CART Models

Introduction to CART Models

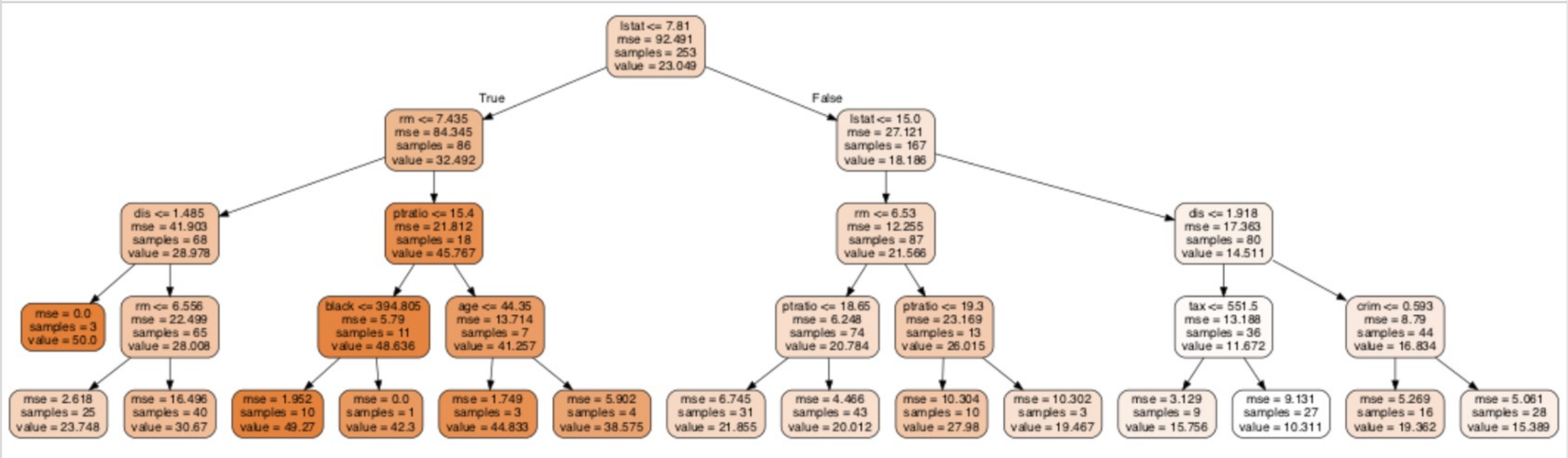
Machine learning trees are binary trees (usually called decision trees) that may be used for prediction and classification.

Their principle is to break the learning process into a set of decisions about a predictor in a sequential manner.

The process can be displayed by the help of a tree diagram.

The process starts at the root of the tree and moving down to the leaves, where the prediction or classification is made.

Tree diagram



Types of CART Models

- Regression Trees

    From sklearn.tree import DecisionTreeRegressor( )

- Classification Trees

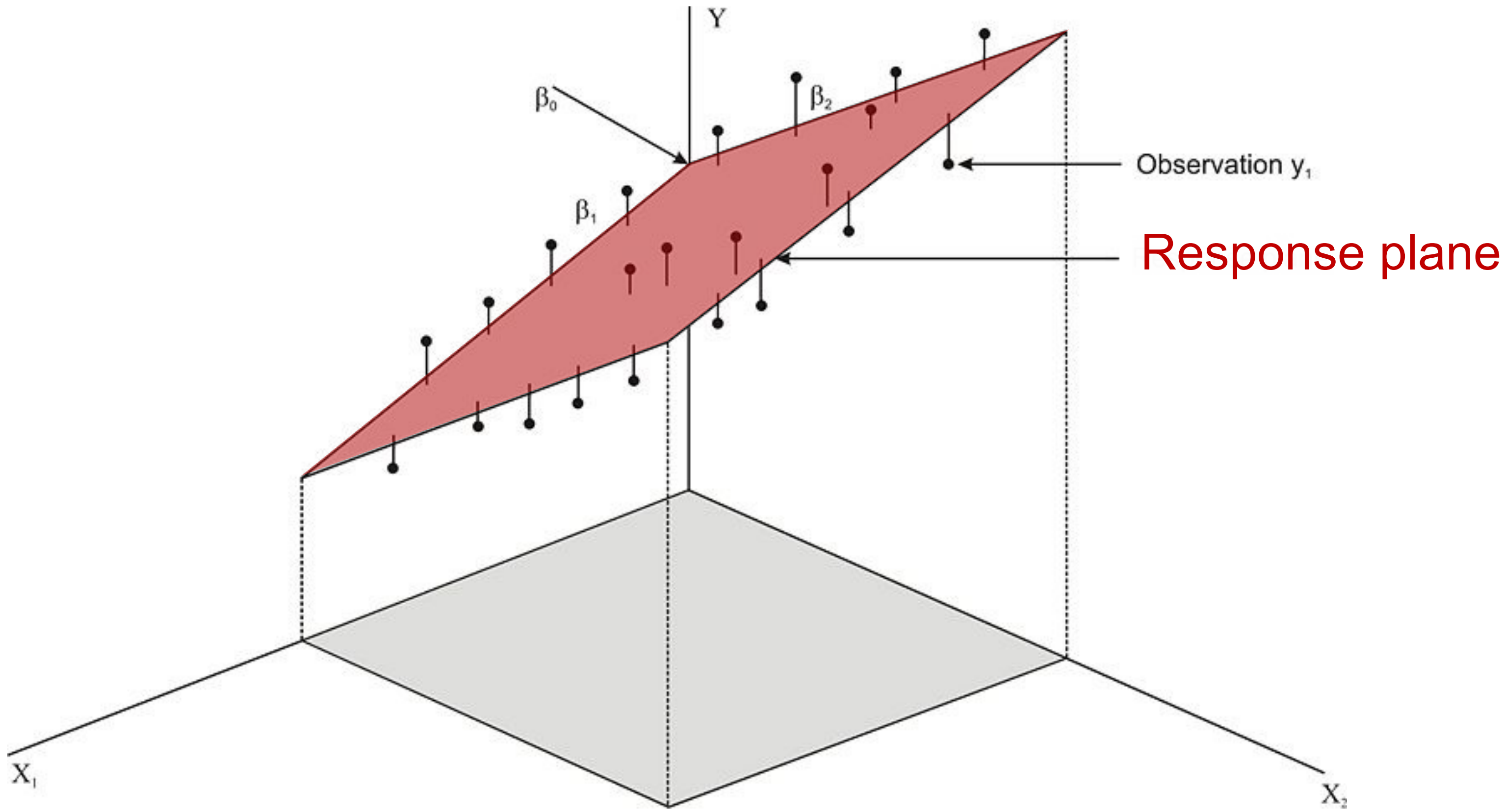
    From sklearn.tree import DecisionTreeClassifier( )

Overview

- Difference
    - Linear Regression
    - Regression Tree
- How to use the Tree
    - For Prediction
    - For feature selection
- How to construct the Tree
- Notes
- Example – Boston dataset

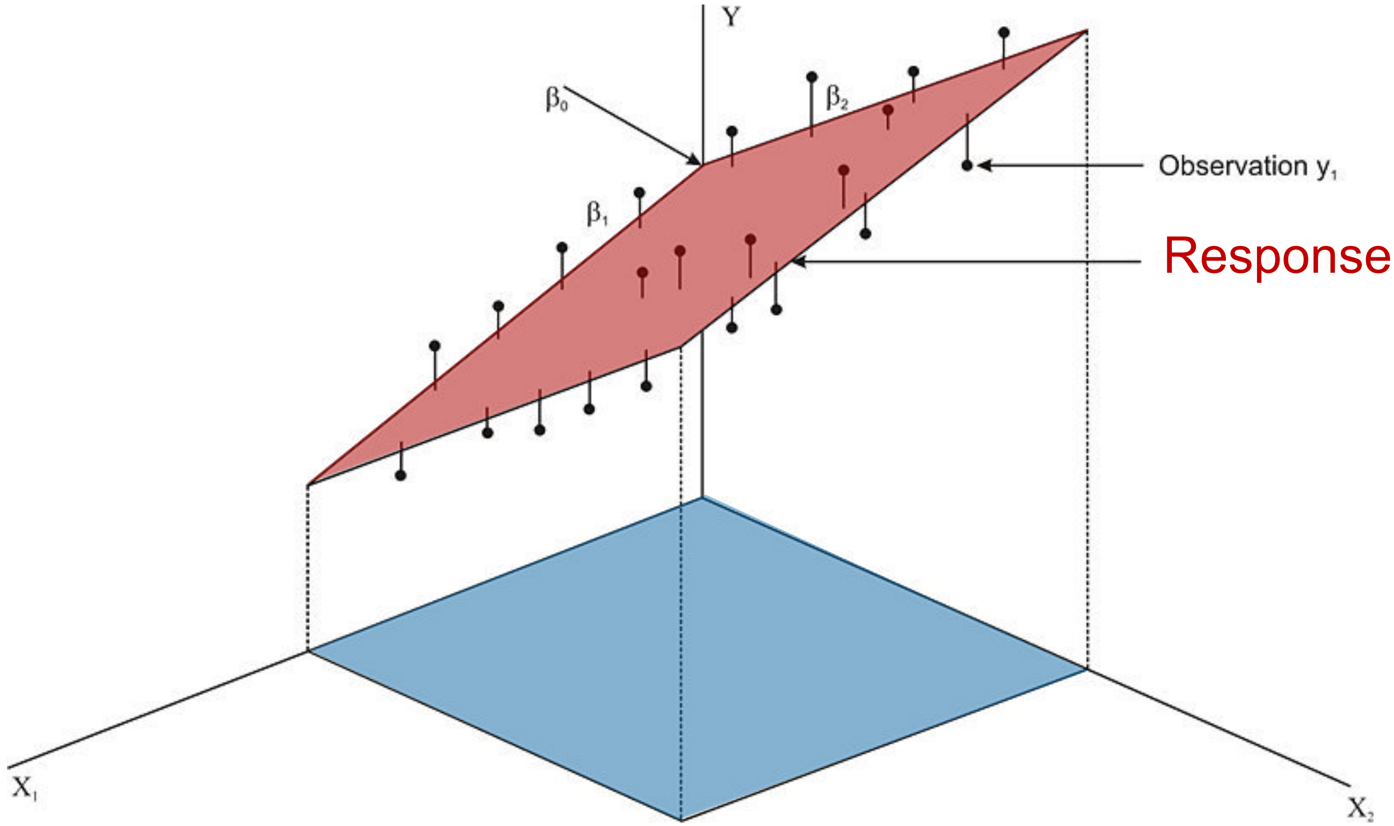
Multiple Linear Regression

.



Predictors Space

.



Predictors space
(set of (x_1, x_2) coordinates
of all data points)

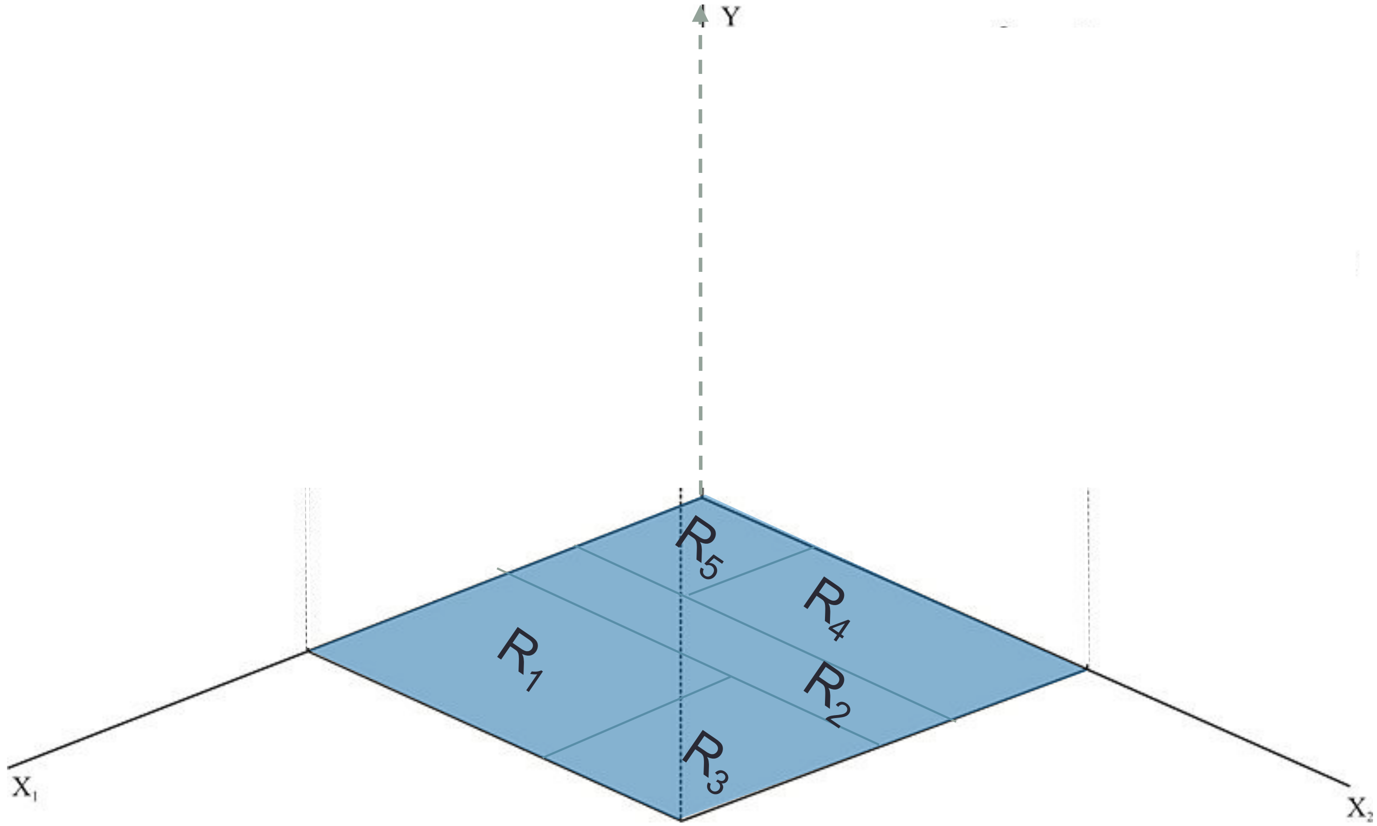
Predictors Space

Partitioning the Predictor Space

- Split the predictors space into non-overlapping regions R_1, R_2,…, R_k

Predictors Space

.



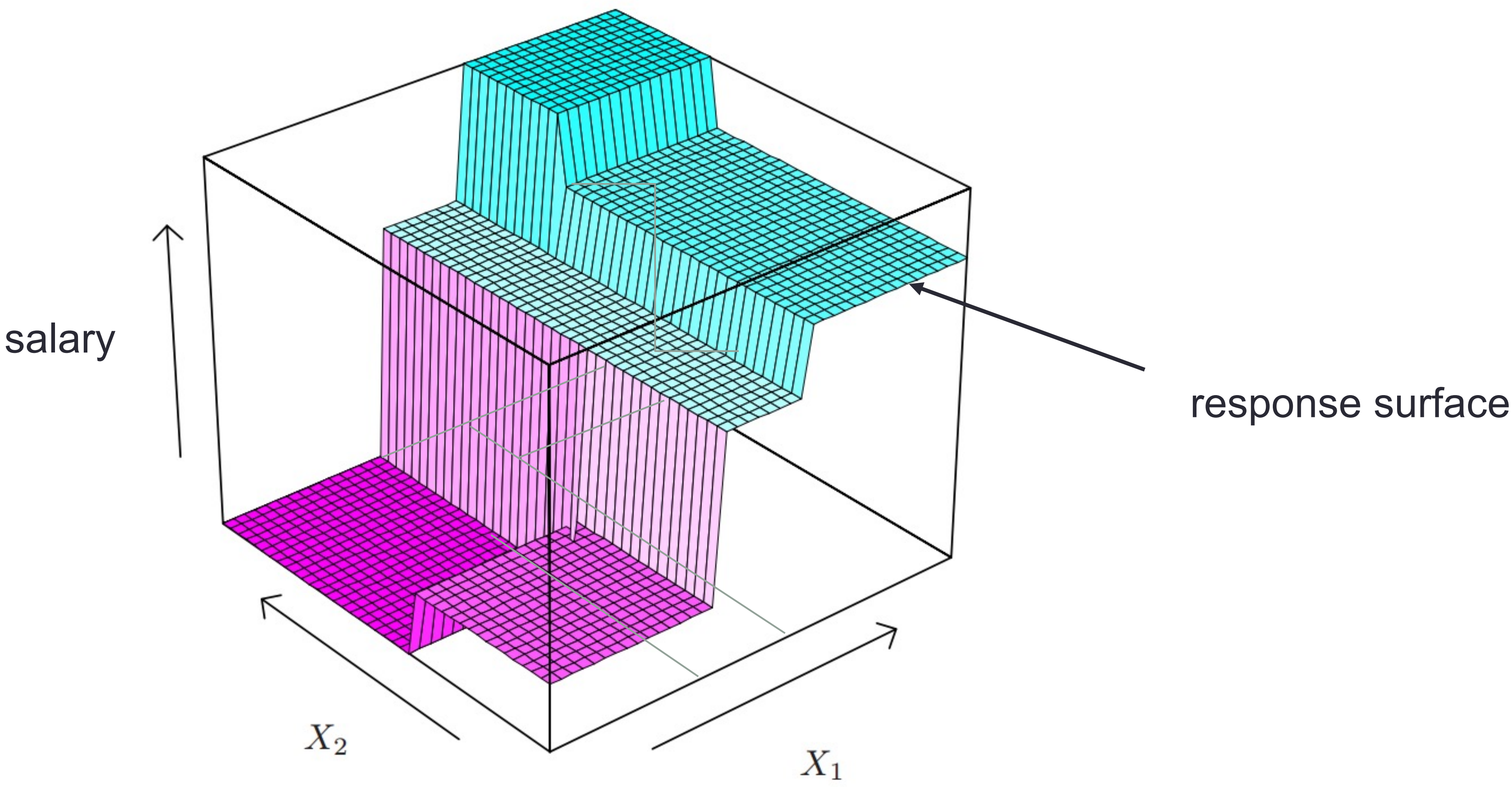
Partition of predictors space

Predictors Space

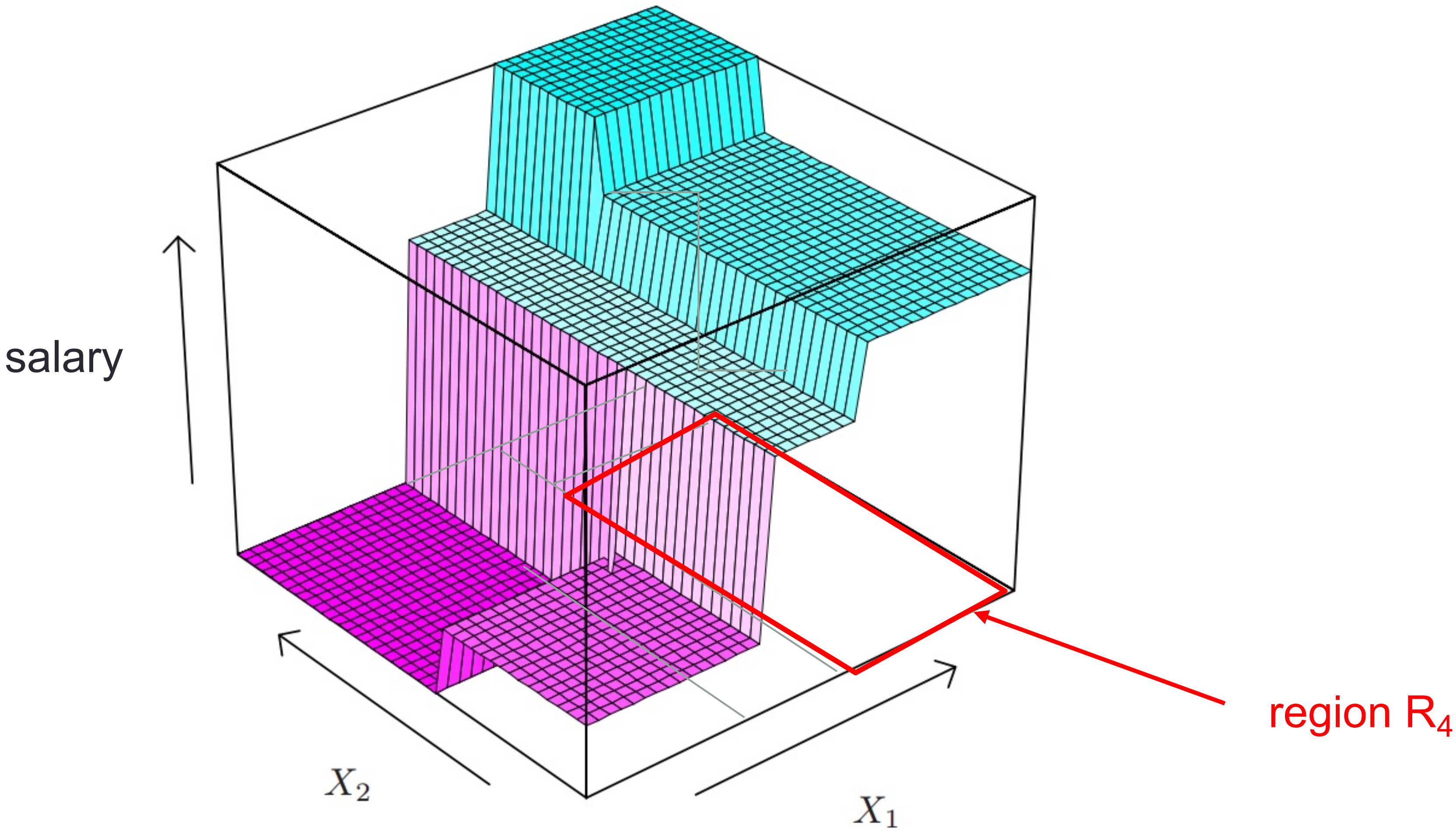
Partitioning the Predictor Space

- Split the predictors space into non-overlapping regions $R_1, R_2, \ldots, R_k$
- For each region, the prediction is the average response of the observations in that region

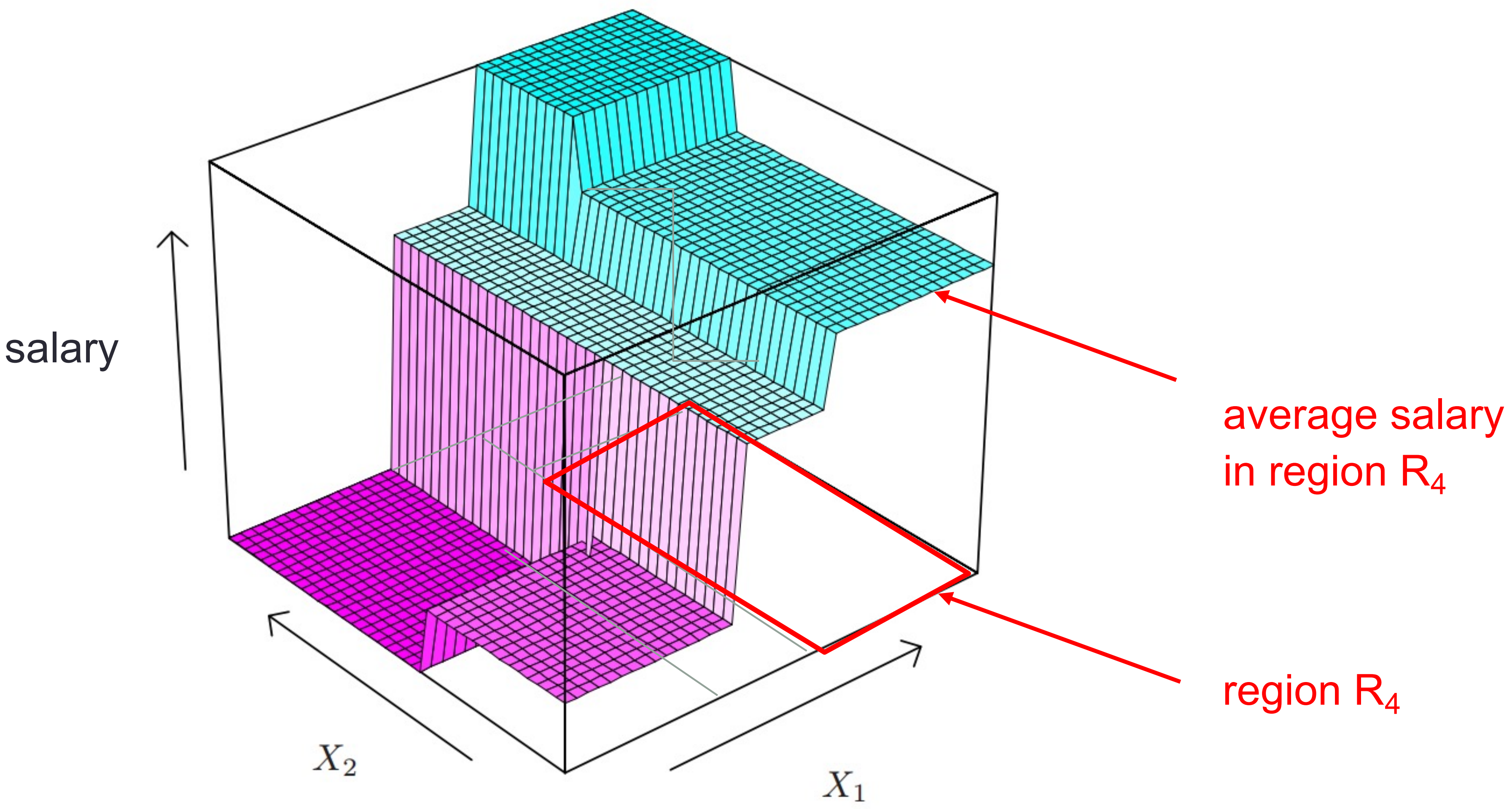
Regression Tree



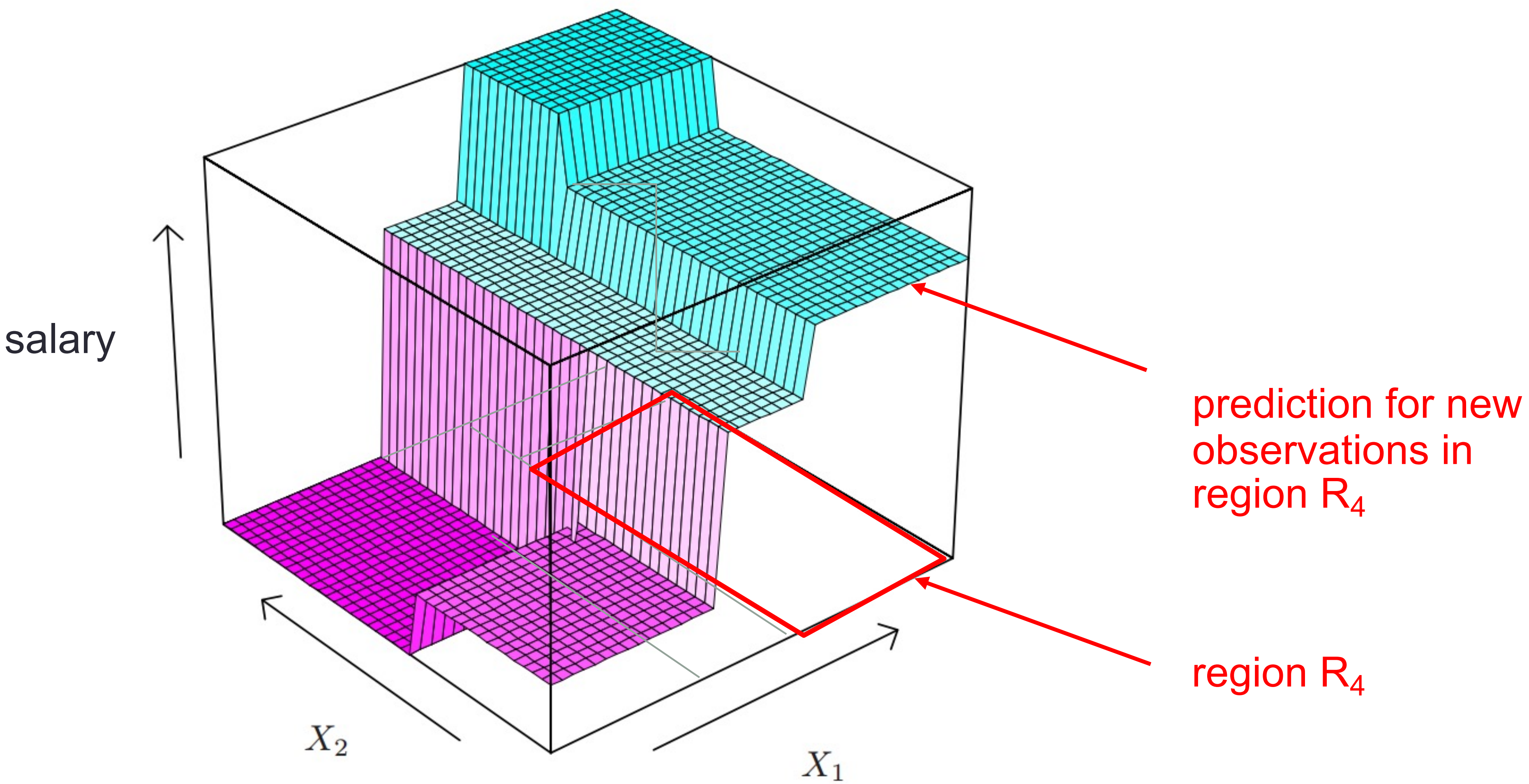
Regression Tree



Regression Tree



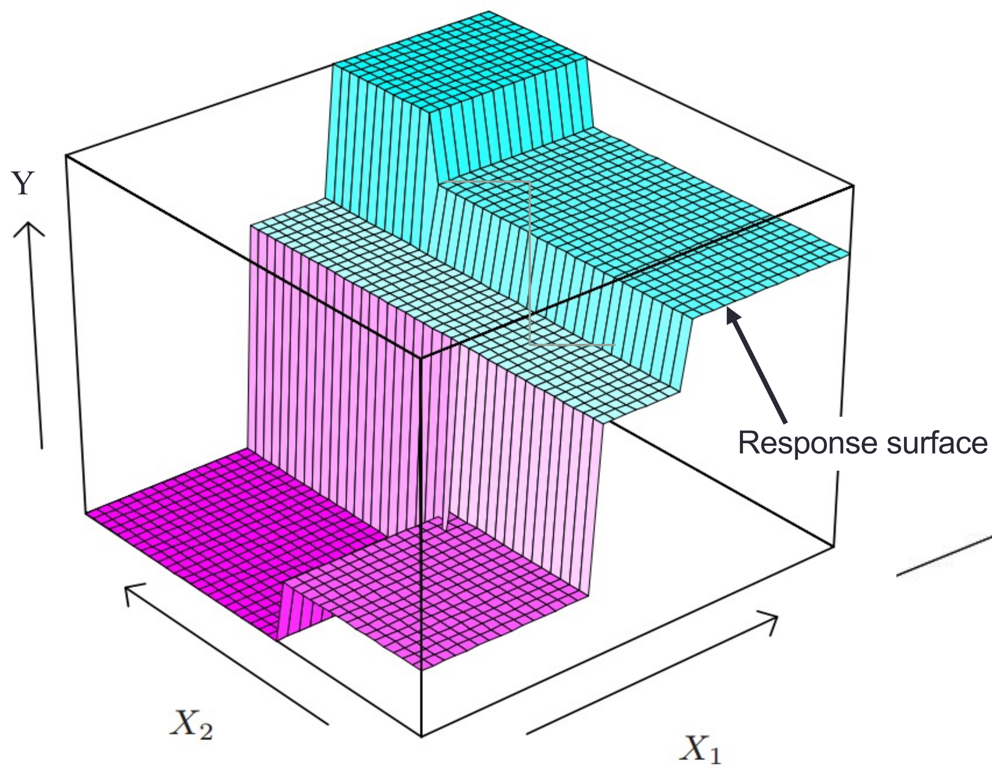
Regression Tree



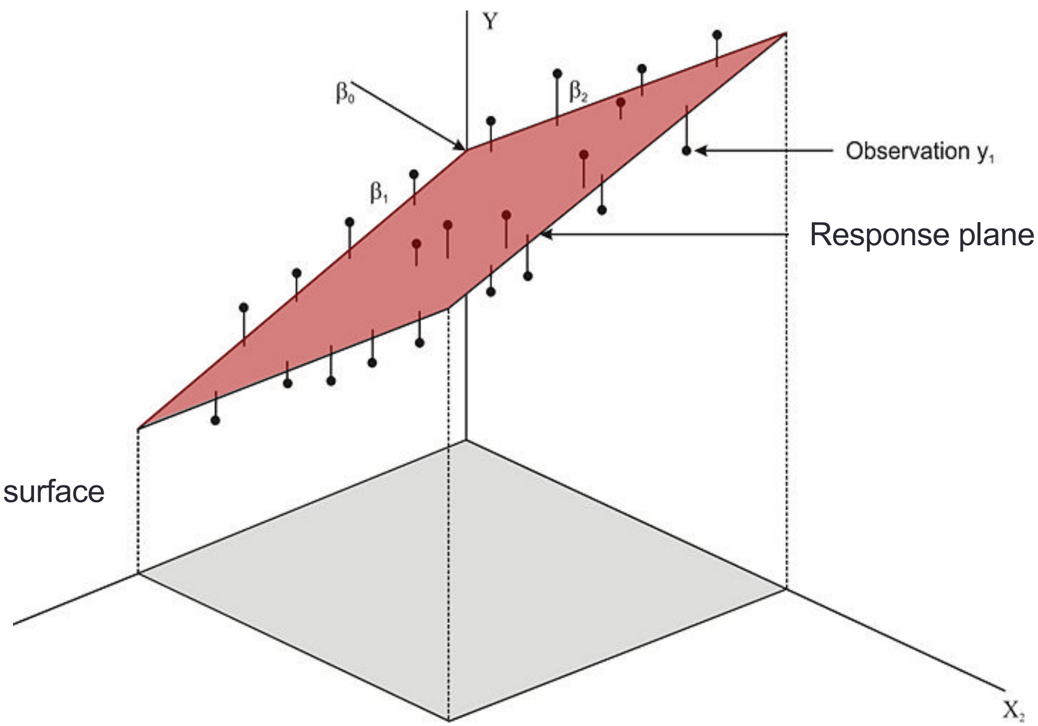
Regression Trees

If the observations in Region R_1
have average response 100,
we would predict 100,
for any new observation that falls in R_1

Regression Tree



Linear Regression



Example: Baseball Players

Salary	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
475	81	7	24	38	39	14	3449	835	69	321
480	130	18	66	72	76	3	1624	457	63	224
500	141	20	65	78	37	11	5628	1575	225	828
91.5	87	10	39	42	30	2	396	101	12	48
750	169	4	74	51	35	11	4408	1133	19	501
70	37	1	23	8	21	2	214	42	1	30
100	73	0	24	24	7	3	509	108	0	41
75	81	6	26	32	8	2	341	86	6	32
1100	92	17	49	66	65	13	5206	1332	253	784
517.143	159	21	107	75	59	10	4631	1300	90	702
512.5	53	4	31	26	27	9	1876	467	15	192
550	113	13	48	61	47	4	1512	392	41	205
700	60	0	30	11	22	6	1941	510	4	309
240	43	7	29	27	30	13	3231	825	36	376
775	158	20	89	75	73	15	8068	2273	177	1045
175	46	2	24	8	15	5	479	102	5	65
135	32	8	16	22	14	8	727	180	24	67
100	92	16	72	48	65	1	413	92	16	72

Example: Baseball Players dataset

Y

19 Predictors (features)

Salary	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
475	81	7	24	38	39	14	3449	835	69	321
480	130	18	66	72	76	3	1624	457	63	224
500	141	20	65	78	37	11	5628	1575	225	828
91.5	87	10	39	42	30	2	396	101	12	48
750	169	4	74	51	35	11	4408	1133	19	501
70	37	1	23	8	21	2	214	42	1	30
100	73	0	24	24	7	3	509	108	0	41
75	81	6	26	32	8	2	341	86	6	32
1100	92	17	49	66	65	13	5206	1332	253	784
517.143	159	21	107	75	59	10	4631	1300	90	702
512.5	53	4	31	26	27	9	1876	467	15	192
550	113	13	48	61	47	4	1512	392	41	205
700	60	0	30	11	22	6	1941	510	4	309
240	43	7	29	27	30	13	3231	825	36	376
775	158	20	89	75	73	15	8068	2273	177	1045
175	46	2	24	8	15	5	479	102	5	65
135	32	8	16	22	14	8	727	180	24	67
100	92	16	72	48	65	1	413	92	16	72

…

Example: Baseball Players' features

`AtBat`
: Number of times at bat in 1986

`Hits`
: Number of hits in 1986

`HmRun`
: Number of home runs in 1986

`Runs`
: Number of runs in 1986

`RBI`
: Number of runs batted in in 1986

`Walks`
: Number of walks in 1986

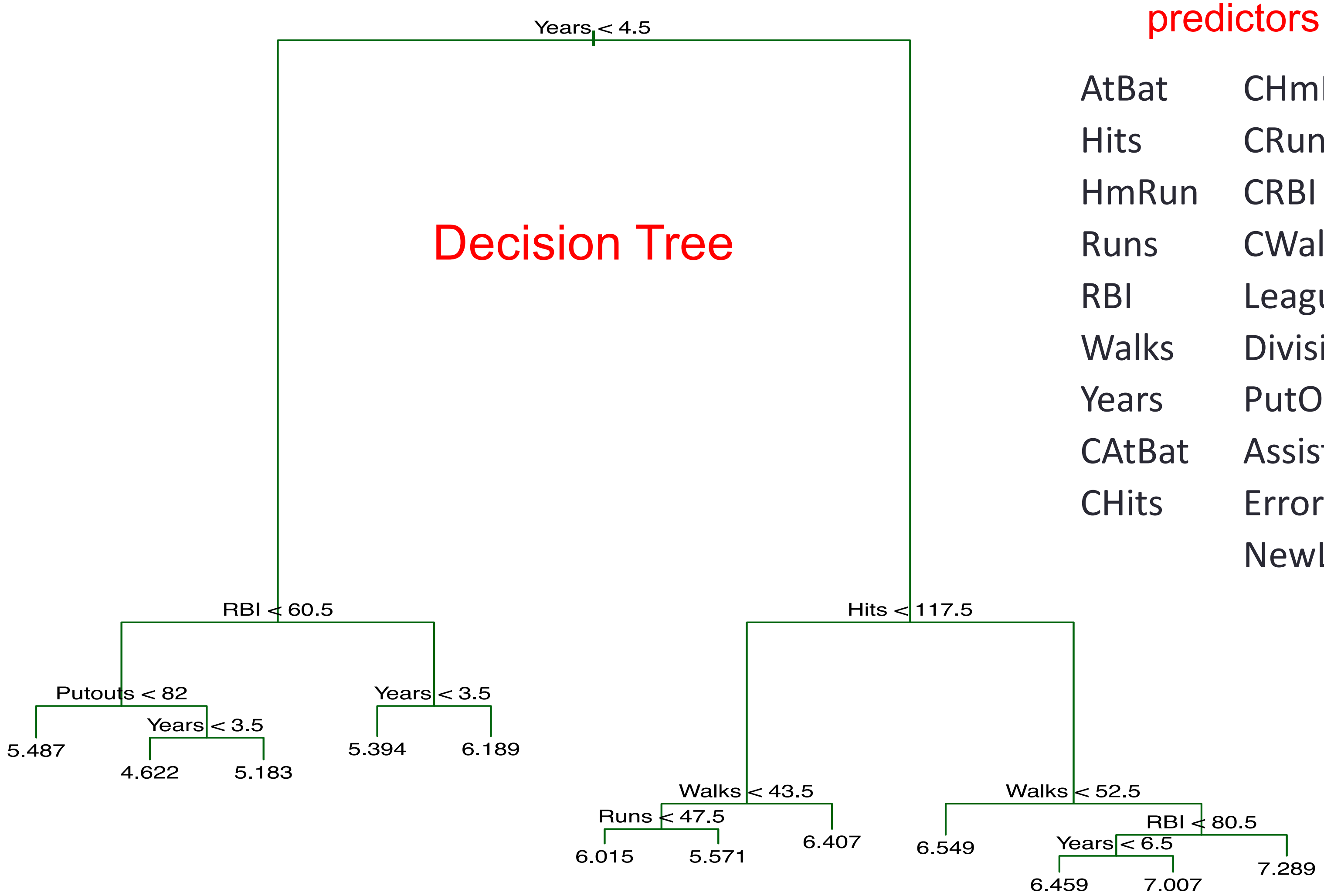
`Years`
: Number of years in the major leagues

`CAtBat`
: Number of times at bat during his career

`CHits`
: Number of hits during his career

`CHmRun`
: Number of home runs during his career

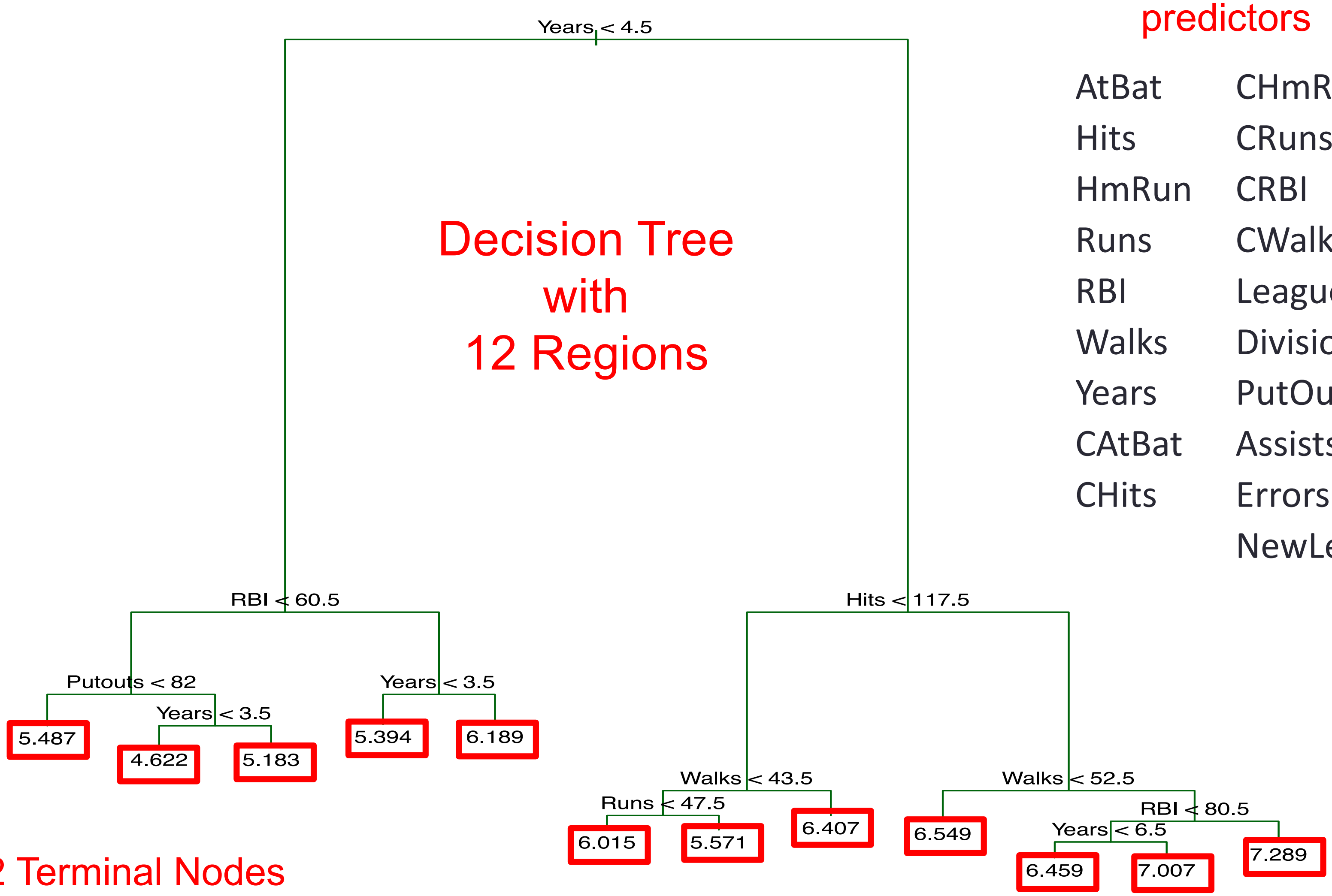
Example: Baseball Players



predictors

AtBat	CHmRun
Hits	CRuns
HmRun	CRBI
Runs	CWalks
RBI	League
Walks	Division
Years	PutOuts
CAtBat	Assists
CHits	Errors
	NewLeague

Example: Baseball Players

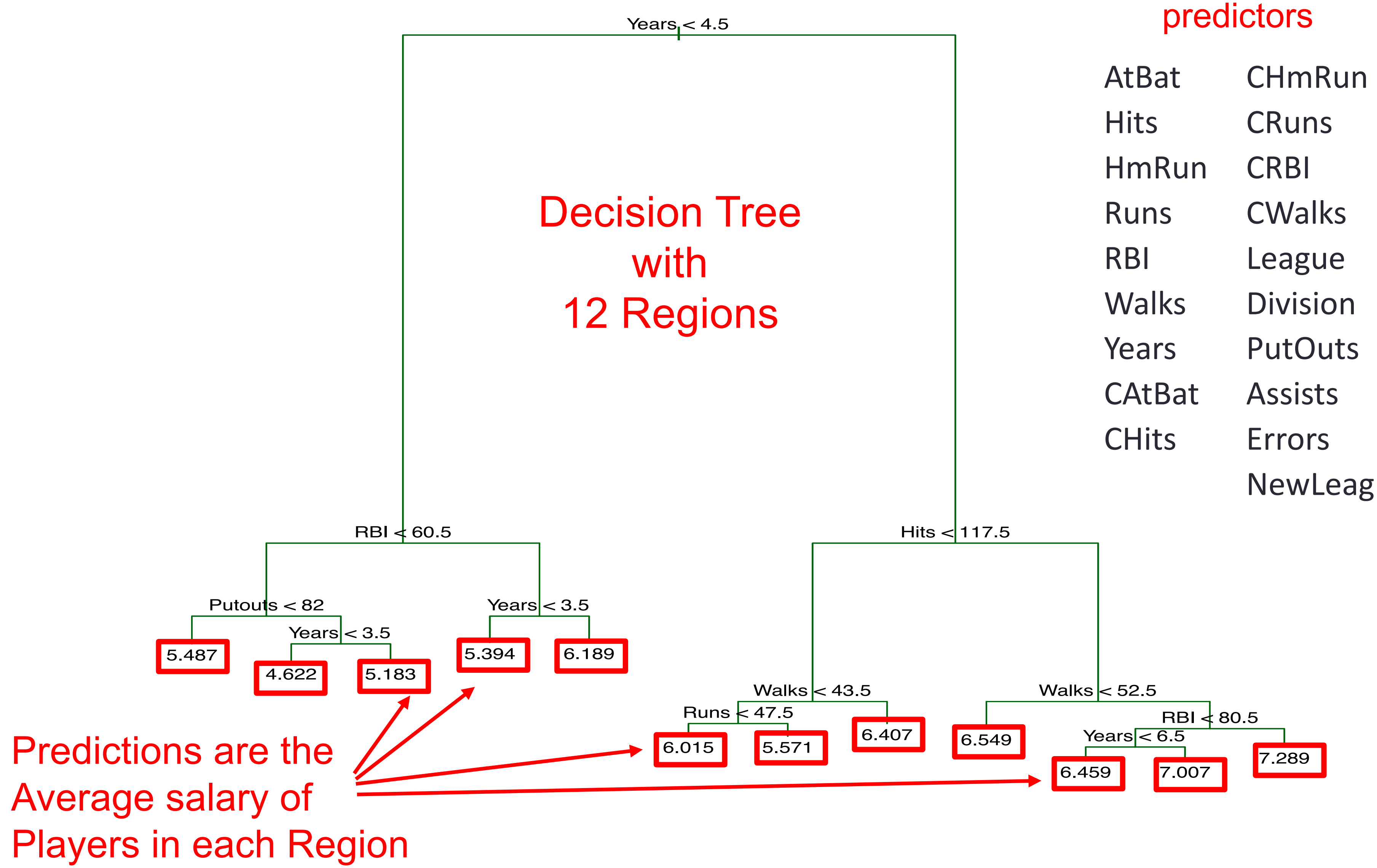


12 Terminal Nodes

predictors

AtBat	CHmRun
Hits	CRuns
HmRun	CRBI
Runs	CWalks
RBI	League
Walks	Division
Years	PutOuts
CAtBat	Assists
CHits	Errors
	NewLeague

Example: Baseball Players

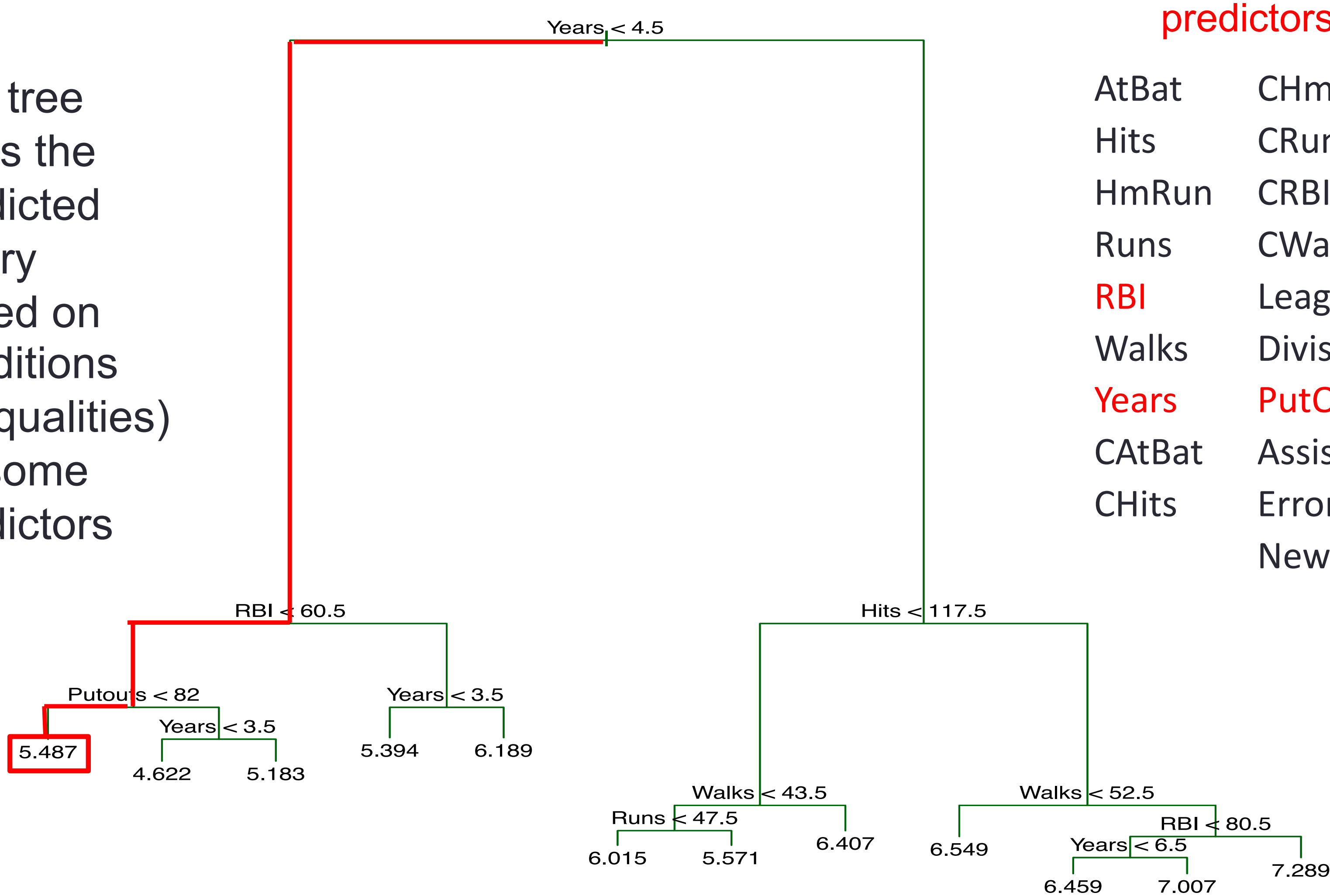


predictors

AtBat	CHmRun
Hits	CRuns
HmRun	CRBI
Runs	CWalks
RBI	League
Walks	Division
Years	PutOuts
CAtBat	Assists
CHits	Errors
	NewLeague

Example: Baseball Players

The tree gives the predicted salary based on conditions (inequalities) on some predictors

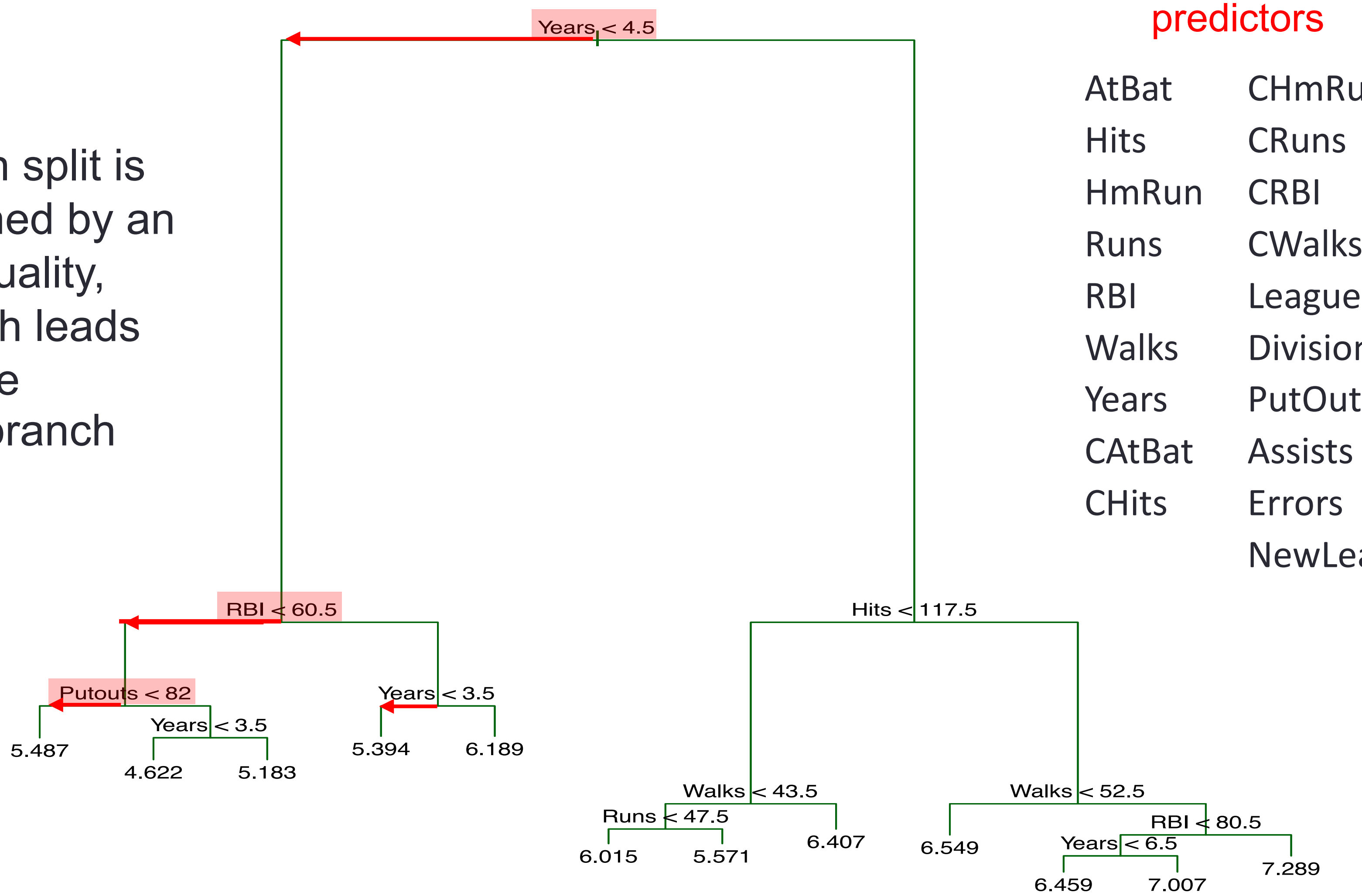


predictors

AtBat	CHmRun
Hits	CRuns
HmRun	CRBI
Runs	CWalks
RBI	League
Walks	Division
Years	PutOuts
CAtBat	Assists
CHits	Errors
	NewLeague

Example: Baseball Players

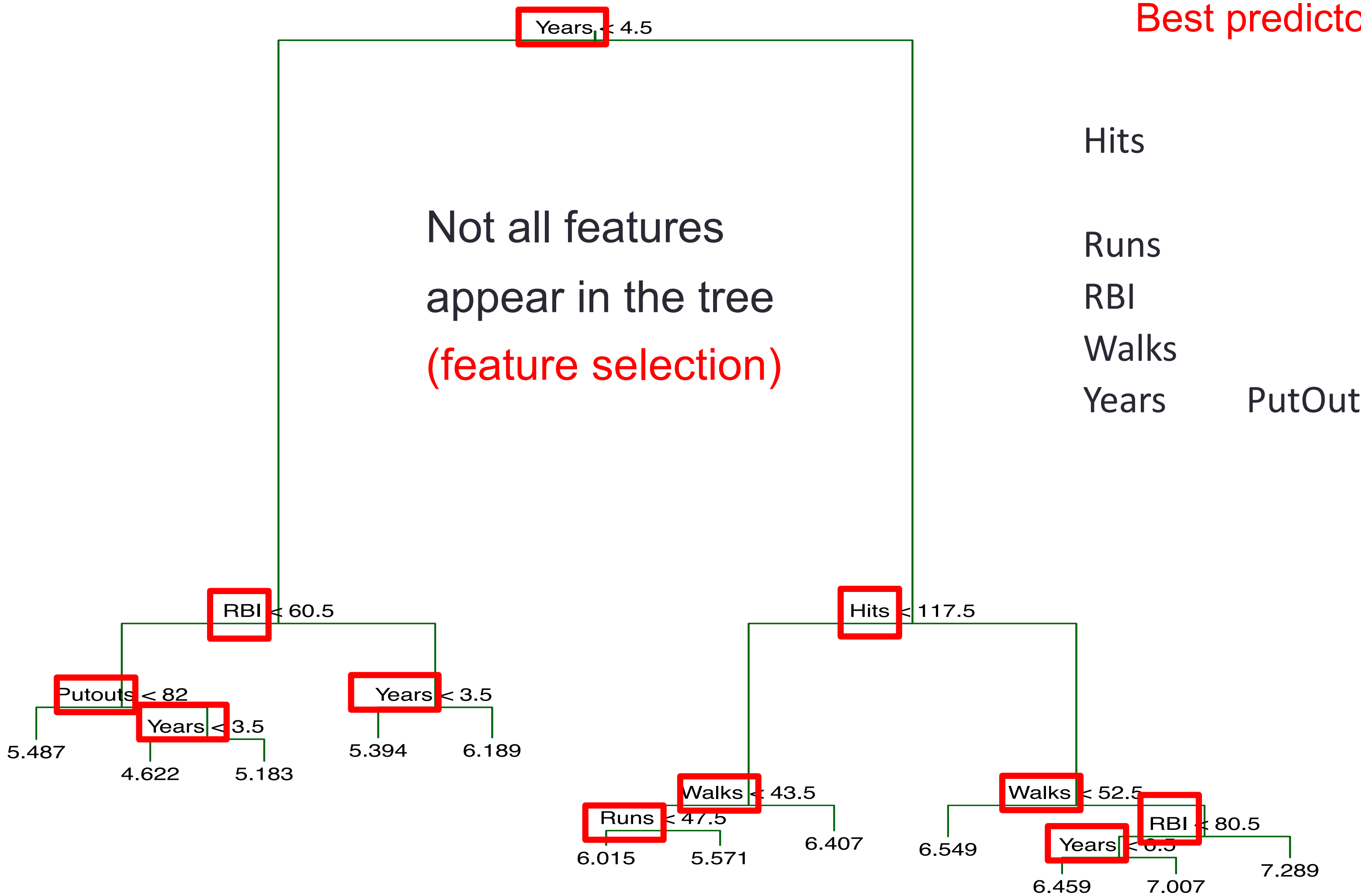
Each split is defined by an inequality, which leads to the left branch



predictors

AtBat	CHmRun
Hits	CRuns
HmRun	CRBI
Runs	CWalks
RBI	League
Walks	Division
Years	PutOuts
CAtBat	Assists
CHits	Errors
	NewLeague

Tree as a **Feature selection** tool



Not all features appear in the tree (feature selection)

Best predictors

Hits

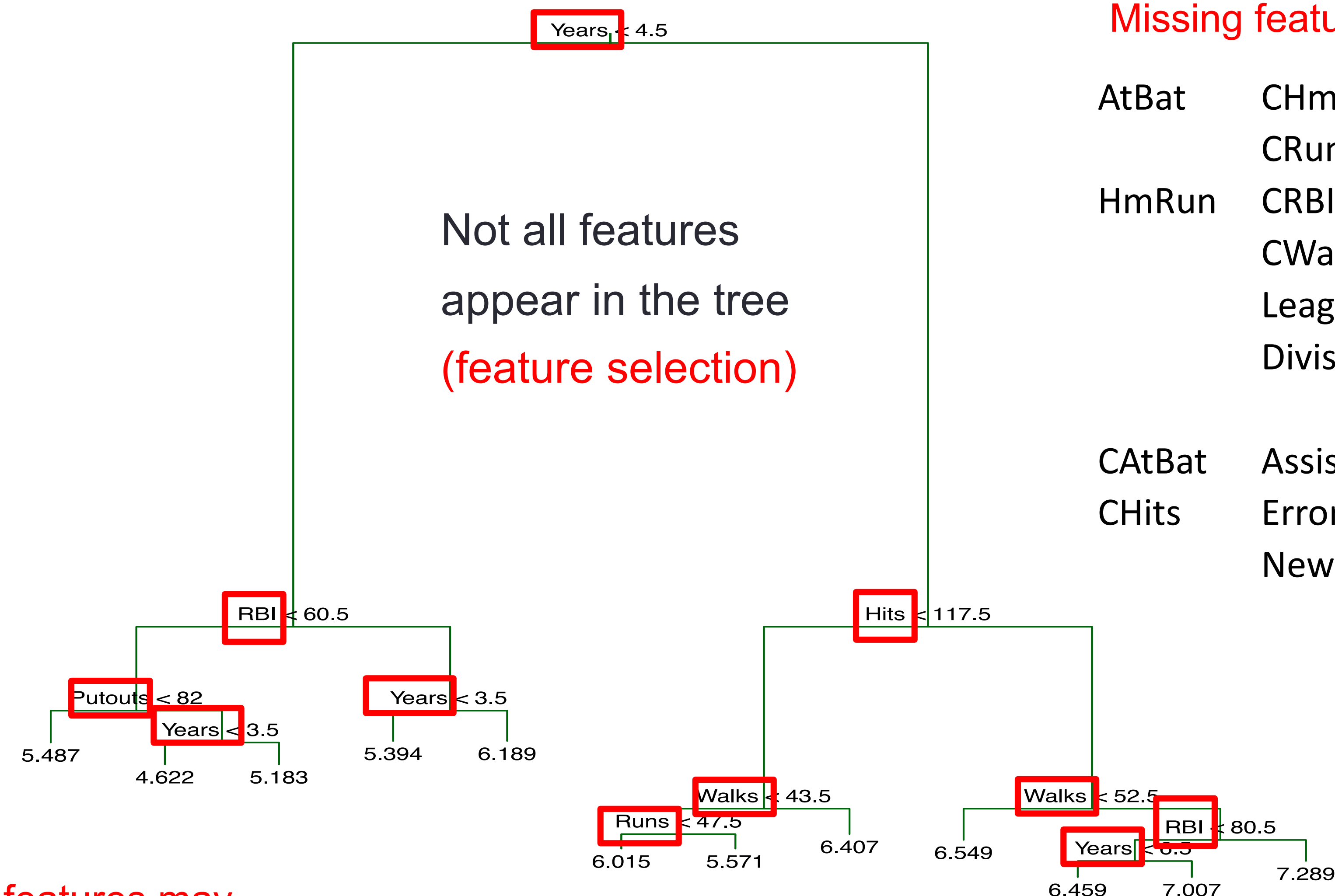
Runs

RBI

Walks

Years PutOuts

Tree as a **Feature selection** tool



Not all features appear in the tree (feature selection)

Missing features

AtBat	CHmRun
	CRuns
HmRun	CRBI
	CWalks
	League
	Division
CAtBat	Assists
CHits	Errors
	NewLeague

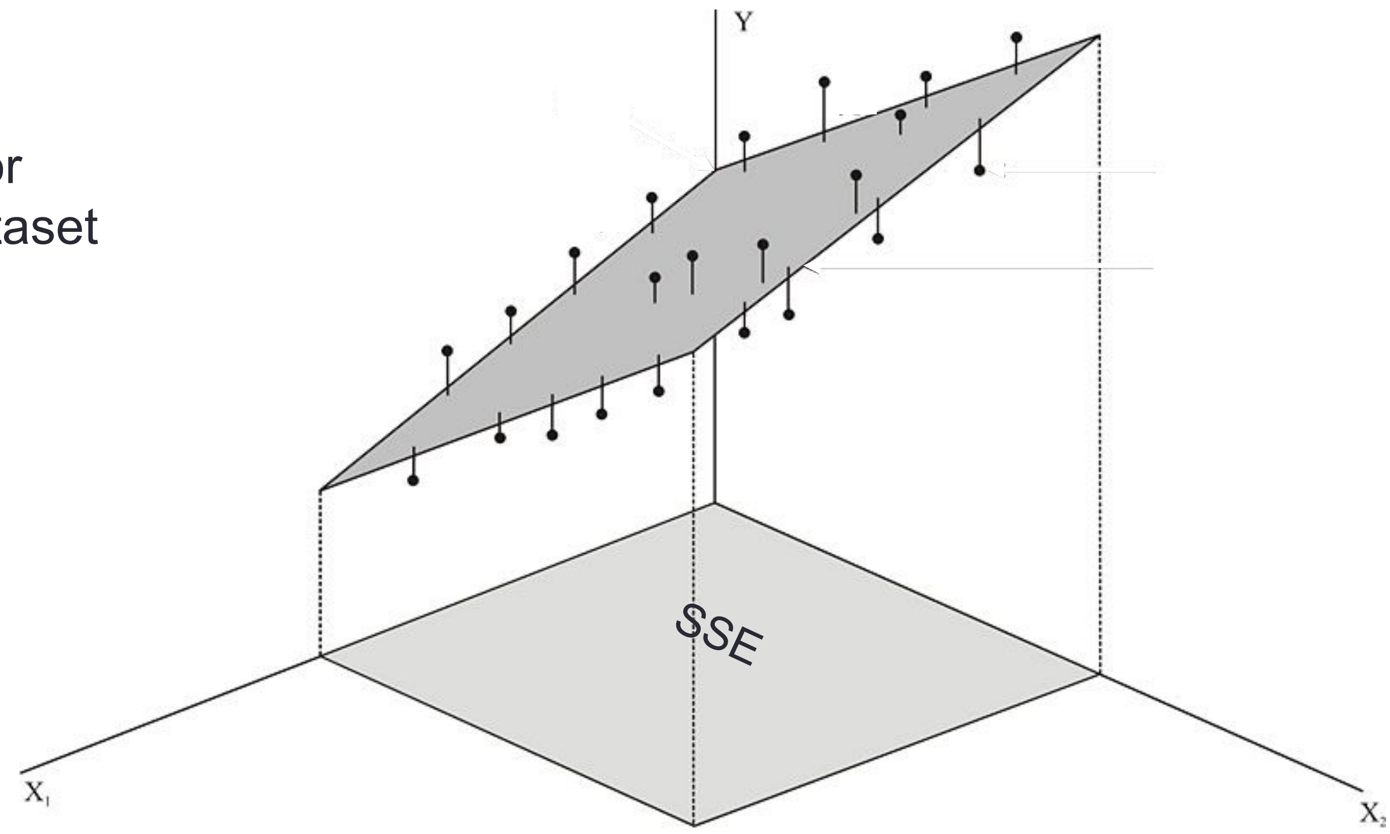
Missing features may be dropped from model

Regression Trees

How to build a Tree?

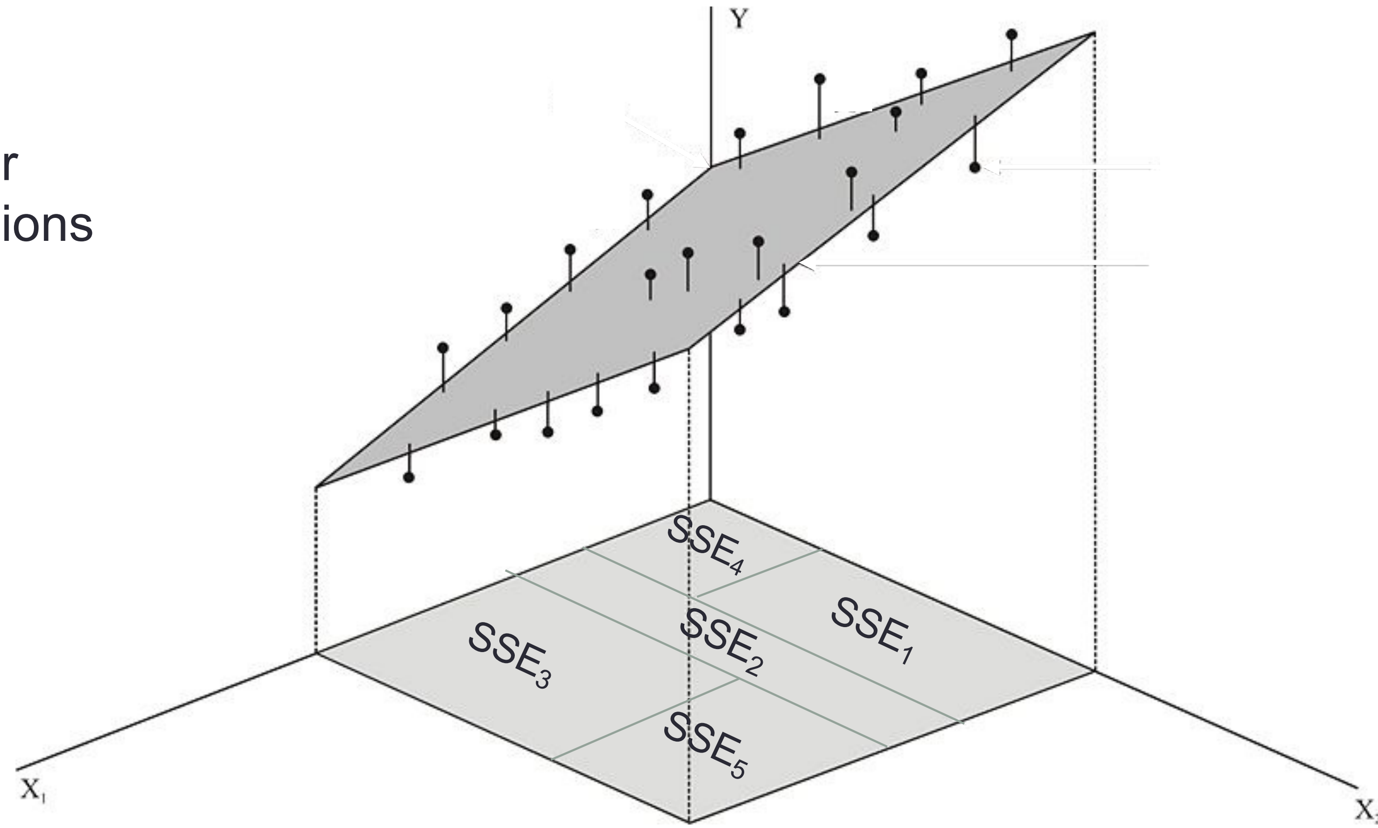
Linear Regression vs. Regression Tree

SSE for
the dataset



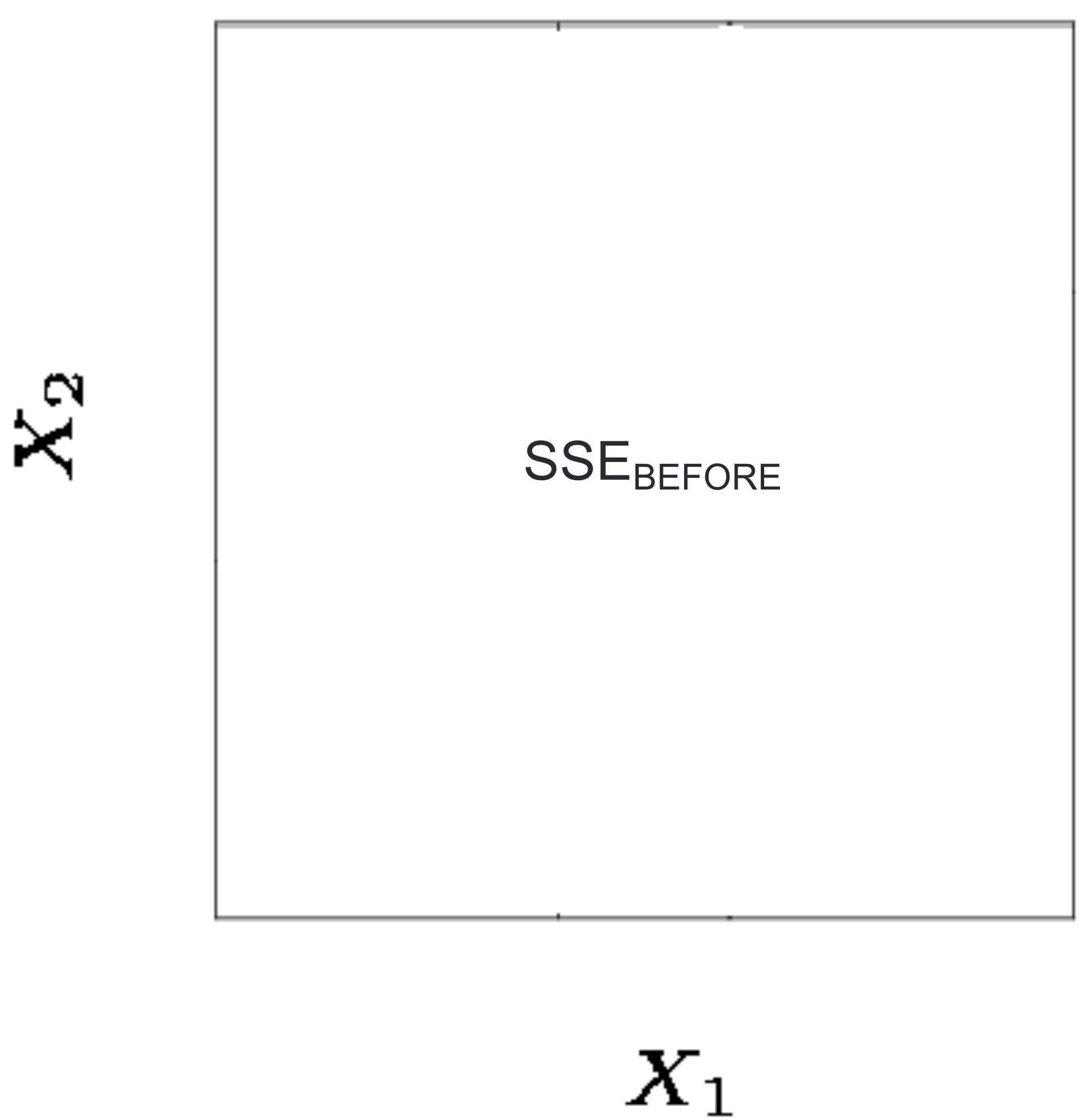
Linear Regression vs. Regression Tree

SSE for
the regions



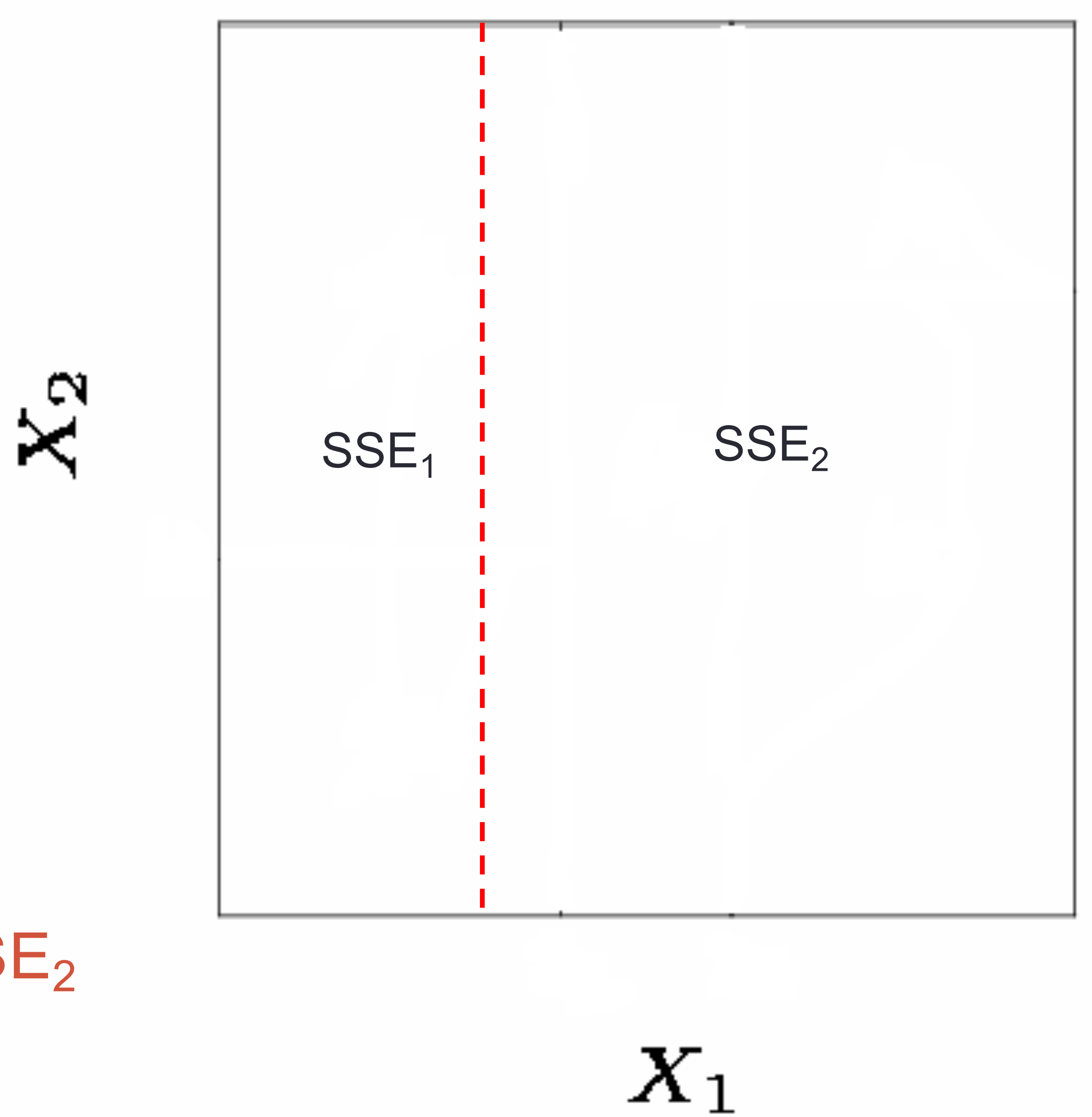
Partitioning Up the Predictor Space

- SSE_{BEFORE} is the SSE that results from finding the average of all observations



Partitioning Up the Predictor Space

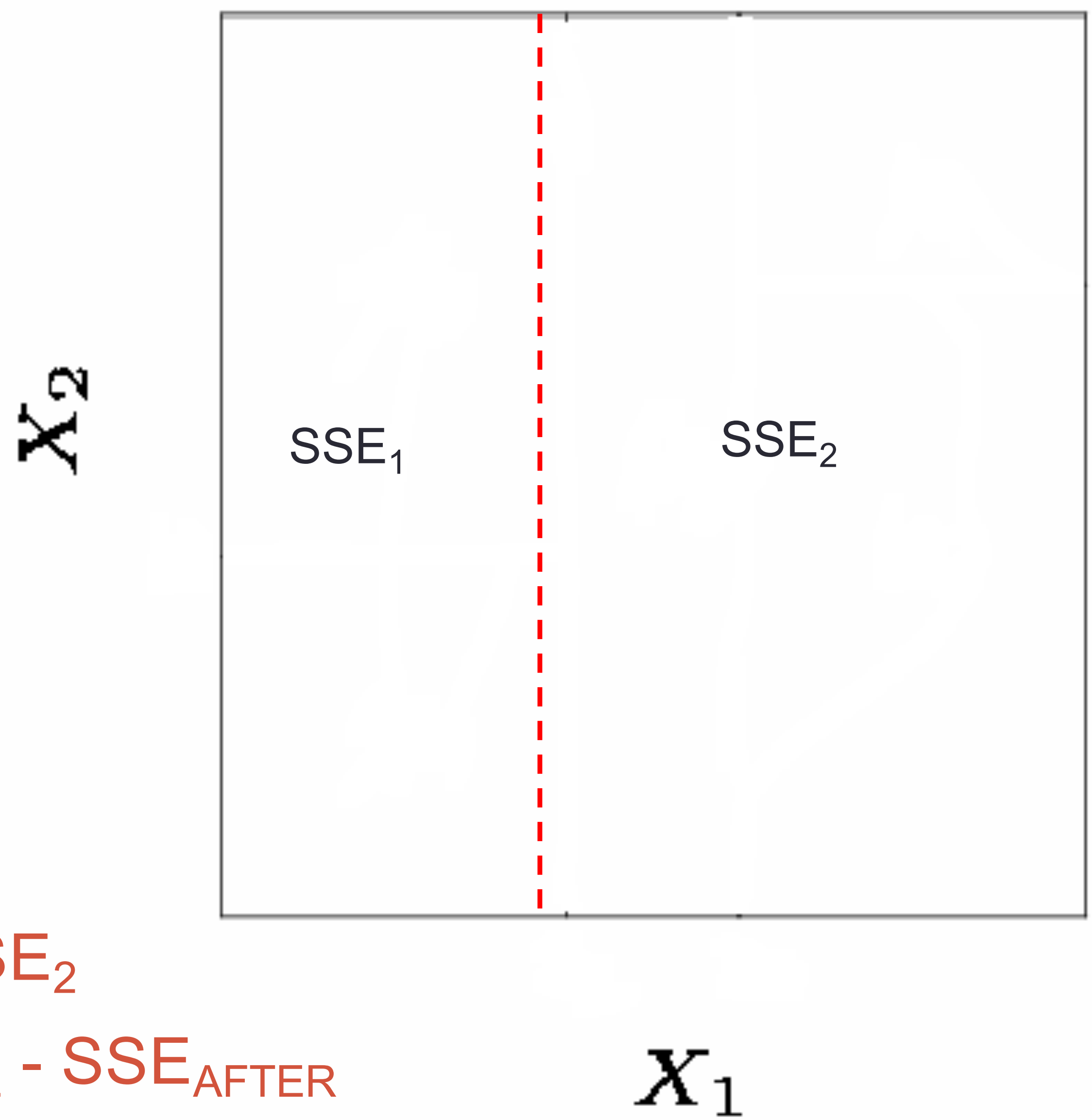
- SSE_1 is the SSE that results from finding the average of observations falling in Region 1
- Similarly, SSE_2 results from the observations falling in Region 2



$SSE_{AFTER} = SSE_1 + SSE_2$

Partitioning Up the Predictor Space

- Any split creates new SSE_1, SSE_2
- Try different splits on X_1

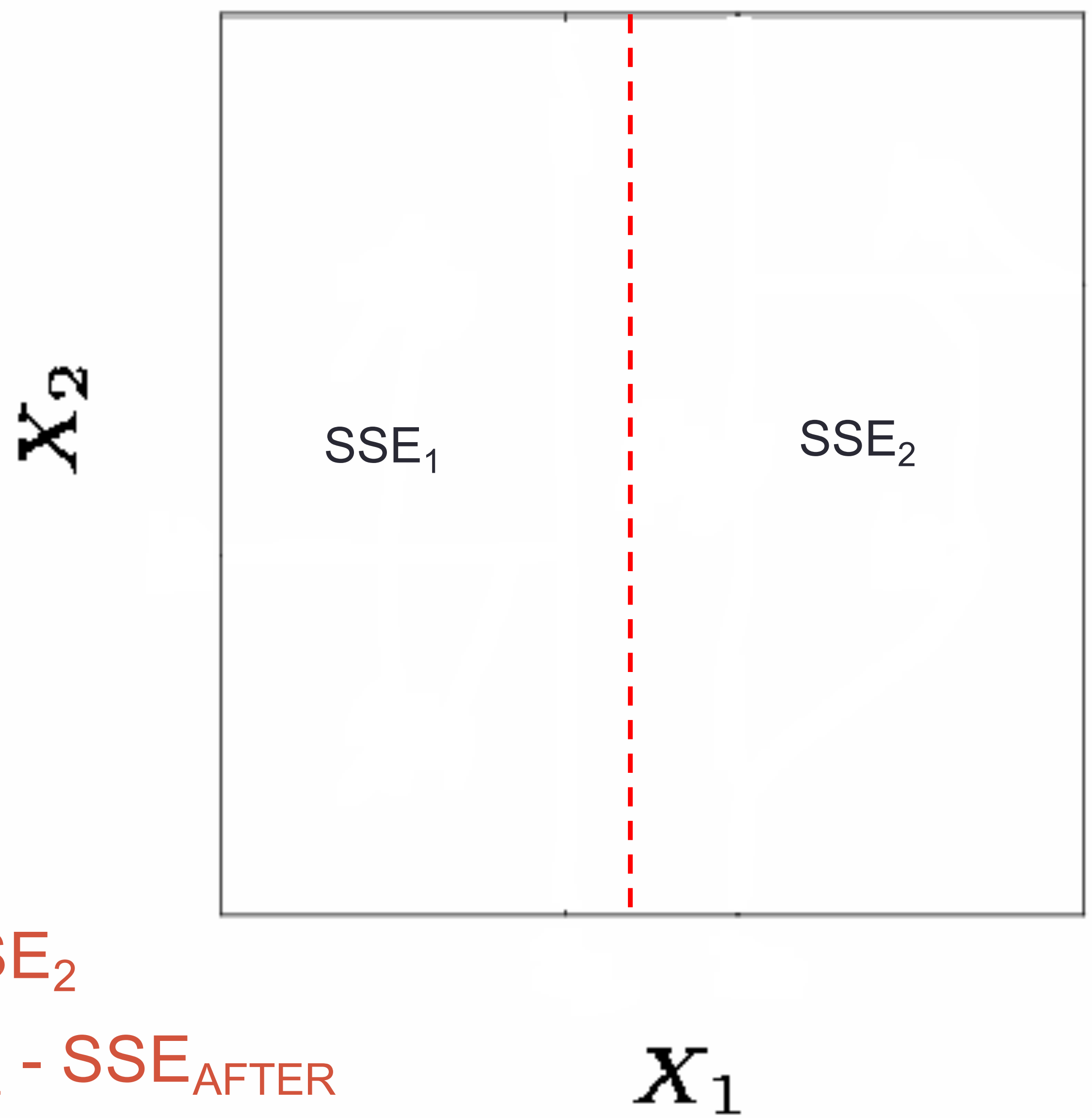


$SSE_{AFTER} = SSE_1 + SSE_2$

$Reduction = SSE_{BEFORE} - SSE_{AFTER}$

Partitioning Up the Predictor Space

- Any split creates new SSE_1, SSE_2
- Try different splits on X_1

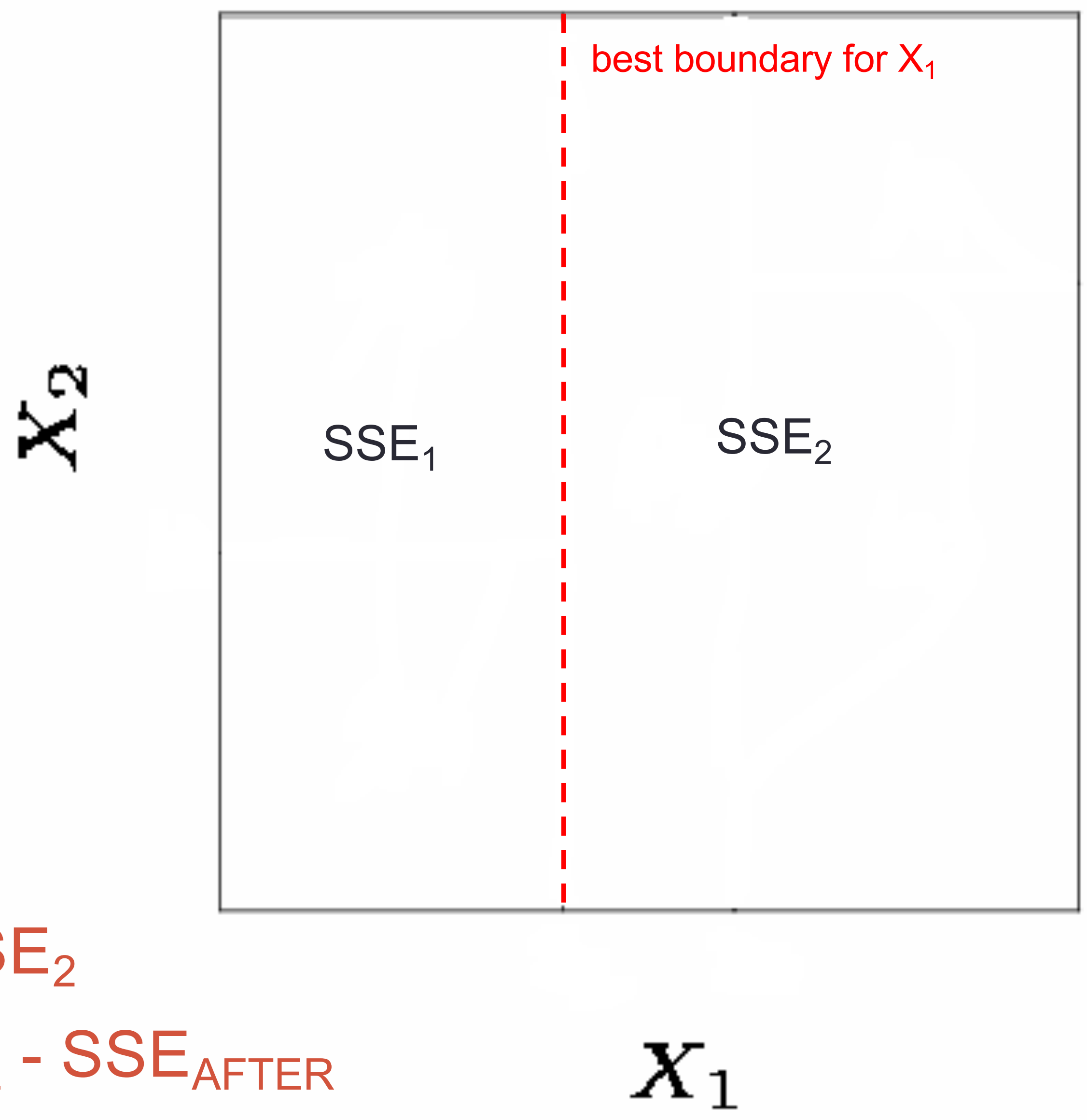


$SSE_{AFTER} = SSE_1 + SSE_2$

$\text{Reduction} = SSE_{BEFORE} - SSE_{AFTER}$

Partitioning Up the Predictor Space

- Any split creates new SSE_1, SSE_2
- Try different splits on X_1
- Select the boundary yielding the largest Reduction

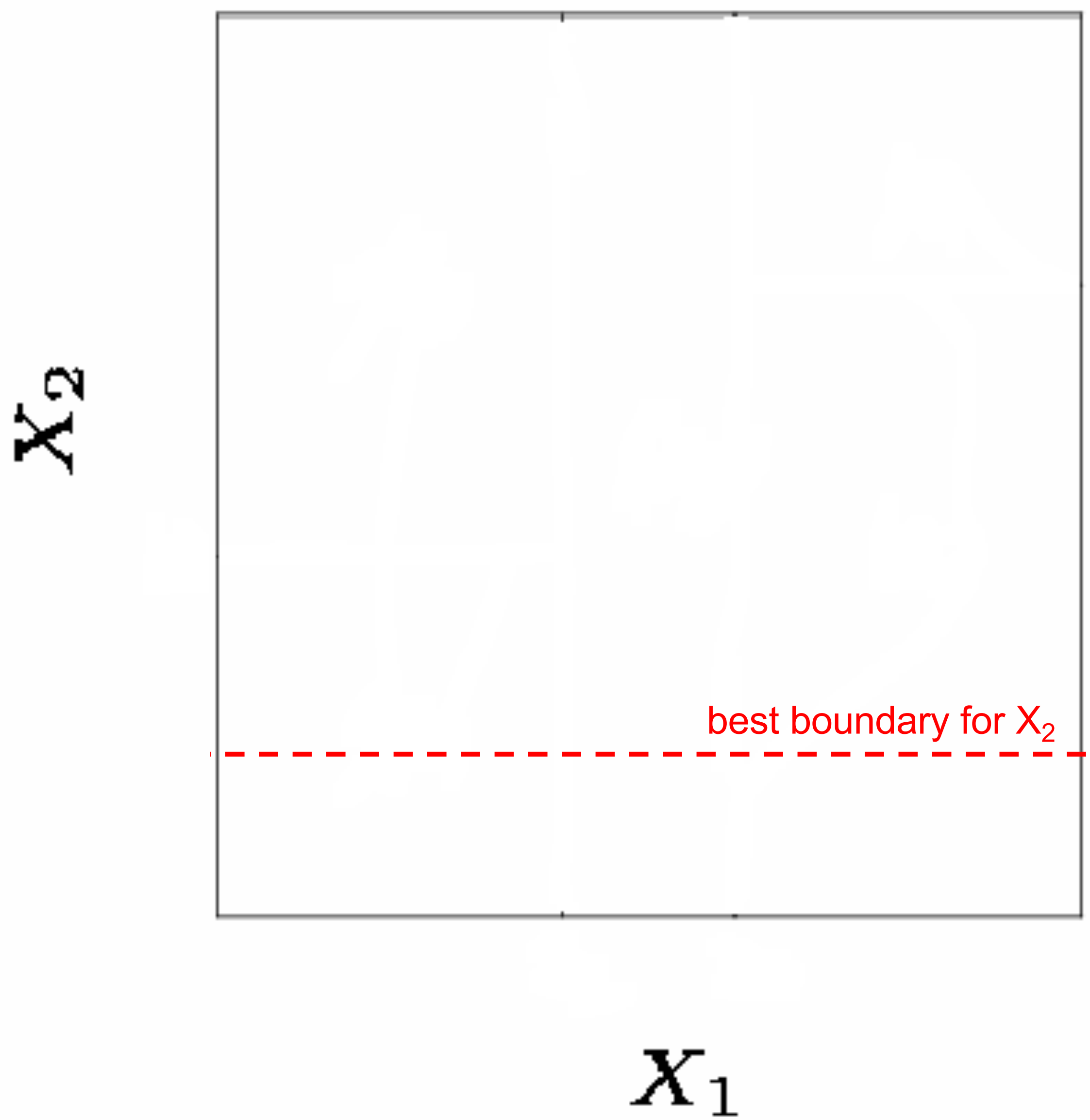


$SSE_{AFTER} = SSE_1 + SSE_2$

$Reduction = SSE_{BEFORE} - SSE_{AFTER}$

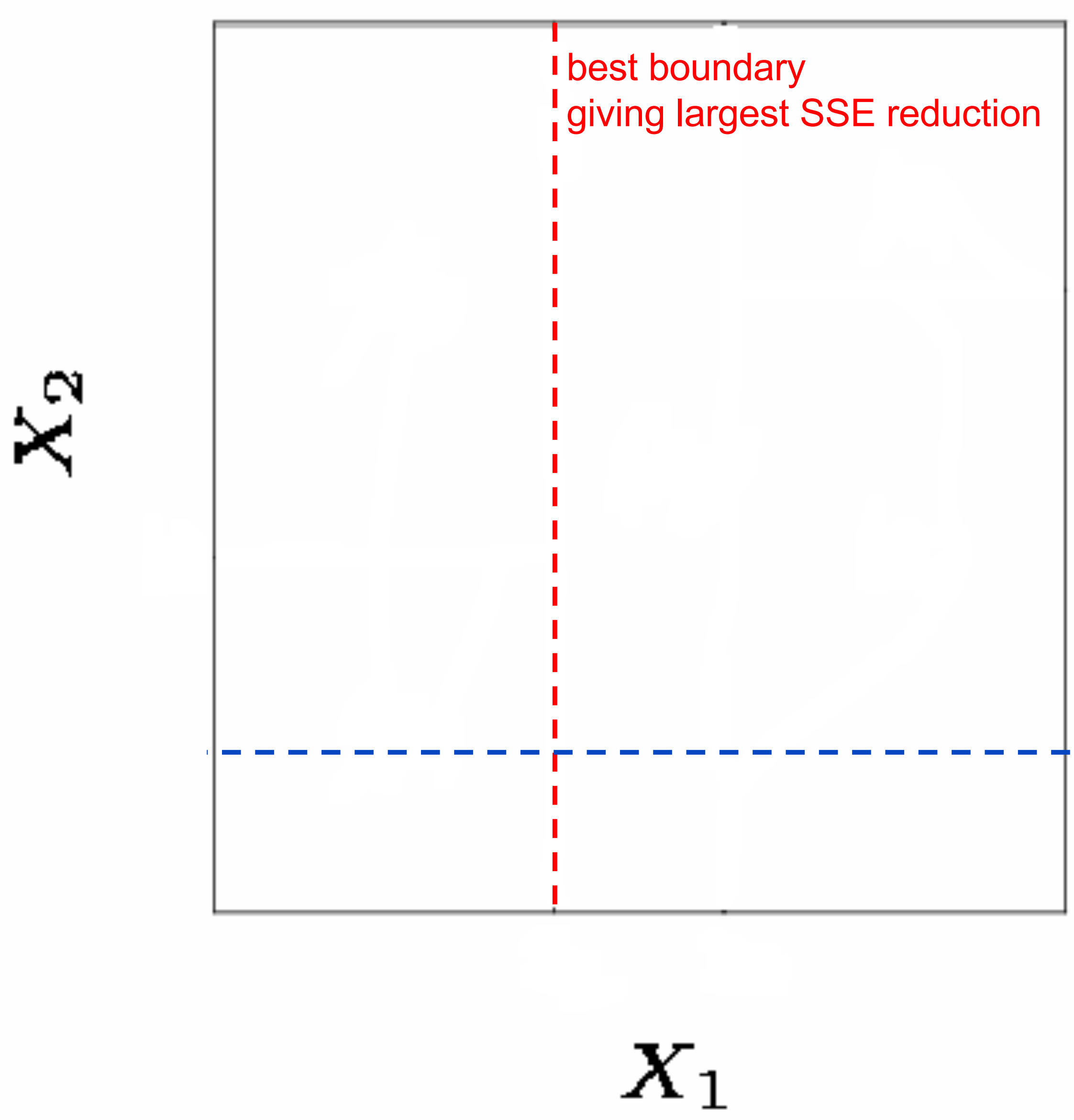
Partitioning Up the Predictor Space

- Repeat for X_2
- Selecting the boundary that results in the largest SSE reduction



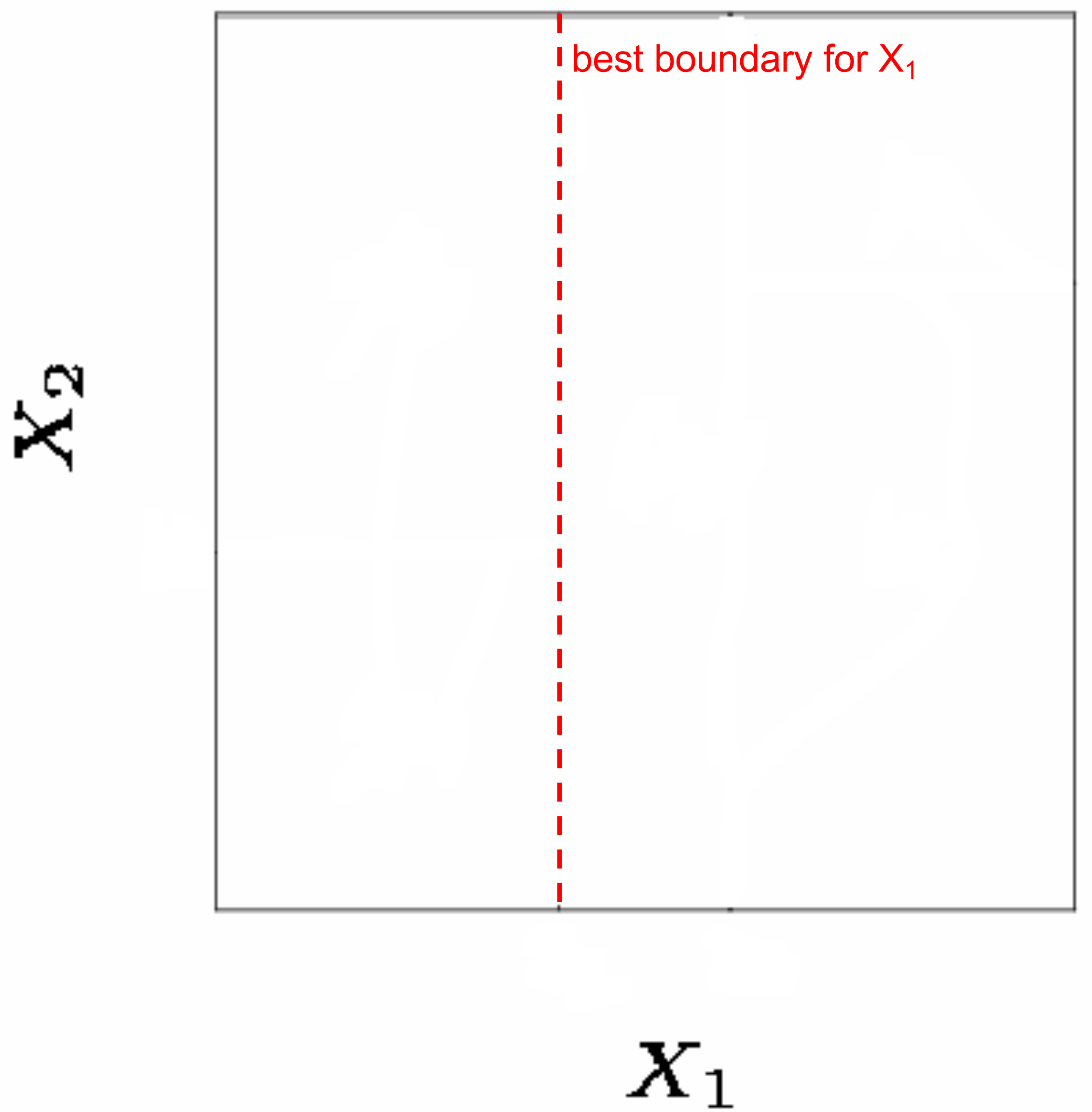
Partitioning Up the Predictor Space

- Compare best boundaries found for X_1 and X_2
- Choose one
- The boundary that results in the largest SSE reduction



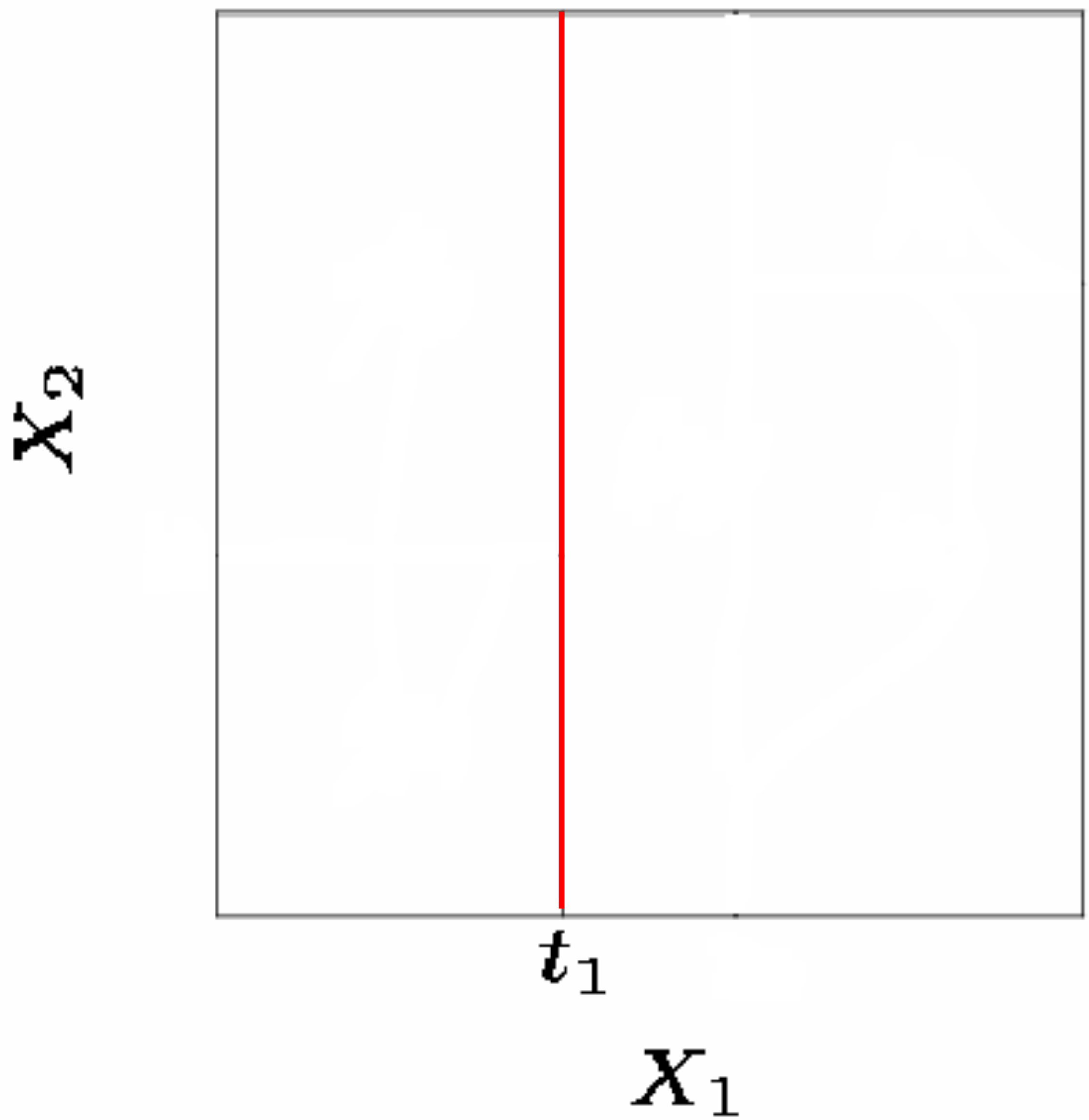
Partitioning Up the Predictor Space

- Compare best boundaries found for X_1 and X_2
- Choose one
- The boundary that results in the largest SSE reduction
- Split Tree using that predictor



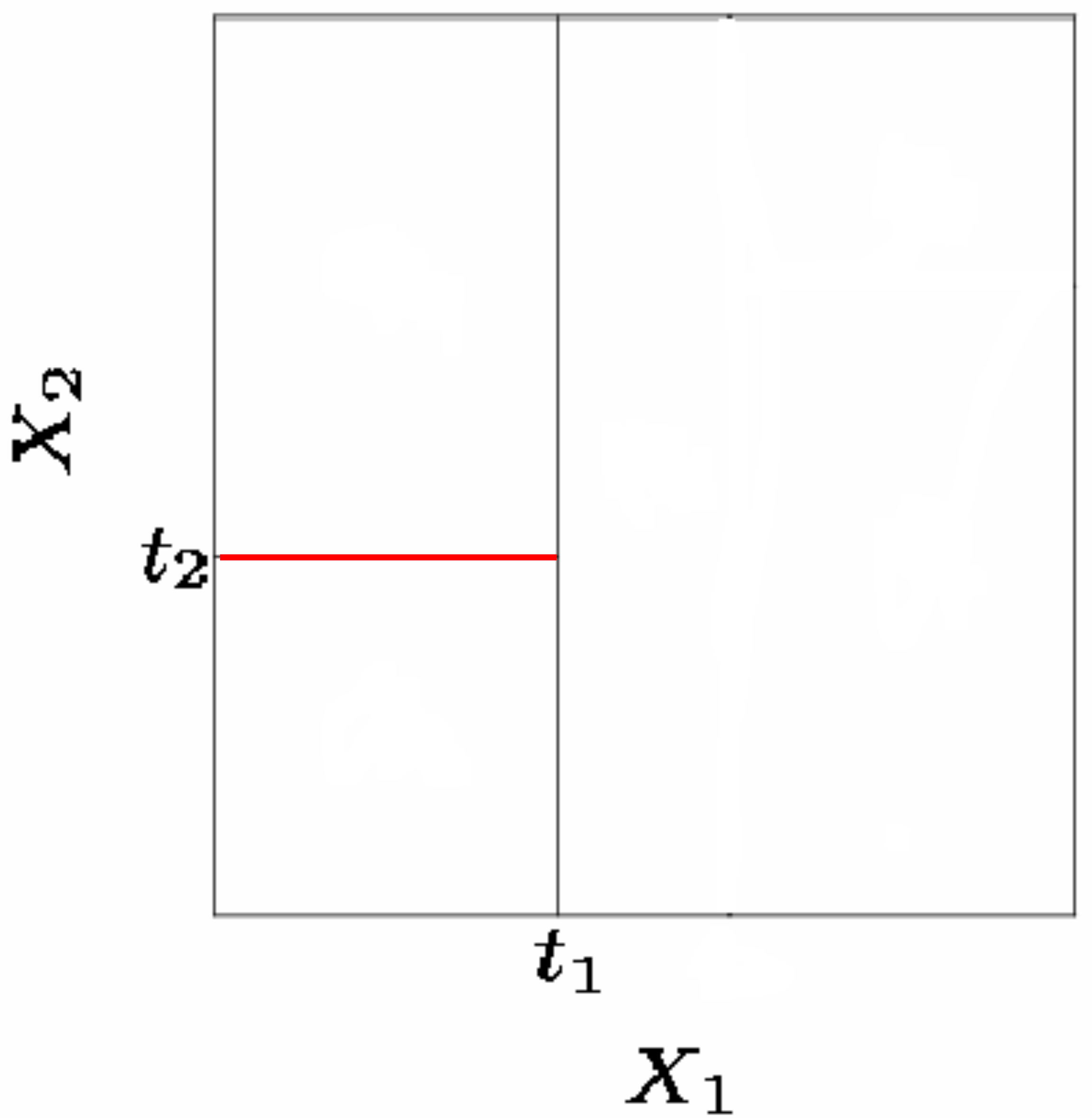
Partitioning Up the Predictor Space

1. First split on $X_1 = t_1$



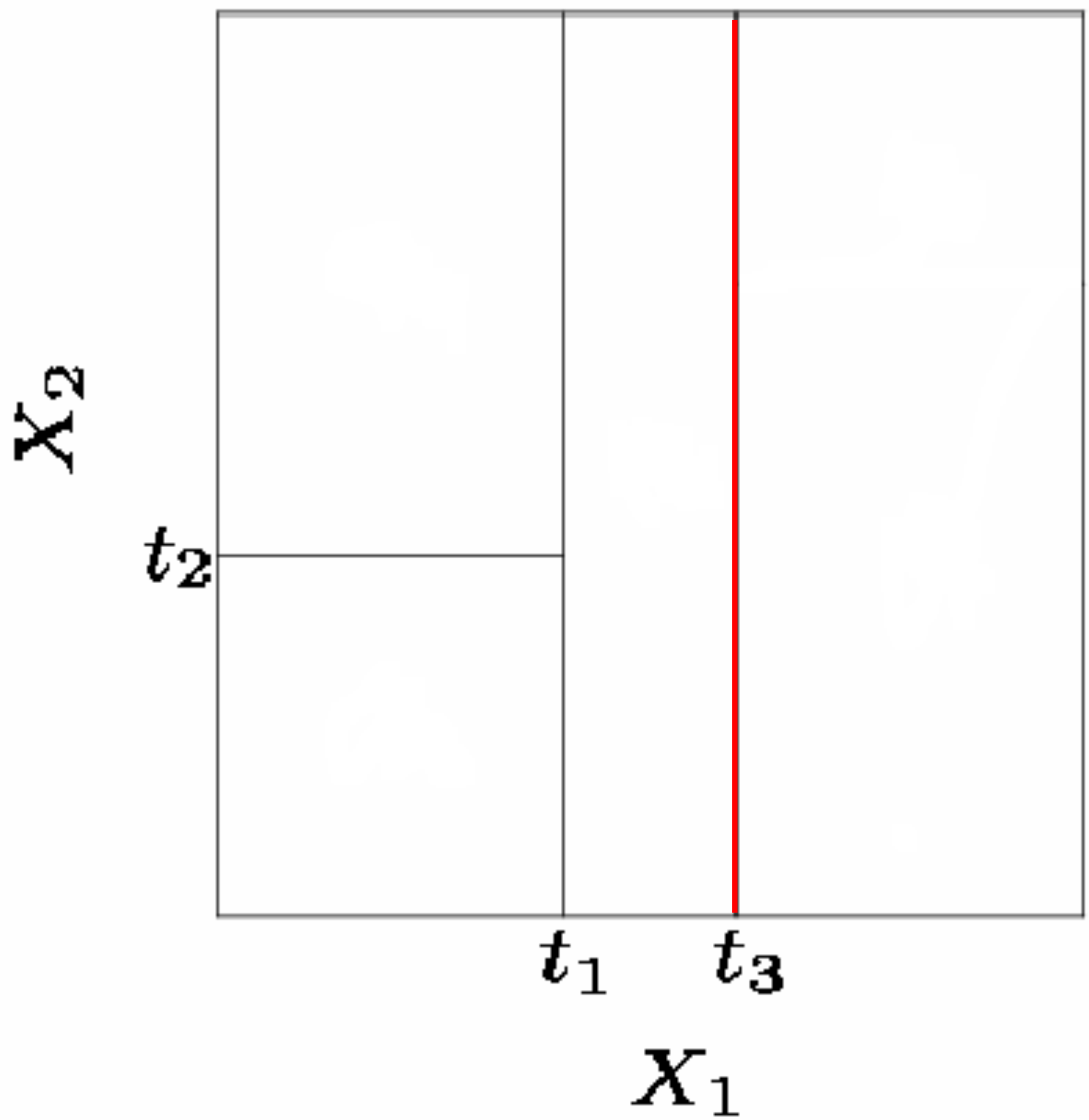
Partitioning Up the Predictor Space

1. First split on $X_1 = t_1$
2. If $X_1 < t_1$, split on $X_2 = t_2$



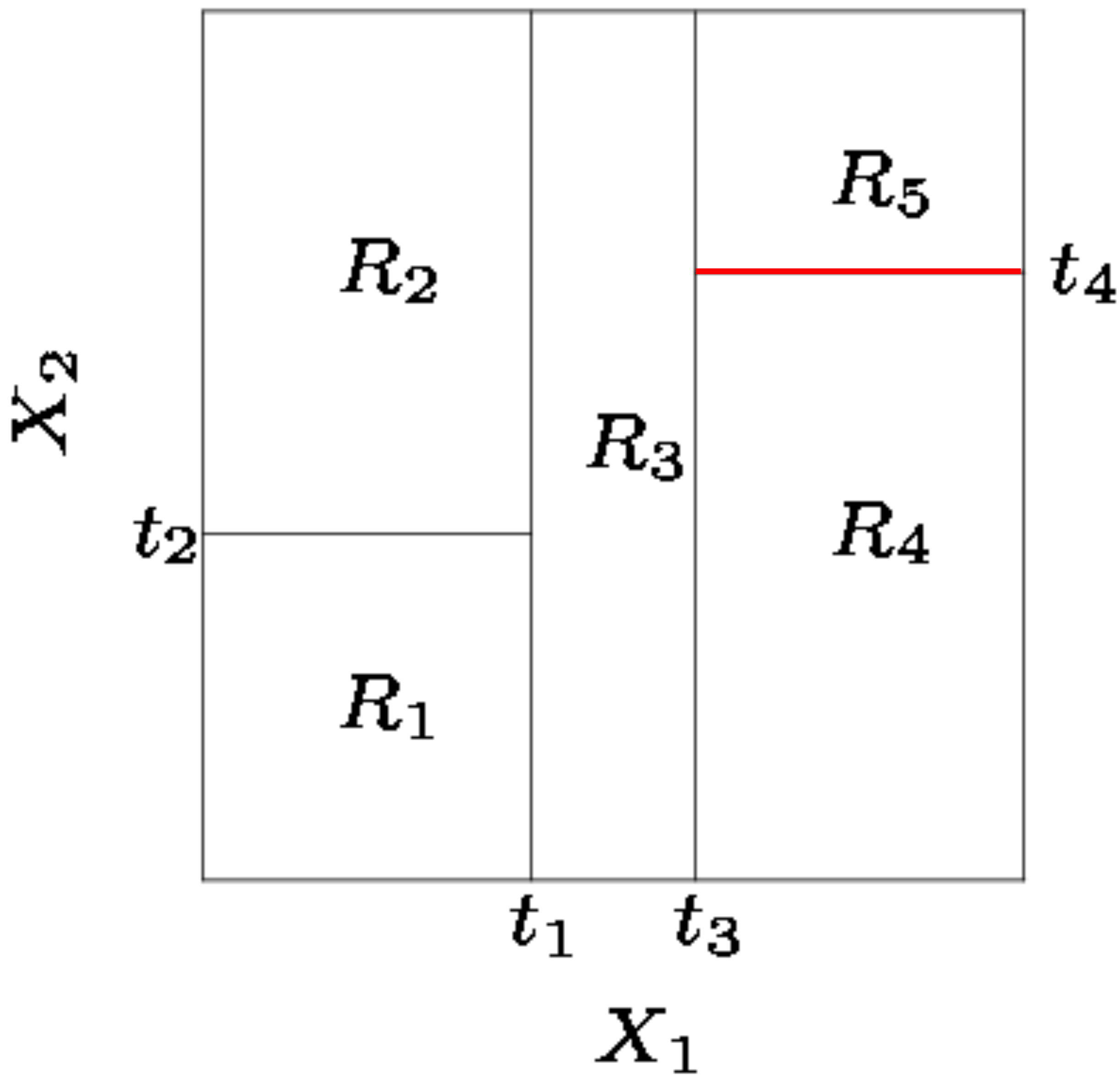
Partitioning Up the Predictor Space

1. First split on $X_1 = t_1$
2. If $X_1 < t_1$, split on $X_2 = t_2$
3. If $X_1 > t_1$, split on $X_1 = t_3$



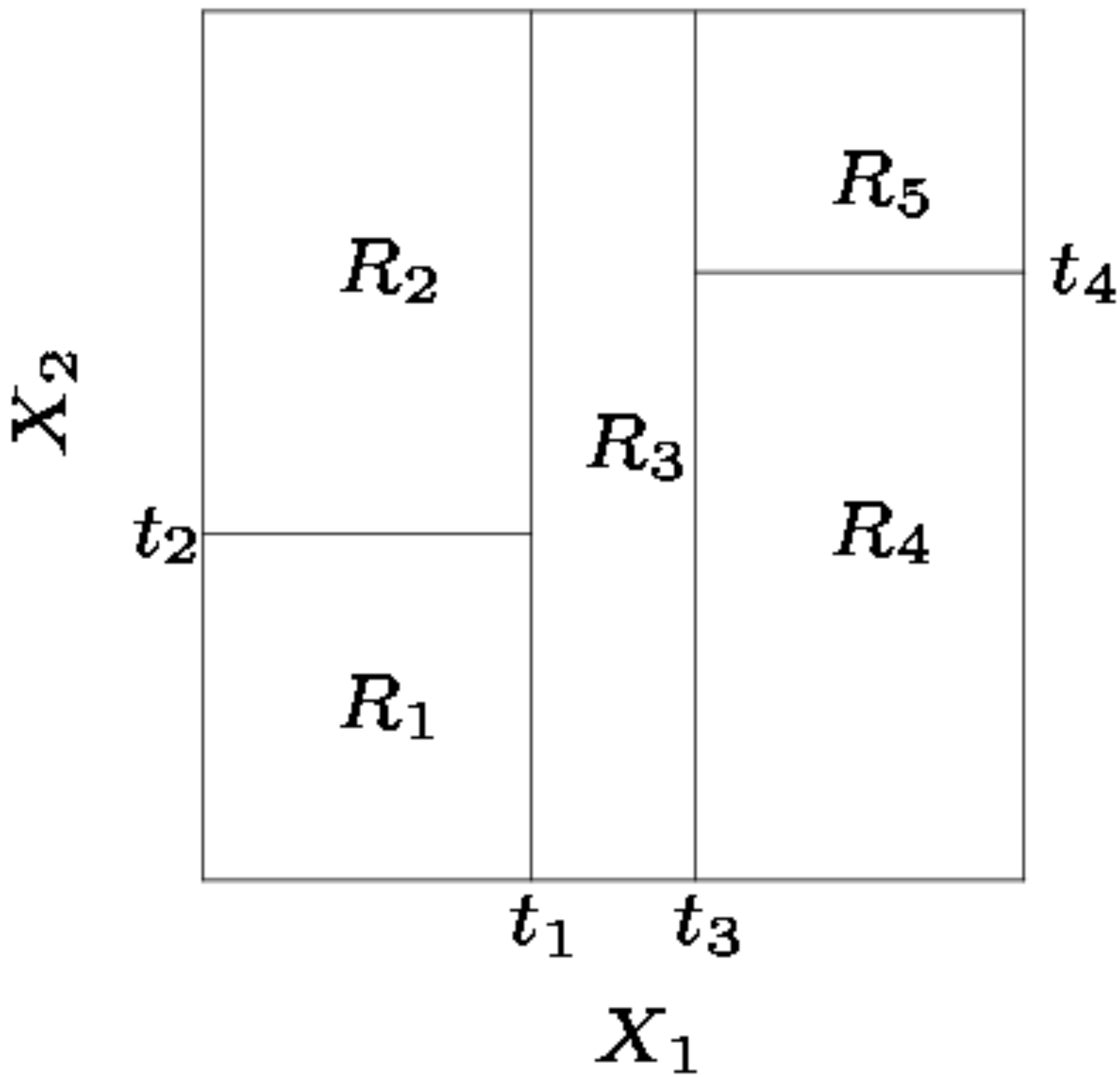
Partitioning Up the Predictor Space

1. First split on $X_1 = t_1$
2. If $X_1 < t_1$, split on $X_2 = t_2$
3. If $X_1 > t_1$, split on $X_1 = t_3$
4. If $X_1 > t_3$, split on $X_2 = t_4$



Partitioning Up the Predictor Space

1. First split on $X_1=t_1$
2. If $X_1<t_1$, split on $X_2=t_2$
3. If $X_1>t_1$, split on $X_1=t_3$
4. If $X_1>t_3$, split on $X_2=t_4$
5. stop

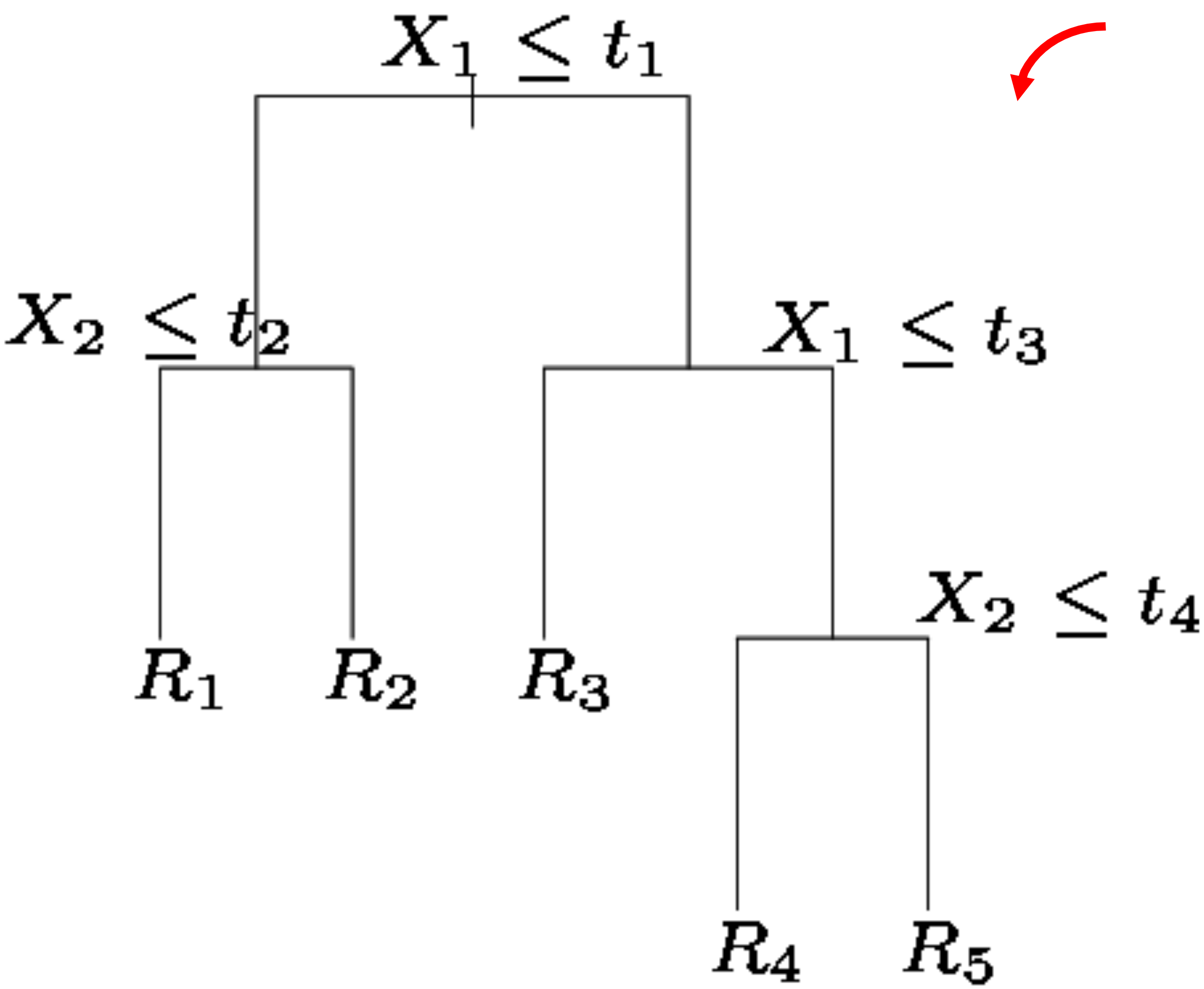


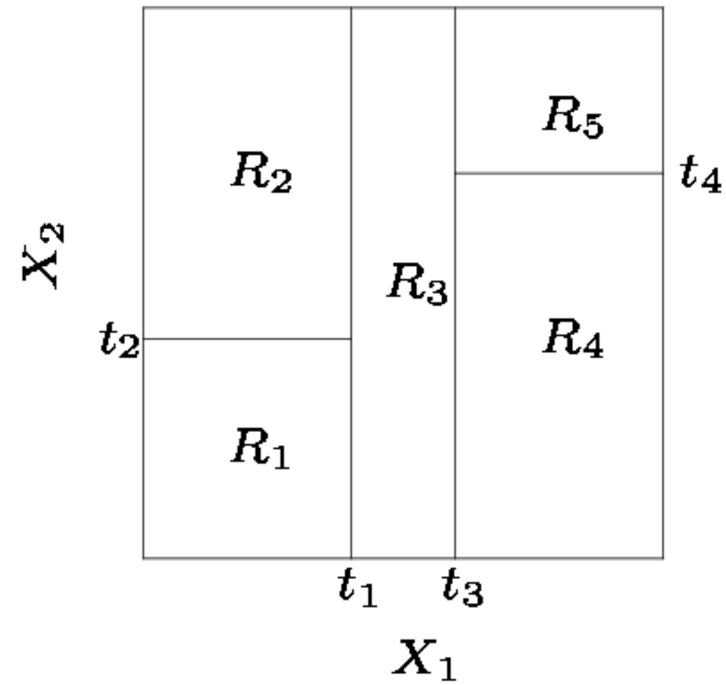
Stopping criteria

As the number of splits increase, the regions become smaller, and the number of observations in the regions decrease

- Criteria 1: Fix (in advance) the number of splits
- Criteria 2: Stop when number of obs in regions is small enough
- Criteria 3: Stop when the resulting SSE decrease is small enough

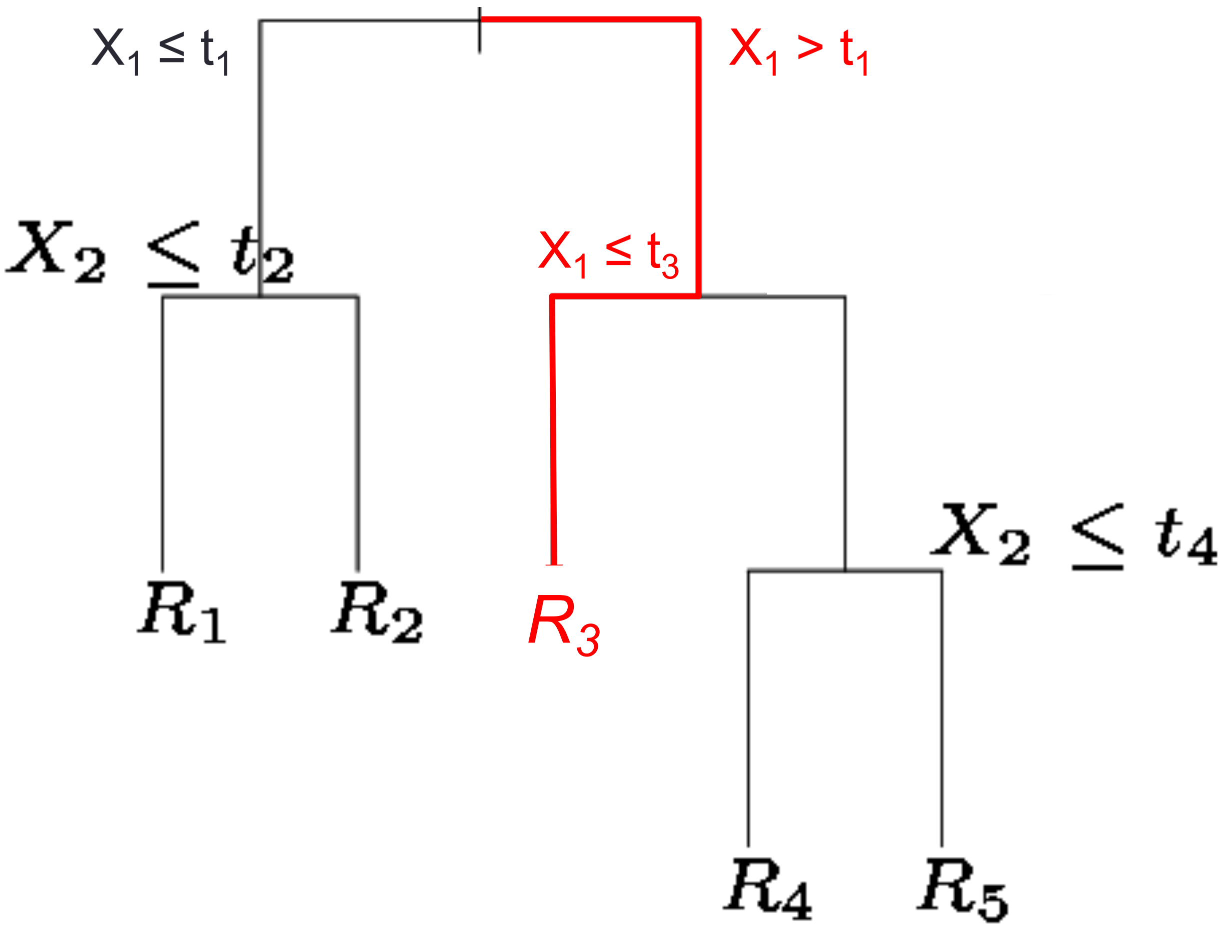
Decision Tree

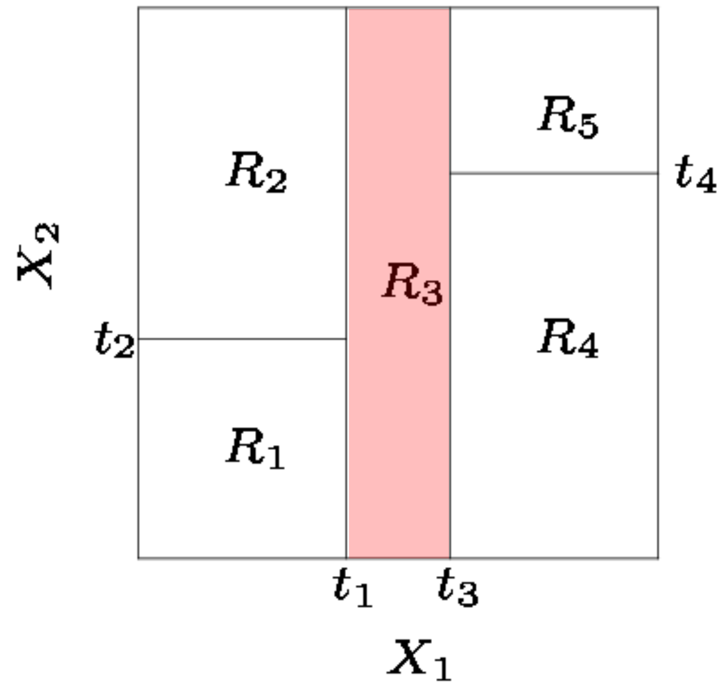




A Tree is a graphical representation of the splits

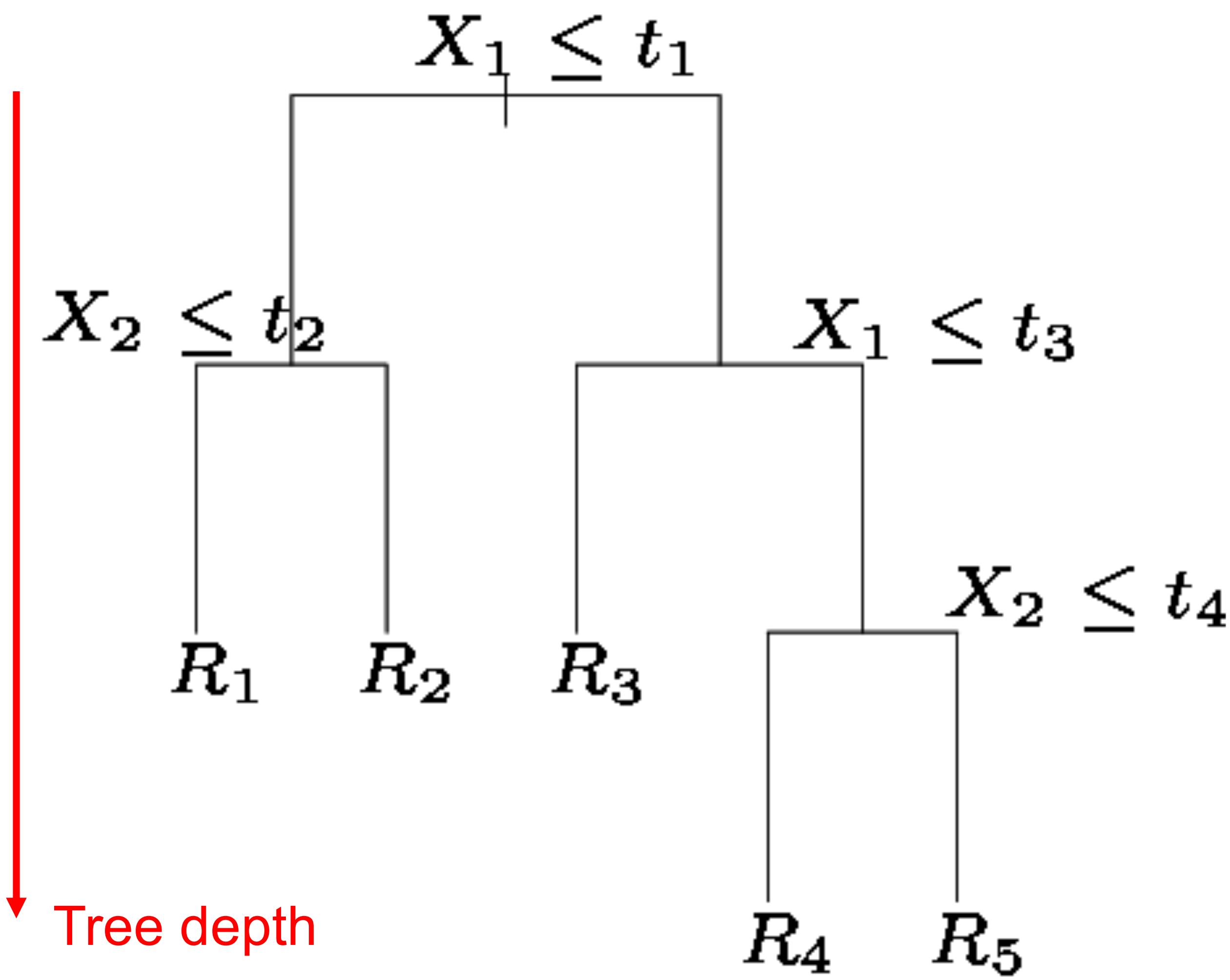
Decision Tree

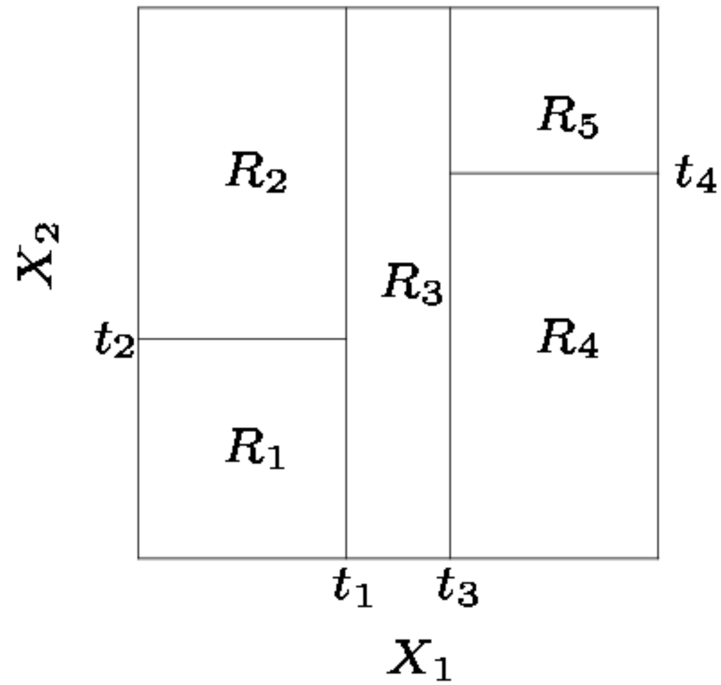




Each region is defined by the inequalities leading to that region

Decision Tree





As the number of splits increases, the tree depth increases

Prunning a tree

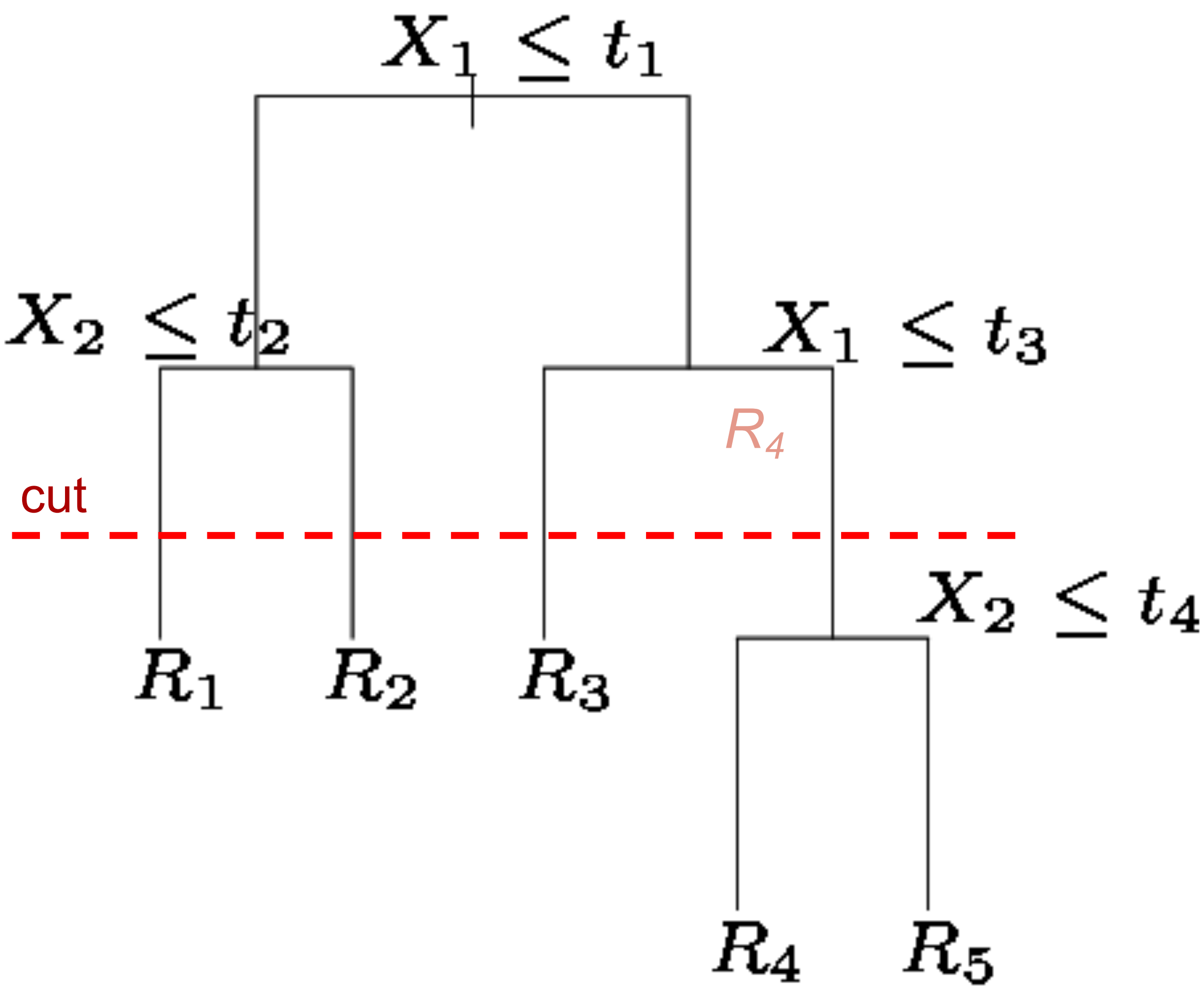
- Large (deep) trees have a large number of regions T
- Large (deep) trees tend to overfit the data
- Adjust the tree loss function by adding a penalty term
- Let αT be the penalty for trees with T regions
- For each α find number of regions T minimizing

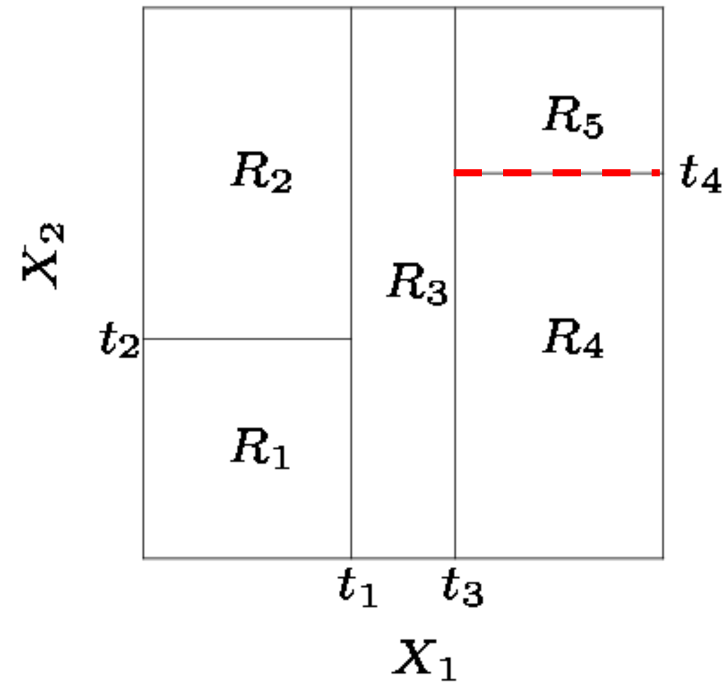
$$\text{RSS} = \sum_{i \,\epsilon\, R_1} (y_1 - \hat{y}_1)^2 + \sum_{i \,\epsilon\, R_2} (y_2 - \hat{y}_2)^2 + \cdots + \alpha\, T$$

penalty

- Use CV to find the best number of regions T

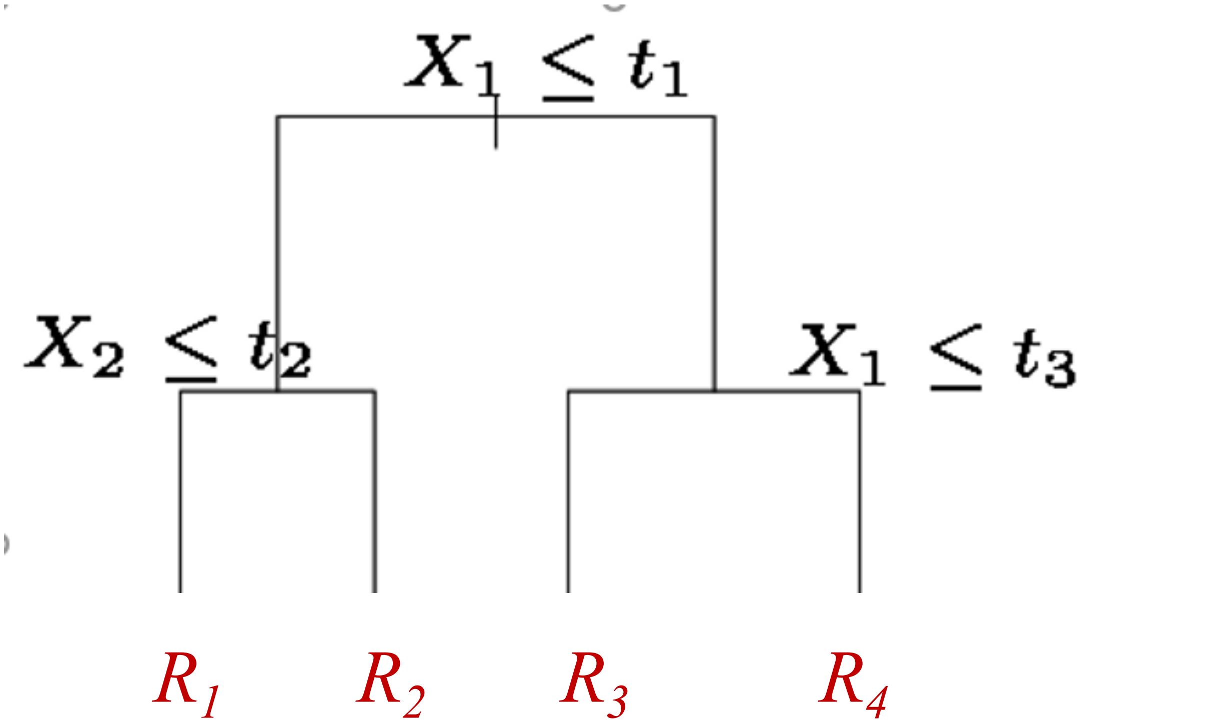
Tree prunning

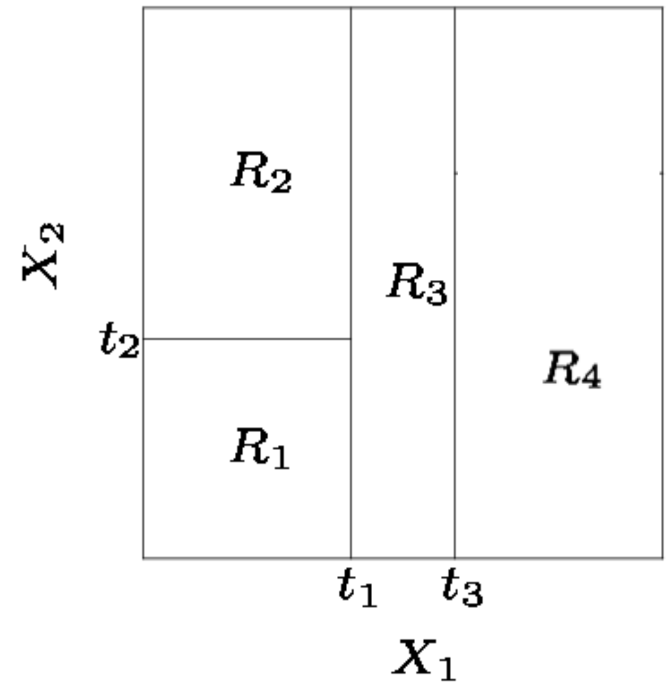




- Pruning a tree results from cutting a tree
- Cutting a tree is equivalent to merging region back

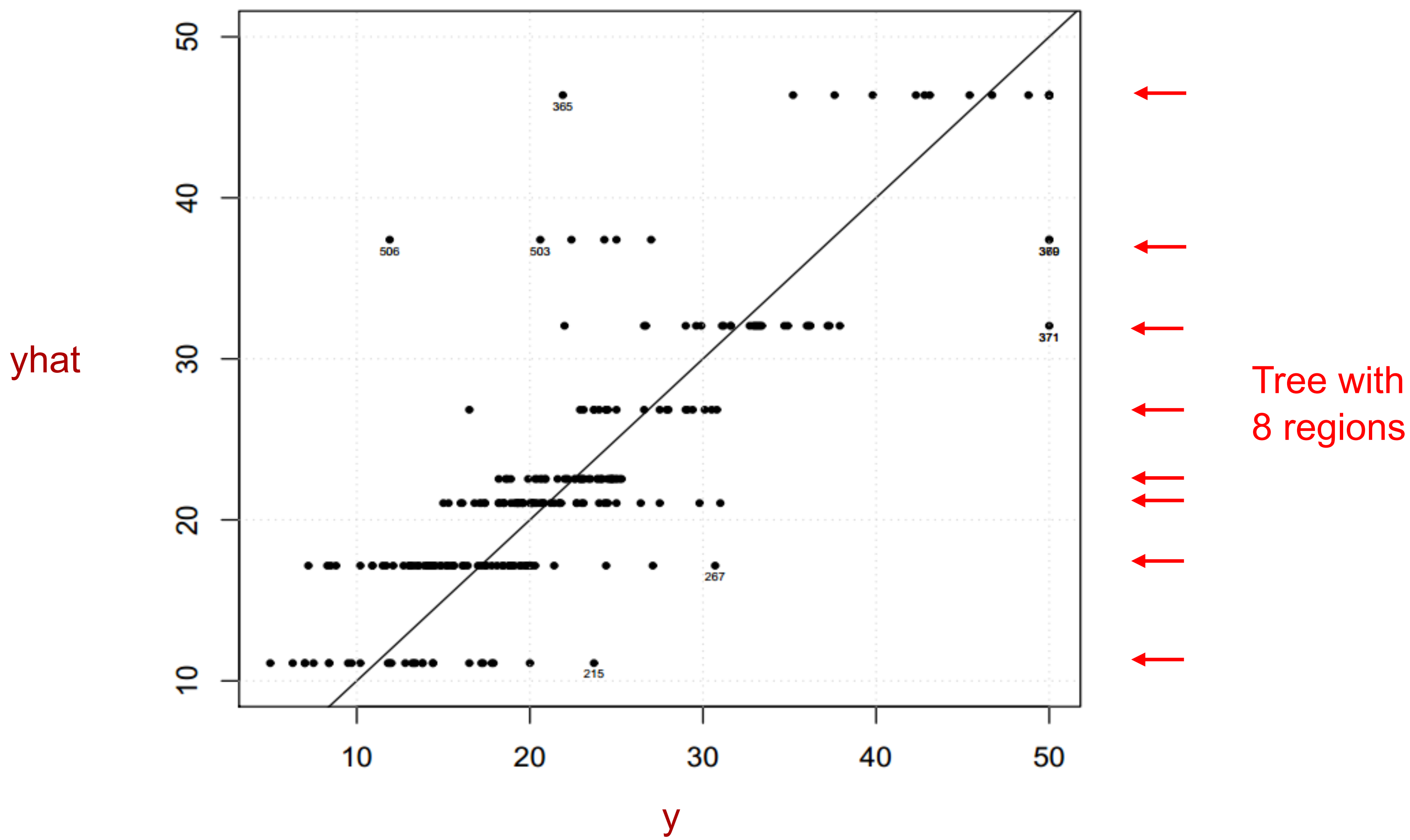
Pruned Tree





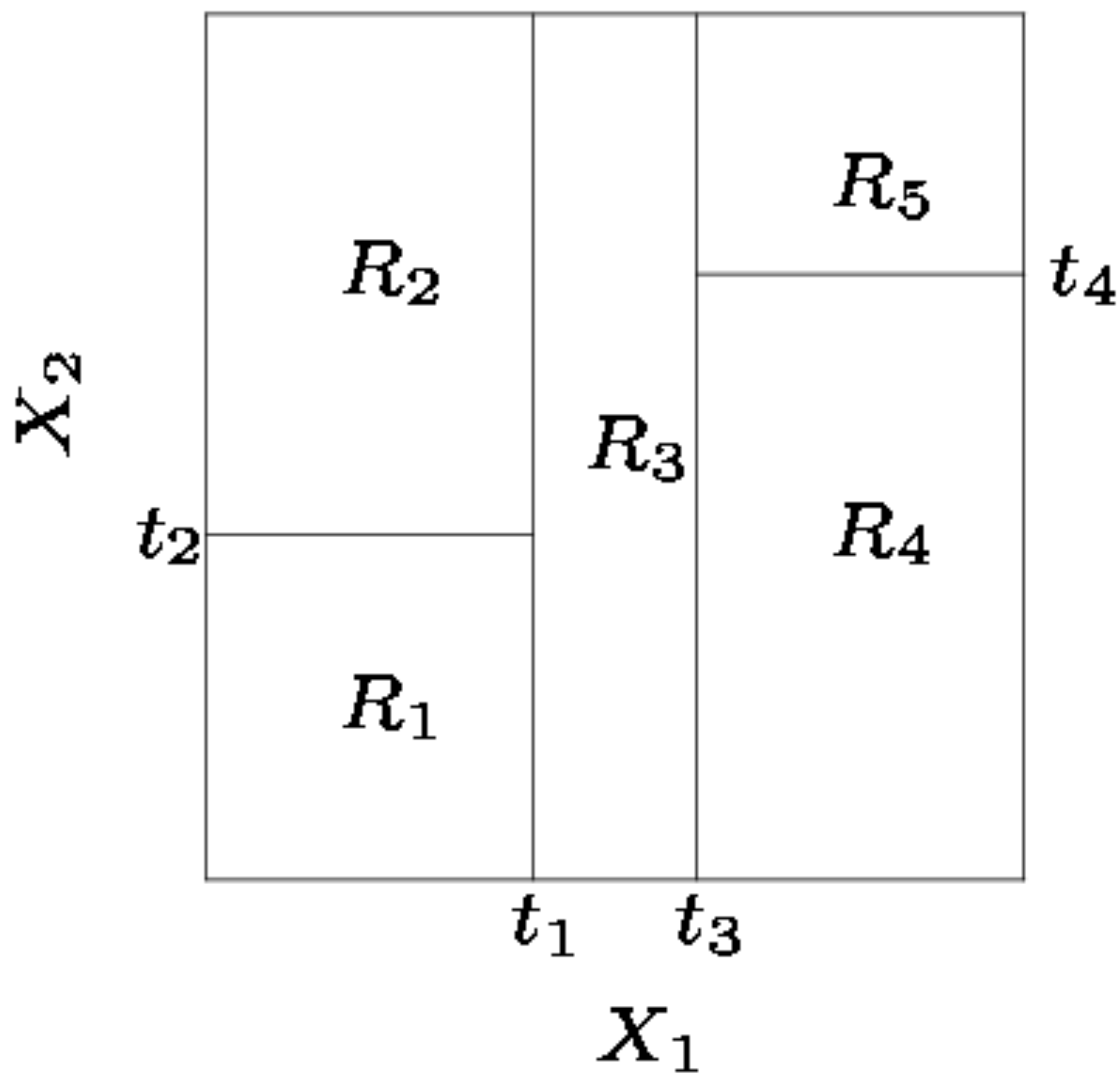
Pruning a tree reduces the number of regions

Display the tree performance: Yhat vs Y plot



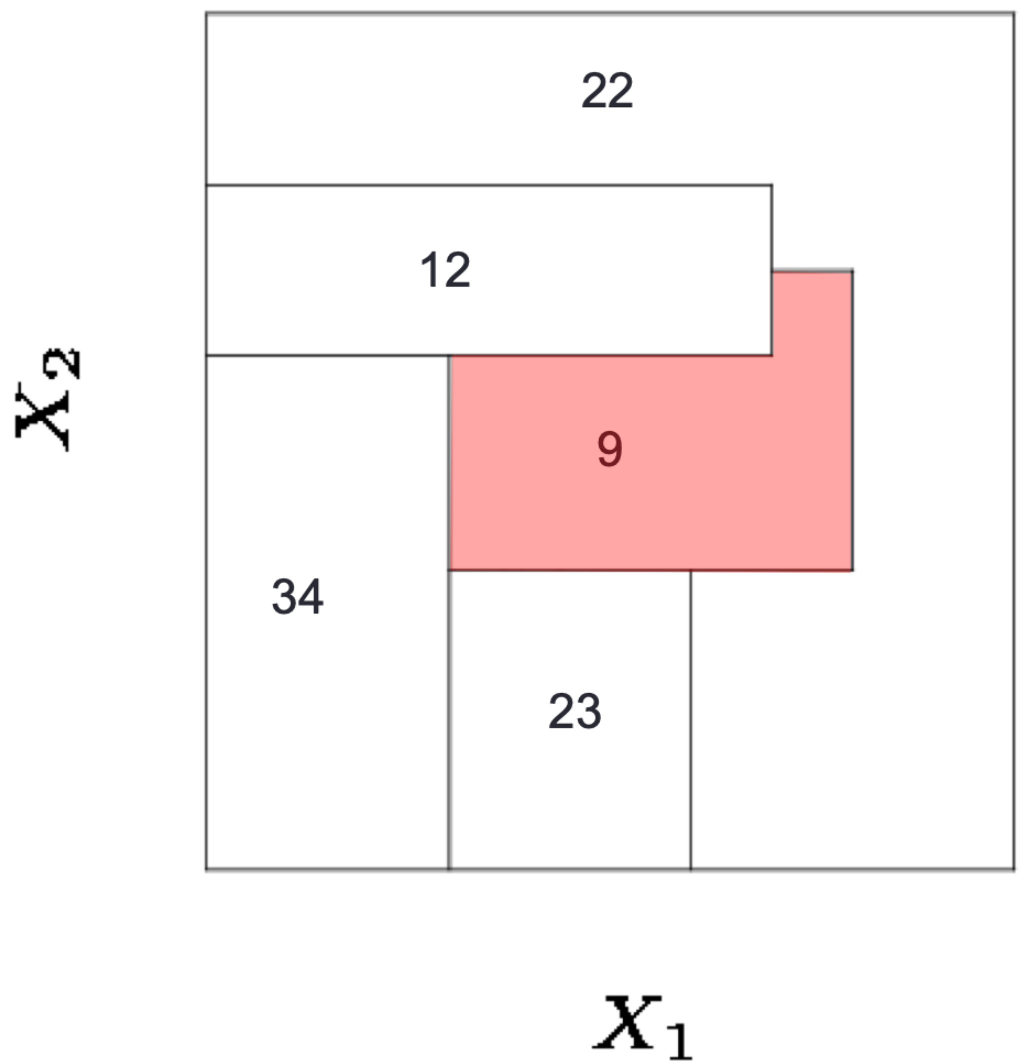
Rectangular regions

CART models partition the predictor space into rectangular non-overlapping regions



Not possible

- This partitioning cannot result from a regression tree
- Region 9 is not rectangular



Regression Trees - Notes

- At each split the number of observations in the 2 new regions decrease
- If following a split, all observations in a splitted region have the same y-value then
    - prediction (ybar) is equal to that value
    - That region becomes terminal region
- No scaling is needed

Example – Boston dataset

Example – Boston dataset

Information of 506 houses in the area of Boston, Mass

Example – 13 predictors, 1 response

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 (000’s)

Example – Boston dataset

Predict the price of houses and identify which variables are most useful for prediction.

- Divide the dataset into a training (50%) and a test set.
- Fit a regression tree. Plot the resulting tree.
- Plot the test MSPE as a function of the tree depth.
- Use cross validation to find the best number of regions.
- Which predictors are the most important?

Example – Boston dataset

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
```

```
from sklearn.tree import DecisionTreeRegressor
```

```
# Choose one of the following:
# !pip install graphviz
# conda install python-graphviz
```

```
from sklearn.tree import export_graphviz
import graphviz
```

```
# !pip install pydotplus
import pydotplus
from IPython.display import Image
```

Example – Boston dataset

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

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	Y 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

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

Example – Boston dataset

Validation Approach

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

max_depth = 2

```
regr_tree_boston = DecisionTreeRegressor(max_depth = 2)
regr_tree_boston.fit(X_train, y_train)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=2, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

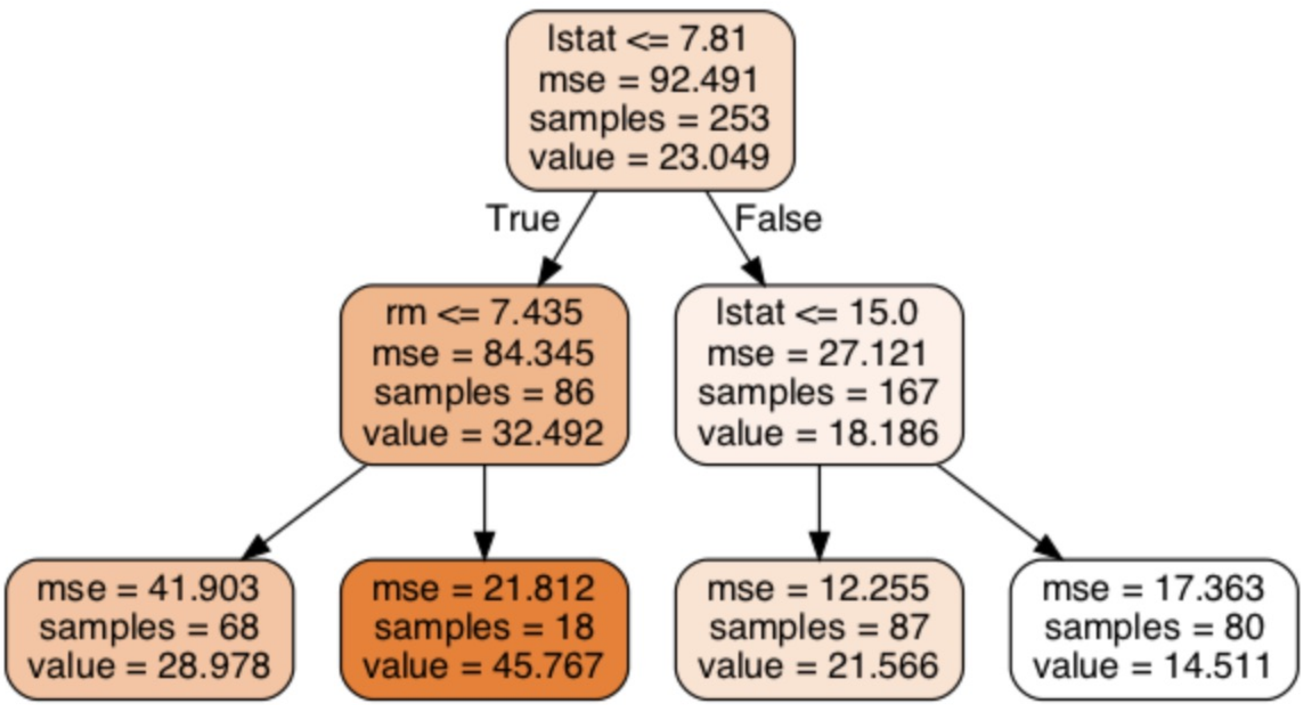
Example – Boston dataset

```
dot_data = export_graphviz(regr_tree_boston,
                                feature_names=col_names,
                                out_file=None,
                                filled=True,
                                rounded=True)
pydot_graph = pydotplus.graph_from_dot_data(dot_data)
pydot_graph.set_size('"6,6!"')
pydot_graph.write_png('resized_tree.png');
Image(pydot_graph.create_png())
```

root

depth 1

depth 2

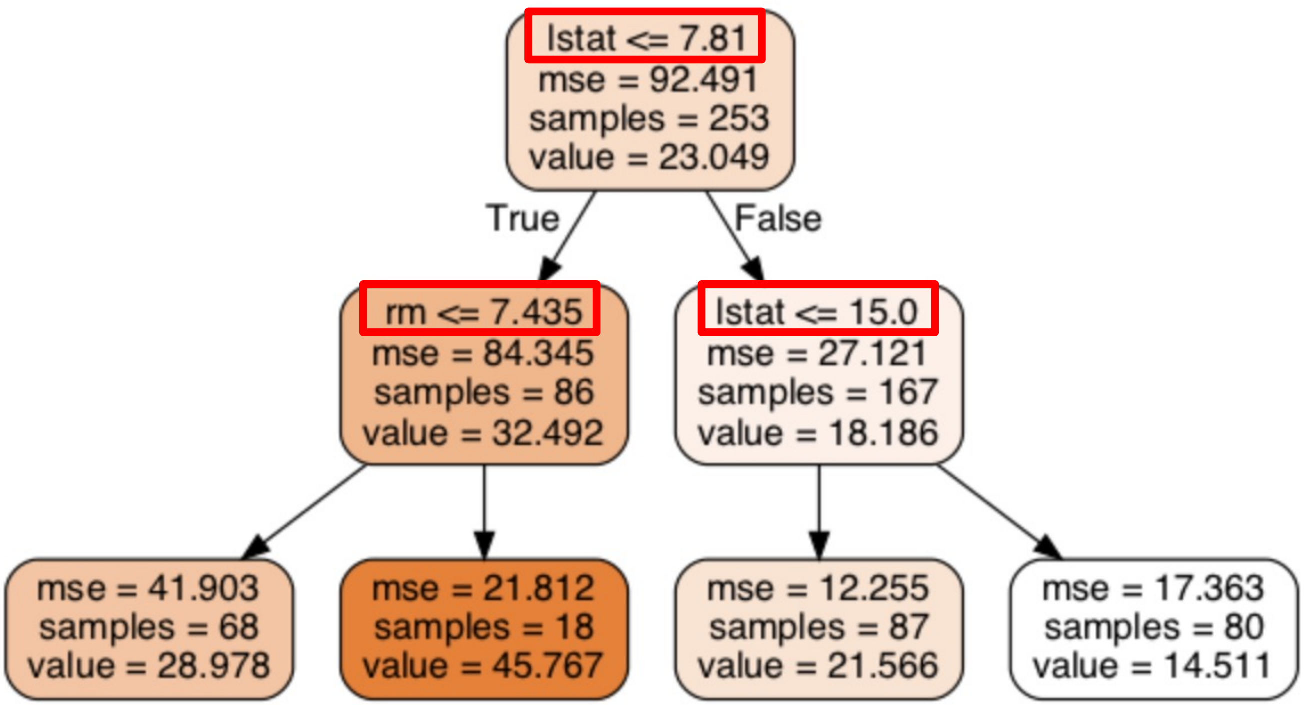


terminal nodes or leaves

There are 4 terminal regions

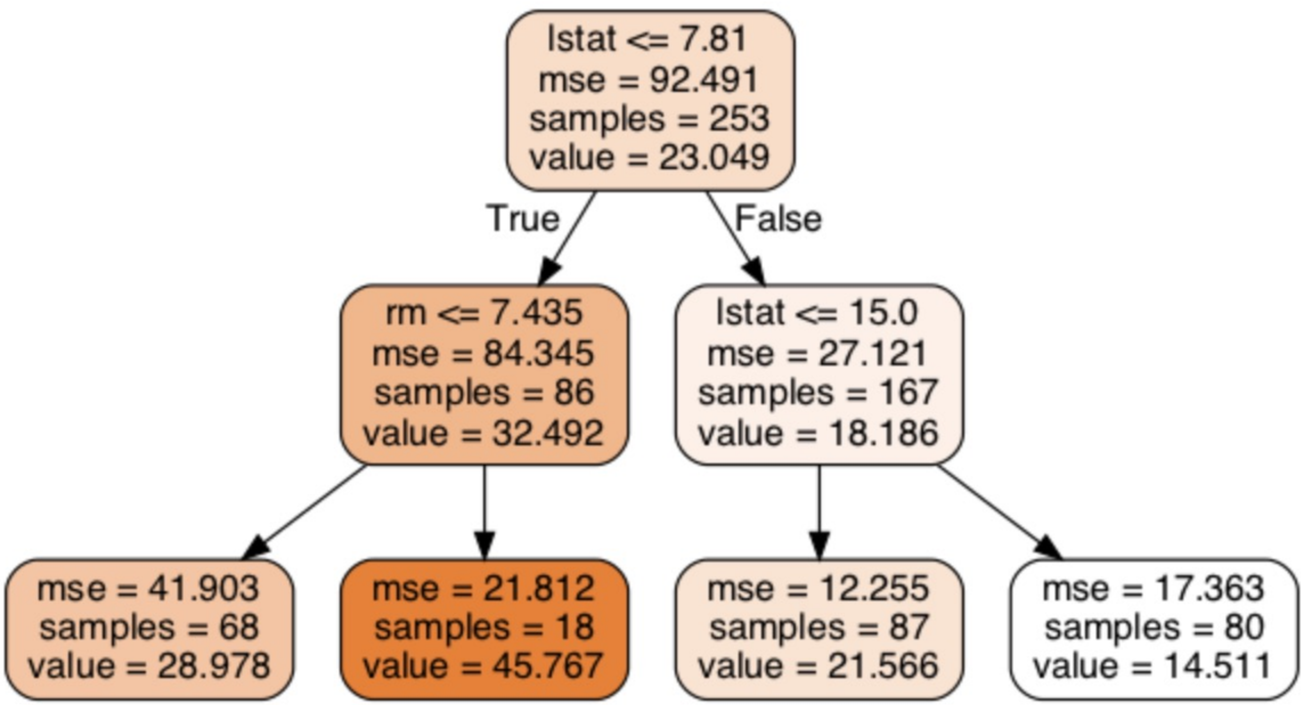
Example – Inequalities (in all non-terminal nodes)

.

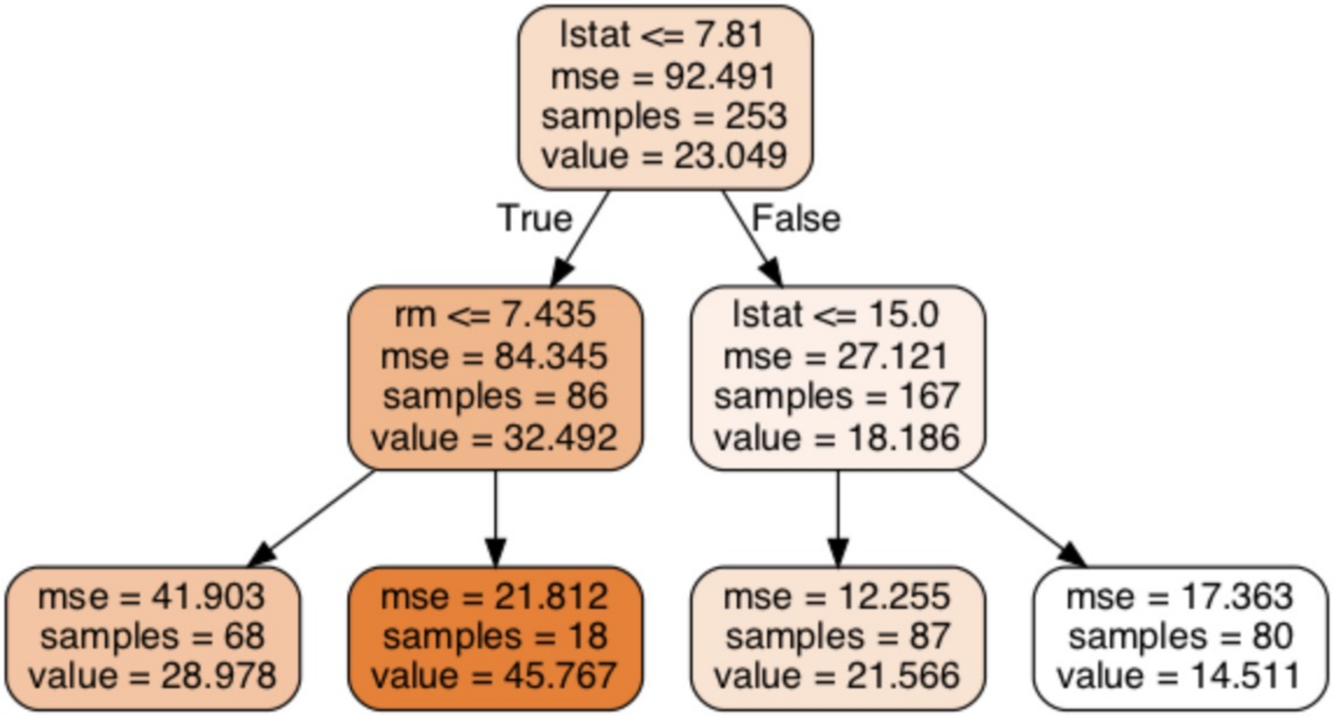


Example – Boston dataset

- samples = number of observations in the region
- value = average response Y in the region
- mse = train MSE in the region



Example – Predict price of a new house

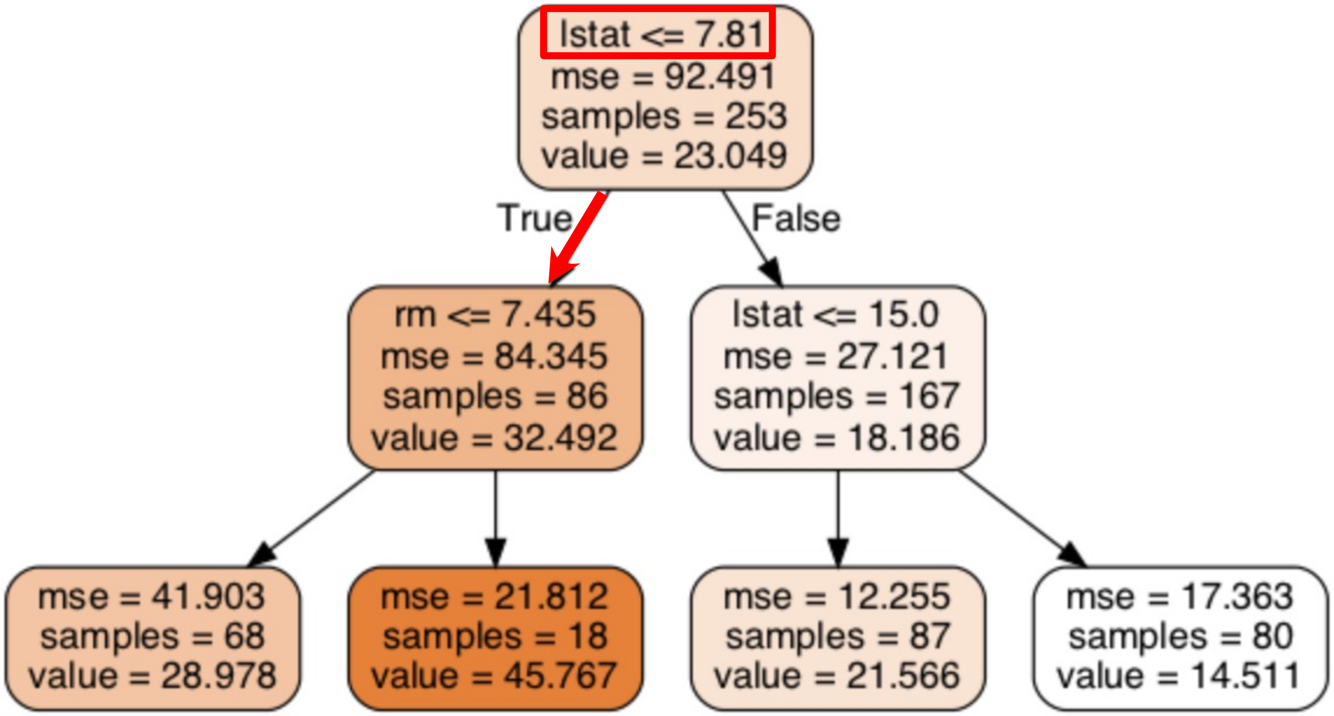


newval

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98

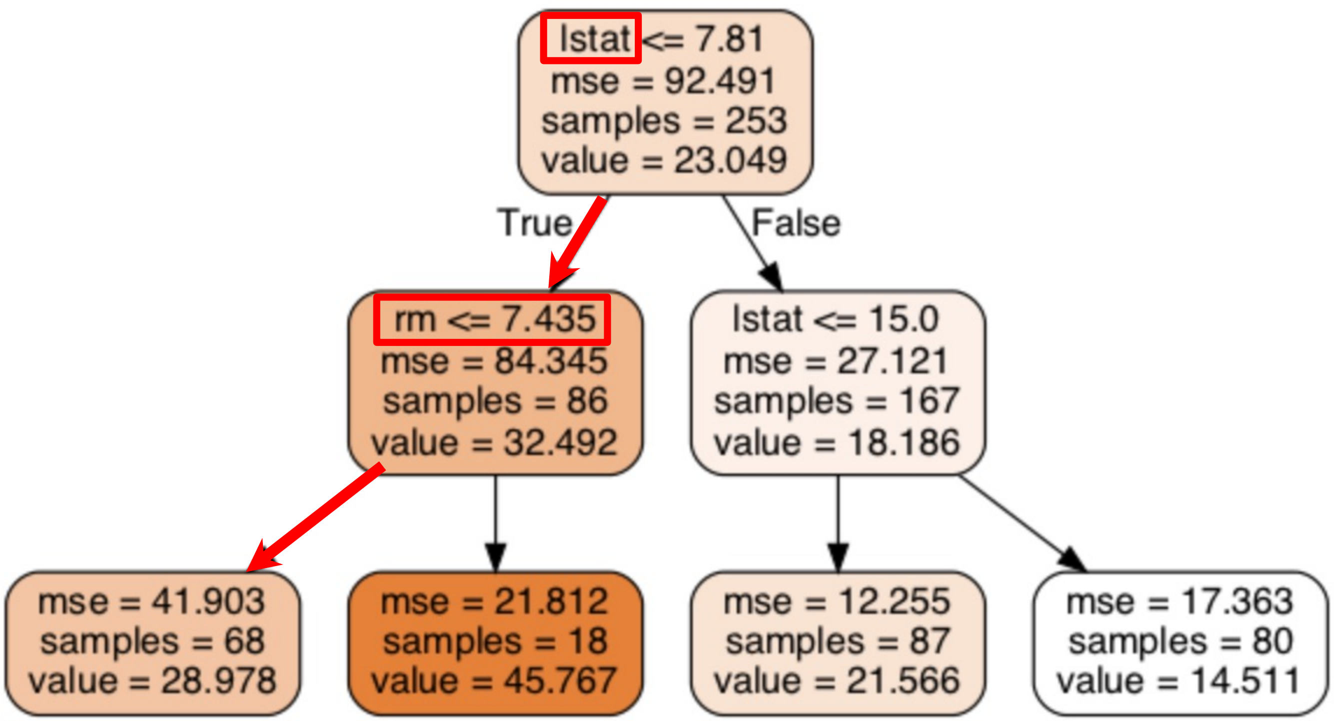
← new house

Example – Predict price of a new house



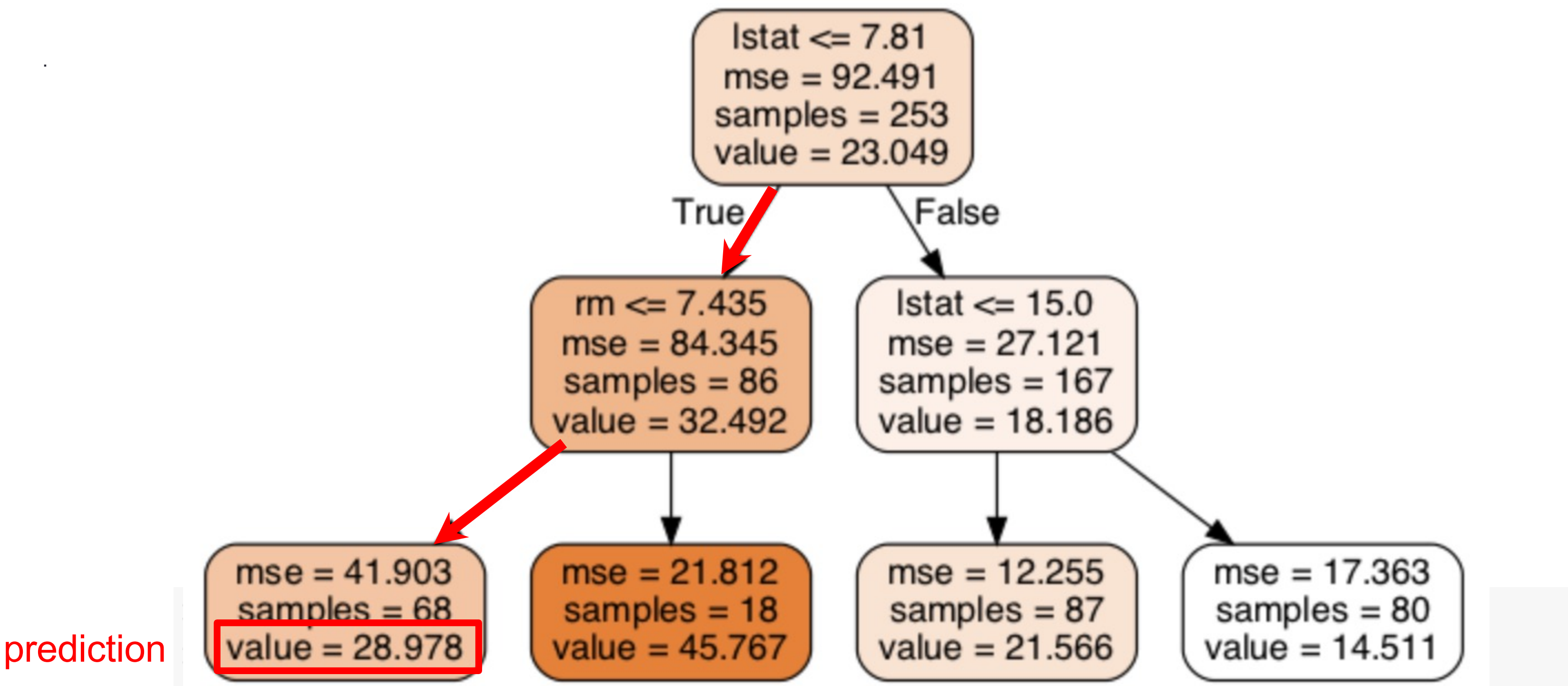
	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98

Example – Predict price of a new house



	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98

Example – Predict price of a new house



	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98

```
regr_tree_boston.predict(newval)
```

```
array([28.97794118])
```

house price is predicted at 28977.97 dollars

Example – Test MSE and Tree depth

max_depth = 2

```
regr_tree_boston = DecisionTreeRegressor(max_depth = 2)
regr_tree_boston.fit(X_train, y_train)
pred = regr_tree_boston.predict(X_test)
mspe = mean_squared_error(y_test, pred)
mspe
```

```
28.80154486445794
```

max_depth = 4

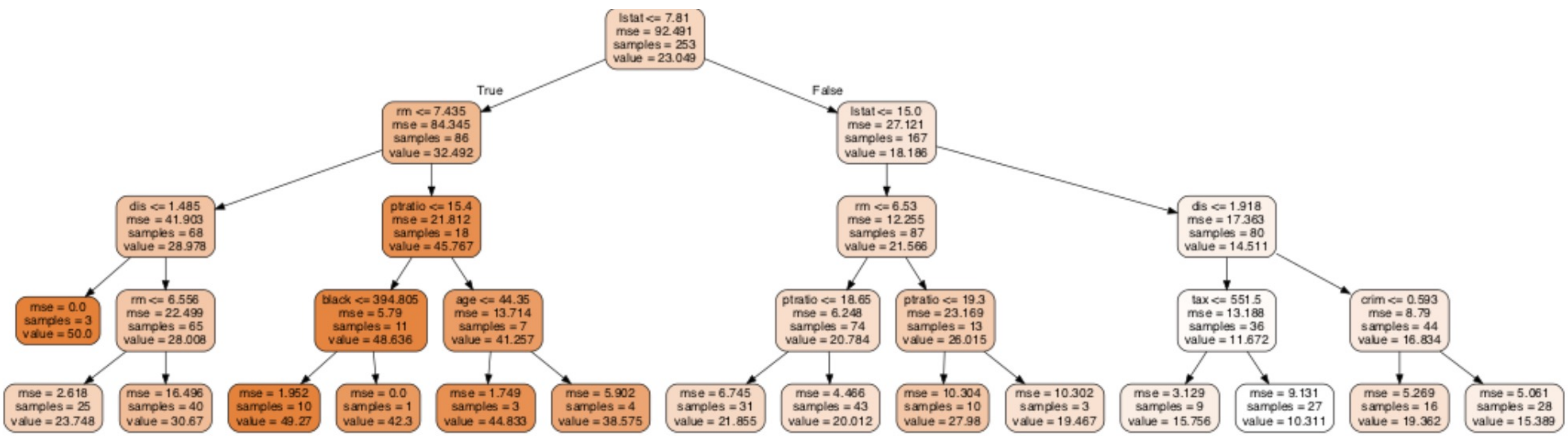
```
regr_tree_boston = DecisionTreeRegressor(max_depth = 4)
regr_tree_boston.fit(X_train, y_train)
pred = regr_tree_boston.predict(X_test)
mspe = mean_squared_error(y_test, pred)
mspe
```

```
23.81737151382862
```

increasing depth reduces test MSE

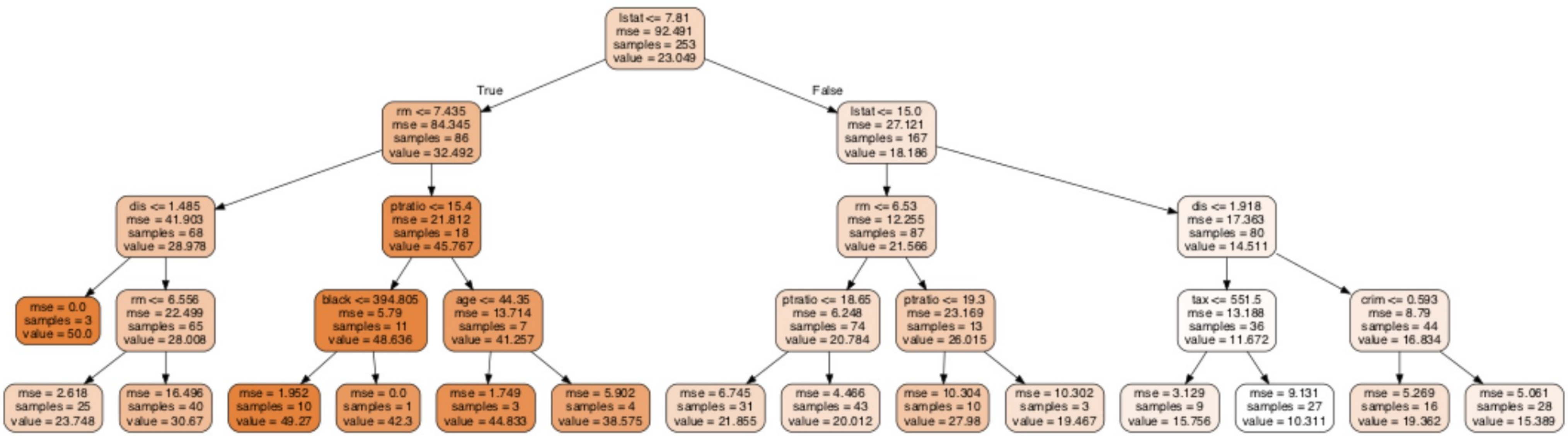
Example – Tree depth = 4

```
regr_tree_boston = DecisionTreeRegressor(max_depth = 4)
regr_tree_boston.fit(X_train, y_train)
```



Example – Display the Tree

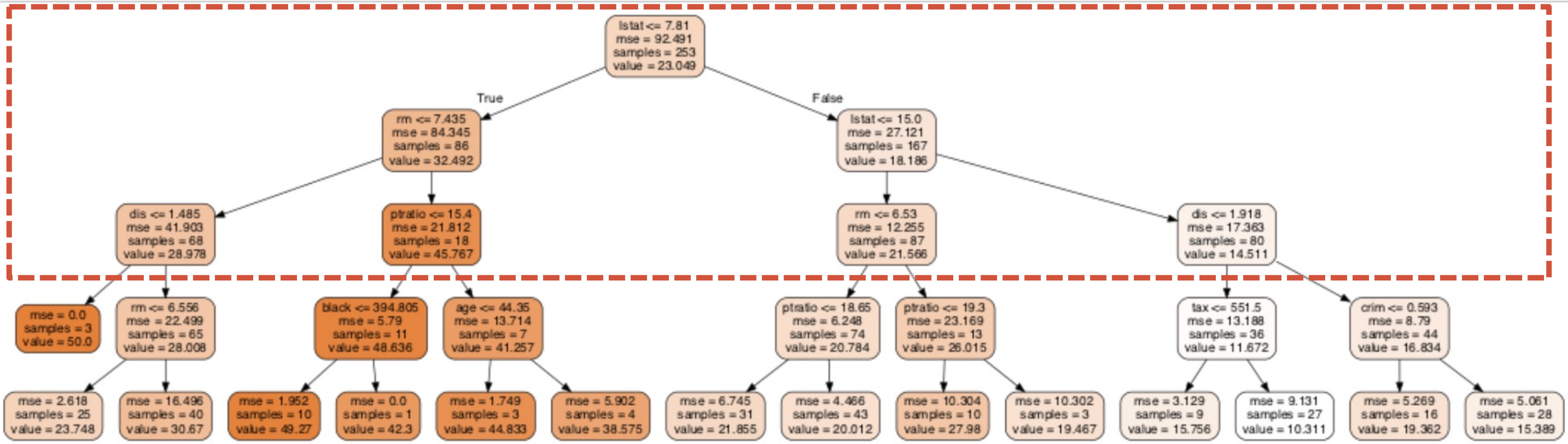
```
dot_data = export_graphviz(regr_tree_boston,
                           feature_names=col_names,
                           out_file=None,
                           filled=True,
                           rounded=True)
pydot_graph = pydotplus.graph_from_dot_data(dot_data)
pydot_graph.set_size('"11,11!"')
pydot_graph.write_png('resized_tree.png');
Image(pydot_graph.create_png())
```



Example – Compare Trees

```
dot_data = export_graphviz(regr_tree_boston,
                                feature_names=col_names,
                                out_file=None,
                                filled=True,
                                rounded=True)
pydot_graph = pydotplus.graph_from_dot_data(dot_data)
pydot_graph.set_size('"11,11!"')
pydot_graph.write_png('resized_tree.png');
Image(pydot_graph.create_png())
```

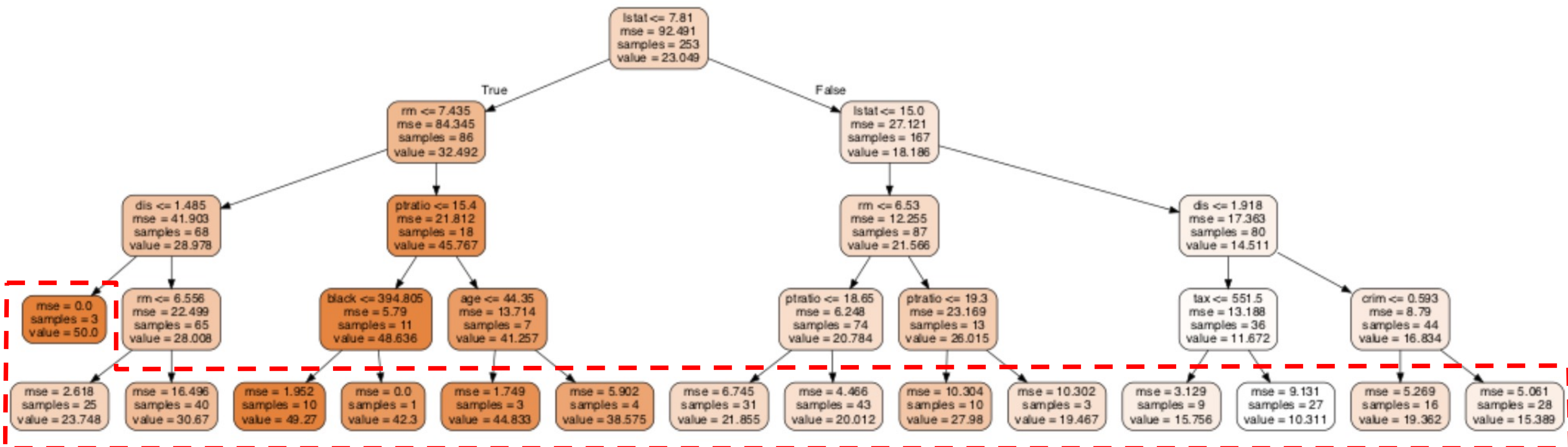
depth = 2



depth = 2 resulted in 4 Terminal Regions

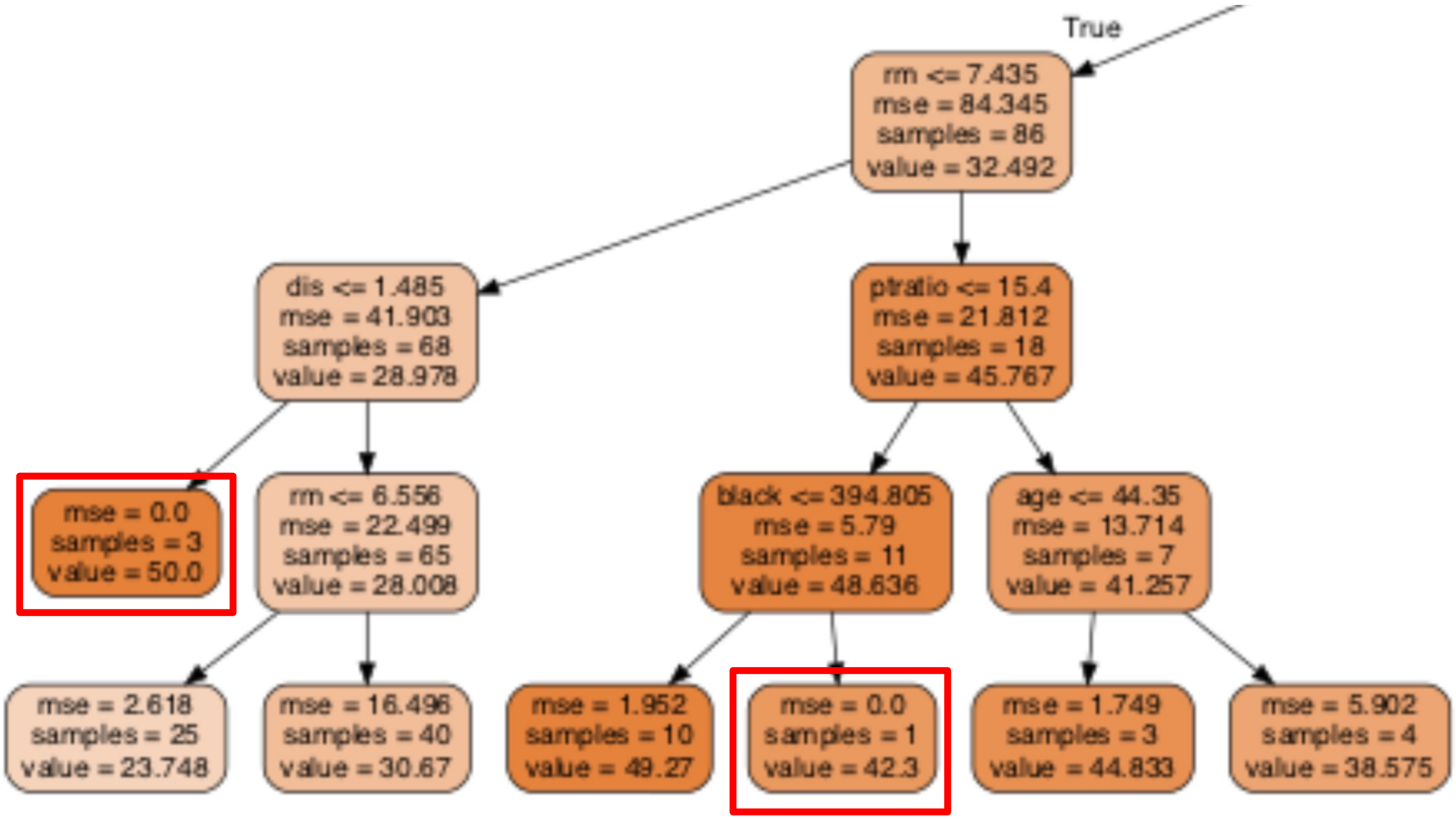
Example – Boston dataset

```
dot_data = export_graphviz(regr_tree_boston,
                                feature_names=col_names,
                                out_file=None,
                                filled=True,
                                rounded=True)
pydot_graph = pydotplus.graph_from_dot_data(dot_data)
pydot_graph.set_size('"11,11!"')
pydot_graph.write_png('resized_tree.png');
Image(pydot_graph.create_png())
```



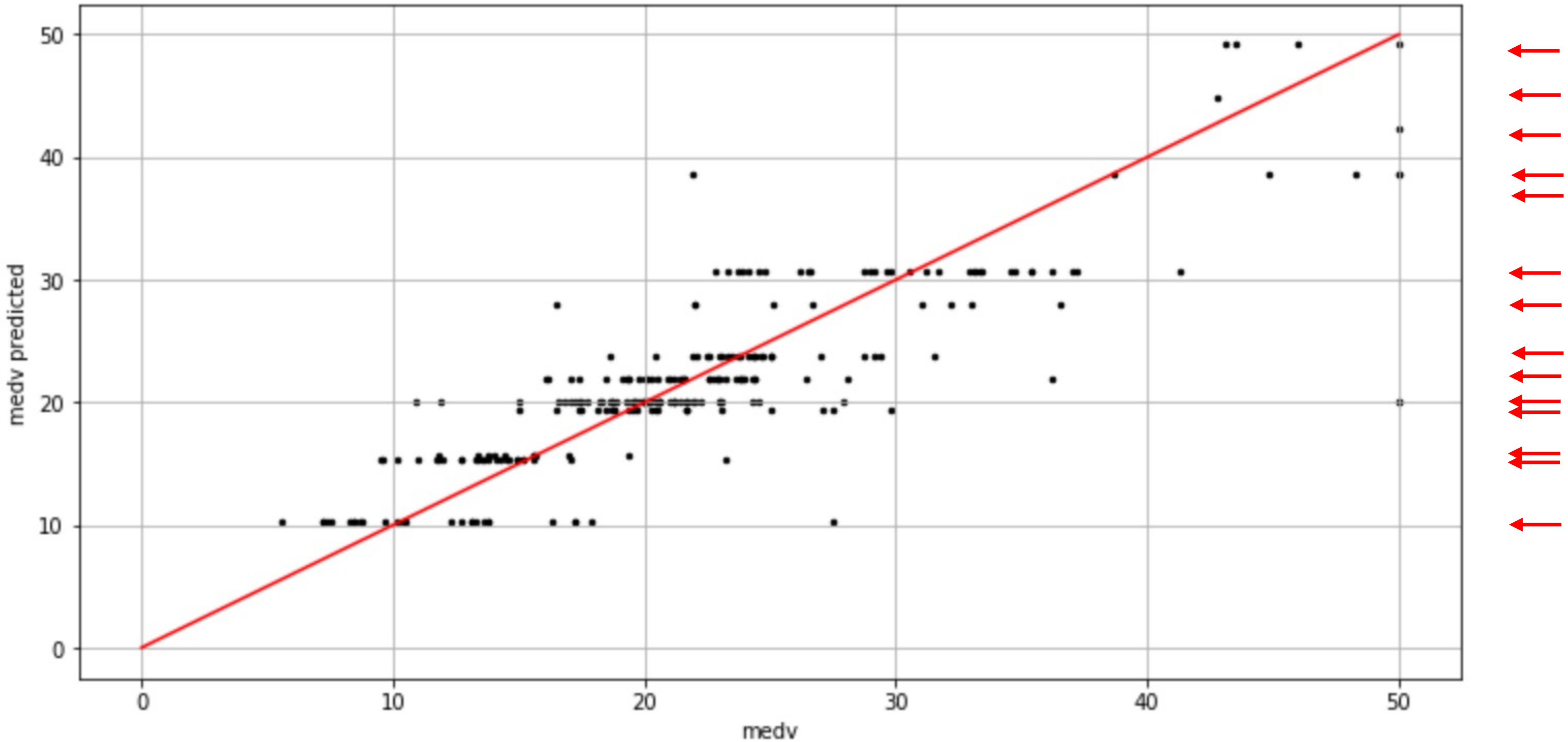
depth = 4, results in 15 Terminal Regions

Example – Boston dataset



Some terminal nodes with MSE = 0

Example – Boston dataset



depth = 4, results in 15 predictions

Holdout Cross Validation

Example – MSPE of a tree with depth = 4

```
from sklearn.metrics import mean_squared_error
```

max_depth = 4

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

```
23.81737151382862
```

increasing depth reduces test MSE

Holdout Validation - tuning hyperparameter max_depth

```
X_nontest, X_test, y_nontest, y_test = train_test_split(X, y,
                                                    train_size = 0.60,
                                                    test_size = 0.40,
                                                    random_state = 0)
```

```
X_train, X_validation, y_train, y_validation = train_test_split(X_nontest,
                                                                y_nontest,
                                                    train_size = 0.5,
                                                    test_size = 0.5,
                                                    random_state = 0)
```

```
model = DecisionTreeRegressor(random_state=1)

validation_mspe = []

for i in range(2,22):
    model.set_params(max_depth = i,random_state=1)
    model.fit(X_train, y_train)
    pred = model.predict(X_validation)
    mspe = mean_squared_error(y_validation, pred)
    validation_mspe.append(mspe)
```

Holdout Cross Validation for max_depth

```
df = pd.DataFrame(validation_mspe,columns = ['MSPE'])
df.index = depths
df.index.name = 'depth'
df
```

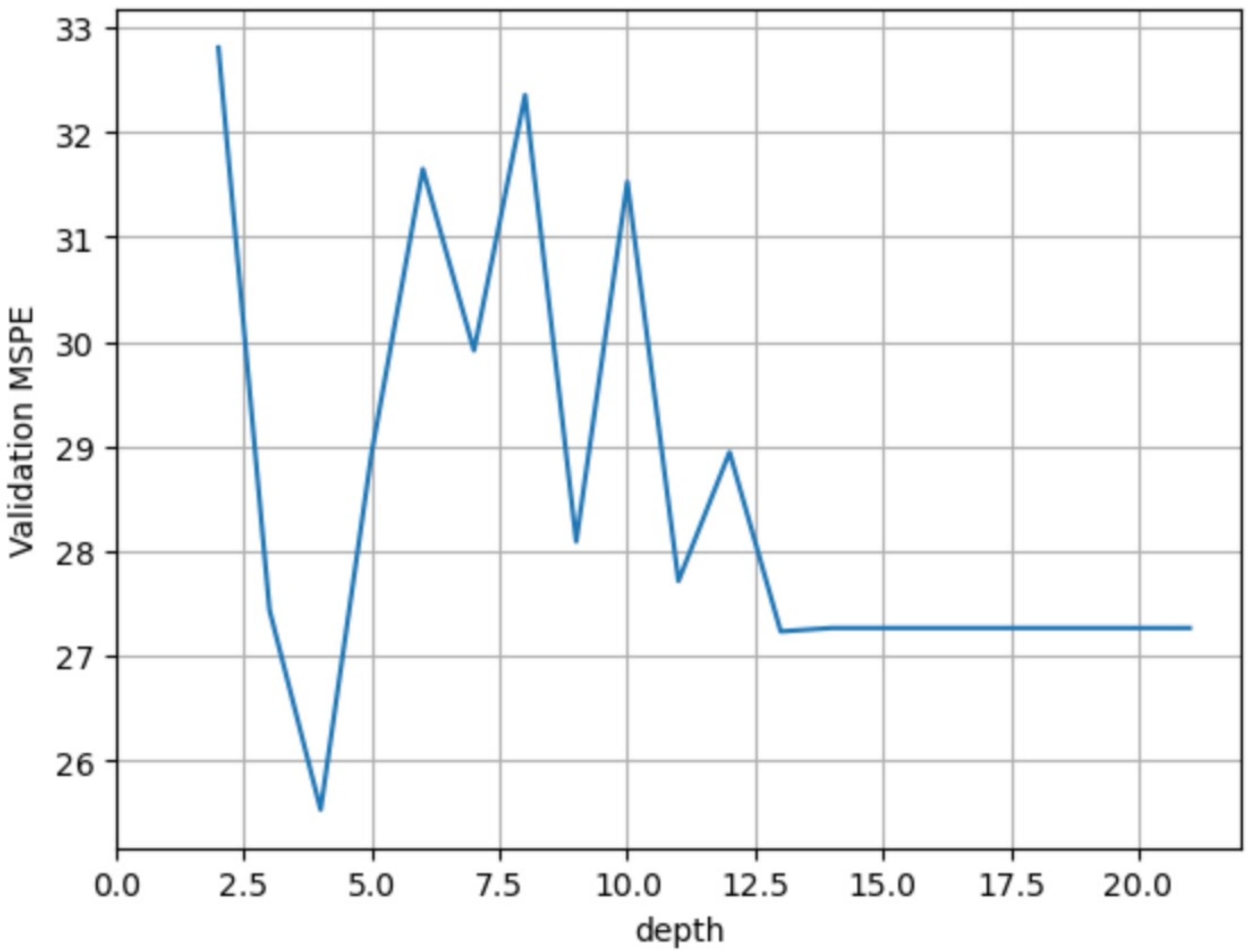
depth	MSPE
2	32.807268
3	27.436607
4	25.529841
5	28.941534
6	31.648445
7	29.914798
8	32.354046

depth	MSPE
9	28.091195
10	31.524097
11	27.713525
12	28.942931
13	27.231859
14	27.264539
15	27.264539

depth	MSPE
16	27.264539
17	27.264539
18	27.264539
19	27.264539
20	27.264539
21	27.264539

Holdout Cross Validation for max_depth

```
df.plot(grid=True,legend=False,xlim = (0,22))
plt.ylabel('Validation MSPE');
```



Holdout Cross Validation for max_depth

```
# Test MSE
model6 = DecisionTreeRegressor(max_depth = 4,
                               random_state=1)
model6.fit(X_nontest, y_nontest)
pred = model6.predict(X_test)
mspe = mean_squared_error(y_test, pred)
mspe
```

```
15.279743835776737
```

Example – Feature Importance

- What is the importance of X_1?
- The most X_1 is used, the more important it is
- How much SSE decreases due to splits due to X_1
- Feature importance is between 0 and 1
  - 0 (if X_1 is not used for splits)
  - 1 (if X_1 is used for all splits)

Example – Feature Importance

```python
# Test MSE
model6 = DecisionTreeRegressor(max_depth = 4,
                               random_state=1)
model6.fit(X_nontest, y_nontest)
pred = model6.predict(X_test)
mspe = mean_squared_error(y_test, pred)
mspe
```

```
15.279743835776737
```

Feature Importances for best model

```python
model6.feature_importances_
```

```
array([0.03035261, 0.        , 0.        , 0.        , 0.00636331,
       0.27144691, 0.00194428, 0.04951238, 0.        , 0.00494618,
       0.02941881, 0.        , 0.60601552])
```

Example – Feature Importance

```
# Test MSE
model6 = DecisionTreeRegressor(max_depth = 4,
                               random_state=1)
model6.fit(X_nontest, y_nontest)
pred = model6.predict(X_test)
mspe = mean_squared_error(y_test, pred)
mspe
```

```
15.279743835776737
```

Feature Importances for best model

predictors with zero feature importance

```
model6.feature_importances_
```

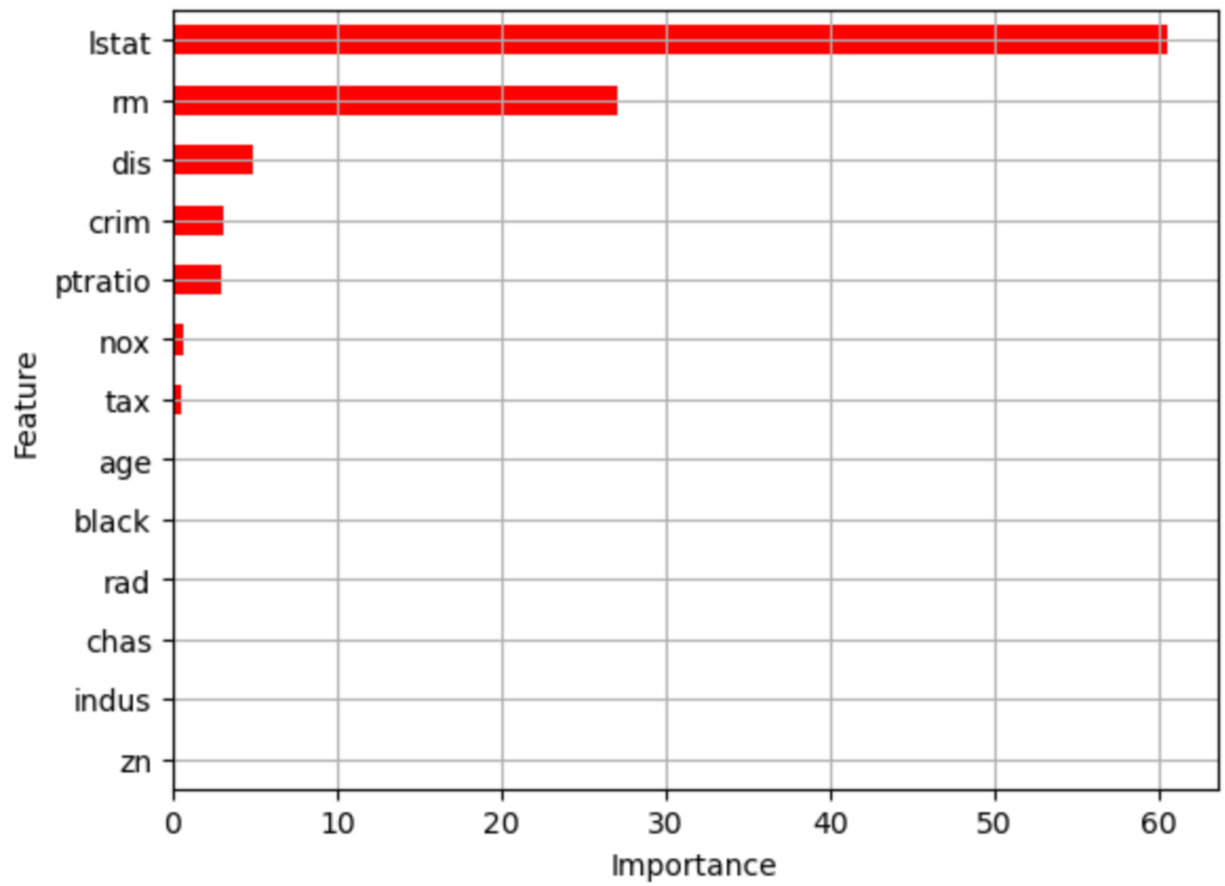
```
array([0.03035261, 0.        , 0.        , 0.        , 0.00636331,
       0.27144691, 0.00194428, 0.04951238, 0.        , 0.00494618,
       0.02941881, 0.        , 0.60601552])
```

```
df9 = pd.DataFrame(100*model6.feature_importances_,
                   index = X.columns,
                   columns=['importance'])
df9 = df9.sort_values(by = 'importance',axis=0,
                      ascending=False)
```

Example – Feature Importance

	importance
lstat	60.601552
rm	27.144691
dis	4.951238
crim	3.035261
ptratio	2.941881
nox	0.636331
tax	0.494618
age	0.194428
zn	0.000000
indus	0.000000
chas	0.000000
rad	0.000000
black	0.000000

```
df9 = df9.sort_values(by = 'importance',axis=0,
                      ascending=True)
df9.plot(kind='barh',color='r',legend = False)
plt.xlabel('Importance')
plt.ylabel('Feature')
```



K-Fold Cross Validation

Kfold cross validation

.

5-fold cross validation with max_depth = 6

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

```
model = DecisionTreeRegressor(max_depth=6,random_state=1)
mspes = cross_val_score(model,X,y,cv=kfold,
                        scoring = 'neg_mean_squared_error')
```

```
mspes
```

```
array([-16.04374666, -51.6247105 , -24.23051498, -28.52711802,
       -13.78521779])
```

```
# Test MSE
```

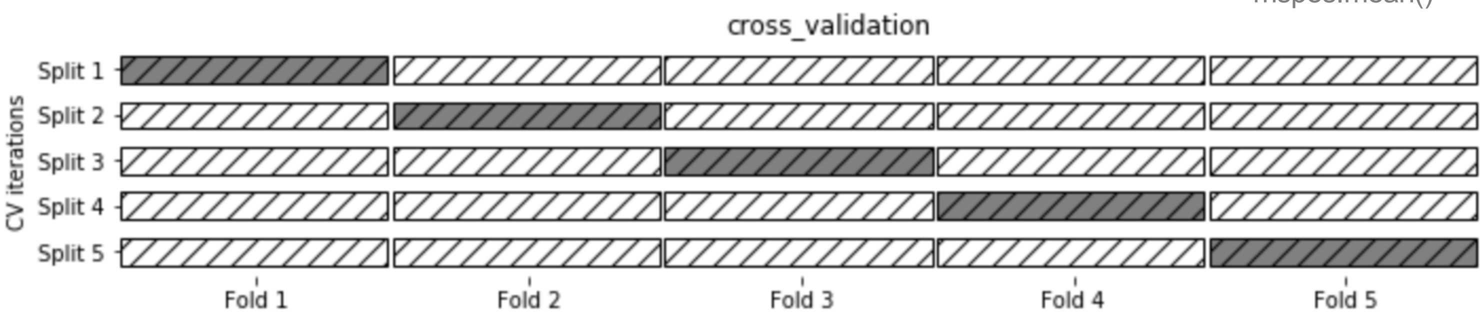
```
-mspes.mean()
```

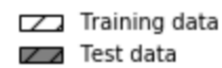
```
26.84226159037823
```

Kfold cross validation – no hyperparameters

```
model  = DecisionTreeRegressor(max_depth=6)
mspes = cross_val_score(model, X, y, cv=kfold)
mspes.mean()
```

.

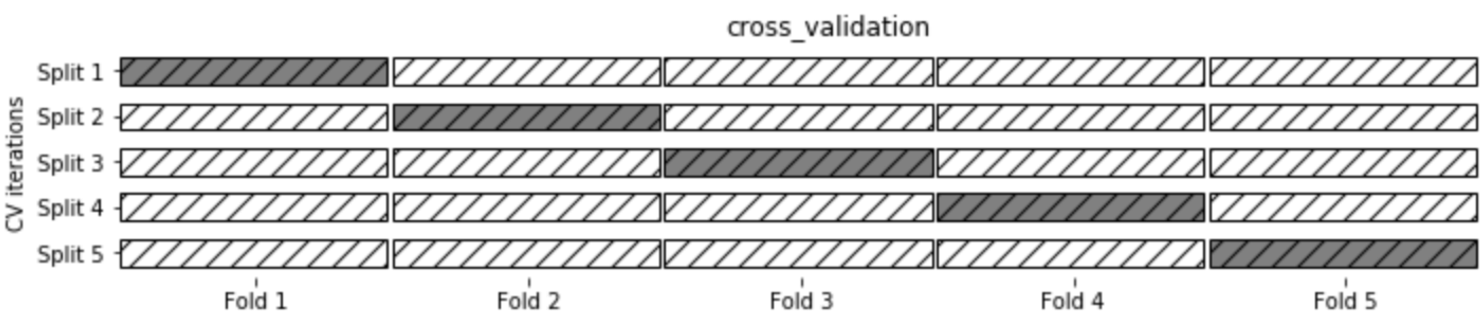




5 test sets

```
X_train,X_test,y_train,y_test = train_test_split(X,y)
model  = DecisionTreeRegressor(random_state = 1)
parameters = {' ',:[ ],… }
grid = GridSearchCV(model,parameters,cv=5)
grid.fit(X_train,y_train)
grid.score(X_test,y_test)
```

Kfold cross validation hyperparameter tuning



Only 1 test set

K-fold cross – hyperparameter tuning

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

```
model = DecisionTreeRegressor(random_state=1)
parameters = {'max_depth':range(3,20)}
```

```
grid = GridSearchCV(model, parameters, cv=5,
                    scoring= 'neg_mean_squared_error')
grid.fit(X_train, y_train);
```

K-fold cross – hyperparameter tuning

.

```
grid = GridSearchCV(model, parameters, cv=5,
                    scoring= 'neg_mean_squared_error')
grid.fit(X_train, y_train);
```

```
# display best depth
grid.best_params_
```

```
{'max_depth': 7}
```

```
# Validation set MSE
-grid.best_score_
```

```
23.346767350514753
```

K-fold cross – hyperparameter tuning

.

both are equivalent

```
# Test MSE
-grid.score(X_test,y_test)
```

```
24.229018548709014
```

```
# Test MSE
```

```
best_model = grid.best_estimator_
best_model.fit(X_train, y_train)
ypred = best_model.predict(X_test)
mean_squared_error(y_test, ypred)
```

```
24.229018548709014
```

K-fold cross validation – Feature importance

```
best_model = grid.best_estimator_
best_model.fit(X_train, y_train)
ypred = best_model.predict(X_test)
mean_squared_error(y_test, ypred)
```

```
24.229018548709014
```

Feature Importances for best model

```
best_model.feature_importances_
```

```
array([0.01006972, 0.00067524, 0.00243003, 0.00066191, 0.00097239,
       0.23605734, 0.01446142, 0.08366883, 0.00218774, 0.01565828,
       0.02521014, 0.00762343, 0.60032354])
```

```
df9 = pd.DataFrame(100*best_model.feature_importances_,
                   index = X.columns,
                   columns=['importance'])
df9 = df9.sort_values(by = 'importance',axis=0,
                      ascending=False)
```

K-fold cross validation – Feature importance

	importance
lstat	60.032354
rm	23.605734
dis	8.366883
ptratio	2.521014
tax	1.565828
age	1.446142
crim	1.006972
black	0.762343
indus	0.243003
rad	0.218774
nox	0.097239
zn	0.067524
chas	0.066191

```
df9 = df9.sort_values(by = 'importance',axis=0,
                      ascending=True)
df9.plot(kind='barh',color='r',legend = False)
plt.xlabel('Importance')
plt.ylabel('Feature')
```