ENSEMBLES for Classification

Ensembles

- Methods combining multiple ML models to create low-bias, low-variance, models
- They combine multiple models to create new more accurate models
- Types of ensembles of trees
    - Bagged trees
    - Random Forest
    - Gradient boosting trees

Hyperparameters

Random Forest

- max_features
- n_estimators
- max_depth

Gradient Boosting

- learning_rate
- max_features
- n_estimators
- max_depth

BAGGING

Bootstrap samples (from dataframes)

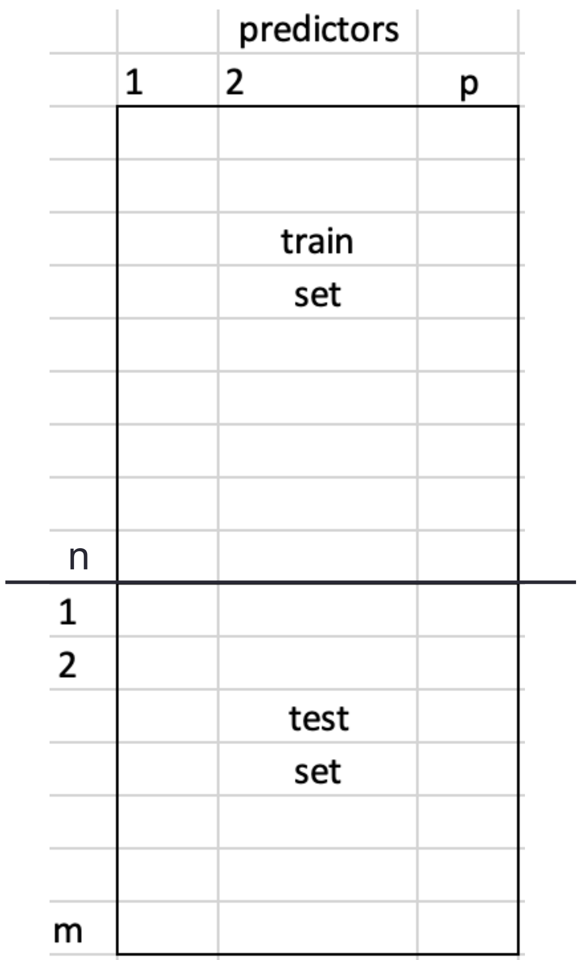
- A bootstrap sample is a sample *with* replacement
- May include same row many times
- Bootstrap samples are usually of the same size

Bagging for **Classification Trees**

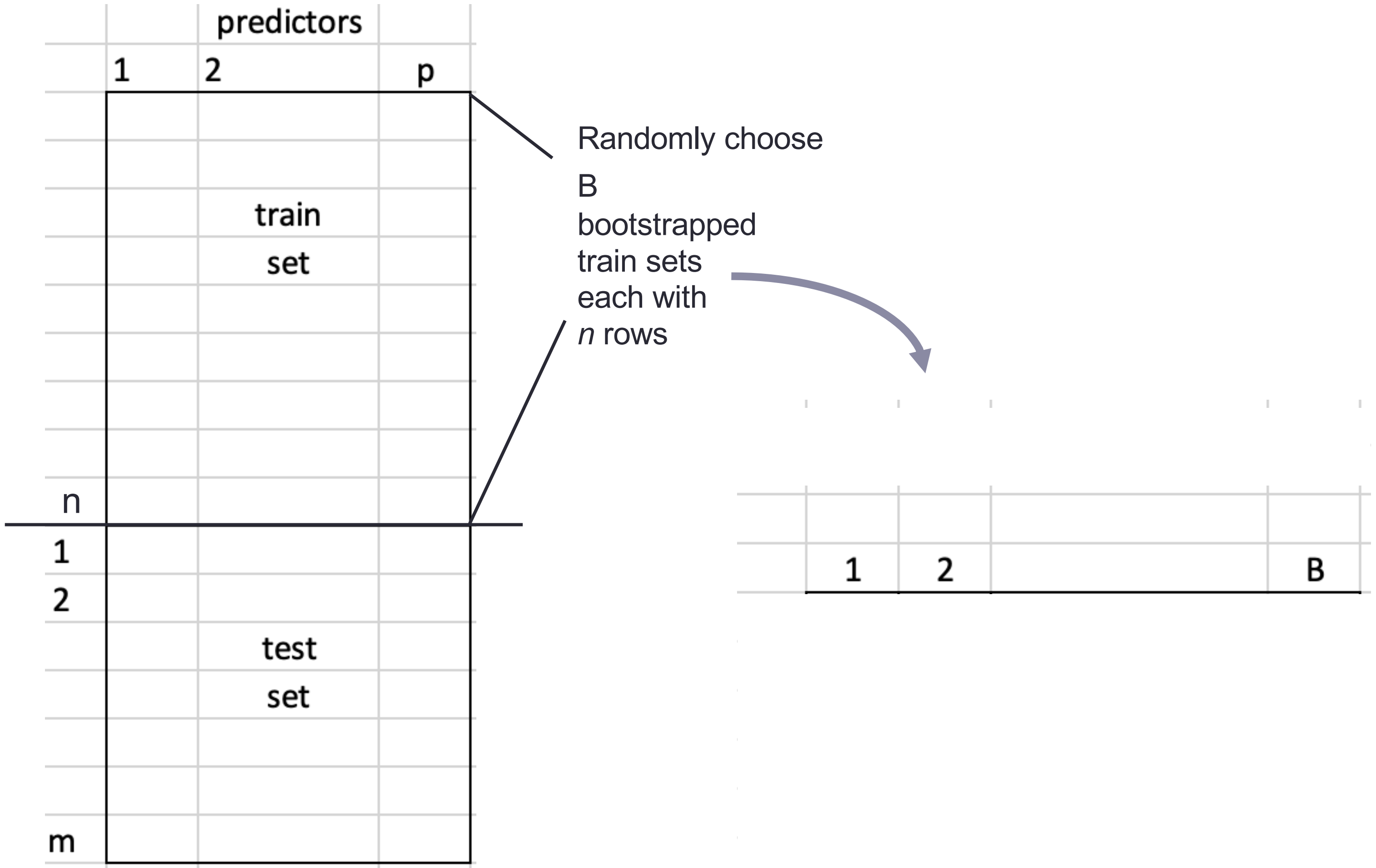
There are two approaches

- Predict the most frequent category (majority vote)
- If the model yields probability estimates, average the probabilities for each class, then predict the class with the highest average

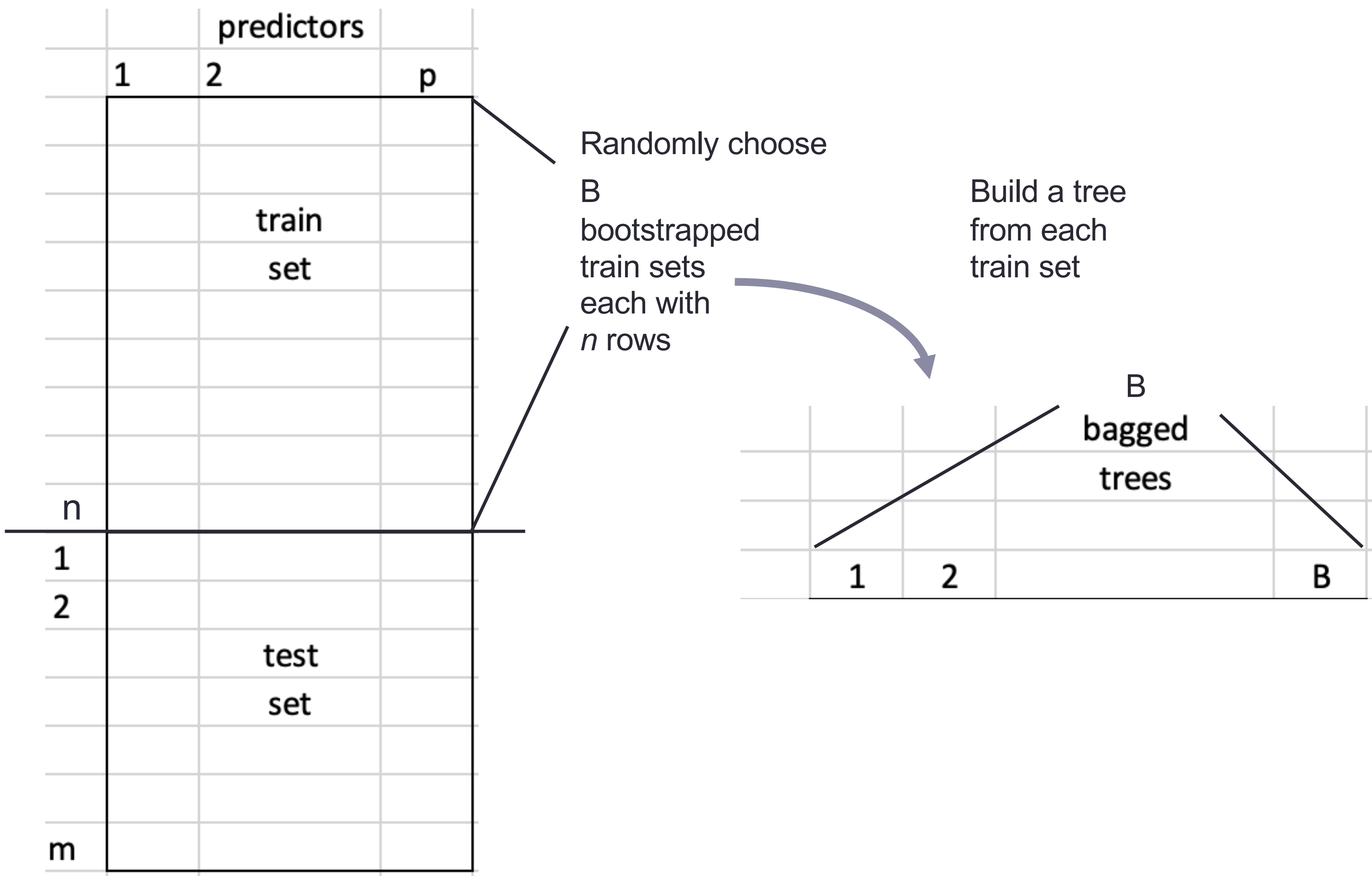
Bagging for Trees



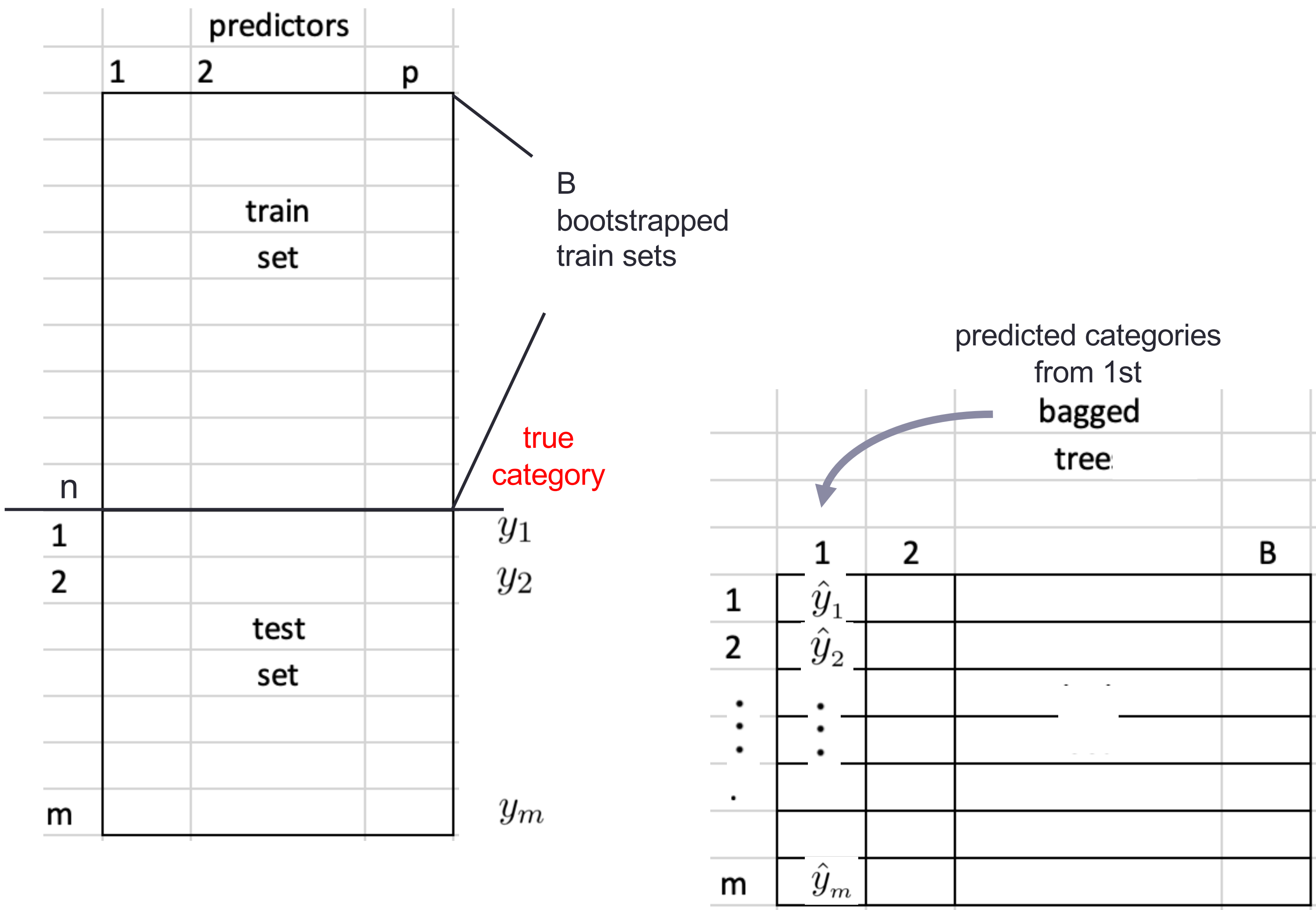
Bagging for Trees



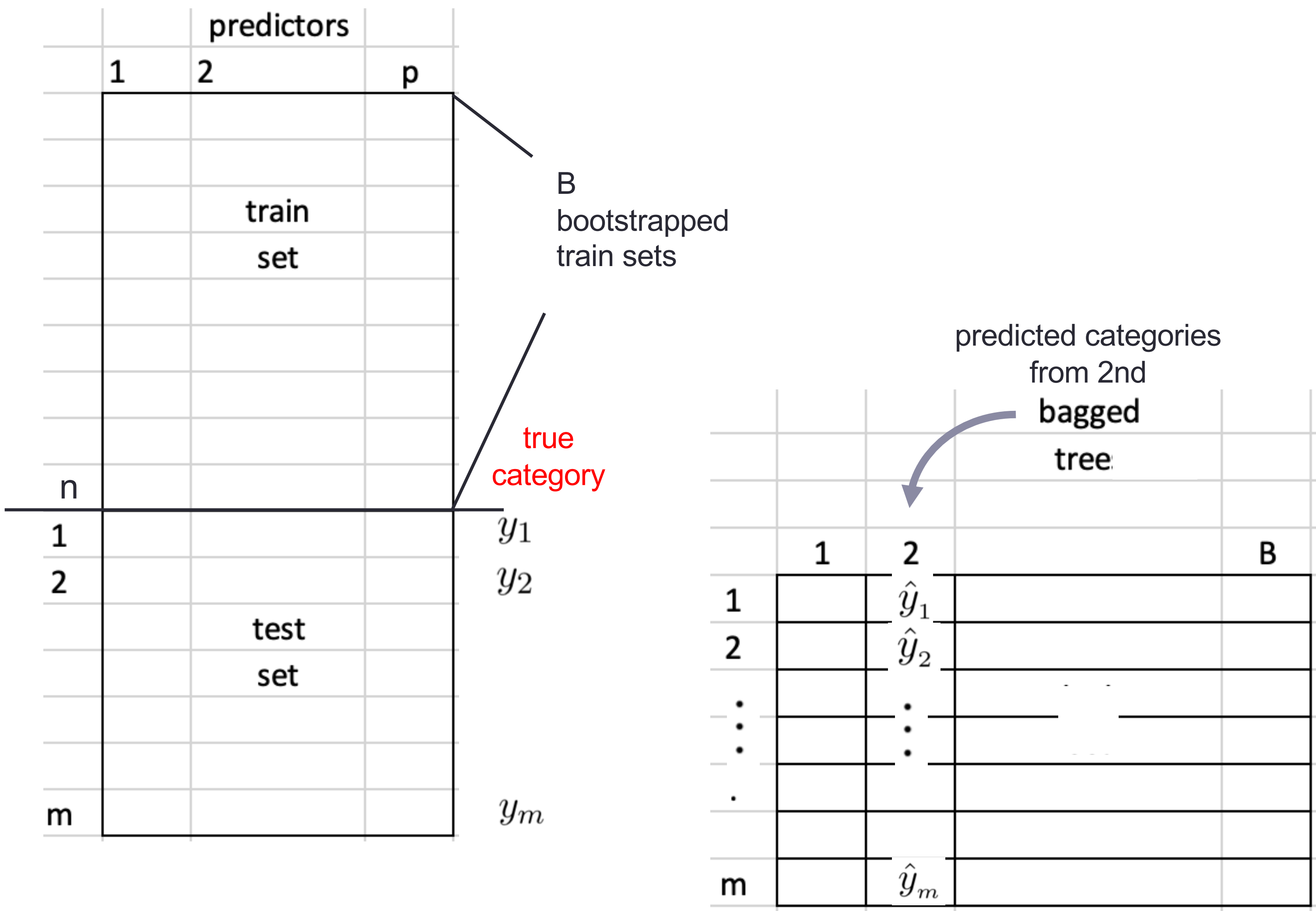
Bagging for Classification Trees



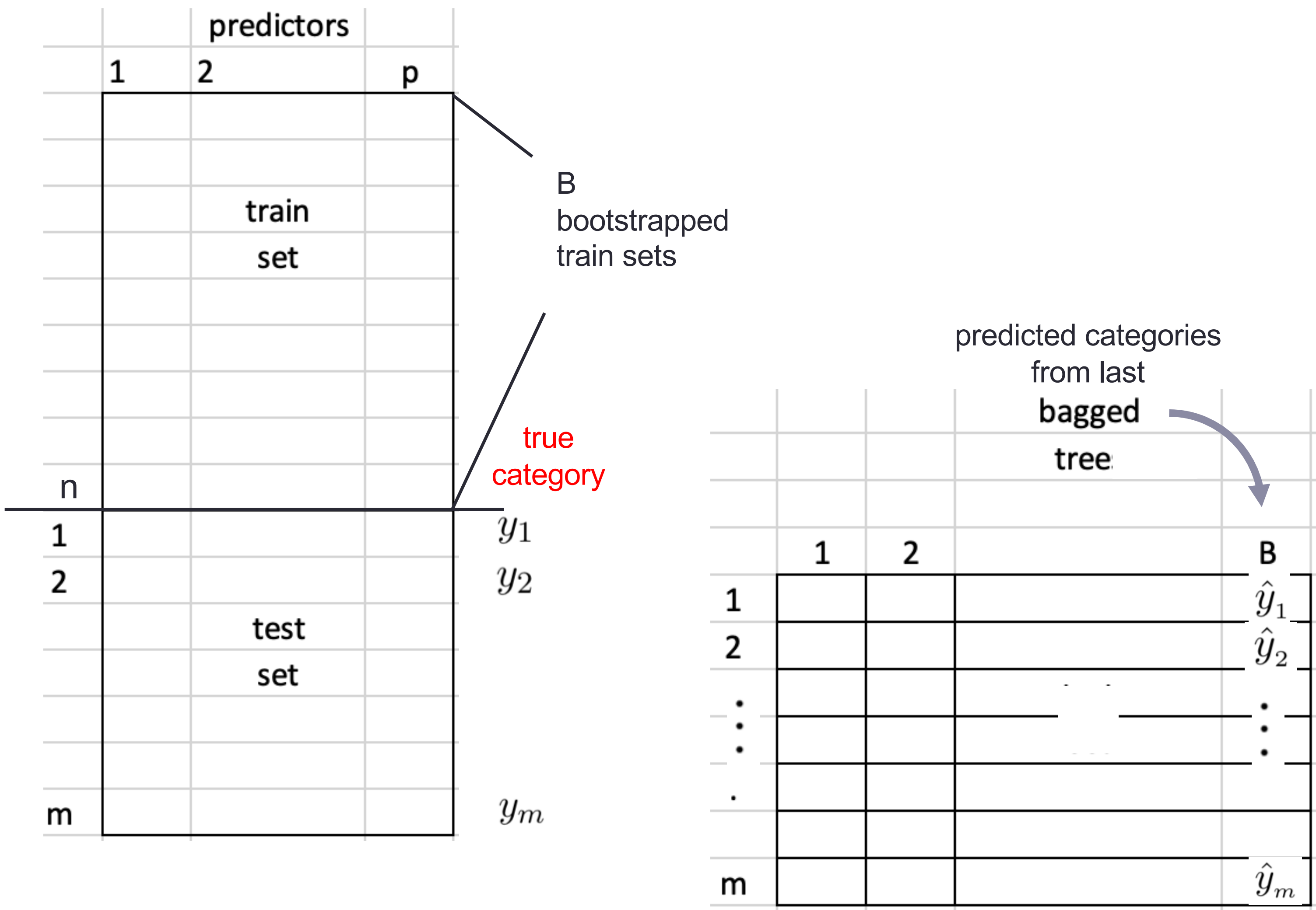
Bagging for Classification Trees



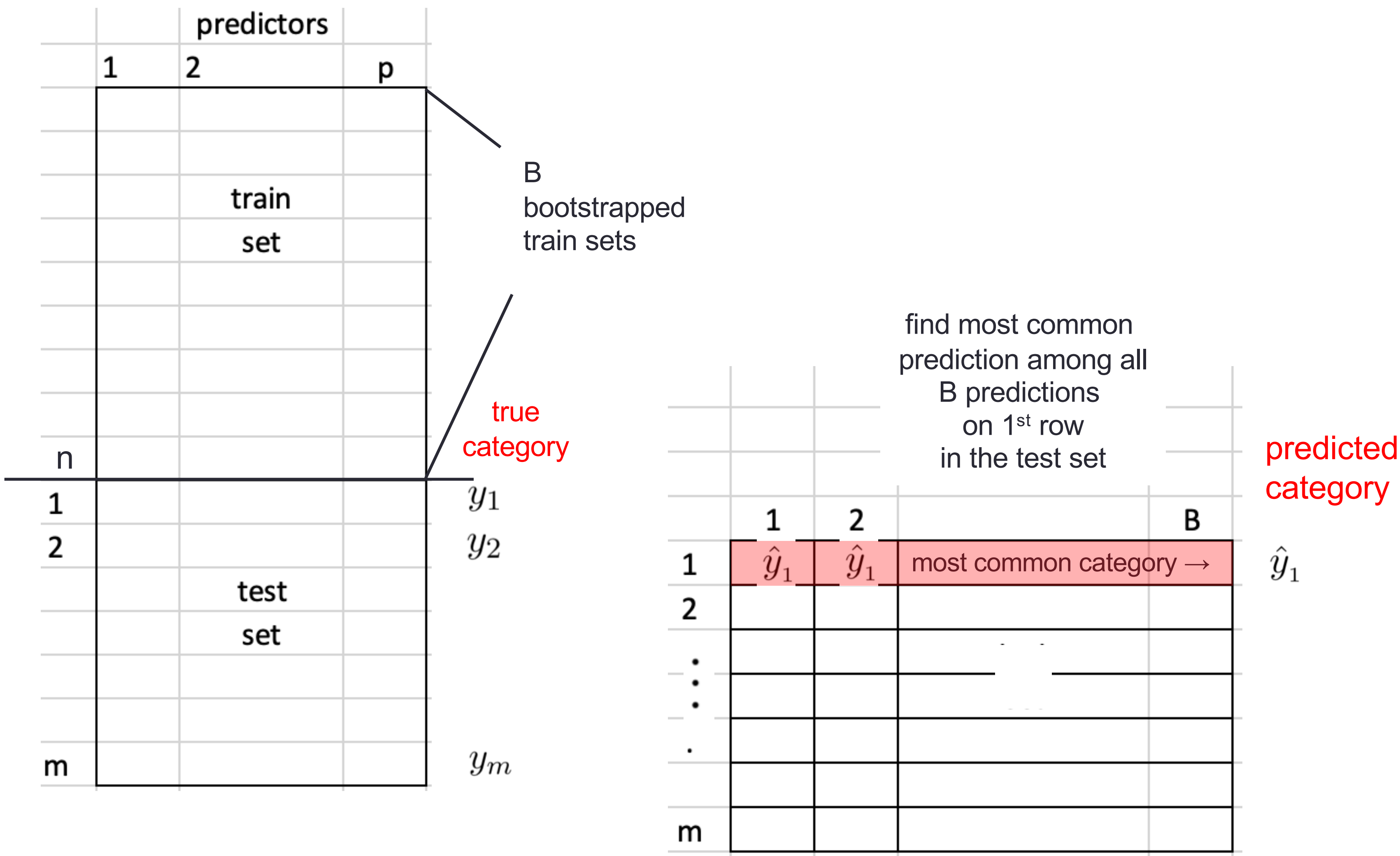
Bagging for Classification Trees



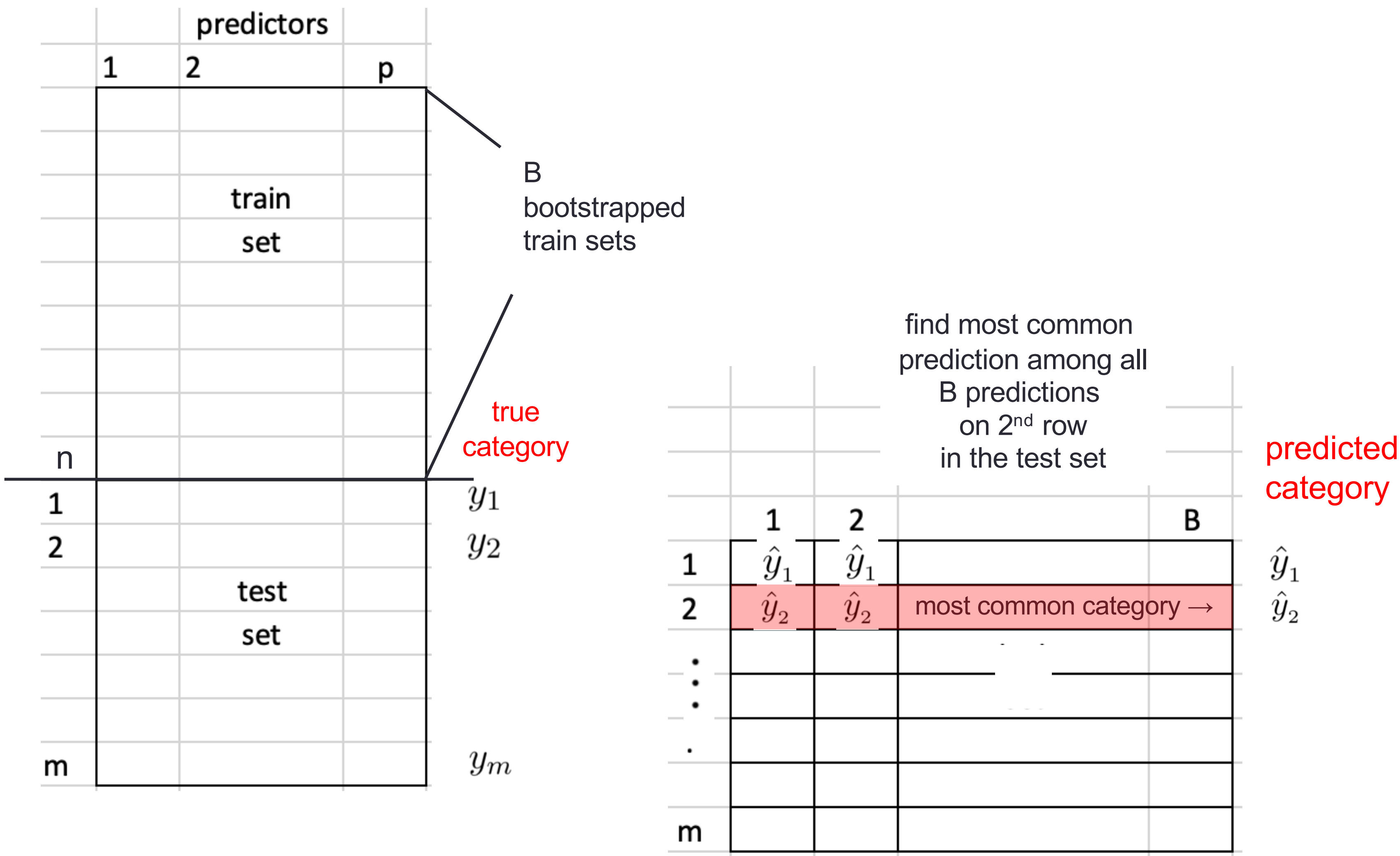
Bagging for Classification Trees



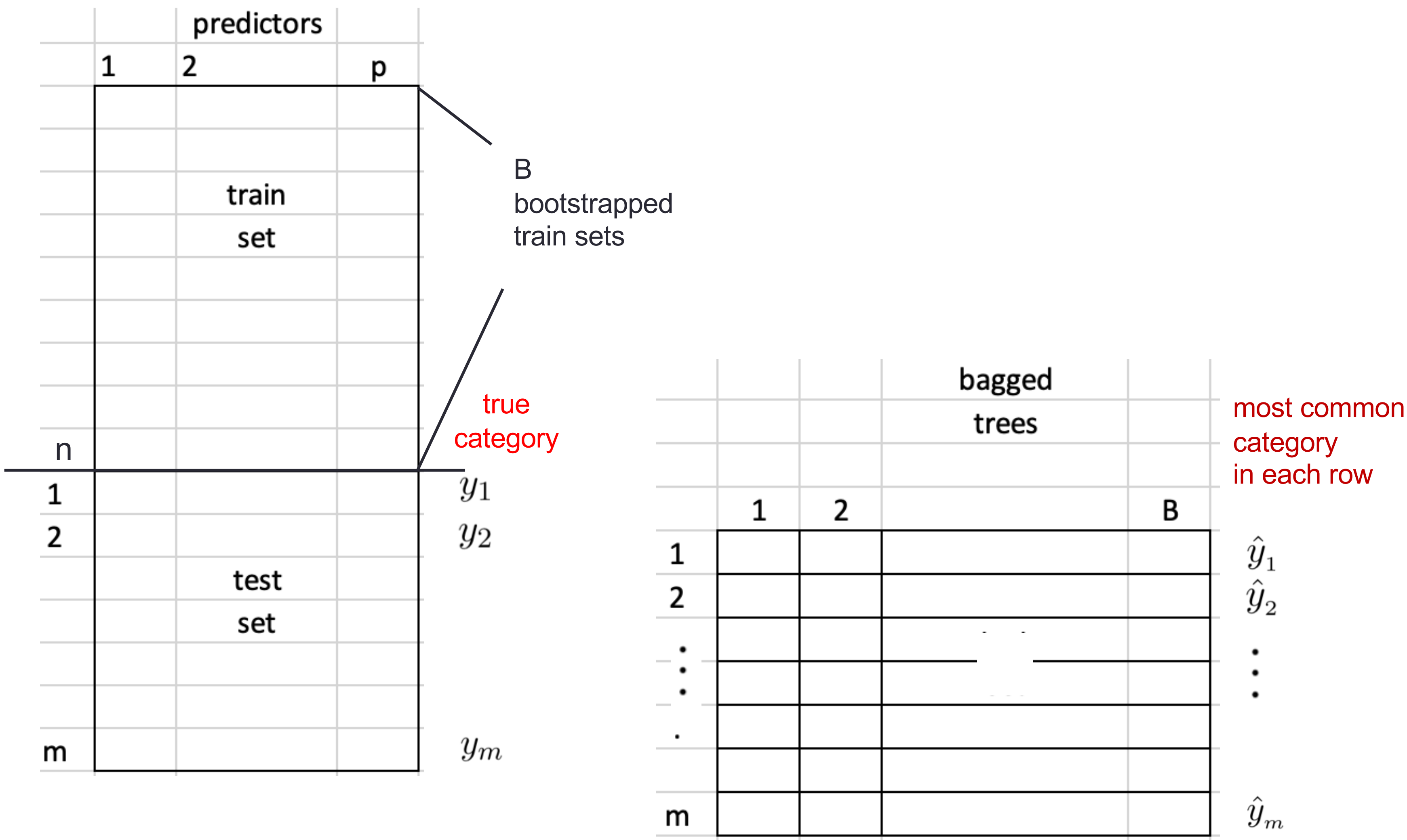
Bagging for Classification Trees



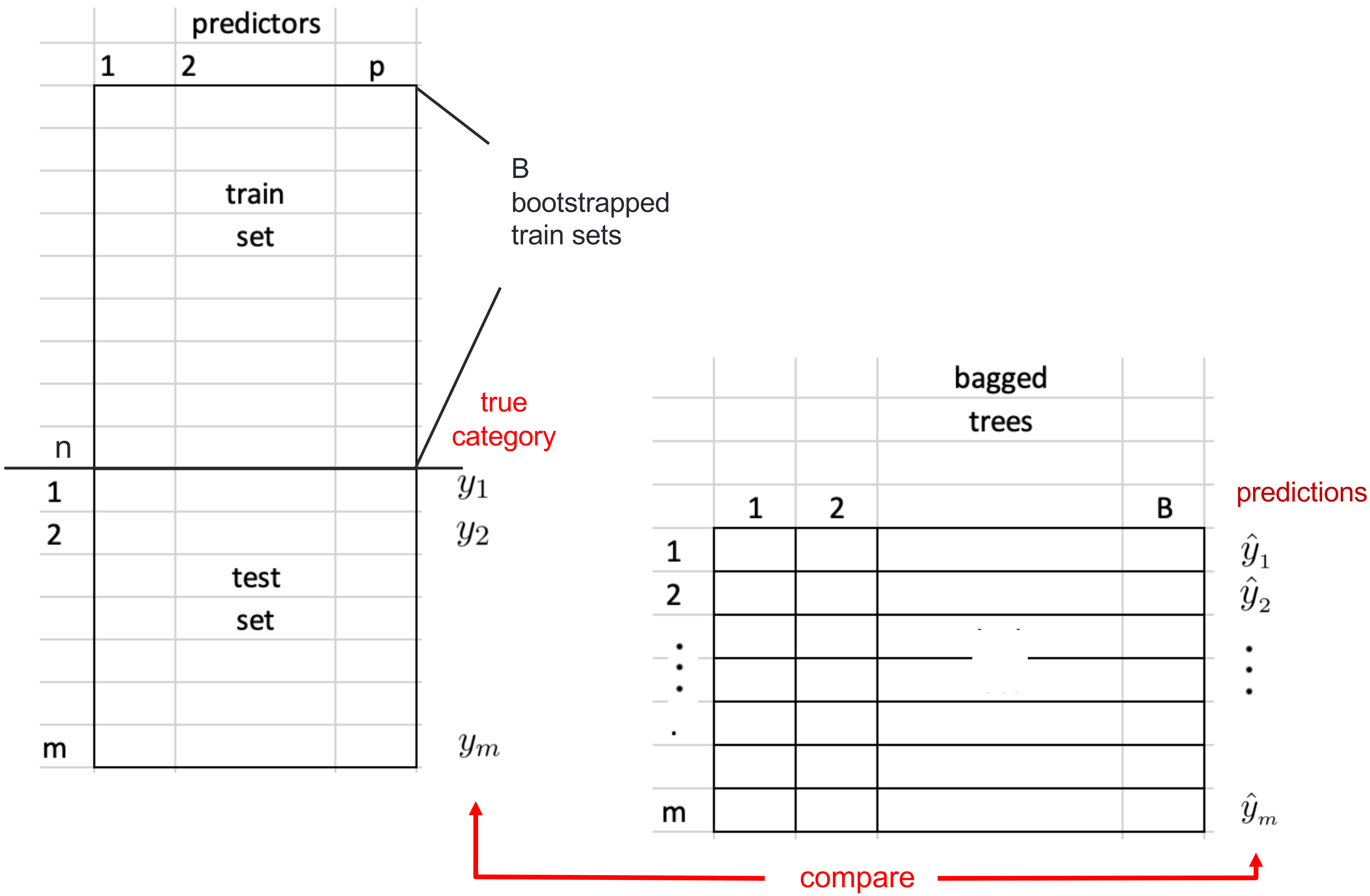
Bagging for Classification Trees



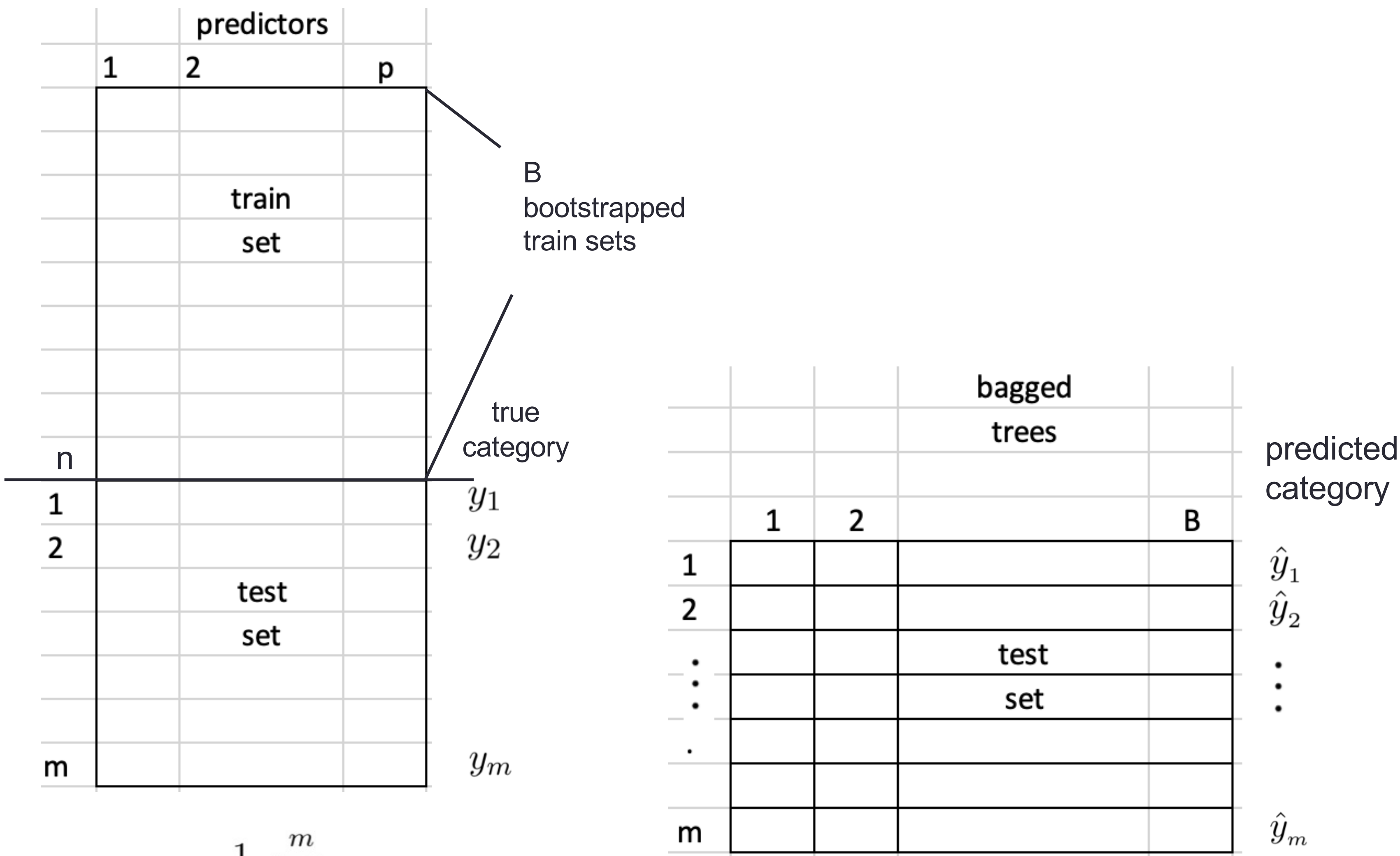
Bagging for Classification Trees



Bagging for Classification Trees



Bagging for Classification Trees



Test
Accuracy Rate

$$\mathrm{AR} = \frac{1}{m}\sum_{i=1}^{m} I(y_i = \hat{y}_i)$$

Bagging - Notes

- Sometimes a few predictors are very good while many are poor predictors
- If so, many of the trees may contain the same set of powerful predictors
- Then the trees would yield similar predictions
- We say that the predictions are co-rrelated
- We need a way to de-correlate them

RANDOM FORESTS

Why are we selecting *m* predictors instead of all *p* predictors for splitting?

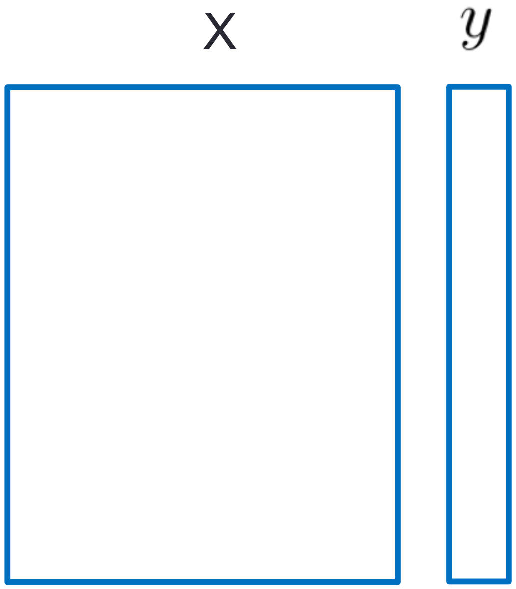
- If there is a single strong predictor, most bagged trees will choose it for the first split (and for the following splits too)
- Most trees will look similar
- As a result their predictions will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction
- By selecting the predictors for splits, from different subsets of predictors, Random Forest “de-correlates” the bagged trees leading to a reduction in variance

GRADIENT BOOSTING

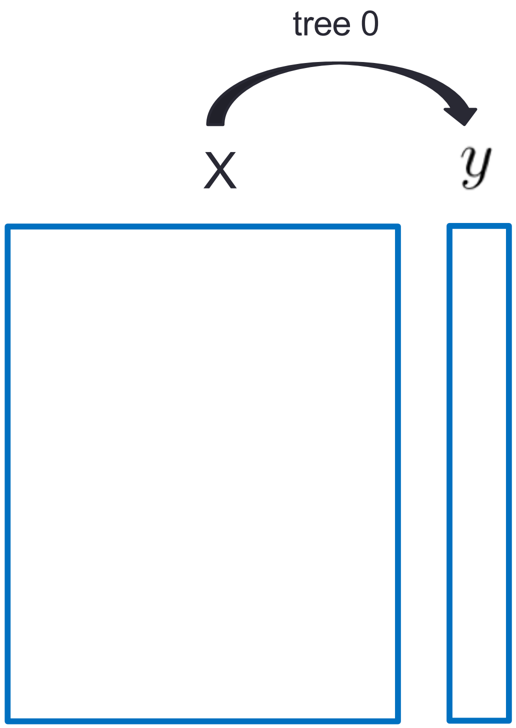
Gradient Boosting

- Trees are built sequentially to improve upon the errors made by their predecessor trees
- Each new tree fits the data to the error made by the previous tree, predicting that error
- The new prediction is equal to the prediction of the previous tree plus α times the predicted error
- Parameter $0 < \alpha < 1$ is called the learning rate

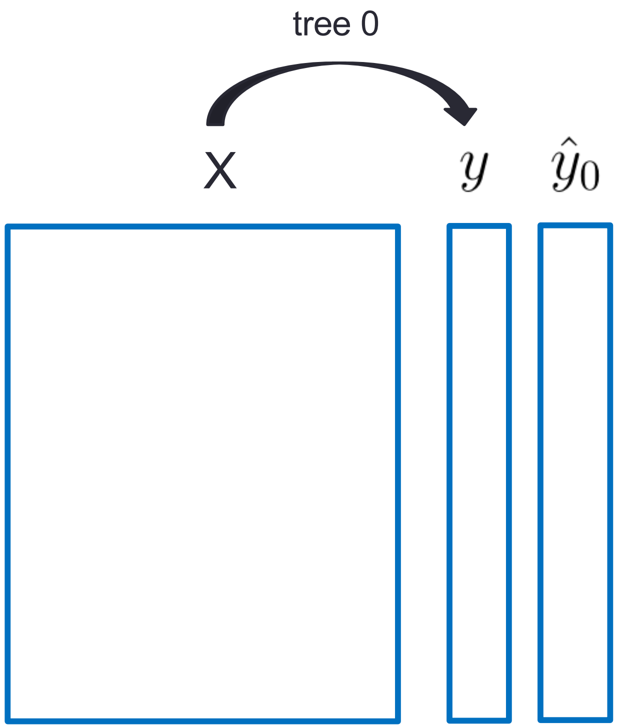
Boosting procedure



Boosting procedure

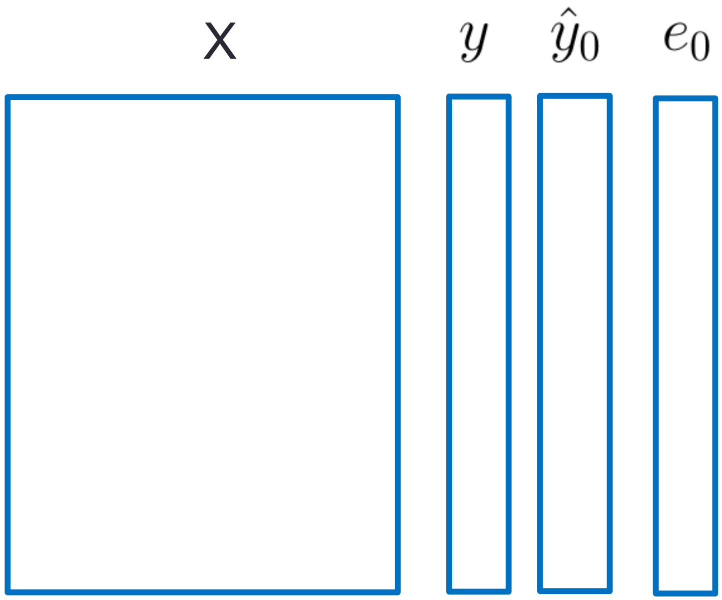


Boosting procedure



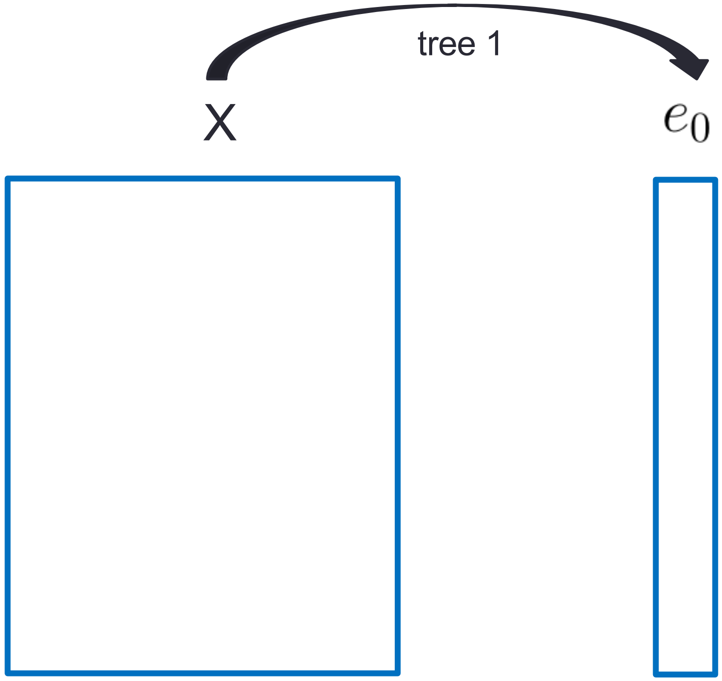
tree 0 predicts $\hat{y}_0$

Boosting procedure

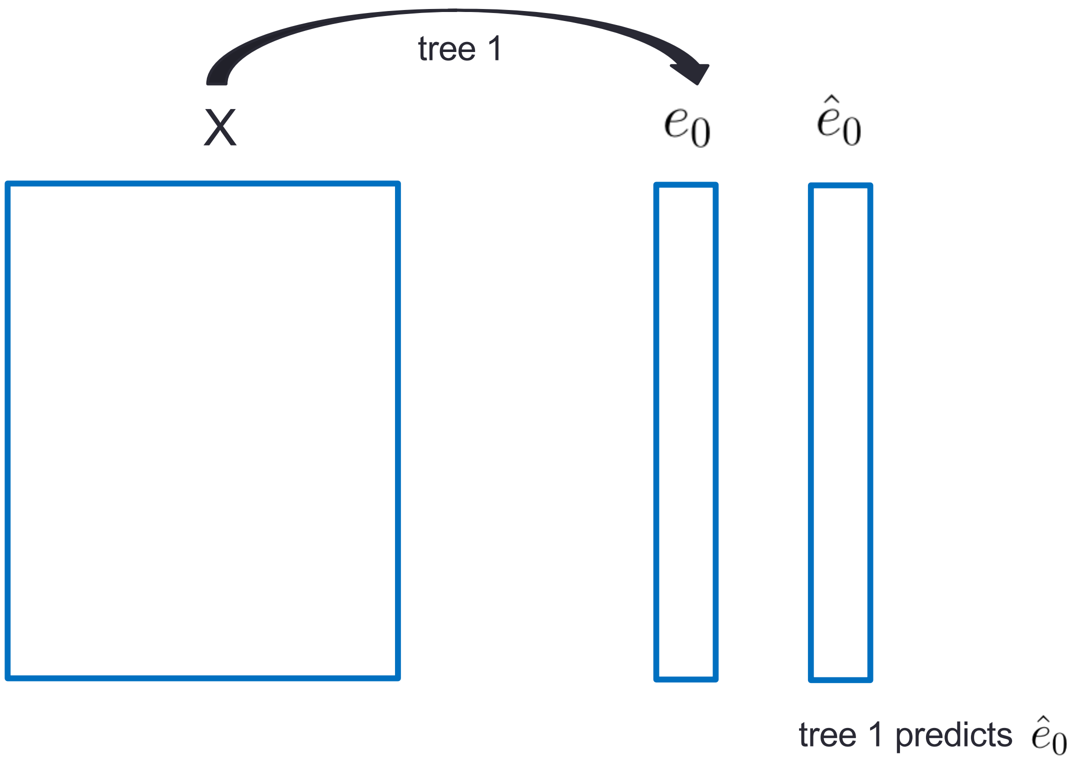


$$e_0 = y - \hat{y}_0$$

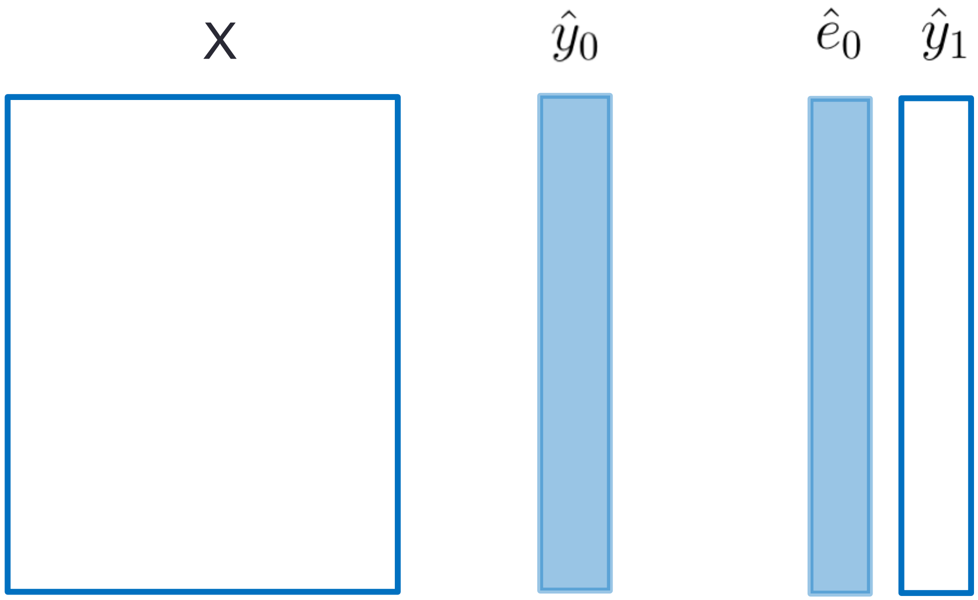
Boosting procedure



Boosting procedure



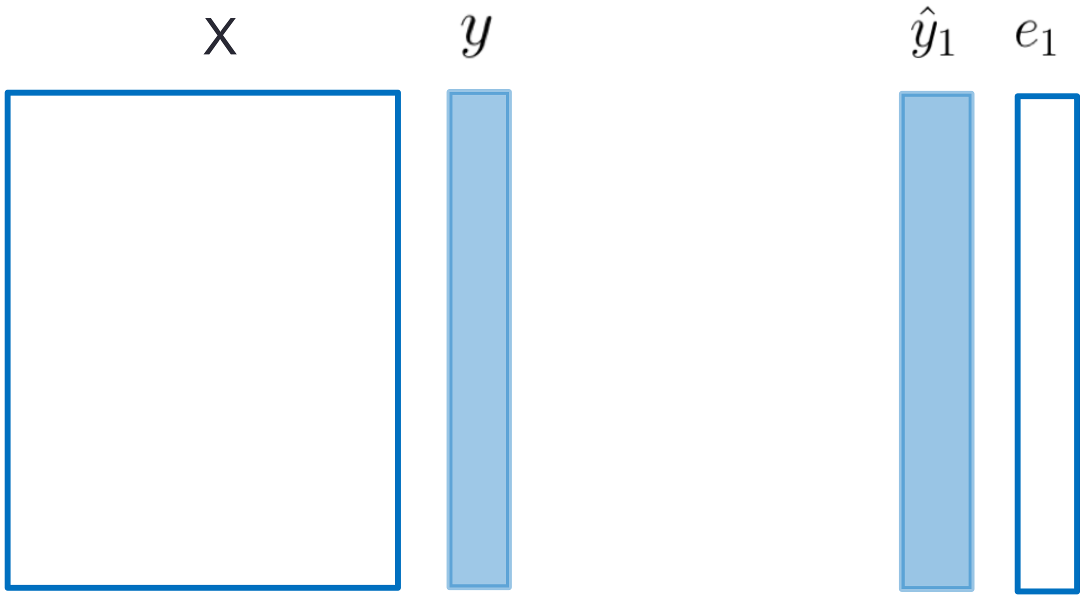
Boosting procedure



tree 1 predicts $\hat{e}_0$

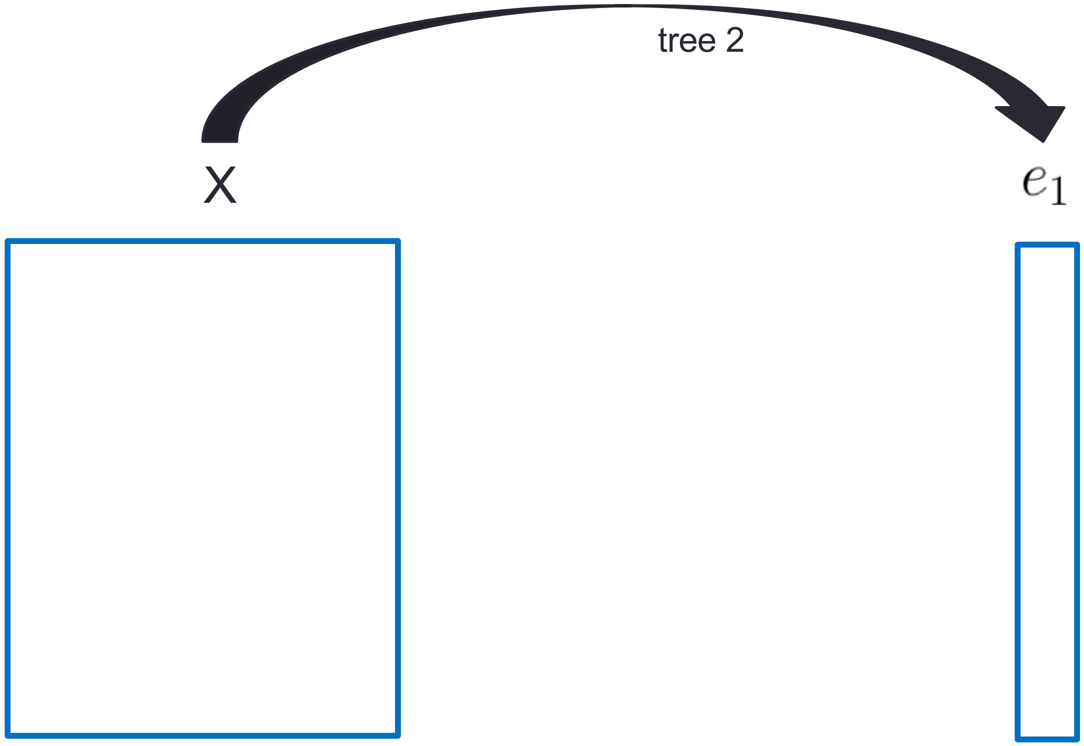
$$\hat{y}_1 \quad = \quad \hat{y}_0 + \alpha\,\hat{e}_0$$

Boosting procedure

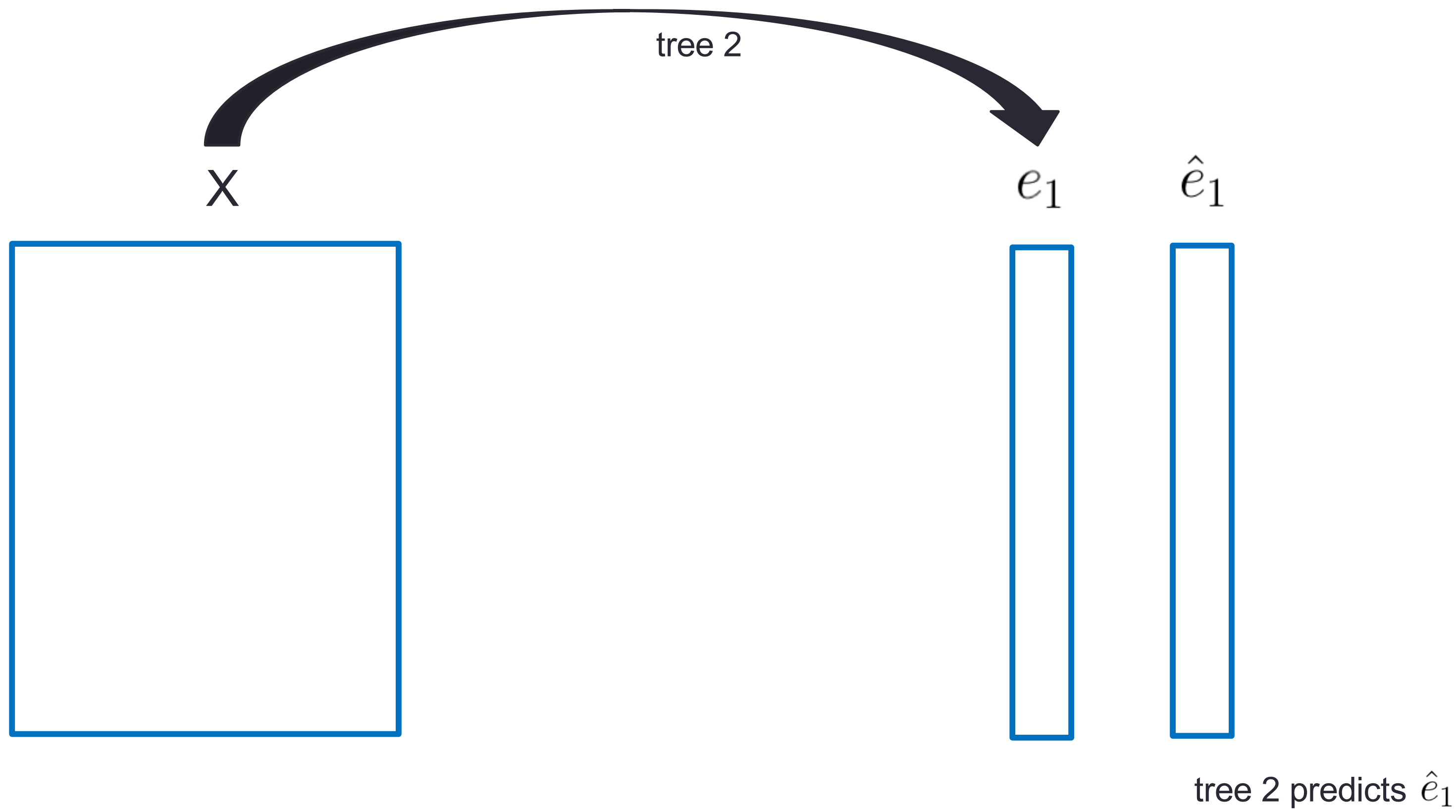


$$e_1 = y - \hat{y}_1$$

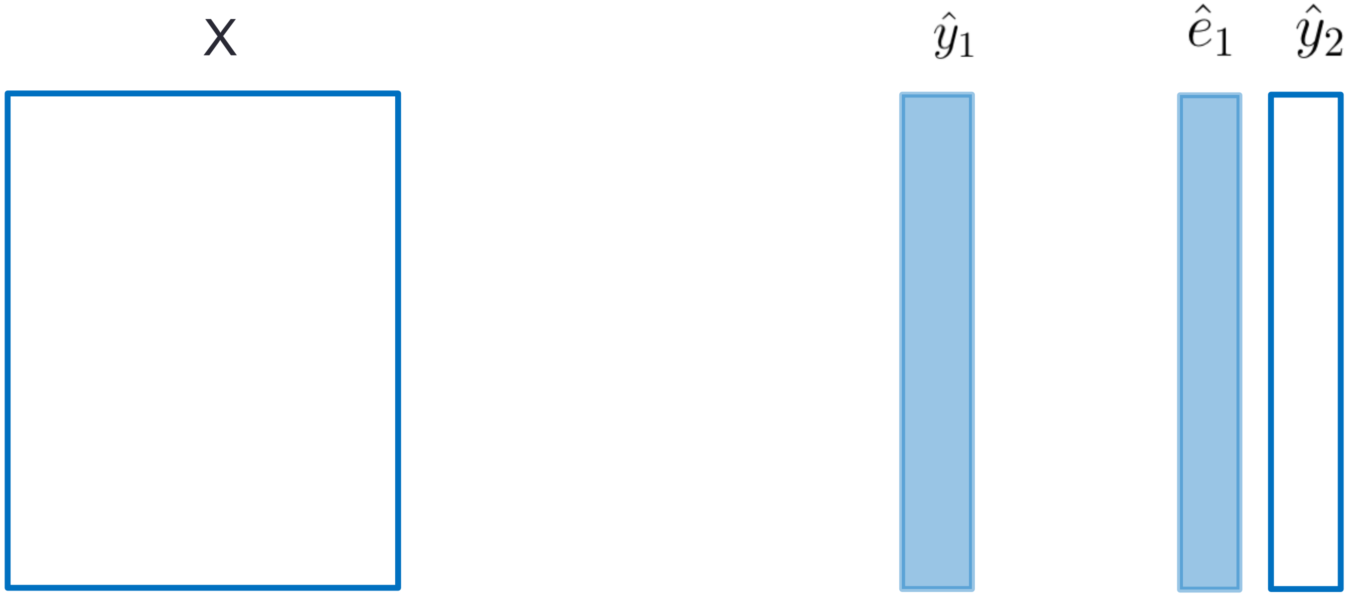
Boosting procedure



Boosting procedure



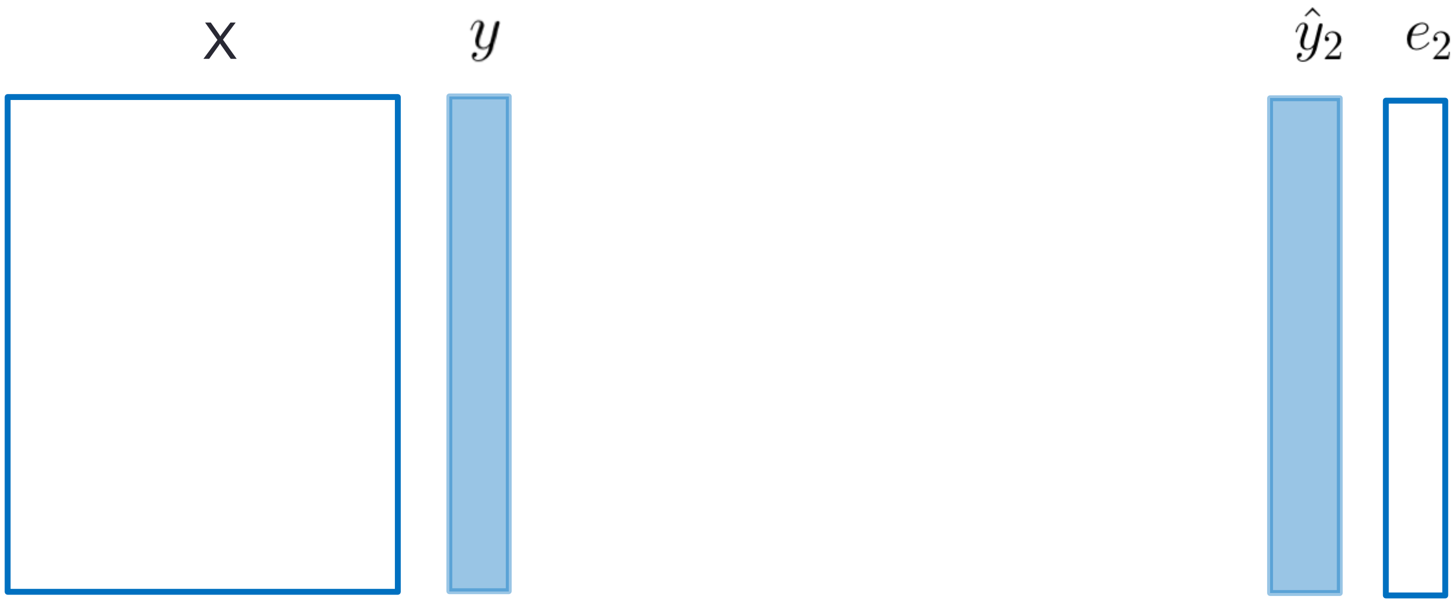
Boosting procedure



tree 2 predicts $\hat{e}_1$

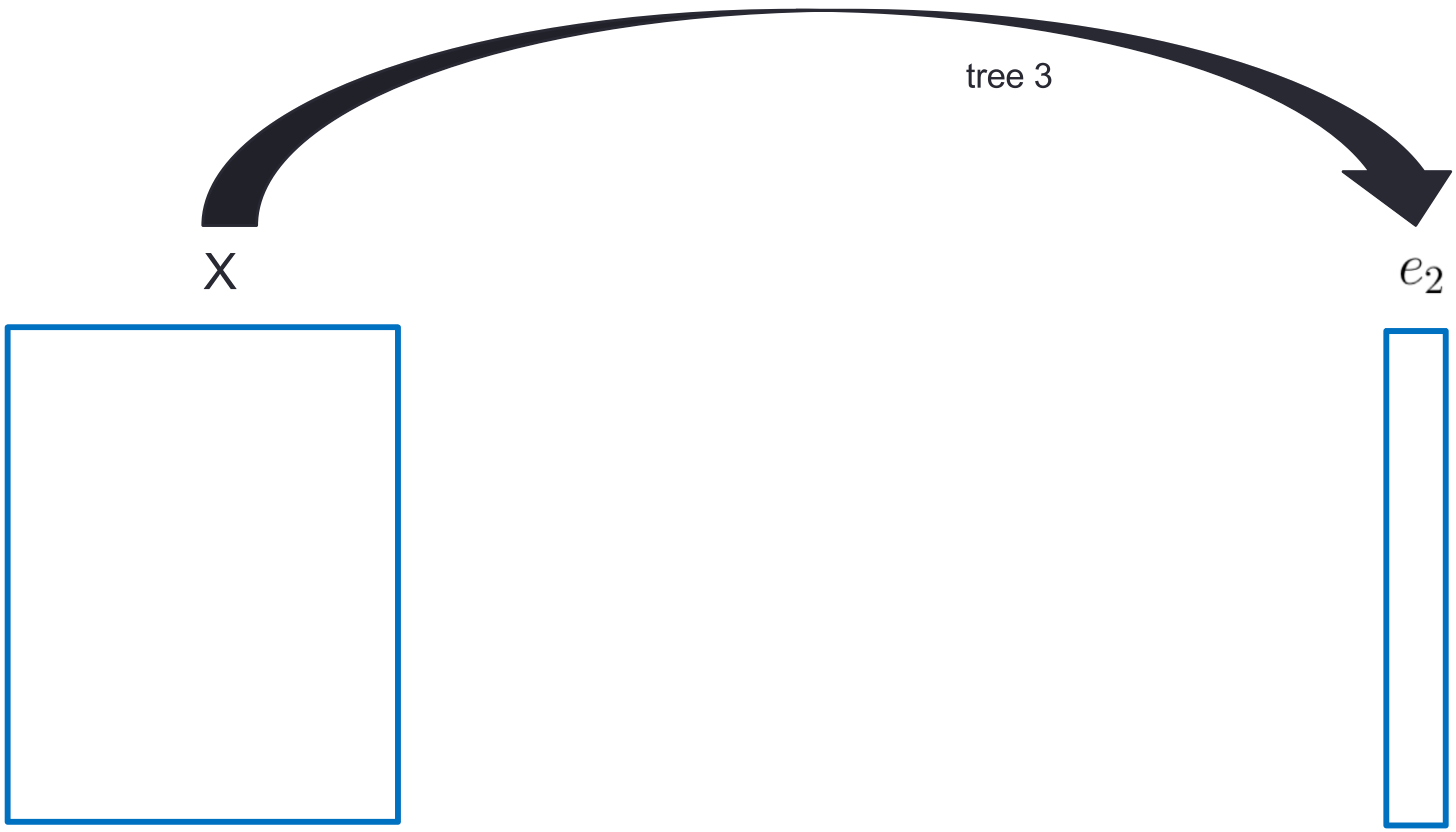
$$\hat{y}_2 = \hat{y}_1 + \alpha\, \hat{e}_1$$

Boosting procedure

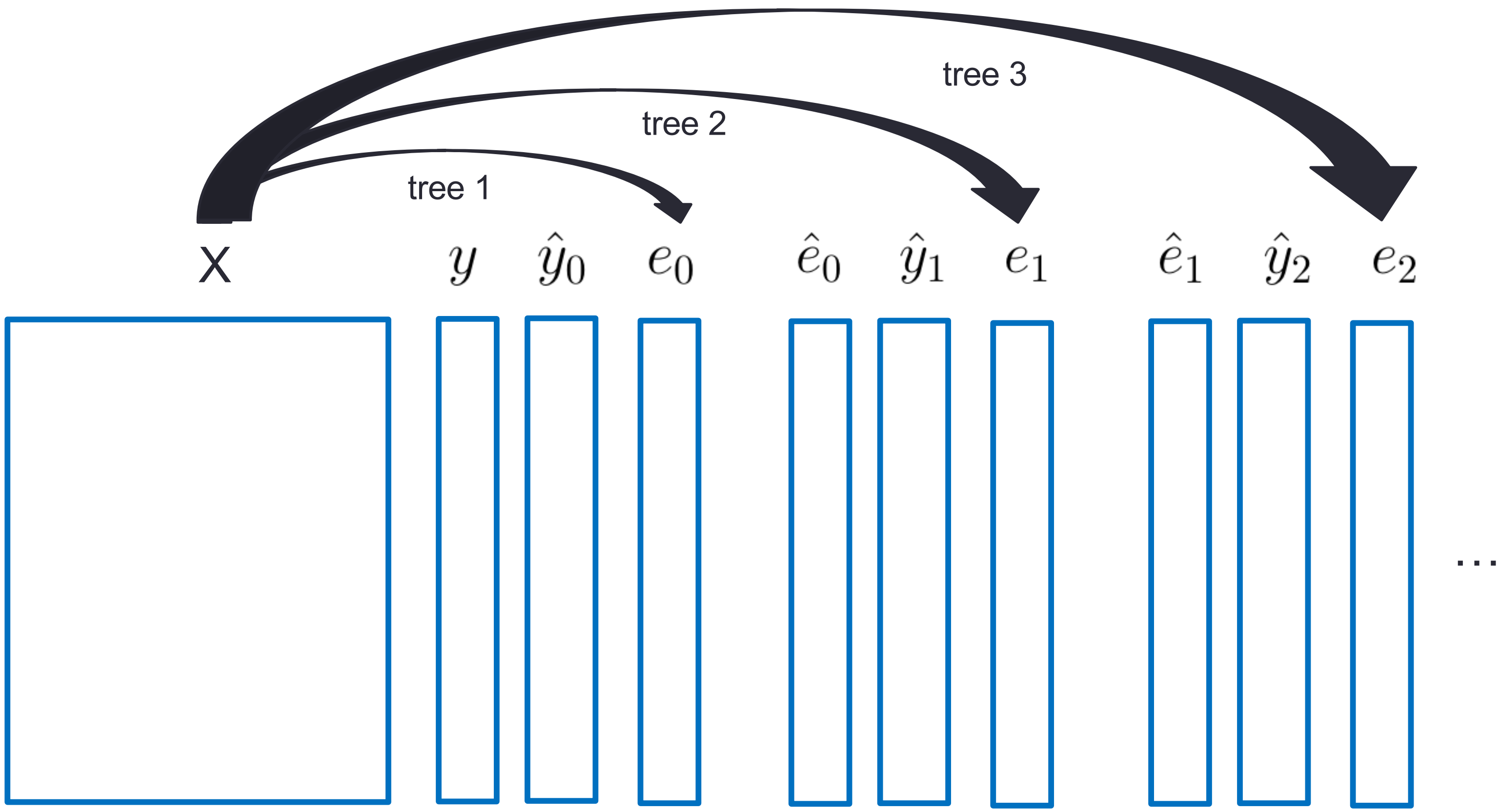


$$e_2 = y - \hat{y}_2$$

Boosting procedure



Boosting procedure



tree 1 predicts $\hat{e}_0$

tree 2 predicts $\hat{e}_1$

$$e_0 = y - \hat{y}_0$$

$$\begin{aligned} \hat{y}_1 &= \hat{y}_0 + \alpha\, \hat{e}_0 \\ e_1 &= y - \hat{y}_1 \end{aligned}$$

$$\begin{aligned} \hat{y}_2 &= \hat{y}_1 + \alpha\, \hat{e}_1 \\ e_2 &= y - \hat{y}_2 \end{aligned}$$

Example 1

Example – Polynomial data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

```
df = pd.read_csv('dataset.csv')
df[:5]
```

	X	y
0	-0.125460	0.051573
1	0.450714	0.594480
2	0.231994	0.166052
3	0.098658	-0.070178
4	-0.343981	0.343986

```
df.shape
```

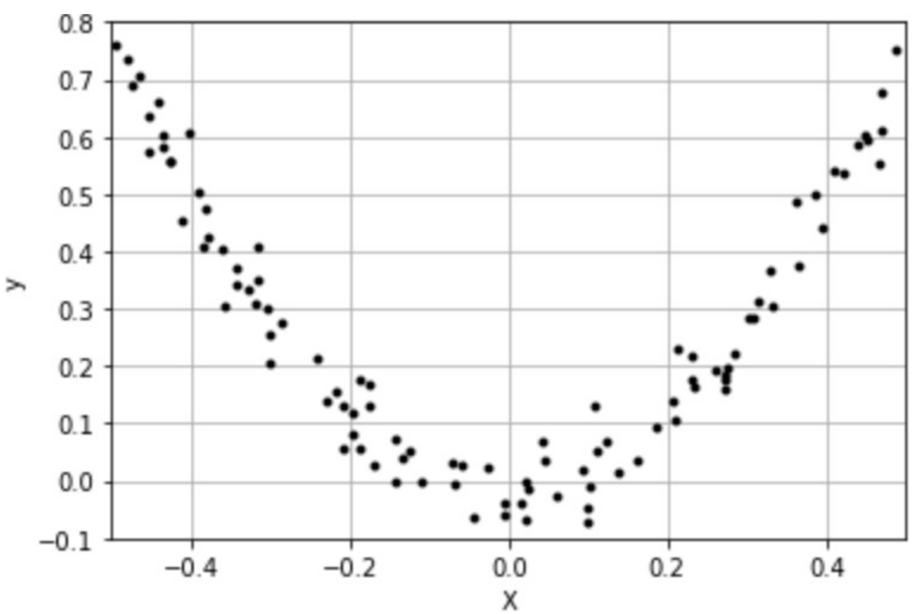
```
(100, 2)
```

```
y = df.y
X = df.drop(['y'],axis =1)
```

Example – Polynomial data

```
plt.plot(X, y,'k.')

# boundary for x and y axes
axes=[-0.5, 0.5, -0.1, 0.8]
plt.axis(axes)
plt.xlabel('X')
plt.ylabel('y')
plt.grid();
```




```
# make sequence of x-coordinate values
seq = np.linspace(-0.5,0.5, 500)
```

```
# transform seq to a dataframe
x1 = pd.DataFrame()
x1['X'] = seq
x1[:5]
```

	X
0	-0.500000
1	-0.497996
2	-0.495992
3	-0.493988
4	-0.491984

```
x1.shape
```

```
(500, 1)
```

Example – Model 1

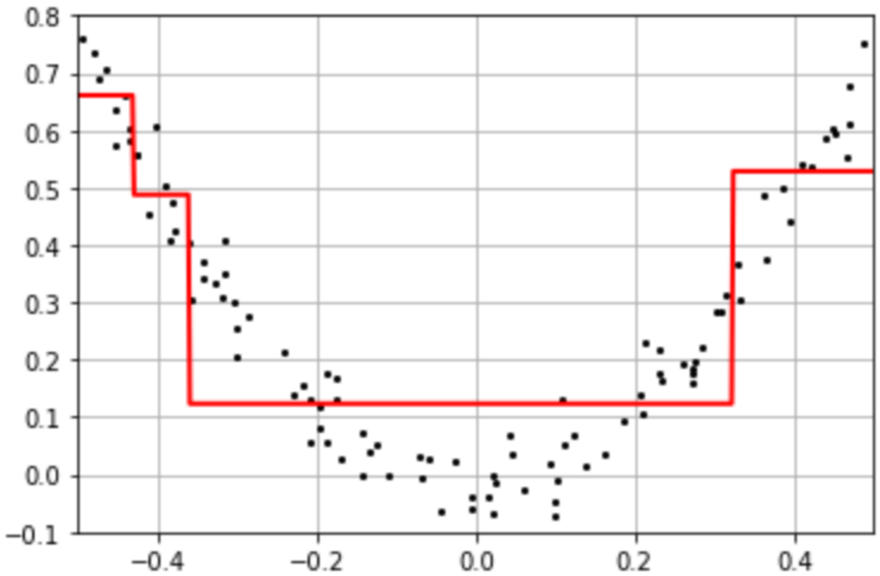
```
# build the regression tree
tree = DecisionTreeRegressor(max_depth=2, random_state=42)
```

```
tree.fit(X,y)
red_line1 = tree.predict(x1)
```

← X is a 1D array with 100 values

← x1 is a 1D array with 500 values

```
plt.plot(X,y,"k.",markersize=4)
plt.plot(x1,red_line1,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```




```
yhat1 = tree.predict(X)
e1 = y - yhat1
```

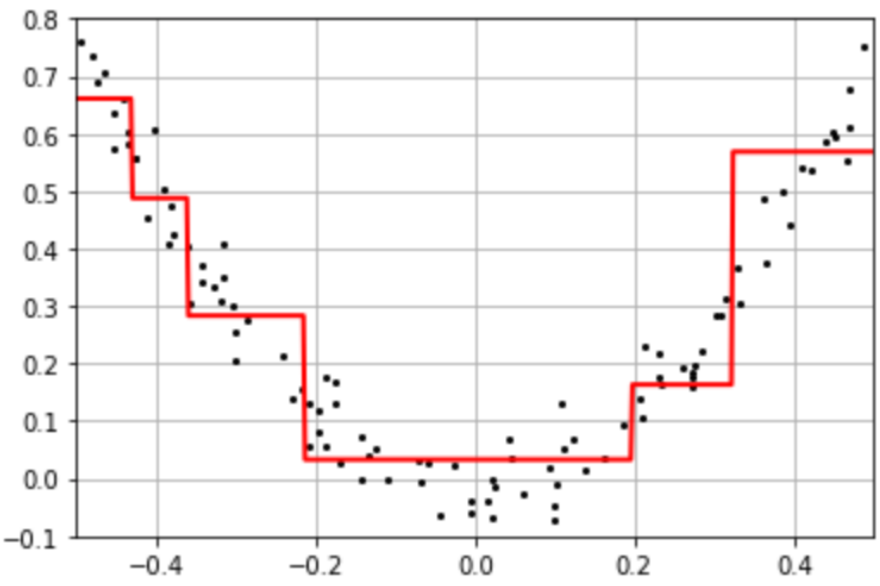
Example – Model 2

.

$\underbrace{}$ e_hat1

```
tree.fit(X, e1)
red_line2 = red_line1 + tree.predict(x1)
```

```
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1, red_line2,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```



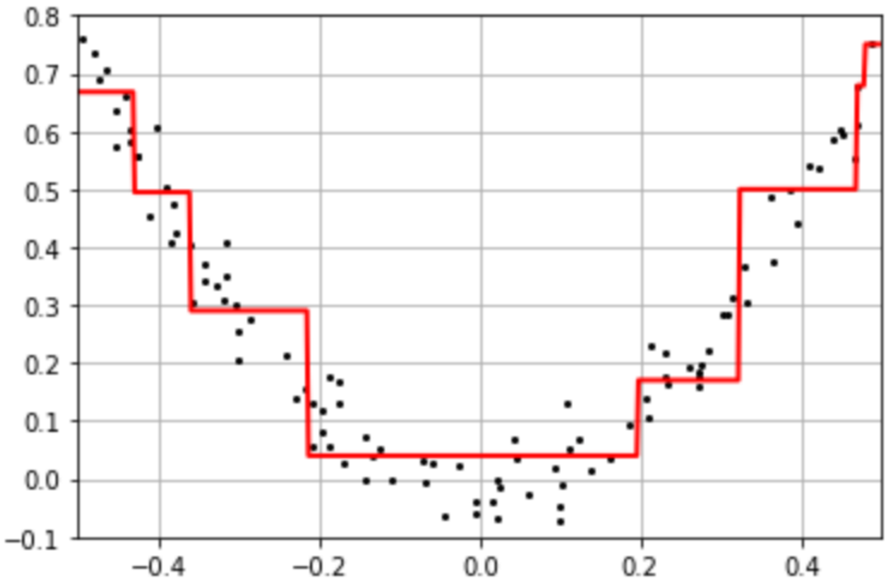

```
yhat2 = tree.predict(X)
e2 = e1 - yhat2
```

Example – Model 3

e_hat2

```
tree.fit(X, e2);
red_line3 = red_line2 + tree.predict(x1)
```

```
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1,red_line3,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```

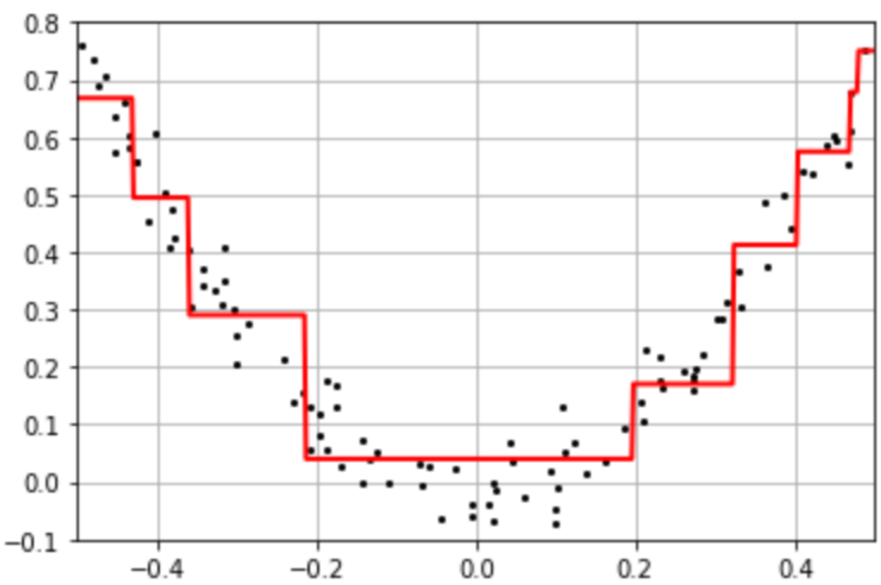



```
yhat3 = tree.predict(X)
e3 = e2 - yhat3
```

Example – Model 4

```
tree.fit(X,e3)
red_line4 = red_line3 + tree.predict(x1)
```

```
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1,red_line4,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```

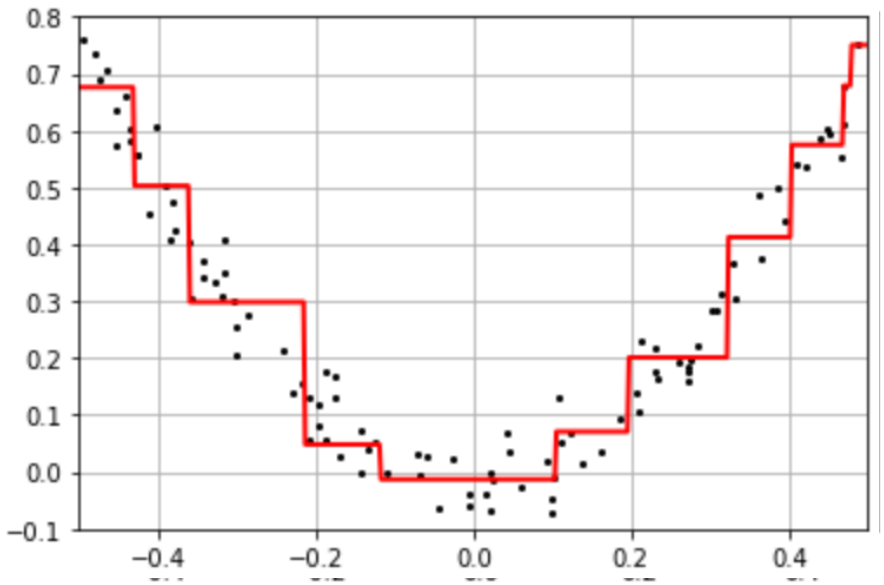



```
yhat4 = tree.predict(X)
e4 = e3 - yhat4
```

Example – Model 5

```
tree.fit(X,e4)
red_line5 = red_line4 + tree.predict(x1)
```

```
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1,red_line5,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```

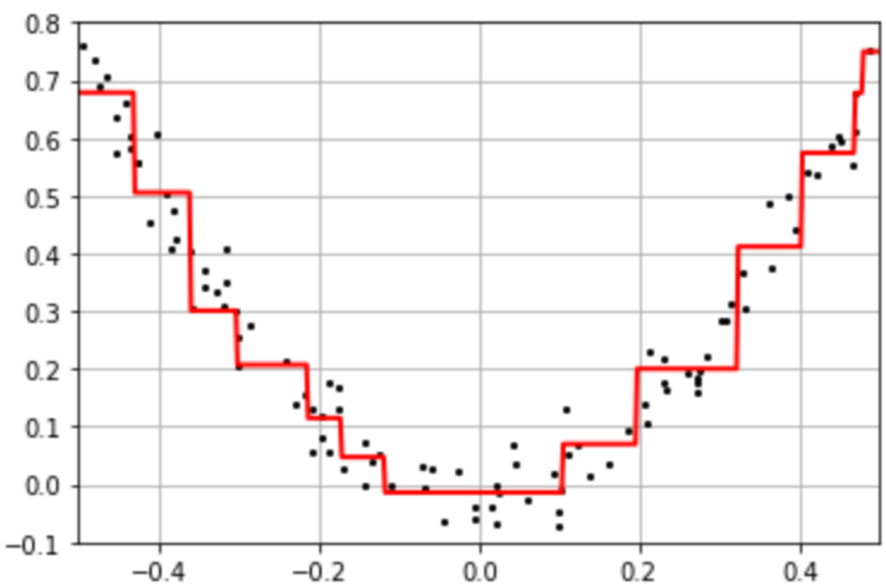



```
yhat5 = tree.predict(X)
e5 = e4 - yhat5
```

Example – Model 6

```
tree.fit(X,e5)
red_line6 = red_line5 + tree.predict(x1)
```

```
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1,red_line6,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```

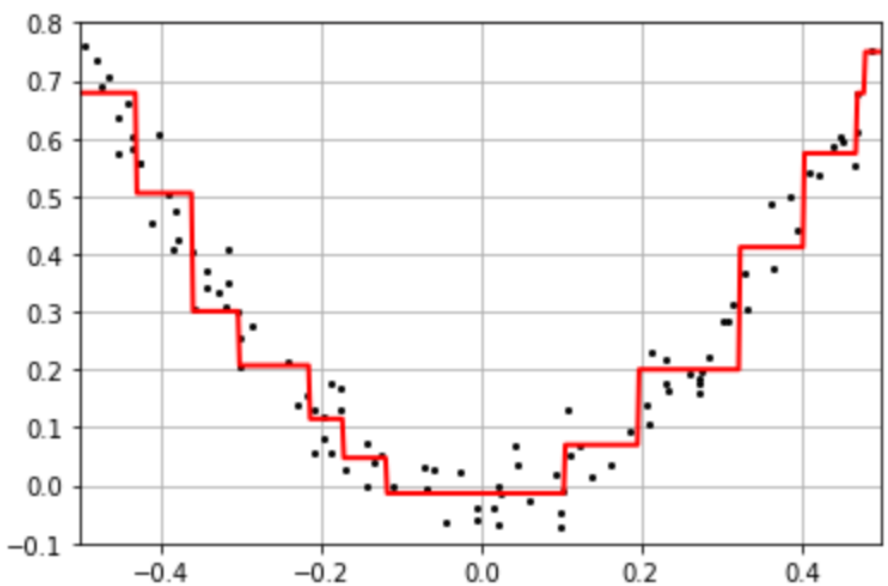


Example

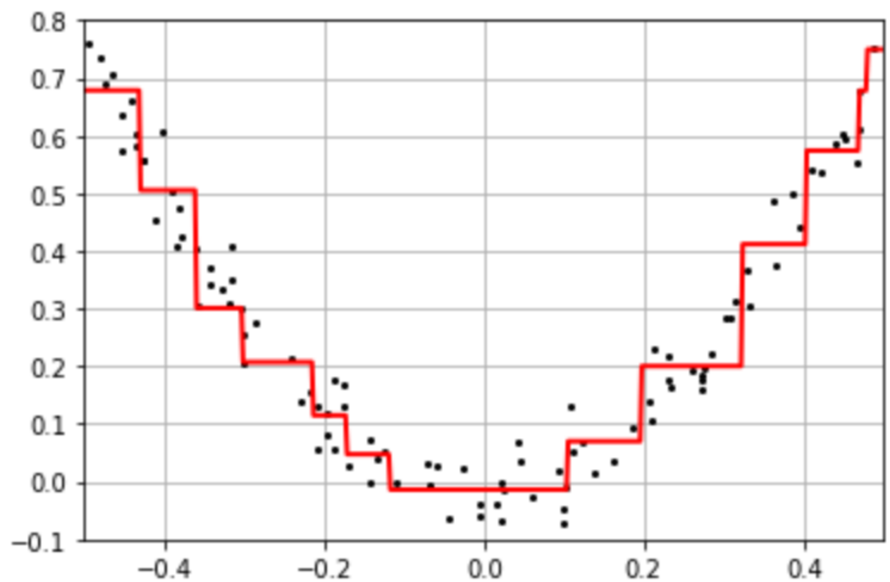
Gradient Boosting with 6 steps

```
tree.fit(X,e5)
red_line6 = red_line5 + tree.predict(x1)
```

```
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1,red_line6,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```

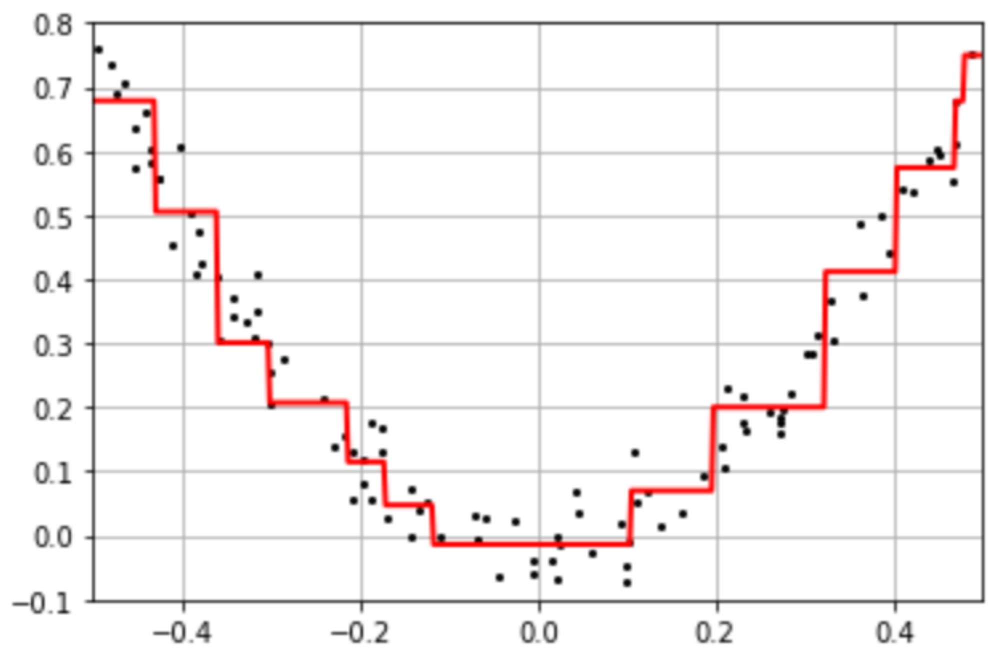



```
gbrt = GradientBoostingRegressor(max_depth=2,
                                 n_estimators=6,
                                 learning_rate=1.0,
                                 random_state=42)
gbrt.fit(X, y)
y_pred = gbrt.predict(x1)
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1, y_pred,"r-",linewidth=2)
plt.axis(axes)
plt.grid();
```



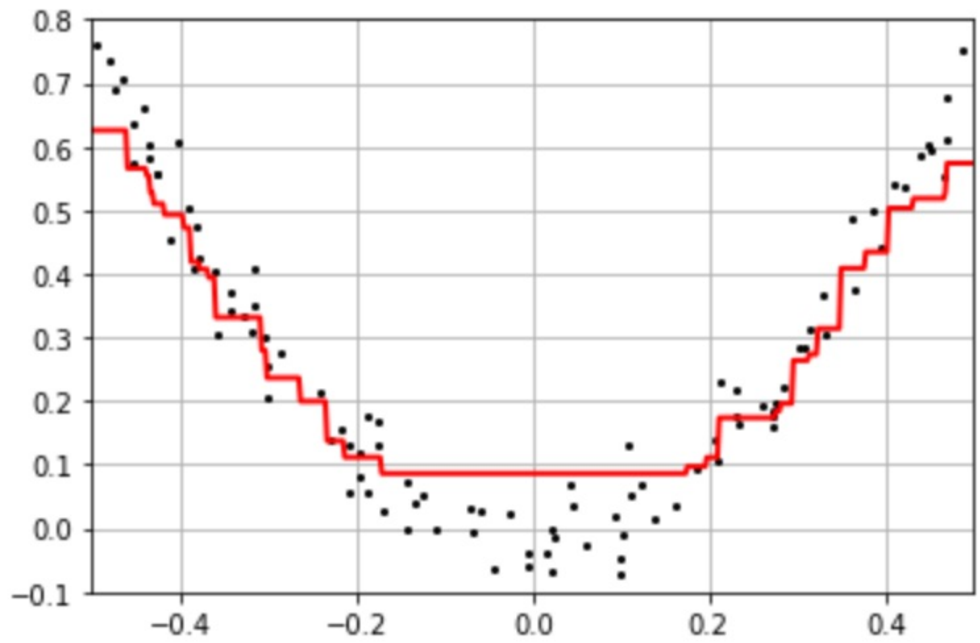
Example – Gradient Boosting with 6 steps

```
gbrt = GradientBoostingRegressor(max_depth=2,
                                 n_estimators=6,
                                 learning_rate=1.0,
                                 random_state=42)
gbrt.fit(X, y)
y_pred = gbrt.predict(x1)
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1, y_pred,"r-",linewidth=2)
plt.axis(axes)
plt.grid();
```



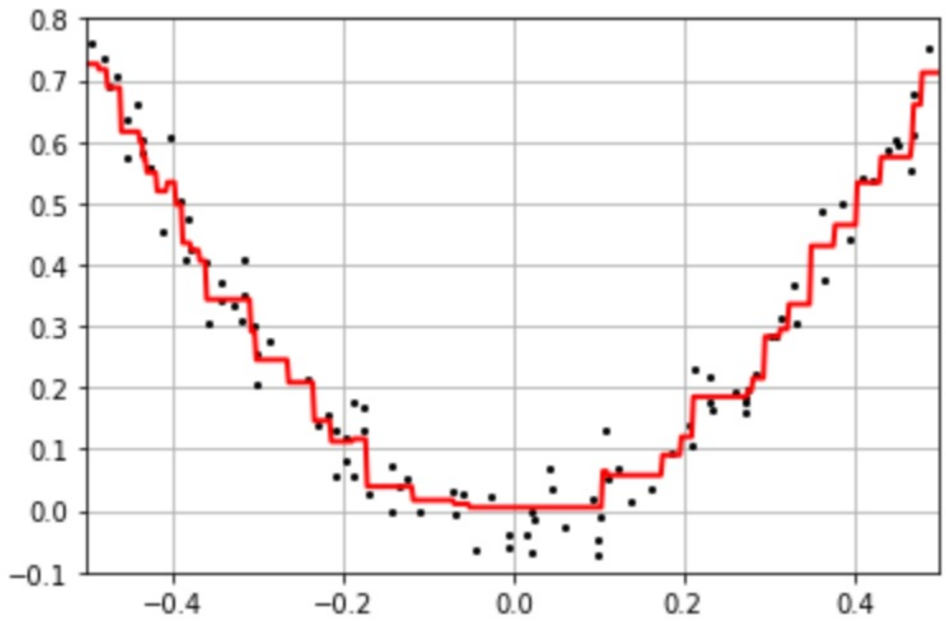
Example – Gradient Boosting with 20 steps

```
gbrt2 = GradientBoostingRegressor(max_depth=2,
                                  n_estimators=20,
                                  learning_rate=0.1,
                                  random_state=42)
gbrt2.fit(X, y);
y_pred = gbrt2.predict(x1)
plt.plot(X,y,"k.",markersize=4)
plt.plot(x1, y_pred,"r-",linewidth=2)
plt.axis(axes)
plt.grid();
```



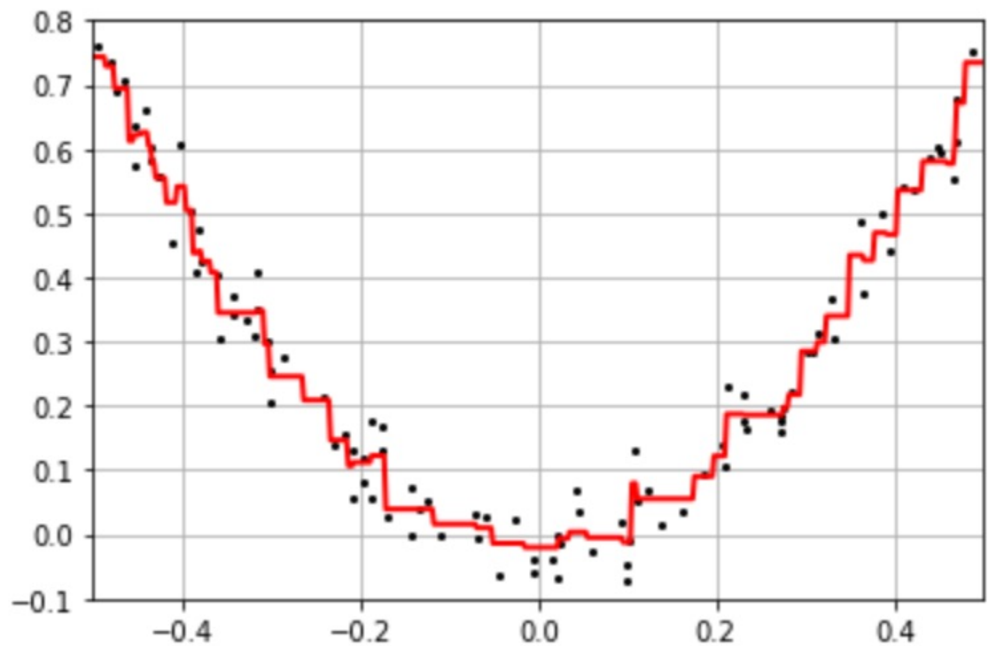
Example – Gradient Boosting with 50 steps

```
gbrt2 = GradientBoostingRegressor(max_depth=2,
                                  n_estimators=50,
                                  learning_rate=0.1,
                                  random_state=42)
gbrt2.fit(X, y);
y_pred = gbrt2.predict(x1)
plt.plot(X,y,"k.", label=None,markersize=4)
plt.plot(x1,y_pred,"r-",linewidth=2)
plt.axis(axes)
plt.grid();
```



Example – Gradient Boosting with 80 steps

```
gbrt2 = GradientBoostingRegressor(max_depth=2,
                                  n_estimators=80,
                                  learning_rate=0.1,
                                  random_state=42)
gbrt2.fit(X, y);
y_pred = gbrt2.predict(x1)
plt.plot(X,y,"k.",markersize=4)
plt.plot(x1, y_pred,"r-",linewidth=2)
plt.axis(axes)
plt.grid();
```



Example 2 – Ensembles on Classification

Example – Cancer dataset

The Cancer data from *sklearn* contains data from 569 patients with breast tumors. It is of interest to predict whether the tumor of a patient is malignant.

- Compare test AR of bagged trees with 25 and 500 trees. Find the test accuracy rate.
- Fit Random Forest models with 25 and 500 trees (and max_features = 4). Which predictors are most important?
- Fit 500 Gradient boosted trees with max_depth = 4, and α = 0.01, 0.20. Which predictors are found most important?

Example – libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

Example – Cancer dataset

```
cancer = load_breast_cancer()
```

```
list1 = list(cancer.feature_names)
```

```
df0 = pd.DataFrame(cancer.data,columns = list1)
df0[:5]
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...

5 rows × 30 columns

```
y = cancer.target
X = cancer.data
```

```
X.shape
```

```
(569, 30)
```

$p = 30$

Example 2 – Bagging Model

Decision Tree vs. Bagging

Single Tree

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
tree1 = DecisionTreeClassifier(max_depth = 4)
tree1.fit(X_train,y_train)
tree1.score(X_test,y_test)
```

```
0.9230769230769231
```

Bagging model (500 Trees)

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
bag_model = RandomForestClassifier(max_features = 30,
                                   max_depth = 4,
                                   n_estimators = 500,
                                   random_state=0)
bag_model.fit(X_train,y_train)
bag_model.score(X_test,y_test)
```

```
0.9300699300699301
```

Decision Tree vs. Bagging

Single Tree

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                   random_state = 0)
```

```
tree1 = DecisionTreeClassifier(max_depth = 4)
tree1.fit(X_train,y_train)
tree1.score(X_test,y_test)
```

```
0.9230769230769231
```

Bagging model (25 Trees)

```
bag_model2 = RandomForestClassifier(max_features = 30,
                                    max_depth = 4,
                                    n_estimators = 25,
                                     random_state=0)
bag_model2.fit(X_train,y_train)
bag_model2.score(X_test,y_test)
```

```
0.9230769230769231
```

Bagging – Find best n_estimators

GridSearchCV on n_estimators

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
model1 = RandomForestClassifier(max_depth=4, random_state=1)
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True,
                                    random_state = 1)
```

```
estimators = range(100,1000,100)
params = dict(n_estimators = estimators)
```

```
grid1 = GridSearchCV(model1,param_grid = params,cv = kfold)
grid1.fit(X_train,y_train);
```

Bagging – Find best n_estimators

GridSearchCV on n_estimators

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
model1 = RandomForestClassifier(max_depth=4, random_state=1)
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True,
                                    random_state = 1)
```

```
estimators = range(100,1000,100)
params = dict(n_estimators = estimators)
```

```
grid1 = GridSearchCV(model1,param_grid = params,cv = kfold)
grid1.fit(X_train,y_train);
```

```
# Best Validation Accuracy rate
grid1.best_score_
```

```
0.9647058823529413
```

```
grid1.best_params_
```

```
{'n_estimators': 500}
```

Test Accuracy rate

```
grid1.score(X_test,y_test)
```

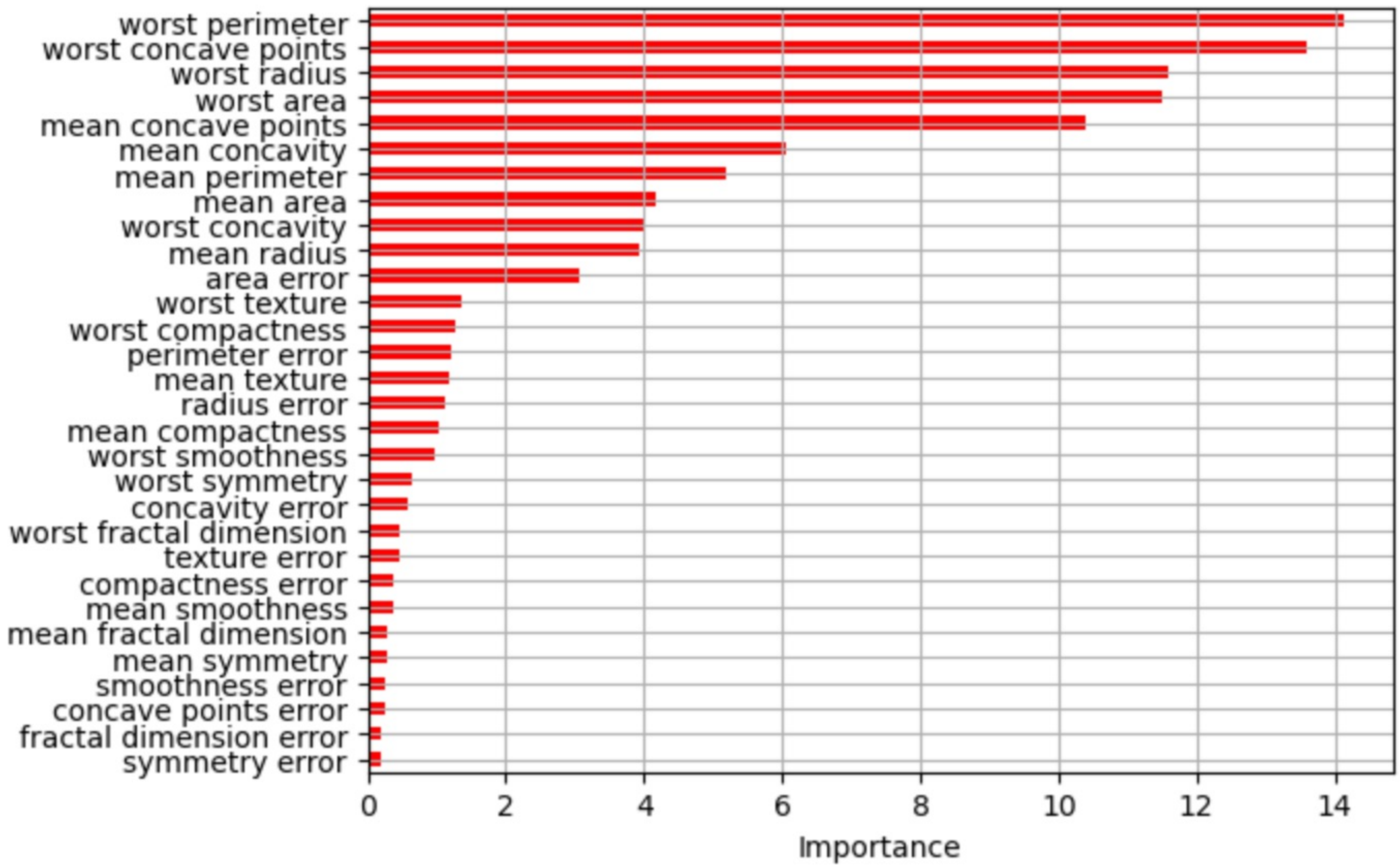
```
0.9440559440559441
```

Bagging – Feature Importance

```
df9 = pd.DataFrame(grid1.best_estimator_.feature_importances_*100,
                   index = cancer.feature_names,
                   columns = ['Importance'])
df9.columns.name = 'Feature'
df9 = df9.sort_values(by='Importance',axis=0,ascending = False)
```

Feature	Importance
worst perimeter	14.129173
worst concave points	13.576649
worst radius	11.579474
worst area	11.495622
mean concave points	10.388860
mean concavity	6.064789
mean perimeter	5.184134
mean area	4.173711
worst concavity	4.033597
mean radius	3.920080
area error	3.068875
worst texture	1.350636
worst compactness	1.269216
perimeter error	1.208892
mean texture	1.166103
radius error	1.117846
mean compactness	1.016690
worst smoothness	0.969281
worst symmetry	0.626952
concavity error	0.567973
worst fractal dimension	0.473491
texture error	0.446715
compactness error	0.383106
mean smoothness	0.359460
mean fractal dimension	0.280126
mean symmetry	0.269401
smoothness error	0.265304
concave points error	0.259535
fractal dimension error	0.177464
symmetry error	0.176844

Bagging – Feature Importance



Example 2 – Random Forest

Random Forest 500 vs 25 trees

RandomForest on 500 Trees (max_features = 5)

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                  random_state = 0)
```

```
forest = RandomForestClassifier(max_features = 5,n_estimators = 500,
                                max_depth = 4,random_state = 1)
forest.fit(X_train,y_train)
forest.score(X_test,y_test)
```

```
0.9440559440559441
```

RandomForest on 25 Trees

```
forest2 = RandomForestClassifier(max_features = 5,n_estimators = 25,
                                max_depth = 4,random_state = 1)
forest2.fit(X_train,y_train)
forest2.score(X_test,y_test)
```

```
0.9370629370629371
```

Random Forest vs Bagging (25 trees)

Bagging model (25 Trees)

```
bag_model2 = RandomForestClassifier(max_features = 30,
                                    max_depth = 4,
                                    n_estimators = 25,
                                     random_state=0)
bag_model2.fit(X_train,y_train)
bag_model2.score(X_test,y_test)
```

```
0.9230769230769231
```

RandomForest on 25 Trees

```
forest2 = RandomForestClassifier(max_features = 5,n_estimators = 25,
                                 max_depth = 4,random_state = 1)
forest2.fit(X_train,y_train)
forest2.score(X_test,y_test)
```

```
0.9370629370629371
```

Random Forest – Find best max_features

GridSearchCV on max_features

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
model3 = RandomForestClassifier(max_depth=4,n_estimators = 500,
                                random_state=1)
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True, random_state = 1)
```

```
features = range(1,31)
params = dict(max_features = features)
```

```
grid1 = GridSearchCV(model3,param_grid = params,cv = kfold)
grid1.fit(X_train,y_train);
```

Random Forest – Find best max_features

GridSearchCV on max_features

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
model3 = RandomForestClassifier(max_depth=4,n_estimators = 500,
                                random_state=1)
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True, random_state = 1)
```

```
features = range(1,31)
params = dict(max_features = features)
```

```
grid1 = GridSearchCV(model3,param_grid = params,cv = kfold)
grid1.fit(X_train,y_train);
```

```
# Best Validation Accuracy rate
grid1.best_score_
```

```
0.967086183310533б
```

```
grid1.best_params_
```

```
{'max_features': 14}
```

Test Accuracy rate

```
grid1.score(X_test,y_test)
```

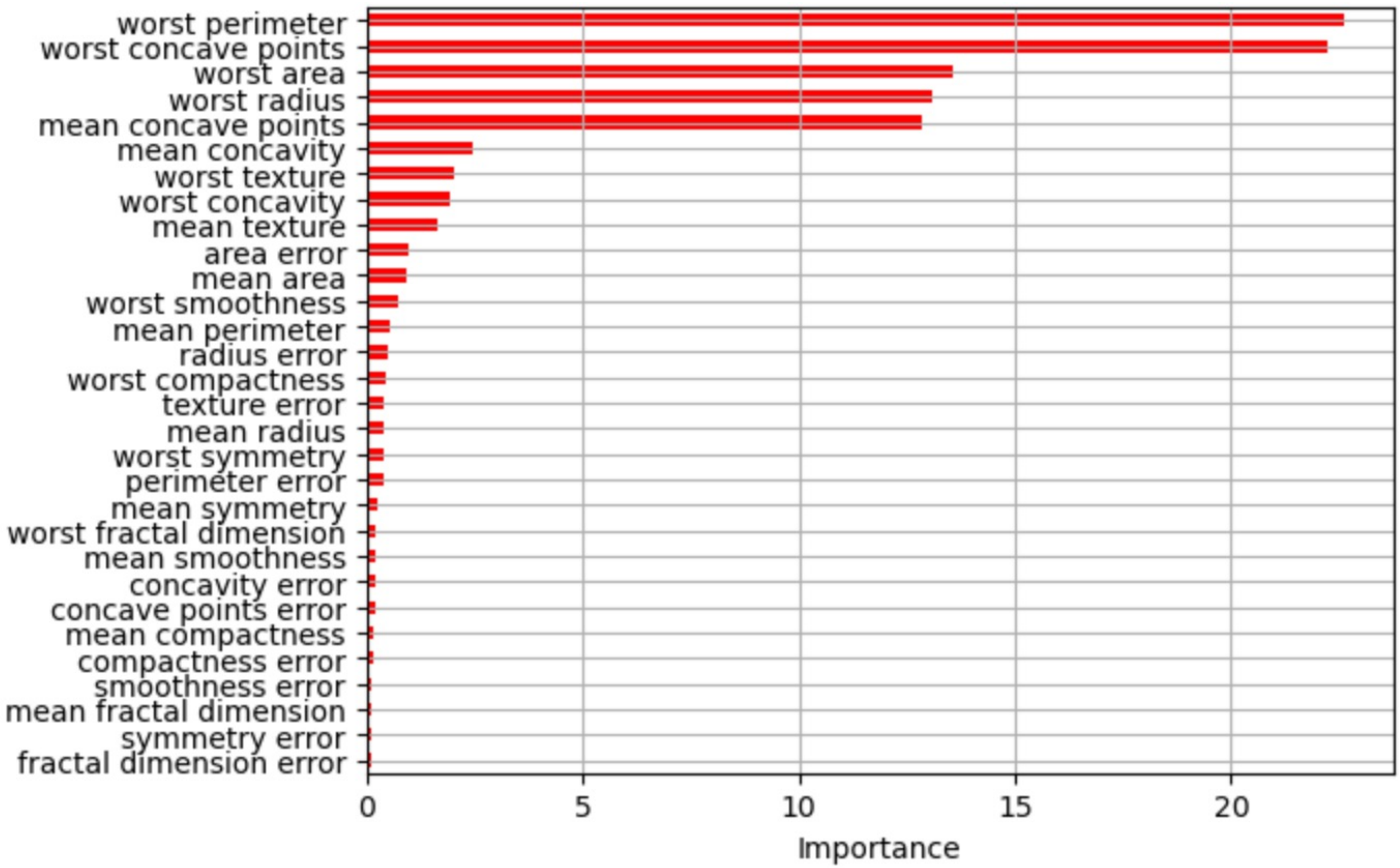
```
0.9440559440559441
```

Random Forest – Feature Importance

```python
df9 = pd.DataFrame(grid1.best_estimator_.feature_importances_*100,
                   index = cancer.feature_names,
                   columns = ['Importance'])
df9.columns.name = 'Feature'
df9 = df9.sort_values(by='Importance',axis=0,ascending = False)
```

Feature	Importance
worst perimeter	22.611353
worst concave points	22.232967
worst area	13.548250
worst radius	13.065276
mean concave points	12.862108
mean concavity	2.457333
worst texture	2.010432
worst concavity	1.912220
mean texture	1.644063
area error	0.999653
mean area	0.937828
worst smoothness	0.758059
mean perimeter	0.569221
radius error	0.511625
worst compactness	0.450272
texture error	0.397110
mean radius	0.392667
worst symmetry	0.387934
perimeter error	0.382765
mean symmetry	0.263526
worst fractal dimension	0.225117
mean smoothness	0.202968
concavity error	0.197187
concave points error	0.193128
mean compactness	0.149347
compactness error	0.144200
smoothness error	0.133866
mean fractal dimension	0.131247
symmetry error	0.125688
fractal dimension error	0.102590

Random Forest – Feature Importance



Example 2 – Gradient Boosting

Gradient Boosting Classification Trees

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state=0)
```

learning rate to 0.60

```
model2 = GradientBoostingClassifier(n_estimators = 25,
                                    learning_rate = 0.6,
                                    max_depth = 4, random_state =1)
model2.fit(X_train,y_train)
model2.score(X_test,y_test)
```

```
0.9370629370629371
```

learning rate to 0.10

```
model2 = GradientBoostingClassifier(n_estimators = 25,
                                    learning_rate = 0.1,
                                    max_depth = 4, random_state =1)
model2.fit(X_train,y_train)
model2.score(X_test,y_test)
```

```
0.951048951048951
```

Gradient Boosting – Find best learning_rate

GridSearchCV on learning_rate

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
model3 = GradientBoostingClassifier(n_estimators = 25,
                                    max_depth = 4, random_state =1)
```

```
lrates = np.linspace(0.0,1.0,20)
lrates
```

```
array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True, random_state = 1)
```

```
params = dict(learning_rate = lrates)
```

```
grid1 = GridSearchCV(model3,param_grid = params,cv = kfold)
grid1.fit(X_train,y_train);
```

Gradient Boosting – Find best learning_rate

GridSearchCV on learning_rate

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 random_state = 0)
```

```
model3 = GradientBoostingClassifier(n_estimators = 25,
                                    max_depth = 4, random_state =1)
```

```
lrates = np.linspace(0.0,1.0,20)
lrates
```

```
array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True, random_state = 1)
```

```
params = dict(learning_rate = lrates)
```

```
grid1 = GridSearchCV(model3,param_grid = params,cv = kfold)
grid1.fit(X_train,y_train);
```

```
# Best Validation Accuracy rate
grid1.best_score_
```

```
0.9671135430916553
```

```
grid1.best_params_
```

```
{'learning_rate': 0.63157894736
```

Test Accuracy rate

```
grid1.score(X_test,y_test)
```

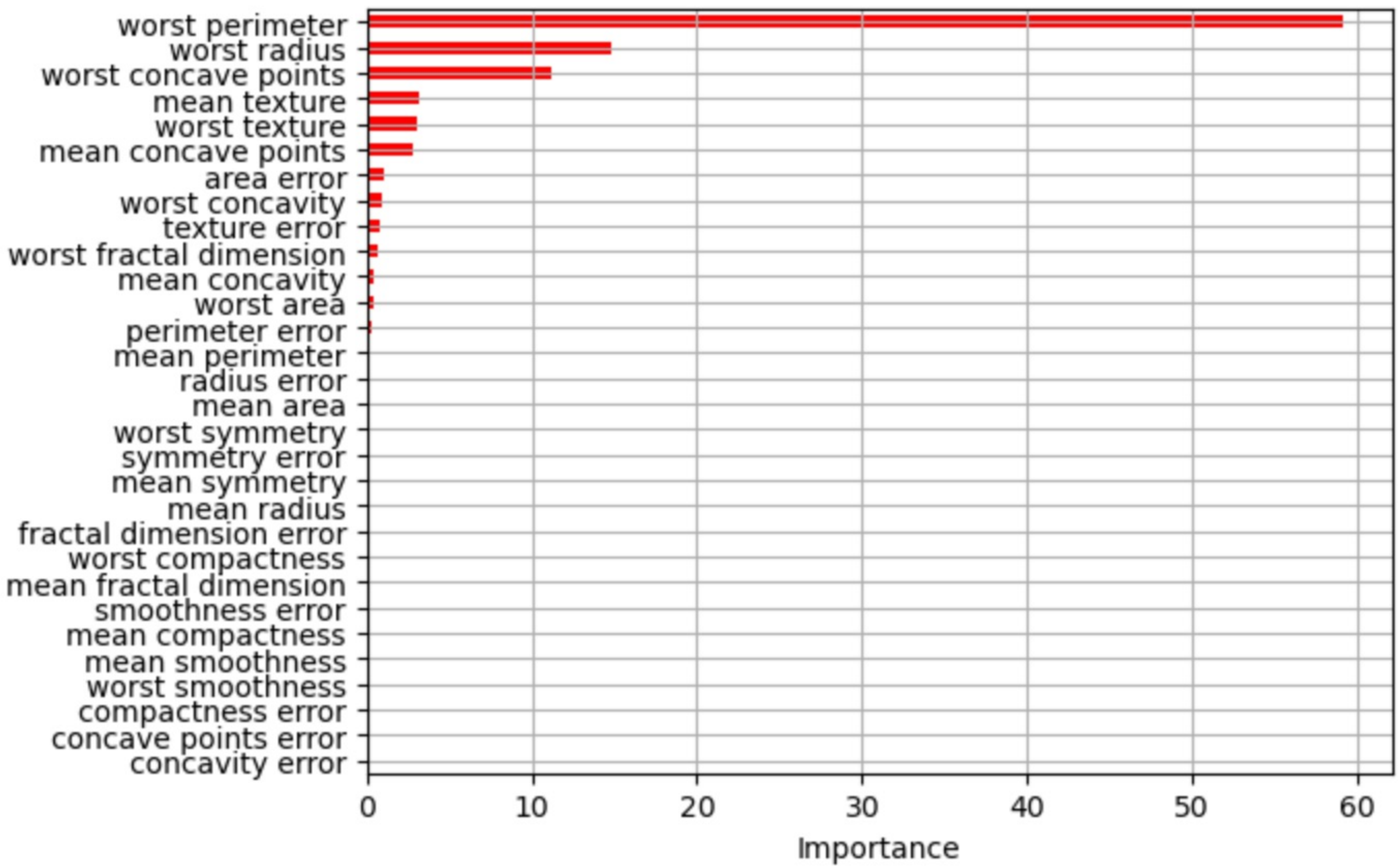
```
0.9300699300699301
```

Gradient Boosting– Feature Importance

```
Importances = grid1.best_estimator_.feature_importances_*100
df9 = pd.DataFrame({'Importance':Importances},
                   index = cancer.feature_names)
df9 = df9.sort_values(by = 'Importance',axis = 0,
                      ascending = False)
df9
```

	Importance
worst perimeter	59.099340
worst radius	14.847550
worst concave points	11.161615
mean texture	3.231188
worst texture	3.098257
mean concave points	2.843215
area error	1.067154
worst concavity	0.949111
texture error	0.799159
worst fractal dimension	0.725911
mean concavity	0.453931
worst area	0.452600
perimeter error	0.332176
mean perimeter	0.217580
radius error	0.204401
mean area	0.165851
worst symmetry	0.148986
symmetry error	0.089275
mean symmetry	0.062010
mean radius	0.020612
fractal dimension error	0.013000
worst compactness	0.008984
mean fractal dimension	0.003803
smoothness error	0.002376
mean compactness	0.000650
mean smoothness	0.000562
worst smoothness	0.000505
compactness error	0.000130
concave points error	0.000067
concavity error	0.000003

Gradient Boosting – Feature Importance



GridSearchCV - Tuning 2 hyperparameters

```
# Consider 6 values for each parameter
params = {'learning_rate': np.linspace(0.2,0.7,6),
              'max_features': list(range(3,9))}
params
```

```
{'learning_rate': array([0.2, 0.3, 0.4, 0.5, 0.6, 0.7]),
 'max_features': [3, 4, 5, 6, 7, 8]}
```

```
grid2 = GridSearchCV(model3, param_grid = params,cv = kfold)
grid2.fit(X_train,y_train);
```

```
# Best Validation Accuracy rate
grid2.best_score_
```

```
0.974172366210671
```

```
grid2.best_params_
```

```
{'learning_rate': 0.6, 'max_features': 6}
```

GridSearchCV on 2 hyperparameters

- cv_results_ has the accuracy rates of each fold and their average in column mean_test_score

```
# store the results into dataframe
results = pd.DataFrame(grid2.cv_results_)
results.dtypes
```

```
mean_fit_time            float64
std_fit_time             float64
mean_score_time          float64
std_score_time           float64
param_learning_rate       object
param_max_features        object
params                    object
split0_test_score        float64
split1_test_score        float64
split2_test_score        float64
split3_test_score        float64
split4_test_score        float64
mean_test_score          float64
std_test_score           float64
rank_test_score            int32
```

GridSearchCV on 2 hyperparameters

- cv_results_ has the accuracy rates of each fold and their average in column mean_test_score

```
# store the results into dataframe
results = pd.DataFrame(grid2.cv_results_)
results.dtypes
```

```
mean_fit_time              float64
std_fit_time               float64
mean_score_time            float64
std_score_time             float64
param_learning_rate         object
param_max_features          object
params                      object
split0_test_score          float64
split1_test_score          float64
split2_test_score          float64
split3_test_score          float64
split4_test_score          float64
mean_test_score            float64
std_test_score             float64
rank_test_score              int32
```

GridSearchCV on 2 hyperparameters

```
list1 = list([4,5,12])
df9 = results.iloc[:,list1].copy()
df9[:13]
```

Validation
Accuracy rate

	param_learning_rate	param_max_features	mean_test_score
0	0.02	3	0.941423
1	0.02	4	0.953160
2	0.02	5	0.950807
3	0.02	6	0.950752
4	0.02	7	0.941341
5	0.02	8	0.948372
6	0.025	3	0.943776
7	0.025	4	0.953160
8	0.025	5	0.953133
9	0.025	6	0.953105
10	0.025	7	0.948372
11	0.025	8	0.955458
12	0.03	3	0.946101

GridSearchCV on 2 hyperparameters

```
df1 = df9.pivot_table('mean_test_score',
                      columns = 'param_learning_rate',
                      index = 'param_max_features')
df1
```

param_learning_rate	0.2	0.3	0.4	0.5	0.6	0.7
param_max_features						
3	0.957674	0.962408	0.969412	0.967059	0.974118	0.967086
4	0.960055	0.955376	0.967086	0.971792	0.960055	0.955376
5	0.960027	0.967086	0.964761	0.962408	0.962435	0.967059
6	0.957729	0.962435	0.960055	0.967114	0.974172	0.967086
7	0.964733	0.960109	0.971765	0.969439	0.964733	0.969466
8	0.962380	0.969412	0.962408	0.955349	0.967114	0.964761

GridSearchCV on 2 hyperparameters

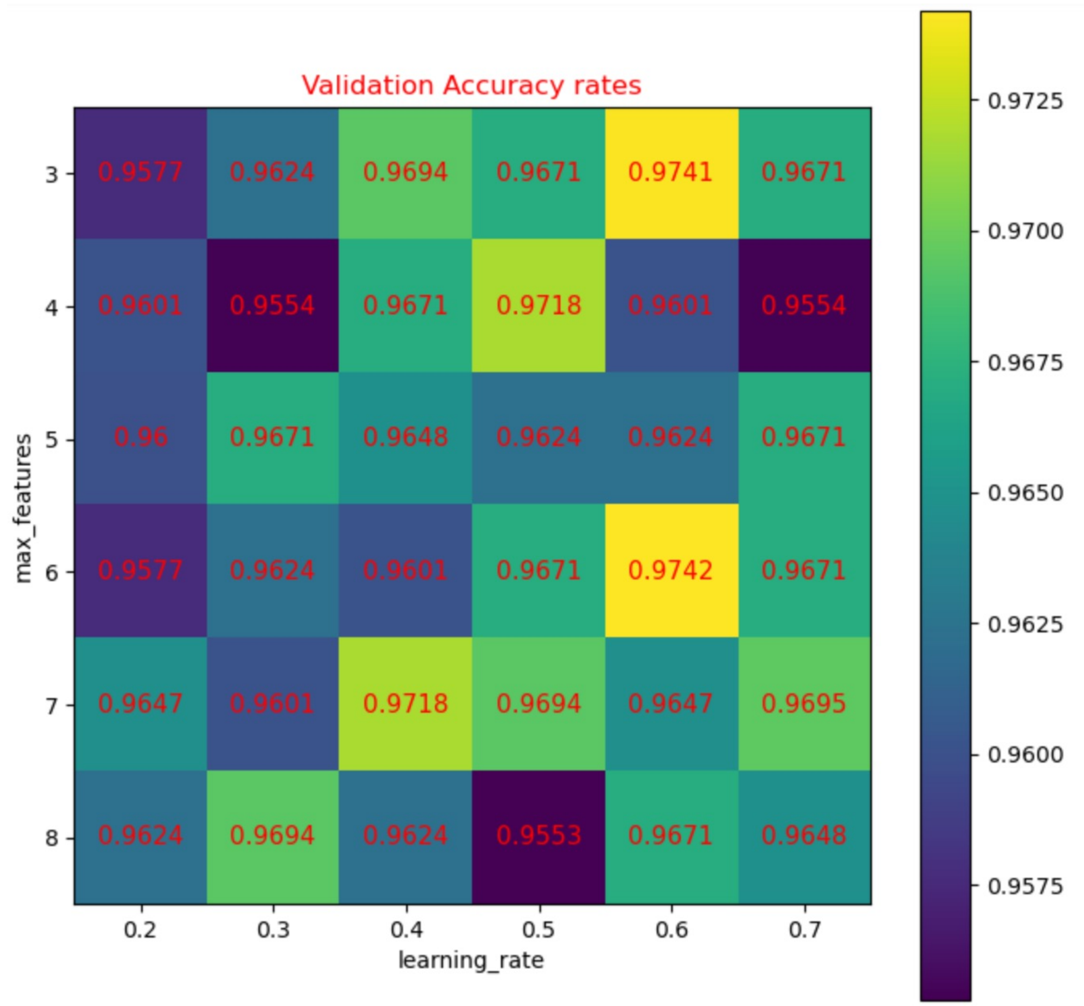
```
# transform df1 dataframe to numpy array
arates = df1.values
arates = np.round(arates,4)
arates
```

```
array([[0.9577, 0.9624, 0.9694, 0.9671, 0.9741, 0.9671],
       [0.9601, 0.9554, 0.9671, 0.9718, 0.9601, 0.9554],
       [0.96  , 0.9671, 0.9648, 0.9624, 0.9624, 0.9671],
       [0.9577, 0.9624, 0.9601, 0.9671, 0.9742, 0.9671],
       [0.9647, 0.9601, 0.9718, 0.9694, 0.9647, 0.9695],
       [0.9624, 0.9694, 0.9624, 0.9553, 0.9671, 0.9648]])
```

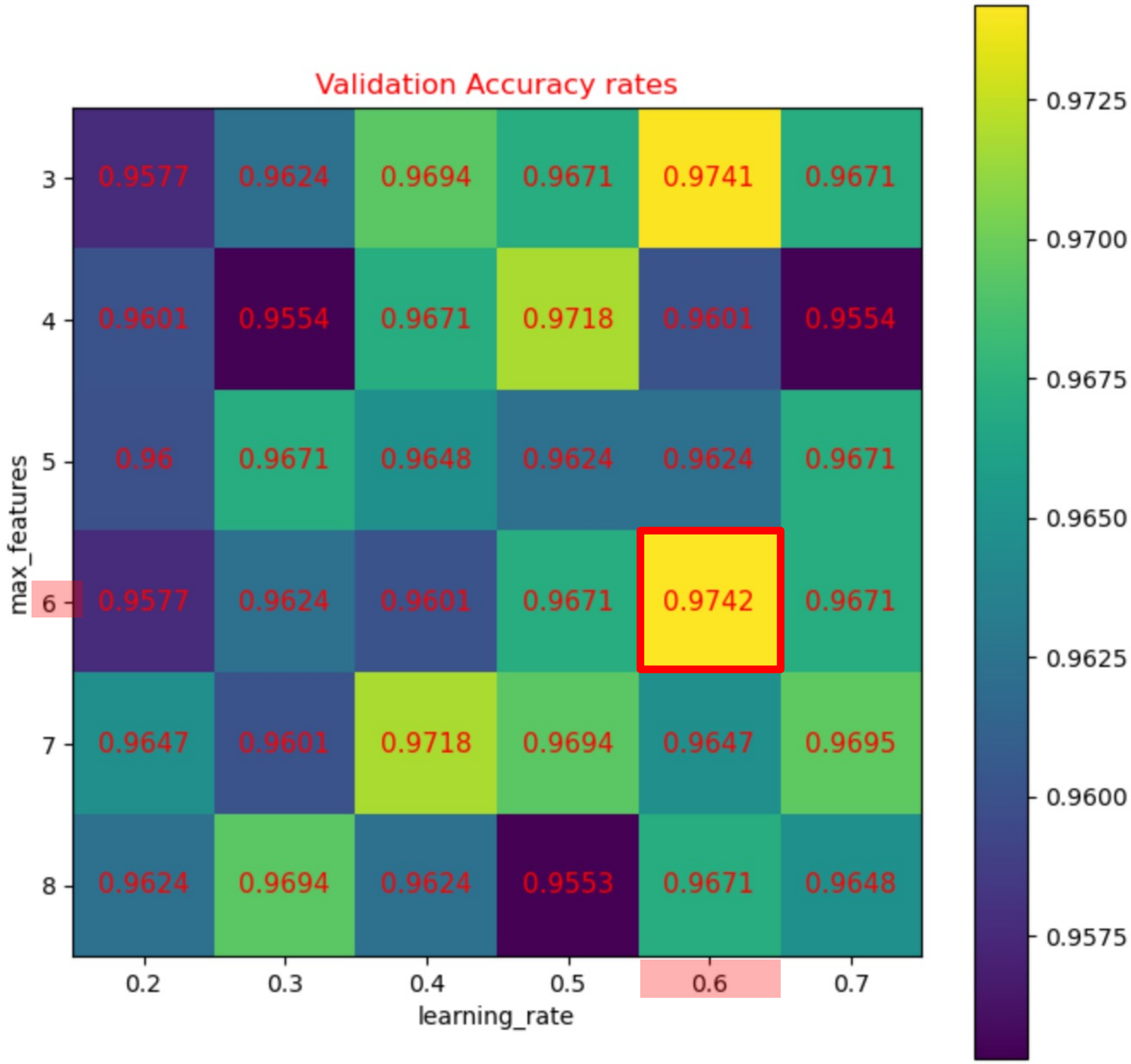
```
plt.figure(figsize=(8,8))
plt.xticks(range(6),df1.columns)
plt.yticks(range(6),df1.index)
plt.ylabel('max_features')
plt.xlabel('learning_rate')
plt.title('Validation Accuracy rates',c='r')
plt.imshow(arates)
for i in range(6):
    for j in range(6):
        text = plt.text(j,i,arates[i,j],
                        ha="center",
                        va="center",
                        color="r",
                        size = 11)
plt.colorbar();
```

GridSearchCV results on a Heatmap

```
plt.figure(figsize=(8,8))
plt.xticks(range(6),df1.columns)
plt.yticks(range(6),df1.index)
plt.ylabel('max_features')
plt.xlabel('learning_rate')
plt.title('Accuracy rates',c='r')
plt.imshow(arates)
for i in range(6):
    for j in range(6):
        text = plt.text(j,i,arates[i,j],
                        ha="center",
                        va="center",
                        color="r",
                        size = 13)
plt.colorbar();
```

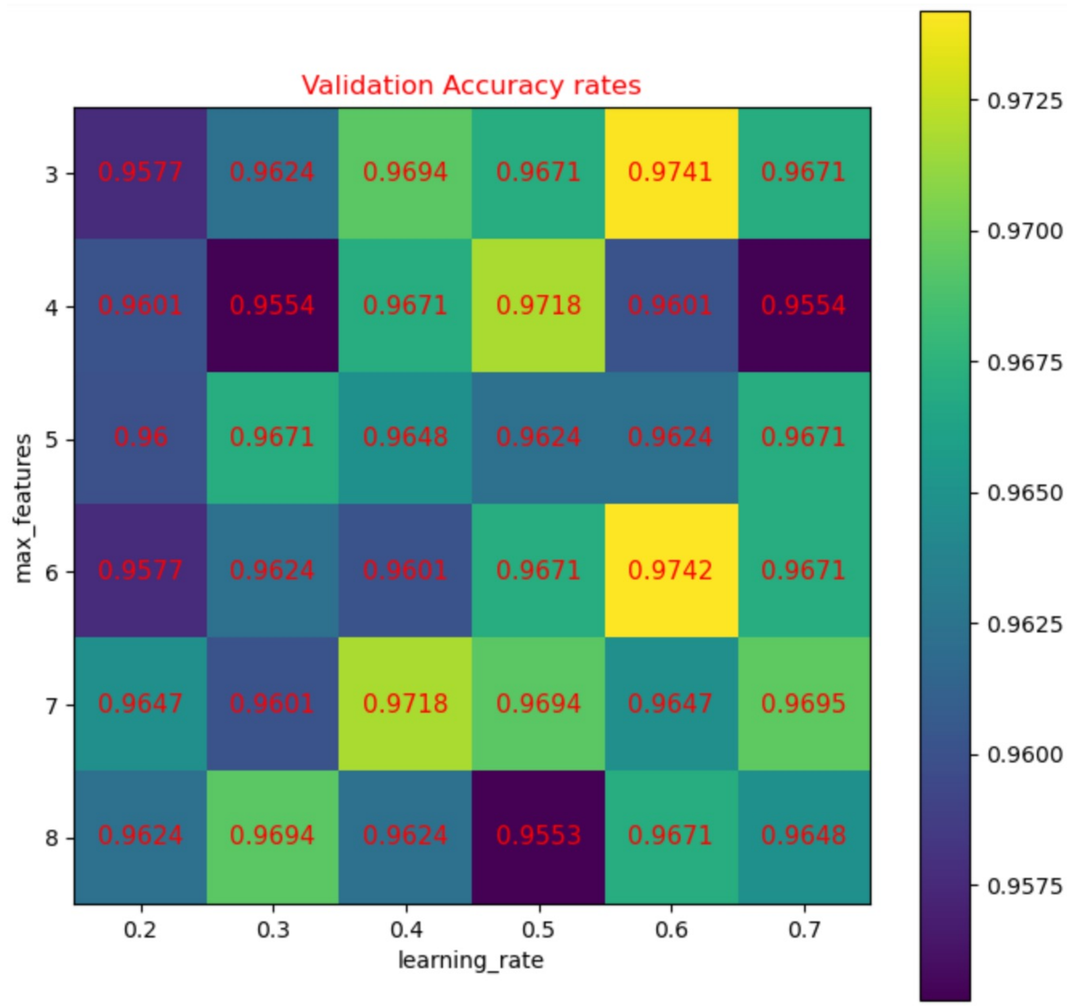


GridSearchCV results on a Heatmap



GridSearchCV results on a Heatmap

```python
plt.figure(figsize=(8,8))
plt.xticks(range(6),df1.columns)
plt.yticks(range(6),df1.index)
plt.ylabel('max_features')
plt.xlabel('learning_rate')
plt.title('Accuracy rates',c='r')
plt.imshow(arates)
for i in range(6):
    for j in range(6):
        text = plt.text(j,i,arates[i,j],
                        ha="center",
                        va="center",
                        color="r",
                        size = 13)
plt.colorbar();
```



Test Accuracy rate

```python
grid2.score(X_test,y_test)
```

```
0.9440559440559441
```

Gradient Boosting – Feature Importance