# Module 5 – Part I
# Machine Learning Fundamentals (review)



Artificial intelligence: Any technique which enables machines to mimic human behavior
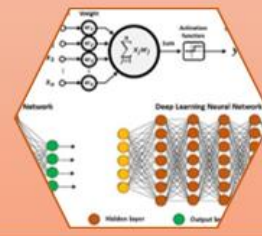
Machine Learning: Subset of AI that enables computers to learn from data. the model is trained with a set of algorithms

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

1950's          1980's          2010's

# Road map!

- Module 1- Introduction to Deep Forecasting
- Module 2- Setting up Deep Forecasting Environment
- Module 3- Exponential Smoothing
- Module 4- ARIMA models
- **Module 5- Machine Learning for Time series Forecasting**
- Module 6- Deep Neural Networks
- Module 7- Deep Sequence Modeling (RNN, LSTM)
- Module 8- Transformers (Attention is all you need!)
- Module 9- Prophet and Neural Prophet

*Pedram Jahangiry*

# Machine Learning Fundamentals

- ML vs traditional programming
- Types of ML
- The Model
- Evaluation metrics
- Bias-Variance tradeoff, overfitting
- Train, Test, Validation
- Resampling methods
- Cost Function
- Solvers/learners (GD, SGD)
- How do machines actually learn?

# 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! The purpose is to generalize.

- This is done by minimizing the loss on the training data.

- The goal is to maximize the performance on the unseen data.
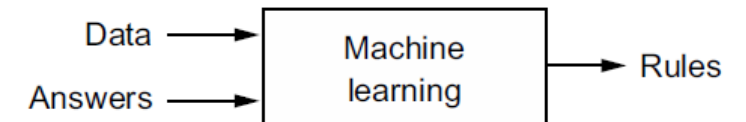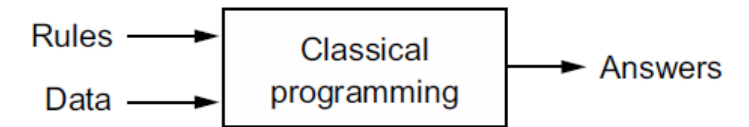
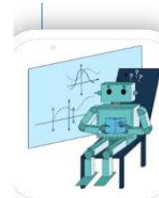# Types of Machine Learning

### Supervised
- Regression
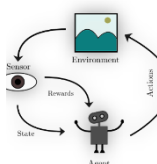- Classification

### Unsupervised
- Clustering
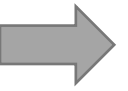- Anomaly detection
- Dimensionality reduction

### Semi-supervised

### Self-supervised

### Reinforcement Learning

Pedram Jahangiry

# The Model

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

y : response, dependent variables, output, Target

X: predictors, independent variables, input, Features

$\theta$: estimates, specifications, Parameters

✓ It is all about estimating $f$ by $\hat{f}$ for two purposes:
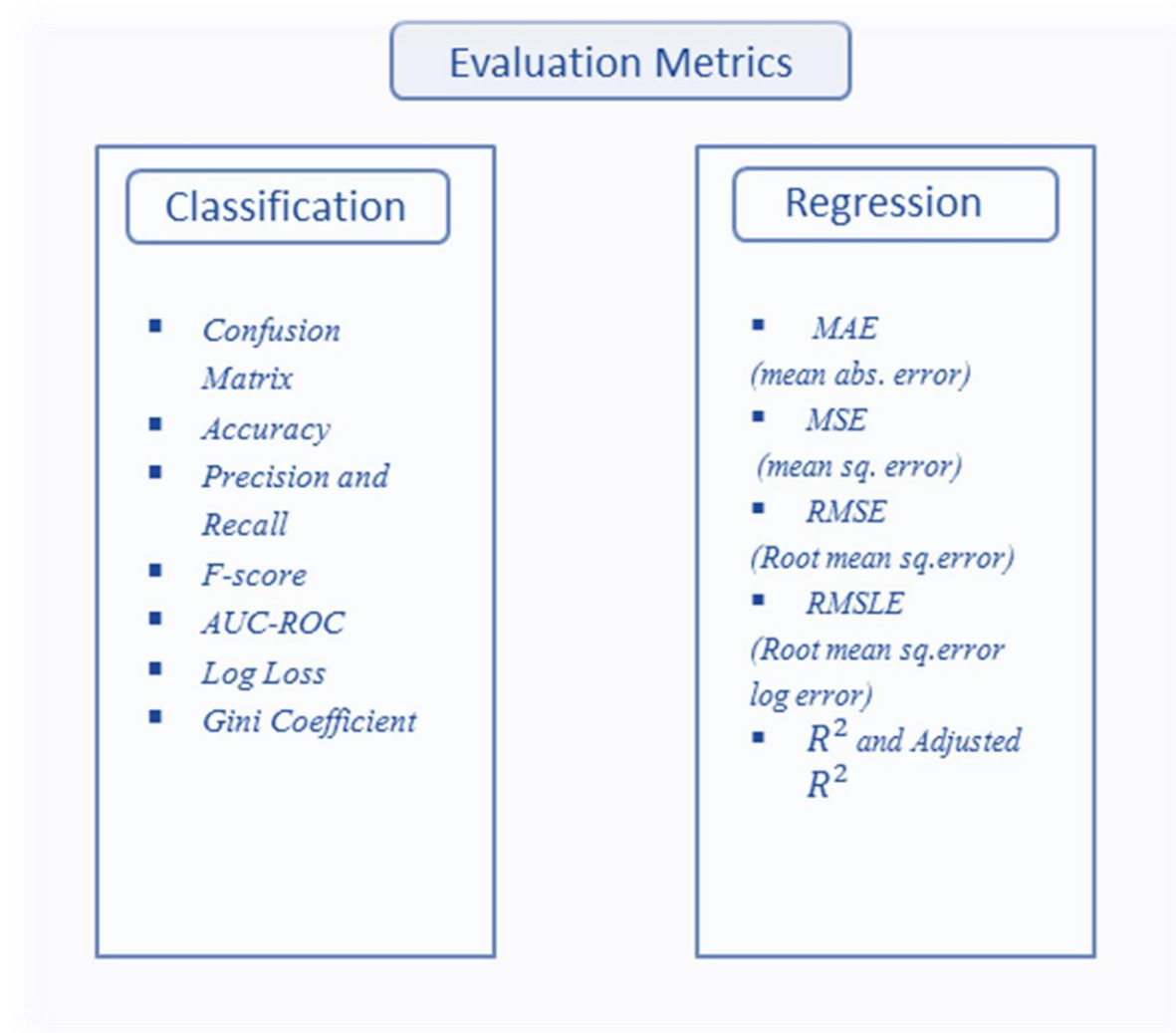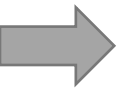
   1)  Inference (interpretable ML)

   2)  Prediction

# Evaluation metrics

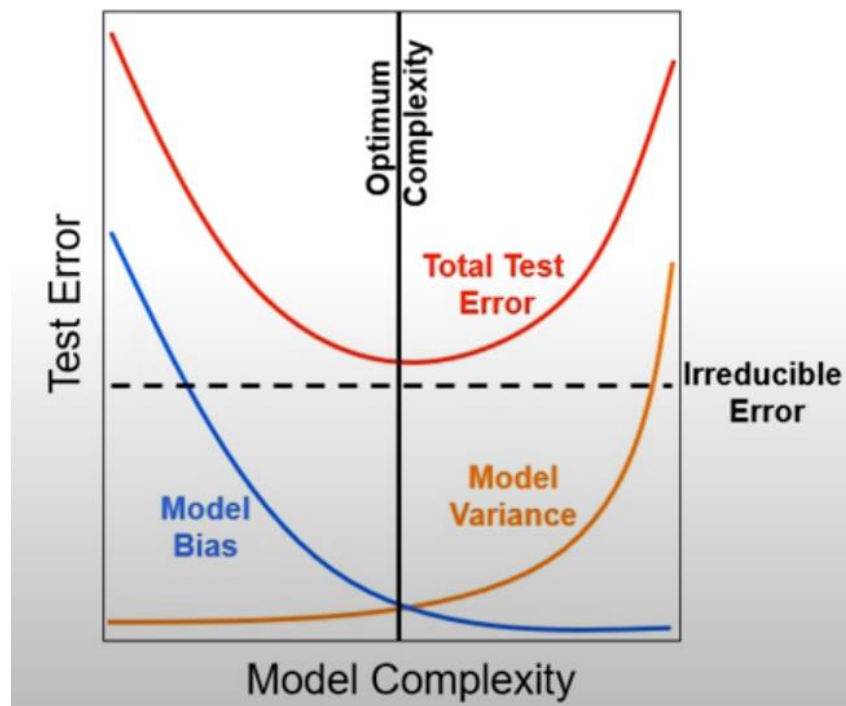In general, we want to compare how close are the predictions to the actual numbers in the test set.

This is typically assessed using
- MSE for quantitative response
- Misclassification rate for qualitative response



**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 $R^2$

# Representations of the bias-variance tradeoff
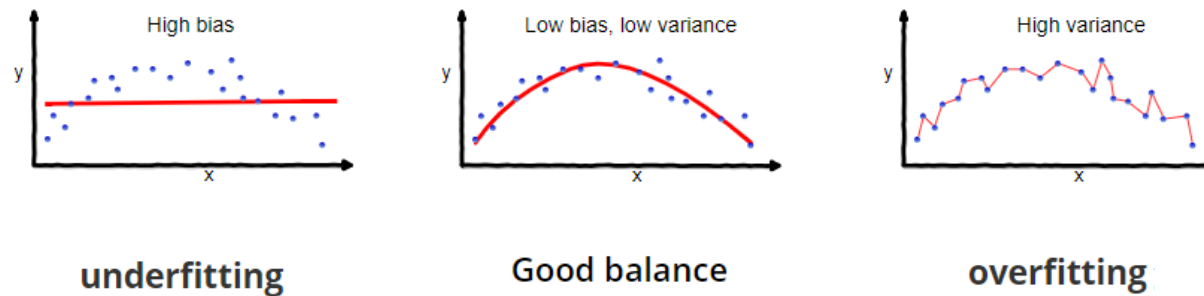


Optimization
Vs
Generalization

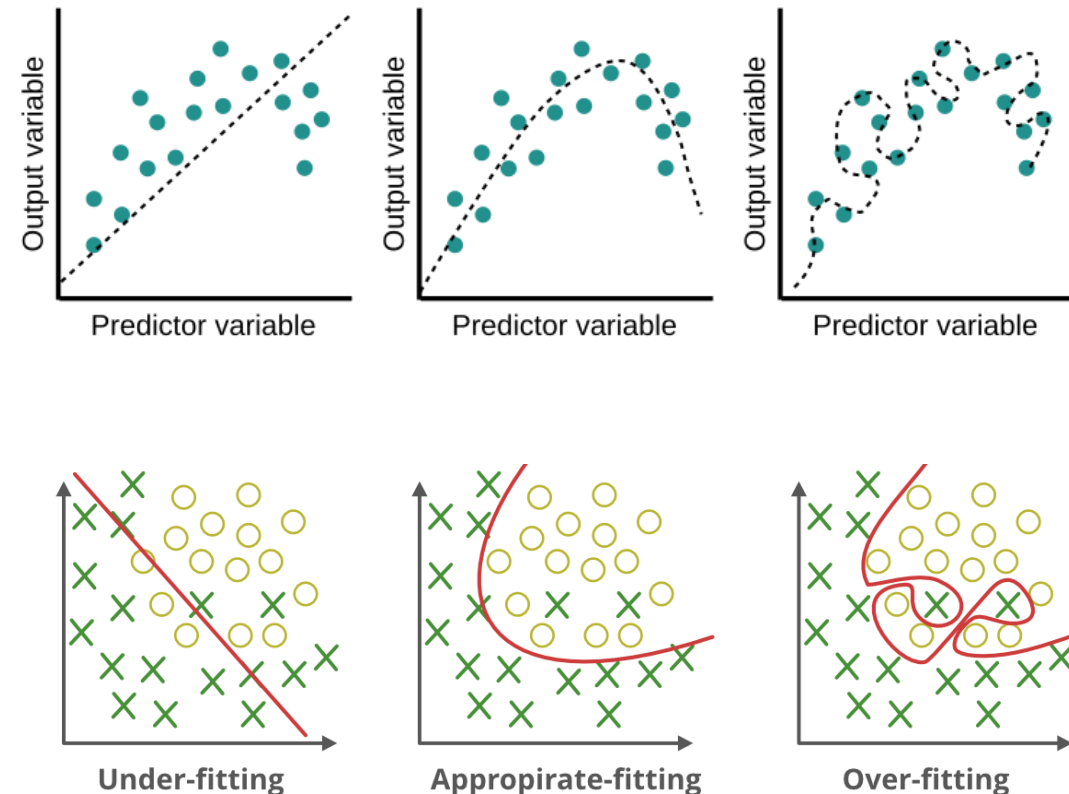# Overfitting

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

- The model fits the training data too well while not predicts well in the new data

- The model fits the noise ($\epsilon$) 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!

# Overfitting



Under-fitting          Appropirate-fitting          Over-fitting
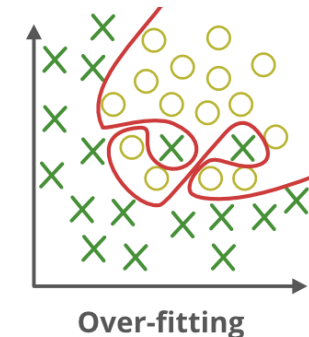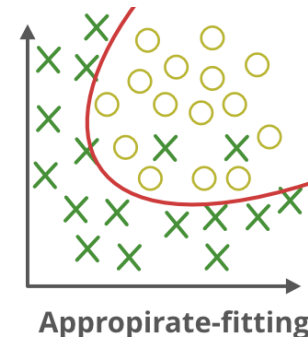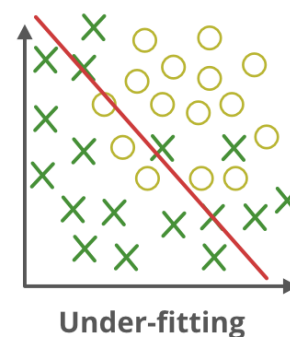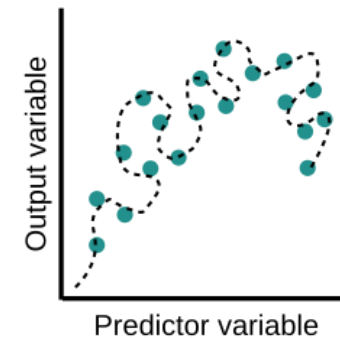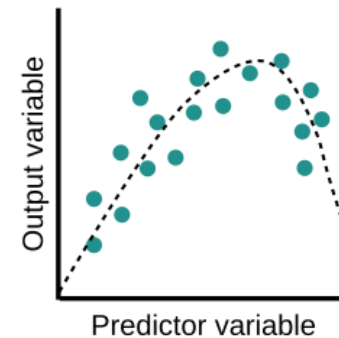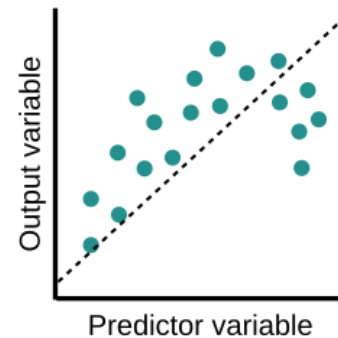
# Mitigate overfitting

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

1) Collect more data (Can reduce bias AND variance)

2) Complexity reduction (regularization, feature selection)

3) Cross validation (estimate the performance in test set)
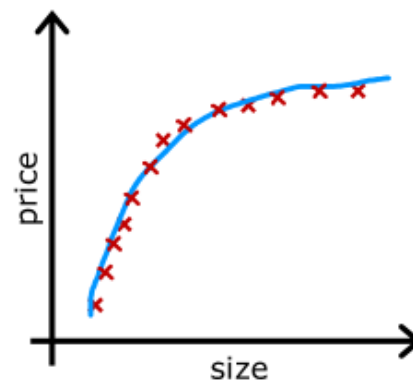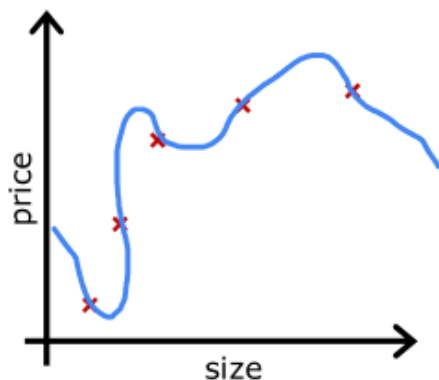


With more training example

# Partitioning of the dataset

The data set is typically divided into three <u>non-overlapping</u> samples:

1) Training set: to train the model

2) Validation set: to validate and tune the model

3) Test set: to test the model's ability to predict well on new data (<mark>generalize</mark>)
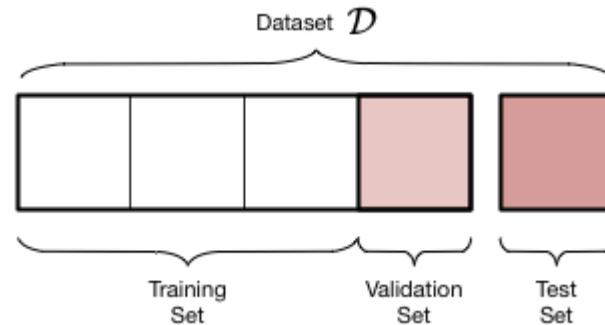


To be <u>valid</u> and <u>useful</u>, any supervised machine learning model must generalize well beyond the training data.

Large dataset is needed! But what if we don't have it?

# Resampling methods

- 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! Unstable model performance in validation set!

- Solution: combining the training and validation sets and use cross validation!
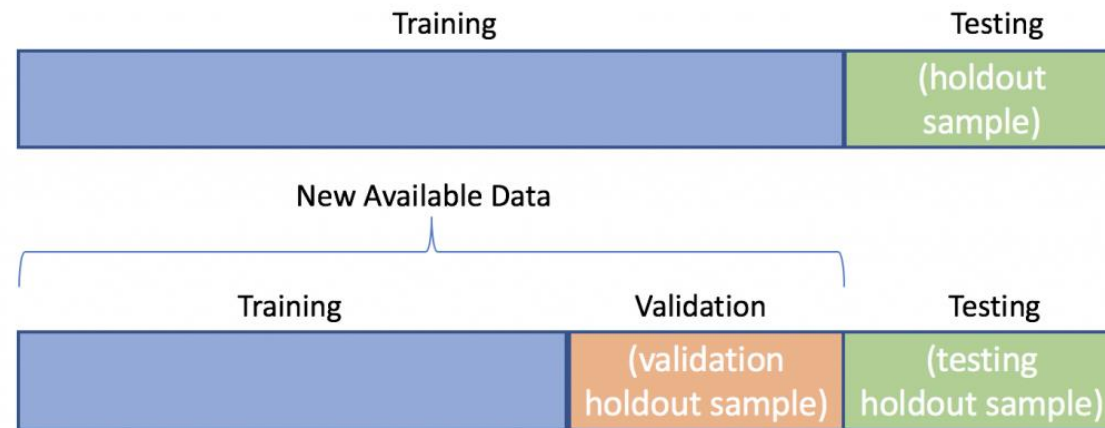
# K-fold Cross Validation

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) This is done in turn for each part $k$ and then the results are combined.
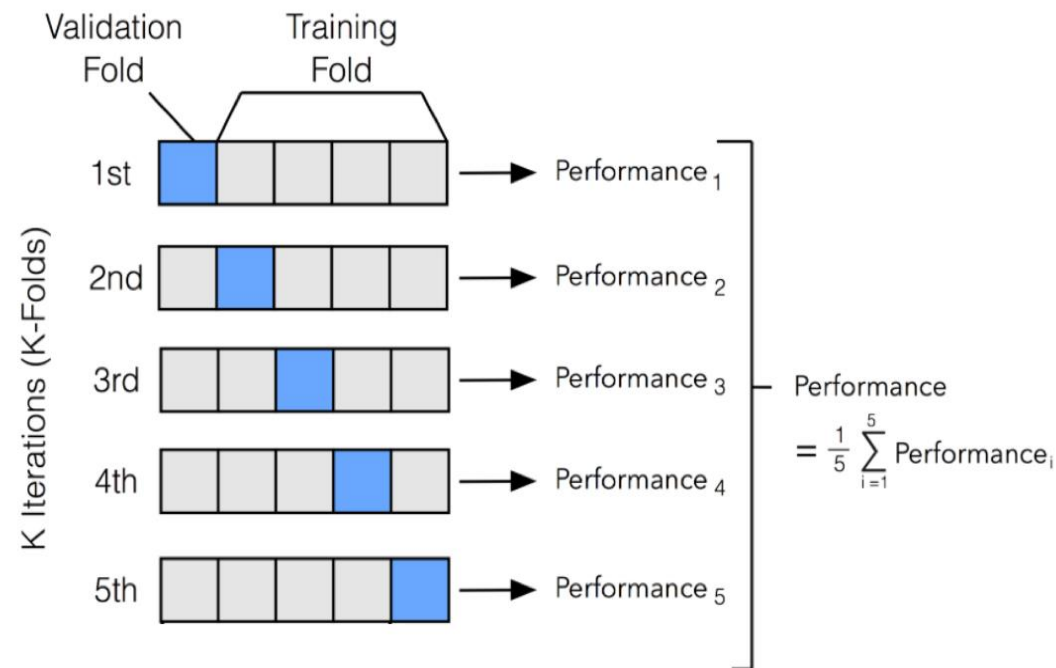
# Why do we use Cross Validation?

Cross validation is mainly used for two purposes:

1. Model architecture selection (optimization vs generalization)

2. Estimation of model performance in the test set

- After selecting the best model architecture, we estimate the generalization error using the test set.

- Different model comparison is based on test set performance!

# The Learning Curve: Do we need to collect more data?

- A learning curve is a plot that shows the relationship between the amount of training data and the performance of a machine learning model.

- It is used to diagnose whether a model has high bias, high variance, or is just right.

-- Training score
-- Cross validation score
-- Benchmark performance
(common sense performance)
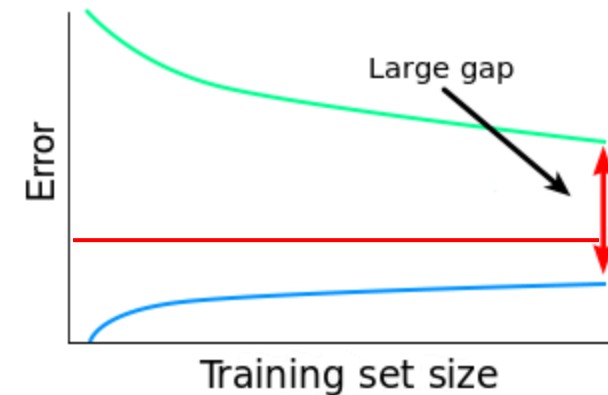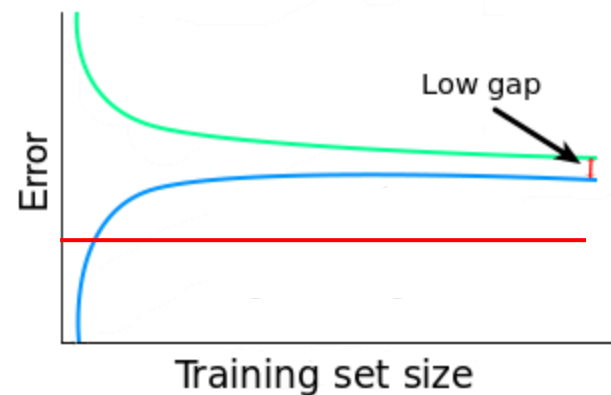
# How do Machines Learn?

# Terminology

- Learning: Finding the model weights (parameters' values)

- Cost 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 cost function is continuous and differentiable are Gradient Descent (GD) and Stochastic GD.
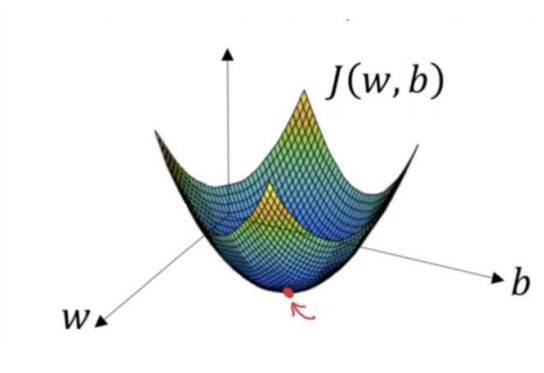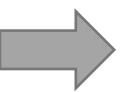
# Solvers (learners)!

- Gradient Descent: is an <mark>iterative</mark> optimization algorithm for finding the minimum of a function.

- We starts at some random point and take steps proportional to the negative of the gradient of the function at the current point.

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

- $\theta_j$ is the model's $j^{th}$ parameter
- $\alpha$ is the learning rate
- $J(\theta)$ is the cost function (which is differentiable)

# Choice of learning rate

- If $\alpha$ is too small, gradient descent can be slow

- If $\alpha$ is too large, the gradient descent can even **diverge**.



**Too low**

$J(\theta)$

$\theta$

A small learning rate requires many updates before reaching the minimum point

**Just right**

$J(\theta)$

$\theta$

The optimal learning rate swiftly reaches the minimum point

**Too high**

$J(\theta)$

$\theta$

Too large of a learning rate causes drastic updates which lead to divergent behaviors
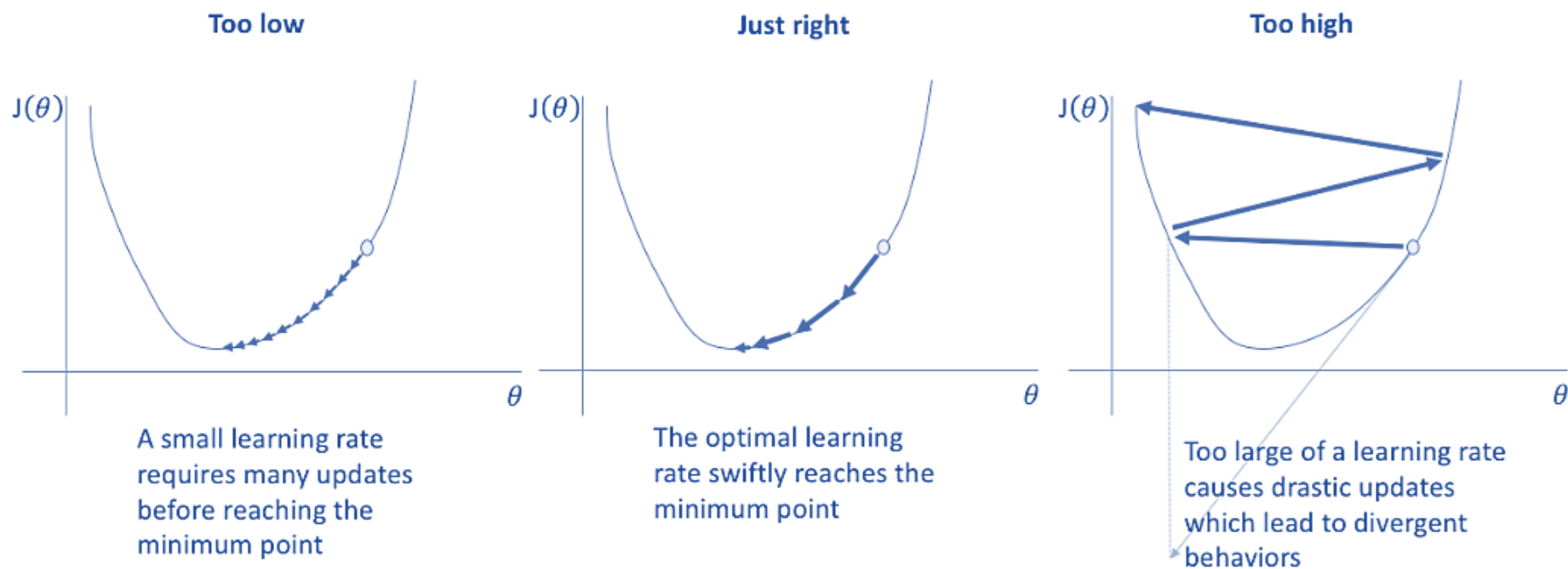
# Beyond Gradient Descent?

Disadvantages of gradient descent:

- Single batch: use the entire training set to update parameters!

- Sensitive to the choice of the learning rate

- Slow for large datasets

(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".

# SGD vs GD
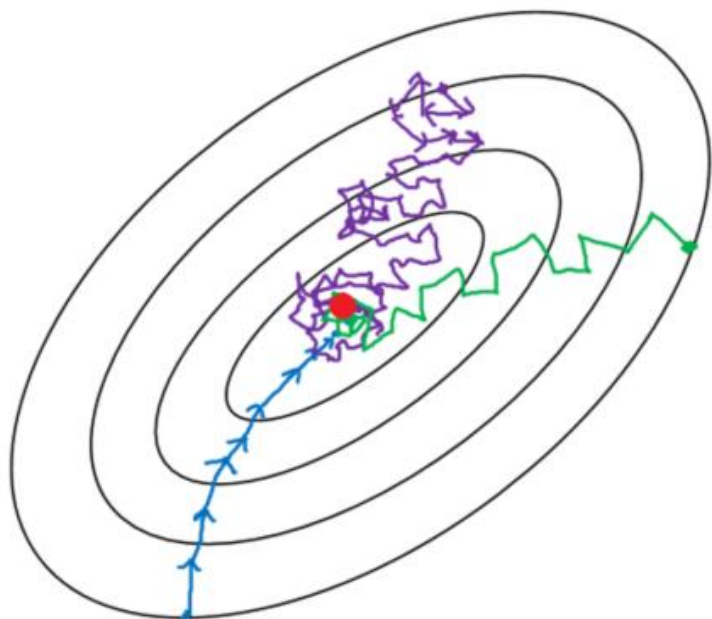


- Batch gradient descent
- Mini-batch gradient Descent
- Stochastic gradient descent
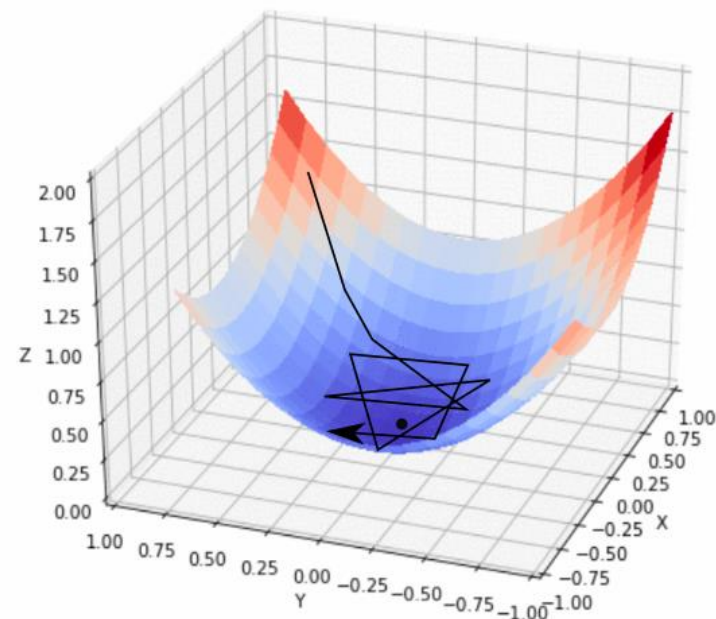
# Beyond SGD?

- Loss functions can be difficult to optimize!

- *Visualizing the loss landscape of neural nets*, Li et all, 2018
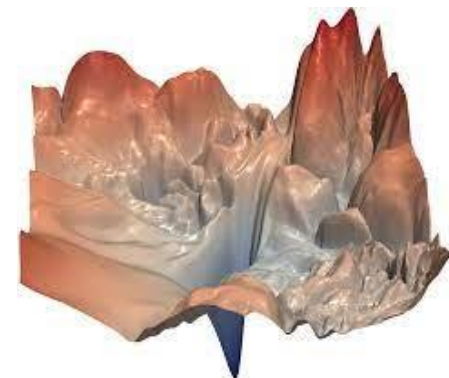


- Solution: Designing an adaptive learning rate that can adapt to the loss landscape.

- Rather than just looking at the current gradient, consider the previous weight updates.

- This is called, momentum!

- Examples: Adam, Adadelta, Adagrad, RMSProp!

# How do machines actually learn?
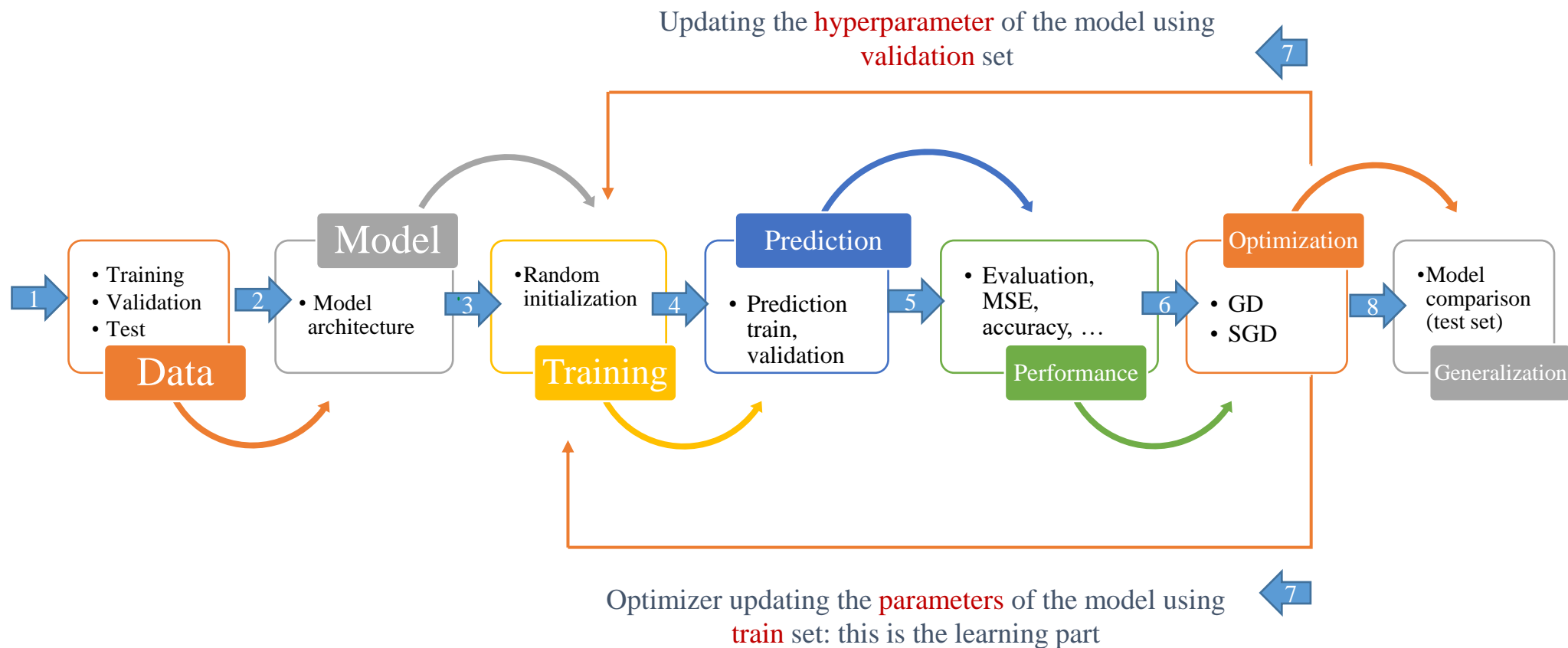
# Module 3 – Part II
# Machine Learning Boosting models

# The modern machine learning landscape

- From 2016 to 2020, the entire machine learning and data science industry has been dominated by these two approaches:
    1. Deep learning
    2. Gradient boosted trees

- Most practitioners of deep learning use Keras, often in combination with its parent framework TensorFlow.

- This means you'll need to be familiar with Scikit-learn, XGBoost, and Keras
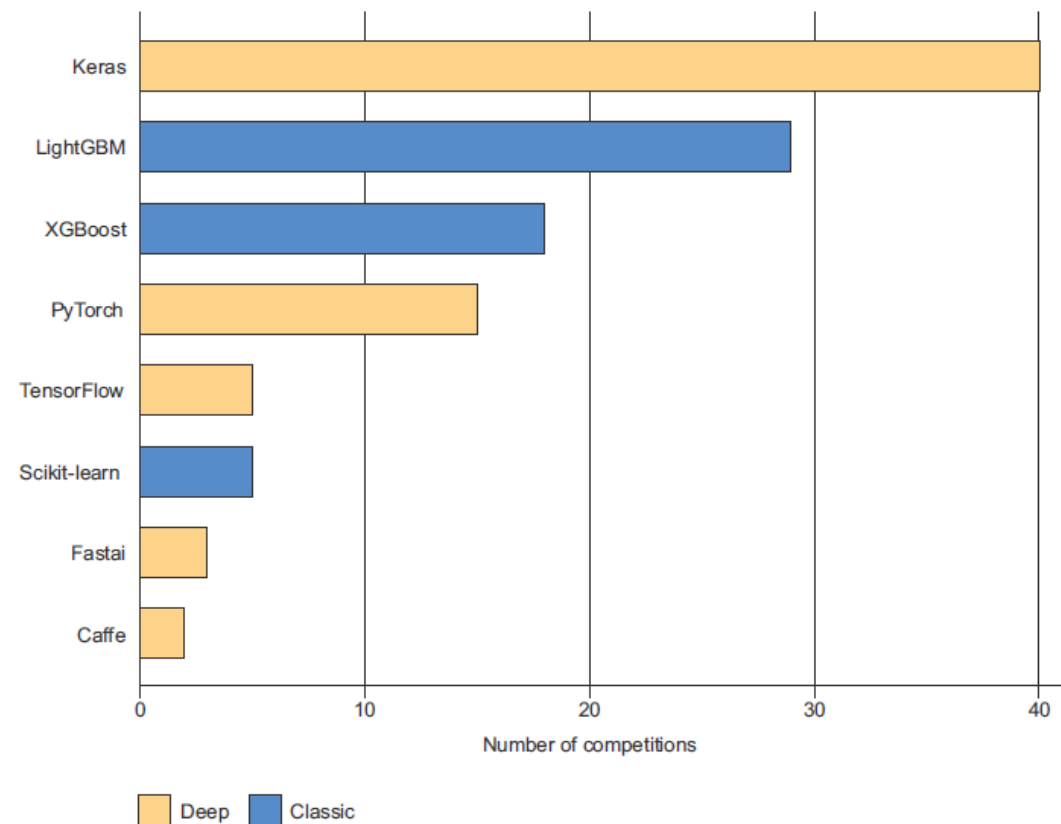


Figure 1.12    Machine learning tools used by top teams on Kaggle
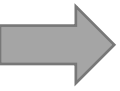
# Decision Trees Fundamental questions

- Four fundamental questions to be answered:
1) What feature and cut off to start with?
2) How to split the samples?
3) How to grow a tree?
4) How to combine trees?
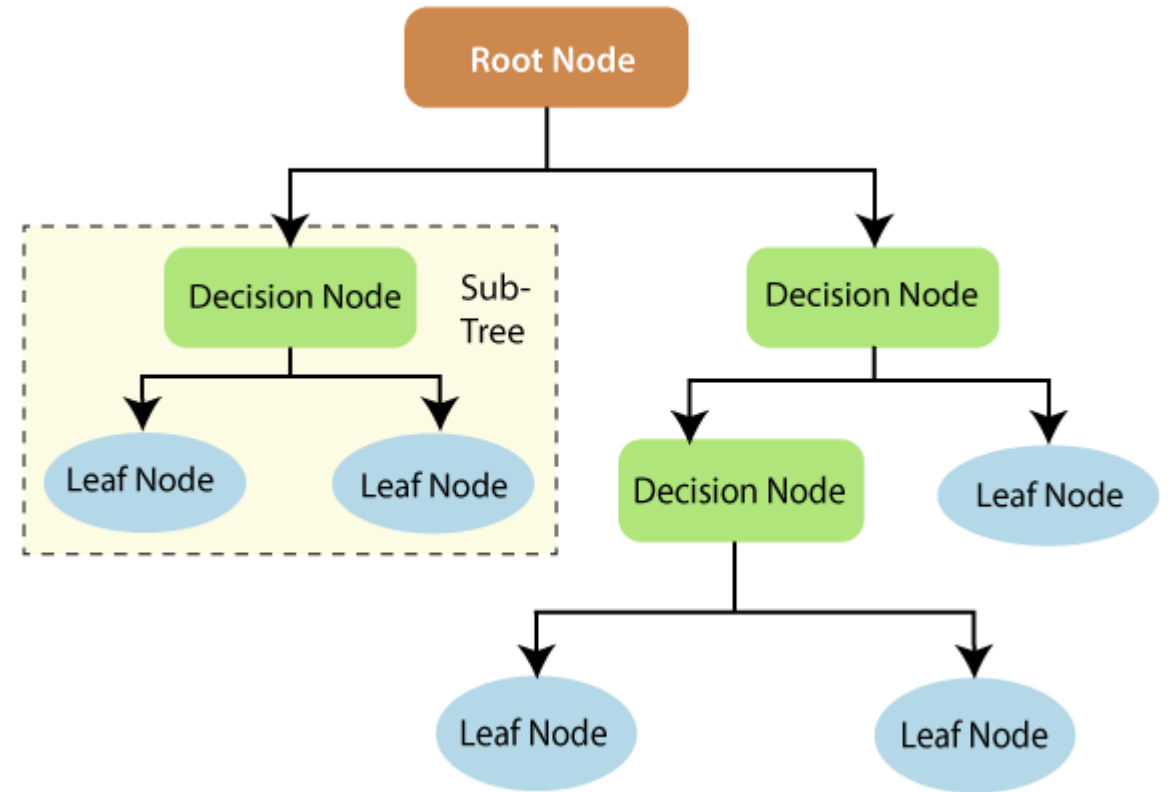
# What feature and cut off to start with?

- Which feature and cut off adds the most information gain (minimum impurity)?

  - Regression trees: MSE

  - Classification trees:

    1. Error rate

    2. Entropy

    3. Gini Index
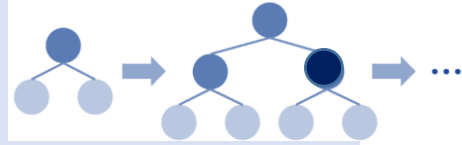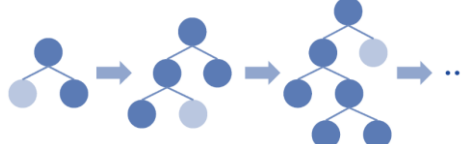
Control how a Decision Tree decides to split the data
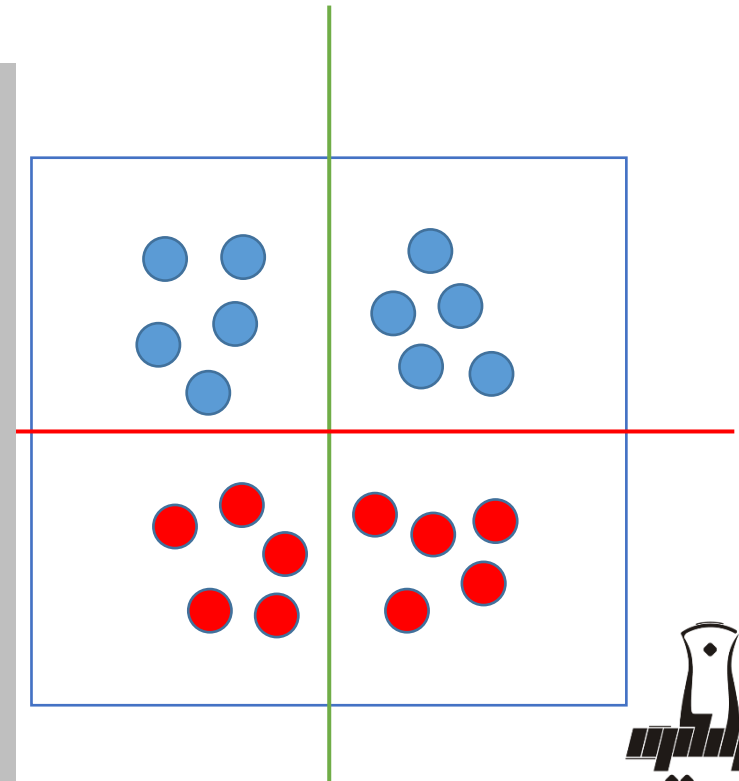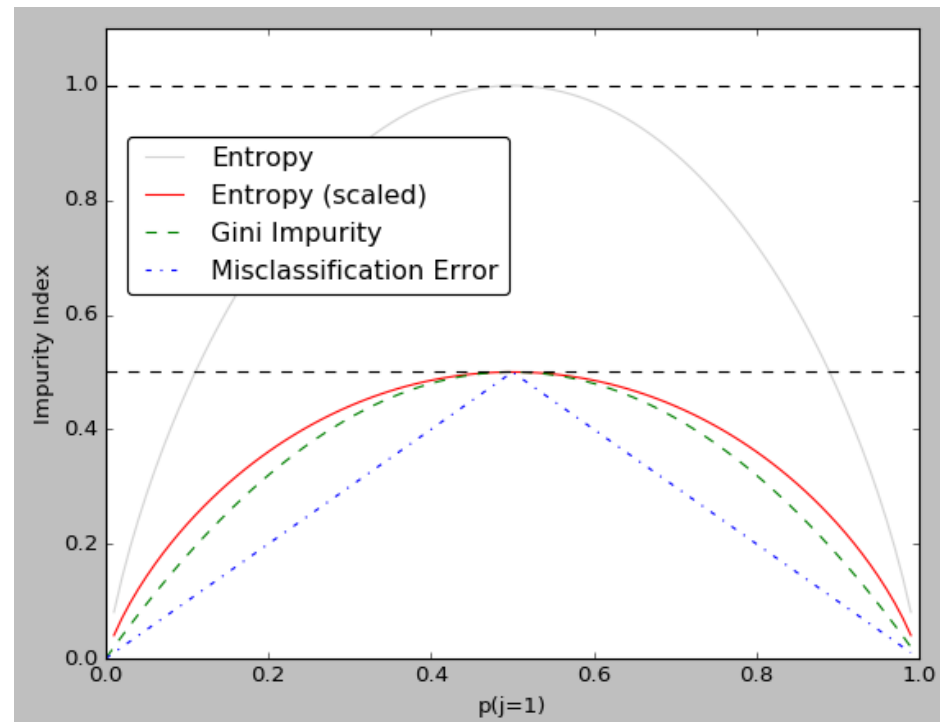
# How to split the samples?

| Method | Description |
| --- | --- |
| Pre-sorted and histogram based | This method sorts the data and creates histograms of the values before splitting the tree. This allows for faster splits but can result in less accurate trees. |
| GOSS (Gradient-based One-Side Sampling) | This method uses gradient information as a measure of the weight of a sample for splitting. Keeps instances with large gradients while performing random sampling on instances with small gradients. |
| Greedy method | This method selects the best split at each step without considering the impact on future splits. This method May result in suboptimal trees |

# How to grow a tree?

| Algorithm | Description | |
|---|---|---|
| Depth-Wise Level-Wise | This strategy grows the tree one level at a time, ensuring all nodes at the current level are fully expanded before moving on to the next. This results in a balanced tree structure. | |
| Leaf-wise | This strategy, rather than growing by levels, focuses on expanding the tree by adding nodes to the leaves, specifically those that result in the highest decrease in impurity or error. his can lead to a more unbalanced tree but potentially more efficient learning. | |
| Symmetric | This strategy attempts to maintain balance not just in the depth of the tree but also in how the features are split, aiming for a tree that grows evenly across all paths. | |

All the methods, repeatedly split the data along the feature with the highest information gain

# How to combine trees?

- 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.

- 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 tries to "fix" the errors which previous models make.

- In boosting, each tree is grown using information from previous tree.


Bagging

Parallel


Boosting

Sequential

Pedram Jahangiry

# Evolution of XGBoost



Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism

**Bagging**

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

**Boosting**

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias

**XGBoost**

**Decision Trees**

A graphical representation of possible solutions to a decision based on certain conditions

**Random Forest**

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

**Gradient Boosting**

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

# XGBoost: eXtreme Gradient Boosting

- XGBoost is an open-source gradient boosting library developed by Tianqi Chen (2014) focused on developing efficient and scalable machine learning algorithms.

- 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.

- XGBoost includes several other features that can improve model performance, such as handling missing values, automatic feature selection, and model ensembling.



dmlc
**XGBoost**

Cache awareness and out-of-core computing

Regularization for avoiding overfitting

Tree pruning using depth-first approach

Efficient handling of missing data

Parallelized tree building

In-built cross-validation capability
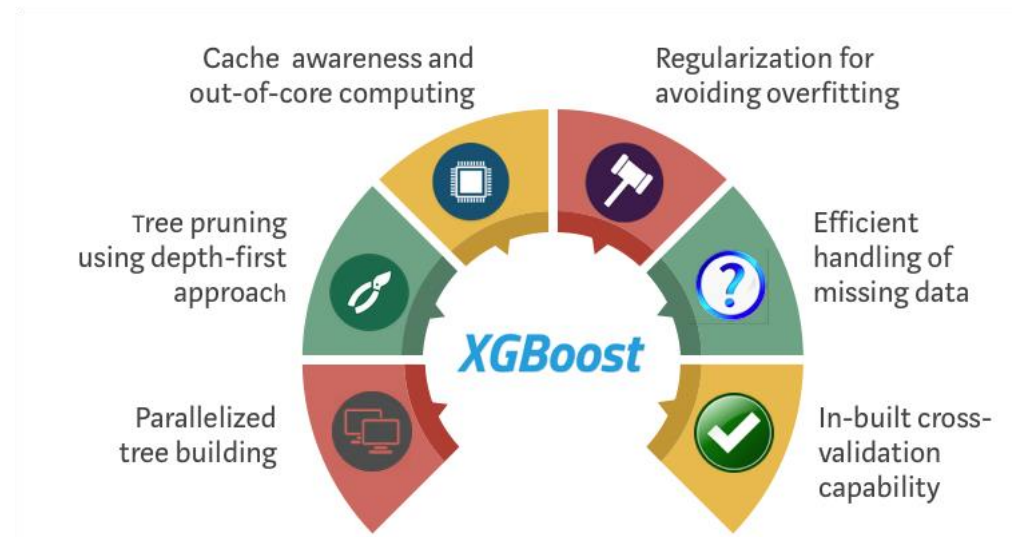
# LightGBM (Light Gradient Boosted Machine)

- LightGBM is an open-source gradient boosting library developed by Microsoft (2016) that is fast and efficient, making it suitable for large-scale learning tasks.

- LightGBM can handle categorical features, but requires one-hot encoding, ordinal encoding or other preprocessing

- LightGBM includes several other features that can improve model performance, such as handling missing values, automatic feature selection, and model ensembling.

# CatBoost (Category Boosting)

- CatBoost is an open-source gradient boosting library developed by Yandex (2017) that is specifically designed to handle categorical data.

- CatBoost can handle categorical features directly, without the need for one-hot encoding or other preprocessing.

- CatBoost includes several other features that can improve model performance, such as handling missing values, automatic feature selection, and model ensembling.

# XGBoost vs LightGBM vs CatBoost

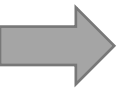| | **XGBoost** | **LightGBM** | **CatBoost** |
|---|---|---|---|
| Developer | Tianqi Chen (2014) | Microsoft (2016) | Yandex (2017) |
| Base Model | Decision Trees | Decision Trees | Decision Trees |
| Tree growing algorithm | Depth-wise tree growth Leaf-wise is also available | Leaf-wise tree growth | Symmetric tree growth |
| Parallel training | Single GPU | Multiple GPUs | Multiple GPUs |
| Handling categorical features | Encoding required (one-hot, ordinal, target, label, …) | Automated encoding using categorical feature binning | No encoding required |
| Splitting method | Pre-sorted and histogram based | GOSS and histogram based | Greedy method |

dmlc
**XGBoost**

LightGBM

CatBoost

JON M. HUNTSMAN
SCHOOL OF BUSINESS
UtahStateUniversity

Pedram Jahangiry

# Module 3 – Part III
# Challenges in Time Series Machine Learning

# Challenges in Time Series Machine Learning

Non-stationarity

Seasonality

Time series challenges

Limited data
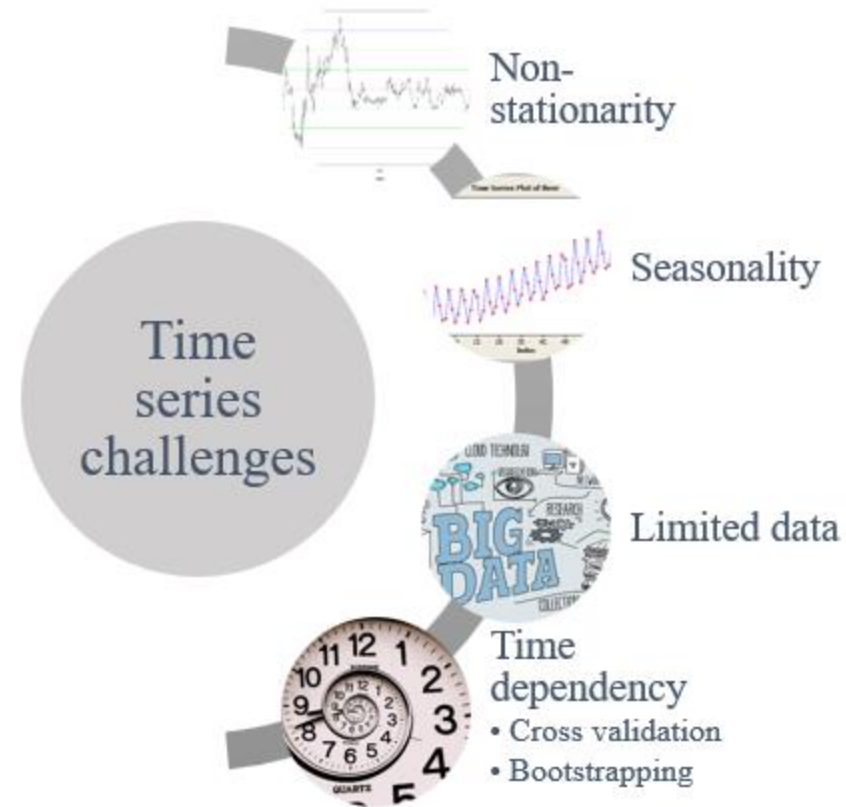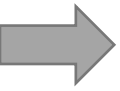
Time dependency
• Cross validation
• Bootstrapping
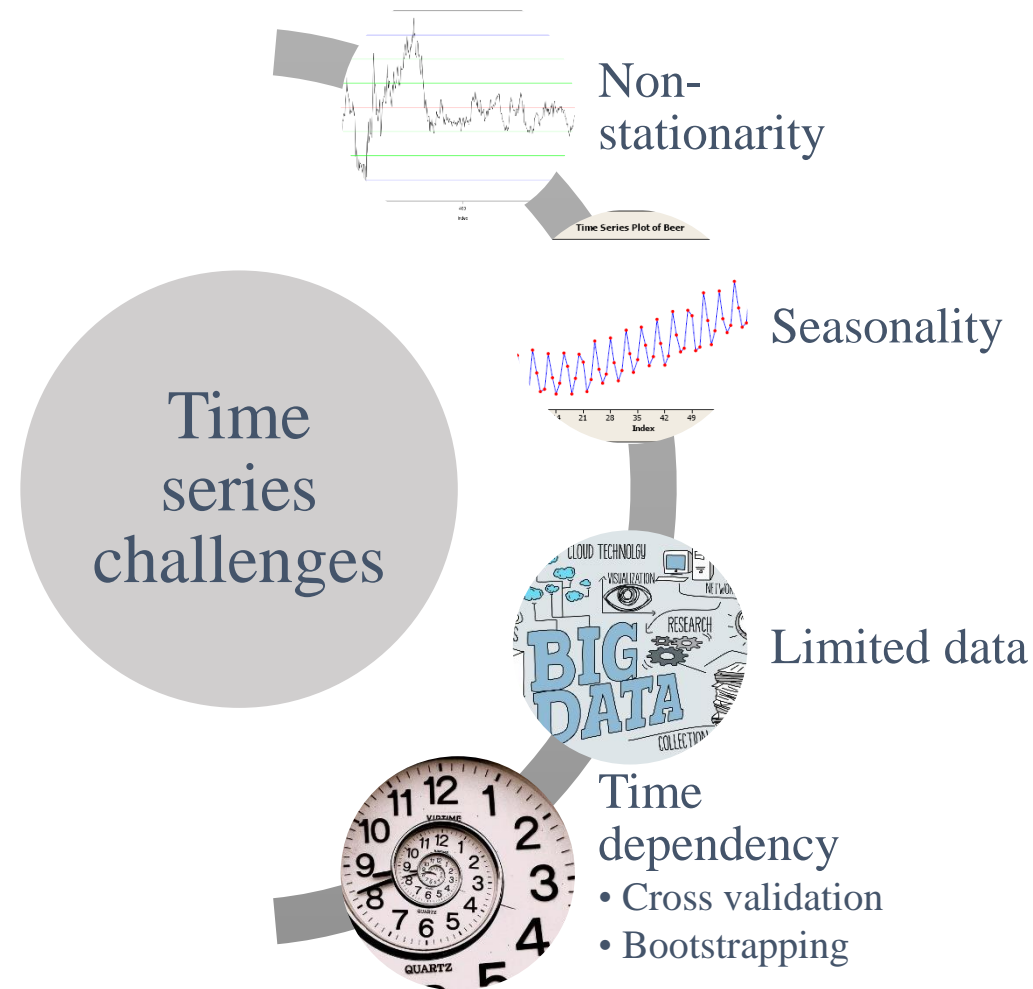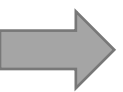
# Stationarity

- Stationary vs Non-Stationary Data. What makes a data set Stationary?

- In a stationary timeseries, the statistical properties do not depend on the time



| Stationary | Non-Stationary | Non-Stationary | Non-Stationary |
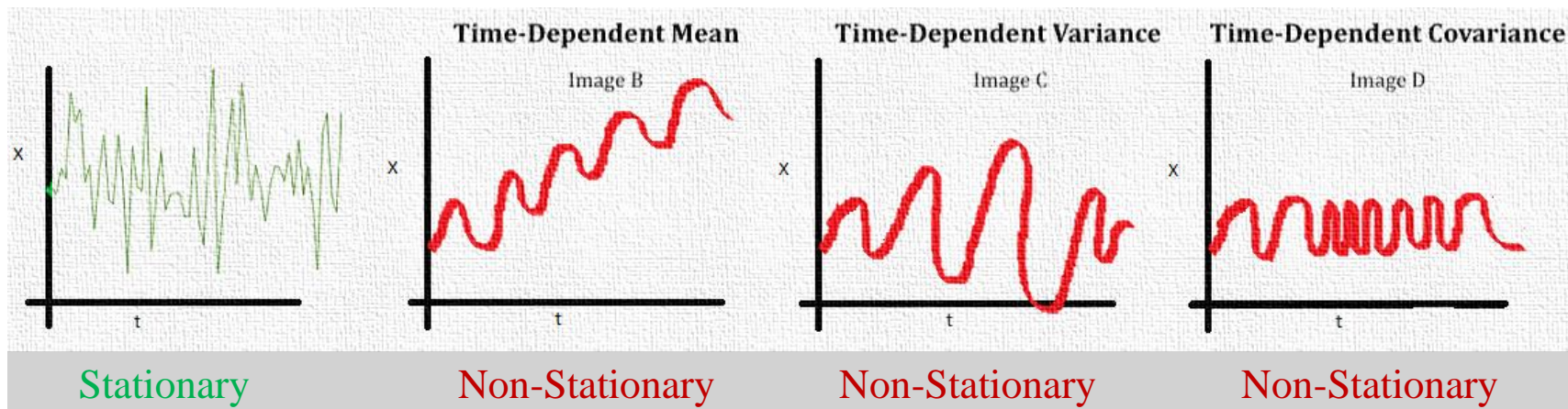
- Data with trend and seasonality are NOT stationary!

# Time Series Cross Validation

- With time series data, we **cannot shuffle** the data! TS data is not IID.

- We also need to avoid **data leakage**!

- The main time series CV methods are:

  1) **Purged** K-Fold CV

  2) Walk forward rolling / expanding window

  3) Combinatorial purged CV

# Purged K-Fold CV

- **Leakage** takes place when the training set contains information that also appears in the testing set.

- Leakage will enhance the model performance

- Solution: Purging and Embargoing

- <u>Purged K-Fold CV</u>: Adding purging and embargoing whenever we produce a train/test split in K-Fold CV.
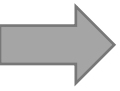


| Iteration 1 | Test | Train | Train | Train | Train |
| Iteration 2 | Train | Test | Train | Train | Train |
| Iteration 3 | Train | Train | Test | Train | Train |
| Iteration 4 | Train | Train | Train | Test | Train |
| Iteration 5 | Train | Train | Train | Train | Test |
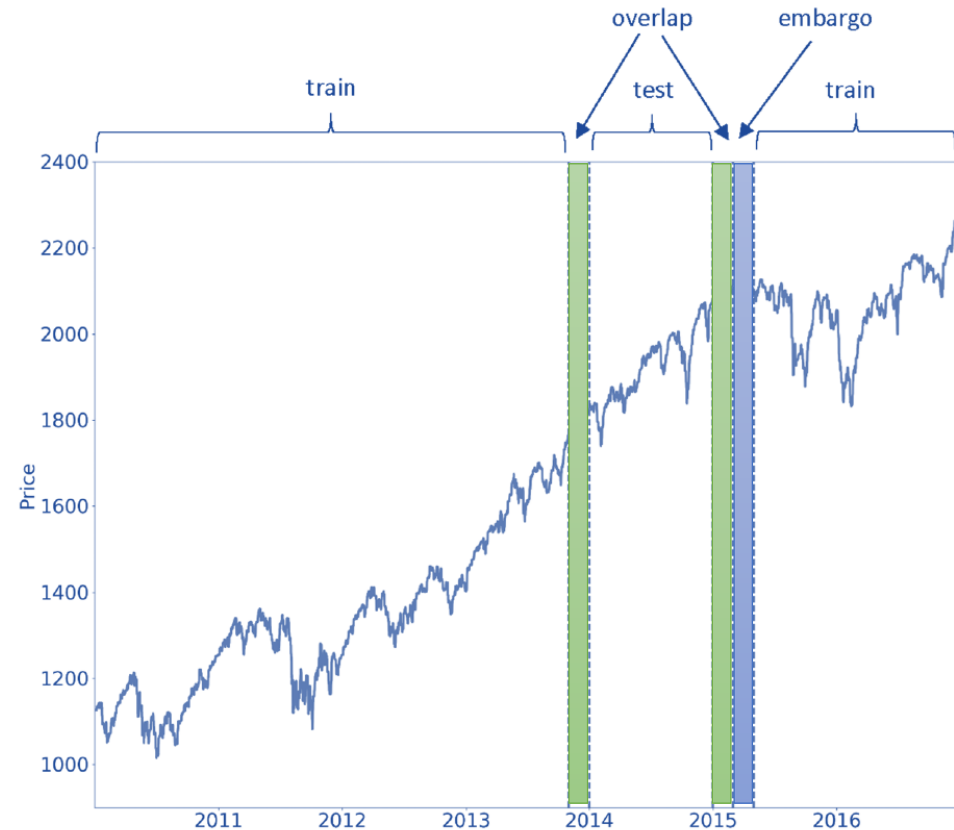


**FIGURE 7.3** Embargo of post-test train observations

# Walk Forward Cross Validation
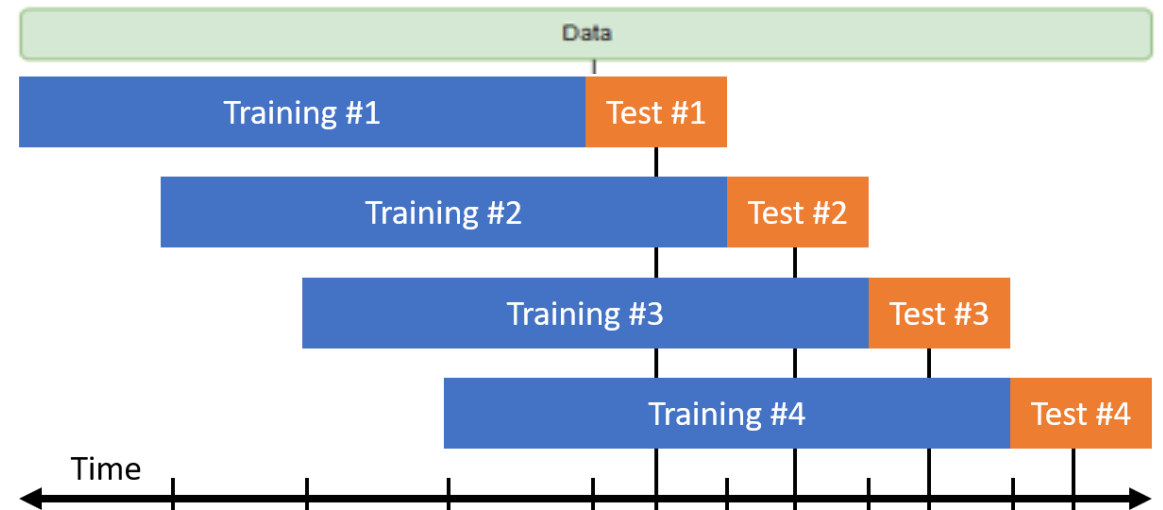
# Combinatorial Purged Cross Validation (CPCV)

- The goal is to generate multiple unique back-test path that span the entire data set.

- In each path, we can look at the model's OOS performance for the entire time period.
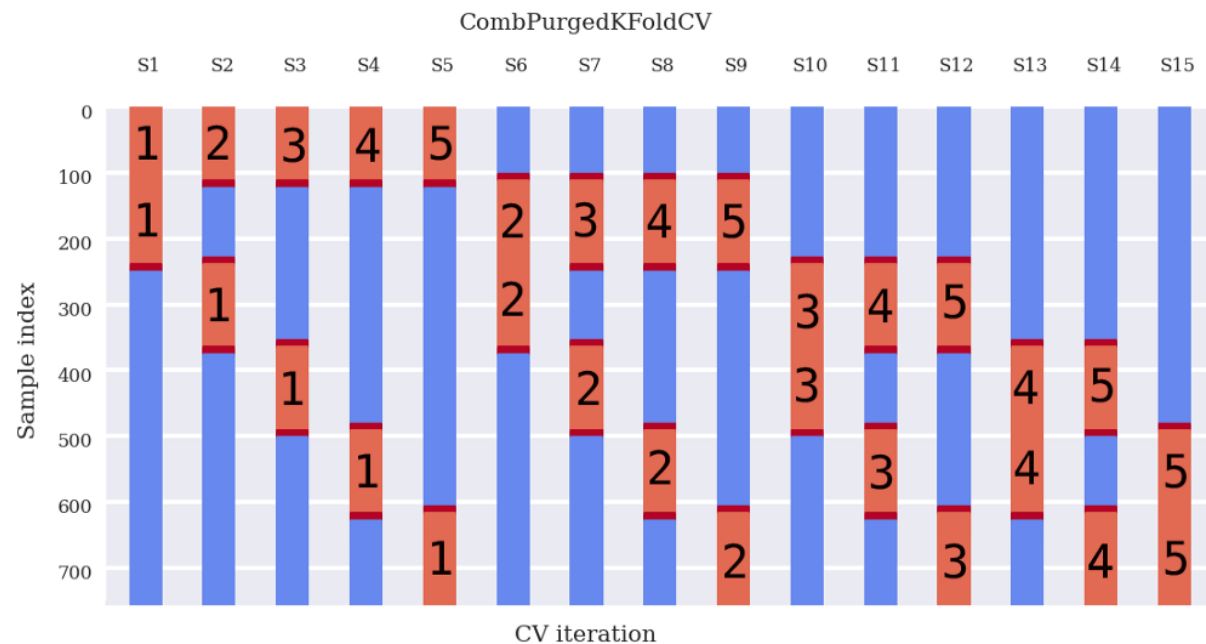
# Time Series Bootstrapping

- IID bootstrapping (random sample with replacement) does not work for time series data with temporal dependency.

- Time series Bootstrapping methods:

  - Parametric (based on models with iid residuals and resampling from residuals. Example: ARIMA bootstrap)

  - Non-parametric block bootstrap (data is directly resampled. Assumption: blocks can be samples so that they are approximately iid)

    - Moving Block Bootstrap (MBB)

    - Circular Block Bootstrap (CBB)

    - Stationary Bootstrap (SB)

# Moving Block Bootstrap (MBB)

- Moving Block Bootstrap, samples overlapping fixed size blocks of m consecutive observations.

- Blocks starts at indices 1, … , T-m+1
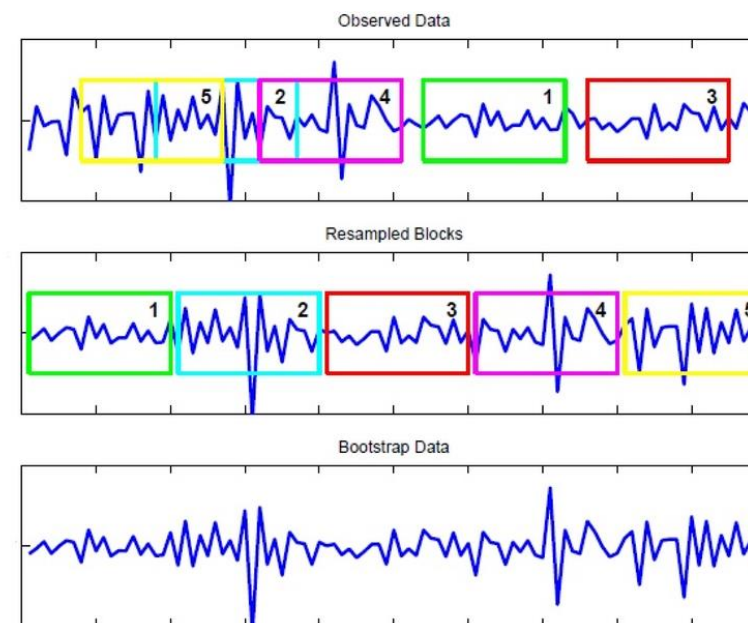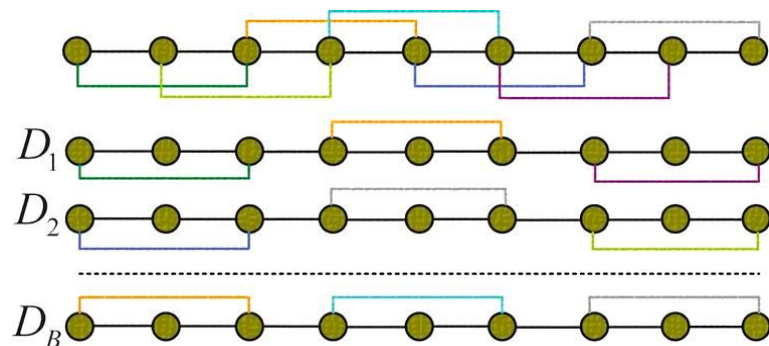
# Circular Block Bootstrap (CBB)

- CBB is a simple extension of MBB which assumes the data live on a circle so that $y_{T+1} = y_1$, $y_{T+2} = y_2$, etc.

- CBB has better finite sample properties since all data points get sampled with equal probability.
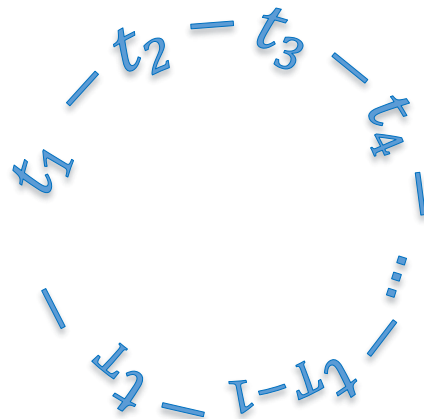
# Stationary Bootstrap (SB)

- In SB, the block size is no longer fixed.

- Chooses an average block size of m rather than an exact block size.

- Popularity of SB stems from difficulty in determining optimal m

- Once applied to stationary data, the resampled pseudo time series by SB are stationary. This is not the case for MBB and CBB.
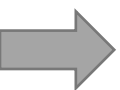
# Preparing data for machine learning

- How to convert time series problems into machine learning problems?

| Features (X) | | | | Target (y) |
|:---:|:---:|:---:|:---:|:---:|
| $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ |
| $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ |
| $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ |
| $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
| $y_5$ | $y_6$ | $y_7$ | $y_8$ | $y_9$ |
| $y_6$ | $y_7$ | $y_8$ | $y_9$ | $y_{10}$ |
| $y_7$ | $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ |
| $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ |

# Single-Output (one-step ahead)

| Input Features | | | | Single-Output |
|:---:|:---:|:---:|:---:|:---:|
| Lag 3 | Lag 2 | Lag 1 | Lag 0 | Lead 1 |
| $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ |
| $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ |
| $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ |
| $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
| $y_5$ | $y_6$ | $y_7$ | $y_8$ | $y_9$ |
| $y_6$ | $y_7$ | $y_8$ | $y_9$ | $y_{10}$ |
| $y_7$ | $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ |
| $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ |
| $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ |
| $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ | $y_{14}$ |

# Single-Output (multiple-step ahead)

| Input Features | | | | Single-Output |
|:---:|:---:|:---:|:---:|:---:|
| Lag 3 | Lag 2 | Lag 1 | Lag 0 | Lead 1 |
| $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ |
| $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ |
| $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ |
| $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
| $y_5$ | $y_6$ | $y_7$ | $y_8$ | $y_9$ |
| $y_6$ | $y_7$ | $y_8$ | $y_9$ | $y_{10}$ |
| $y_7$ | $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ |
| $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ |
| $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ |
| $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ | $y_{14}$ |

# Multi-Output

| Input Features | | | | Multi-Output | | |
|:---:|:---:|:---:|:---:|:---:|:---:|:---:|
| Lag 3 | Lag 2 | Lag 1 | Lag 0 | Lead 1 | Lead 2 | Lead 3 |
| $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ |
| $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ |
| $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ | $y_9$ |
| $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ | $y_9$ | $y_{10}$ |
| $y_5$ | $y_6$ | $y_7$ | $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ |
| $y_6$ | $y_7$ | $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ |
| $y_7$ | $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ |
| $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ | $y_{14}$ |
| $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ | $y_{14}$ | $y_{15}$ |
| $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ | $y_{14}$ | $y_{15}$ | $y_{16}$ |

Pedram Jahangiry

# Road map!

✓ Module 1- Introduction to Deep Forecasting

✓ Module 2- Setting up Deep Forecasting Environment

✓ Module 3- Exponential Smoothing

✓ Module 4- ARIMA models

✓ Module 5- Machine Learning for Time series Forecasting

• Module 6- Deep Neural Networks

• Module 7- Deep Sequence Modeling (RNN, LSTM)

• Module 8- Transformers (Attention is all you need!)

• Module 9- Prophet and Neural Prophet