IS4303 Week 6
Model Tuning & Ensemble Learning

Agenda

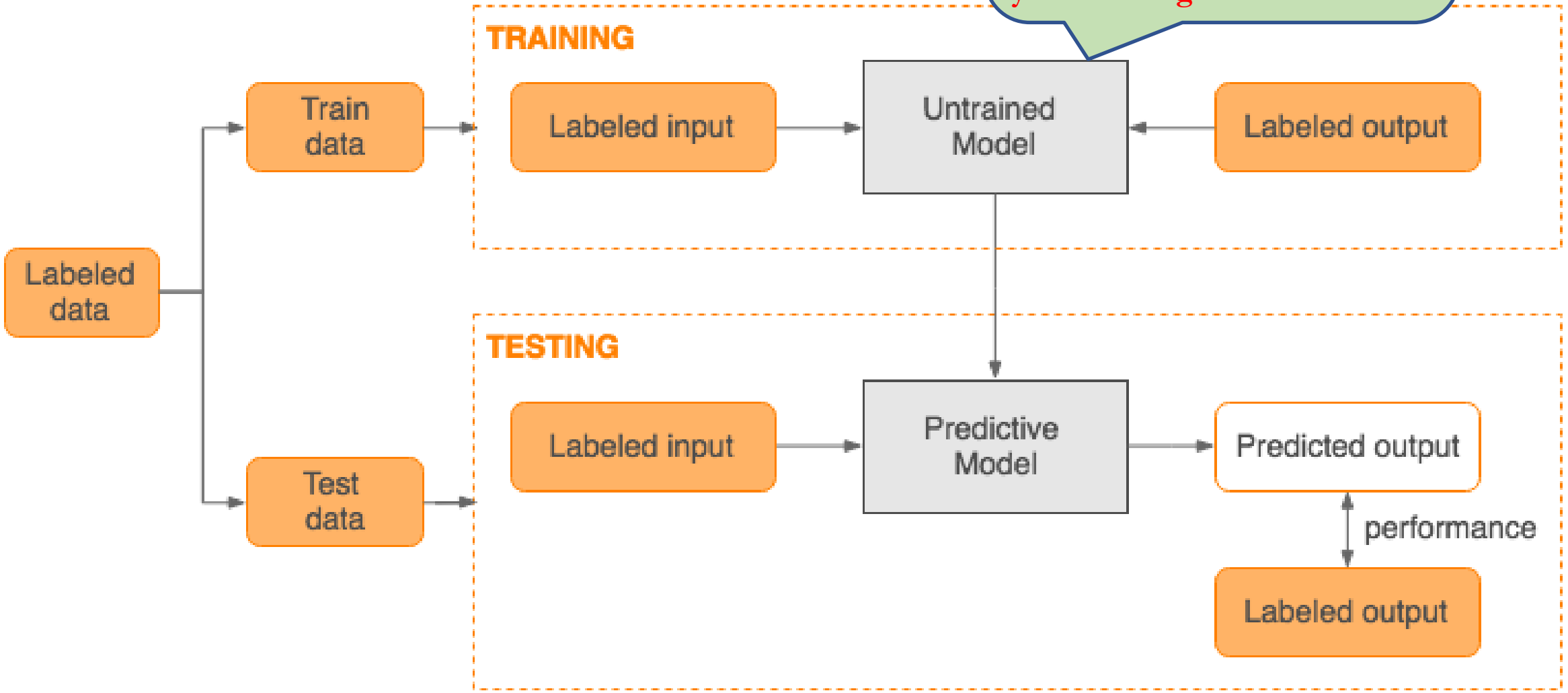
- Model Tuning
  - Hyperparameter Optimization: Grid-Search
  - Cross-Validation
- Alternative Performance Metrics
  - Confusion Matrix: Sensitivity v.s. Precision
  - ROC and AUC
- Tree-Based Models
  - Decision Tree and Ensemble Learning Methods

Model Prediction

1. Decision Tree
2. Lasso/Ridge Regression
3. K Nearest Neighbor

More……

♠ **What hyperparameters are you choosing?**



Hyperparameter Optimization

$$\text{Lasso: } \min_{\beta_0,\ldots,\beta_j} \mathrm{L}(\beta) = RSS(\beta) + \lambda \sum_{j=1}^{p} |\beta_j| = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- Case:
  - Finding the BEST lambda value (regularization parameter value) in Lasso regression
  - BEST: This lambda results in best performance (e.g., highest accuracy)

- Grid Search on hyperparameters (e.g., exhaustive method):
  - Fix a subset of possible hyperparameter values
  - Specify the performance metrics (e.g., accuracy, or others)
  - Simple validation (or cross-validation) on all these values
  - Select the BEST hyperparameter value

Hyperparameter Optimization

- Fix a subset of possible hyperparameter values
  - Use: numpy.logspace(start, stop, number)
  - https://docs.scipy.org/doc/numpy/reference/generated/numpy.logspace.html
  - Example: lbd = numpy.logspace(-3, 3, 5)
  - First, generate 5 values that are evenly spaced between -3 and 3 (i.e., arithmetic sequence): [-3, -1.5, 0, 1.5, 3]
  - Then, return their exponential values [10^{-3}, $10^{-1.5}$, 10^{0}, $10^{1.5}$, 10^{3}] as "lbd" values

```
> # Fix a subset of possible hyperparameter values
> import numpy as np
> lbd = np.logspace(-9, 9, 1000)
> print(lbd) # Show values
```

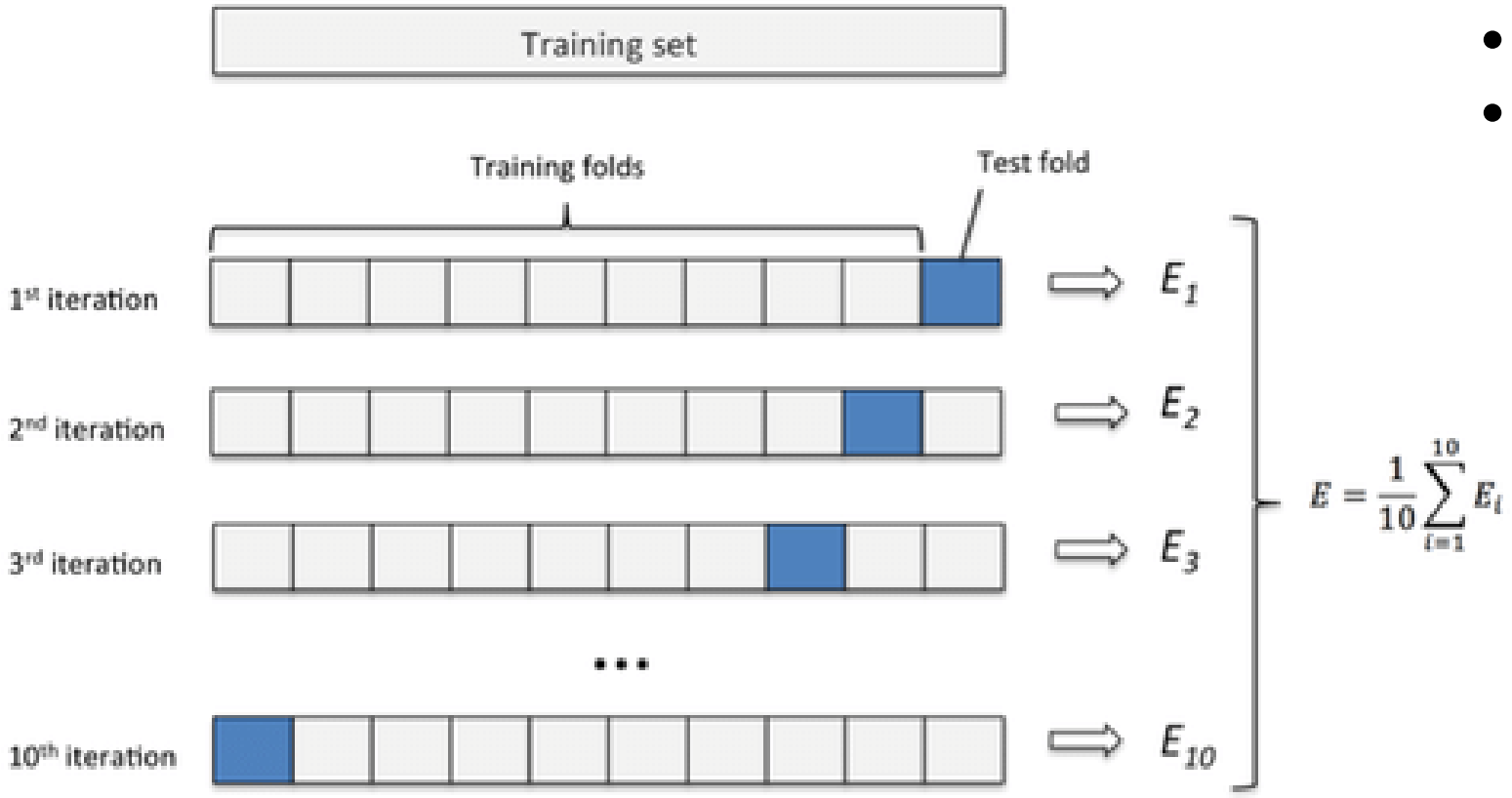
Hyperparameter Optimization

- Specify the performance metrics (e.g., accuracy, or others)
- Simple validation (or cross-validation) on all these values
- Select the BEST hyperparameter value

Please check “Step 3.2” in the Jupyter Notebook

Cross Validation

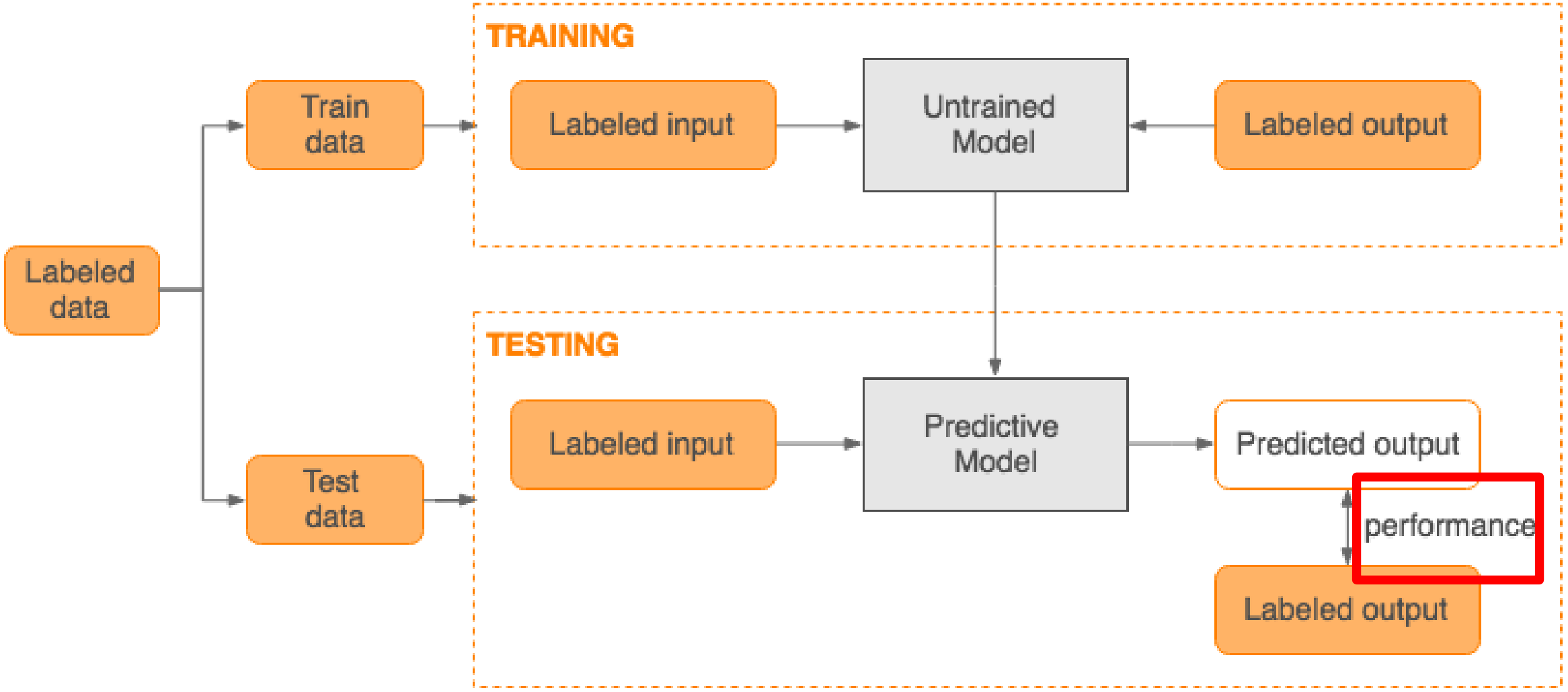
- K-Fold Cross Validation (e.g., K=10)



- Model Evaluation
- Model Comparison
- **Model Tuning**

K = 3: ≈ 70/30 split;
K = 5: = 80/20 split;
K = 10: = 90/10 split

Model Prediction



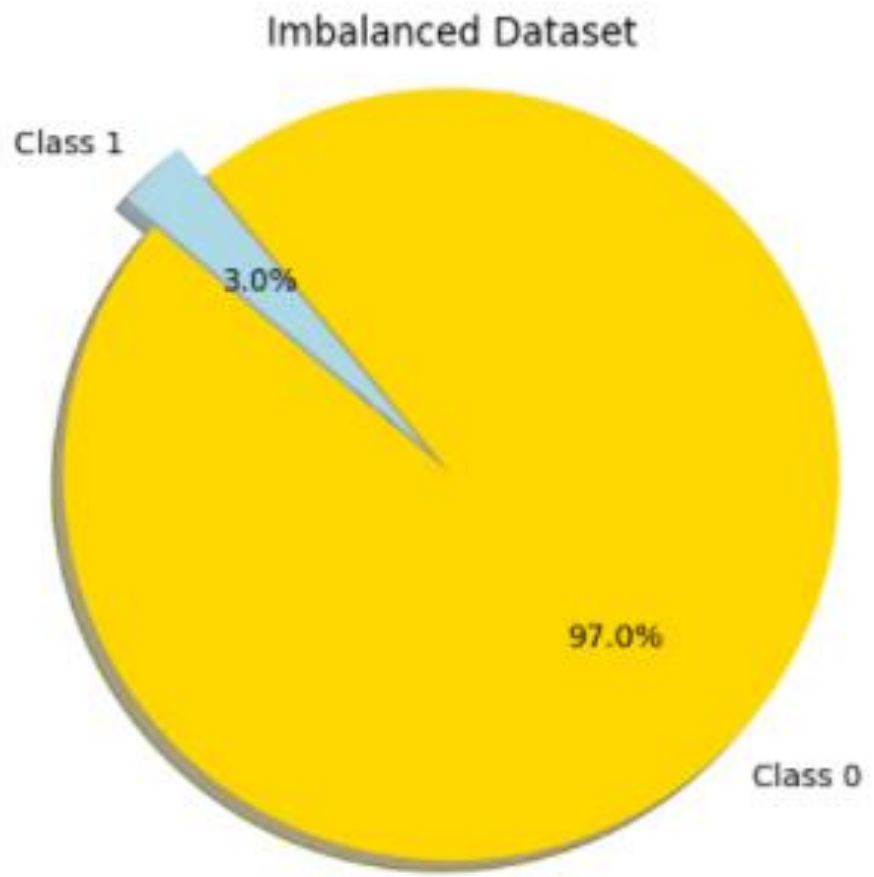
Performance of Binary Classifier

- Problems of **Accuracy/Error** metrics:

- Example: Cancer Detection
  - Class 1 (Positive): Having cancer
  - Class 0 (Negative): Being Healthy

- What if your model misclassified all the "Class 1" cases but correctly classified all the "Class 0" cases?

♦ **Your model accuracy is 97%, but do you think this is a good model?**



Performance of Binary Classifier

- Confusion Matrix:

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

Sensitivity (or Recall, TPR) = TP ÷ (TP+FN)

Precision (or PPR, PPV) = TP ÷ (TP+FP)

True Positive Rate or Hit Rate or Recall or Sensitivity or TP Rate	TP/P	The proportion of positive instances that are correctly classified as positive
False Positive Rate or False Alarm Rate or FP Rate	FP/N	The proportion of negative instances that are erroneously classified as positive
False Negative Rate or FN Rate	FN/P	The proportion of positive instances that are erroneously classified as negative = 1 − True Positive Rate

True Negative Rate or Specificity or TN Rate	TN/N	The proportion of negative instances that are correctly classified as negative
Precision or Positive Predictive Value	TP/(TP+FP)	Proportion of instances classified as positive that are really positive
F1 Score	(2 × Precision × Recall) /(Precision + Recall)	A measure that combines Precision and Recall
Accuracy or Predictive Accuracy	(TP + TN)/(P + N)	The proportion of instances that are correctly classified
Error Rate	(FP + FN)/(P + N)	The proportion of instances that are incorrectly classified

Sensitivity v.s. Precision

Sensitivity (or Recall, TPR) = TP ÷ (TP+FN)

Precision (or PPR, PPV) = TP ÷ (TP+FP)

Do you care more about **FN** or **FP** ?

Case I: New HIV Test Method

	Predicted: NO	**Predicted: YES**
Actual: NO	TN=900	FP=0
Actual: YES	FN=90	TP=10

- 1000 people: 100 are real HIV patients, 900 are healthy people
- Accuracy = (900+10)/1000=91%
- **Sensitivity = 10/(10+90) = 10% ♣**
- **Precision = 1/(1+0) = 100%**

♥ FN in HIV test kills people !

Case II: Advertising Target on Credit Card Users

	Predicted: NO	**Predicted: YES**
Actual: NO	TN=909	FP=81
Actual: YES	FN=1	TP=9

- 1000 people: 10 are really interested in, 900 are not interested at all
- Accuracy=(909+9)/1000=91.8%
- **Sensitivity = 9/(9+1) = 90%**
- **Precision = 9/(9+81) = 10% ♣**

♥ FP in advertising target wastes money !

Performance of Binary Classifier

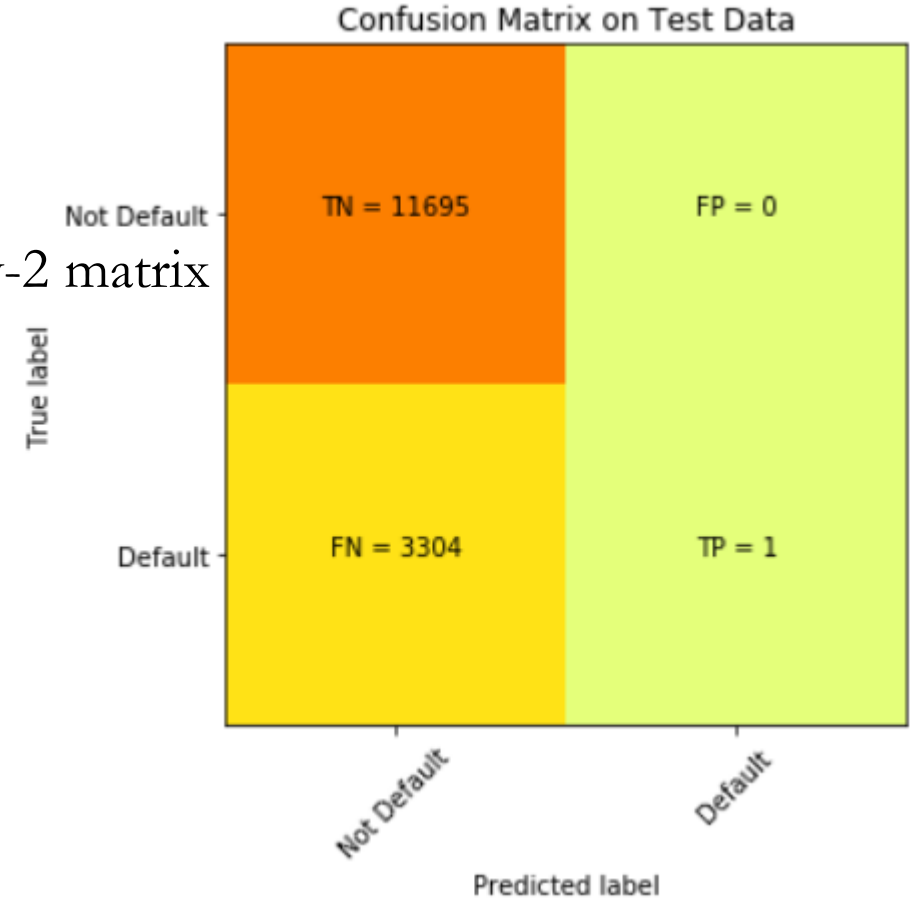
- Confusion Matrix in Python (See notebook):

```
> # Import packages
> from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, classification_report, confusion_matrix

> # Fit a model
> # Get predicted labels for test data
> y_pred = model.predict(X_test)

> cm = confusion_matrix(y_test, y_pred) # Confusion matrix
> tn, fp, fn, tp = cm.ravel() # Get values of tn, fp, fn, and tp
```

Performance of Binary Classifier

- Confusion Matrix in Python (See notebook):

```
> # You have another way to plot confusion matrix
> plt.figure(figsize=(5, 5))
> plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia) # 2-by-2 matrix
> labels = ['Not Default','Default'] # 1=Default, 0=Not Default
> plt.title('Confusion Matrix on Test Data')
> plt.ylabel('True label')
> plt.xlabel('Predicted label')
> tick_marks = np.arange(len(labels)) # [0, 1]
> plt.xticks(tick_marks, labels, rotation=45)
> plt.yticks(tick_marks, labels)
> s = [['TN','FP'], ['FN', 'TP']]
> for i in range(2):
>     for j in range(2):
>         plt.text(j, i, str(s[i][j])+" = "+str(cm[i][j]), horizontalalignment="center")
> plt.show()
```

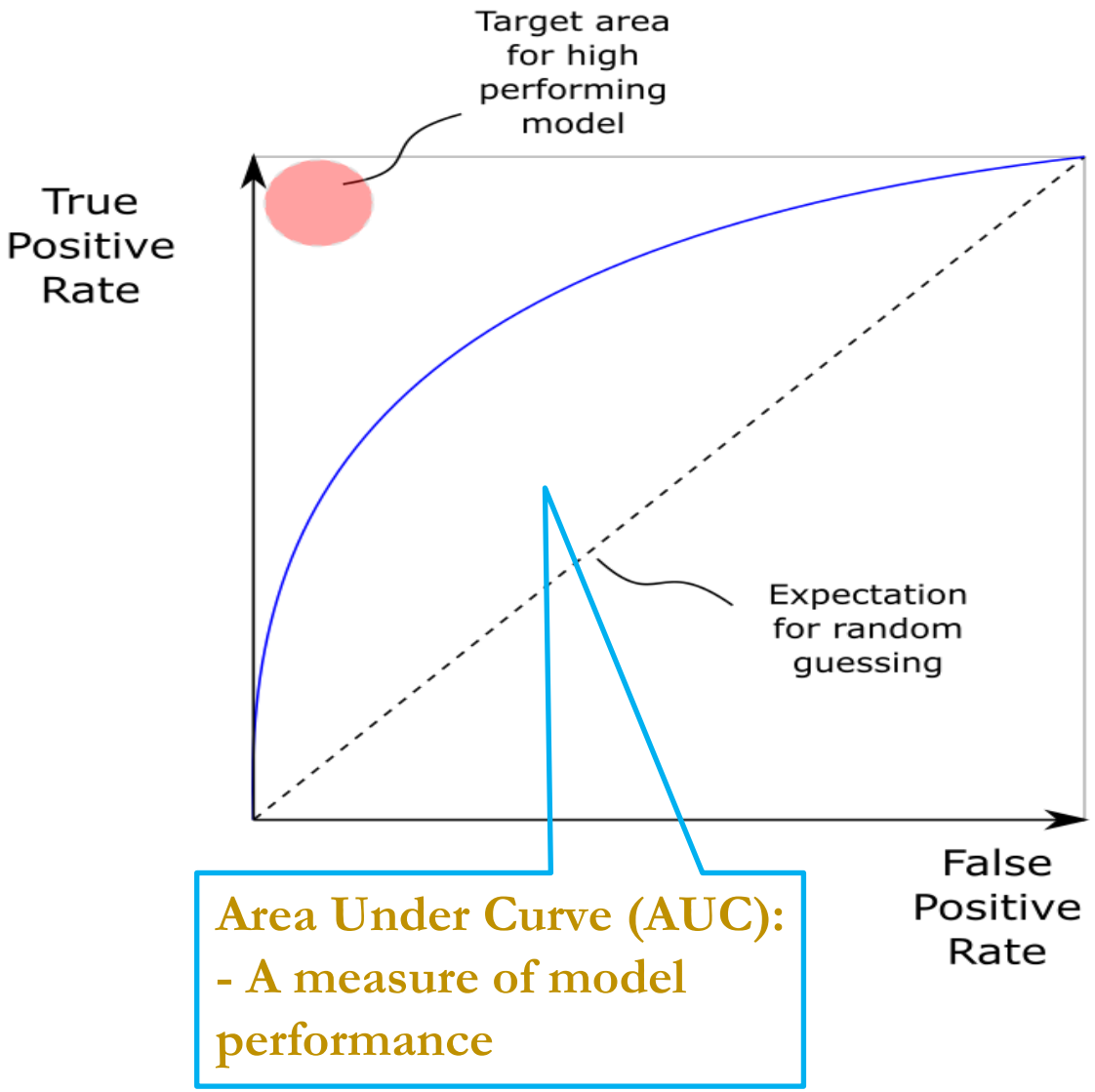


Performance of Binary Classifier

- Sensitivity and Precision in Python (See notebook):

```
> # Import packages
> from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

> print("Sensitivity (True positive rate, or Recall): ", recall_score(y_test, y_pred)) # Get sensitivity/recall score

> print("Precision (Positive predictive value): ", precision_score(y_test, y_pred)) # Get precision score
```

Performance of Binary Classifier



- The **upper left-hand triangle** corresponds to classifiers that are **better** than "random guessing".
- The **lower right-hand triangle** corresponds to classifiers that are **worse** than "random guessing"
- The closer the curve follows the **left-hand border and the top border** of the ROC space (i.e., closer to the upper left-hand circle), the **more accurate** the test.
- The closer the curve comes to the **45-degree diagonal** of the ROC space, the **less accurate** the test.

ROC and AUC

Set Threshold:
If Score >= Threshold: Predict 1;
If Score < Threshold: Predict 0.

TPR=1/6, FPR=0/4

TPR=2/6, FPR=1/4

TPR=6/6, FPR=3/4

ID	True Output	Score: P(y=1|Data, β)	Predicted Output (Threshold 90%)	Predicted Output (Threshold 70%)	…	Predicted Output (Threshold 10%)
1	1	0.9	1	1	…	1
2	1	0.8	0	1	…	1
3	0	0.7	0	1	…	1
4	1	0.6	0	0	…	1
5	1	0.55	0	0	…	1
6	1	0.47	0	0	…	1
7	0	0.39	0	0	…	1
8	0	0.21	0	0	…	1
9	1	0.19	0	0	…	1
10	0	0.03	0	0	…	0

Performance of Binary Classifier

- ROC and AUC in Python (See notebook):

```
> # Import packages
> from sklearn.metrics import roc_curve, auc

> # Get predicted scores Pr(y=1): Used as thresholds for calculating TP Rate and FP Rate
> scores = Lasso_model.predict_proba(X_test)[:, 1]

> # fpr: FP Rate, tpr: TP Rate, thresholds: Pr(y=1)
> fpr, tpr, thresholds = roc_curve(y_test, scores)
> roc_auc = auc(fpr, tpr) # Area under curve
```

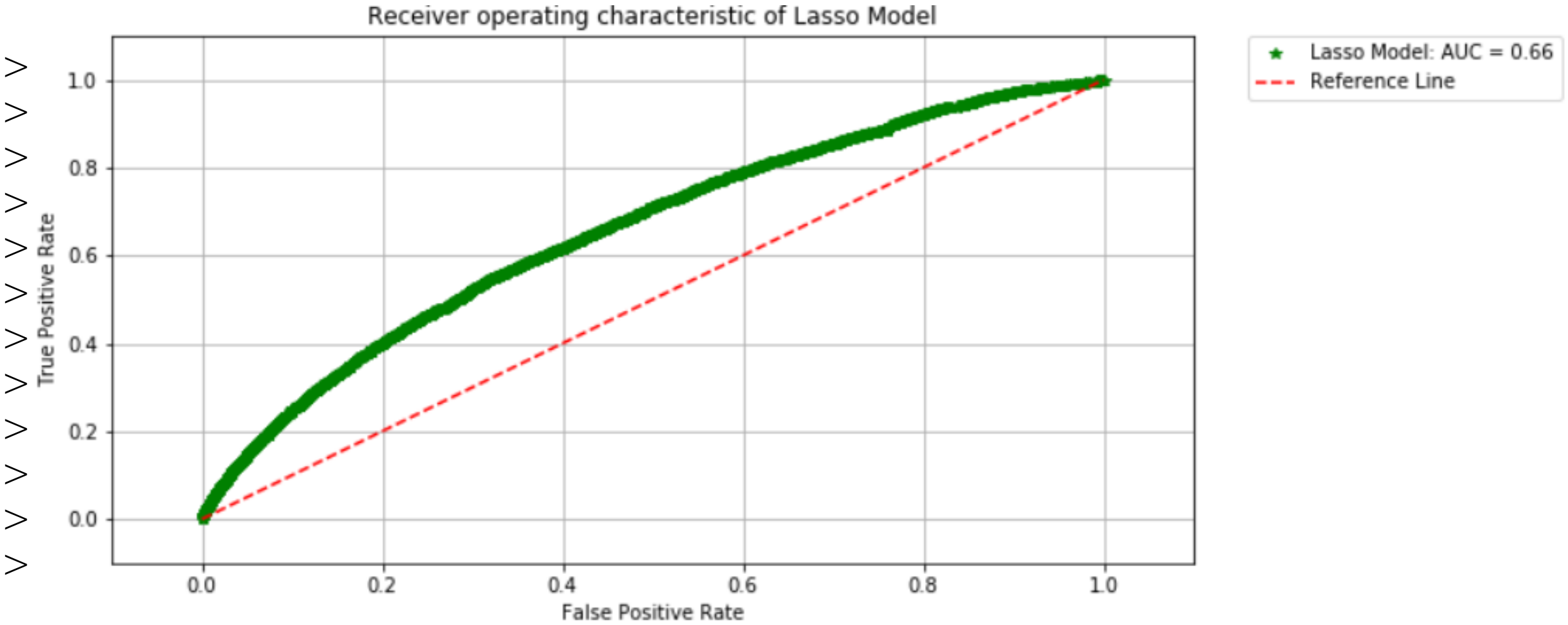
Performance of Binary Classifier

- ROC and AUC in Python (See notebook):

```
> # Visualize ROC and AUC of test data
> plt.figure(figsize=(10,5))
> plt.grid(True)
> plt.plot(fpr, tpr, 'g*', label='Lasso Model: AUC = %0.2f'% roc_auc)
> plt.plot([0,1], [0,1], 'r--', label='Reference Line')
> plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
> plt.xlim([-0.1,1.1])
> plt.ylim([-0.1,1.1])
> plt.title('Receiver operating characteristic of Lasso Model')
> plt.ylabel('True Positive Rate')
> plt.xlabel('False Positive Rate')
> plt.show()
```

Performance of Binary Classifier

- ROC and AUC in Python (See notebook):

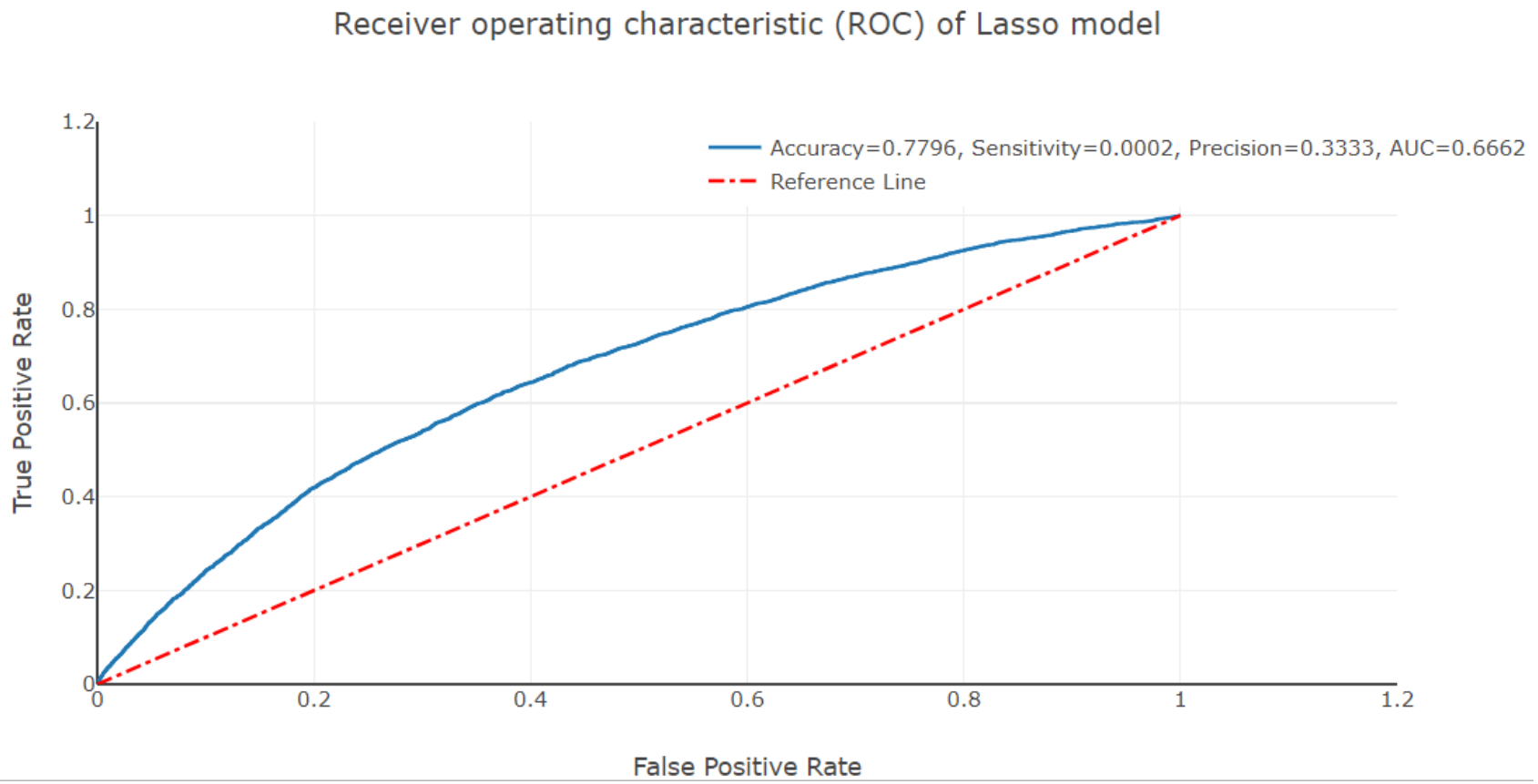


Hyperparameter Tuning and Model Performance

- How does model performance change with hyperparameters (See notebook):

Please drag and slide:

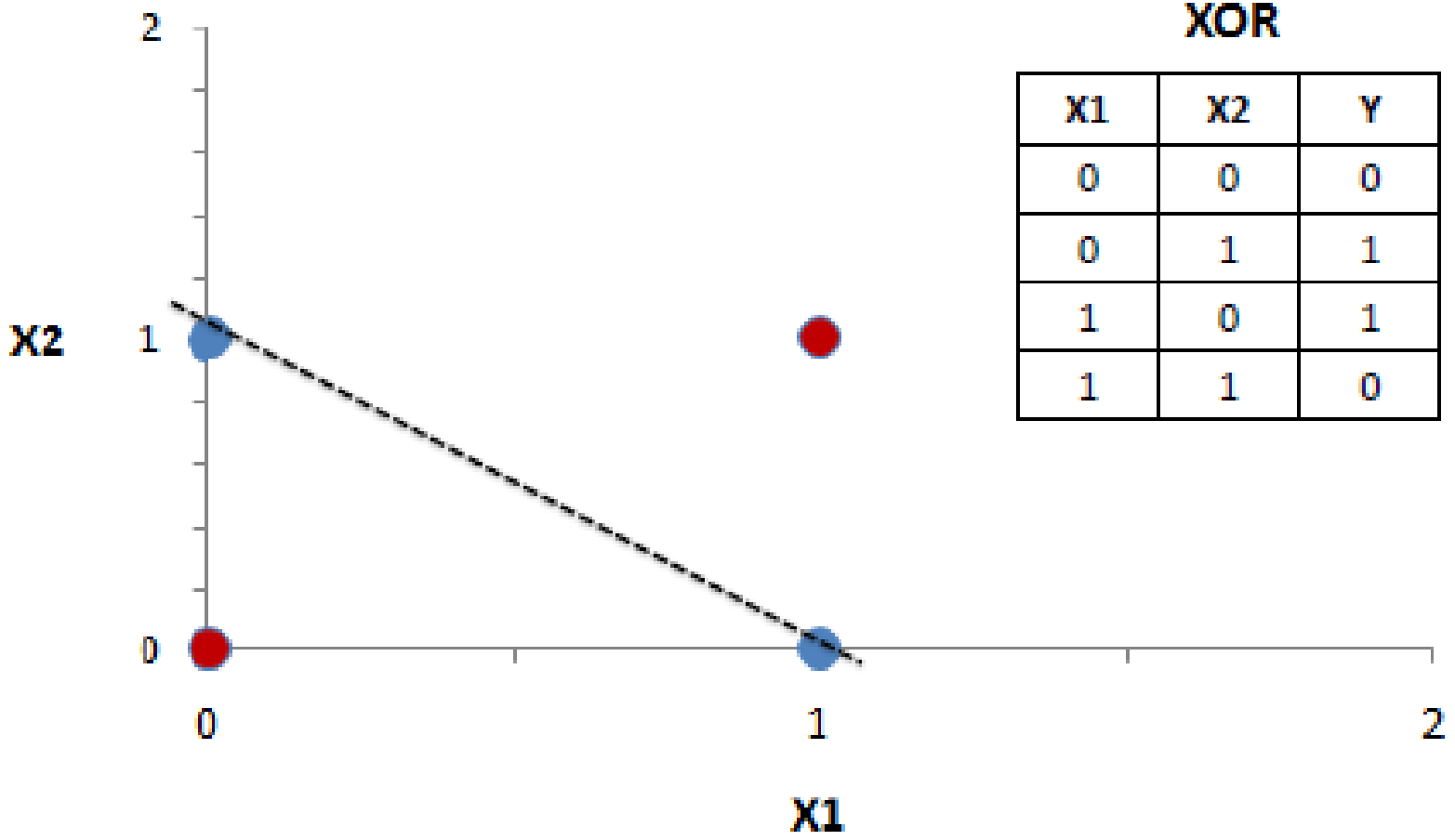
lambdas: 1.00



Tree-Based Models

Making Predictions With Regression

- Problems:
  - Nonlinear Data Pattern
- Exclusive-OR:



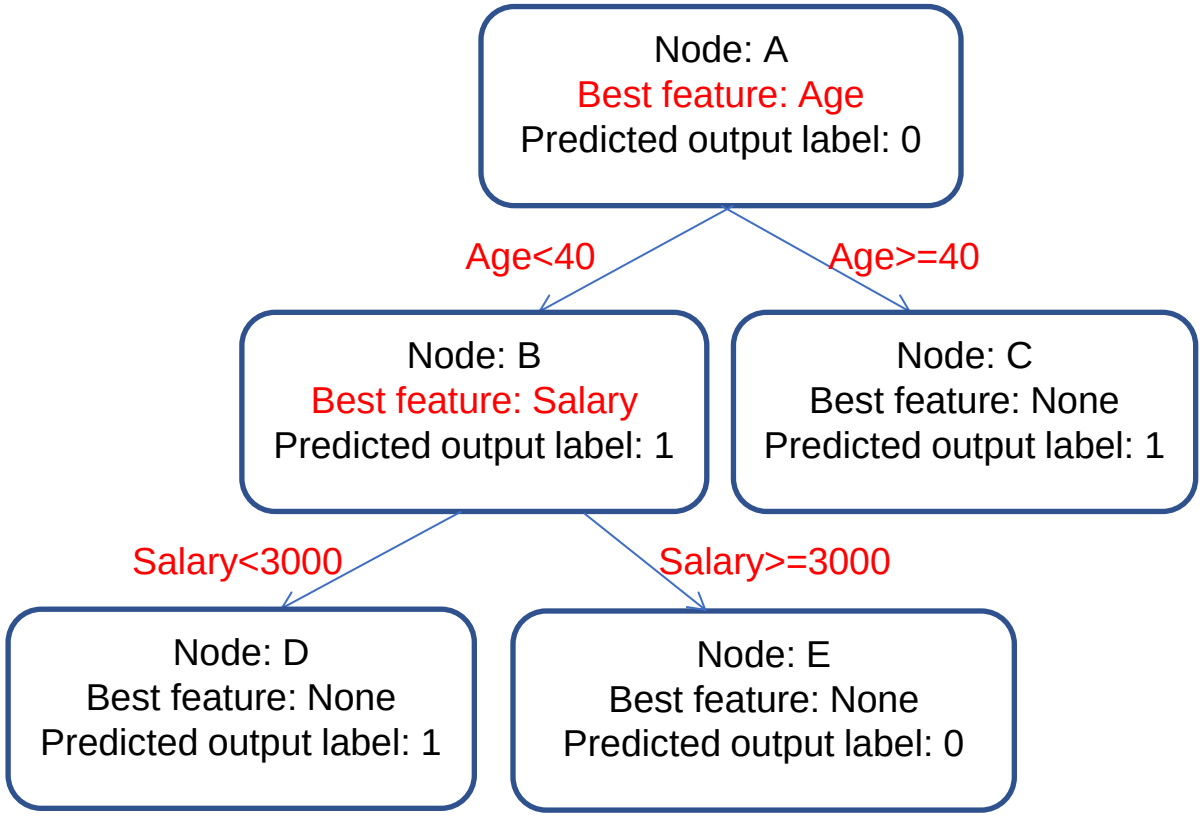
XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

How about Tree Method ?
(1) If x1>0.5 and x2>0.5, then y=0 (Red); (2) If x1>0.5 and x2<=0.5, then y=1 (Blue);
(3) If x1<=0.5 and x2>0.5, then y=1 (Blue); (4) If x1<=0.5 and x2<=0.5, then y=0 (Red);

Decision Tree

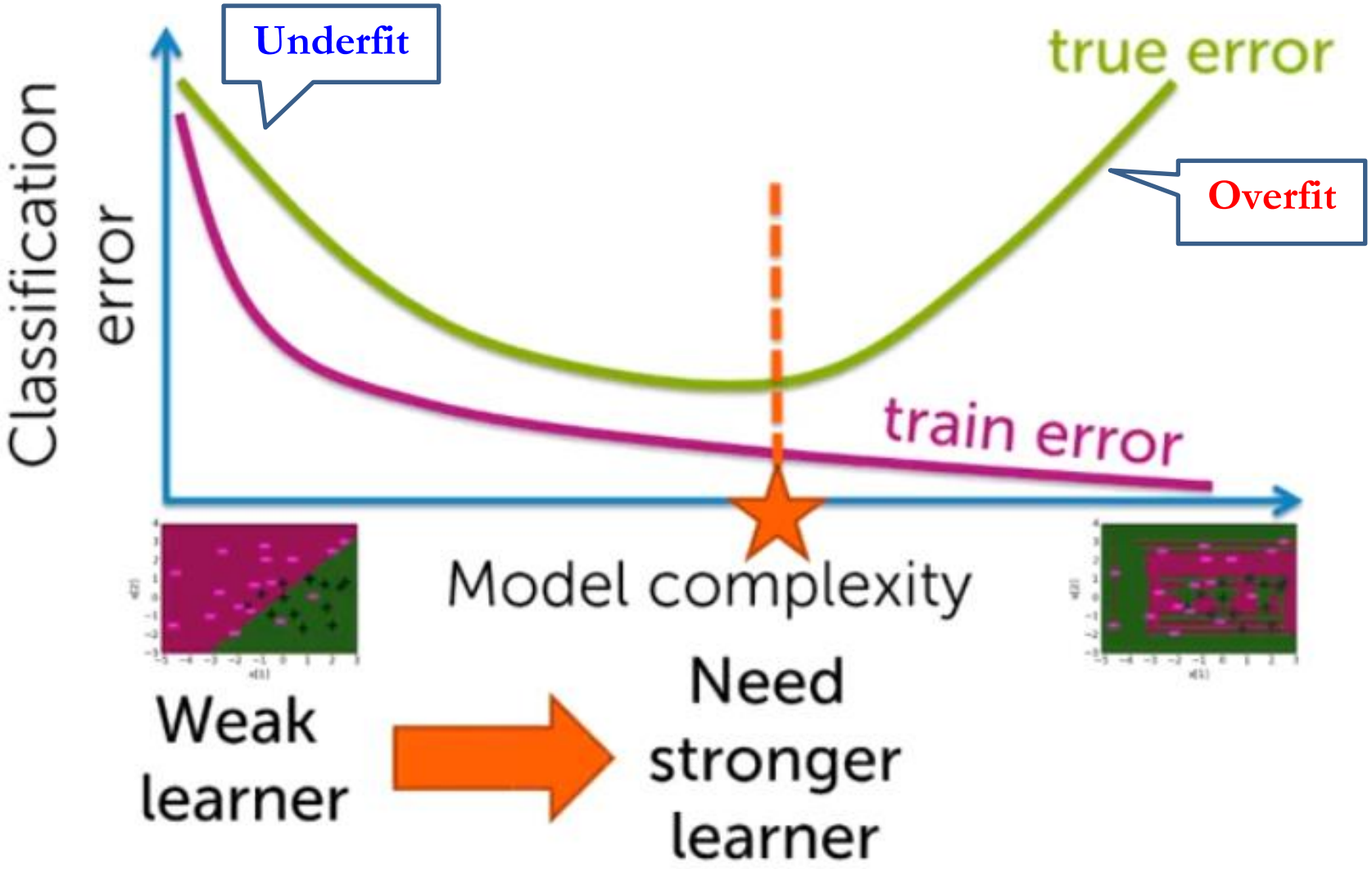
- Example: Build a tree with **training data samples** and get:



ID	Age	Salary	Output: Default
1	20	3000	1
2	25	4000	0
3	50	1500	0
4	45	2500	1
5	33	4500	1
…	…	…	…

Underfit And Overfit

- Finding a single model that performs well on prediction is not so easy.
- Data Scientists focus more on overfit issue.



How To Make Better Predictions ?

- What we've learned from previous weeks:
  - Regularization: Penalized regression
  - Cross-Validation
  - Feature Selection (Week4 Tutorial)
  - Pruning in decision tree (e.g., early-stopping when training a tree)
  - **Ensemble Learning methods (This Tutorial)**
  - Dropout in deep learning and neural network
  - More…

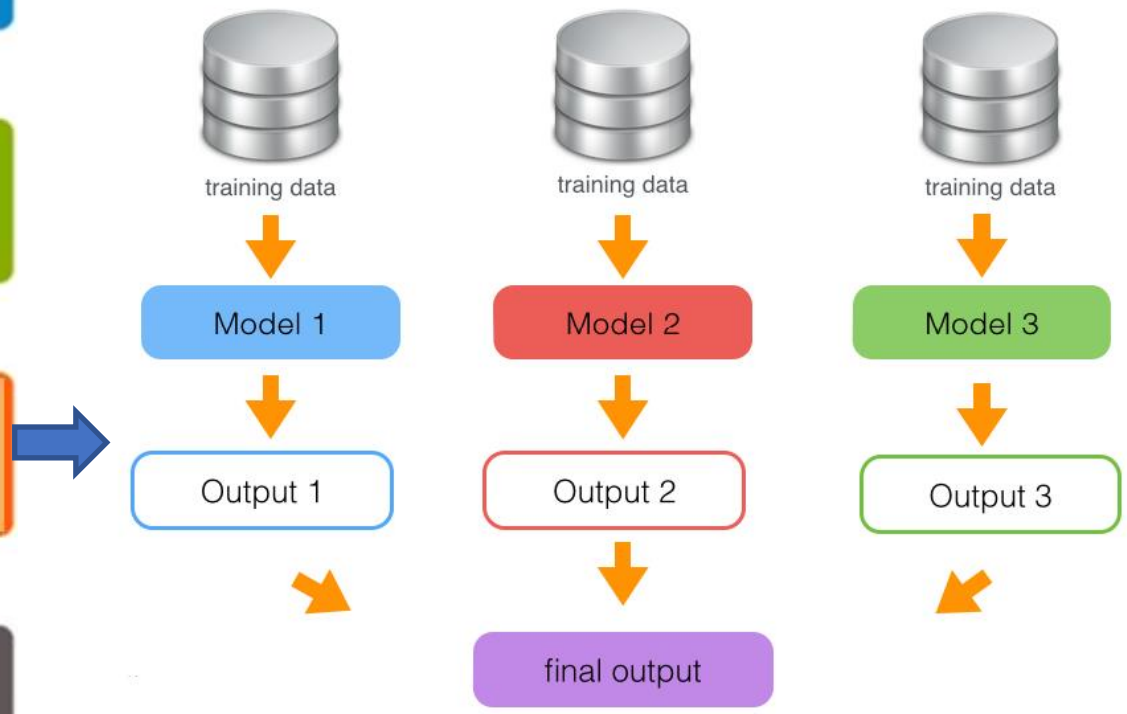
Ensemble learning

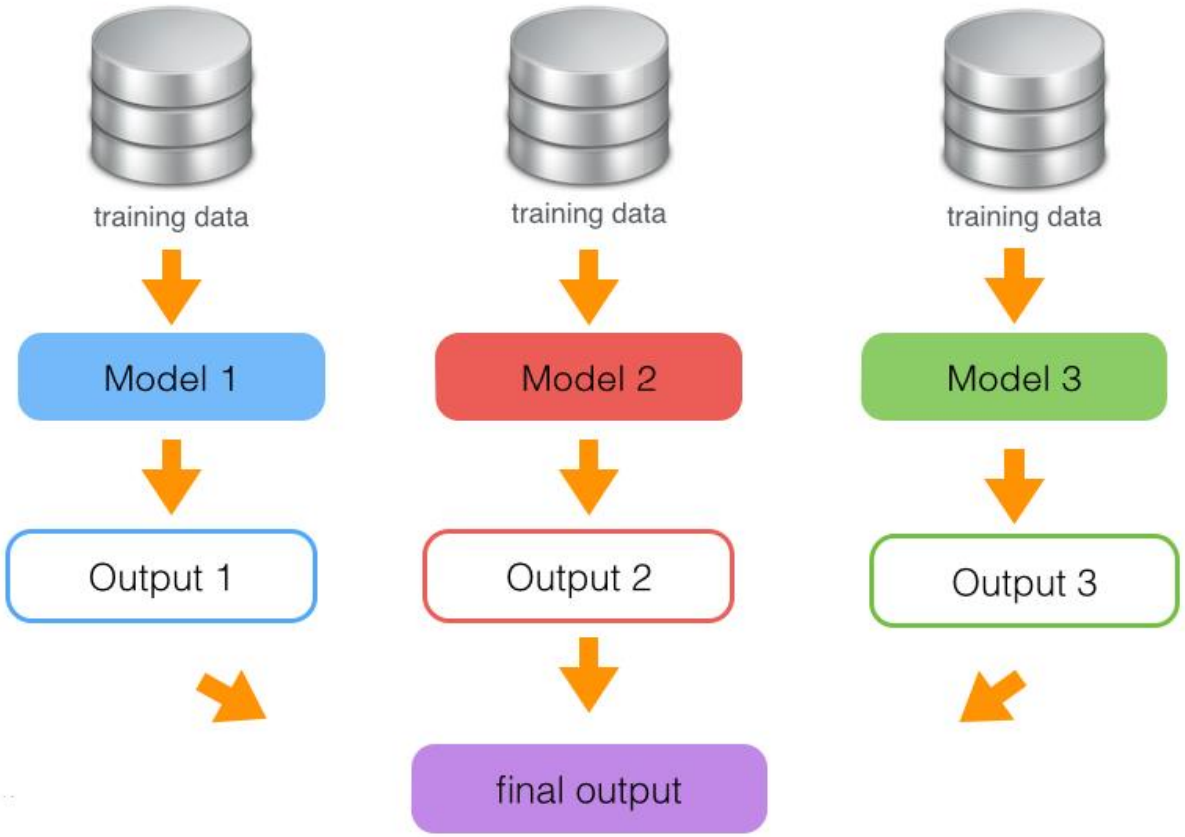
"Can a set of weak learners be combined to create a stronger learner?" *Kearns and Valiant (1988)*

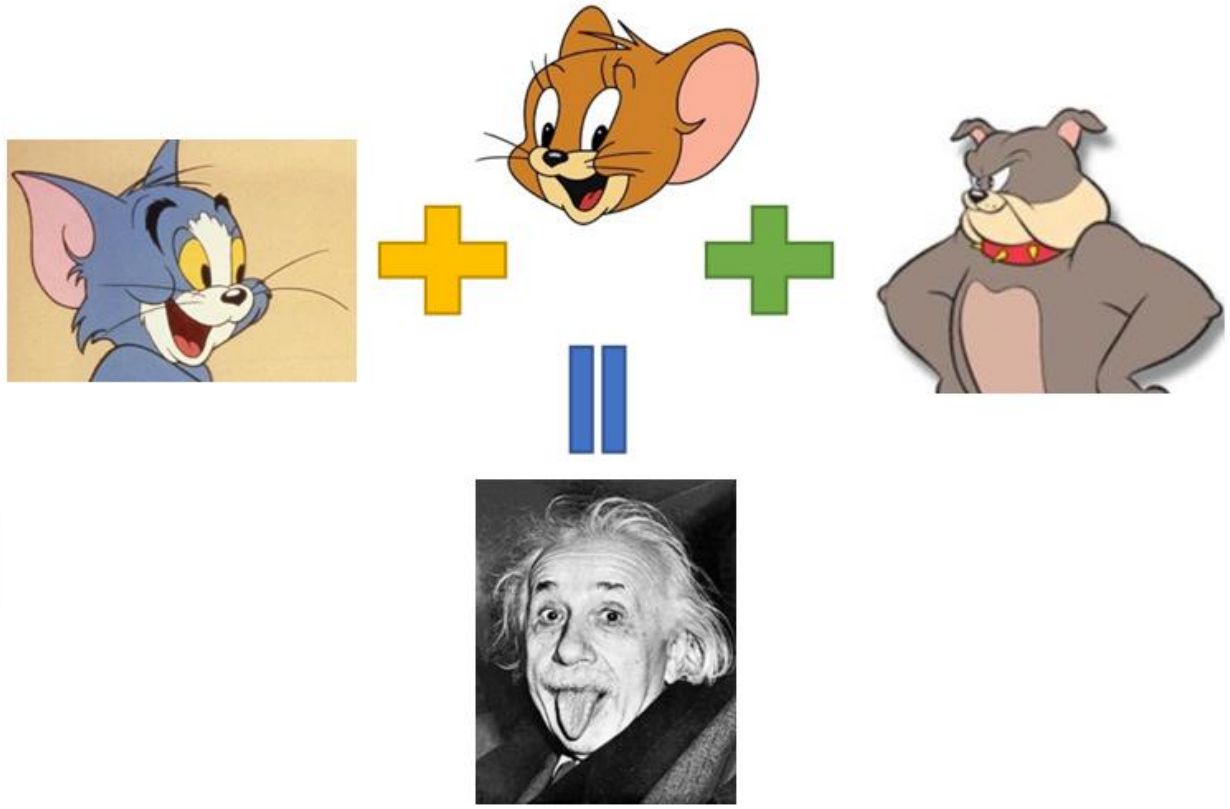
Yes! *Schapire (1990)*

Ensemble Learning Method

Amazing impact: • simple approach • widely used in industry • wins most Kaggle competitions







Learn from not just one learner/model but a set of base learners/models, and **combine** their predictions for the unseen instances using some **aggregation methods** (e.g., taking average, majority voting, logistic regression, etc.)

A set of weak models are combined to create a strong model

History of Ensemble Learning

1980s

- Hansen and Salamon 1989: Basic Concepts
- Schapire 1990: Boosting

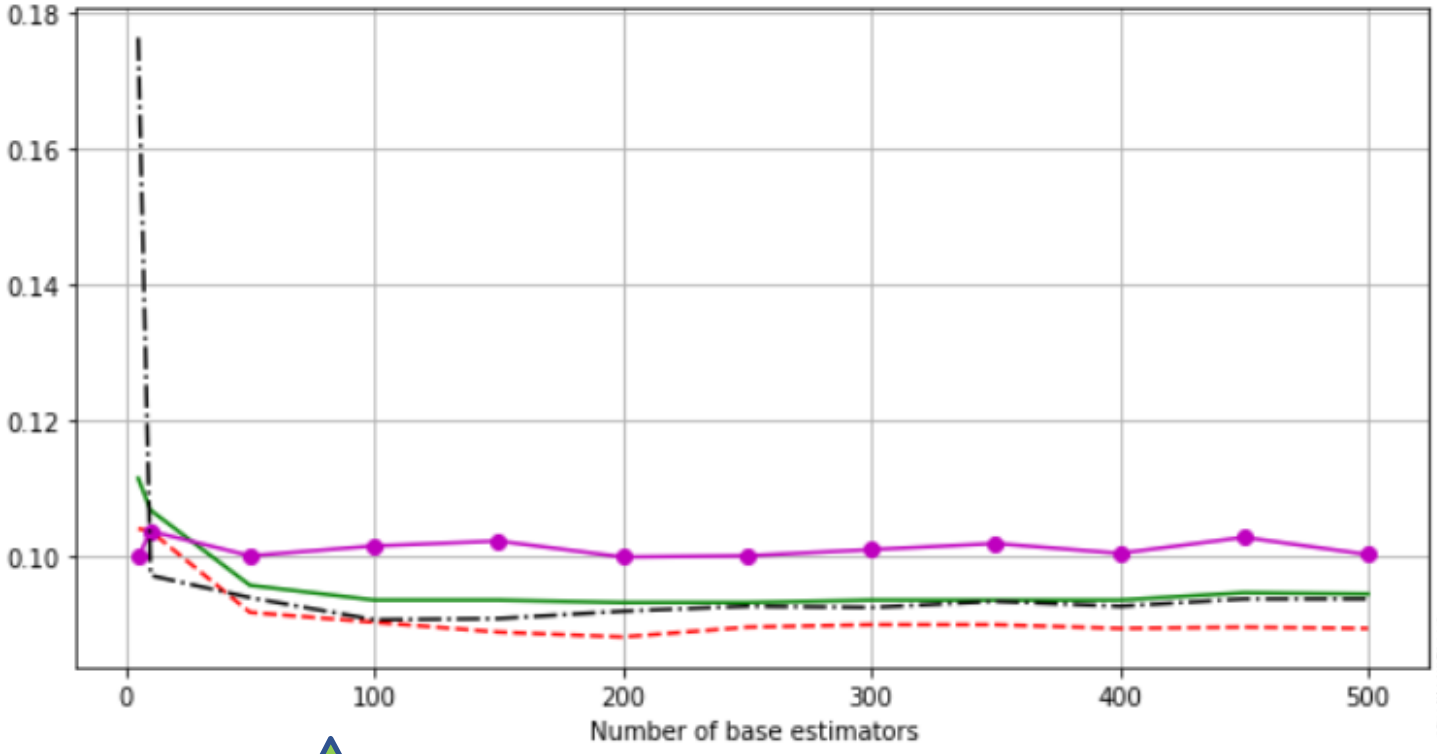
1990s

- Breiman 1994: Bagging
- Tin K.H. 1995: Random Forest
- Schapire and Freund 1997: Adaptive Boosting (Adaboost)

2000s

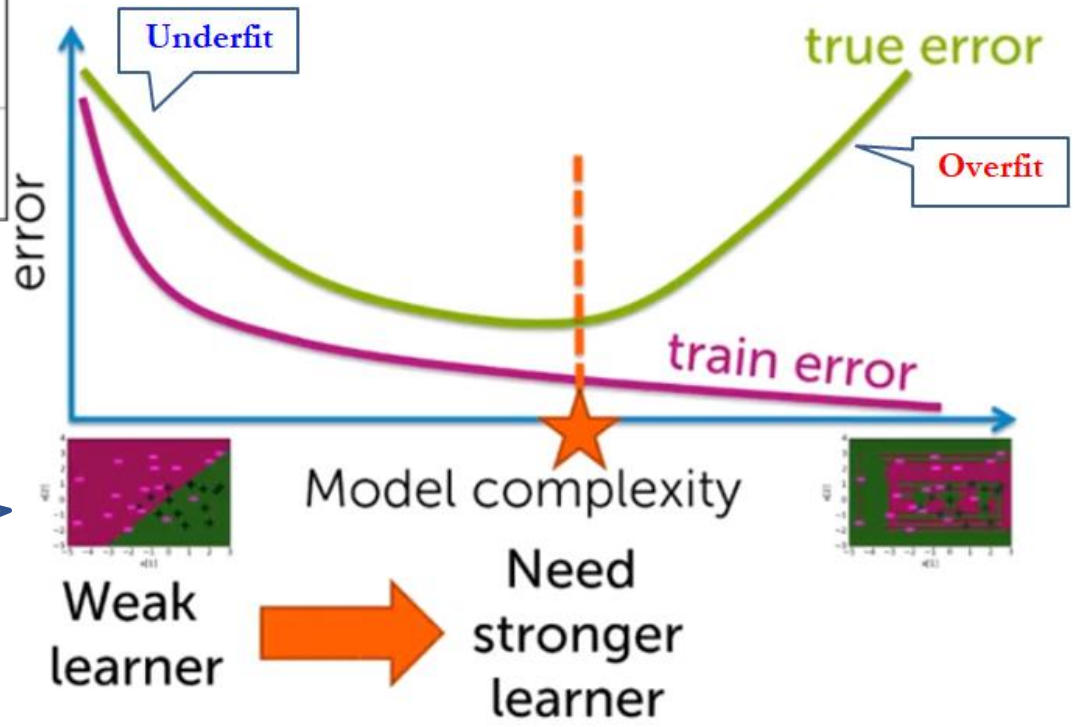
- Friedman 1999: Gradient Boosting Machine
- Chen Tianqi 2016: XGBoost
- Microsoft 2017: LightGBM

Ensemble Method



Ensemble Learning

Single Model



Ensemble Method

- Parallel Learning
  - Bagging (Bootstrapping Aggregation)
  - Random Forest
  - Stacking: Combination of different-type base models

- Sequential Learning (Error-Based or Residual-Based)
  - Adaboost (Adaptive Boosting): Increase weights on misclassified data
  - Gradient Boosting: Fit base models on residuals
  - XGBoost: An extension from Gradient Boosting

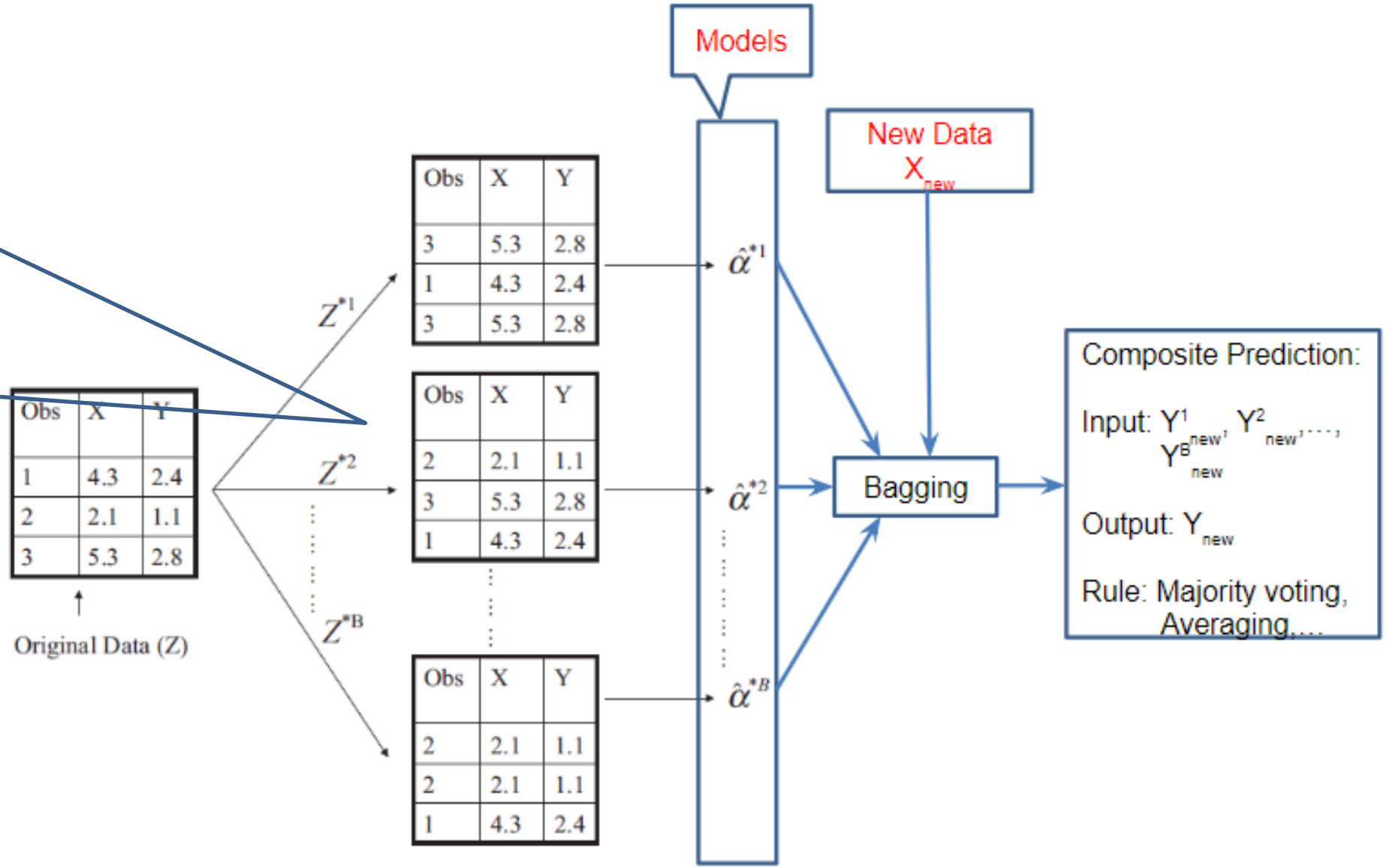
Bagging

Explanation:

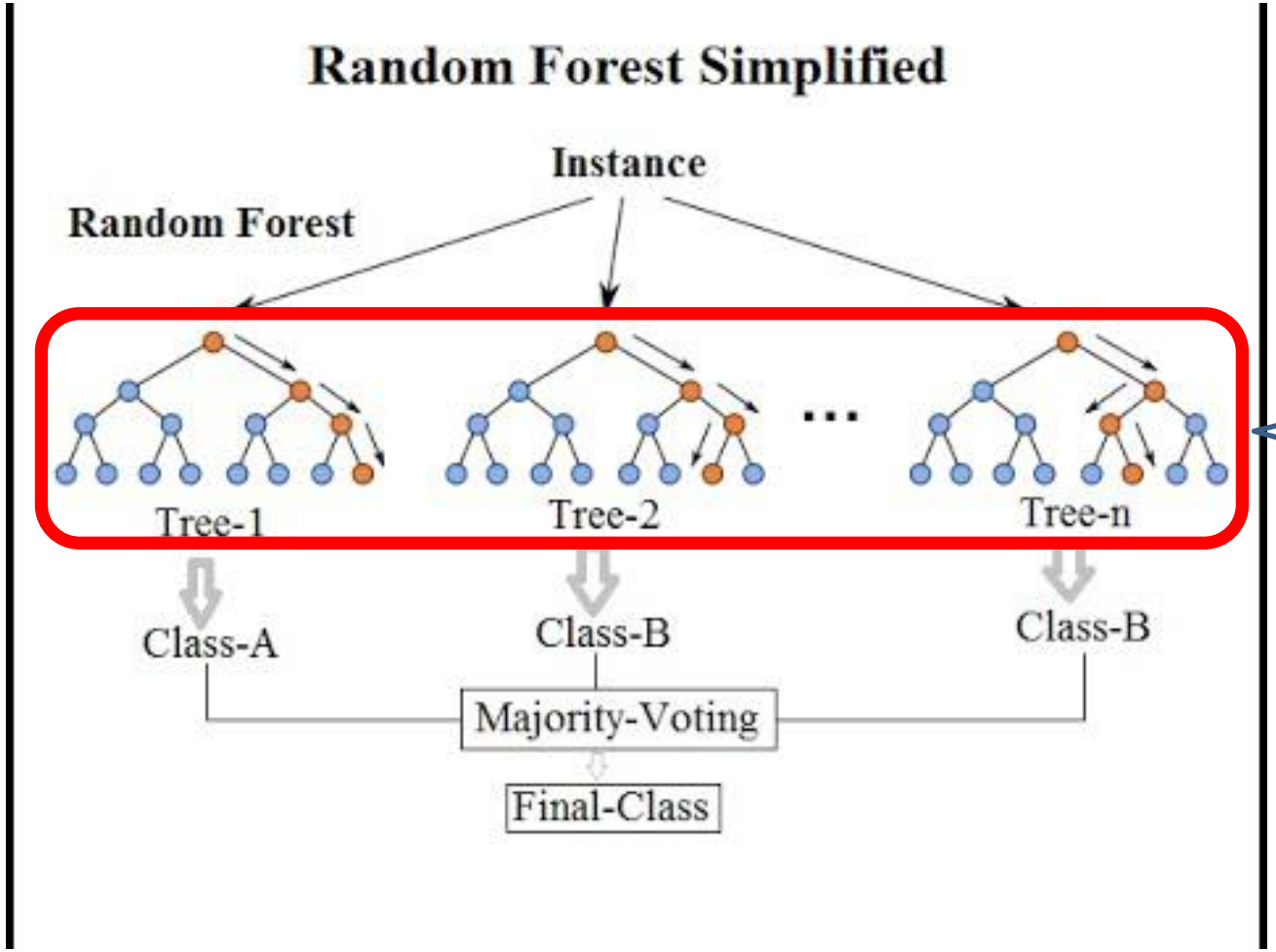
Bootstrap sampling (with replacement) from the original dataset B times

You get B bootstrap samples

Each bootstrap sample size is N (usually same size as original dataset)



Random Forest



Explanation:

Create n base models:
Each time, randomly select a subset of original features (m) to get the base model

Suppose you have p features in total, each time select:

$$m = \sqrt{p}$$

features

Bagging v.s. Random Forest

Bagging

- **Bootstrap Resampling**
- Subsample dimension is the same as original data

They can be combined

Random Forest

- **Randomly Select Features**
- Subsample dimension is different from original data

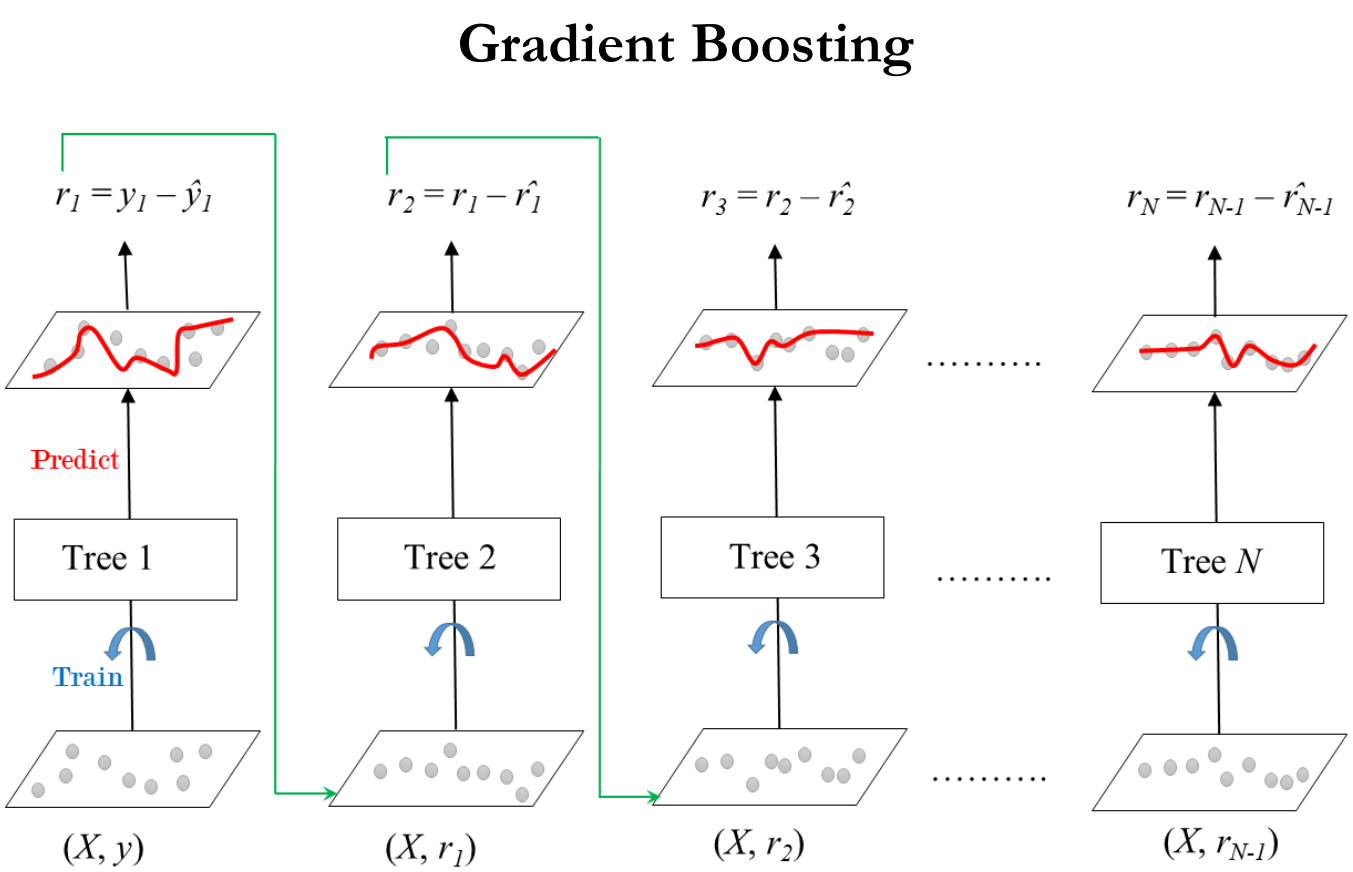
Random Forest With Bootstrapping

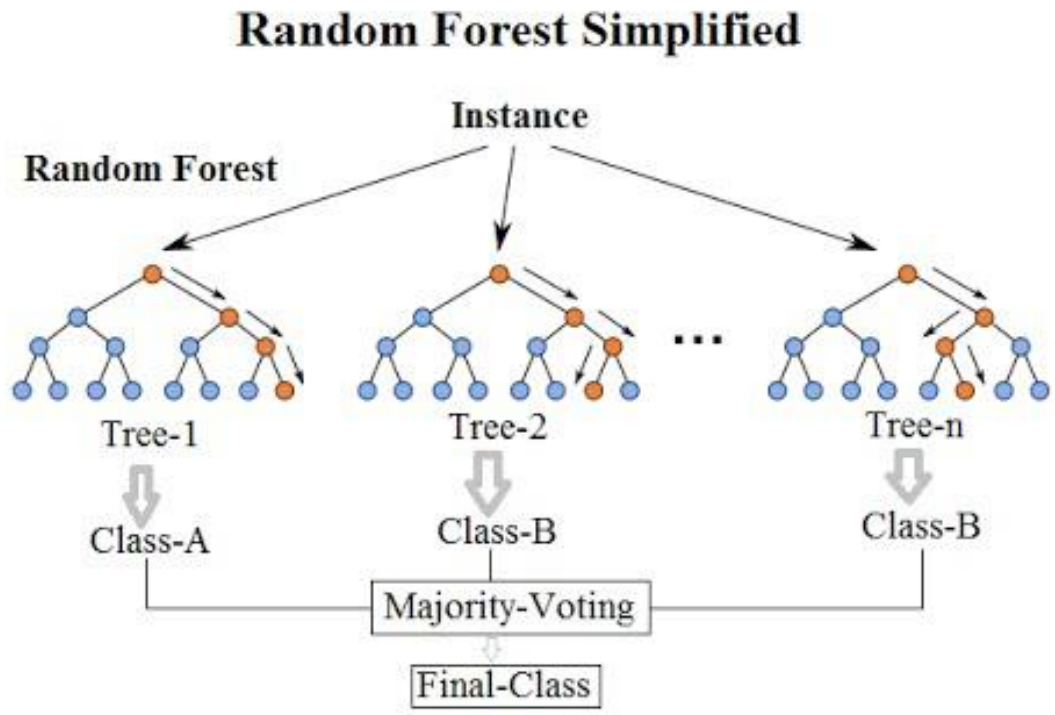
- A Random Forest Tree (See notebook):

```
> # Import packages
> from sklearn.ensemble import RandomForestClassifier

> # Random Forest model: By default, bootstrap=True
> # https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
> N = 100 # Number of base tree models
> RF = RandomForestClassifier(n_estimators=N, random_state=12345)
> RF_model = RF.fit(X_train, y_train)
```

Boosting

Difference?





Gradient Boost

- A Gradient Boost Tree (See notebook):

```
> # Import packages
> from sklearn.ensemble import GradientBoostingClassifier

> # Gradient Boost model
> # https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
> N = 100 # Number of base tree models
> GDB = GradientBoostingClassifier(n_estimators=N, random_state=12345)
> GDB_model = GDB.fit(X_train, y_train)
```

Comparison

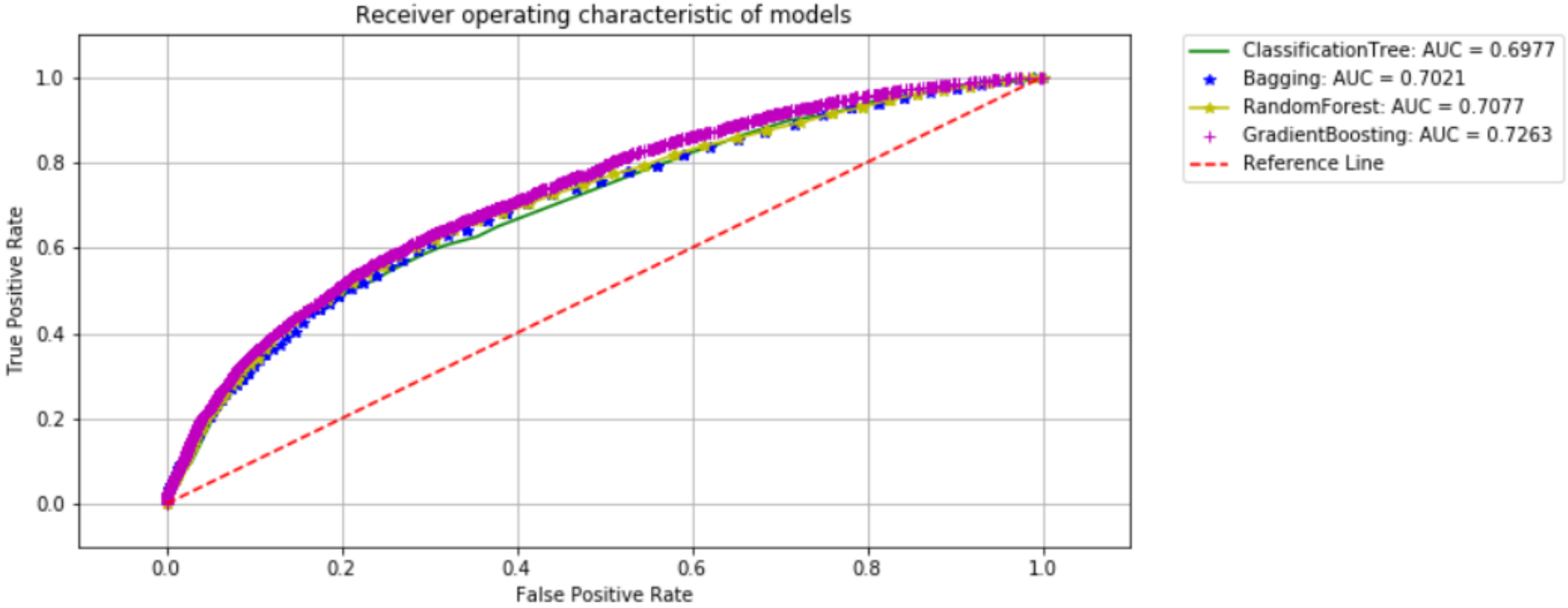
- A single Decision Tree v.s. Ensemble Methods (See notebook):

Model	Accuracy	Sensitivity	Precision	AUC
Single Tree	0.7807	0.0978	0.4861	0.6977
Random Forest	0.7907	0.1905	0.5590	0.7077
Gradient Boost	0.7926	0.1651	0.5870	0.7263

- Ensemble Methods are effective in improving model performance

Comparison

- A single Decision Tree v.s. Ensemble Methods (See notebook):



- Ensemble Methods are effective in improving model performance

Any Questions ?

Thank You !