ESS2222

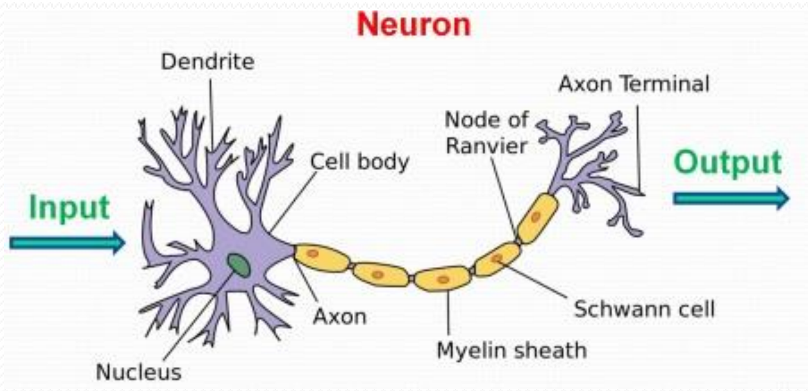
Lecture 9 – Decision Trees and Random Forest

Hosein Shahnas

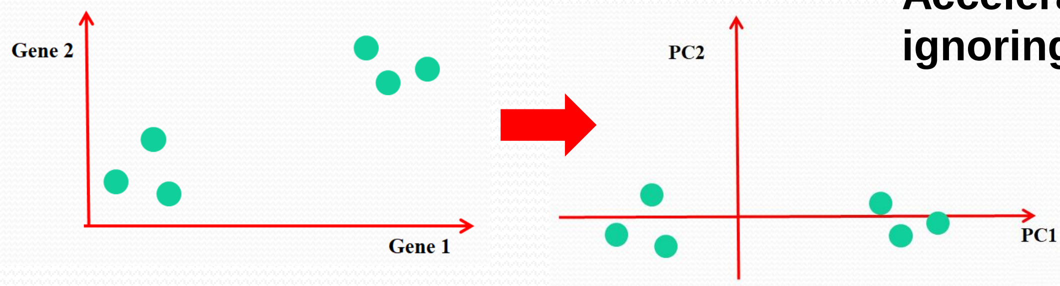
University of Toronto, Department of Earth Sciences,

Outline

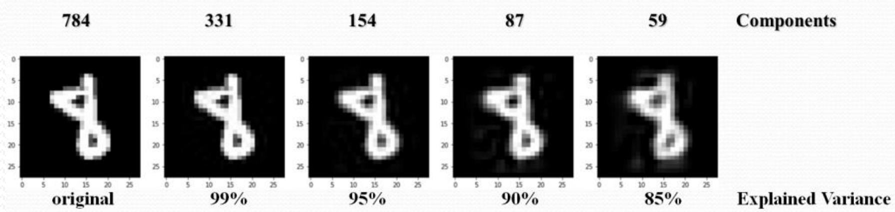
- ❑ Decision Trees
- ❑ Gini index
- ❑ Decision Tree – Iris Classification Problem
- ❑ Random Forest Algorithm
- ❑ Bagging – Bootstrap Sampling
- ❑ Random Forest – Image recognition
- ❑ Feature Importance



Review of Lecture 8

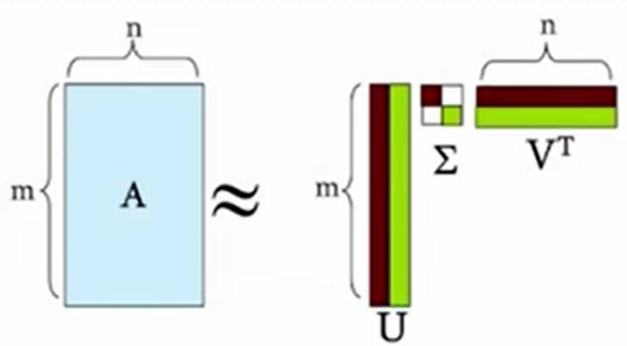


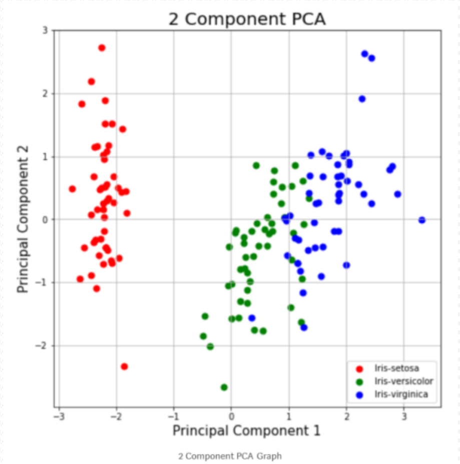
Accelerating the computational process by ignoring less important features



Singular Value Decomposition (SVD)

$$A_{[mn]} = U_{[mr]}\, \Sigma_{[rr]}\, V_{[nr]}^{T}$$





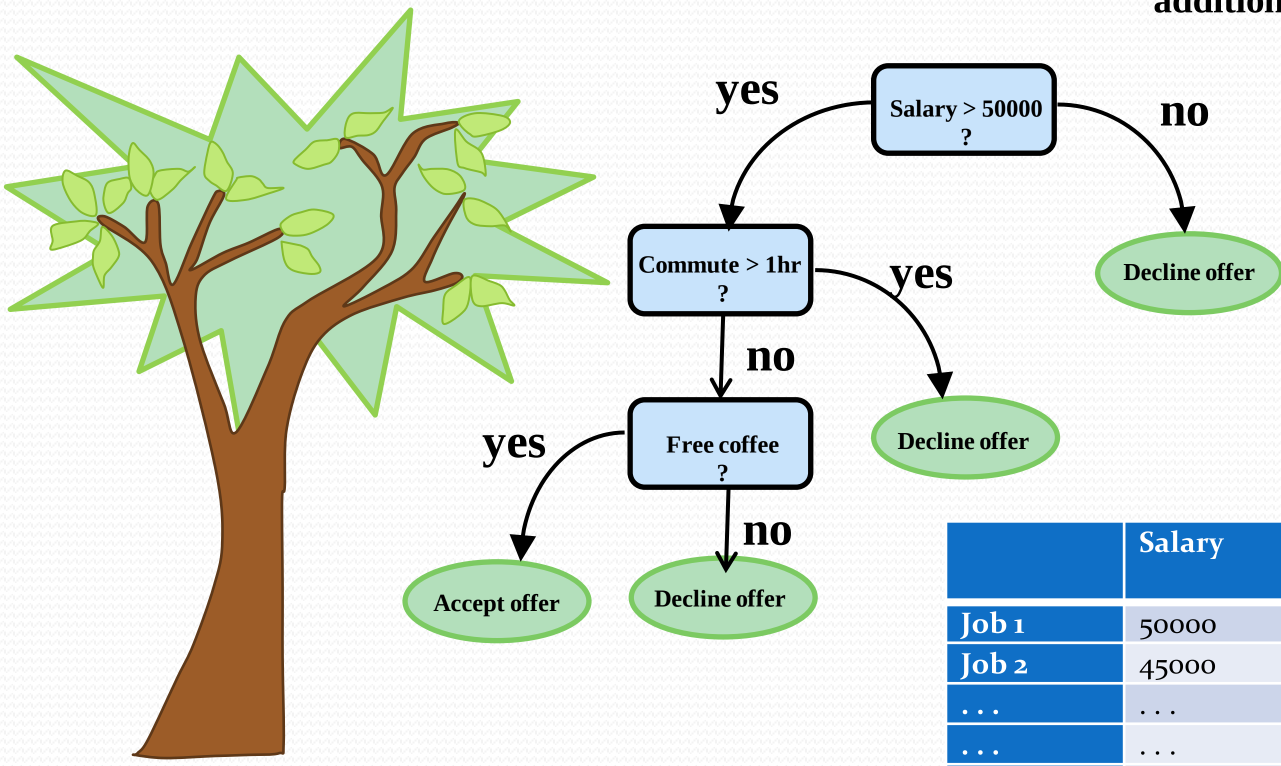
Visualization

Classification and Regression Tree (CART)

Decision Tree:

Should I accept the job?

Root node: The top most decision node
Decision nodes: Nodes that have two or more braches
Leaf nodes: Nodes of the tree that have no additional nodes coming off them



	Salary	Commute	Free coffee	Class
Job 1	50000	2 hr	Yes	1
Job 2	45000	3 hr	No	0
. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .
Job n	53000	1 hr	No	1

Tree-based methods are one of the commonly used supervised learning algorithms in machine learning, both for regression and classification problems.

Decision Trees

Decision trees:

A decision tree is a simple but powerful supervised learning method that uses tree-like model of decisions and their possible consequences. They are used in both classification and regression problems.

Unlike the highly dimensional SVM, where it is impossible for a human brain to even imagine how the hyperplane built looks like, the decision trees provide very good visualization on the steps of decisions and the relative importance of the features.

class sklearn.tree.DecisionTreeRegressor *for classification*
class sklearn.tree.DecisionTreeClassifier *for regression*

Gini Index

Gini index: The gini index is a number describing the quality of the split of a node on a variable (feature).

If a data set D contains samples from C classes, gini index is defined as:

$$gini(D) = 1 - \sum_{c=1}^{C} {P^2}_c$$

where P_c is the relative frequency of class c in D

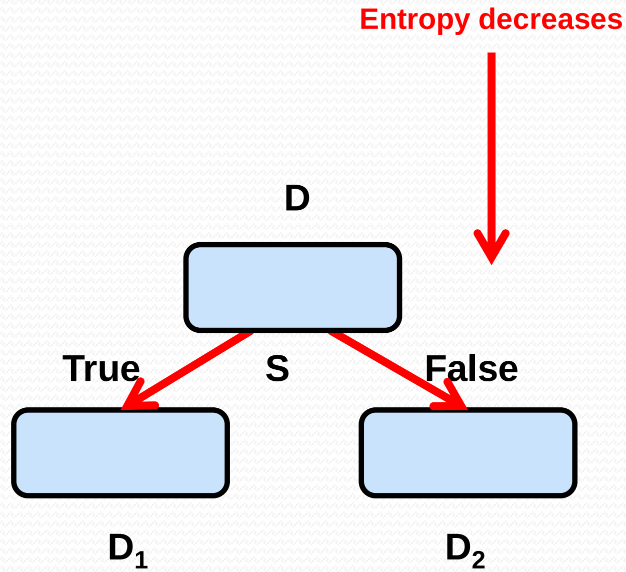
If a data set **D** splits on **S** into two subsets D_1 and D_2, the gini index is defined as:

$$gini_S(D) = \frac{D_1}{D} gini(D_1) + \frac{D_2}{D} gini(D_2)$$

where $gini(D_1) < gini(D)$, $gini(D_2) < gini(D)$

Reduction in impurity:

$$\Delta\, gini\,(S) = gini\,(D) - giniS\,(D)$$



Information Gain Entropy:

Information gain is a measure of decrease in entropy after the data is split.

Information gain entropy is another criterion for choosing the features in splitting. Entropy is a measure of the degree of disorder or randomness in the system.

$$H(D) = 1 - \sum_{c=1}^{C} P_c\, log(P_c)$$

Gini Index

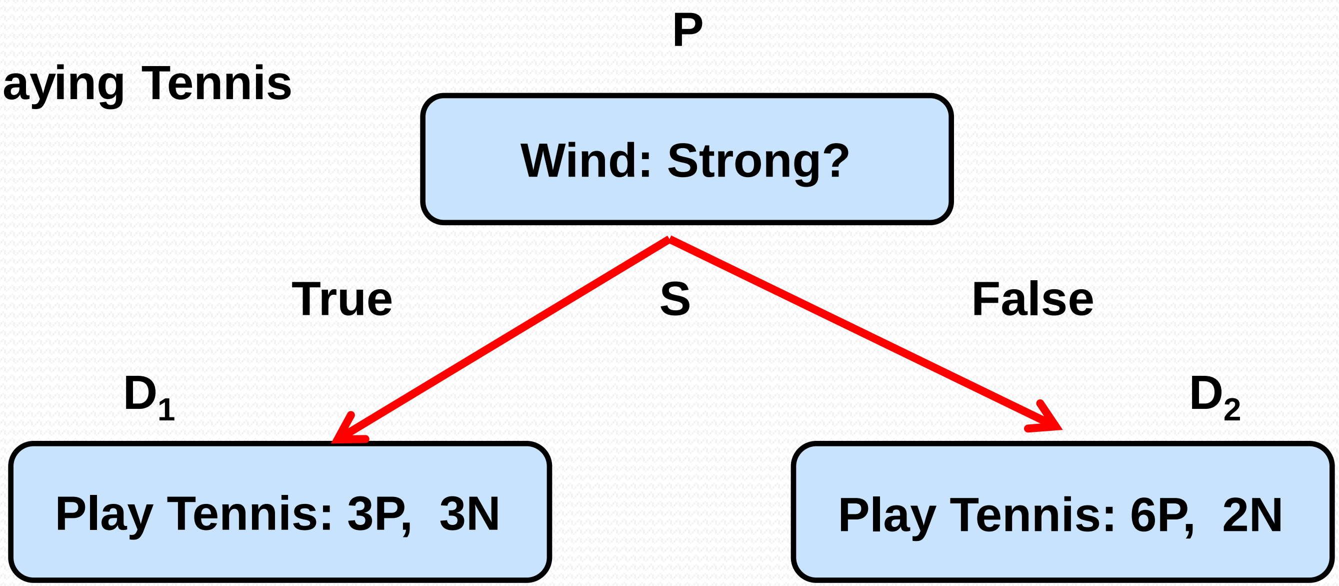
Example: **Playing Tennis**

Features **Class labels**

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Gini Index (Gini Impurity)

Example: **Playing Tennis**



gini (Play Tennis | Wind =True) = 1 $- \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2$ **= 0.5**

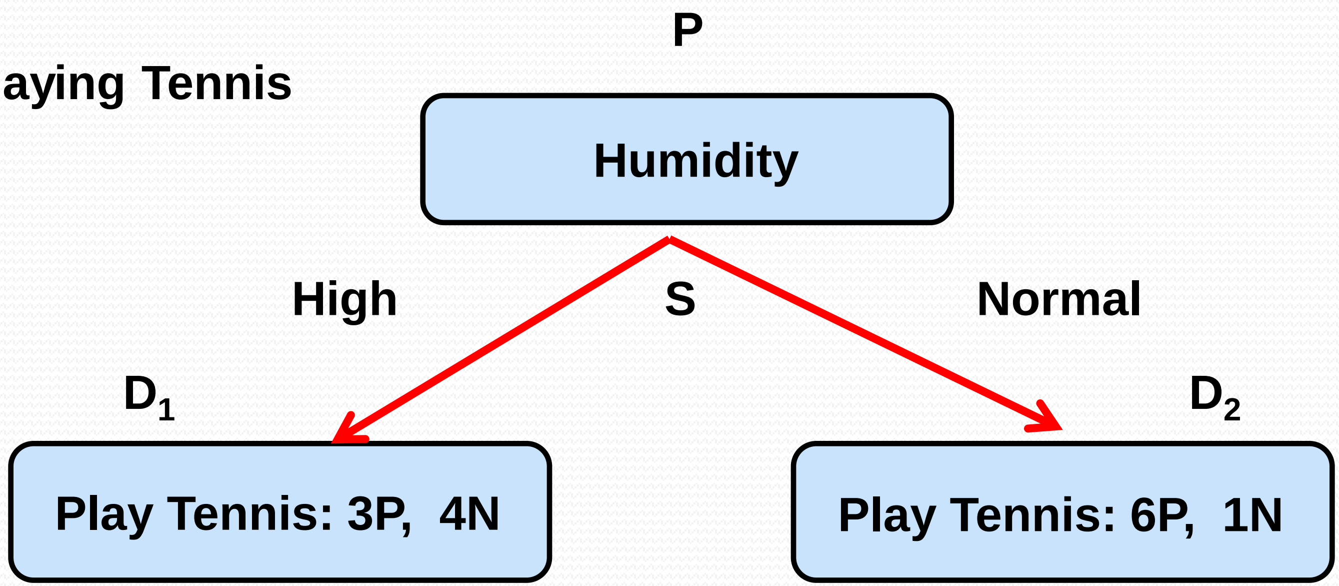
gini (Play Tennis | Wind = False) = 1 $- \left(\frac{6}{8}\right)^2 - \left(\frac{2}{8}\right)^2$ **= 0.375**

$$gini_S(D) = \frac{D_1}{D} gini(D_1) + \frac{D_2}{D} gini(D_2)$$

$$= \frac{6}{14} \times 0.5 + \frac{8}{14} \times 0.375 = 0.4286$$

Gini Index (Gini Impurity)

Example: **Playing Tennis**



gini (Play Tennis | Humidity = High) = $1 - \left(\frac{3}{7}\right)^2 - \left(\frac{4}{7}\right)^2$ = 0.4898

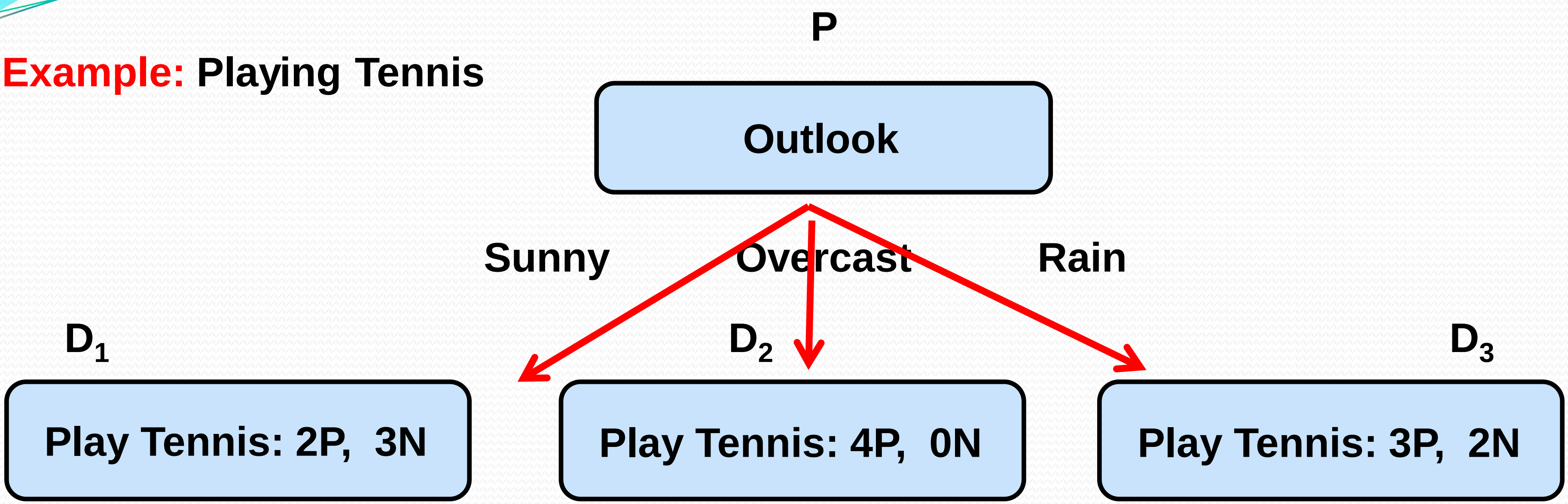
gini (Play Tennis | Humidity = Normal) = $1 - \left(\frac{6}{7}\right)^2 - \left(\frac{1}{7}\right)^2$ = 0.2449

$$gini_S(D) = \frac{D_1}{D}\, gini(D_1) + \frac{D_2}{D}\, gini(D_2)$$

$$= \frac{6}{14} \times 0.4898 + \frac{8}{14} \times 0.2449 = 0.3674$$

Gini Index (Gini Impurity)

Example: **Playing Tennis**



gini (Play Tennis | Outlook = Sunny) = 1 $- \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2$ **= 0.48**

gini (Play Tennis | Outlook = Overcast) = 1 $- \left(\frac{4}{4}\right)^2 - \left(\frac{0}{4}\right)^2$ **= 0**

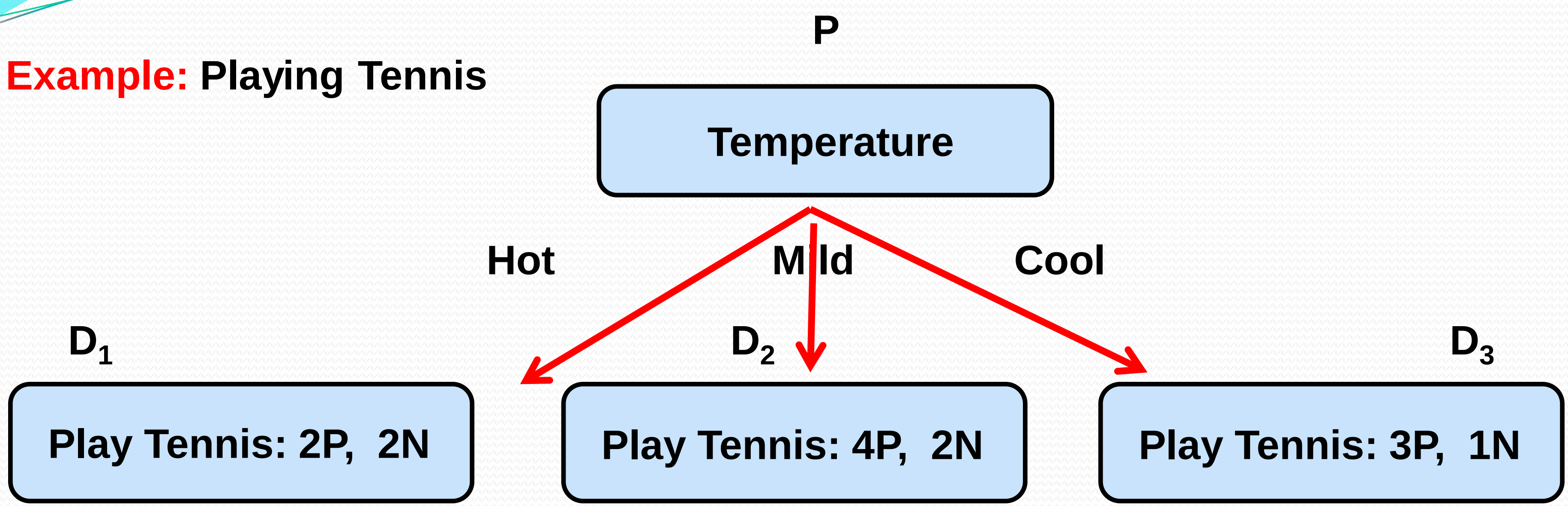
gini (Play Tennis | Outlook = Rain) = 1 $- \left(\frac{3}{5}\right)^2 - \left(\frac{3}{5}\right)^2$ **= 0.48**

$$gini_S(D) = \frac{D_1}{D} gini(D_1) + \frac{D_2}{D} gini(D_2) + \frac{D_3}{D} gini(D_3)$$

$$= \frac{5}{14} \times 0.48 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.48 = 0.3429$$

Gini Index (Gini Impurity)

Example: Playing Tennis



$$\text{gini (Play Tennis | Temperature = Hot)} = 1 - \left(\frac{2}{4}\right)^2 - \left(\frac{3}{4}\right)^2 = 0.5$$

$$\text{gini (Play Tennis | Temperature = Mild)} = 1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2 = 0.4444$$

$$\text{gini (Play Tennis | Temperature = Cool)} = 1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.375$$

$$gini_S(D) = \frac{D_1}{D} gini(D_1) + \frac{D_2}{D} gini(D_2) + \frac{D_3}{D} gini(D_3)$$

$$= \frac{4}{14} \times 0.5 + \frac{6}{14} \times 0.4444 + \frac{4}{14} \times 0.375 = 0.4405$$

Gini Index (Gini Impurity)

Example: Playing Tennis

$gini_S$(Windy) = 0.4286
$gini_S$ (Humidity) = 0.3674
$gini_S$ (Outlook) = 0.3429
$gini_S$ (Temperature) = 0.4405

The splitting gain for Temperature feature is high, therefore we chose temperature as the feature for splitting.

$$gini(D) = 1 - \sum_{c=1}^{C} P^2{}_c$$

$$gini(D) = 1 - \left(\frac{5}{14}\right)^2 - \left(\frac{9}{14}\right)^2 = 0.541$$

Parameters for DecisionTreeClassifier

criterion : string, optional (default=”gini”)
The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

splitter : string, optional (default=”best”)
The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

max_depth : int or None, optional (default=None)
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : int, float, optional (default=2)
The minimum number of samples required to split an internal node.

min_samples_leaf : int, float, optional (default=1)
The minimum number of samples required to be at a leaf node.

max_features : int, float, string or None, optional (default=None)
The number of features to consider when looking for the best split.

max_leaf_nodes : int or None, optional (default=None)
Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

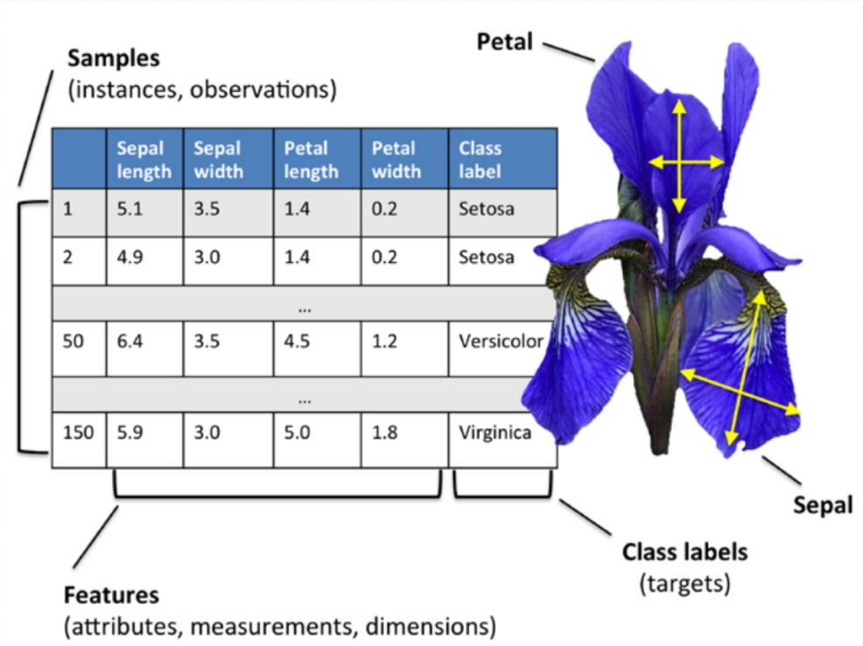
DecisionTreeRegressor

criterion : strstring, optional (default=”mse”)
The function to measure the quality of a split. Supported criteria are “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss

Decision Tree - Iris Classification Problem

Iris problem:
Three class problem with four features.

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV *for grid search*



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Decision Tree - Iris Problem

```python
#==============================================
import pandas as pd
#import graphviz
#from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
import sys
import os.path
# Setting random seed.
seed = 10
#==============================================
save_path = os.path.dirname(os.path.abspath(__file__))
print(save_path)
#==============================================
#============================================== read data
# Loading Iris dataset.
data = pd.read_csv('data/iris.csv')
print('data.head(3) = ', data.head(3))
#============================================== read data

#============================================== cunstruct arrays from raw data
n_s = 150  #number of samples
y = data.iloc[0:n_s, 4].values    # get data at the fifth (4) column for the first 100 lines
X = data.iloc[0:n_s, [0,1 , 2, 3]].values
print('X[0] = ', X[0])
print('y[0] = ', y[0])

Feature_size = int(X.size/y.size)
print('Feature_size = ', Feature_size)
print('X.shape = ', X.shape, 'X.shape[0] = ', X.shape[0], 'X.shape[1] = ', X.shape[1])

y = np.reshape(y, (n_s,1))

print('X[0]2 = ', X[0])
print('y[0]2 = ', y[0])
#============================================== cunstruct arrays from raw data

```

Decision Tree - Iris Problem

```
#============================================= train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=seed)
print('X_train.shape = ', X_train.shape)
print('y_train.shape = ', y_train.shape)
print('X_test.shape = ', X_test.shape)
print('y_test.shape = ', y_test.shape)
print('X_test.size  = ', X_train.size)
print('y_train.size  = ', y_train.size)
print('X_test.size  = ', X_test.size)
print('y_test.size  = ', y_test.size)
n_test = y_test.size
n_train = y_train.size
#============================================= train-test split

#============================================= grid search
'''
from sklearn.model_selection import GridSearchCV
parm_grid = dict(min_samples_leaf = [1,3],
                 min_samples_split = [2,5],
                 max_depth = [1,20],
                 max_features = [1,4])
clf_tree=DecisionTreeClassifier(criterion='gini')
grid=GridSearchCV(clf_tree,parm_grid, cv=7)
grid.fit(X_train,y_train)
print('==============================================================')
print("grid.cv_results_ {}".format(grid.cv_results_))
print('==============================================================')
sys.exit()
'''
#============================================= grid search

#============================================= decision tree
'''
tree = DecisionTreeClassifier(criterion='gini',
                              splitter='best',
                              max_depth=None,
                              min_samples_split=2,
                              min_samples_leaf=1,
                              min_weight_fraction_leaf=0.0,
```

Decision Tree - Iris Problem

```
                              max_features=None,
                              random_state=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0.0,
                              min_impurity_split=None,
                              class_weight=None,
                              presort=False)
'''
tree = DecisionTreeClassifier(criterion='gini',
                              min_samples_leaf=5,
                              min_samples_split=5,
                              max_depth=None,
                              random_state=seed)
#================================================ decision tree

#================================================ train
tree.fit(X_train, y_train)
#================================================ train

#================================================ feature importance
print()
print('tree.feature_importances_ = ', tree.feature_importances_)
print()
#================================================ feature importance

#================================================ predit test samples
y_pred = tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('DecisionTreeClassifier accuracy score: {}'.format(accuracy))
print('y_test.shape = ', y_test.shape)
print('y_pred.shape = ', y_pred.shape)
#print(y_test,y_pred)
#================================================ predit test samples

#================================================ predit train samples
y_pred_train = tree.predict(X_train)
#================================================ predit train samples

#====================================================================== Acc. Test data
```

Decision Tree - Iris Problem

```
file_name = 'Mis_class_deg_Test'
completeName = os.path.join(save_path, file_name+".dat")
file1 = open(completeName, "w")
file_name = 'Score_Test'
completeName = os.path.join(save_path, file_name+".dat")
file2 = open(completeName, "w")
file_name = 'Prediction_Test'
completeName = os.path.join(save_path, file_name+".dat")
file3 = open(completeName, "w")

rs = np.zeros(n_test)
rs2 = np.ones((n_test), dtype=bool)

for i in range(0,n_test):
    if (y_pred[i] == y_test [i] ):
        rs[i] = 0.0
    else:
        rs[i] = 1.0

    if rs[i]==0.0:
        rs2[i] = True
        file3.write("%s %s \n" % (y_pred[i], y_test[i]))
    else:
        rs2[i] = False
        file3.write("%s %s %s\n" % (y_pred[i], y_test[i], str(rs2[i])))

    file1.write('{:4.2f}\n'.format(rs[i]))
    file2.write(str(rs2[i]))
    file2.write('\n')

file1.close()
file2.close()
file3.close()

err_test = 0.0
for i in range(0,n_test):
    err_test = err_test + rs[i]
print('Number of errors (Test samples) = ', err_test)
print('Error (Test samples) = ', err_test/float(n_test))
```

Decision Tree - Iris Problem

```python
print('Accuracy (Test samples) = ', 1.0 - err_test/float(n_test))
#====================================================================== Acc. Test data

#====================================================================== Acc. Train data
file_name = 'Mis_class_deg_Train'
completeName = os.path.join(save_path, file_name+".dat")
file1 = open(completeName, "w")
file_name = 'Score_Train'
completeName = os.path.join(save_path, file_name+".dat")
file2 = open(completeName, "w")
file_name = 'Prediction_Train'
completeName = os.path.join(save_path, file_name+".dat")
file3 = open(completeName, "w")

rs = np.zeros(n_train)
rs2 = np.ones((n_train), dtype=bool)

for i in range(0,n_train):
    if (y_pred_train[i] == y_train [i] ):
        rs[i] = 0.0
    else:
        rs[i] = 1.0

    if rs[i]==0.0:
        rs2[i] = True
        file3.write("%s %s \n" % (y_pred_train[i], y_train[i]))
    else:
        rs2[i] = False
        file3.write("%s %s %s\n" % (y_pred_train[i], y_train[i], str(rs2[i])))

    file1.write('{:4.2f}\n'.format(rs[i]))
    file2.write(str(rs2[i]))
    file2.write('\n')

file1.close()
file2.close()
file3.close()

err_train = 0.0
```

Decision Tree - Iris Problem

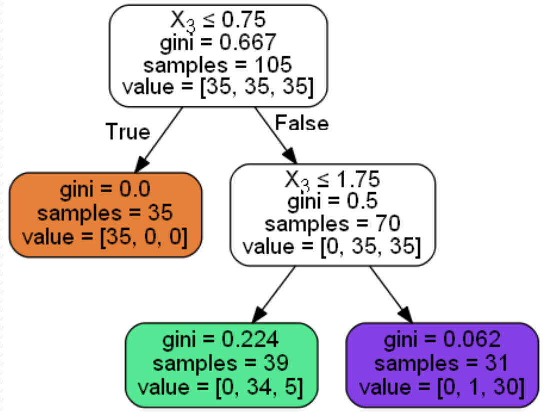
```
for i in range(0,n_train):
    err_train = err_train + rs[i]

print('Number of errors (Train samples) = ', err_train)
print('Error (Train samples) = ', err_train/float(n_train))
print('Accuracy (Train samples) = ', 1.0 - err_train/float(n_train))
#====================================================================== Acc. Train data

#====================================================================== write results
file_name = 'Results'
completeName = os.path.join(save_path, file_name+".dat")
file1 = open(completeName, "w")

file1.write("%s %4.2f \n" % ('Number of errors (Test samples) = ', err_test))
file1.write('\n')
file1.write("%s %4.2f \n" % ('Error (Test samples) = ', err_test/float(n_test)))
file1.write('\n')
file1.write("%s %4.2f \n" % ('Accuracy (Test samples) = ', 1.0 - err_test/float(n_test)))
file1.write('\n')

file1.write("%s %4.2f \n" % ('Number of errors (Train samples) = ', err_train))
file1.write('\n')
file1.write("%s %4.2f \n" % ('Error (Train samples) = ', err_train/float(n_train)))
file1.write('\n')
file1.write("%s %4.2f \n" % ('Accuracy (Train samples) = ', 1.0 - err_train/float(n_train)))
file1.write('\n')

file1.write("%s %s \n" % ('tree.feature_importances_ = ', tree.feature_importances_))
print('tree.feature_importances_ = ', tree.feature_importances_)
file1.write('\n')
file1.close()
#====================================================================== write results

#============================================== predit using test samples

```

Decision Tree - Iris Problem

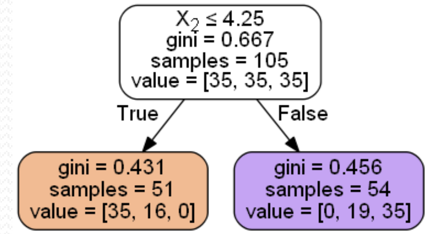
```
#============================================== visualization of tree
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

'''
export_graphviz(decision_tree,
                out_file=None,
                max_depth=None,
                feature_names=None,
                class_names=None,
                label='all',
                filled=False,
                leaves_parallel=False,
                impurity=True,
                node_ids=False,
                proportion=False,
                rotate=False,
                rounded=False,
                special_characters=False,
                precision=3
'''
dot_data = StringIO()

export_graphviz(tree, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())

# Create PDF
graph.write_pdf("iris.pdf")

# Create PNG
graph.write_png("iris.png")
#============================================== visualization of tree
sys.exit()
```

Example 1: **Iris problem**

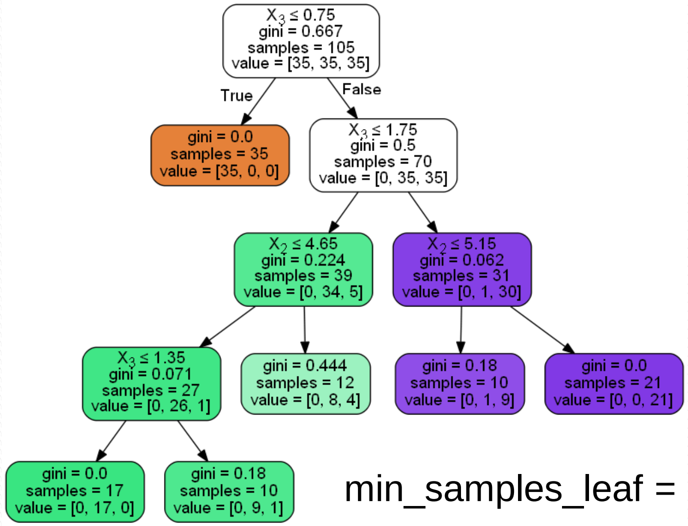
criterion='gini',
splitter='best',
min_samples_split=5,
min_samples_leaf = ?,
max_depth=20,
max_features=None,



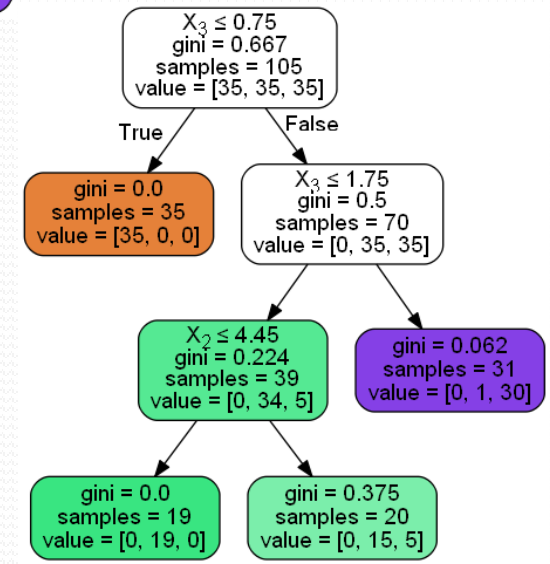
min_samples_leaf = 25



min_samples_leaf = 50



min_samples_leaf = 10

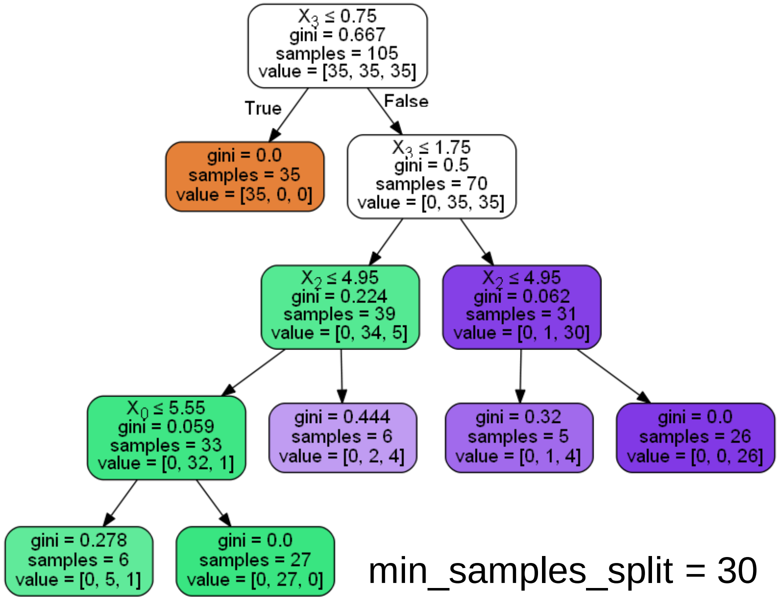


min_samples_leaf = 17

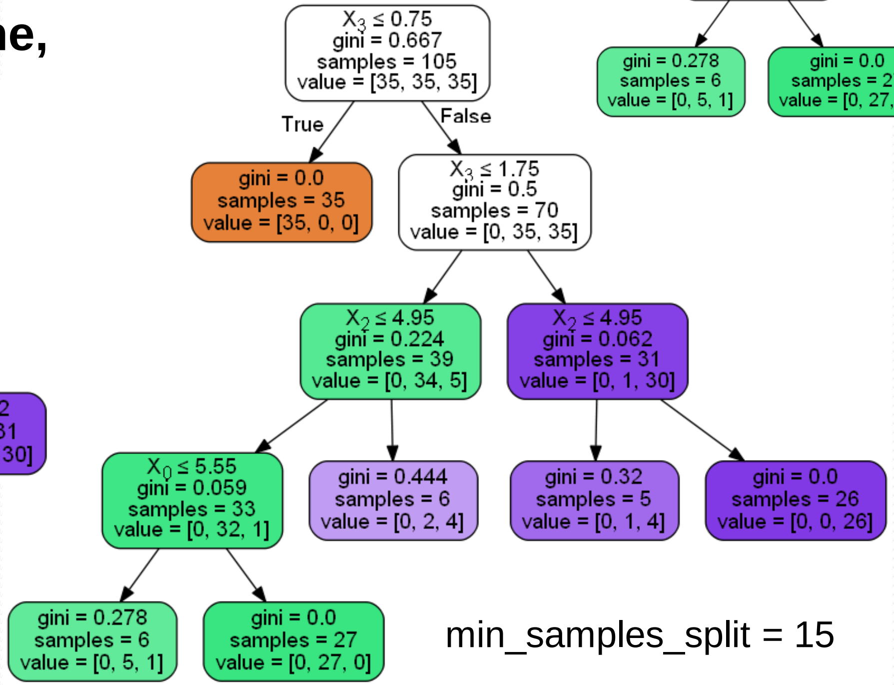
Decision Tree

Example 1: Iris problem

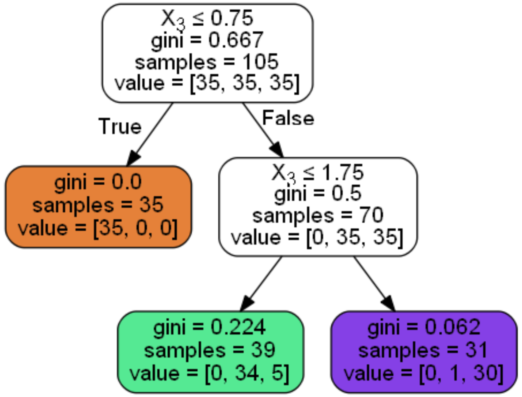
criterion='gini',
splitter='best',
min_samples_split=?,
min_samples_leaf=5,,
max_depth=20,
max_features=None,



min_samples_split = 30



min_samples_split = 15

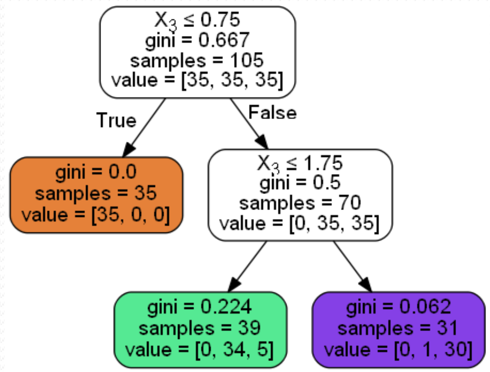


min_samples_split = 65

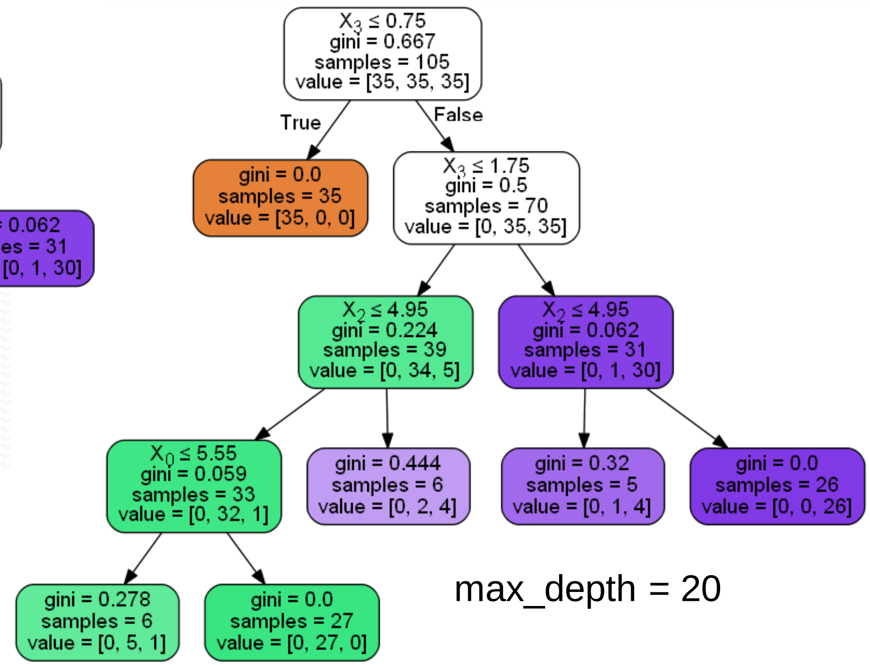
Decision Tree

Example 1: Iris problem

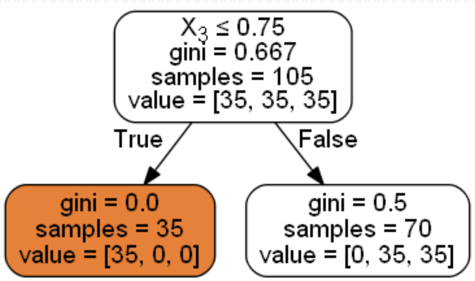
criterion='gini',
splitter='best',
min_samples_split=5,
min_samples_leaf=5,
max_depth = ?,
max_features=None,



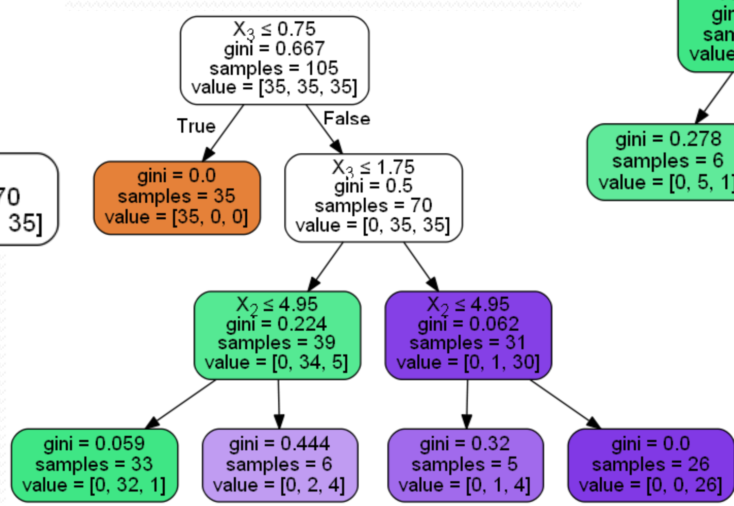
max_depth = 2



max_depth = 20



max_depth = 1

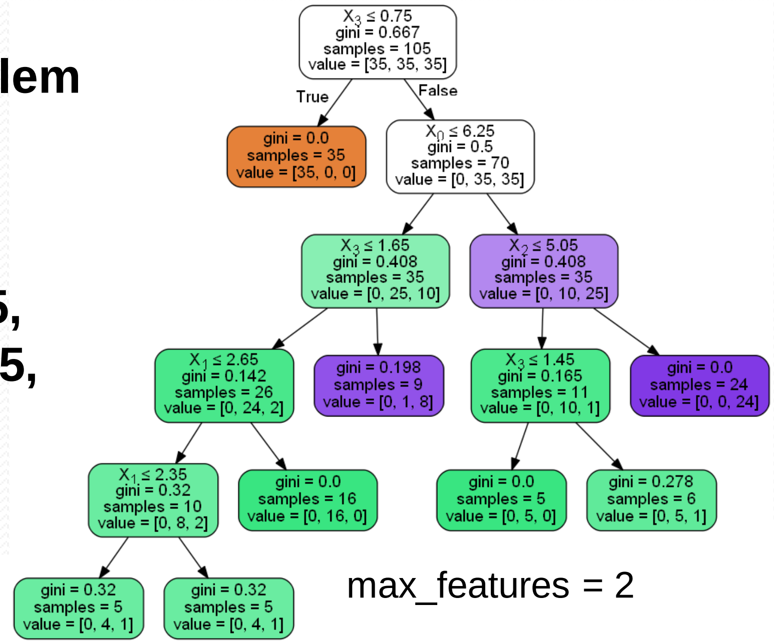


max_depth = 3

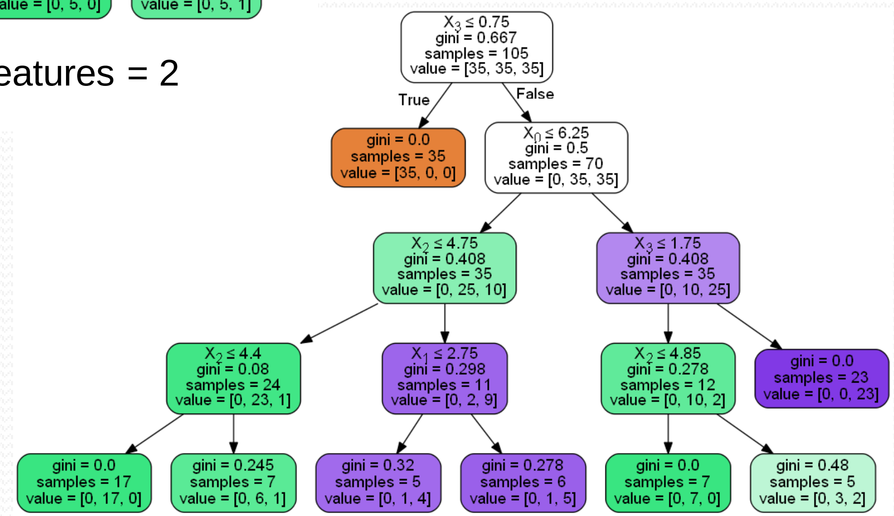
Decision Tree

Example 1: Iris problem

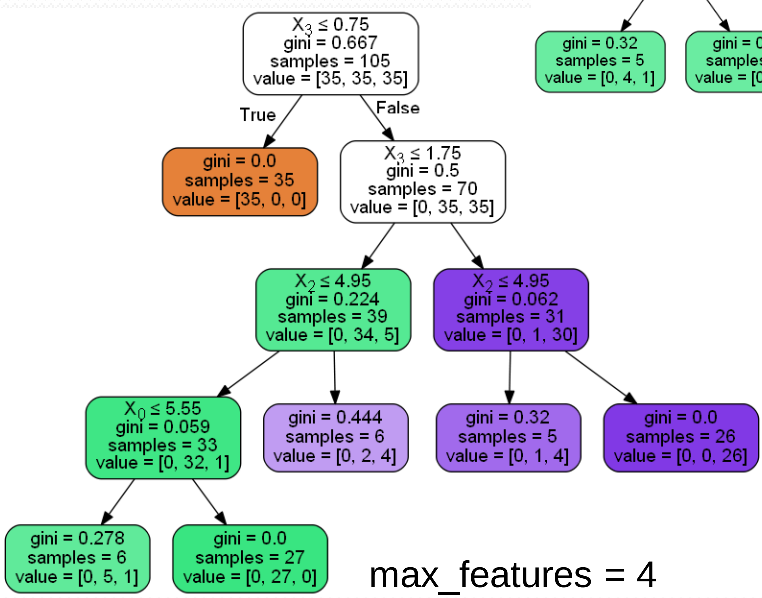
criterion='gini',
splitter='best',
min_samples_leaf=5,
min_samples_split=5,
max_depth=20,
max_features = ?,



max_features = 2



max_features = 1

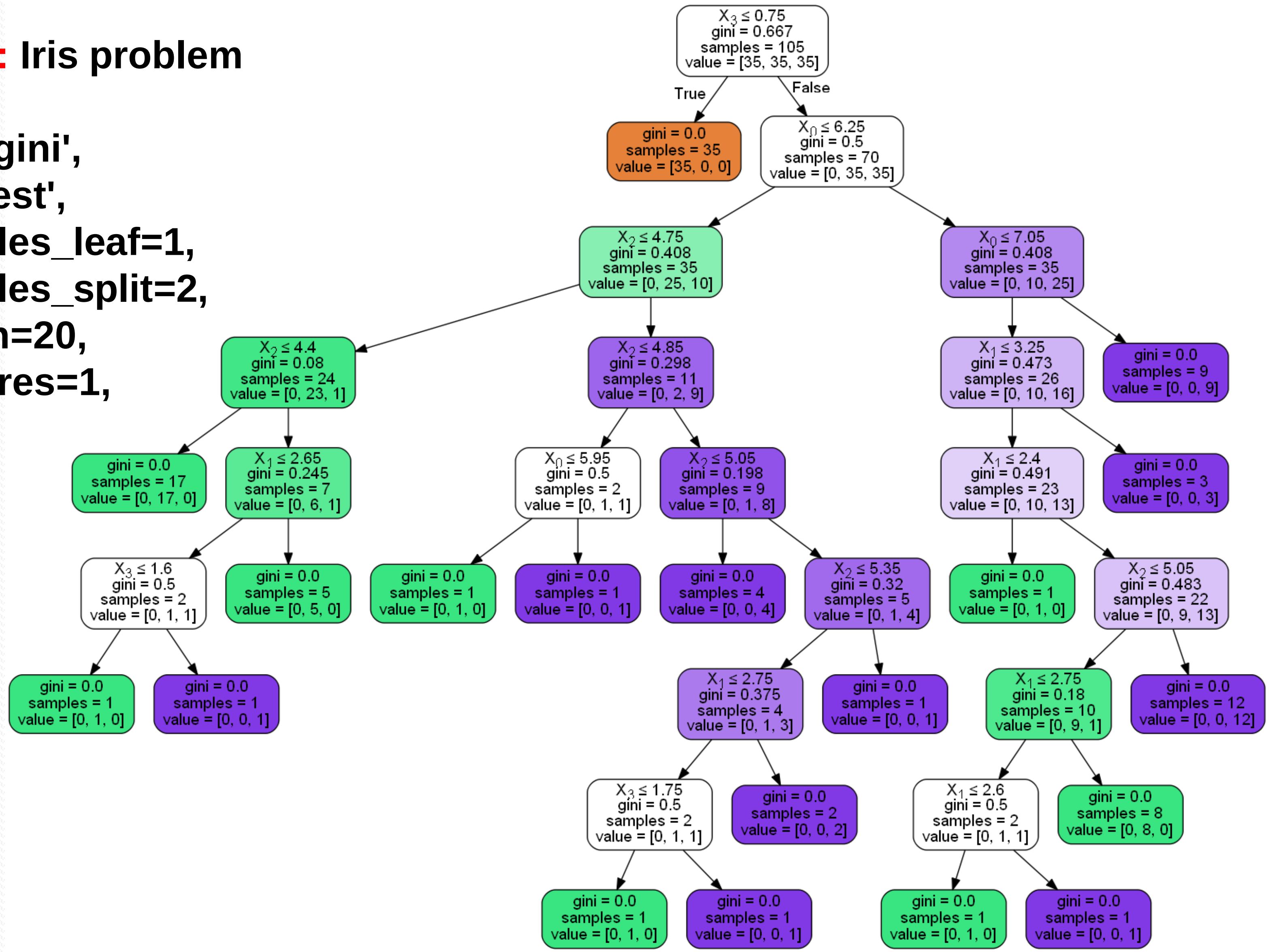


max_features = 4

Decision Tree

Example 1: Iris problem

criterion='gini',
splitter='best',
min_samples_leaf=1,
min_samples_split=2,
max_depth=20,
max_features=1,



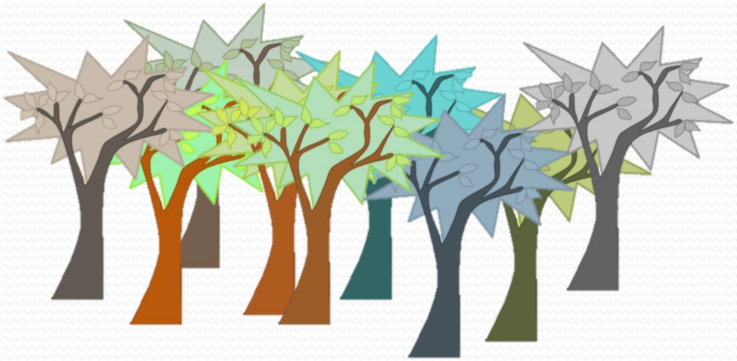
Random Forest Algorithm

The random forest method is another way to elaborate nonlinear problems. Classification and Regression Trees (CART) were first introduced by Leo Breiman (2001) for classification or regression predictive modeling problems.

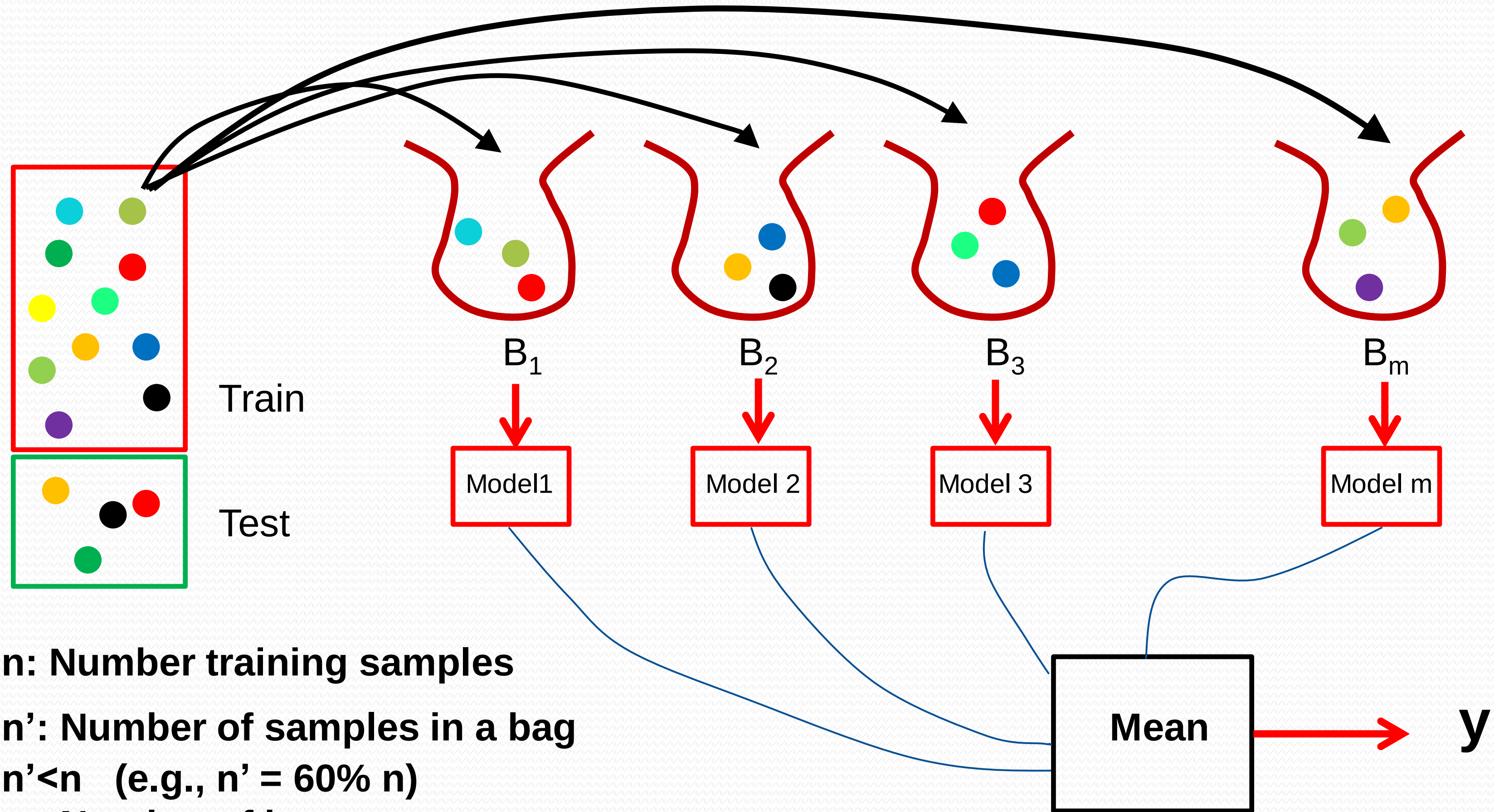
Trees: The trees are grown from different subsets of the training data by a bagging procedure (Breiman, 1996) which ensures the diversity of the trees and minimizes the similarities between them.

The bagging procedure is based on a bootstrap technique. In this method the mean value of a variable is calculated from the average of the mean values of the random subsets of that variable with replacement (i.e., the same value of the variable can be selected few times randomly in a subset). This reduces the correlation between the trees that may cause overfitting. The prediction for unseen data is then based on the average estimations from all regression or classification trees.

It has been shown that the generalization error for the forest converges as the number of the trees in the forest increases which prevents an overfitting problem. On the other hand, a reduction in the number of features in splitting reduces the correlation among trees and increases the model prediction accuracy (Breiman, 1996).

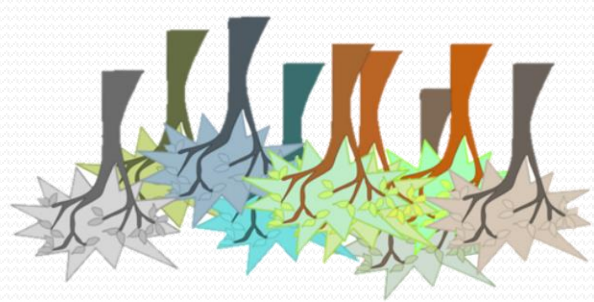


Bagging – Bootstrap Sampling



n: Number training samples

n’: Number of samples in a bag
n’<n   (e.g., n’ = 60% n)
m: Number of bags



Parameters

The random forest method has a number of tuning parameters (e.g., Raschka, 2015) which can be adjusted by a grid search or manually.

n_estimators : integer, optional (default=10)

The number of trees in the forest. Increasing this parameter to a certain level reduces the possibility of overfitting to the cost of computational time. However, a very large number of random trees may not result in a significant gain in the prediction accuracy and the score could even drop.

criterion : string, optional (default=”gini”)

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

max_depth : integer or None, optional (default=None)

If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. The depth of the tree is proportional to the number of splits of the nodes into the child nodes. Deeper trees are more complex and are more likely to overfit the training data.

.

Parameters

min_samples_split: int, float, optional (default=2)
The minimum number of samples required to split at each internal node of a tree (from one to all sample). However, splitting a larger number of samples at each node will cause underfitting and may abruptly cause a decrease in learning. This parameter may be declared as integer (the number of samples required to split at each node) or be set as the fraction of the samples in the range of 0<min_samples_split≤1 (floating).

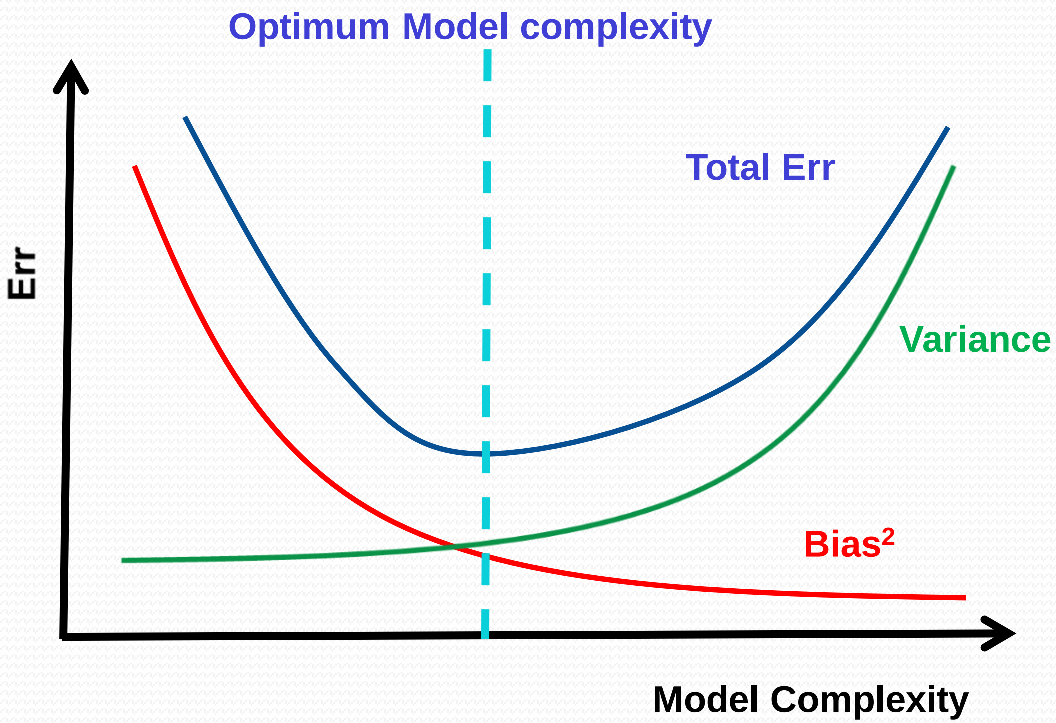
min_samples_leaf : int, float, optional (default=1)
The minimum number of samples at a leaf node to avoid further splitting (integer or floating). Increasing this parameter may cause underfitting.

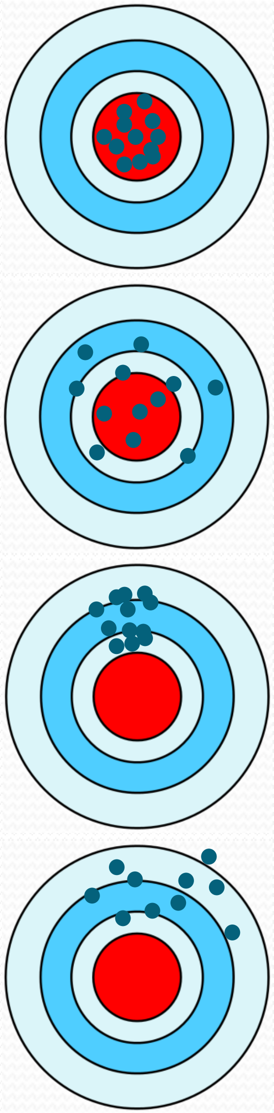
The parameters "max_depth" and "min_samples_leaf" control the level of regularization; that is, decreasing "max_depth" and increasing "min_samples_leaf" will result in a better regularization.

max_features : int, float, string or None, optional (default="auto")
The number of features to consider when looking for the best split.
The smaller values prevents overfitting, but too small values may introduce underfitting.

max_leaf_nodes: The maximum number of leaves in the tree.

Bias-Variance Trade-off





$$Err(x) = \left(\bar{g}(x) - f(x)\right)^2 + \left(g(x) - \bar{g}(x)\right)^2 + \sigma^2$$

Bias Variance Irreducible error

Random Forest Model
Image Recognition – Fashion Data

Fashion problem:

$precision$ **= t_p / (t_p + f_p)**

$recall$ **= t_p / (t_p + f_n)**

t_p: ***true positive***
f_p : ***false positive***
f_n : ***true positive***

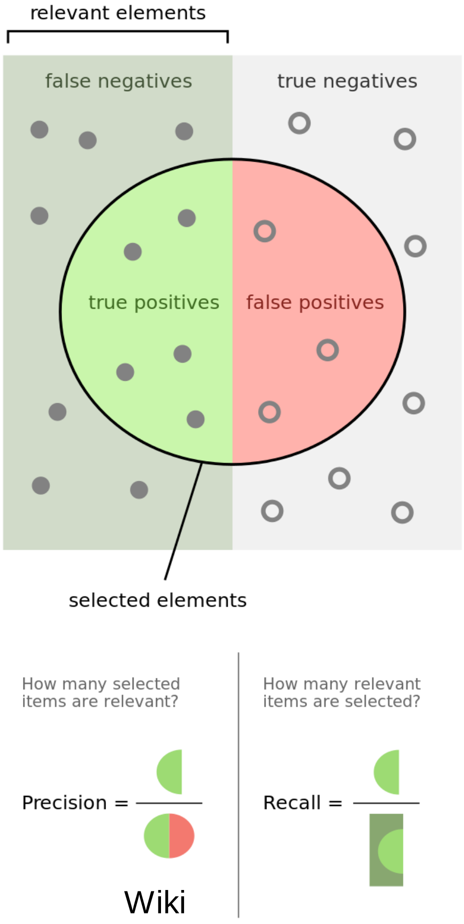
$$f_\beta = (1+\beta^2)\frac{precision \times recall}{\beta^2\ precision + recall}$$

$$f_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

=================================== final test
Test data metrics:

	precision	recall	f1-score	support
class 0	0.64	0.81	0.72	91
class 1	0.93	0.91	0.92	92
class 2	0.52	0.75	0.61	91
class 3	0.79	0.79	0.79	105
class 4	0.75	0.54	0.62	99
class 5	0.74	0.74	0.74	105
class 6	0.37	0.28	0.32	99
class 7	0.73	0.76	0.74	94
class 8	0.91	0.83	0.87	115
class 9	0.87	0.84	0.86	109
avg / total	0.73	0.73	0.72	1000

=================================== final test



The f_β score can be interpreted as a weighted harmonic mean of the precision and recall, where an f_β score reaches its best value at 1 and worst score at 0.

The support is the number of occurrences of each class in y_true.

Random Forest Model
Image Recognition – Fashion Data

```
#=======================================================
#https://github.com/PraveenDubba/Image-Classification-using-Random-Forest/blob/master/Random_Forest_latest.py
import sys
import os
import pandas as pd
import numpy as np
import sklearn
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.grid_search import GridSearchCV
from sklearn.cluster import KMeans
from sklearn.cross_validation import train_test_split
#=======================================================

#======================================================= import data
train_data = pd.read_csv("fashion_train.csv")
final_test_data = pd.read_csv("fashion_test.csv")
#======================================================= import data

#======================================================= split data
print('======================================= split data')
X_train = train_data.iloc[:,0:784]
y_train = train_data.iloc[:,784:]
print('train_data.shape = ', train_data.shape)
print('X_train.shape = ', X_train.shape)
#print('X_train = ', X_train.tail(5))
print('y_train = ', y_train.tail(5))

X_final_test = final_test_data.iloc[:,0:784]
y_final_test = final_test_data.iloc[:,784:]
print('final_test_data.shape = ', final_test_data.shape)
print('X_final_test.shape = ', X_final_test.shape)
#print('X_final_test = ', X_final_test.tail(5))
print('y_final_test = ', y_final_test.tail(5))

# Splitting train data into training and validation datasets
x_train_v, x_test_v, y_train_v, y_test_v = train_test_split(X_train,y_train, test_size = 0.3, random_state = 2)

```

Random Forest Model
Image Recognition – Fashion Data

```
print('x_train_v.shape = ', x_train_v.shape)
print('x_test_v.shape = ', x_test_v.shape)
print('y_train_v.shape = ', y_train_v.shape)
print('y_test_v.shape = ', y_test_v.shape)
y_train_v = np.ravel(y_train_v)
print('y_train_v.shape = ', y_train_v.shape)
y_test_v = np.ravel(y_test_v)
print('y_test_v.shape = ', y_test_v.shape)
print('====================================== split data')
#====================================================== split data

#====================================================== a simple random forest without grid search
# ================= Using Random Forest without hyper paramter tuning and clustering ===================
print('====================================== simple model')
rf = RandomForestClassifier(n_estimators=6)
rf.fit(x_train_v,y_train_v)

# Predictions on training and validation
y_pred_train = rf.predict(x_train_v)
    # predictions for test
y_pred_test = rf.predict(x_test_v)

my_target_names=['class 0', 'class 1', 'class 2','class 3', 'class 4', 'class 5', 'class 6', 'class 7', 'class 8', 'class 9']

print('====================================== metrics 1')
print('====================================== train')
    # training metrics
print("Training metrics:")
print(sklearn.metrics.classification_report(y_true= y_train_v, y_pred= y_pred_train, target_names=my_target_names))
print('====================================== train')

print('====================================== test')
    # test data metrics
print("Test data metrics:")
print(sklearn.metrics.classification_report(y_true= y_test_v, y_pred= y_pred_test, target_names=my_target_names))
print('====================================== test')
```

Random Forest Model
Image Recognition – Fashion Data

```
print('======================================= final test')
# Predictions on testset
y_pred_test = rf.predict(X_final_test)
    # test data metrics
print("Test data metrics:")
print(sklearn.metrics.classification_report(y_true= y_final_test, y_pred= y_pred_test, target_names=my_target_names))
print('======================================= final test')
print('y_pred_test[113] = ', y_pred_test[113]) #check one of them
print('======================================= metrics 1')
print('======================================= simple model')
#========================================================= a simple random forest without grid search
#sys.exit()
#========================================================= random forest with grid search
print('======================================= model with grid search')
print("================random forest with gird search started================")
#Using Grid Search for hyper parameter tuning
#========================================================= grid search
print("================grid search================")
rf = RandomForestClassifier()
param_grid = dict(n_estimators=[10,20],
                  min_samples_leaf=[2,3])
grid = GridSearchCV(rf, param_grid, cv=5, scoring=None)
model = grid.fit(x_train_v,y_train_v)
print('======================================= grid best estimator')
print ('grid.best_estimator_ = ', grid.best_estimator_ )
print('======================================= grid best estimator')
#========================================================= grid results
print("================grid scores================")
print("grid.cv_results_ {}".format(grid.grid_scores_))
print("================grid scores================")
print("================grid best score================")
print("-grid.best_score_ (r2 / variance) = {} ".format(-grid.best_score_))
print("================grid best score================")
print("================grid best parameters================")
print("grid.best_params_ = {} ".format(grid.best_params_))
print("================grid best parameters================")
print("================grid best estimator================")
print("grid.best_estimator_ = {} ".format(grid.best_estimator_))
print("================grid best estimator================")
#========================================================= grid results
```

Random Forest Model
Image Recognition – Fashion Data

```

#========================================================= predict
y_pred_train = model.predict(x_train_v)
    # predictions for test
y_pred_test = model.predict(x_test_v)
print('======================================== metrics 2')
    # training metrics
print("Training metrics:")
print(sklearn.metrics.classification_report(y_true= y_train_v, y_pred= y_pred_train))

    # test data metrics
print("Test data metrics:")
print(sklearn.metrics.classification_report(y_true= y_test_v, y_pred= y_pred_test))


# Predictions on testset
y_pred_test = model.predict(X_final_test)
    # test data metrics
print("Test data metrics:")
print(sklearn.metrics.classification_report(y_true= y_final_test, y_pred= y_pred_test))
print('======================================== metrics 2')
#========================================================= predict
print("================grid search================")
#========================================================= grid search
print('======================================== model with grid search')
#========================================================= a simple random forest without grid searc
sys.exit('Program finished')


```

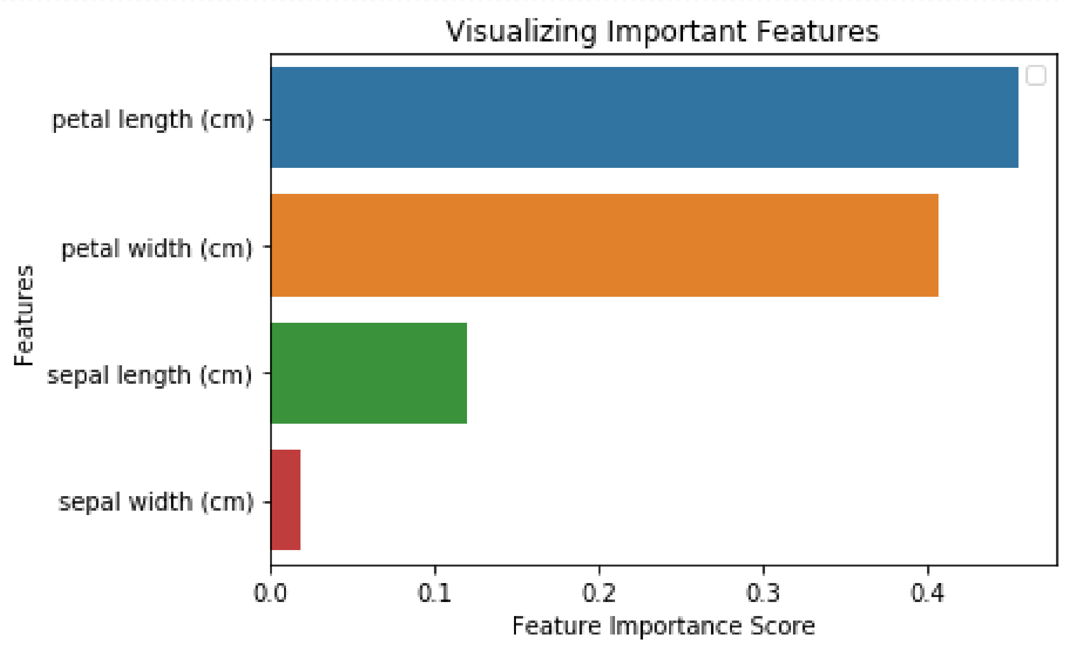
Feature Importance

```
tree.fit(X_train, y_train)
print('tree.feature_importances_ = ', tree.feature_importances_)
```

tree.feature_importances_ = [0.00425693 0.   0.06941788 0.92632519]

Or

```
clf.fit(X_train,y_train)
import pandas as pd
feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascending=False)
```



Feature Importance

```
#===============================================
from sklearn import datasets  # scikit-learn dataset library
import sys
#=============================================== load iris data
iris = datasets.load_iris()
print('iris.target_names = ', iris.target_names)  # labels (setosa, versicolor,virginica)
print('iris.feature_names = ', iris.feature_names)  # feature names
print('iris.data[0:5] = ', iris.data[0:5])     # print data (features) from row 0 to row 4
print('iris.target[0:5] = ', iris.target[0:5])     # print target (class lables) from row 0 to row 4
#print('iris.target = ', iris.target)
#print('iris.data = ', iris.data)
print('iris.target.shape = ', iris.target.shape)
print('iris.data.shape = ', iris.data.shape)
#=============================================== load iris data

#===============================================  create a DataFrame of iris dataset
import pandas as pd
data=pd.DataFrame({
    'sepal length':iris.data[:,0],
    'sepal width':iris.data[:,1],
    'petal length':iris.data[:,2],
    'petal width':iris.data[:,3],
    'species':iris.target
})
data.head()
print('data.head(3) = ', data.head(3))
print('data.tail(3) = ', data.tail(3))
#===============================================  create a DataFrame of iris dataset

#=============================================== split data into train and test
from sklearn.model_selection import train_test_split
X=data[['sepal length', 'sepal width', 'petal length', 'petal width']]  # Features
y=data['species']  # Labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 19) # 70% training and 30% test

print('=======================================')
#print('y = ', y)
#print('X = ', X)
print('y_train.shape = ', y_train.shape)
print('X_train.shape = ', X_train.shape)
```

Feature Importance

```
print('y_test.shape = ', y_test.shape)
print('X_test.shape = ', X_test.shape)
print('========================================')
#=============================================== split data into train and test

#=============================================== random forest model (training)
from sklearn.ensemble import RandomForestClassifier
#clf=RandomForestClassifier(n_estimators=100)     # n_estimators: the number of trees in the forest

clf=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,            # set random_state = i and get the ave.
            warm_start=False)
clf.fit(X_train,y_train)    # train
#=============================================== random forest model (training)

#=============================================== testing
y_pred=clf.predict(X_test)
y_pred2=clf.predict(X_train)
print('y_pred = ', y_pred)
from sklearn import metrics #scikit-learn metrics module for accuracy calculation
print("Accuracy (test):",metrics.accuracy_score(y_test, y_pred))
print("Accuracy (train):",metrics.accuracy_score(y_train, y_pred2))
#=============================================== testing

#=============================================== predict a single item (can be used for plotting decision boundaries)
single_pred = clf.predict([[3, 5, 4, 2]])
print('single_pred = ', single_pred)
#=============================================== predict a single item (can be used for plotting decision boundaries)
#sys.exit()
#==============================================feature importance
import pandas as pd
feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascending=False)
print('feature_imp = ', feature_imp)
#==============================================feature importance
```

Feature Importance

```

#===============================================feature importance (visualization)
import matplotlib.pyplot as plt
import seaborn as sns   # statistical data visualization
#%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
#===============================================feature importance (visualization)

#=============================================== Generating the Model on Selected Features
#from sklearn.cross_validation import train_test_split
#from sklearn.ensemble import RandomForestClassifier
#Create a Gaussian Classifierclf=RandomForestClassifier(n_estimators=100)
#from sklearn import metrics   #scikit-learn metrics module for accuracy calculation

# Split dataset into features and labels
X=data[['petal length', 'petal width','sepal length']]  # Removed feature "sepal length"
y=data['species']
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.70, random_state=5) # 70% training and 30% test

clf.fit(X_train,y_train)         # train
y_pred=clf.predict(X_test)       # test
y_pred2=clf.predict(X_train)
print("Accuracy (test with reduced features):",metrics.accuracy_score(y_test, y_pred))
print("Accuracy (train with reduced features):",metrics.accuracy_score(y_train, y_pred2))

#=============================================== Generating the Model on Selected Features

```