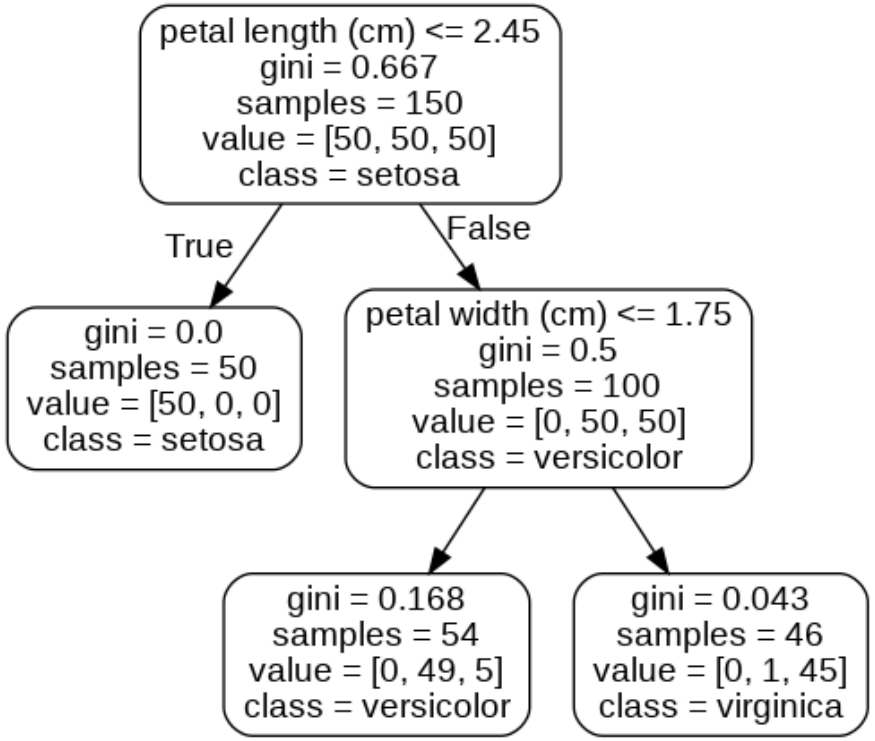
AI: Module 01 Week 02

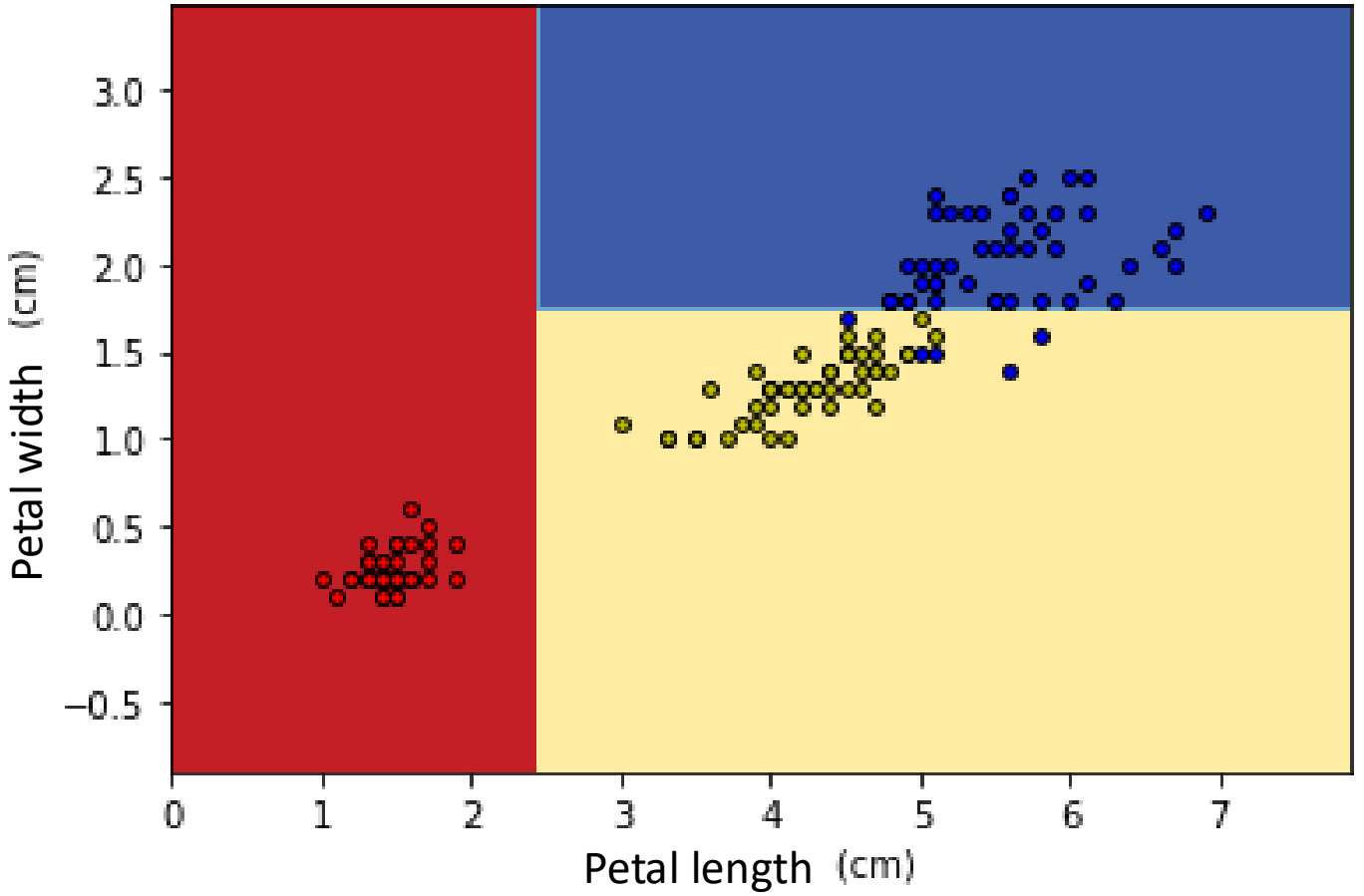
Deepak Subramani
Assistant Professor
Dept. of Computational and Data Science
Indian Institute of Science Bengaluru

Week 01 Summary

- Part 1
  - Decision Tree – Structure, Prediction, Attributes
  - Decision Boundary – Fundamental Concept in AI for Classification
  - Cart Training Algorithm
  - Collection of Decision Trees – Random Forest
- Part 2
  - Metrics of performance – Regression and Classification
  - Development-Testing Paradigm
  - Overfit vs Underfit – Regularization to prevent overfit
  - Hyperparameter Selection through k-fold CV

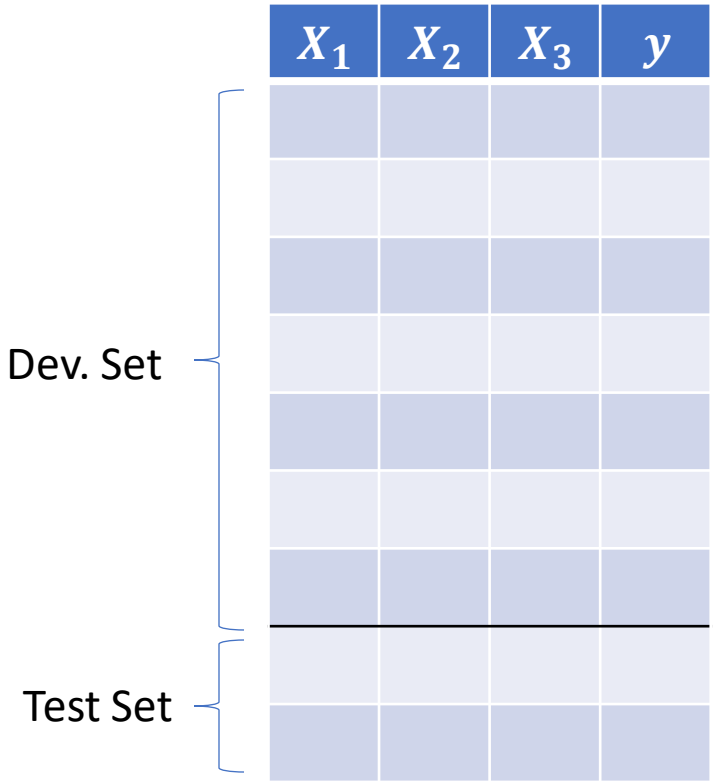
Decision Boundaries





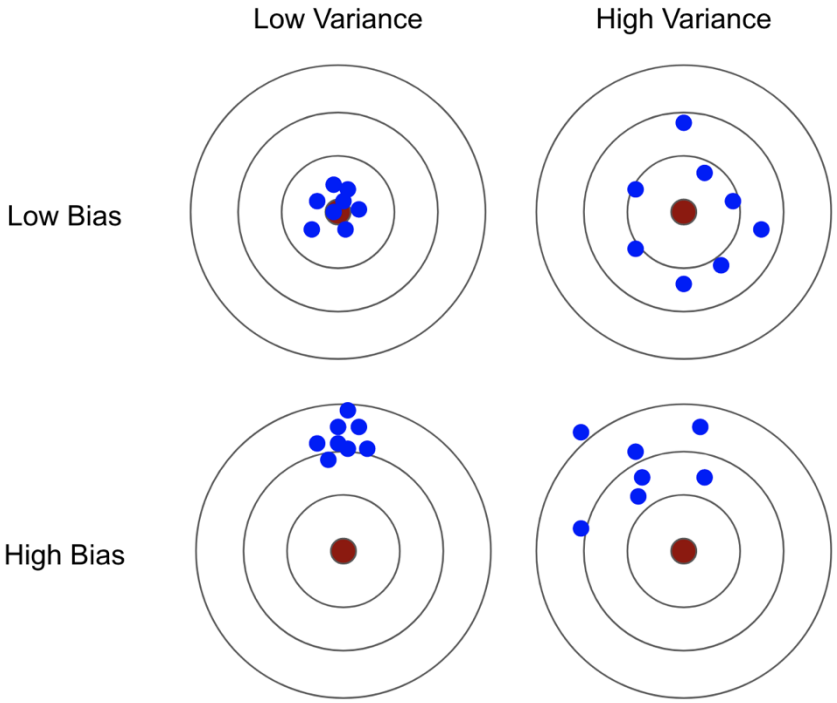
Development-Testing Paradigm

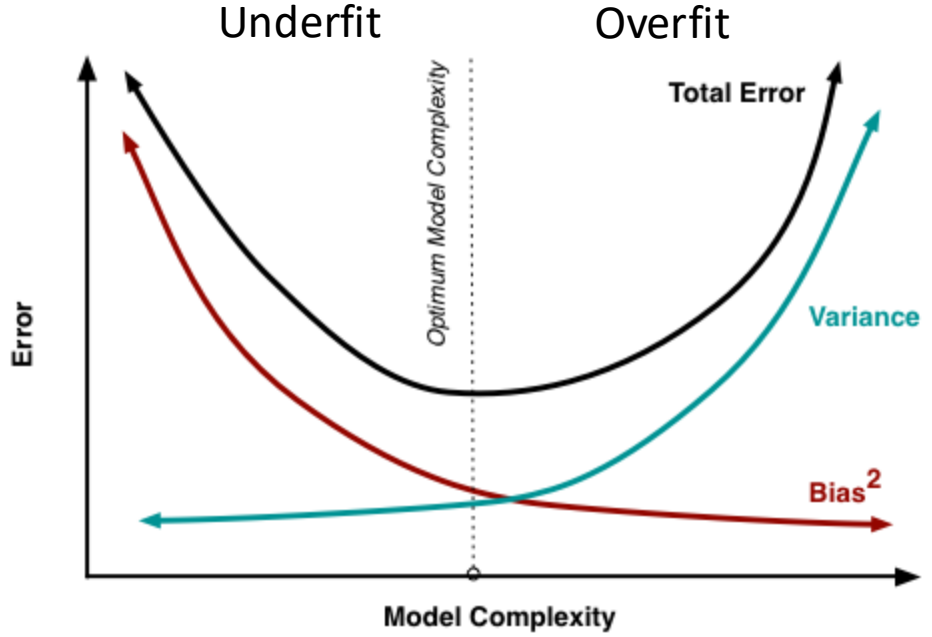
- Consider a data set with m rows and n columns
- Development Set
  - Used to train a ML model
- Training a model involves
  - Finding parameters or growing trees
  - Uses data and an optimization algorithm working on some loss
- Development involves training and hyper-parameter tuning
  - K-Fold Cross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



Bias/Variance 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





Week 02

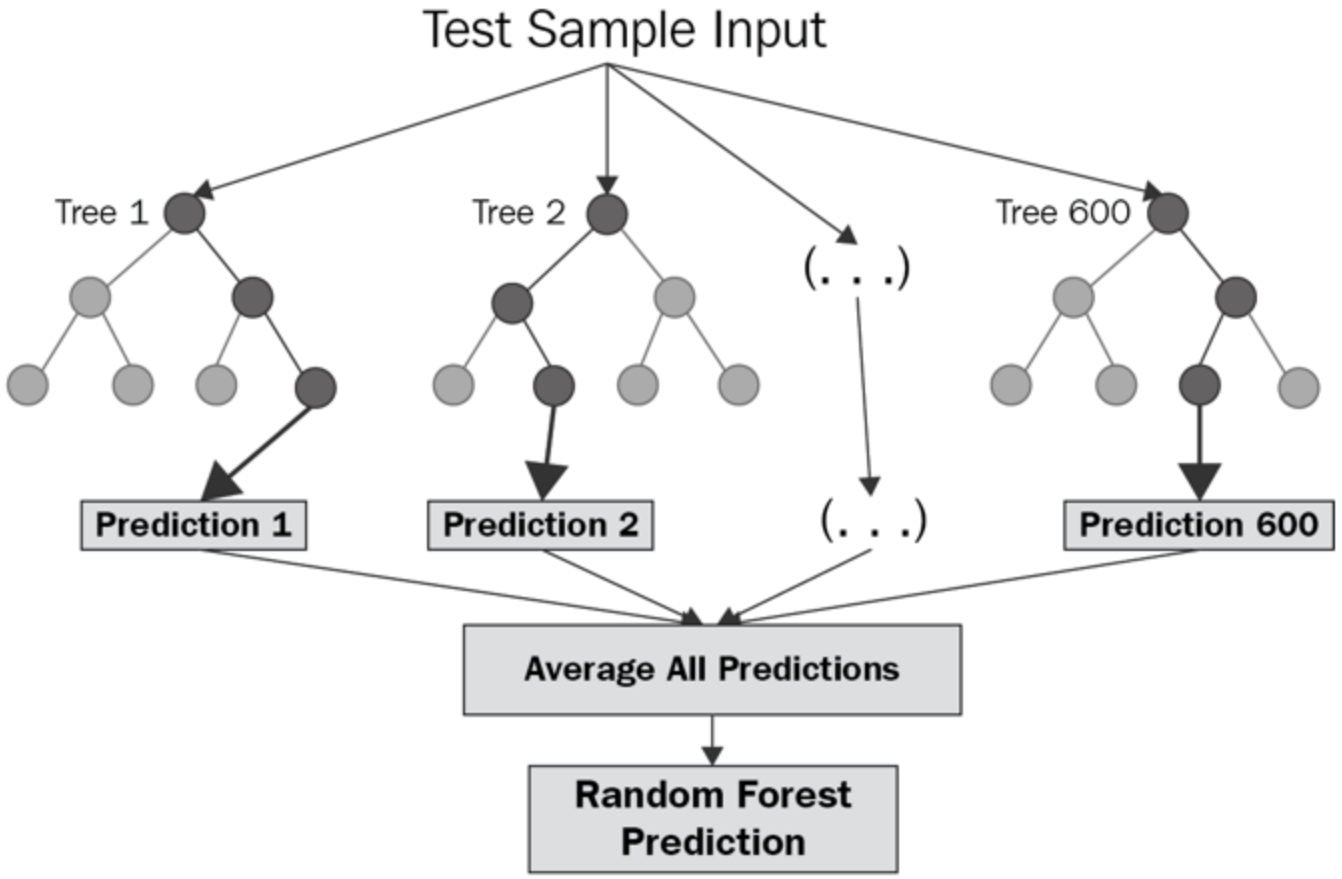
- Part 1
  - Random Forest
  - Feature Importance
- Part 2
  - Gradient Boosted Tree Models
  - Extreme Gradient Boosting – Key innovations
  - XGBoost vs LightGBM vs CatBoost

Deepak Subramani, deepakns@iisc.ac.in

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

RF Illustration



Feature Importance Example

- from sklearn.datasets import load_iris
- iris = load_iris()
- rnd_clf = RandomForestClassifier(n_estimators=600, n_jobs=-1)
- rnd_clf.fit(iris["data"], iris["target"])
- for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
  print(name, score)

- In iris, petal length and width have importance of 44% and 42%, sepal length and width only 11% and 2%

Further Reading

- The impurity-based feature importances computed on tree-based models suffer from two flaws that can lead to misleading conclusions.
- First they are computed on statistics derived from the training dataset and therefore **do not necessarily inform us on which features are most important to make good predictions on held-out dataset**.
- Secondly, **they favor high cardinality features**, that is features with many unique values.
- Permutation feature importance is an alternative to impurity-based feature importance that does not suffer from these flaws.
- These two methods of obtaining feature importance are explored in: Permutation Importance vs Random Forest Feature Importance (MDI).
- https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html

Permutation Feature Importance

- Inputs: fitted predictive model m, tabular dataset (training or validation) D.
- Compute the reference score s of the model m on data D (for instance the accuracy for a classifier or the R^2 for a regressor).
- For each feature j (column of D):
    - For each repetition k in $1, \ldots, K$:
        - Randomly shuffle column j of dataset D to generate a corrupted version of the data named $\tilde{D}_{k,j}$.
        - Compute the score $s_{k,j}$ of model m on corrupted data $\tilde{D}_{k,j}$.
    - Compute importance i_j for feature f_j defined as:

$$i_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j}$$

Audience Poll

1. Ensemble Learning can be used only with Decision Trees
   - True, False
2. Feature importance can be calculated ONLY for tree-based models
   - True, False
3. Permutation feature importance measures the importance of each feature by using an adversarial approach of permuting that feature and evaluating the model
   - True, False

Week 02

- Part 1
  - Random Forest
  - Feature Importance
- Part 2
  - Gradient Boosted Tree Models
  - Extreme Gradient Boosting – Key innovations
  - XGBoost vs LightGBM vs CatBoost

Deepak Subramani, deepakns@iisc.ac.in

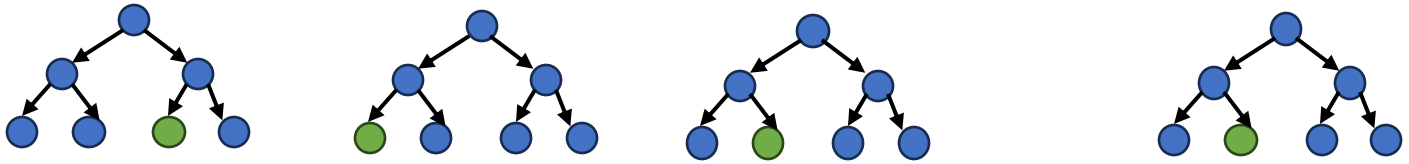
Boosting

- Boosting is a Sequential Ensemble method
- General Idea – Train models sequentially so that each successive model 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)

Gradient Boosted Regression Tree

Training Data

Input to all trees is all features of ith data point $\boldsymbol{x}_i$



Out:	$\widehat{y_{i,1}}$	$\widehat{y_2}$	$\widehat{y_3}$	$\widehat{y_n}$
Model:	F_1	F_2	F_3	F_n
Target:	y	$y - \hat{y}_1$	$y - \hat{y}_1 - \hat{y}_2$	

Final Output is Summation of output from each tree

$$\widehat{y_i} = F_1(x_i)$$

$$\widehat{y_i} = F_1(x_i) + F_2(x_i)$$

$$\widehat{y_i} = F_1(x_i) + F_2(x_i) + F_3(x_i)$$

$$\widehat{y_i} = \sum_{t=1}^{n} F_t(x_i)$$

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 $loss(y - \hat{y}_1, y_2)$
3. So on until n_estimators
4. Final prediction is $\hat{y} = \hat{y}_1 + \hat{y}_2 + \cdots + \hat{y}_n$

Deepak Subramani, deepakns@iisc.ac.in

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 = tree_reg1.predict(X_new) + tree_reg2.predict(X_new) + tree_reg3.predict(X_new)

Extreme Gradient Boosting

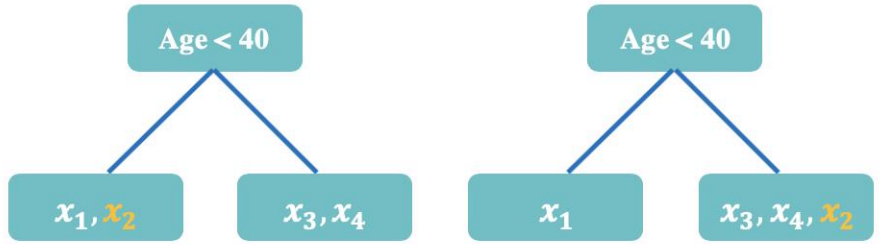
- A significant upgrade from Gradient Boosting
- Improvements to
  - Handle missing features (Sparsity aware split finding)
  - Gaining speed
    - Weighted quantile sketch with merging and pruning for approximate split-finding
    - Parallel computing by sorting and compressing data into blocks and using these blocks for split-finding
    - Cache aware prefetching for efficient read/write
    - Block compression and sharding to reduce read time from disk
  - Accuracy Gains through regularization
    - A new mathematical derivation for representing decision tree training with leafs and weights
    - A regularization term is added to prevent overfitting

Handling Missing Data

- XGBoost can handle missing data directly
- The sparsity aware split finding method calculates the score change if missing data is sent to left and right both, and pick the best option for dealing with missing feature data

ID	Gender	Age
1	F	10
2	M	??
3	F	40
4	M	80

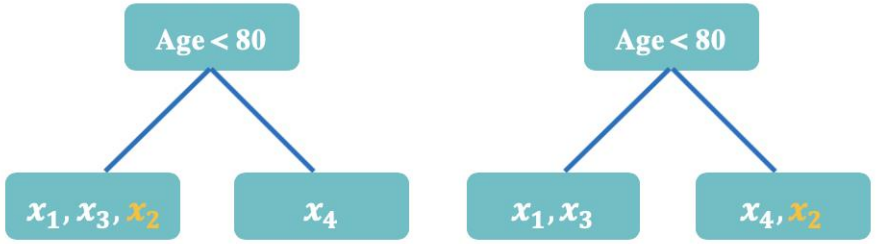
Split point 1:



The change in the loss function is $\Delta S_{40,L}$

The change in the loss function is $\Delta S_{40,R}$

Split point 2:



The change in the loss function is $\Delta S_{80,L}$

The change in the loss function is $\Delta S_{80,R}$

Most Important Hyperparameters

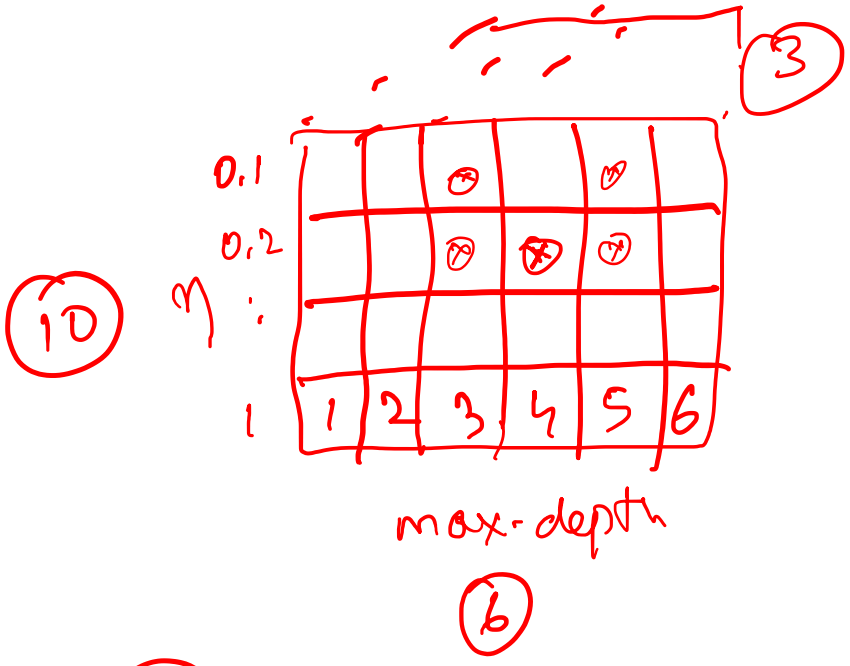
- **Tree-specific hyperparameters control the construction and complexity of the decision trees:**
  - max_depth: maximum depth of a tree. Deeper trees can capture more complex patterns in the data, but may also lead to overfitting.
  - min_child_weight: minimum sum of instance weight (hessian) needed in a child. This can be used to control the complexity of the decision tree by preventing the creation of too small leaves.
  - subsample: percentage of rows used for each tree construction. Lowering this value can prevent overfitting by training on a smaller subset of the data.
  - colsample_bytree: percentage of columns used for each tree construction. Lowering this value can prevent overfitting by training on a subset of the features.
- **Learning task-specific hyperparameters control the overall behavior of the model and the learning process:**
  - eta (also known as learning rate): step size shrinkage used in updates to prevent overfitting. Lower values make the model more robust by taking smaller steps.
  - gamma: minimum loss reduction required to make a further partition on a leaf node of the tree. Higher values increase the regularization.
  - lambda: L2 regularization term on weights. Higher values increase the regularization.
  - alpha: L1 regularization term on weights. Higher values increase the regularization.

- https://blog.dataiku.com/narrowing-the-search-which-hyperparameters-really-matter

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

Intuition of Optuna or HyperOpt



180

Bayesian Optimization

Loss landscape

proposal distribu

MCMC

Alternatives of XGBoost

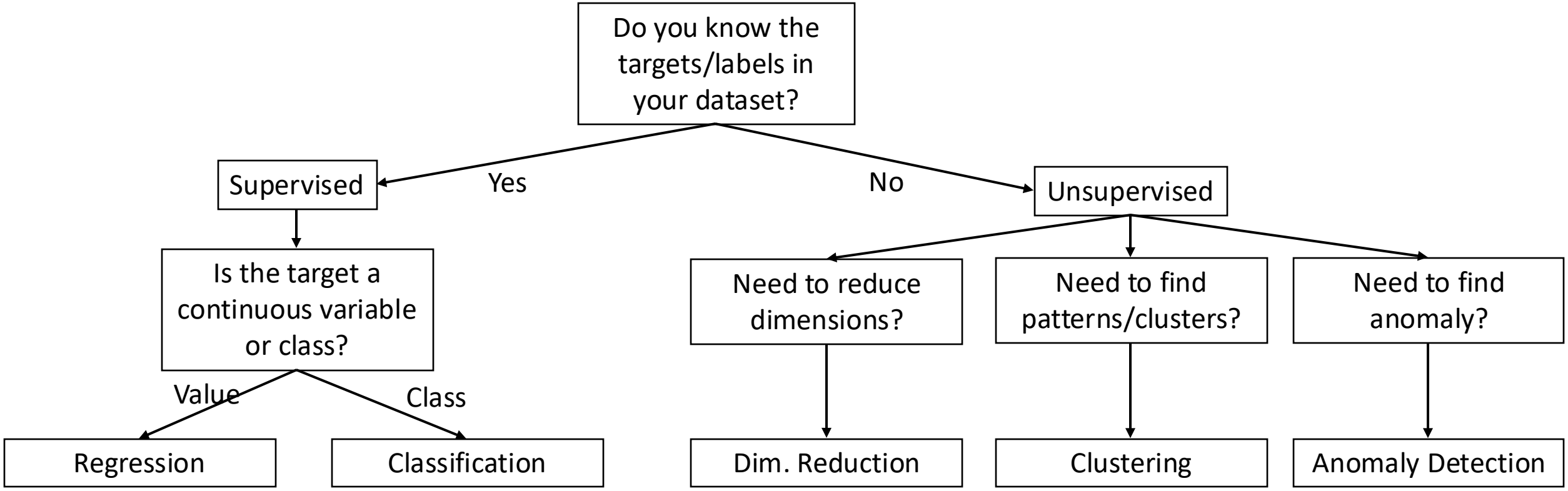
- CatBoost: https://catboost.ai/en/docs/concepts/tutorials
- HistGradientBoosting: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html
- LightGBM: https://github.com/Microsoft/LightGBM

XGBoost (2016) vs LightGBM (2017) vs CatBoost (2018)

- Three worthy competitors for gradient boosting
- Missing Value Handling
  - XGB and LGBM try both options (send missing to L or R) and choose the best
  - CatBoost assumes missing values are <min (or >max) and proceed
- Split finding
  - XGB uses a weighted quantile split strategy (histogram-based)
  - LGB uses Gradient Based One Side Sampling and Exclusive Feature Bundling to reduce the number of data points and features per node for split finding
  - CatBoost uses Ordered Target Statistics, a histogram-based split
- Tree growth
  - LGB uses leaf-wise splitting – leads to faster solution but at the cost of accuracy
  - CatBoost uses level-wise splitting
  - XGB has support for both

Deepak Subramani, deepakns@iisc.ac.in

Summary of ML



1.Regularized Linear Regression
2.Decision Tree Regression
3.Random Forest Regression
4.Boosted Tree Models
5.Neural Networks

1.Logistic Regression
2.Decision Tree Classification
3.Random Forest
4. Boosted Tree Model
5.Neural Networks

1.Projection - PCA
2.Embedding - LLE
3.Nonlinear methods - Auto Encoders

1.K-Means
2.DBSCAN
3.Hierarchical Clustering
4.GMM

1.GMM
2.Isolation Forest

What do you need?
Should I worry about Model XYZ?

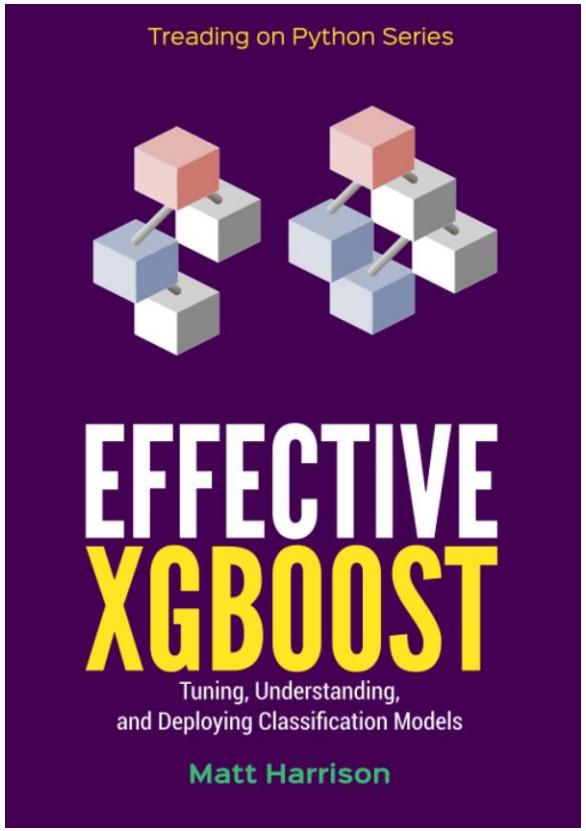
- Tabular Data: Deep Learning is Not All You Need
  - On comparing Deep Networks with Gradient Boosting Models (GBM), evidence shows that GBM is better for Tabular Data
  - https://arxiv.org/pdf/2106.03253.pdf

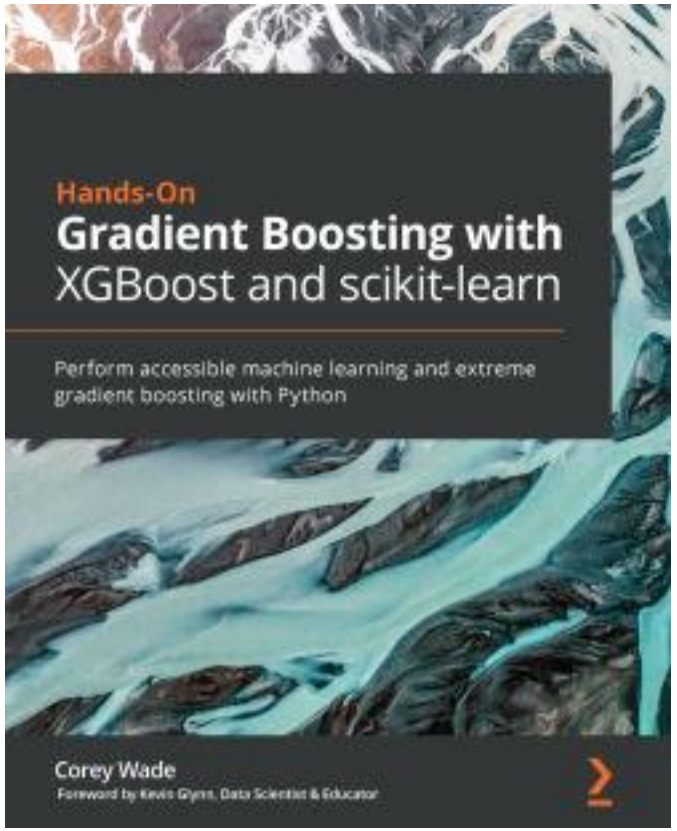
- There are 100s of ML Models and algorithms out in the wild
- Fortunately, most of it is flexing and noise
- In AI&MLOps, you rarely need beyond
  - Boosting Models for tabular Data
  - Convolutional Models for Image data
  - Transformer Models for Text and Speech Data (and Generative Image)
- In Classical Data and ML
  - PCA or t-SNE for visualization
  - K-Means, DBSCAN and Agglomerative Clustering for Clustering
  - K-NN for regression/classification baselines

Key Concepts So Far

1. Decision Trees
2. Decision Boundaries
3. Development-Testing Paradigm
   - K-Fold CV
   - Hyperparameter Tuning
4. Overfitting –
   - Bias/Variance Tradeoff
   - Regularization
5. Evaluation Metrics for Classification and Regression
6. Random Forests
7. Boosting

Resources





https://github.com/mattharrison/effective_xgboost_book

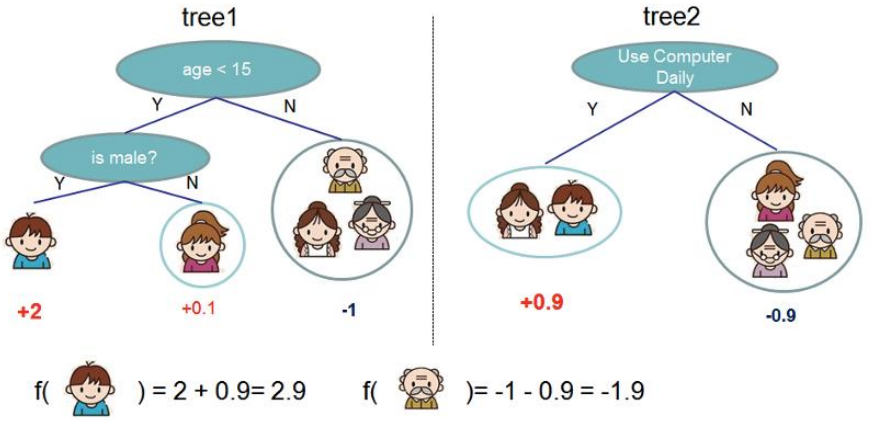
https://github.com/PacktPublishing/Hands-On-Gradient-Boosting-with-XGBoost-and-Scikit-learn

Additional Material

- More Details about XGBoost, LightGBM and CatBoost to help in interview preparation

XGBoost Objective

- Training of XGBoost is modified from regular GBRT
- Consider trees $F_t \in \mathcal{F}$ family of CART
- Let there be T_t leaves for F_t
- Define $q_t(x_i) \rightarrow 1:T_t$ a map that returns the leaf reached by a data point x_i
- $F_t(x_i) = w_{q_t(x_i)}$ where the vector w is a vector of weights of the leaves of F_t
- Final prediction from an XGBoost model is the sum of the leaf scores returned by n trees in the model
- $\hat{y}_i = \sum_{t=1}^{n} F_t(x_i) = \sum_{t=1}^{n} w_{q_t(x_i)}$



XGBoost Objective

- Process of training involves greedily adding a new tree or splitting the leaves of an existing tree
- Reduction in the objective decides whether to add a new tree or to split the existing tree
- XGBoost uses Taylor's theorem and loss reduction criterion to arrive at efficient conditions to determine split finding, and leaf scores

Training Objective:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Additive Model Training:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

Taylor Expansion - XGBoost needs only g and h

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

XGBoost Objective

Taylor Theorem upto 2nd Derivative

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

Substitution and Expansion of Regularization

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T\end{aligned}$$

Optimal w and objective Function

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

Loss Functions

- XGBoost only needs the first and second derivatives of the loss function
- It can handle any loss function
- Usually MSE is used for regression and Cross Entropy for Classification
- The algorithm is called gradient boosting as the residual of the MSE is the target in the vanilla Gradient Boosted Regression Tree (GBRT)

Approximate Split Finding

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

LightGBM

- LGBM uses a different set of code optimizations than XGB
  - Speed and memory usage (histogram based split finding)
  - Leaf wise tree growth (instead of level wise)
  - Optimal split into categorical features (by sorting the histogram of partitioned classes)
  - Optimization in network communication (during distributed learning)
  - Feature parallel, data parallel and voting parallel
- Key Algorithm Advance: It uses Gradient Based One Sided Sampling and Exclusive Feature Bundling to reduce the number of data points and features in a node for split finding
- LightGBM is the fastest model among all GBRT

CatBoost

- Main difference from XGBoost is the way categorical data is handled
- CatBoost can handle categorical data, text data and encodings out of the box
- It uses a histogram-based split in a special form called as Ordered Target Statistics
- It uses level-wise split finding