Machine Learning

Lecture 5: Tree-Based and Ensemble Models

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 5 - Topics

- Decision tree classification and regression
- Bagging
- Boosting

Decision tree classification and regression

https://drive.google.com/file/d/12eegDnFPtqL42rJcuUOViJjbnFuet2uO/view?usp=sharing

Predictive Modeling as Supervised Segmentation

- How can we segment the population into groups that differ from each other with respect to some quantity of interest?

Quantity of interest
=
Things we would like to predict or estimate

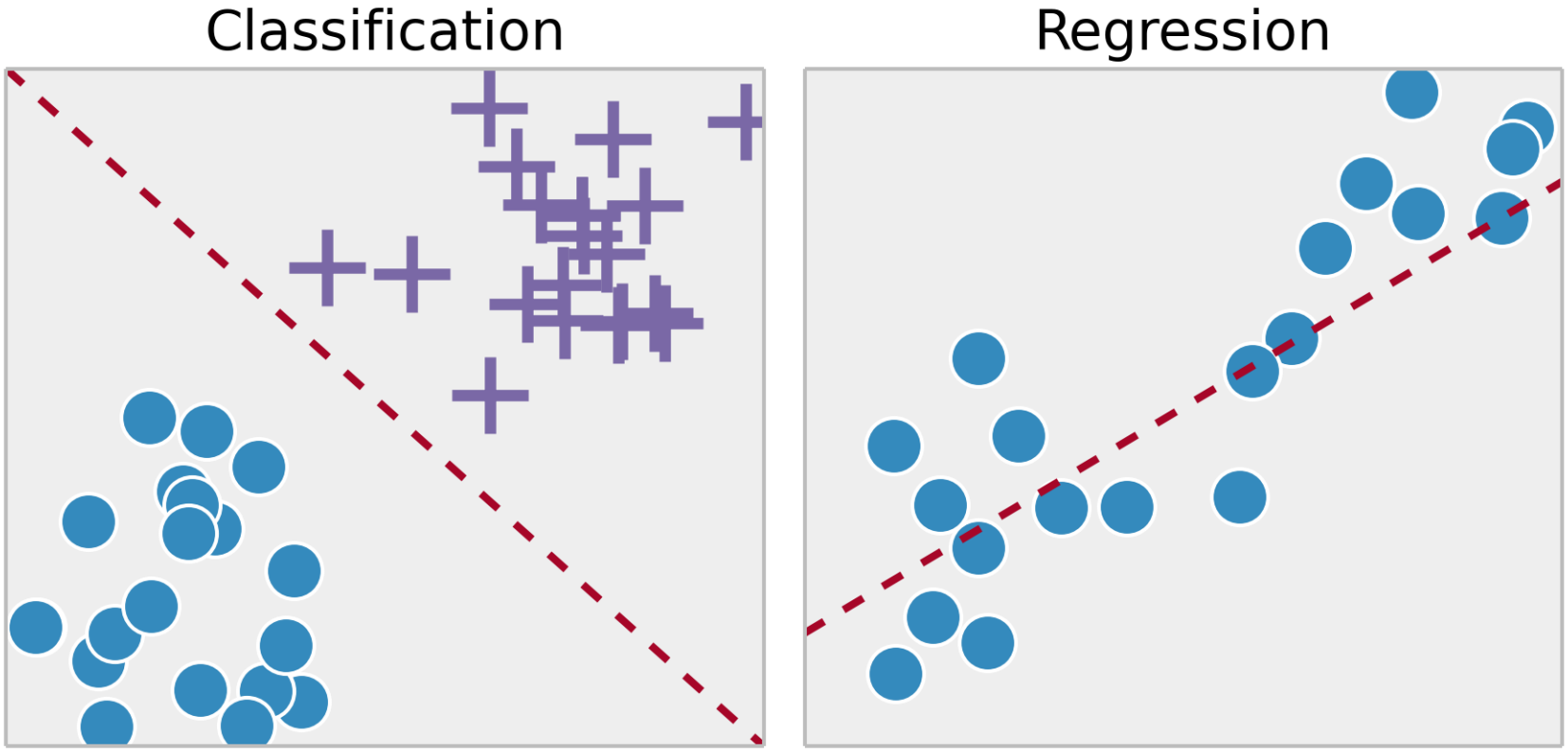
Target and Attribute

target	*Attribute 1*	*Attribute 2*	…	*Attribute k*
Data record → y_1	x_{11}	x_{12}	⋯	x_{1k}
y_2	x_{21}	x_{22}	⋯	x_{2k}
⋮	⋮	⋮		⋮
y_n	x_{n1}	x_{n2}	⋯	x_{nk}

Target attribute / class to predict

Attributes

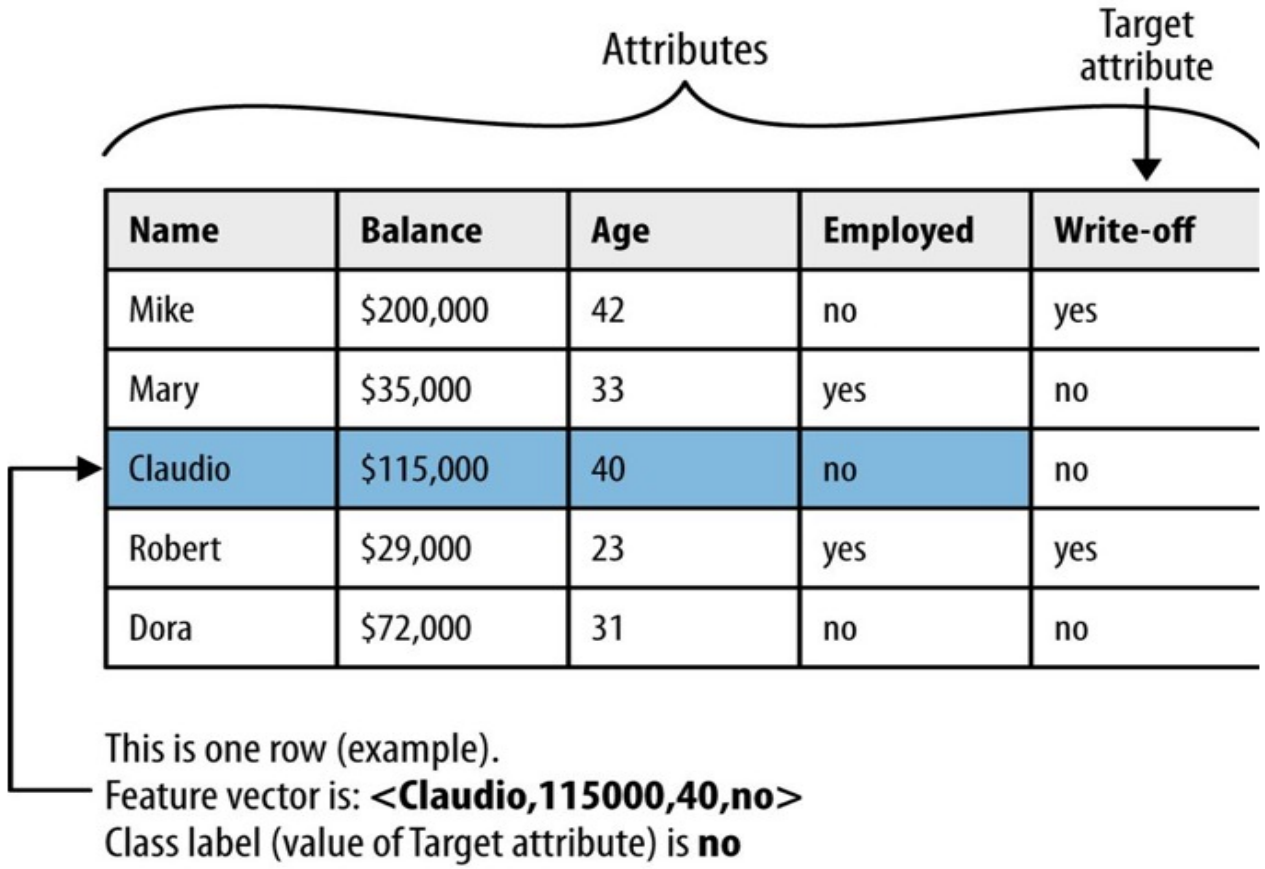
Classification vs Regression



Discrete Target (class)

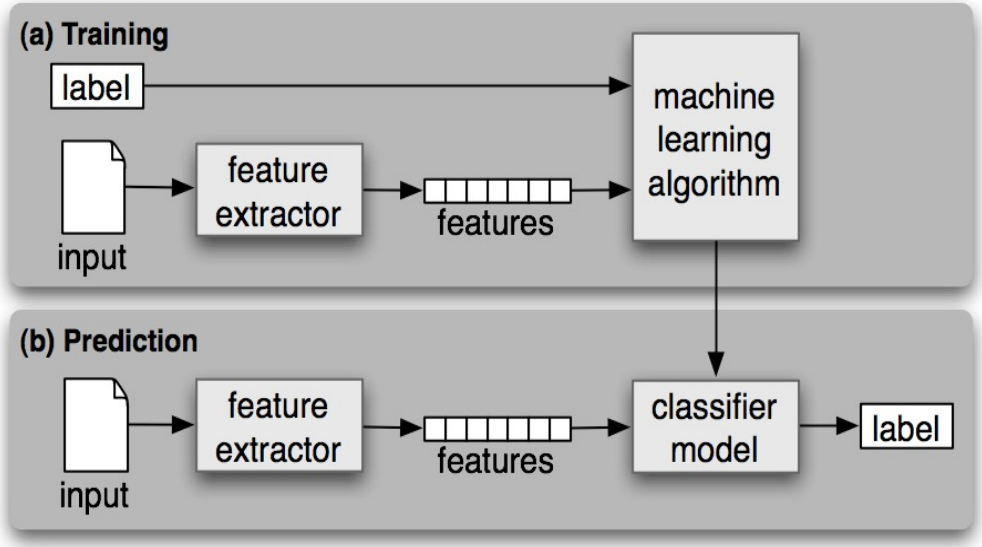
Continuous Target (value)

Supervised classification Example



Induction (Training)

- The creation of models from data is known as model induction (training).
- Induction is a term from philosophy that refers to generalizing from specific cases to general rules
- Our models are general rules in a statistical sense
- Most inductive procedures have variants that induce models both for classification and regression



Which attribute should be used to segment?

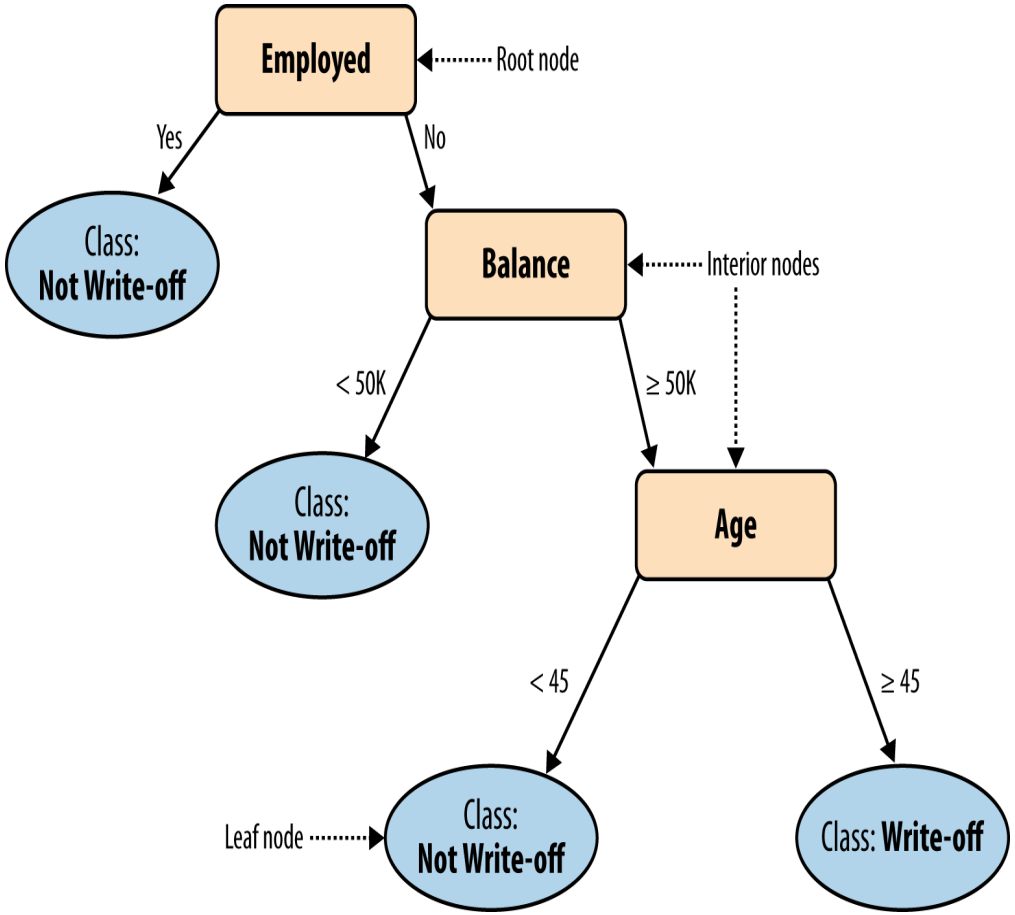
แบ่งยังไงให้ data กลุ่มเดียวกัน เหมือนกัน

- **Fundamental concept**: Which variable contains the most information?

- **Aims**: automatic selection, ranking

Decision Tree

- A tree consists of nodes: interior and terminal
- Interior node contains a test of an attribute
- Terminal node is a segment

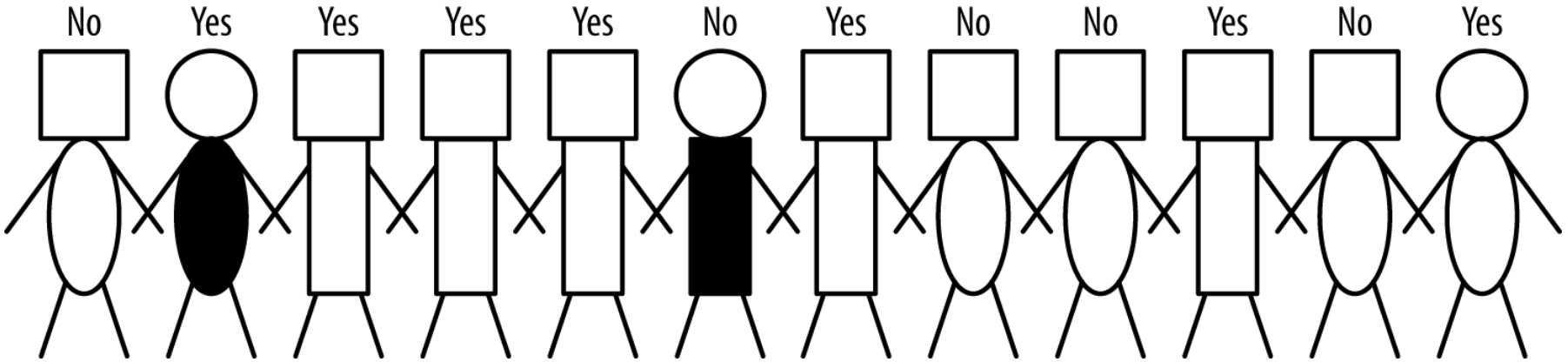


<Claudio, 115000, 40, No>?

Tree Induction

- How do we create a decision tree from data?
- Tree induction takes a divide-and-conquer approach,
  1. starting with the whole dataset
  2. applying variable selection to create subgroups
  3. Recursively repeating step 2 for each subgroup
- We will illustrate this using the write-off example

Example: Write-off



- Attributes
  - head-shape: square, circular
  - body-shape: rectangular, oval
  - body-color: black, white
- Target variable
  - write-off: Yes, No

Which attribute should be best to segment these people into groups, in a way to distinguish write-off from non-write-off?

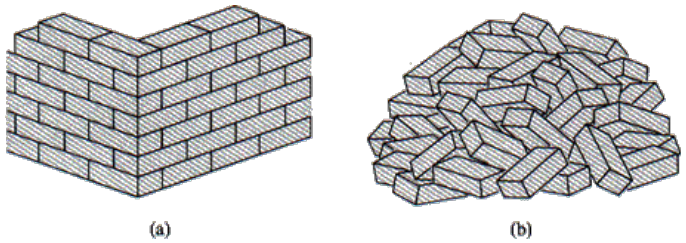
Purity

- Technically, we would like the group to be as pure as possible.
- Pure means homogeneous with respect to the target variable
- If some member in the group has a different target then the group is impure
- Comparing
  - G1 = {Y, Y, Y, Y}
  - G2 = {Y, N, N, Y}

In real data, however, we rarely find pure segments.

Entropy

- Entropy is a measure of disorder that can be applied to a set, such as one of our individual segments
- Disorder corresponds to how mixed (impure) the segment is with respect to the target
- For example, a mixed up segment with lots of write-offs and lots off non-write-offs would have high entropy

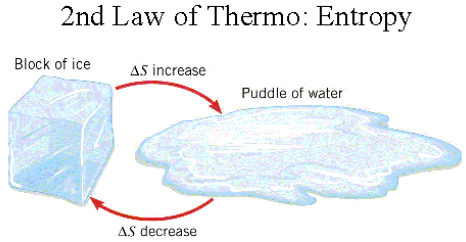


Entropy Formula

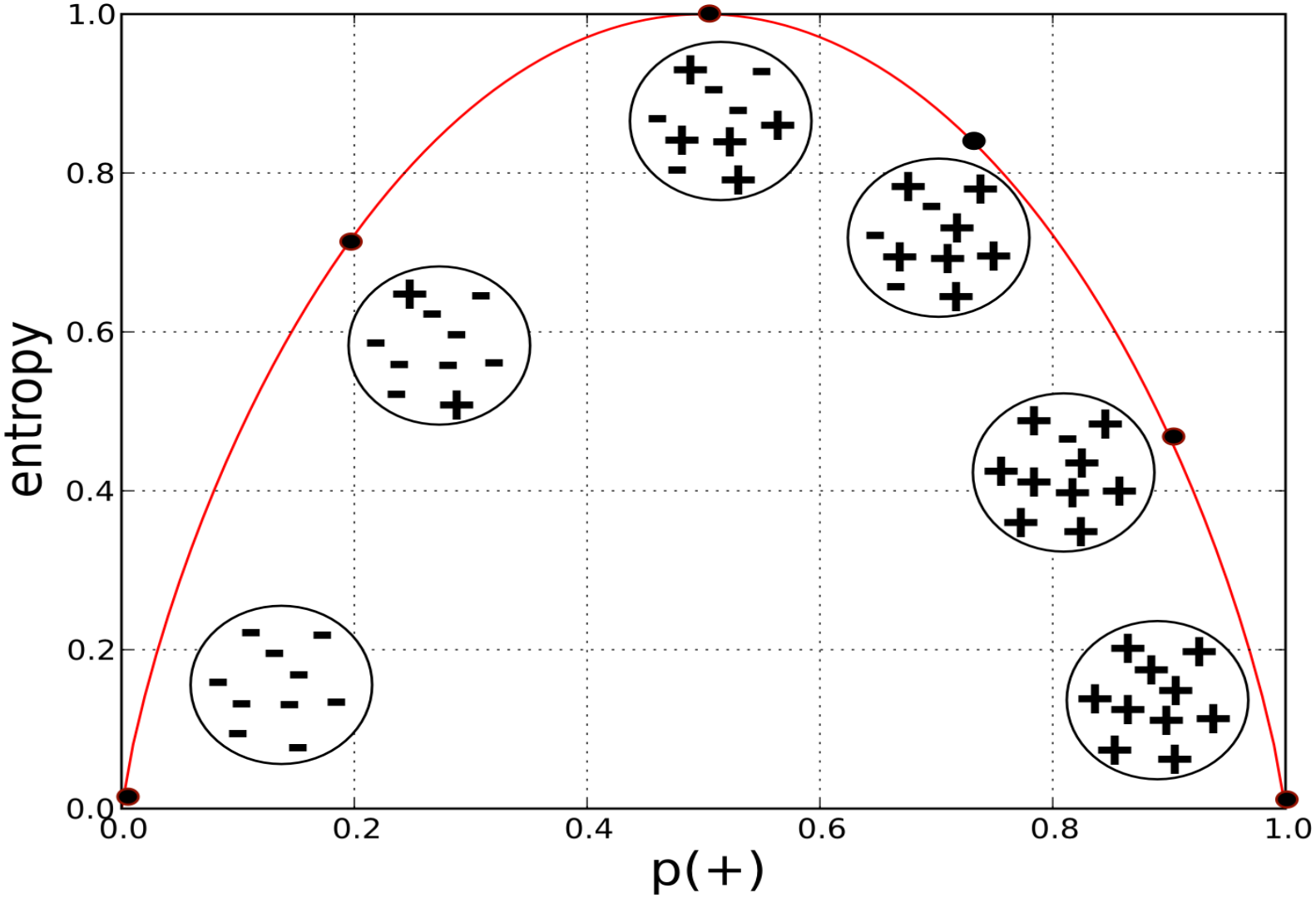
- More technically, entropy is defined as Note: $p_1 + p_2 + \ldots = 1$

$$entropy = -p_1\log(p_1) - p_2\log(p_2) - \ldots$$

- This is based on Gibbs entropy in thermodynamics
- Each p_i is the probability (the relative percentage) of property i (e.g. write-offs/non-write-offs) of the target.

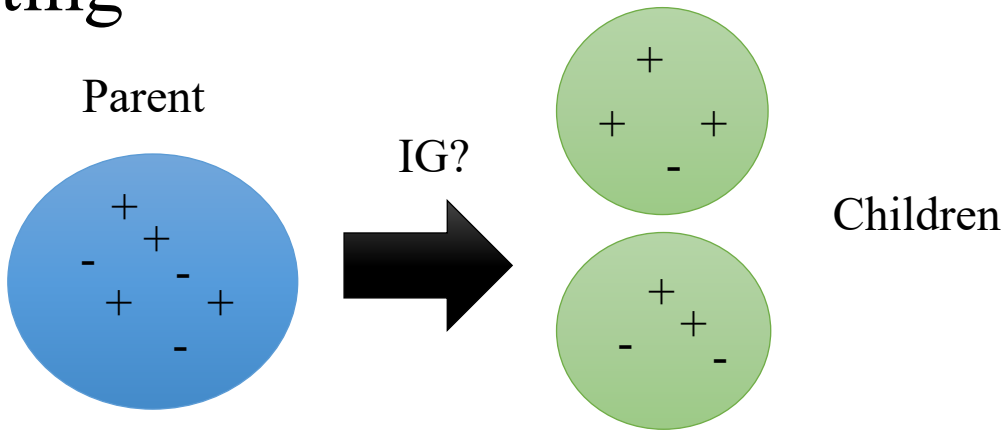


Entropy of a two-class set



Information Gain

- Entropy is only part of the story. We would like to measure how informative an attribute is with respect to target: how much gain in information it gives us about the target?
- Information gain (IG) measure how much attribute improves (decreases) entropy over the whole segmentation it creates.
- In our context, IG measures the change in entropy due to further splitting



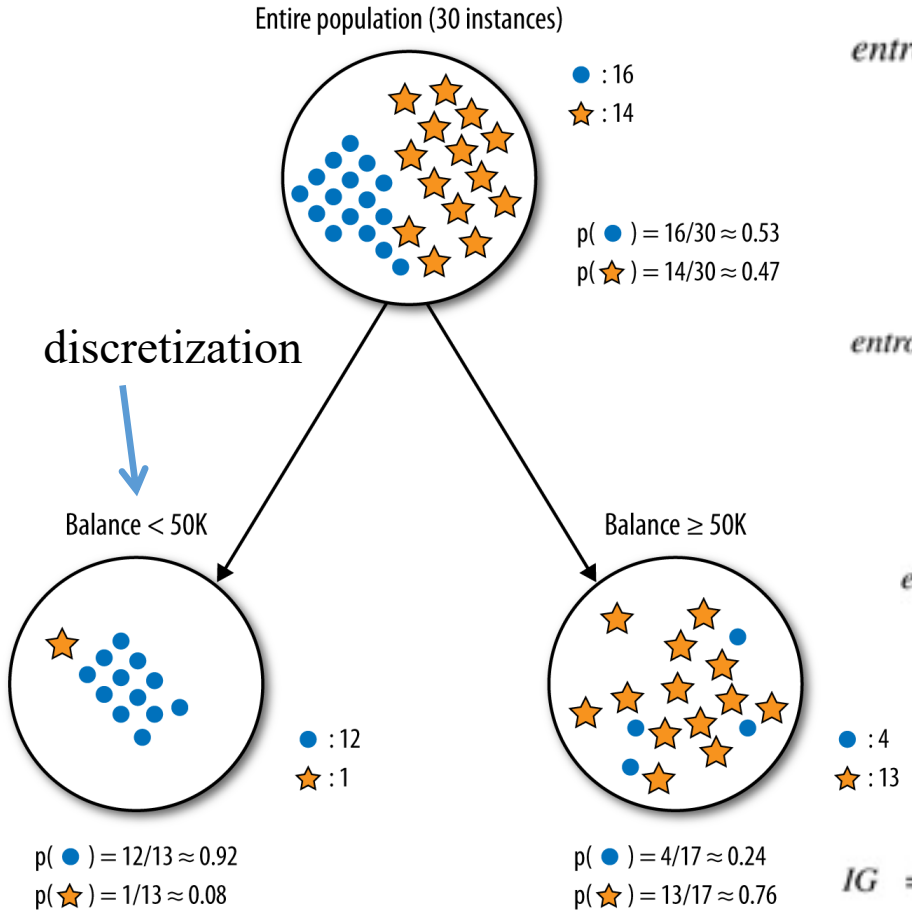
Information Gain Formula

IG(*parent*, *children*)

$$= \text{entropy}(parent) - [p(c_1) \times \text{entropy}(c_1) + p(c_2) \times \text{entropy}(c_2) + \ldots]$$

- The entropy for each child (c_i) is weighted by the proportion of instances belonging to that child, $p(c_i)$.

Example 1



$$\begin{aligned} entropy(parent) &= -[p(\bullet)\times\log_2 p(\bullet)+p(\star)\times\log_2 p(\star)] \\ &\approx -[0.53\times -0.9+0.47\times -1.1] \\ &\approx 0.99 \quad (\text{very impure}) \end{aligned}$$

Entropy of the left child is

$$\begin{aligned} entropy(Balance<50K) &= -[p(\bullet)\times\log_2 p(\bullet)+p(\star)\times\log_2 p(\star)] \\ &\approx -[0.92\times(-0.12)+0.08\times(-3.7)] \\ &\approx 0.39 \end{aligned}$$

Entropy of the right child is

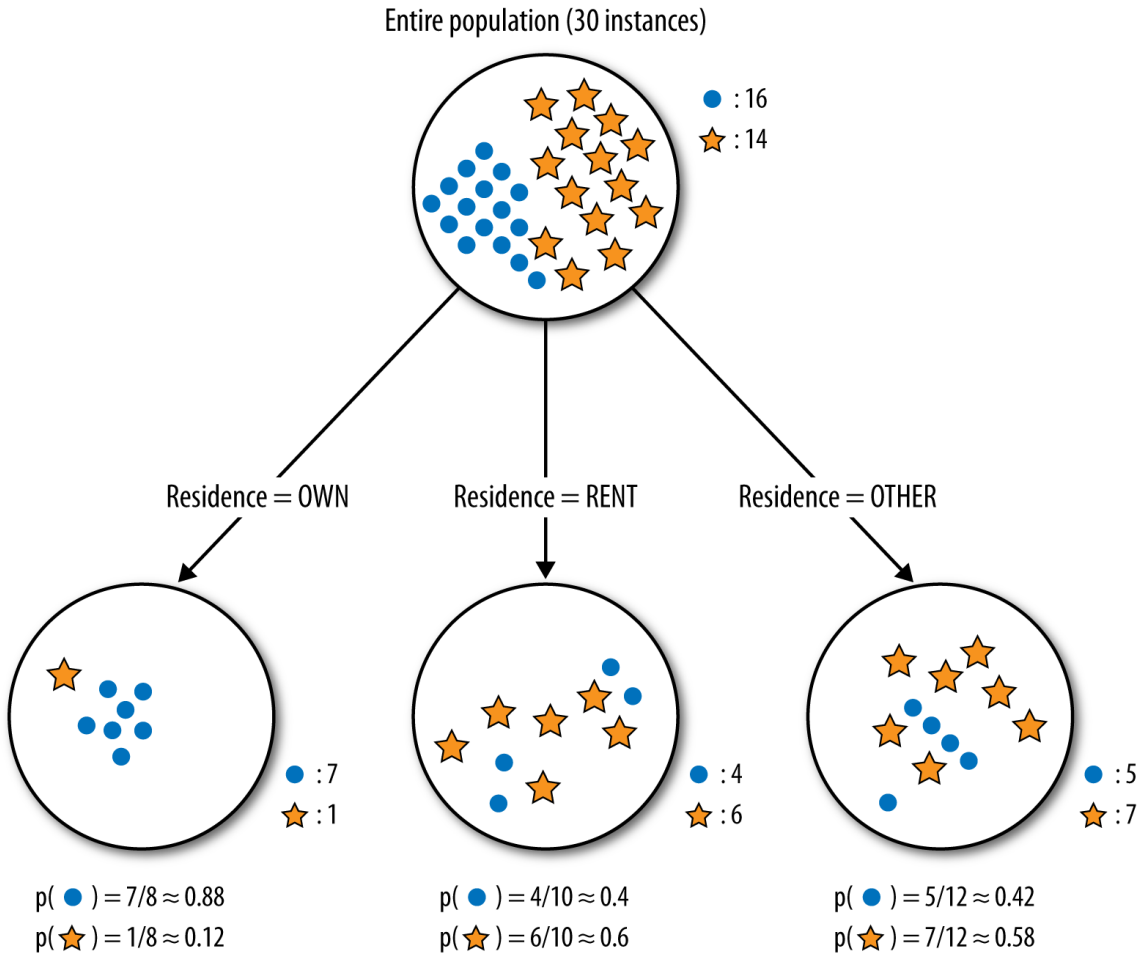
$$\begin{aligned} entropy(Balance\geq 50K) &= -[p(\bullet)\times\log_2 p(\bullet)+p(\star)\times\log_2 p(\star)] \\ &\approx -[0.24\times(-2.1)+0.76\times(-0.39)] \\ &\approx 0.79 \end{aligned}$$

Information gain is

$$\begin{aligned} IG &= entropy(parent)-[p(\text{Balance}<50\text{K})\times entropy(\text{Balance}<50\text{K}) \\ &\qquad +p(\text{Balance}\geq 50\text{K})\times entropy(\text{Balance}\geq 50\text{K})] \\ &\approx 0.99-[0.43\times 0.39+0.57\times 0.79] \\ &\approx 0.37 \end{aligned}$$

13/30

Example 2



Calculations are omitted

entropy(*parent*) ≈ 0.99

entropy(Residence=OWN) ≈ 0.54

entropy(Residence=RENT) ≈ 0.97

entropy(Residence=OTHER) ≈ 0.98

IG ≈ 0.13

Residence variable is less informative than Balance.

Balance(prev. page) ให้ข้อมูลได้ดีกว่า Residence

IG(Balance) > IG(Residence)

Splitting criteria

Regression: residual sum of squares

$$RSS = \sum_{left} (y_i - y_L^*)^2 + \sum_{right} (y_i - y_R^*)^2$$

where y_L^* = mean y-value for left node

y_R^* = mean y-value for right node

→ ให้ผลเหมือน entropy

Classification: Gini criterion (Similar to Information Gain)

$$Gini = N_L \sum_{k=1,\ldots,K} p_{kL} (1 - p_{kL}) + N_R \sum_{k=1,\ldots,K} p_{kR} (1 - p_{kR})$$

where p_{kL} = proportion of class k in left node

p_{kR} = proportion of class k in right node

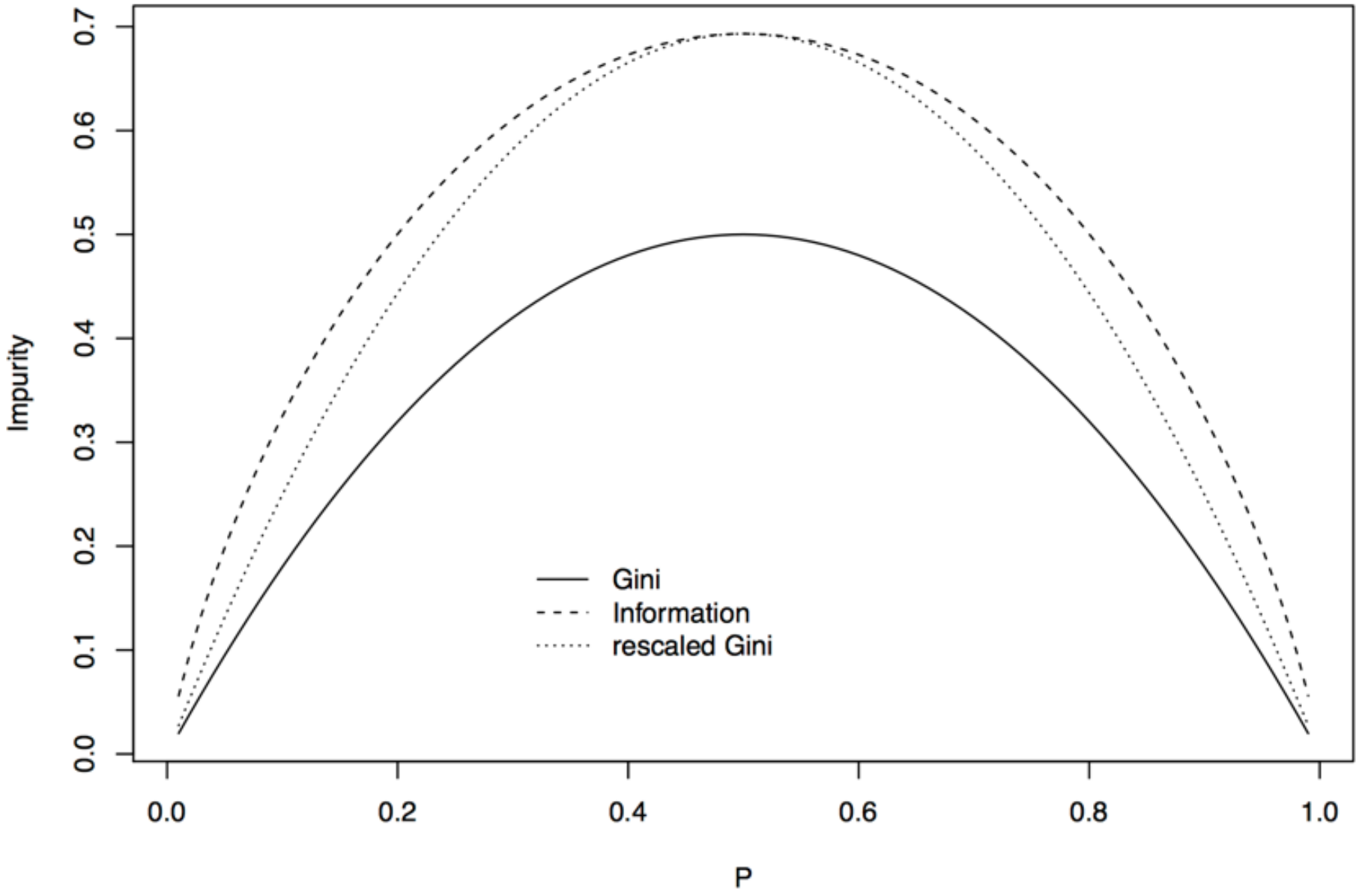
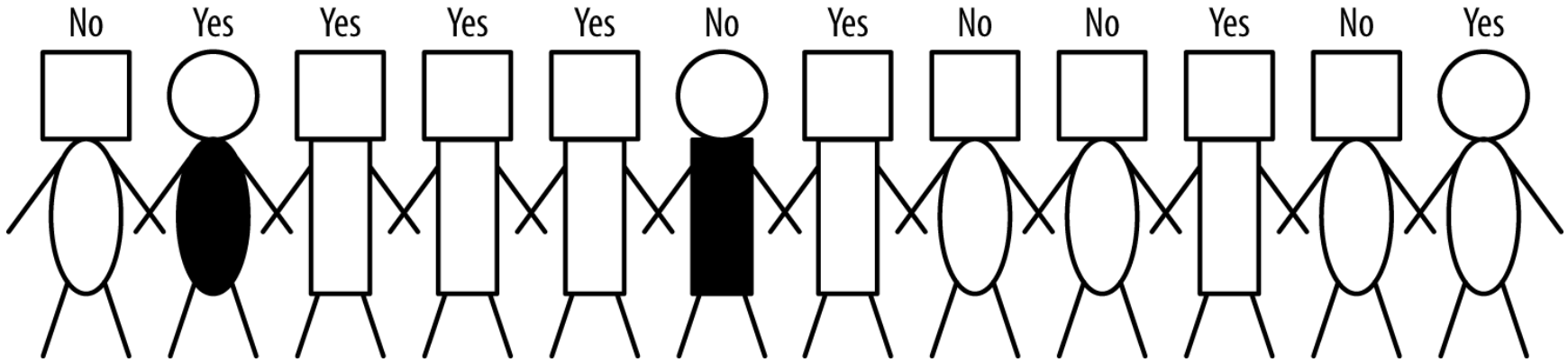


Figure 2: Comparison of Gini and Information impurity for two groups.

Example

No : Yes
5 : 7



Attributes

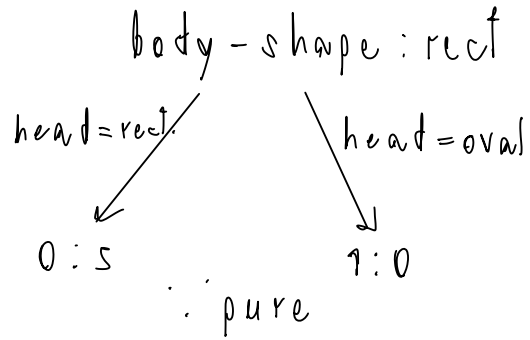
head-shape: square, circular (4:5, 1:2)

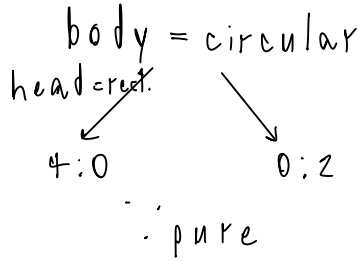
body-shape: rectangular, oval (1:5, 4:2)

body-color: gray, white (1:1, 4:6)

Target variable

write-off: Yes, No



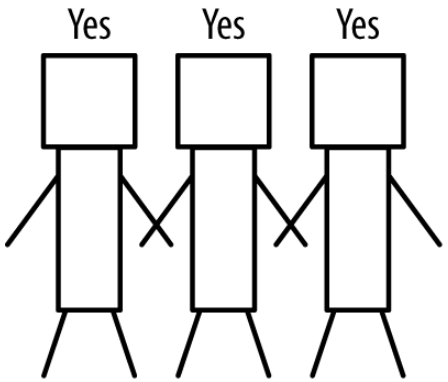


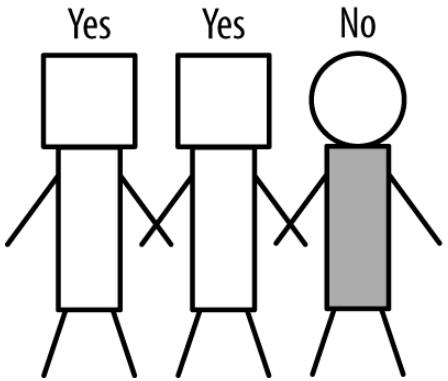
∴ ใช้แค่ body, head shape ก็เพียงพอ

First Partitioning: body-shape

No : Yes
1 : 5

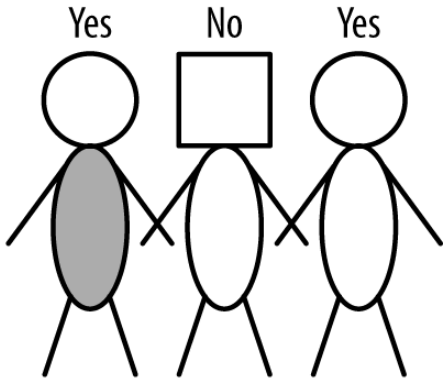
Rectangular Bodies

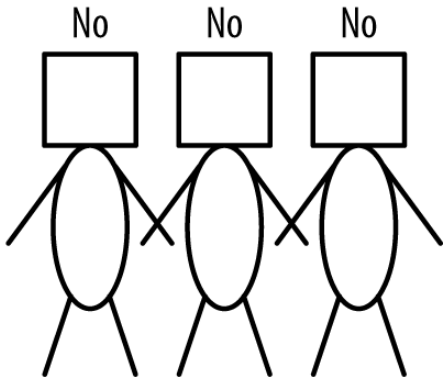




4 : 2

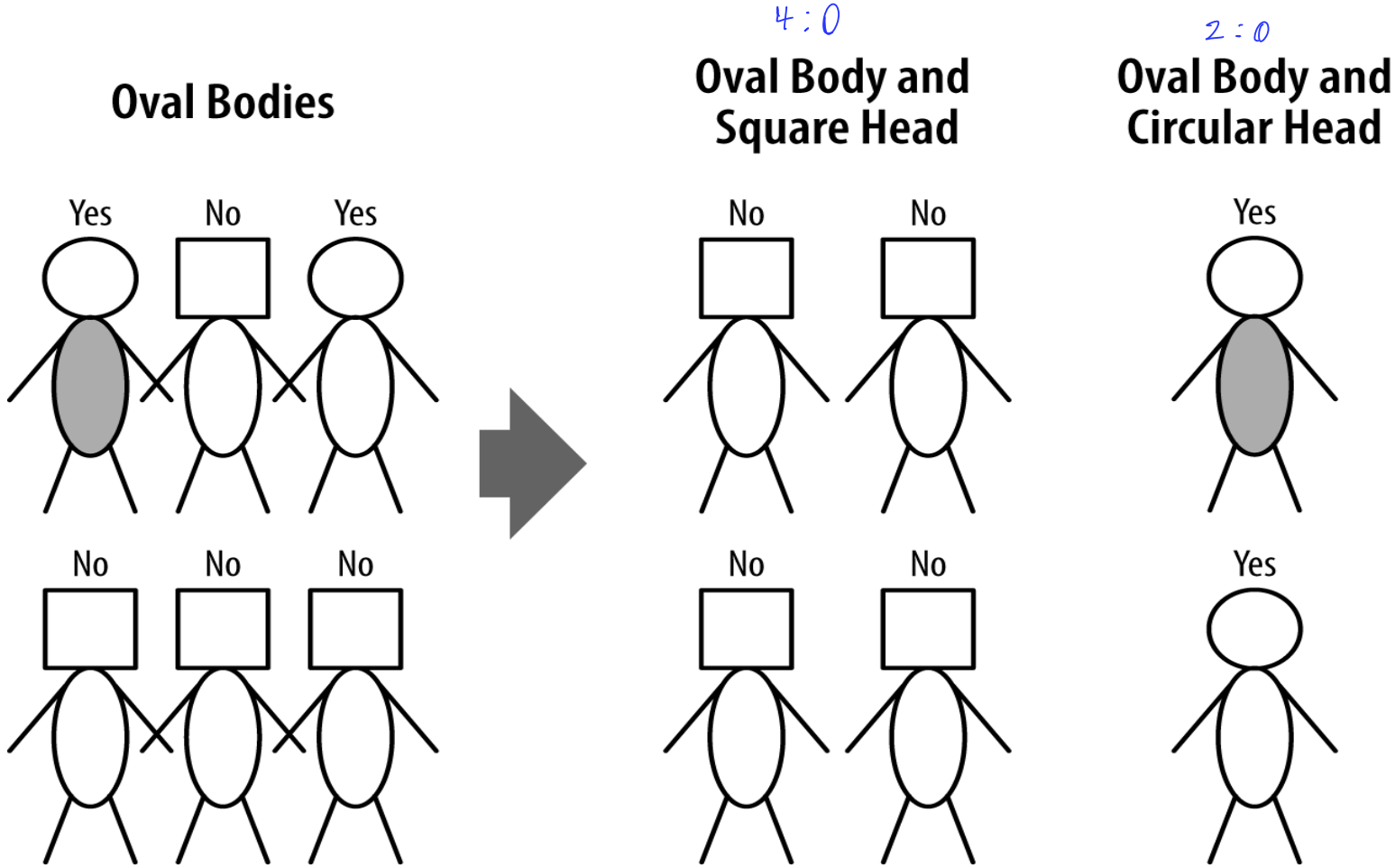
Oval Bodies



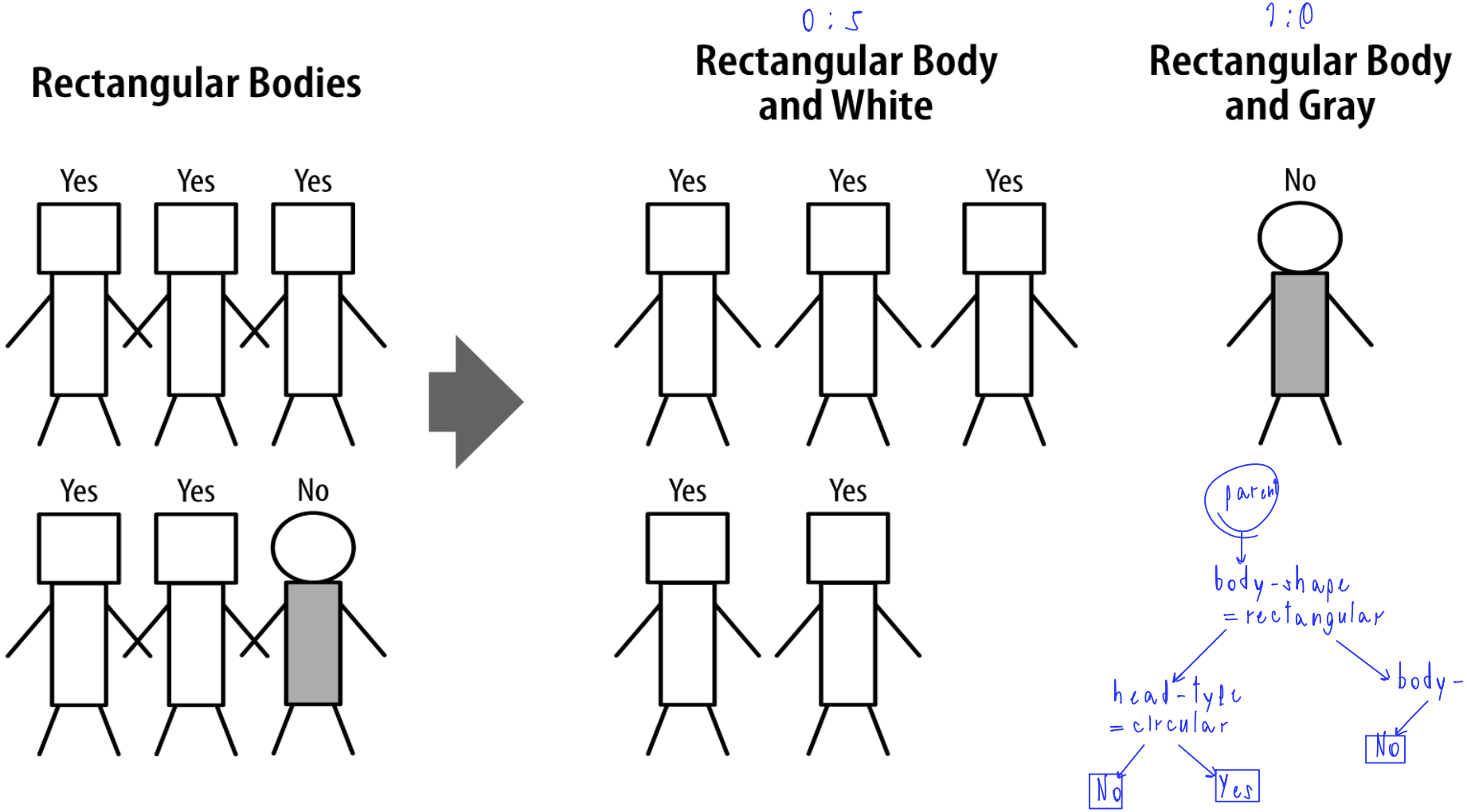


body-shape has the highest IG, so it is selected as the first attribute

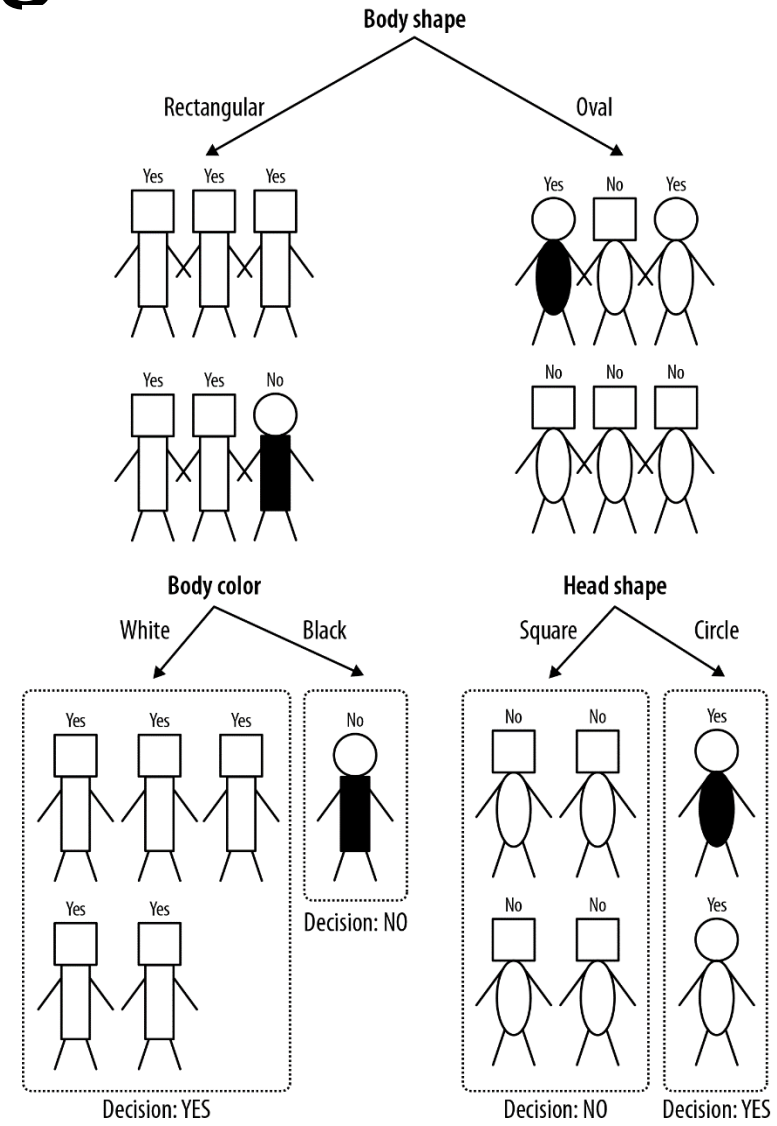
2nd partitioning: oval-body, head-type

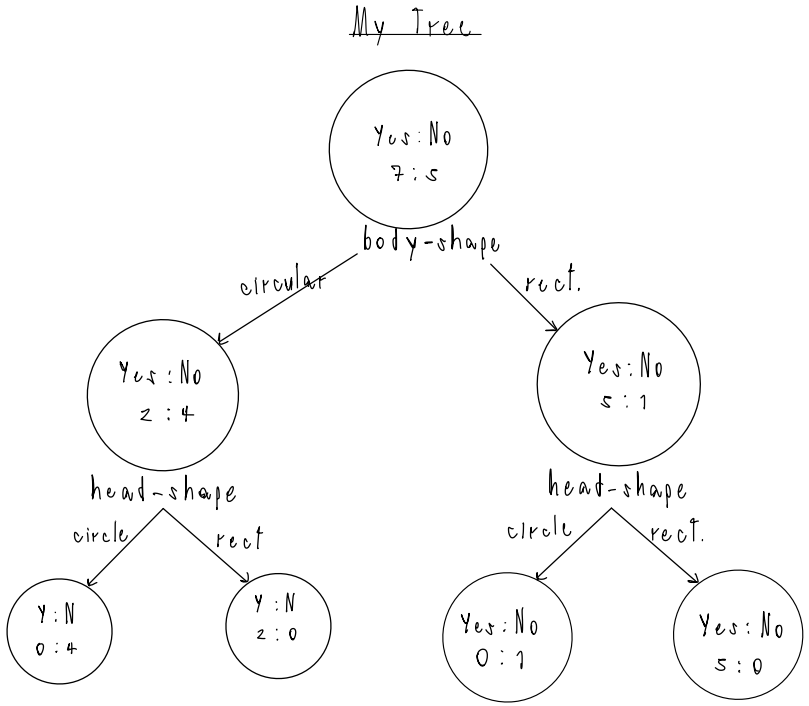


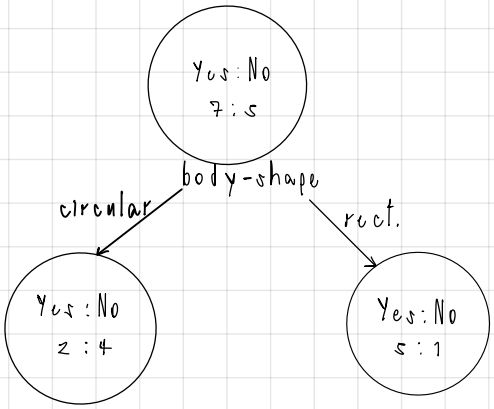
3rd partitioning: rectangular-body, body-color



Resulting Decision Tree







$$entropy(parent) = -\sum p_i \log(p_i)$$

$$= -\left[\frac{7}{12}\log_2\left(\frac{7}{12}\right) + \frac{5}{12}\log_2\left(\frac{5}{12}\right)\right]$$

$$\approx 0.98$$

body shape

$$entropy(body = circular) = -\left[\frac{2}{6}\log_2\left(\frac{2}{6}\right) + \frac{4}{6}\log_2\left(\frac{4}{6}\right)\right]$$

$$\approx 0.92$$

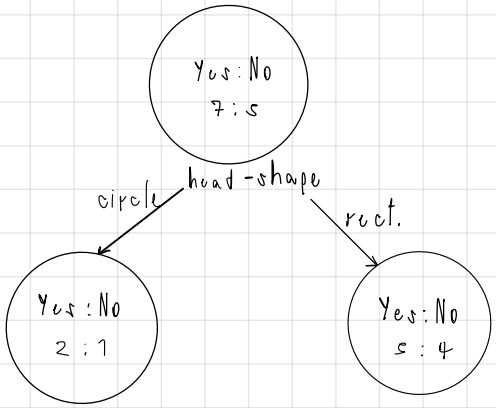
$$entropy(body = rect.) = -\left[\frac{5}{6}\log_2\left(\frac{5}{6}\right) + \frac{1}{6}\log_2\left(\frac{1}{6}\right)\right]$$

$$\approx 0.65$$

$$IG(body\ shape) = entropy(parent) - [p(cir.)\cdot entropy(cir.) + p(rec.)\cdot entropy(rec.)]$$

$$= 0.98 - \left[\frac{6}{12}(0.92) + \frac{6}{12}(0.65)\right]$$

$$= 0.135 \quad \text{—} \circledast$$



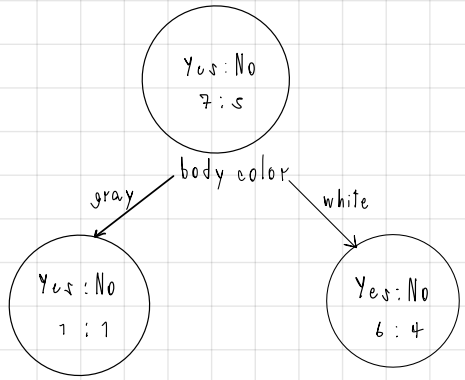
head shape

$$entropy(rect.) = -\left[\frac{5}{9}\log_2\left(\frac{5}{9}\right) + \frac{4}{9}\log_2\left(\frac{4}{9}\right)\right] \approx 0.99$$

$$entropy(cir.) = -\left[\frac{2}{3}\log_2\left(\frac{2}{3}\right) + \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right] \approx 0.92$$

$$IG = 0.98 - \left[\frac{9}{12}(0.99) + \frac{3}{12}(0.92)\right]$$

$$= 0.0075 \quad \text{—} \circledast$$



body color

$$entropy(gray) = -\left[\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right]$$

$= 1 \longrightarrow$ ครึ่ง : ครึ่ง ∵ maximum entropy

$$entropy(white) = -[0.6\log_2(0.6) + 0.4\log_2(0.4)]$$

$$= 0.97$$

$$IG(color) = 0.98 - \left[\frac{2}{12}(1) + \frac{10}{12}(0.97)\right]$$

$$= 0.005 \quad \text{—} \circledast$$

∴ ใช้ body-shape เป็น condition เนื่องจาก ให้ข้อมูลได้มากที่สุด (IG เยอะสุด)

Node ต่อๆไปก็ทำเช่นนี้

Accuracy, Precision and Recall

	Actual Positive (p)	Actual Negative (n)
The model says "Yes" = positive (y)	True positives	False positives
The model says "No" = not positive (n)	False negatives	True negatives

model ทาย ถูก/ผิด

ผลที่ model ทาย

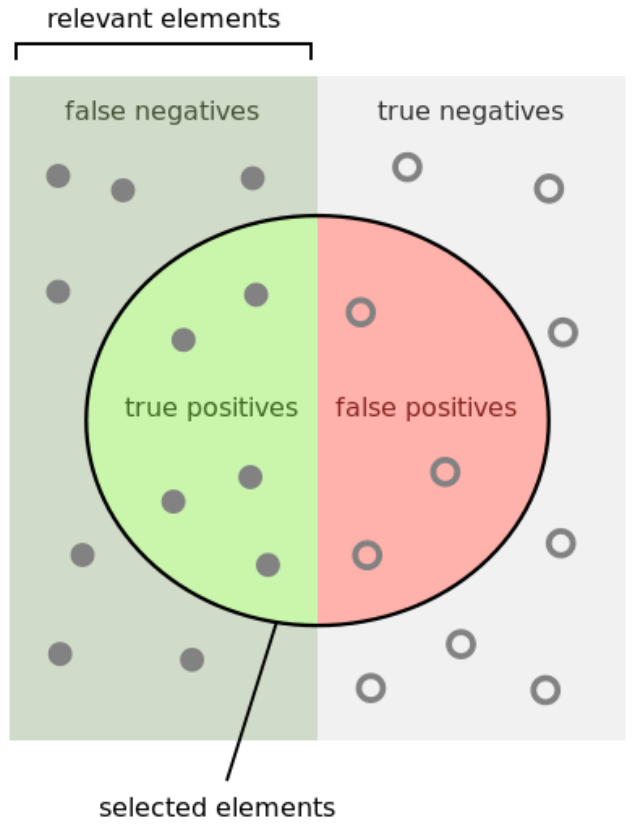
- Accuracy = (TP + TN)/(TP + FP + TN + FN)
- Recall (Completeness) = true positive rate = TP/(TP + FN)
- Precision (Exactness) = the accuracy over the cases predicted to be positive, TP/(TP + FP)
- F-measure = the harmonic mean of precision and recall

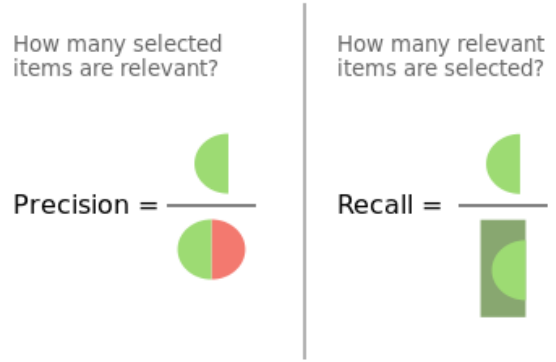
  = the balance between recall and precision

  $= 2 \cdot \frac{precision * recall}{precision + recall}$

โฟกัสที่ actual data

โฟกัสที่ model





Receiver operating characteristics Area under the ROC curve

True Positive Rate (**TPR**) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (**FPR**) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

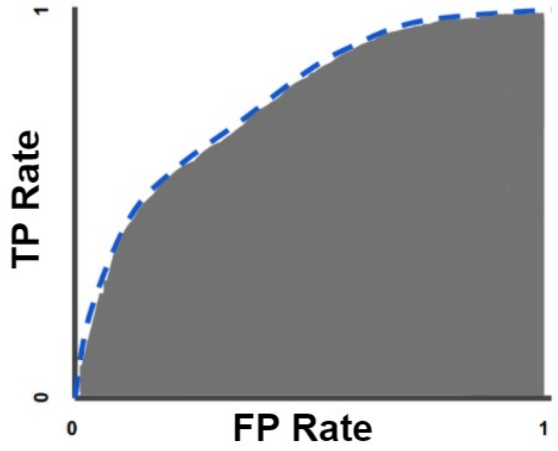


Figure 5. AUC (Area under the ROC Curve).

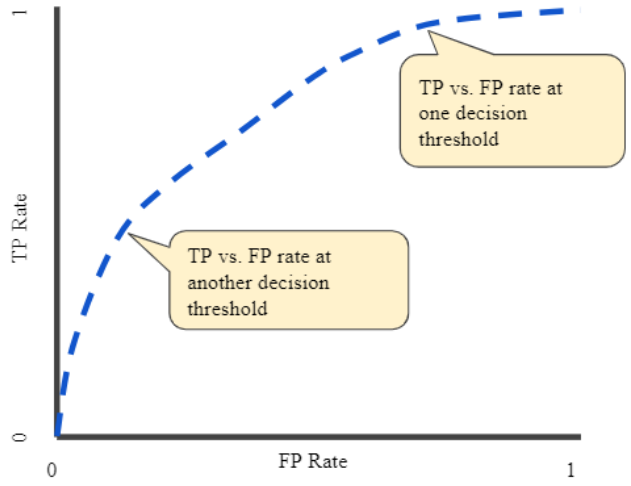
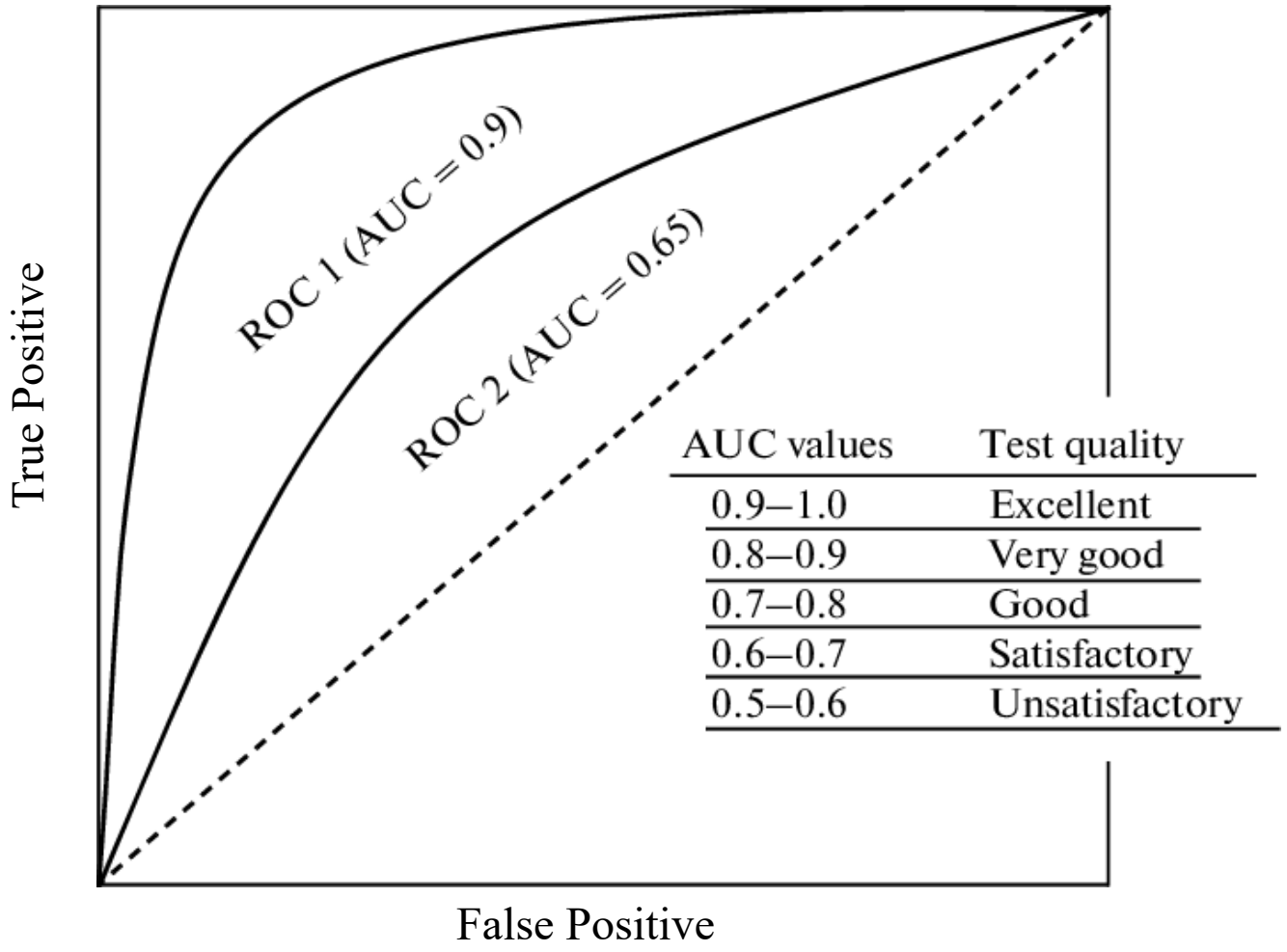


Figure 4. TP vs. FP rate at different classification thresholds.

AUC - ROC



AUC values	Test quality
0.9–1.0	Excellent
0.8–0.9	Very good
0.7–0.8	Good
0.6–0.7	Satisfactory
0.5–0.6	Unsatisfactory

Ensemble Models

https://drive.google.com/file/d/1sp2quqCmxkzy2ksoHCqWgNdJImAjRrsx/view?usp=sharing

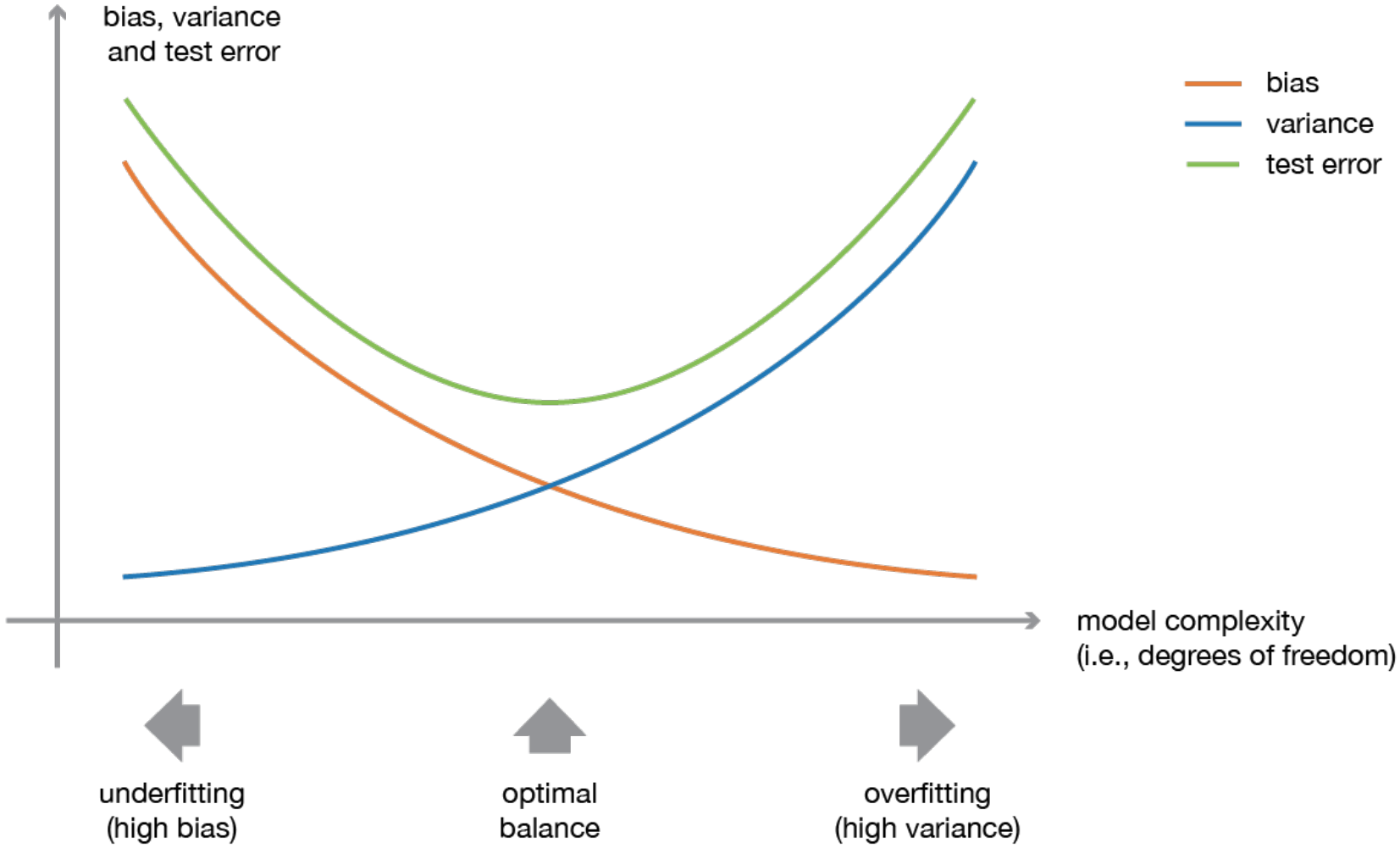
Classification and Regression Trees Disadvantages

- *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

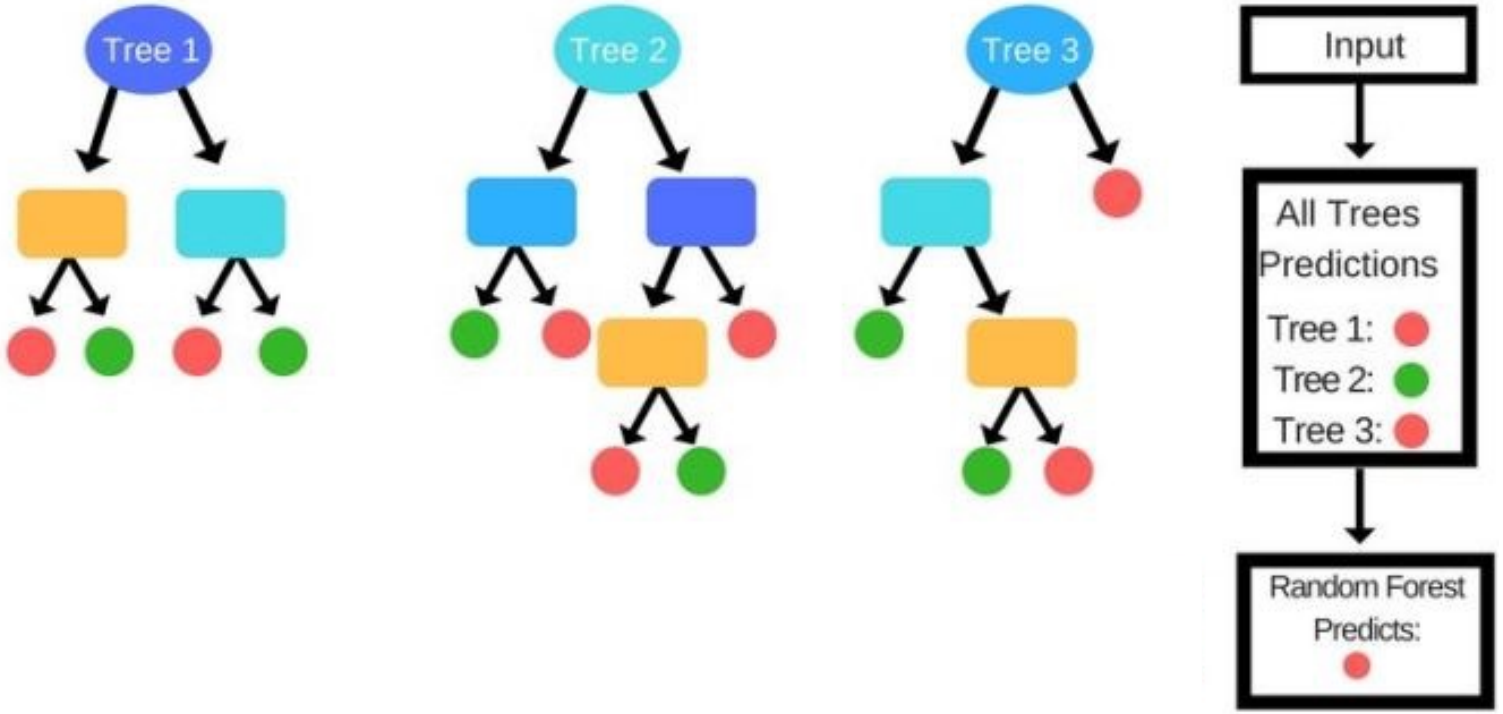
Bias variance tradeoff



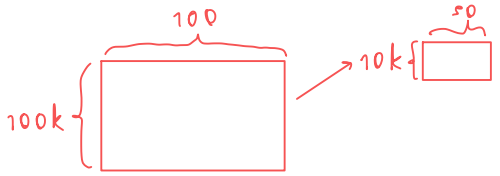
Random Forest

for regression and classification

Random Forest



Random forest



main idea: ให้ tree แต่ละตัวเห็นภาพแค่บางส่วน

หยิบแบบใส่คืน

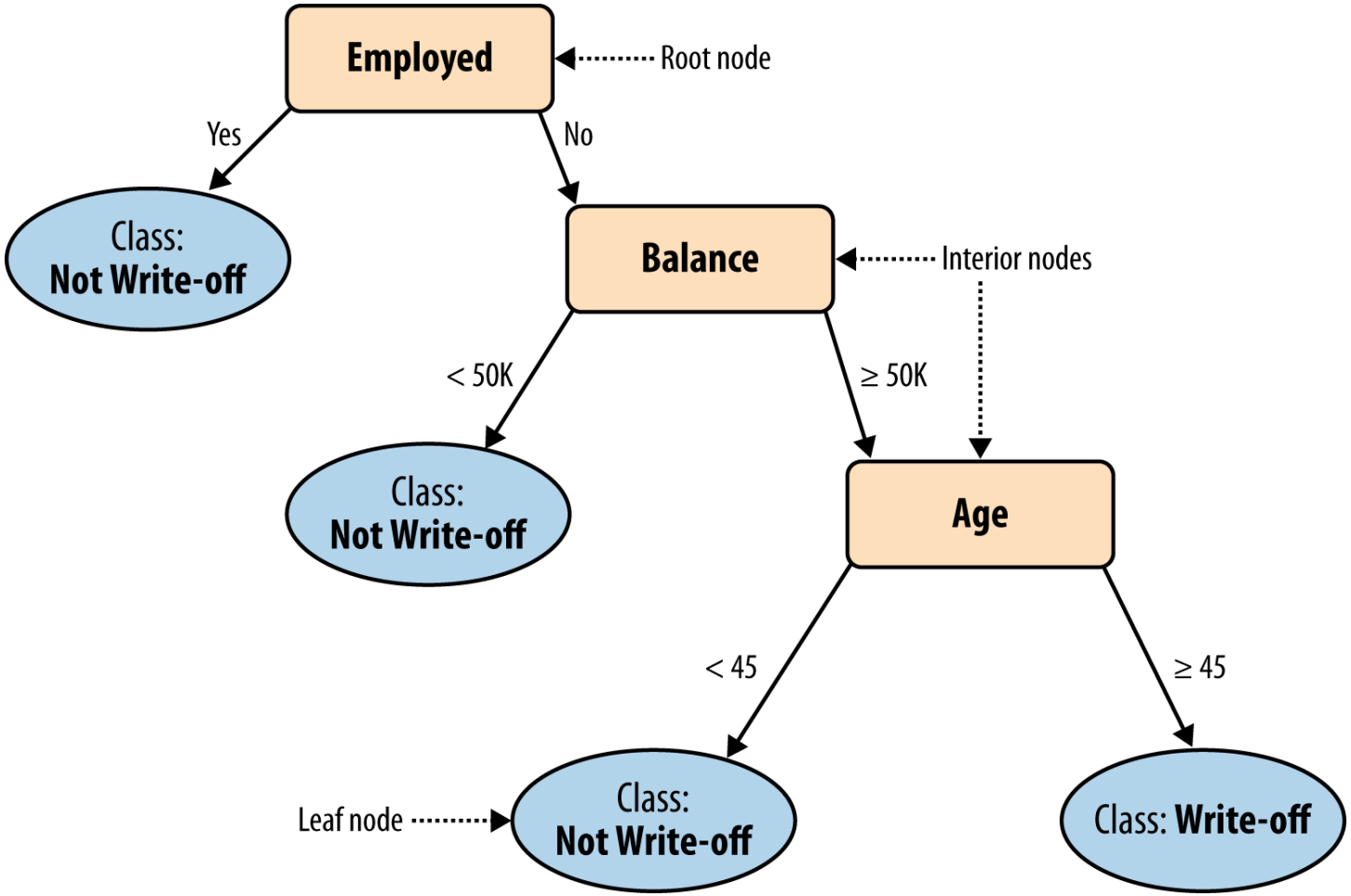
- An ensemble of decision tree trained by bootstrap sampling and random feature selection
- It is based on decision trees: classifiers constructed greedily using the conditional entropy
- The extension hinges on two ideas:
  - building an ensemble of trees by training on subsets of data
  - considering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

Classification and Regression Trees Pioneers

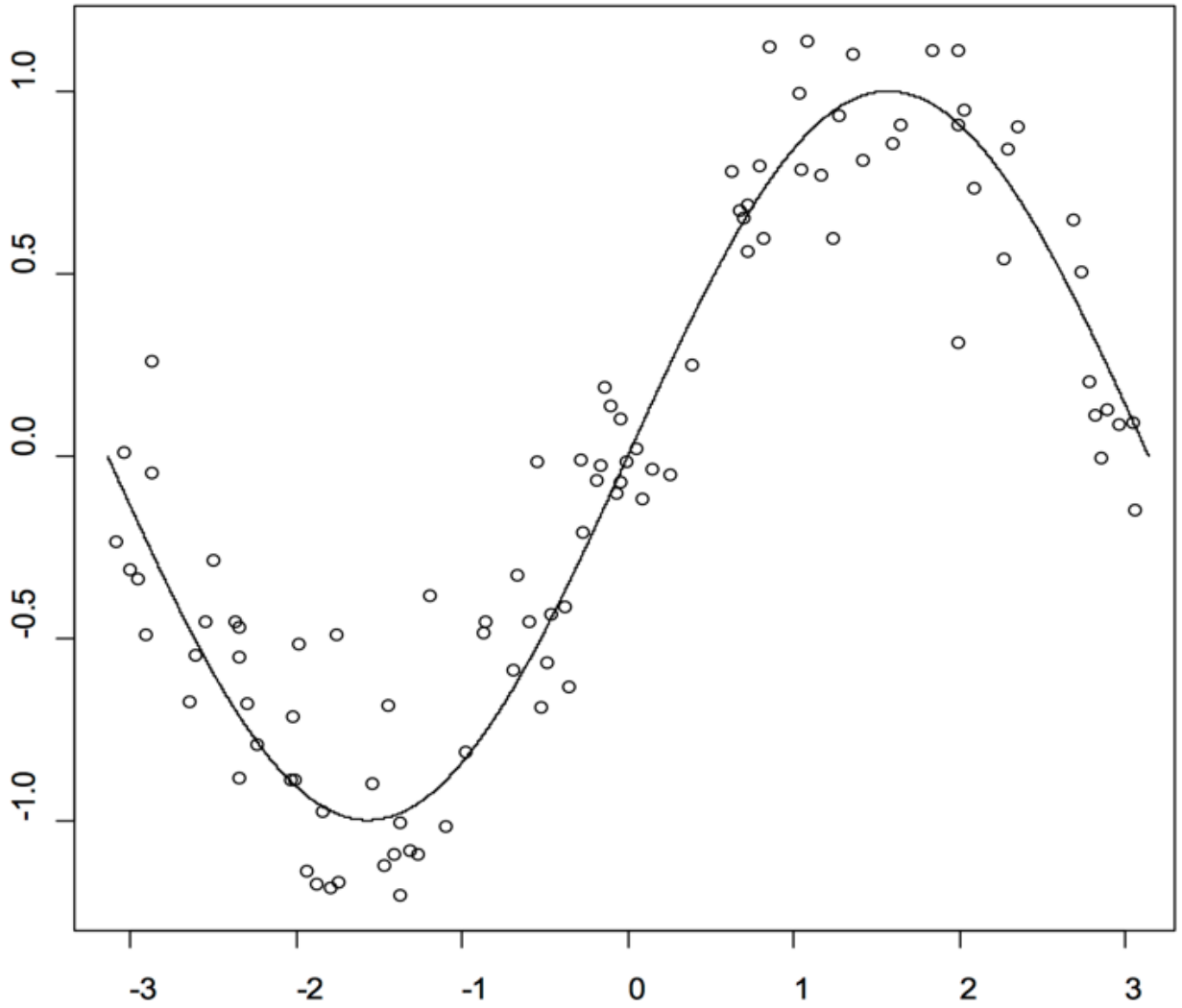
Pioneers:

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

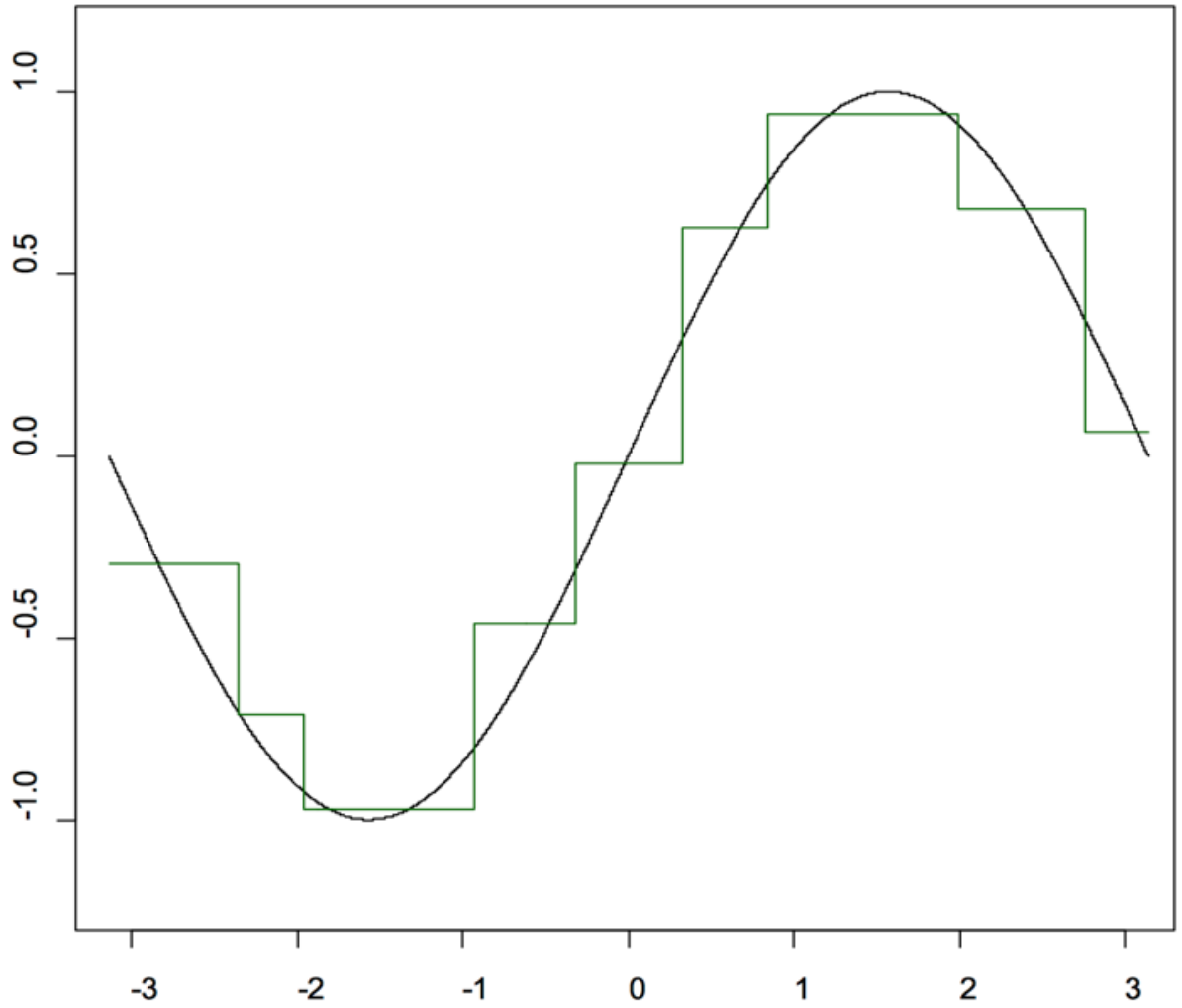
A Classification Tree



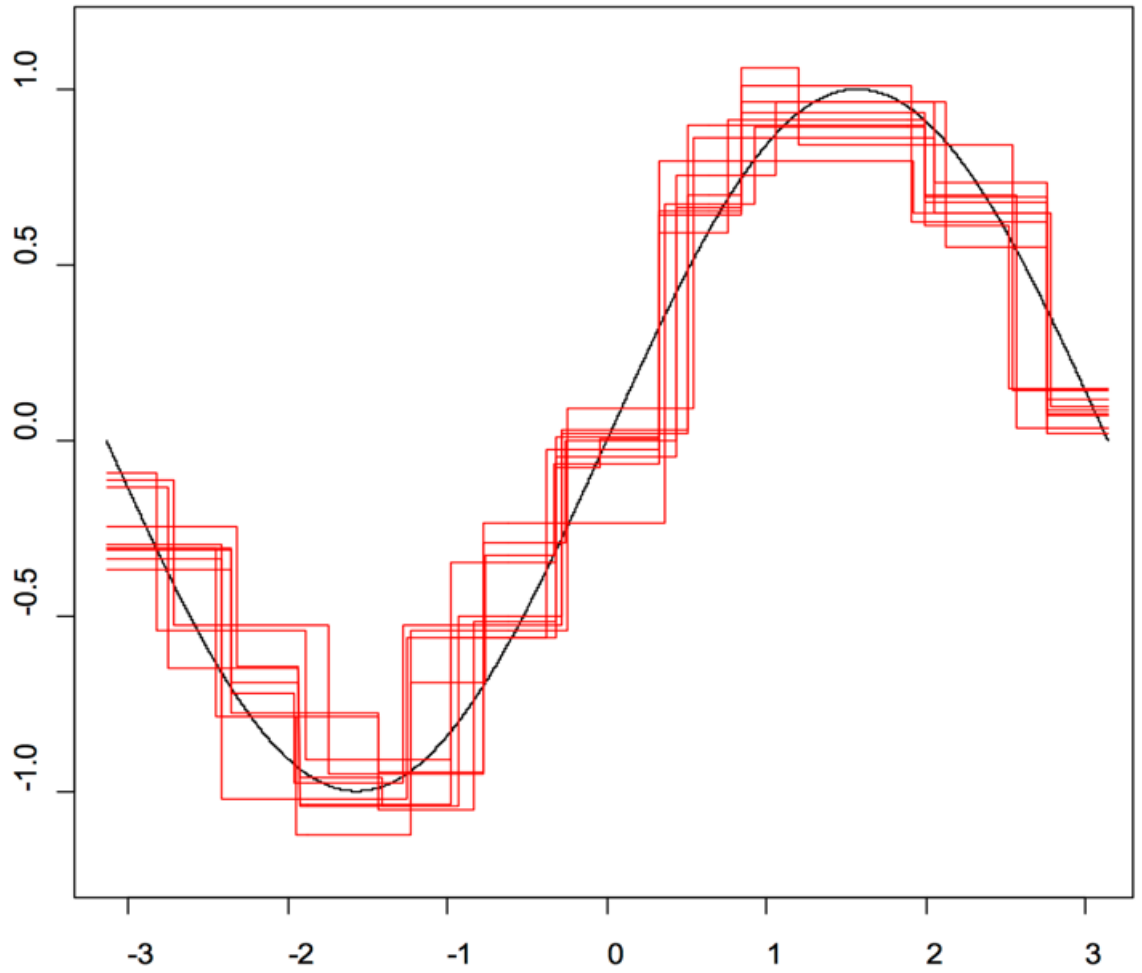
Data and Underlying Function



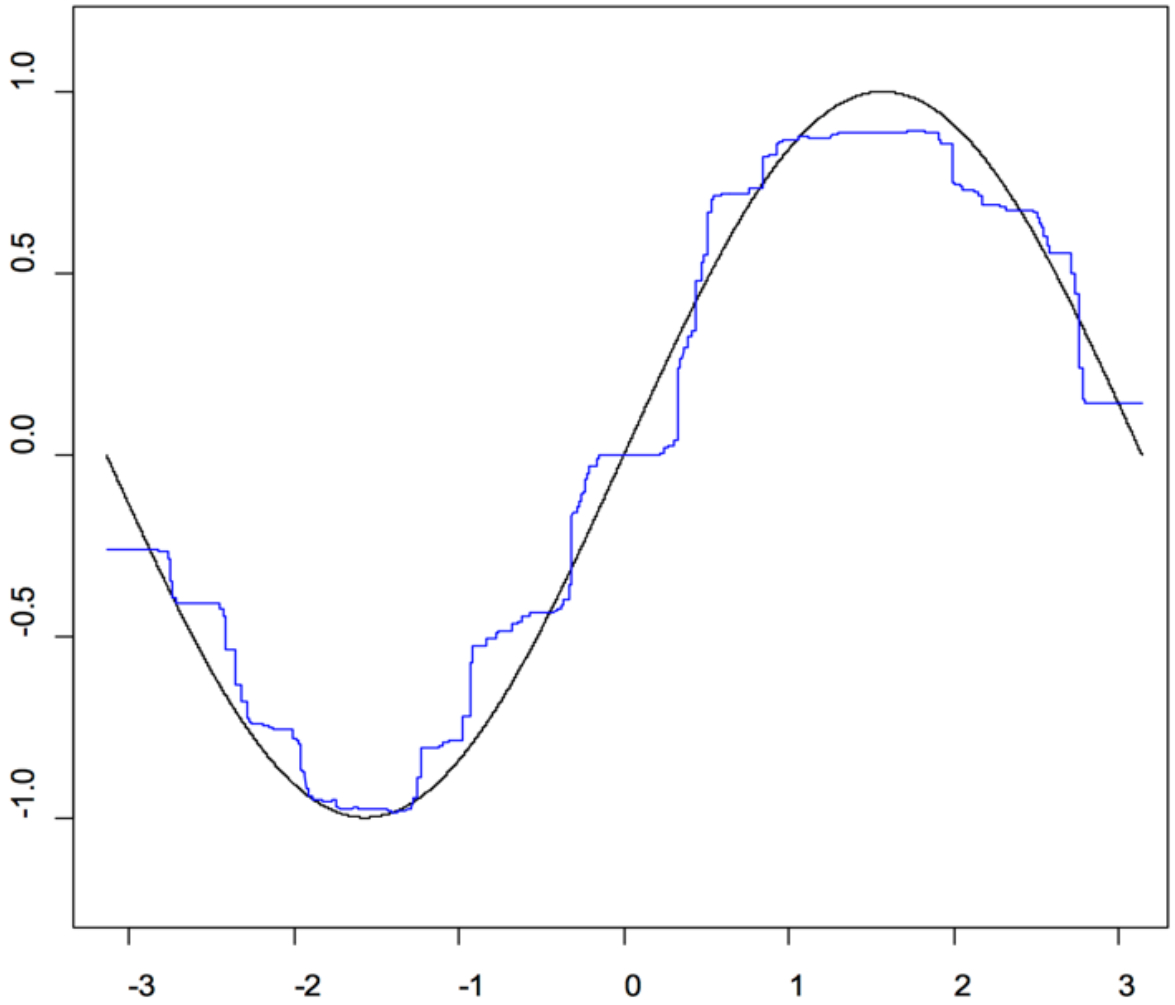
Single Regression Trees



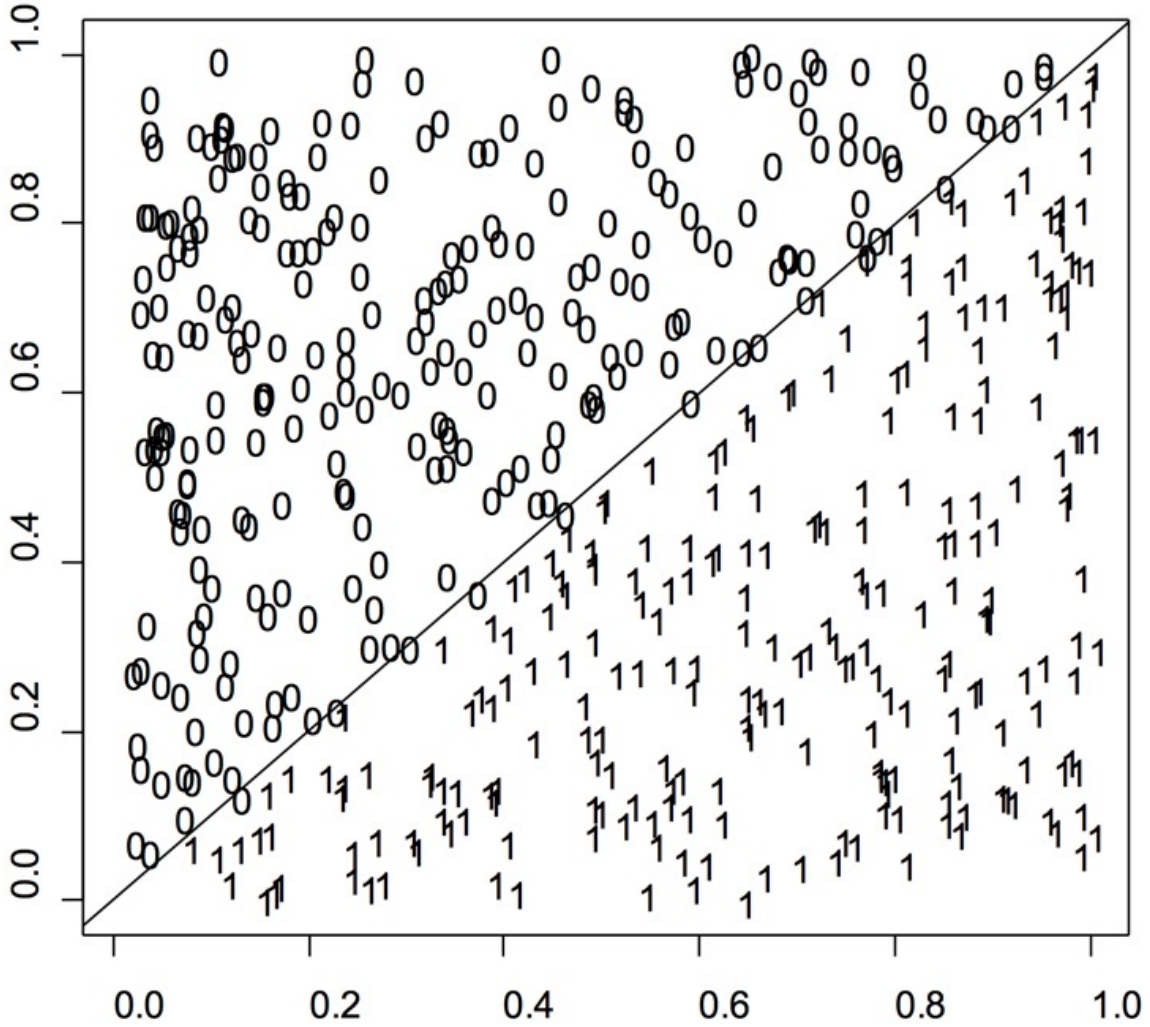
10 Regression Trees



Average of 100 Regression Trees

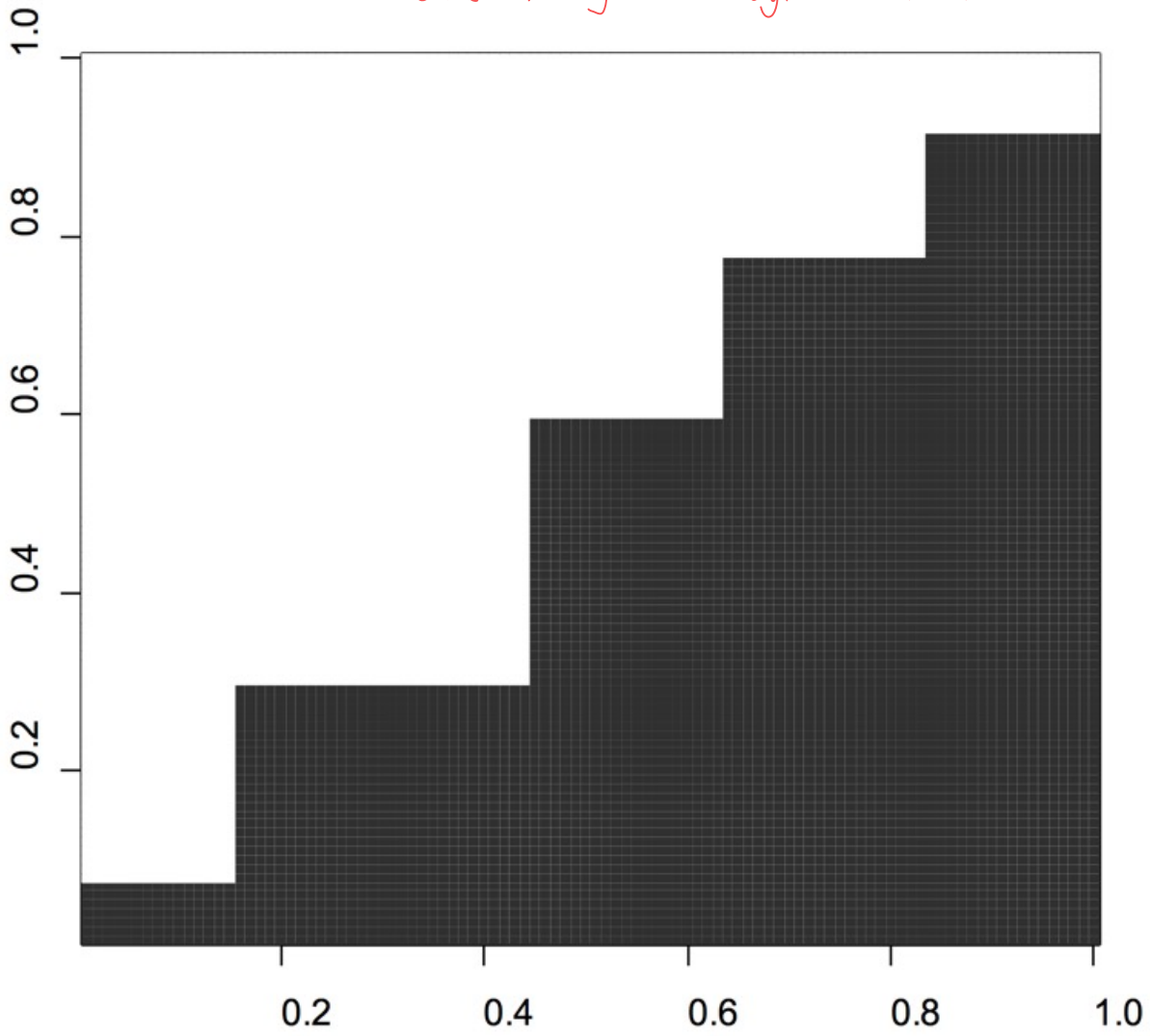


Hard problem for a single tree

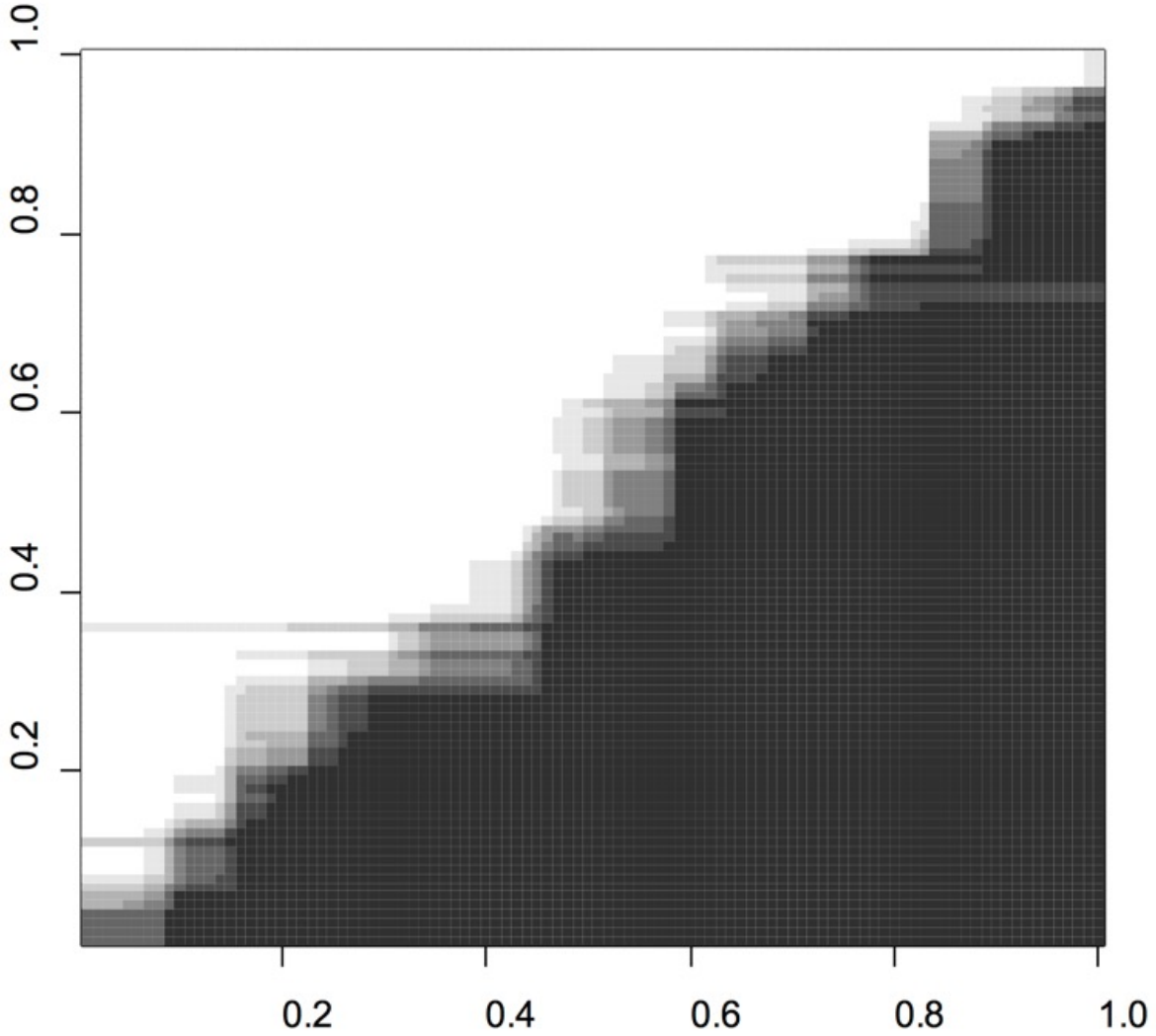


Single Tree

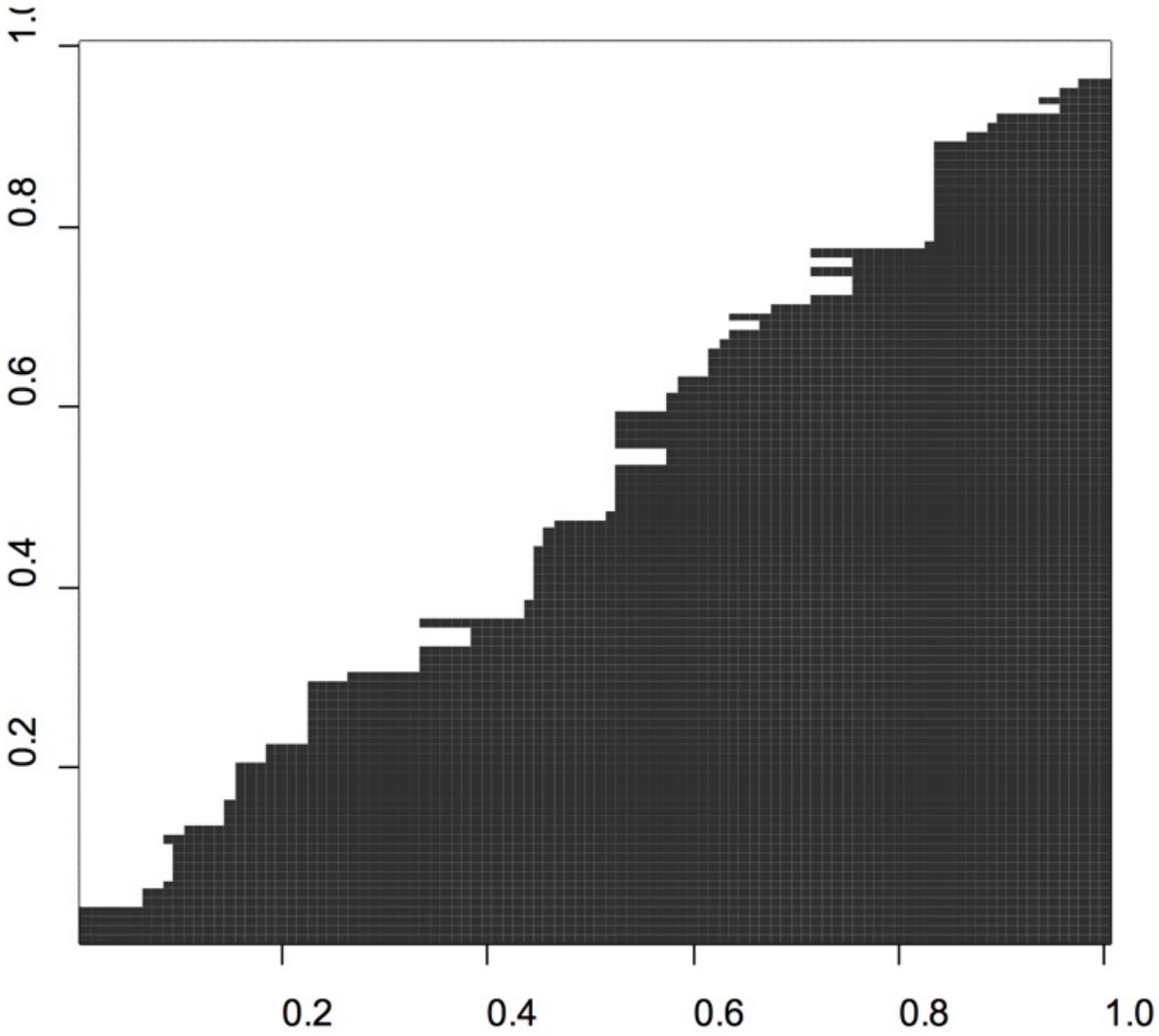
∴ error กระจาย เพราะใช้หลักการในการแบ่งกลุ่ม
case นี้ logistic regression น่าจะเหมาะกว่า เพราะ data ดูเป็น linear separable



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)

Bootstrap sample → $f_1(x)$
Bootstrap sample → $f_2(x)$
Bootstrap sample → $f_3(x)$
…
Bootstrap sample → $f_M(x)$

Combine $f_1(x), \ldots, f_M(x)$ → $f(x)$

$f_i(x)$'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.

model ที่ overfit ง่ายๆ (high variance)

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)
```

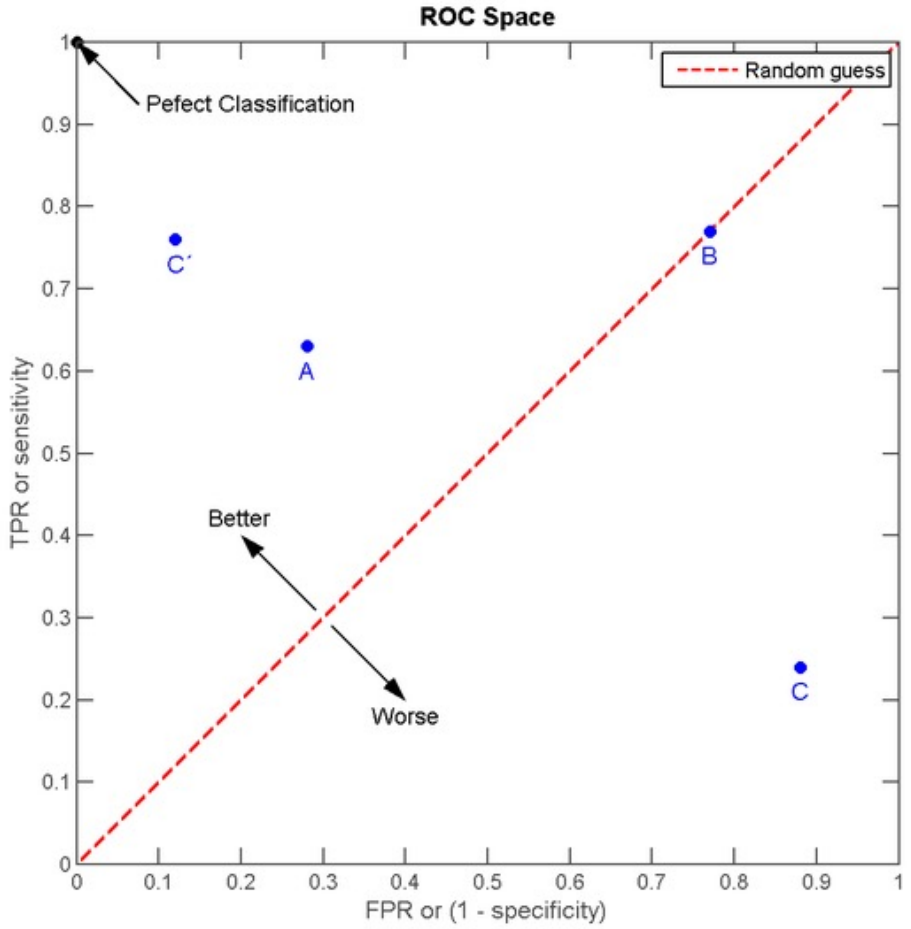
```
RandomForestClassifier(class_weight='balanced', max_depth=8,
                       min_samples_leaf=20)
```

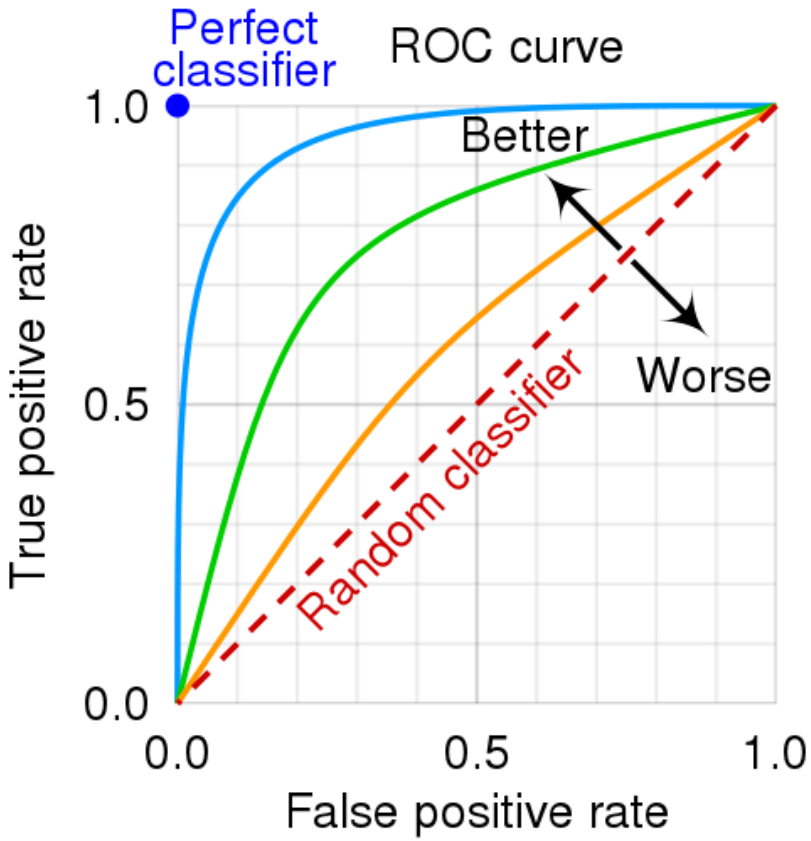
Random Forest – Feature Importance

```
[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



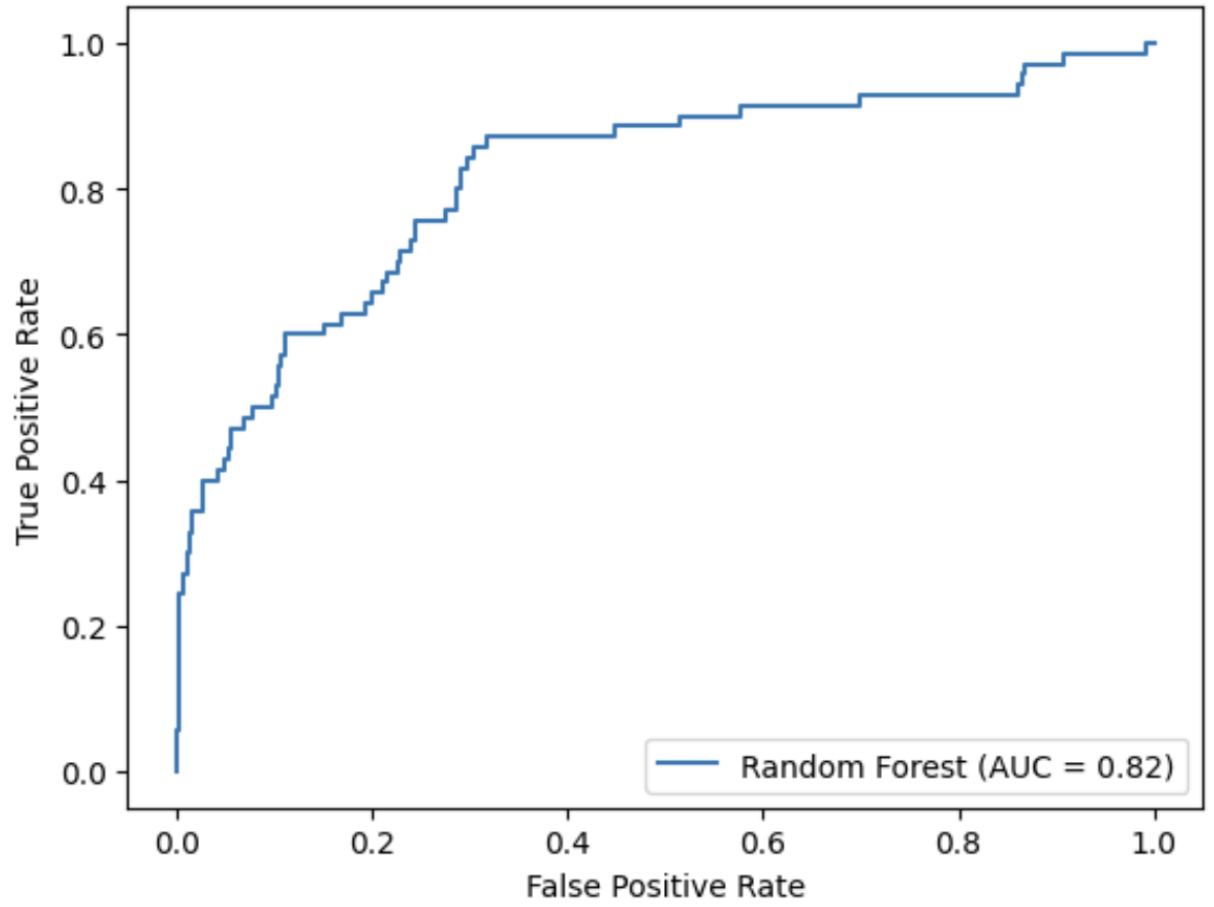


Evaluation

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.

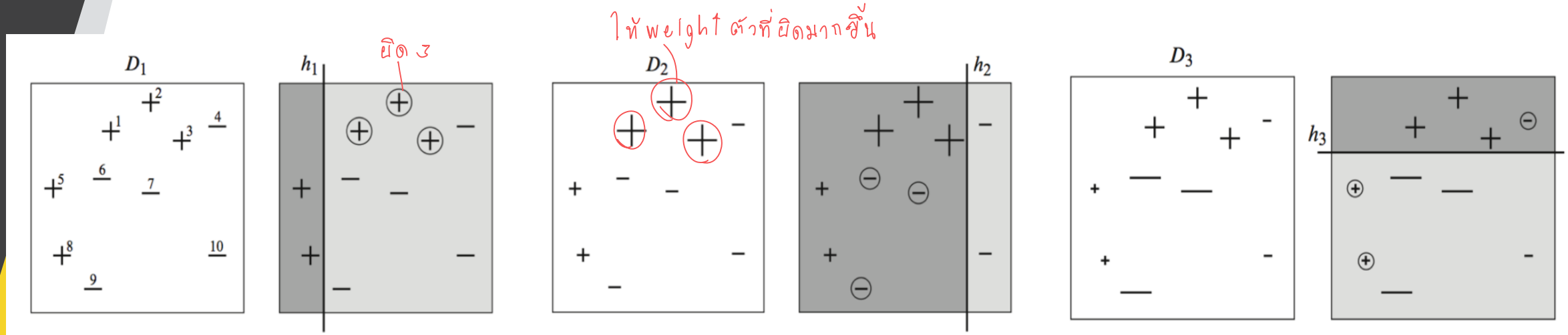
- Gradient Boosting performs well when you have unbalanced data such as in real time risk assessment.

sample ในแต่ละ class ไม่เท่ากัน

Boosting

AdaBoost (Schapire and Freund, 2012)

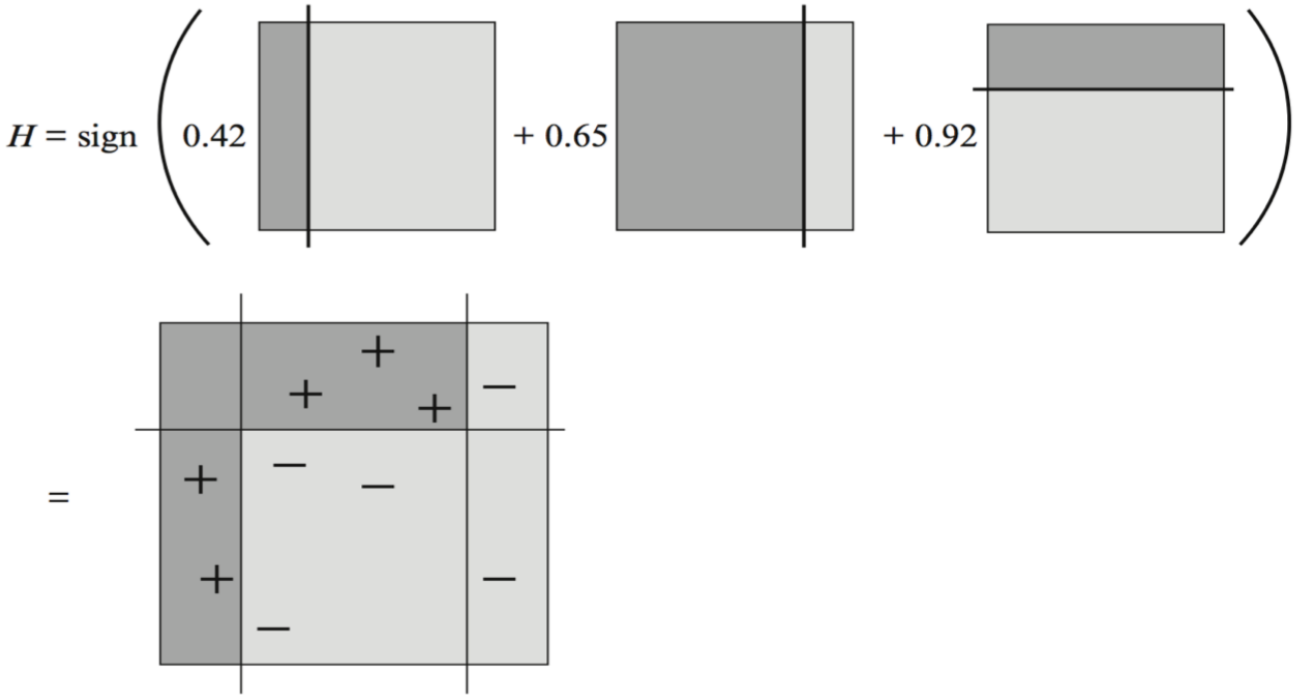
- Fit additive model $\sum_t c_t h_t(x)$ (weight, ตัวแบ่ง)
- In each stage, introduce a **weak learner** to compensate the shortcoming of existing weak learner



AdaBoost

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

Idea: เอา weak learner มาเรียน
แบบ serial ให้เรียนรู้ตัวที่ผิด



Gradient Boosting

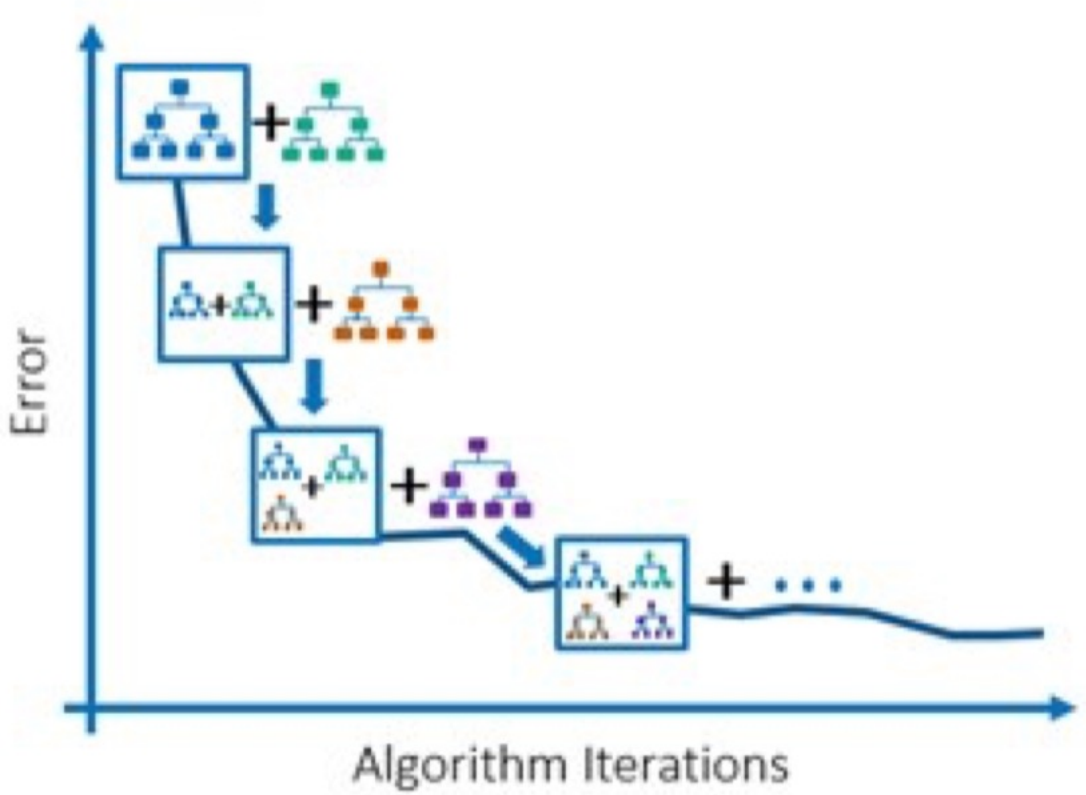
- Gradient boosting = gradient descent + boosting
- Fit additive model $\sum_t c_t h_t(x)$
- In each stage, introduce a weak learner to compensate the shortcoming of existing weak learner
- In Gradient Boosting,"shortcomings" are identified by gradients. (residual)
- Both high-weight data points and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

Gradient

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

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

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

Feature importance

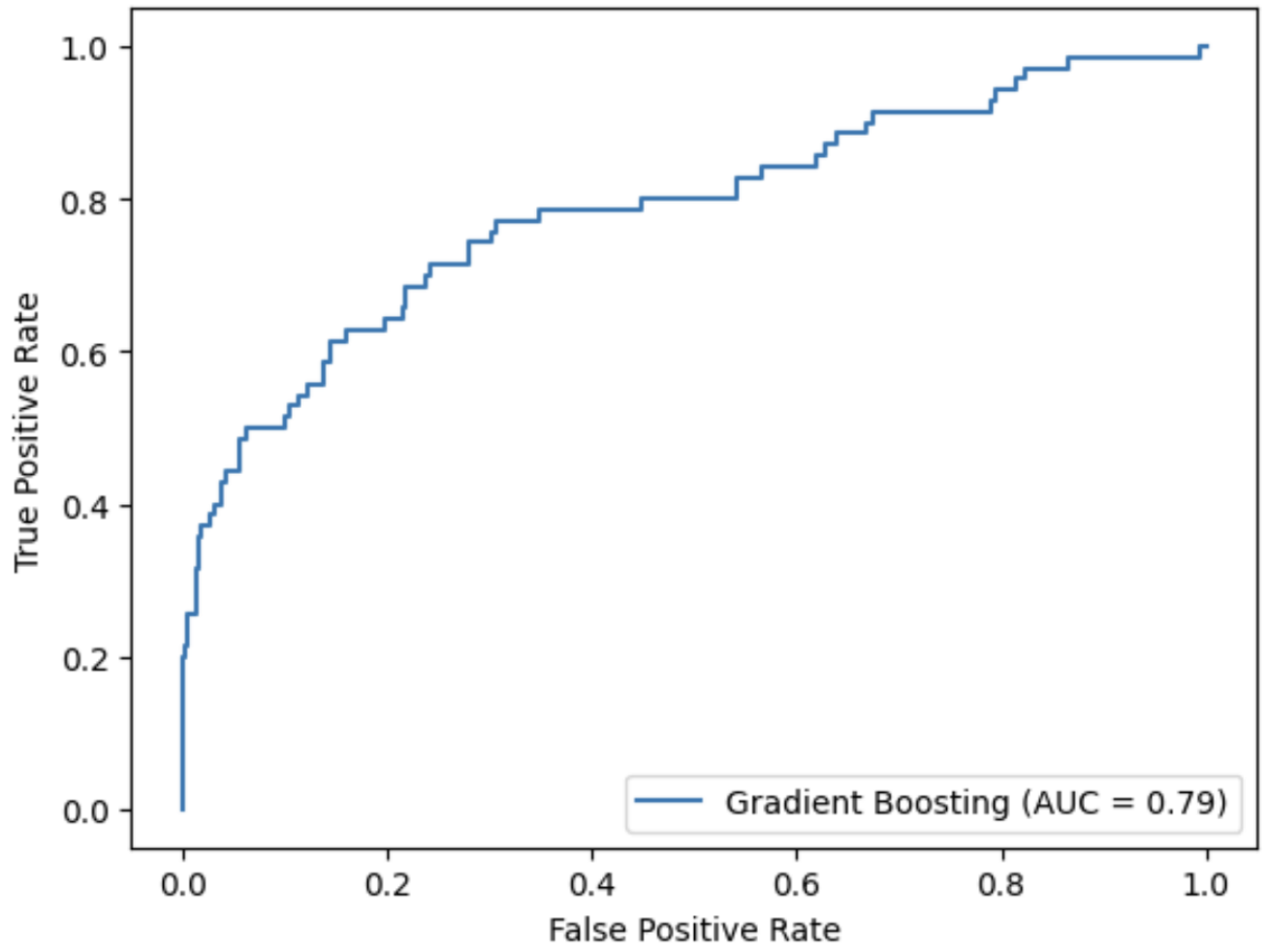
```
[14] import pandas as pd

     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.

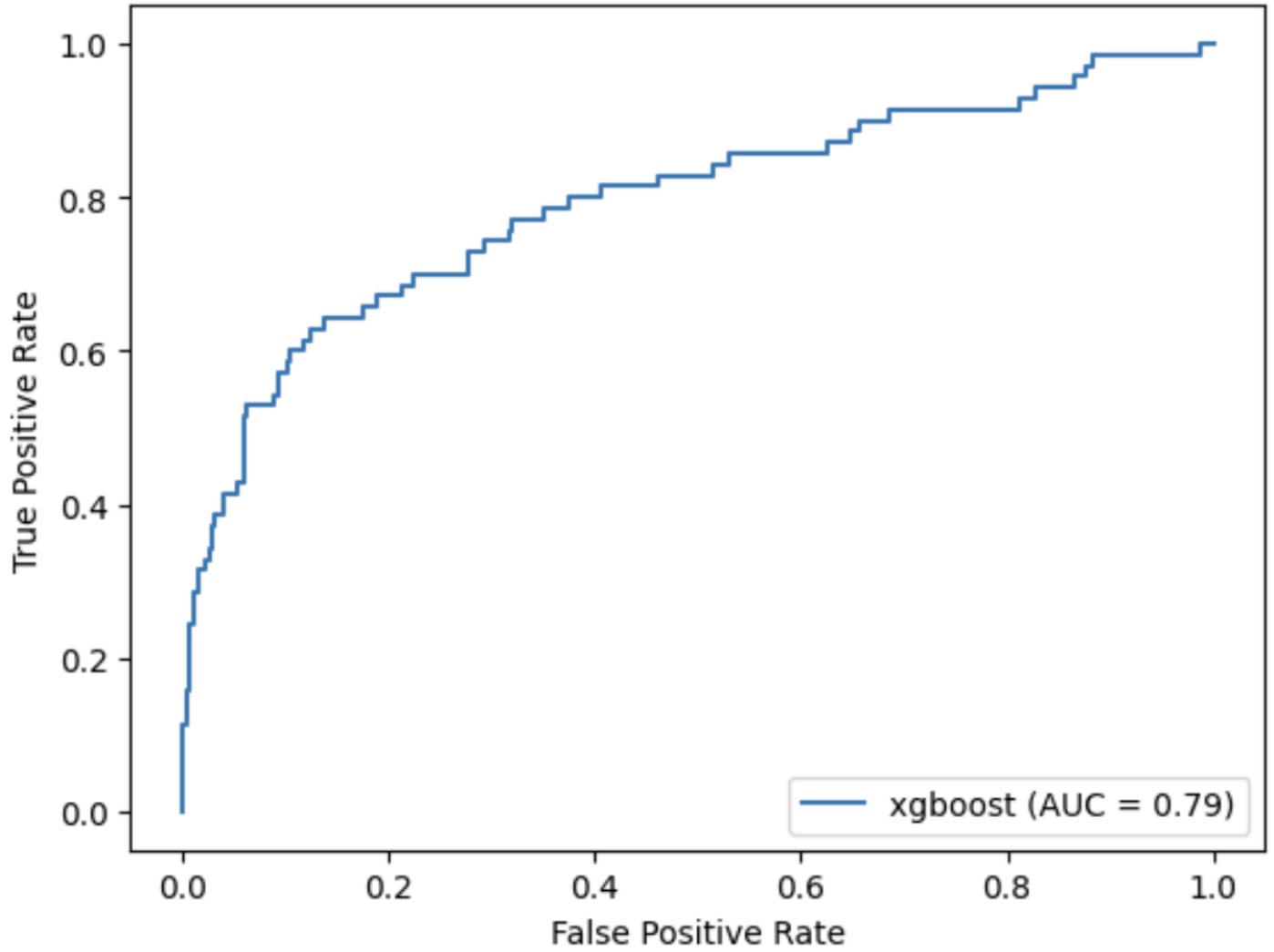
Modeling

```
[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

```
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', 49.44914245605469),
 ('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

End of Lecture 5