Data Science

9 – Modeling with Ensemble Learning

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut’s University of Technology Thonburi

BIG DATA
EXPERIENCE
CENTER

Lecture 9 - topics

- Bagging
- Boosting

Ensemble Models

https://colab.research.google.com/drive/1gNx94lEp7PCV5_ZbqTijGr1I0odmndLE

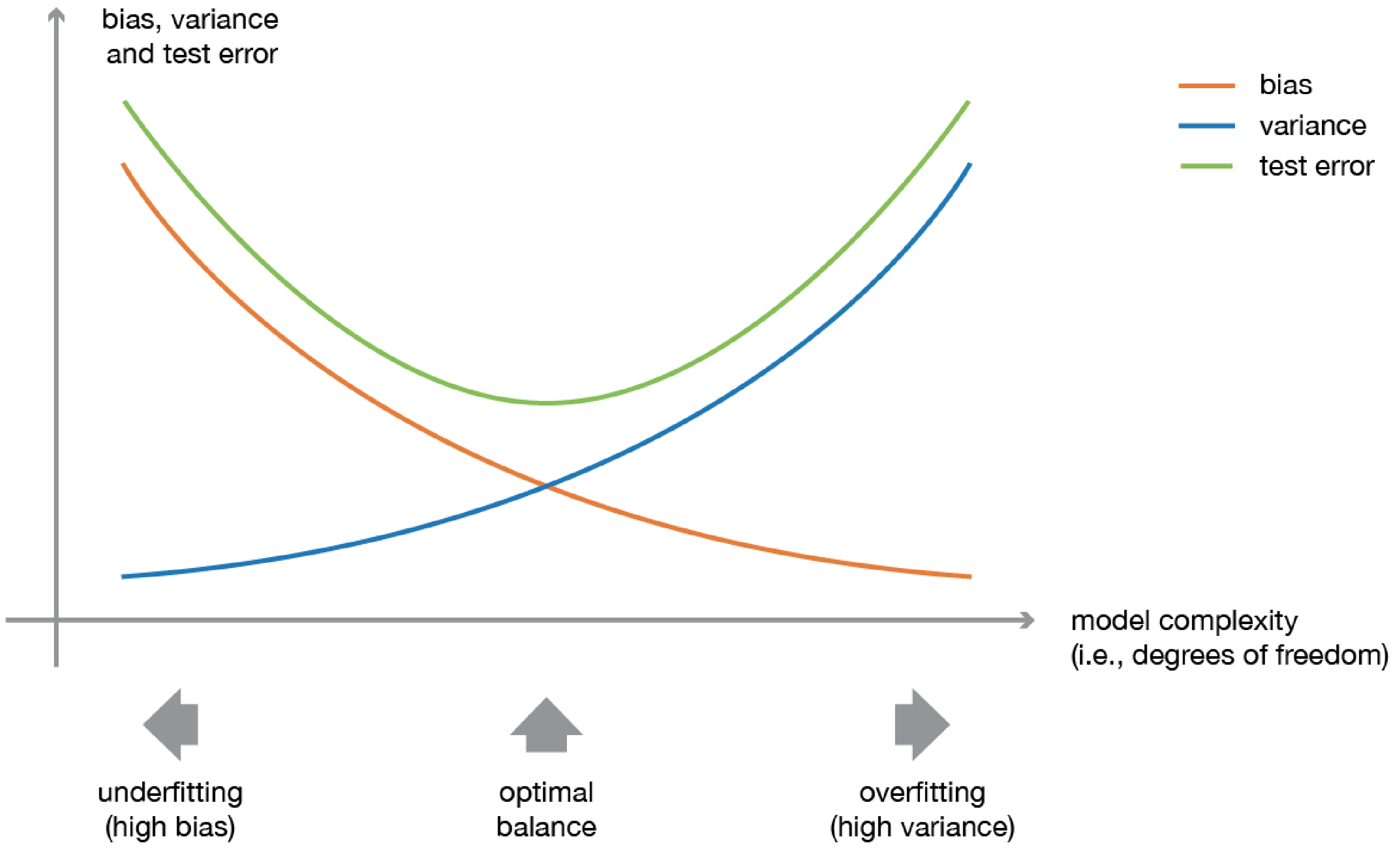
Classification and Regression Trees Disadvantages

- *Accuracy* - current methods, such as support vector machines and ensemble classifiers often have 30% lower error rates than CART.
- *Instability* – if we change the data a little, the tree picture can change a lot. So the interpretation is not as straightforward as it appears.

- We can do better!

Random Forests

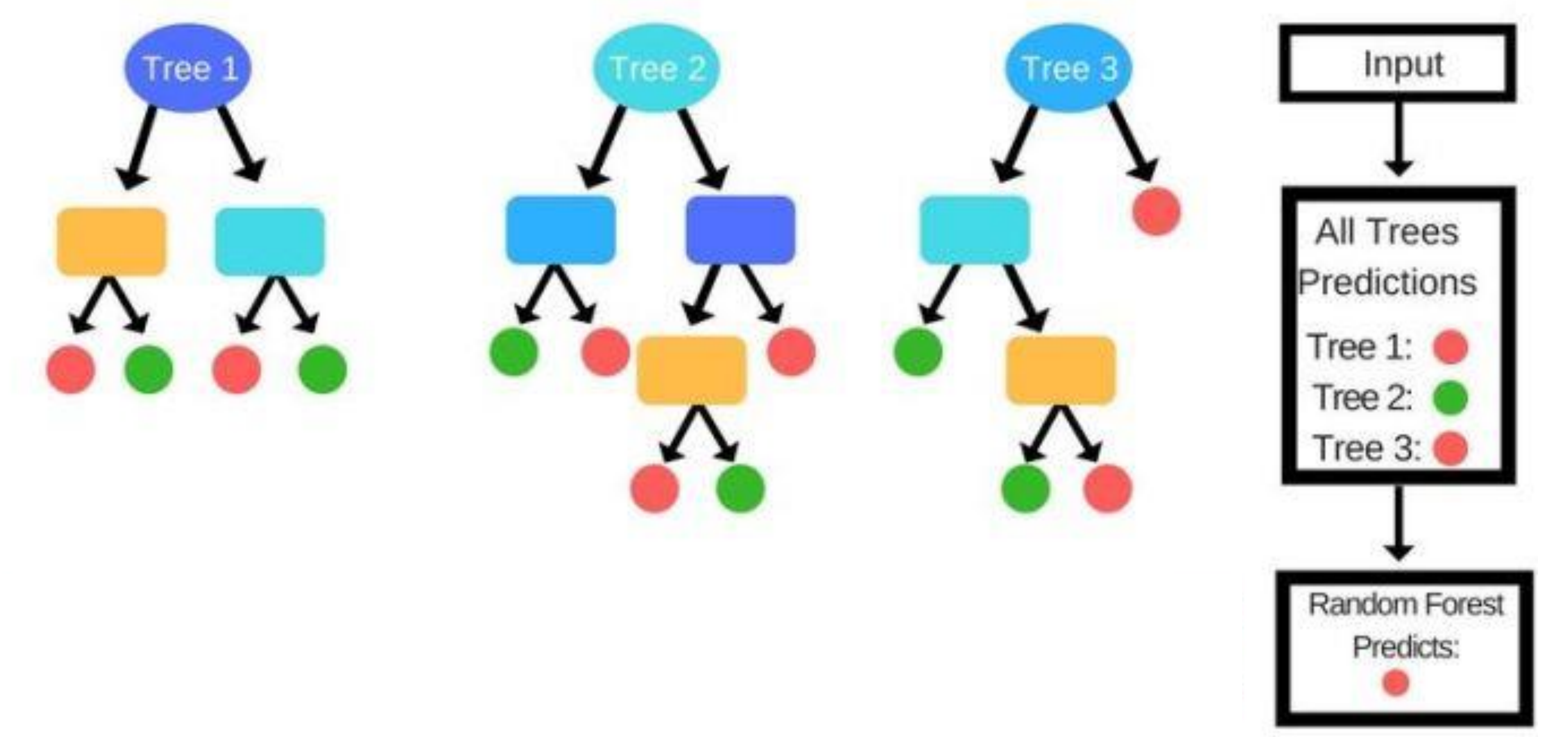
Bias variance tradeoff



Random Forest

for regression and classification

Random Forest



Random forest

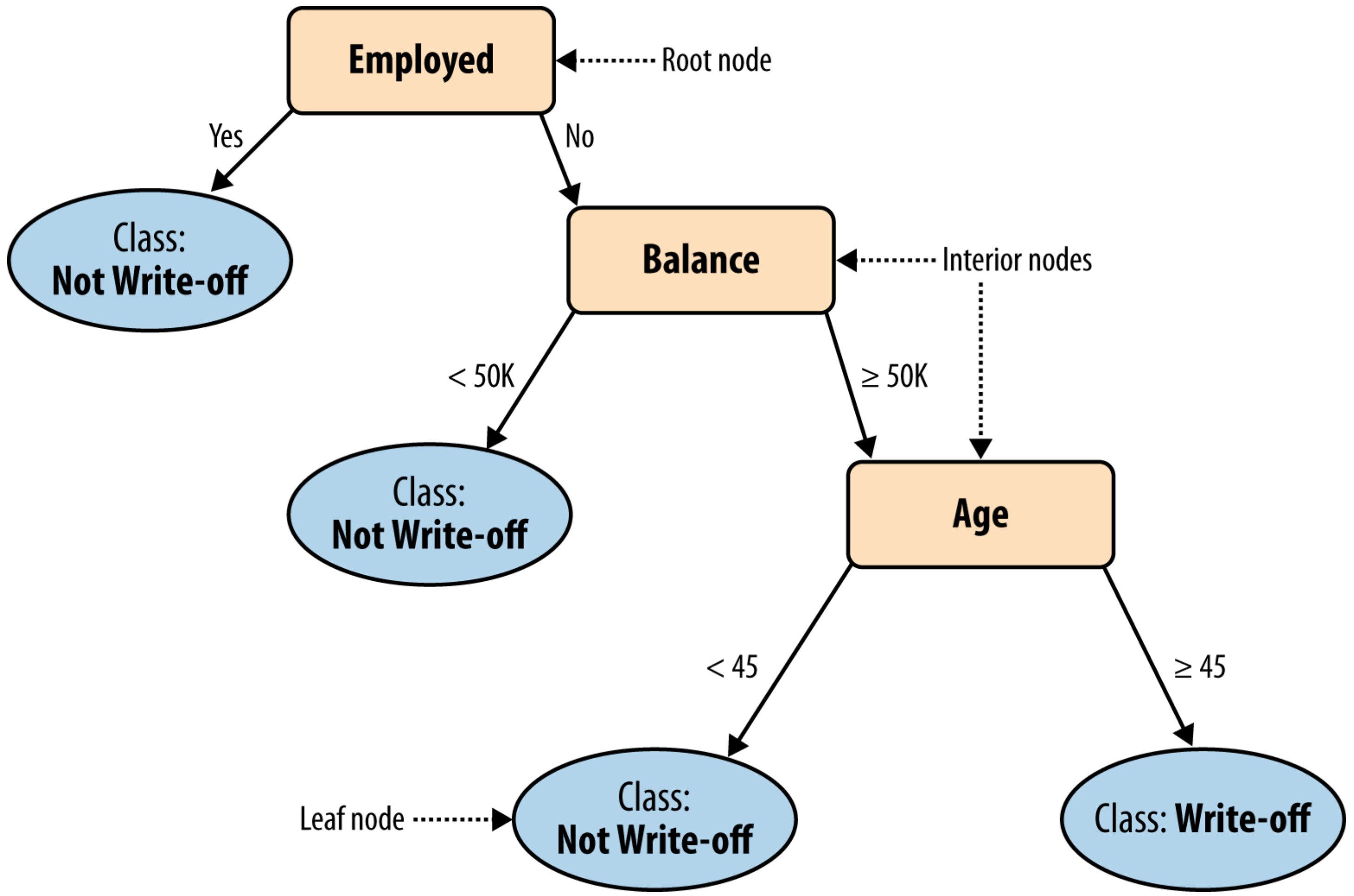
- An ensemble of decision tree trained by bootstrap sampling and random feature selection
- It is based on decision trees: classifiers constructed greedily using the conditional entropy
- The extension hinges on two ideas:
  - building an ensemble of trees by training on subsets of data
  - considering a reduced number of possible variables at each node

Classification and Regression Trees Pioneers

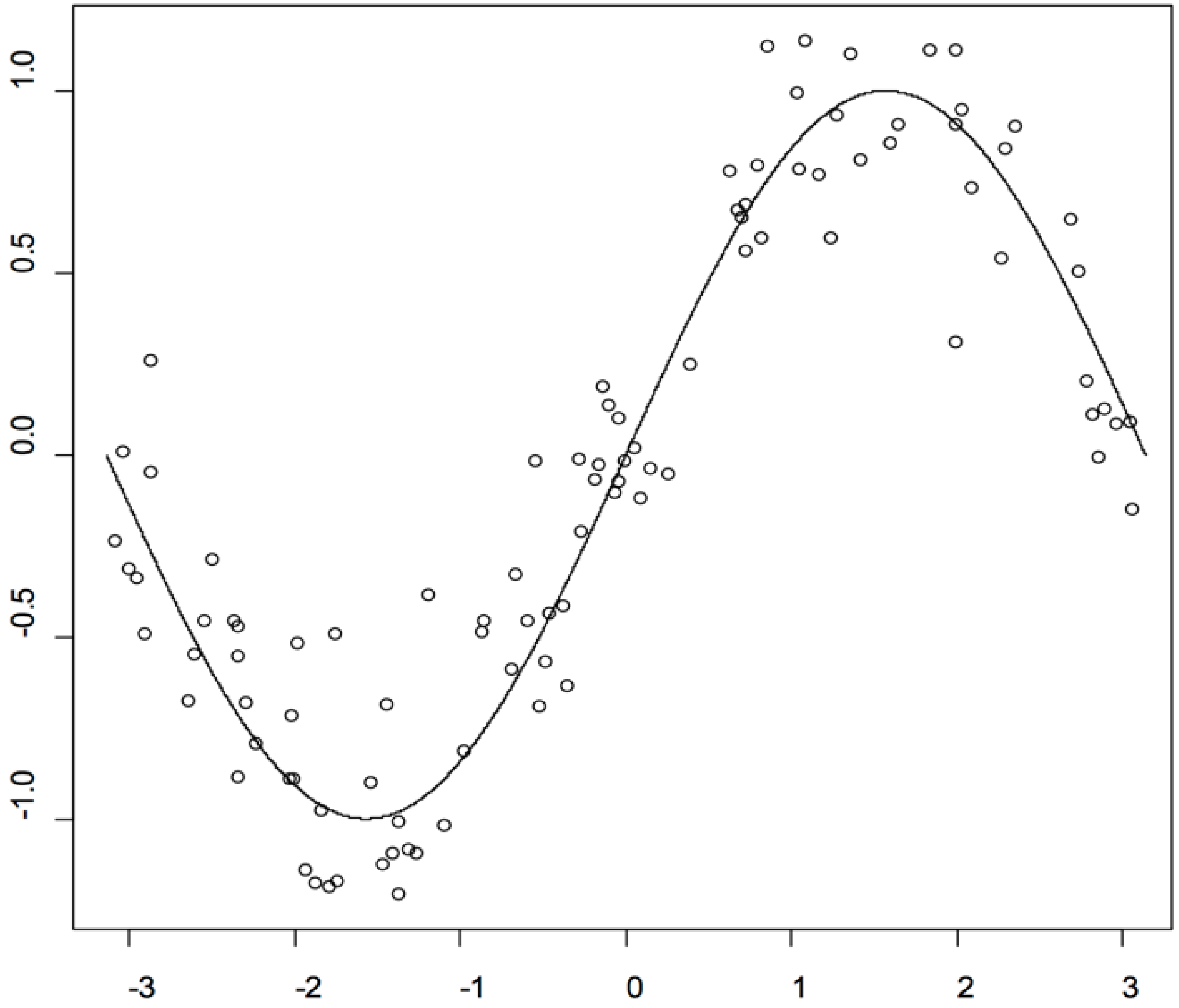
Pioneers:

- Morgan and Sonquist (1963).
- **Breiman, Friedman, Olshen, Stone (1984).** ***CART***
- Quinlan (1993). *C4.5*

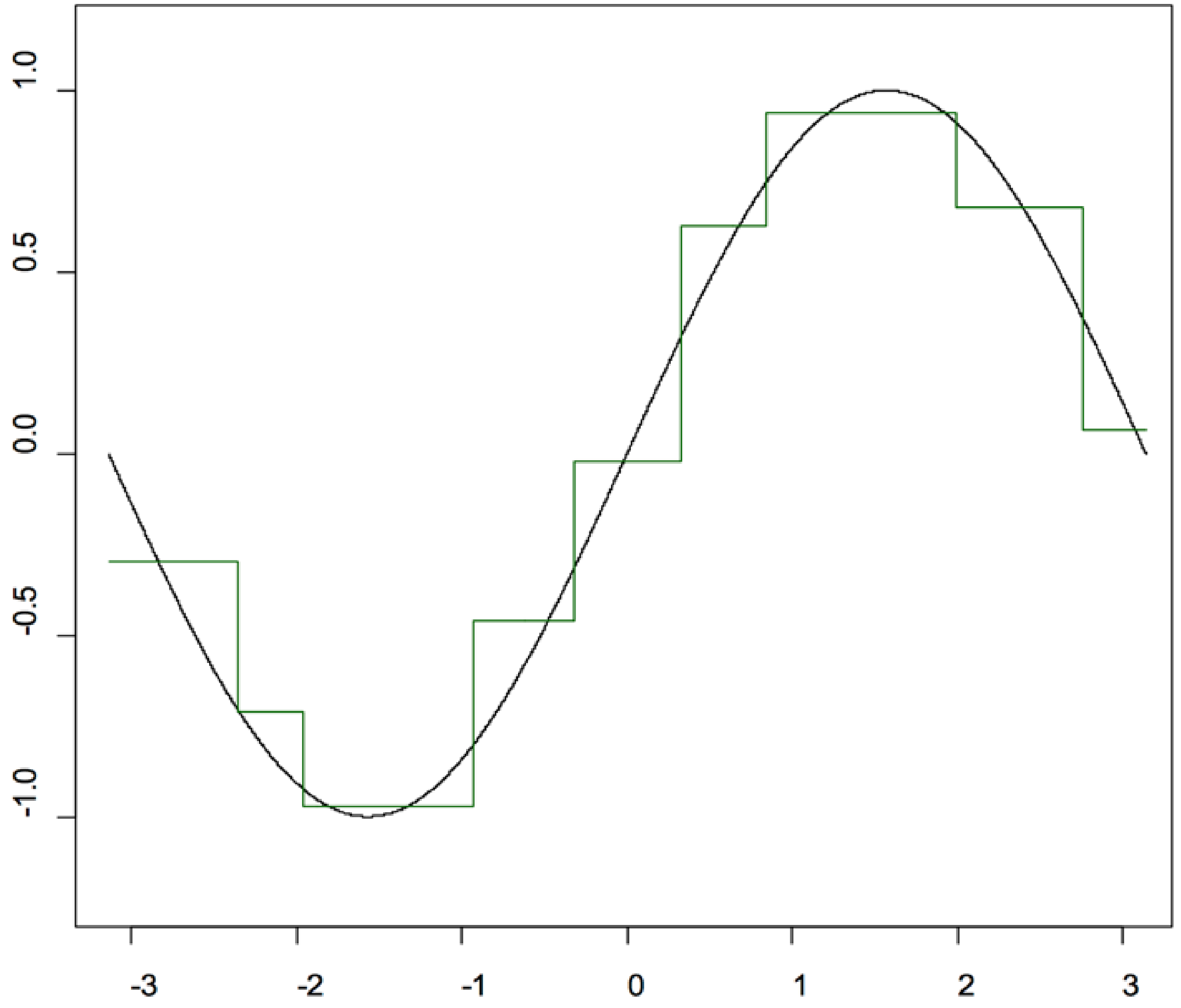
A Classification Tree



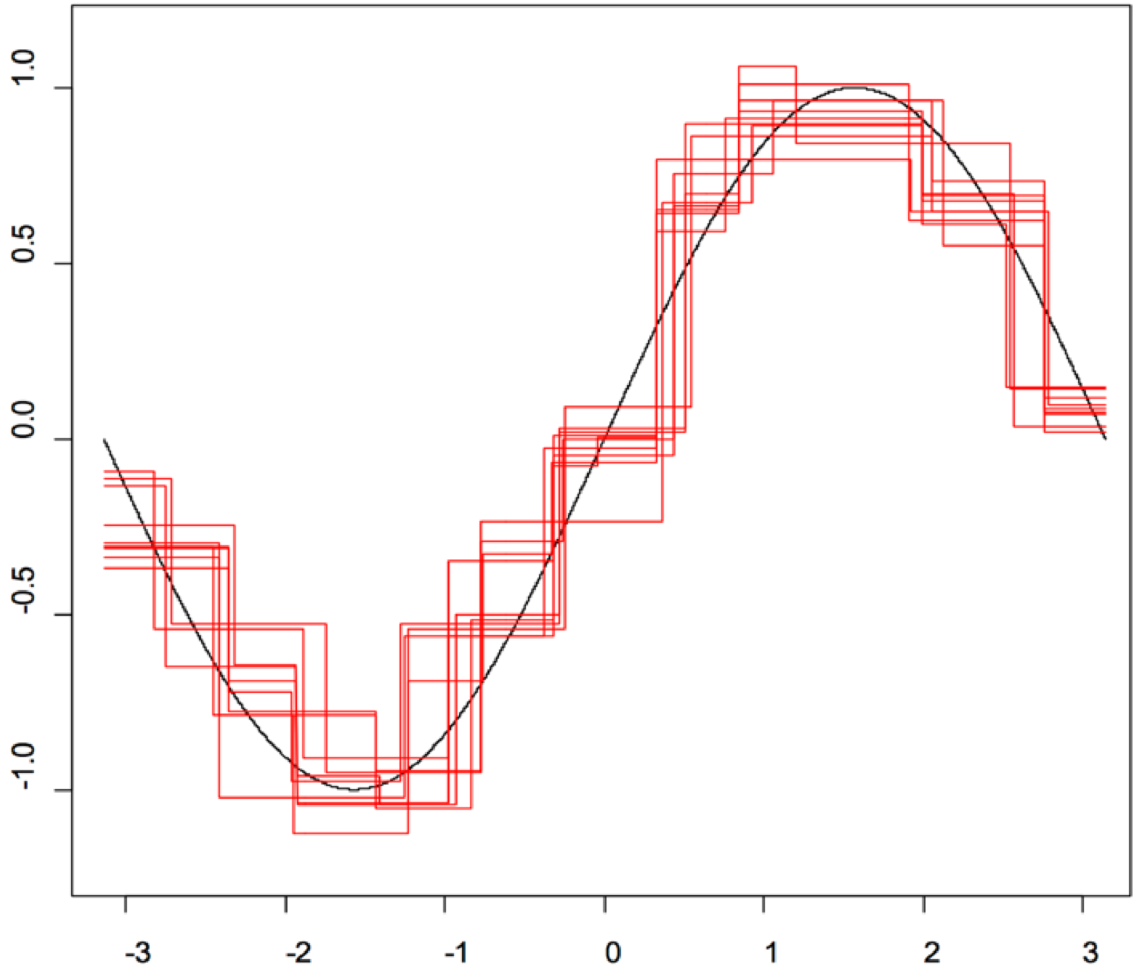
Data and Underlying Function



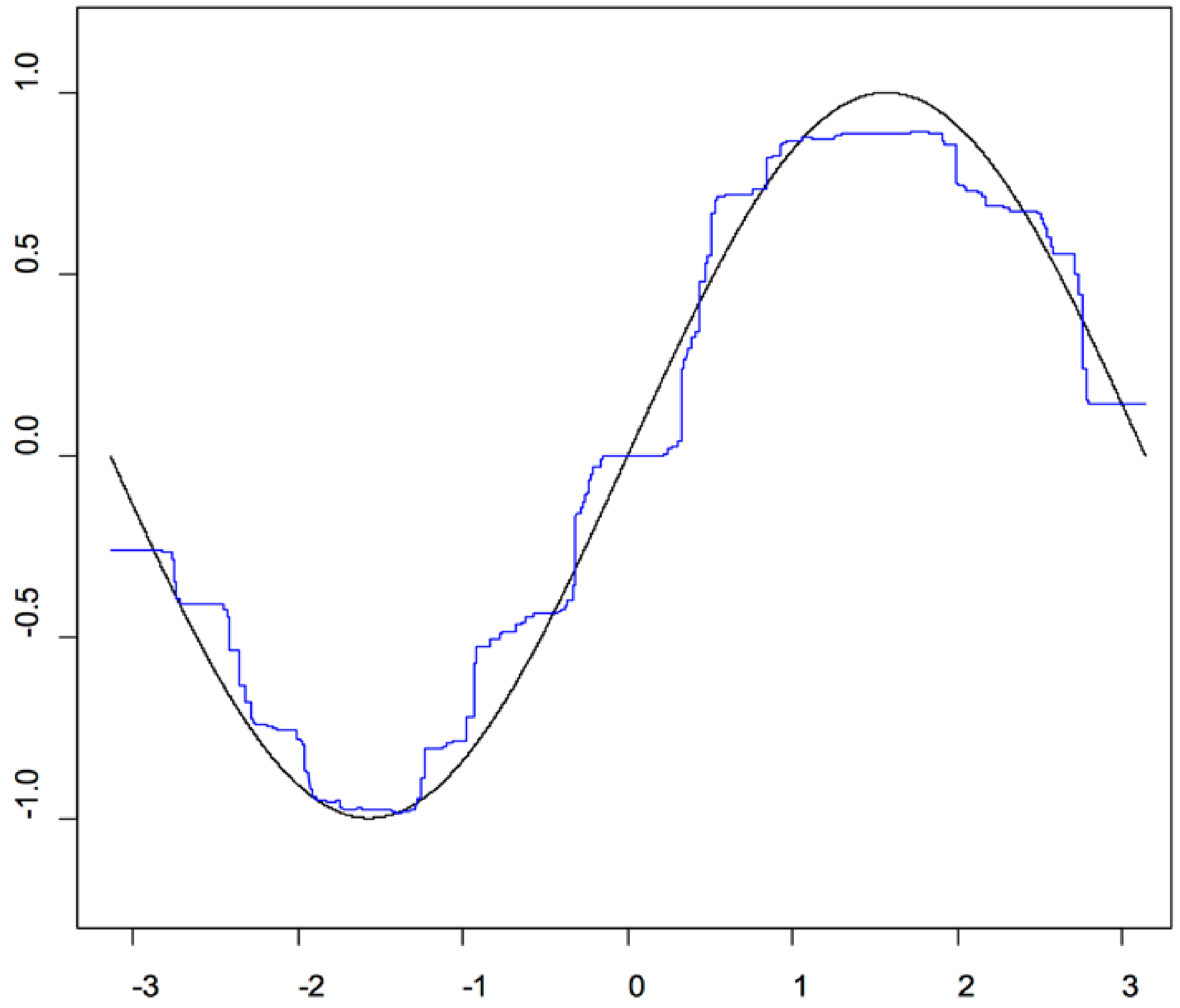
Single Regression Trees



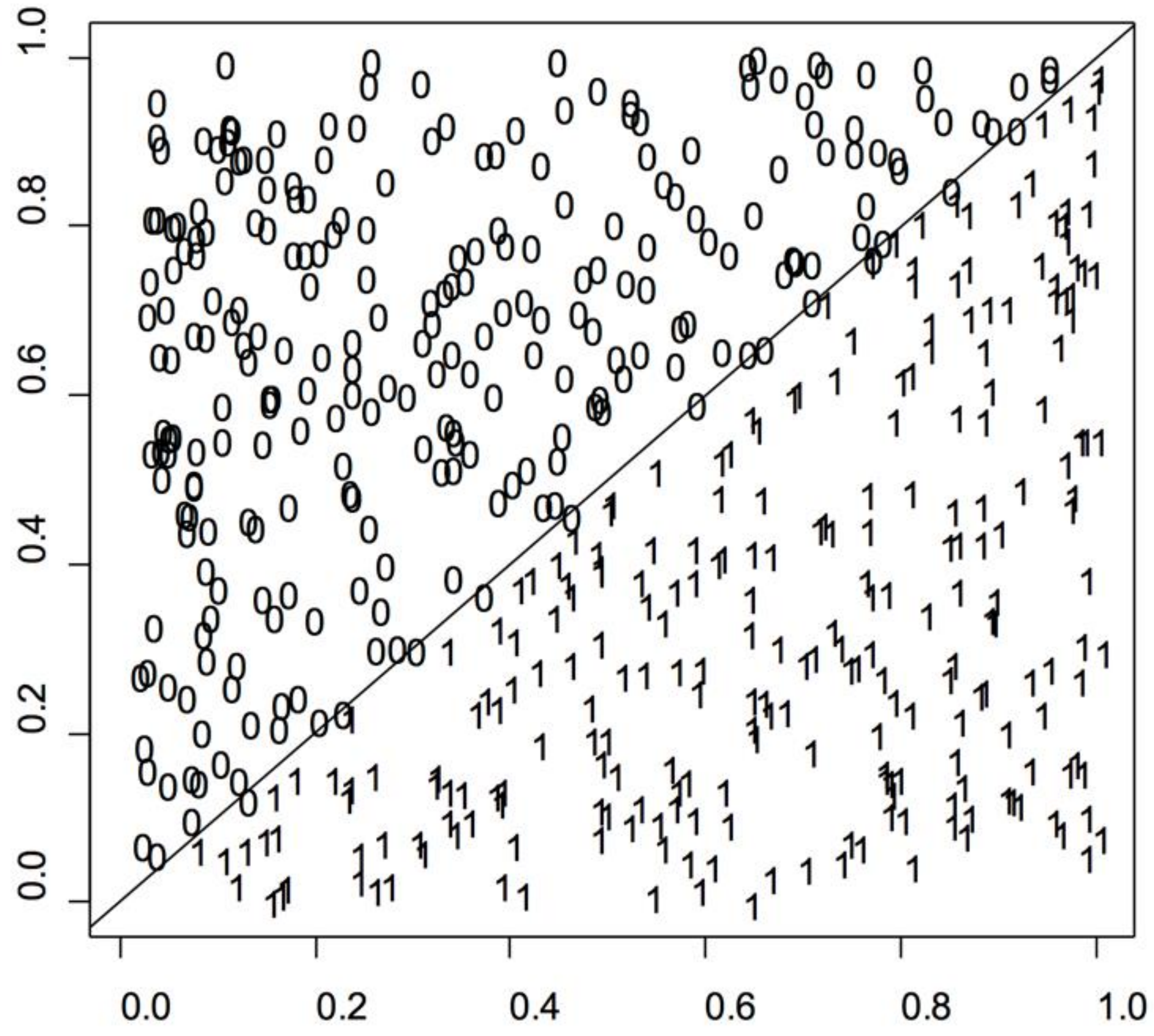
10 Regression Trees



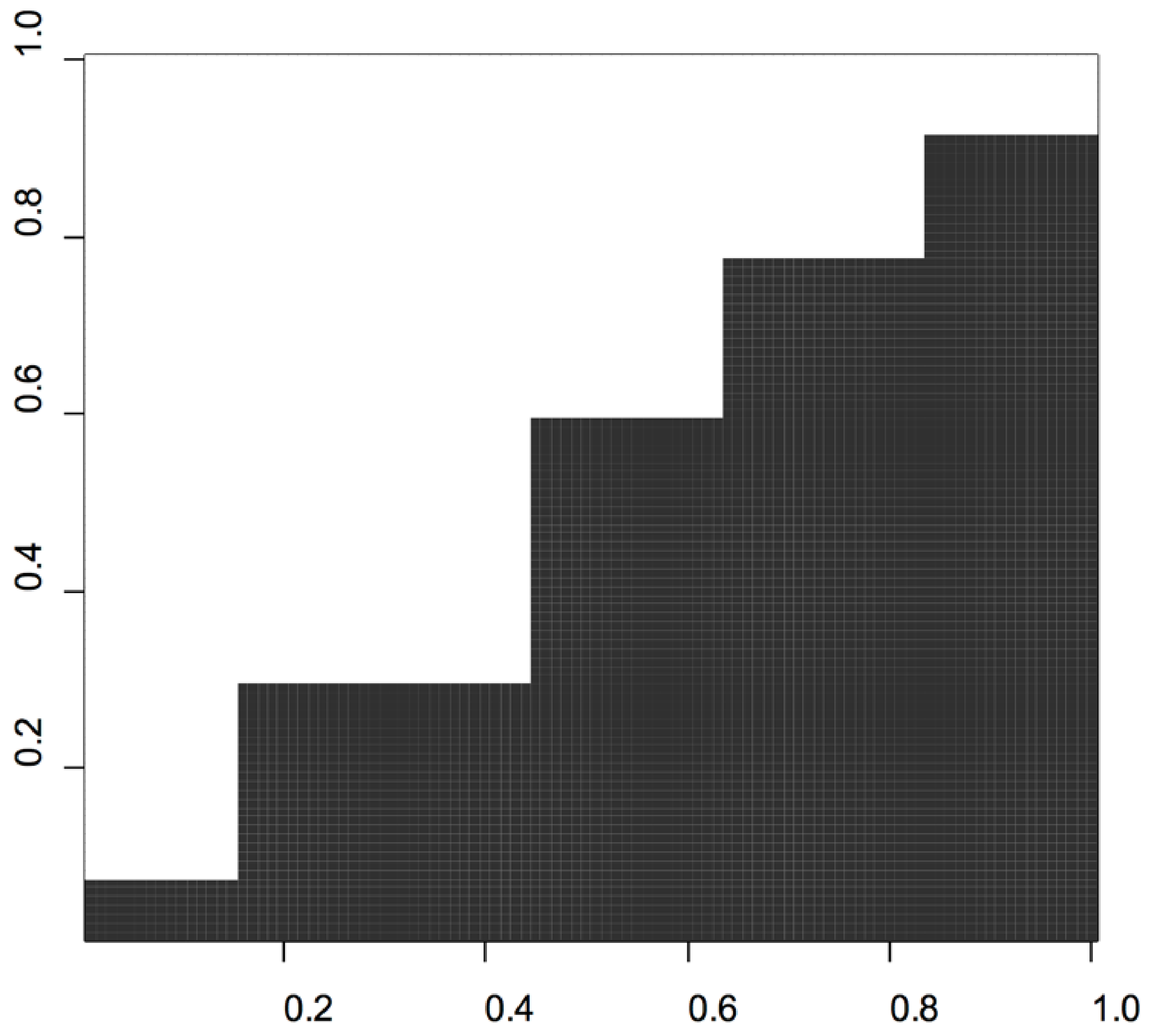
Average of 100 Regression Trees



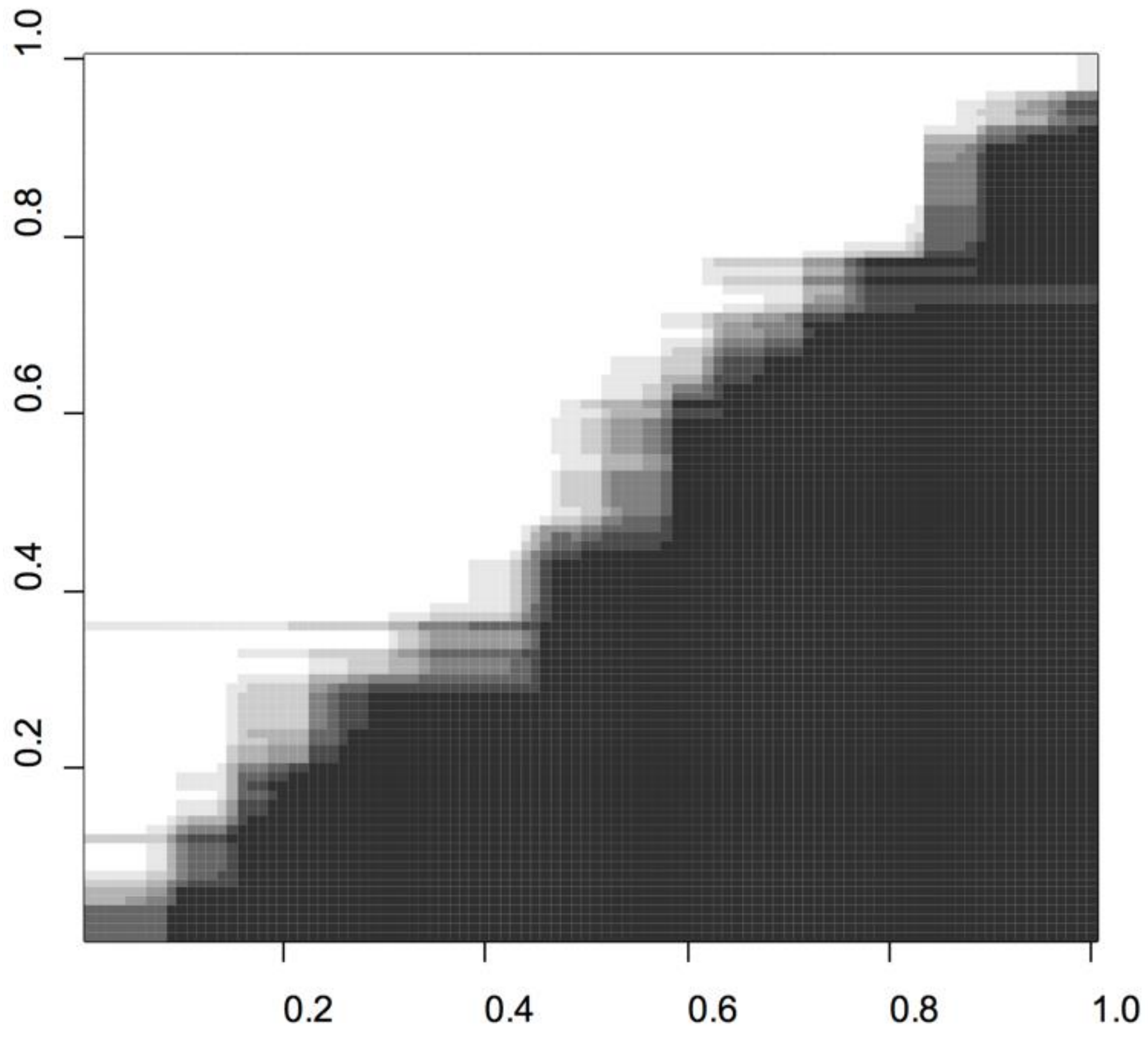
Hard problem for a single tree



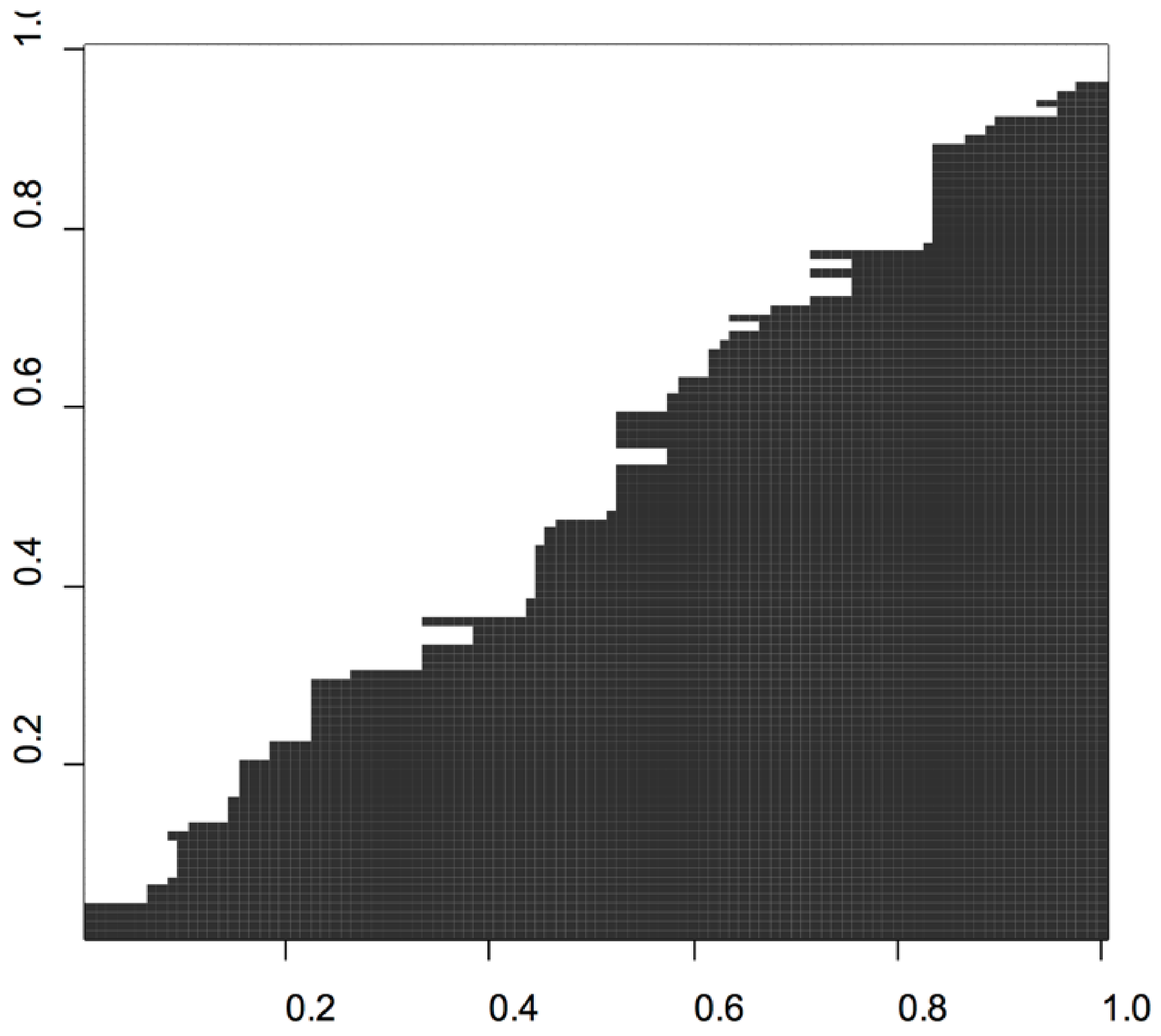
Single Tree



25 Averaged Trees



25 Voted Trees



Bagging (Bootstrap Aggregating)

- Fit classification or regression models to bootstrap samples from the data and combine by voting (classification) or averaging (regression)

$$\left.\begin{array}{l}
\text{Bootstrap sample} \rightarrow f_1(x) \\
\text{Bootstrap sample} \rightarrow f_2(x) \\
\text{Bootstrap sample} \rightarrow f_3(x) \\
\dots \\
\text{Bootstrap sample} \rightarrow f_M(x)
\end{array}\right\} \text{Combine } f_1(x), \dots, f_M(x) \rightarrow f(x)$$

$f_i(x)$'s are "base learners"

Bagging (Bootstrap Aggregating)

- A bootstrap sample is chosen at random *with* replacement from the data. Some observations end up in the bootstrap sample more than once, while others are not included (“out of bag”).
- Bagging reduces the *variance* of the base learner but has limited effect on the *bias*.
- It’s most effective if we use *strong* base learners that have very little bias but high variance (unstable). E.g. trees.

Random Forests

- Grow a **forest** of many trees. (100-500)
- Grow each tree on an independent **bootstrap sample*** from the training data.
- At each node:
  1. Select ***m*** variables **at random** out of all ***M*** possible variables (independently for each node).
  2. Find the best split on the selected ***m*** variables.
- Grow the trees to maximum depth (classification). Vote/average the trees to get predictions for new data.

*Sample N cases at random with replacement.

Random Forests

Inherit many of the advantages of CART:

- Applicable to both regression and classification problems. Yes.
- Handle categorical predictors naturally. Yes.
- Computationally simple and quick to fit, even for large problems. Yes.
- No formal distributional assumptions (non-parametric). Yes.
- Can handle highly non-linear interactions and classification boundaries. Yes.
- Automatic variable selection. Yes. But need variable importance too.
- Handles missing values ~~through surrogate variables.~~ Using proximities.
- ~~Very easy to interpret if the tree is small~~. NO!

Random Forests

Improve on CART with respect to:

- *Accuracy* – Random Forests is competitive with the best known machine learning methods.
- *Instability* – if we change the data a little, the individual trees may change but the forest is relatively stable because it is a combination of many trees.

Two natural questions

- ***Why bootstrap? (Why subsample?)***
  - Bootstrapping → out-of-bag data →
    - Estimated error rate and confusion matrix
    - Variable importance
- ***Why trees?***
  - Trees → proximities →
    - Missing value fill-in
    - Outlier detection
    - Illuminating pictures of the data (clusters, structure, outliers)

Load data

```
[1]  from google.colab import drive
     drive.mount('/content/drive')
```

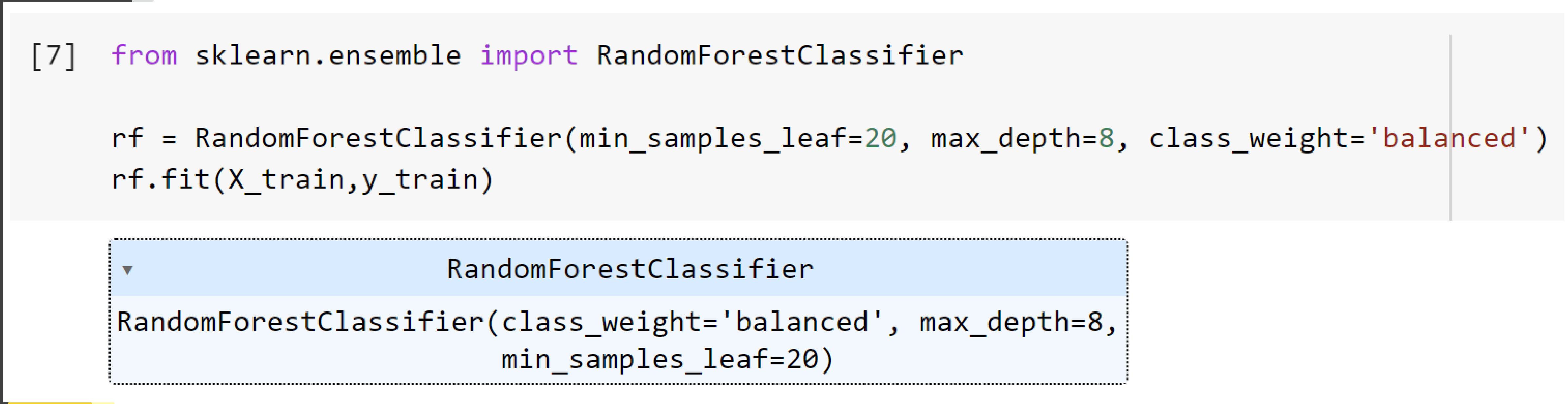
```
Mounted at /content/drive
```

```
[5]  import pickle

     X_train, X_test, y_train, y_test = \
         pickle.load(open('/content/drive/MyDrive/GSB/hr_attrition.data','rb'))
```

```
[6]  print(X_train.shape)
     print(X_test.shape)
     print(y_train.shape)
     print(y_test.shape)
```

```
(1029, 44)
(441, 44)
(1029,)
(441,)
```

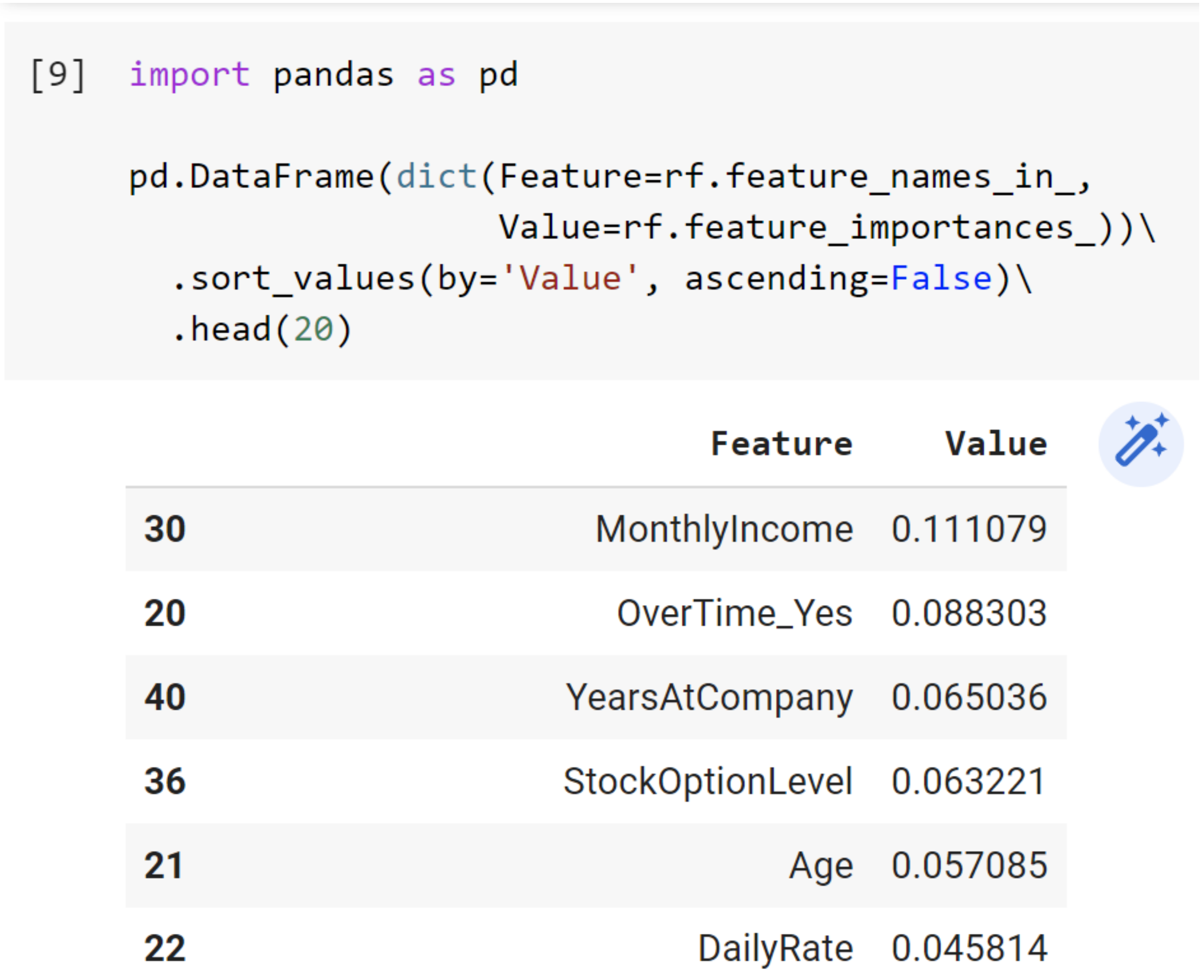
Modeling - RandomForestClassifier



```
[7]  from sklearn.ensemble import RandomForestClassifier

     rf = RandomForestClassifier(min_samples_leaf=20, max_depth=8, class_weight='balanced')
     rf.fit(X_train,y_train)
```

```
▾                    RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=8,
                       min_samples_leaf=20)
```

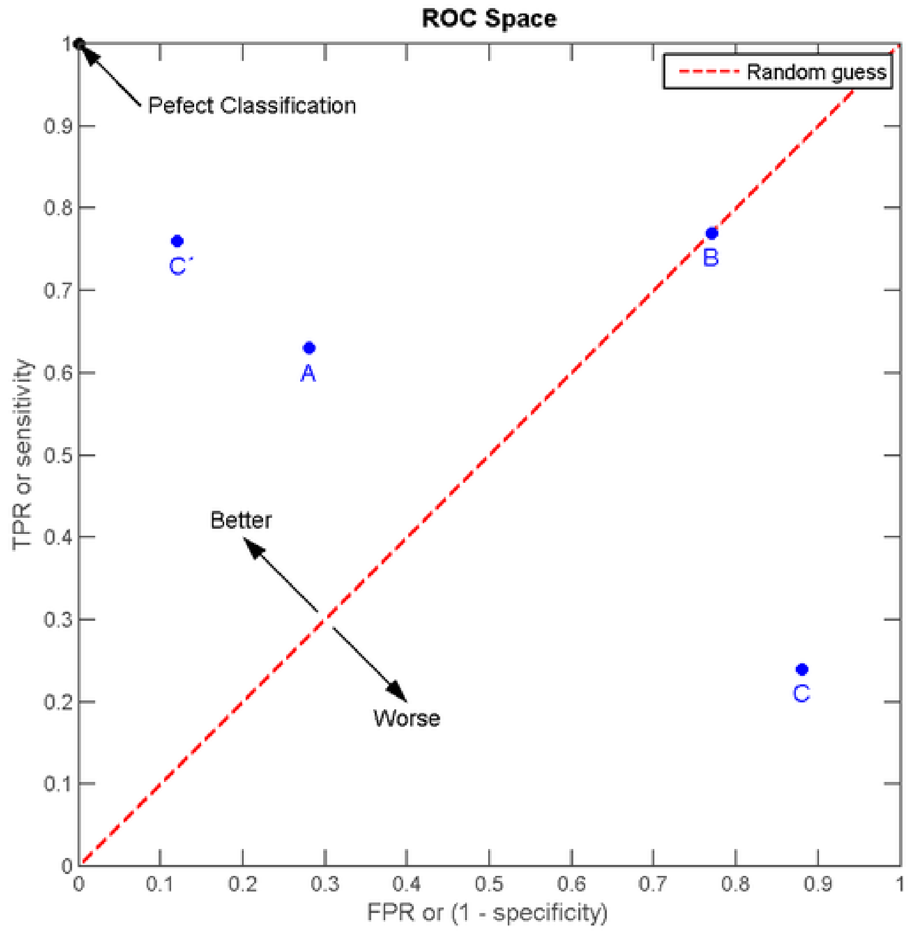
Random Forest – Feature Importance

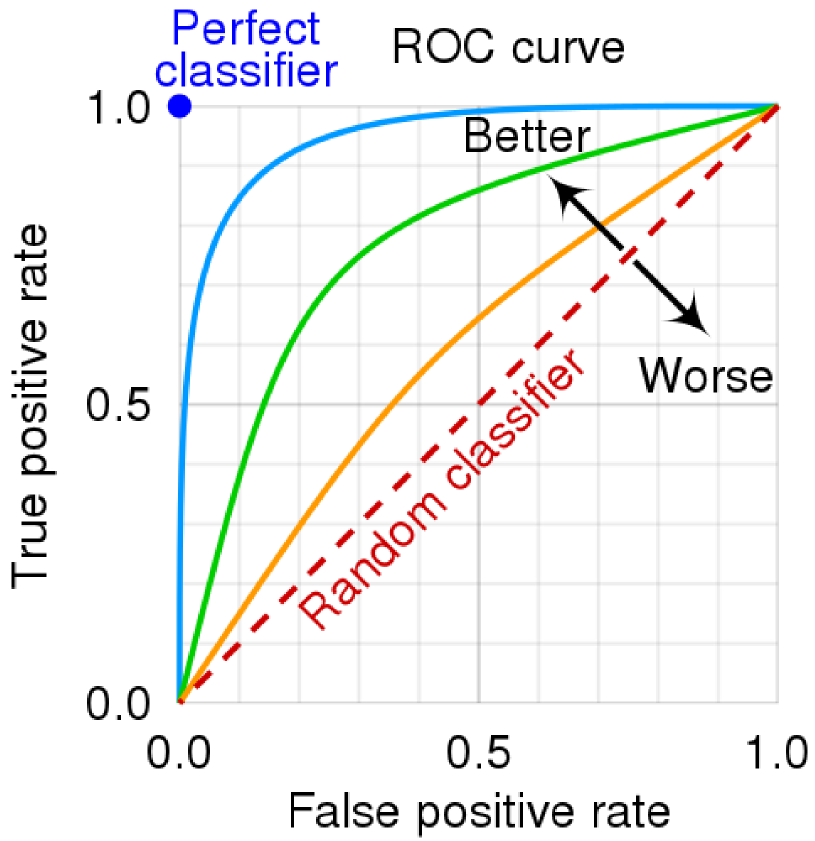


```
[9] import pandas as pd

    pd.DataFrame(dict(Feature=rf.feature_names_in_,
                      Value=rf.feature_importances_))\
      .sort_values(by='Value', ascending=False)\
      .head(20)
```

```
             Feature     Value
30     MonthlyIncome  0.111079
20      OverTime_Yes  0.088303
40    YearsAtCompany  0.065036
36  StockOptionLevel  0.063221
21               Age  0.057085
22         DailyRate  0.045814
```

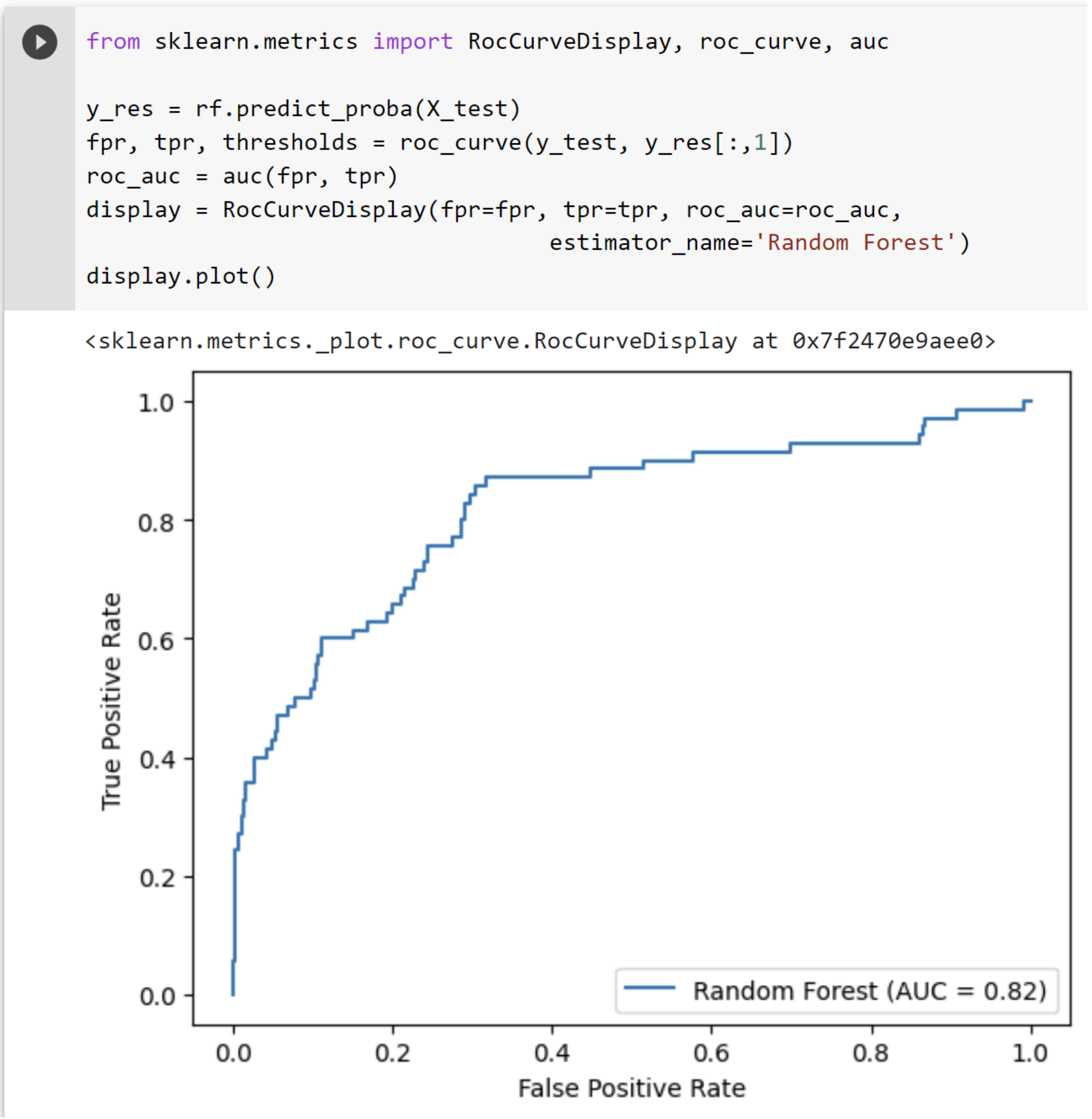
Receiver Operating Characteristics





Evaluation

AUC = 0.82



```
from sklearn.metrics import RocCurveDisplay, roc_curve, auc

y_res = rf.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_res[:,1])
roc_auc = auc(fpr, tpr)
display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,
                                  estimator_name='Random Forest')
display.plot()
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f2470e9aee0>
```

Bagging vs Boosting

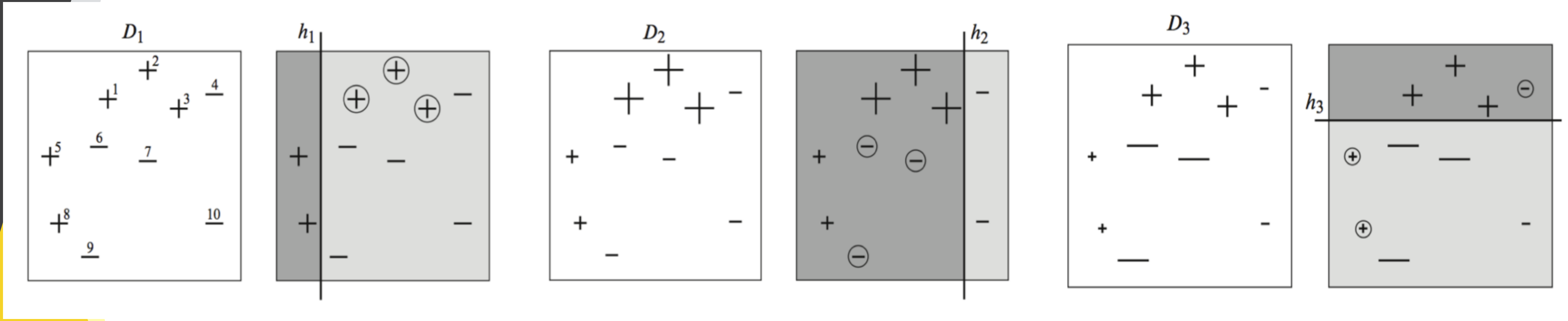
- Bagging (random forests) perform well for multi-class object detection, which tends to have a lot of statistical noise.

- Gradient Boosting performs well when you have unbalanced data such as in real time risk assessment.

Boosting

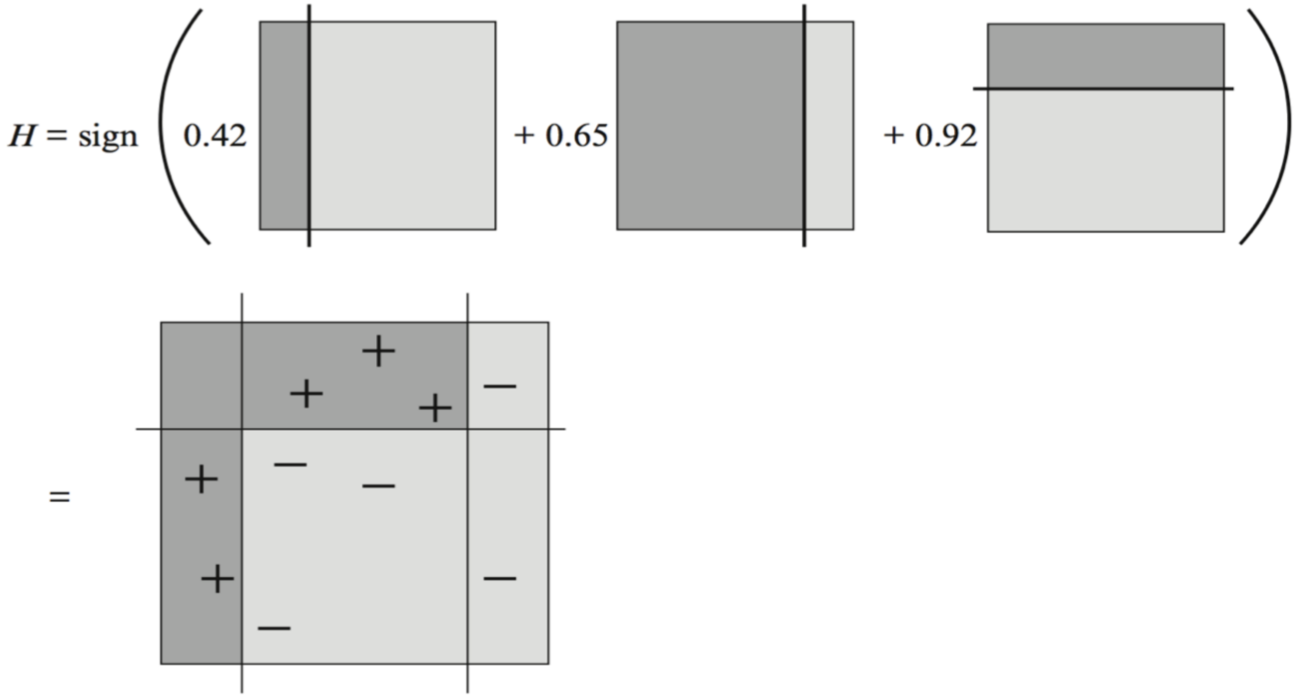
AdaBoost (Schapire and Freund, 2012)

- Fit additive model $\sum_t c_t h_t(x)$
- In each stage, introduce a weak learner to compensate the shortcoming of existing weak learner



AdaBoost

$$H(x) = \sum_t \rho_t h_t(x)$$



Gradient Boosting

- Gradient boosting = gradient descent + boosting
- Fit additive model $\sum_t c_t h_t(x)$
- In each stage, introduce a weak learner to compensate the shortcoming of existing weak learner
- In Gradient Boosting,“shortcomings” are identified by gradients. (residual)
- Both high-weight data points and gradients tell us how to improve our model.

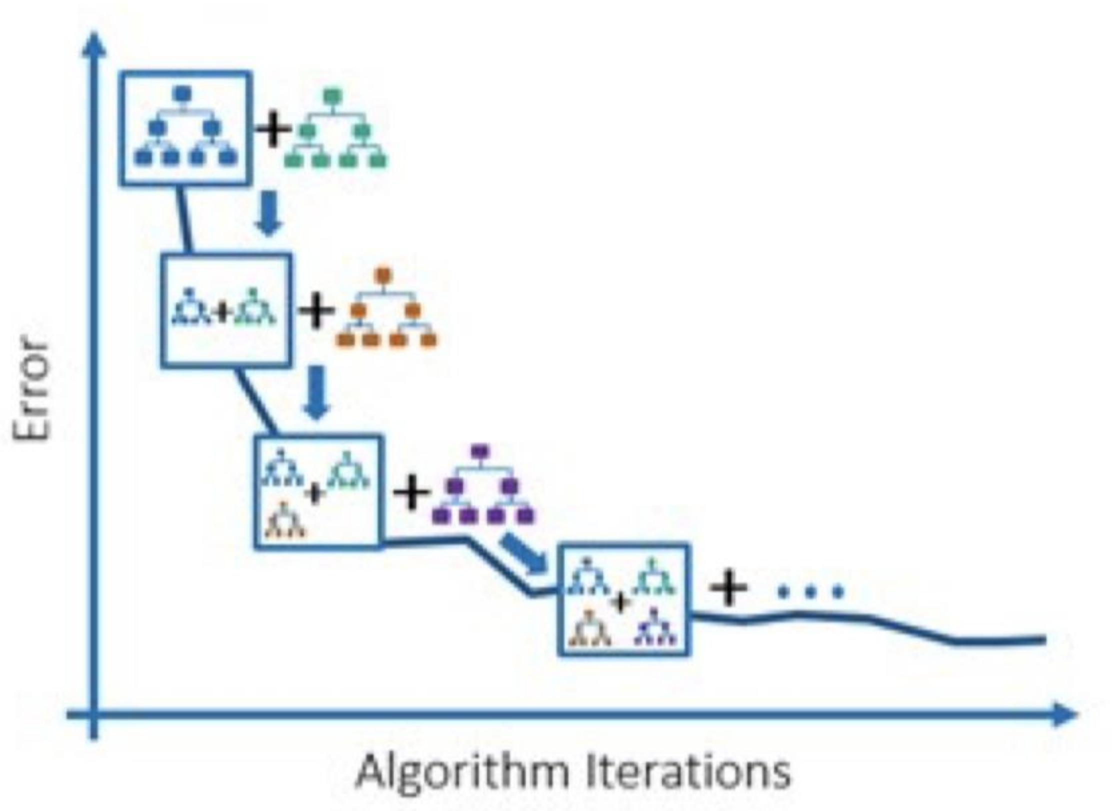
Gradient

- Gradient is calculated from the derivative of the loss function.

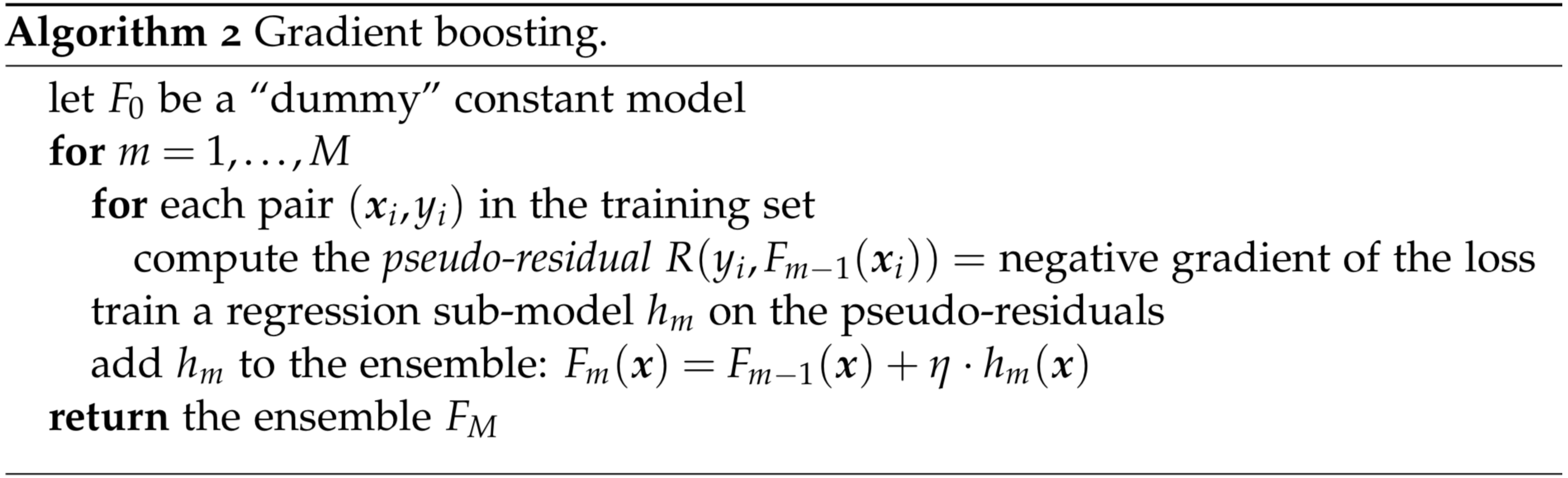
$$\text{Loss}(y_i, \hat{y}) = (y_i - \hat{y})^2$$

$$-\nabla_{\text{Loss}}(\hat{y}) = 2 \cdot (y_i - \hat{y})$$

Gradient boosting



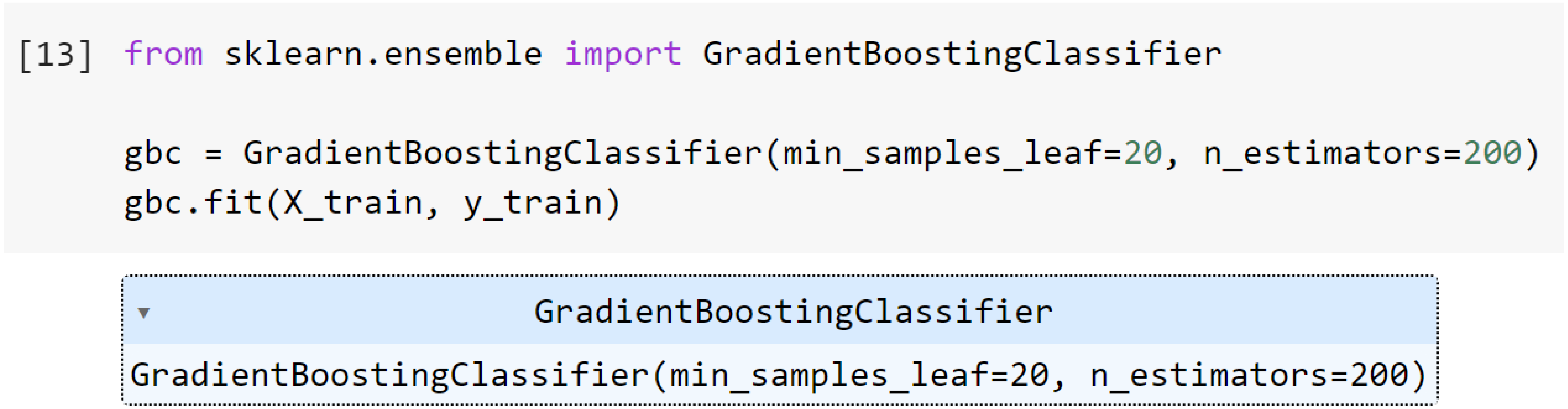
Gradient boosting algorithm



Algorithm 2 Gradient boosting.

```
let $F_0$ be a "dummy" constant model
for $m = 1, \ldots, M$
    for each pair $(\boldsymbol{x}_i, y_i)$ in the training set
        compute the *pseudo-residual* $R(y_i, F_{m-1}(\boldsymbol{x}_i))$ = negative gradient of the loss
    train a regression sub-model $h_m$ on the pseudo-residuals
    add $h_m$ to the ensemble: $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \eta \cdot h_m(\boldsymbol{x})$
return the ensemble $F_M$
```

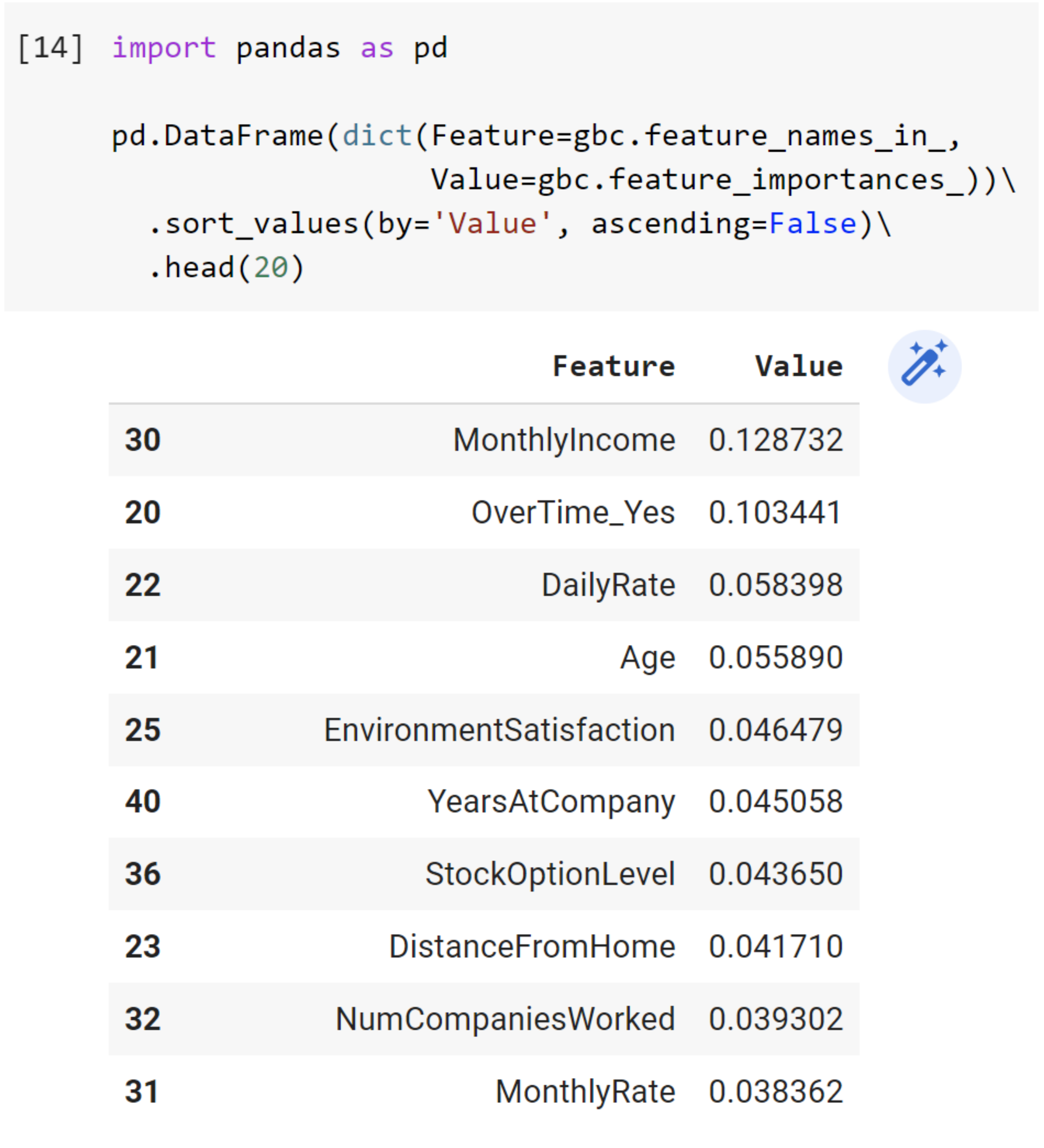
Modeling – GradientBoostingClassifier



```
[13] from sklearn.ensemble import GradientBoostingClassifier

     gbc = GradientBoostingClassifier(min_samples_leaf=20, n_estimators=200)
     gbc.fit(X_train, y_train)
```

```
▾                GradientBoostingClassifier
GradientBoostingClassifier(min_samples_leaf=20, n_estimators=200)
```

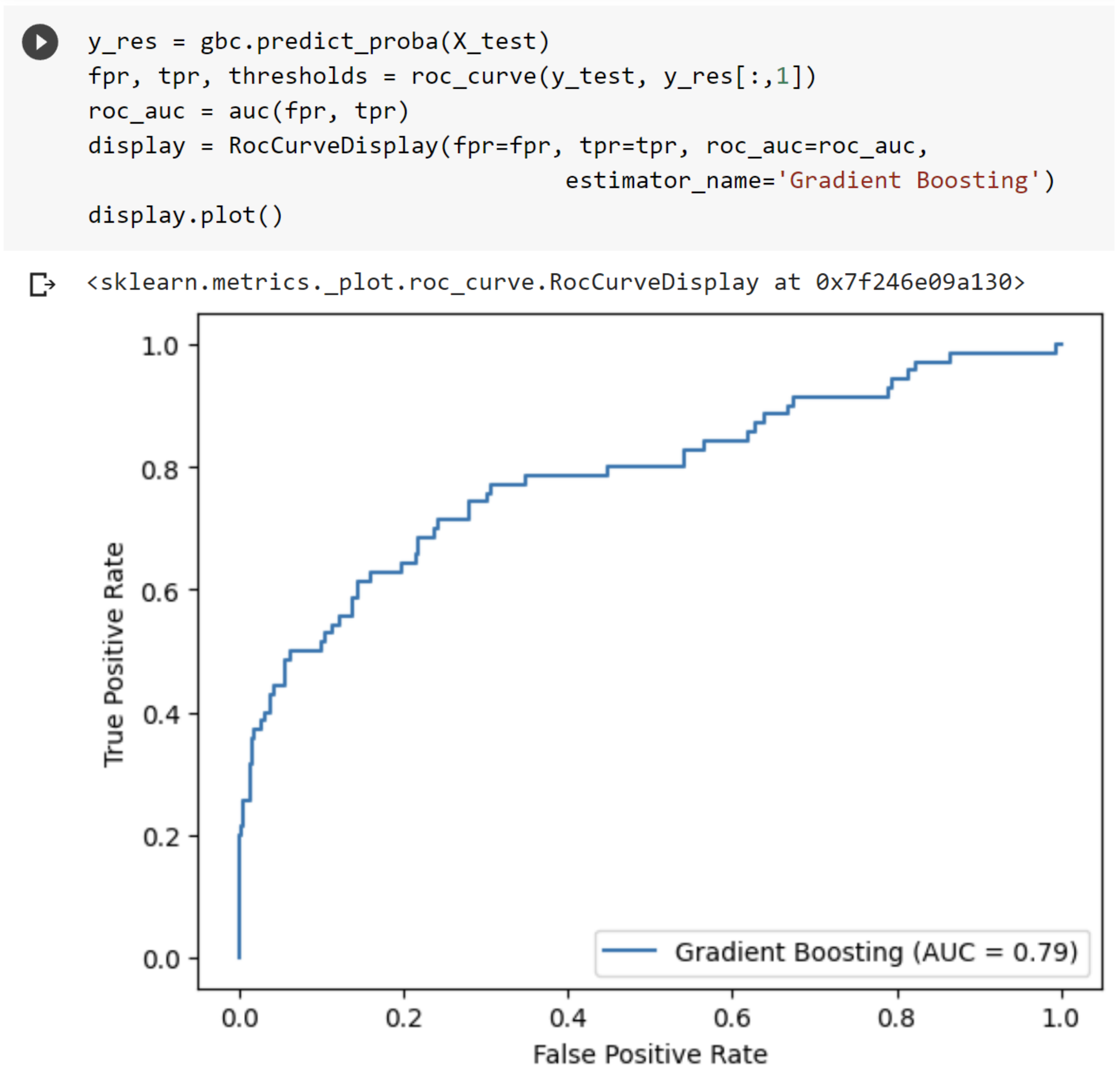
Feature importance



```
[14] import pandas as pd

     pd.DataFrame(dict(Feature=gbc.feature_names_in_,
                       Value=gbc.feature_importances_))\
       .sort_values(by='Value', ascending=False)\
       .head(20)
```

	Feature	Value
30	MonthlyIncome	0.128732
20	OverTime_Yes	0.103441
22	DailyRate	0.058398
21	Age	0.055890
25	EnvironmentSatisfaction	0.046479
40	YearsAtCompany	0.045058
36	StockOptionLevel	0.043650
23	DistanceFromHome	0.041710
32	NumCompaniesWorked	0.039302
31	MonthlyRate	0.038362

Evaluation



```
y_res = gbc.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_res[:,1])
roc_auc = auc(fpr, tpr)
display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,
                                  estimator_name='Gradient Boosting')
display.plot()
```

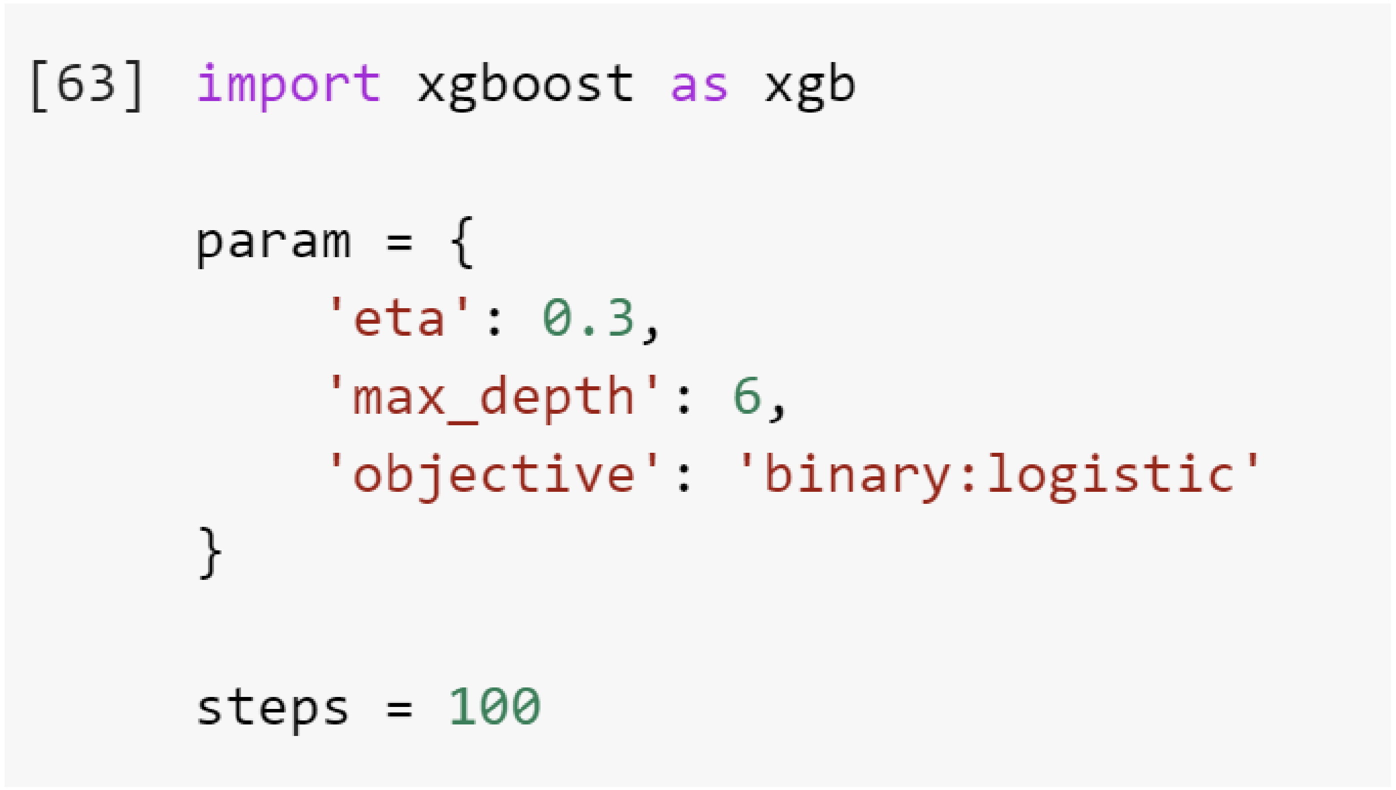
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f246e09a130>
```

eXtreme Gradient Boosting (xgboost)

xgboost is an optimized version of gradient boosting with these added features

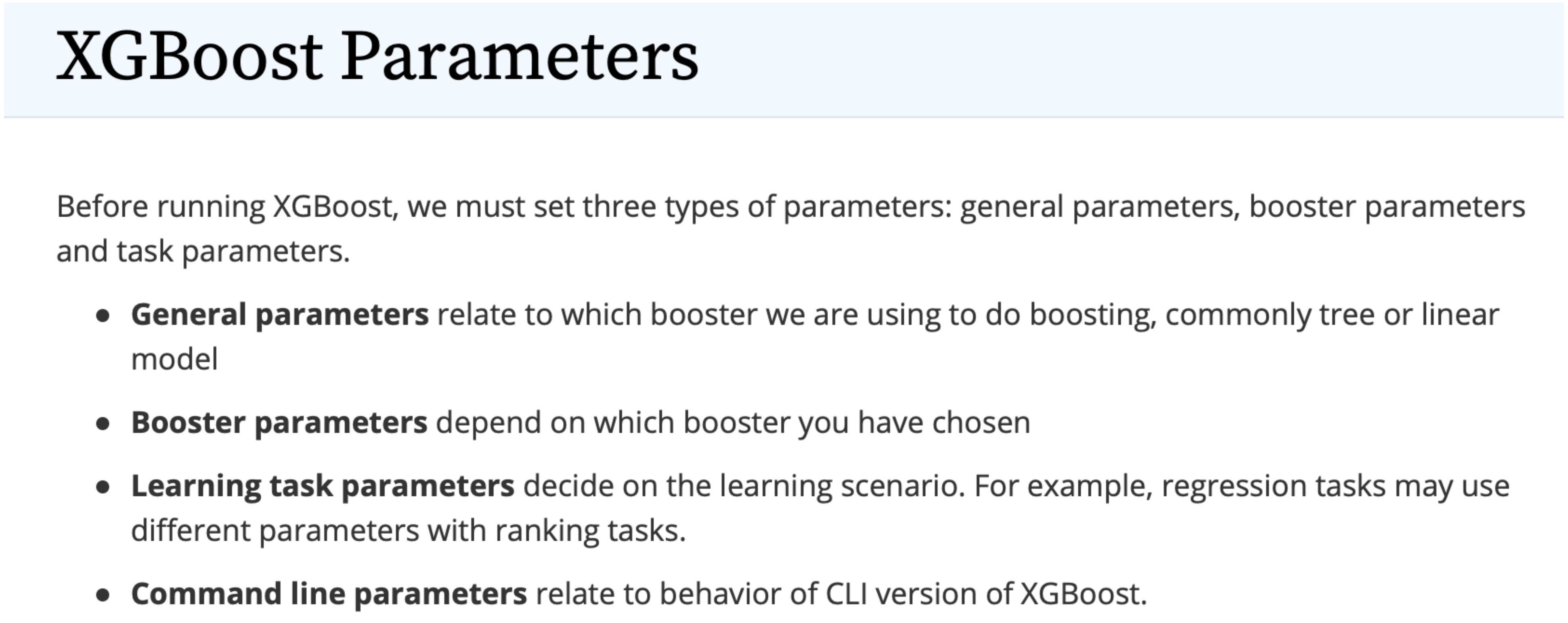
- Regularization – xgboost provides both L1 and L2 regularization in model training
- Sparsity awareness – Incorporate missing data as criteria in feature selection
- Parallelization – allow parallelism of algorithm in either parallel or distributed environment
- Cache aware access – the learning algorithm efficiently utilizes the cache

Prepare parameters

- 

```
[63] import xgboost as xgb

     param = {
         'eta': 0.3,
         'max_depth': 6,
         'objective': 'binary:logistic'
     }

     steps = 100
```

XGBoost Parameters



XGBoost Parameters

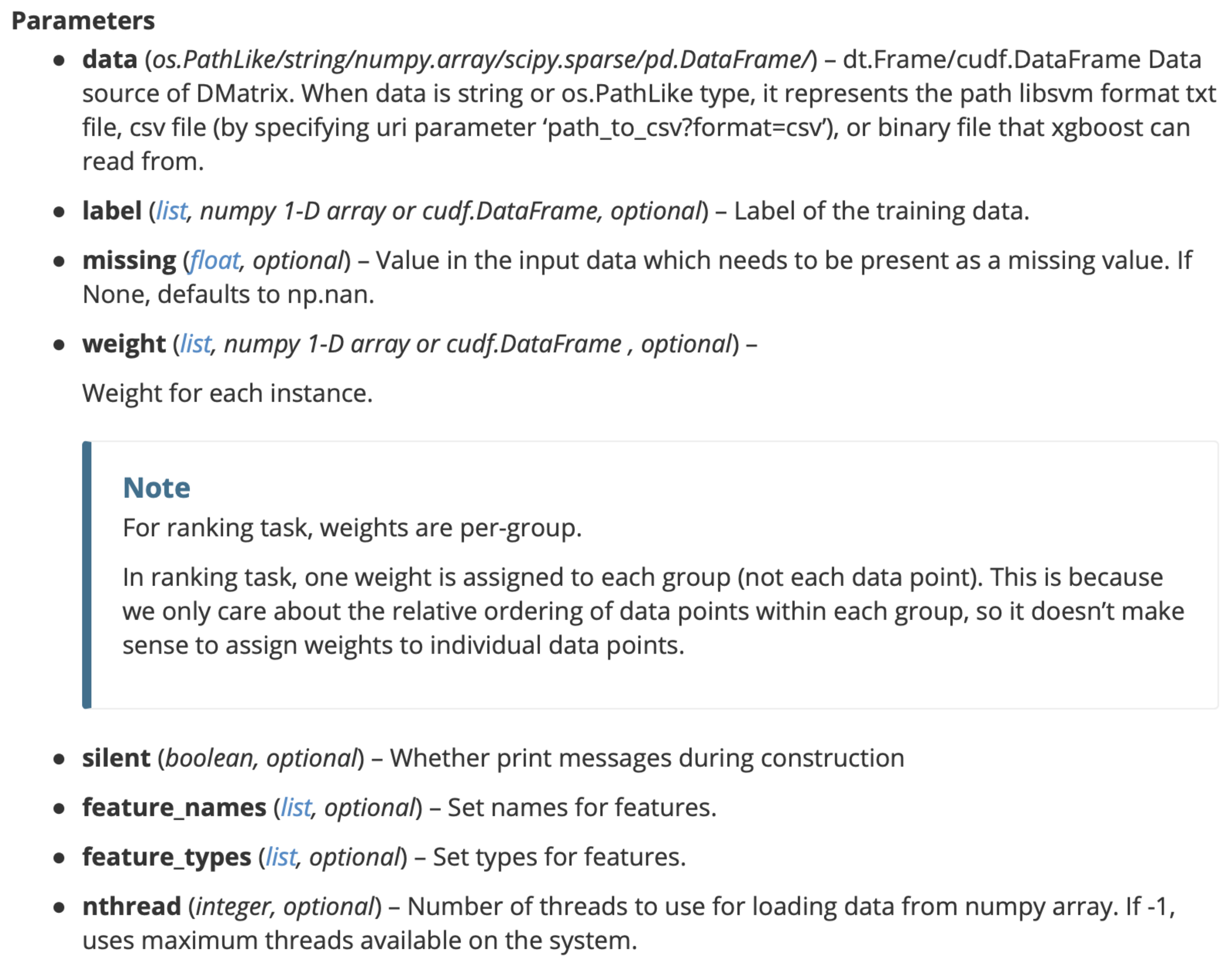
Before running XGBoost, we must set three types of parameters: general parameters, booster parameters and task parameters.

- **General parameters** relate to which booster we are using to do boosting, commonly tree or linear model
- **Booster parameters** depend on which booster you have chosen
- **Learning task parameters** decide on the learning scenario. For example, regression tasks may use different parameters with ranking tasks.
- **Command line parameters** relate to behavior of CLI version of XGBoost.

https://xgboost.readthedocs.io/en/latest/parameter.html

Prepare data

DMatrix is a internal data structure that used by XGBoost which is optimized for both memory efficiency and training speed.



Parameters

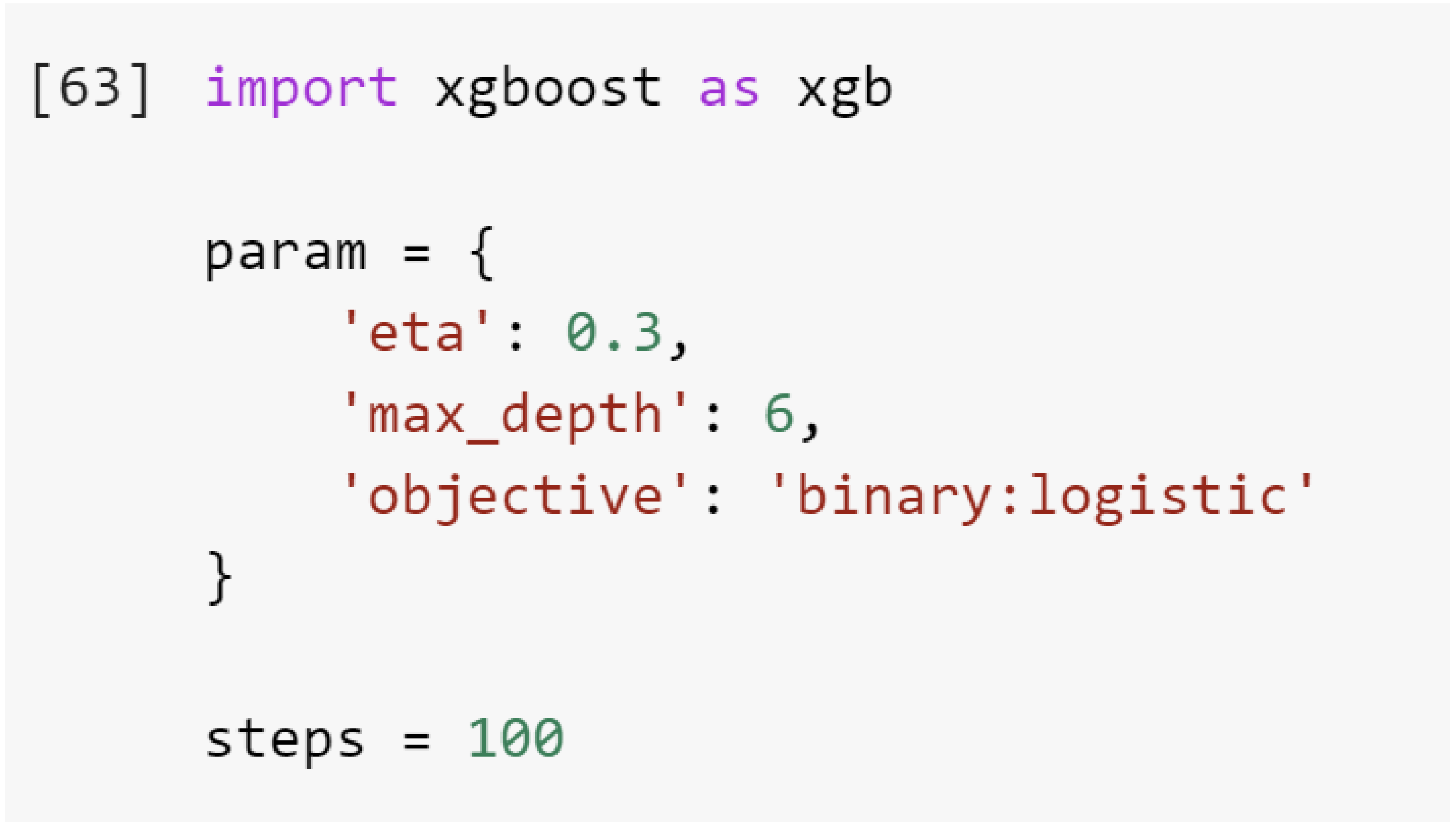
- **data** (*os.PathLike/string/numpy.array/scipy.sparse/pd.DataFrame/*) – dt.Frame/cudf.DataFrame Data source of DMatrix. When data is string or os.PathLike type, it represents the path libsvm format txt file, csv file (by specifying uri parameter 'path_to_csv?format=csv'), or binary file that xgboost can read from.
- **label** (*list, numpy 1-D array or cudf.DataFrame, optional*) – Label of the training data.
- **missing** (*float, optional*) – Value in the input data which needs to be present as a missing value. If None, defaults to np.nan.
- **weight** (*list, numpy 1-D array or cudf.DataFrame , optional*) –

  Weight for each instance.

  > **Note**
  >
  > For ranking task, weights are per-group.
  >
  > In ranking task, one weight is assigned to each group (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn't make sense to assign weights to individual data points.

- **silent** (*boolean, optional*) – Whether print messages during construction
- **feature_names** (*list, optional*) – Set names for features.
- **feature_types** (*list, optional*) – Set types for features.
- **nthread** (*integer, optional*) – Number of threads to use for loading data from numpy array. If -1, uses maximum threads available on the system.

Modeling

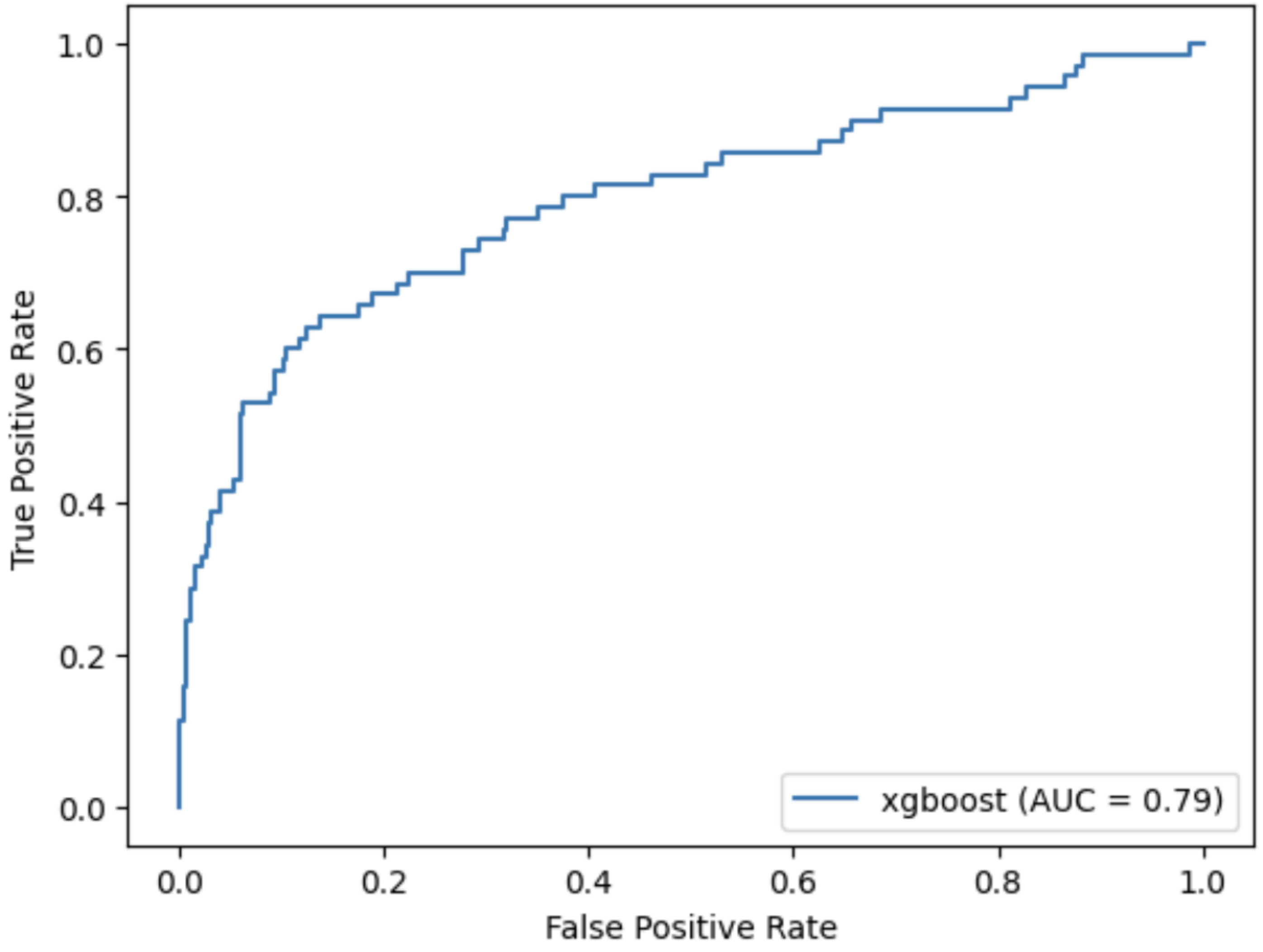


```
[63] import xgboost as xgb

     param = {
         'eta': 0.3,
         'max_depth': 6,
         'objective': 'binary:logistic'
     }

     steps = 100
```

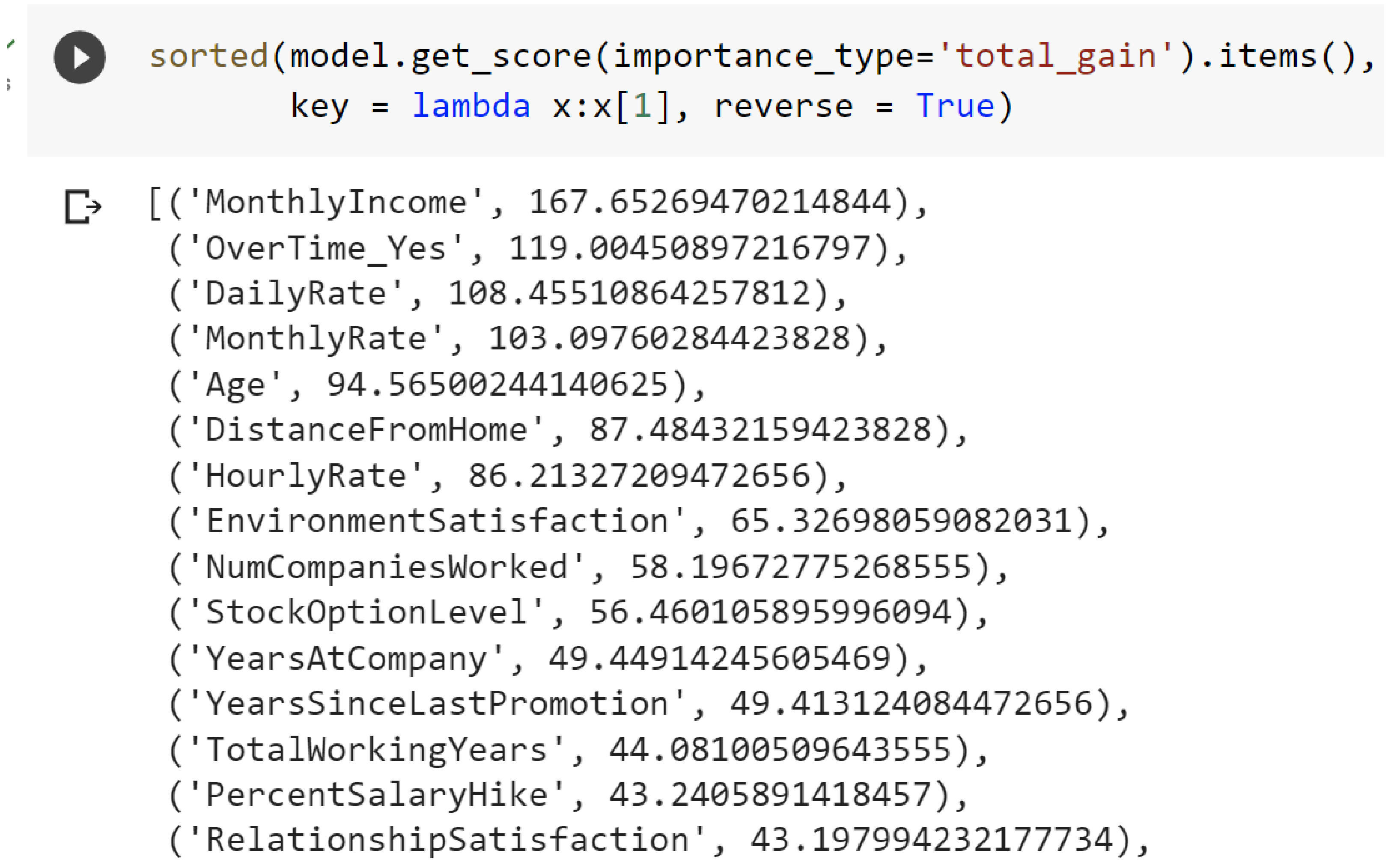
Evaluation

```
fpr, tpr, thresholds = roc_curve(y_test, p)
roc_auc = auc(fpr, tpr)
display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,
                                  estimator_name='xgboost')
display.plot()
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f245a17c250>
```



Feature importance



```python
sorted(model.get_score(importance_type='total_gain').items(),
       key = lambda x:x[1], reverse = True)
```

```
[('MonthlyIncome', 167.65269470214844),
 ('OverTime_Yes', 119.00450897216797),
 ('DailyRate', 108.45510864257812),
 ('MonthlyRate', 103.09760284423828),
 ('Age', 94.56500244140625),
 ('DistanceFromHome', 87.48432159423828),
 ('HourlyRate', 86.21327209472656),
 ('EnvironmentSatisfaction', 65.32698059082031),
 ('NumCompaniesWorked', 58.19672775268555),
 ('StockOptionLevel', 56.460105895996094),
 ('YearsAtCompany', 49.4491424560546 9),
 ('YearsSinceLastPromotion', 49.413124084472656),
 ('TotalWorkingYears', 44.08100509643555),
 ('PercentSalaryHike', 43.2405891418457),
 ('RelationshipSatisfaction', 43.197994232177734),
```

Summary

- Ensemble learning allows us to generalize the machine learning model to further improve the model performance
- Bagging is robust to noise
- Boosting allows us to squeeze the performance by modeling the residuals

Lab

- Select your own classification problem data from Kaggle.com website
- Prepare the data for classification modeling
- Build a decision tree classifier for the problem
- Evaluate the modeling result

End of Lecture 5