Department of Computational and Data Sciences

INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

AI: Module 01

Deepak Subramani

Assistant Professor

Dept. of Computational and Data Science

Indian Institute of Science Bengaluru

ML: Mental Model

Data that can be collected

$$\begin{tikzcd} X \arrow[r] & \boxed{\text{Real World}} \arrow[r] & y \end{tikzcd}$$

Quantity that must be predicted to make money

Data that can be collected

$$\begin{tikzcd} X \arrow[r] & \boxed{h_\theta(X;\theta)} \arrow[r] & \hat{y} \end{tikzcd}$$

Machine's Prediction

The Machine Learning Workflow

1. Frame the ML problem by looking at the business need
   a. Identify subproblems (One/more of the 5 tasks a computer can do)
   b. Establish a current baseline (What is currently done?)
   c. Define success
2. Gather the data and do Data Munging/Wrangling + Baselines
   a. Explore the data
   b. Clean data and prepare for the downstream ML models
   c. Establish a data, domain and SoTA baseline
3. Explore different models, improve them through Cross Validation and perhaps new model design
4. Form an ensemble of multiple models and solutions
5. Present your solution
   a. Say a story with the data
6. Deploy

Decision Trees

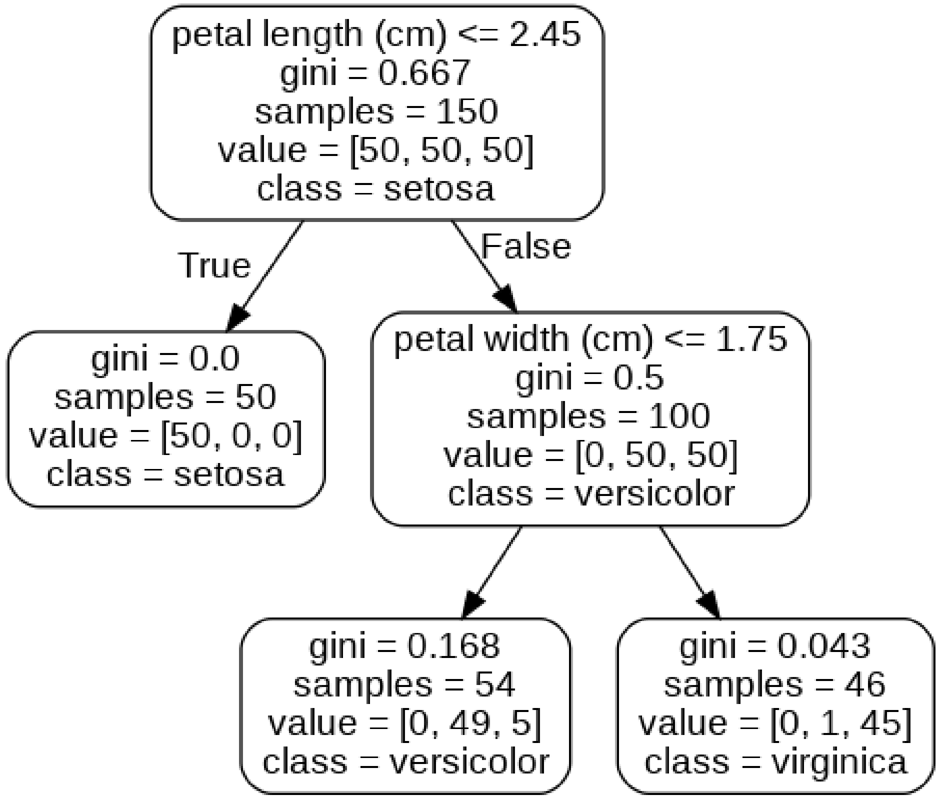
- Decision Trees are data driven models for classification and regression
- They are a versatile ML Algorithm capable of fitting complex data
- They are trained by a greedy optimization algorithm called CART
- Plan of action:
    1. See commands for training DT in sklearn
    2. Visualize DT
    3. Predict with DT and see how they make a decision
    4. Understand the math of the optimization algorithm
    5. Discuss pros and cons

Training a DT in sklearn

- from sklearn.datasets import load_iris
- from sklearn.tree import DecisionTreeClassifier
- iris = load_iris()
- X = iris.data[:,2:] #Make 2 features in the data
- y = iris.target
- tree_clf = DecisionTreeClassifier(max_depth=2)
- tree_clf.fit(X,y)

Making Predictions with DT

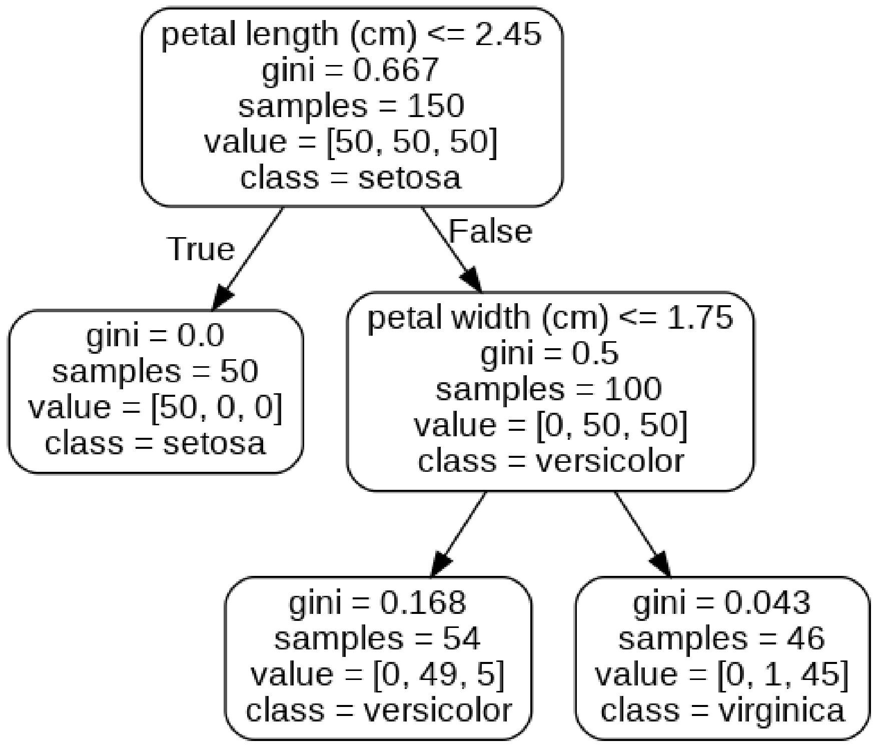
- How does DT classify a new data point?
- Start from the top, and move down asking the question at each node and following the answer
- Is petal length (cm) <= 2.45?
  - True – move left; False – move right
- Keep moving until you reach a leaf node
  - Leaf node – no children
- The class of the leaf node a datapoint ends up in is its class!
- That simple!

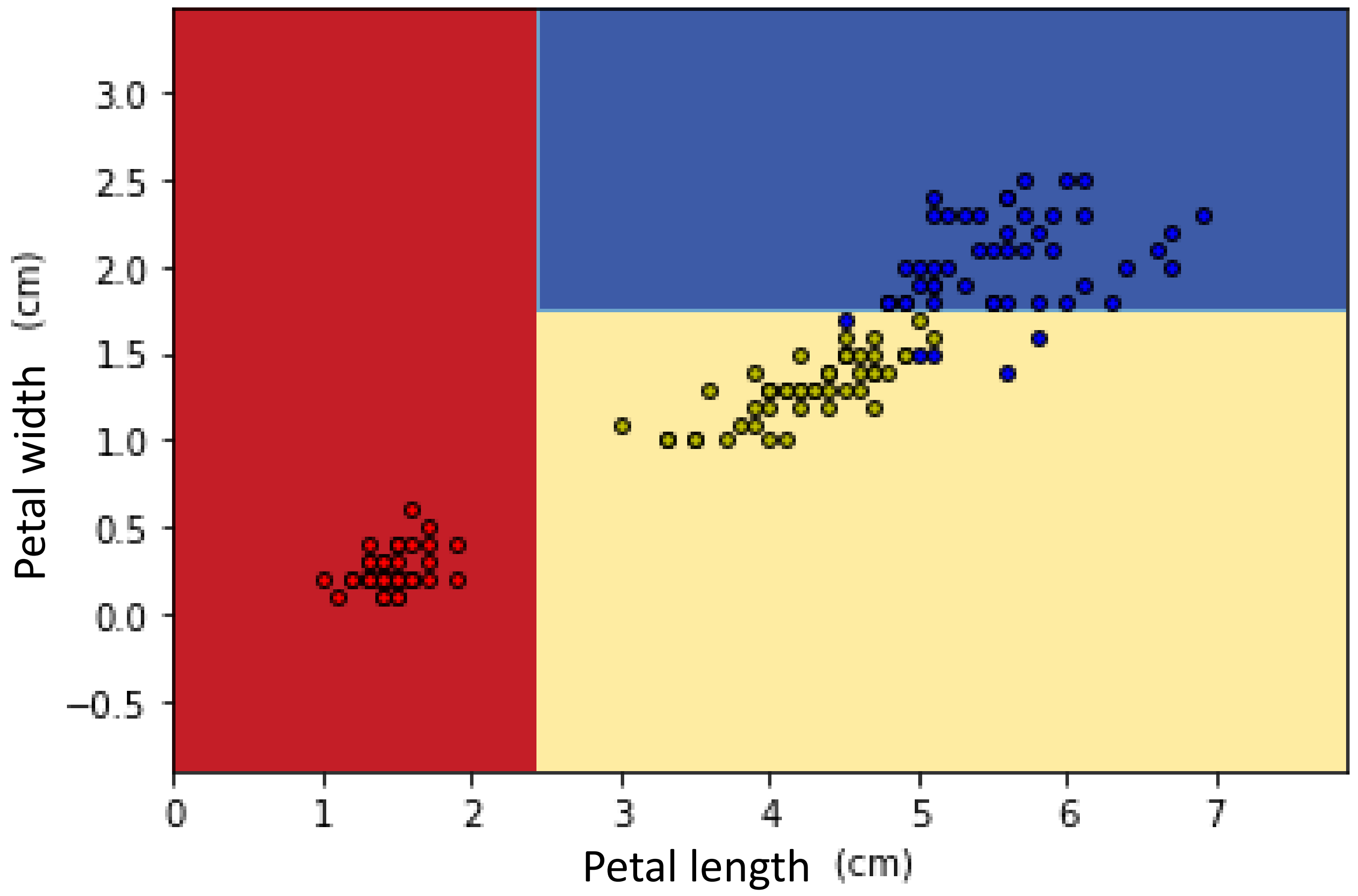


Understanding DT Attributes

- Gini attribute:
  - Measures impurity at a node
  - $G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$
  - $p_{i,k}$ is the fraction of kth class in ith node. n=total classes
- Samples attribute:
  - Number of training instances for which this question was applied
- Value attribute:
  - How many training instances of each class this node applies to
- Class attribute:
  - The label which will be applied to the instance if we stop there
- Probability of a class at a particular node
  - Value/samples

Decision Boundaries





CART Training Algorithm

1. At every node, split the training set into two subsets based on one single feature x_k and a threshold t_k on it
2. The pair (x_k, t_k) is chosen by what results in the purest subset measured by Gini Coefficient or Entropy

$$J(x_k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

$G_{l/r}$ is the impurity and $m_{l/r}$ is the number of instances

3. Continue the same at the two child nodes
4. Stop when no further split reduces impurity or maximum depth is reached

Evaluation Metrics – Classification Confusion Matrix

	C+	C-
T+	True Positive	False Positive
T-	False Negative	True Negative

- Recall = TP/(TP+FN) – Also called Sensitivity in Statistics
- False Negative Rate = FN/(TP+FN)

- Specificity = TN/(FP+TN)
- False Positive Rate = FP/(FP+TN)

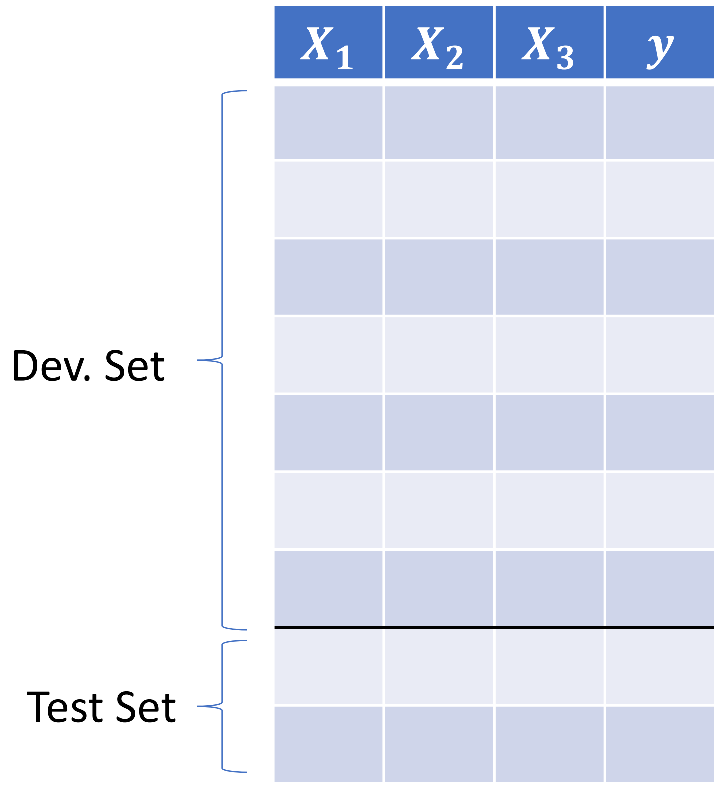
- Precision = TP/(TP+FP)
- F1 Score = Harmonic mean of Recall and Precision

Evaluation Metrics – Regression

- Root Mean Square Error
- Mean Absolute Error
- Relative Errors
- R2 = 1 – MSE/Variance

Development-Testing Paradigm

- Consider a data set with m rows and n columns
- Development Set
  - Used to train a ML model
- Training a model involves
  - Finding parameters or growing trees
  - Uses data and an optimization algorithm working on some loss
- Development involves training and hyper-parameter tuning
  - K-Fold Cross Validation is used
- Testing involves using a developed model on totally unseen data and evaluating an expected real world performance

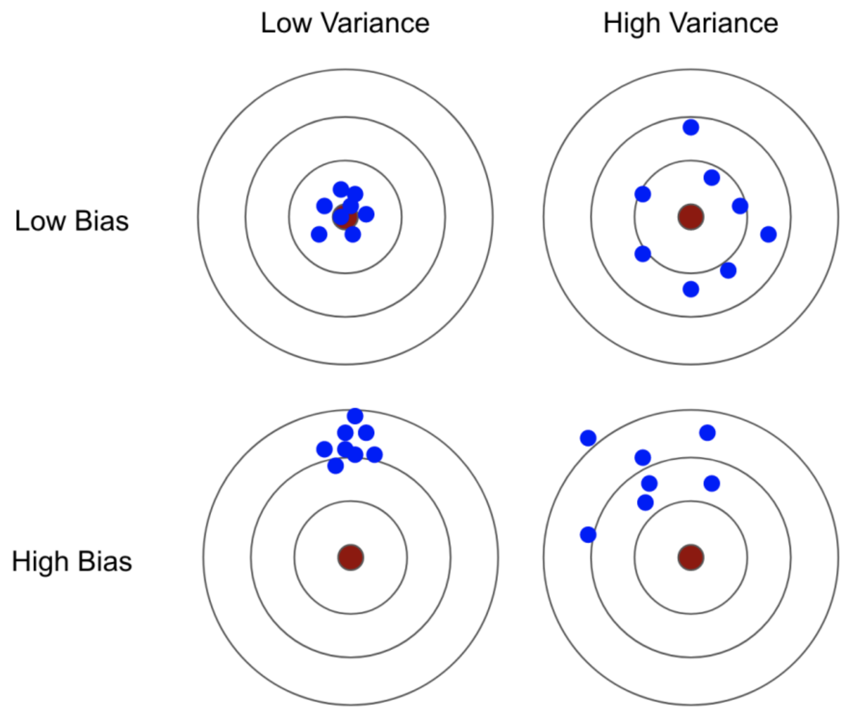


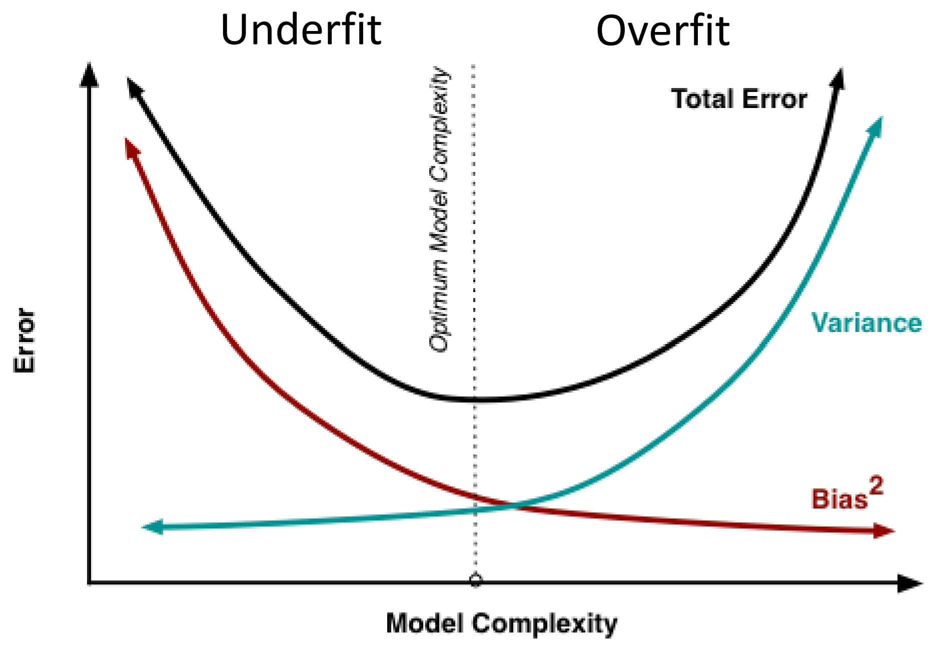
Bias-Variance Tradeoff

- Error = Bias + Variance + Irreducible Error
- Bias: Error from erroneous assumptions in the model
    - High bias -> underfitting
    - Model has limited flexibility to learn
    - Analogy: Overly simplistic assumptions about people make you a biased person
- Variance: Error from sensitivity of model to small perturbations in training data
    - High variance -> overfitting
    - Model has too much flexibility to learn
    - If you have a lot of data and over complex model, you can make each parameter of the model "by-heart" one data point
    - When a new data point comes, then the model will change wildly to accommodate the new point

Bias/Variance Equation

- $Error \; = E[(y-\hat{y})^2]$ –> RMSE
- $Error = (E[\hat{y}]-y)^2 + E[(\hat{y}-E[\hat{y}])^2] + \sigma_e^2$
- Error = Bias + Variance + Irreducible Error

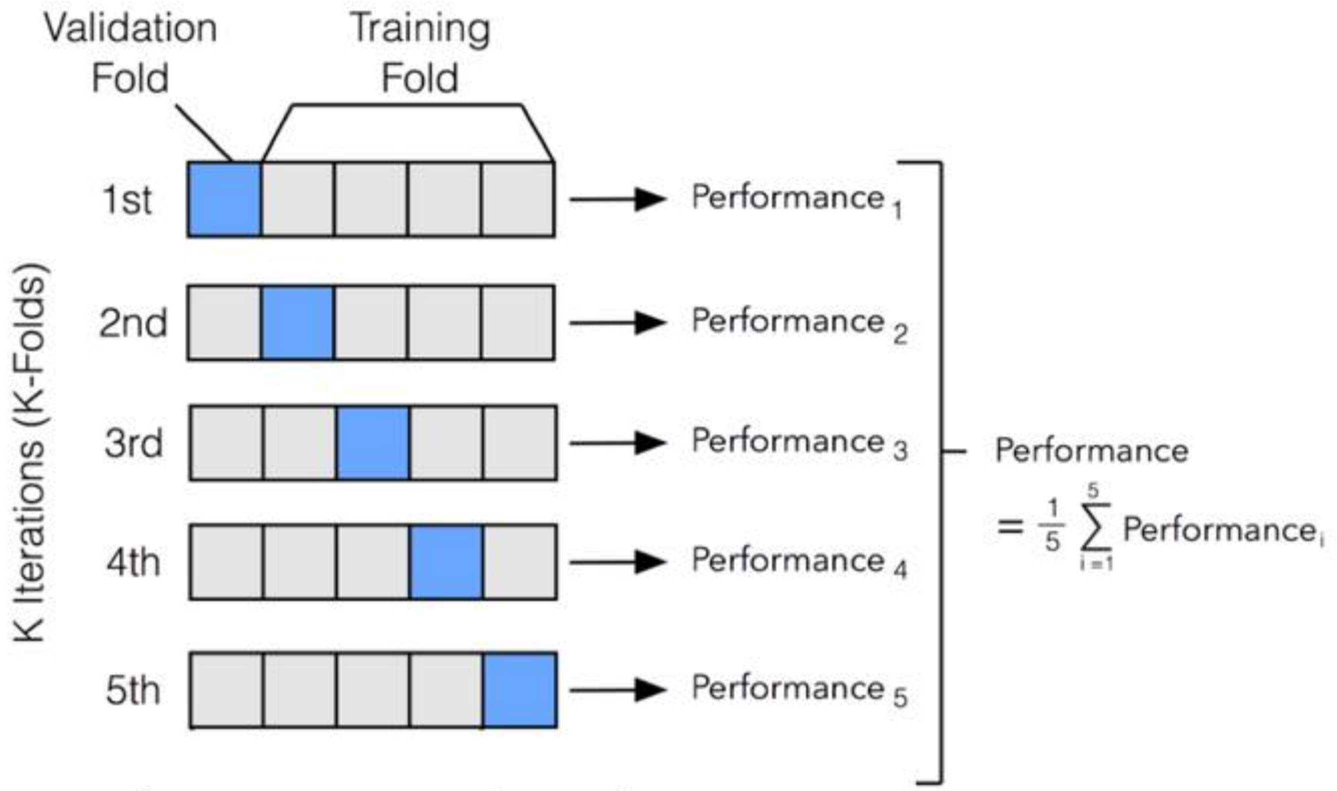




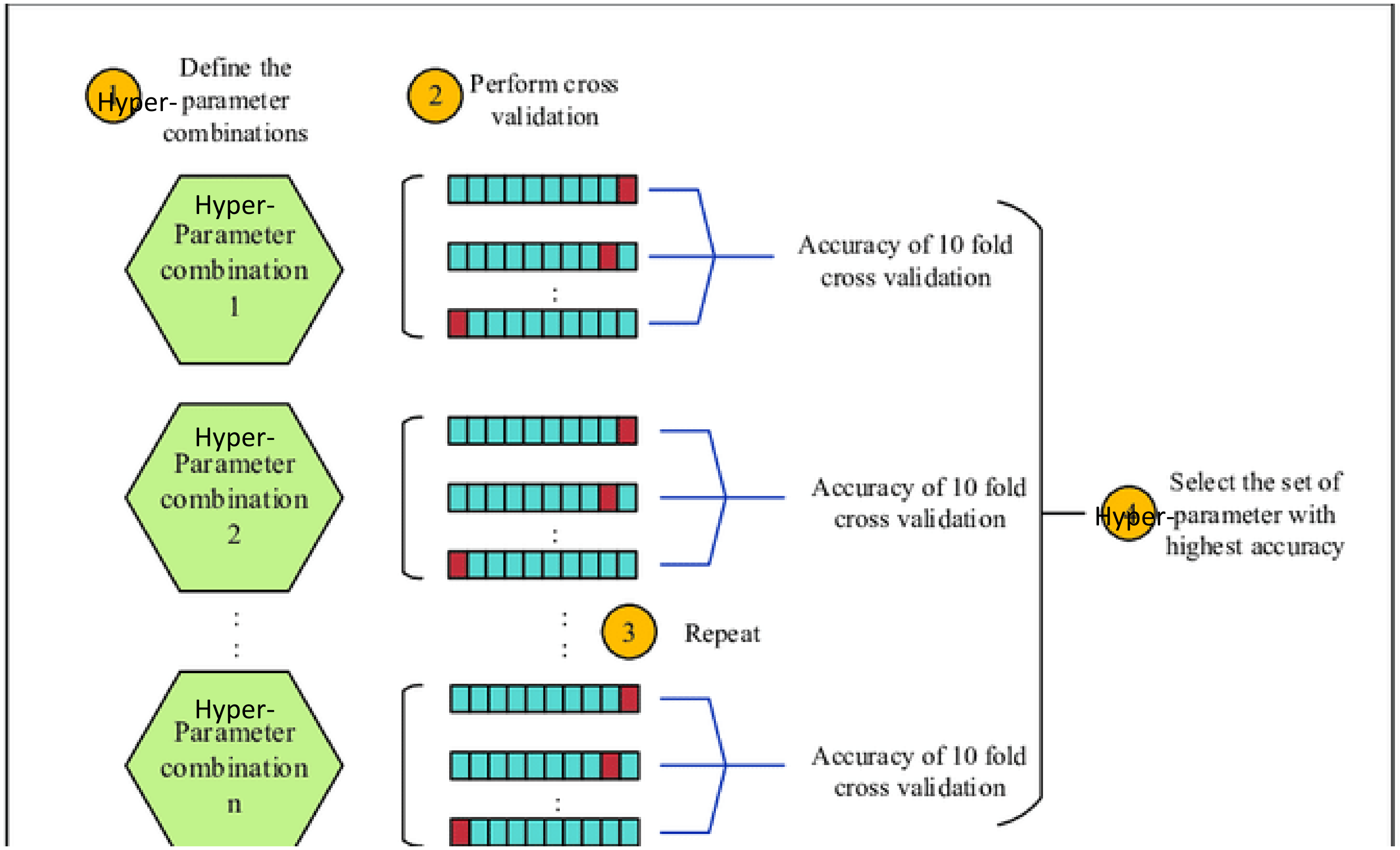
Regularization

- DT are what is called as non-parametric models
  - They don’t have a pre-defined number of parameters
  - On the other hand, linear models are parametric, with predefined number of parameters
- Thus DT can overfit any complex training data
- To avoid overfitting, we restrict a DT’s degrees of freedom
- Use the following hyperparameters to regularize and avoid overfitting:
  - max_depth: the maximum number of levels
  - min_samples_split: the minimum number of samples a node must have before it can be split,
  - min_samples_leaf: the minimum number of samples a leaf node must have,
  - min_weight_fraction_leaf: same as min_samples_leaf but expressed as a fraction of the total number of weighted instances,
  - max_leaf_nodes: the maximum number of leaf nodes, and
  - max_features: the maximum number of features that are evaluated for splitting at each node.
  - Increasing min_ hyperparameters or reducing max_ hyperparameters will regularize the model.
- Can also build a tree without restrictions and prune

K-Fold Cross Validation

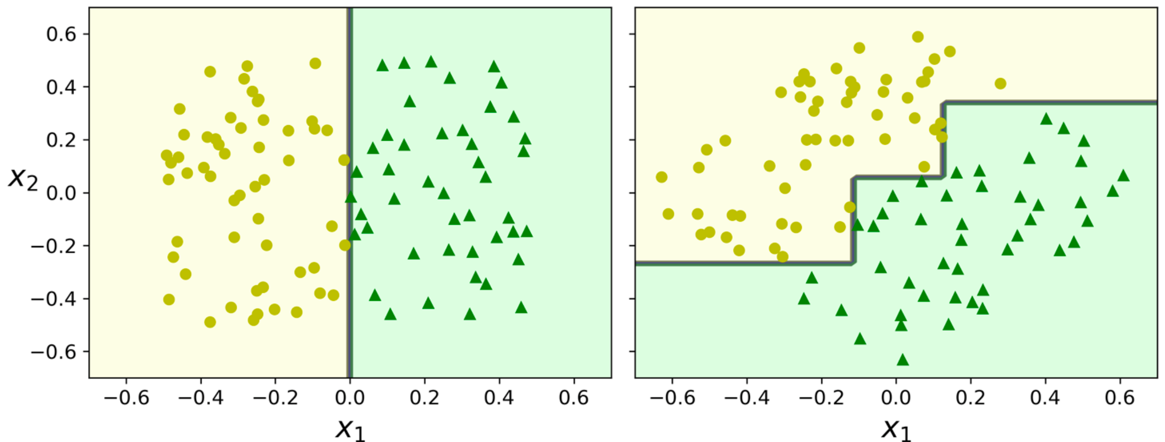


Hyper-parameter Tuning


1 Define the Hyper-parameter combinations
2 Perform cross validation
Hyper-Parameter combination 1
Accuracy of 10 fold cross validation
Hyper-Parameter combination 2
Accuracy of 10 fold cross validation
3 Repeat
Hyper-Parameter combination n
Accuracy of 10 fold cross validation
4 Select the set of Hyper-parameter with highest accuracy

Issues with DT

- Decision trees produce orthogonal decision boundaries
- Hence, rotating data makes DT unnecessarily convoluted
  - Can use PCA to overcome this issue
- Decision Trees are very sensitive to small variations in the training data
  - Removing an outlier data point may drastically change the DT
- Random forests and XGBoost can overcome this instability by ensemble learning!



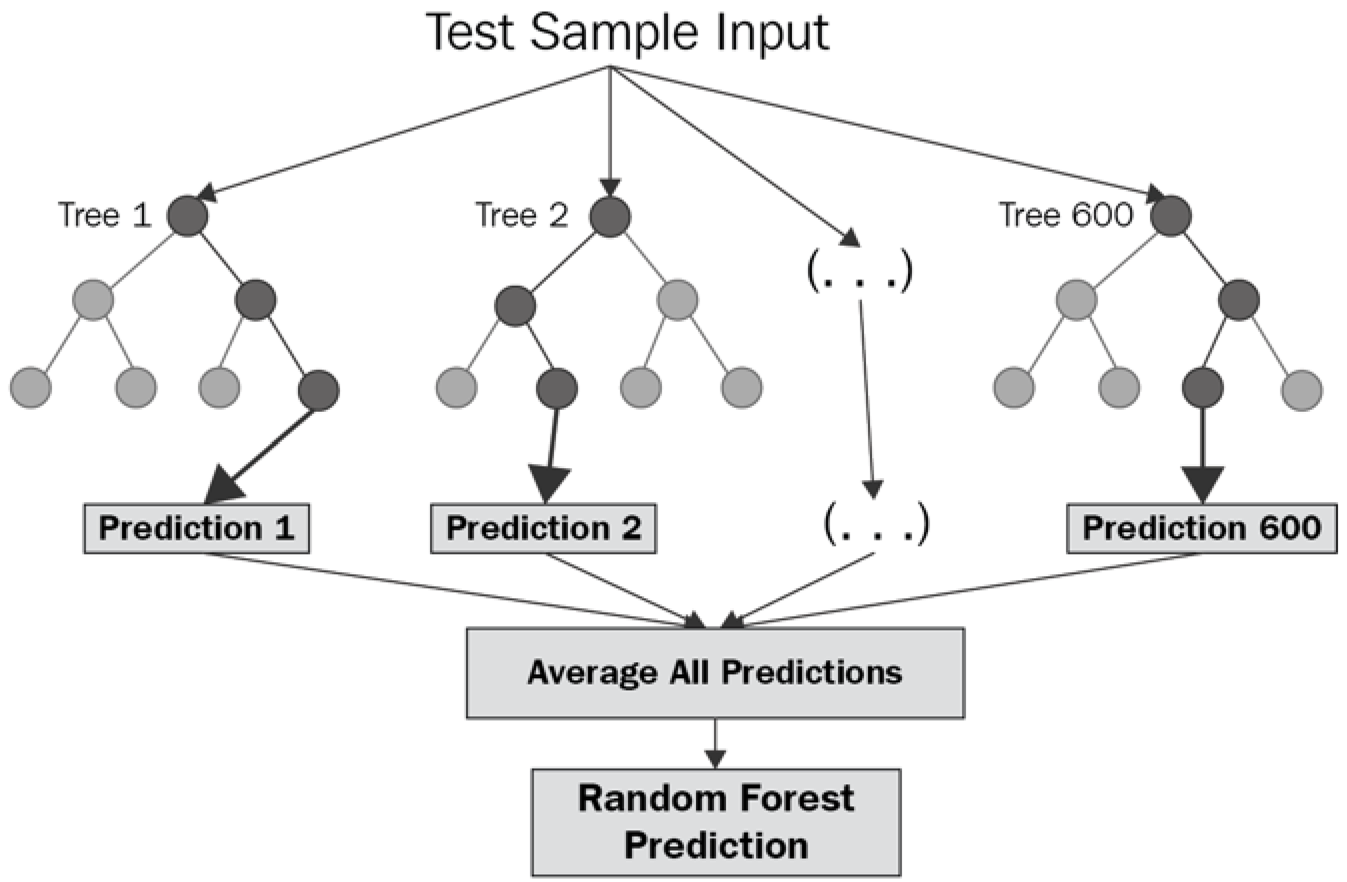
Ensemble Learning

- Audience poll/Wisdom of the crowd: Pose a complex question to 1000s of random knowledgeable people and aggregate their responses
  - The average response is usually better than one experts response, even if the random people are only knowledgeable
- This idea is called Ensemble Learning
- Example:
  - Train different Decision Trees on random subsets of the training data
  - Make the final prediction as an average of all the trees
  - Multiple trees trained on random subsets -> Random Forests!
- Different ensemble techniques: Voting, Bagging, Boosting, Stacking

Random Forests

- Random Forest - Ensemble of Decision Trees trained via Bagging
- Sklearn provides RandomForestClassifier and RandomForestRegressor classes
  - A more efficient implementation for Bagging with Decision Trees
  - ✓ Usually trained with a random subset (max_samples) of data
  - ✓ Usually trained with a random subset (max_features) of features at each tree
- RandomForestClassifier has all the hyperparameters of DecisionTreeClassifier and BaggingClassifier with a few obvious exceptions (no base_estimator)

RF Illustration



Department of Computational and Data Sciences

INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

XG Boost is All You Need

Deepak Subramani

Assistant Professor

Dept. of Computational and Data Science

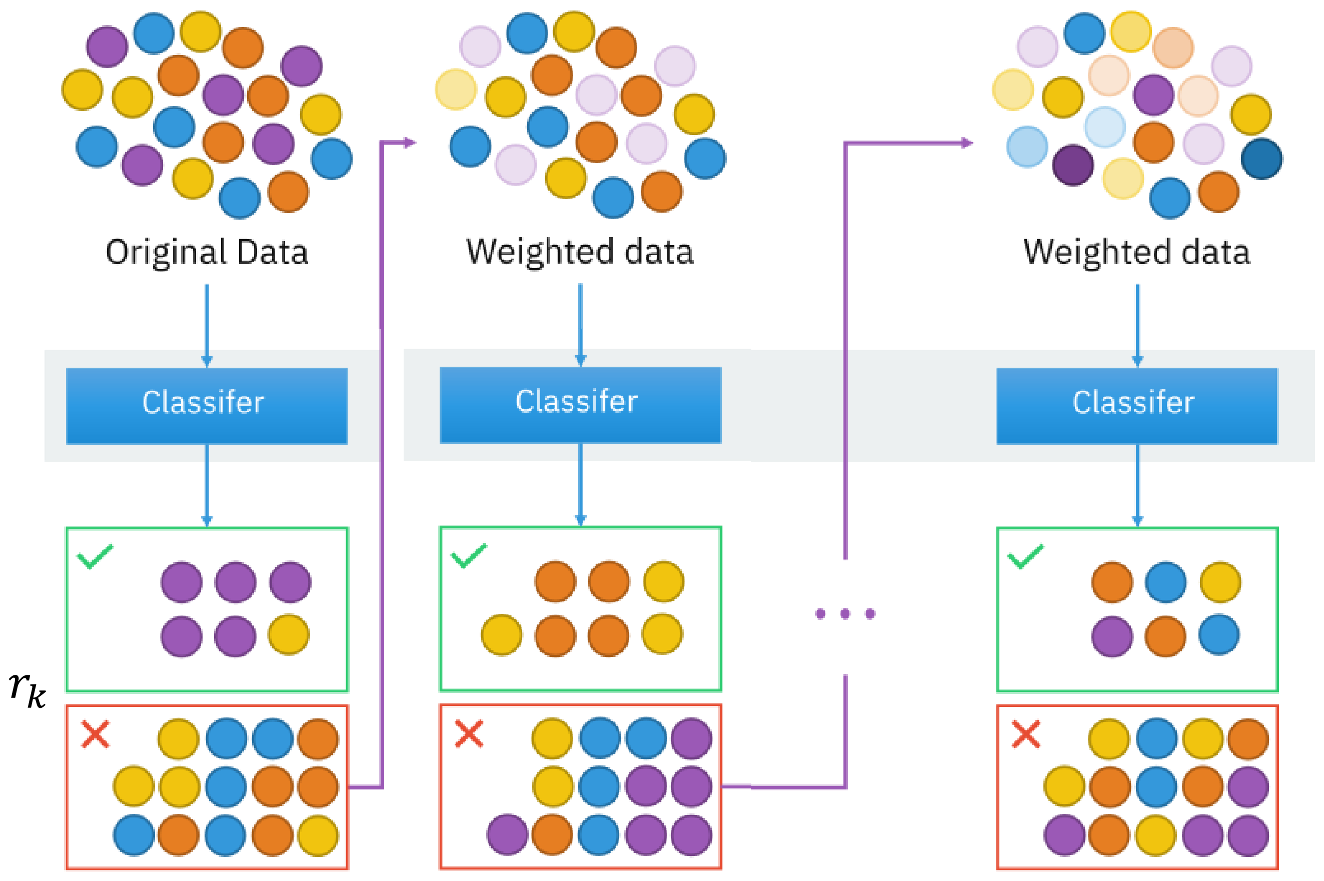
Indian Institute of Science Bengaluru

Boosting

- Boosting is a Sequential Ensemble method
- General Idea – Train predictors sequentially so that each successive predictor corrects the errors of its predecessor
- Imagine a sequential learning system
  - The first learner uses all the data for training. There may be some wrong predictions
  - The second learner tries to learn from all the wrong predictions. There may still be some more wrong predictions
  - Repeat
- Among several Boosting Methods available, the most popular are
  - Adaptive Boosting (AdaBoost)
  - Gradient Boosting (e.g., XGBoost)

AdaBoost - Illustration

$$w^{(j)} = w^{(j)} \exp(\alpha_k)$$



$$\alpha_k = \eta \log \frac{1 - r_k}{r_k}$$

1. Start with all samples having the same weight
2. Train a base classifier and make predictions on the training set
3. Find error r_k
4. Set Classifier weight $\alpha_k = \eta \log \frac{1-r_k}{r_k}$
5. Update weights of the misclassified instances:
6. Train a second classifier with updated weights and make predictions on the training set
7. Update instance weights and continue
8. Final Prediction – Weighted Ensemble

$$\hat{y} = argmax_{label} \sum_{k=1}^{N} \alpha_k \, [pred_k(X) == label]$$

Department of Computational and Data Sciences

Work Out Problem

Consider the following data set where each entry is $[x^{(j)}, y^{(j)}]$:

[(0,1), (1,-1), (2,1), (3,-1), (4,1), (5,-1)]

We will use decision stumps as the ensemble weak learner. Recollect that a decision stump is a decision tree with depth=1, i.e., it is a classifier of the form

```
if x>v then s else -s
```

Here, the value v is usually chosen from the midpoints between the data points, which is [0.5, 1.5, 2.5, 3.5, 4.5] in this case and $s \in \{1, -1\}$

1. Compute the α (leave it in the log form, don't write decimals) for the stumps chosen in the first two rounds by AdaBoost.

Round	V	S	Alpha (Fill this column)
1	0.5	-1	
2	4.5	-1	

3. What are the predictions that the resulting classifier (after these two rounds of boosting) makes for each of the training points?

AdaBoost

- AdaBoost can be used by both classifier and regressor
- Each classifier here can be any algorithm that we have seen so far - LogisticRegression, SVC, DecisionTreeClassifier
- If a weight update increases the weight of an instance, it is a “boost” to those instances

Department of Computational and Data Sciences

Gradient Boosting

- In Adaptive Boosting, each sample weights are updated sequentially based on the wrong predictions
- In Gradient Boosting, subsequent predictors are trained on the residual errors made by the previous tree
    - This setting is more natural for regression
    - Classification is done as a regression using deviance (or exponential) loss function as the error to which subsequent predictors are fit

Gradient Boosted Regression Tree

1. Train a predictor $\widehat{y_1} = h_1(x;\theta_{f1})$ to minimize $loss(y, y_1)$

2. Train a predictor $\widehat{y_2} = h_2(x;\theta_{f2})$ to minimize $loss(y - y_1, y_2)$

3. So on until n_estimators

4. Final prediction is $\hat{y} = \hat{y}_1 + \hat{y}_2 + \cdots$

Example of Gradient Boosting with DT

- from sklearn.tree import DecisionTreeRegressor
- tree_reg1 = DecisionTreeRegressor(max_depth=2)
- tree_reg1.fit(X, y)
- y2 = y - tree_reg1.predict(X)
- tree_reg2 = DecisionTreeRegressor(max_depth=2)
- tree_reg2.fit(X, y2)
- y3 = y2 - tree_reg2.predict(X)
- tree_reg3 = DecisionTreeRegressor(max_depth=2)
- tree_reg3.fit(X, y3)
- y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))

GradientBoostingRegressor

- from sklearn.ensemble import GradientBoostingRegressor
- gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0)
- gbrt.fit(X, y)

- Think of Gradient Boosting just like Gradient Descent algorithm
  - In each stage a regression tree is fit on the negative gradient of the given loss function.
  - For MSE, it is simply the residual as we have described
- XGBoost – a python package (similar API as sklearn) implements excellent gradient boosting

Regularization

- Gradient Boosting can easily overfit the training samples
- Shrinkage via Learning Rate
  - Shrinkage is a way to perform regularization
  - Idea is to use a learning_rate hyperparameter that skrinks the contribution of each consecutive decision tree
- Subsampling
  - Stochastic gradient boosting: Every predictor is trained only on a subset of the data which is sampled without replacement
  - One can also subsample the max_features to be used in each subset [Subspace]
  - Subsampling acts as a regularization

XGBoost+Optuna or HyperOpt

- XGBoost: https://xgboost.readthedocs.io/en/stable/tutorials/model.html

- HyperOpt: http://hyperopt.github.io/hyperopt/

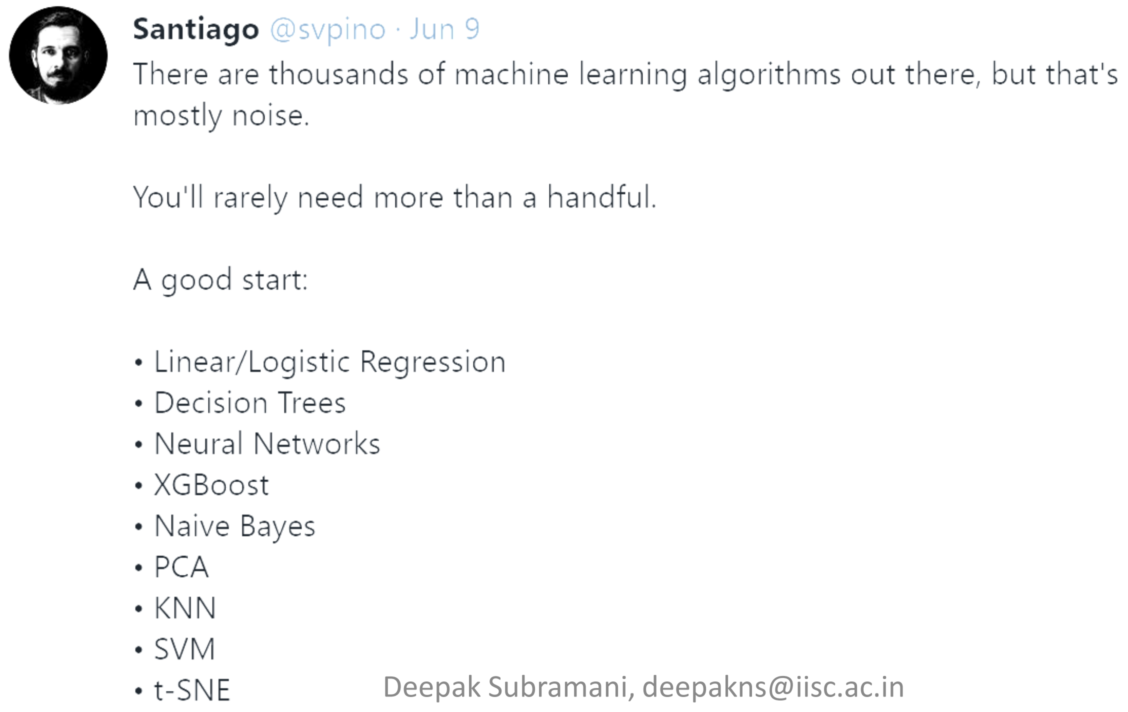
- Walk through Code Example: https://medium.com/optuna/using-optuna-to-optimize-xgboost-hyperparameters-63bfcdfd3407

Alternatives of XGBoost

- CatBoost: https://catboost.ai/en/docs/concepts/tutorials
- HistGradientBoosting: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html
- LightGBM: https://github.com/Microsoft/LightGBM

What do you need? Is it Neural Net?

- Tabular Data: Deep Learning is Not All You Need
  - On comparing Deep Networks with Gradient Boosting Models (GBM), evidence shows that GBM is better for Tabular Data
  - https://arxiv.org/pdf/2106.03253.pdf


Santiago @svpino · Jun 9

There are thousands of machine learning algorithms out there, but that's mostly noise.

You'll rarely need more than a handful.

A good start:

• Linear/Logistic Regression
• Decision Trees
• Neural Networks
• XGBoost
• Naive Bayes
• PCA
• KNN
• SVM
• t-SNE