ENSEMBLES

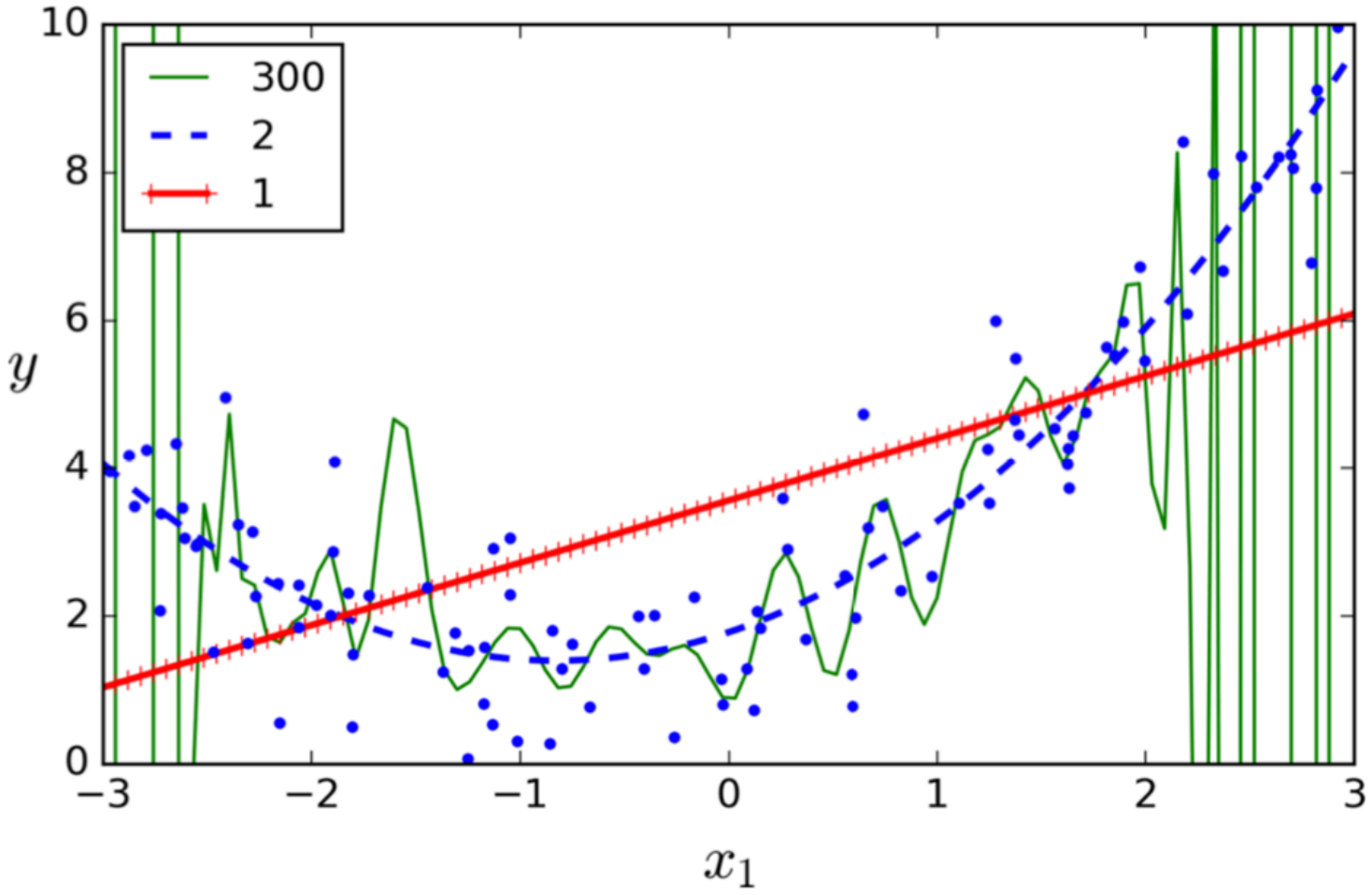
Outline

- Definition of Ensembles
- Bias-Variance Tradeoff
- Bootstrap samples
- Ensembles
- Bagging
- Random Forest
- Gradient Boosting

Ensembles

- Methods combining multiple machine learning models to create low-bias, low-variance, prediction models
- What are low-bias and low-variance models?

Bias – Variance Tradeoff



Bias – Variance Tradeoff

The model error (MSE, error rate) is the sum of 3 parts

- Bias
- Variance
- Random error

Bias – Variance Tradeoff

The model error (MSE, error rate) is the sum of 3 parts

- Bias

Model is unable to follow the structural variation underlying the data. (i.e., a linear model used with nonlinear data).

Models with high-bias are likely to underfit the data.

Bias – Variance Tradeoff

The model error (MSE, error rate) is the sum of 3 parts

- Bias

  Model is unable to follow the structural variation underlying the data. (i.e., a linear model used with nonlinear data).

  Models with high-bias are likely to underfit the data.

- Variance

  Model is highly sensitive, trying to capture random variations in the data. (i.e., a high-degree polynomial model).

  Models with high-variance are likely to overfit the data.

Bias – Variance Tradeoff

The model error (MSE, error rate) is the sum of 3 parts

- Random error

  Data portion that cannot be predicted

We look for models that do not underfit or overfit the data

These are low-bias, low-variance models

Model Complexity

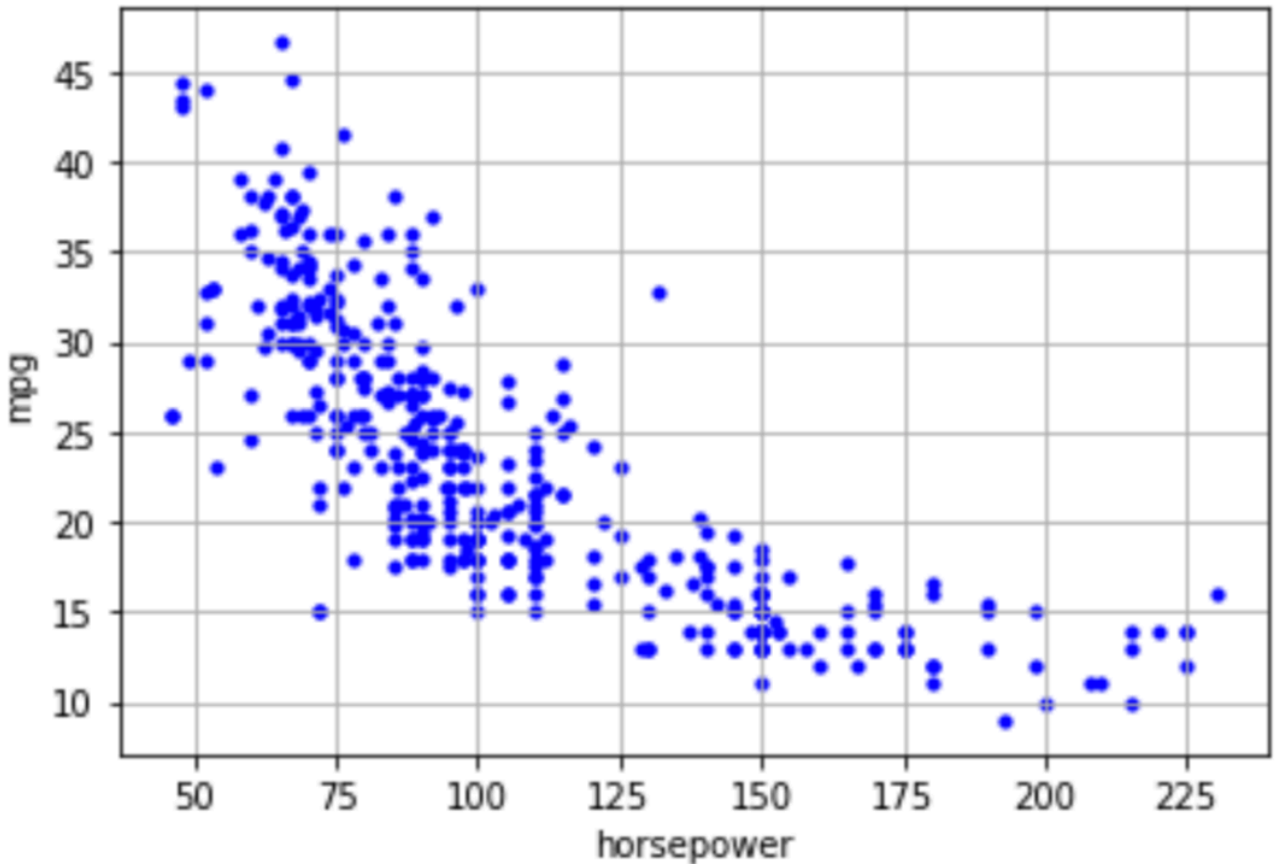
- Model Complexity is given by the number of predictor columns in the model.
- Adding new features or interactions, and polynomial terms increases the model complexity

Model Complexity

- Increasing model complexity usually increases the model's variance and reduces its bias.
- Decreasing model complexity usually increases the model's bias and reduces its variance.
- This relation between the error portions of a model is called the Bias – Variance Tradeoff

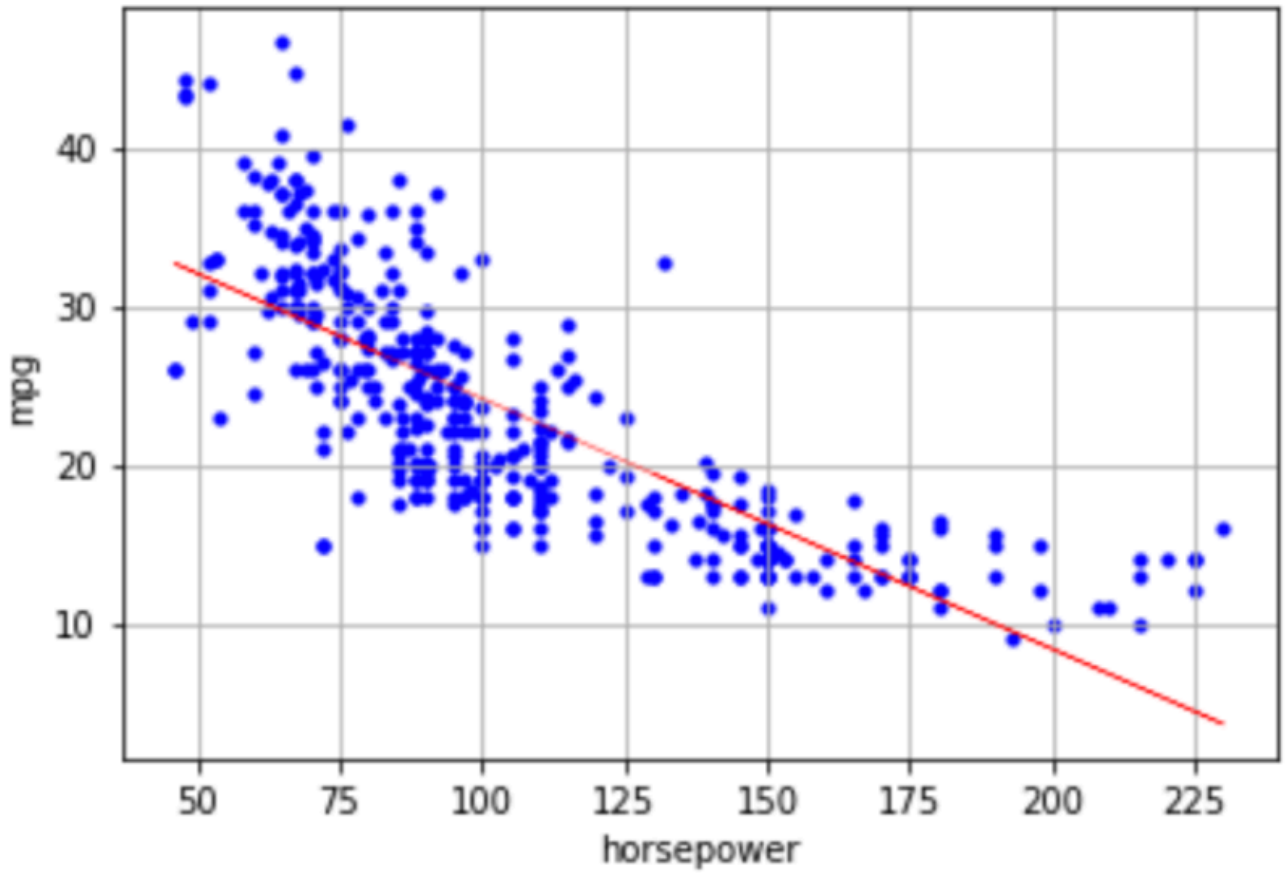
Bias – Variance Tradeoff

Non-linear relation



Bias – Variance Tradeoff

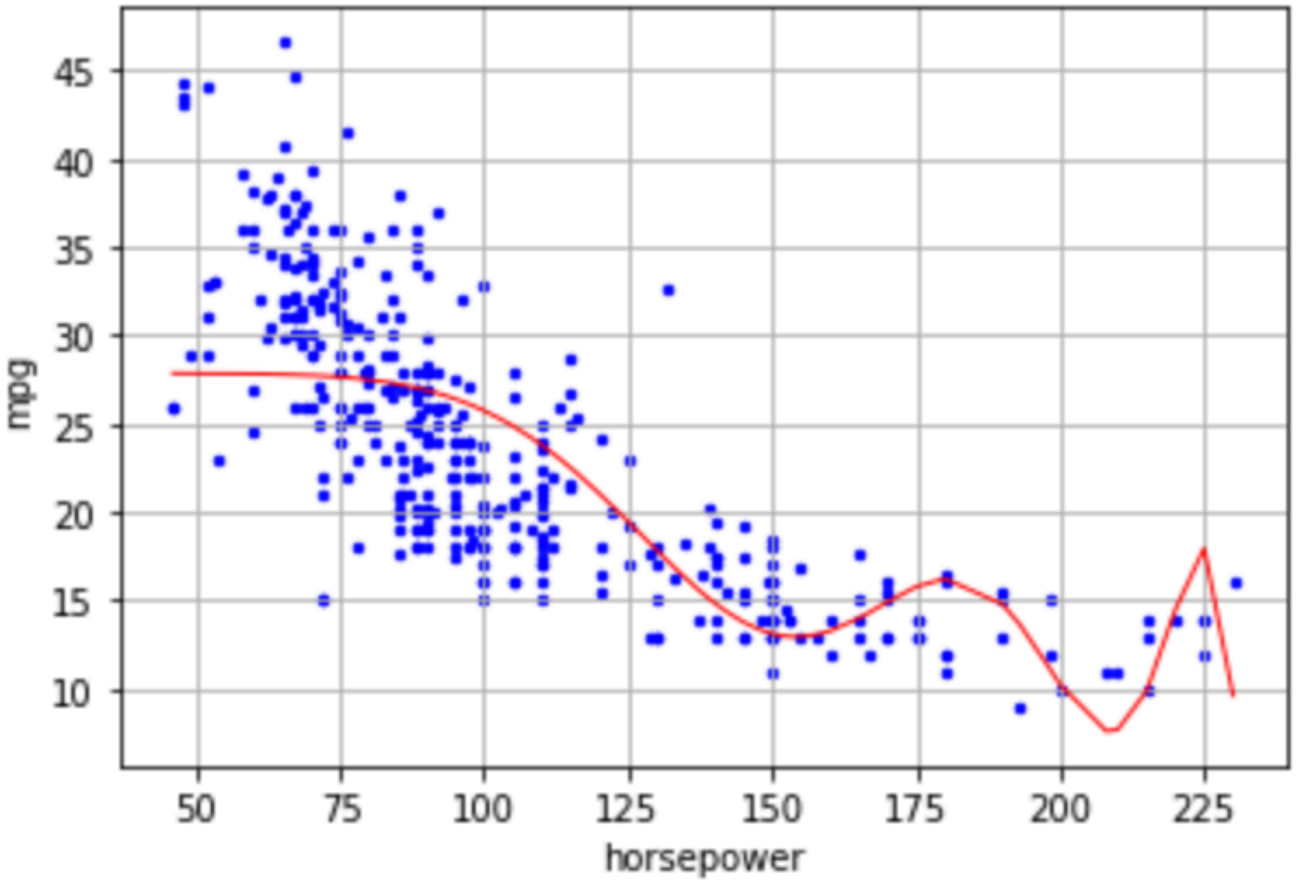
Model with High-Bias (does not follow underlying pattern)



underfits the data

Bias – Variance Tradeoff

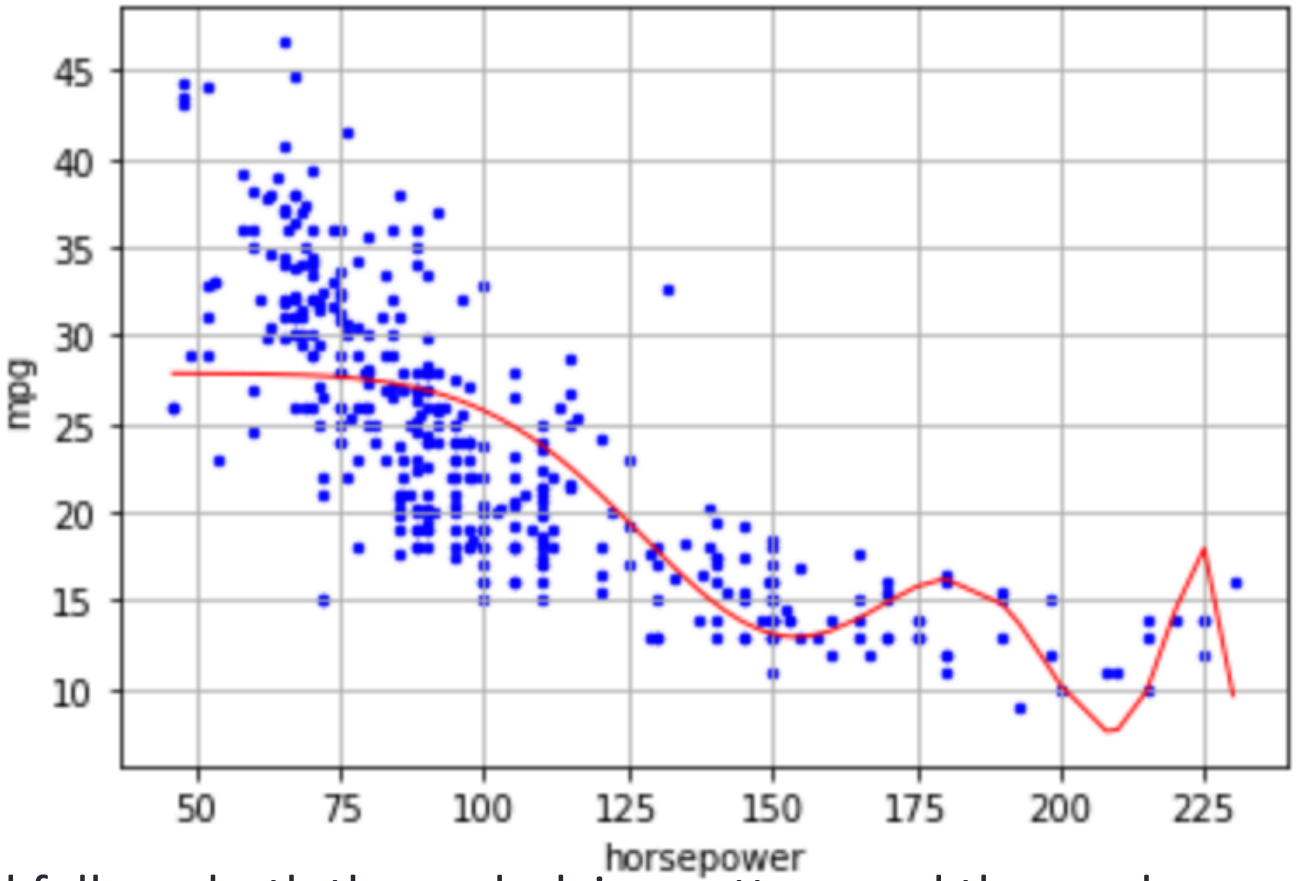
Model with High-Variance (high-degree polynomial)



overfits the data

Bias – Variance Tradeoff

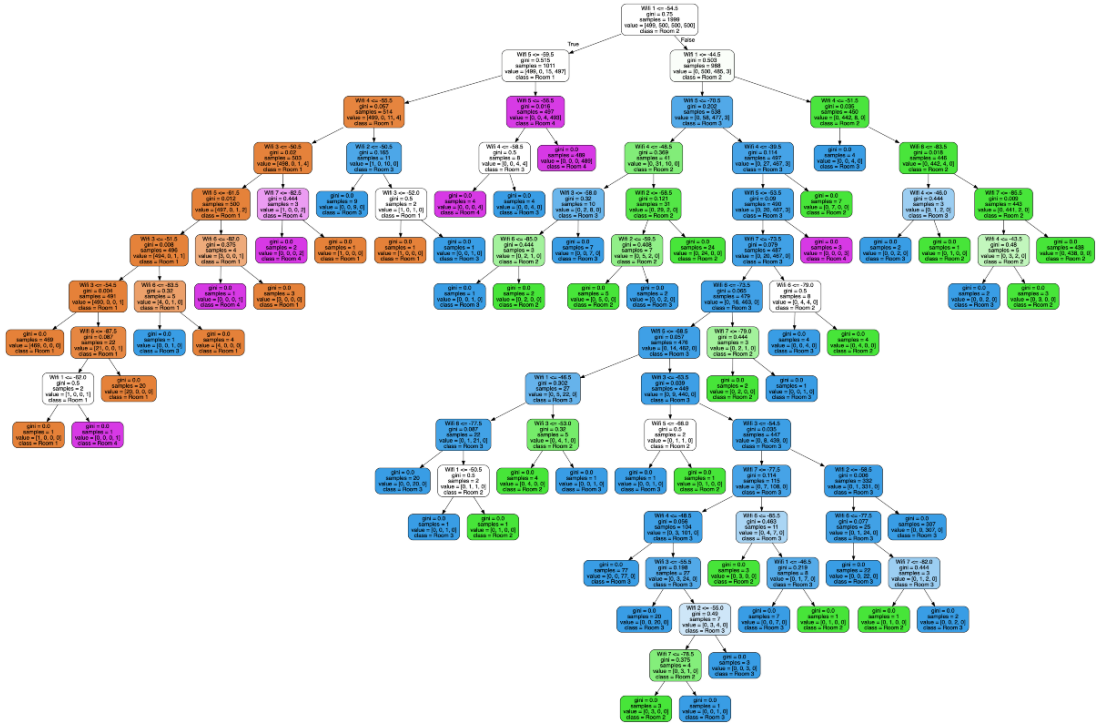
Model with High-Variance (high-degree polynomial)



the model follows both the underlying pattern and the random errors in the data

Tree complexity

- A decision Tree with a large depth is a high variance model
- To avoid it, we may choose small depth
- But then predictions may become less accurate



Balancing Tree Complexity

- Ensembles are combination of models (trees)
- They tend to be low-bias, low-variance models

ENSEMBLES

Ensembles

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

BAGGING

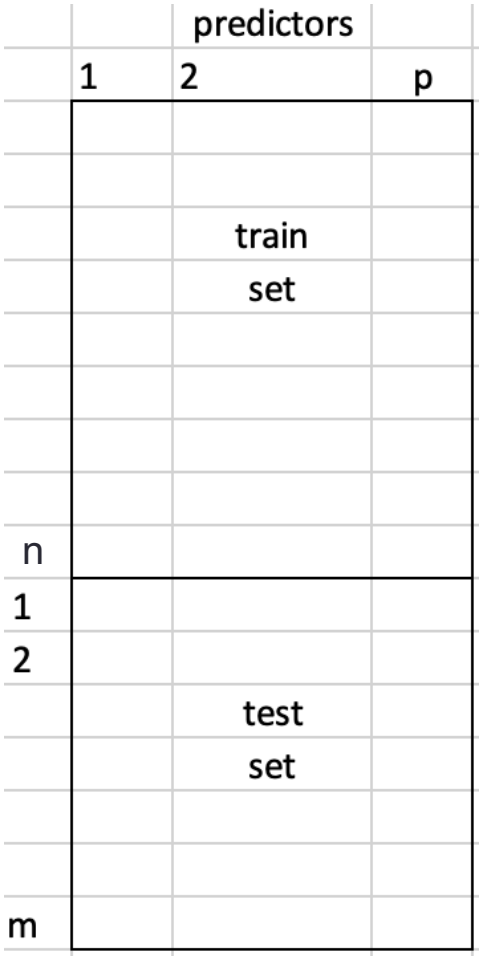
Bootstrap samples (from dataframes)

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

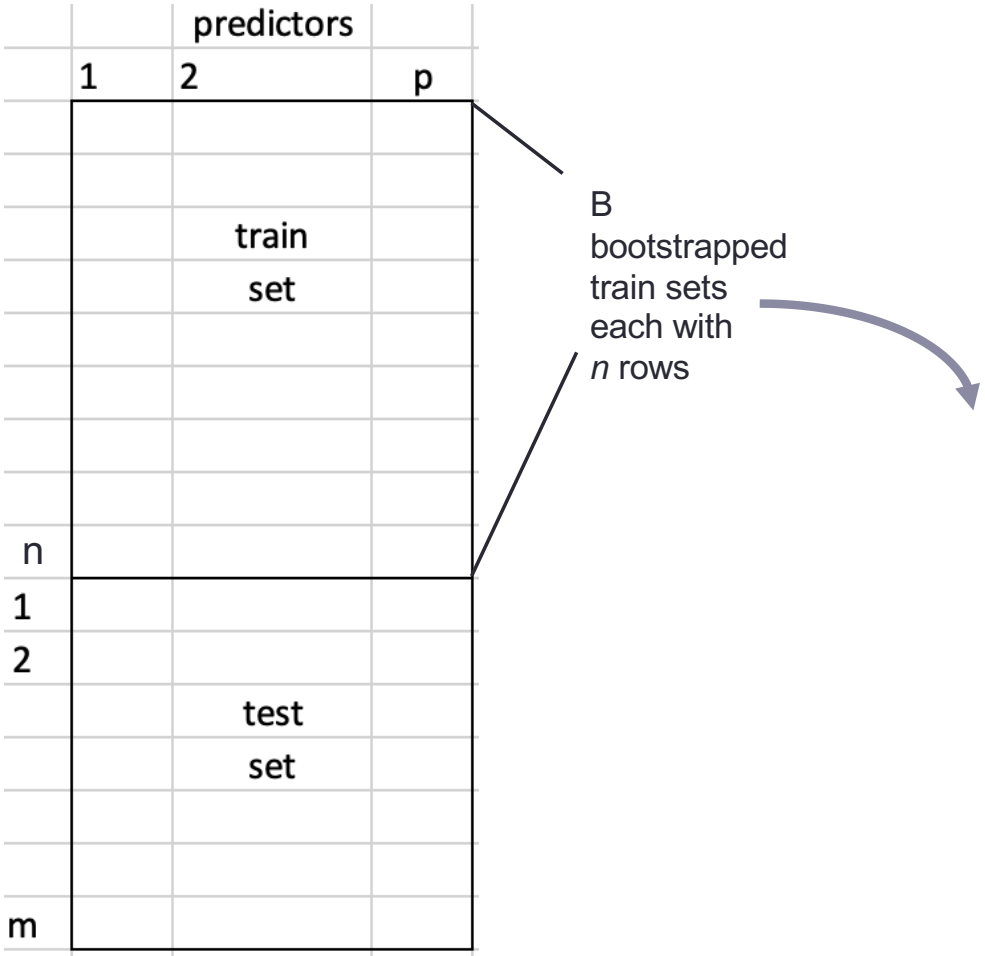
Bagging

- Individual trees suffer from high variance
- That is, if a dataset is randomly split into 2 sets and a tree is fit to each half, the predictions may be *very* different
- On the other hand, a low-variance model would yield predictions that are not much different
- Averaging trees predictions help avoid high-variance models

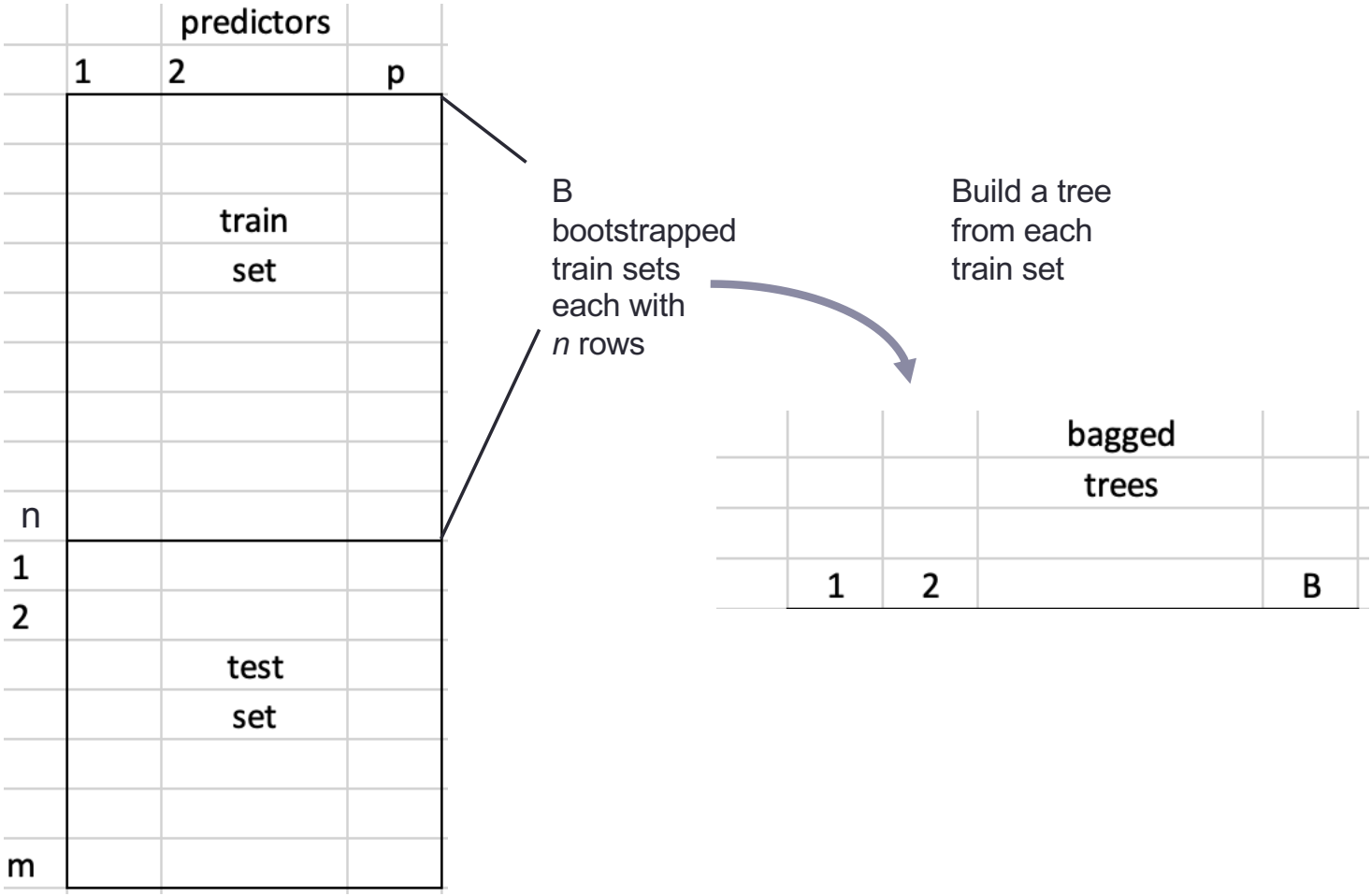
Bagging for **Regression Trees**



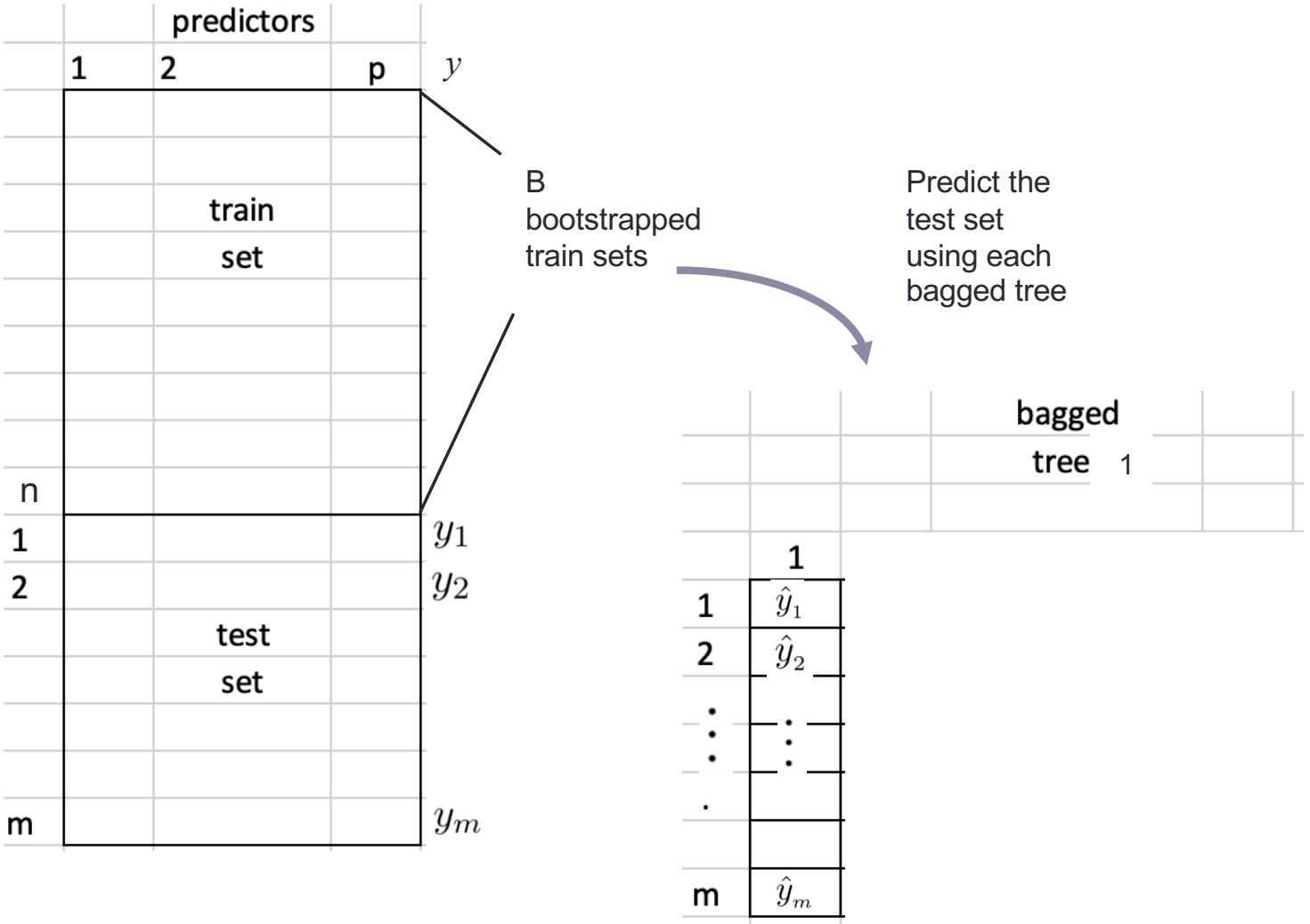
Bagging for Regression Trees



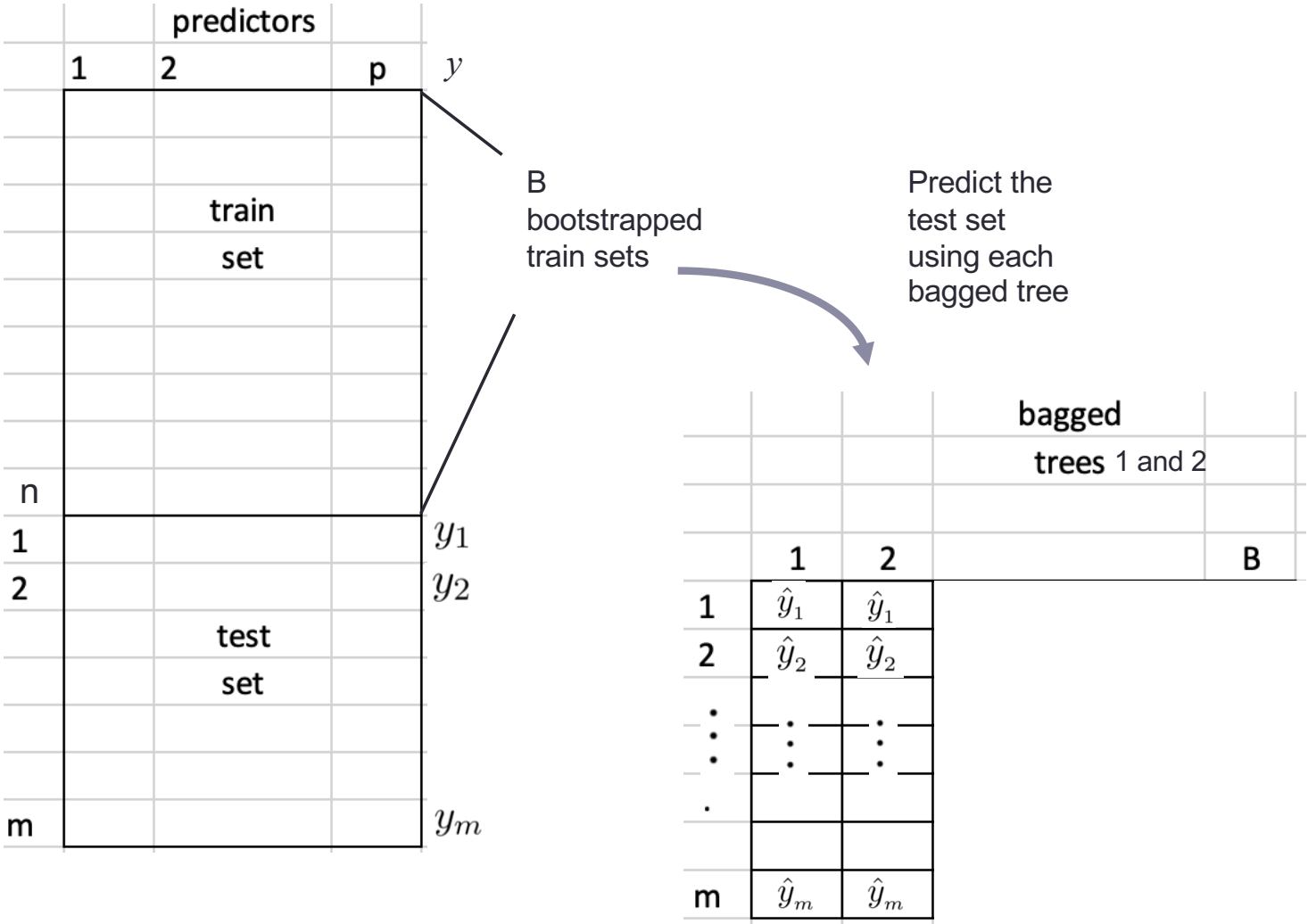
Bagging for Regression Trees



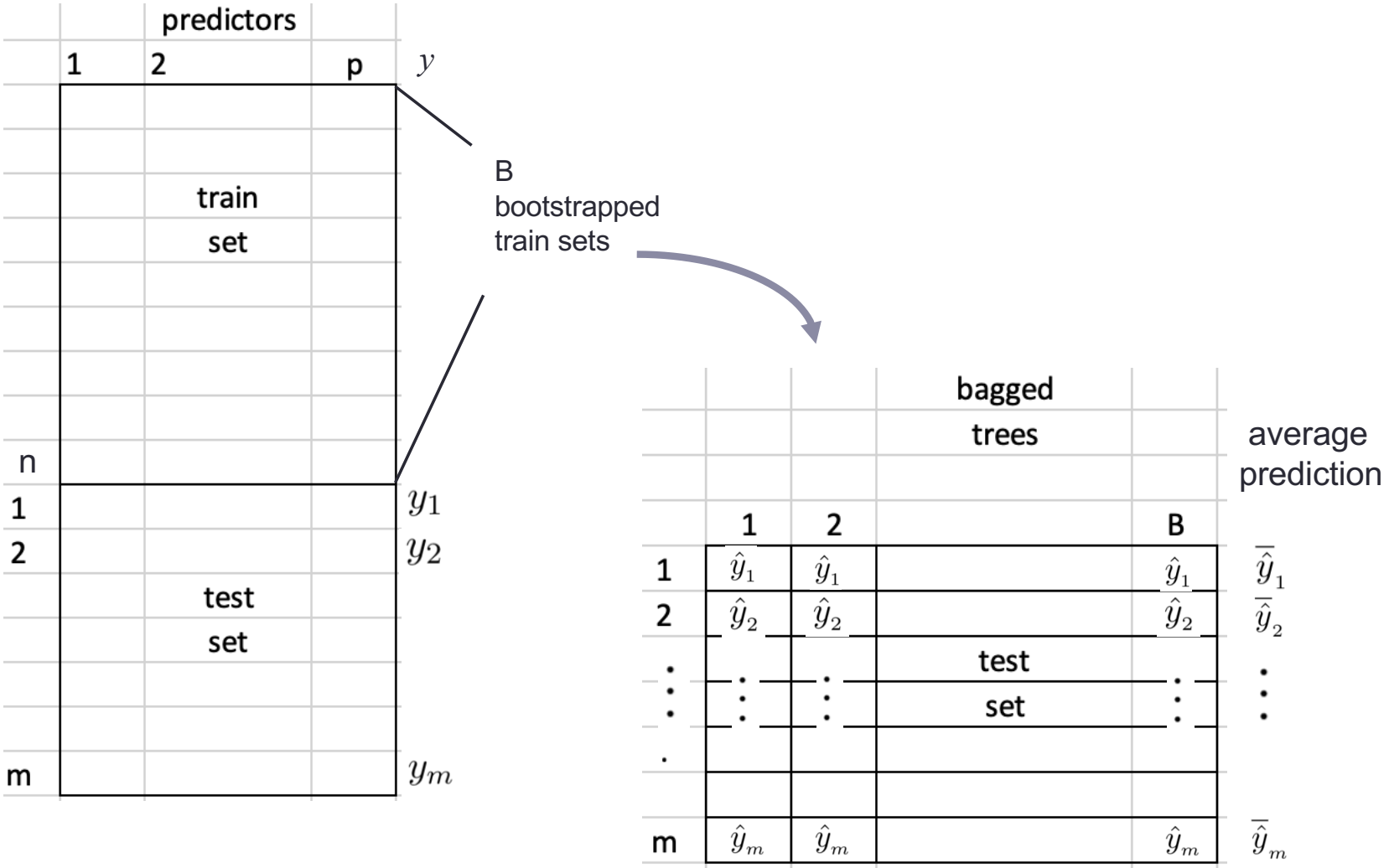
Bagging for Regression Trees



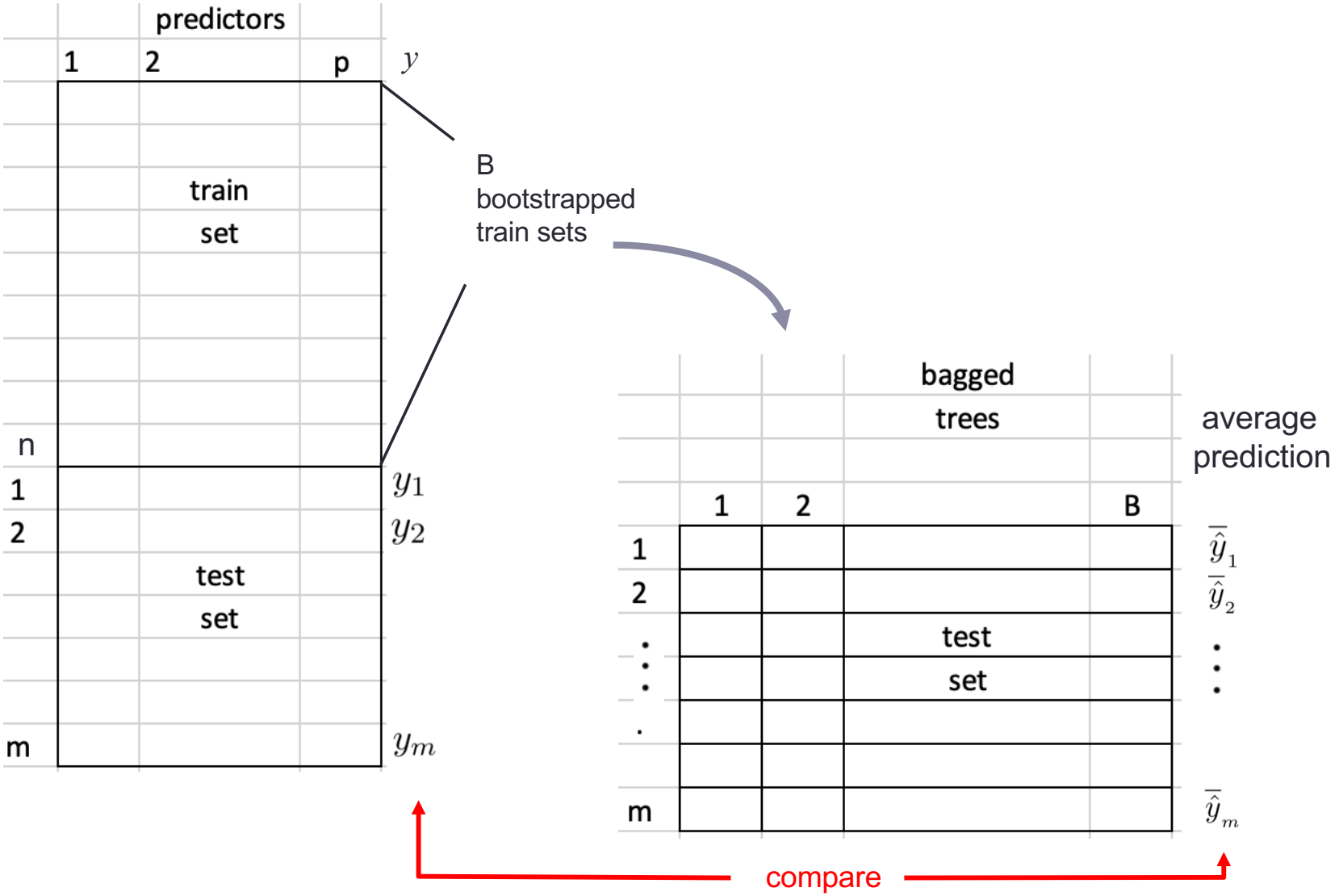
Bagging for Regression Trees



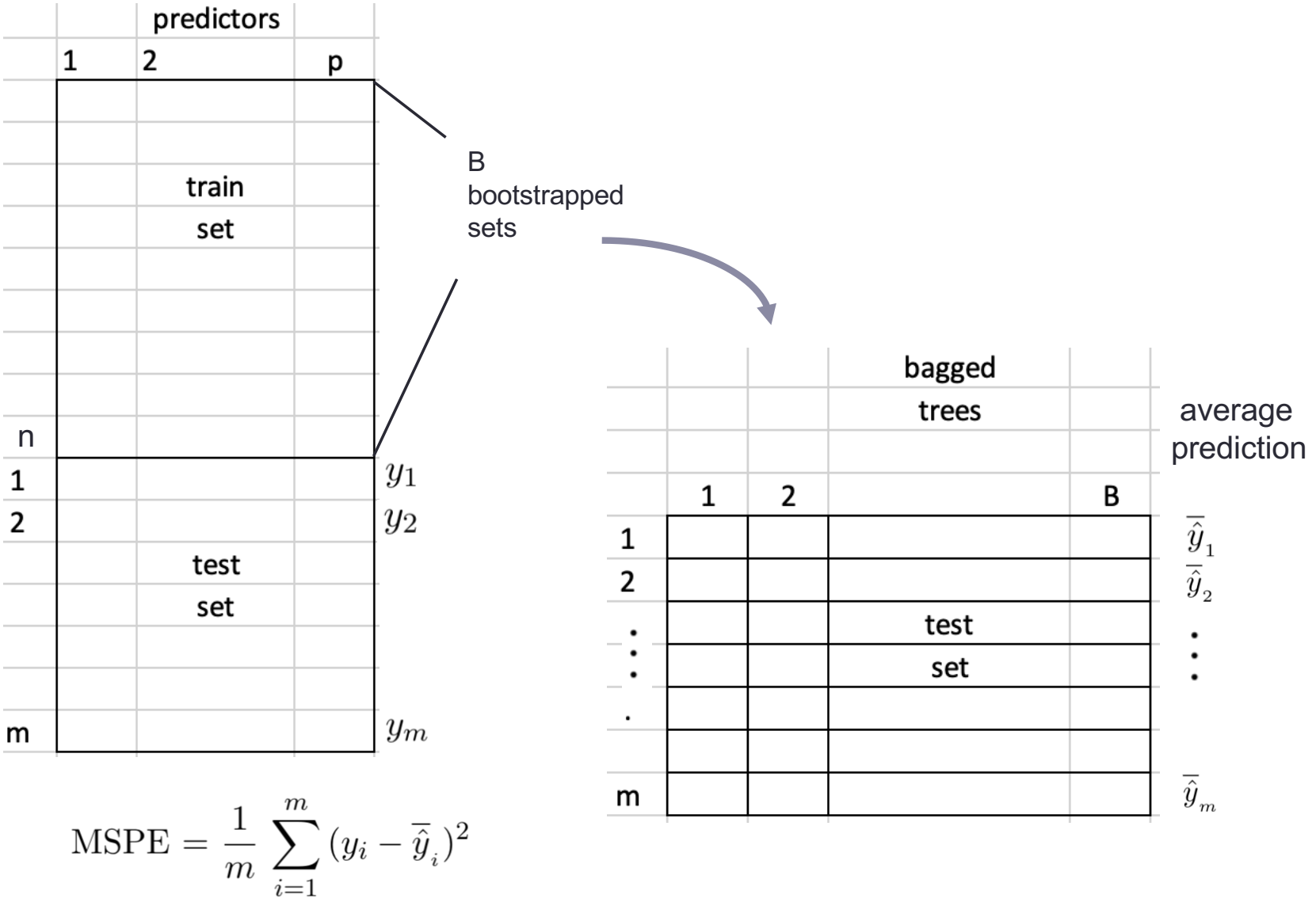
Bagging for Regression Trees



Bagging for Regression Trees



Bagging for Regression Trees



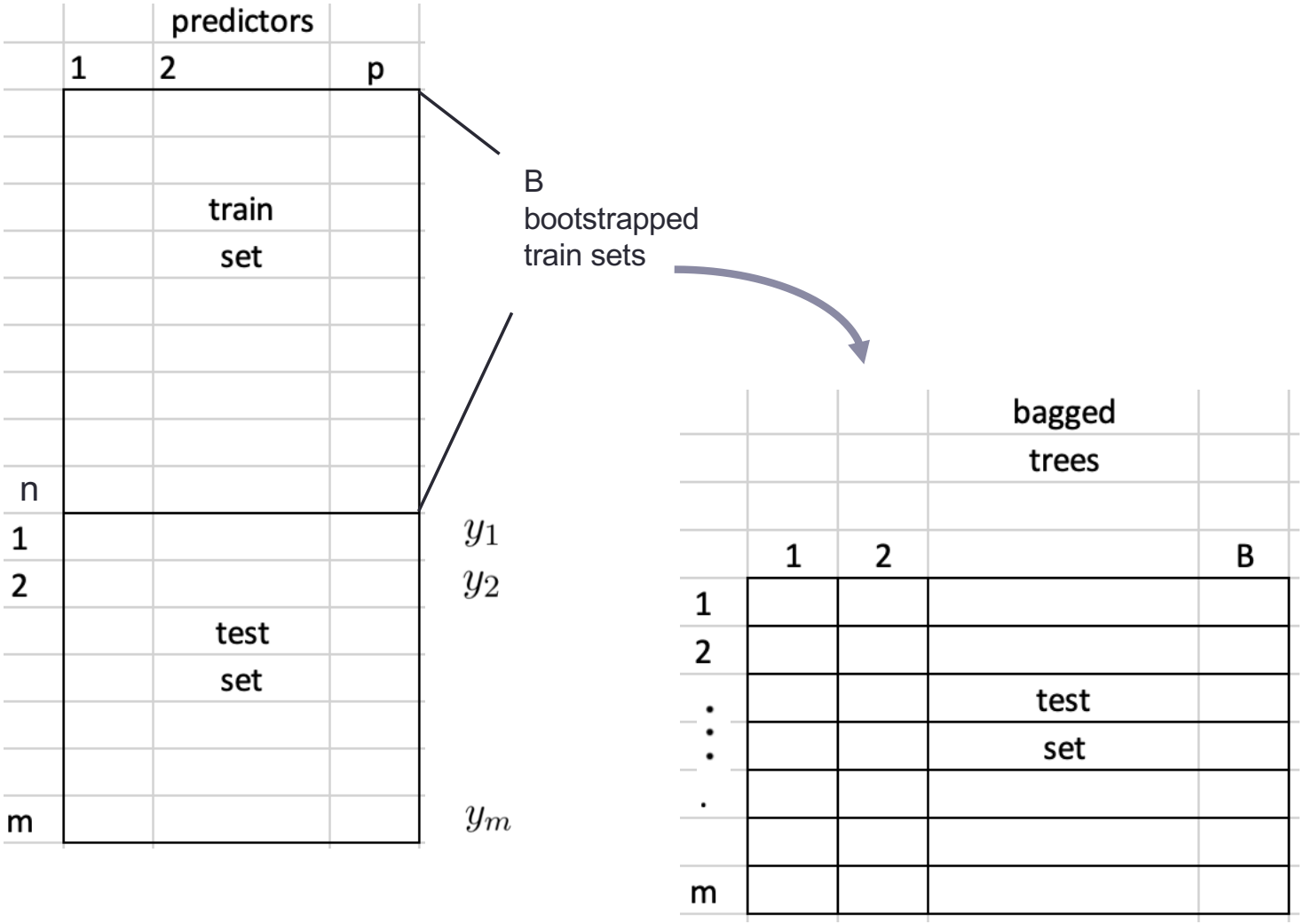
$$\text{MSPE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \overline{\hat{y}}_i)^2$$

Bagging for **Classification Trees**

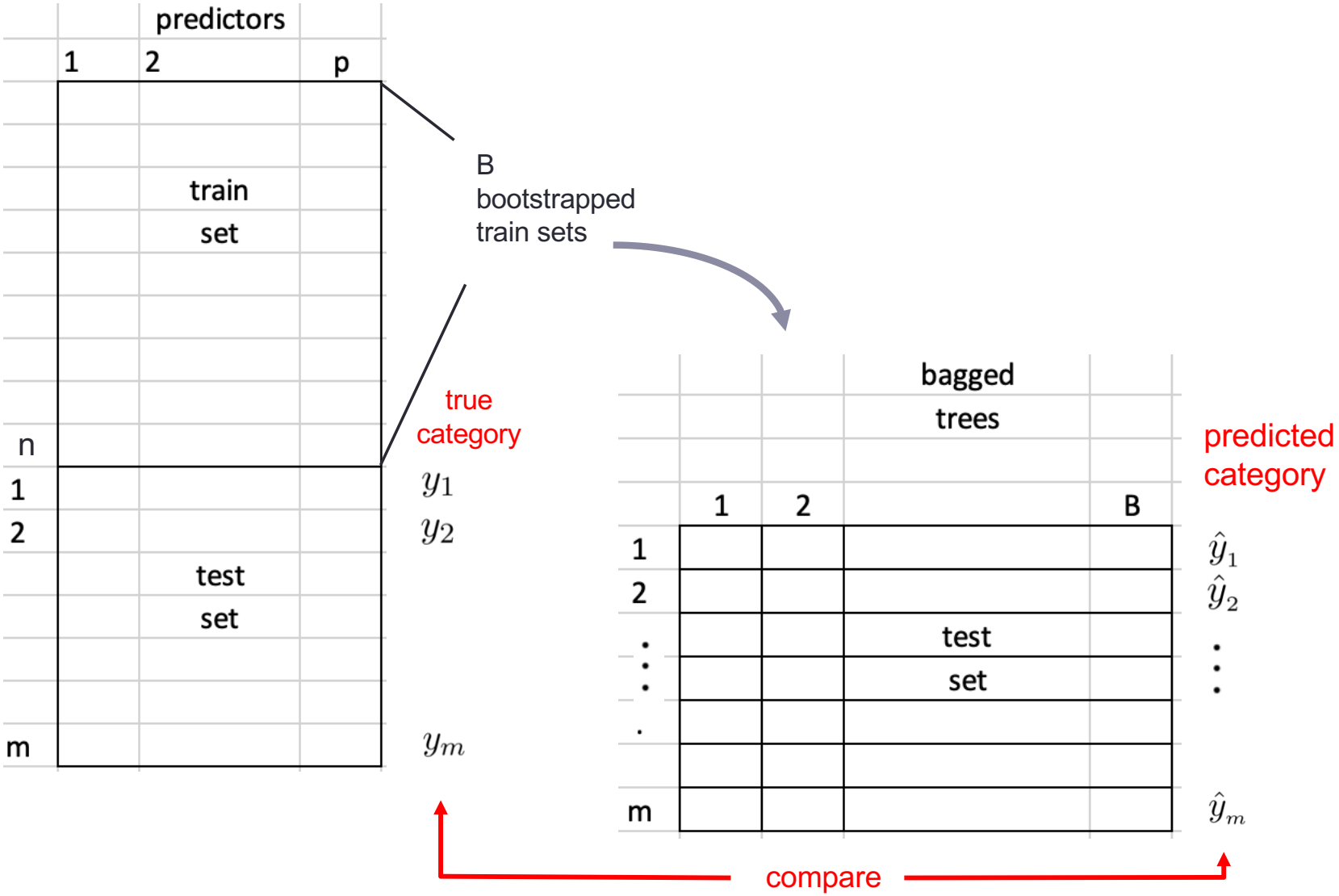
There are two approaches

- 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

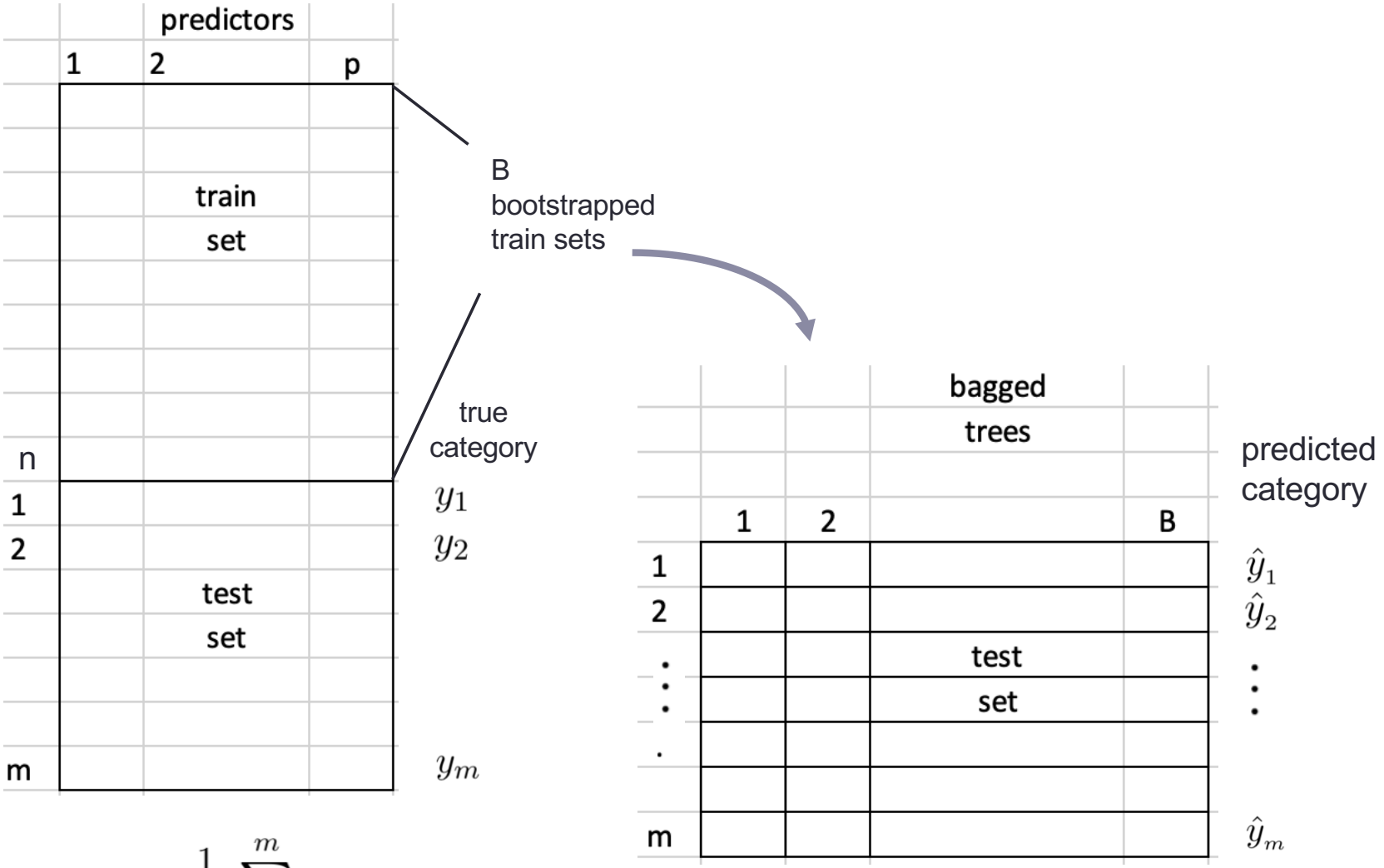
Bagging for Classification Trees



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
- Then many of the trees may contain the same set of powerful predictors
- The trees would yield similar predictions
- We say that the predictions are co-rrelated
- We need a way to de-correlate them

RANDOM FORESTS

Random Forests

A simple modification on bagging

How does it work?

- Before each split, randomly select $m < p$ predictors as candidates to make the split

  then choose that one giving

  the largest MSE reduction

- Bagging uses $m = p$ (all the predictors)
- Random Forest $m < p$ predictors

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

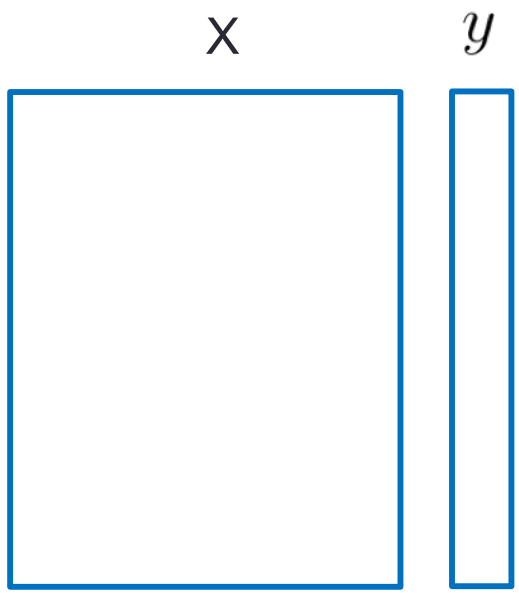
- If there is a single strong predictor, most bagged trees will choose it for the first split (and for the following splits too)
- Most trees will look similar
- As a result their predictions will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction
- 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

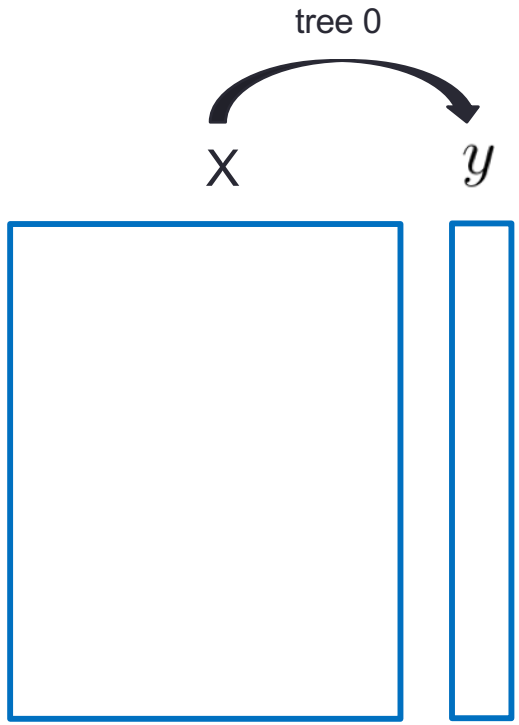
Gradient Boosting

- 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

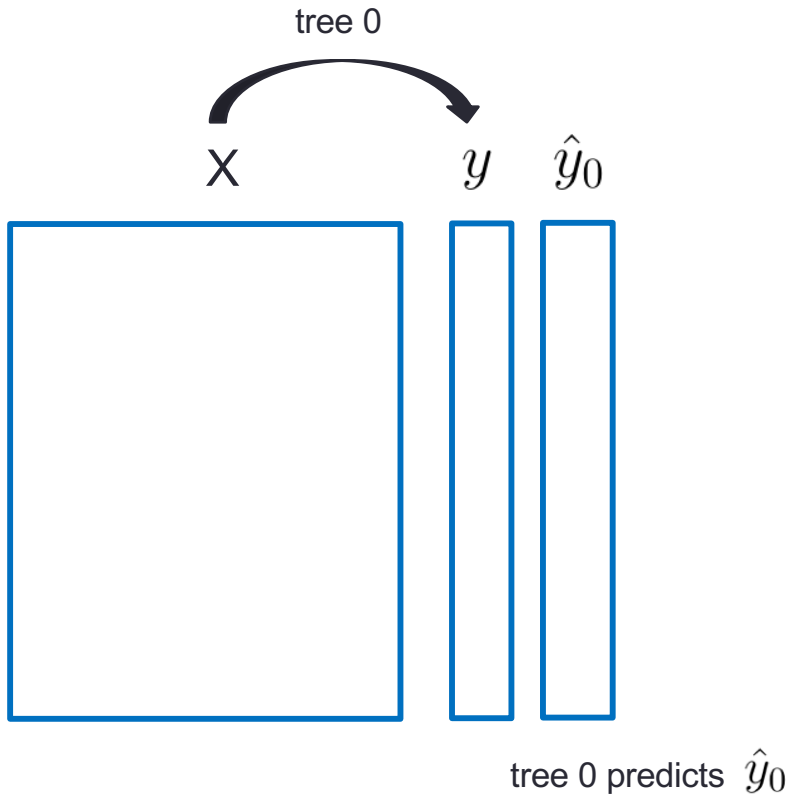
Boosting procedure



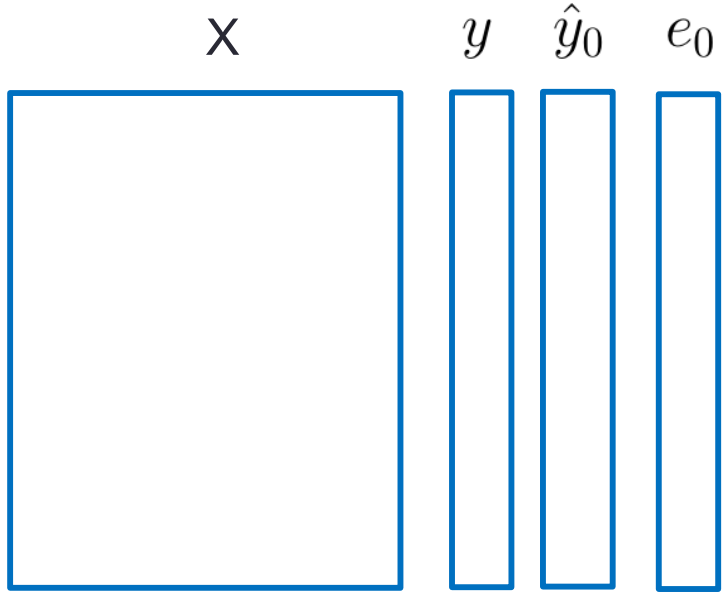
Boosting procedure



Boosting procedure

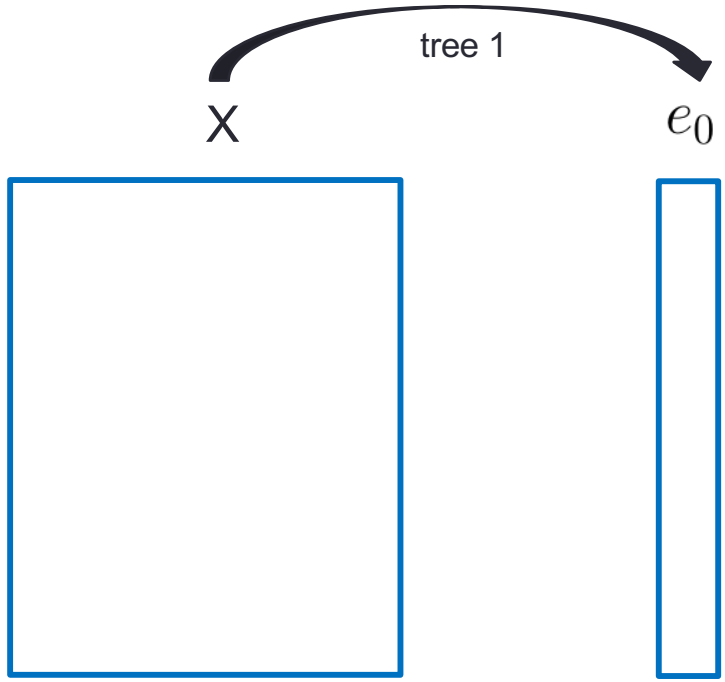


Boosting procedure

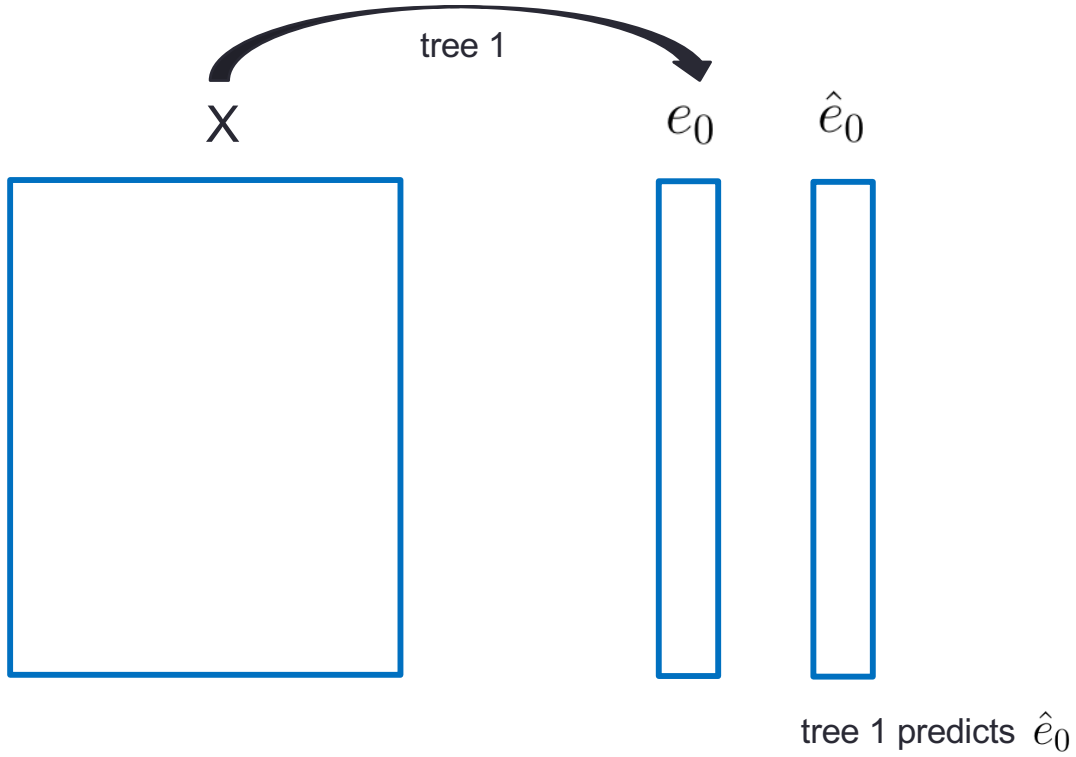


$$e_0 = y - \hat{y}_0$$

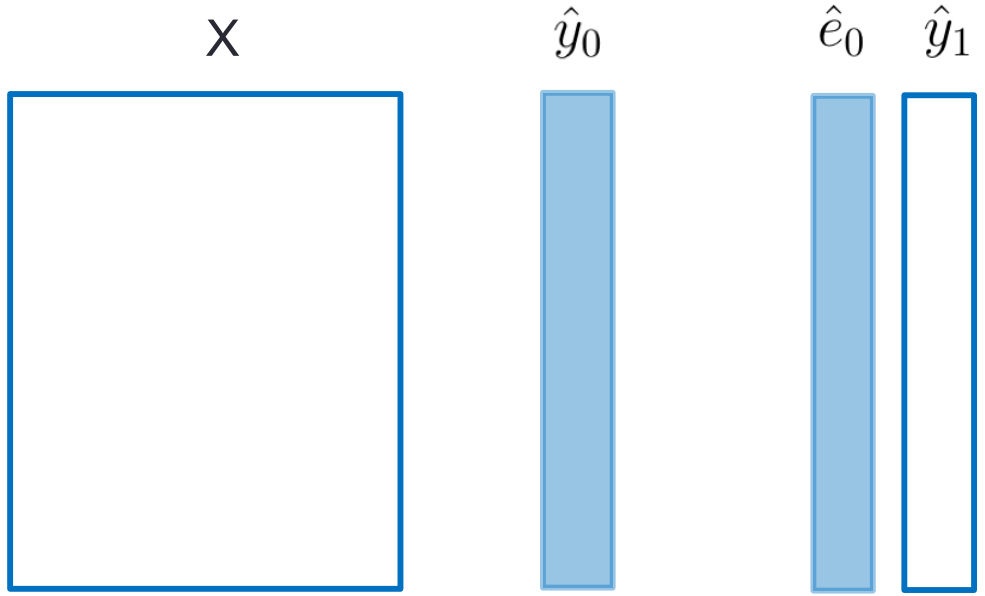
Boosting procedure



Boosting procedure



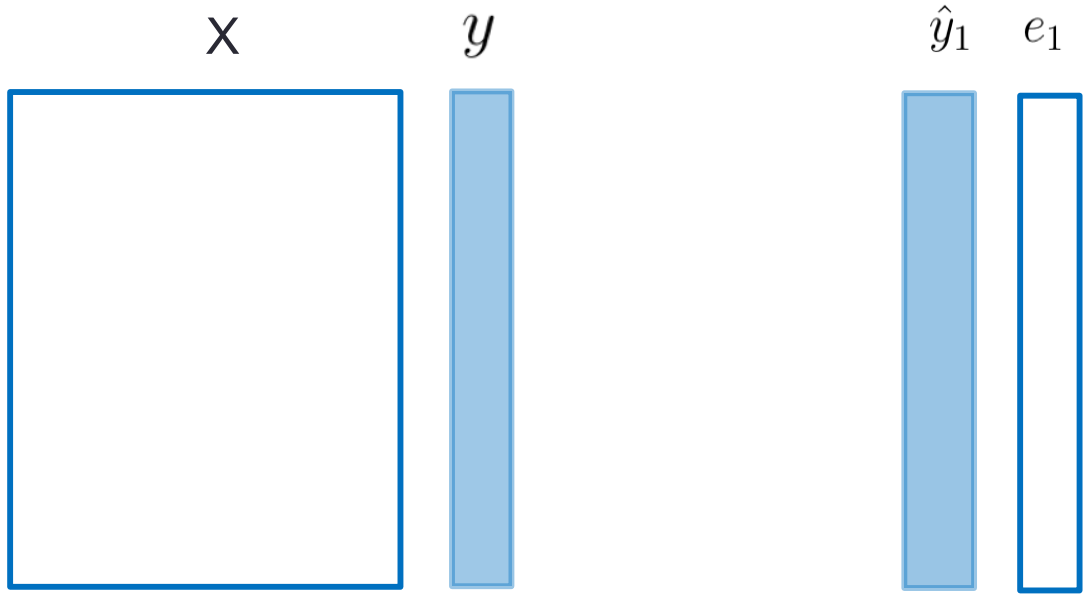
Boosting procedure



tree 1 predicts $\hat{e}_0$

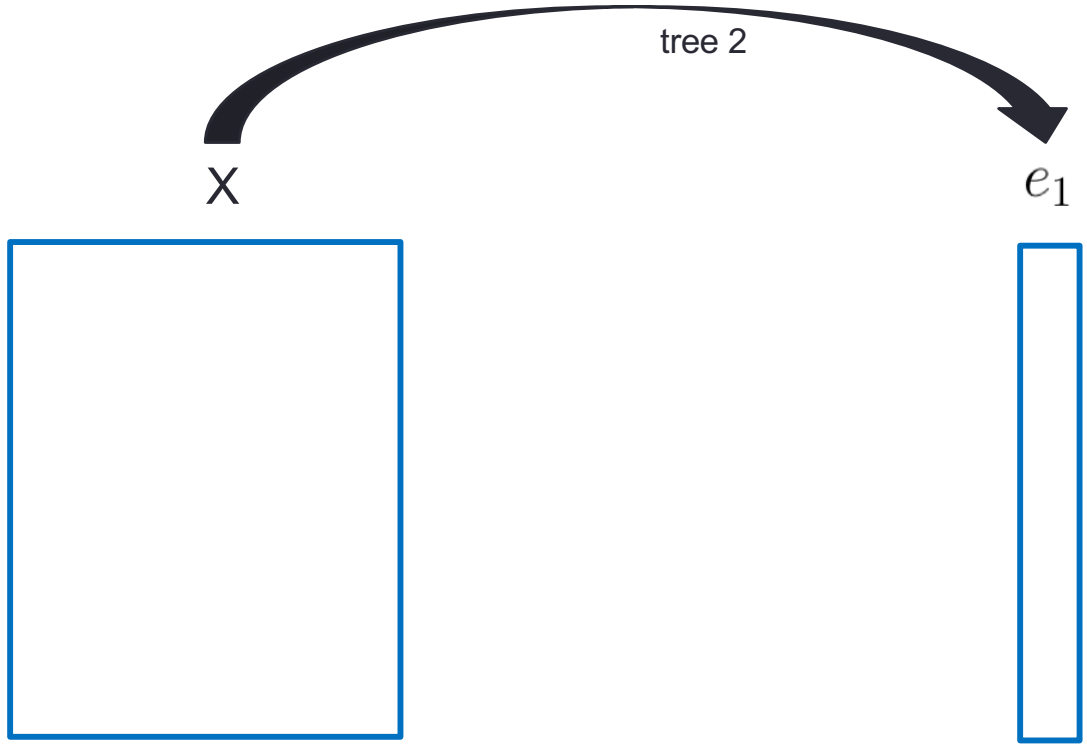
$$\hat{y}_1 = \hat{y}_0 + \alpha\,\hat{e}_0$$

Boosting procedure

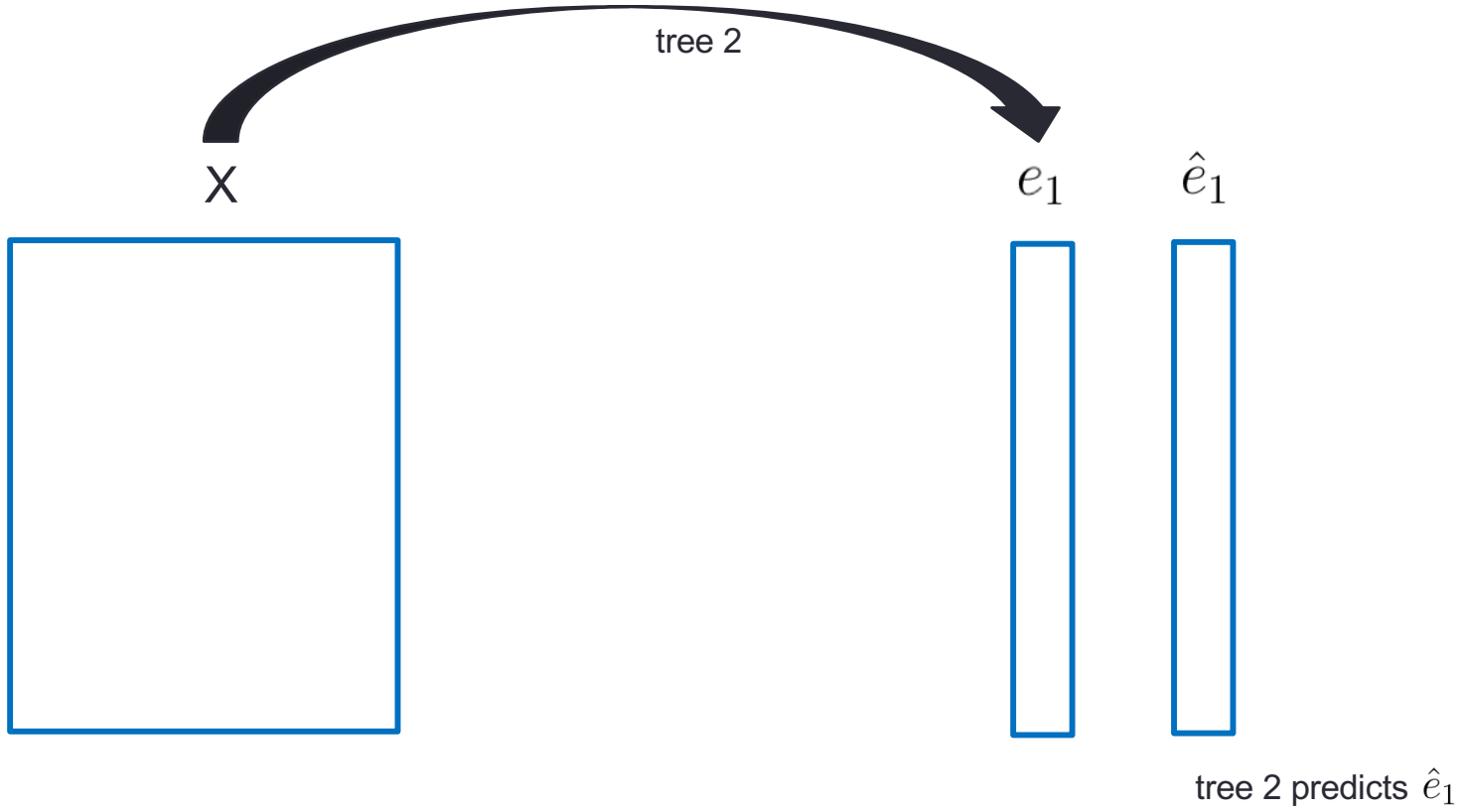


$$e_1 = y - \hat{y}_1$$

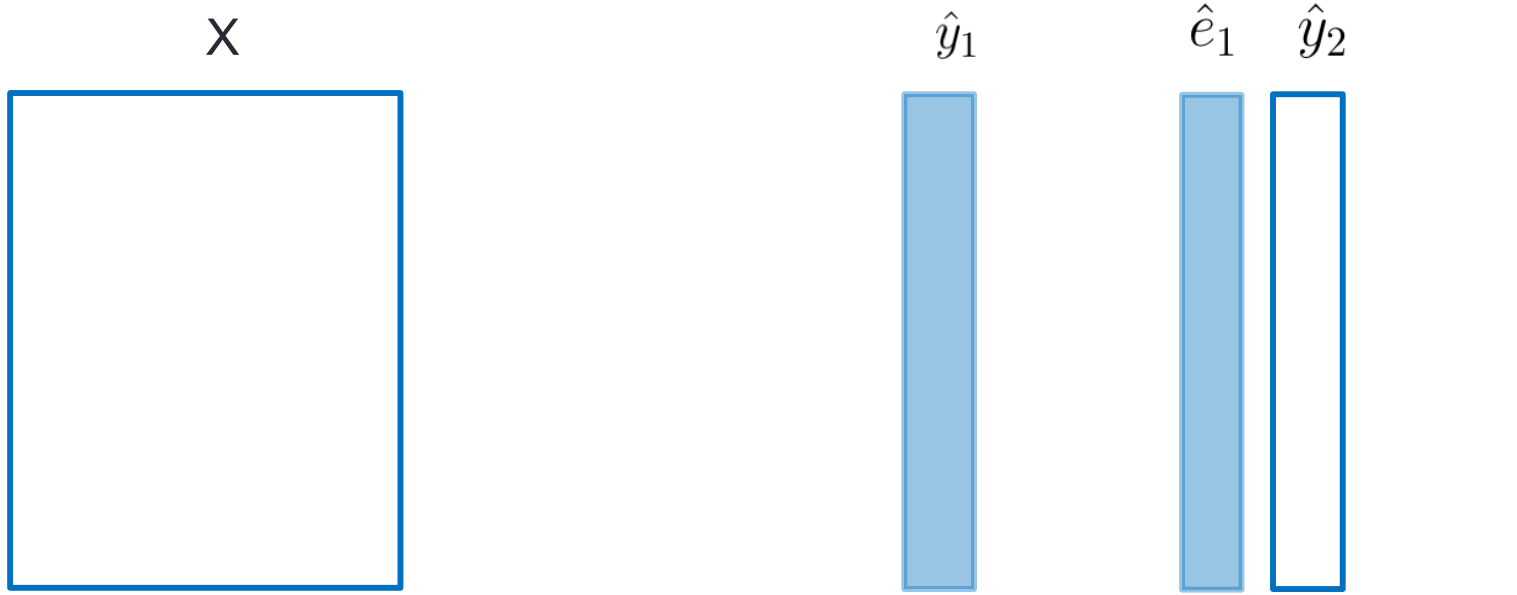
Boosting procedure



Boosting procedure



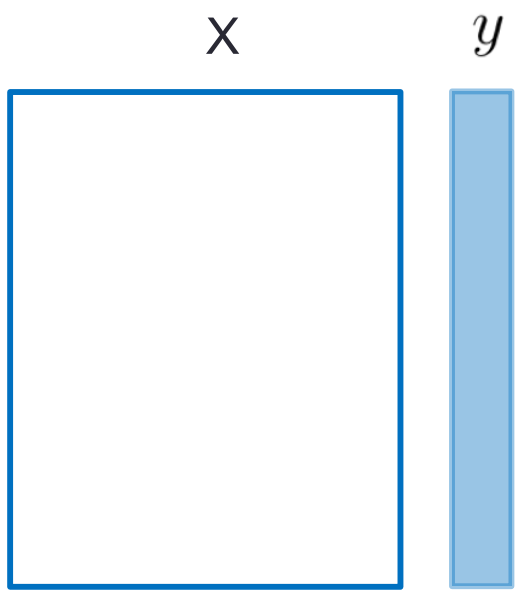
Boosting procedure

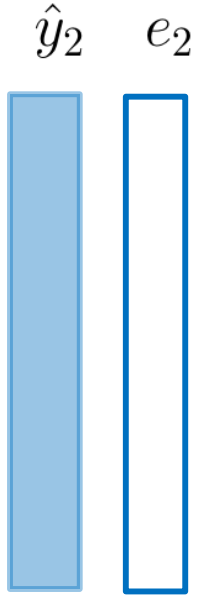


tree 2 predicts $\hat{e}_1$

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

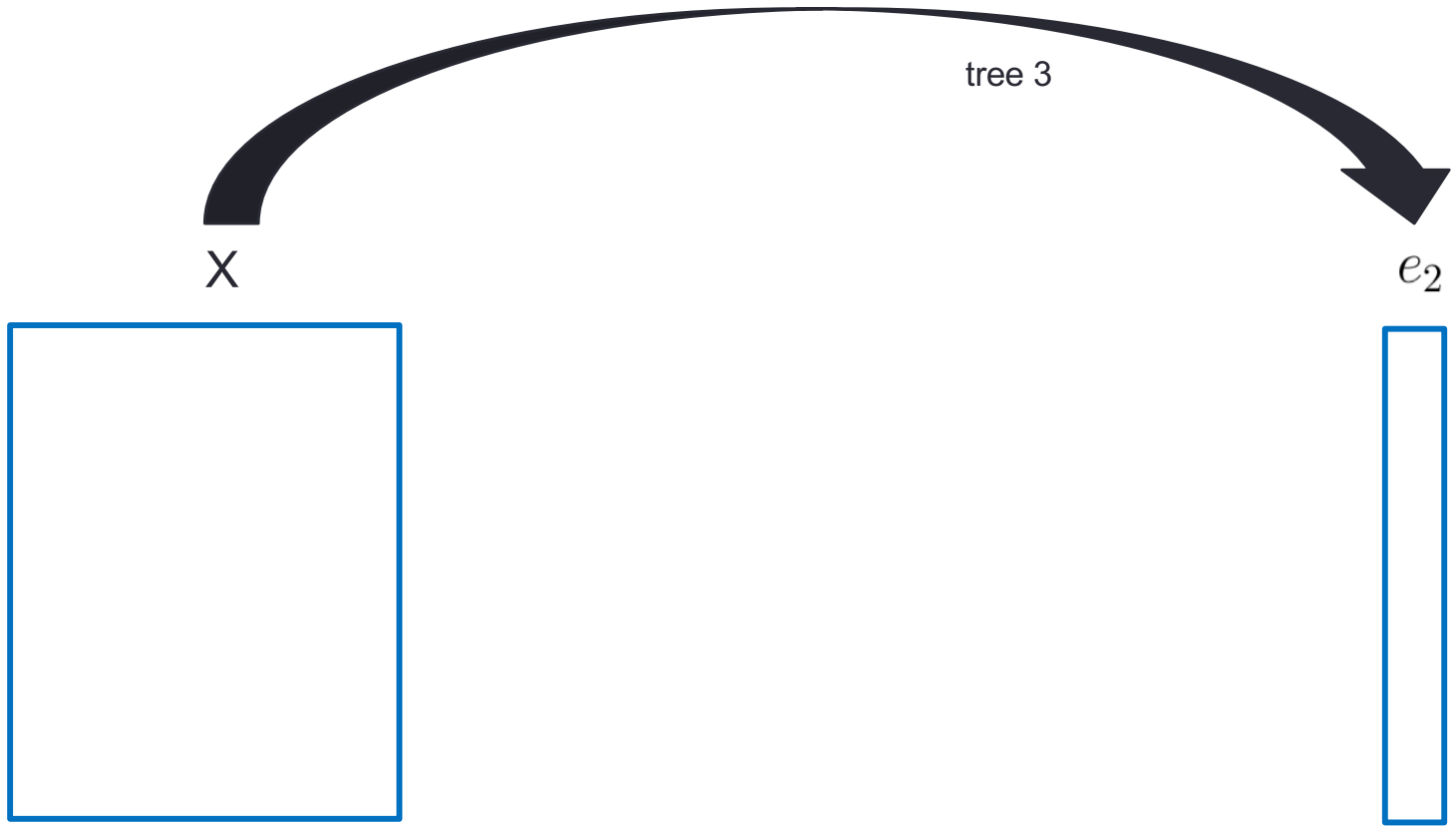
Boosting procedure



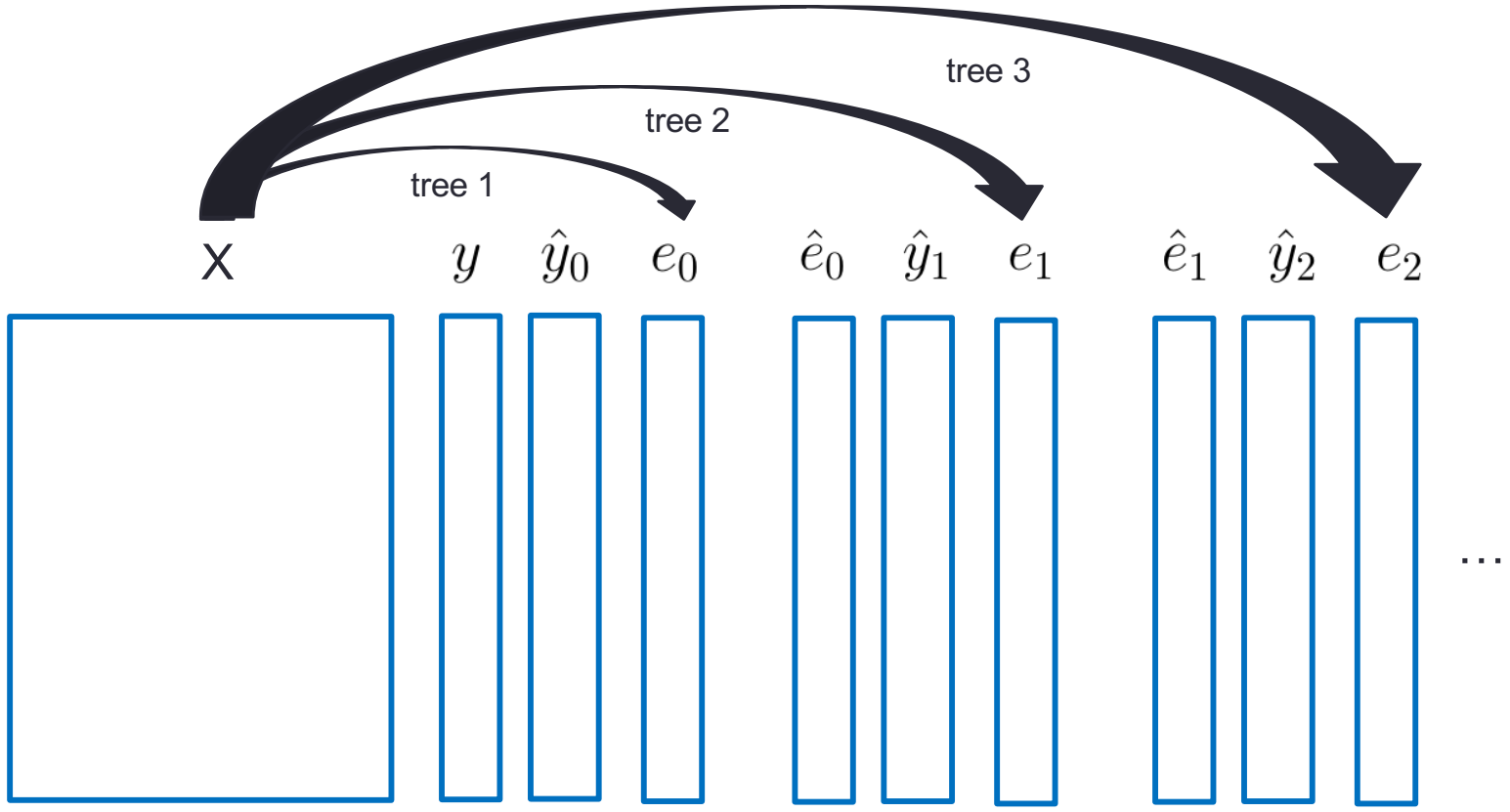


$$e_2 = y - \hat{y}_2$$

Boosting procedure



Boosting procedure



$$e_0 = y - \hat{y}_0 \qquad \hat{y}_1 = \hat{y}_0 + \alpha\,\hat{e}_0 \qquad \hat{y}_2 = \hat{y}_1 + \alpha\,\hat{e}_1$$

$$e_1 = y - \hat{y}_1 \qquad e_2 = y - \hat{y}_2$$

Hyperparameters

Random Forest

- max_features
- n_estimators
- max_depth

Gradient Boosting

- learning_rate
- max_features
- n_estimators
- max_depth

Example 0

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
```

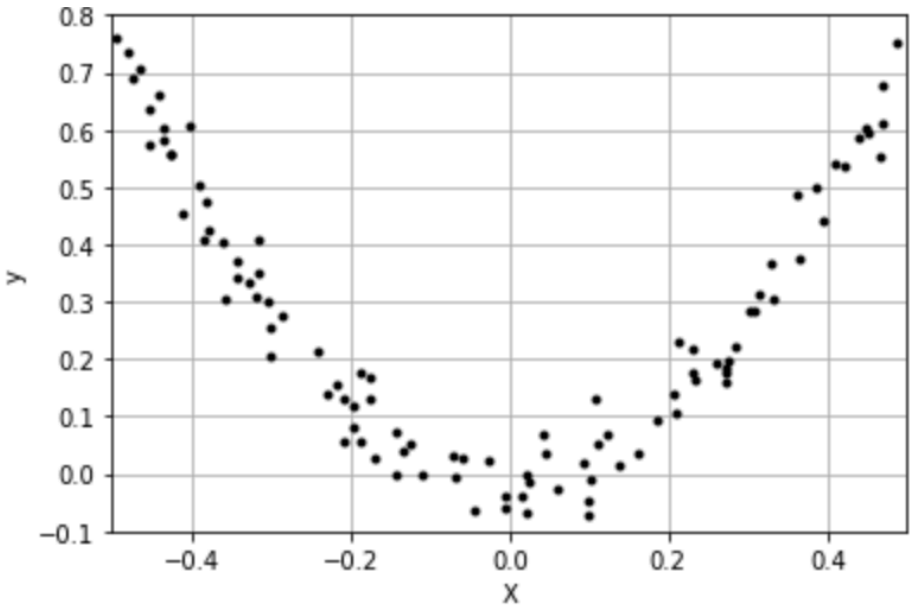
```
(100, 2)
```

```
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
```

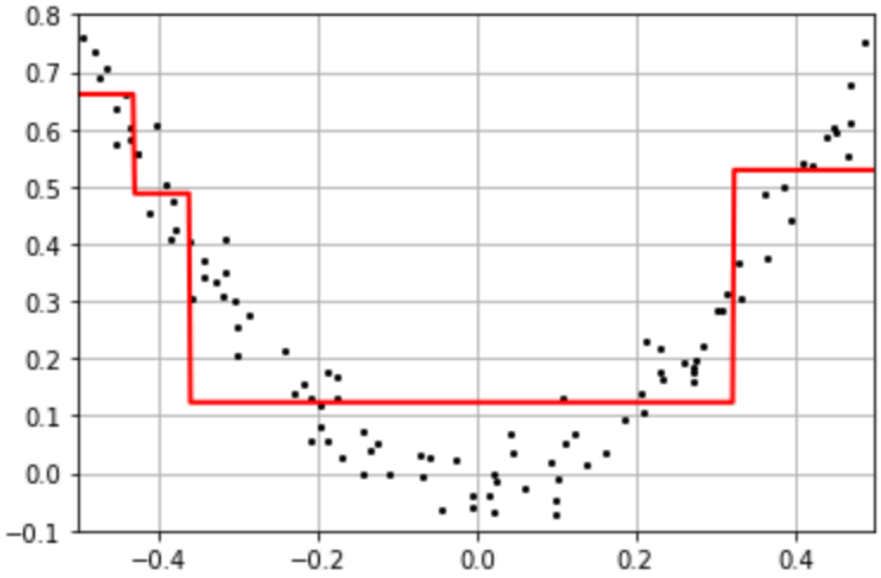
```
(500, 1)
```

Example – Model 1

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

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

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

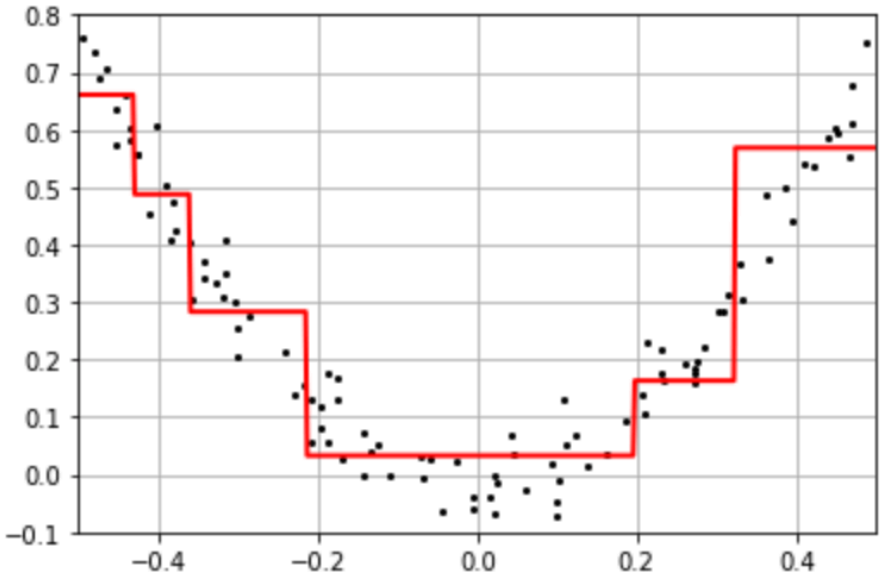



```
yhat1 = tree.predict(X)
e1 = y - yhat1
```

Example – Model 2

```
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();
```

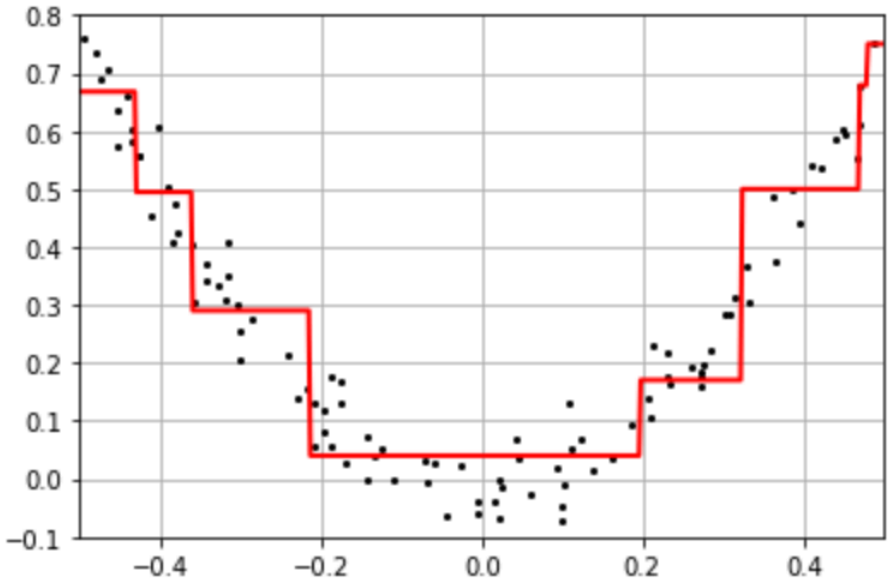



```
yhat2 = tree.predict(X)
e2 = e1 - yhat2
```

Example – Model 3

```
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();
```

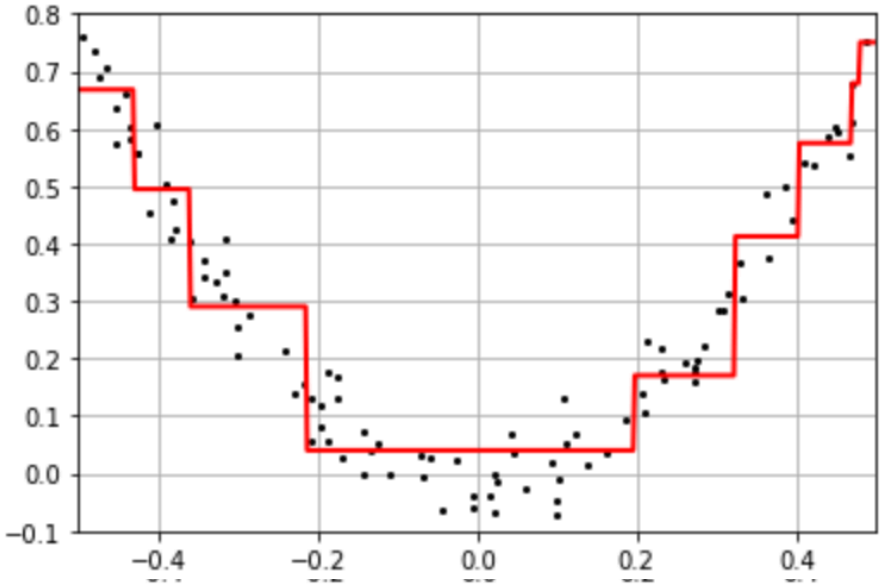



```
yhat3 = tree.predict(X)
e3 = e2 - yhat3
```

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();
```

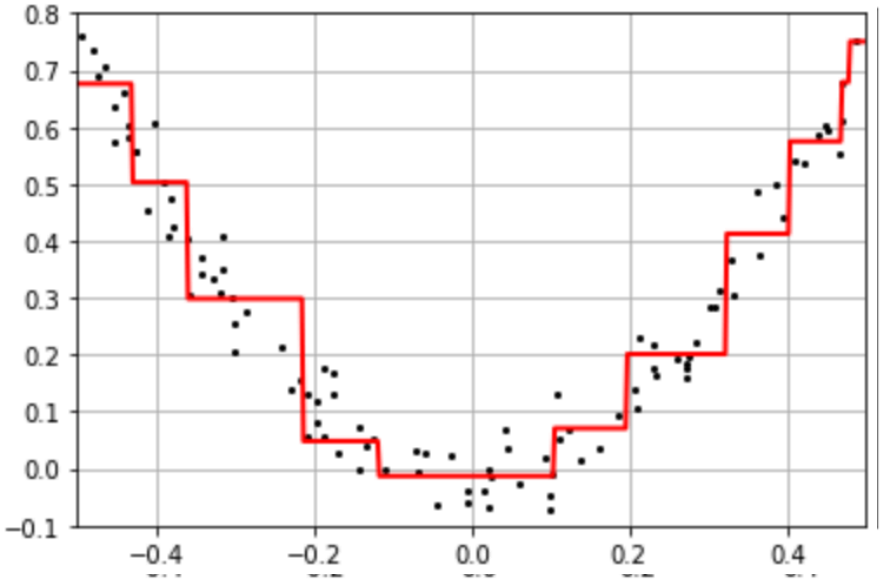



```
yhat4 = tree.predict(X)
e4 = e3 - yhat4
```

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();
```

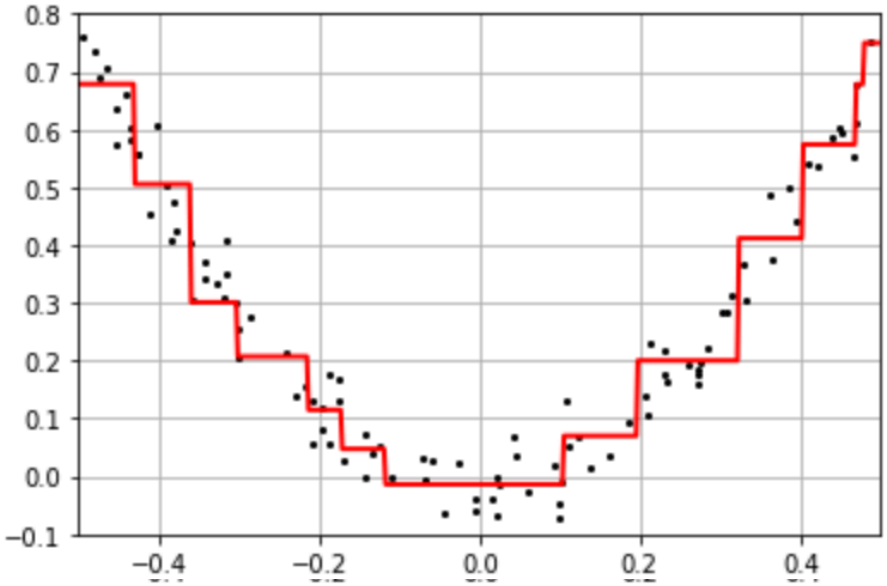



```
yhat5 = tree.predict(X)
e5 = e4 - yhat5
```

Example – Model 6

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

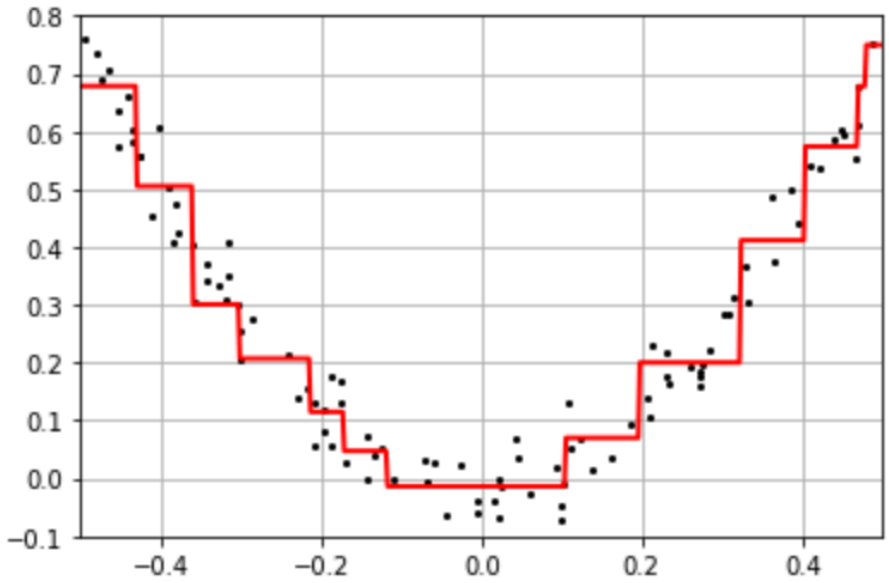
```
plt.plot(X, y, "k.",markersize=4)
plt.plot(x1,red_line6,"r-", linewidth=2)
plt.axis(axes)
plt.grid();
```



Example

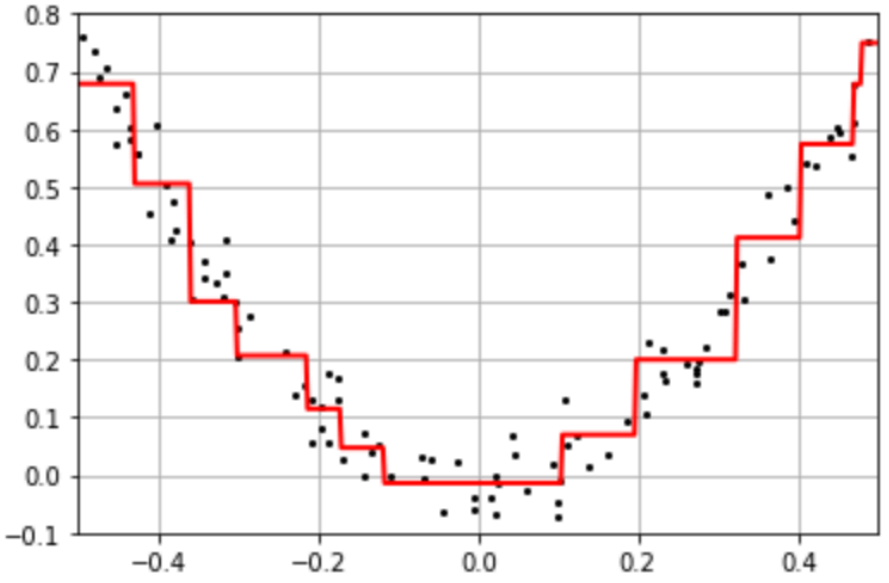
```
tree.fit(X,e5)
red_line6 = red_line5 + tree.predict(x1)
```

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



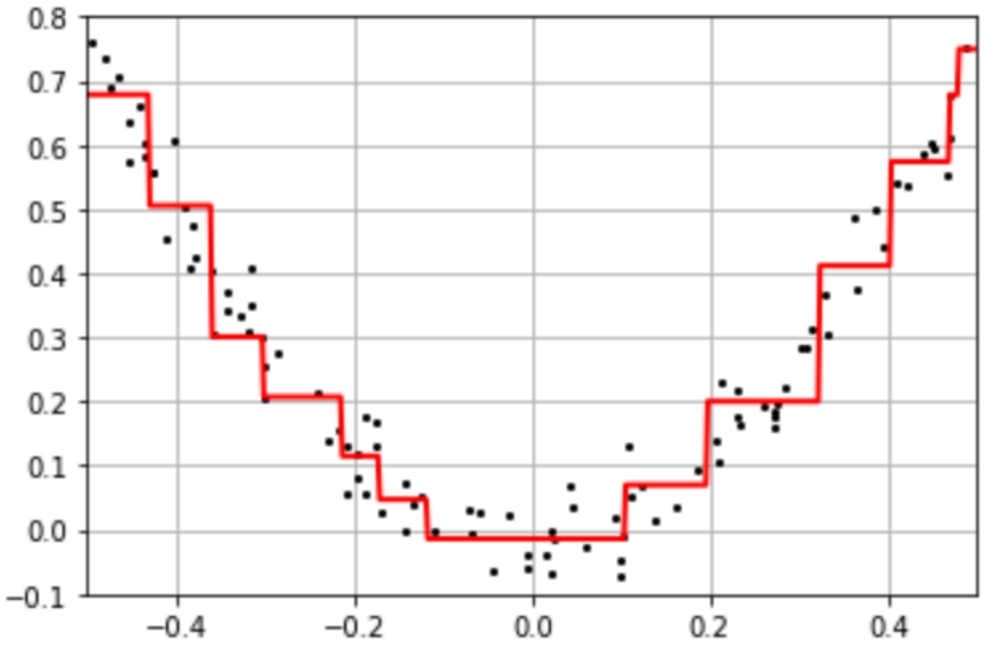
Gradient Boosting with 6 steps

```
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();
```



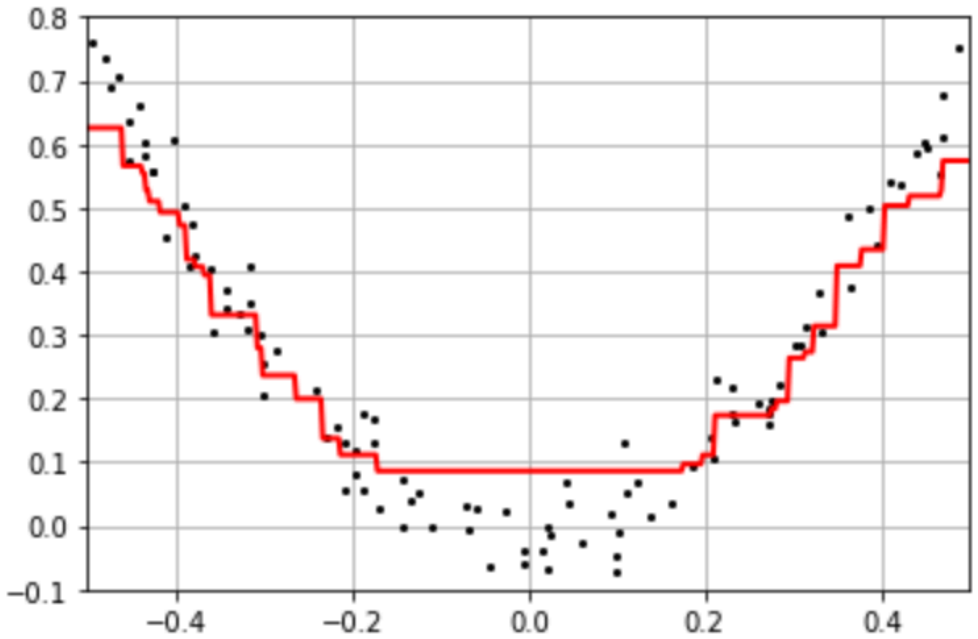
Example – Gradient Boosting with 6 steps

```
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();
```



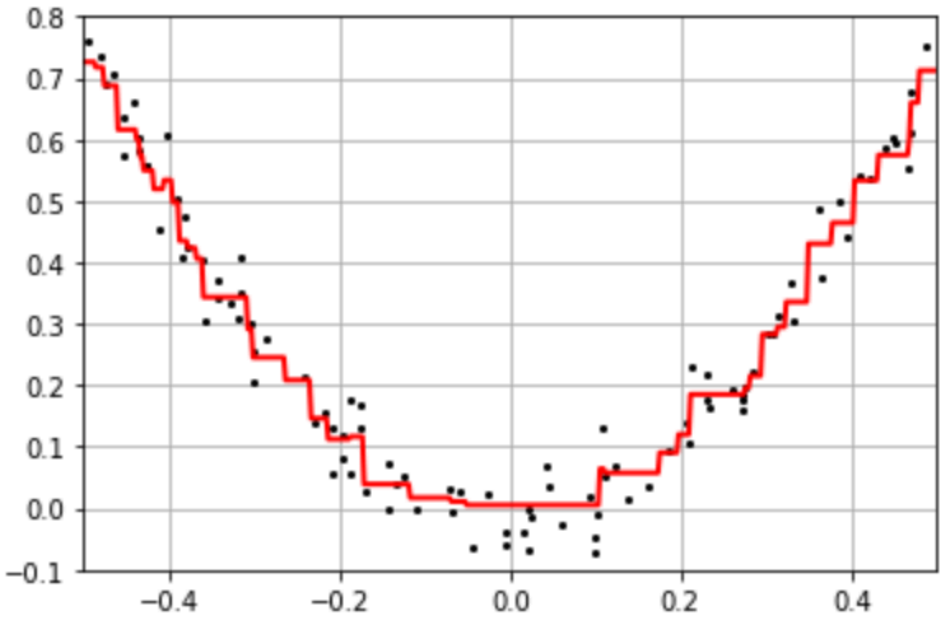
Example – Gradient Boosting with 20 steps

```
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();
```



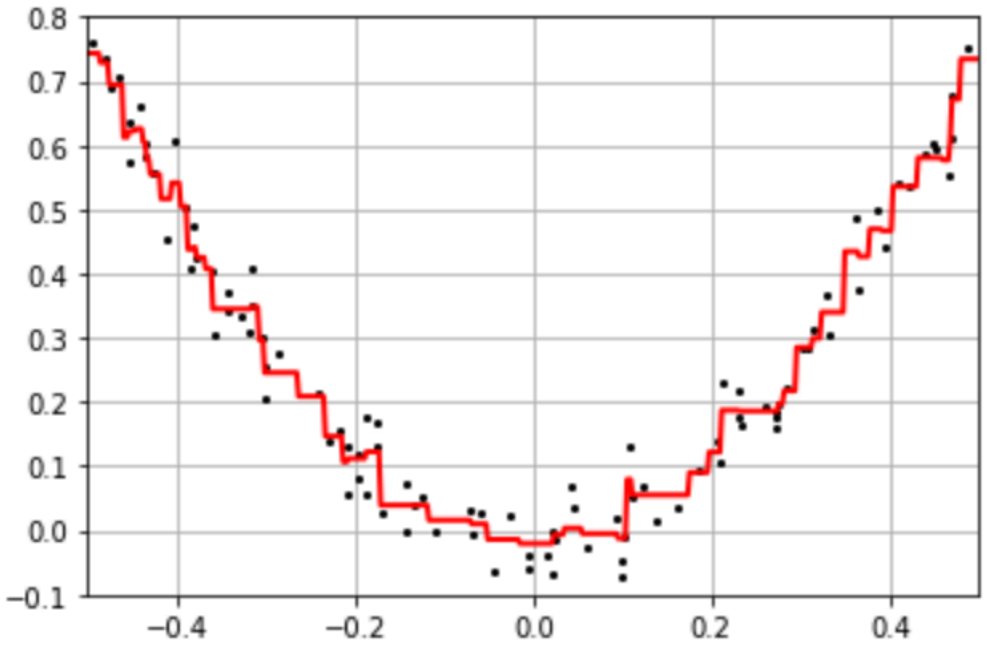
Example – Gradient Boosting with 50 steps

```
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 1 – Ensembles on Regression

Example – Boston dataset

Data of 506 houses in the area of Boston

- Want to predict the price of houses and to identify which variables are most important for prediction
- Split the dataset into a training (50%) and a test set
- Fit and compare bagged trees with 25 and 500 trees. Find the test MSPE.
- Fit a Random Forest with 500 trees and max_features = 6. Which predictors are most important?
- Fit 500 Gradient boosted trees with max_depth = 4, and α = 0.01, 0.20. Which predictors are most important?

Boston dataset -13 features, 1 target

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- BLACK - proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median price of owner-occupied homes in $1000's

Example – libraries

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

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

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

Use RandomForestRegressor for both Bagging and Random Forest

Example – Boston dataset variables

```
boston_df = pd.read_csv('Boston.csv')
boston_df[:5]
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
X = boston_df.drop('medv',axis =1)
y = boston_df.medv
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.5,
                                                 random_state=0)
```

Example – Single Tree

Single Tree

```
tree1 = DecisionTreeRegressor(max_depth=4)
tree1.fit(X_train,y_train)
pred1 = tree1.predict(X_test)
mspe = mean_squared_error(y_test,pred1)
mspe
```

```
23.817371513828615
```

Example – Single Tree

Single Tree

```
tree1 = DecisionTreeRegressor(max_depth=4)
tree1.fit(X_train,y_train)
pred1 = tree1.predict(X_test)
mspe = mean_squared_error(y_test,pred1)
mspe
```

```
23.817371513828615
```

Bagging on B=500 Regression Trees

All p=13 predictors will be considered at each split of the tree
m = max_features = n. of predictors = p

```
bag500 = RandomForestRegressor(max_features=13,
                               max_depth=4,
                               n_estimators = 500,
                               random_state=1)
bag500.fit(X_train,y_train);
```

```
pred2 = bag500.predict(X_test)
mean_squared_error(y_test,pred2)
```

```
17.489095605372235
```

Example – Single Tree

Single Tree

```
tree1 = DecisionTreeRegressor(max_depth=4)
tree1.fit(X_train,y_train)
pred1 = tree1.predict(X_test)
mspe = mean_squared_error(y_test,pred1)
mspe
```

```
23.817371513828615
```

Bagging on B=500 Regression Trees

All p=13 predictors will be considered at each split of the tree
m = max_features = n. of predictors = p

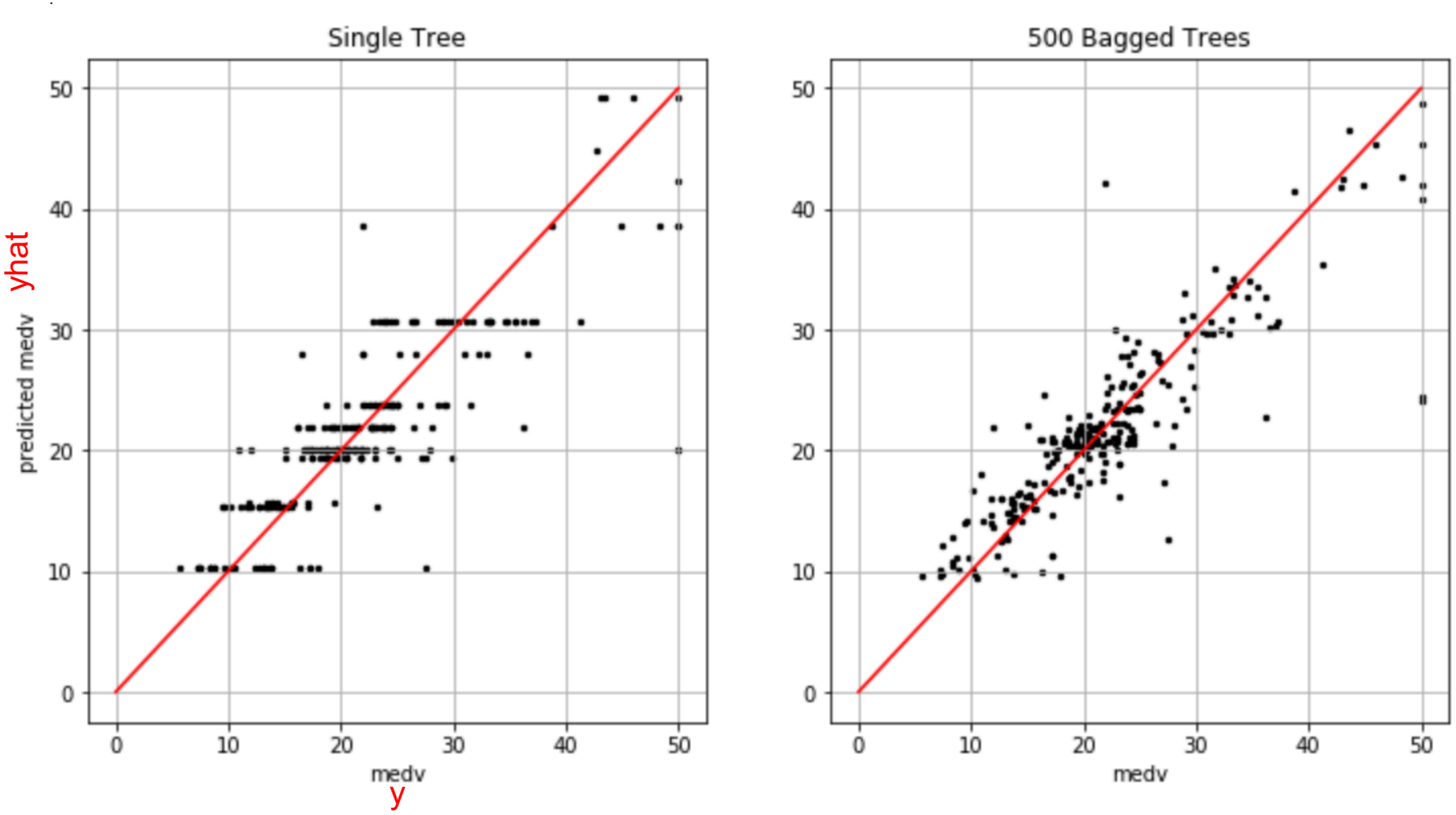
```
bag500 = RandomForestRegressor(max_features=13,
                               max_depth=4,
                               n_estimators = 500,
                               random_state=1)
bag500.fit(X_train,y_train);
```

max_features = p

```
pred2 = bag500.predict(X_test)
mean_squared_error(y_test,pred2)
```

```
17.489095605372235
```

Example – Single Tree vs Bagging



Boston data – Bagging 500 vs 25 trees

```
bag500 = RandomForestRegressor(max_features=13,
                               max_depth=4,
                               n_estimators = 500,
                               random_state=1)
bag500.fit(X_train,y_train);
```

```
pred2 = bag500.predict(X_test)
mean_squared_error(y_test,pred2)
```

```
17.489095605372235
```

```
bag25 = RandomForestRegressor(max_features=13,max_depth=4,
                               n_estimators = 25,
                               random_state = 1)
bag25.fit(X_train,y_train);
```

```
pred = bag25.predict(X_test)
mean_squared_error(y_test,pred)
```

```
18.37639273778574
```

Boston data – Bagging 500 vs 25 trees

```
bag500 = RandomForestRegressor(max_features=13,
                               max_depth=4,
                               n_estimators = 500,
                               random_state=1)
bag500.fit(X_train,y_train);
```

```
pred2 = bag500.predict(X_test)
mean_squared_error(y_test,pred2)
```

```
17.489095605372235
```

small loss

```
bag25 = RandomForestRegressor(max_features=13,max_depth=4,
                               n_estimators = 25,
                               random_state = 1)
bag25.fit(X_train,y_train);
```

```
pred = bag25.predict(X_test)
mean_squared_error(y_test,pred)
```

```
18.37639273778574
```

Comparing Bagging vs Random Forest

```
bag500 = RandomForestRegressor(max_features=13,
                               max_depth=4,
                               n_estimators = 500,
                               random_state=1)
bag500.fit(X_train,y_train);
```

Bagging
$p = 13$

```
pred2 = bag500.predict(X_test)
mean_squared_error(y_test,pred2)
```

```
17.489095605372235
```

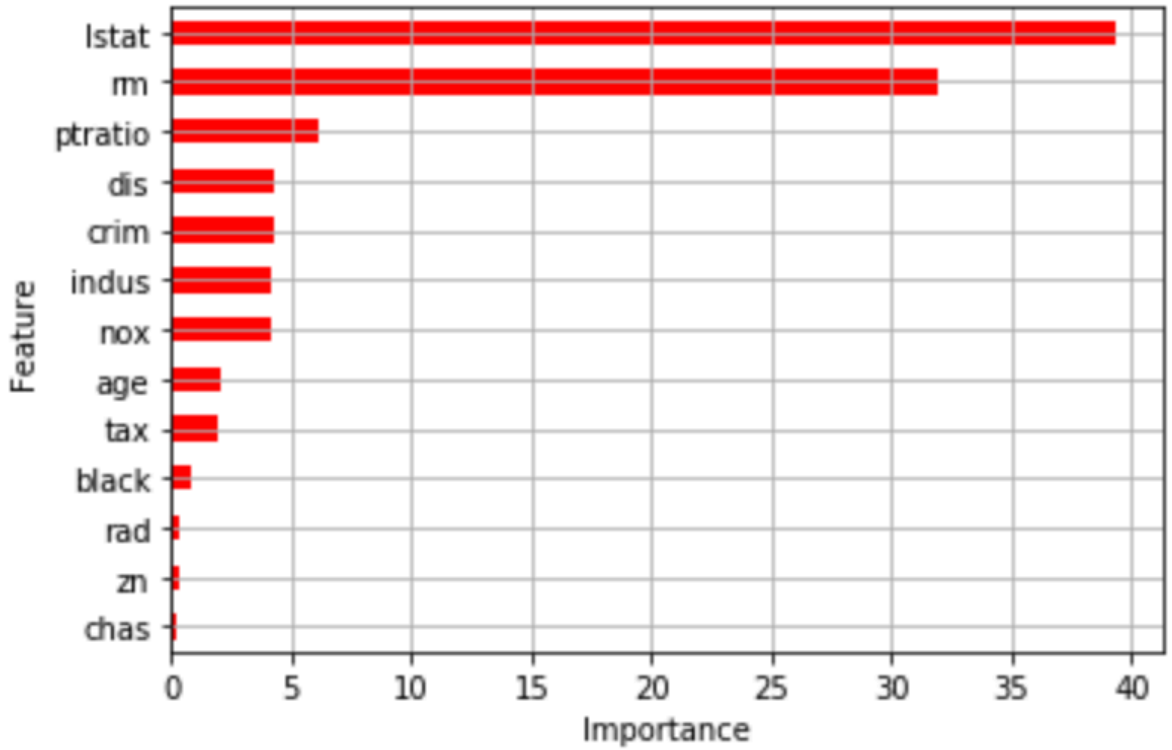
```
forest1 = RandomForestRegressor(max_features = 6,
                                max_depth = 4,
                                n_estimators = 500,
                                random_state = 1)
forest1.fit(X_train,y_train)
pred = forest1.predict(X_test)
mean_squared_error(y_test,pred)
```

Random Forest
$m = 6 < p$

```
17.533313043949654
```

Random Forest – Feature Importance

```
Importance = pd.DataFrame({'Importance':forest1.feature_importances_*100},
                          index = X.columns)
df8 = Importance.sort_values(by = 'Importance',axis = 0,
                             ascending = True)
df8.plot(kind = 'barh',color = 'r',legend = False)
```



Gradient Boosting Regression Trees

```
boost10 = GradientBoostingRegressor(n_estimators = 500,
                                    learning_rate = 0.1,
                                    max_depth = 4,
                                    random_state =1)
boost10.fit(X_train,y_train)
mean_squared_error(y_test,boost10.predict(X_test))
```

```
17.143788835794222
```

Gradient Boosting Trees - learning_rate

```
boost10 = GradientBoostingRegressor(n_estimators = 500,
                                    learning_rate = 0.1,
                                    max_depth = 4,
                                    random_state =1)
boost10.fit(X_train,y_train)
mean_squared_error(y_test,boost10.predict(X_test))
```

```
17.143788835794222
```

```
boost40 = GradientBoostingRegressor(n_estimators = 500,
                                    learning_rate = 0.40,
                                    max_depth = 4,
                                    random_state=1)
boost40.fit(X_train,y_train)
mean_squared_error(y_test,boost40.predict(X_test))
```

```
16.65787780390907
```

GridSearchCV on the learning_rate

```
from sklearn.model_selection import GridSearchCV
```

```
lrates = np.linspace(0.01,1,9)
lrates
```

try these learning rates

```
array([0.01   , 0.13375, 0.2575 , 0.38125, 0.505  , 0.62875, 0.7525 ,
       0.87625, 1.     ])
```

GridSearchCV on the learning_rate

```
from sklearn.model_selection import GridSearchCV
```

```
lrates = np.linspace(0.01,1,9)
lrates
```

```
array([0.01   , 0.13375, 0.2575 , 0.38125, 0.505  , 0.62875, 0.7525 ,
       0.87625, 1.     ])
```

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

use the sklearn parameter name

```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.5,
                                                 random_state=0)
```

GridSearchCV on the learning_rate

```
model = GradientBoostingRegressor(n_estimators = 500,
                                  max_depth = 4,
                                  random_state=1)
```

```
# I will use 5-fold CV in the search
```

```
grid1 = GridSearchCV(model,param_grid = params,
                     scoring = 'neg_mean_squared_error',cv = 5)
grid1.fit(X_train,y_train)
-grid1.score(X_test,y_test)
```

```
15.16231211322207
```

Test MSE

```
grid1.best_params_
```

```
{'learning_rate': 0.2575}
```

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 AR.
- Fit Random Forest models with 25 and 500 trees (and max_features = 4). Which predictors are found 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.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
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 30 columns

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

```
X.shape
```

```
(569, 30)
```

p = 30

Cancer dataset – Single tree vs Bagging

Single Tree (max_depth = 4)

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

AR = `0.916083916083916`

Bagging on B = 500 Classification Trees

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

Bagging $p = 30$

AR = `0.9300699300699301`

Cancer dataset – Bagging 500, 25 trees

Bagging on B = 500 Classification Trees

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

```
0.9300699300699301
```

Changing B=500 to B=25 trees

small loss

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

```
0.9230769230769231
```

Random Forest: max_features < *p*

```
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
```

Random Forest 500 vs 25 trees

```
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
```

Changing B=500 to B=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
```

Cancer dataset – RF feature importance

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

```
array([0.0392008 , 0.01166103, 0.05184134, 0.04173711, 0.0035946 ,
       0.0101669 , 0.06064789, 0.1038886 , 0.00269401, 0.00280126,
       0.01117846, 0.00446715, 0.01208892, 0.03068875, 0.00265304,
       0.00383106, 0.00567973, 0.00259535, 0.00176844, 0.00177464,
       0.11579474, 0.01350636, 0.14129173, 0.11495622, 0.00969281,
       0.01269216, 0.04033597, 0.13576649, 0.00626952, 0.00473491])
```

```
cancer.feature_names
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

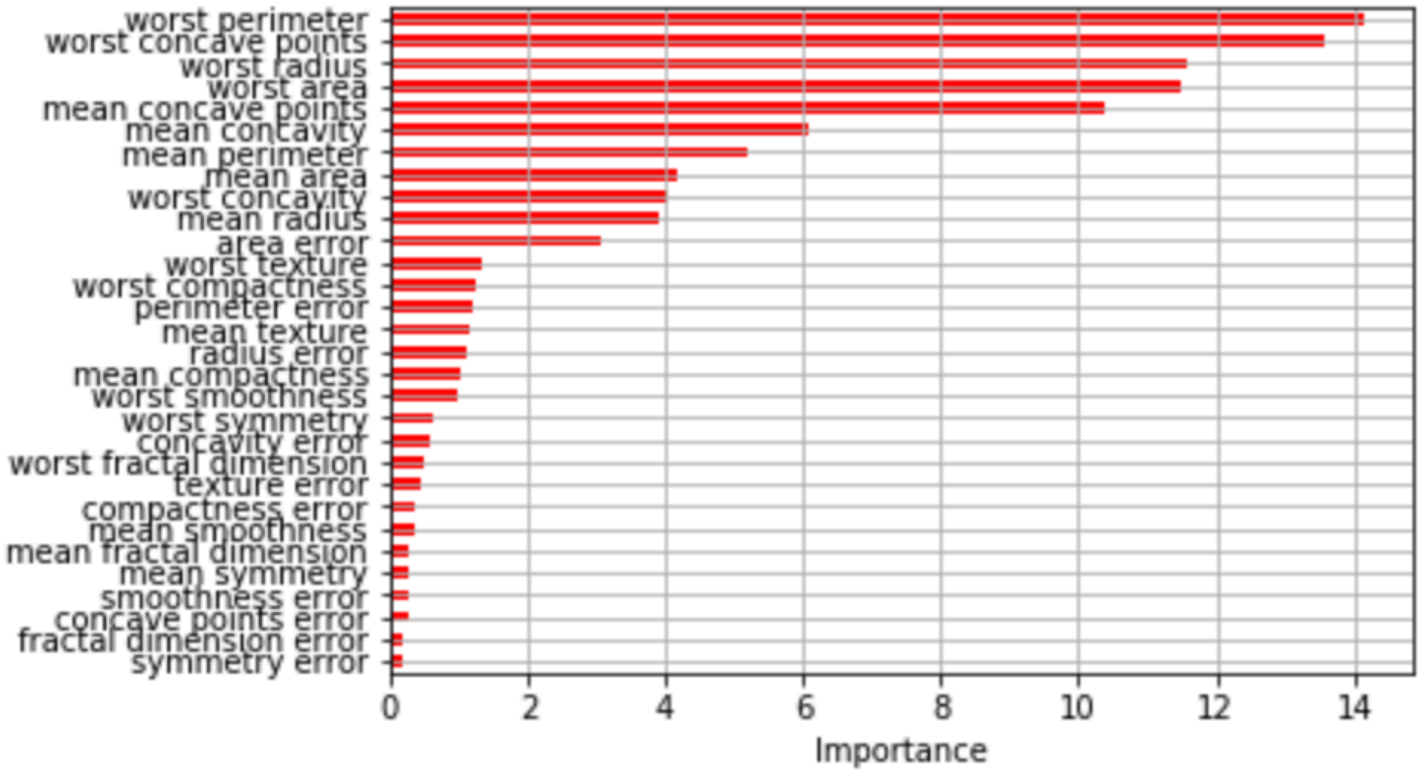
Cancer dataset – RF feature importance

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

	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

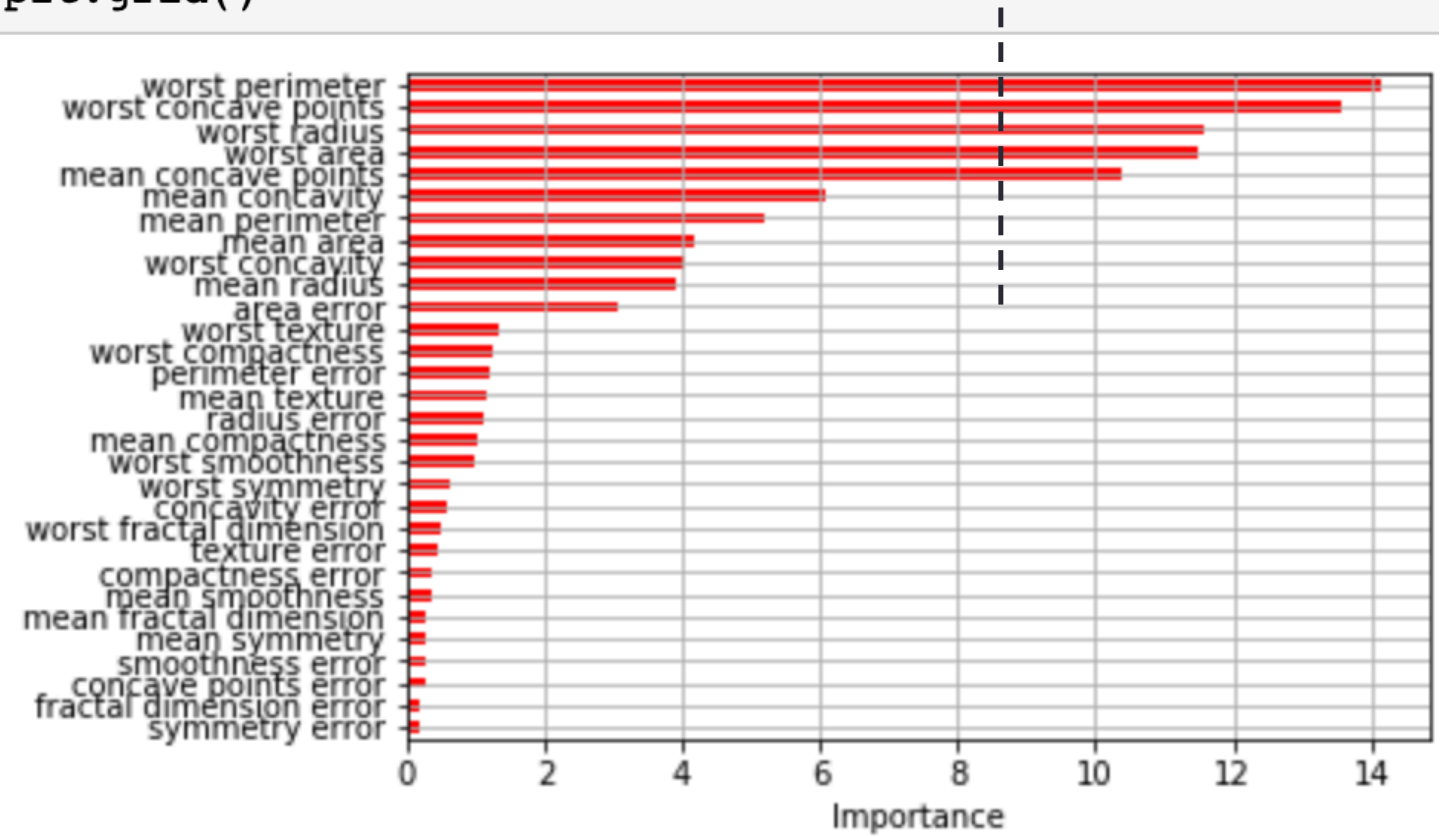
Cancer dataset – Random Forest

```
df9.plot(kind='barh',color='r',legend=False)
plt.xlabel('Importance')
plt.grid()
```



Cancer dataset – Random Forest

```
df9.plot(kind='barh',color='r',legend=False)
plt.xlabel('Importance')
plt.grid()
```



Gradient Boosting Classification Trees

Boosting (0.01 learning rate)

```
# limit the depth of the trees to 4 splits
```

```
model1 = GradientBoostingClassifier(n_estimators = 25,
                                    learning_rate = 0.01,
                                    max_depth = 4, random_state =1)
model1.fit(X_train,y_train)
model1.score(X_test,y_test)
```

```
0.9230769230769231
```

Change 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
```

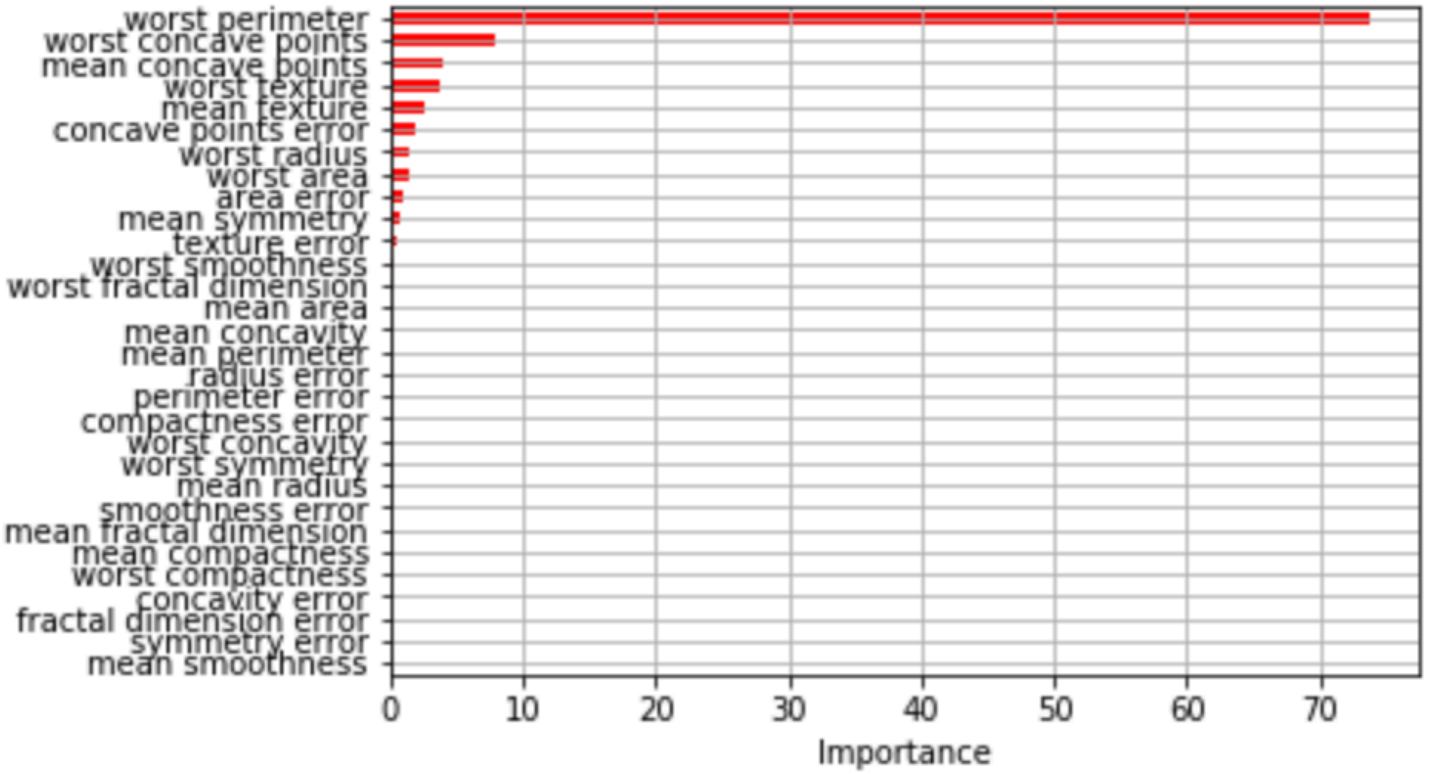
Boosting feature importance

```
# feature importance from model2
df9 = pd.DataFrame(model2.feature_importances_*100,
                   index = cancer.feature_names,
                   columns = ['Importance'])
df9 = df9.sort_values(by='Importance',axis=0,ascending = False)
df9[:11]
```

	Importance
worst perimeter	73.709612
worst concave points	7.801738
mean concave points	4.010190
worst texture	3.783620
mean texture	2.604245
concave points error	1.915741
worst radius	1.541126
worst area	1.366006
area error	1.000423
mean symmetry	0.706863
texture error	0.420015

Cancer dataset – Boosting feature importance

```
df9 = df9.sort_values(by='Importance',axis=0)
df9.plot(kind='barh',color='r',legend=False)
plt.xlabel('Importance')
plt.grid()
```



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(0.01,0.4,8)
lrates
```

```
array([0.01      , 0.06571429, 0.12142857, 0.17714286, 0.23285714,
       0.28857143, 0.34428571, 0.4       ])
```

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

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(0.01,0.4,8)
lrates
```

```
array([0.01      , 0.06571429, 0.12142857, 0.17714286, 0.23285714,
       0.28857143, 0.34428571, 0.4       ])
```

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

```
grid1 = GridSearchCV(model3,param_grid = params,cv = 5)
grid1.fit(X_train,y_train)
grid1.score(X_test,y_test)
```

```
0.9440559440559441
```

Test accuracy rate

```
grid1.best_params_
```

Best learning rate

```
{'learning_rate': 0.12142857142857143}
```

Tuning 2 hyperparameters

```
# Consider 6 values for each parameter
params = {'learning_rate': np.arange(0.02,0.05,0.005),
              'max_features': list(range(3,9))}
params
```

```
{'learning_rate': array([0.02 , 0.025, 0.03 , 0.035, 0.04 , 0.045]),
 'max_features': [3, 4, 5, 6, 7, 8]}
```

```
grid2 = GridSearchCV(model3, param_grid = params,cv = 5)
grid2.fit(X_train,y_train)
grid2.score(X_test,y_test)
```

```
0.958041958041958
```

```
grid2.best_params_
```

```
{'learning_rate': 0.04000000000000001, 'max_features': 8}
```

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]
```

	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

```
# Show accuracy rate for each combination of: max_features, learning_rate
df1 = df9.pivot_table('mean_test_score',
                      columns = 'param_learning_rate',
                      index = 'param_max_features')
df1
```

param_learning_rate	0.020	0.025	0.030	0.035	0.040	0.045
param_max_features						
3	0.941423	0.943776	0.946101	0.943748	0.946074	0.948427
4	0.953160	0.953160	0.957866	0.960192	0.960192	0.957839
5	0.950807	0.953133	0.960164	0.957811	0.962490	0.962490
6	0.950752	0.953105	0.948399	0.955458	0.964843	0.962490
7	0.941341	0.948372	0.948345	0.955404	0.957756	0.957811
8	0.948372	0.955458	0.962462	0.962462	0.967168	0.964788

GridSearchCV on 2 hyperparameters

```
# column names
df1.columns
```

```
Float64Index([0.02, 0.025, 0.03, 0.035, 0.04, 0.045], dtype='float64',
```

```
# row names
df1.index
```

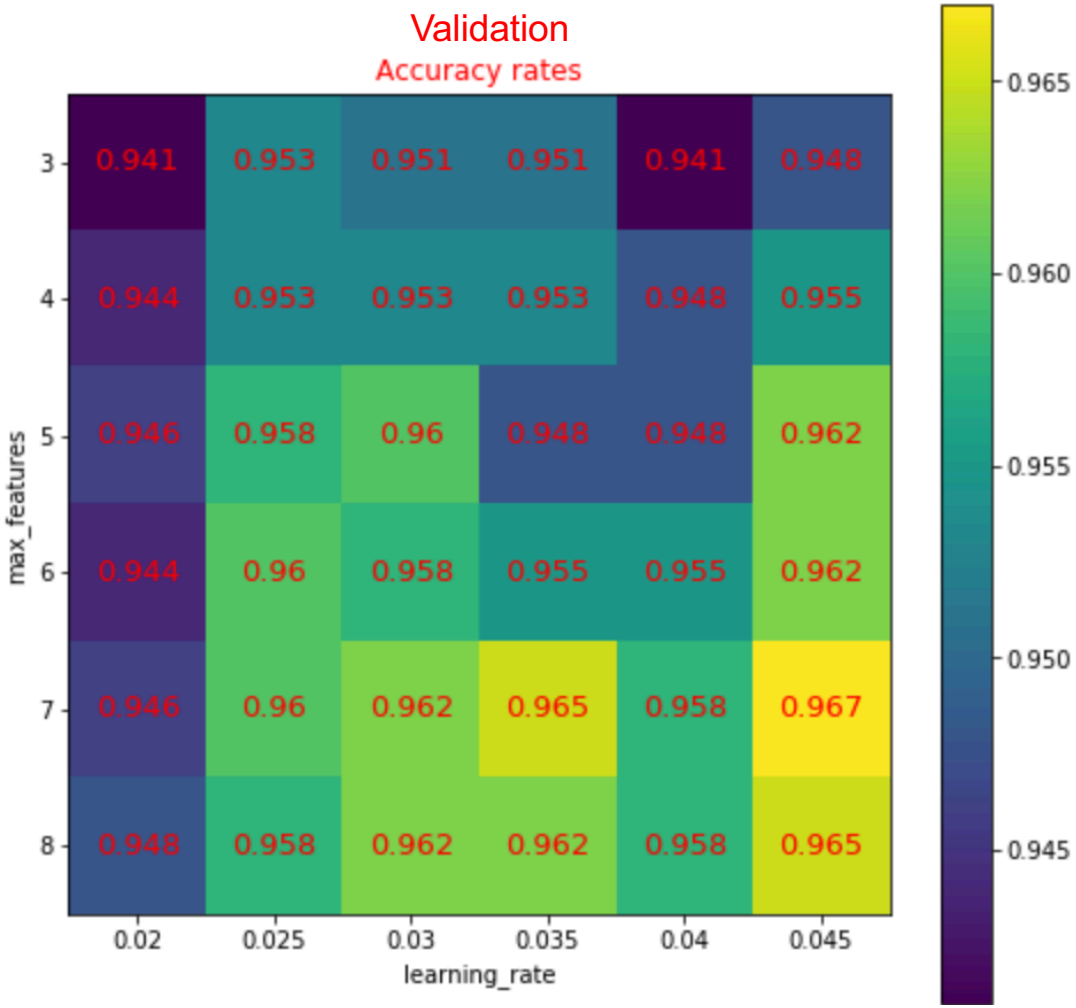
```
Int64Index([3, 4, 5, 6, 7, 8], dtype='int64', name='param_max_features'
```

```
# transform df1 dataframe to numpy array
arates = df1.values
arates = np.round(arates,3)
arates = arates.transpose()
arates
```

```
array([[0.941, 0.953, 0.951, 0.951, 0.941, 0.948],
       [0.944, 0.953, 0.953, 0.953, 0.948, 0.955],
       [0.946, 0.958, 0.96 , 0.948, 0.948, 0.962],
       [0.944, 0.96 , 0.958, 0.955, 0.955, 0.962],
       [0.946, 0.96 , 0.962, 0.965, 0.958, 0.967],
       [0.948, 0.958, 0.962, 0.962, 0.958, 0.965]])
```

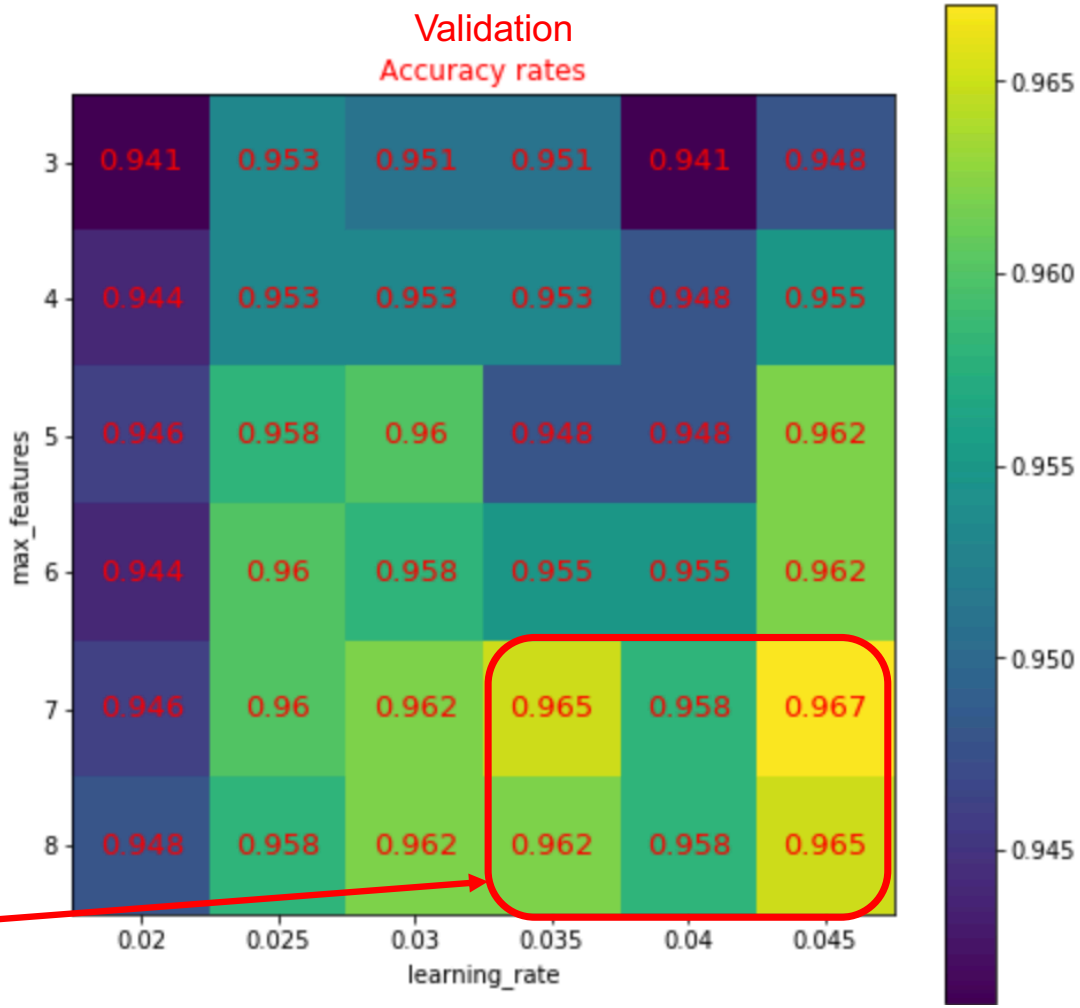
GridSearchCV results on a Heatmap

```
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();
```



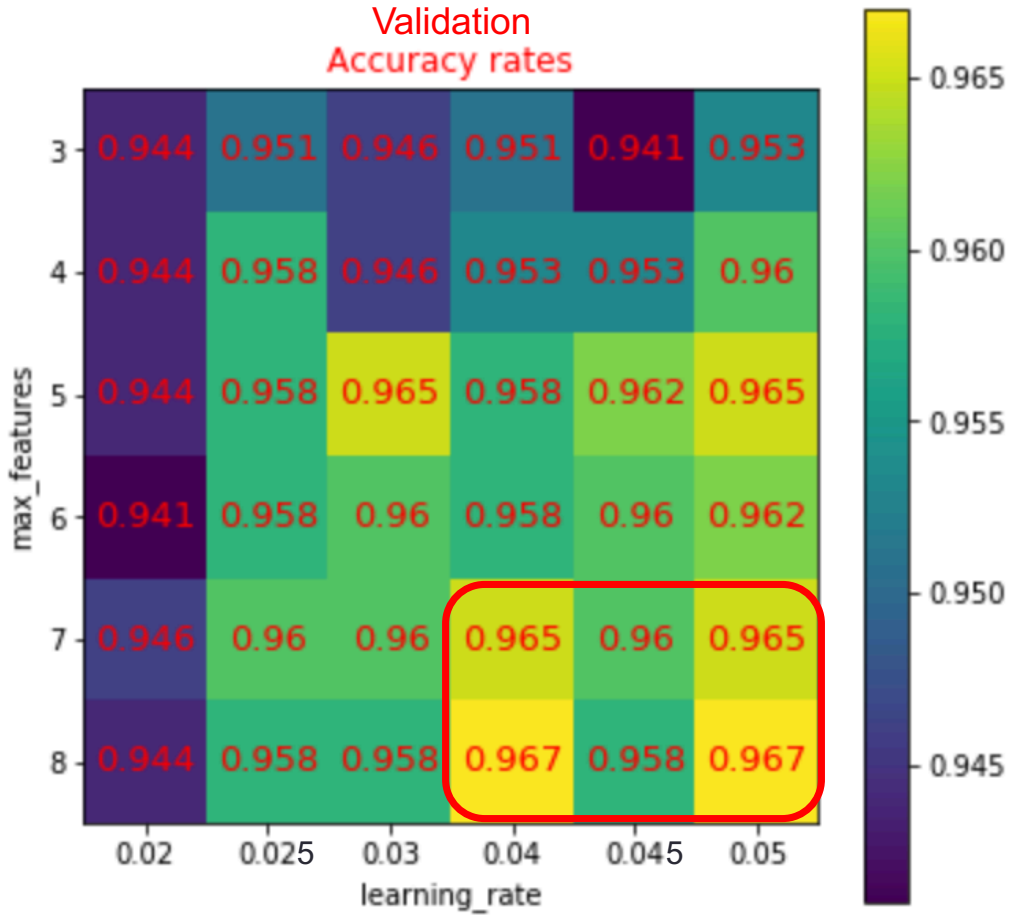
GridSearchCV results on a Heatmap

```
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

```
plt.figure(figsize=(6,6))
plt.xticks(range(6),colnames)
plt.yticks(range(6),rownames)
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();
```



best region for parameter values