동국대학교
dongguk university

Decision Tree, K-Nearest Neighbor, SVM

Machine Learning

(PM chap 3)

Decision Trees

- The Model
  - Representing the model as a **tree** - not a parameterized function
  - x : (x_1, x_2, …, x_n), y = C_1 or C_2 or … C_m
    - Internal nodes: a feature (x_i)
    - Edges: different feature values
    - Leaves: class label y (C_k)

x		y
size	weight	class
large	high	Y
small	low	N
large	high	Y
large	low	N
small	high	N
…	…	…

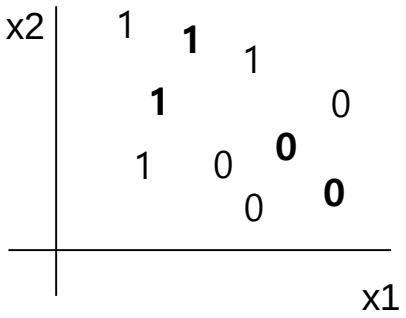
input x (size, weight) → **rules** → output y (cancer)

If (weight=low) then class=N

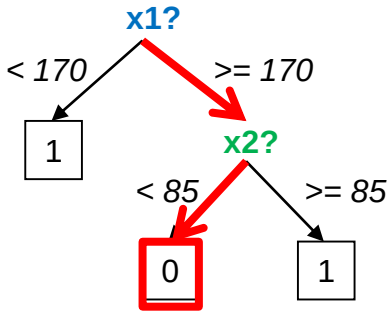
Decision Trees

- Example
  - Training data:

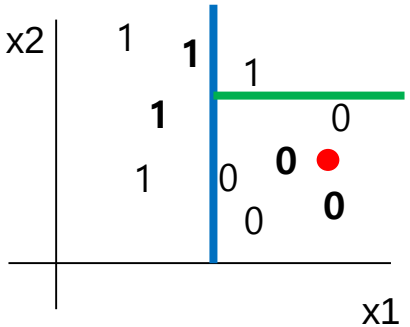
x		y
185	65	0
162	80	1
175	70	0
...	...	...



  - The model:



  - Prediction:



  - Learning: Build tree by recursively partition the training data

Learning Decision Trees

- Learning
  1. At start, all the training examples are at the root
  2. A feature is selected based on a statistical measure (e.g., information gain)
  3. Examples are partitioned recursively based on selected features
- Conditions for stop partitioning
  - All samples belong to the same class
  - No samples left → majority class
  - No attributes left → majority sample

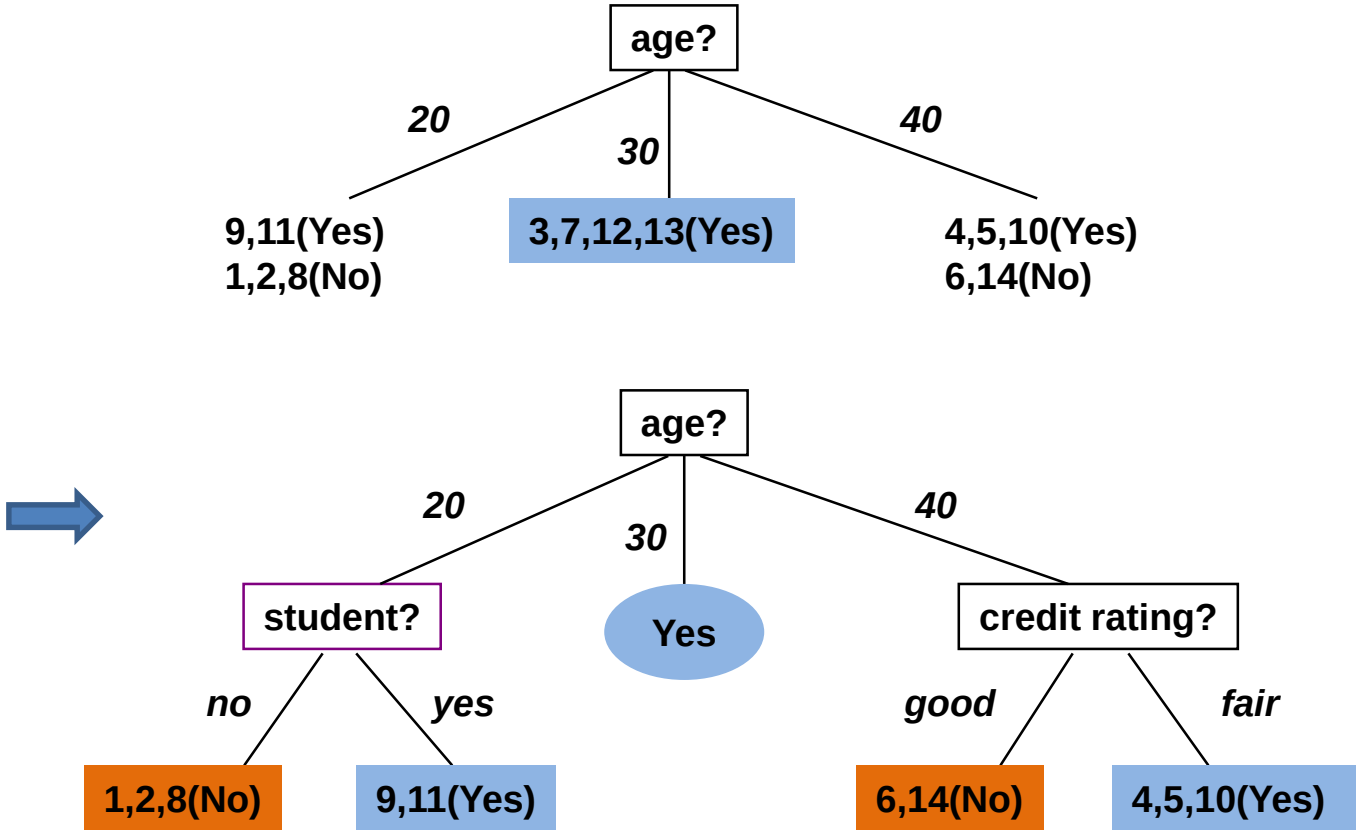
Learning Decision Trees

- Learning example

No.	age	income	student	credit	BUY?
1	20	high	no	fair	N
2	20	high	no	good	N
3	30	high	no	fair	Y
4	40	med	no	fair	Y
5	40	low	yes	fair	Y
6	40	low	yes	good	N
7	30	low	yes	good	Y
8	20	med	no	fair	N
9	20	low	yes	fair	Y
10	40	med	yes	fair	Y
11	20	med	yes	good	Y
12	30	med	no	good	Y
13	30	high	yes	fair	Y
14	40	med	no	good	N

Learning Decision Trees

- Learning example



Entropy

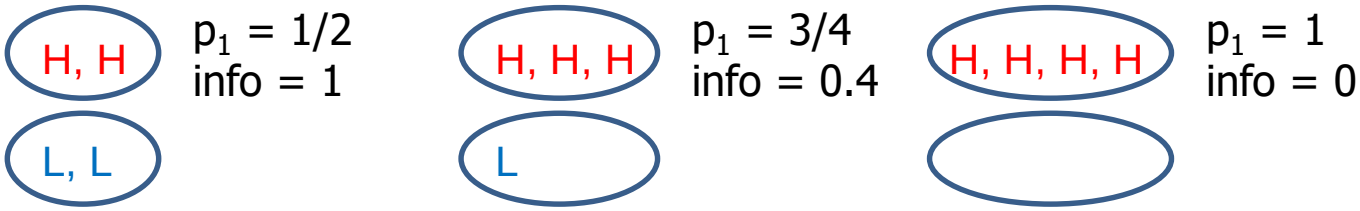
- Amount of Information
  - There are $C_1, C_2, \ldots, C_n$ classes, each of which has $s_1, s_2, \ldots, s_n$ samples

$$p_1 = \frac{s_1}{s} \quad p_2 = \frac{s_2}{s}$$

  - Information required to classify a sample to C_i
    - High prob. → low info.
    - Prob. = 1 → zero info.

$$Info. = \log_2 \frac{1}{p_i} = \boxed{-\log_2 p_i} = -\log_2 \frac{s_i}{s}$$

- Example



Entropy

- Entropy of a set S
  - Measure of uncertainty, or *mixedupness*
  - Expected information to classify a sample in S

$$I(S) = I(s_1, s_2, \ldots, s_m) = \sum_{i=1}^{m} p_i(-\log_2 p_i) = -\sum_{i=1}^{m} \frac{s_i}{s} \log_2 \frac{s_i}{s}$$

- Example

H, H, L, L

$$I = \left(-\frac{1}{2} \cdot \log\frac{1}{2}\right) + \left(-\frac{1}{2} \cdot \log\frac{1}{2}\right) = 1.0$$

H, H, H, L

$$I = \left(-\frac{3}{4} \cdot \log\frac{3}{4}\right) + \left(-\frac{1}{4} \cdot \log\frac{1}{4}\right) = 0.81$$

H, H, H, H

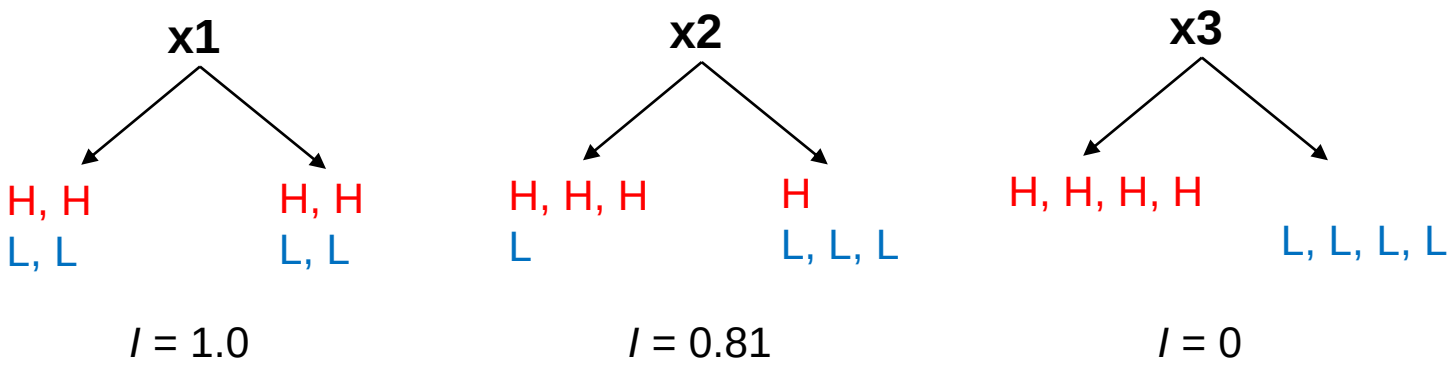
$$I = \left(-\frac{4}{4} \cdot \log\frac{4}{4}\right) + \left(-\frac{0}{4} \cdot \log\frac{0}{4}\right) = 0.0$$

Entropy

- Example

H, H, H, H
L, L, L, L

$$I = -\frac{1}{2}log_2\frac{1}{2} - \frac{1}{2}log_2\frac{1}{2} = 1.0$$



I = 1.0 *I* = 0.81 *I* = 0

➡ x1 < x2 < x3 reduces more entropy

➡ x1 < x2 < x3 has more information for H / L decision

Gini

- Gini Index
  - A measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution
    - Summing the probability p_i of an item i being chosen * the probability of a mistake

$$I(S) = \sum_{i=1}^{m} p_i (1 - p_i) = \sum_{i=1}^{m} (p_i - p_i^2) = 1 - \sum_{i=1}^{m} p_i^2$$

- Example

H, H, L, L

$$I_G = 1 - \left(\frac{1}{2}^2 + \frac{1}{2}^2\right) = 0.5$$

H, H, H, L

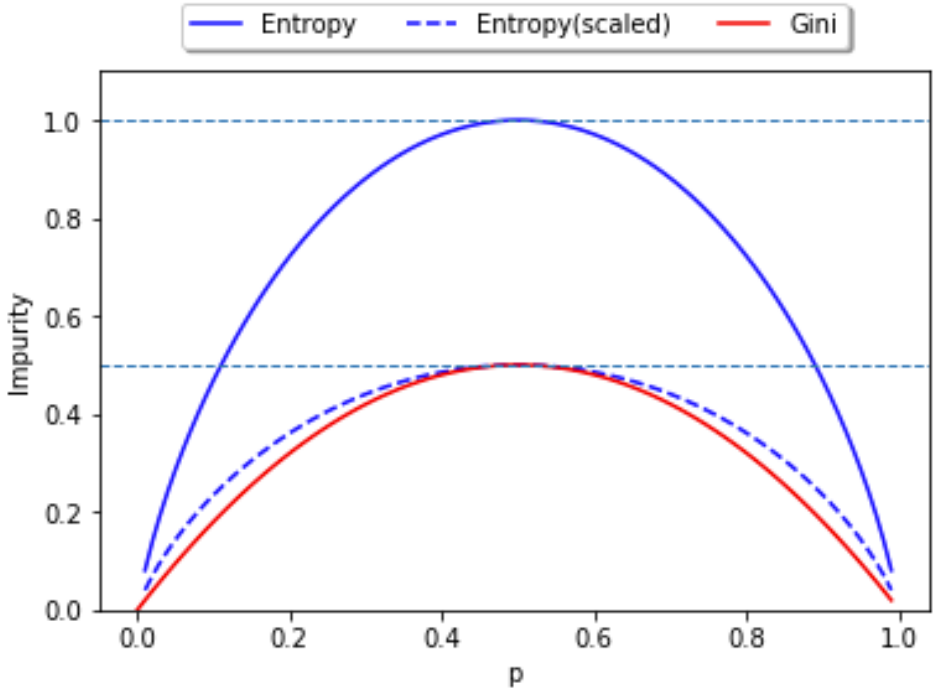
$$I_G = 1 - \left(\frac{3}{4}^2 + \frac{1}{4}^2\right) = 0.375$$

H, H, H, H

$$I_G = 1 - \left(\frac{4}{4}^2 + \frac{0}{4}^2\right) = 0.0$$

Entropy and Gini Impurity

- Entropy and Gini graph



Selecting Features

- Maximizing Information Gain (IG)
  - Objective function

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$

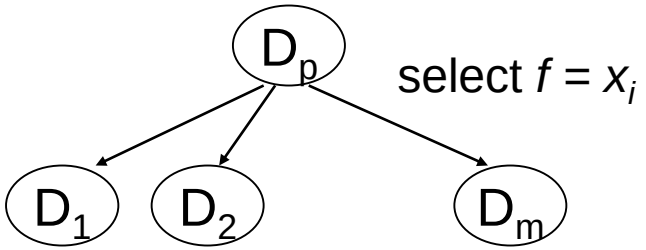
f : features for classification

D_p: Parent node

D_j : j[th] child node

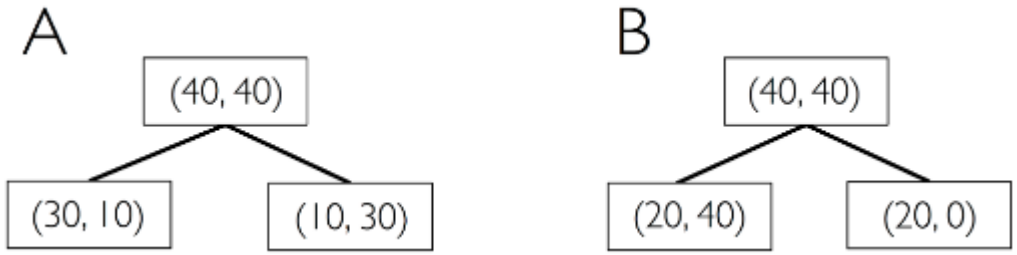
N_p: Number of samples in parent node

N_j : Number of samples in j[th] child node



Example - Entropy

- Maximizing information gain



$$I(D_P) = -\frac{1}{2}log_2\frac{1}{2} - \frac{1}{2}log_2\frac{1}{2} = 1$$

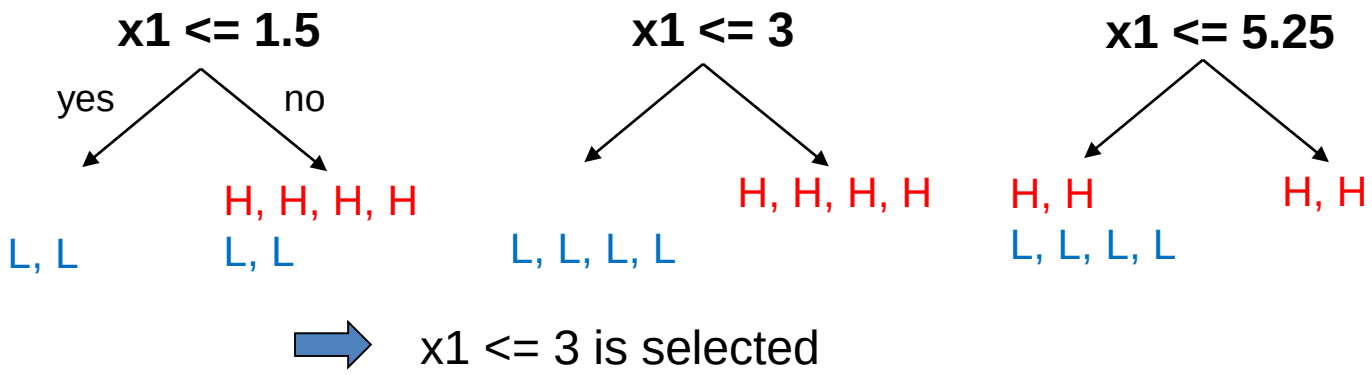
$$I(D_{left}) = -\frac{3}{4}log_2\frac{3}{4} - \frac{1}{4}log_2\frac{1}{4} = 0.81 \qquad I(D_{left}) = -\frac{2}{6}log_2\frac{2}{6} - \frac{4}{6}log_2\frac{4}{6} = 0.92$$

$$I(D_{right}) = -\frac{1}{4}log_2\frac{1}{4} - \frac{3}{4}log_2\frac{3}{4} = 0.81 \qquad I(D_{right}) = -\frac{2}{2}log_2\frac{2}{2} - \frac{0}{2}log_2\frac{0}{2} = 0$$

$$IG(D_P, A) = 1 - \frac{4}{8}\cdot 0.81 - \frac{4}{8}\cdot 0.8 = 0.19 \qquad IG(D_P, B) = 1 - \frac{6}{8}\cdot 0.92 - \frac{2}{8}\cdot 0 = 0.31$$

Continuous Feature Values

- Find the binary split points
  - Sort the values of x in increasing order
  - The midpoints between adjacent values are possible *split points*
    - Example : [1, 2, 4, 6.5] → [1.5, 3, 5.25]
- Select best split point → binarize the feature
  - For each possible split points, compute the information gain
  - The point with the *maximum information gain* is selected as the split-point for x

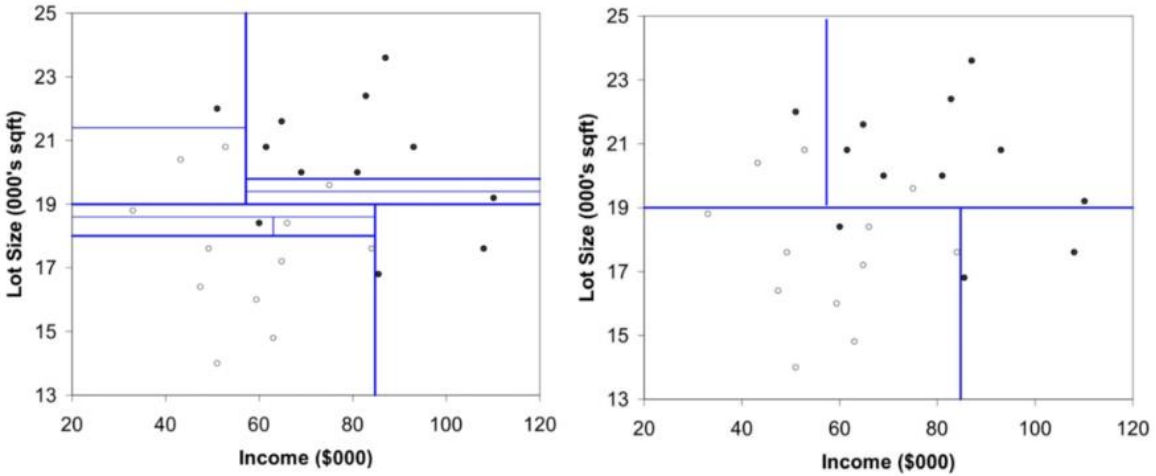


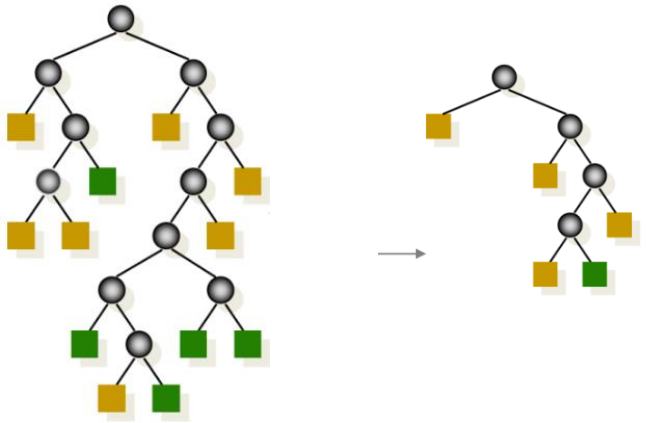
→ x1 <= 3 is selected

Pruning

- The generated tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
- Prepruning
  - Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
- Postpruning
  - Remove branches from a "fully grown" tree
  - If pruning a node lead to a smaller error rate (with test set), prune it

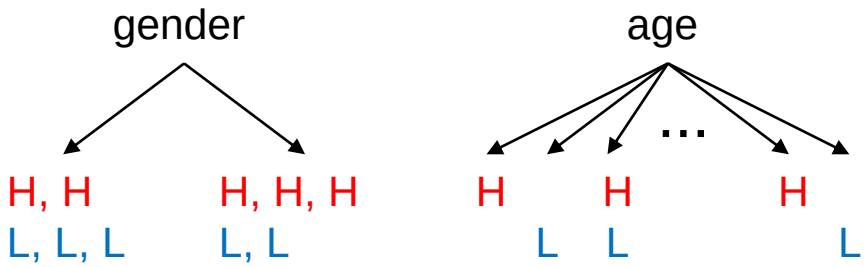
Pruning


Lot Size (000's sqft)
Income ($000)
25
23
21
19
17
15
13
20
40
60
80
100
120



Advantages

- Advantage
  - The model is interpretable (understand why the prediction is made)
    - Ex> x is classified as 'yes' because (Age > 40 and Credit = fair)
- Disadvantage
  - It is unstable
    - a small change in the data can → a large change in the structure of the tree
  - Data including categorical variables with different number of levels
    - information gain in decision trees is biased in favor of attributes with more levels
    - Example



Decision Tree Learning with Scikit-learn

- Iris dataset

```
iris = datasets.load_iris()

X = iris.data[:, [2, 3]]
y = iris.target
```

- DT learning on Iris dataset

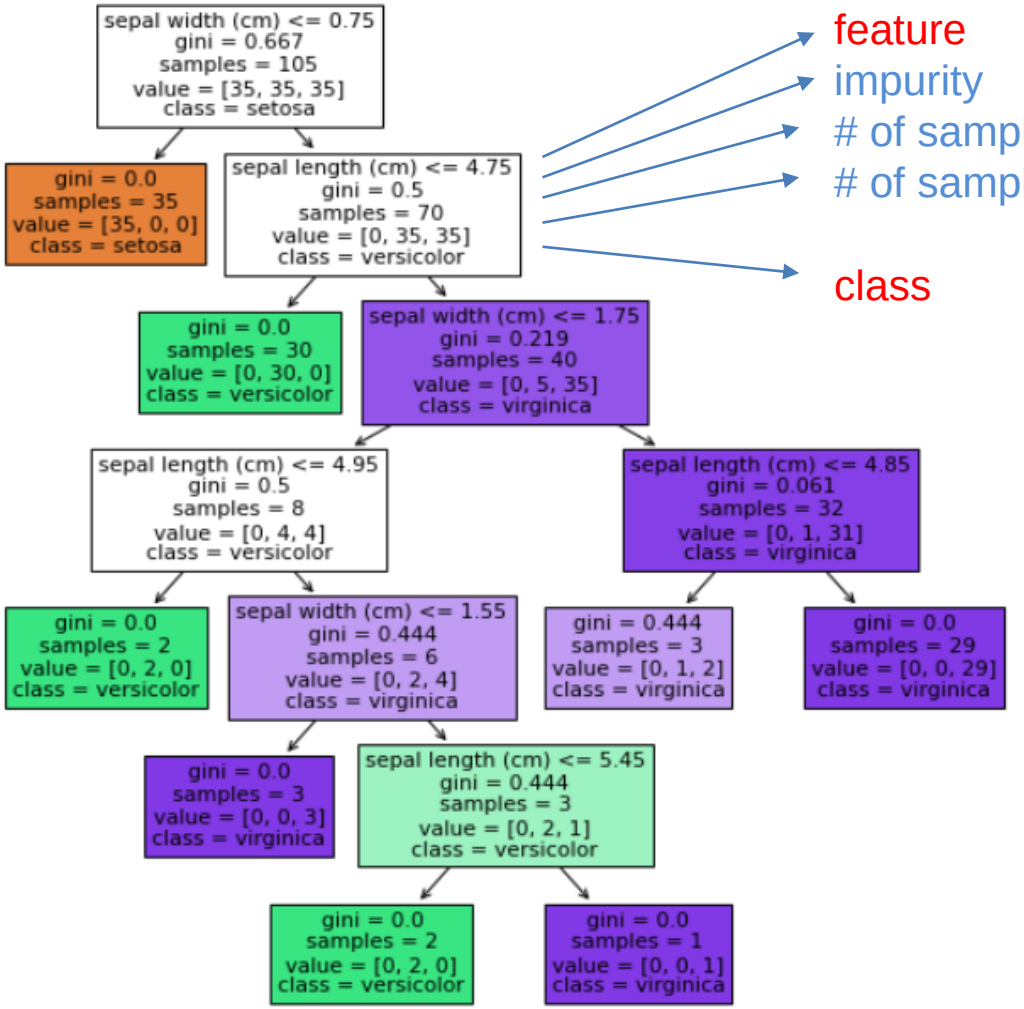
```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion='gini', random_state=1)
tree.fit(X_train, y_train)
```

- The model

```
plot_tree(tree, feature_names=iris.feature_names, class_names=iris.target_names, filled=True, fontsize=11)
plt.show()
```

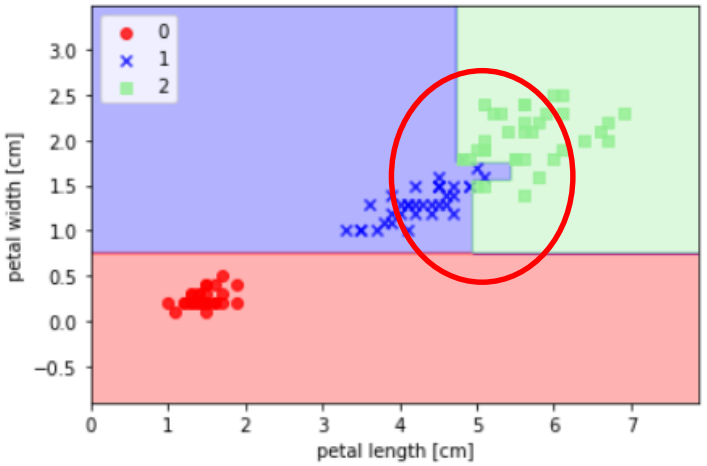
Decision Tree Learning with Scikit-learn



Decision Tree Learning with Scikit-learn

- Decision boundary

```
plot_decision_regions(X_train, y_train, classifier=tree)
```



- Model accuracy

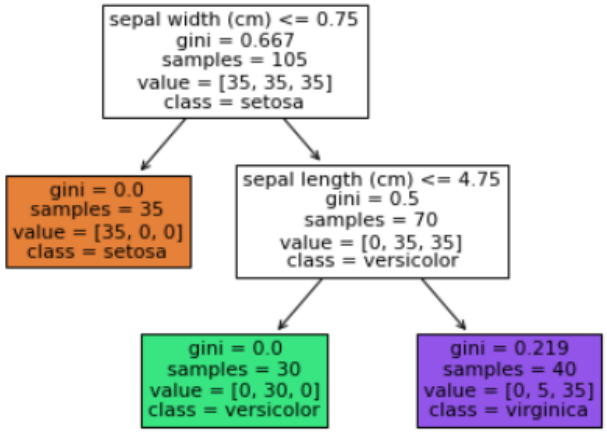
```
print("Train Accuracy : ", tree.score(X_train, y_train))
print("Test Accuracy : ", tree.score(X_test, y_test)
```

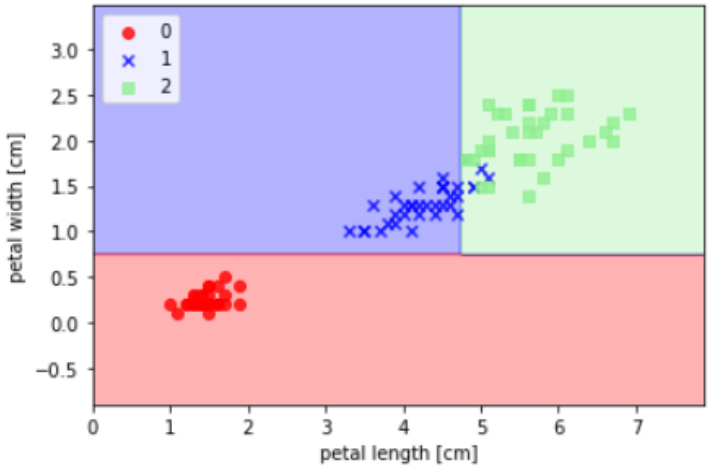
```
Train Accuracy :  0.9904761904761905
Test Accuracy :  0.9777777777777777
```

Decision Tree Learning with Scikit-learn

- DT on Iris dataset – reduced depth

```
tree = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=1)
tree.fit(X_train, y_train)
```



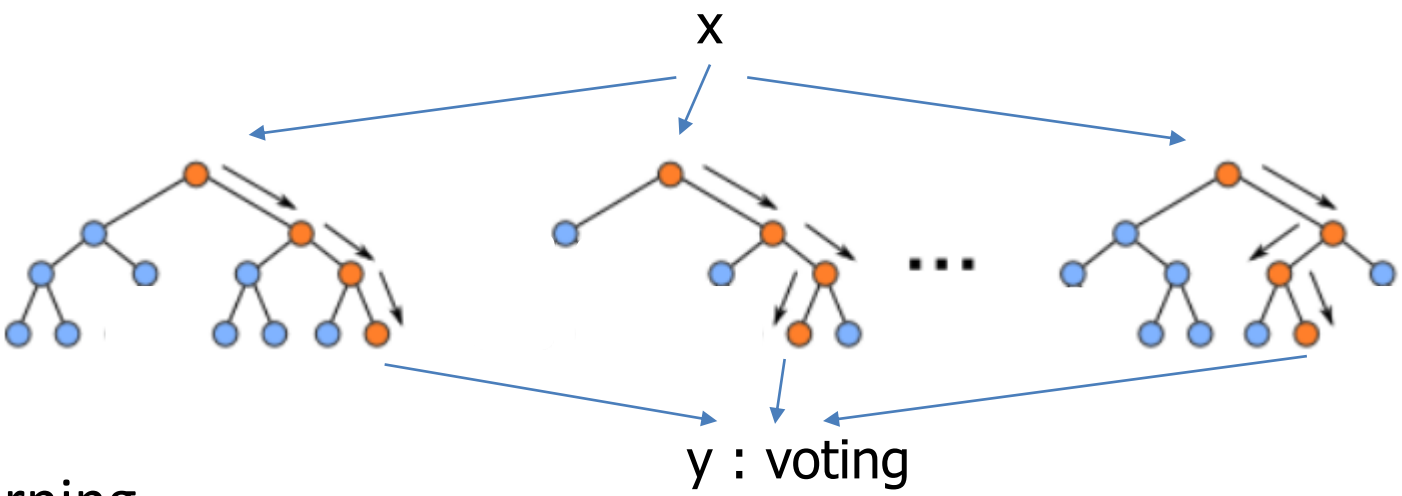



```
print("Train Accuracy : ", tree.score(X_train, y_train))
print("Test Accuracy : ", tree.score(X_test, y_test))
```

```
Train Accuracy :  0.9523809523809523
Test Accuracy :  0.9555555555555556
```

Random Forest

- The model
  - A set of *K* decision trees → y(class label) is determined by majority voting



- Learning
  - Each tree is built from *N* sample data randomly drawn with replacement
  - Each tree is built using *D* features randomly selected from *n* features ($D = \sqrt{n}$)

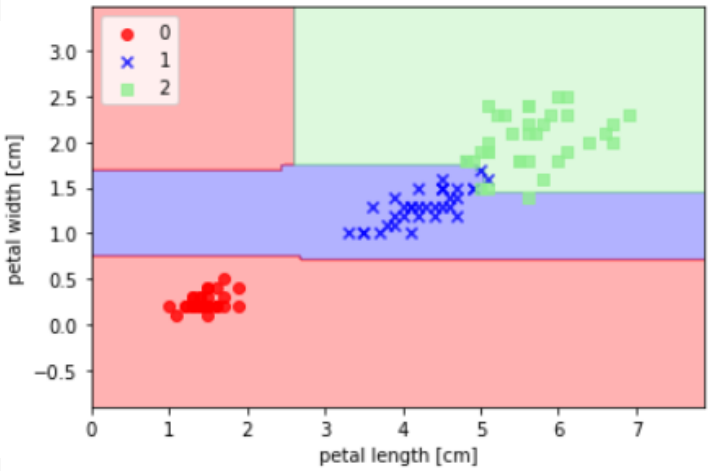
⇒ *better generalization (reduce overfitting)*

Random Forest with Scikit-learn

- DT on Iris dataset
  - No. of trees = 100, depth = 2
  - No. of features = sqrt(n)

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(criterion='gini', n_estimators=100, max_depth=2,
                                max_features='sqrt', random_state=1)
forest.fit(X_train, y_train)
```



- Model accuracy

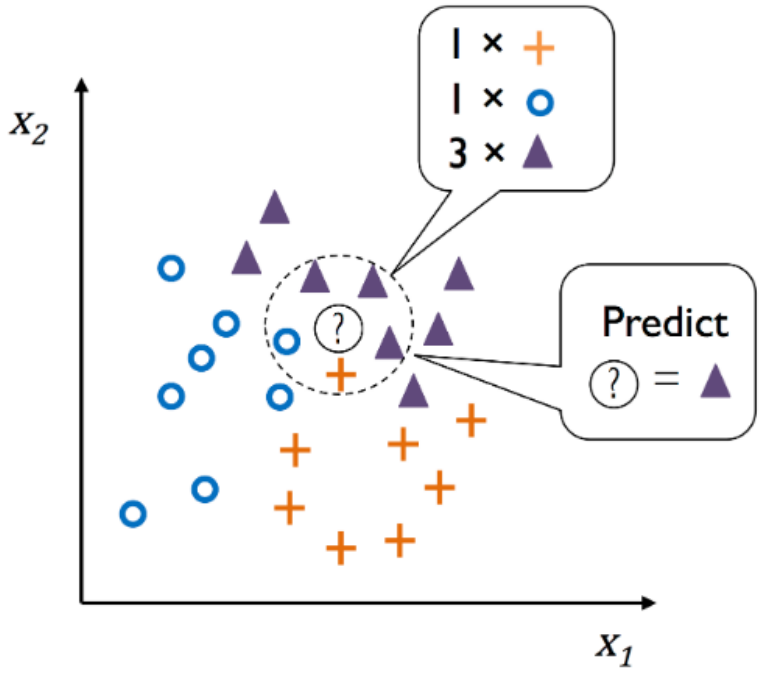
```
print("Train Accuracy : ", forest.score(X_train, y_train))
print("Test Accuracy : ", forest.score(X_test, y_test))
```

```
Train Accuracy :  0.9523809523809523
Test Accuracy :  0.9777777777777777
```

K-Nearest Neighbors

- Non-parametric method
  - The K-Nearest Neighbors algorithm(K-NN) is non-parametric method used for classification and regression
  - No explicit model - *lazy learning*
- Method
  1. Choose K and distance metric
  2. Find K nearest neighbors of x

     from the training data $x^{(i)}$
  3. Assign label y by majority vote



K-Nearest Neighbors

- Minkowski distance
  - Distance between **x** = (x_1, x_2, … , x_n) and **y** = (y_1, y_2, … , y_n) is

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

- P = 1 : Manhattan distance

$$d(\mathbf{x}, \mathbf{y}) = |x_1 - y_1| + |x_2 - y_2| + \cdots$$

- P = 2 : Euclidean distance

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots}$$

K-Nearest Neighbors

- For discrete-valued target y
  - The k-NN returns the most common value among the k training examples nearest to **x**
    - Ex> y is Yes or No : 9-NN, 5 Yes, 4 No → Yes
- For continuous-valued target y
  - Calculate the mean values of the k nearest neighbors
- Distance-weighted method
  - Weight k neighbors according to their distance to $\mathbf{x}^{(i)}$, d(**x**, $\mathbf{x}^{(i)}$)
    - Larger weight to closer neighbor
  - Calculate the weighted average of $y^{(i)}$

$$w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}^{(i)})} \quad W = \sum w_i \qquad\qquad \hat{y} = \sum_{i=1}^{k} \frac{w_i}{W} y^{(i)}$$

Example

- Learning example

No.	age	income	student	credit	BUY?
1	0	1.0	0	0	0
2	0	1.0	0	1.0	0
3	0.5	1.0	0	0	1.0
4	1.0	0.5	0	0	1.0
5	1.0	0	1.0	0	1.0
6	1.0	0	1.0	1.0	0
7	0.5	0	1.0	1.0	1.0
8	0	0.5	0	0	0
9	0	0	1.0	0	1.0
10	1.0	0.5	1.0	0	1.0
11	0	0.5	1.0	1.0	1.0
12	0.5	0.5	0	1.0	1.0
13	0.5	1.0	1.0	0	1.0
14	1.0	0.5	0	1.0	0

Example

- Training data → map to 4-D space
  - d(**x**, $\mathbf{x}^{(i)}$) : manhattan distance
  - K = 3
- Classify new data

  x: (age="20", income="med", student="yes", credit="fair")
  → (0, 0.5, 1, 0)

  - 3-NN: 9. (0, 0, 1, 0), d = 0.5, w = 1/0.5 → 1(yes),
    8. (0, 0.5, 0, 0), d = 1.0, w = 1/1.0 → 0(no)
    11. (0, 0.5, 1, 1), d = 1.0, w = 1/1.0 → 1(yes)
  - W = 4
  - y = 2/4 x 1(yes) + 1/4 x 0(no) + 1/4 x 1(yes) = 0.75

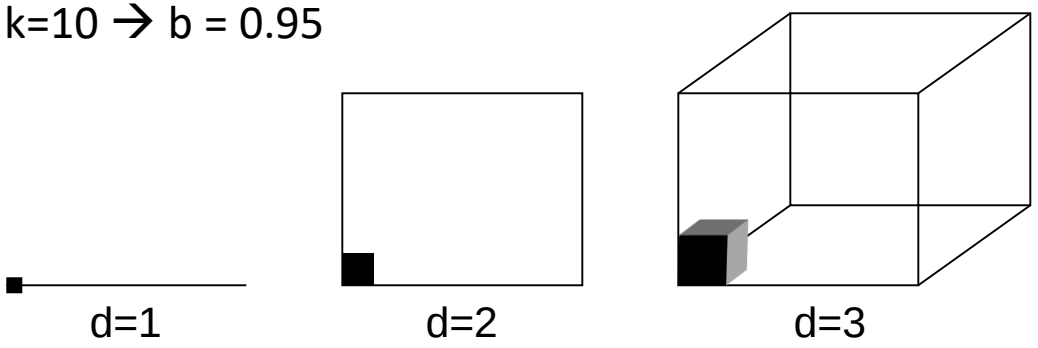
  ⇒ Classify x as "**Yes**"

Curse of Dimensionality

- For high dimensional data (large number of features)
  - Neighbors can be far away → k-NN is not appropriate
    - Distance between neighbors can be dominated by irrelevant features
  - Number of data: N, dimension: d, all values are in [0, 1] → total volume: 1^d
  - All neighborhood distance is less than b ($0 \leq b \leq 1$) → total volume: b^d
    - To contain k data, the neighbors should occupy k/N fraction, i.e. $b^d = (k/N)*1$

      → **$b = (k/N)^{1/d}$**
- Example
  - N=1000, d=2, k=10 → b = 0.1
  - N=1000, d=100, k=10 → b = 0.95

For k/ N = 1/100 :



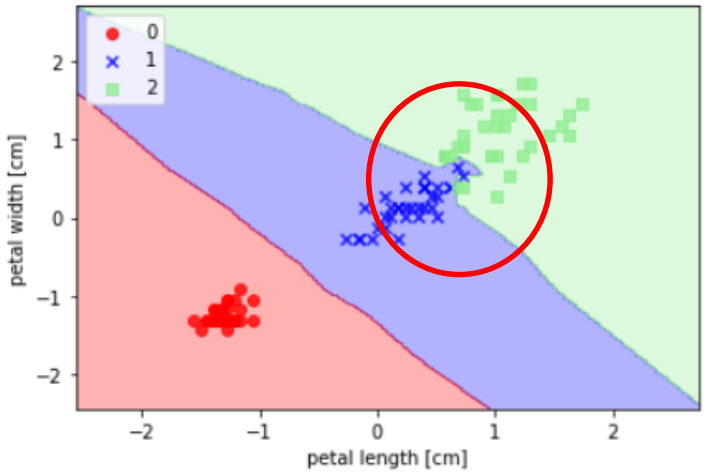
K-Nearest Neighbors Using Scikit-learn

- K-NN on Iris dataset (binary classification)

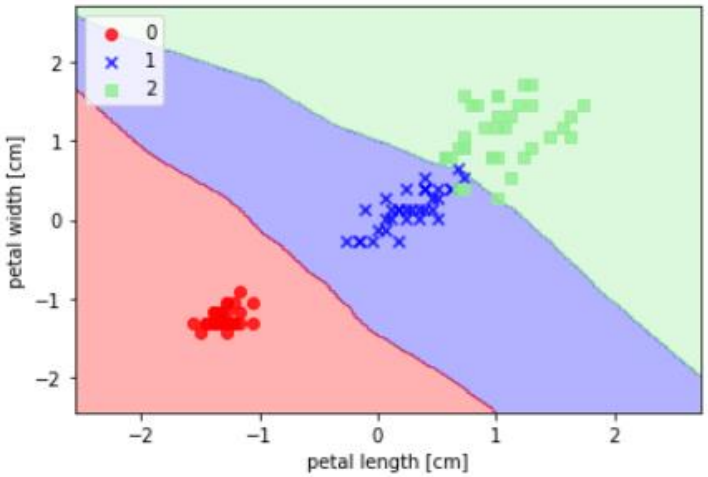
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3, p=2)
knn.fit(X_train_std, y_train)
```

K = 5



K = 9



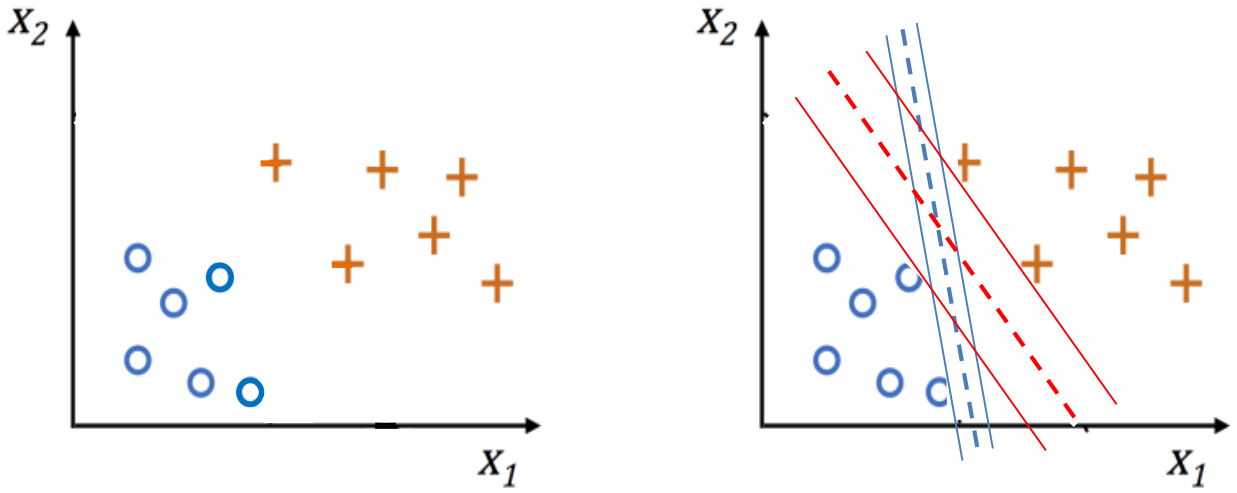
- Model accuracy

```
Train Accuracy :  0.9904761904761905
Test Accuracy :  0.9777777777777777
```

```
Train Accuracy :  0.9523809523809523
Test Accuracy :  0.9777777777777777
```

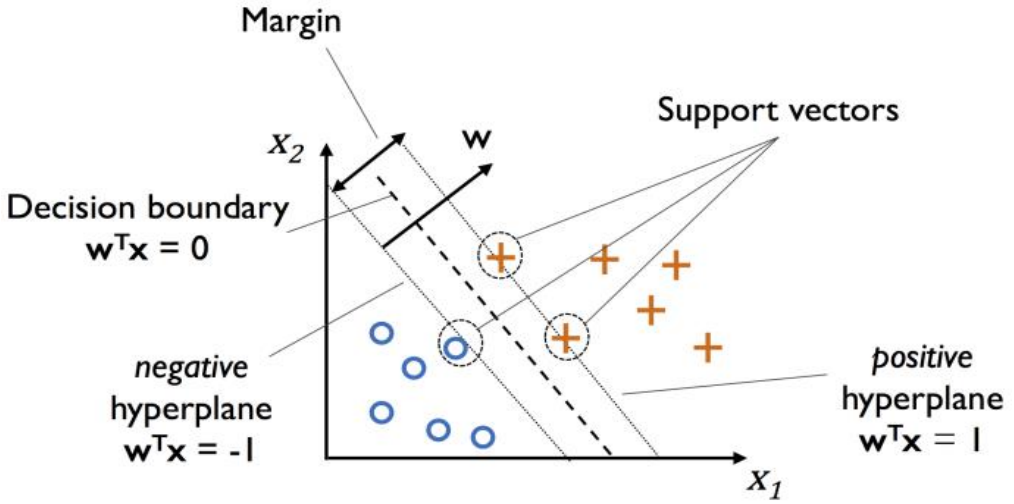
Support Vector Machines

- SVM is a linear classifier
  - Given dataset $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, y_i = +1 or -1, decision boundary is $\mathbf{w}^T\mathbf{x} + b = 0$
  - Margin - width that the boundary can be increased before hitting a data point
  - SVM tries to find a hyperplane that maximize the margin
  - The maximum-margin separator is determined by a subset of the data points
    – support vectors



Support Vector Machines

- Decision boundary
  - Positive hyperplane = $\mathbf{w}^T\mathbf{x} + b = +1$
  - Negative hyperplane = $\mathbf{w}^T\mathbf{x} + b = -1$
- Classification of data x
  - $y = +1 \quad if \quad \mathbf{w}^T\mathbf{x} + b \geq 1$
  - $y = -1 \quad if \quad \mathbf{w}^T\mathbf{x} + b \leq -1$



Support Vector Machines

- Finding maximum margin M
  - $\mathbf{w}$ is perpendicular to the hyperplane
    - Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two vectors on the same hyperplane
      Then $\mathbf{w}^T\mathbf{x}_1 + b = \mathbf{w}^T\mathbf{x}_2 + b = 0$. $\mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0$
  - Let $\mathbf{x}^-$: any point on the negative hyperplane

    $\mathbf{x}^+$: closest point on the positive hyperplane

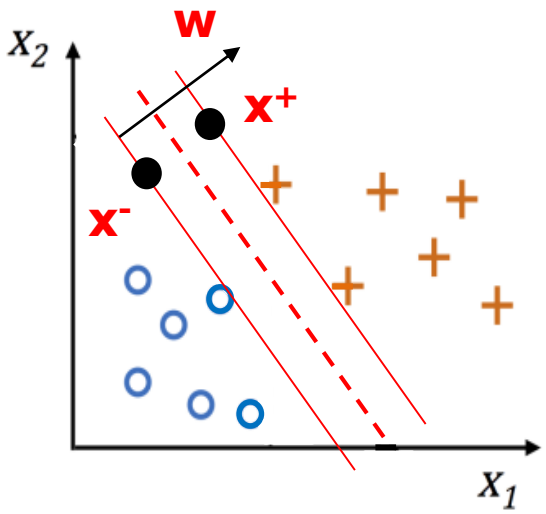
    → $\mathbf{x}^+ = \mathbf{x}^- + \lambda\mathbf{w}$ → $\mathbf{x}^+ - \mathbf{x}^- = \lambda\mathbf{w}$
  - Subtract $\mathbf{w}^T\mathbf{x}^+ + b = +1$

    $\mathbf{w}^T\mathbf{x}^- + b = -1$

    → $\mathbf{w}^T(\mathbf{x}^+ - \mathbf{x}^+) = 2$

    $\mathbf{w}^T\lambda\mathbf{w} = 2$

    $\lambda = \frac{2}{\|\mathbf{w}\|^2}$
  - Margin $M = \|\mathbf{x}^+ - \mathbf{x}^+\| = \lambda\|\mathbf{w}\| = \frac{2}{\|\mathbf{w}\|^2}\|\mathbf{w}\| = \frac{2}{\|\mathbf{w}\|}$
  - So, maximize M = minimize $\frac{\|\mathbf{w}\|}{\mathbf{2}}$



Support Vector Machines

- The optimization problem
  - Minimize $\|\mathbf{w}\|^2$

    Constraints : $\mathbf{w}^T\mathbf{x}_i + b \geq 1 \quad if\ y_i = 1$

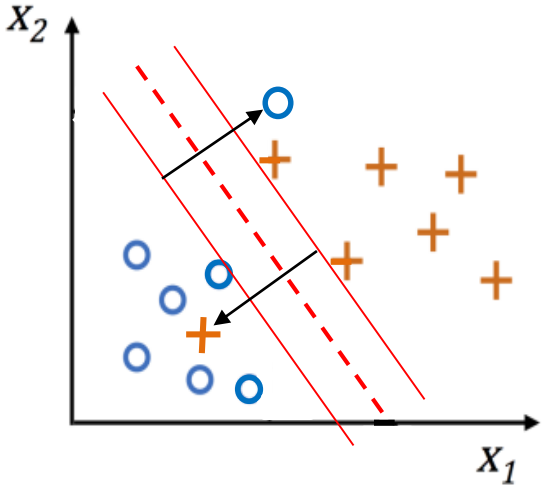
    $\mathbf{w}^T\mathbf{x}_i + b \leq -1 \quad if\ y_i = -1$

- Dealing with noise points (soft margin)
  - Minimize $\|\mathbf{w}\|^2 + C\sum \epsilon_i$

    Constraints : $\mathbf{w}^T\mathbf{x}_i + b \geq 1 - \epsilon_i \quad if\ y_i = 1$

    $\mathbf{w}^T\mathbf{x}_i + b \leq -1 + \epsilon_i \quad if\ y_i = -1$

    $\epsilon_i \geq 0$



⇒ Solve by *Quadratic Programming* method

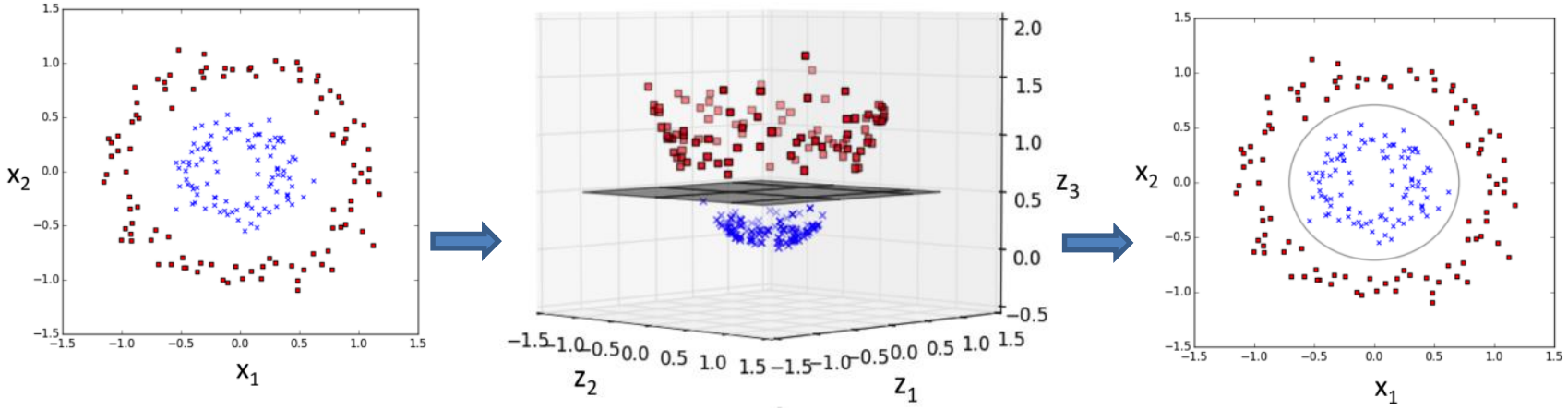
on its dual form (* can be very expensive for big datasets)

⇒ $\mathbf{w} = \sum c_i y_i \mathbf{x}_i$, $\mathbf{x}_i$ are support vectors

Support Vector Machines

- Kernel trick
  - For non-linear decision boundary, transform data $\mathbf{x}$ into high-dimensional space of feature vectors → $\phi(\mathbf{x})$
  - Then we can get a linear boundary in the high-dimensional space
  - Example:

$$\text{for } \mathbf{x} = (x_1, x_2) \text{ (2D)} \quad \rightarrow \quad \phi(\mathbf{x}) = (x_1, x_2, x_3 = {x_1}^2 + {x_2}^2) \text{ (3D)}$$



Support Vector Machines

- Kernel function
  - The kernel function is a function that has property:

$$K(\mathbf{x}_i, \mathbf{x}_i) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$$

- Example
  - $x \rightarrow \phi(x) = (x^4, 2x^3, \sqrt{6x^2}, 2x, 1)$
  - $\phi(x) \cdot \phi(y) = (x^4, 2x^3, \sqrt{6x^2}, 2x, 1) \cdot (y^4, 2y^3, \sqrt{6y^2}, 2y, 1)$

$$= x^4y^4 + 4x^3y^3 + 6x^2y^2 + 4xy + 1$$
$$= (xy + 1)^4$$
$$= K(x, y)$$

Support Vector Machines

- Using kernel function
  - $K(\mathbf{x}_i, \mathbf{x}_i) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$
  - Solving the optimization problem, **w** can be obtained by using transformed support vectors $\phi(\mathbf{x}_i)$ in a form like:

$$\mathbf{w} = \sum c_i y_i \phi(\mathbf{x}_i)$$

  - The prediction for **x** can be done by computing

$$\begin{aligned} & \mathbf{w}^T \phi(\mathbf{x}) + b \\ &= \sum c_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b \\ &= \sum c_i y_i \, K(\mathbf{x}_i, \mathbf{x}) + b \end{aligned}$$

→ much less computation

  - Example kernel functions:
    - Polynomial Function $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}.\mathbf{y} + 1)^p$
    - Gaussian Radial Basis Function $K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2 / 2\sigma^2}$

SVM Using Scikit-learn

- SVM on a sample dataset (binary classification, non-linear boundary)

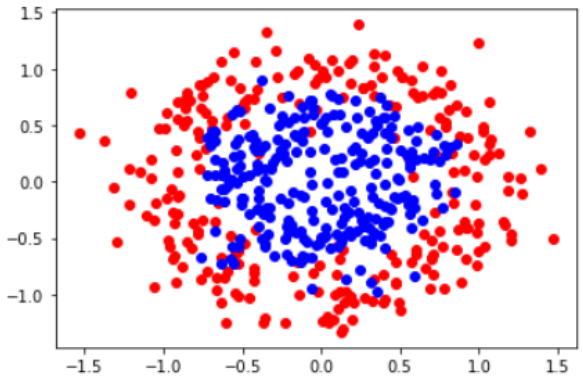
```
X, y = make_circles(n_samples=500, noise=0.2, factor=0.5, random_state=0)
```

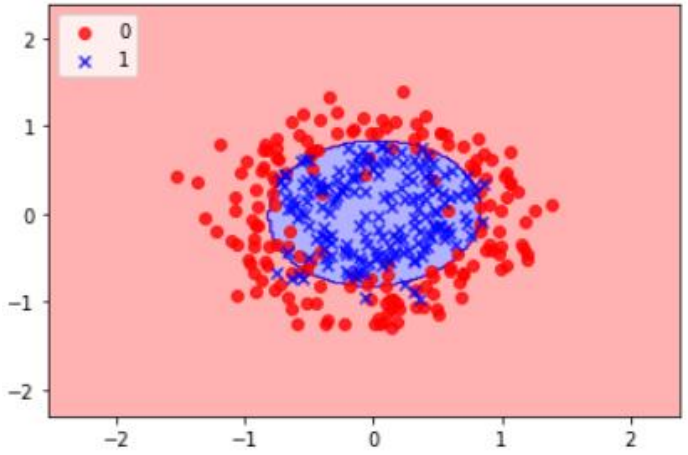
```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=1, gamma=0.2, C=1.0)
svm.fit(X_train, y_train)
```

C: Regularization parameter
gamma: Kernel coefficient

Controls shape of boundary and overfitting





Comparison

- Non-linear decision boundaries for different models