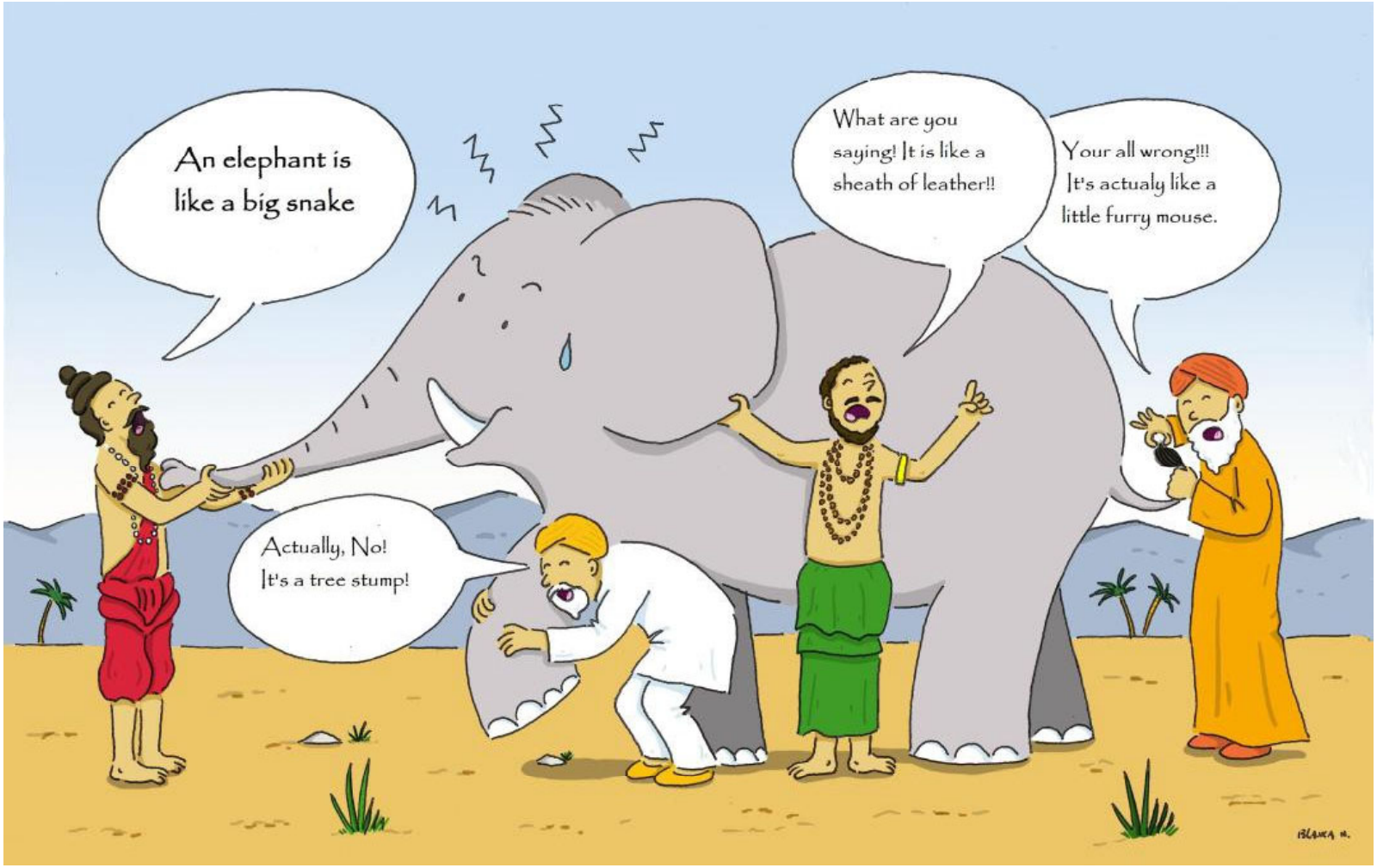
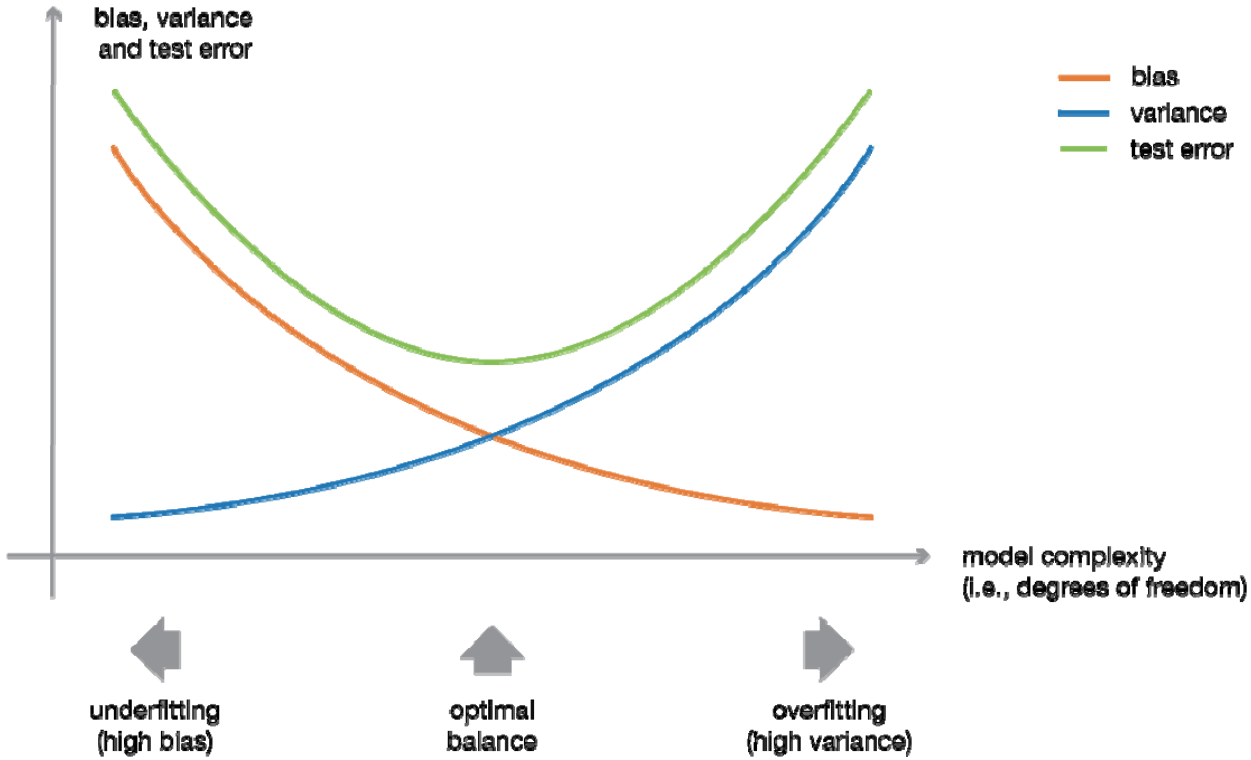


BLIND MEN & ELEPHANT PARABLE. [Source]

Bias Variance Tradeoff



Ensemble Learning

- Ensemble learning is a machine learning paradigm where multiple models (often called **“weak learners**”) are trained to solve the same problem and combined to get better results.
- The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models.

- Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to **decrease variance** (bagging), **bias** (boosting), or **improve predictions** (stacking).

Ensemble Methods

Ensemble methods can be divided into two groups:

- ***sequential*** ensemble methods where the base learners are generated sequentially (e.g. AdaBoost).
  The basic motivation of sequential methods is to **exploit the dependence between the base learners.** The overall performance can be boosted by weighing previously mislabeled examples with higher weight.

- ***parallel*** ensemble methods where the base learners are generated in parallel (e.g. Random Forest).
  The basic motivation of parallel methods is to **exploit independence between the base learners** since the error can be reduced dramatically by averaging.

Simple Ensemble Techniques

- Max Voting
- Averaging
- Weighted Averaging

Max Voting

- The max voting method is generally used for classification problems.
- The predictions which we get from the majority of the models are used as the final prediction.

```
from sklearn.ensemble import VotingClassifier
model1 = LogisticRegression(random_state=1)
model2 = tree.DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='har
d')
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

Averaging

- In this method, we take an average of predictions from all the models and use it to make the final prediction.

- Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

```
model1 = tree.DecisionTreeClassifier()

model2 = KNeighborsClassifier()

model3= LogisticRegression()


model1.fit(x_train,y_train)

model2.fit(x_train,y_train)

model3.fit(x_train,y_train)


pred1=model1.predict_proba(x_test)

pred2=model2.predict_proba(x_test)

pred3=model3.predict_proba(x_test)


finalpred=(pred1+pred2+pred3)/3
```

Weighted Average

- This is an extension of the averaging method.
- All models are assigned different weights defining the importance of each model for prediction.

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict_proba(x_test)
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1*0.3+pred2*0.3+pred3*0.4)
```

Advanced Ensemble techniques

1. Stacking
2. Blending
3. Bagging
4. Boosting

Stacking

- Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor.

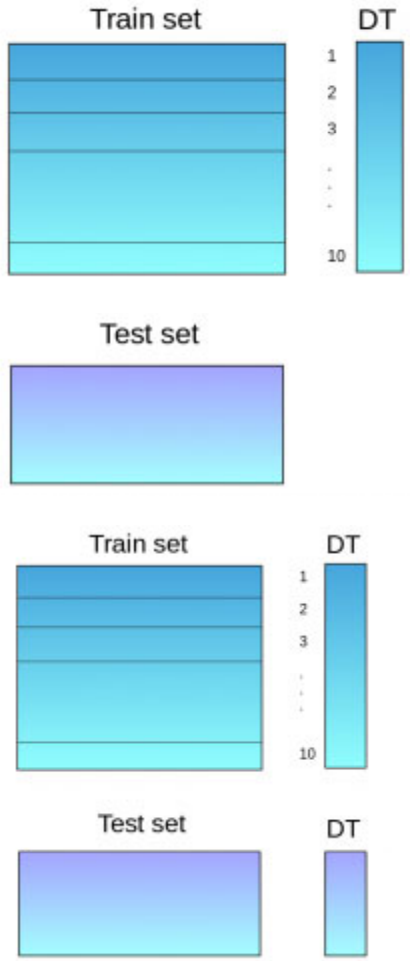
- The base level models are trained based on a complete training set, then the **meta-model is trained on the outputs of the base level model as features**.
- The base level often consists of different learning algorithms and therefore stacking ensembles are often **heterogeneous**.

https://towardsdatascience.com/stacking-made-easy-with-sklearn-e27a0793c92b

Stacking Steps

1. The train set is split into 10 parts.
2. A base model (suppose a decision tree) is fitted on 9 parts and predictions are made for the 10th part. This is done for each part of the train set.

3. The base model (in this case, decision tree) is then fitted on the whole train dataset.
4. Using this model, predictions are made on the test set.

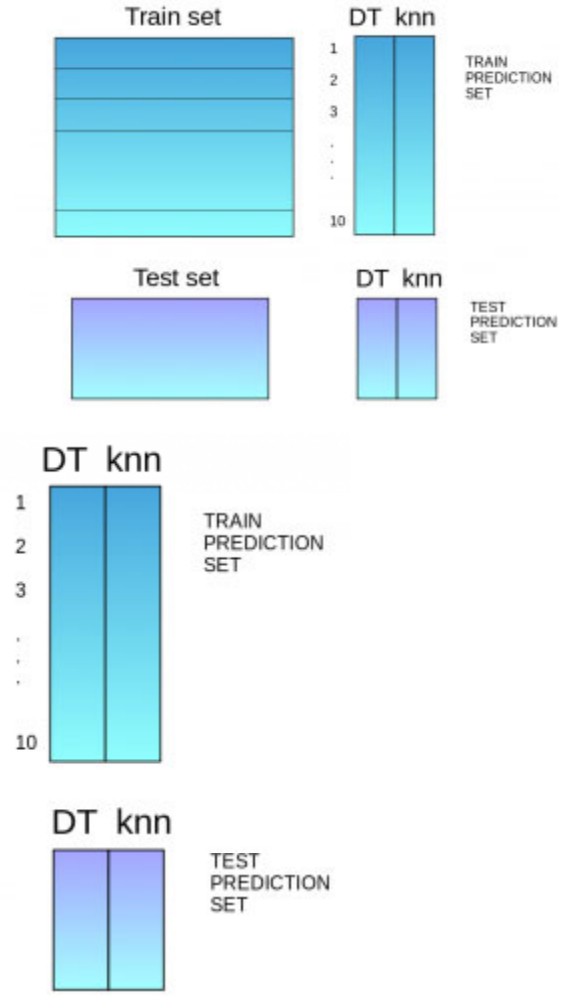


Stacking Steps

5. Steps 2 to 4 are repeated for another base model (say knn) resulting in another set of predictions for the train set and test set.

6. The predictions from the train set are used as features to build a new model.

7. This model is used to make final predictions on the test prediction set.



Stacking

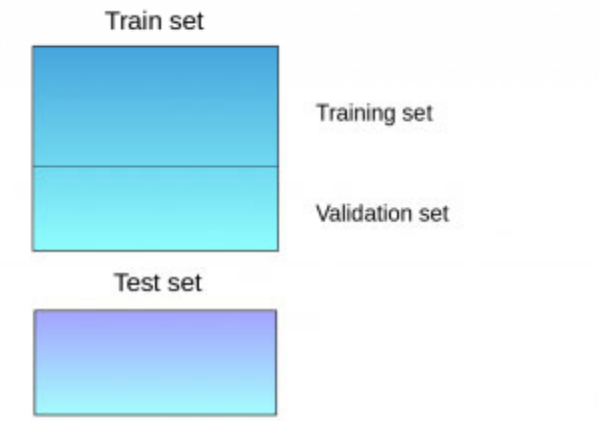
```python
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier
X, y = load_iris(return_X_y=True)
estimators = [
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
    ('svr', make_pipeline(StandardScaler(),
                          LinearSVC(random_state=42)))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression()
)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=42
)
clf.fit(X_train, y_train).score(X_test, y_test)
```

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html

Blending

- Blending follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions.
- In other words, unlike stacking, the predictions are made on the holdout set only.
- The holdout set and the predictions are used to build a model which is run on the test set.

1. The train set is split into training and validation sets



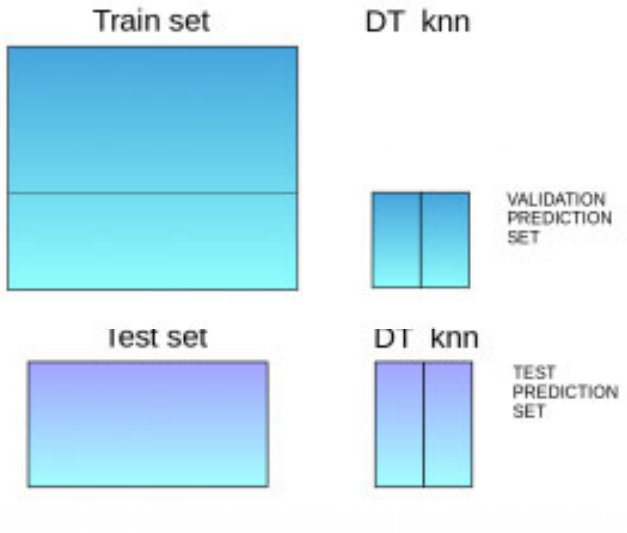
Blending Steps

2. Model(s) are fitted on the training set.

3. The predictions are made on the validation set and the test set.

4. The validation set and its predictions are used as features to build a new model.

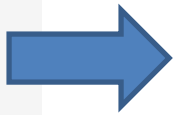
5. This model is used to make final predictions on the test and meta-features.



Blending Code

```
model1 = tree.DecisionTreeClassifier()
model1.fit(x_train, y_train)
val_pred1=model1.predict(x_val)
test_pred1=model1.predict(x_test)
val_pred1=pd.DataFrame(val_pred1)
test_pred1=pd.DataFrame(test_pred1)

model2 = KNeighborsClassifier()
model2.fit(x_train,y_train)
val_pred2=model2.predict(x_val)
test_pred2=model2.predict(x_test)
val_pred2=pd.DataFrame(val_pred2)
test_pred2=pd.DataFrame(test_pred2)
```



```
df_val=pd.concat([x_val, val_pred1,val_pred2],axis=1)
df_test=pd.concat([x_test, test_pred1,test_pred2],axis=1)

model = LogisticRegression()
model.fit(df_val,y_val)
model.score(df_test,y_test)
```

Bagging

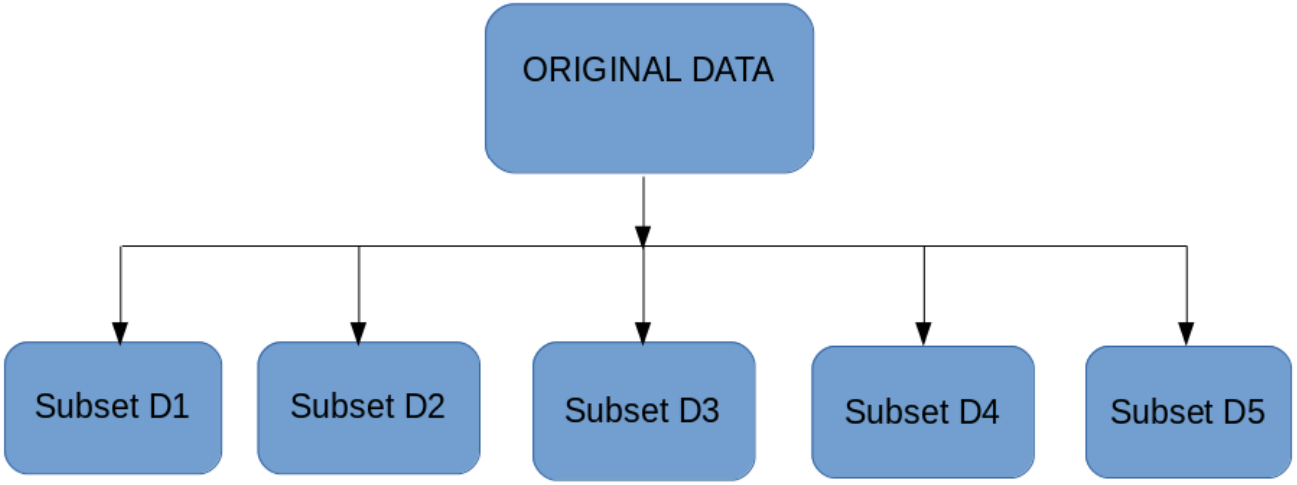
- **Bagging stands for bootstrap aggregation.** One way to reduce the variance of an estimate is to average together multiple estimates.

$$f(x) = 1/M \sum_{m=1}^{M} f_m(x)$$

- Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses *voting for classification* and *averaging for regression*.

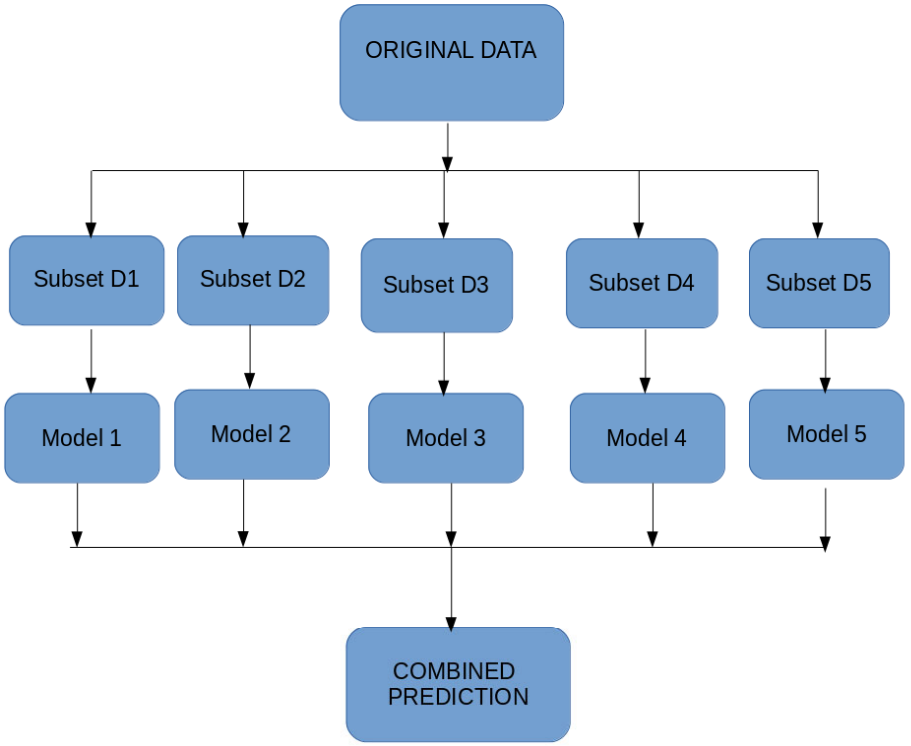
Bagging

- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**. The size of the subsets is the same as the size of the original set.
- Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.



Bagging - Steps

- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.



Random Forest

- The base estimators in random forest are decision trees.
- Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.
- Random subsets are created from the original dataset (bootstrapping).
- **At each node in the decision tree, only a random set of features are considered to decide the best split.**
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

RF Parameters:

- **n_estimators:**
  - It defines the number of decision trees to be created in a random forest.
  - Generally, a higher number makes the predictions stronger and more stable, but a very large number can result in higher training time.
- **criterion**:
  - It defines the function that is to be used for splitting.
  - The function measures the quality of a split for each feature and chooses the best split.
- **max_features** :
  - It defines the maximum number of features allowed for the split in each decision tree.
  - Increasing max features usually improve performance but a very high number can decrease the diversity of each tree.

RF Parameters:

- **max_depth**:
  - Random forest has multiple decision trees. This parameter defines the maximum depth of the trees.

- **min_samples_split:**
  - Used to define the minimum number of samples required in a leaf node before a split is attempted.
  - If the number of samples is less than the required number, the node is not split.

- **min_samples_leaf:**
  - This defines the minimum number of samples required to be at a leaf node.
  - Smaller leaf size makes the model more prone to capturing noise in train d

RF Parameters:

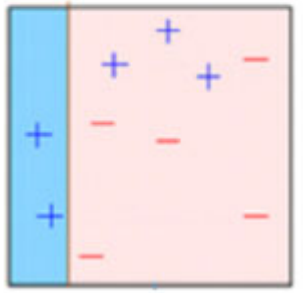
- **max_leaf_nodes:**
  - This parameter specifies the maximum number of leaf nodes for each tree.
  - The tree stops splitting when the number of leaf nodes becomes equal to the max leaf node.

- **n_jobs**:
  - This indicates the number of jobs to run in parallel.
  - Set value to -1 if you want it to run on all cores in the system.
- **random_state**:
  - This parameter is used to define the random selection.
  - It is used for comparison between various models.

Boosting

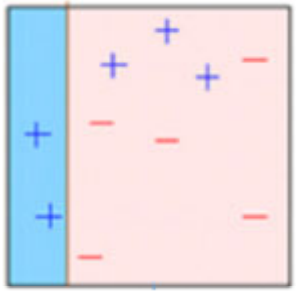
- **Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.** The succeeding models are dependent on the previous model.

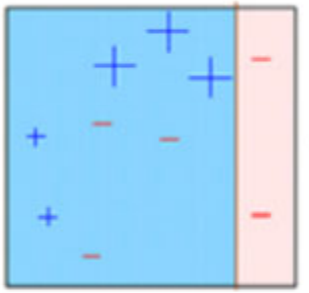
1. A subset is created from the original dataset.
2. Initially, all data points are given equal weights.
3. A base model is created on this subset.
4. This model is used to make predictions on the whole dataset.



Boosting - Steps

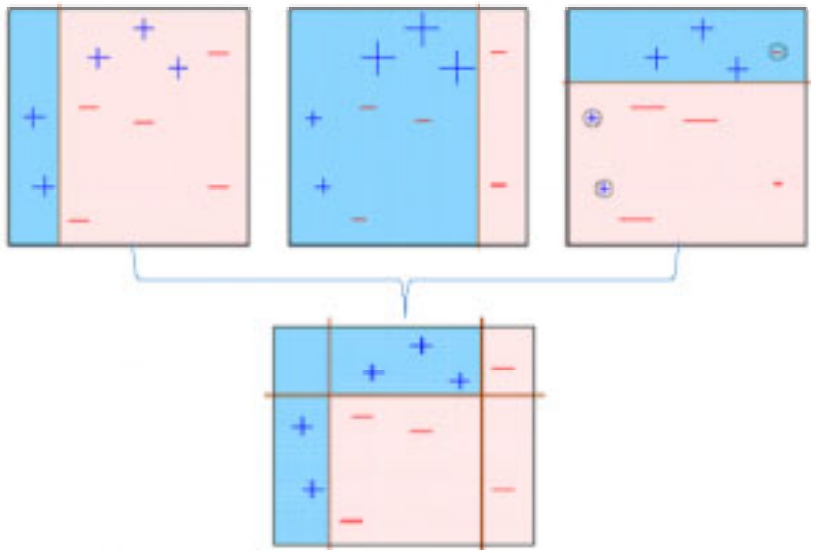
5. Errors are calculated using the actual values and predicted values.
6. The observations which are incorrectly predicted, are given higher weights.
   (Here, the three misclassified blue-plus points will be given higher weights)
7. Another model is created and predictions are made on the dataset.
   (This model tries to correct the errors from the previous model)





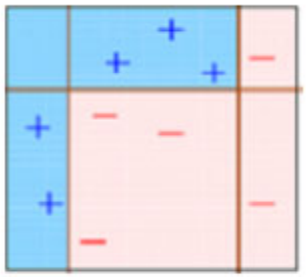
Boosting - Steps

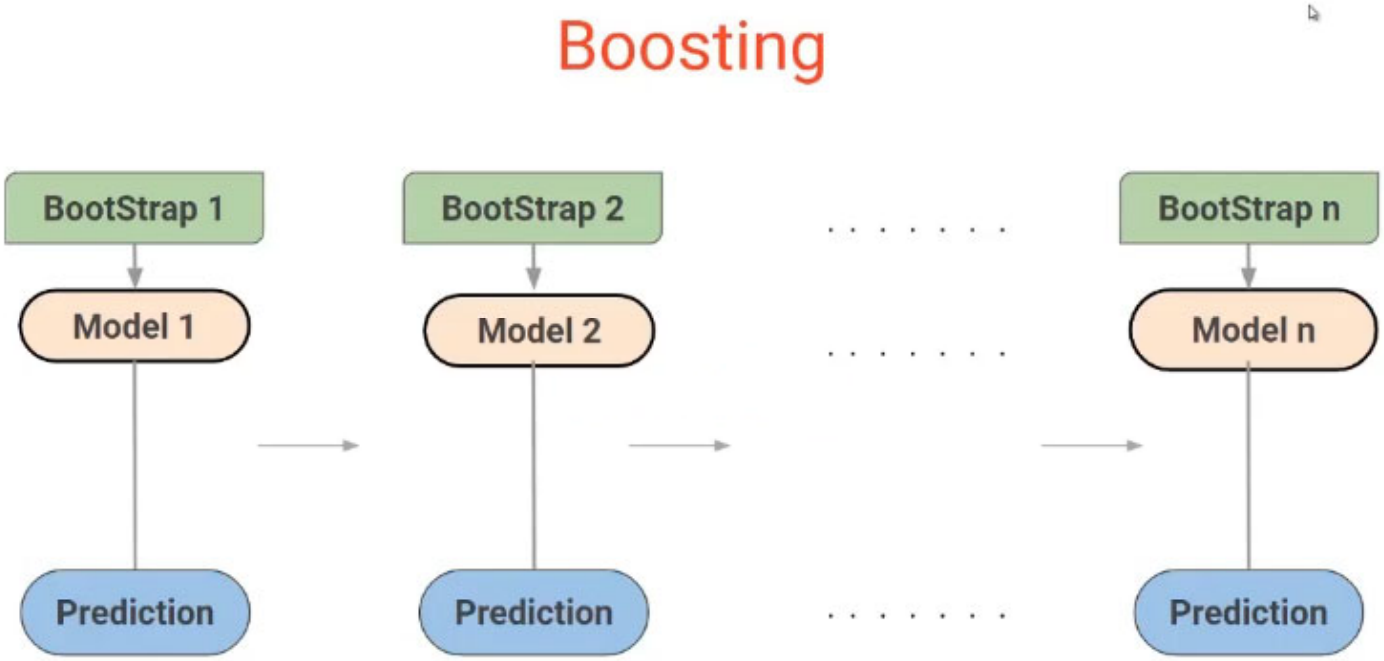
8. Similarly, multiple models are created, each correcting the errors of the previous model.
9. The final model (strong learner) is the weighted mean of all the models (weak learners).



Boosting - Steps

- Thus, the boosting algorithm combines a number of weak learners to form a strong learner.
- The individual models would not perform well on the entire dataset, but they work well for some part of the dataset.
- Thus, each model actually boosts the performance of the ensemble.





Stopping Criterion for Boosting:

1. Error Becomes zero (all sample points are correctly classified/regressed
2. There are constraint on the number of estimators to be used

Boosting Algorithms

- AdaBoost
- GBM
- XGBM
- Light GBM
- CatBoost

Bagging vs Boosting: Summary

	Bagging	Boosting
Similarities	• Uses voting • Combines models of the same type	
Differences	Individual models are built separately	Each new model is influenced by the performance of those built previously
	Equal weight is given to all models	Weights a model's contribution by its performance

Advantages/Benefits of ensemble methods

1. **More accurate prediction results**
2. **Stable and more robust model**- The aggregate result of multiple models is always less noisy than the individual models. This leads to model stability and robustness.
3. Ensemble models can be used to **capture the linear as well as the non-linear relationships** in the data. This can be accomplished by using 2 different models and forming an ensemble of the two.

Disadvantages of ensemble methods

1. **Reduction in model interpretability-** Using ensemble methods reduces the model interpret-ability due to increased complexity and makes it very difficult to draw any crucial business insights at the end.

2. **Computation and design time is high-** It is not good for real time applications.

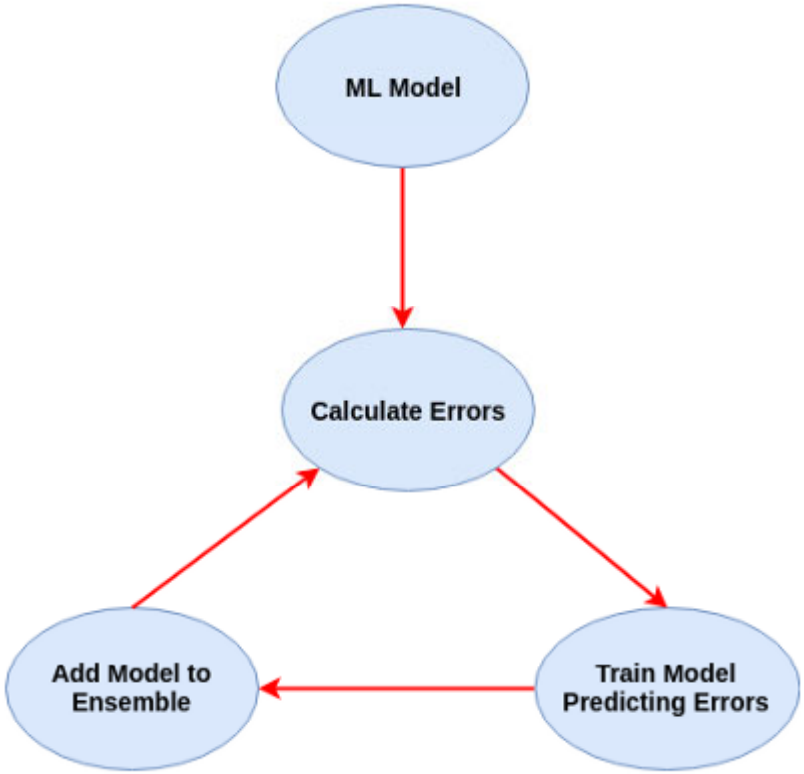
3. The selection of models for creating an ensemble is an art which is really hard to master.

Boosting Algorithms

- Boosting algorithms:
- AdaBoost
- GBM
- XGBM
- Light GBM
- CatBoost

Gradient Boosting

- Models are built sequentially
- Subsequent models focus on reducing the error
- Models are created over the residuals



Steps to build GB Model

1. Build a model and make predictions
2. Cal the error and set this as Target
3. Build model on the errors and make predictions
4. This predicted error is added to the predicted target in step 1.
5. Repeat steps 2-4.

GBM-1

ID	Age	City	Income	Model 1 Income
1	32	A	51000	53500
2	30	B	78000	61000
3	21	A	20000	28500
4	27	B	44000	61000
5	36	B	89000	90500
6	25	A	37000	28500
7	47	A	56000	53500
8	54	B	92000	90500

TARGET	PREDICTION	RESIDUAL
Income	Predictions	
51000	53500	
78000	61000	
20000	28500	
44000	61000	
89000	90500	
37000	28500	
56000	53500	
92000	90500	

GBM-2

ID	Age	City	Income	Model 1 Income
1	32	A	51000	53500
2	30	B	78000	61000
3	21	A	20000	28500
4	27	B	44000	61000
5	36	B	89000	90500
6	25	A	37000	28500
7	47	A	56000	53500
8	54	B	92000	90500

TARGET	PREDICTION	RESIDUAL
Income	Predictions	Error
51000	53500	-2500
78000	61000	17000
20000	28500	-8500
44000	61000	-17000
89000	90500	-1500
37000	28500	8500
56000	53500	2500
92000	90500	1500

TARGET
Error
-2500
17000
-8500
-17000
-1500
8500
2500
1500

GBM-3

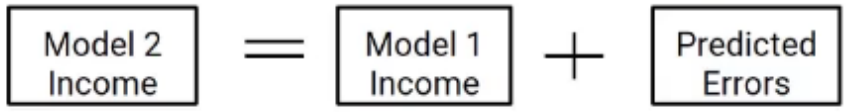
ID	Age	City	Income	Model 1 Income
1	32	A	51000	53500
2	30	B	78000	61000
3	21	A	20000	28500
4	27	B	44000	61000
5	36	B	89000	90500
6	25	A	37000	28500
7	47	A	56000	53500
8	54	B	92000	90500

TARGET	PREDICTION	RESIDUAL
Error	Predictions	
-2500	-5500	
17000	8000	
-8500	-5500	
-17000	-4300	
-1500	8000	
8500	8000	
2500	-4300	
1500	-4300	

GBM-4

ID	Age	City	Income	Model 1 Income	Model 2 Income
1	32	A	51000	53500	48000
2	30	B	78000	61000	69000
3	21	A	20000	28500	23000
4	27	B	44000	61000	56700
5	36	B	89000	90500	98500
6	25	A	37000	28500	36500
7	47	A	56000	53500	49200
8	54	B	92000	90500	86200

TARGET	PREDICTION	RESIDUAL
Error	Predictions	
-2500	-5500	
17000	8000	
-8500	-5500	
-17000	-4300	
-1500	8000	
8500	8000	
2500	-4300	
1500	-4300	



GBM-5

ID	Age	City	Income	Model 1 Income	Model 2 Income
1	32	A	51000	53500	48000
2	30	B	78000	61000	69000
3	21	A	20000	28500	23000
4	27	B	44000	61000	56700
5	36	B	89000	90500	98500
6	25	A	37000	28500	36500
7	47	A	56000	53500	49200
8	54	B	92000	90500	86200

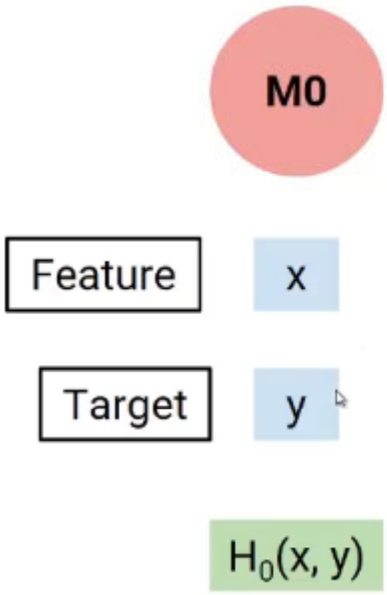
TARGET	PREDICTION	RESIDUAL
Error	Predictions	New Error
-2500	-5500	3000
17000	8000	9000
-8500	-5500	-3000
-17000	-4300	-12700
-1500	8000	-9500
8500	8000	500
2500	-4300	6800
1500	-4300	5800

GBM-6

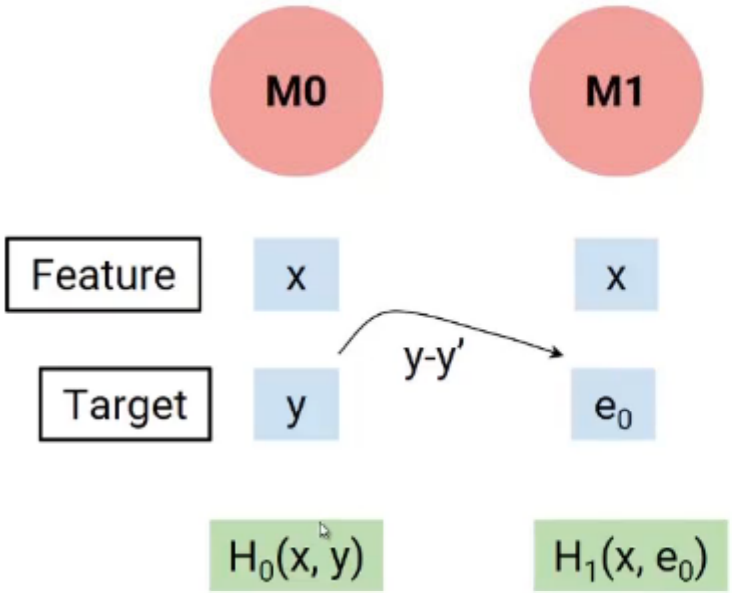
ID	Age	City	Income	Model 1 Income	Model 2 Income
1	32	A	51000	53500	48000
2	30	B	78000	61000	69000
3	21	A	20000	28500	23000
4	27	B	44000	61000	56700
5	36	B	89000	90500	98500
6	25	A	37000	28500	36500
7	47	A	56000	53500	49200
8	54	B	92000	90500	86200

TARGET	PREDICTION	RESIDUAL
New Error		
3000		
9000		
-3000		
-12700		
-9500		
500		
6800		
5800		

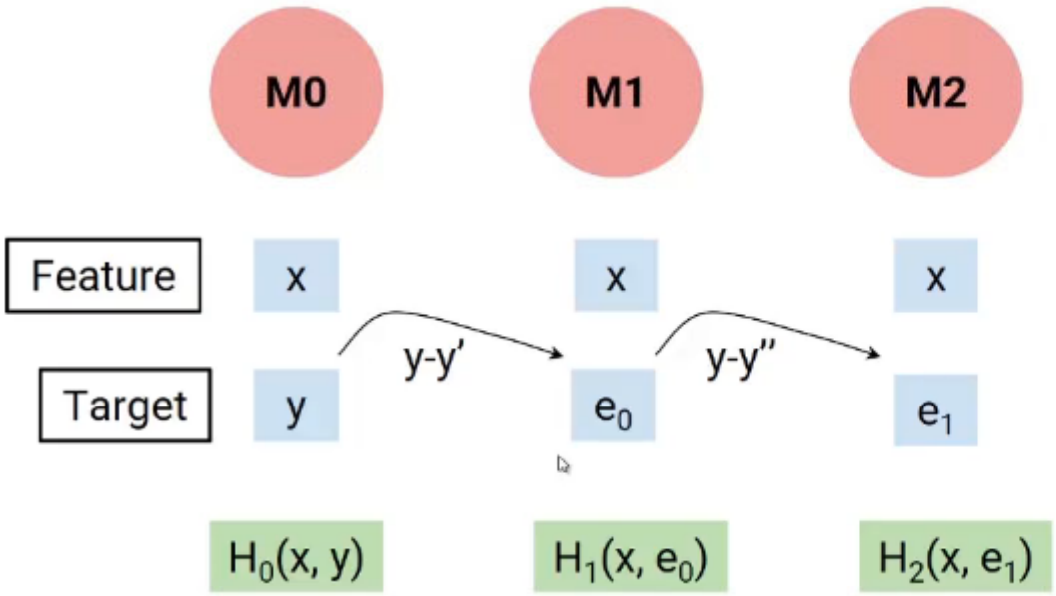
Gradient Boosting Model



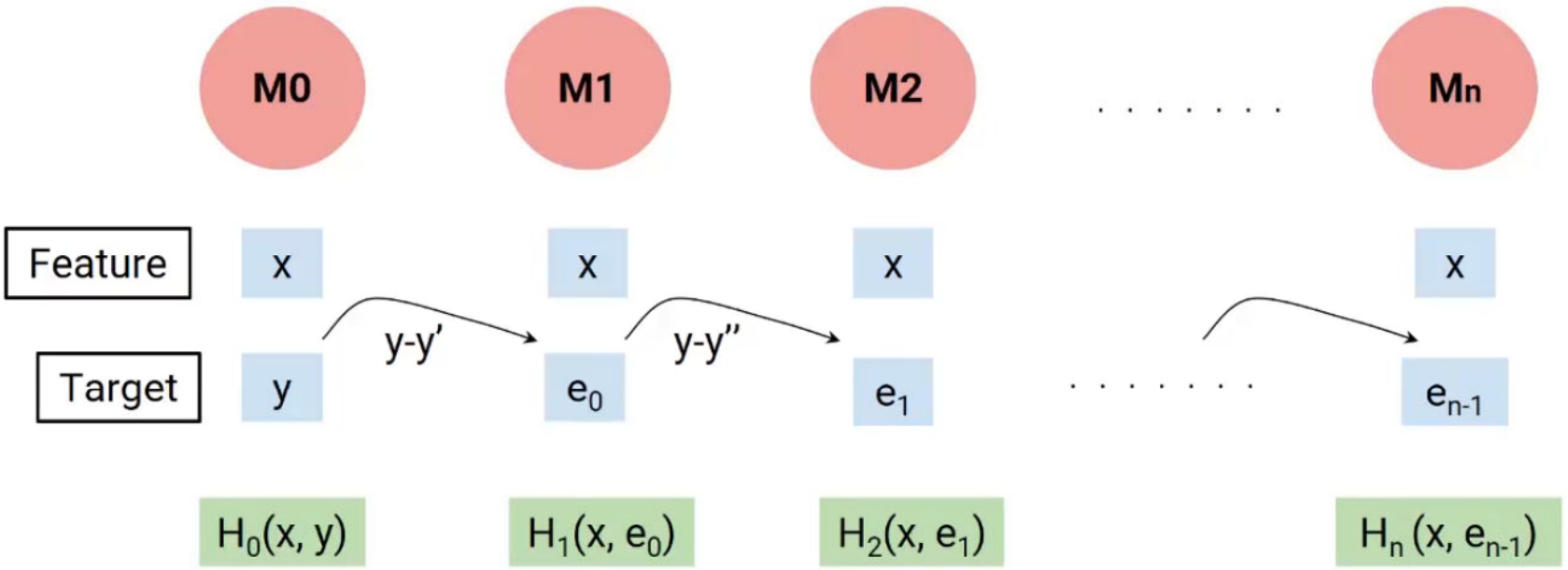
Gradient Boosting Model



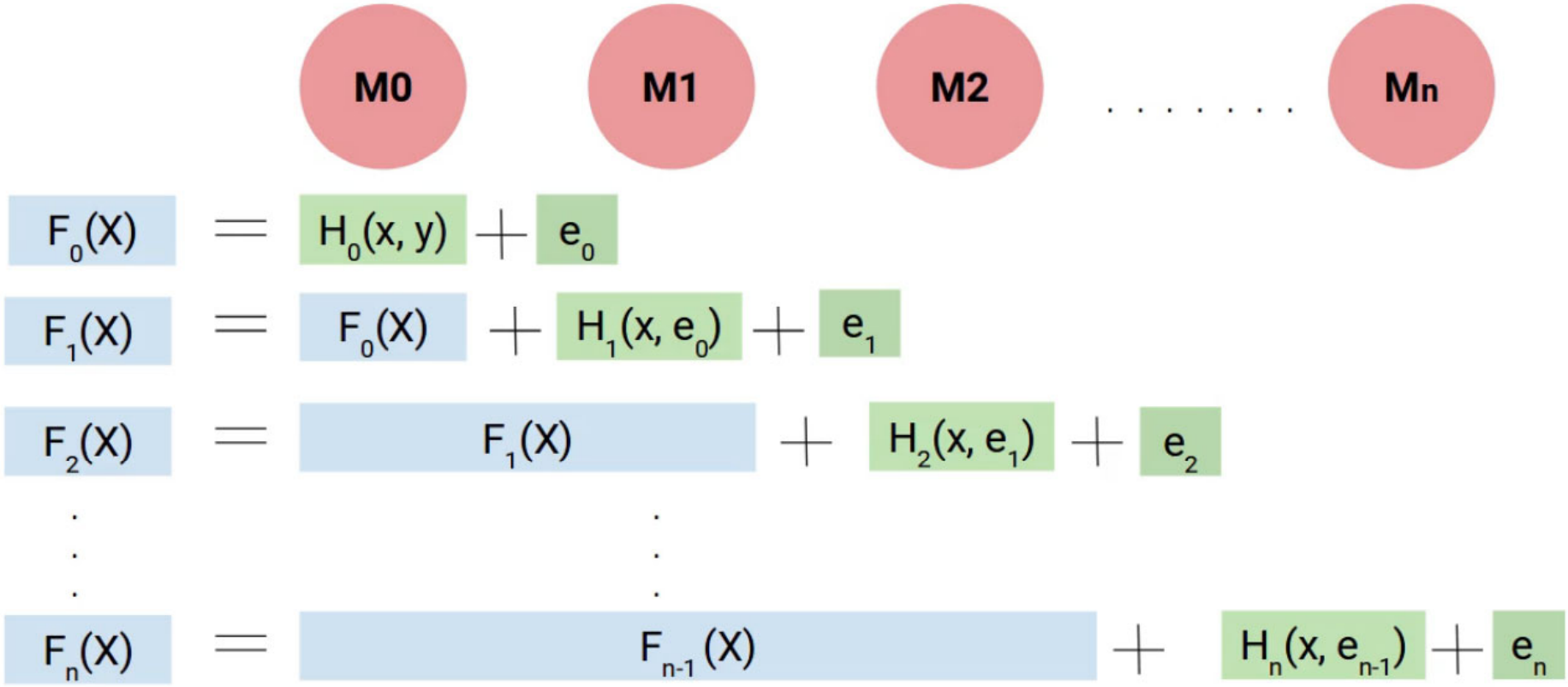
Gradient Boosting Model



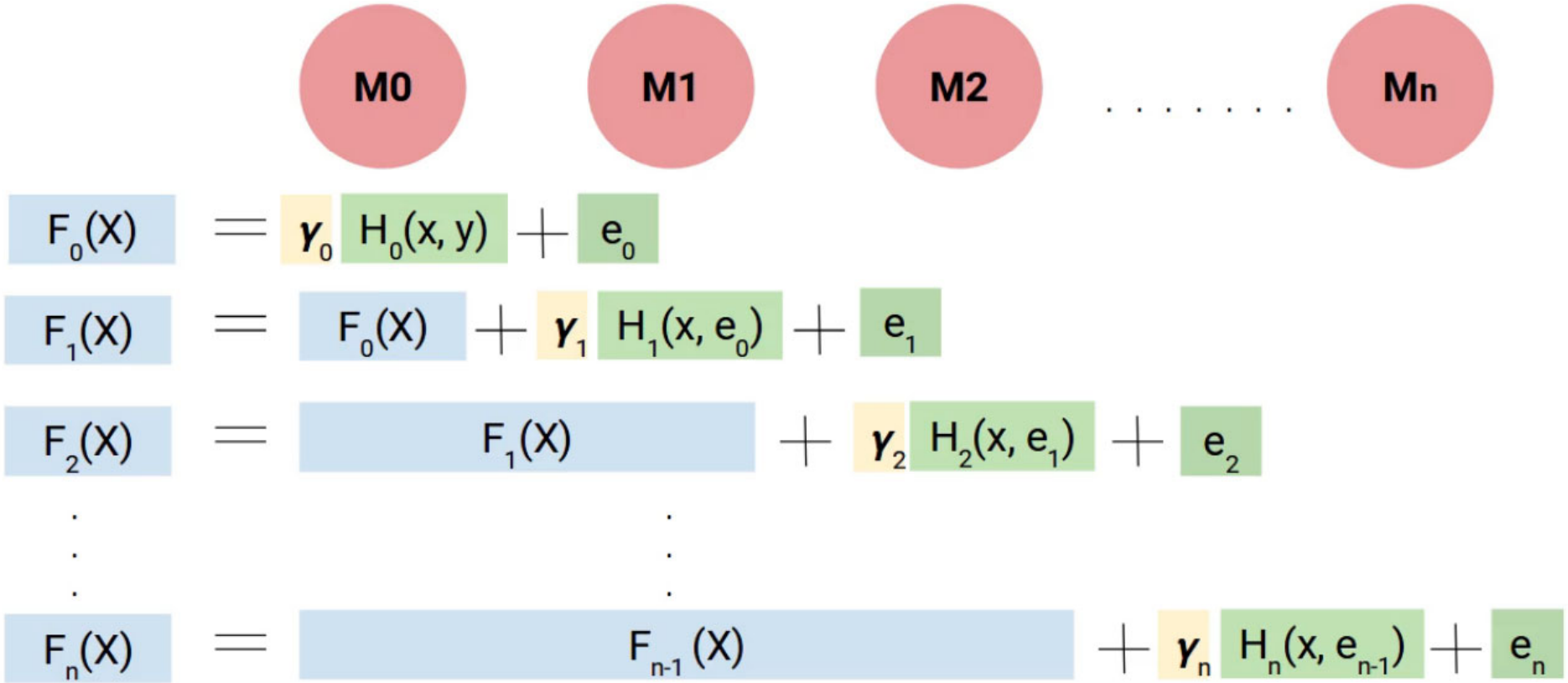
Gradient Boosting Model



Gradient Boosting Model



Gradient Boosting Model



Gradient Boosting Model

$$F_{n+1}(X) = F_n(X) + \gamma_n H(x, e_n)$$

$$F_0(X) = \gamma_0 H_0(x, y) + e_0$$

$$F_1(X) = F_0(X) + \gamma_1 H_1(x, e_0) + e_1$$

$$F_2(X) = F_1(X) + \gamma_2 H_2(x, e_1) + e_2$$

$$\vdots$$

$$F_n(X) = F_{n-1}(X) + \gamma_n H_n(x, e_{n-1}) + e_n$$

Gradient Boosting Model

$$F_{n+1}(X) = F_n(X) + \gamma_n H(x, e_n)$$

$$F_0(X) = \gamma_0 H_0(x, y) + e_0$$

$$F_1(X) = F_0(X) + \gamma_1 H_1(x, e_0) + e_1$$

$$F_2(X) = F_1(X) + \gamma_2 H_2(x, e_1) + e_2$$

$$\vdots \qquad \vdots$$

$$F_n(X) = F_{n-1}(X) + \gamma_n H_n(x, e_{n-1}) + e_n$$

Gradient Boosting Model

$$F_{n+1}(X) = F_n(X) + \gamma_n H(x, e_n)$$

$$L = (y - y')^2$$

$$L = (y - F_n(x))^2$$

$$-\frac{dL}{dF_n(x)} = 2(y - F_n(x))$$

Gradient Boosting Model

$$F_{n+1}(X) = F_n(X) + \gamma_n \, H\left(x, -\frac{dL}{dF_n(x)}\right)$$

$$L = (y - y')^2$$

$$L = (y - F_n(x))^2$$

$$-\frac{dL}{dF_n(x)} = 2(y - F_n(x))$$

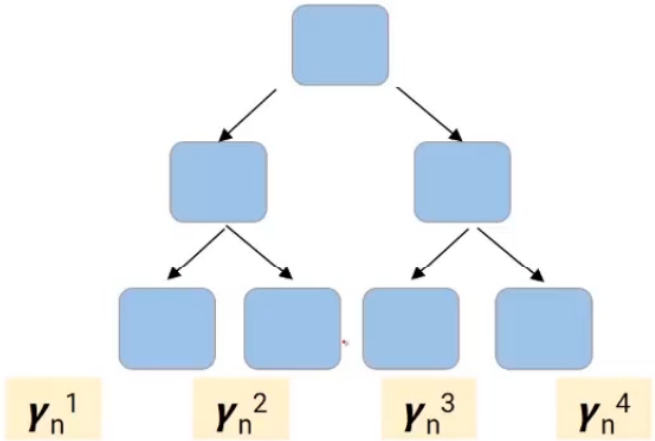
Gradient Boosting Model

$$F_{n+1}(X) = F_n(X) + \gamma_n \, H\left(x, -\frac{dL}{dF_n(x)}\right)$$

$$\text{Loss} = L(y, F_n(x)) + \gamma_n \, L\left(H\left(x, -\frac{dL}{dF_n(x)}\right)\right)$$

Gradient Boosting Decision Tree

$$F_{n+1}(X) = F_n(X) + \gamma_n \; H\left(x, -\frac{dL}{dF_n(x)}\right)$$



γ is calculated at the "Leaf-Level", not node level !

Extreme Gradient Boosting

- Extreme Gradient Boosting Machine (XGBoost)
- Working procedure same as GBM
- Subsequent trees focus on reducing the error
- Designed for speed and performance

Features of Extreme Gradient Boosting

Regularization

Parallel Processing

Handling Missing Values

Out of core Computing

- XGBoost has an option to penalize complex models through both L1 and L2 regularization

Features of Extreme Gradient Boosting

Regularization

Parallel Processing

Handling Missing Values

Out of core Computing

- XGBoost implements parallel processing and hence is faster than GBM
- Parallelizing the node building at each level

Features of Extreme Gradient Boosting

Regularization

Parallel Processing

Handling Missing Values

Out of core Computing

- XGBoost has an in-built routine to handle missing values

Features of Extreme Gradient Boosting

Regularization

Parallel Processing

Handling Missing Values

Out of core Computing

- XGBoost is designed to be memory efficient
- Uses out-of-core computing for very large datasets that don't fit in the memory

Features of Extreme Gradient Boosting

Regularization

Parallel Processing

Handling Missing Values

Out of core Computing

Built-in cross validation

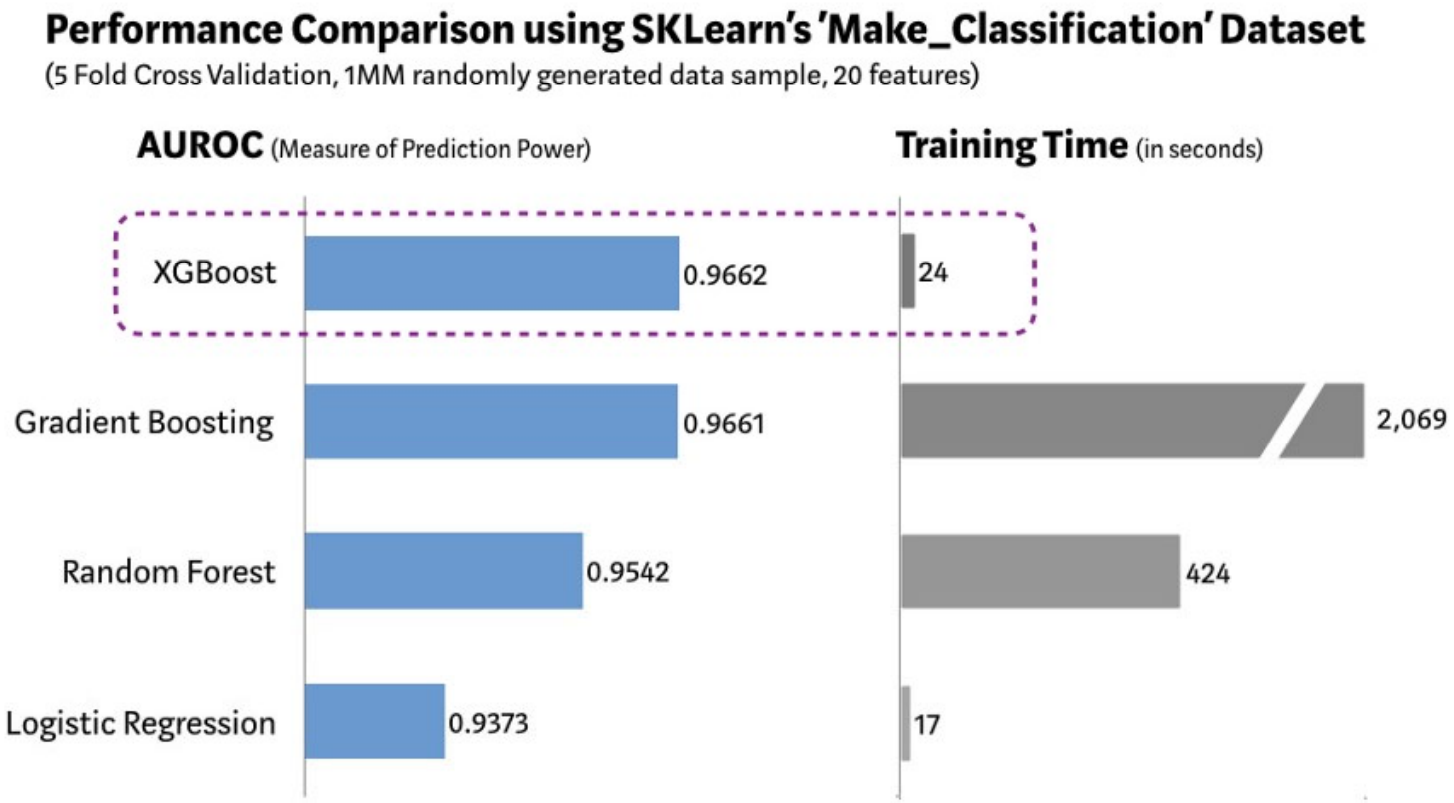
- XGB allows user to run a cross validation at each iteration of the boosting process

Why does XGBoost perform so well?

Algorithmic Enhancements in XGBoost

1. **Regularization**: It penalizes more complex models through both LASSO (L1) and Ridge (L2) regularization to prevent overfitting.
2. **Sparsity Awareness**: XGBoost naturally admits sparse features for inputs by automatically 'learning' best missing value depending on training loss and handles different types of sparsity patterns in the data more efficiently.
3. **Weighted Quantile Sketch:** XGBoost employs the distributed weighted Quantile Sketch algorithm to effectively find the optimal split points among weighted datasets.
4. **Cross-validation**: The algorithm comes with built-in cross-validation method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run.

Need some Proof?



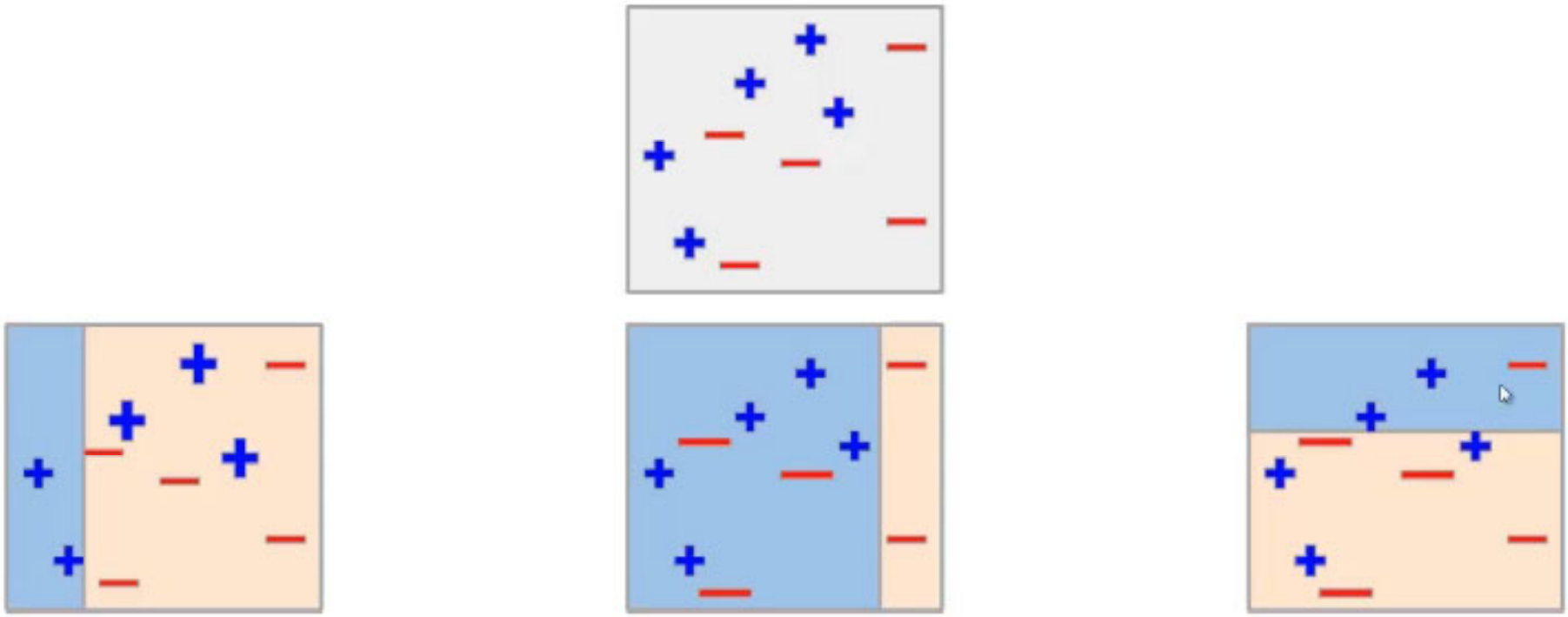
Adaptive Boosting

Higher weights are assigned to data points which are incorrectly predicted.

Steps to build Adaptive Boosting Model

- **Step 1:** Build a model and make predictions
- **Step 2:** Assign higher weight to misclassified points
- **Step 3:** Build next model
- **Step 4:** Repeat Step 2 and Step 3
- **Step 5:** Weighted average of individual models

Steps to build Adaptive Boosting Model



Steps to build Adaptive Boosting Model