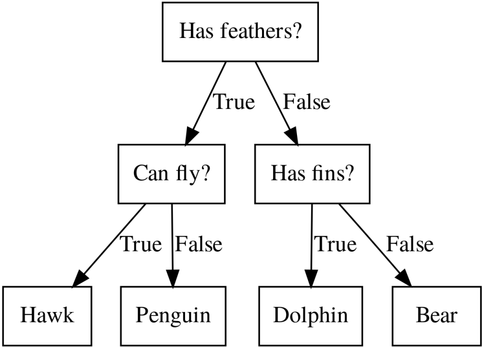
Module-4.5-Decision Trees and Ensemble Models

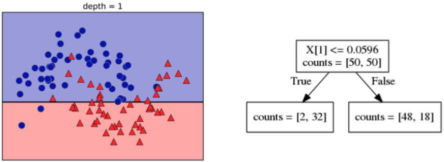
Presented by Yasin Ceran

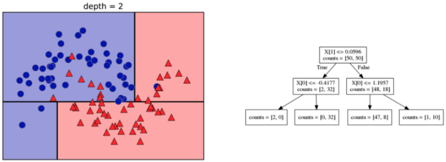
Table of Contents

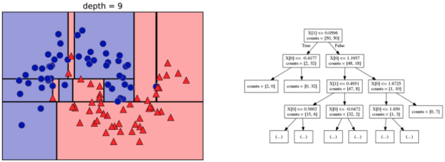
Idea: Series of Binary of Binary Questions



Building Trees







Continuous features:

- “questions” are thresholds on single features.
- Minimize impurity

Criteria for Classification

- Gini Index:

$$H_{\text{gini}}(X_m) = \sum_{k \in \mathcal{Y}} p_{mk}(1 - p_{mk})$$

- Cross-Entropy:

$$H_{\text{CE}}(X_m) = -\sum_{k \in \mathcal{Y}} p_{mk} \log(p_{mk})$$

- X_m observations in node m
- $\mathcal{Y}$ classes
- $p_{m\cdot}$ distribution over classes in node m

An Example

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Entropy

Entropy **H**(**S**) is a measure of the amount of uncertainty in the data set **S** (i.e., entropy characterizes the data set **S**.

$$H(S) = \sum_{c \in C} -p(c) log_2 p(c)$$

where

- S - The current data set for which the entropy is being calculated
- C - Set of classes in S, for example, $C = \{yes, no\}$
- $p(c)$ - The proportion of the number of elements in class c to the number of elements in set S

When $H(S) = 0$, the set S is perfectly classified.

Information Gain

Information gain **IG**(**A**) is the measure of the difference in entropy from before to after the set **S** is split on an attribute **A**. In other words, how much uncertainty in **S** was reduced after splitting **S** on attribute **A**.

$$IG(A,S) = \mathbf{H}(S) - \sum_{t \in T} p(t)H(t)$$

where

- $H(S)$ - Entropy of set **S**
- T - The subsets created from splitting set **S** by attribute **A** such that $\mathbf{S} = \bigcup_{t \in T} t$
- $p(t)$ - The proportion of the number of elements in t to the number of elements in set S
- $H(t)$ - Entropy of subset t

Compute the Entropy for the Weather Data Set

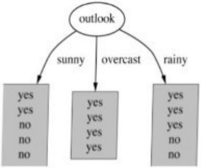
$$H(S) = \sum_{c \in C} -p(c) log_2 p(c)$$

$$C = \{yes, no\}$$

Out of 14 instances, 9 are clasified as 'yes' and 5 as 'no'

- $p(yes) = -(9/14) * log_2(9/14)$
- $p(no) = -(5/14) * log_2(5/14)$
- $H(S) = p(yes) + p(no) = 0.94$

Calculate Entropy and IG for All Features



E (Outlook=sunny) = $-\frac{2}{5}\log\left(\frac{2}{5}\right)-\frac{3}{5}\log\left(\frac{3}{5}\right)=0.971$

E (Outlook=overcast) = $-1\log(1)-0\log(0)=0$

E (Outlook=rainy) = $-\frac{3}{5}\log\left(\frac{3}{5}\right)-\frac{2}{5}\log\left(\frac{2}{5}\right)=0.971$

H(S,Outlook)

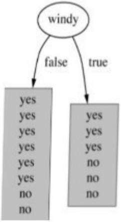
Average Entropy information for Outlook

I (Outlook) = $\frac{5}{14}*0.971+\frac{4}{14}*0+\frac{5}{14}*0.971=0.693$

$$\sum_{t\in T} p(t)H(t)$$

Gain (Outlook) = E(S) – I (outlook) = 0.94-.693 = 0.247

$$IG(A,S)=H(S)-\sum_{t\in T} p(t)H(t)$$



E (Windy=false) = $-\frac{6}{8}\log\left(\frac{6}{8}\right)-\frac{2}{8}\log\left(\frac{2}{8}\right)=0.811$

E (Windy=true) = $-\frac{3}{6}\log\left(\frac{3}{6}\right)-\frac{3}{6}\log\left(\frac{3}{6}\right)=1$

Average entropy information for Windy

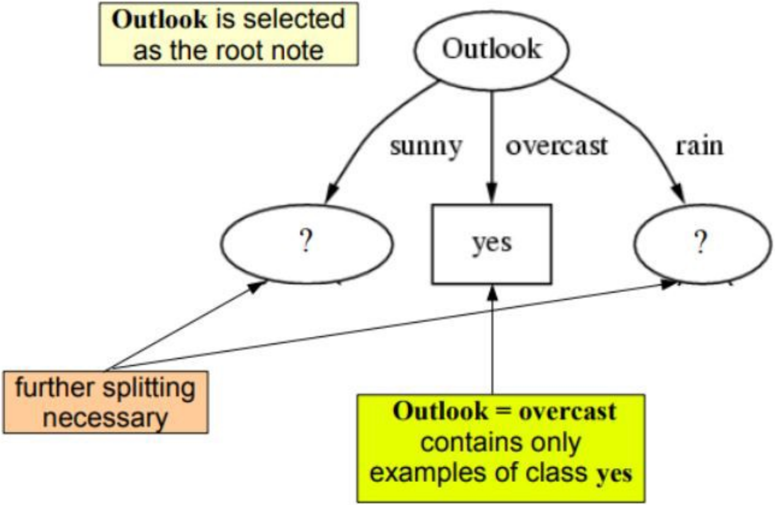
I (Windy) = $\frac{8}{14}*0.811+\frac{6}{14}*1=0.892$

Gain (Windy) = E(S) – I (Windy) = 0.94-0.892=0.048

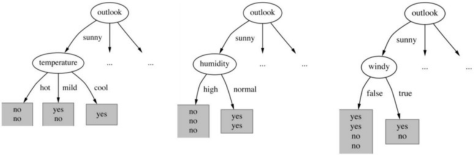
Pick the Highest Gain Attribute

Outlook		Temperature	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	$\boxed{0.247}$	Gain: 0.940-0.911	0.029
Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048

Which One is the Root?

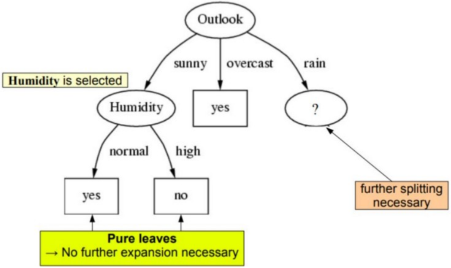


Create the Tree



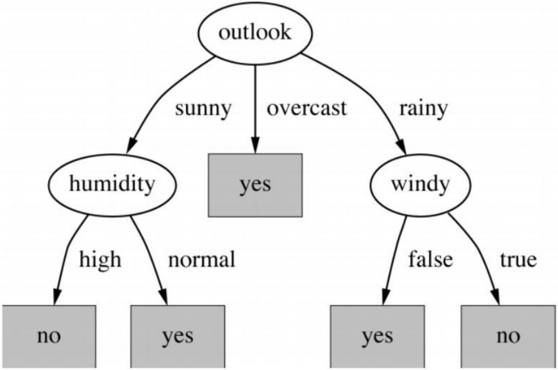
$$
\left.\begin{aligned}
\text{Gain}(\textit{Temperature}) &= 0.571 \text{ bits}\\
\text{Gain}(\textit{Humidity}) &= 0.971 \text{ bits}\\
\text{Gain}(\textit{Windy}) &= 0.020 \text{ bits}
\end{aligned}\right\}
$$

Humidity is selected

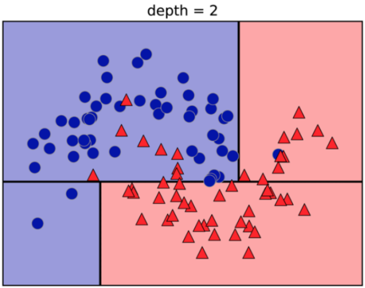


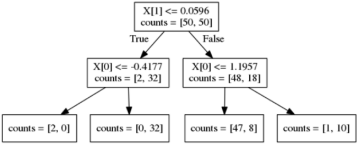
Final Decision

Final decision tree



Predictions with Decision Trees





Regression Trees

- Prediction:

$$\bar{y}_m = \frac{1}{N_m} \sum_{i \in N_m} y_i$$

- Mean Squared Error:

$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - \bar{y}_m)^2$$

- Mean Absolute Error:

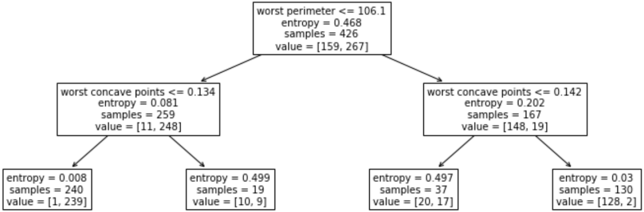
$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} |y_i - \bar{y}_m|$$

A Simple Example with Tree

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data,cancer.target,
                                                    stratify=cancer.target,
                                                    random_state=0)


from sklearn.tree import DecisionTreeClassifier, export_graphviz
tree = DecisionTreeClassifier(max_depth=2)
tree.fit(X_train, y_train)
```

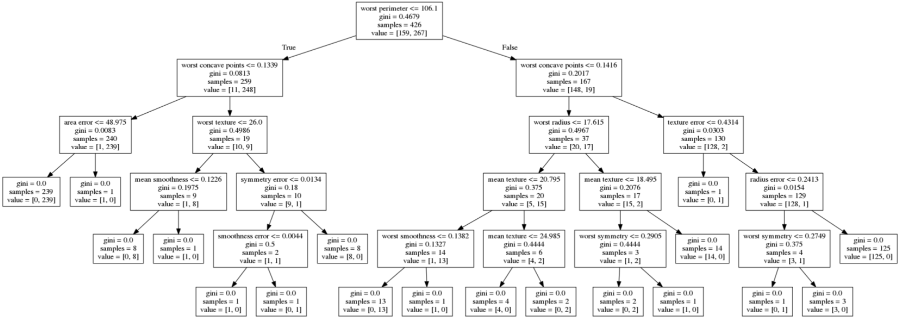
```
from sklearn.tree import plot_tree
tree_dot = plot_tree(tree, feature_names=cancer.feature_names)
```



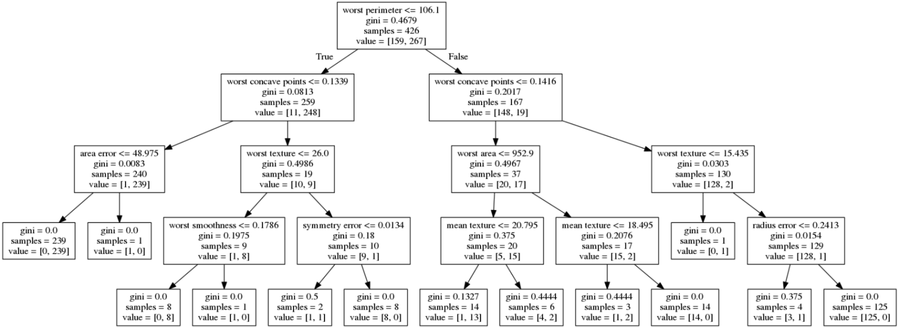
Parameter Tuning

- Limit tree size (pick one, maybe two):
  - max_depth
  - max_leaf_nodes
  - min_samples_split
  - min_impurity_decrease

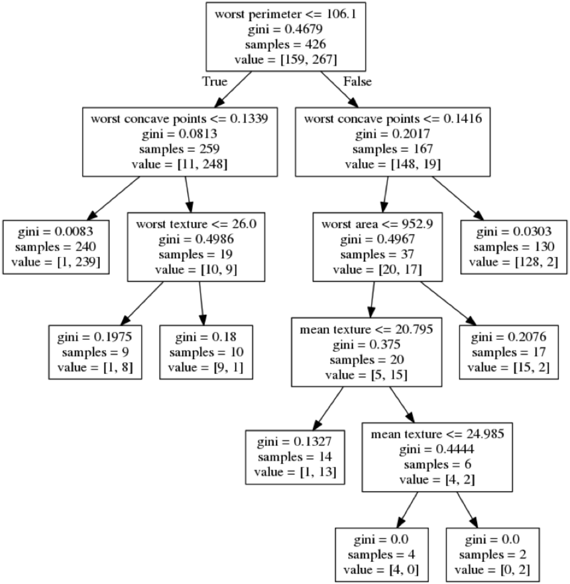
No Pruning


worst perimeter <= 106.1
gini = 0.4679
samples = 426
value = [159, 267]
True
False
worst concave points <= 0.1339
gini = 0.0813
samples = 259
value = [11, 248]
worst concave points <= 0.1416
gini = 0.2017
samples = 167
value = [148, 19]
area error <= 48.975
gini = 0.0083
samples = 240
value = [1, 239]
worst texture <= 26.0
gini = 0.4986
samples = 19
value = [10, 9]
worst radius <= 17.615
gini = 0.4967
samples = 37
value = [20, 17]
texture error <= 0.4314
gini = 0.0303
samples = 130
value = [128, 2]
gini = 0.0
samples = 239
value = [0, 239]
gini = 0.0
samples = 1
value = [1, 0]
mean smoothness <= 0.1226
gini = 0.1975
samples = 9
value = [1, 8]
symmetry error <= 0.0134
gini = 0.18
samples = 10
value = [9, 1]
mean texture <= 20.795
gini = 0.375
samples = 20
value = [5, 15]
mean texture <= 18.495
gini = 0.2076
samples = 17
value = [15, 2]
gini = 0.0
samples = 1
value = [0, 1]
radius error <= 0.2413
gini = 0.0154
samples = 129
value = [128, 1]
gini = 0.0
samples = 8
value = [0, 8]
gini = 0.0
samples = 1
value = [1, 0]
smoothness error <= 0.0044
gini = 0.5
samples = 2
value = [1, 1]
gini = 0.0
samples = 8
value = [8, 0]
worst smoothness <= 0.1382
gini = 0.1327
samples = 14
value = [1, 13]
mean texture <= 24.985
gini = 0.4444
samples = 6
value = [4, 2]
worst symmetry <= 0.2905
gini = 0.4444
samples = 3
value = [1, 2]
gini = 0.0
samples = 14
value = [14, 0]
worst symmetry <= 0.2749
gini = 0.375
samples = 4
value = [3, 1]
gini = 0.0
samples = 125
value = [125, 0]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 13
value = [0, 13]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 4
value = [4, 0]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 3
value = [3, 0]

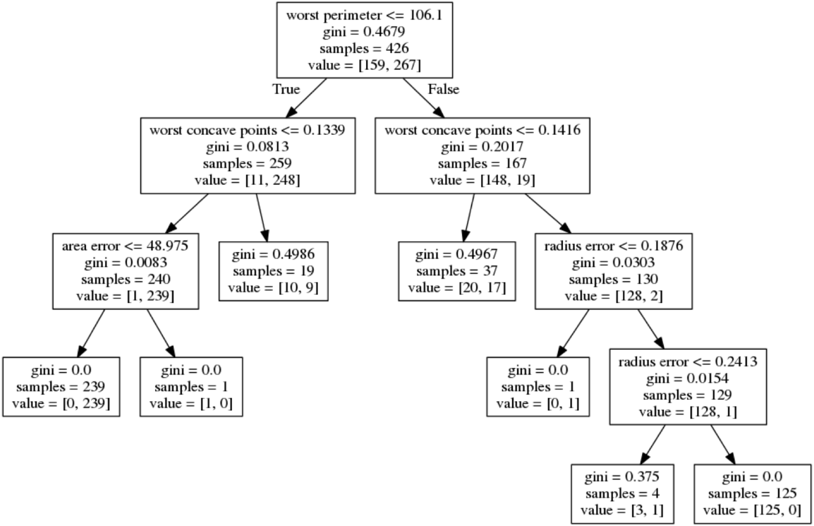
max_depth = 4


worst perimeter <= 106.1
gini = 0.4679
samples = 426
value = [159, 267]
True
False
worst concave points <= 0.1339
gini = 0.0813
samples = 259
value = [11, 248]
worst concave points <= 0.1416
gini = 0.2017
samples = 167
value = [148, 19]
area error <= 48.975
gini = 0.0083
samples = 240
value = [1, 239]
worst texture <= 26.0
gini = 0.4986
samples = 19
value = [10, 9]
worst area <= 952.9
gini = 0.4967
samples = 37
value = [20, 17]
worst texture <= 15.435
gini = 0.0303
samples = 130
value = [128, 2]
gini = 0.0
samples = 239
value = [0, 239]
gini = 0.0
samples = 1
value = [1, 0]
worst smoothness <= 0.1786
gini = 0.1975
samples = 9
value = [1, 8]
symmetry error <= 0.0134
gini = 0.18
samples = 10
value = [9, 1]
mean texture <= 20.795
gini = 0.375
samples = 20
value = [5, 15]
mean texture <= 18.495
gini = 0.2076
samples = 17
value = [15, 2]
gini = 0.0
samples = 1
value = [0, 1]
radius error <= 0.2413
gini = 0.0154
samples = 129
value = [128, 1]
gini = 0.0
samples = 8
value = [0, 8]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.5
samples = 2
value = [1, 1]
gini = 0.0
samples = 8
value = [8, 0]
gini = 0.1327
samples = 14
value = [1, 13]
gini = 0.4444
samples = 6
value = [4, 2]
gini = 0.4444
samples = 3
value = [1, 2]
gini = 0.0
samples = 14
value = [14, 0]
gini = 0.375
samples = 4
value = [3, 1]
gini = 0.0
samples = 125
value = [125, 0]

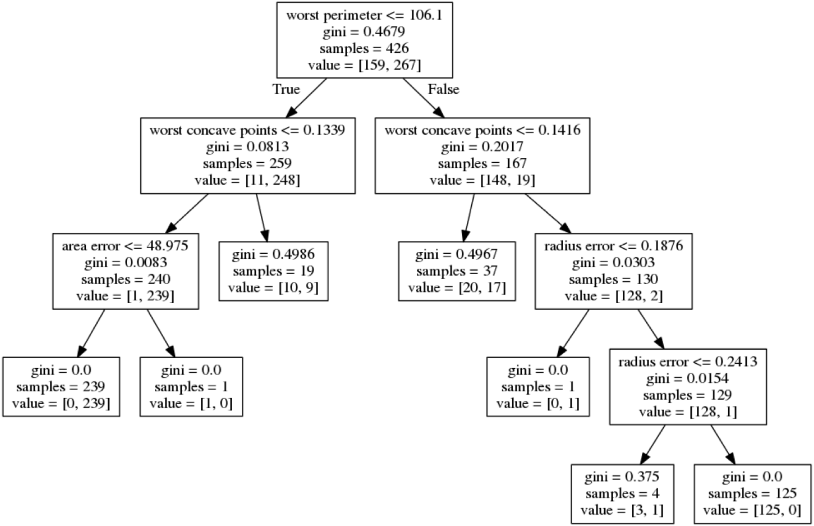
max_leaf_nodes = 8



min_samples_split = 50

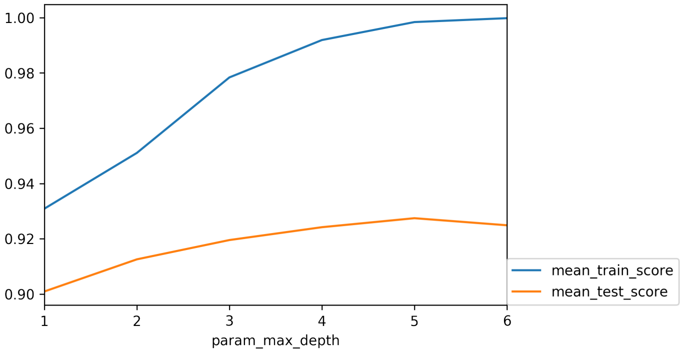


min_samples_split = 50



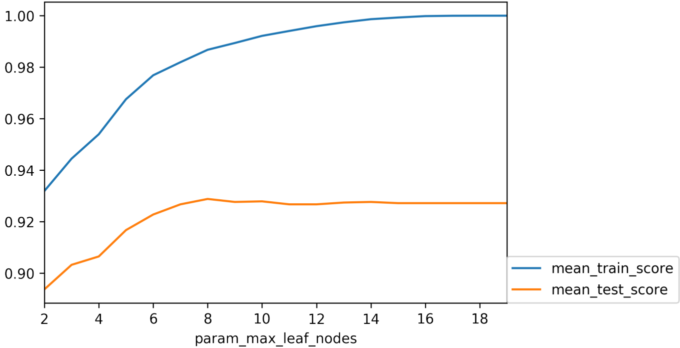
Decision Tree Accuracy for Different max_depth

```
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth':range(1, 7)}
grid = GridSearchCV(DecisionTreeClassifier(random_state=0),param_grid=param_grid,
                    cv=10)
grid.fit(X_train, y_train)
```



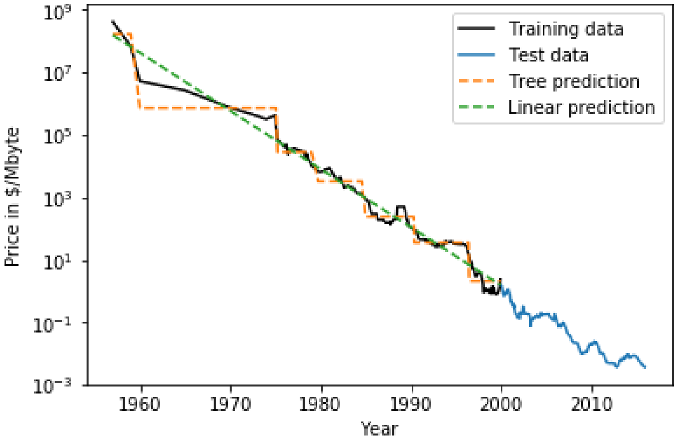
Decision Tree Accuracy for Different max_leaf_nodes

```python
from sklearn.model_selection import GridSearchCV
param_grid = {'max_leaf_nodes':range(2, 20)}
grid = GridSearchCV(DecisionTreeClassifier(random_state=0),
                    param_grid=param_grid, cv=10)
grid.fit(X_train, y_train)
```

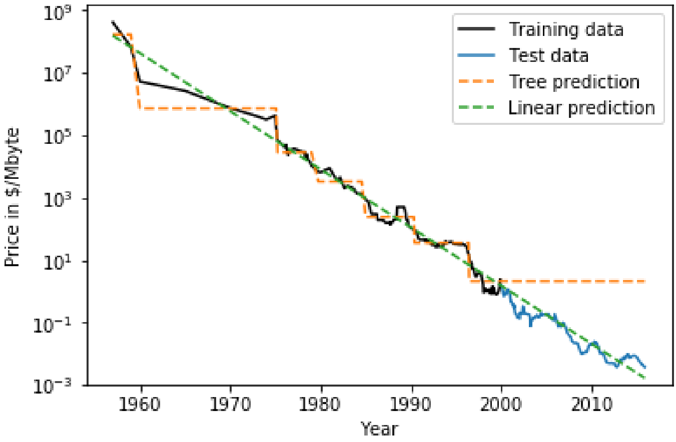


Relation to Nearest Neighbors

- Predict average of neighbors – either by k, by epsilon ball or by leaf.
- Trees are much faster to predict.
- Both can't extrapolate



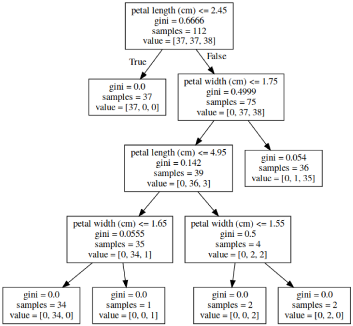
Extrapolation

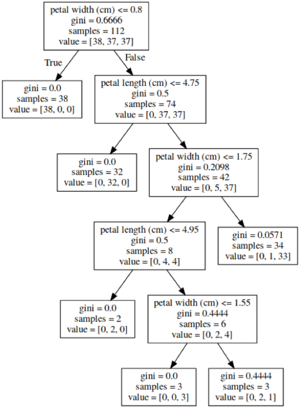


Instability

```
X_train, X_test, y_train, y_test =
train_test_split(iris.data, iris.target,
        stratify=iris.target, random_state=0)
tree = DecisionTreeClassifier(max_leaf_nodes=6)
tree.fit(X_train, y_train)
```

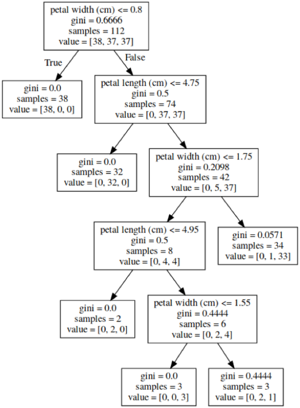
```
X_train, X_test, y_train, y_test =
train_test_split(iris.data, iris.target,
        stratify=iris.target, random_state=1)
tree = DecisionTreeClassifier(max_leaf_nodes=6)
tree.fit(X_train, y_train)
```



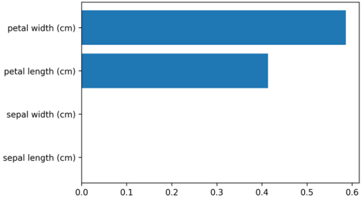


Feature Importance

```
X_train, X_test, y_train, y_test =
        train_test_split(
    iris.data, iris.target, stratify=iris.target
    , random_state=1)
tree = DecisionTreeClassifier(max_leaf_nodes=6)
tree.fit(X_train,y_train)
```




```
tree.feature_importances_
array([0.0, 0.0, 0.414, 0.586])
```



Predicting Probabilities

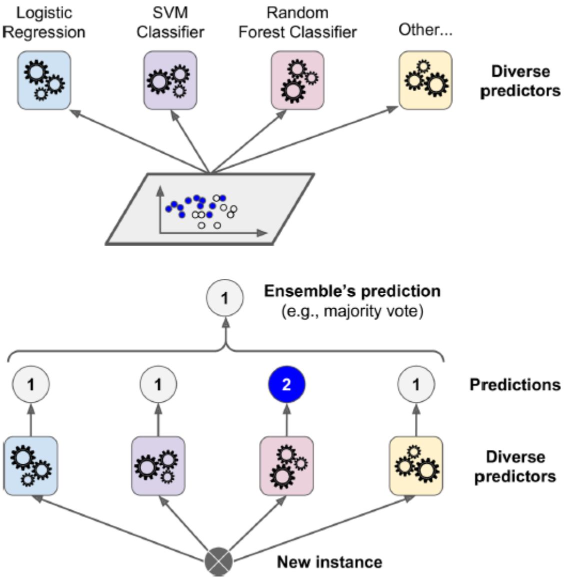
- Fraction of class in leaf.
- Without pruning: Always 100% certain!
- Even with pruning might be too certain.

Table of Contents

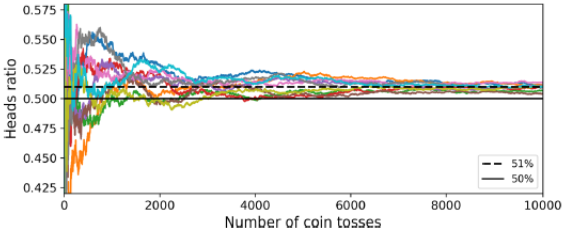
Poor Man's Ensembles

- Build different models
- Average the result
- More models are better – if they are not correlated.
- Also works with neural networks
- You can average any models as long as they provide calibrated ("good") probabilities

Voting Classifiers



Voting Classifier and Law of Large Numbers

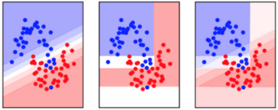


- A slightly biased coin with 51% chance of coming heads
- Toss it 1,000 times, it is 75% more likely to have heads as majority
- Toss it 10,000 times, it is 97% more likely to have heads as majority
- Keep in mind that each toss is independent from the others

Voting Classifier

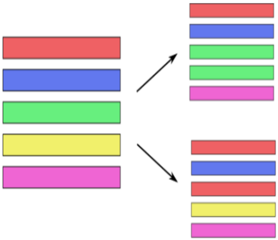
```
voting = VotingClassifier(
    [('logreg', LogisticRegression(C=100)),
     ('tree', DecisionTreeClassifier(max_depth=3, random_state=0))],
    voting='soft')
voting.fit(X_train, y_train)
lr, tree = voting.estimators_
voting.score(X_test, y_test), lr.score(X_test, y_test), tree.score(X_test, y_test)
```

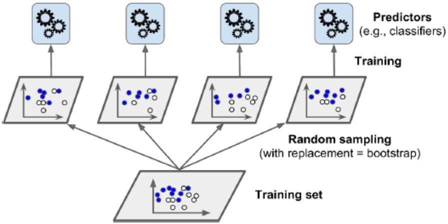
```
0.88 0.84 0.80
```



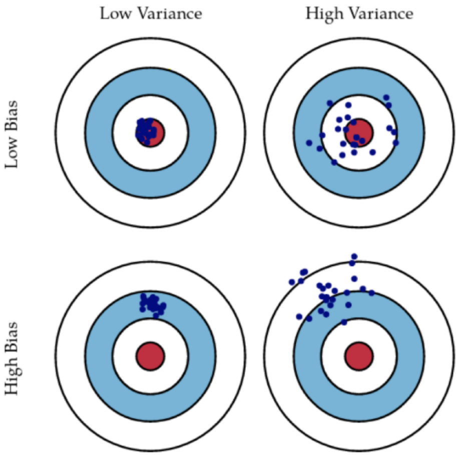
Bagging (Bootstrap AGGregation)

- Generic way to build "slightly different" models
- Use the same training algorithm for every predictor and train them on different random subsets of the training set

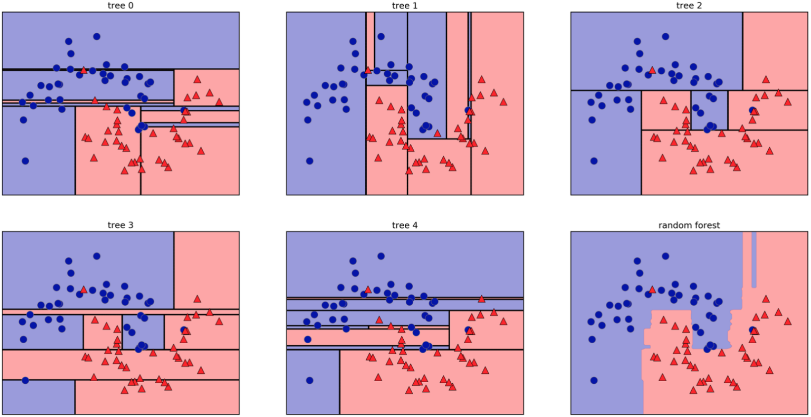




Bias and Variance

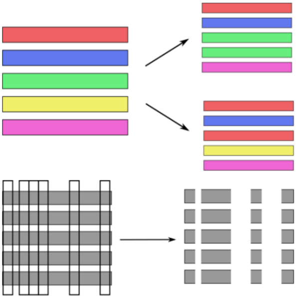


Random Forest



Randomize in Two Ways

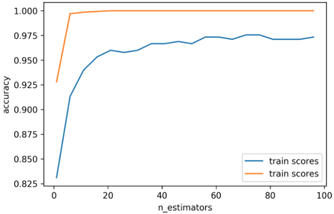
- For each tree: Pick bootstrap sample of data
- For each split: Pick random sample of features
- More trees are always better



A Simple Example for Random Forest

```
train_scores = []
test_scores = []

rf = RandomForestClassifier(warm_start=True)
estimator_range = range(1, 100, 5)
for n_estimators in estimator_range:
    rf.n_estimators = n_estimators
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))
```

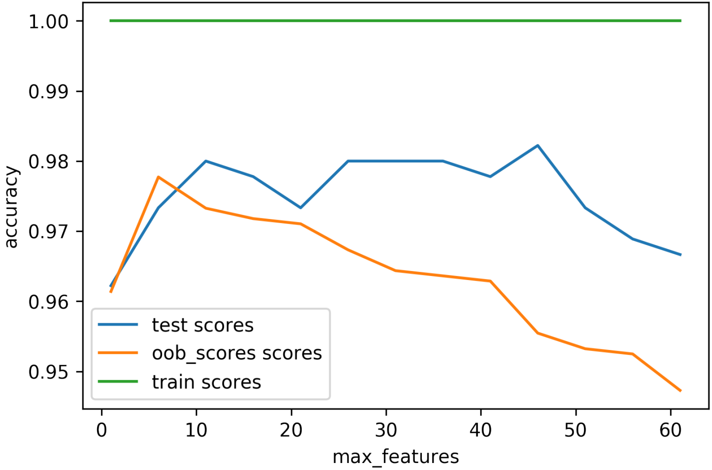


Out of Bag Estimates

```python
train_scores = []
test_scores = []
oob_scores = []
feature_range = range(1, 64, 5)
for max_features in feature_range:
    rf = RandomForestClassifier(max_features=max_features, oob_score=True,
                                n_estimators=200, random_state=0)
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))
    oob_scores.append(rf.oob_score_)
```

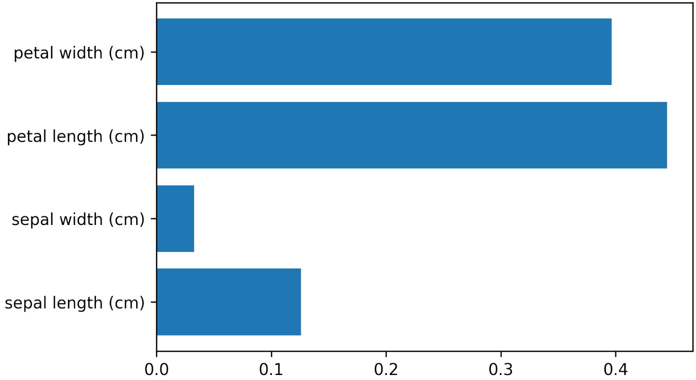
- Each tree only uses 66% of data
- Can evaluate it on the rest!
- Make predictions for out-of-bag, average, score.
- Each prediction is an average over different subset of trees

Out of Bag Example



Variable Importance

```
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, stratify=iris.target, random_state=1)
rf = RandomForestClassifier().fit(X_train, y_train)
rf.feature_importances_
plt.barh(range(4), rf.feature_importances_)
plt.yticks(range(4), iris.feature_names);
```



Summary

- Decision trees are robust to data scaling, but are very unstable models
- To cope with the high variance in the decision trees, we use Random Forest models