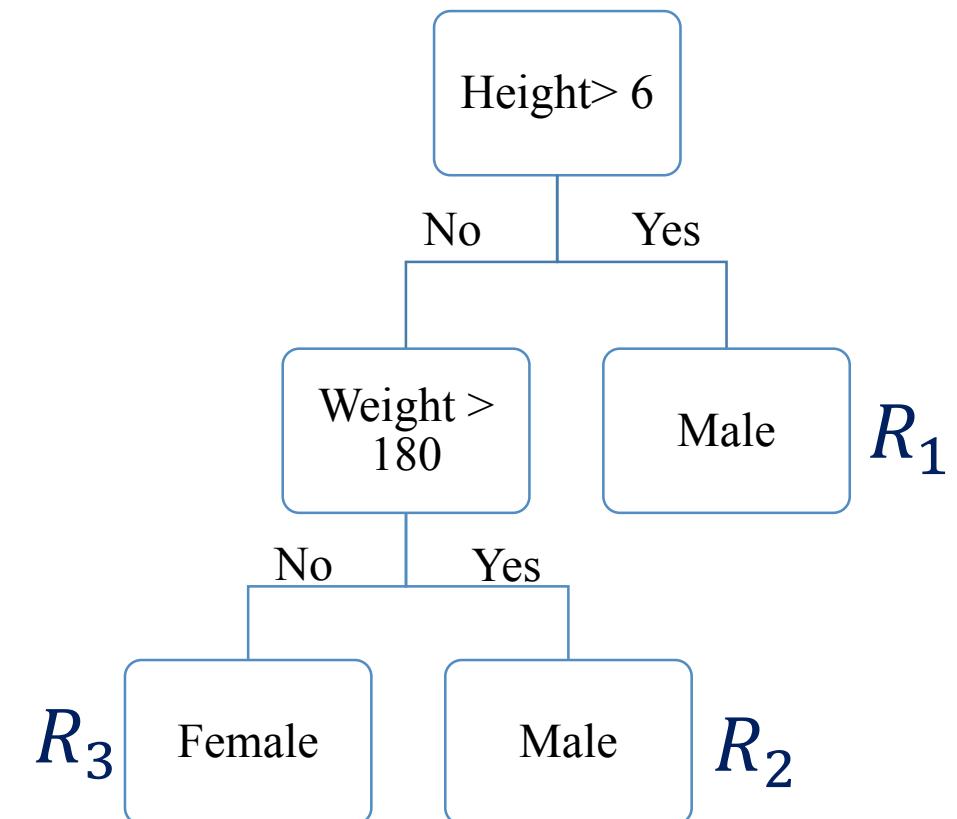
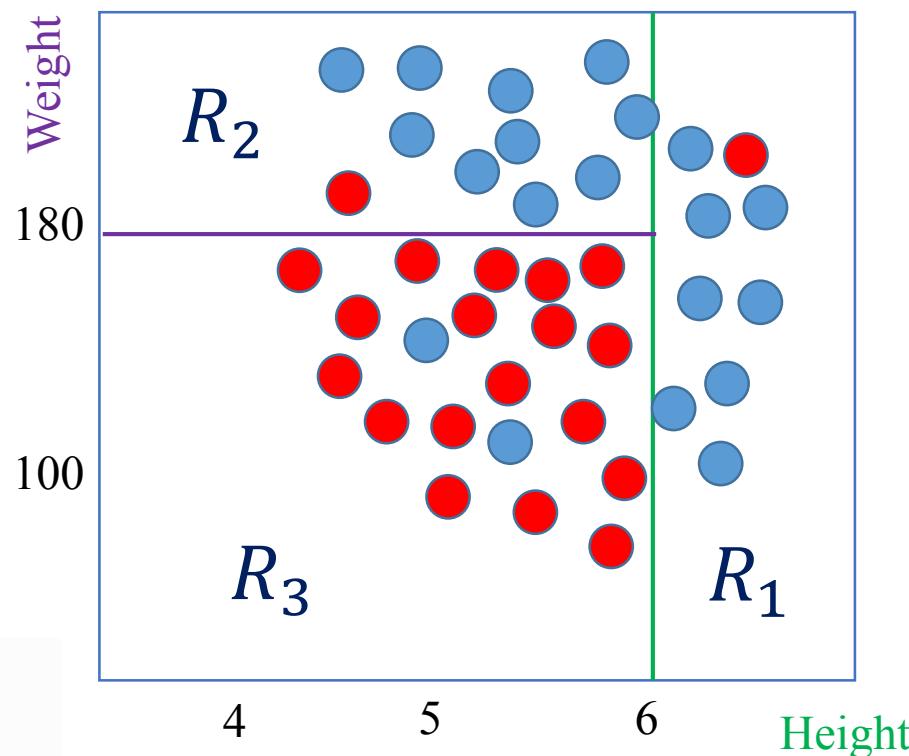
Part I

Decision Trees definitions and criteria



Decision Trees Definitions

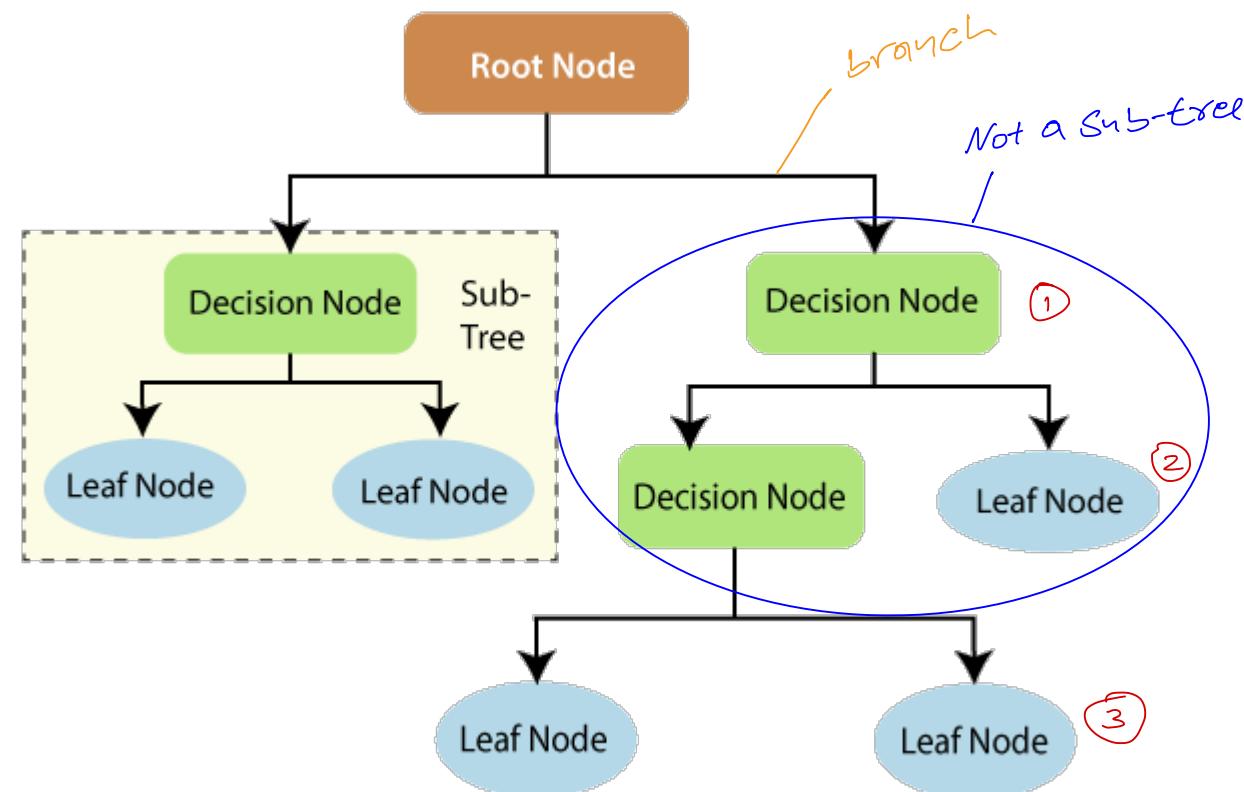
- DTs are ML algorithms that **progressively** divide data sets into **smaller data groups** based on a **descriptive feature**, until they reach sets that are small enough to be described by some **label**.
- DTs apply a top-down approach to data, trying to group and **label** observations that are similar.



→ Decision Trees Definitions

- When the target variable consists of **real numbers**: regression trees
- When the target variable is **categorical**: classification trees
- Terminology:

- ✓ Root node (*Beginning of Tree*) (*No input*)
- ✓ Splitting (*Split node to 2 or more nodes*)
- ✓ Branch (*Line connecting nodes*)
- ✓ Decision node (internal node) (*Has input & output*)
- ✓ Leaf node (terminal node) (*Has input, no output*)
- ✓ Sub-tree (*Contains node & terminal node*)
- ✓ Depth (level) (*longest Path from root node to terminal node*)
- ✓ Pruning (*Removing sub-nodes from a tree & replace with terminal node*)



Depth = 3

Decision Trees Criteria

- Which split adds the most information gain (minimum impurity)?

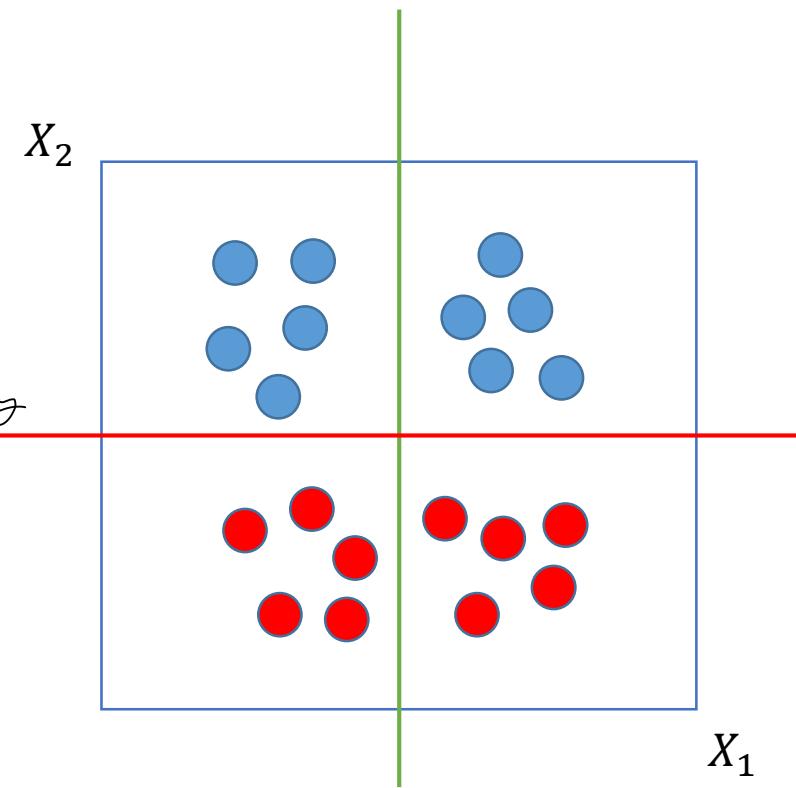
- Regression trees: MSE

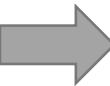
- Classification trees:

1. Error rate
2. Entropy
3. Gini Index

They all measure
impurity

Control how a Decision Tree
decides to **split** the data





Decision Trees Criteria

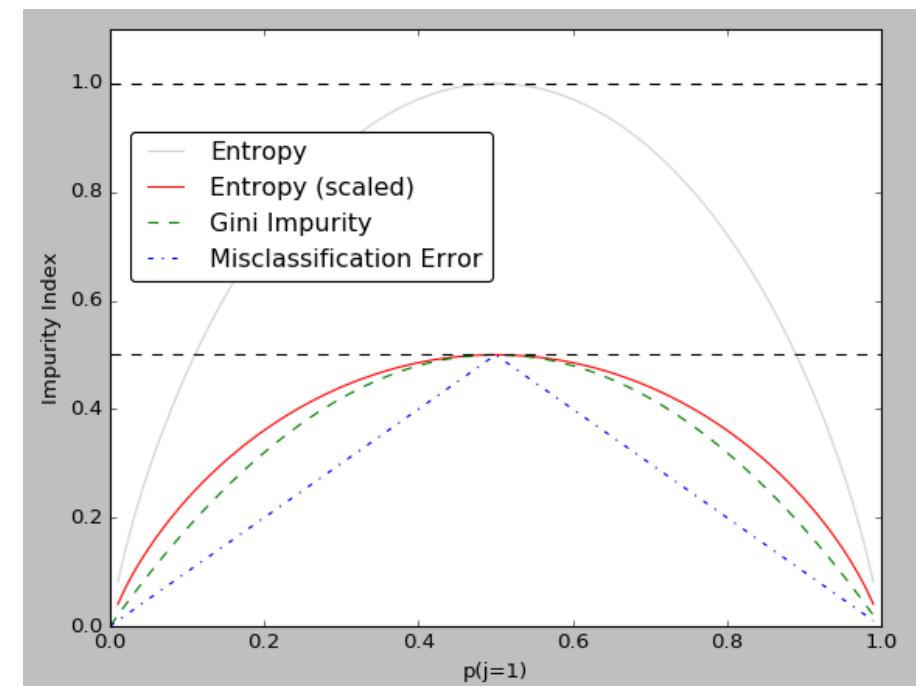
[In entropy formula, we multiply by minus because log₂ probabilities are -ve as 0 ≤ p ≤ 1.]

- **Entropy:** Measures the impurity or randomness (uncertainty) in the data points
- **Gini Index:** Measure how often a randomly chosen element would be incorrectly labeled
- For both Entropy and Gini, 0 expresses all the elements belong to a specified class (**pure**)
- Different decision tree algorithms utilize different impurity metrics

$$\text{entropy} = - \sum_j p_j \log_2(p_j)$$
$$Gini = 1 - \sum_j p_j^2$$

Goal is to minimize these.

p_j = Probability of element j belonging to Correct class

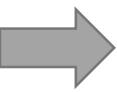


- Don't mix data Variance & model Variance. The trade off between Bias & Variance is Model Variance.
Entropy is maximum when distribution of your target Variable is Uniform. Ideally, we want Entropy=0 in Test set.

Part II

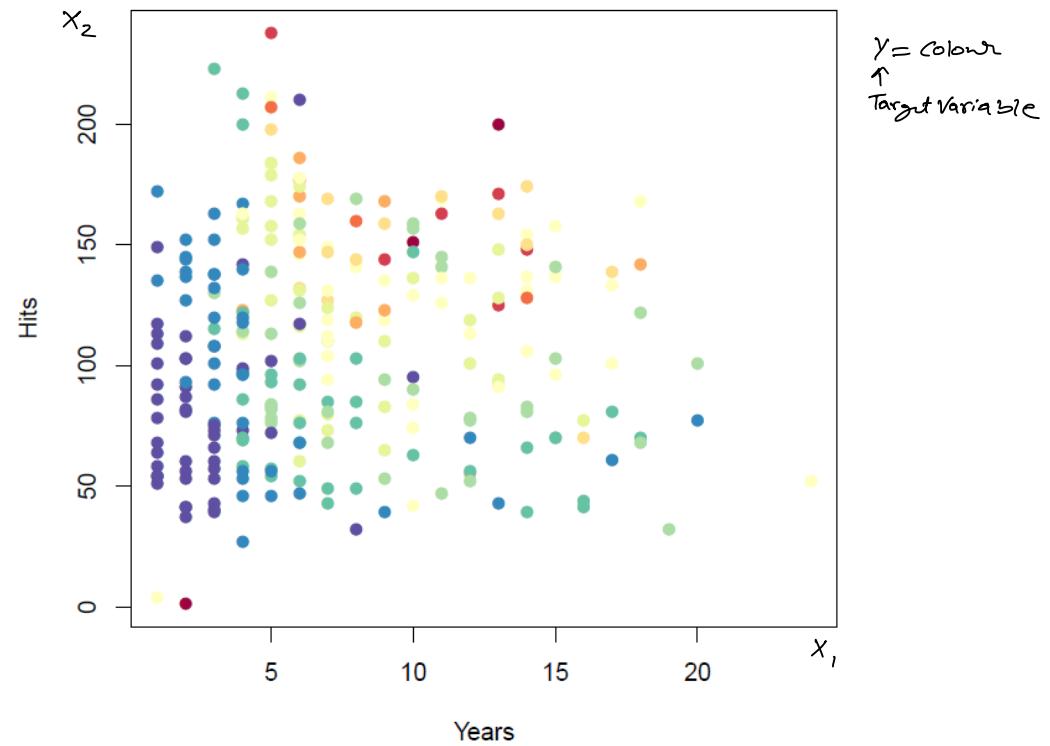
Regression / Classification Trees!

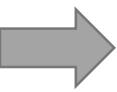
How does a decision tree work?



Regression Trees

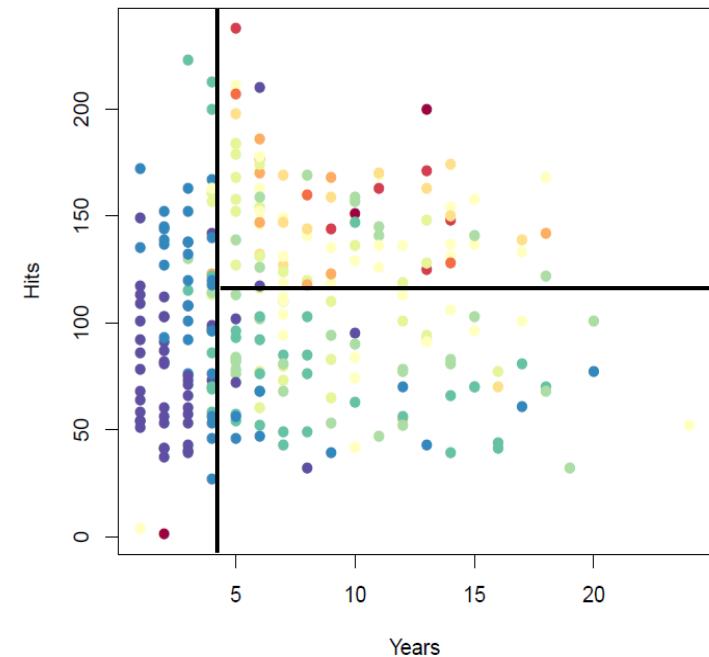
- Baseball Salary is color-coded from low (blue, green) to high (yellow, red)
- DTs apply a top-down approach to data, trying to group and label observations that are similar.
- The main questions in every decision-making process:
 1. Which feature to start with? *Years*
 2. Where to put the split (cut off)? *9.8*

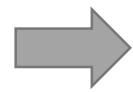




Interpreting the results

- Based on color-coded salary, it seems that **years** is the most important factor in determining **salary**.
- For less experienced players, the number of **hits** seems irrelevant.
- Among more experienced players thought, players with more **hits** tend to have higher **salaries**.
- As one can see, the model is very **easy to display, interpret and explain**.



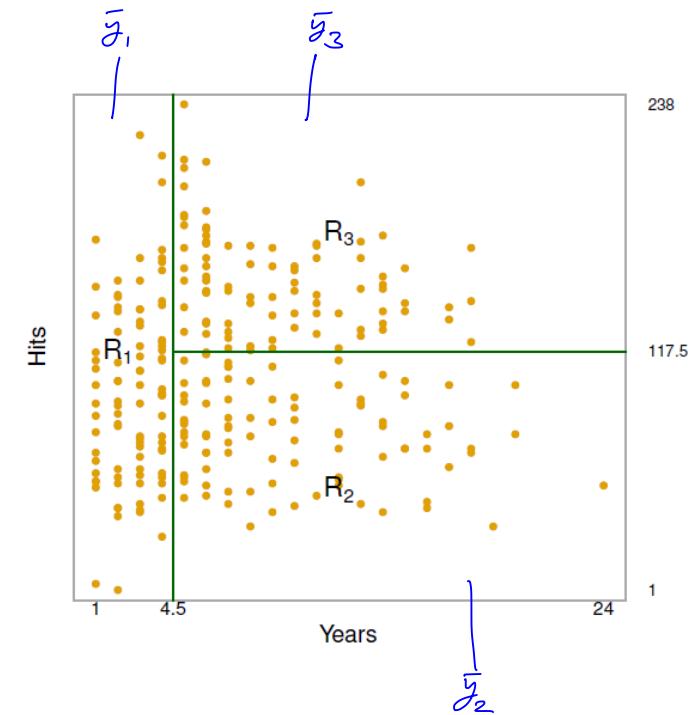


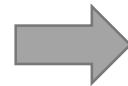
Tree building process

- Divide the feature space into **J distinct** and **non-overlapping** regions.
- For every observation that falls into the region R_j , we make the **same prediction**, which is simply the **mean of the target values** for the training observations in R_j .
- The goal is to find rectangles R_1, R_2, \dots, R_j that minimize the **RSS**:

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

- Where \hat{y}_{R_j} is the mean target for the training observations within the j^{th} rectangle.





Tree building process: Recursive Binary Splitting

(Top Down Greedy Algorithm)

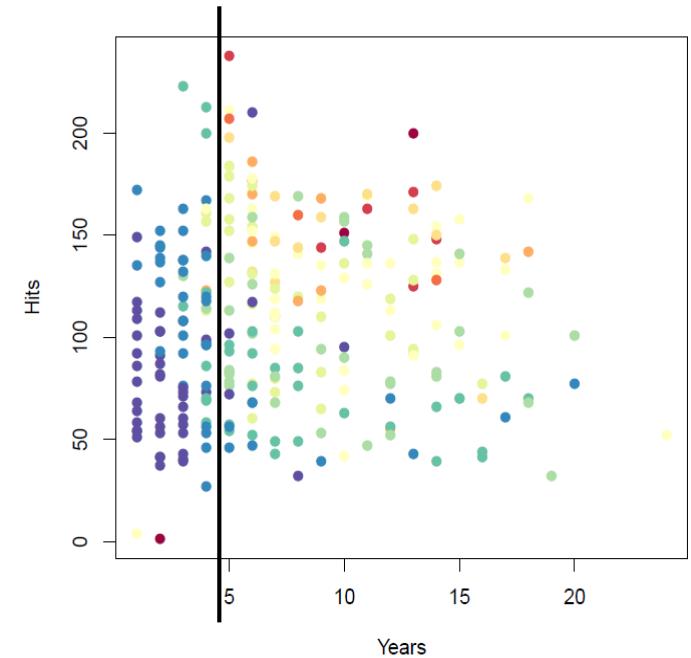
- How does the algorithm select X_j and the split s ?
what feature to start with ↑ *when to put the split.* ↑
- X_j and s are selected such that splitting the feature space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the largest possible reduction in RSS.

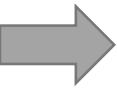
$$R_1(j, s) = \{X|X_j < s\} \text{ and } R_2(j, s) = \{X|X_j \geq s\}$$

- Seeking for the value of j and s that **minimized** the following equation:

$$\underset{j, s}{\text{Min}} \left\{ \sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2 \right\}$$

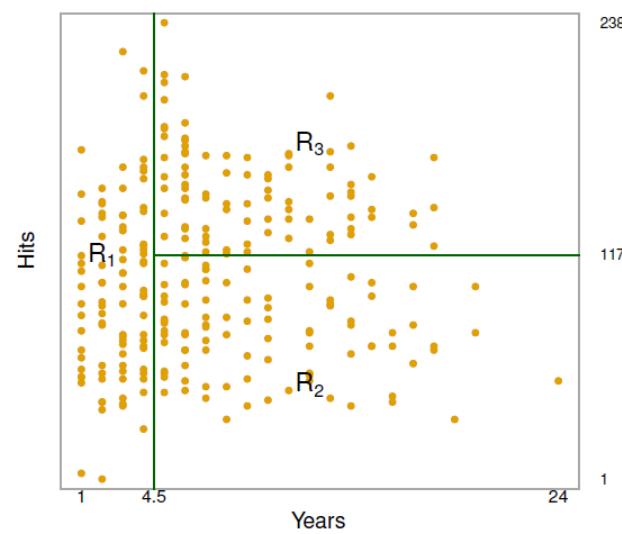
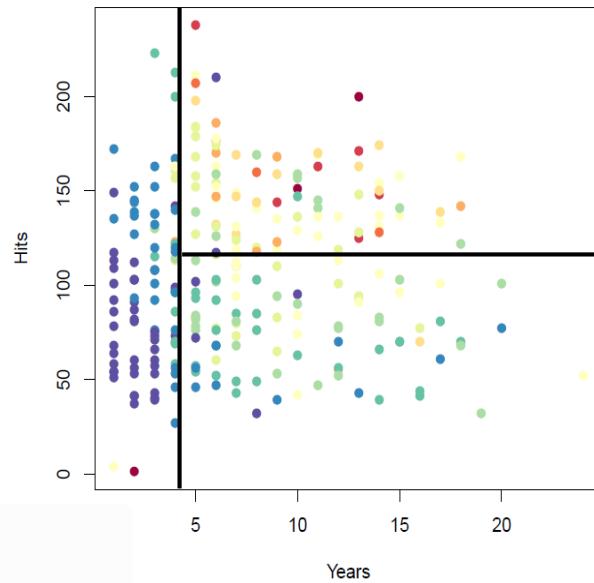
- The **best split** is made at that particular step, rather than **looking ahead** and picking a split that will lead to a better tree in some **future step**.

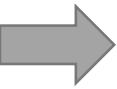




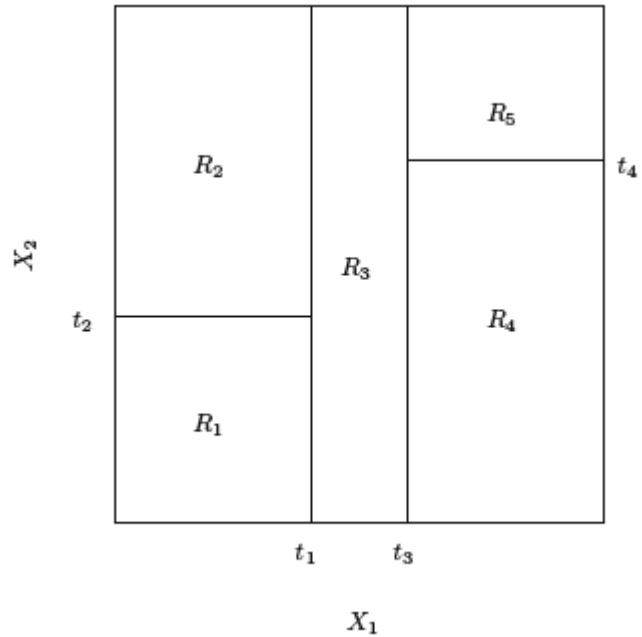
Tree building process: Recursive Binary Splitting

- Next, the algorithm repeats the process, looking for the **best feature and best split** in order to split the data further **to minimize the RSS** within each of the resulting regions.
- The process continues **until a stopping criterion** is reached; for instance, continues until no region contains more than a fixed number of observations.

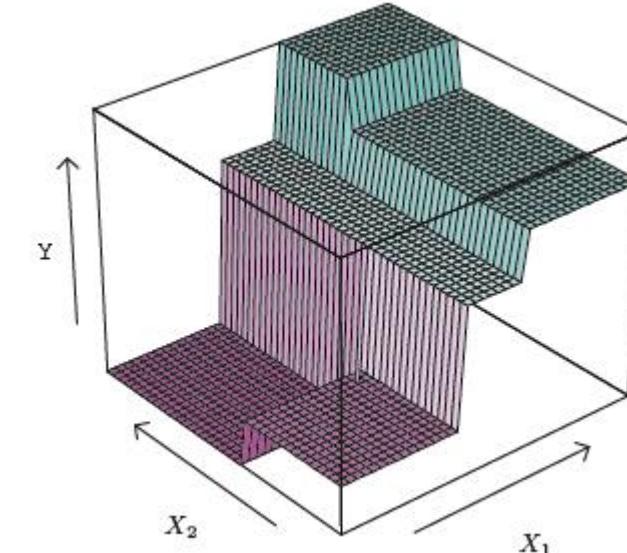
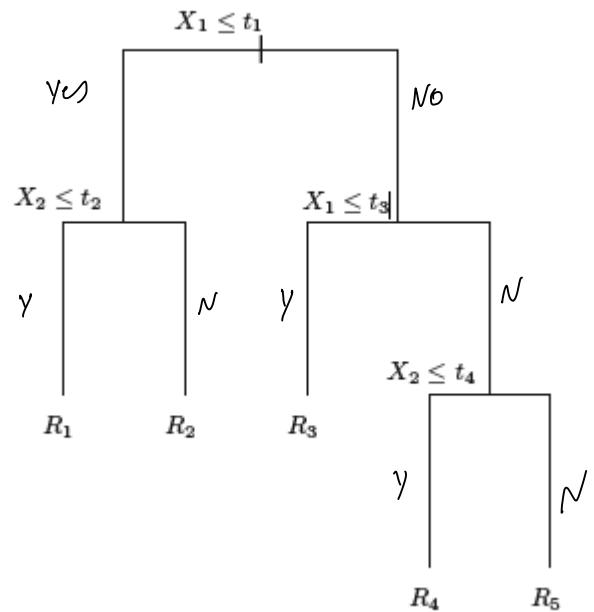


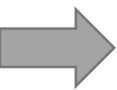


A Five-Region Example of Recursive Binary Splitting



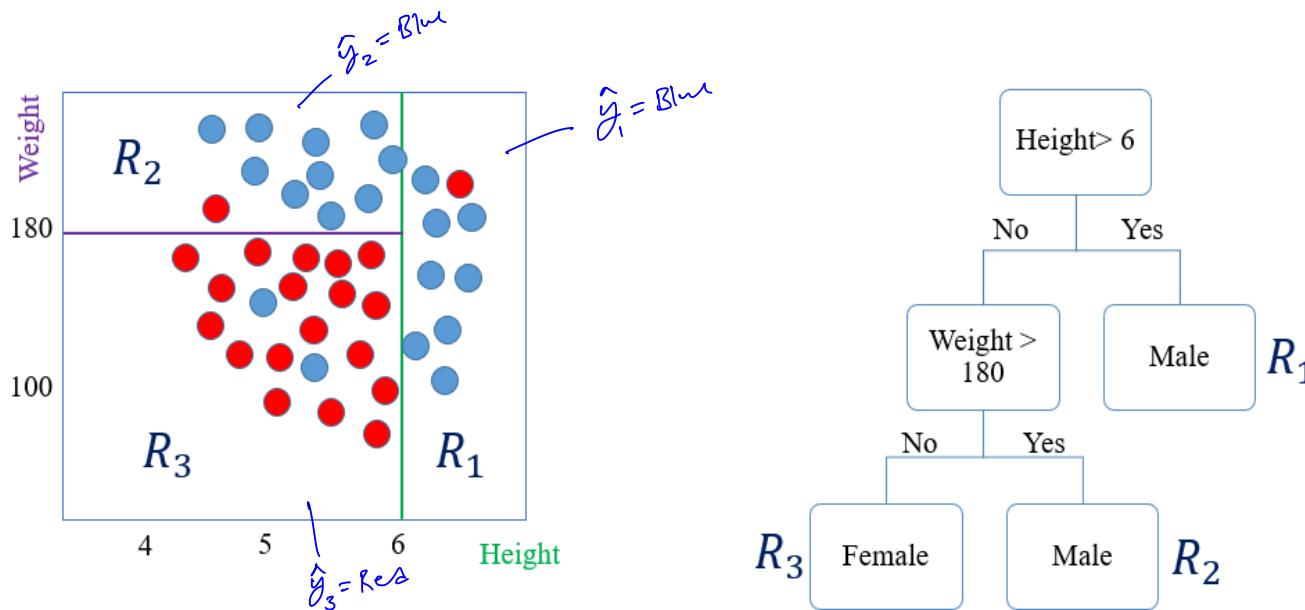
The output of **recursive binary splitting** on a two-dimensional example

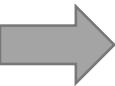




Classification Trees

- Classification trees are very similar to regression trees, except that it is used to predict a **qualitative** response rather than a **quantitative** one.
- The prediction of the algorithm at each terminal node will be the category with the **majority of data points** i.e., the **most commonly occurring class**.





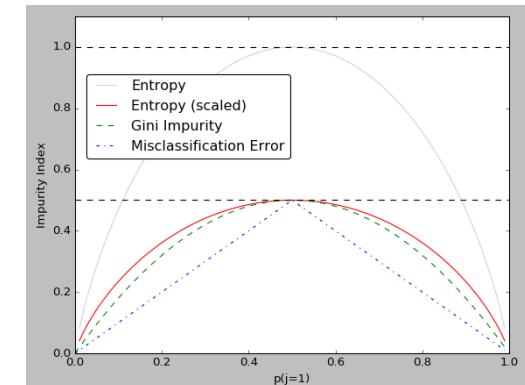
Classification Trees (details)

- Just as in the regression setting, the **recursive binary splitting** is used to grow a classification tree. However, instead of RSS we will be using one of the following impurity criteria:

1. Classification error rate: $\longrightarrow E = 1 - \max_k(\hat{p}_{mk})$

2. Gini index: $\longrightarrow G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk})$

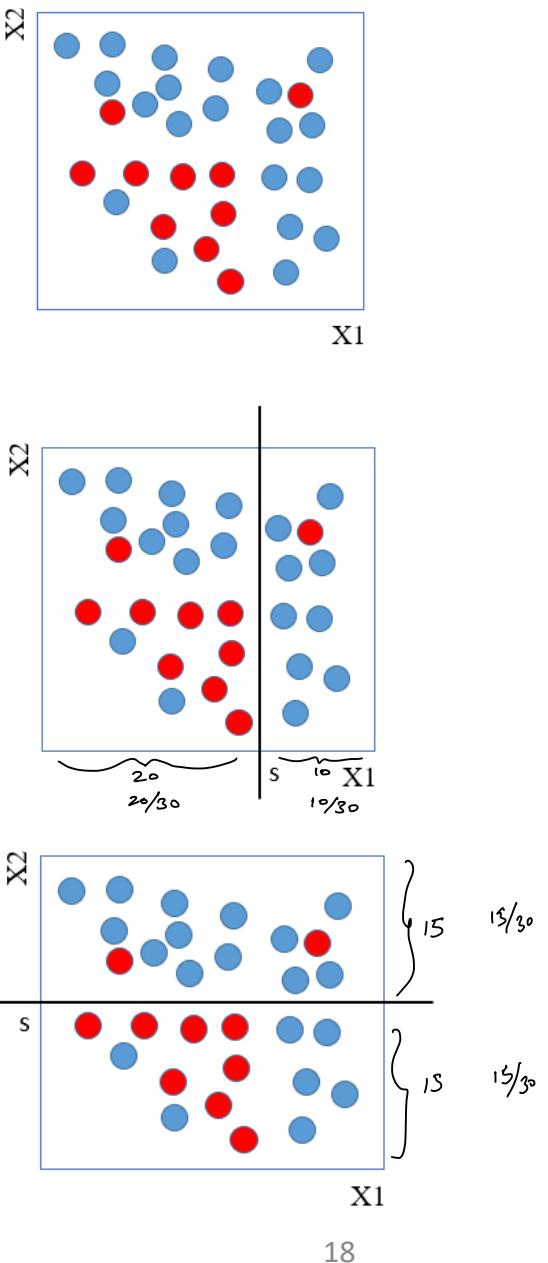
3. Cross entropy:
 \downarrow
When we use Natural log (\ln) $\longrightarrow D = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$

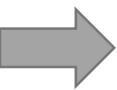


- \hat{p}_{mk} represents the proportion of training observations in the m^{th} region from the k^{th} class.
- Classification error rate is **not sufficiently sensitive to node purity** and in practice either **Gini** or **Cross entropy** is preferred.

Decision Tree Metrics (Simple Example)

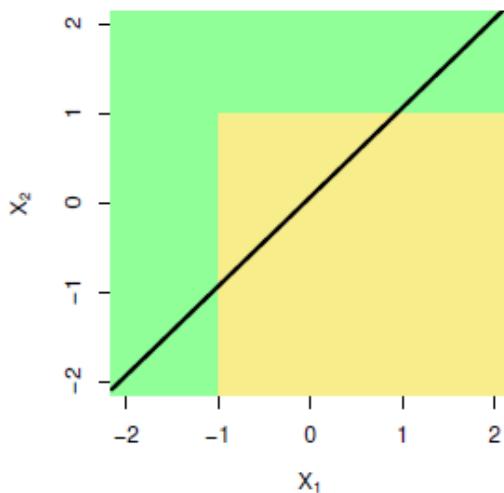
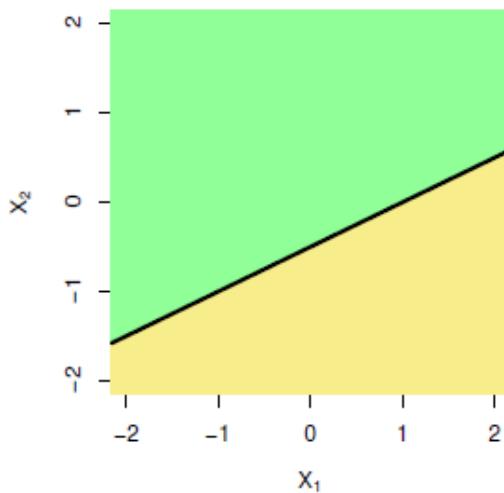
Node	Gini $1 - \sum_j p_j^2$	Cross entropy $-\sum_j p_j \log(p_i)$	Error rate $1 - \max(p_i)$
Entire training data before split	$\left\{1 - \left(\left(\frac{10}{30}\right)^2 + \left(\frac{20}{30}\right)^2\right)\right\} = 0.44$	$-\left\{\left(\frac{10}{30} \log \frac{10}{30} + \frac{20}{30} \log \frac{20}{30}\right)\right\} = 0.64$	$1 - \max\left\{\frac{10}{30}, \frac{20}{30}\right\} = 1 - \frac{20}{30} = 0.333$
Root node: $X1 > s$	$\frac{20}{30} * \left\{1 - \left(\left(\frac{9}{20}\right)^2 + \left(\frac{11}{20}\right)^2\right)\right\} + \frac{10}{30} * \left\{1 - \left(\left(\frac{1}{10}\right)^2 + \left(\frac{9}{10}\right)^2\right)\right\} = \frac{20}{30} * 0.495 + \frac{10}{30} * 0.18 = 0.39$	$-\frac{20}{30} \left\{\left(\frac{9}{20} \log_2 \frac{9}{20} + \frac{11}{20} \log_2 \frac{11}{20}\right)\right\} + \frac{10}{30} \left\{\left(\frac{1}{10} \log_2 \frac{1}{10} + \frac{9}{10} \log_2 \frac{9}{10}\right)\right\} = \frac{20}{30} * 0.69 + \frac{10}{30} * 0.325 = 0.57$	$\frac{20}{30} * \frac{9}{20} + \frac{10}{30} * \frac{1}{10} = 0.33$
Root node: $X2 > s$	$\frac{15}{30} * \left\{1 - \left(\left(\frac{2}{15}\right)^2 + \left(\frac{13}{15}\right)^2\right)\right\} + \frac{15}{30} * \left\{1 - \left(\left(\frac{8}{15}\right)^2 + \left(\frac{7}{15}\right)^2\right)\right\} = \frac{15}{30} * 0.231 + \frac{15}{30} * 0.497 = 0.37$	$-\frac{15}{30} \left\{\left(\frac{2}{15} \log_2 \frac{2}{15} + \frac{13}{15} \log_2 \frac{13}{15}\right)\right\} + \frac{15}{30} \left\{\left(\frac{8}{15} \log_2 \frac{8}{15} + \frac{7}{15} \log_2 \frac{7}{15}\right)\right\} = \frac{15}{30} * 0.39 + \frac{15}{30} * 0.69 = 0.54$	$\frac{15}{30} * \frac{2}{15} + \frac{15}{30} * \frac{7}{15} = 0.3$



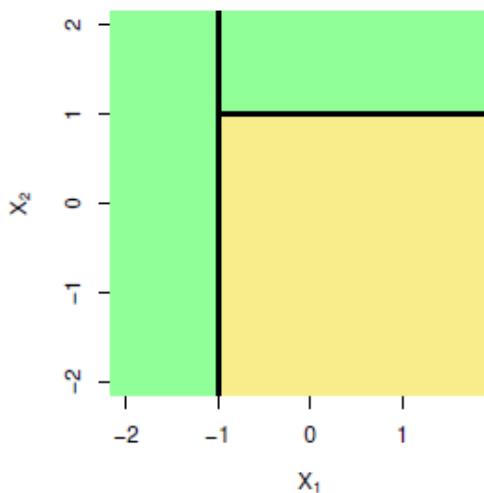
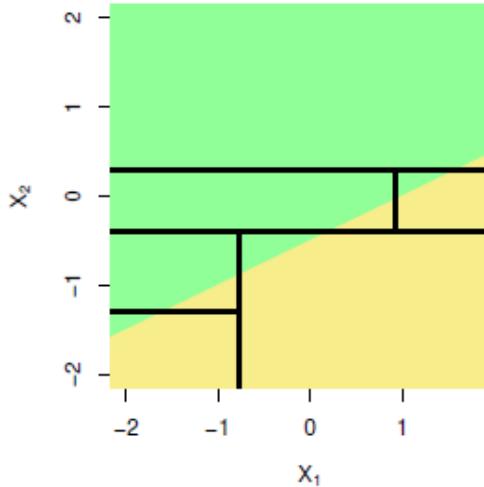


Trees Versus Linear Models

Left column: linear model; Right column: tree-based model



Top Row: True linear boundary
Bottom row: true non-linear boundary.

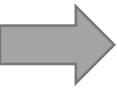


- If the True boundary is linear do logistic Regression but if it is Non-linear do Decision Tree.

Part III

Pruning a tree

Tunning hyper parameters



Pruning a tree

- Decision tree may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. (why?) *Variance high.*
- A **smaller tree** with fewer splits may lead to **lower variance** and better interpretation at the cost of **a little bias**.
- This strategy may result in smaller trees, but is too short-sighted:
“a seemingly worthless split early on in the tree might be followed by a very good split, a split that leads to a large reduction in RSS/impurity index later on”
- A better strategy is to **grow a very large tree T_0** , and then prune it back in order to obtain a subtree.
- **Cost complexity pruning** is used to do this.

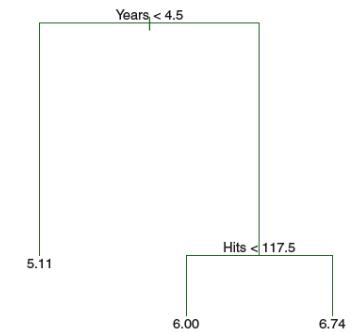
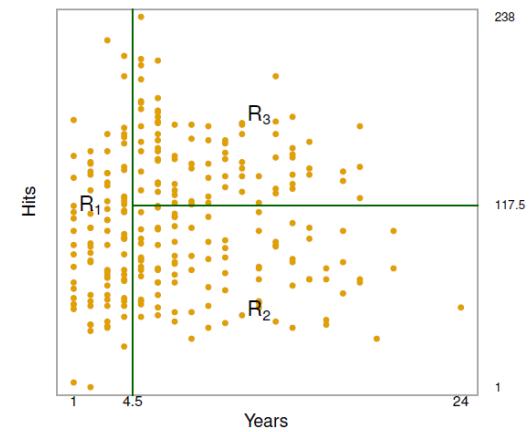


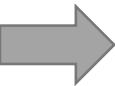
Cost complexity pruning (weakest link pruning)

- Consider a sequence of trees indexed by a nonnegative tuning parameter α .
- For each value of α there corresponds a subtree $T \subset T_0$ such that the following objective function is minimized.

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

- $|T|$ indicates the number of terminal nodes of the tree T
- R_m is the rectangle corresponding to m^{th} terminal node and
- \hat{y}_{R_m} is the mean of the training observations in R_m
- α controls the bias variance trade off and is determined by cross validation.
- Lastly, we return to full data set and obtain the subtree corresponding to α





Building a Regression Tree algorithm

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
 2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α .
 3. Use K-fold cross-validation to choose α . That is, divide the training observations into K folds. For each $k = 1, \dots, K$:
 - (a) Repeat Steps 1 and 2 on all but the k th fold of the training data.
 - (b) Evaluate the mean squared prediction error on the data in the left-out k th fold, as a function of α .Average the results for each value of α , and pick α to minimize the average error.
 4. Return the subtree from Step 2 that corresponds to the chosen value of α .
-

Source: An introduction to Statistical Learning

We could literally start from alpha = 0 and increase it gradually (0.01, 0.02, ...). BUT this is not the most efficient way of doing that (because for regression analysis, where target variable is not scaled, the alpha can be a very large number in scale of thousands). A better way is to find the best alpha for each subtree programmatically and this is exactly what "cost_complexity_pruning_path()" function does in sklearn.

* The cost_complexity_pruning_path function generates a sequence of candidate values for alpha by exploring the pruning path from the largest subtree to the smallest subtree. The function then computes the effective value of alpha for each subtree as the difference in cost complexity between the parent subtree and the current subtree, divided by the number of leaves removed in the pruning step. The effective value of alpha for the largest subtree is set to zero.

* The function returns an array of the effective values of alpha for each subtree in the pruning path, which can be used to select the optimal value of alpha for pruning the decision tree.

The cost-complexity pruning process is as follows:

1. Grow the full tree: Build an initial decision tree using the training data without any stopping criteria.
2. Calculate the cost complexity for the entire tree: For each subtree, compute the impurity (e.g., total Gini impurity or mean squared error, depending on the task). Compute the number of leaves in each subtree. Calculate the cost complexity for each subtree using the formula:
$$\text{cost_complexity} = \text{subtree_impurity} + \alpha * \text{num_leaves}$$

Here, alpha is a user-defined parameter that controls the trade-off between fit and complexity.

3. Prune the tree: Starting from the bottom of the tree and working upwards, compare the cost complexity of a subtree with the cost complexity of its parent node after pruning (i.e., treating the parent as a leaf node). If pruning the subtree results in a lower cost complexity, replace the subtree with its parent node (effectively pruning the tree).
4. Repeat the pruning process: Continue pruning the tree until no further reductions in cost complexity can be achieved.
5. Find the best-pruned tree: Evaluate the pruned trees on a validation set or using cross-validation to find the tree with the lowest prediction error. This tree represents the best balance between fitting the training data and maintaining a simple structure, reducing the risk of overfitting.

Keep in mind that alpha is a user-defined parameter, and you don't need to compute a separate alpha for each subtree. The same alpha value is used for all subtrees during the pruning process. You can experiment with different alpha values to find the best balance between model complexity and prediction accuracy.
Do not confuse alphas with effective alphas from the cost_complexity_pruning_path() method.

The cost_complexity_pruning_path() function in scikit-learn computes the effective alphas and the corresponding impurities at each step of the pruning process. The effective alphas are not random; they are the minimum values of alpha required to prune a subtree at each step.

Here's how the function works:

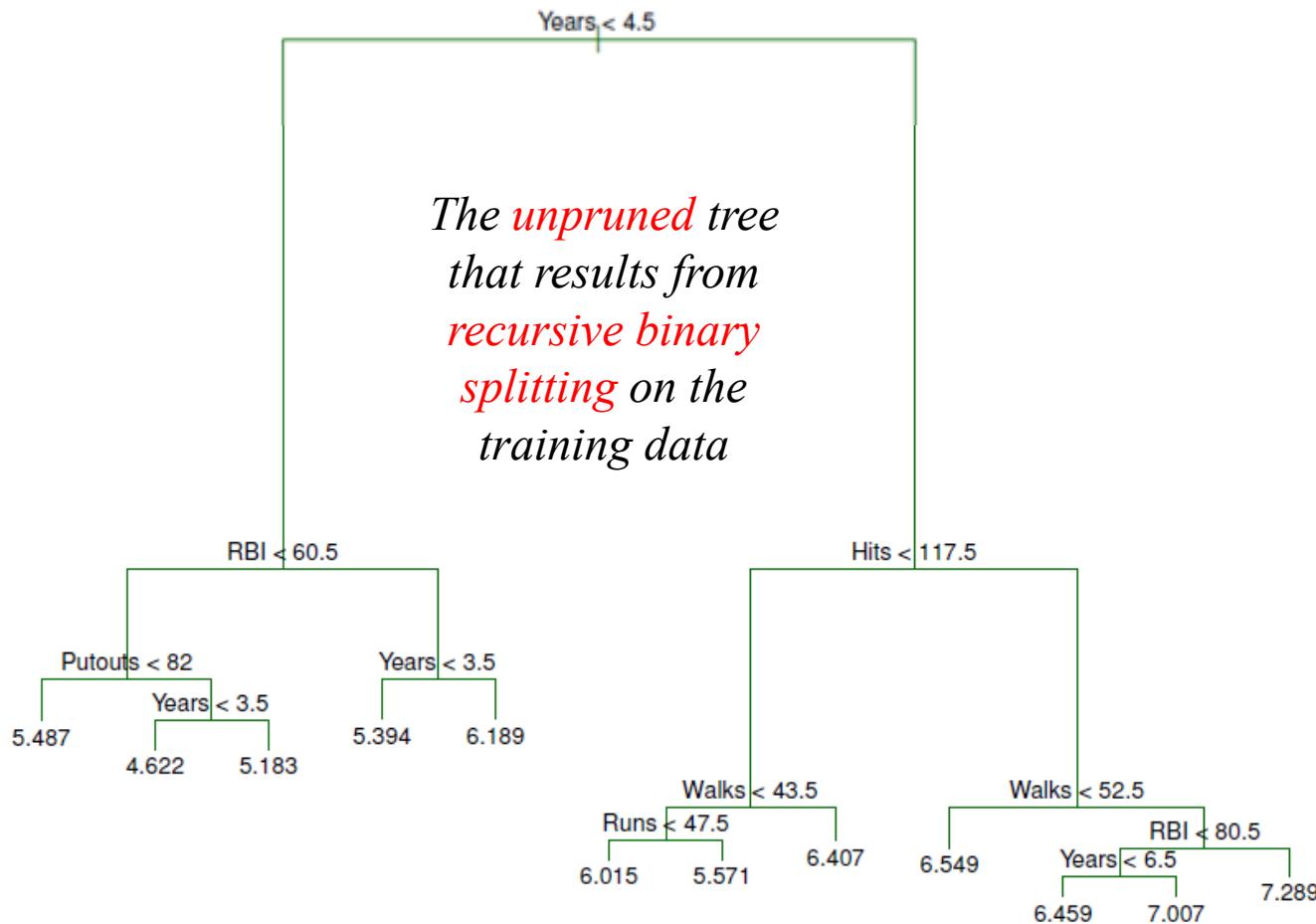
1. It starts with the full decision tree and computes the impurities for each node.
2. For each node, it calculates the effective alpha, which is the change in impurity divided by the change in the number of leaves when that node is pruned:

```
effective_alpha = (node_impurity - sum(child_impurities)) / (num_leaves_after_pruning - num_leaves_before_pruning)
```

3. The function selects the node with the smallest effective alpha, prunes it, and updates the tree.
4. The process is repeated until the tree is fully pruned (i.e., only the root node remains).

The resulting alphas and impurities are returned in two arrays, with each element corresponding to a step in the pruning process. The alphas array represents the sequence of effective alphas used to prune the tree, while the impurities array represents the total impurity of the tree after each pruning step. These effective alphas can be used as candidate values for the cp_alpha parameter when fitting a decision tree with cost complexity pruning. By choosing different values of cp_alpha, you can control the trade-off between model complexity and prediction accuracy, selecting the best-pruned tree based on cross-validation or a validation set.

→ Salary example continued



→ Finding the optimal α or T

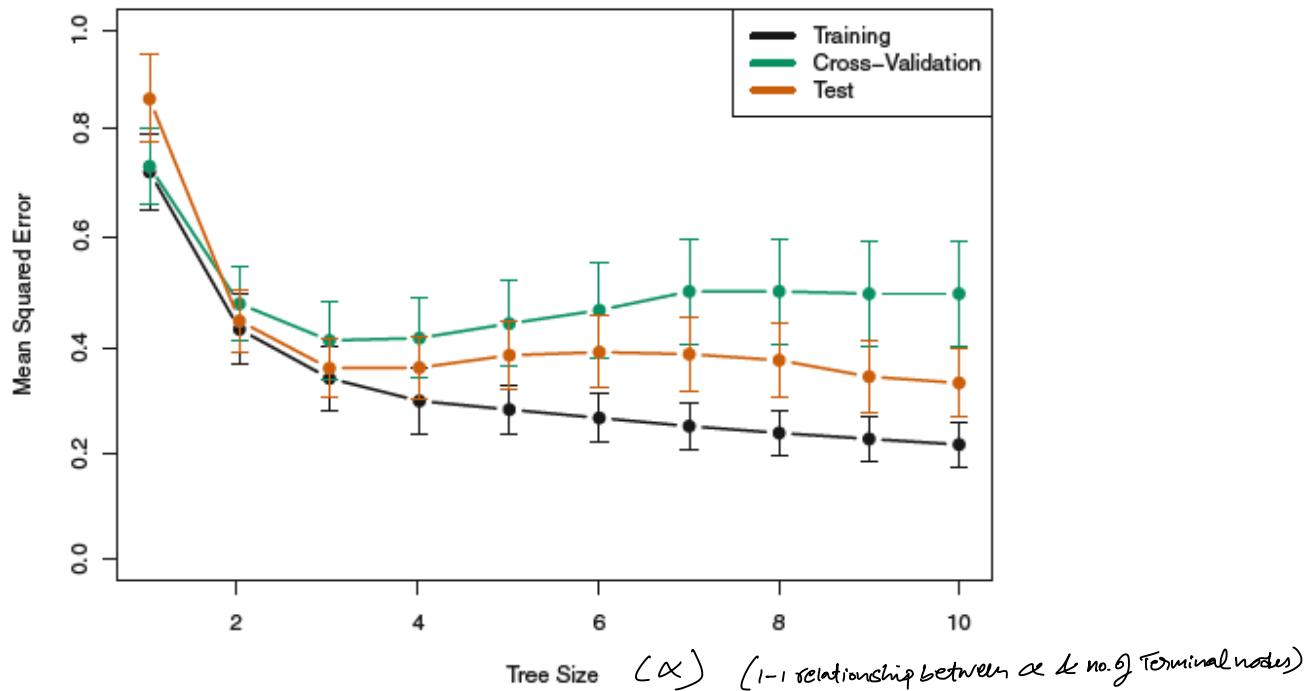
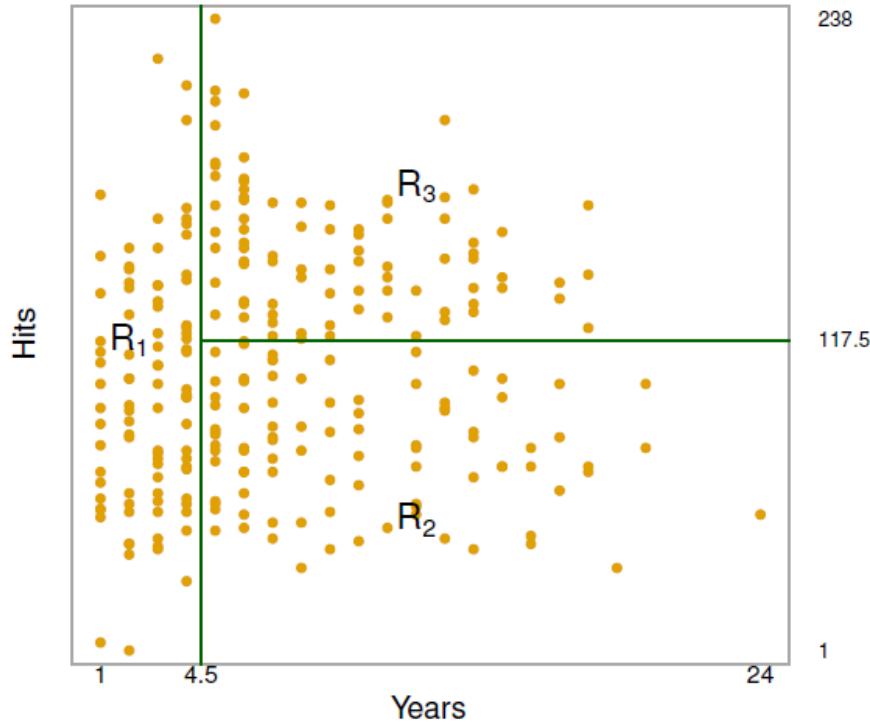
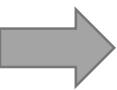


FIGURE 8.5. Regression tree analysis for the **Hitters** data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.

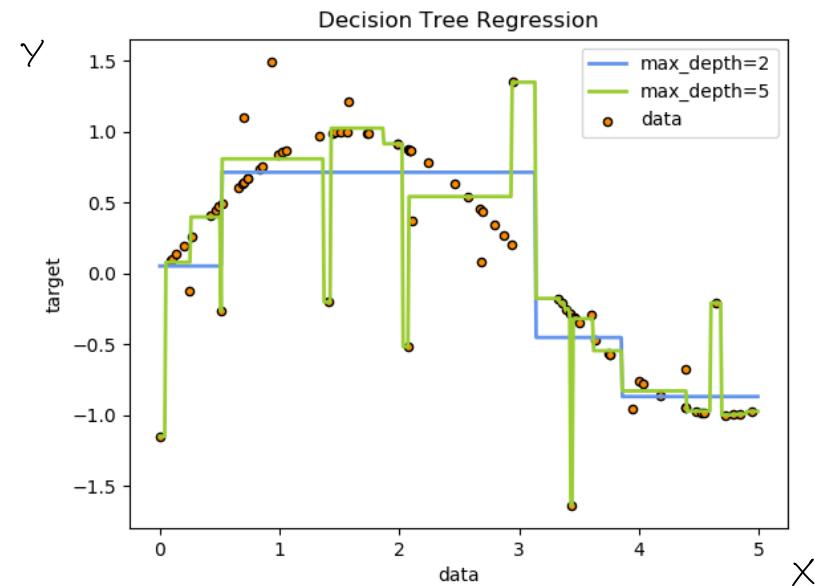
→ The optimal (pruned) tree





Other hyperparameters

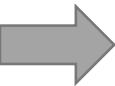
- ✓ To avoid overfitting, **regularization parameters** can be added to the model such as:
 - Maximum depth of the tree
 - Minimum population at a node
 - Maximum number of decision nodes
 - Minimum impurity decrease (info gain)
 - Alpha (complexity parameter)
- ✓ Other hyperparameters are:
 - Criterion: gini, entropy
 - Splitter: best, random
 - Class weight: balanced, none



Part IV

Pros and Cons

Applications in finance



DTs' Pros and Cons



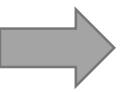
Pros:

- Easy to interpret and visualize
- Can easily handle categorical data without the need to create dummy variables
- Can easily capture Non-linear patterns
- Can handle data in its raw form (no preprocessing needed). Why? *No Distance Metric.*
- Has no assumptions about distribution because of the non-parametric nature of the algorithm

Cons:

- Poor level of predictive accuracy.
- Sensitive to noisy data. It can overfit noisy data. Small variations in data can result in the different decision tree*.

*This can be reduced by bagging and boosting algorithms.



DTs' Applications in finance

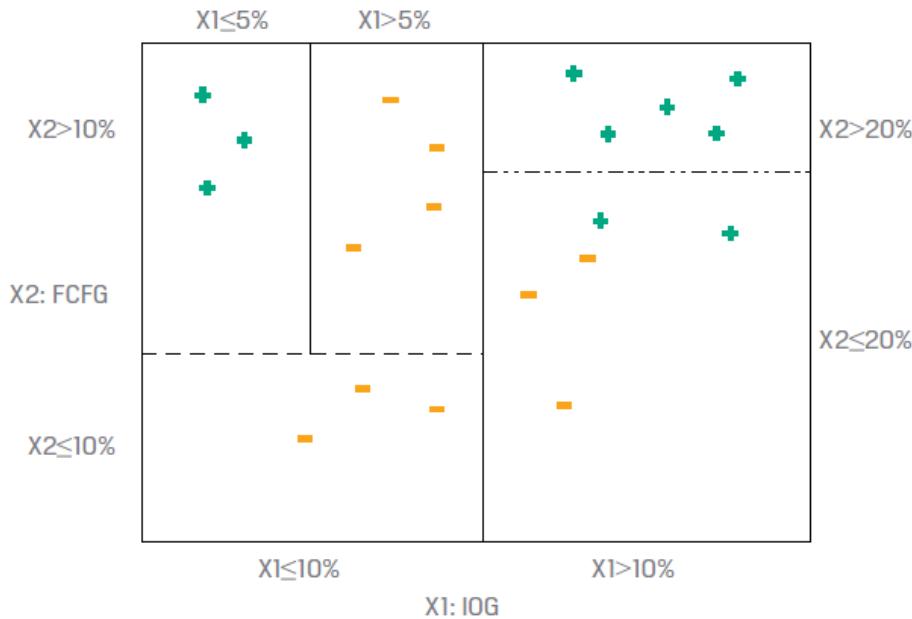
- Enhancing detection of fraud in financial statements,
- Generating consistent decision processes in equity and fixed-income selection
- Simplifying communication of investment strategies to clients.
- Portfolio allocation problems.



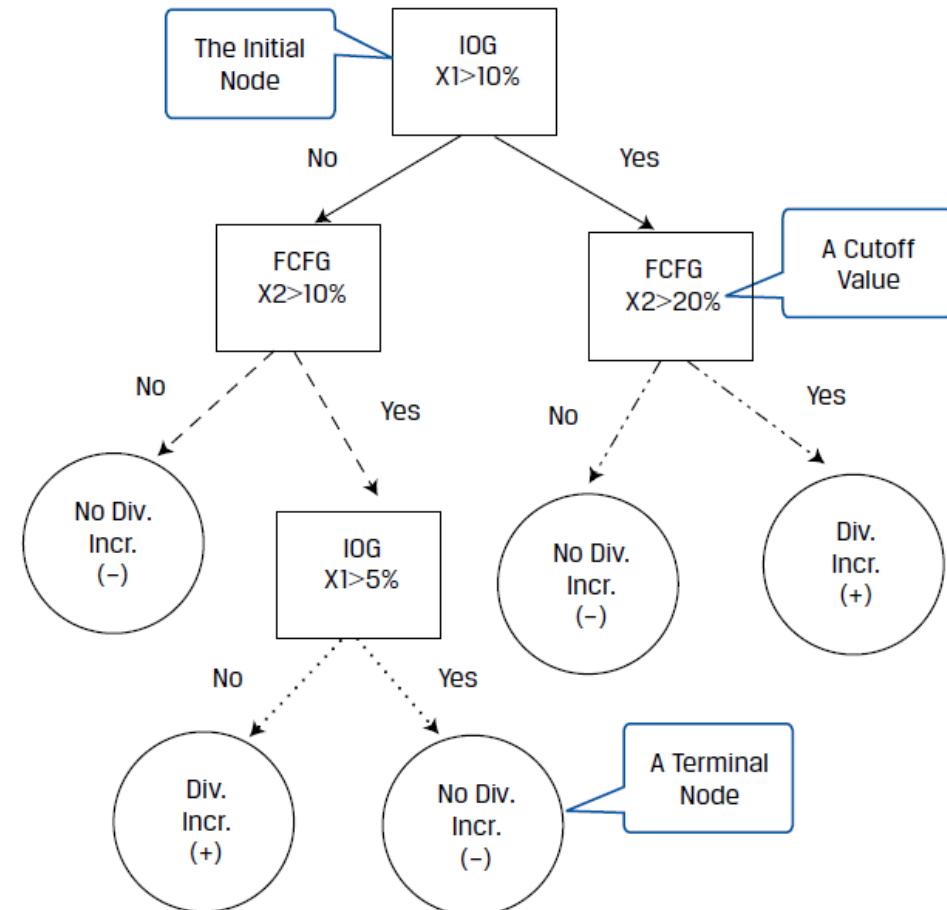
→ DT example in finance

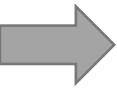
X1: Investment Opportunities Growth (IOG)
X2: Free Cash Flow Growth (FCFG)

Features



Source: CFA PROGRAM. Level II . Reading 7



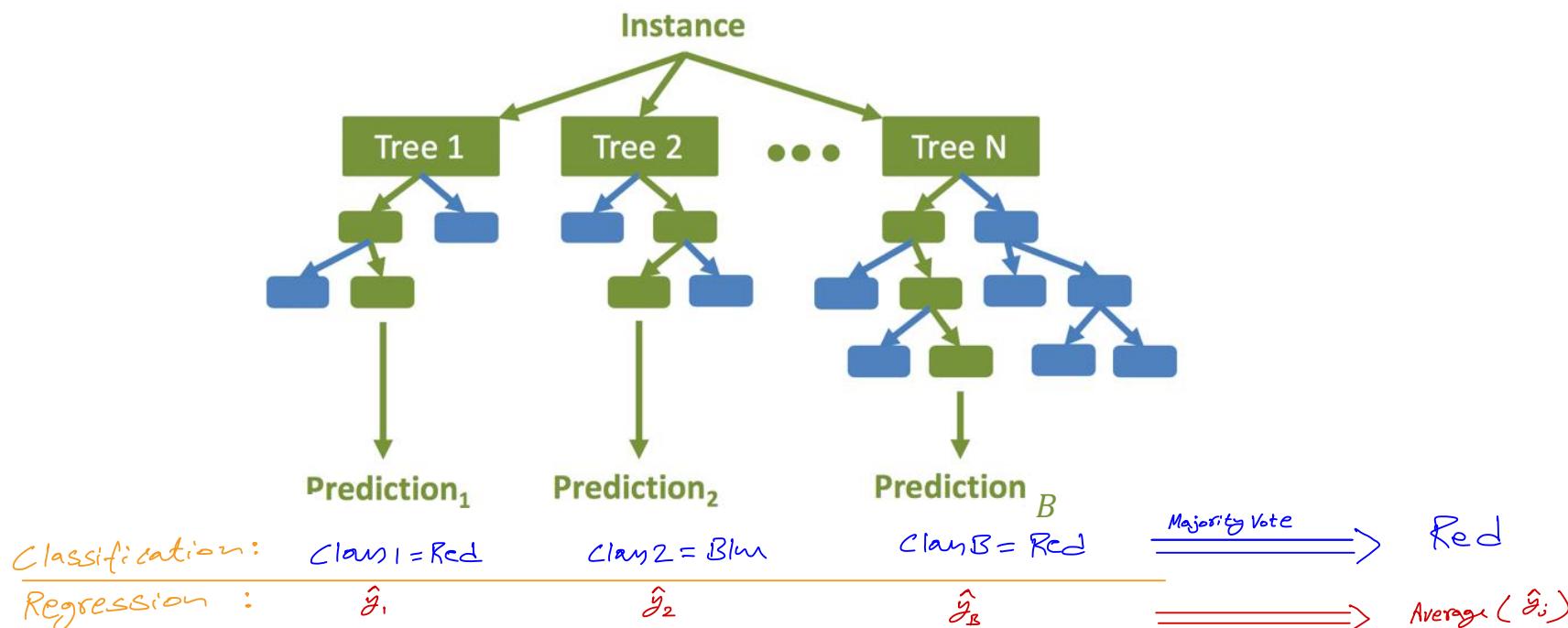


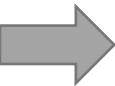
Students' questions

1. Should we be concerned about the computational expense of 'MSE' when working with large trees of continuous data?
2. Having hard time understanding gini impurity and what conceptually the value means.
3. What is the Gini impurity role in classification?
4. A little confused on how to pick which method should be used for a given data set.
5. Would like to go over when the algorithm decides to create a leaf node vs continuing the decision tree.

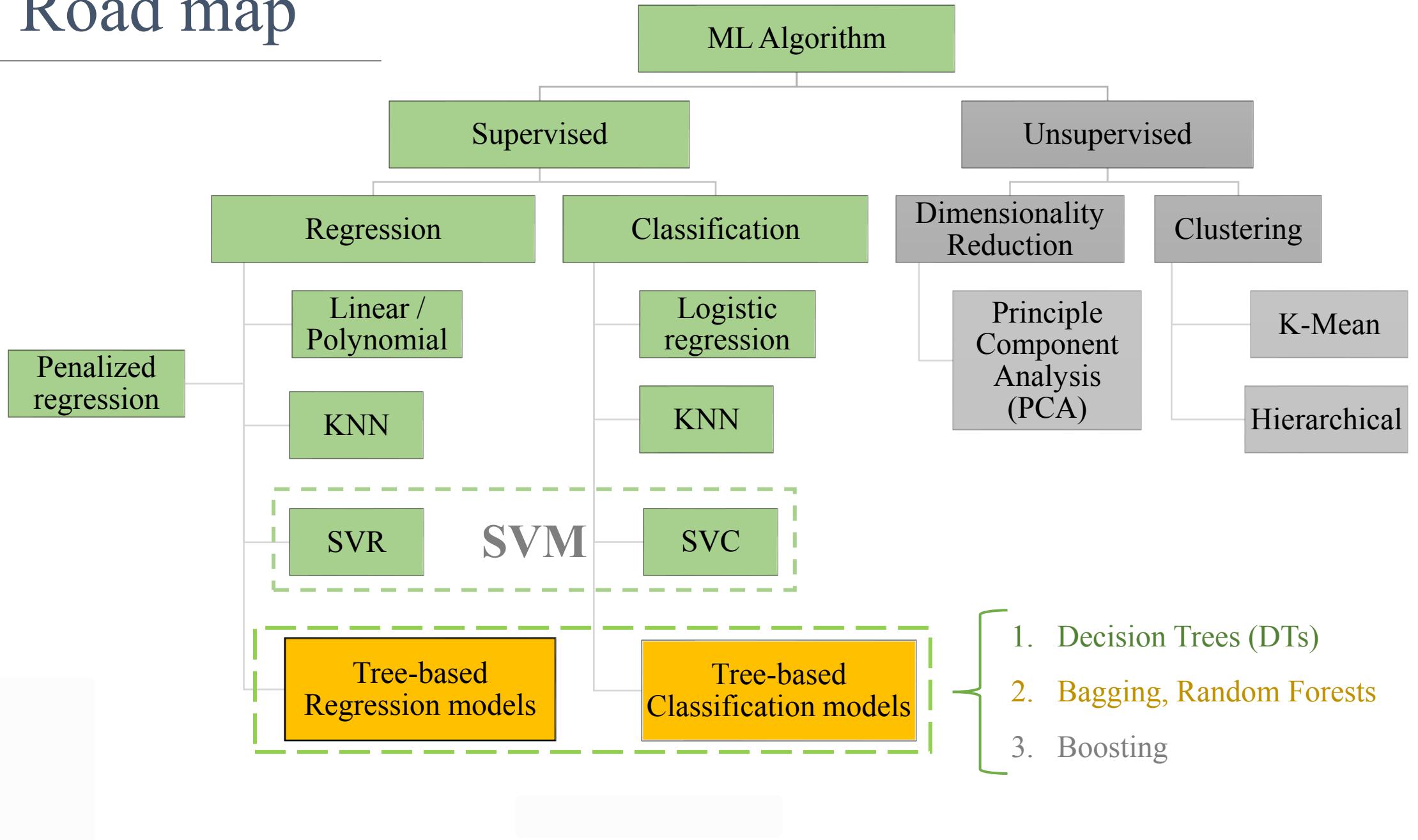
Class -20

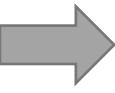
Bagging and Random Forests





Road map





Topics

Part I

1. Why not a simple tree?
2. Ensemble Learning

Part III

1. Hyperparameters
2. Feature importance

Part II

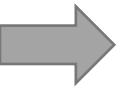
1. Bootstrap Aggregation (Bagging)
2. Random Forests

Part IV

1. Pros and Cons
2. Applications in Finance

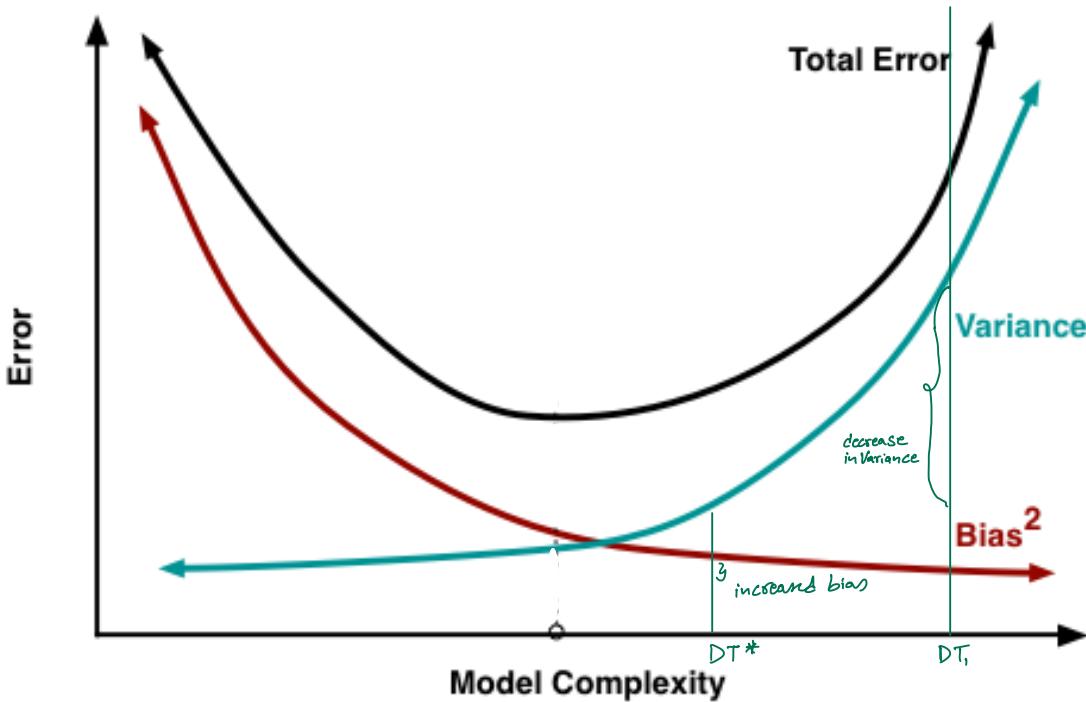
Part I

Motivation



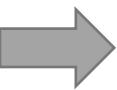
Why not a simple tree?

- Goal: Reduce the bias and variance at the same time?!



Previously, we said we start with very bushy tree with low bias & high variance (DT_i). This tree will overfit in train set & perform badly in Test set. We then solve this by pruning the bushy tree to DT^* which increases the bias by little amount but drastically decreases variance.

- Bagging focuses on reducing the Model Variance i.e. shift the Variance Curve downward. By doing it we reduce the error in Test set.
- Boosting focuses on reducing the Model Bias i.e. shift the Bias Curve downward. By doing it we reduce the error in Test set.



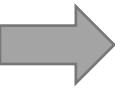
Ensemble Learning

- Why not use the predictions of a group (an ensemble) of models?
- Ensemble Learning: Combining the predictions from a collection of models
- Idea: instead of trying to learn one super-accurate model, focusing on training many low-accuracy models and then combining the predictions given by those weak models.
- Ensemble learning typically produces more accurate and more stable predictions than the best single model.
- Ensemble learning methods are based on:
 1. Aggregation of heterogeneous learners (voting classifiers / average predictions) SVM, KNN, DT etc.
 2. Aggregation of homogeneous learners (**bagging** and **boosting**)

$$\text{Var}(\bar{x}) = \frac{\sigma^2}{n} \text{ So as } n \uparrow \text{ Sample Variance} \downarrow \text{ Similarly,}$$
$$\text{Var}(\hat{f}(x)) = \text{Var}(\underbrace{\hat{f}_1 + \hat{f}_2 + \dots}_{k \text{ no. of trees}}) = \frac{\text{Var}(\hat{f}_1) + \text{Var}(\hat{f}_2) + \text{Cov}(\hat{f}_1, \hat{f}_2) + \dots}{k}$$

Unique Model (DTs) but work with different Variations of same data set.
i.e. resample from data set thousands of times.

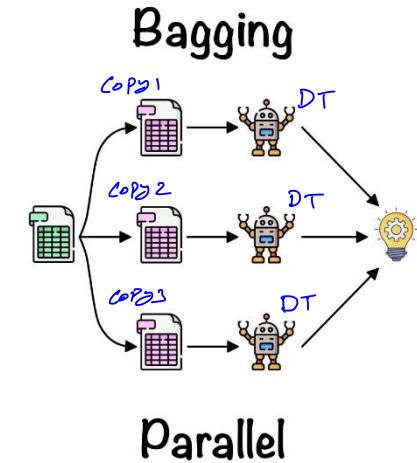
So, the Variance of Ensemble methods depends on Variance of individual methods & Covariance between them. Idea is to reduce individual Variances by a lot & then divide by k , the overall Var. of ensemble method is smaller than that of a Single Tree.



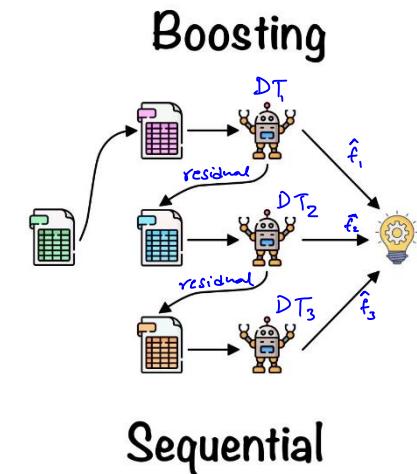
Bagging vs Boosting

→ Bootstrap Aggregation

- **Bagging** consists of creating many “**copies**” of the training data (each copy is slightly different from another) and then apply the weak learner to each copy to obtain multiple weak models and then combine them.



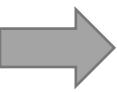
- **Boosting** consists of using the “**original**” training data and **iteratively** creating multiple models by using a weak learner. Each new model would be different from the previous ones in the sense that the weak learner, by building each new model tries to “**fix**” the **errors** which previous models make.



Part II

Bagging and Random Forest

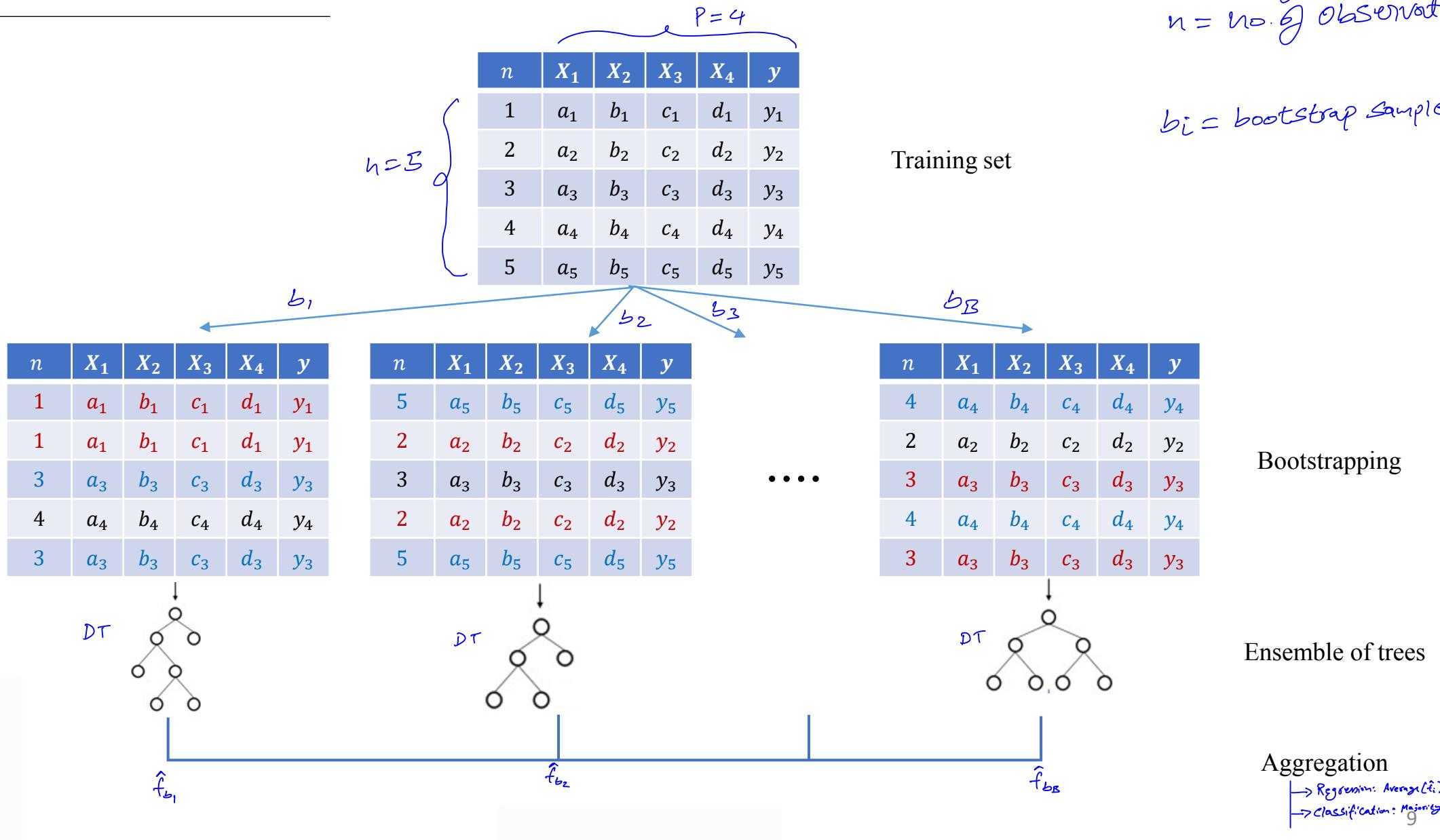
- Bootstrapping → Repeated Random Sampling from data with replacement.

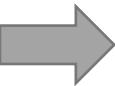


Bootstrap Aggregation (Bagging)

$p = \text{no. of features}$
 $n = \text{no. of observations}$

$b_i = \text{bootstrap sample}$





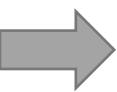
Bagging procedure

$$\text{Var}(\hat{f}_{\text{ensemble}}) = \frac{\text{Var}(\hat{f}_1) + \text{Var}(\hat{f}_2) + \text{Cov}(\hat{f}_1, \hat{f}_2) + \dots}{B}$$

- Bootstrap aggregating (or bagging) is a general-purpose procedure for reducing the variance of a learning method.
- Given a training set, first we create B random samples (with replacement) of the training set.
- For each sample b , build a decision tree model f_b (*weak learner*)
- After training, we have B decision trees. The predictions for a new test observation x is obtained as the **average** of B predictions (for regression) or a **majority vote** (for classification).

$$y \leftarrow \hat{f}(x) \stackrel{\text{def}}{=} \frac{1}{B} \sum_{b=1}^B f_b(x) \quad \text{or} \quad \text{most frequent } f_b(X)$$

- Bagging is a very useful technique because it helps to improve the **stability** of predictions and protects against **overfitting** the model. (why?) Reducing Model Variance \Rightarrow Reduction in Overfitting

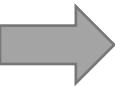


Random Forests

- Random forests provide an improvement over The “vanilla” bagging algorithm by using a small trick that decorrelates the trees. (Reducing $\text{Cov}(\hat{t}_i, \hat{t}_j)$ term reduces $\text{Var}(\hat{f}_{ensemble})$).
- RF uses a modified tree learning algorithm that at each split, inspects a random subset of the features (instead of inspecting the entire feature space!)
- The reason for doing this is to avoid the correlation of the trees in our forest:

“Suppose there is one very strong feature in the data set. When using bagged trees, most of the trees will use that feature as the top split, resulting in an ensemble of similar trees that are highly correlated”



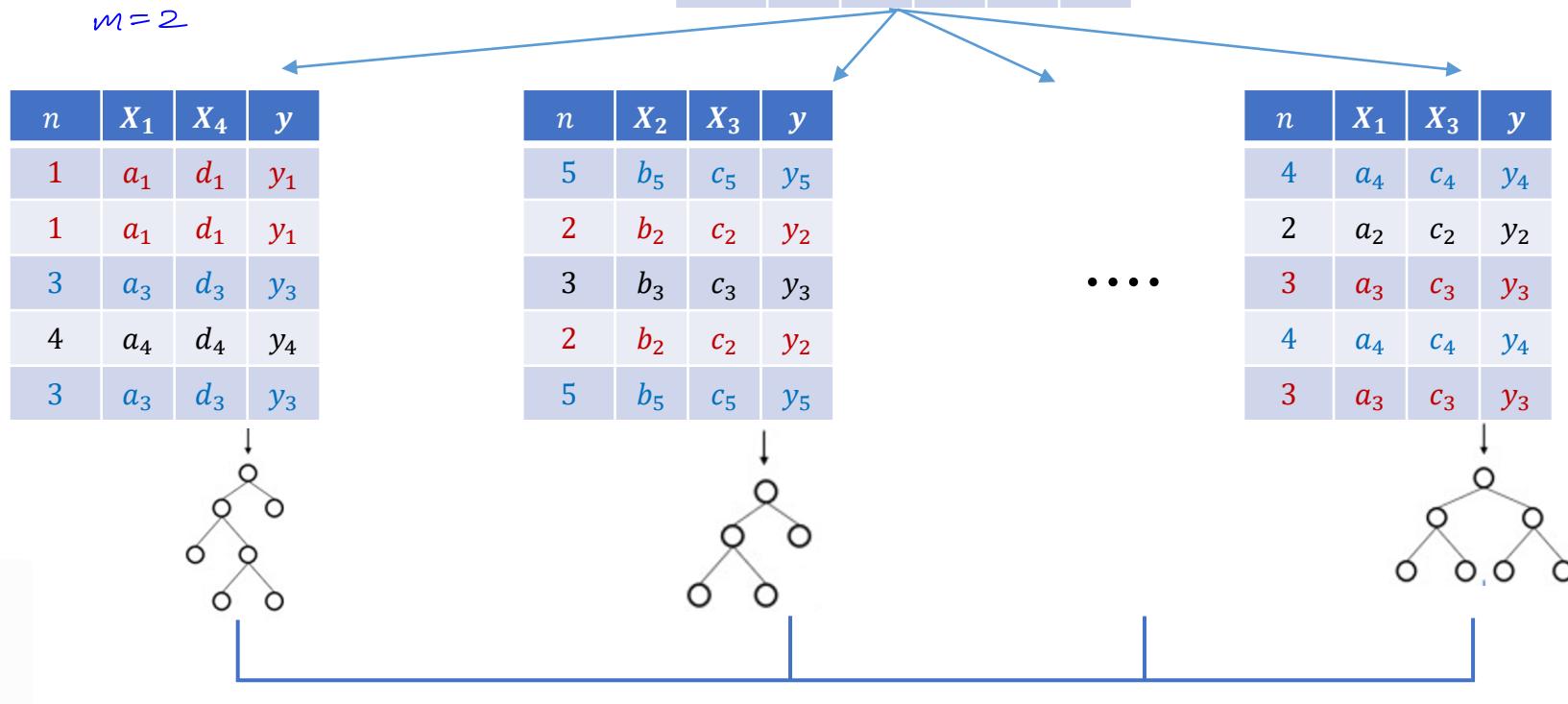


Random Forests

$m = \text{Subset of features}$



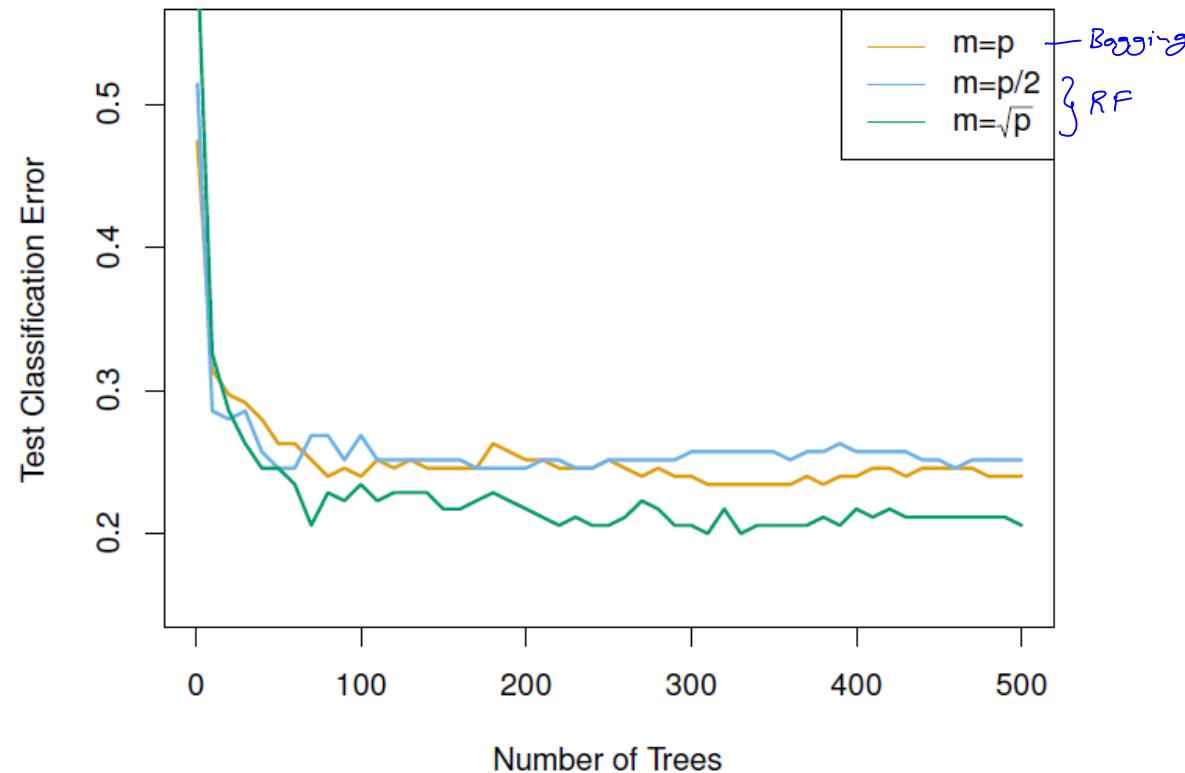
Random subset of features
happens at each split!



If $m=p$, RF = Bagging

Random Forests vs Bagging

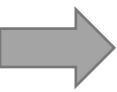
- A random subset \mathbf{m} , can be any subset of \mathbf{p} features like $\frac{p}{2}$ or \sqrt{p} or $\log(p)$ etc.



Part III

Hyper parameters

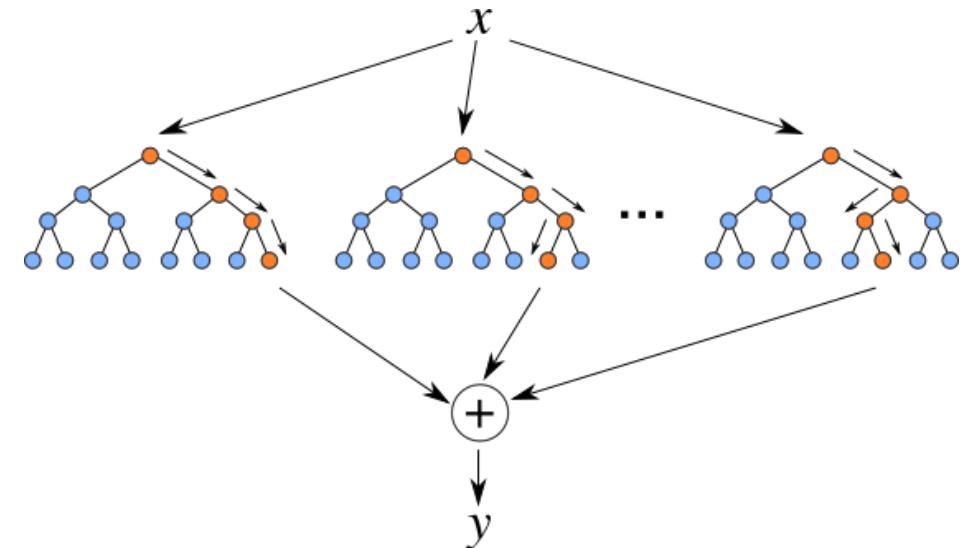
Feature importance



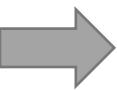
Hyperparameters

- The most important hyper parameters are:

- ✓ The number of subset features (**m**)
- ✓ The number of trees to use (**B**) *(called n_estimators in python)*
- ✓ The minimum **size** of each node (or leaf)
- ✓ The maximum **number** of leaf nodes
- ✓ The maximum **depth** of each tree
- ✓ Criterion: gini, entropy



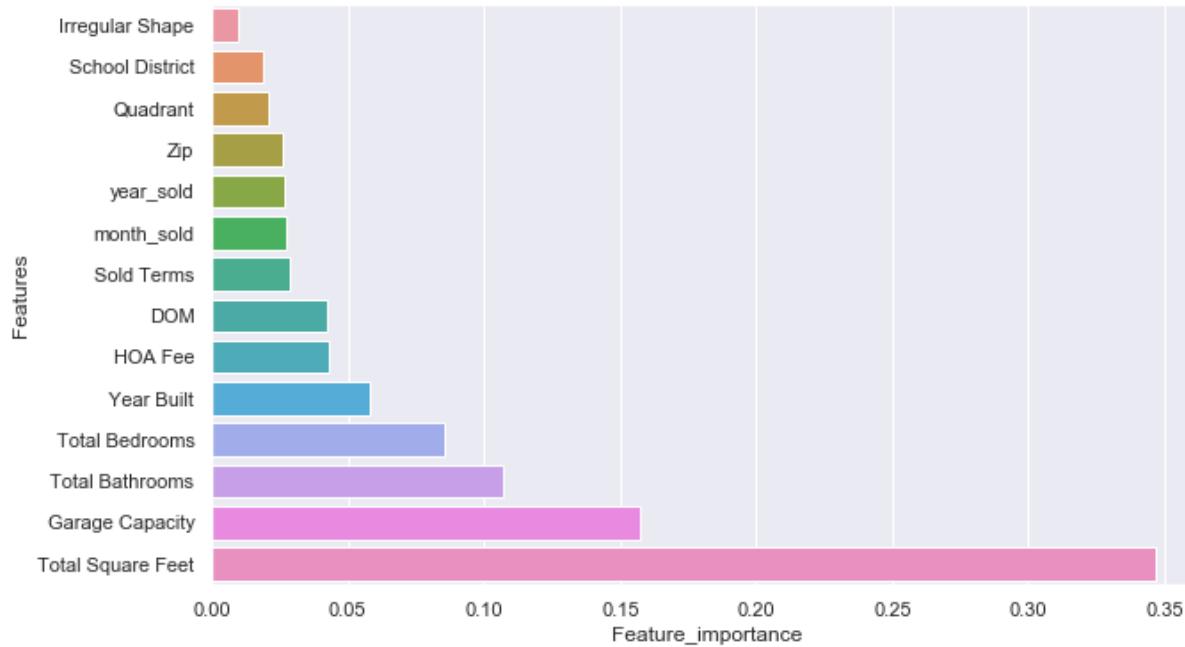
- Grid search CV could be used to tune a combination of hyper parameters.



Feature importance

RSS
gini/entropy

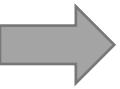
- Feature importance refers to techniques that assign a **score** to input features based on how **useful** they are at predicting a target variable.
- For RF , the total amount that the **RSS is decreased** / **Gini index or entropy is decreased** due to splits over a given predictor, averaged over all B trees. A **large** value indicates an **important** predictor.



Part IV

Pros and Cons

Applications in finance



Random Forests' Pros and Cons

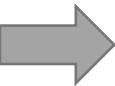
Pros:

- One of the most widely used ensemble learning algorithms. Because on top of all the advantages of a Decision Tree model (handling categorical data, learning nonlinear patterns, no preprocessing needed, nonparametric), it can:
- Avoid overfitting by reducing the model variance.
- Parallelizable! *(Split large data & apply RF to each)*
- Great with high dimensionality
- Quick training and prediction speed (why?) *Works with Subset of features hence reducing dimensionality.*



Cons:

- Lacks the ease of interpretability of individual trees! Relatively **black box** type of algorithm.
- Can overfit without tuning the hyperparameters.

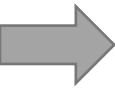


Random Forests' Applications in finance

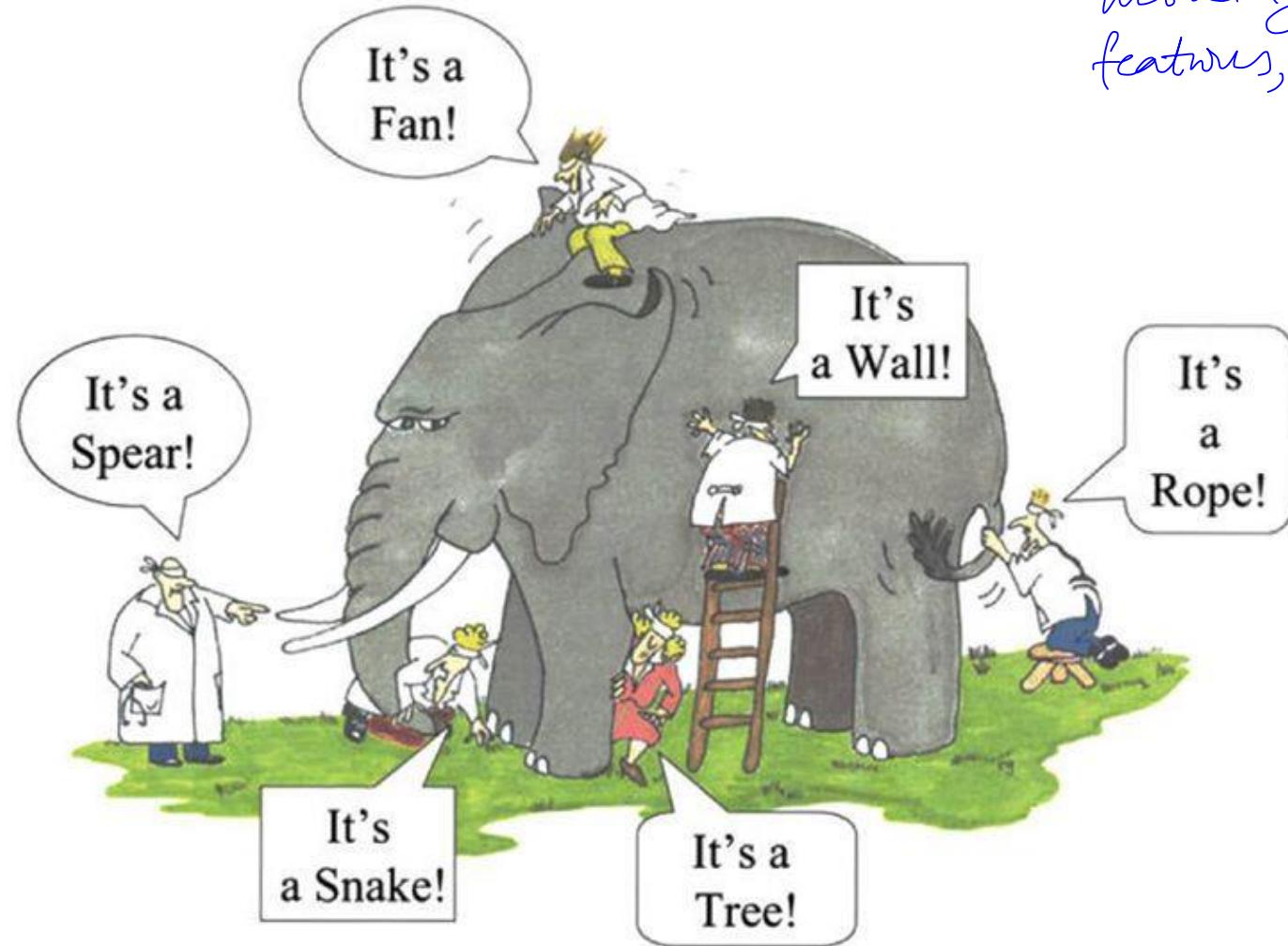
- Factor based investment strategies for **asset allocation** or **investment selection**
- Predicting whether an IPO will be **successful** based on some ranked factors including:
 - Percent oversubscribed,
 - First trading day close price
 - First trading day volume
 - ...
- Fundamental factor modeling!
 - P/E, EV/EBITDA, P/S, P/B, ...



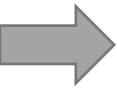
[For Time series data don't blindly use RF as there are subtle complexities, like Can't do simple bootstrap have to do sequential/stationary bootstrap, Can't do Simple Cross Validation (need to use timeseries Cross Validation).]



Question of the day: Wisdom of Crowds



Working with subset of features, so this is R.F.

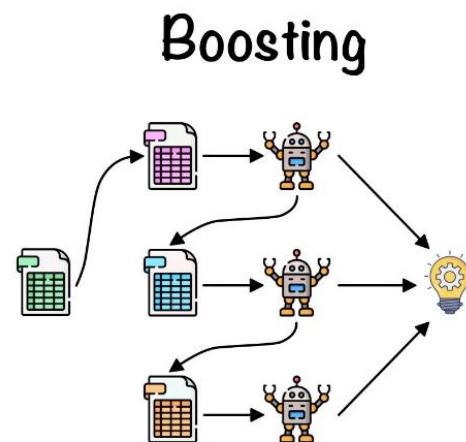


Students' questions

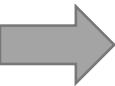
1. I've heard the term bootstrapping in an econometric context before. Are the two concepts the same?
2. Are random forest trees allowed to be deeper with little cost to over-fitting when compared to decision trees?
3. I didn't really get what the disadvantages of random forest. I also don't really understand what bootstrapping does and how it works.
4. I don't know when you would use decision trees over random forest.
5. I don't get what's meant by a 'weaker learner' that's being applied to the copies of the training data.
6. When making a random forest, does the code make ALL possible decision trees with the data? If not, how do we decide which, and if yes, would that not take too long with large data sets?
7. Do we set the number of nodes for the tree or is this truly a "black box" that we just look at the output?
8. Is bootstrapping attempting to accomplish the same thing as splitting the data into test sets?

Class -22

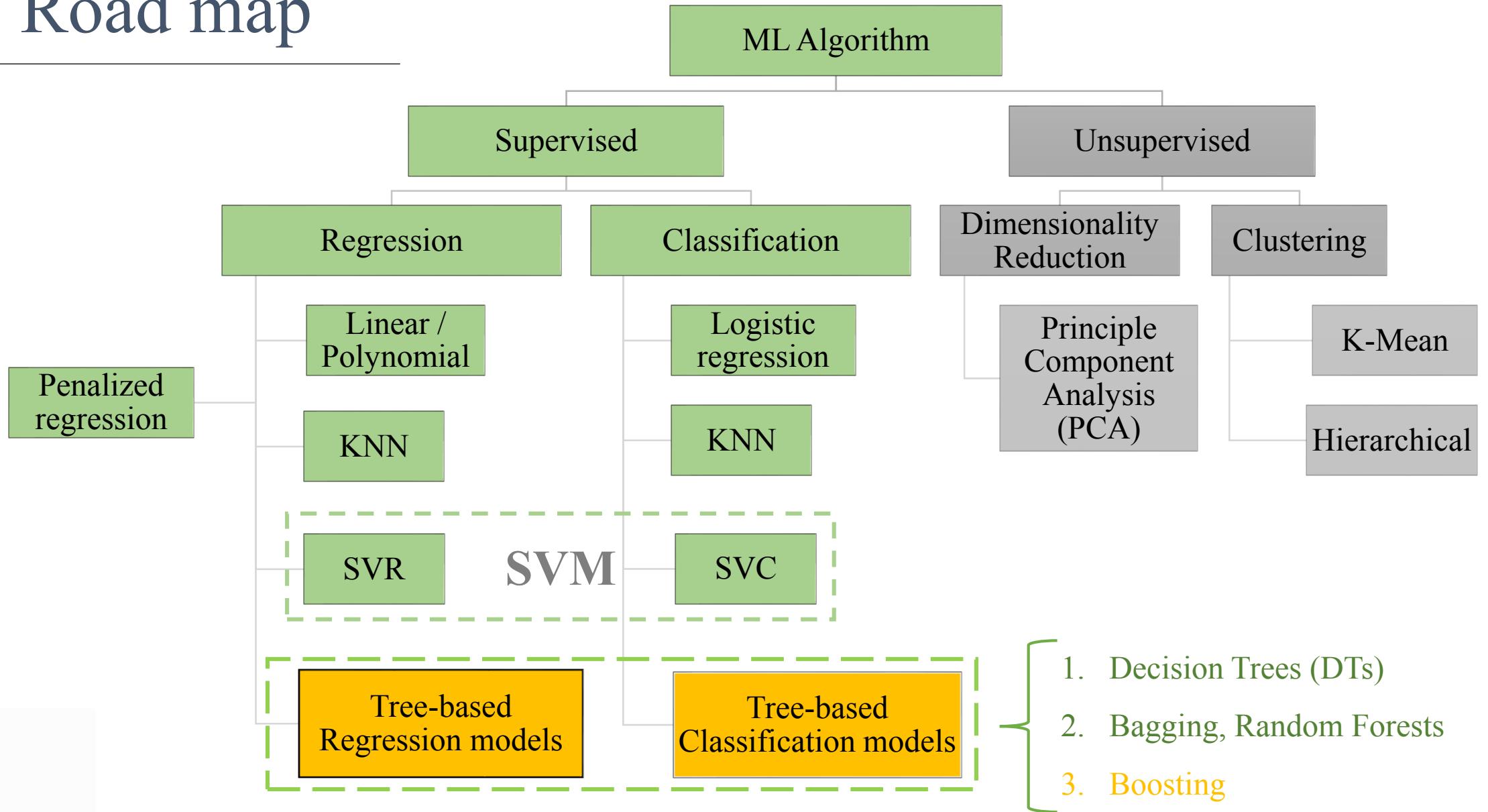
Boosting algorithms (AdaBoost, GBM, XGBoost)

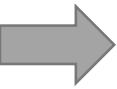


Sequential



Road map





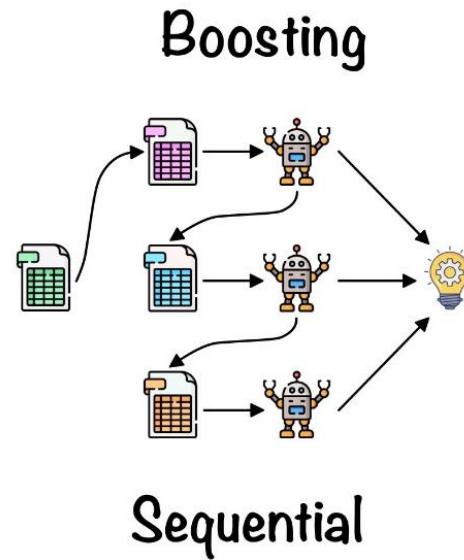
Topics

Part I

1. Bagging vs Boosting
2. AdaBoost
3. Gradient Boosting Machine (GBM)
4. XGBoost

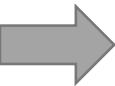
Part II

Pros and Cons



Part I

1. Bagging vs Boosting
2. AdaBoost
3. Gradient Boosting Machine (GBM)
4. XGBoost

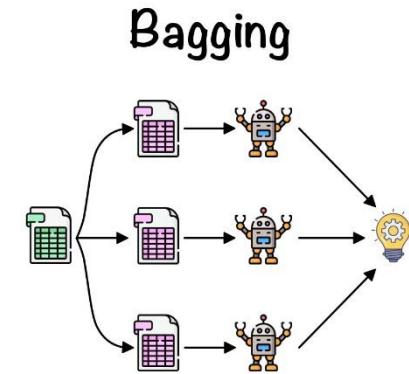


Bagging vs Boosting

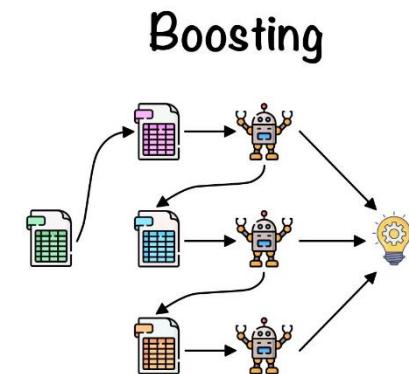
[In Bagging we do bootstrap but in Boosting We do not do bootstrap.]

- Bagging consists of creating many “copies” of the training data (each copy is slightly different from another) and then apply the weak learner to each copy to obtain multiple weak models and then combine them.
DT ↪
- In bagging, the bootstrapped trees are independent from each other.
- Boosting consists of using the “original” training data and iteratively creating multiple models by using a weak learner. Each new model would be different from the previous ones in the sense that the weak learner, by building each new model tries to “fix” the errors which previous models make.
- In boosting, each tree is grown using information from previous tree.

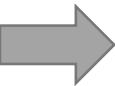
In boosting, the aggregation happens in additive manner $\hat{f}_1 + \gamma \hat{f}_2 + \gamma \hat{f}_3 + \dots$
& not averages or majority like in Bagging. γ is a shrinkage parameter so the contribution of \hat{f}_2 is not as much as \hat{f}_1 .



Parallel



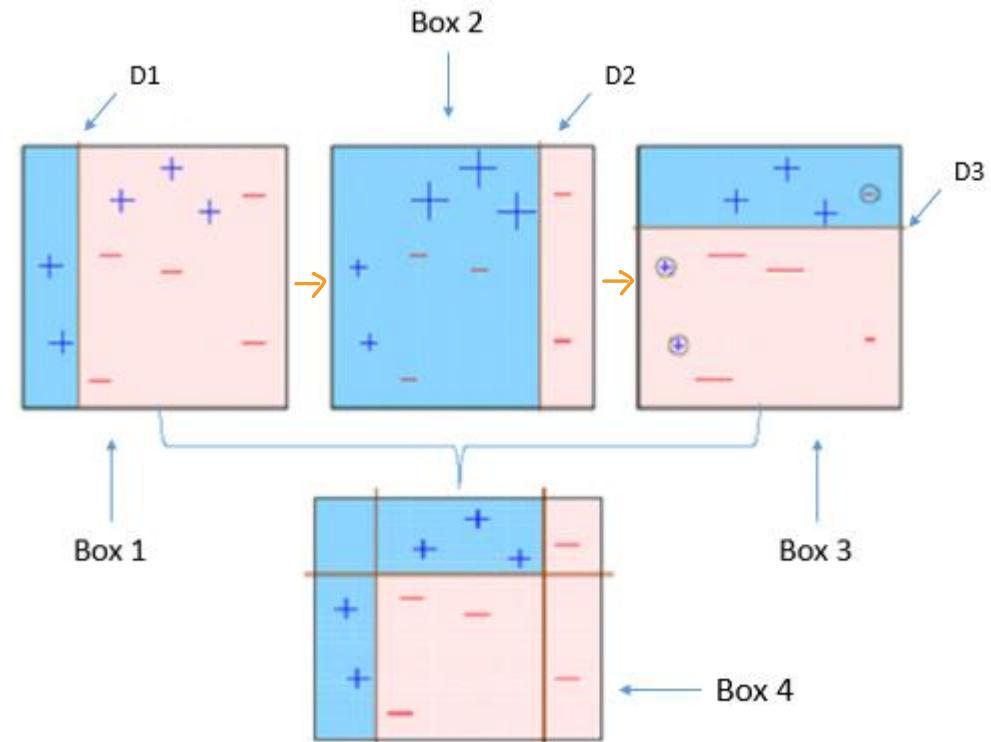
Sequential



AdaBoost (Adaptive Boosting)

- Forest of weak learners (trees with only 1 feature; **stumps**).
- Each tree (stump) depends on the previous tree's **errors** rather than being independent.

- 1) Starting with usual splitting criteria! (*impurity index like gini index or entropy*)
- 2) Each tree (stump) gets **different weight** based on its prediction accuracy.
- 3) Each **observation** gets a weight inversely related to its predicted outcome. (ex, misclassified ones get more weight).
- 4) **Aggregation** is done based on each weak learner's weight.



Source: [Towards data science](#) Classification ex.

Adaboost (Adaptive Boosting)

Adaboost combines multiple weak learners into a single strong learner.

This method does not follow Bootstrapping. However, it will create different decision trees with a single split (one depth), called decision stumps.

The number of decision stumps it will make will depend on the number of features in the dataset. Suppose there are M features then, Adaboost will create M decision stumps.

1. We will assign an equal sample weight to each observation. Initial Weight = $1/N$, where N is the no. Of observations.
2. We will create M decision stumps, for M number of features.
3. Out of all M decision stumps, I first have to select one best decision tree model. For selecting it, we will either calculate the Entropy or Genie coefficient. The model with lesser entropy will be selected (means model that is less disordered).
4. Now, after the first decision stump is built, an algorithm would evaluate this decision and check how many observations the model has misclassified. Suppose out of N observations, The first decision stump has misclassified T number of observations.
For this, we will calculate the total error (TE),

$$\text{Total error} = T/N$$

Now we will calculate the performance of the first decision stump.

$$\text{Performance of stump} = 0.5 \ln[(1-\text{TE})/(\text{TE})]$$

5. Now we will update the weights assigned before. To do this, we will first update the weights of those observations, which we have misclassified. The weights of wrongly classified observations will be increased and the weights of correctly classified weights will be reduced.

By using this formula:

$$\text{New Sample Weight for incorrect classification} = \text{old weight} * e^{(\text{performance})}$$

$$\text{New Sample Weight for correct classification} = \text{old weight} * e^{(-\text{performance})}$$

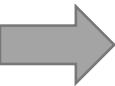
$$\text{Normalised weight} = (\text{New sample weight}) / (\text{sum of new sample weights})$$

Now respectively for each observation, we will add and subtract the updated weights to get the final weights. But these weights are not normalized as their sum is not equal to one. To do this, we will sum them and divide each final weight with that sum.

6. After this, we have to make our second decision stump. For this, we will make a class intervals for the normalized weights.

We want to make a second weak model. But to do that, we need a sample dataset on which the second weak model can be run. For making it, we will run N number of iterations. On each iteration, it will calculate a random number ranging between 0-1 and this number will be compared with class intervals we created and on which class interval it lies and that row will be selected for sample data set. So new sample data set would also be of N observation.

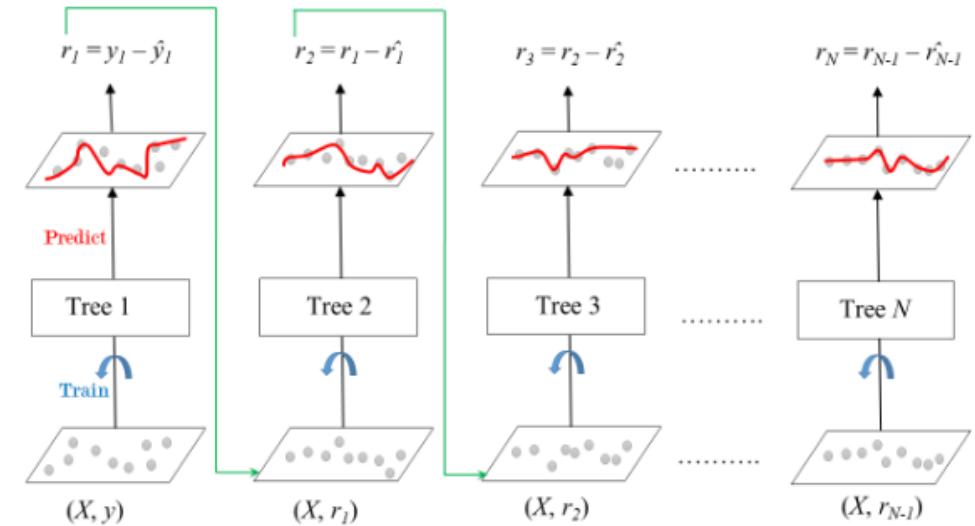
7. This whole process will continue from step 2 iteratively for M decision stumps. The final sequential tree would be considered as the final tree.



Gradient Boosting Machine (GBM)

- In **gradient boosting**, each weak learner corrects its predecessor's error.
- Unlike AdaBoost, the **weights** of the training instances are not tweaked, instead, each predictor is trained using the **residual errors** of predecessor as labels.
- Unlike AdaBoost, each tree can be **larger than a stump**. However, the trees are still small. By fitting a small tree to the residuals, the GBM slowly improve \hat{f} in areas where it does not perform well.
- Learning rate shrinks the contribution of each tree. There is a trade-off between learning rate and number of trees. Learning rate slows down the process even further, allowing for more and different shaped trees to attack the residuals.
- Aggregation is done by adding the first tree predictions and a scaled (shrunk) version of the following trees.

Source: [Geeksforgeeks](#)



GBM PSEUDO ALGORITHM

Inputs:

- ① $\{(x_i, y_i)\}_{i=1}^n$ ith row of Data containing features & Target Variable.
- ② Differentiable Loss fun: $L(y, F(x))$
- ③ No. of Trees.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M .

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma). \quad \leftarrow \frac{d}{d\hat{y}} \sum_{i=1}^n \frac{1}{2} (y_i - \hat{y})^2 = 0 \Rightarrow \hat{y} = F_0(x) = \text{Average of } y$$

2. For $m = 1$ to M : In Practice $M=100$ Trees.

1. Compute so-called pseudo-residuals:

$$r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n.$$

$$\begin{aligned} \text{Loss} &= \frac{1}{2} (\hat{y}_i - \hat{y})^2 \\ \frac{\partial L}{\partial \hat{y}} &= -(y_i - \hat{y}) \Rightarrow -\frac{\partial L}{\partial \hat{y}} = (y_i - \hat{y}) = r_{im} \rightarrow \text{residual for } i^{\text{th}} \text{ observation with model } m. \end{aligned}$$

2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$. $\text{Fit a base learner (tree) } h_m(x) \text{ with } \{(x_i, r_{im})\}$

3. Compute multiplier γ_{jm} by solving the following one-dimensional optimization problem: for $j = 1, \dots, J_m$

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \quad \leftarrow \frac{d}{d\hat{y}} \sum_{i=1}^n \frac{1}{2} [y_i - (F_0(x) + \hat{y})]^2 = 0 \Rightarrow \hat{y} = \gamma$$

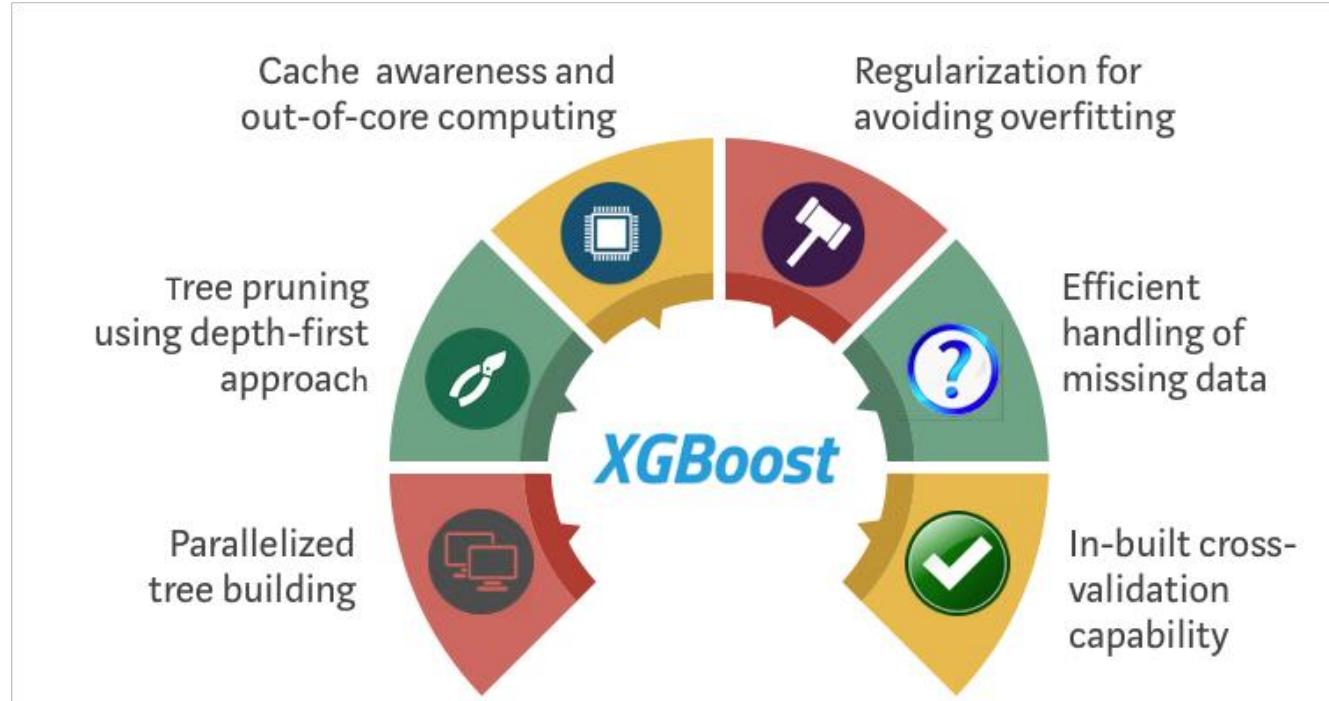
4. Update the model:

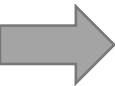
$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{(x \in R_{jm})}, \quad \nu \in [0, 1]$$

3. Output $F_M(x)$.

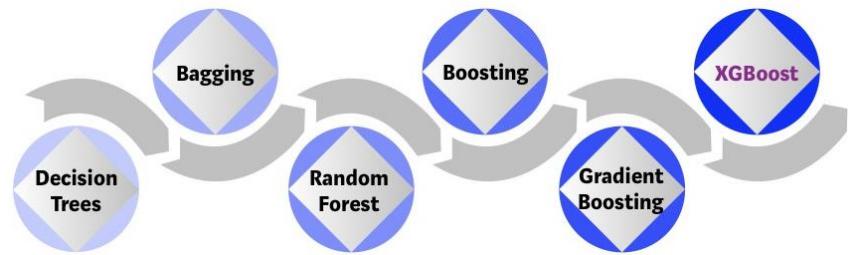
Extreme Gradient Boosting (XGBoost)

- XGBoost is a refined and customized version of a gradient boosting decision tree system, created with **performance** and **speed** in mind.
- **Extreme** refers to the fact that the algorithms and methods have been customized to push the limit of what is possible for gradient boosting algorithms.



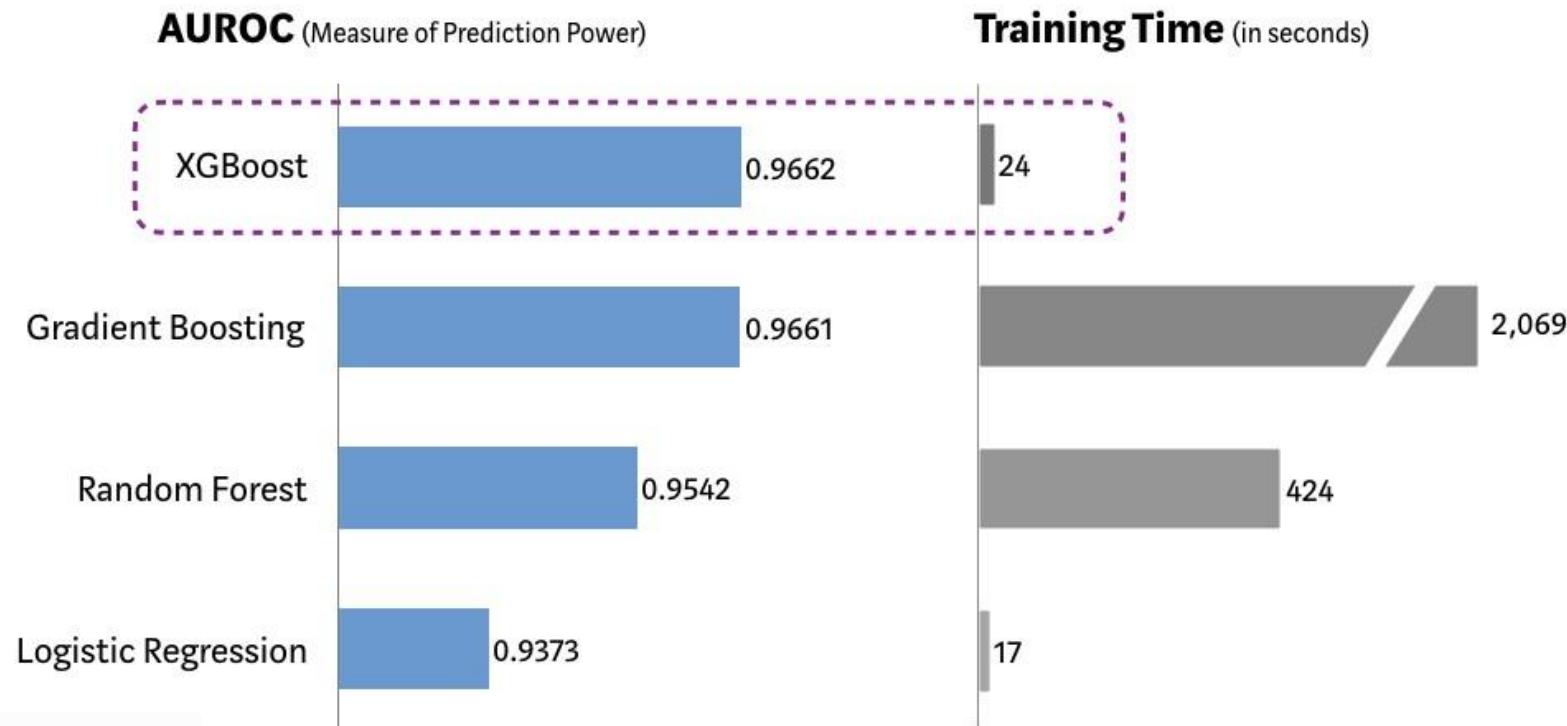


Put it all together!



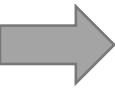
Performance Comparison using SKLearn's 'Make_Classification' Dataset

(5 Fold Cross Validation, 1MM randomly generated data sample, 20 features)



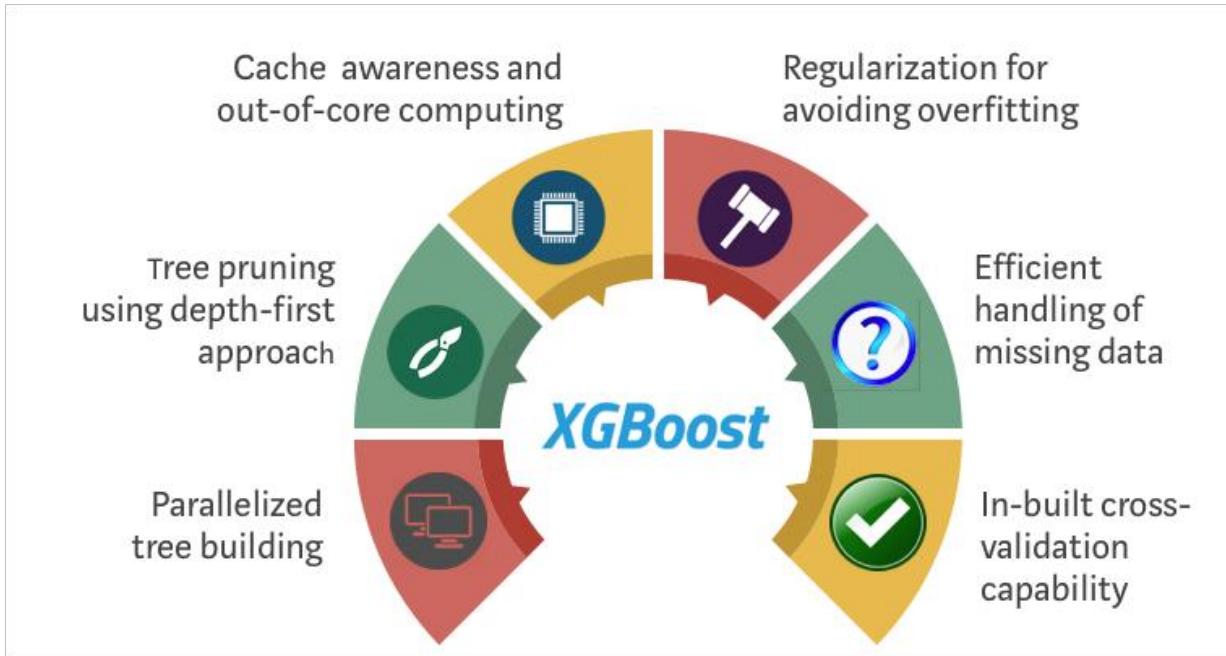
Part II

Pros and Cons



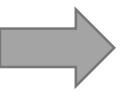
XGBoost's Pros and Cons

Pros:



Cons:

- XGBoost is **more difficult to** understand, visualize and to **tune** compared to AdaBoost and random forests. There is a multitude of hyperparameters that can be tuned to increase performance.



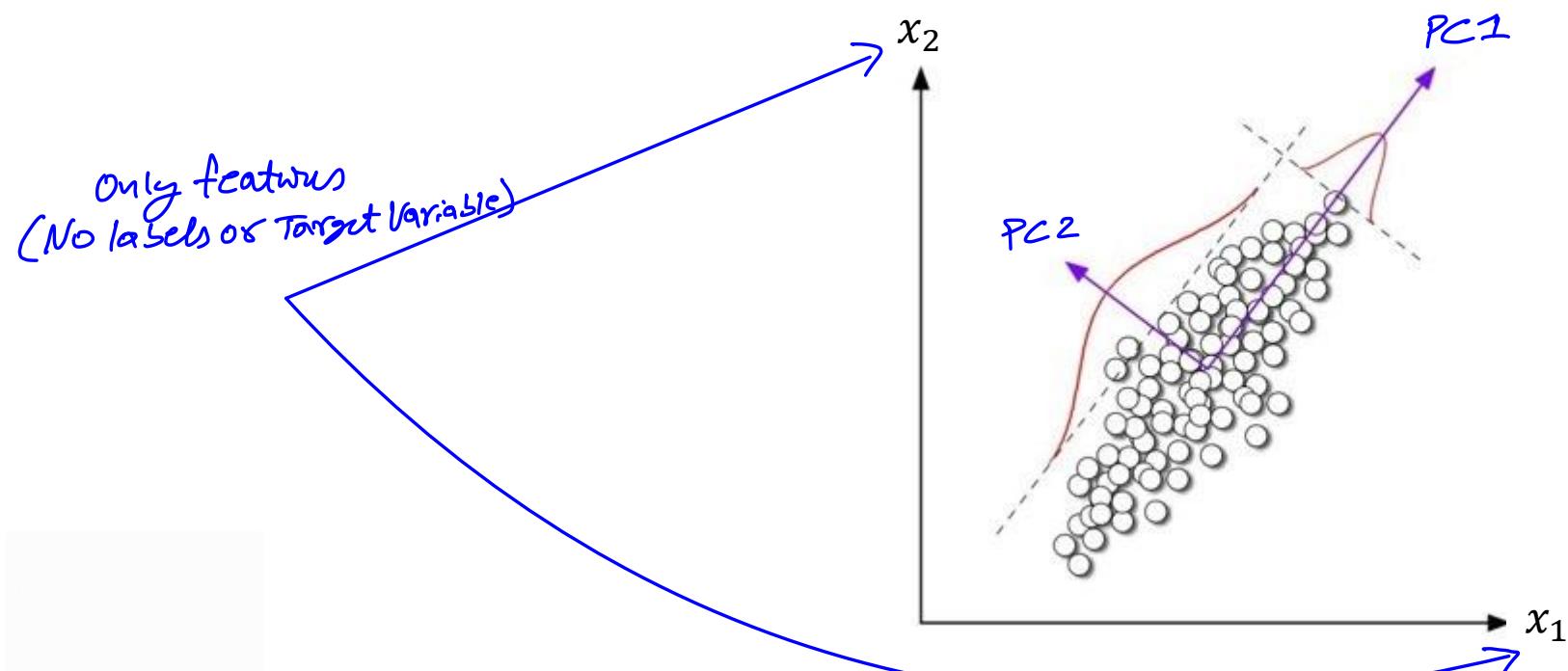
Students' questions

1. I'm still a little unclear on what exactly boosting is and how it differs from bagging.
2. Is boosting just a decision tree thing? Or can the same technique be applied to other models?
3. Why would you use something other than XGboost?
4. What is the "additive" model utilized by Gradient Boosting? *sequential.*

- Decision Trees are best weak learners as they are the most interpretable & can handle Non-linearity. They are also very fast. By changing depth of a tree we can easily balance trade-off between Bias & Variance.

Class -23

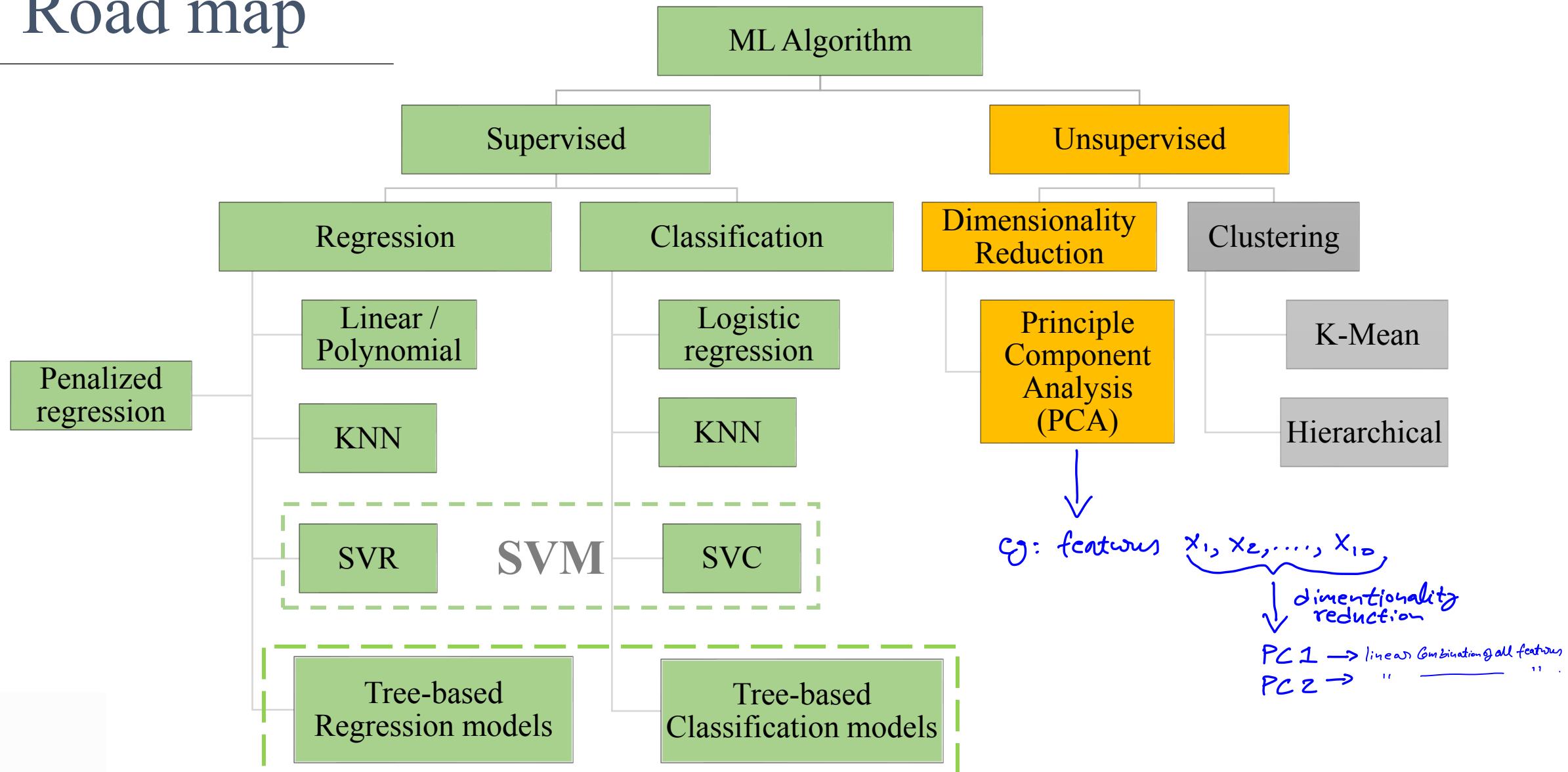
Principal Component Analysis (PCA)

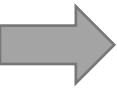


Goal: Discover patterns,
Subgroups among data.

PCA looks for most Variation
in dataset. It sorts the
axis of new feature space that
the variation of data is maximized.

Road map

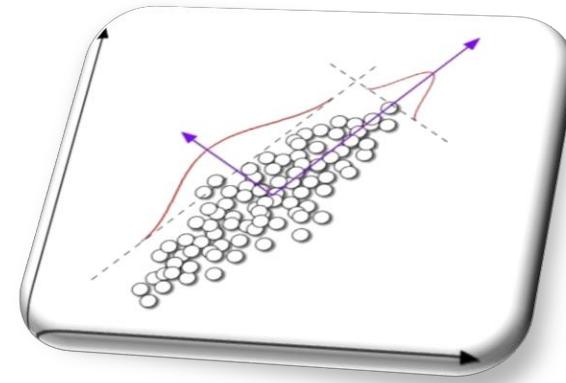




Topics

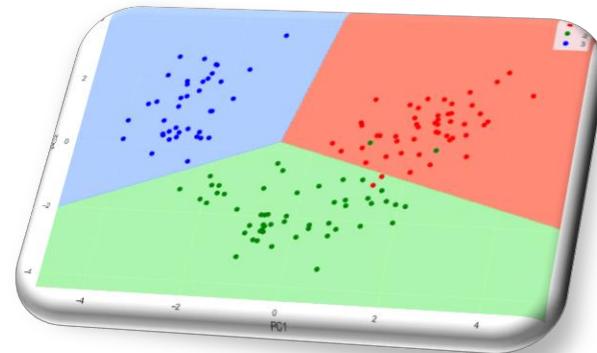
Part I

1. Unsupervised Machine Learning
2. Principal Component terminology



Part II

1. Principal Components Analysis (PC)
2. Scree plot

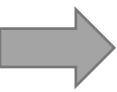


Part III

1. Why PCA? Pros and Cons!
2. Applications of PCA

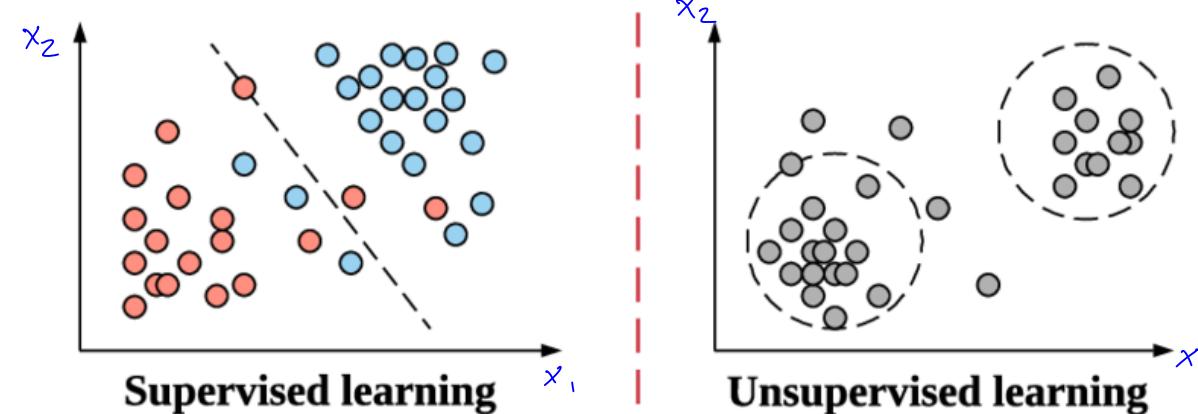
Part I

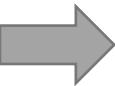
1. Unsupervised Machine Learning
2. PC terminology



Unsupervised Learning

- **Unsupervised learning** is a machine learning technique that does not use labeled data (no target variable)
- The **goal** is to discover the underlying patterns and find groups of samples that behave similarly.
- The two main types of unsupervised learning algorithms are:
 - 1) Dimension reduction algorithm
 - Principal Component Analysis
 - 2) Clustering techniques
 - K-Mean
 - Hierarchical

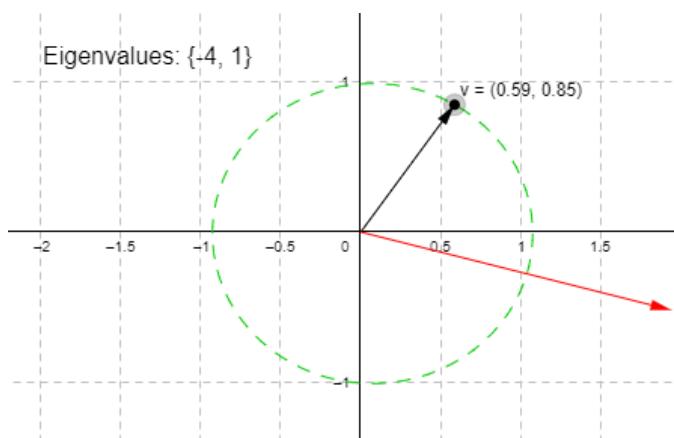
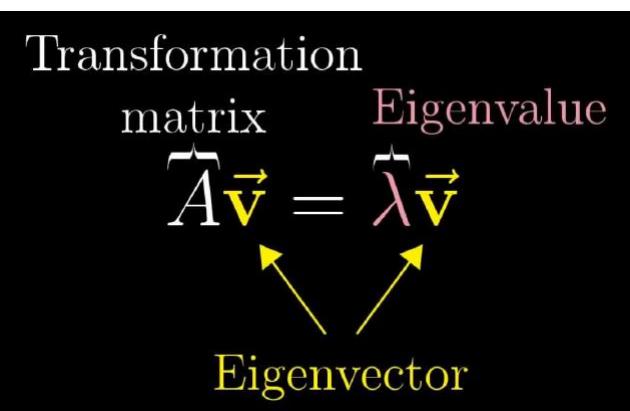
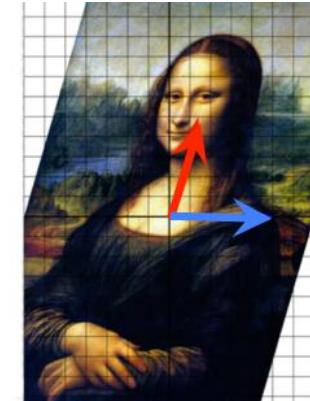
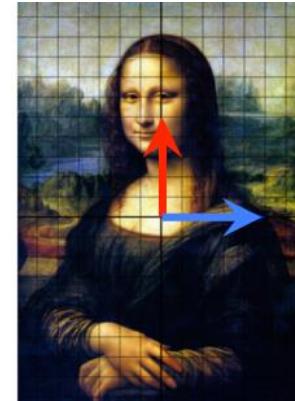




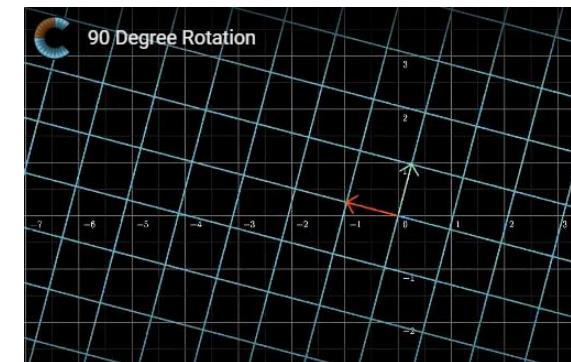
Eigen things!

But its eigenvalue (magnitude) may change.

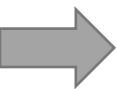
- Eigenvector does not change direction in a transformation.
- In this mapping, the blue arrow is eigenvector (why?) *doesn't change direction.*
- Its eigenvalue = 1 (why?) *magnitude didn't change.*
- For a square matrix A, an Eigenvector and Eigenvalue are defined as follow:



What is a transformation matrix?

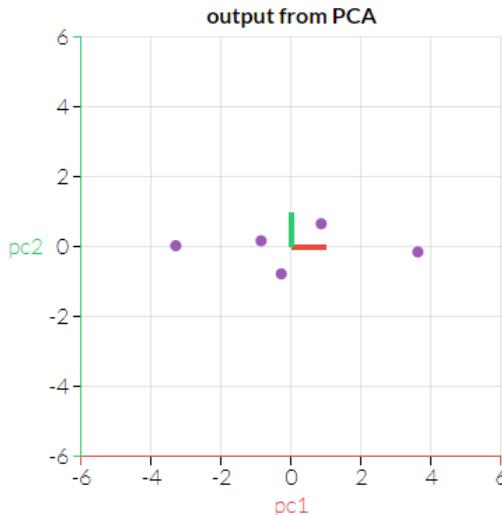
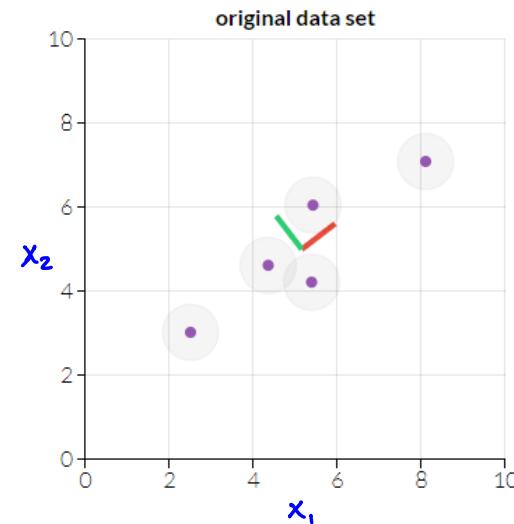


- After Transformation eigenvectors are vectors that do not change direction & eigenvalues are the magnitude of eigenvector after transformation.



PC terminology (connecting PC and Eigenthings)

- The **eigenvectors** and **eigenvalues** of a **covariance matrix** represent the “core” of a PCA
- Consider the following data points:



- We want to find a **direction(s)** in the data set that **explain the most variation**. So, our transformation matrix will be the **covariance matrix**.
- The **principal components** (**eigenvectors** of the covariance matrix) determine the **directions** of the new feature space, and the **eigenvalues** determine their **magnitude**. In other words, the eigenvalues explain the variance of the data **along the new feature axes**.

$$A = \text{Cov}(X)$$

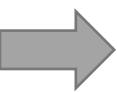
Covariance of feature space

e.g. for 2 features

$$\text{Cov}(X) = \begin{pmatrix} \text{Var}(x_1) & \text{Cov}(x_1, x_2) \\ \text{Cov}(x_2, x_1) & \text{Var}(x_2) \end{pmatrix}$$

If you want to reduce the dimension of the data to 1, which PC would you drop? **PC 2**





PC terminology (PC definition)

In summary:

- Principal components are vectors that define a new coordinate system in which the first axis goes in the direction of the highest variance in the data.
- The second axis is orthogonal to the first one and goes in the direction of the second highest variance in the data.

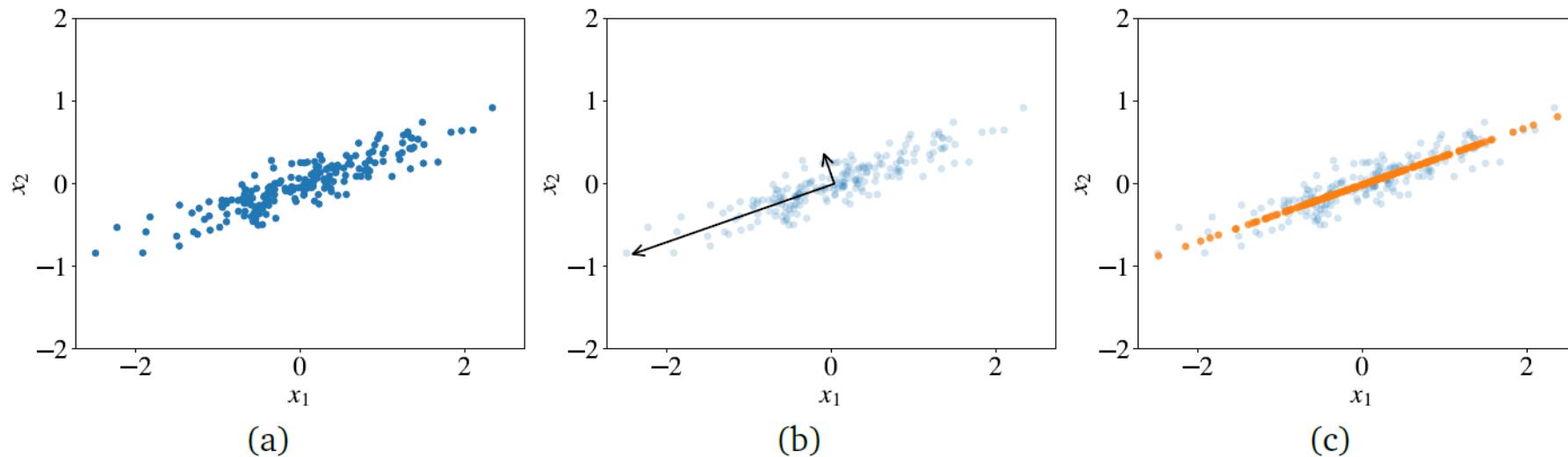
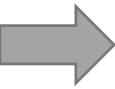
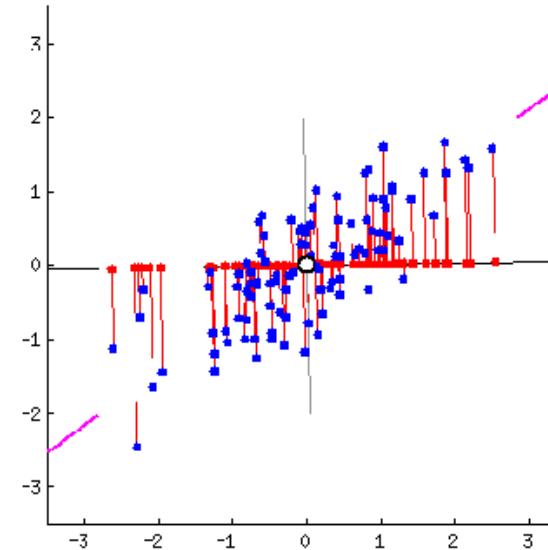
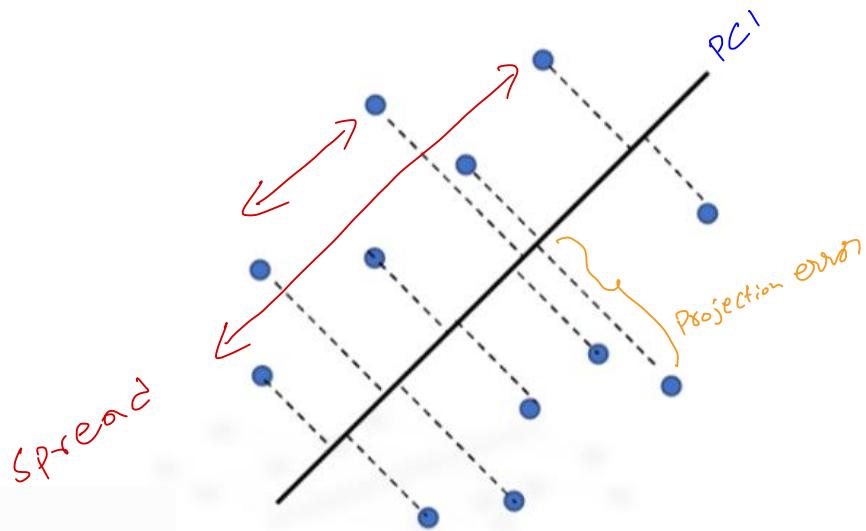


Figure 9.7: PCA: (a) the original data; (b) two principal components displayed as vectors; (c) the data projected on the first principal component. | Source: The hundred-page machine learning book



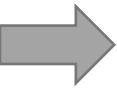
PC terminology (the objective function)

- **Projection errors:** The perpendicular distance (Euclidian) between the data point and a Principal Component.
- **Spread:** Variation of the data along Principal component.
- In PCA the **goal** is to **minimize the projection errors** (or equivalently maximize the spreads)

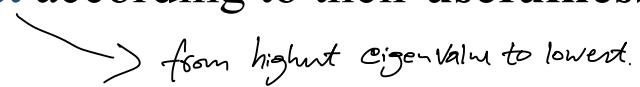


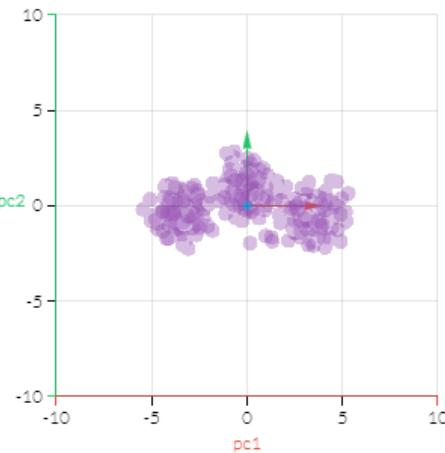
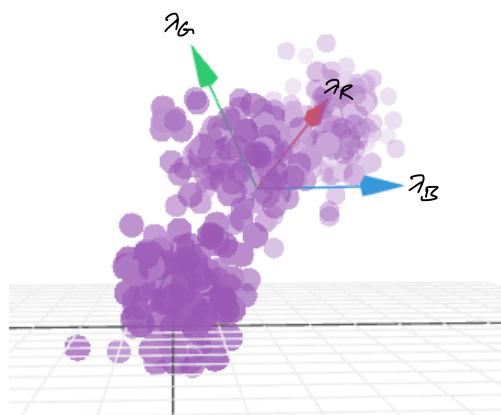
Part II

1. Principal Component Analysis (PCA)
2. Proportion Variance Explained
3. Scree plot



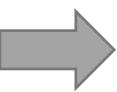
Principal Component Analysis (PCA)

- **Dimension reduction** aims to represent a dataset with many typically correlated features by a smaller set of features that still does well in describing the data.
- When **many features** in a dataset, **visualizing** the data or **fitting models** to the data may become extremely complex and “noisy”.
- **Principal components analysis (PCA)** is used to summarize or transform highly correlated features of data into a few main, uncorrelated **components**. (PC_1 is orthogonal to PC_2)
- The PCA algorithm orders the eigenvectors from highest to lowest according to their **usefulness** in explaining the **total variance** in the initial data (i.e., eigenvalues) 



e.g:
 $\gamma_R > \gamma_G > \gamma_B$
60% 30% 10%
↳ Explains the % variation in data

γ_R & γ_G explains 90% of variation in data



PCA details

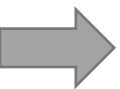
- The PC1 of a set of features X_1, X_2, \dots, X_p is the **normalized** linear combination of the features that has **the largest variance**.

$$PC_1 = \phi_{11}X_1 + \phi_{21}X_2 + \dots + \phi_{p1}X_p \quad \text{where } \sum_{j=1}^p \phi_{j1}^2 = 1$$

- The elements $\phi_{11}, \dots, \phi_{p1}$ are referred to as **loadings** of PC1.
- Note that the X features are **standardized** (why?) distance metric (projection error)
- The loading vector ϕ_1 defines a direction in feature space along which the data vary the most i.e., maximizing the variance in that direction!

$$\underset{\phi_{11}, \dots, \phi_{p1}}{\text{maximize}} \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^p \phi_{j1} x_{ij} \right)^2 \quad \text{subject to } \sum_{j=1}^p \phi_{j1}^2 = 1$$

Objective fn.



USA arrests data: Biplot

- USA arrests data contains the number of arrests per 100k residents for each of the 50 states.
- The **features** are murder, assault, rape and urban population.
- PCA was performed after **standardizing** each feature!
The loadings are as follow:

		PC1	PC2
x_1	Murder	0.5358995	-0.4181809
x_2	Assault	0.5831836	-0.1879856
x_3	UrbanPop	0.2781909	0.8728062
x_4	Rape	0.5434321	0.1673186

- Biplot displays both the PC **scores** and PC **loadings**.

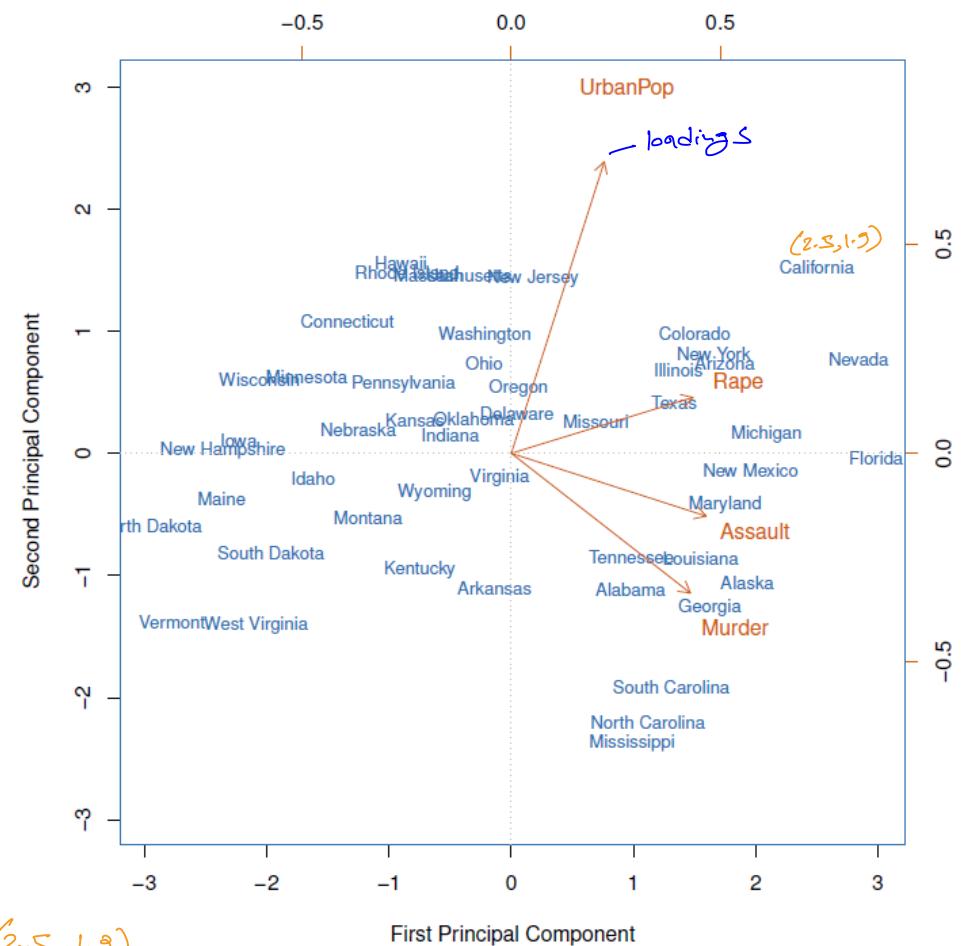
↓
final values for states in 2-D.

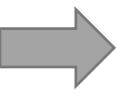
Source: ISLR first edition

$$\text{Score}_{\text{PC1}}(\text{California}) = (0.53 \times 0.27) + (0.58 \times 1.26) + (0.27 \times 1.75) + (0.54 \times 2.06) \approx 2.5$$

$$\text{Score}_{\text{PC2}}(\text{California}) = (-0.41 \times 0.27) + \dots + (0.16 \times 2.06) \approx 1.9$$

	x_1	x_2	x_3	x_4
California	0.2782682	1.262814	1.7589234	2.067820
Florida	1.7476714	1.970778	0.9989801	1.138967
New Hampshire	-1.3059321	-1.365049	-0.6590781	-1.252564

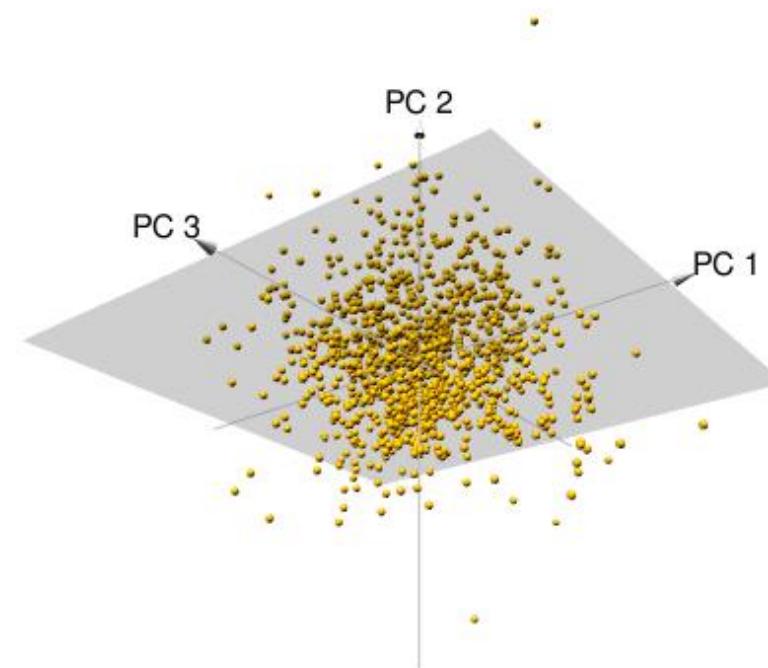
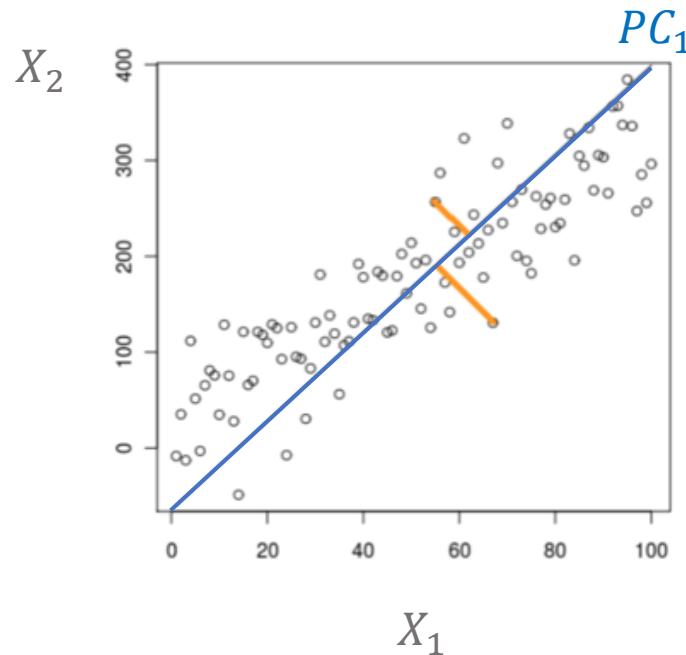


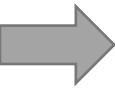


Another interpretation of PCA

- PCA find the **hyperplane closest** to the observations!
- What is the difference between **PCA** and linear regression then?

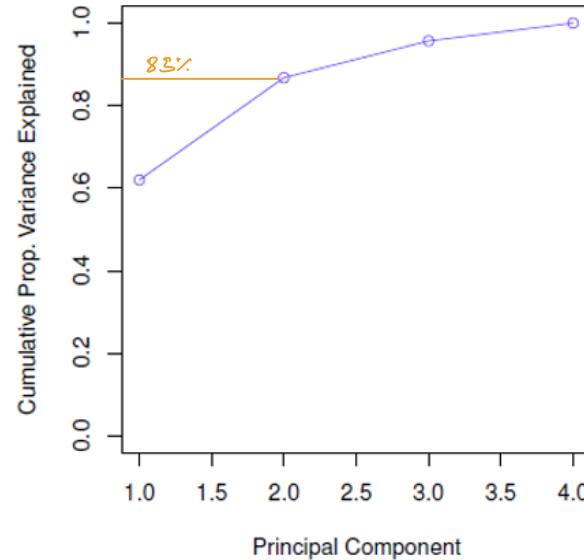
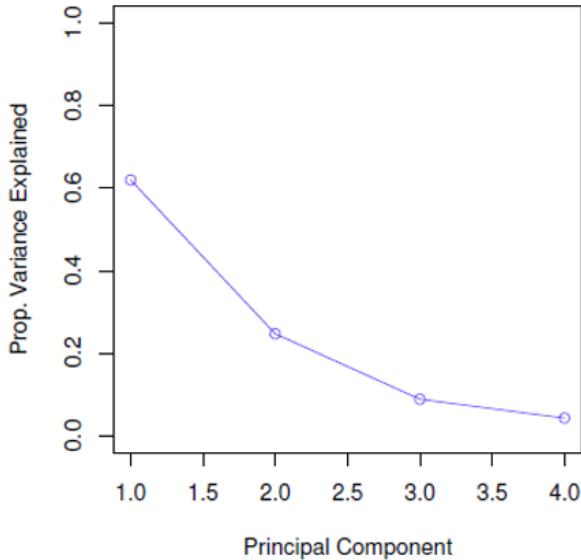
linear Regression is Supervised & the residual is not the \perp distance.





Scree plot

- Scree plot shows the proportion of total variance in the data explained by each principal component. This is also called **Proportion Variance Explained (PVE)**
- The PVEs **sum to one**. Sometimes they are displayed as **cumulative PVEs**.



- What is the **optimal number of PCs** here? Why cannot we use **cross validation**?

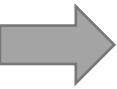
Rule of thumb is 85%
So in this case PC1 & PC2.
i.e. 2 PCs.

There is no target variable
 y to validate the model.

Part III

Why PCA? Pros and cons!

Applications of PCA



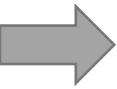
PCA's Pros and Cons

Pros:

- Reducing the number of features to the **most relevant predictors** is very useful in general.
- Dimension reduction **facilitates** the data visualization in two or three dimensions.
- **Before** training another supervised or unsupervised learning model, it can be performed as part of EDA to **identify patterns** and detect **correlations**.
- Machine learning models are **quicker to train**, tend to reduce overfitting (by avoiding the curse of dimensionality), and are easier to interpret if provided with lower-dimensional datasets.

Cons:

- Hard to interpret!

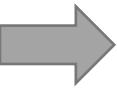


Applications of PCA (Example from CFA II reading 7)

- Consider a hypothetical Diversified Large Cap (**DLC**) 500 and Very Large Cap (**VLC**) 30:
 - DLC 500 can be thought of as a diversified index of **500 large-cap companies** covering all economic sectors
 - VLC 30 is a more concentrated index of the **30 largest publicly traded companies**.
- The dataset consists of index prices and more than **2,000** fundamental and technical **features**
- Multi-collinearity among the features is **inevitable!**
- To mitigate the problem, PCA can be used to capture the information and **variance** in the data.

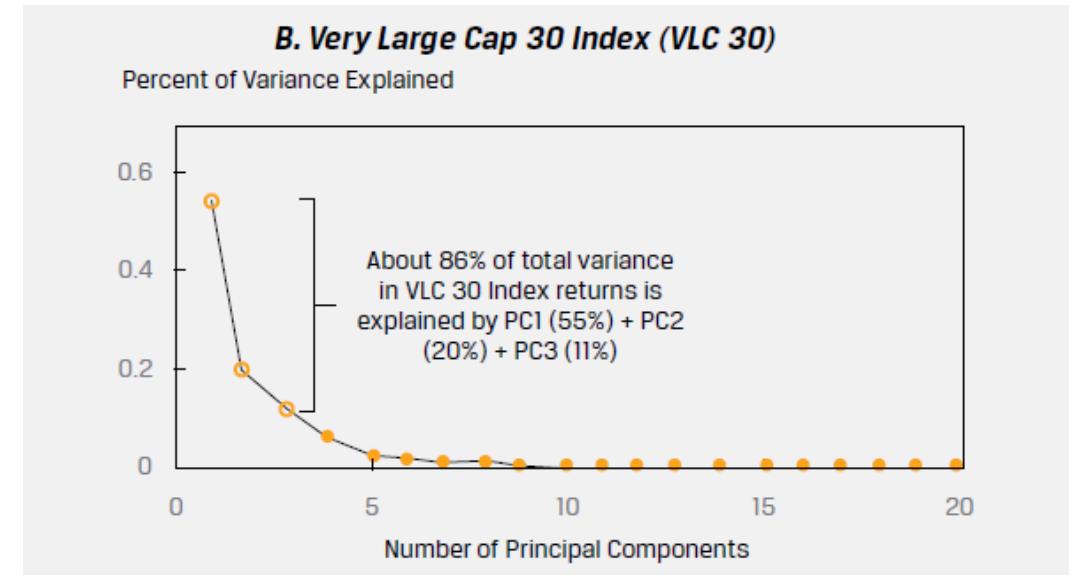
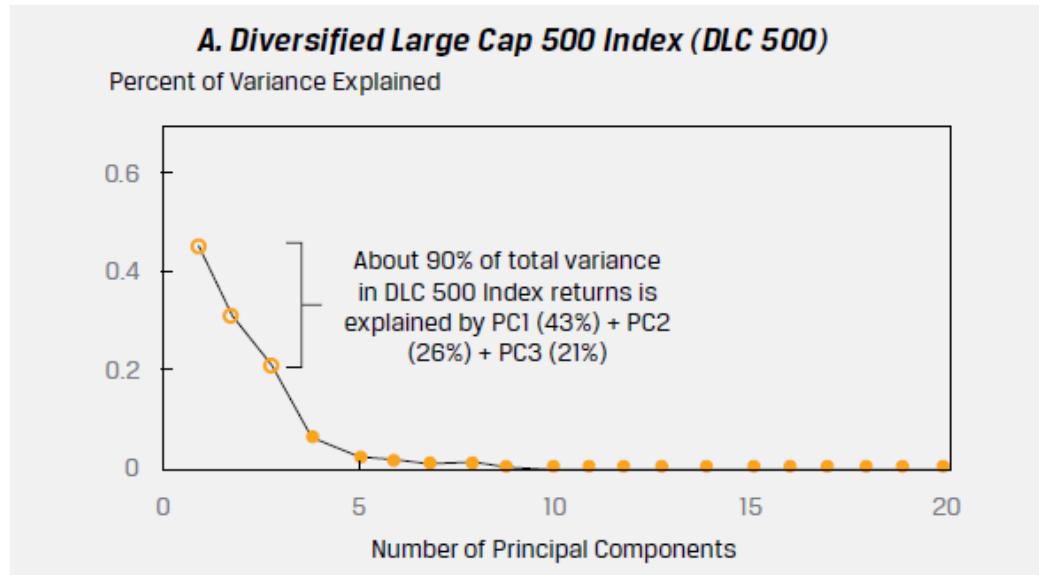
THE FACTOR ZOO

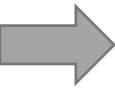




Applications of PCA (Example from CFA II reading 7)

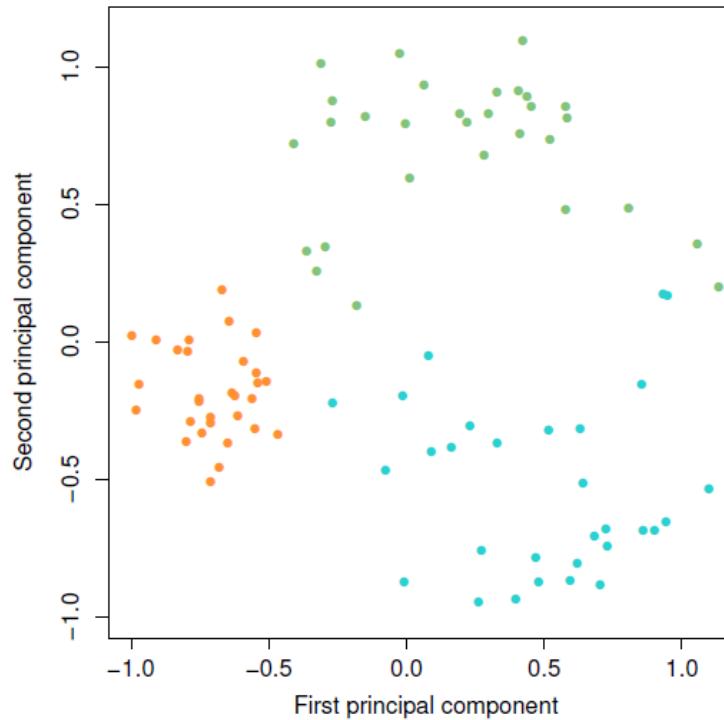
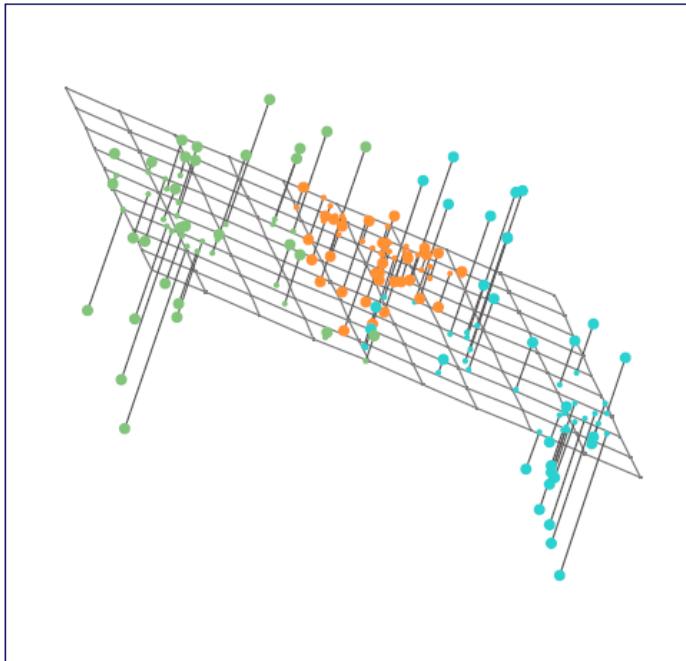
- The following **scree plots** show that of the **20** principal components generated, **the first 3 together** explain about 90% of the variance of DLC 500 and 86% of the VLC 30.

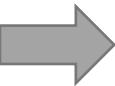




Applications of PCA (Data visualization)

- PCA can be fed into other **unsupervised** or **supervised learning models!**
- Using PCA with an unsupervised model like K-Mean **clustering**:

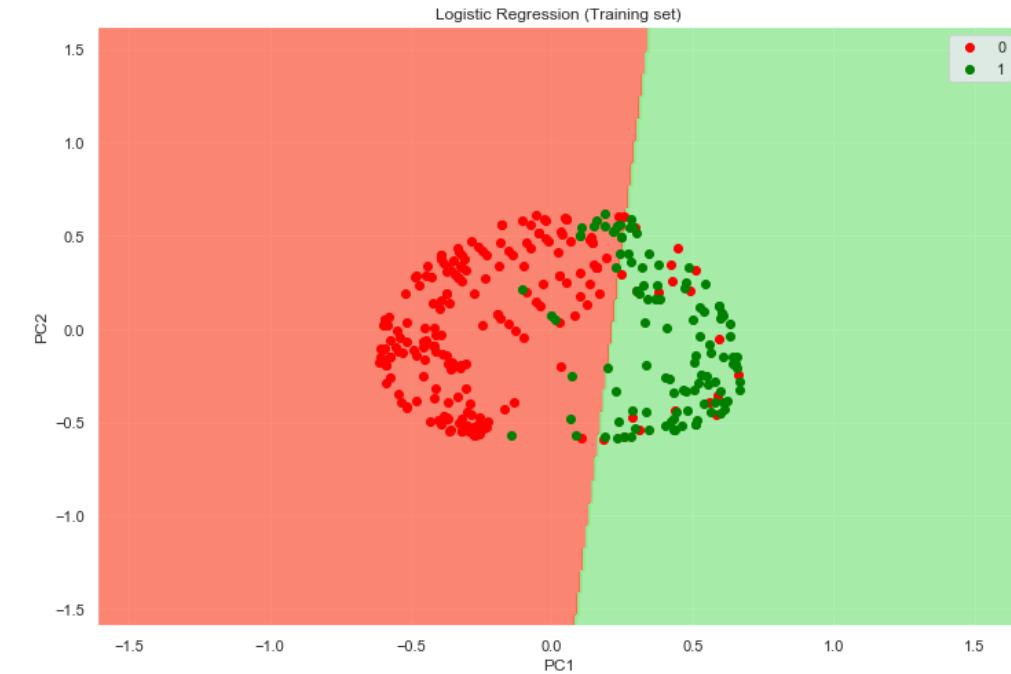


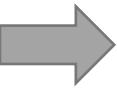


Applications of PCA (Kernel PCA)

(Using PCA as an input for supervised learning model)

- What if we want to use a linear classifier (regressor) but the data is **non-linearly separable**?
- Solution: **Kernel PCA**
- Kernel PCA uses a kernel function to project dataset into a higher dimensional feature space, where it is **linearly separable**. It is like the idea of Support Vector Machines.
- Using PCA with a supervised learning model like logistic regression:



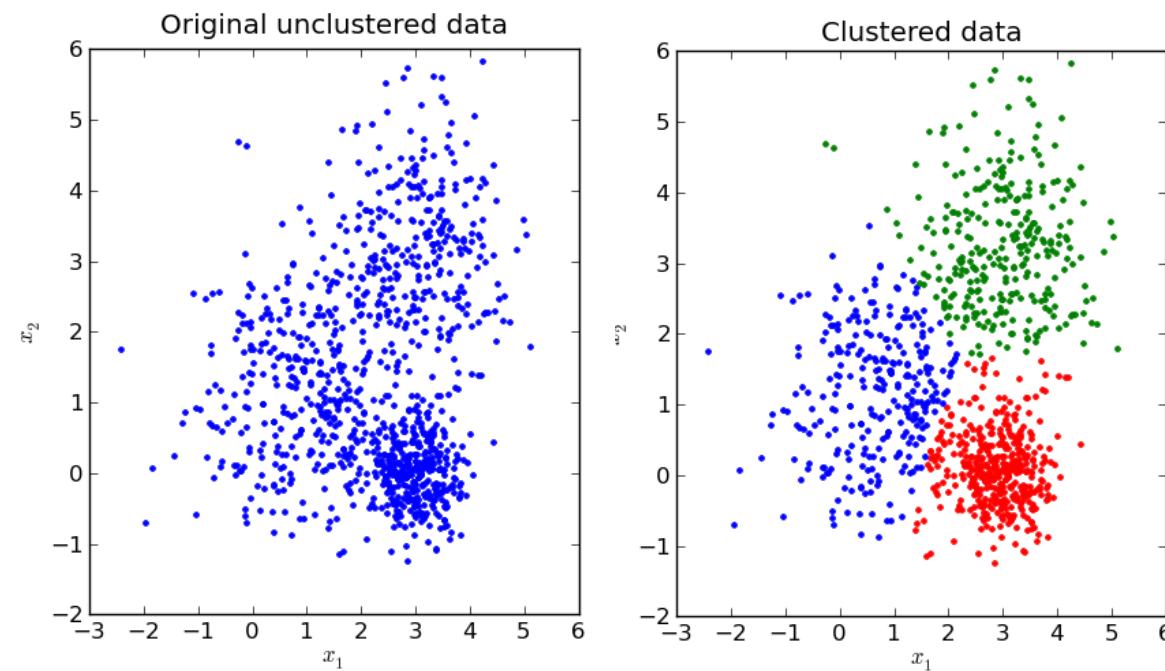


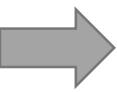
Students' questions

1. I am having a little trouble seeing how PCA is unsupervised learning.
2. If the best performing models can easily handle high-dimensional data, is dimensionality reduction mainly used for visualization?
3. I am still confused as to how PCA takes features that should be represented in 4 or more dimensions and changes them to be seen in a 2-dimensional plot.

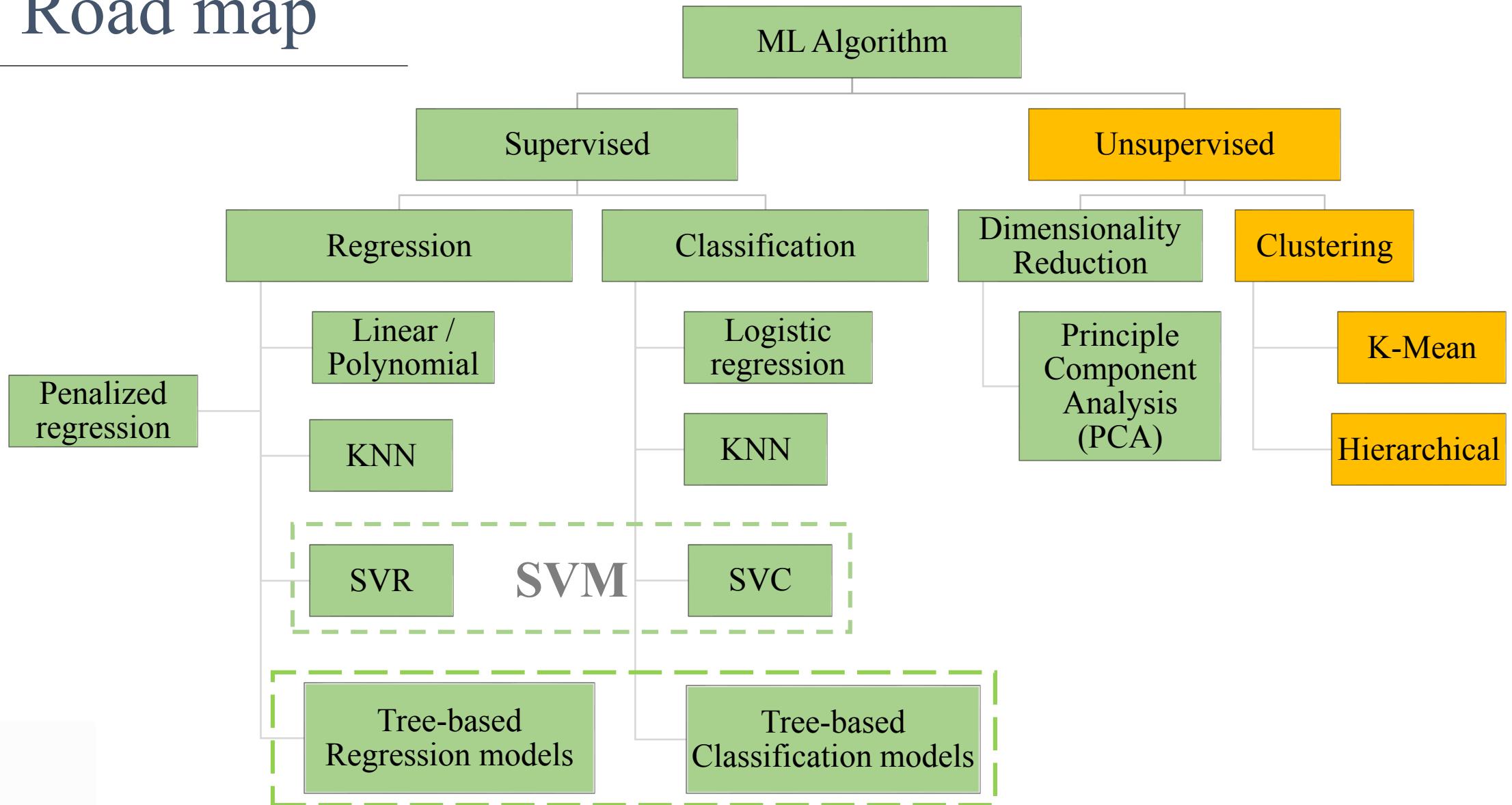
Class -25

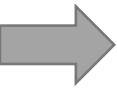
Clustering (K-Mean & Hierarchical)





Road map

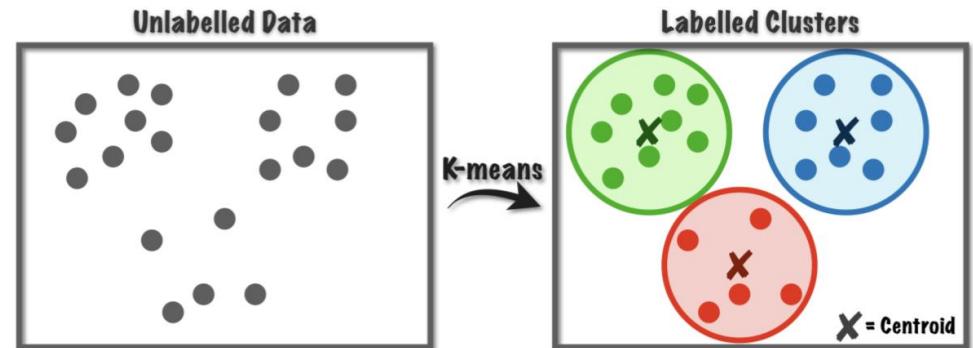




Topics

Part I

1. What is clustering?
2. Similarity/Dissimilarity metrics
3. PCA vs Clustering

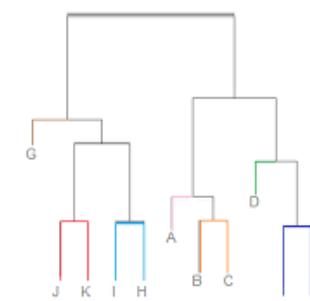
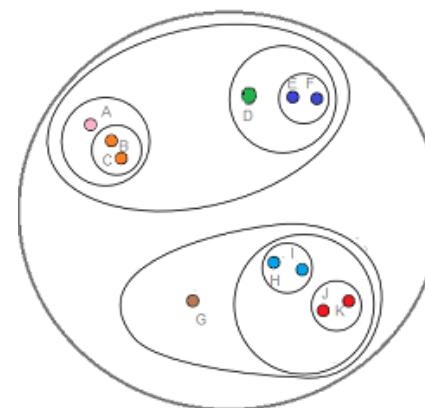


Part II

- ✓ K-Mean clustering

Part III

- ✓ Hierarchical Clustering

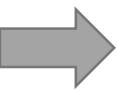


Part IV

- ✓ Applications in finance

Part I

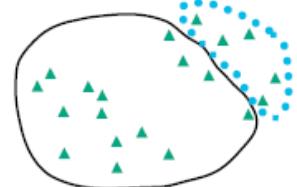
1. What is Clustering?
2. Similarity/Dissimilarity metrics
3. PCA vs Clustering



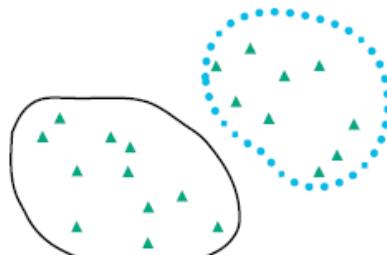
What is Clustering?

- Clustering is an **unsupervised** machine learning which is used to organize data points into **similar groups** called **clusters**.
- A cluster contains a subset of observations from the dataset such that all the observations within the same cluster are “**similar**.”
- The goal is to maximize the **intra-clusters** (within) **similarities** or equivalently to maximize the **inter-clusters** (between) **dissimilarities**.

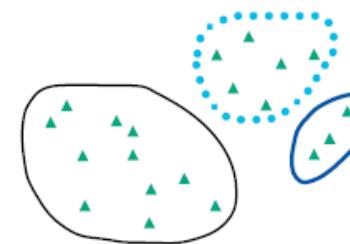
Bad Clustering

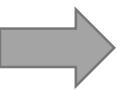


Good Clustering



(Maybe) Better Clustering



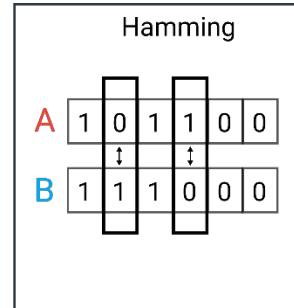
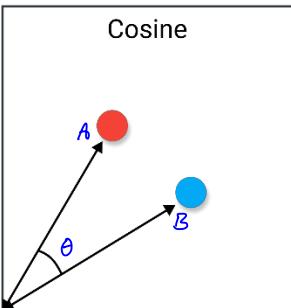
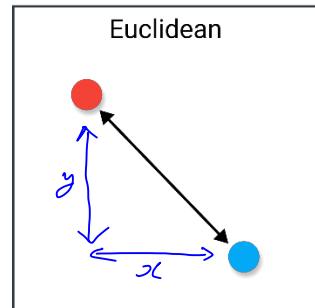


Similarity/Dissimilarity metrics

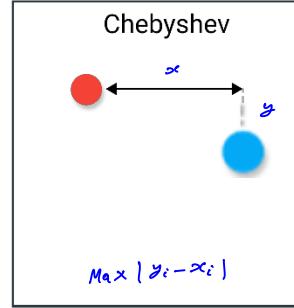
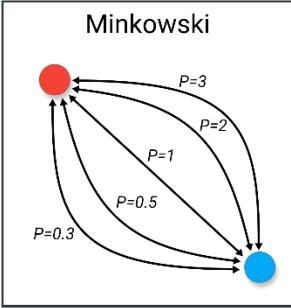
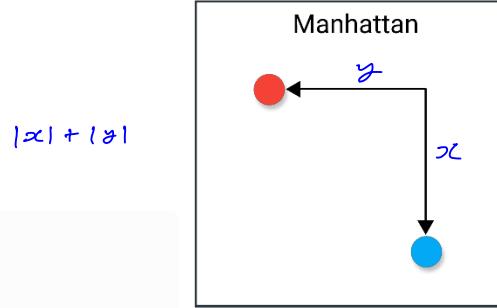
- Similarity/Dissimilarity between observations can be thought of as the distance between them.
- The smaller the distance, the more similar the observations; the larger the distance, the more dissimilar the observations.

$$\cos \theta = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}| |\vec{B}|}$$

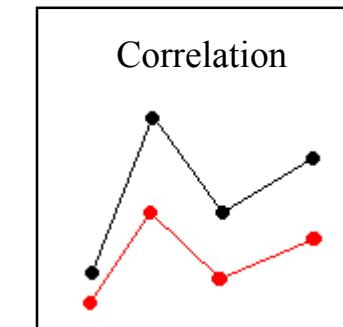
$\begin{cases} 1 \rightarrow A \& B \text{ Similar} \\ 0 \rightarrow A \perp B \\ -1 \rightarrow A \& B \text{ Not Similar} \end{cases}$

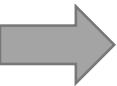


$$\sqrt{x^2 + y^2}$$



One of the distance metric used in Finance.

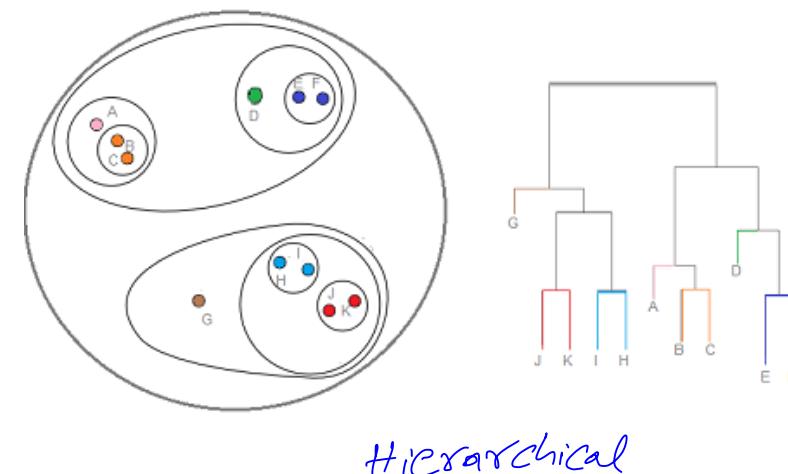
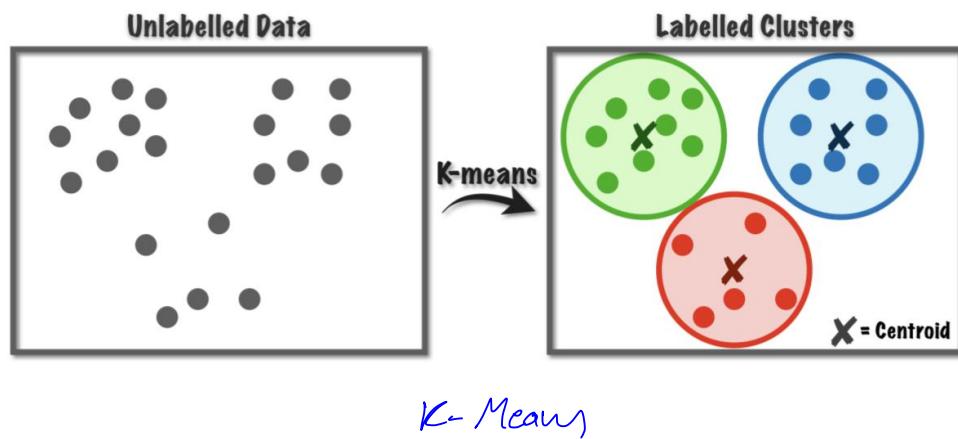




PCA vs Clustering

feature Selection = Select Certain features from feature Space like lasso.
feature extraction = Taking linear combination of features from feature Space eg. T-SNE
3 linear combination of all x's.

- Principle Component Analysis (Dimension reduction; feature extraction) looks for a low-dimensional representation of the observations that explains a good fraction of the variance.
- Clustering looks for homogeneous subgroups among the observations.
- Two of the more popular clustering approaches are:
 - K-Means clustering (*You have to come up with pre-specified K-number.*)
 - Hierarchical clustering



Part II

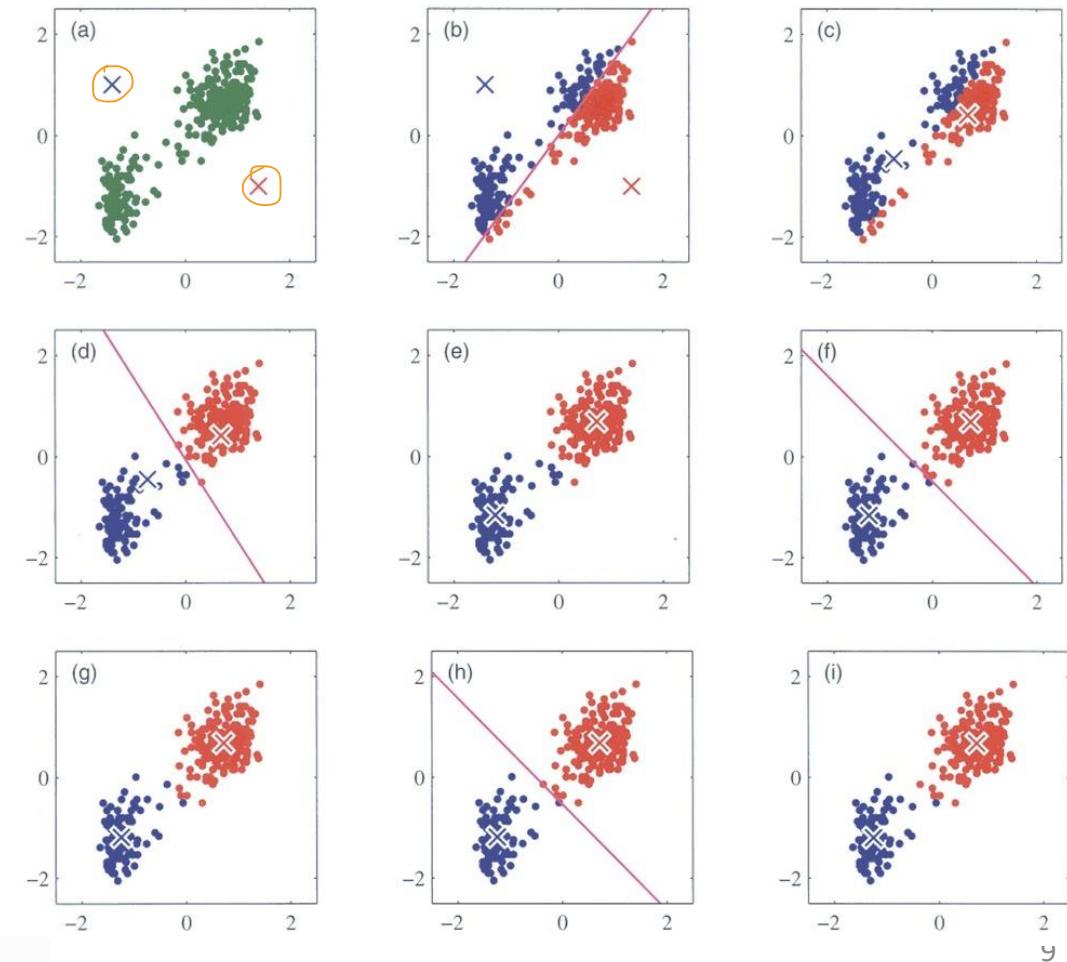
❑ K-Mean clustering

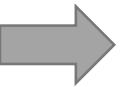
K-Means Clustering

- K-means is an algorithm that repeatedly partitions observations into a
 - fixed
 - pre-specified and
 - non-overlappingnumber of clusters, k (a hyperparameter)
- Each cluster is characterized by its **centroid (arithmetic mean position)**.
- K-means **minimizes** intra-cluster (within-cluster) distance

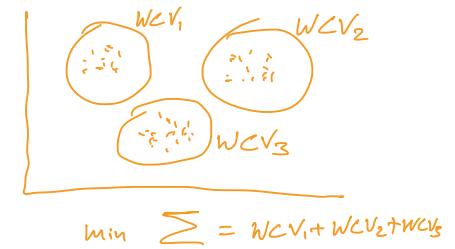
Algorithm 1 k -means algorithm

```
1: Specify the number  $k$  of clusters to assign.  $\text{eg } k=2$ 
2: Randomly initialize  $k$  centroids.  $\circ$ 
3: repeat
4:   expectation: Assign each point to its closest centroid.
5:   maximization: Compute the new centroid (mean) of each cluster.
6: until The centroid positions do not change.
```





K-Means clustering (details)

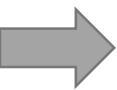


- Objective function: Minimizing the within-cluster variation (WCV)

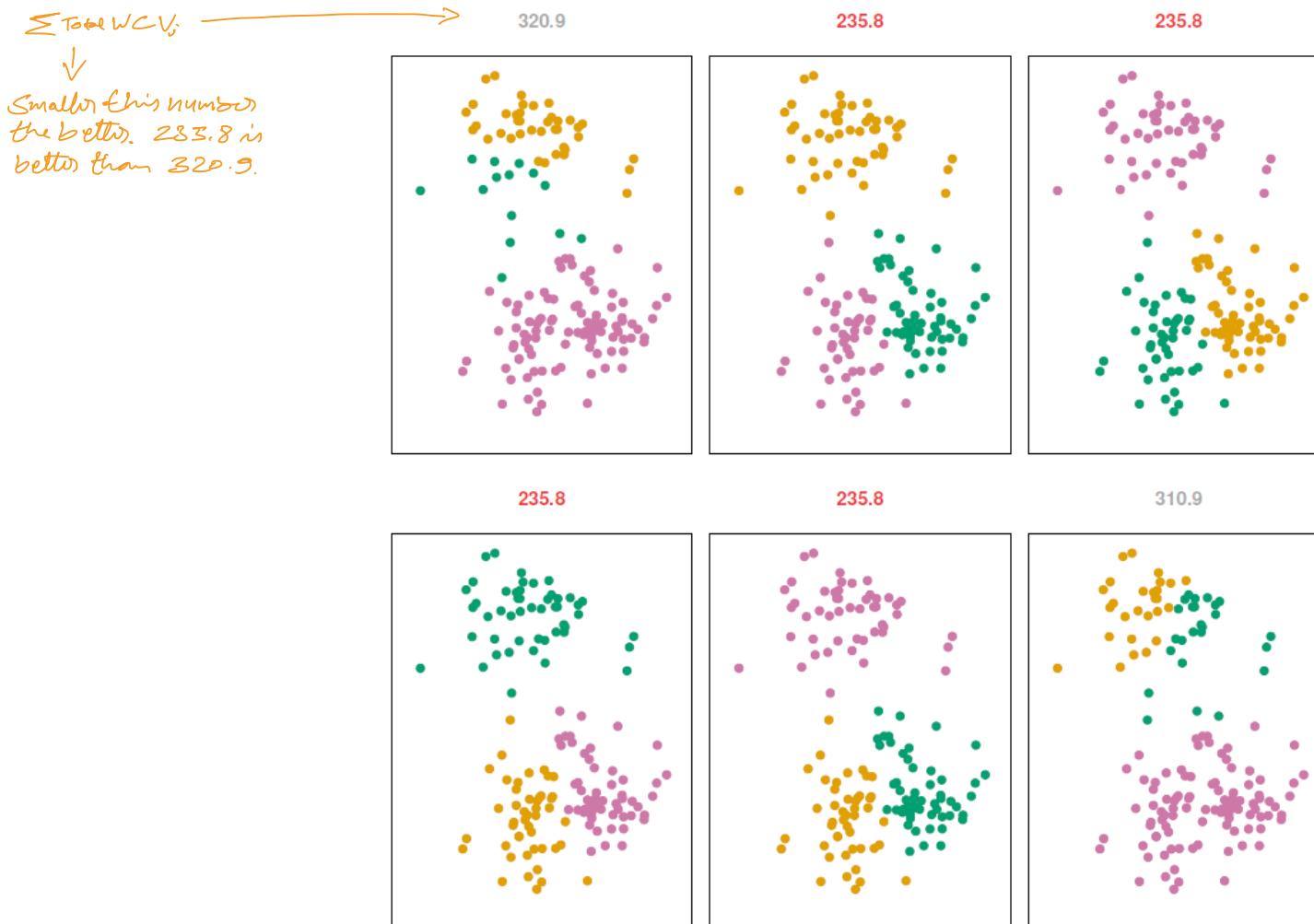
$$\underset{C_1, \dots, C_K}{\text{minimize}} \left\{ \sum_{k=1}^K \text{WCV}(C_k) \right\}$$

- This optimization says that we want to partition the observations into K clusters such that the **total within-cluster variation** is as small as possible.
- If we use Euclidian distance, then:

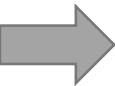
$$\underset{C_1, \dots, C_K}{\text{minimize}} \left\{ \sum_{k=1}^K \frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2 \right\}$$



Initial positioning of the centroids matter!



Initial value of centroid position matters & depending on that we may get different Total WCV. So to solve this & get global optimum rather than stuck in local optimum, we use K-means++. By default, python uses K-means++.



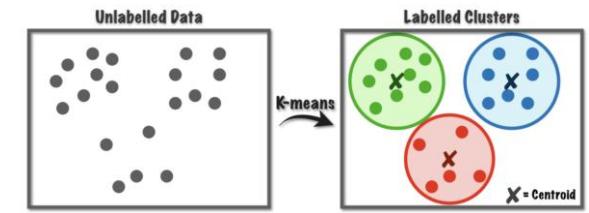
K-Means clustering pros and cons

Pros

- Simple!
- The k-means algorithm is **fast** and works well on **very large datasets**.
- Can help **visualize** the data and facilitate detecting trends or outliers.
- The k-means algorithm is among the most used algorithms in **investment practice**.

Cons

- The hyperparameter, **k**, must be decided **before** the algorithm can be run.
- The final assignment of observations to clusters can **depend on the initial location of the centroids**. Local optimum vs global optimum. (should be rerun with several initialization)

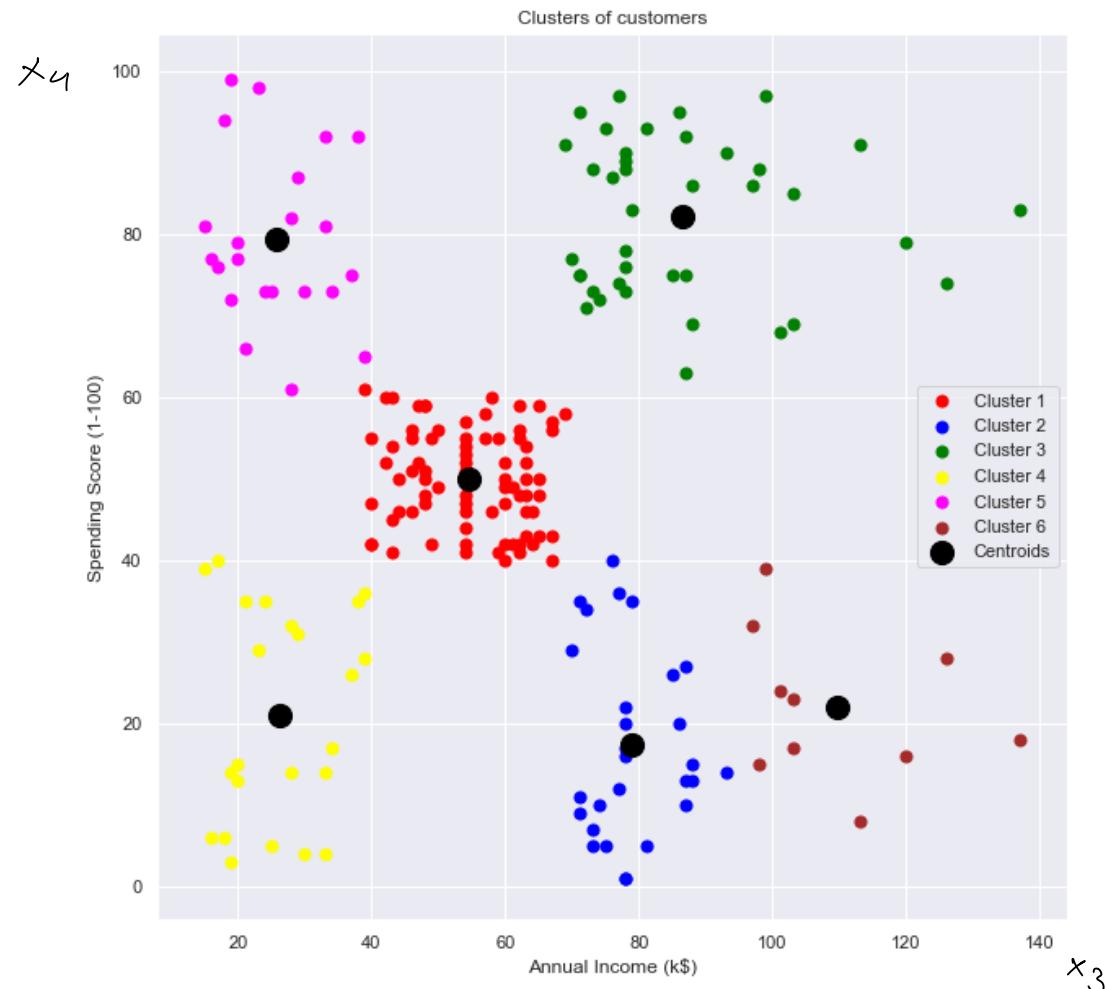
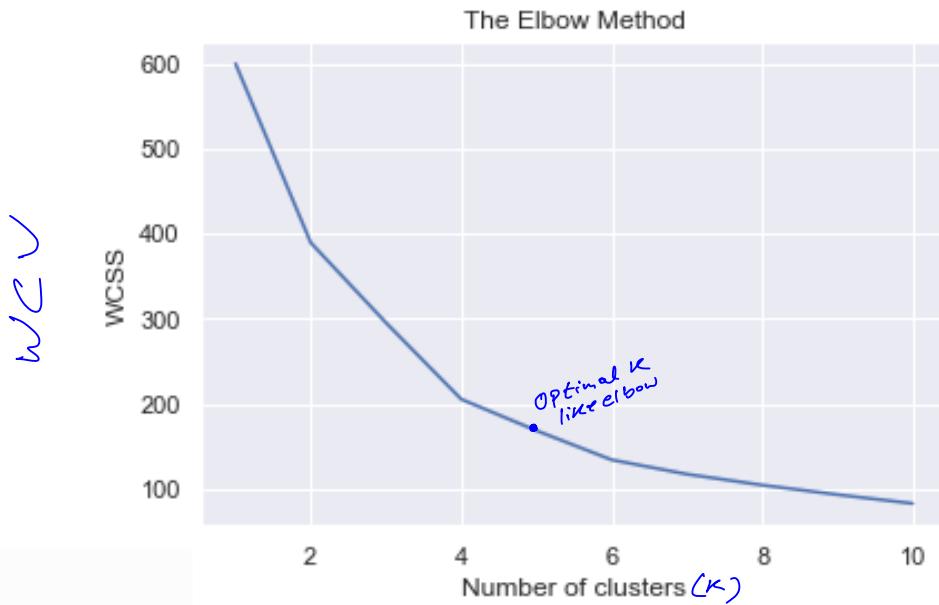


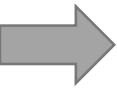
$x_1 = \text{age}$ $x_3 = \text{income}$
 $x_2 = \text{gender}$ $x_4 = \text{Spending score}$

(We could instead do PCA)
to find optimal K

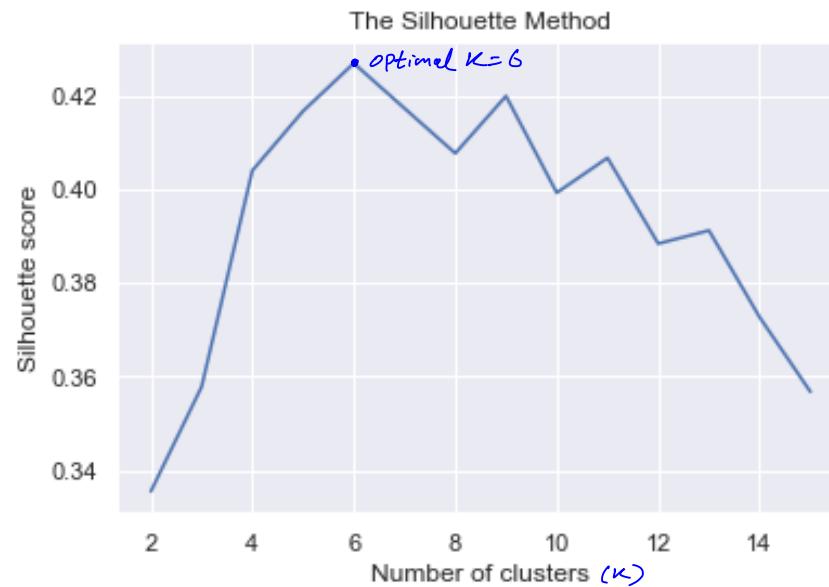
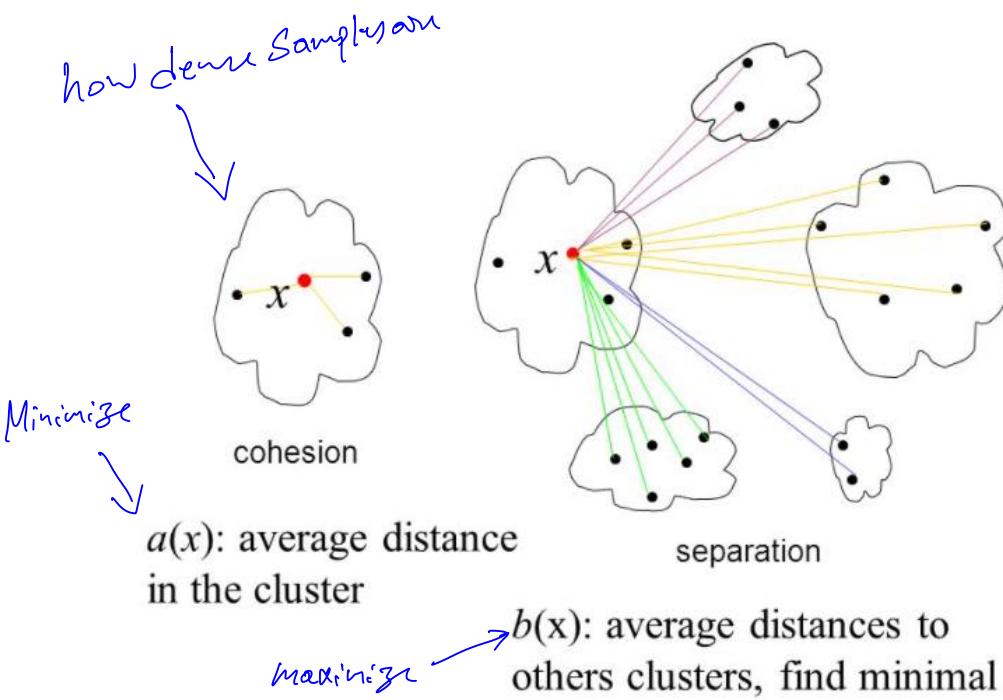
Optimal number of K (the elbow method)

$$\underset{C_1, \dots, C_K}{\text{minimize}} \left\{ \sum_{k=1}^K \text{WCV}(C_k) \right\}$$



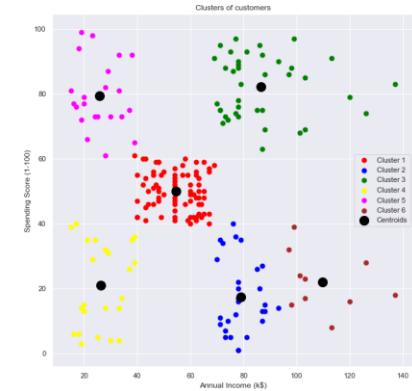


Optimal number of K (the Silhouette method)



$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad -1 \leq s(i) \leq 1$$

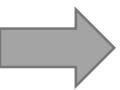
↳ higher the no. better



Part II

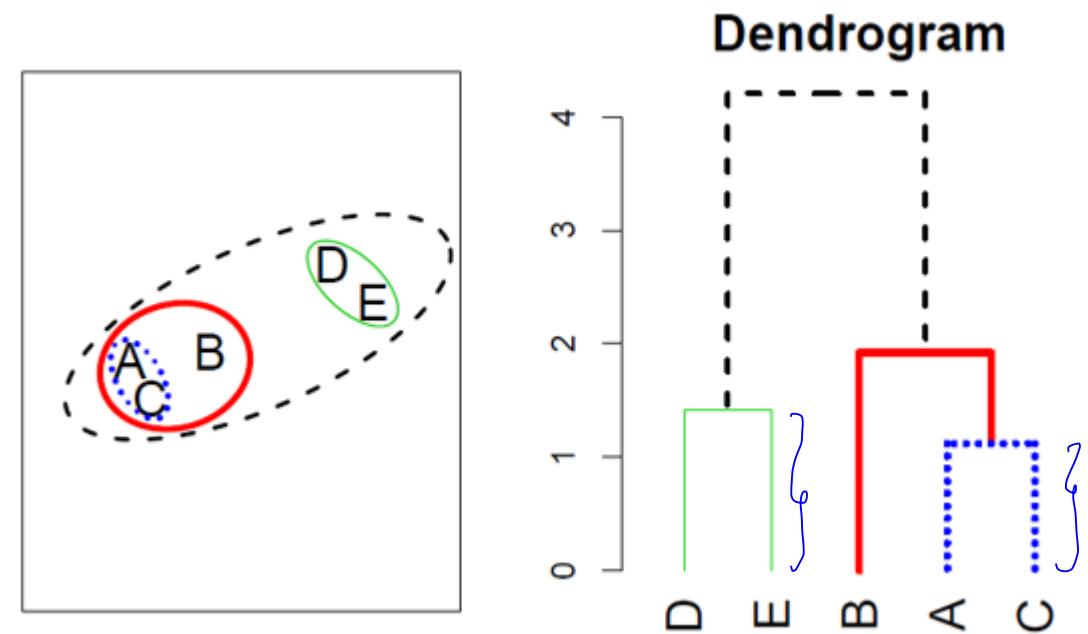
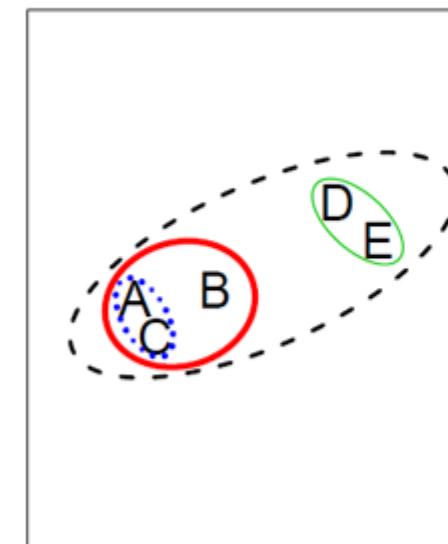
❑ Hierarchical Clustering

1. Agglomerative
2. Divisive

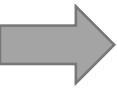


Hierarchical Clustering

- In **k-means** clustering, the algorithm seeks to partition the data into a **pre-specified** number of clusters k. All clusters are found **simultaneously**.
- In **hierarchical** clustering, the algorithm **does not require** a pre-specified choice of K. Clusters are found **sequentially**.
- Hierarchical clustering is an **iterative procedure** used to build a **hierarchy of clusters**.
- Using a **dendrogram** (a type of tree diagram which highlights the hierarchical relationships among the clusters), hierarchical clustering has the advantage of allowing the analyst to examine alternative partitioning of data of **different granularity** **before** deciding which one to use.

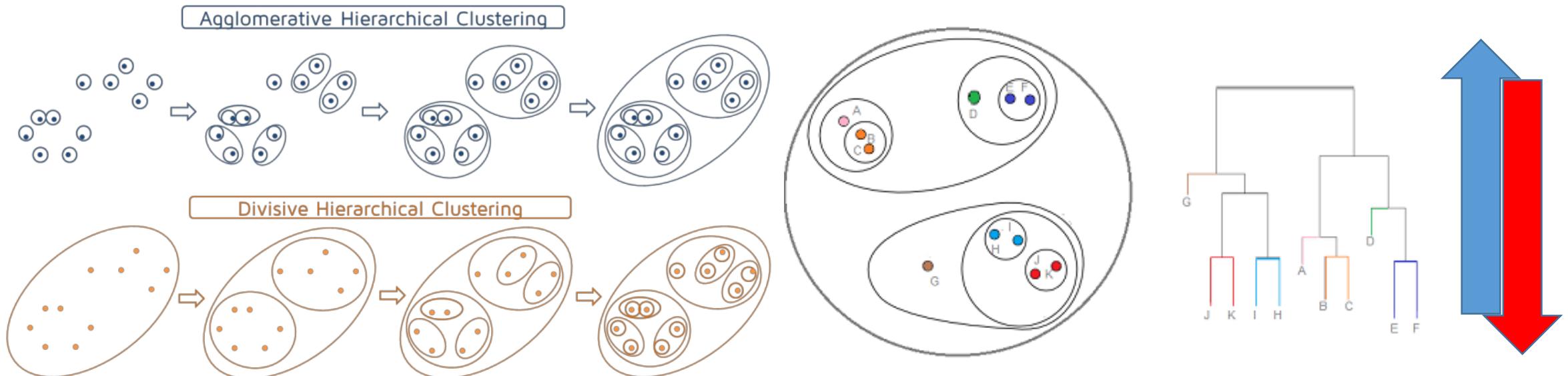


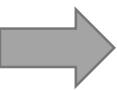
Height in dendrogram is a measure of distance between clusters. longer the height the larger the distance.



Agglomerative (bottom-up) vs Divisive (top-down) HCA

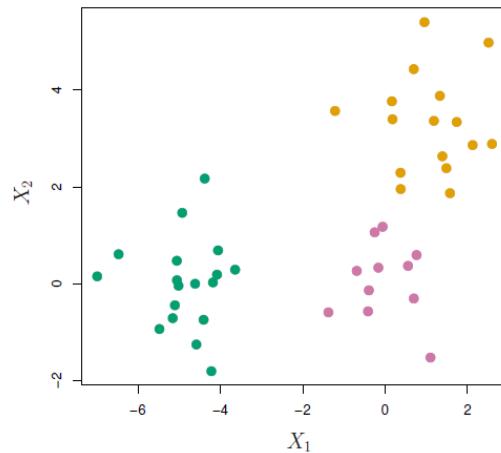
- **Agglomerative**: start with each observation being treated as its own cluster
- **Divisive**: starts with all the observations belonging to a single cluster.



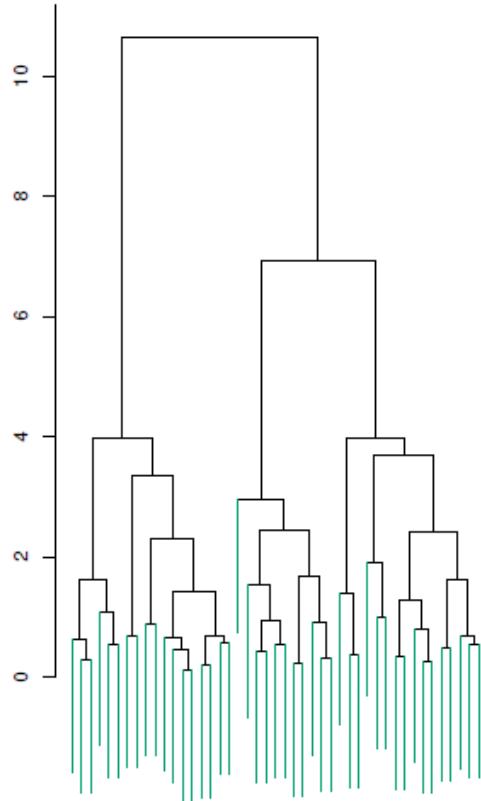


An example

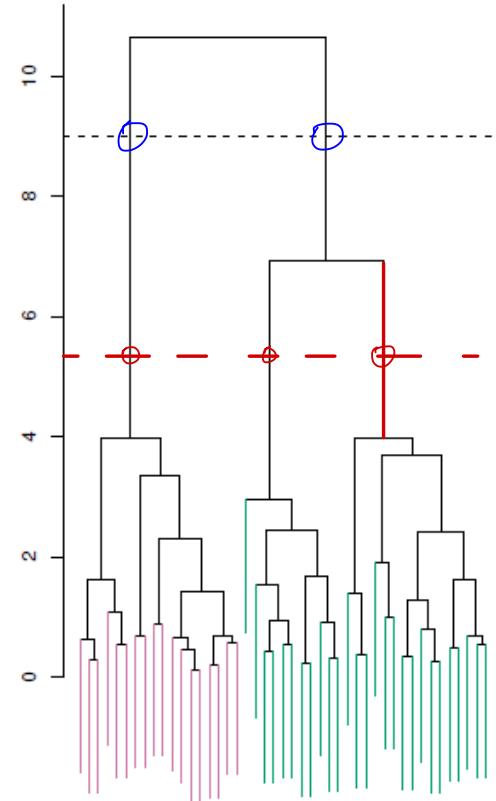
The dotted horizontal line is the threshold that when intersected with height gives no. of clusters. This threshold is your tolerance for total WCV. The rule of thumb to find this threshold is find the longest vertical line that doesn't cut extension of any horizontal line & half it. Shown by Red in this eg.



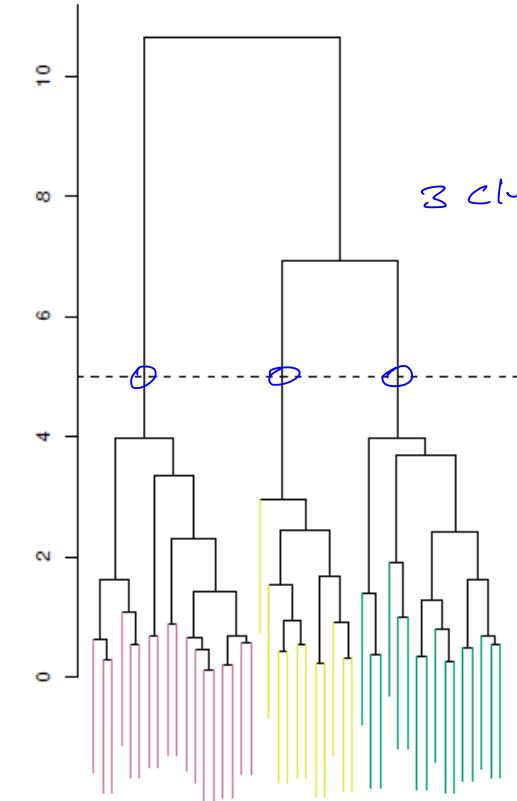
one cluster

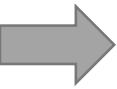


2 Cluster



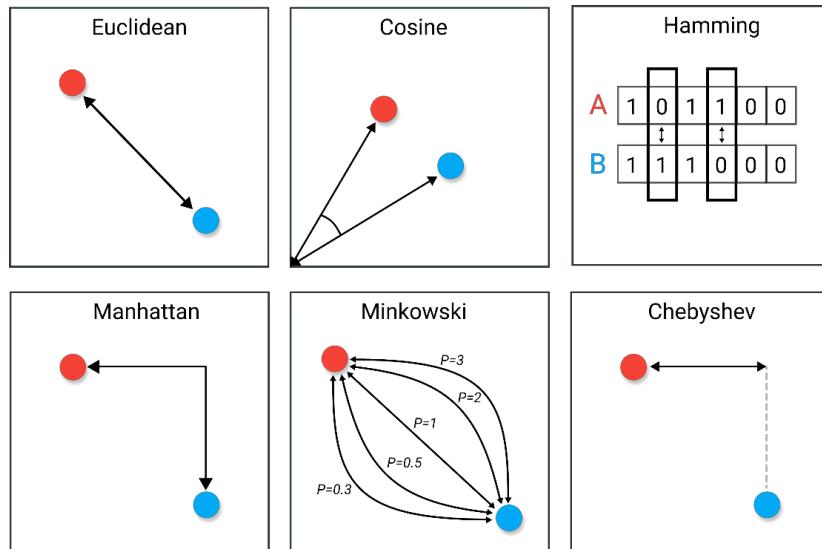
3 cluster





Types of Linkage (distance between two clusters)

- To decide on the **closest clusters**, an explicit definition for the **distance** between two clusters is required (linkage)
- Recall: We have already defined the **within-cluster** distance metrics.



- Single Linkage**

$$D(c_1, c_2) = \min D(x_i, x_j)$$

Minimum distance or distance between closest elements in clusters

- Complete Linkage**

$$D(c_1, c_2) = \max D(x_i, x_j)$$

Maximum distance between elements in clusters

- Average Linkage**

$$D(c_1, c_2) = \frac{1}{|c_1|} \frac{1}{|c_2|} \sum \sum D(x_i, x_j)$$

Average of the distances of all pairs

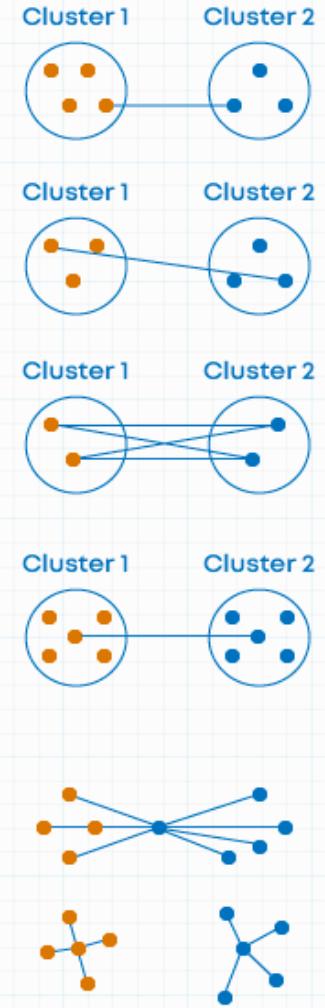
- Centroid Method**

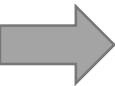
Combining clusters with minimum distance between the centroids of the two clusters

- Ward's Method** (*Mostly Used*)

- Combining clusters where increase in within cluster variance is to the smallest degree.

- Objective is to minimize the total within cluster variance



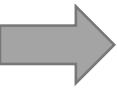


Hierarchical Clustering discussion

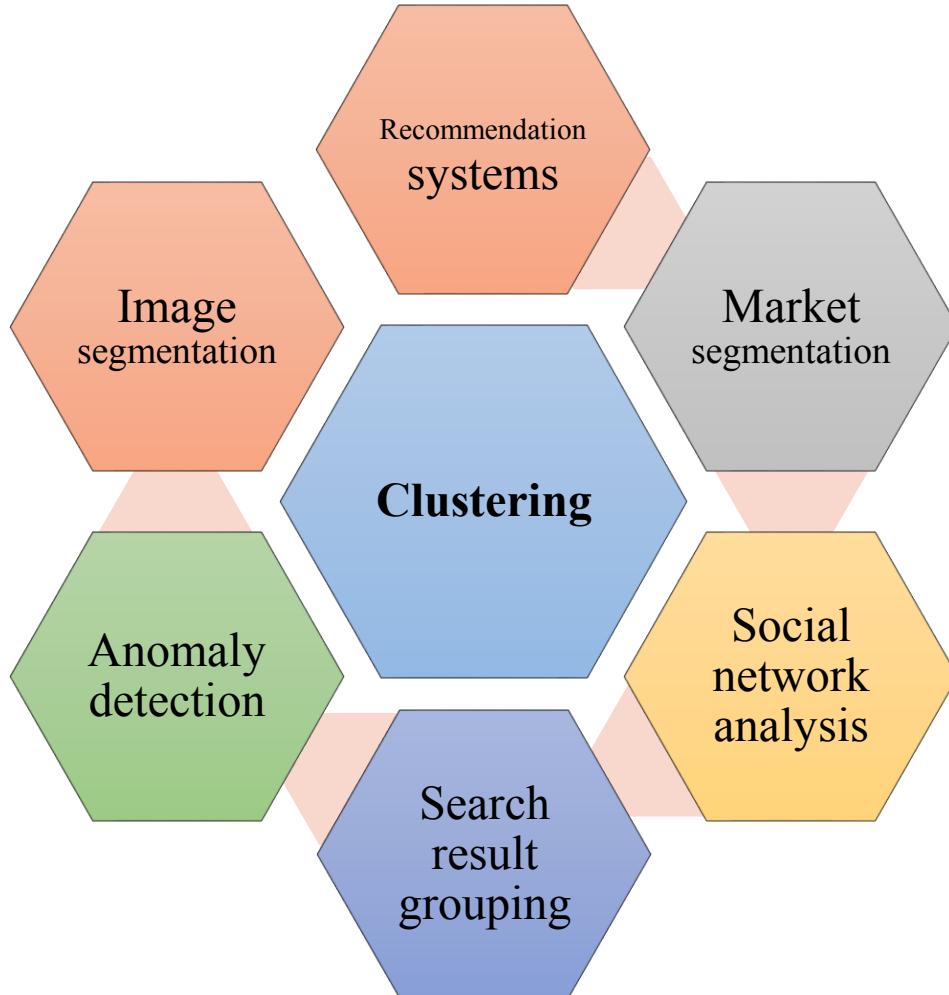
- The **agglomerative** method is the approach typically used with **large** datasets because of the algorithm's fast computing speed.
- The **agglomerative** clustering algorithm makes clustering decisions based on **local** patterns without initially accounting for the global structure of the data. As such, the agglomerative method is well suited **for identifying small clusters**.
- The **divisive** method starts with a **holistic** representation of the data, so it is designed to account for the global structure of the data and thus is better suited **for identifying large clusters**.
- What **dissimilarity measure** should be used?
- What type of **linkage** should be used?
- There is **no commonly agreed-upon** way to decide where to cut the tree.

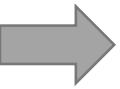
Part III

Applications in finance



Applications of clustering

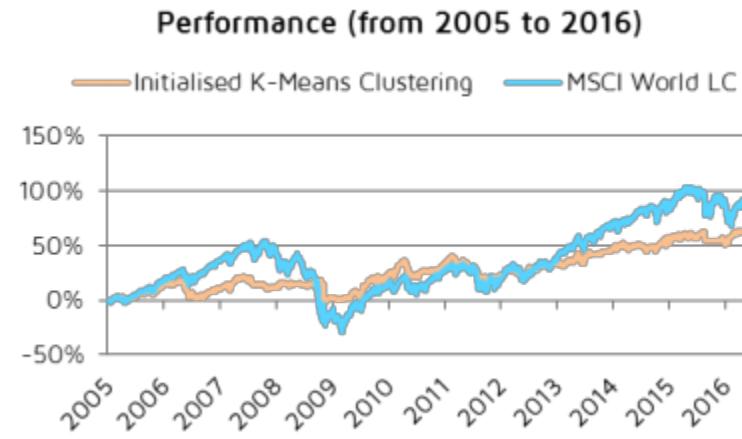
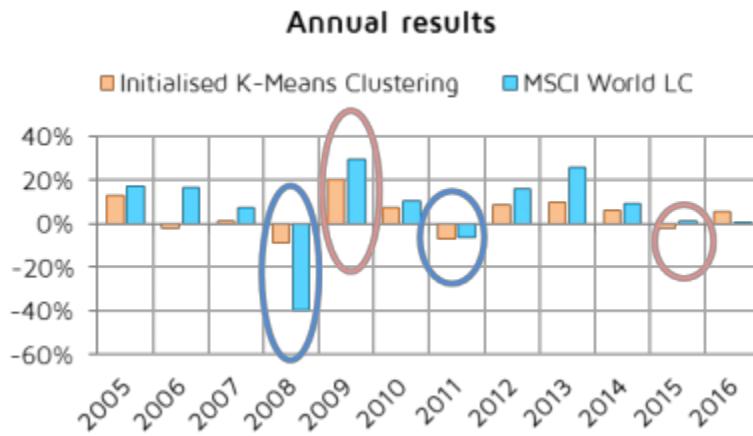


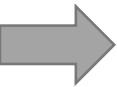


Applications in Finance

- Clustering algorithms are particularly useful in the many investment problems and applications in which the concept of **similarity is important**. *e.g. co-movement of stocks.*
- Applied to grouping companies, for example, clustering may uncover important similarities and differences among companies that are **not captured by standard classifications of companies by industry and sector**.
- In **portfolio management**, clustering methods have been used for improving portfolio diversification by investing in assets from multiple different clusters.

Portfolio constructed by K-means Clustering (orange) compared to MSCI (blue) i.e. index for all large Cap Stocks in the World. Average returns for different years.





Clustering stocks based on co-movement similarity

Exhibit 23 Dataset of Eight Stocks from the S&P 500 Index

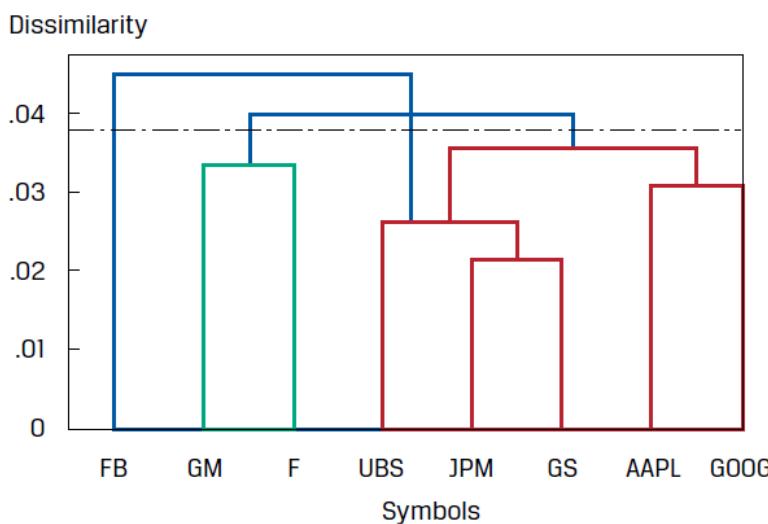
Description: Daily adjusted closing prices of eight S&P 500 member stocks

Trading Dates: 30 May 2017 to 24 May 2019

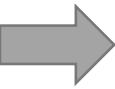
Number of Observations: 501

Stocks (Ticker Symbols): AAPL, F, FB, GM, GS, GOOG, JPM, and UBS

Exhibit 26 Dendrogram for Hierarchical Agglomerative Clustering



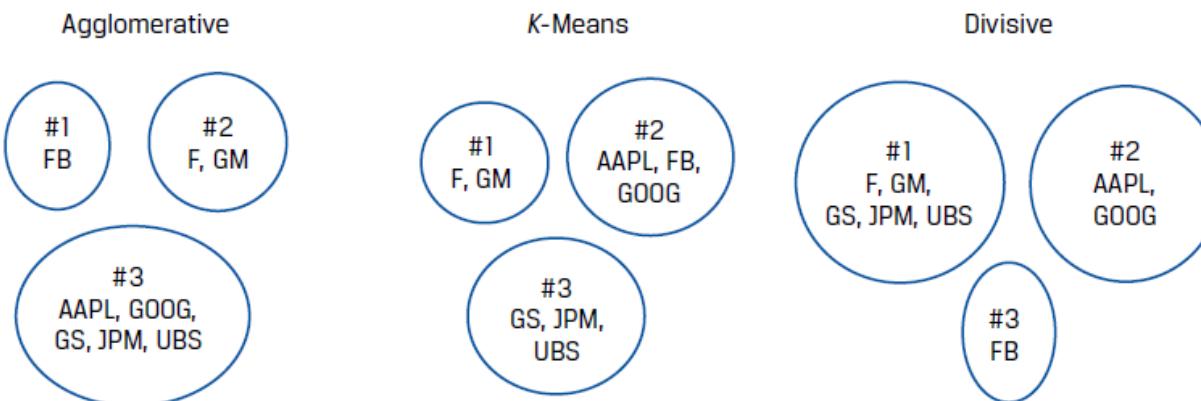
Developed and written by Matthew Dixon, PhD, FRM.



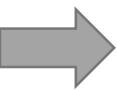
Comparing clustering methods

Exhibit 27 Comparison of Results of Different Clustering Algorithms

	Agglomerative	K-means	Divisive
AAPL	3	2	2
F	2	1	1
FB	1	2	3
GM	2	1	1
GOOG	3	2	2
GS	3	3	1
JPM	3	3	1
UBS	3	3	1



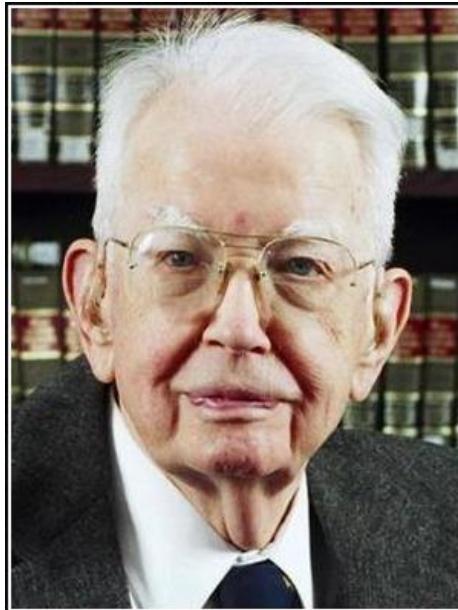
Developed and written by Matthew Dixon, PhD, FRM.



Students' questions

1. How does K-means clustering assist **supervised learning**? Add more linearity to data?
2. I don't know how you would know what situations to use K-mean clustering or when it's better to use hierarchical clustering. *No consensus.*
3. Does clustering lead to false trends in the data? *Yes like any other methods if you make model complex it will overfit.*

↗ No



If you torture the data long enough,
it will confess.

— Ronald Coase —

AZ QUOTES