Lecture 2
Support Vector Machine

Wonjun Lee

Objectives

- Learn powerful machine learning algorithms that are commonly used in academia as well as in industry
- Develop an appreciation of their individual strengths and weaknesses
- Experience scikit-learn library, which offers a user-friendly and consistent interface for using those algorithms

Contents

- Overfitting vs. Underfitting
- Support Vector Machine
- Decision Tree

Overfitting vs. Underfitting

Choosing a Classification Algorithm

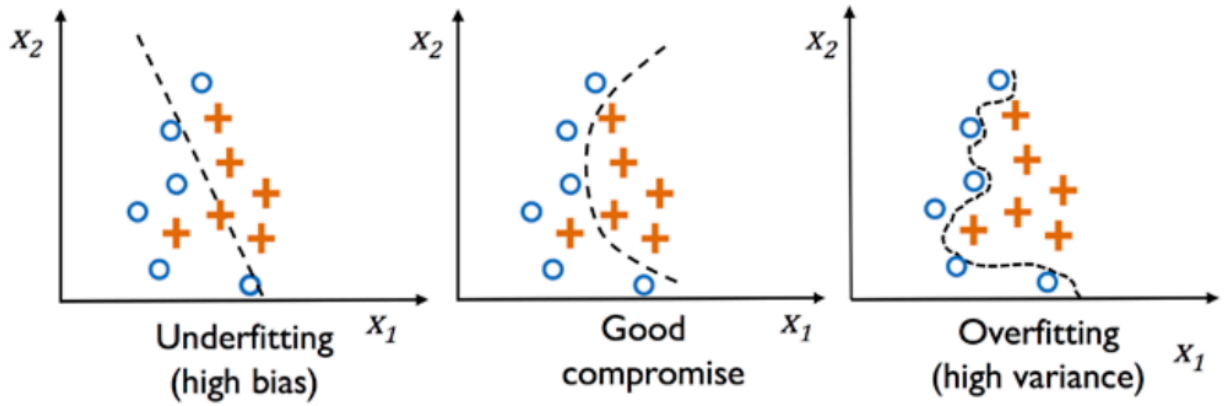
- No Free Lunch Theorem – no single classifier works best across all possible scenarios
  - ✓compare the performance of at least a handful of different learning algorithms to select the best model for the particular problem

Tackling Overfitting Via Regularization

- Fundamental issue: tension between **optimization** and **generalization**
  - ✓ ***Optimization*** refers to the process of adjusting a model to get the best performance possible on the training data
  - ✓ ***Generalization*** refers to how well the trained model performs on data it has never seen before

Tackling Overfitting Via Regularization

- Underfitting
  - ✓High bias
  - ✓Model is not complex enough to capture the pattern in the training data well → suffer from low performance on unseen data
- Overfitting
  - ✓High variance ← too many parameters (complex)
  - ✓Model performs well on training data but does not generalize well to unseen (test) data



Generalization

- The central challenge in machine learning is that algorithm must perform well on new, previously unseen inputs—not just those on which our model was trained:
  - ✓The ability to perform well on previously unobserved inputs is called **generalization**
- Training error: error measuring on training set
  - ✓optimization problem
- Test error: expected value of the error on a new input (generalization error)

Generalization

- Generalization error = test error
- Generalization error: expected value of error on a new input
  - ✓ Expectation is taken across different possible inputs, drawn from the distribution of inputs we expect the system to encounter in practice

$$\frac{1}{m^{(\text{test})}} ||\boldsymbol{X}^{(\text{test})}\boldsymbol{w} - \boldsymbol{y}^{(\text{test})}||_2^2$$

- How can we affect performance on the test set when we can observe only the training set?

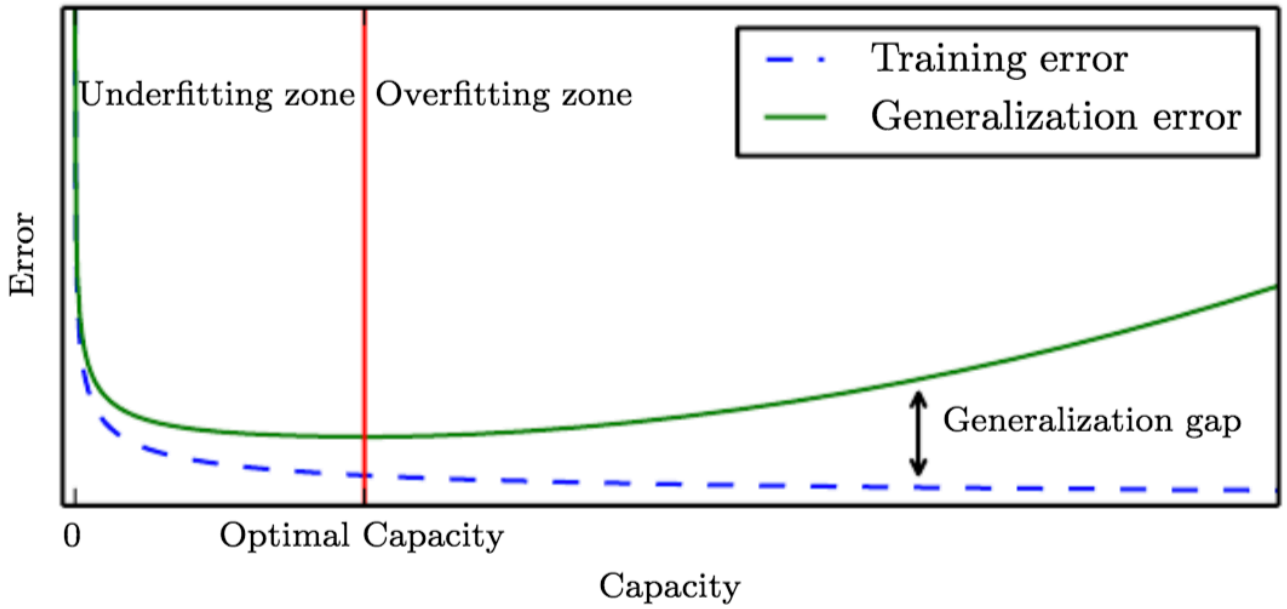
Generalization Error: Statistical Learning Theory

- If the training and the test set are collected arbitrarily, there is indeed little we can do
- But, assumptions – The training and test data are generated by a probability distribution over datasets
- In another, assumptions – I.I.D.
  - ✓Examples in each dataset are **Independent** from each other
  - ✓Training and test set are **Identically distributed**

Generalization Error: Statistical Learning Theory

- At the beginning of training, optimization and generalization are correlated
  - ✓The lower the loss on training data, the lower the loss on test data
  - ✓While this is happening, the model is said to be ***underfit***: there is still progress to be made
- But after a certain number of iterations on the training data, generalization stops improving, and validation metrics stall and then begin to degrade
  - ✓the model is starting to ***overfit***
  - ✓That is, it's beginning to learn patterns that are specific to the training data but that are misleading or irrelevant when it comes to new data

Generalization Error: Statistical Learning Theory



- At the left end of the graph, training error and generalization error are both high
- Underfitting regime: As we increase capacity, training error decreases, but the gap between training and generalization error increases
- Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity

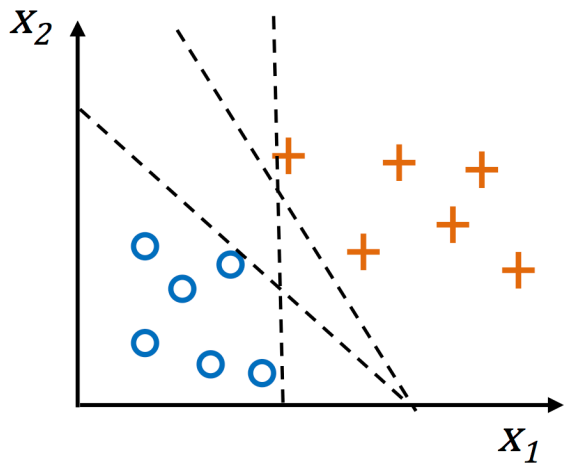
Overfitting and Underfitting

- The best solution is to get more training data
  - ✓A model trained on more data will naturally generalize better
- When that isn't possible, the next-best solution is to modulate the quantity of information that the model is allowed to store or to add constraints on what information it's allowed to store
- If a network can only afford to memorize a small number of patterns, the optimization process will force it to focus on the most prominent patterns, which have a better chance of generalizing well
- The processing of fighting overfitting this way is called ***regularization***

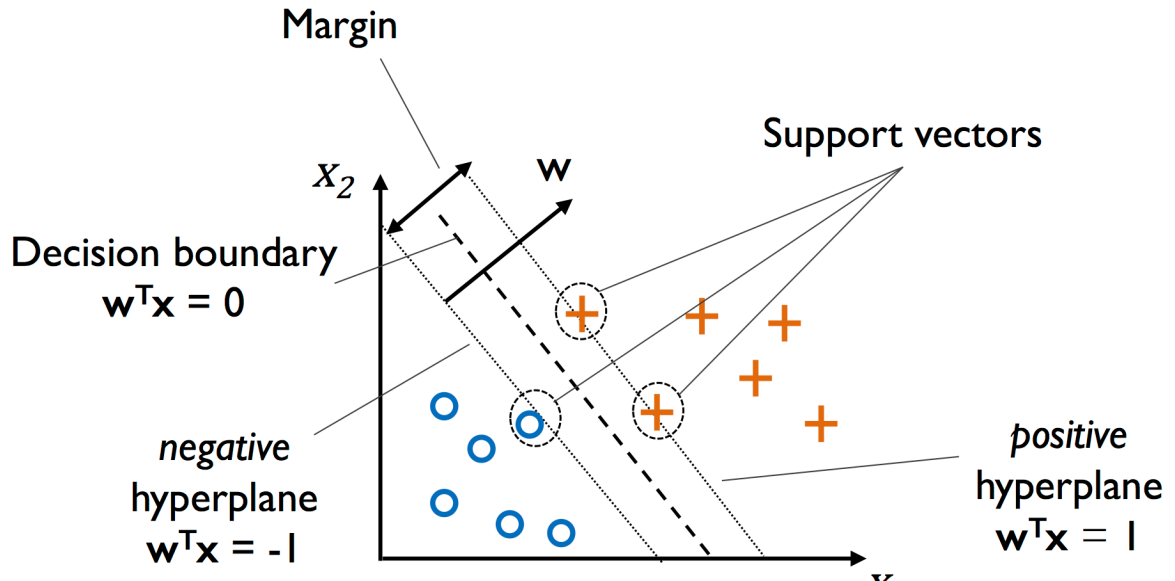
Support Vector Machine

Maximum Margin Classification with Support Vector Machines (SVM)

- Objective of SVM – maximize the margin
  - ✓Margin : distance between the separating hyperplane (decision boundary) and the training examples that are closest to this hyperplane (support vectors)



Which hyperplane?

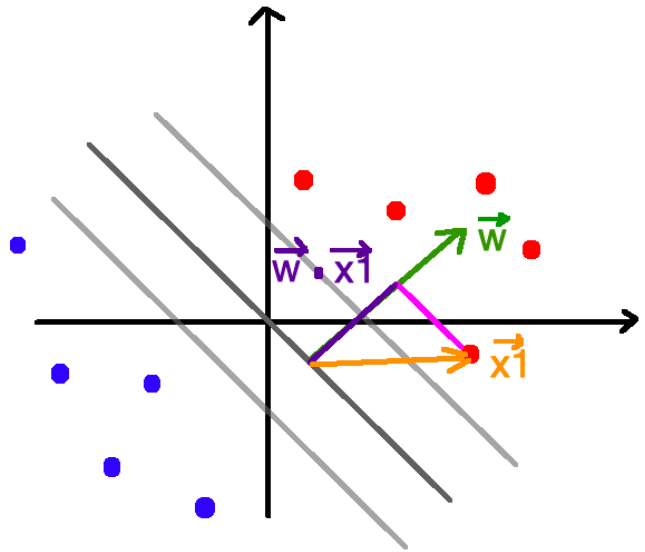


SVM:
Maximize the margin

Maximum Margin Classification with Support Vector Machines (SVM)

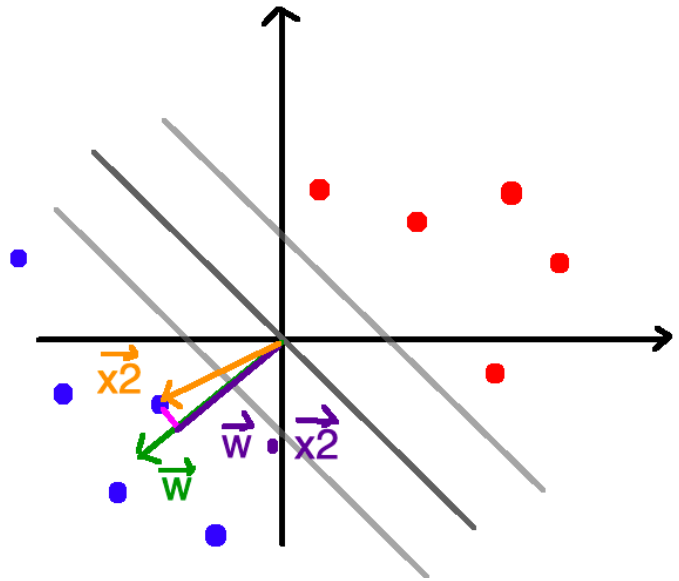
- If it's a positive sample, we're going to insist that the proceeding decision function returns a value greater than or equal to 1

$$w_0 + w^T x_{pos} \geq 1$$



- If it's a negative sample, we're going to insist that the proceeding decision function returns a value smaller than or equal to -1

$$w_0 + w^T x_{neg} \leq -1$$



Maximum Margin Classification with Support Vector Machines (SVM)

- Models with smaller margins are more prone to overfitting
- Positive and negative hyperplane :

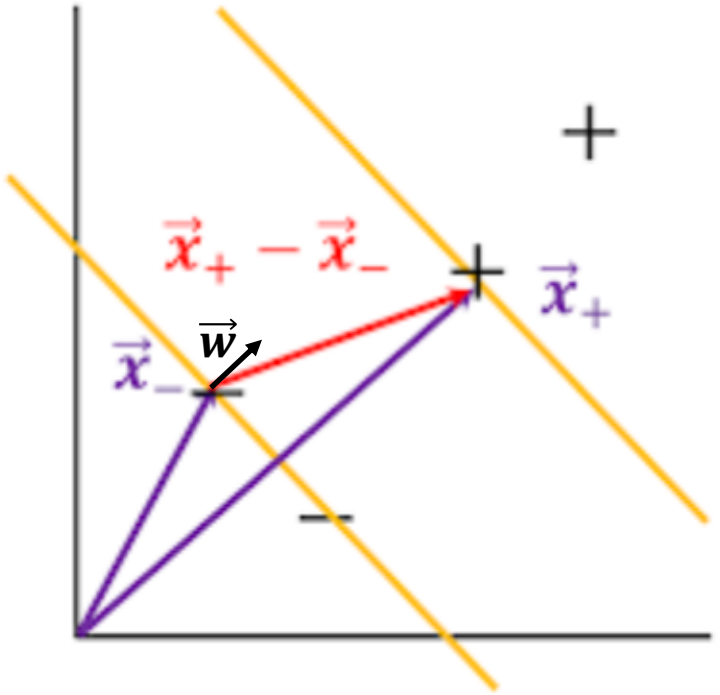
$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1$$

$$\Rightarrow \boldsymbol{w}^T (\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg}) = 2$$

$$\frac{\boldsymbol{w}^T (\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg})}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$$

Width



Maximum Margin Classification with Support Vector Machines (SVM)

- Objective function of the SVM becomes the maximization of this margin by **maximizing** $\frac{2}{\|w\|}$ under constraint that the examples are classified correctly as:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \quad if\ y^{(i)} = 1$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \leq -1 \quad if\ y^{(i)} = -1$$

$$y^{(i)}(w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)}) \geq 1 \qquad \forall_i$$

-

$$y^{(i)}\left(w_0 + w^T x^{(i)}\right) - 1 = 0, for\ x^{(i)}\ in\ gutter$$

for i = 1, …, N (# of examples)

Nonlinearly Separable Case Using Slack Variables

- Soft-margin Classification – linear constraints need to be relaxed for nonlinearly separable data using slack variable ξ
- Positive-valued ξ is simply added to the linear constraints:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 - \xi^{(i)} \quad if\ y^{(i)} = 1$$
$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \leq -1 + \xi^{(i)} \quad if\ y^{(i)} = -1$$

- So, the new objective is minimized:

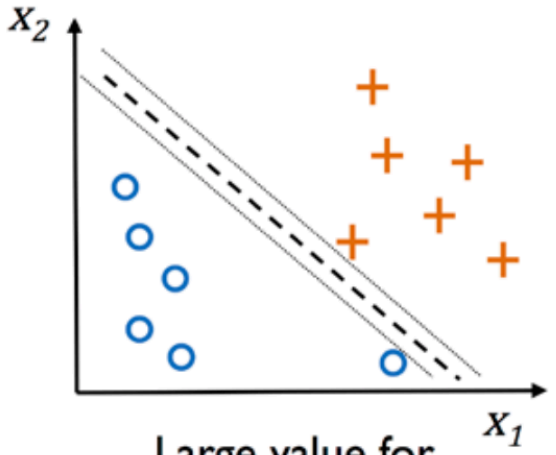
$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i \xi^{(i)}\right)$$

maximizing $\frac{2}{\|\boldsymbol{w}\|}$ → **maximizing** $\frac{1}{\|\boldsymbol{w}\|}$ → minimizing $\| \boldsymbol{W} \|$ →minimizing $\frac{\|\boldsymbol{w}\|^2}{2}$ since $\frac{d}{dx}\frac{x^2}{2}$ = x

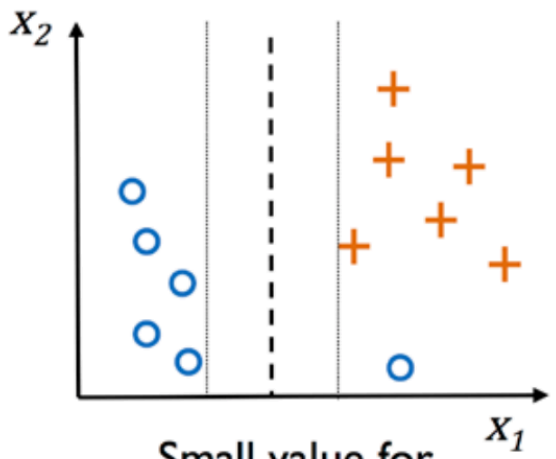
Nonlinearly Separable Case Using Slack Variables

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i \xi^{(i)}\right)$$

- Control the penalty for misclassification via C
- Large C → large error penalties
- Small C → less strict about misclassification errors
  - ✓ decreasing C → increases bias (bad optimization) and lowers variance (better regularization) of model



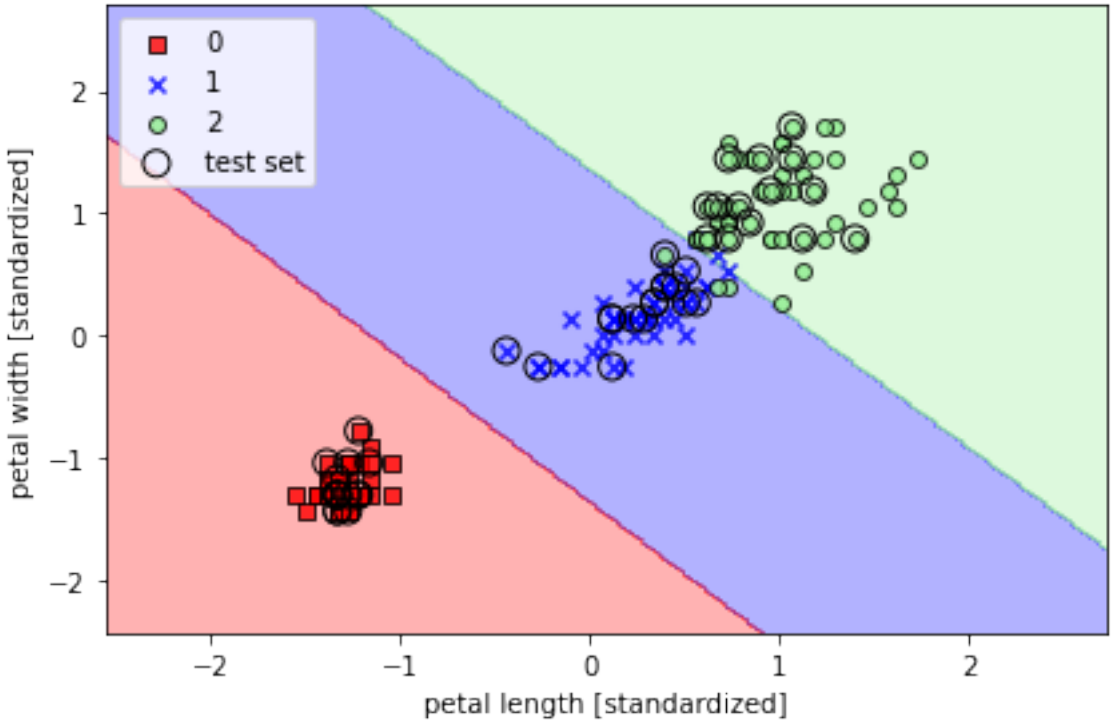
Large value for parameter C



Small value for parameter C

Nonlinearly Separable Case Using Slack Variables

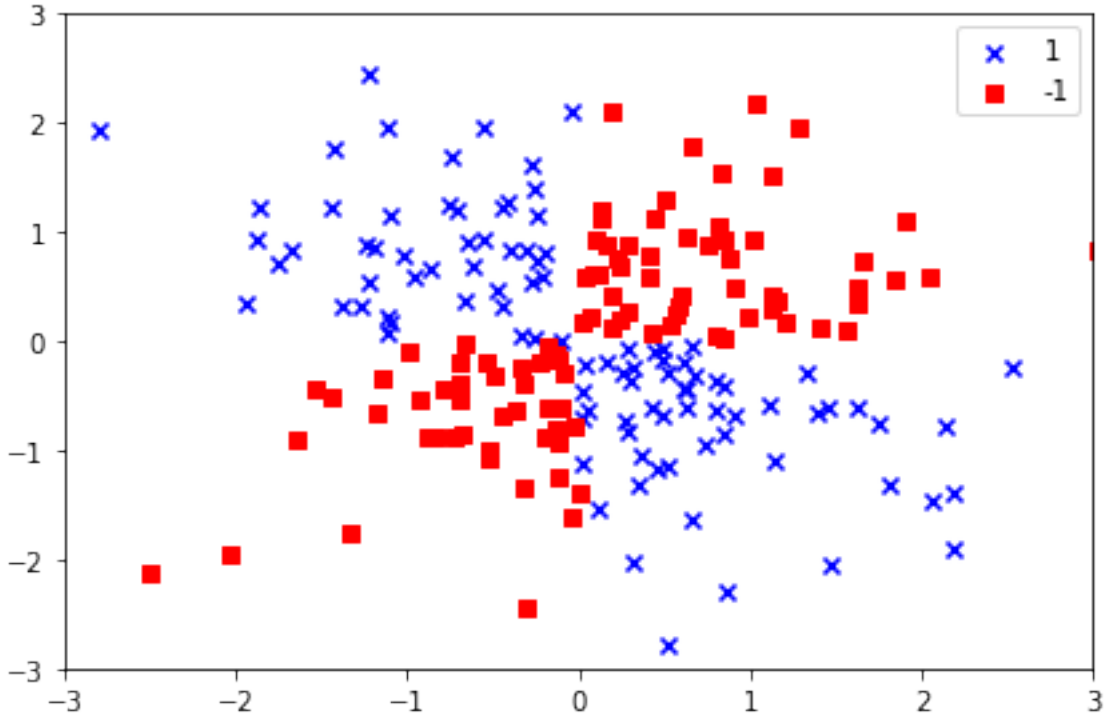
- Code → http://127.0.0.1:8888/notebooks/work/python-machine-learning-book-3rd-edition-master/ch03/ch03.ipynb#Dealing-with-the-nonlinearly-separable-case-using-slack-variables



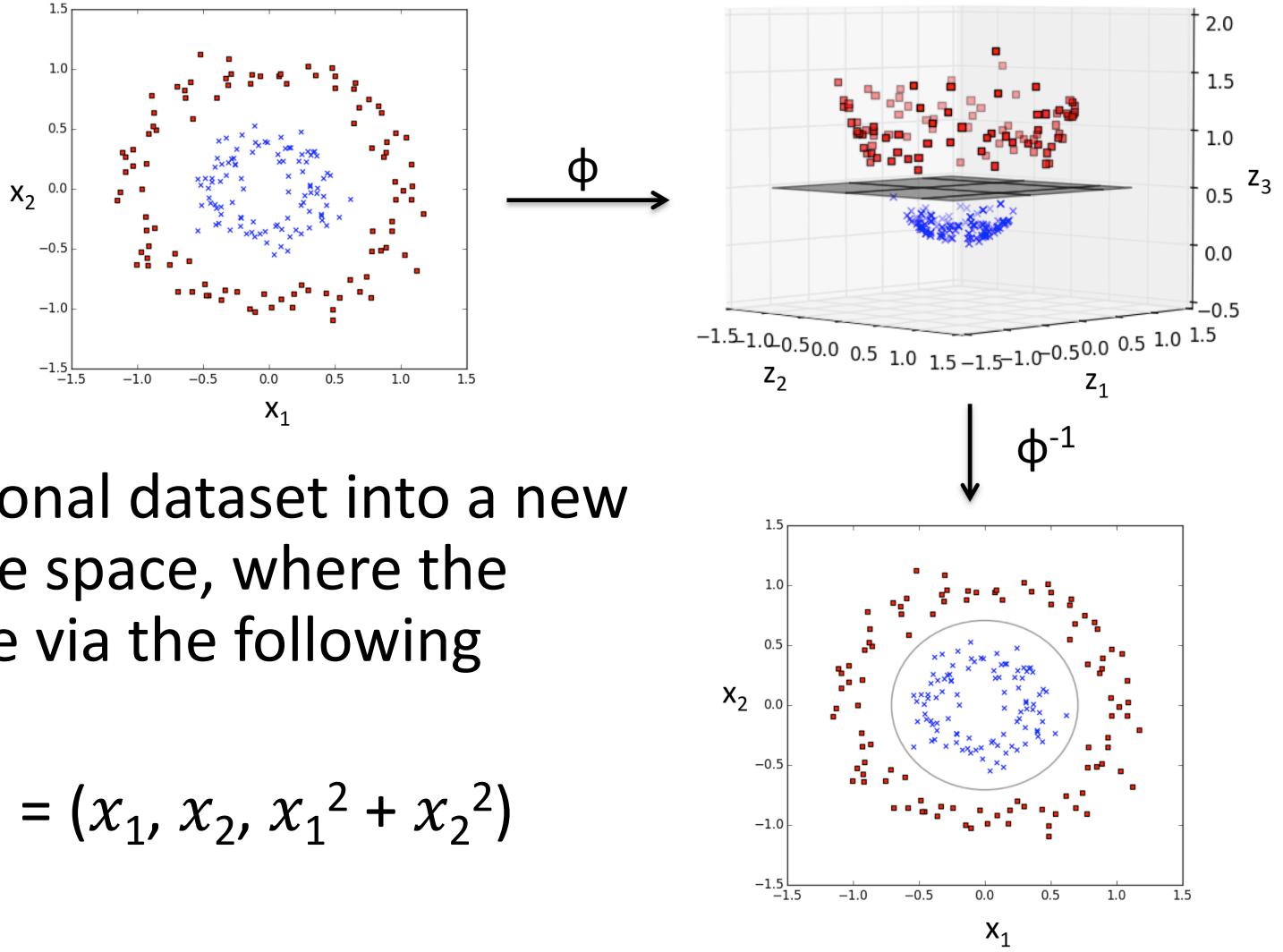
Solving Nonlinear Problems Using a Kernel SVM

- Create a simple dataset : 100 examples → 1, 100 examples → -1
  - ✓Use XOR gate – *logical_xor* (NumPy)

```
np.random.seed(1)
X_xor = np.random.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0,
                       X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)
```



Solving Nonlinear Problems Using a Kernel SVM



- Transform a two-dimensional dataset into a new three-dimensional feature space, where the classes become separable via the following projection:

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

Kernel Trick to Find Separating Hyperplanes in a High-Dimensional Space

- Transforming training data into a higher-dimensional feature space via a mapping function, ϕ is computationally very expensive → Kernel trick is required
- Kernel function: $\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$
- ϕ is a function that projections vectors x into a new vector space. The kernel function computes the inner-product between two projected vectors.
- Computing $\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is cheaper than $\phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$

Kernel Trick to Find Separating Hyperplanes in a High-Dimensional Space

- Optimize: $L_d = \sum a_i - \frac{1}{2}\sum a_i a_j y_i y_j (x_i \bullet x_j)$ where $(x_i \cdot x_j)$ is the dot product of two feature vectors
- If we transform to ϕ, instead of $x_i \cdot x_j$, we will have to compute $\phi(x_i) \bullet \phi(x_j)$ which is expensive
- If there is a ”kernel function” K such that $k(x_i, x_j) = \phi(x_i) \bullet \phi(x_j)$, then we do not need to know or compute ϕ at all ← kernel function defines similarity in the transformed space
- Function we end up optimize: $L_d = \sum a_i - \frac{1}{2}\sum a_i a_j y_i y_j K(x_i \bullet x_j)$

Kernel Trick

$$\mathbb{R}^2 \rightarrow x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}$$

$$\emptyset(x_i) = \begin{bmatrix} {x_{i1}}^2 \\ x_{i1}x_{i2} \\ x_{i2}x_{i1} \\ {x_{i2}}^2 \end{bmatrix}, \emptyset(x_j) = \begin{bmatrix} {x_{j1}}^2 \\ x_{j1}x_{j2} \\ x_{j2}x_{j1} \\ {x_{j2}}^2 \end{bmatrix}$$

$$\mathbb{R}^2 \rightarrow \mathbb{R}^4$$

$$\emptyset(x_i)\emptyset(x_j)$$

- Ex) $x_i = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $x_j = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$

$$\emptyset(x_i) = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 4 \end{bmatrix}, \emptyset(x_j) = \begin{bmatrix} 9 \\ 15 \\ 15 \\ 25 \end{bmatrix}$$

$$\emptyset(x_i)\emptyset(x_j) = 1*9 + 2*15 + 2*15 + 4*25 = 169$$

Kernel trick:

$$K(xi, xj) = (\begin{bmatrix} 1 \\ 2 \end{bmatrix}^T \cdot \begin{bmatrix} 3 \\ 5 \end{bmatrix})^2 = (3+10)^2 = 169$$

Kernel Trick to Find Separating Hyperplanes in a High-Dimensional Space

- A **radial basis function** (**RBF**) **is** a **kernel function** that assigns a real value (measure of distance) to each input :

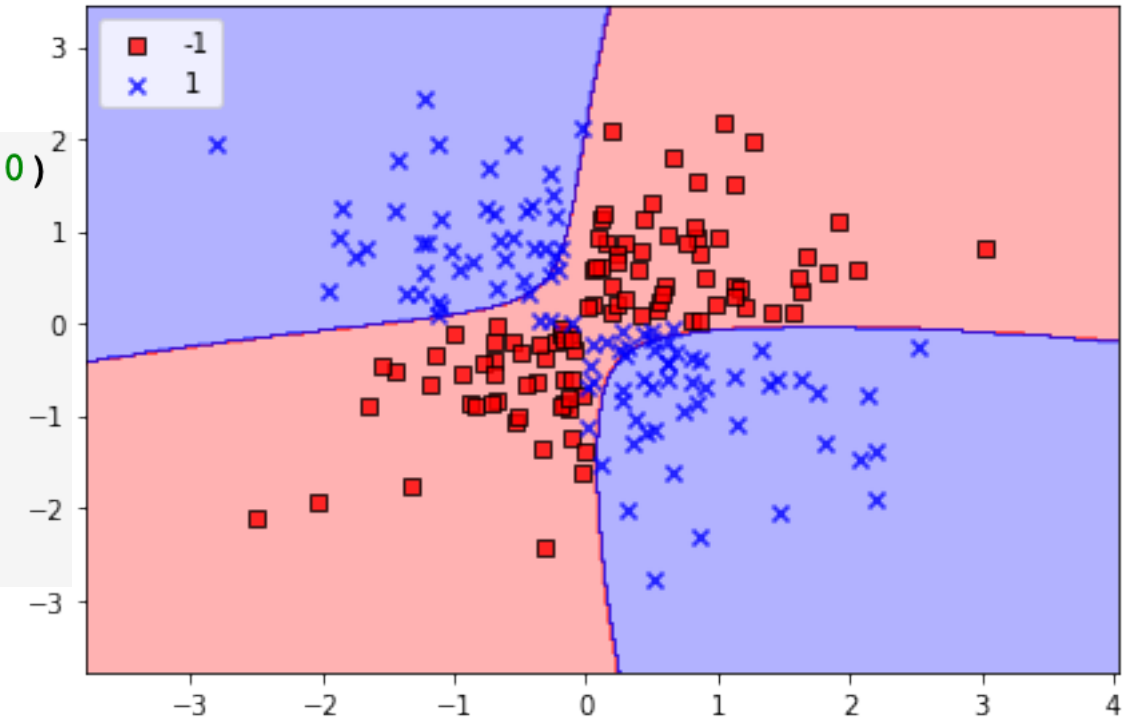
$$\kappa\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \exp\left(-\frac{\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2}{2\sigma^2}\right)$$

- $\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2$ = Euclidean Distance = L 2 norm
  - ✓ $k(x^{(i)}, x^{(j)}) = 1$ → $x^{(i)}$ and $x^{(j)}$ are exactly same from
  - ✓ $k(x^{(i)}, x^{(j)}) = 0$ → $x^{(i)}$ and $x^{(j)}$ are far from each other
  - ✓ (x1, x2) – closeness → similarity
- Gamma (γ) = $\frac{1}{2\sigma^2}$

$$\kappa\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \exp\left(-\gamma\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2\right)$$

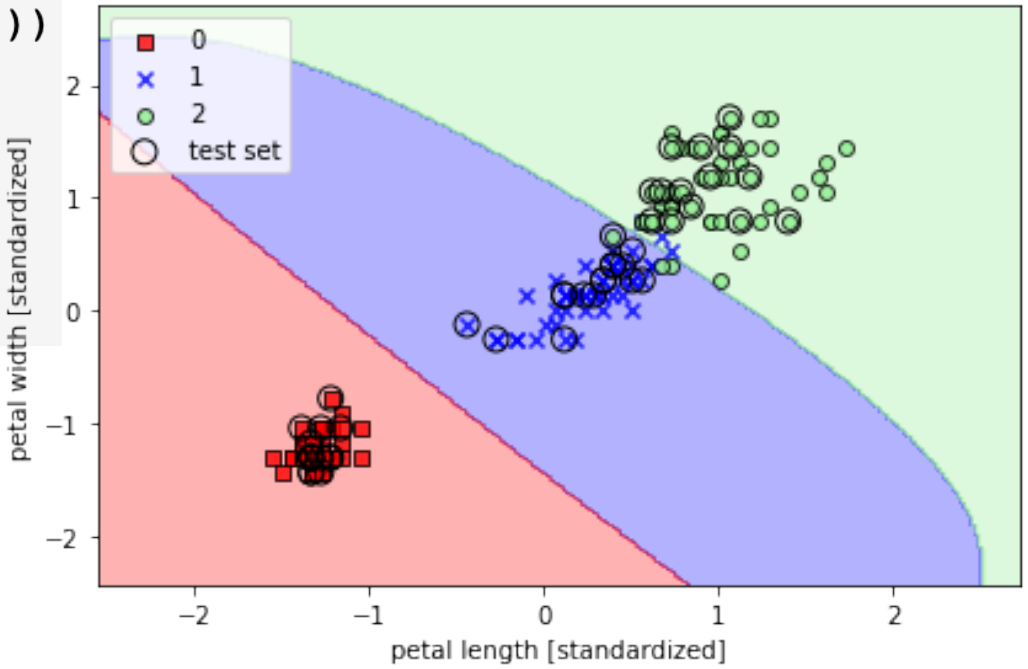
Kernel Trick to Find Separating Hyperplanes in a High-Dimensional Space

```
svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=10.0)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor,
                      classifier=svm)

plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_14.png', dpi=300)
plt.show()
```



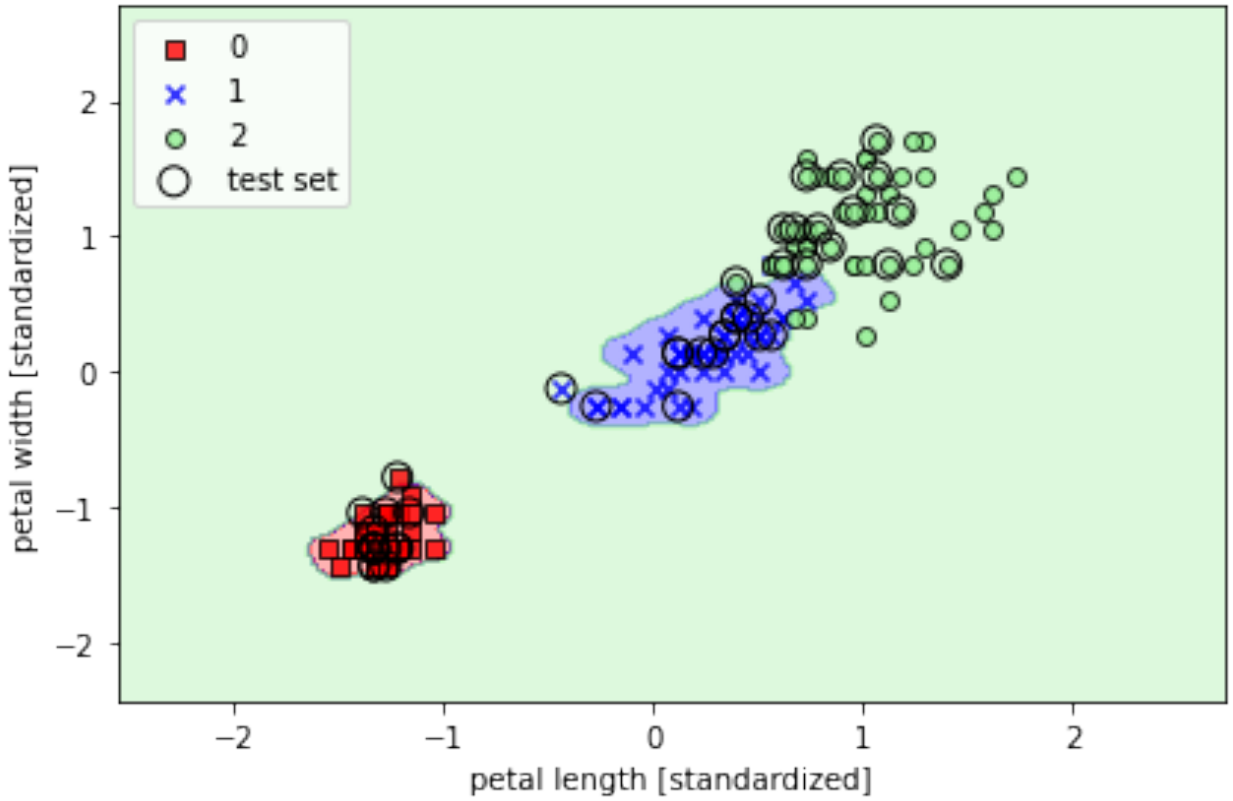
Kernel Trick to Find Separating Hyperplanes in a High-Dimensional Space

```
from sklearn.svm import SVC

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

plot_decision_regions(X_combined_std, y_combined,
                      classifier=svm, test_idx=range(105, 150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_15.png', dpi=300)
plt.show()
```



Kernel Trick to Find Separating Hyperplanes in a High-Dimensional Space

```
svm = SVC(kernel='rbf', random_state=1, gamma=100.0, C=1.0)
```

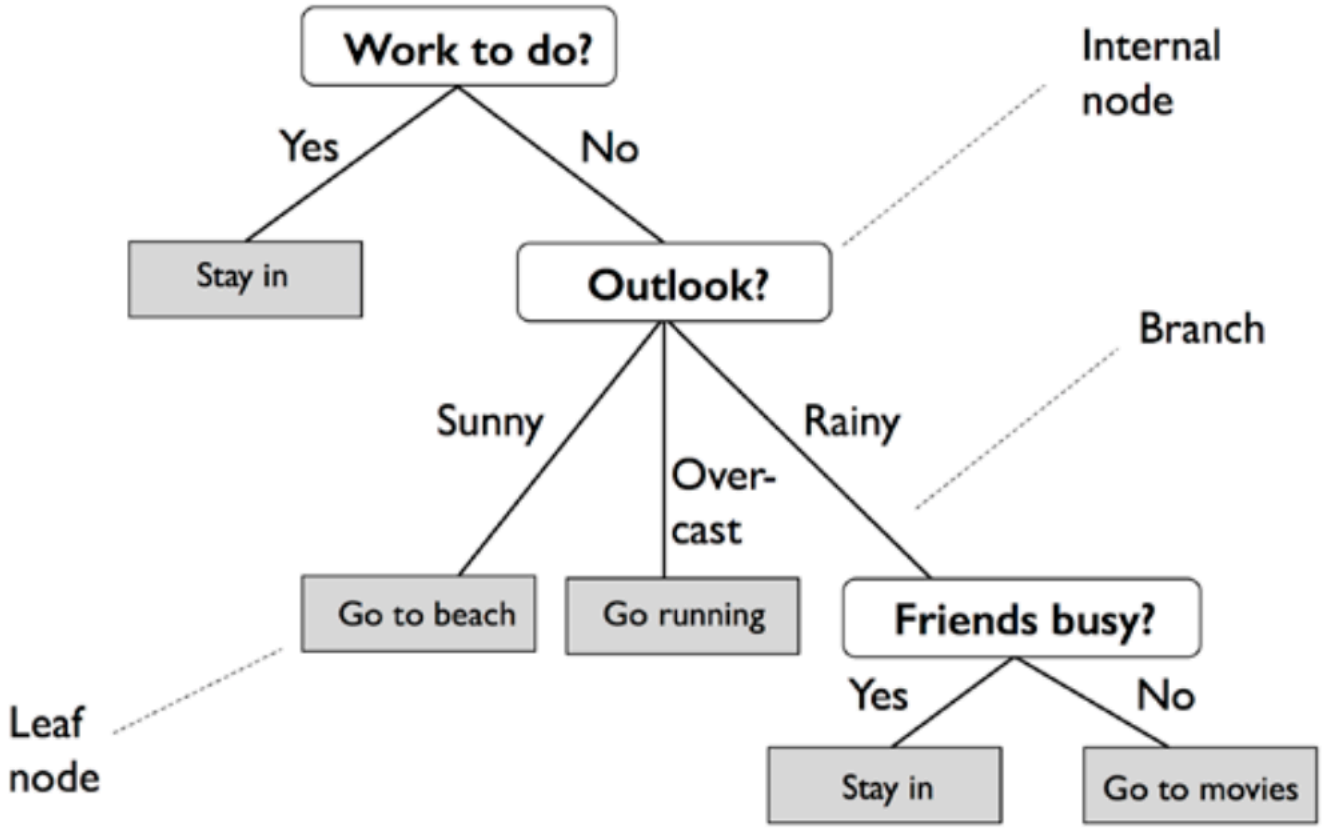


- Such a classifier will likely have a high generalization error on unseen data

Decision Tree

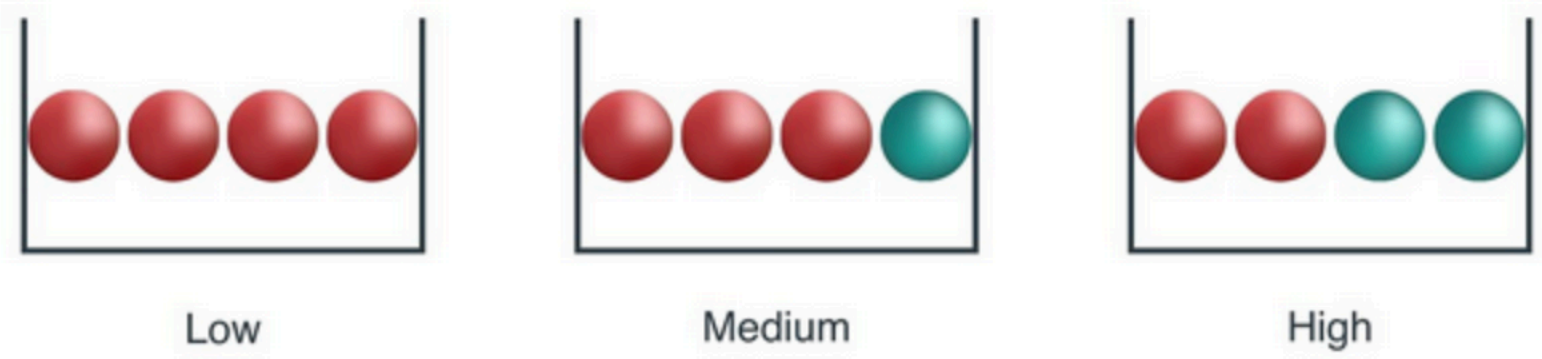
Decision Tree Learning

- Model as breaking down our data by making a decision based on asking a series of questions



Decision Tree Learning

- Impurity – Degree how much of information is needed to tell the color



Pure node:
Needed less info

Less impure:
Needed more info than left

More impure:
Needed max info as both

Decision Tree Learning

- How to measure impurity?
  - ✓Gini index / Gini impurity
  - ✓Entropy

Decision Tree Learning

- Gini index / Gini impurity – measure of inequality in sample
  - ✓[0, 1] : 0 → samples are perfectly homogeneous and all elements are similar
    1 → maximal inequality among elements
  - ✓Sum of square of probabilities of each class

$$1 - \sum_{i=1}^{n} {P_i}^2$$

- Impurity measures homogeneity

Decision Tree Learning

- Example:

  Predict

  '**Play (Yes)**'

  or

  '**not (No)**'

Day	outlook	temperature	humidity	wind	Decision
1	sunny	hot	high	weak	No
2	sunny	hot	high	strong	No
3	overcast	hot	high	weak	Yes
4	rainfall	mild	high	weak	Yes
5	rainfall	cool	normal	weak	Yes
6	rainfall	cool	normal	strong	No
7	overcast	cool	normal	wtrong	Yes
8	sunny	mild	high	weak	No
9	sunny	cool	normal	weak	Yes
10	rainfall	mild	normal	weak	Yes
11	sunny	mild	normal	strong	Yes
12	overcast	mild	high	strong	Yes
13	overcast	hot	normal	weak	Yes
14	rainfall	mild	high	strong	No

Decision Tree Learning

- Outlook:
  - ‘sunny’
  - ‘overcast’
  - ’rainfall’

Outlook	Yes	No	# Instances
sunny	2	3	5
overcast	4	0	4
rainfall	3	2	5

Gini index (outlook = sunny) = $1 – (2/5)^2 – (3/5)^2 = 0.48$

Gini index (outlook = overcast) = $1 – (4/4)^2 = 0$

Gini index (outlook = rainfall) = $1 – (3/5)^2 – (2/5)^2 = 0.48$

Weighted sum of Gini index for outlook features:

Gini index (outlook) = (5/14)*0.48 + (4/14)*0 + (5/14)*0.48 = **0.342**

Decision Tree Learning

- Temperature:
  - ‘hot’
  - ‘cold’
  - ’mild’

Temperature	Yes	No	# Instances
hot	2	2	4
cool	3	1	4
mild	4	2	6

Gini index (temperature = hot) = $1 – (2/4)^2 – (2/4)^2 = 0.5$

Gini index (temperature = cool) = $1 – (3/4)^2 – (1/4)^2 = 0.375$

Gini index (temperature = mild) = $1 – (4/6)^2 – (2/6)^2 = 0.445$

Weighted sum of Gini index for temperature features:

Gini index (temperature) = (4/14)*0.5 + (4/14)*0.375 + (6/14)*0.445 = **0.439**

Decision Tree Learning

- Humidity:
  - ‘high’
  - ‘normal’

Humidity	Yes	No	# Instances
high	3	4	7
Normal	6	1	7

Gini index (humidity = high) = $1 – (3/7)^2 – (4/7)^2 = 0.489$
Gini index (humidity = normal) = $1 – (6/7)^2 – (1/7)^2 = 0.244$

Weighted sum of Gini index for humidity features:
Gini index (humidity) = (7/14)*0.489 + (7/14)*0.244 = **0.367**

Decision Tree Learning

- Wind:
  - ‘weak’
  - ‘strong’

wind	Yes	No	# Instances
weak	6	2	8
strong	3	3	6

Gini index (wind = weak) = $1 – (6/8)^2 – (2/8)^2 = 0.375$

Gini index (wind = strong) = $1 – (3/6)^2 – (3/6)^2 = 0.5$

Weighted sum of Gini index for wind features:

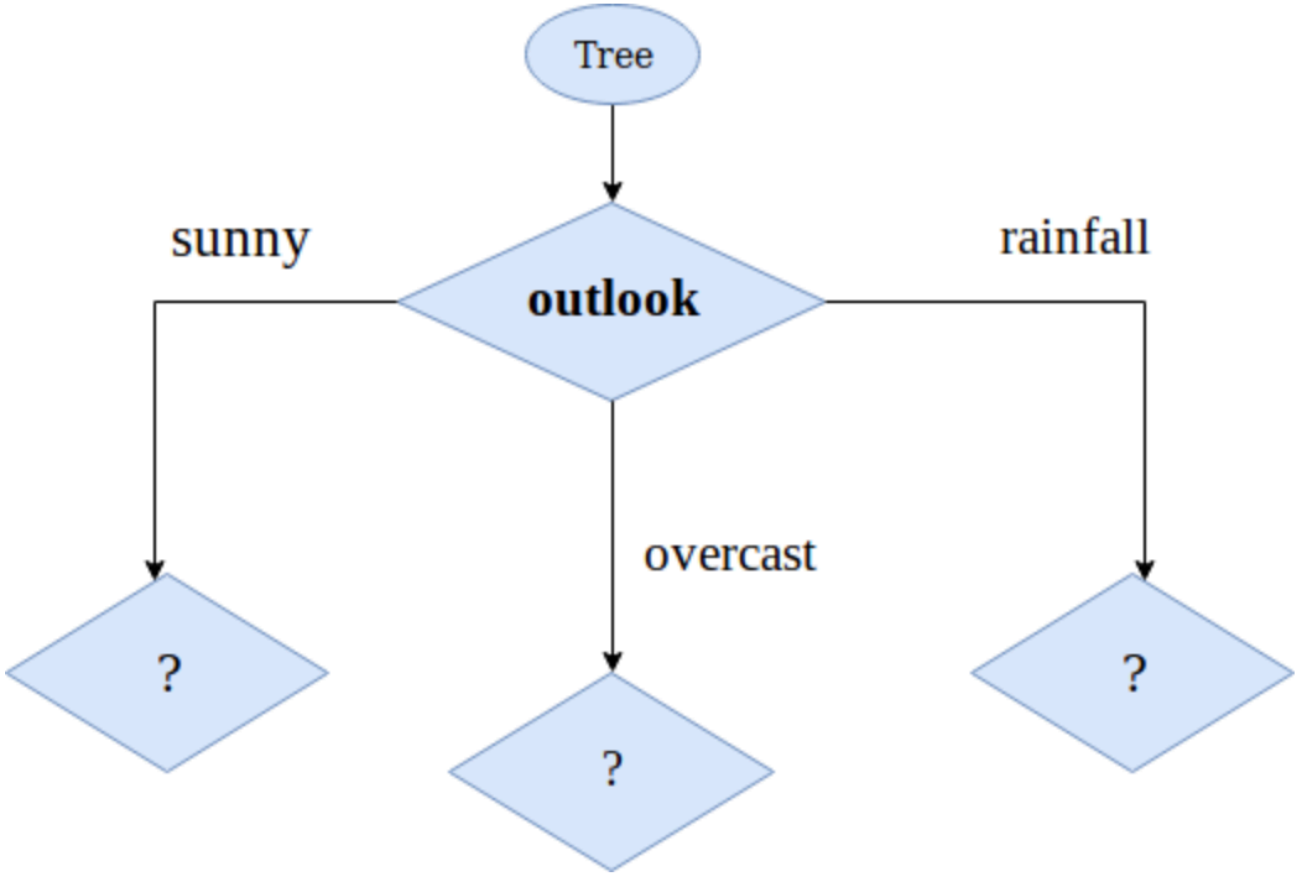
Gini index (wind) = (8/14)*0.375 + (6/14)*0.5 = **0.428**

Decision Tree Learning

- Decision for root node:

  Take the lowest

Features	Gini Index
outlook	0.342
temperature	0.439
humidity	0.367
wind	0.428



Decision Tree Learning

- Focus on sub data on ‘sunny’ outlook feature

Temperature	Yes	No	# Instances
hot	0	2	2
cool	1	1	1
mild	1	1	2

- Gini(outlook=sunny & temperature=hot) = $1 – (0/2)^2 – (2/2)^2 = 0$
- Gini(outlook=sunny & temperature=cool) = $1 – (1/1)^2 – (0/1)^2 = 0$
- Gini(outlook=sunny & temperature=mild) = $1 – (1/2)^2 – (1/2)^2 = 0.5$
- Gini(outlook=sunny & temperature) = $(2/5)*0 +(1/5)*0 + (2/5)*0.5 = 0.2$

Day	outlook	temperature	humidity	wind	decision
1	sunny	hot	high	weak	No
2	sunny	hot	high	strong	No
8	sunny	mild	high	weak	No
9	sunny	cool	normal	weak	Yes
11	sunny	mild	normal	strong	Yes

Decision Tree Learning

- Focus on sub data on ‘sunny’ outlook feature

Humidity	Yes	No	# Instances
high	0	3	3
Normal	2	0	2

Day	outlook	temperature	humidity	wind	decision
1	sunny	hot	high	weak	No
2	sunny	hot	high	strong	No
8	sunny	mild	high	weak	No
9	sunny	cool	normal	weak	Yes
11	sunny	mild	normal	strong	Yes

- Gini(outlook=sunny & humidity=high) =
  $1 - (0/3)^2 - (3/3)^2 = 0$
- Gini(outlook=sunny & humidity=normal) =
  $1 - (2/2)^2 - (0/2)^2 = 0$
- Gini(outlook=sunny & humidity) =
  $(3/5)*0 + (2/5)*0 = 0$

Decision Tree Learning

- Focus on sub data on ‘sunny’ outlook feature

wind	Yes	No	# Instances
weak	1	2	3
strong	1	1	2

Day	outlook	temperature	humidity	wind	decision
1	sunny	hot	high	weak	No
2	sunny	hot	high	strong	No
8	sunny	mild	high	weak	No
9	sunny	cool	normal	weak	Yes
11	sunny	mild	normal	strong	Yes

- Gini(outlook=sunny & wind=weak) =

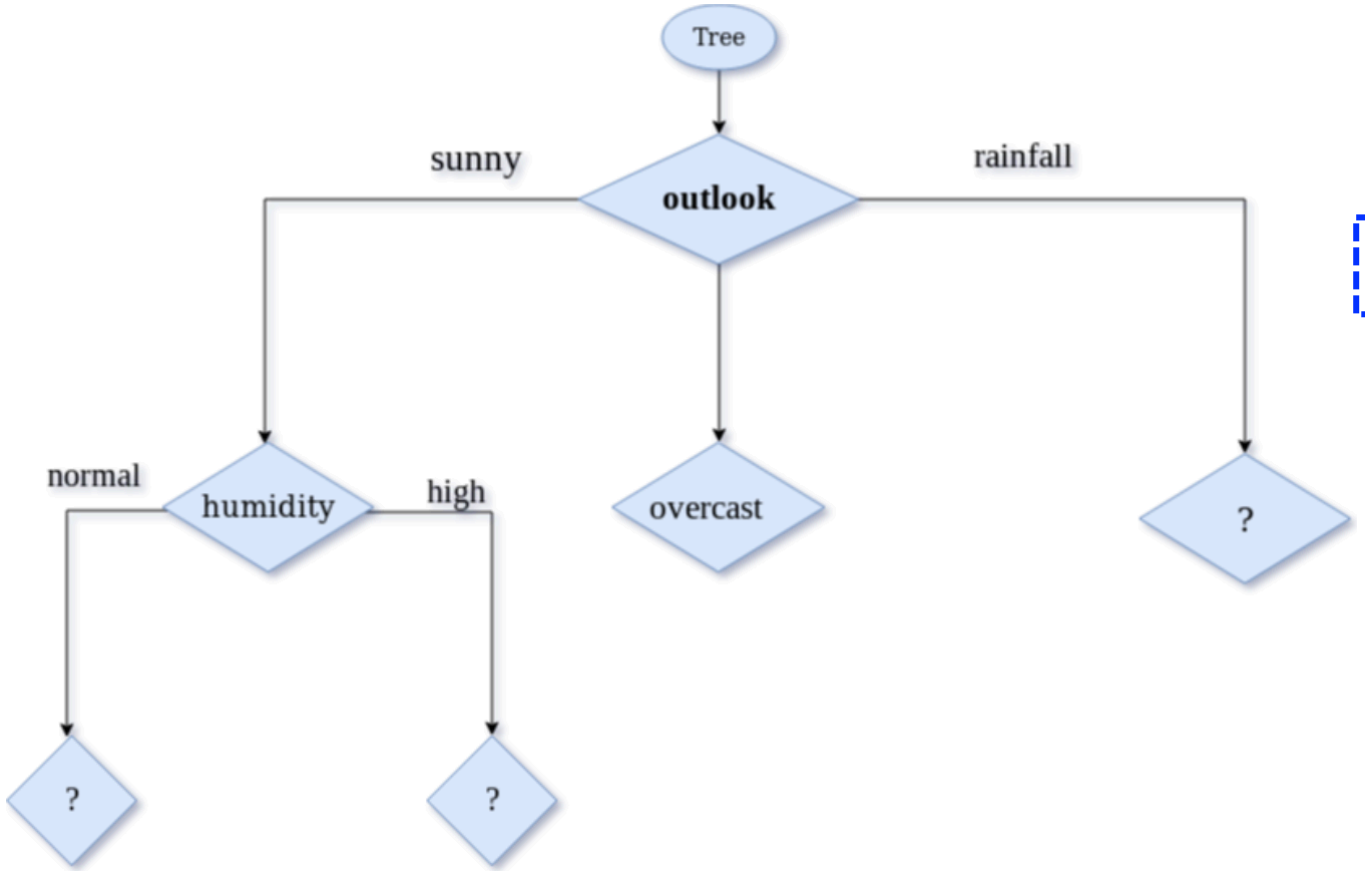
  $1 - (1/3)^2 - (2/3)^2 = 0.44$
- Gini(outlook=sunny & wind=weak) =

  $1 - (1/2)^2 - (1/2)^2 = 0.5$
- Gini(outlook=sunny & wind) =

  $(3/5)*0.44 + (2/5)*0.5 = 0.466$

Decision Tree Learning

- Decision on ‘sunny’ outlook factor



Features	Gini Index
temperature	0.2
humidity	0
wind	0.466

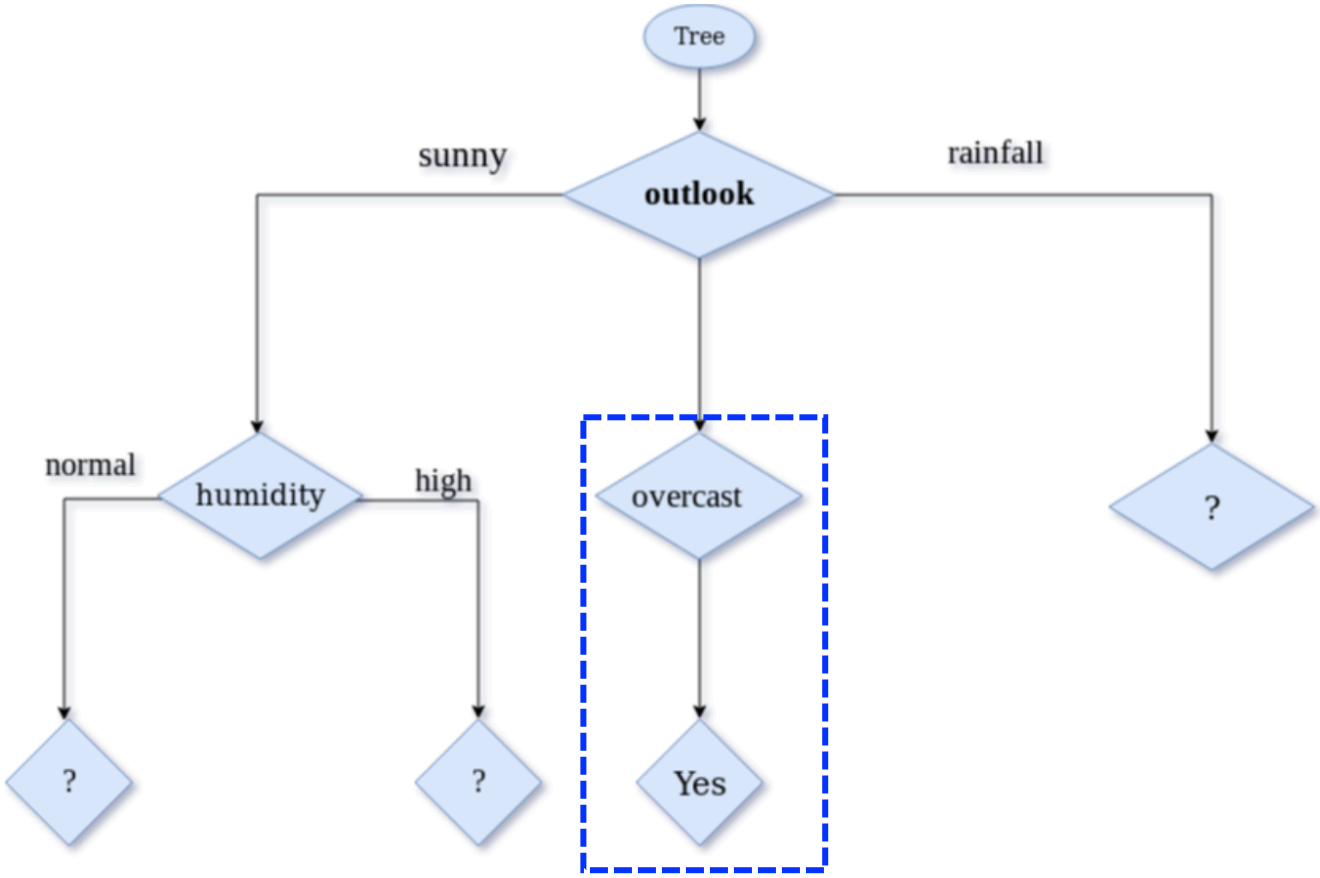
Decision Tree Learning

- Focus on sub data for overcast outlook feature:

Day	outlook	temperature	humidity	wind	decision
3	overcast	hot	high	weak	Yes
7	overcast	cool	normal	strong	Yes
12	overcast	mild	high	strong	Yes
13	overcast	hot	normal	weak	Yes

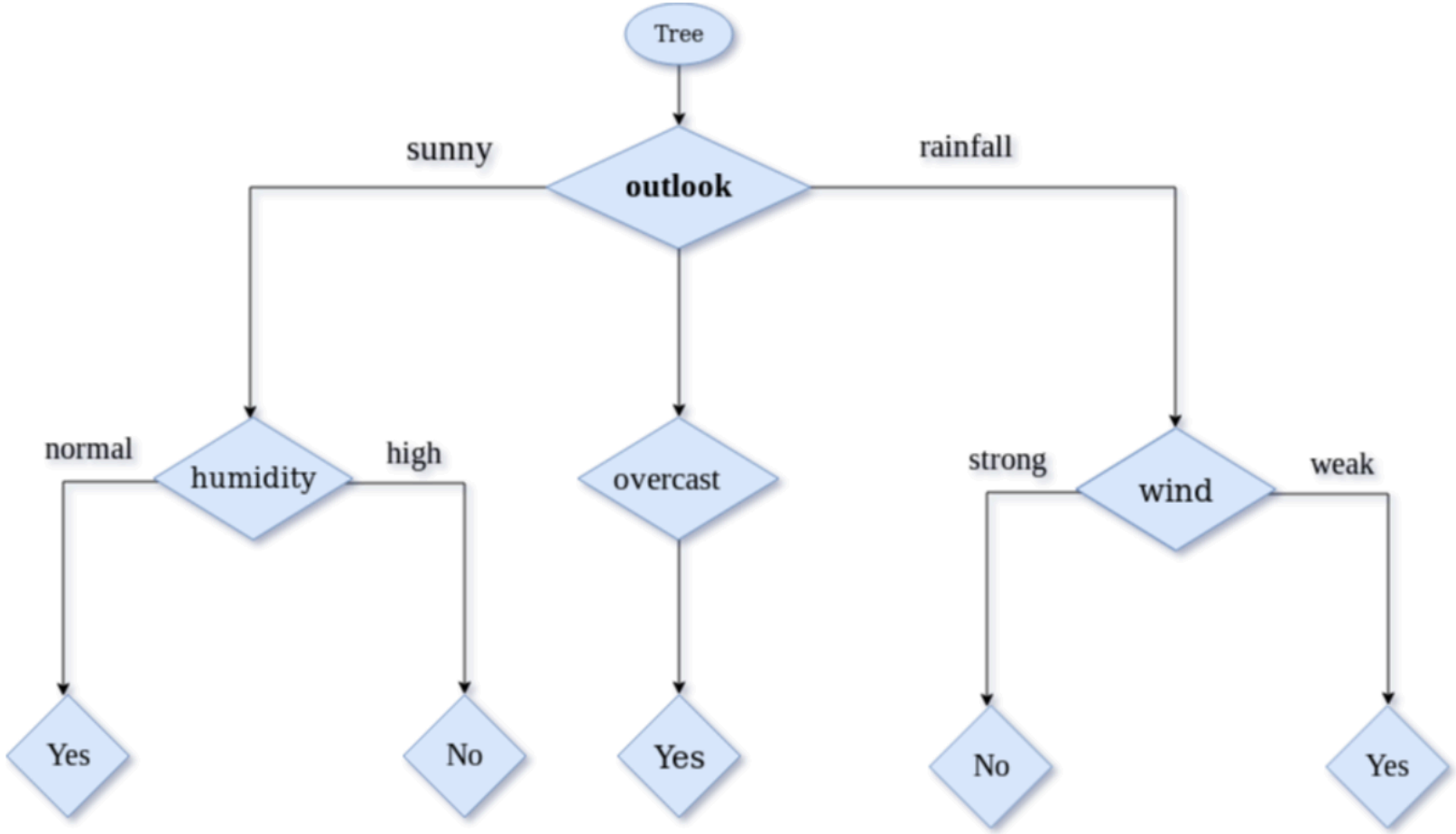
Decision Tree Learning

- Focus on sub data for overcast outlook feature:



Decision Tree Learning

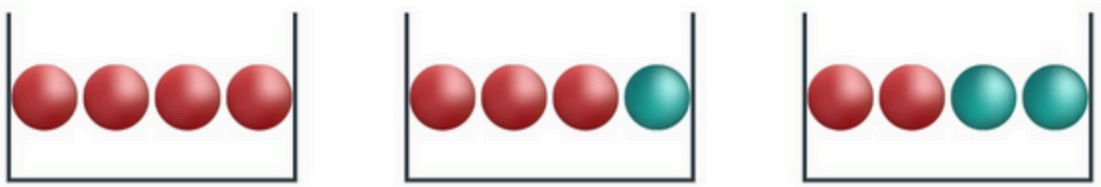
- Focus on sub data for overcast outlook feature:



Decision Tree Learning

- Entropy – a measure of randomness of a system
  - ✓Sample is homogeneous → Entropy is 0

$$E(S) = -\sum_{i=1}^{n} P_i \times \log(P_i)$$



Entropy < Entropy < Entropy

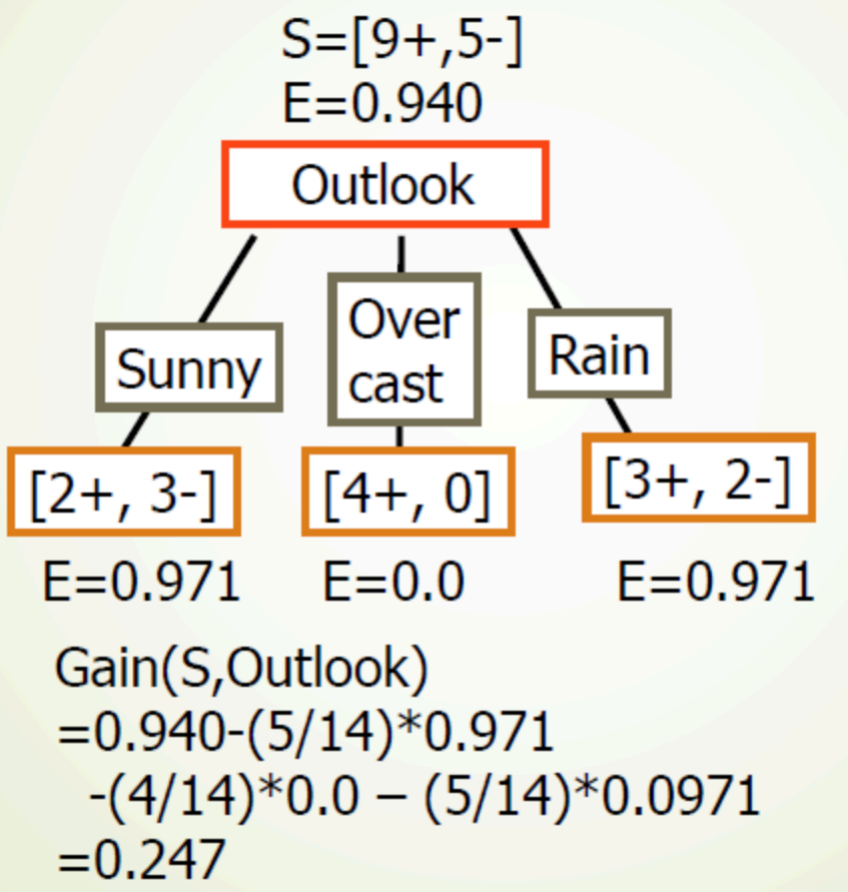
Decision Tree Learning

- Information Gain – amount by which the Entropy of the system reduces due to the split
- How do you decide based on which feature you should split the data?

$$\text{Gain(S, A)} = \text{E(S)} - \sum_{v \in D_A} \frac{|S_v|}{|S|} \text{E}(S_v)$$

Decision Tree Learning

- 9 Yes, 5 No



Day	outlook	temperature	humidity	wind	Decision
1	sunny	hot	high	weak	No
2	sunny	hot	high	strong	No
3	overcast	hot	high	weak	Yes
4	rainfall	mild	high	weak	Yes
5	rainfall	cool	normal	weak	Yes
6	rainfall	cool	normal	strong	No
7	overcast	cool	normal	wtrong	Yes
8	sunny	mild	high	weak	No
9	sunny	cool	normal	weak	Yes
10	rainfall	mild	normal	weak	Yes
11	sunny	mild	normal	strong	Yes
12	overcast	mild	high	strong	Yes
13	overcast	hot	normal	weak	Yes
14	rainfall	mild	high	strong	No

Decision Tree Learning

- Decision for root node:

  Take the highest information gain and repeat the procedure

  Gain(S, Outlook) =0.247

  Gain(S, Humidity) =0.151

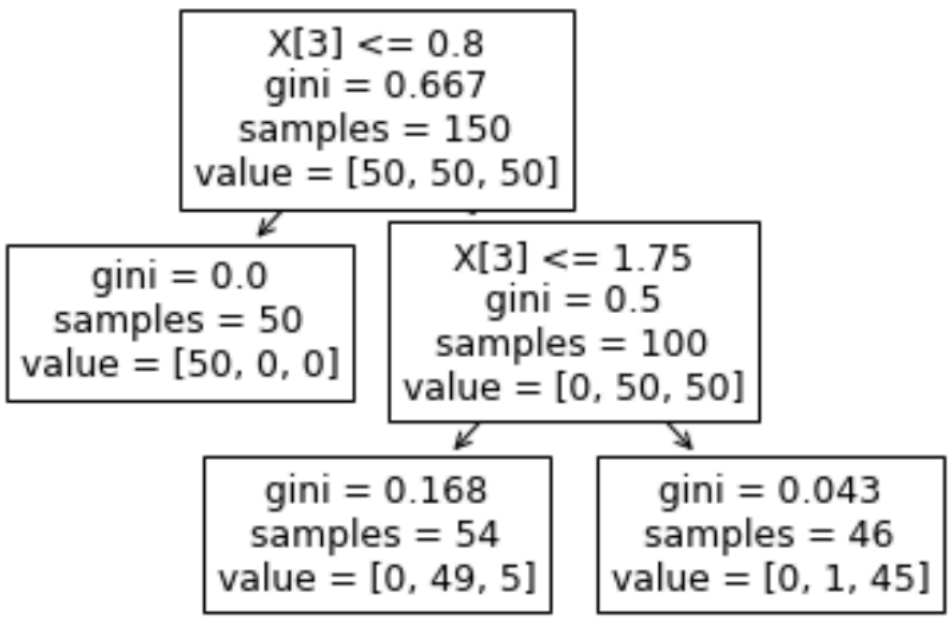
  Gain(S, Wind) =0.048

  Gain(S, Temperature) =0.029

Building a Decision Tree

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
from sklearn import tree

iris = load_iris()
decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
decision_tree = decision_tree.fit(iris.data, iris.target)
r = export_text(decision_tree, feature_names=iris['feature_names'])
print(r)
tree.plot_tree(decision_tree)
```

```
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |--- petal width (cm) >  1.75
|   |   |--- class: 2
```



Building a Decision Tree

- http://127.0.0.1:8888/notebooks/work/python-machine-learning-book-3rd-edition-master/ch03/ch03.ipynb#Building-a-decision-tree

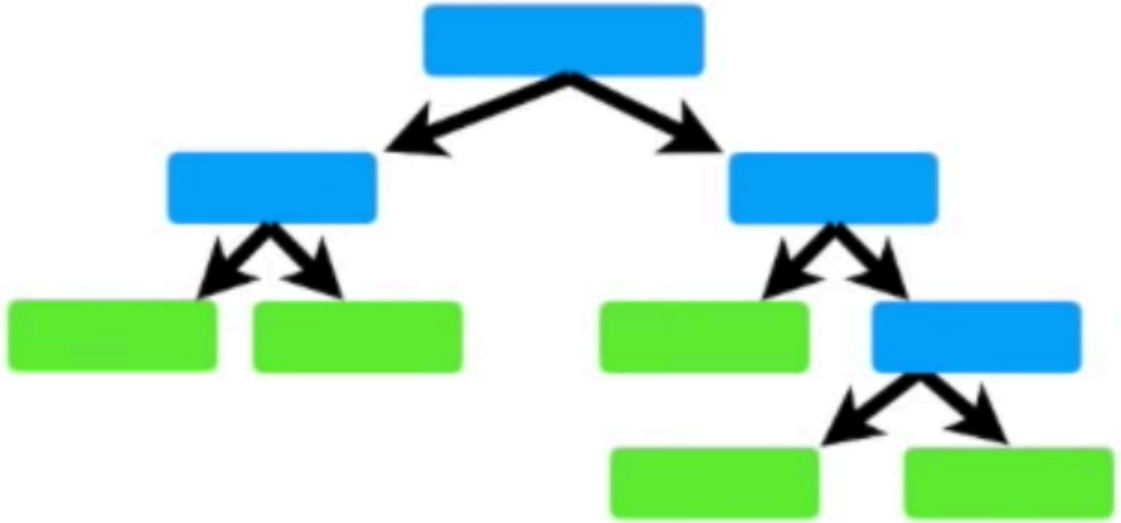
```
tree_model = DecisionTreeClassifier(criterion='gini',
                                    max_depth=4,
                                    random_state=1)
tree_model.fit(X_train, y_train)
```

Decision Trees via Random Forests

Combining Multiple Decision Trees via Random Forests

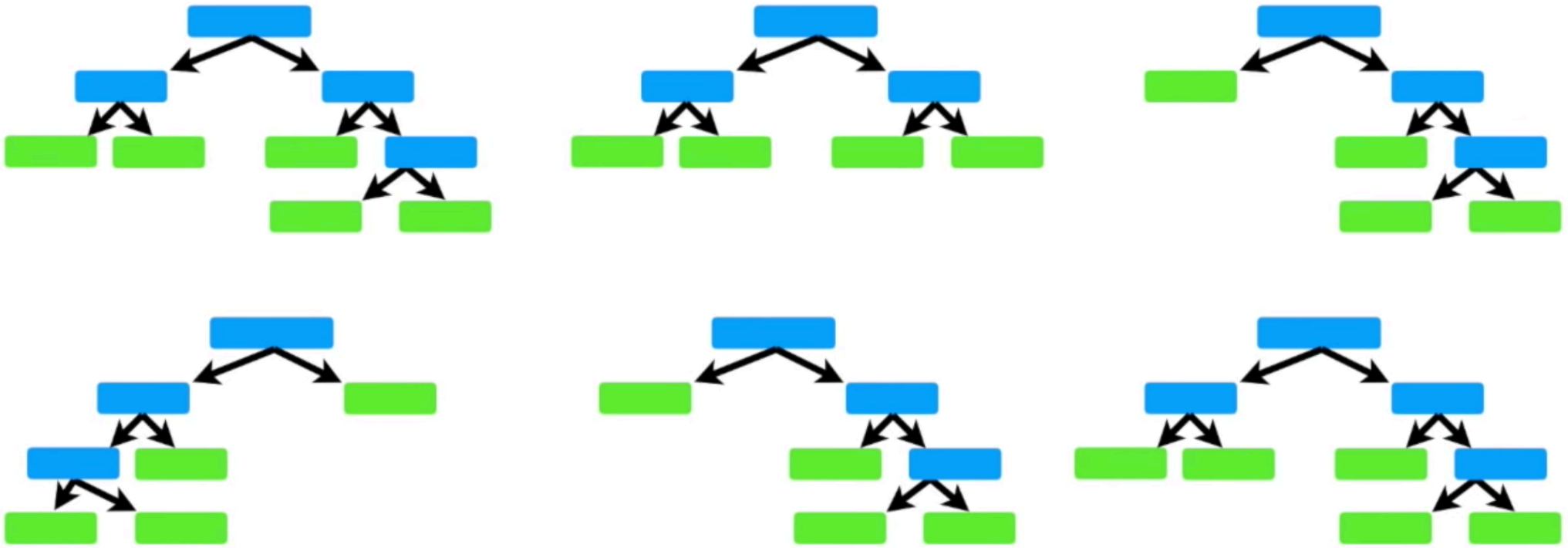
- Decision Tree – work great with data used to create it

  But not flexible for classifying new samples



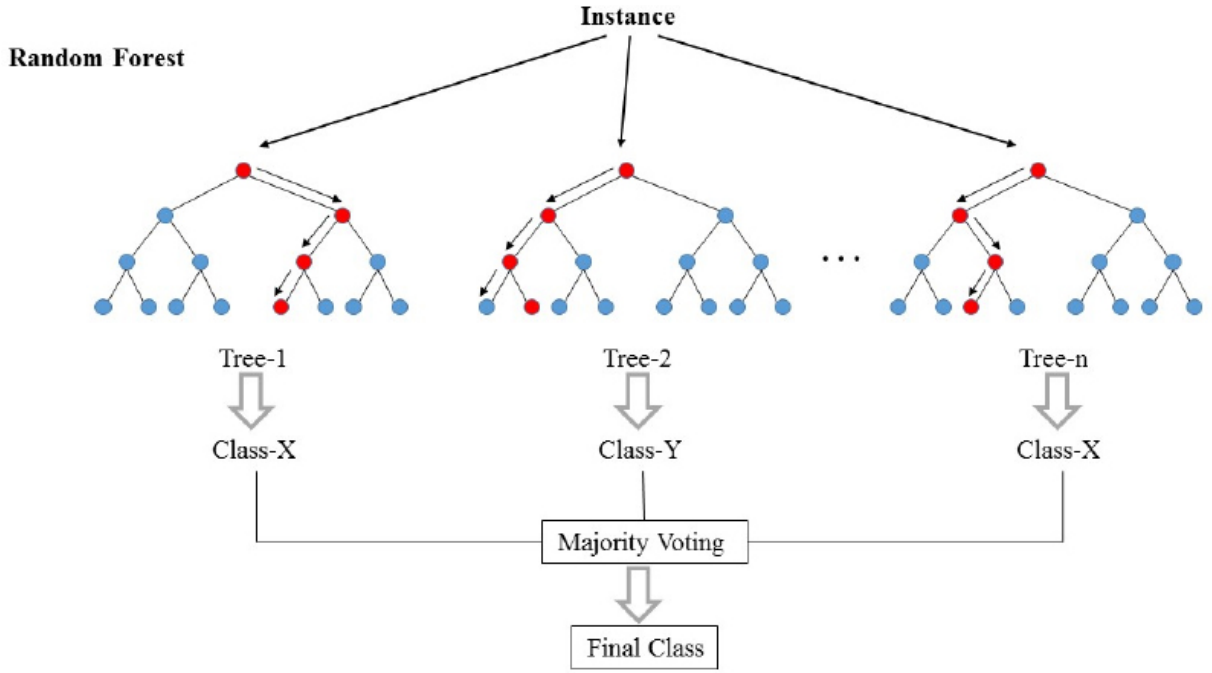
Combining Multiple Decision Trees via Random Forests

- Random Forests (RF) combines simplicity of decision trees with flexibility → improvement in accuracy



Combining Multiple Decision Trees via Random Forests

- Random forest - builds multiple decision trees and merges them together to get a more accurate and stable prediction
- Final prediction is done by taking a vote from all *k* trees - For classification and regression problems



Algorithm of Random Forests

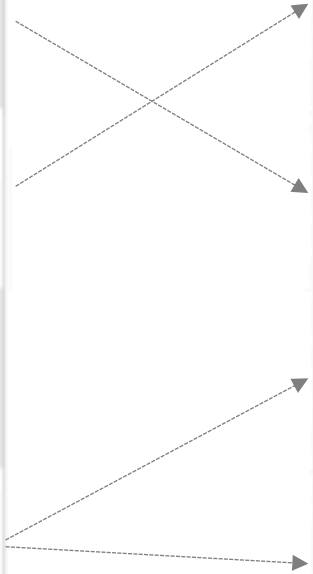
1. Create a random bootstrap sample dataset (sized *n*)
2. Create a decision tree from the bootstrap sample dataset. At each node:
   - ✓Randomly select *d* features w/o replacement
   - ✓Split the node using the feature that provides the best split according to the objective function
3. Repeat steps 1~2 *k* times
4. Aggregate prediction by each tree to assign the class label by majority vote

Step 1. Create a random bootstrap sample dataset

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Randomly select from original dataset



Bootstrapped Dataset

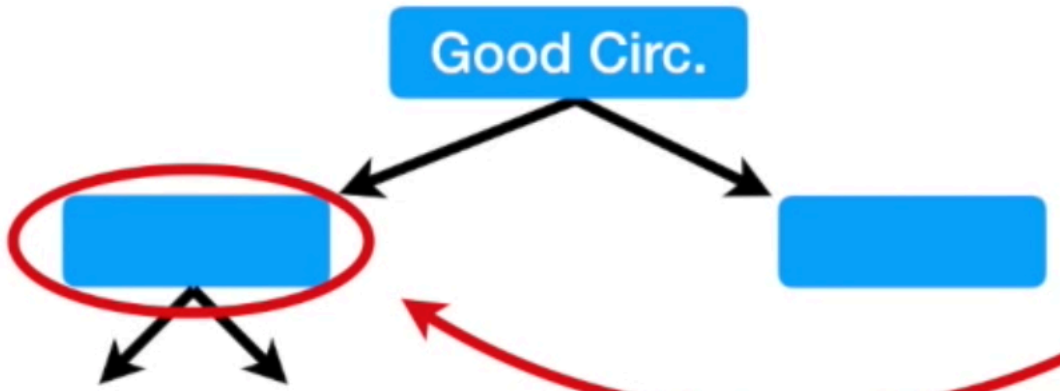
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Step 2. Create a Decision Tree from the Bootstrap Sample Dataset

- Selected 'Good Blood Circ.' and 'Blocked Arteries' as candidates for the root node

Step 2. Create a Decision Tree from the Bootstrap Sample Dataset

- Select 2 (f) variables as candidates, instead of all 3 remaining columns

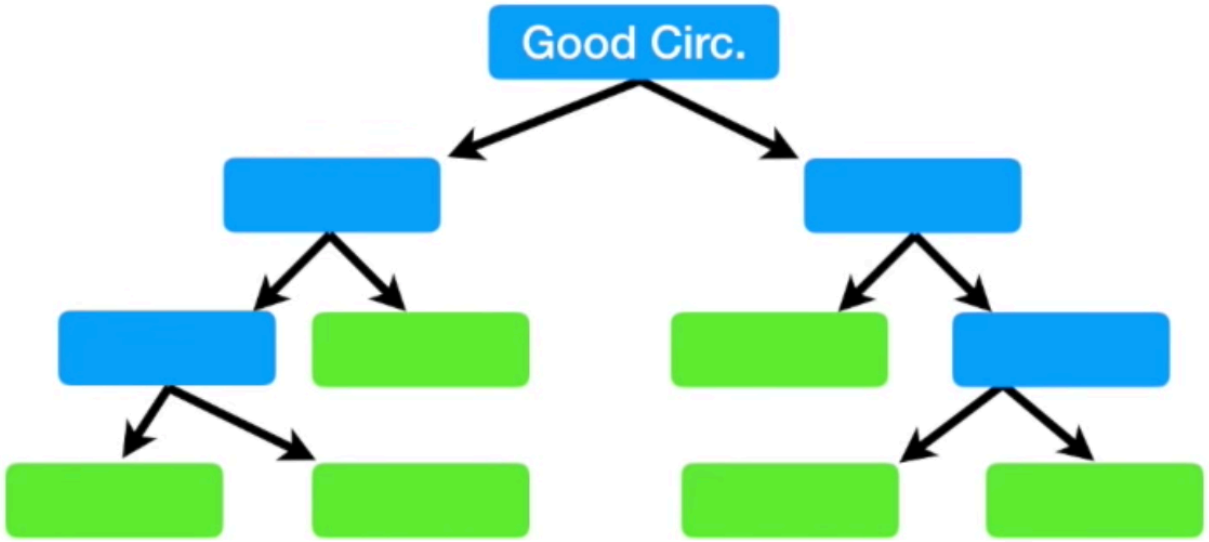


Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No

Step 2. Create a Decision Tree from the Bootstrap Sample Dataset

- Complete building a tree

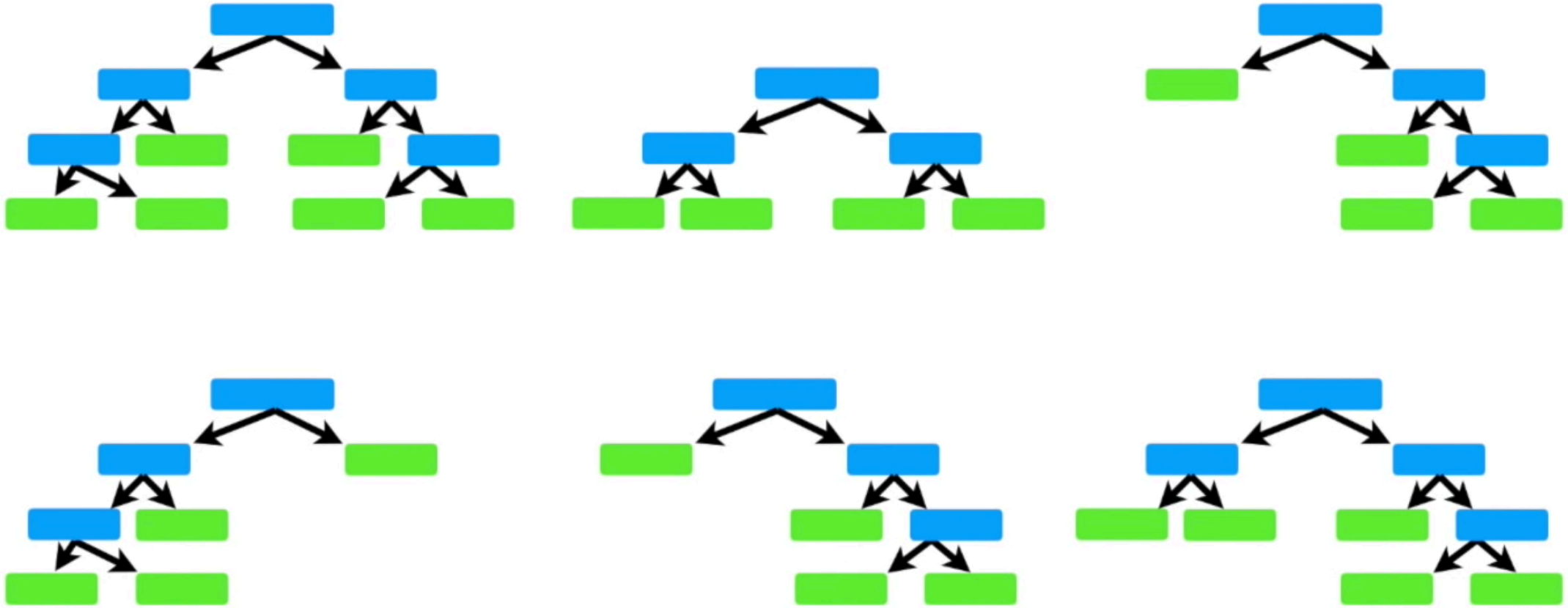


Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

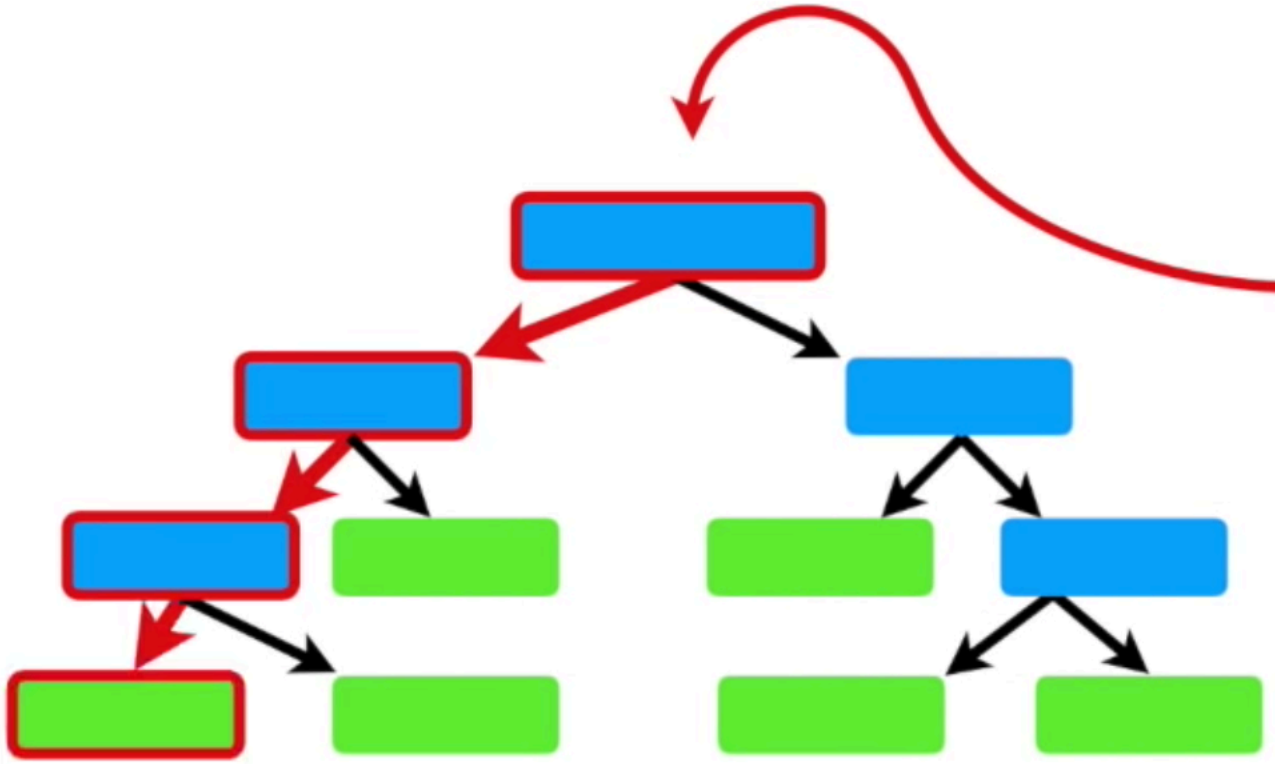
Step 3. Repeat steps 1~2 *k* times

- Results in a wide variety of trees



Step 4. Aggregate prediction by each tree to assign the class label by majority vote

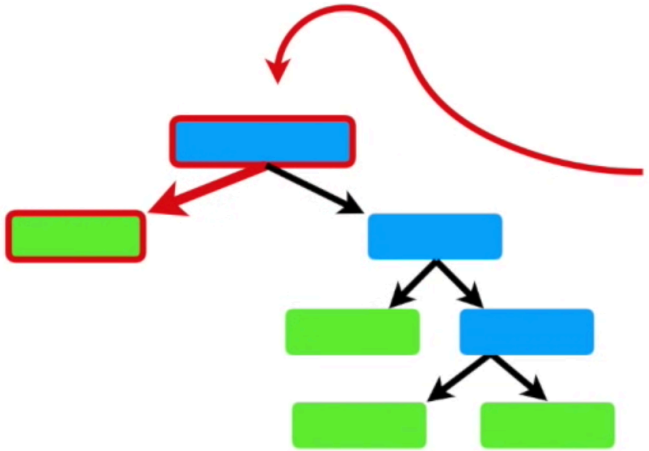
- For a new data



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

Heart Disease	
Yes	No
1	0

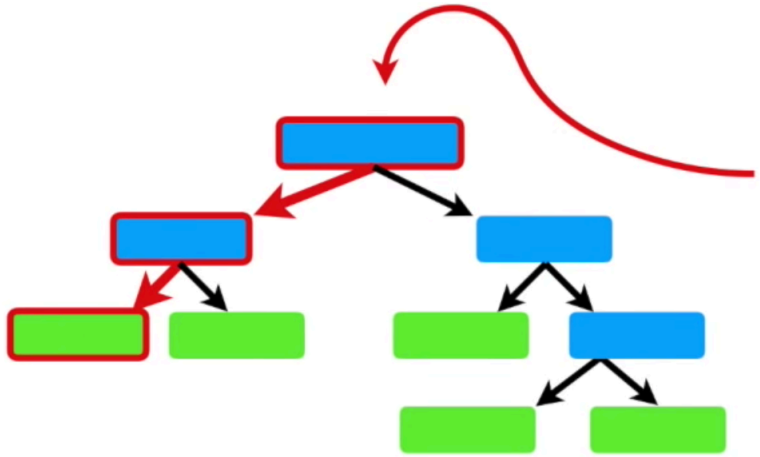
Step 4. Aggregate prediction by each tree to assign the class label by majority vote



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

Heart Disease	
Yes	No
3	0

⋮



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

Heart Disease	
Yes	No
5	1

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	YES

How do we measure accuracy?

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

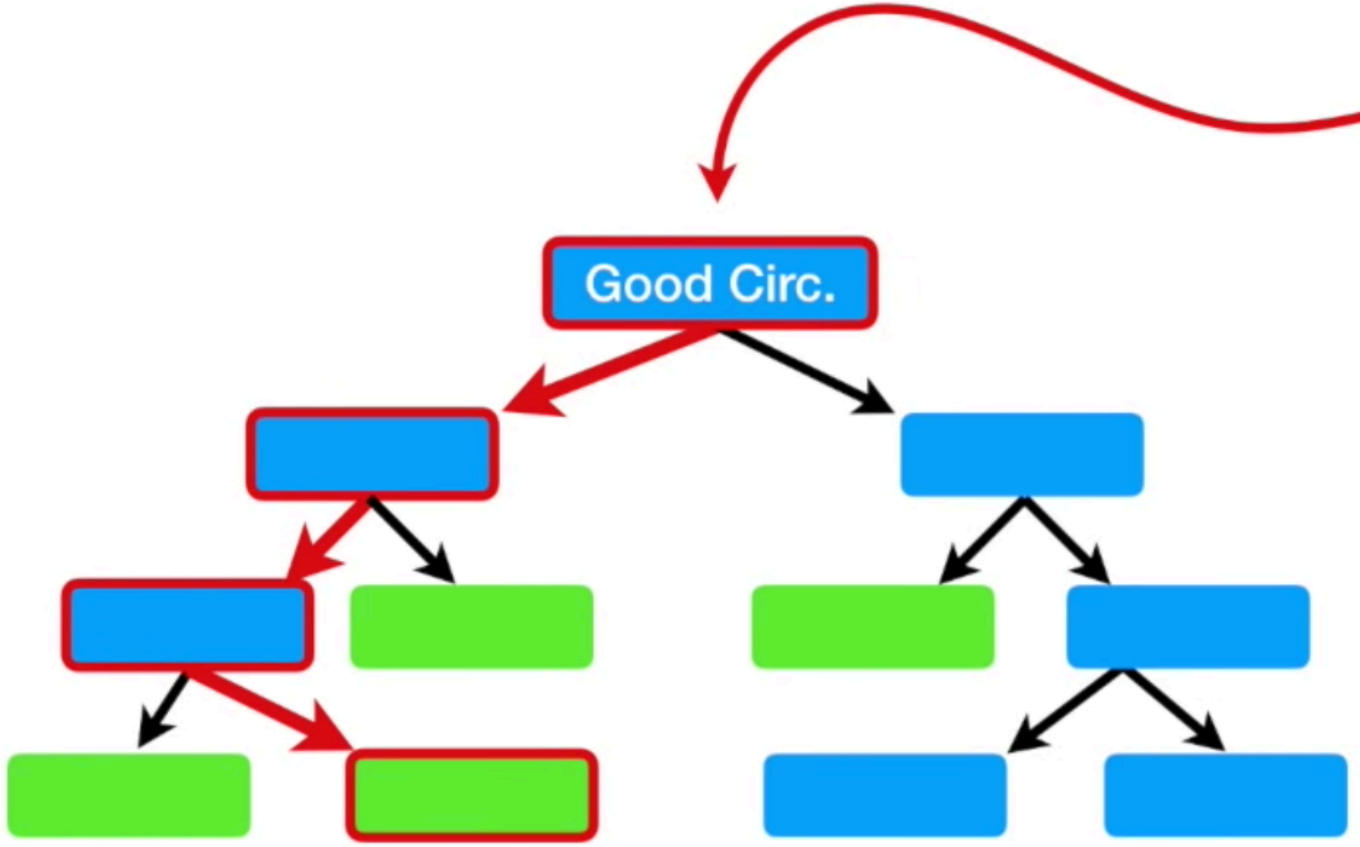
This entry was Not bootstrapped

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

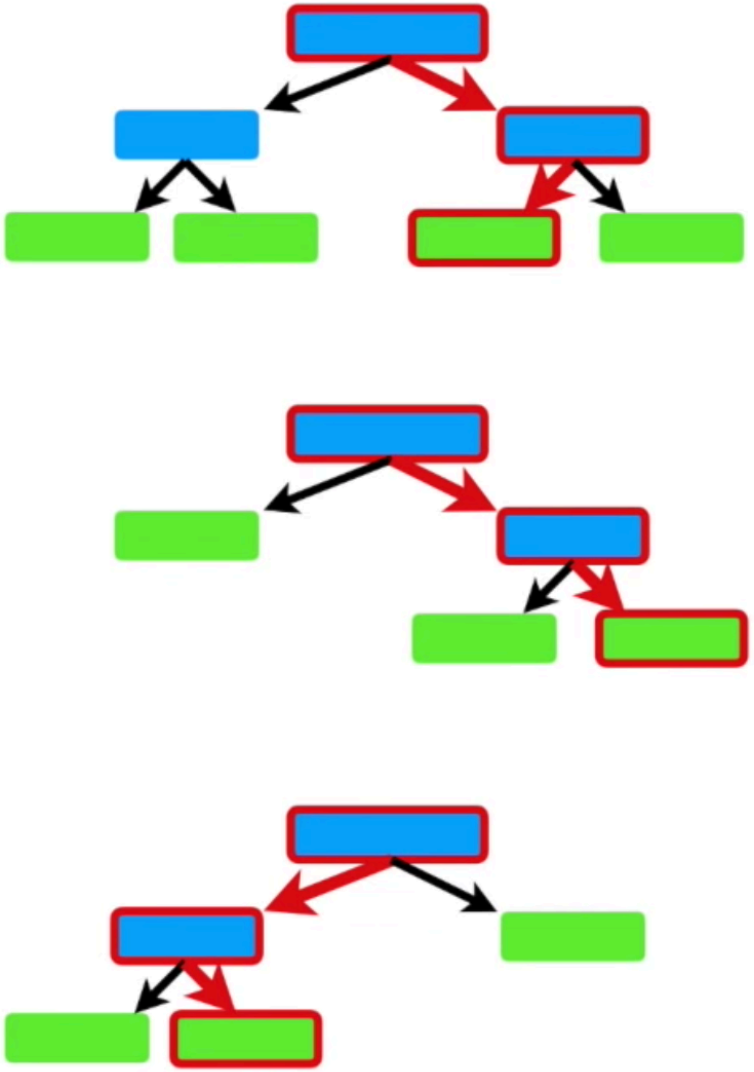
How do we measure accuracy?

- Run it through and see if it correctly classifies the sample as “No”.



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

How do we measure accuracy?



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

Classification of the Out-Of-Bag sample	
Yes	No
1	3

How do we measure accuracy?

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No

Classification of the Out-Of-Bag sample	
Yes	No
3	1

This Out-of-Bag sample was *incorrectly* labeled…

Combining Multiple Decision Trees via Random Forests

- Average multiple decision trees that individually suffer from high variance to build a more robust model → better generalization performance + less susceptible to overfitting
- Parameters to tune for better performance
  - ✓Size of bootstrap sample (n) : control bias-variance tradeoff
  - ✓# of features (d)
  - ✓# of trees (k) : the larger the better for performance (with increased computational cost)

Combining Multiple Decision Trees via Random Forests

- Size of bootstrap sample (*n*)
  - ✓Decreasing n → increases diversity (prob of including a particular example is lowered)→ increases randomness → reduce overfitting
  - ✓Decreasing n → lower overall performance
  - ✓*RandomForestClassifier* in scikit-learn: size *n* = # of training examples in original
- # of features (*d*)
  - ✓A reasonable default in scikit-learn : $d = \sqrt{m}$, where *m* is # of features in training dataset
- # of trees (*k*)

K-Nearest Neighbors