WPI

CS 534
Artificial Intelligence

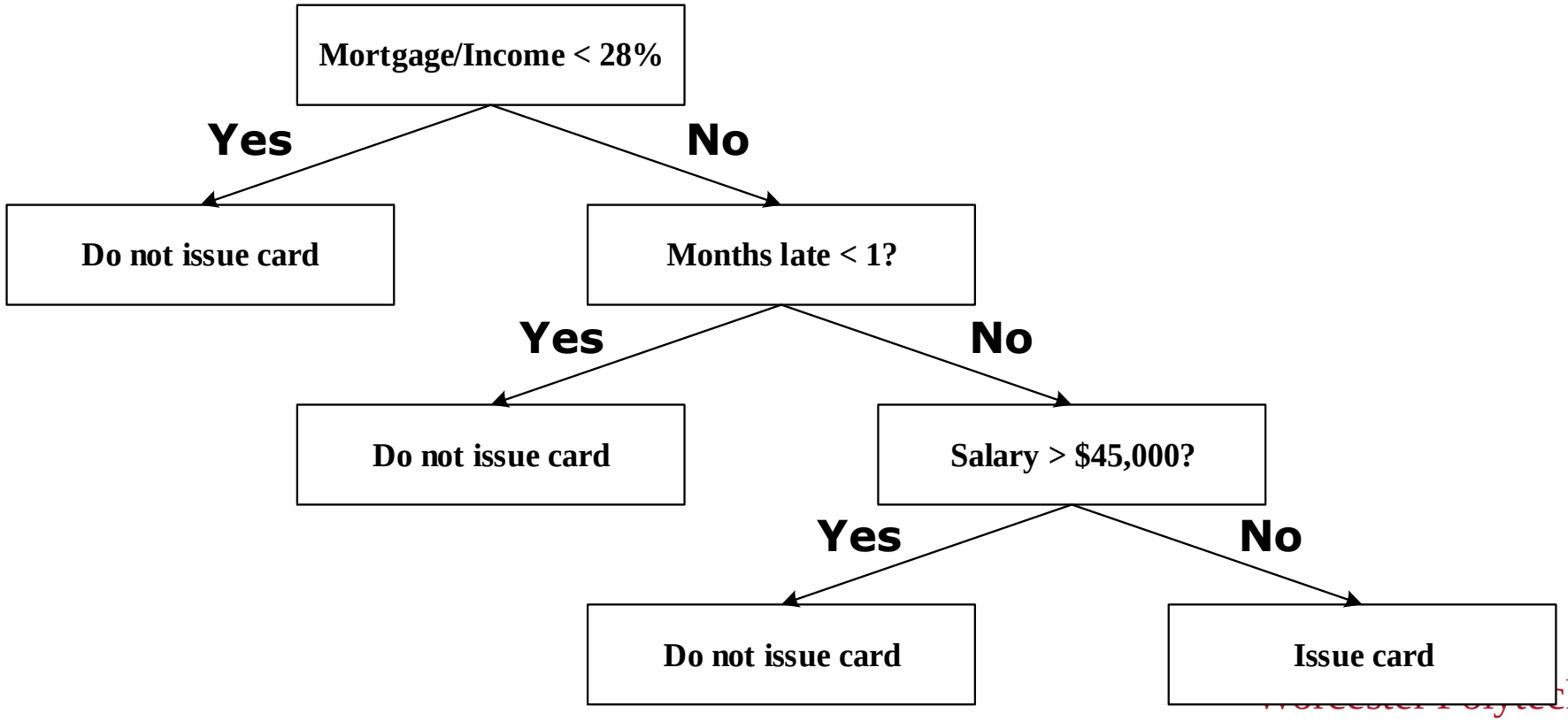
Week 6: Supervised Machine Learning Model II

By

Ben C.K. Ngan

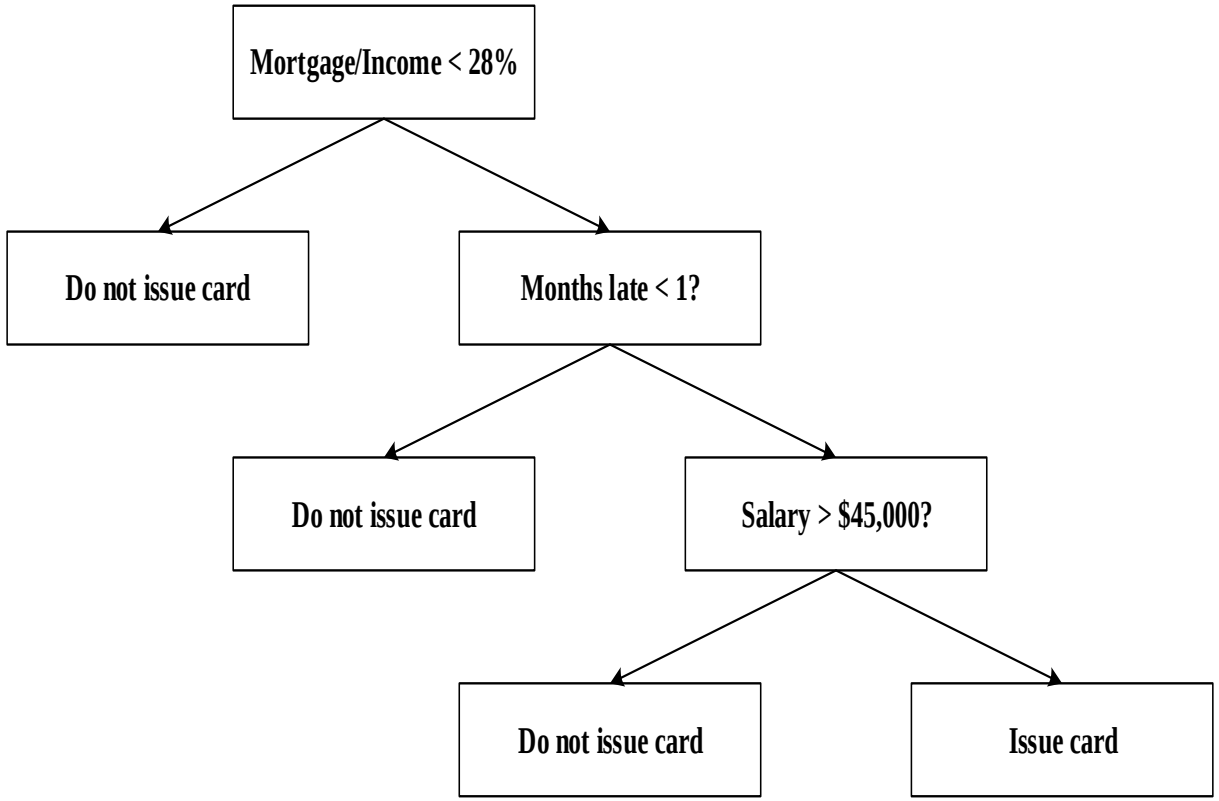
Decision Tree

- What is a decision tree (DT)?
  - A classification model/decision flowchart predicts the value of a target variable by learning simple decision rules inferred from the data attributes/features.
  - Decision rules are **a set of if-then rules**.



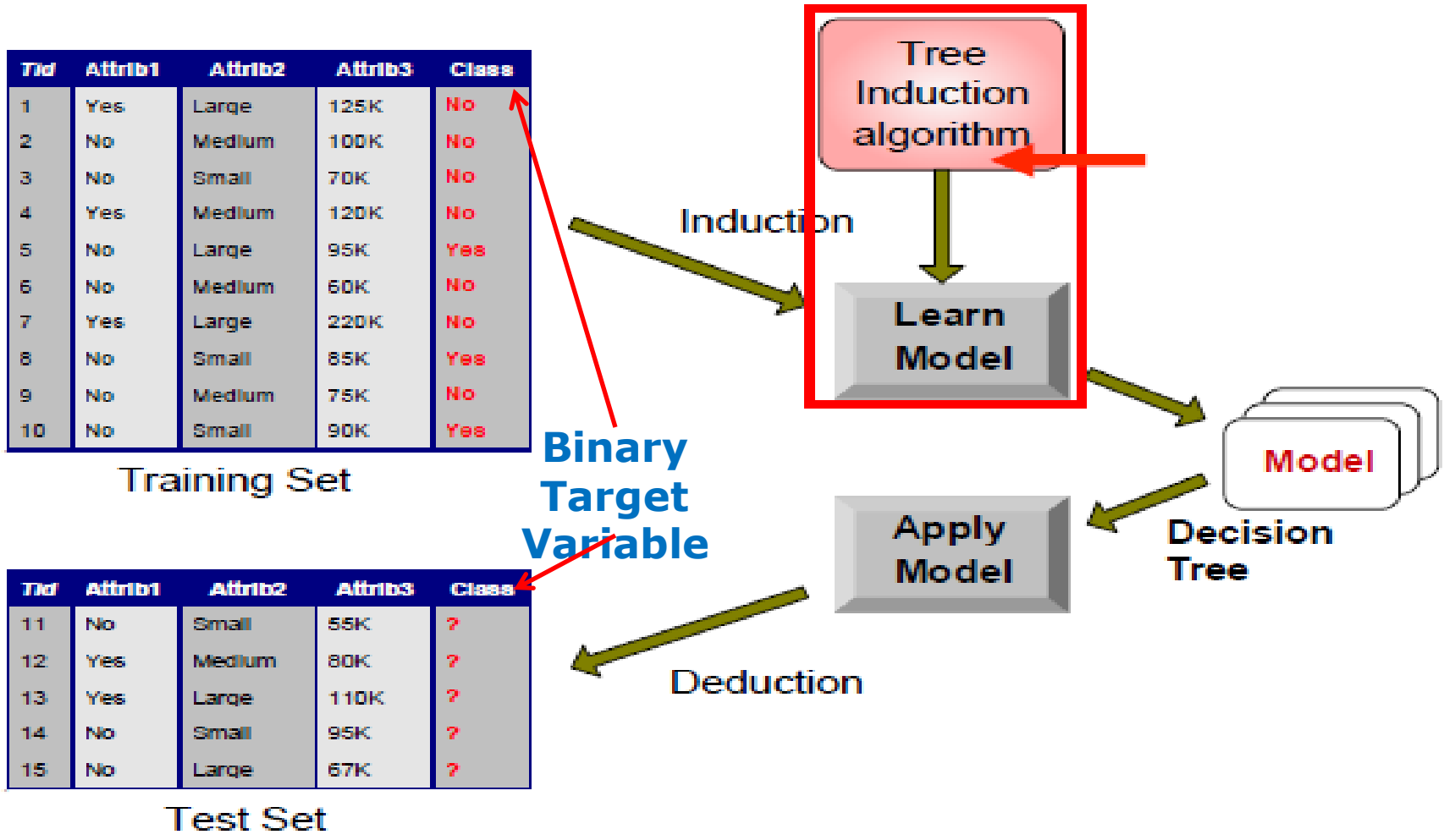
Decision Tree

- Decision Tree Structure
  - A root node
    - No incoming edges
    - Two outgoing edges labeled with **a testing question**
  - Internal nodes
    - Exactly one incoming edge
    - Two outgoing edges labeled with **a testing question**
  - Leaf nodes
    - Exactly one incoming edge
    - No outgoing edges
    - A predicted value assigned to **the target binary variable**



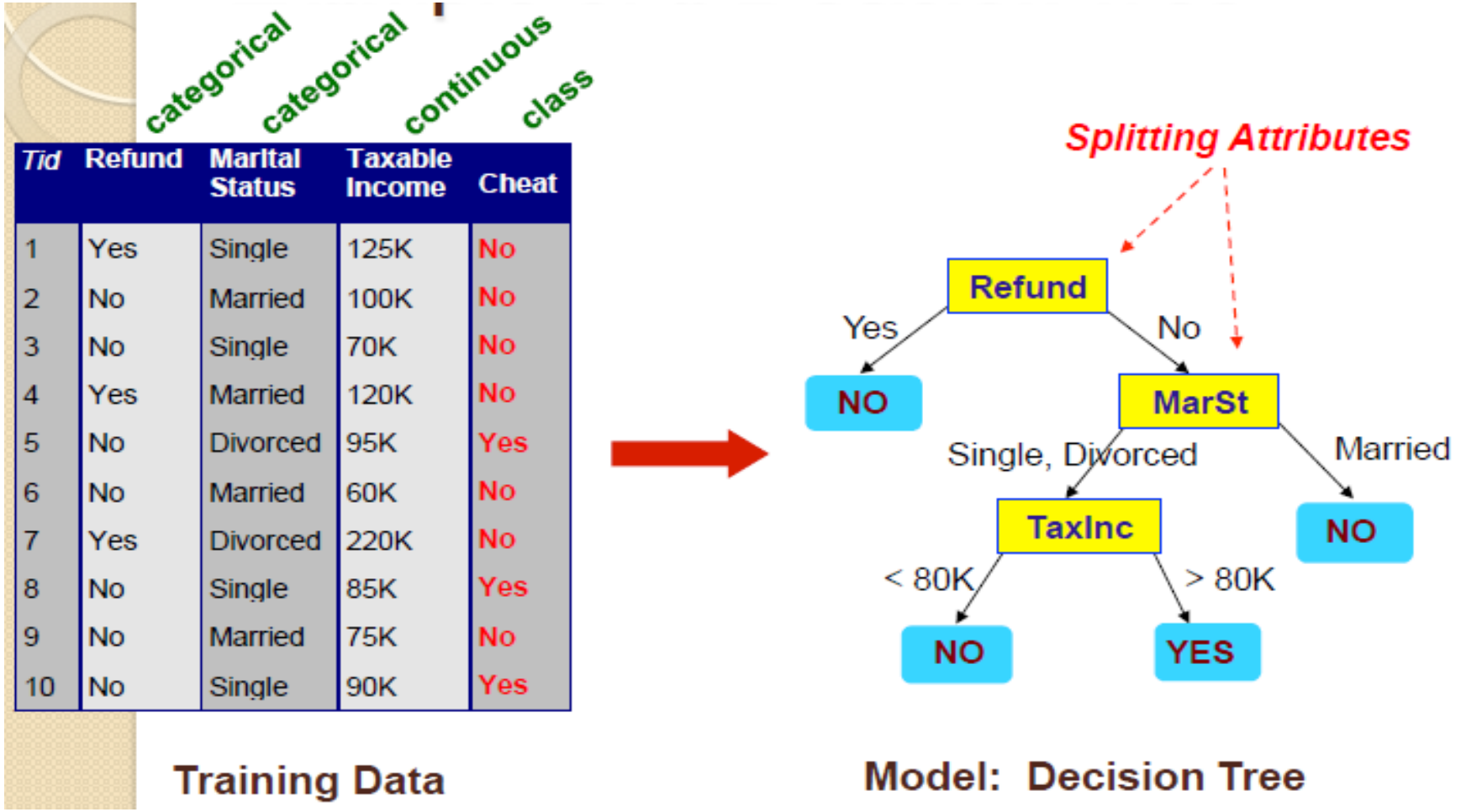
Decision Tree

- Tree Algorithm



Decision Tree

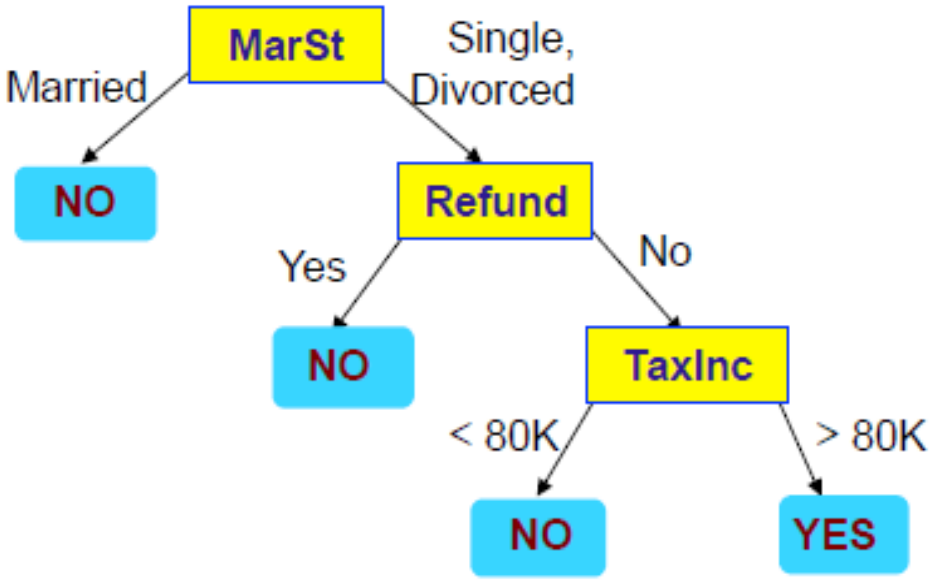
- Example of a Decision Tree Model



Decision Tree

- Example of a Decision Tree Model

	categorical	categorical	continuous	class
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



There could be more than one tree that fits the same data!

Decision Tree

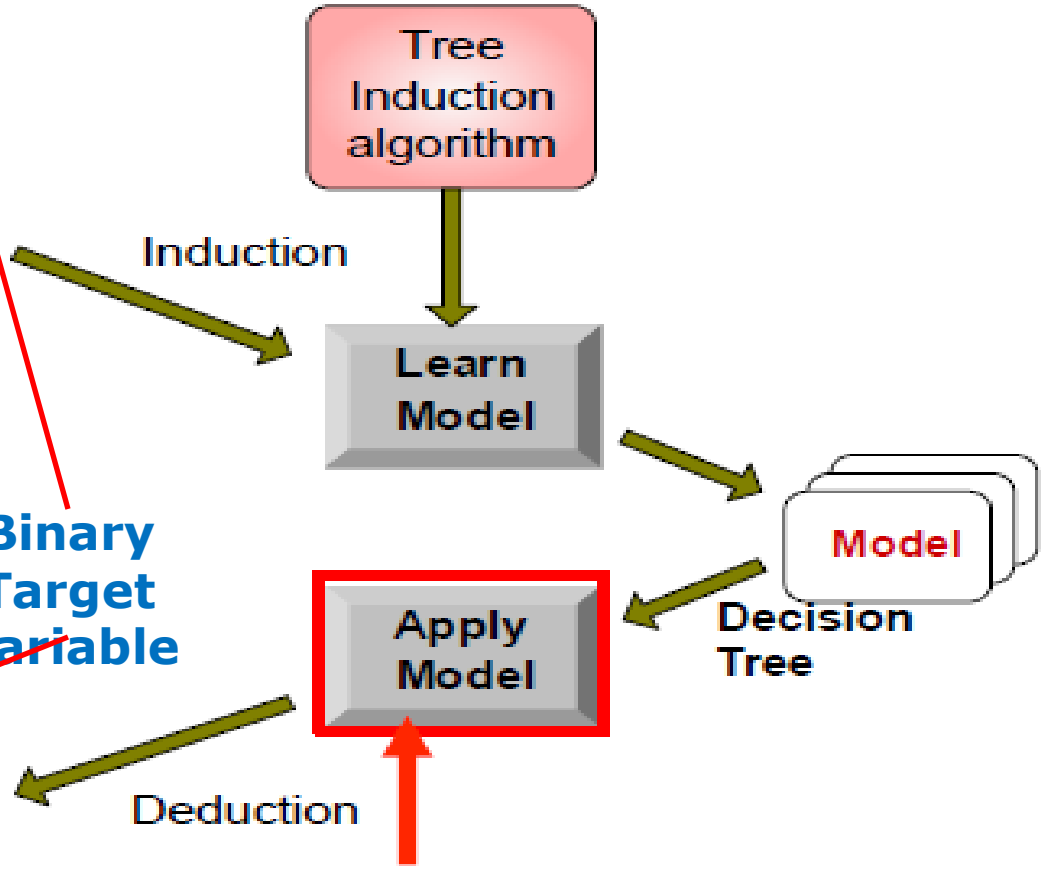
- Tree Algorithm

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

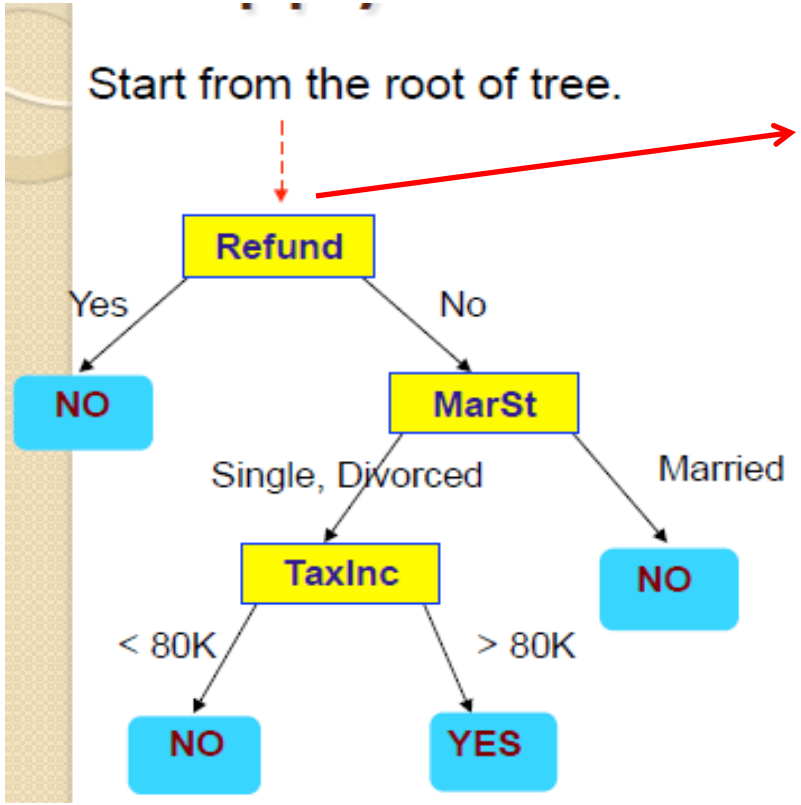
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



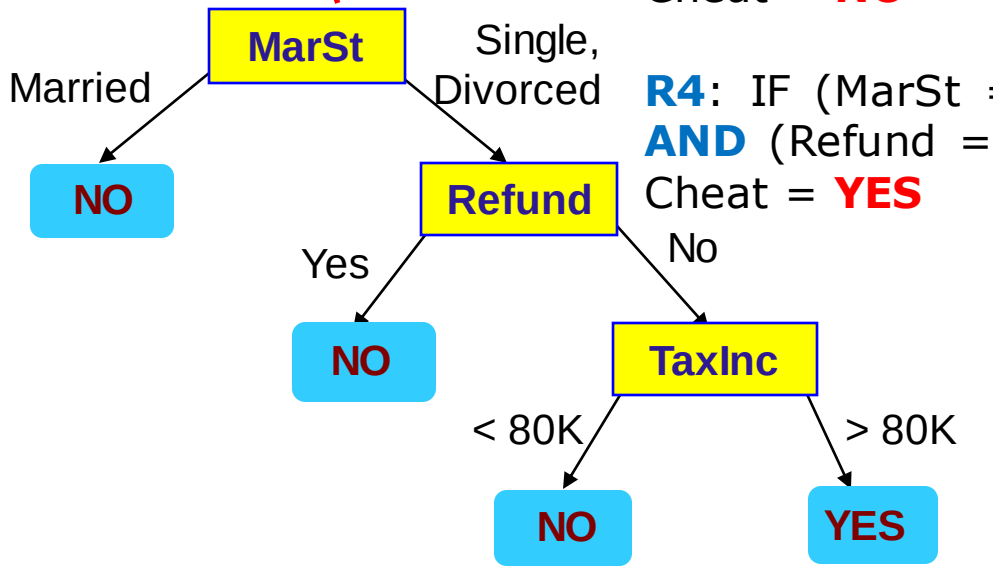
Decision Tree

- Apply Model To Test Data



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Decision Rule: One rule is generated for each leaf {**IF** "condition" **THEN** "result"}

R1: IF (MarSt = Married) THEN Cheat = **NO**

R2: IF (MarSt = Single **OR** MarST = Divorced) **AND** (Refund = Yes) THEN Cheat = **NO**

R3: IF (MarSt = Single **OR** MarST = Divorced) **AND** (Refund = No) **AND** (TaxInc < 80K) THEN Cheat = **NO**

R4: IF (MarSt = Single **OR** MarST = Divorced) **AND** (Refund = No) **AND** (TaxInc > 80K) THEN Cheat = **YES**

Decision Tree

- Constructing and Calibrating the Tree on a Training Sample
  - **Recursive Partitioning Algorithm**

    - Search ***all possible split points*** among ***all possible input variables*** to find the one that splits the cases into the most homogeneous or pure subgroups.
    - This requires a measure of purity. The measure is what is known as the ***Gini Index (Classification And Regression Tree - CART) and Entropy/Information Gain (Iterative Dichotomiser 3 – ID3)***.

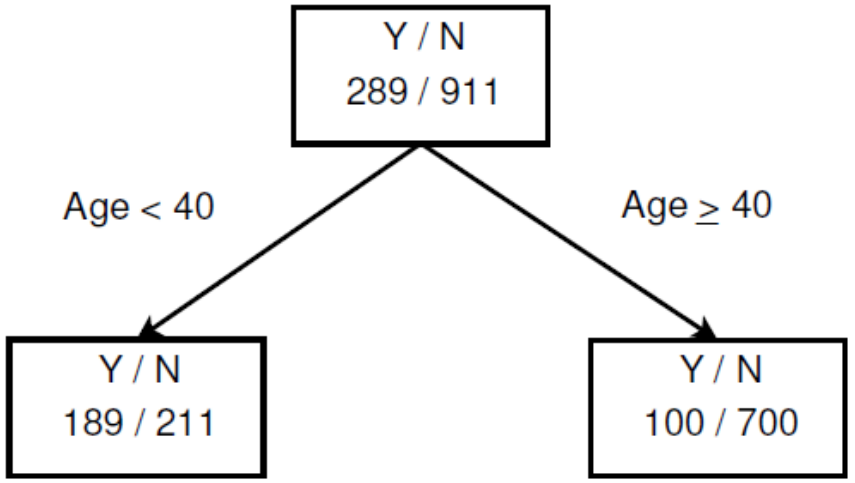
  - **Example**: A bicycle shop has collected information on adult customers entering the store. This information includes the ***customer's age***, ***whether he or she has kids***, and ***whether or not he or she purchased a bicycle (Target Binary Variable)***.

    - 289 buyers and 911 non-buyers
    - The target variable "Buy" takes the values "Yes" or "No"

Decision Tree

- Calibrating the Tree on a Training Sample
  - **Example**: A bicycle shop has collected information on adult customers entering the store. This information includes the ***customer's age***, ***whether he or she has kids***, and ***whether or not he or she purchased a bicycle (Target Binary Variable)***.

    - 289 buyers and 911 non-buyers
    - The target variable "Buy" takes the values "Yes" or "No"



Buy	Age < 40	Age ≥ 40
Yes/No:	189/211	100/700
Total in branch:	400	800
	(47% Yes)	(12.5% Yes)

Decision Tree

- Calibrating the Tree on a Training Sample
  - We need a single number, i.e., ***Gini Index***, that we can use to compare the improvement across all possible splits and use that number to pick the best split.
  - The ***Gini Index*** is calculated for the bicycle shop example (See next slides)
  - $Gini\ Index = 1 - \sum_{i=1}^{n} {p_i}^2$, where p_i is the proportion of the samples that belongs to a Class Label for a particular node.
  - *Gini Index* ranges from 0 to 1, with 0 representing perfect equality and 1 representing perfect inequality.

Decision Tree

- Calibrating the Tree on a Training Sample
  - The ***Gini index*** is calculated as follows for the bicycle shop example:

(1) Compute $\mathbf{P_L}$ and $\mathbf{P_R}$ as the proportions of the original node in the left and right nodes, resulting in:

$\mathbf{P_L}$ = 400/1200 = 0.333

$\mathbf{P_R}$ = 800/1200 = 0.666

Buy	Age $<$ 40	Age $\geq$ 40
Yes/No:	189/211	100/700
Total in branch:	400	800
	(47% Yes)	(12.5% Yes)

(2) Compute the proportion of one type (Yes & No) within the left and right sub-nodes as $\mathbf{p_L}$ and $\mathbf{p_R}$, which gives:

$\mathbf{p_L}$ = 189/400 = 0.4725; $\mathbf{1 - p_L}$ = 211/400 = 1 – 0.4725 = 0.5275

Gini Index (Left Sub-Node: Age < 40) = $[1 - (p_L)^2 - (1 - p_L)^2]$ = $[1 - 0.4725^2 - 0.5275^2] = 0.4985$

$\mathbf{p_R}$ = 100/800 = 0.125; $\mathbf{1 - p_R}$ = 700/800 = 1 – 0.125 = 0.875

Gini Index (Right Sub-Node: Age ≥ 40) = $[1 - (p_R)^2 - (1 - p_R)^2]$ = $[1 - 0.125^2 - 0.875^2] = 0.2188$

(3) Compute the Weighted Sum of the Gini index at Age 40 as the split:

$\mathbf{P_L[1 - (p_L)^2 - (1 - p_L)^2] + P_R[1 - (p_R)^2 - (1 - p_R)^2]}$

= 0.333 * 0.4985 + 0.666 * 0.2188

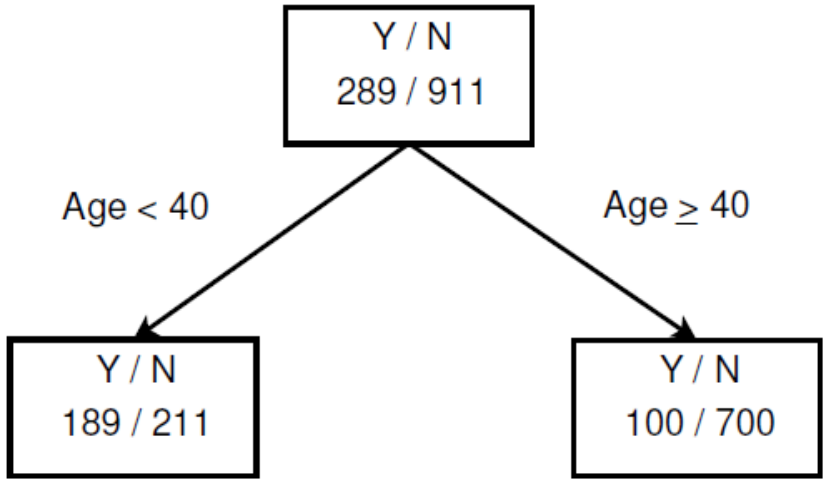
= **0.312**

$$Gini\ Index = 1 - \sum_{i=1}^{n} {p_i}^2$$

Decision Tree

- Calibrating the Tree on an Estimation Sample
  - We can calculate the ***Gini Index*** for all possible splits on all possible input variables.

Possible Split	Gini Index Value
Kids = no, kids = yes	0.321
Age <30, Age ≥ 30	0.910
Age <40, Age ≥ 40	0.312 (The Smallest Gini Index)
Age <50, Age ≥ 50	0.618
Age <60, Age ≥ 60	0.802
Age <70, Age ≥ 70	0.912

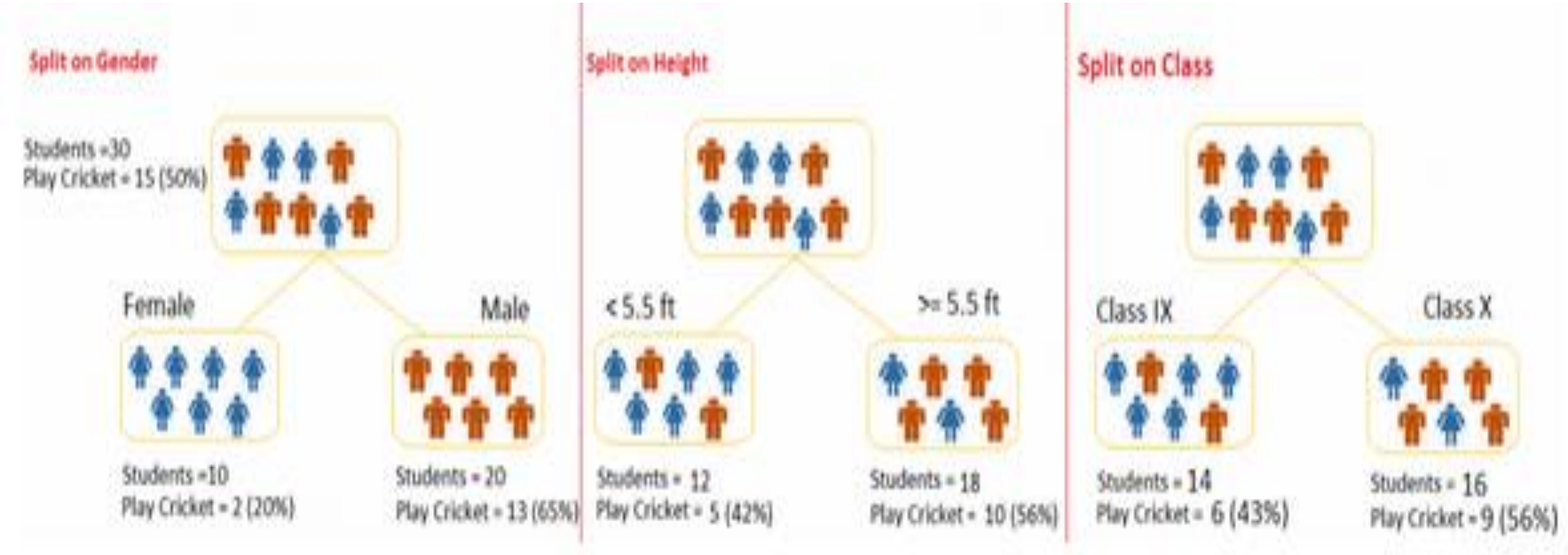


  - Choose the split position that has the lowest Gini Index

Decision Tree

- Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class(IX/X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, **I want to create a model to predict who will play cricket during leisure period?**

- This is where **decision tree** helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other).



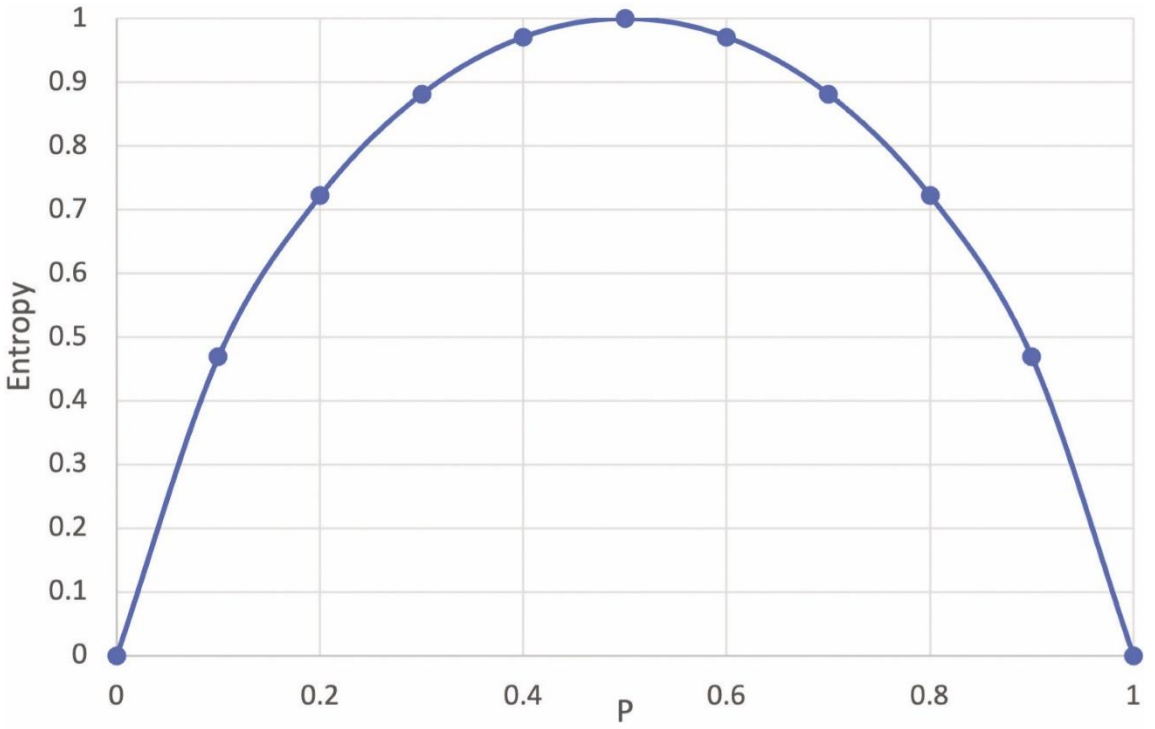
Decision Tree

- Entropy measures the extent of disorder, uncertainly or randomness in a data set.

- Entropy can be calculated using formula, where p and q is probability of success and failure of the decision rule, respectively in that node.

- If the sample is completely homogeneous, then the entropy is **zero** and if the sample is an equally divided (50% – 50%), it has entropy of **one**. See the graph.

$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

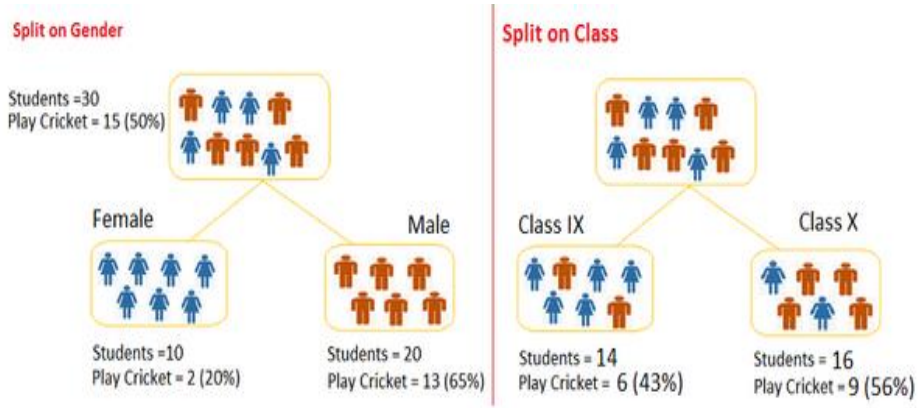


- **Information Gain** = Entropy(parent) – [average Entropy(children)]

Decision Tree

- Steps to calculate entropy for a split:

$$\text{Entropy} = -p \log_2 p - q \log_2 q$$



STEP 1:

Entropy for Gender node = $-(15/30)\log_2(15/30) - (15/30)\log_2(15/30) = 1$

Entropy for Female node = $-(2/10)\log_2(2/10) - (8/10)\log_2(8/10) = 0.72$

Entropy for Male node = $-(13/20)\log_2(13/20) - (7/20)\log_2(7/20) = 0.93$

Entropy for split Gender = Weighted entropy of sub-nodes = $(10/30)*0.72 + (20/30)*0.93 = 0.86$

Information Gain of Gender (IGG) = Entropy(Gender) – [average Entropy(Gender's children)]

Information Gain of Gender (IGG) = 1 - 0.86 = **0.14**

Assume the IG threshold is 0.1

STEP 2:

Entropy for Class node = $-(15/30)\log_2(15/30) - (15/30)\log_2(15/30) = 1$

Entropy for Class IX node = $-(6/14)\log_2(6/14) - (8/14)\log_2(8/14) = 0.99$

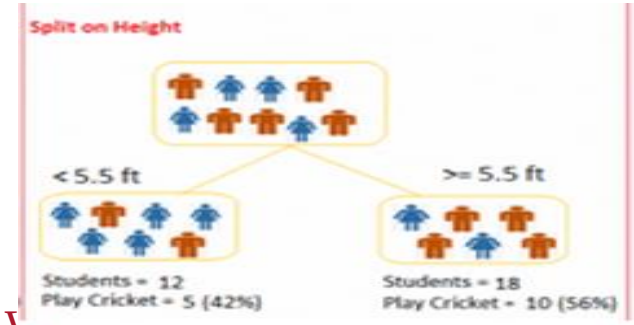
Entropy for Class X node = $-(9/16)\log_2(9/16) - (7/16)\log_2(7/16) = 0.99$.

Entropy for split Class = $(14/30)*0.99 + (16/30)*0.99 = 0.99$

Information Gain of Class (IGC) = Entropy(Class) – [average Entropy(Class's children)]

Information Gain of Class (IGC) = 1 - 0.99 = **0.01**

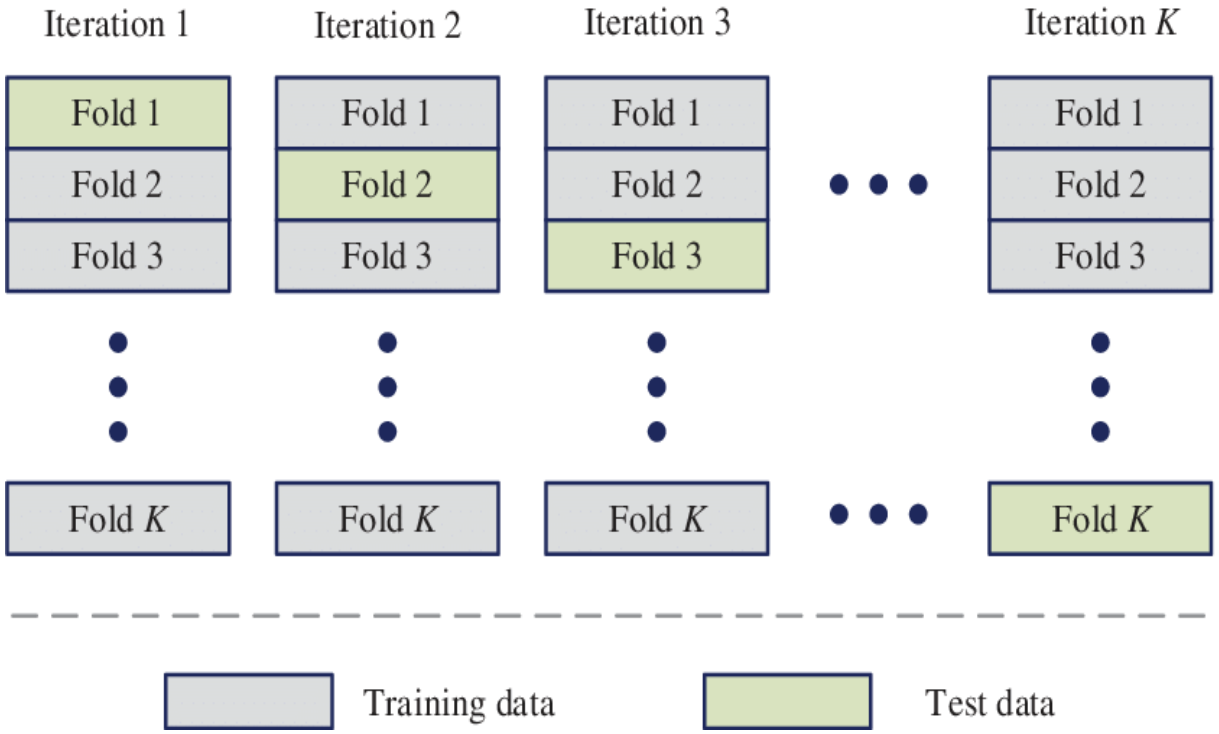
Class Exercise: Compute the information gain for Height using 5.5

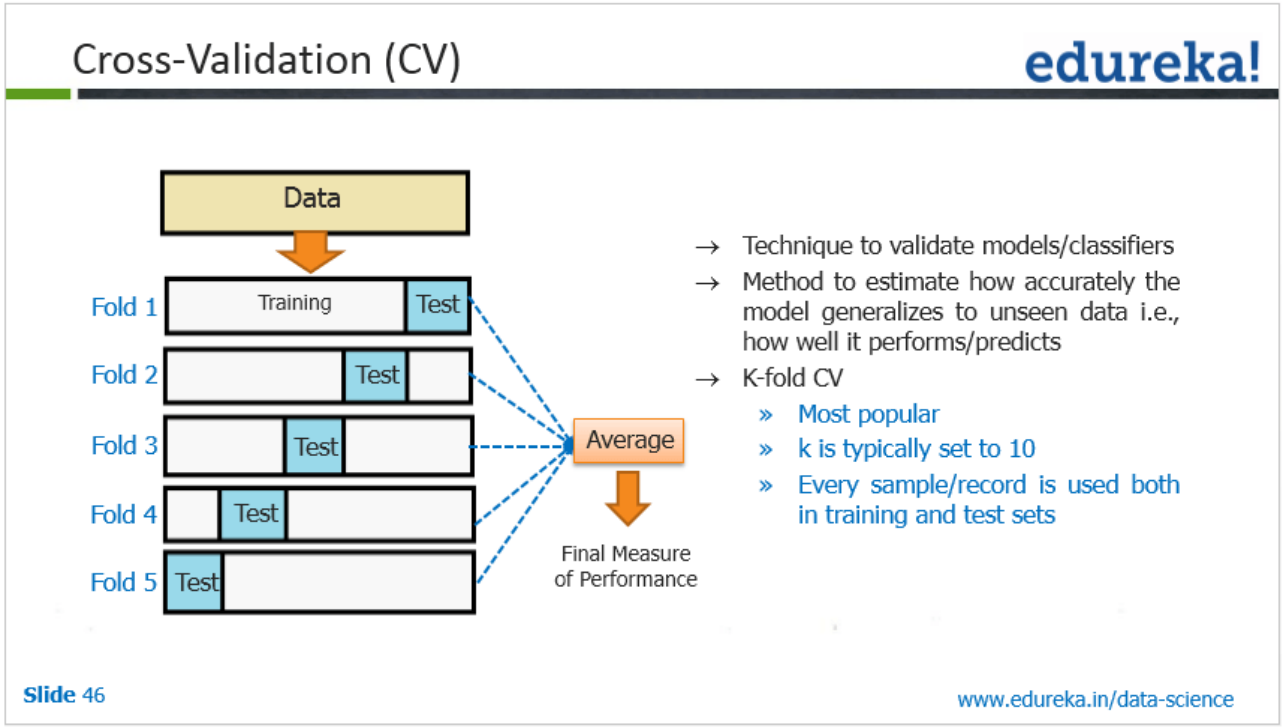


- IGG > IGC, Gender > Class

Decision Tree

- The approached used is to grow a large tree and, at each stage of growth and increasing model complexity, to automatically generate holdout samples as subsets of the training sample. These automatically generated holdout samples are used to validate each of the sequence of tree models.





Decision Tree

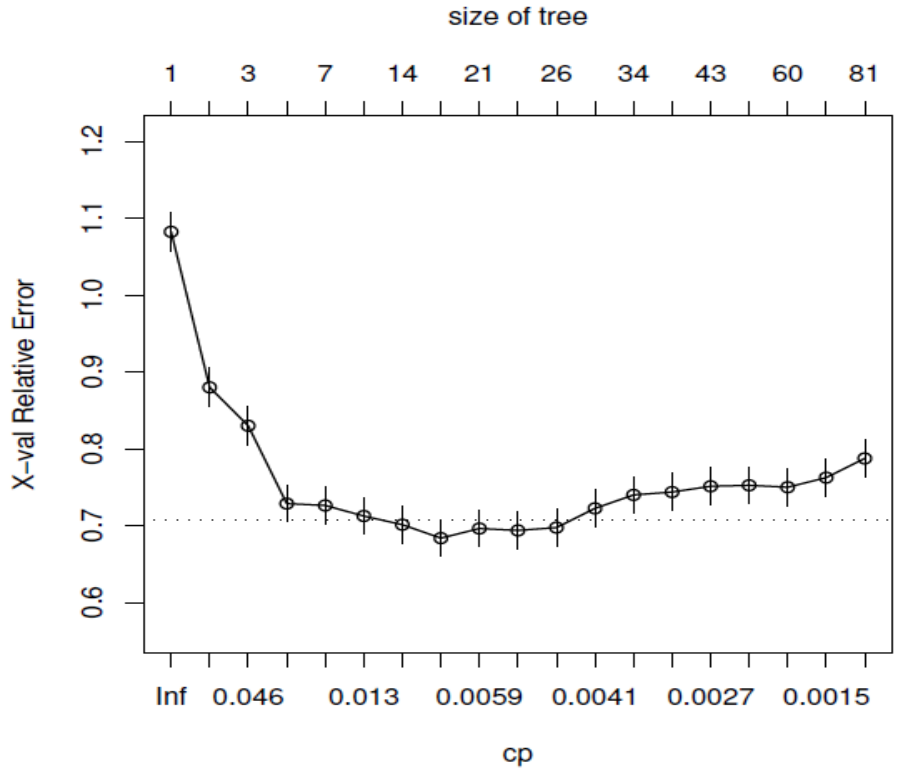
```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)
```

- The vertical axis gives the relative cross-validation error, where "relative" means relative to the root node.
- The horizontal axis is labeled by both the complexity parameter along the bottom and the number of nodes along the top. As the number of nodes, i.e., complexity, increases, the cross-validation error decreases and levels out, and then increases as the model becomes ***ovefitted***.
- The appropriate value of cp can be chosen to generate the final model, i.e., the minimum relative cross-validation error.

The ***stopping rule*** is the ***cost complexity parameter***, ***cp***, i.e., the tree reaches a pre-specified level of complexity as measured by cp.

As the cp increases (decreases), the complexity (or depth) of the tree decreases (increases)

Figure 7.3: A Relative Cross-Validation Error Plot



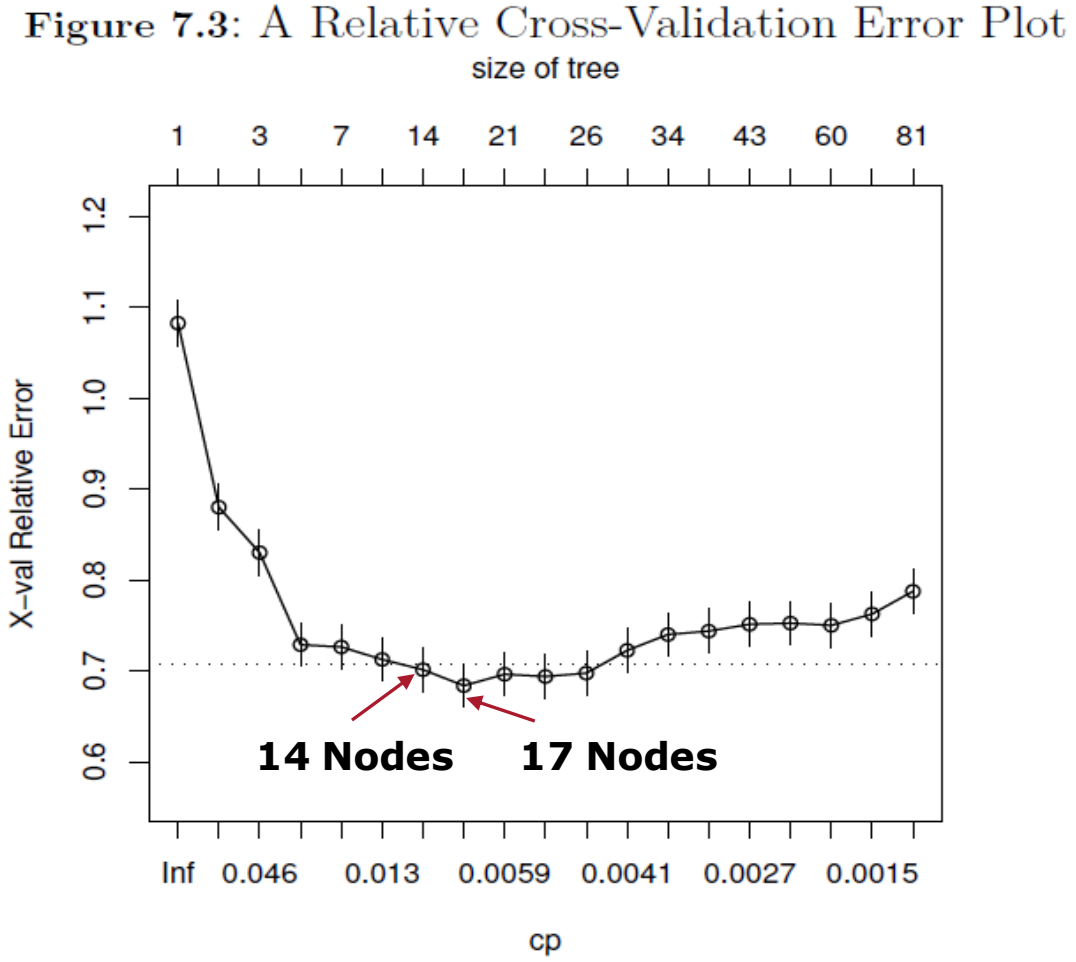
nstitute

Decision Tree

- **Dotted line**: The algorithm calculates the standard deviation of the relative error at the minimum value and plots a horizontal dotted line that is **one standard deviation** away from the estimated value.

- Any point below that **Dotted line** is said to be statistically equivalent to the minimum point.

- ** We could choose the simplest tree that has a cross-validation error statistically equivalent to the minimum error (17 nodes), which corresponds to the tree with 14 nodes, for example.



Hands-on Example: Decision Tree

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, confusion_matrix
import graphviz

def main():
    bal_df = pd.read_csv('balloons.csv')

    # convert categorical data to int representations of unique categories
    for col in bal_df.columns:
        labels, uniques = pd.factorize(bal_df[col])
        bal_df[col] = labels

    X = bal_df.drop(columns='inflated')
    y = bal_df['inflated']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    dtree = DecisionTreeClassifier()
    dtree.fit(X_train, y_train)
    predictions = dtree.predict(X_test)
    print(accuracy_score(y_test, predictions))
    print(confusion_matrix(y_test, predictions))

    dot_data = tree.export_graphviz(dtree, out_file=None,
                                    feature_names=('Color', 'size', 'act', 'age'),
                                    class_names=('0','1'),
                                    filled=True)
    graph = graphviz.Source(dot_data, format="png")
    graph.render('balloons_dt', view=True)

if __name__ == "__main__":
    main()
```

out_file : *object or str, default=None*

Handle or name of the output file. If `None`, the result is returned as a string.

filled : *bool, default=False*

When set to `True`, paint nodes to indicate majority class for classification, extremity of values for regression, or purity of node for multi-output.

Each attribute has only two values.

Color	size	act	age	inflated
YELLOW	SMALL	STRETCH	ADULT	T
YELLOW	SMALL	STRETCH	CHILD	T
YELLOW	SMALL	DIP	ADULT	T
YELLOW	SMALL	DIP	CHILD	F
YELLOW	SMALL	DIP	CHILD	F
YELLOW	LARGE	STRETCH	ADULT	T
YELLOW	LARGE	STRETCH	CHILD	T
YELLOW	LARGE	DIP	ADULT	T
YELLOW	LARGE	DIP	CHILD	F
YELLOW	LARGE	DIP	CHILD	F

Hands-on Example: Decision Tree

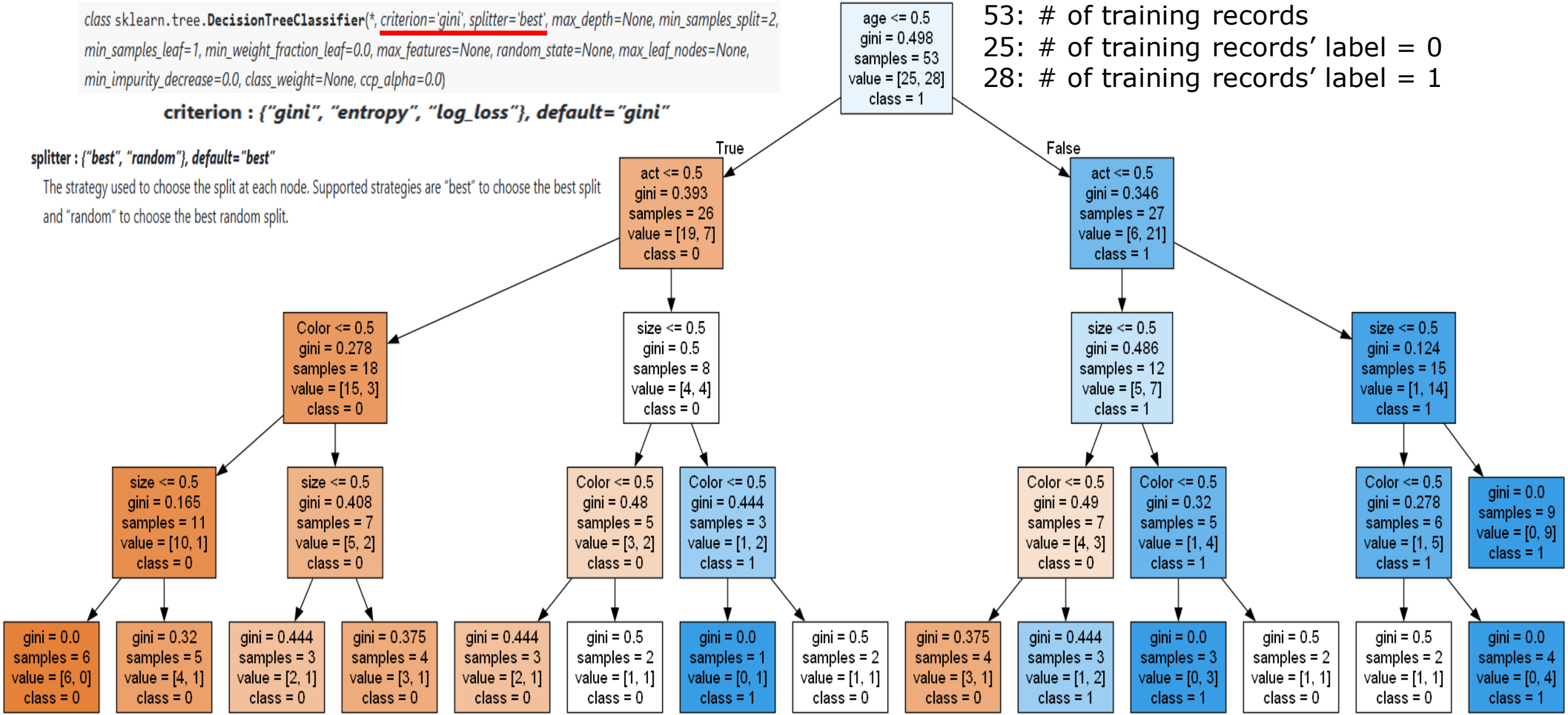
class sklearn.tree.**DecisionTreeClassifier**(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*)

criterion : {"gini", "entropy", "log_loss"}, default="gini"

splitter : {"best", "random"}, 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.

53: # of training records
25: # of training records' label = 0
28: # of training records' label = 1



Decision Tree

- Decision Tree vs. Logistic Regression Model

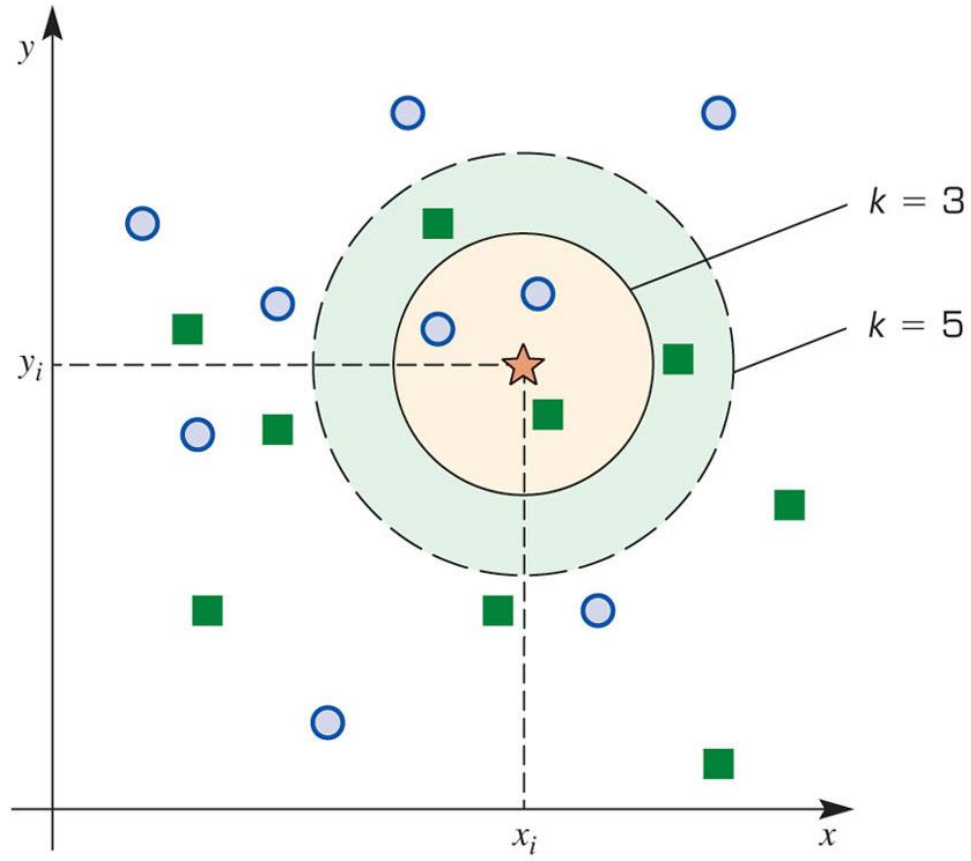
Logistic Regression Model	Decision Tree
Require historical data that contains both predictors and target Variables	
Use historical data to create a model that shows the relationship between the predictor and target Variables	
An algebraic formula	If-then rules
Use predictor variables to predict the value of the target variable	
Calculate the probability of a target variable that split the data into two groups, i.e., Yes or No.	Assign a new case to a leaf node that is either Yes or No for the target variable.

Decision Tree

https://scikit-learn.org/stable/modules/tree.html#classification

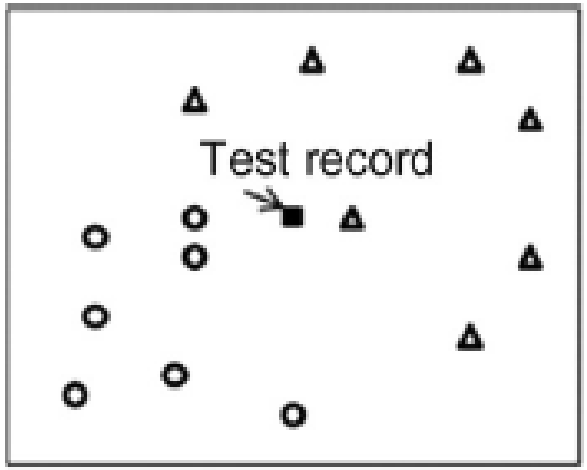
k-Nearest Neighbors

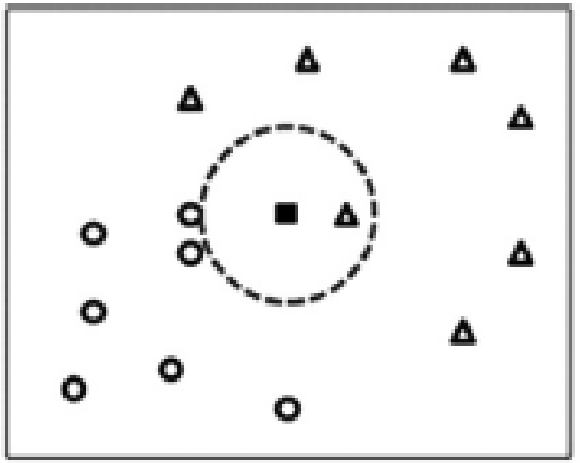
- **What are *k*-Nearest Neighbors (*k*-NNs)?**
  - The entire training dataset is "memorized" and when unlabeled example records need to be classified, the input attributes of the new unlabeled records are compared against the entire training set to find the closest match. The class label of the closest training record is the predicted class label for the unseen test record.
  - ***How can we find the closest training record for each new record?***



k-Nearest Neighbors

- **How can we find the closest training record for each new record?**
  - The *k* in the *k*-NN algorithm indicates the number of close training record(s) that need to be considered when making the prediction for an unlabeled test record.
  - When ***k*=1**, the model tries to find **the first nearest record** and adopts the class label of the first nearest record as the predicted target class value.

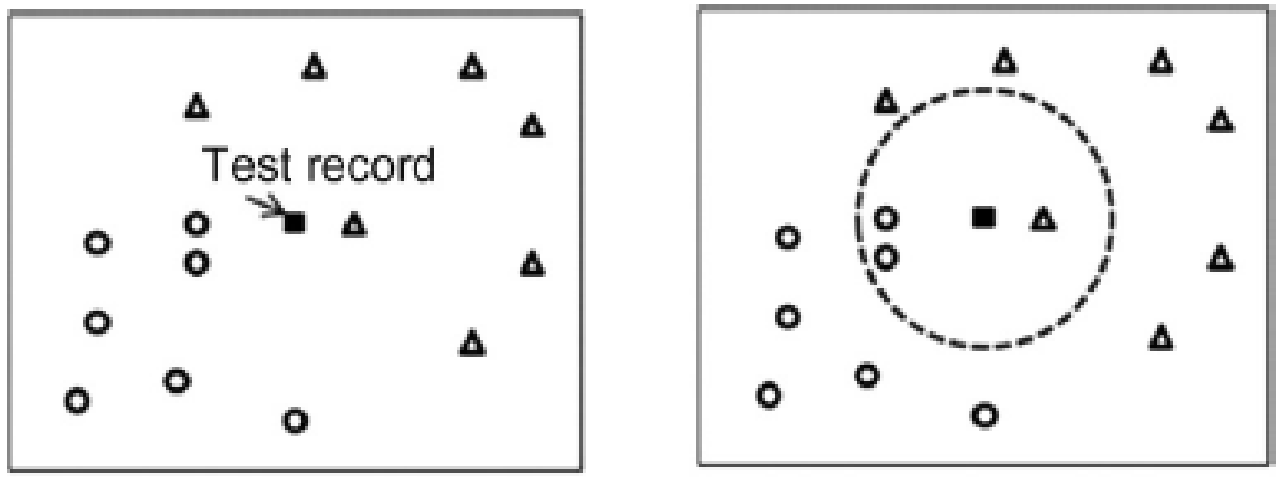




K = 1 Predicted Class is triangle

k-Nearest Neighbors

- **How can we find the closest training record for each new record?**
  - When ***k*=3**, the nearest three training records are considered instead of one. Based on the majority class of **the nearest three training records**, the predicted class of the test record can be concluded as circle.
  - Since the class of the target record is evaluated by voting, ***k*** is usually assigned an **odd** number for **a two-class problem**.



K = 3 Predicted Class is circle

Measure of Proximity - Distance

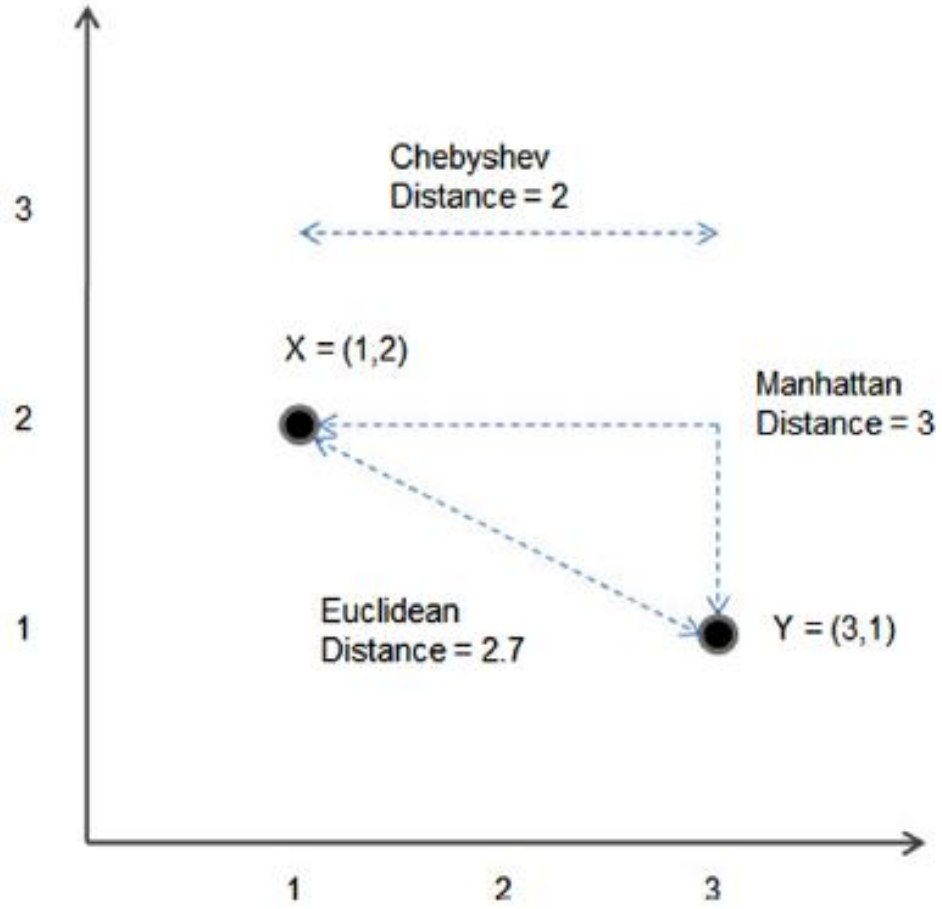
- **Euclidean Distance (ED)**

$$\text{Distance } d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \qquad (4.4)$$

For example, the first two records of a four-dimensional Iris dataset is $X = (4.9, 3.0, 1.4, 0.2)$ and $Y = (4.6, 3.1, 1.5, 0.2)$. The distance between X and Y is: $d = \sqrt{(0.3)^2 + (0.1)^2 + (0.1)^2 + (0)^2} = 0.33$ centimeters.

- **Manhattan Distance (MD):** The sum of the difference between individual attributes, rather than the root of squared difference.

- **Chebyshev Distance (CD):** The maximum difference between all attributes in the dataset.



Measure of Proximity – Correlation Similarity

$$\text{Correlation}(X, Y) = \frac{s_{xy}}{s_x \times s_y}$$

where s_{xy} is the covariance of X and Y, which is calculated as:

$$s_{x\,y} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

and s_x and s_y are the standard deviation of X and Y, respectively. For example, the Pearson correlation of two data points X (1,2,3,4,5) and Y (10,15,35,40,55) is 0.98.

Measure of Proximity – Cosine Similarity

$$X\ (1,2,0,0,3,4,0)\ \text{and}\ Y\ (5,0,0,6,7,0,0).$$

$$\text{Cosine similarity}(|X, Y|) = \frac{x \cdot y}{||x||\ \ ||y||}$$

where $x \cdot y$ is the dot product of the x and y vectors with, for this example,

$$x \cdot y = \sum_{i=1}^{n} x_i y_i \text{ and } ||x|| = \sqrt{x \cdot x}$$

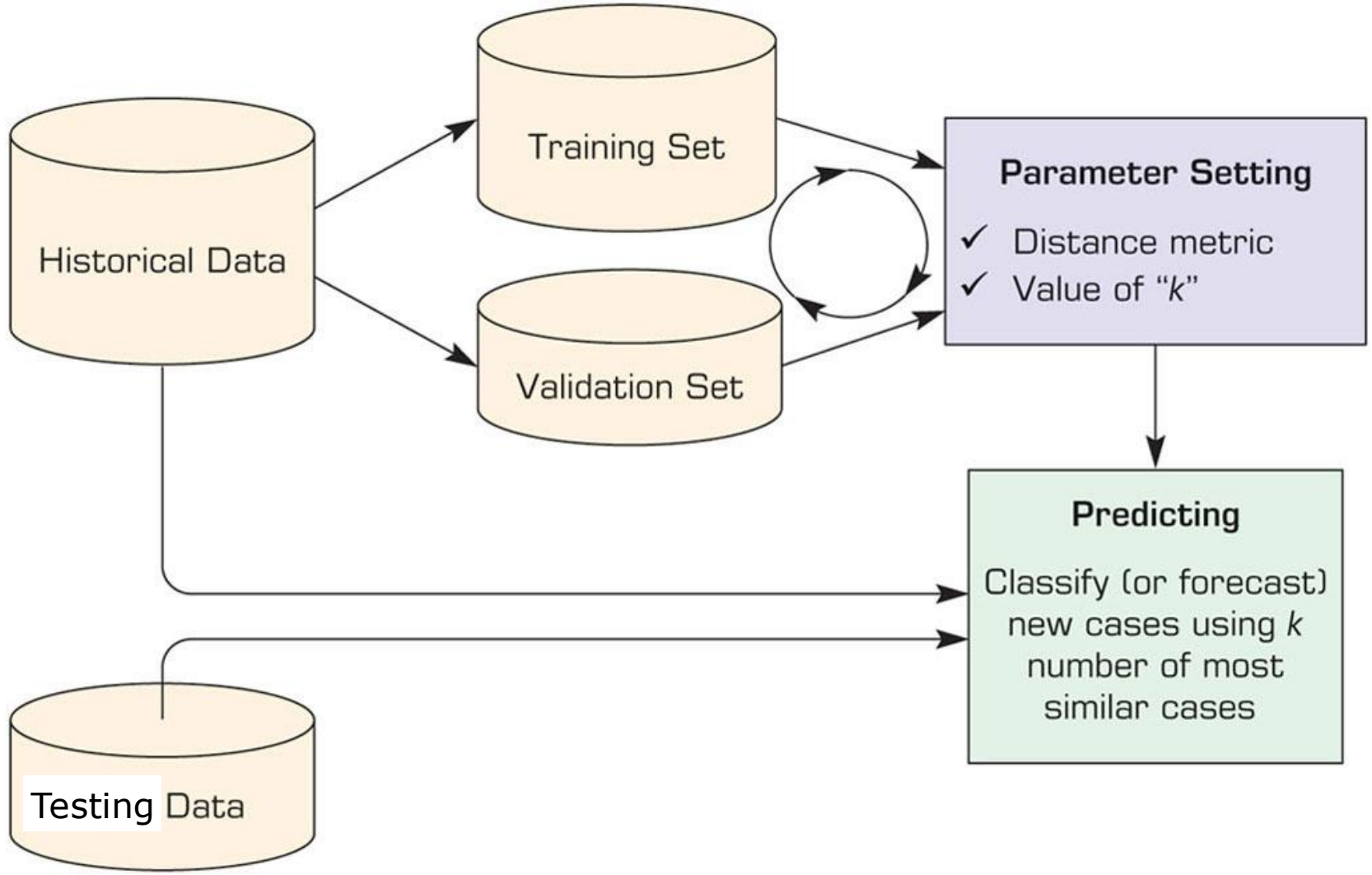
$x \cdot y =$ 1 x 5 + 2 x 0 + 0 X 0 + 0 X 6 + 3 X 7 + 4 X 0 + 0 X 0 = 26

$$||x|| = \sqrt{1 \times 1 + 2 \times 2 + 0 \times 0 + 0 \times 0 + 3 \times 3 + 4 \times 4 + 0 \times 0} = 5.5$$

$$||y|| = \sqrt{5 \times 5 + 0 \times 0 + 0 \times 0 + 6 \times 6 + 7 \times 7 + 0 \times 0 + 0 \times 0} = 10.5$$

$$\text{Cosine similarity}(|x \cdot y|) = \frac{x \cdot y}{||x||\ \ ||y||} = \frac{26}{5.5 \times 10.5} = 0.45$$

Process for Determining the Optimal Values for Distance Metric and *k*



class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*) [sou

Hands-on Example: *k*-Nearest Neighbors

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

def main():
    df = pd.read_csv("wine.csv")

    # Mark about 70% of the data for training and use the rest for
    # testing
    # We will use 'density', 'sulphates', and 'residual_sugar'
    # features for training a classifier on 'high_quality'
    X_train, X_test, y_train, y_test = train_test_split(df[['density','sulphates','residual_sugar']],
                                                        df['quality'], test_size=.3)

    # Define the classifier using kNN function and train it
    classifier = KNeighborsClassifier(n_neighbors=3)
    classifier.fit(X_train, y_train)

    # Test the classifier by giving it test instances
    prediction = classifier.predict(X_test)

    # Count how many were correctly classified
    correct = np.where(prediction==y_test, 1, 0).sum()
    print(correct)

    # Calculate the accuracy of this classifier
    accuracy = correct/len(y_test)
    print(accuracy)

    # Start with an array where the results (k and corresponding
    # accuracy) will be stored
    results = []
```

```
    for k in range(1, 51):
        classifier = KNeighborsClassifier(n_neighbors=k)
        classifier.fit(X_train, y_train)
        prediction = classifier.predict(X_test)
        accuracy = np.where(prediction==y_test, 1, 0).sum() / (len(y_test))
        print ("k=",k," Accuracy=", accuracy)
        results.append([k, accuracy]) # Storing the k,accuracy tuple in results array

    # Convert that series of tuples in a dataframe for easy plotting
    results = pd.DataFrame(results, columns=["k", "accuracy"])

    plt.plot(results.k, results.accuracy)
    plt.title("Value of k and corresponding classification accuracy")
    plt.show()

if __name__ == "__main__":
    main()
```

p : *int, default=2*

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*

Metric to use for distance computation. Default is "minkowski", which results in the standard Euclidean distance when p = 2. See the documentation of scipy.spatial.distance and the metrics listed in `distance_metrics` for valid metric values.

fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color	is_red	high_quality
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	red	1	0
7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5	red	1	0
7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5	red	1	0
11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6	red	1	0
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	red	1	0
7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5	red	1	0
7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5	red	1	0
7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	7	red	1	1
7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	7	red	1	1

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*) [sou

Hands-on Example: *k*-Nearest Neighbors

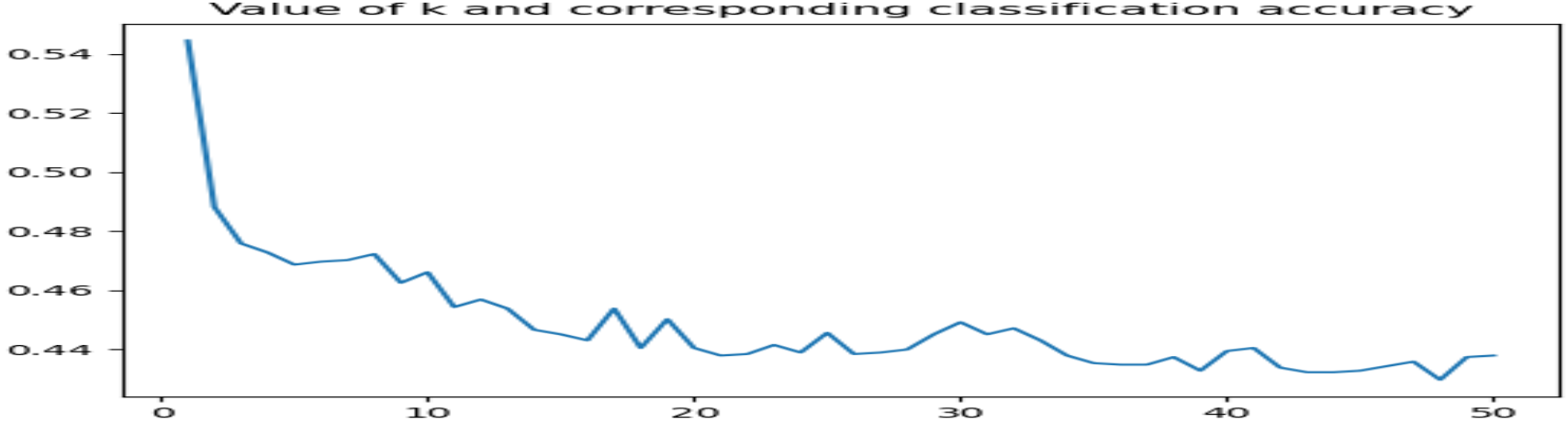
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

def main():
    df = pd.read_csv("wine.csv")

    # Mark about 70% of the data for training and use the rest for
    # testing
    # We will use 'density', 'sulphates', and 'residual_sugar'
    # features for training a classifier on 'high_quality'
    X_train, X_test, y_train, y_test = train_test_split(df[['density','sulphates','residual_sugar']],
                                                        df['quality'], test_size=.3)

    # Define the classifier using kNN function and train it
    classifier = KNeighborsClassifier(n_neighbors=3)
    classifier.fit(X_train, y_train)

    # Test the classifier by giving it test instances
    prediction = classifier.predict(X_test)

    # Count how many were correctly classified
    correct = np.where(prediction==y_test, 1, 0).sum()
    print(correct)

    # Calculate the accuracy of this classifier
    accuracy = correct/len(y_test)
    print(accuracy)

    # Start with an array where the results (k and corresponding
    # accuracy) will be stored
    results = []

    for k in range(1, 51):
        classifier = KNeighborsClassifier(n_neighbors=k)
        classifier.fit(X_train, y_train)
        prediction = classifier.predict(X_test)
        accuracy = np.where(prediction==y_test, 1, 0).sum() / (len(y_test))
        print ("k=",k," Accuracy=", accuracy)
        results.append([k, accuracy]) # Storing the k,accuracy tuple in results array

    # Convert that series of tuples in a dataframe for easy plotting
    results = pd.DataFrame(results, columns=["k", "accuracy"])

    plt.plot(results.k, results.accuracy)
    plt.title("Value of k and corresponding classification accuracy")
    plt.show()

if __name__ == "__main__":
    main()
```

p : *int, default=2*

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*

Metric to use for distance computation. Default is "minkowski", which results in the standard Euclidean distance when p = 2. See the documentation of scipy.spatial.distance and the metrics listed in distance_metrics for valid metric values.



k-Nearest Neighbors

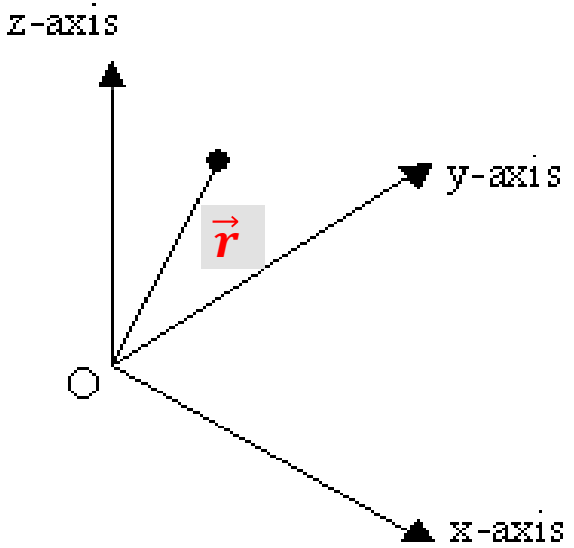
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Support Vector Machine

- **What is a Support Vector Machine (SVM)?**
  - It is based on the concept of **decision hyperplanes** that define decision boundaries.
  - A decision plane in the **two** dimensions separates a set of data instances having **two different class memberships**.
  - Given the training dataset, the SVM algorithm constructs **an optimal decision hyperplane** in a **two-dimensional vector space**, which is able to categorize data instances in **two different classes** respectively.
  - Generalizing to **n dimensions**, if we have **n number of input attributes**, the best **n-1 dimensional decision hyperplane** is found.

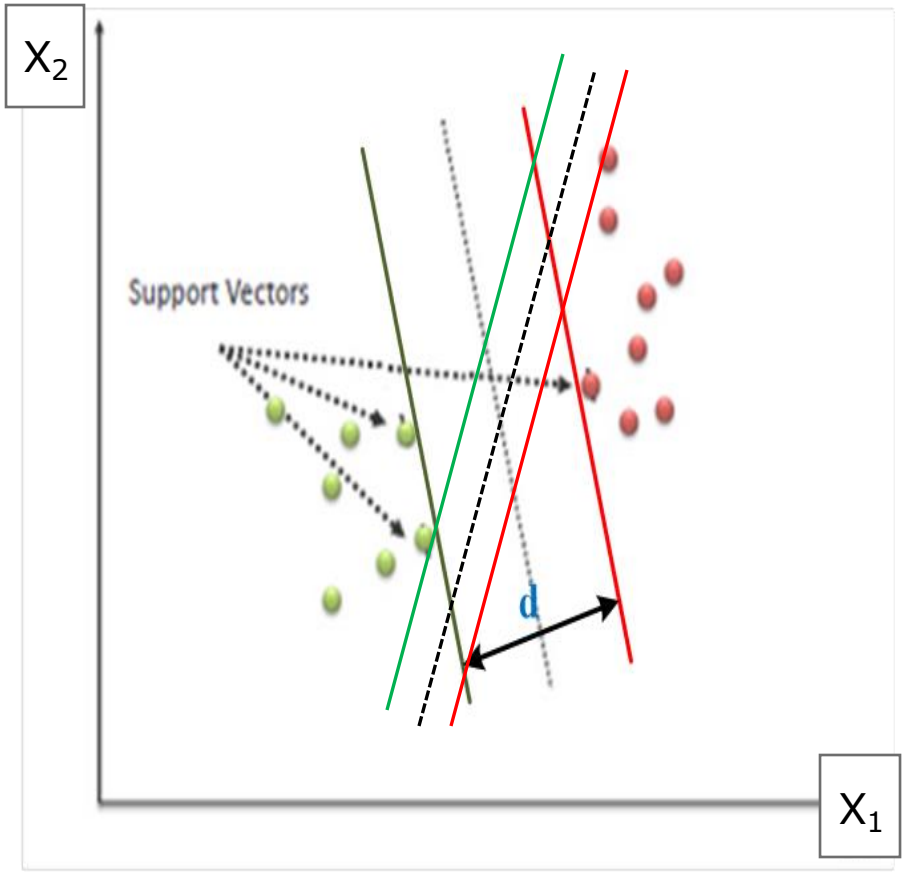
Support Vector Machine

- **What is a n-dimensional vector space?**
  - Each dimension represents an **input attribute** of a data instance.
  - The value of each input attribute is **the value of one coordinate axis** in this vector space.
  - Each data instance is defined as **($\vec{r}$, o)** in the vector space:
    - (1) represents a data point in this vector space
    - (2) is described by a set of different dimensions in the vector space
    - (3) associates with **a particular class value of a target variable**, e.g., Buy or Sell the Stock, Stay or Quit the Job, etc.
  - Using the SVM, once we can **find the optimal decision hyperplane**, we can perform the classifications using that plane to differentiate the classes of data instances.



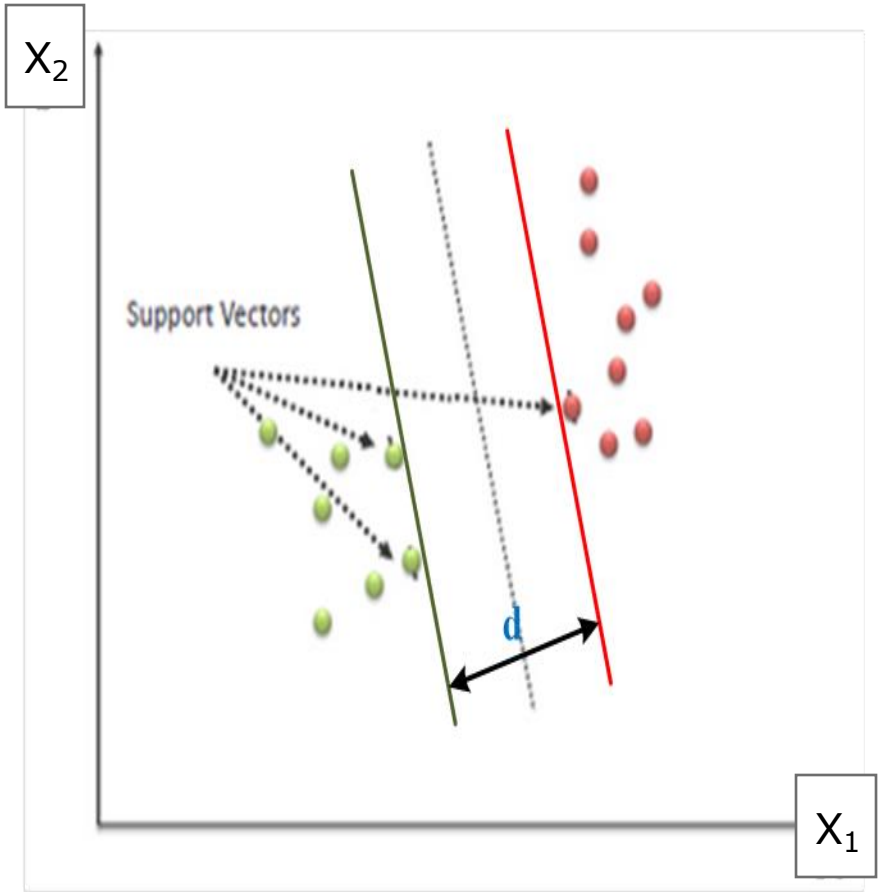
Support Vector Machine

- **Support Vectors**
  - Support vectors are the data instances that (1) are found to **lie at the edge of two planes** that separate the data with **no points** between them and far away from the separating line as possible, i.e., "margin" **AND** (2) the **closest** to the optimal decision **hyperplane** between one of these two classes of data instances (e.g., Buy the Stock) and another class of data instances (e.g., Sell the Stock).
  - **Using the support vectors**, the SVM algorithm searches the **widest** distance (called the **maximal margin (d)**) that separates data instances with different class values of the target variable optimally.
  - Consider the two input variables X_1 and X_2 (i.e., 2-dimensional vector space). **The SVM algorithm purposefully select an optimal decision plane that clearly demonstrate a significant separation (margin) between the data instances**, i.e., one with **GREEN CIRCLE** and the other with **RED CIRCLE**.



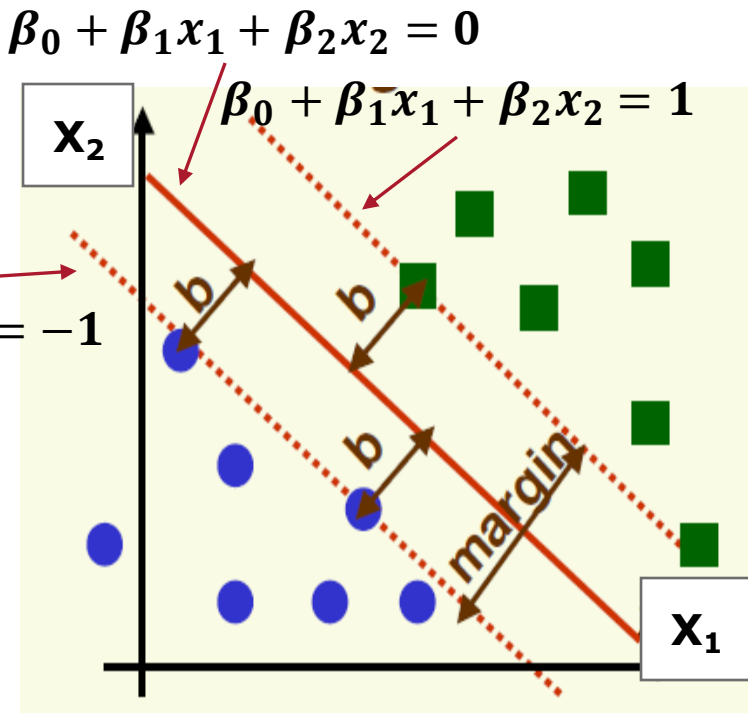
Support Vector Machine

- **Maximal Margin Decision Plane**
  - The middle dash line is the **maximal margin decision hyperplane**, as it makes the distance between the support vectors and the plane is maximal.
  - This maximal margin decision hyperplane also separates all the **green circles** on one side of the plane and all the **red circles** on the other side.
  - The color of each circle denotes the class value of its target variable.
  - This is a **linear SVM**.



Consider the hyperplane: $f(x) = \beta_0 + \beta^T x$

- $f(x) = \beta_0 + \beta^T x$, where x is a set of input attributes, T is a weight vector, namely $T = \{1, 2, \ldots, n\}$, n is the number of input attributes and *f(x) is the output attribute, i.e., y.* By scaling of β_0 and $\beta^T = \{\beta_1, \beta_2, \ldots, \beta_n\}$, the optimal hyperplane can be found.
- Assume that $\boldsymbol{n = 2}$. We have two input attributes $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ and one output attribute $\boldsymbol{y}$.
  - $\boldsymbol{b = \frac{|\beta_0 + \beta^2 X|}{||\beta||} = \frac{|\beta_0 + \beta_1 x_1 + \beta_2 x_2|}{\sqrt{\beta_0^2 + \beta_1^2 + \beta_2^2}}}$
- Extending this same equation to any $\boldsymbol{n}$ input attributes:
  - $\boldsymbol{b = \frac{|\beta_0 + \beta^T X|}{||\beta||}}$
- Consider the canonical hyperplane, the numerator is equal to one and the distance to the support vectors is $\boldsymbol{b_{support\ vectors} = \frac{|\beta_0 + \beta^T X|}{||\beta||} = \frac{1}{||\beta||}}$
- Margin (M) = $\mathbf{2b = \frac{2}{||\beta||}}$

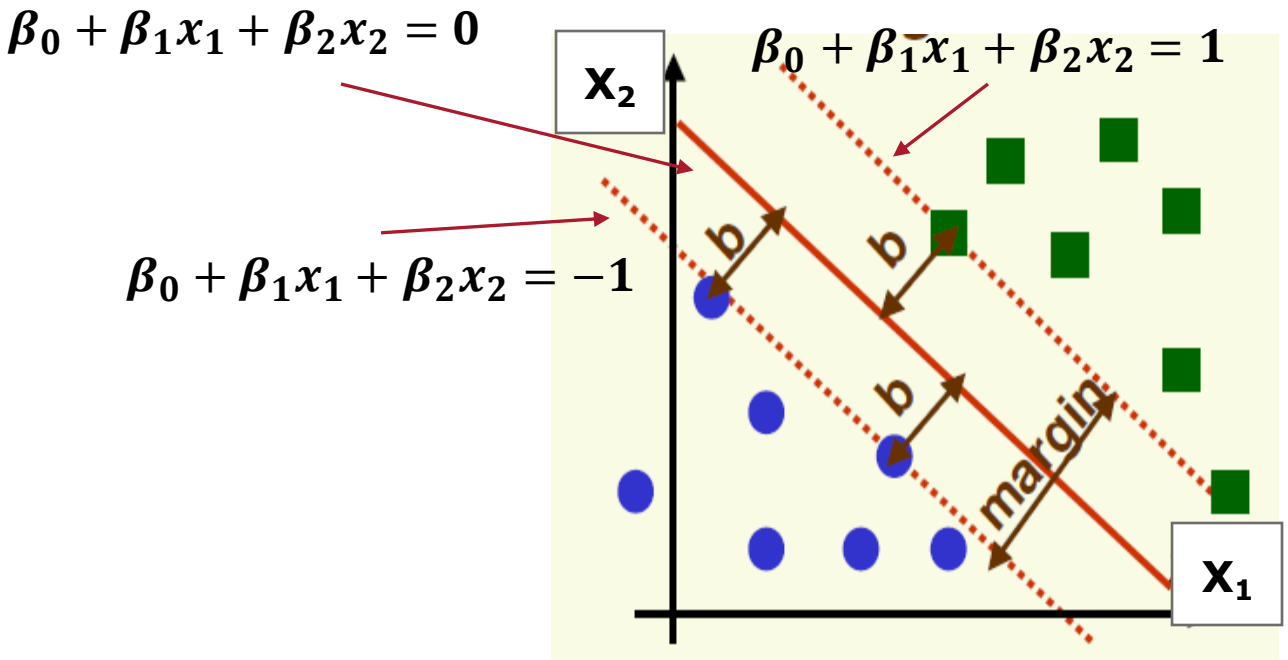


Consider the hyperplane: $f(x) = \beta_0 + \beta^T x$

- Margin (M) = 2b= $\frac{2}{||\beta||}$
- Maximizing M is equivalent to the problem of minimizing a function *L()* subject to some constraints.

$$\min_{\beta^T, \beta_0} L(\beta) = \frac{1}{2}||\beta||^2$$

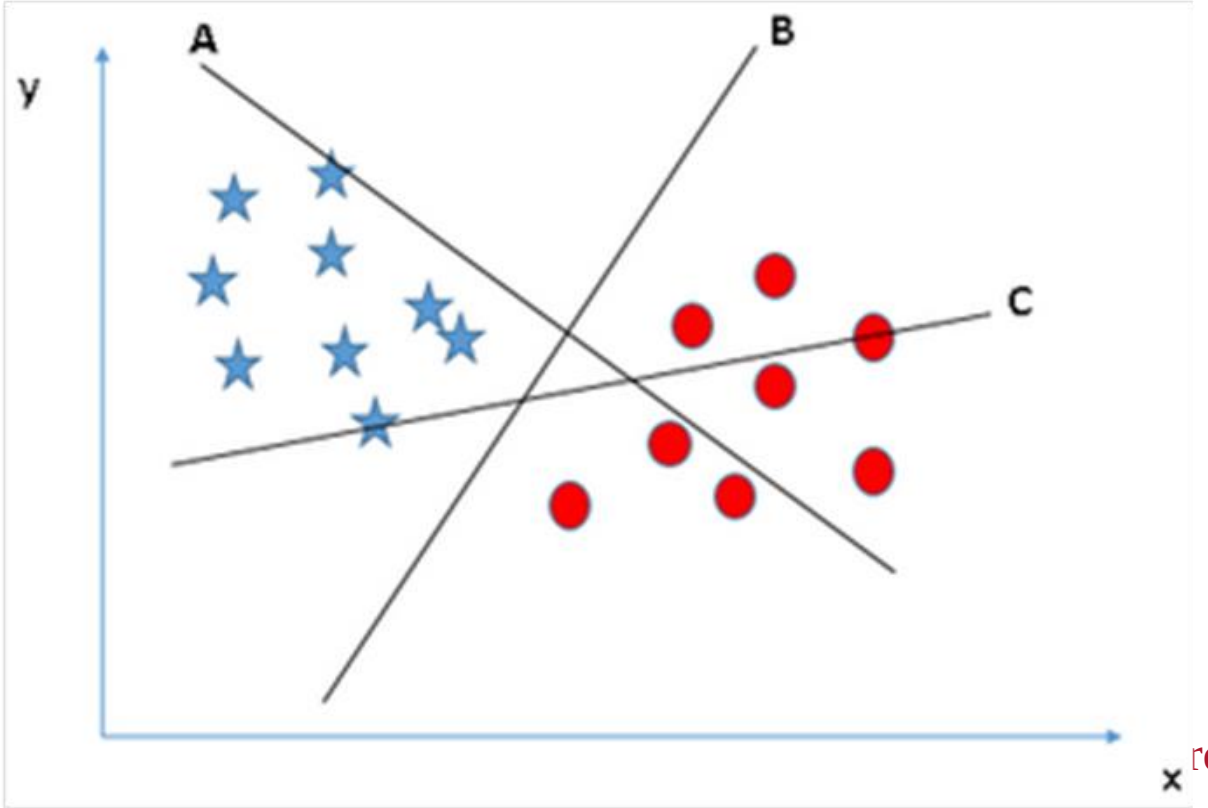
$$s.t.\ y_i(\beta^T x_i + \beta_0) \geq 1 \forall i, where\ y_i\ represents\ each\ of\ the\ labels\ of\ the\ training\ examples.$$



Linear SVM

- **Four Selection Rules for the Optimal Decision Hyperplane by SVM**

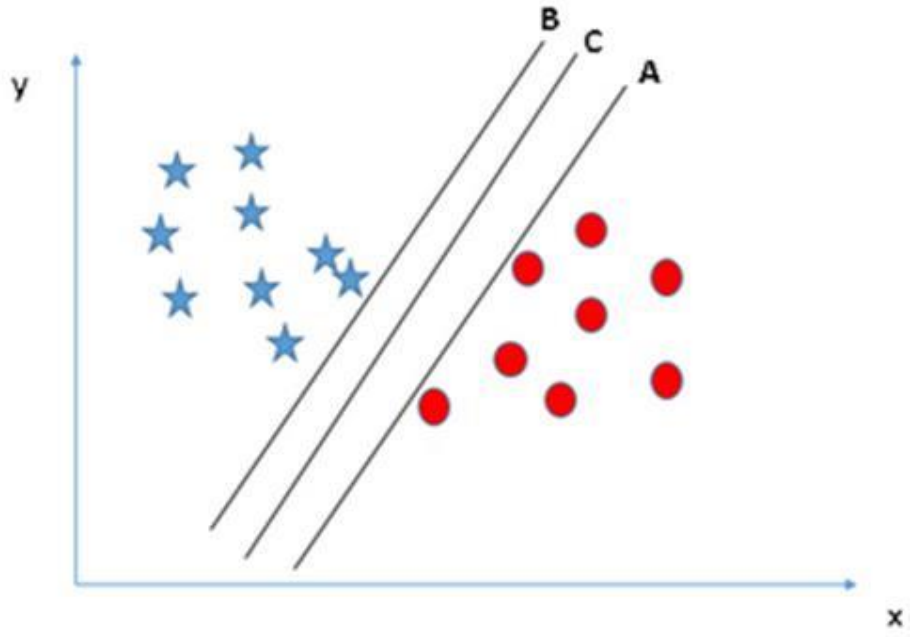
RULE 1: SELECT THE DECISION HYPERPLANE WHICH SEGREGATES TWO SETS OF DATA POINTS ACCURATELY.

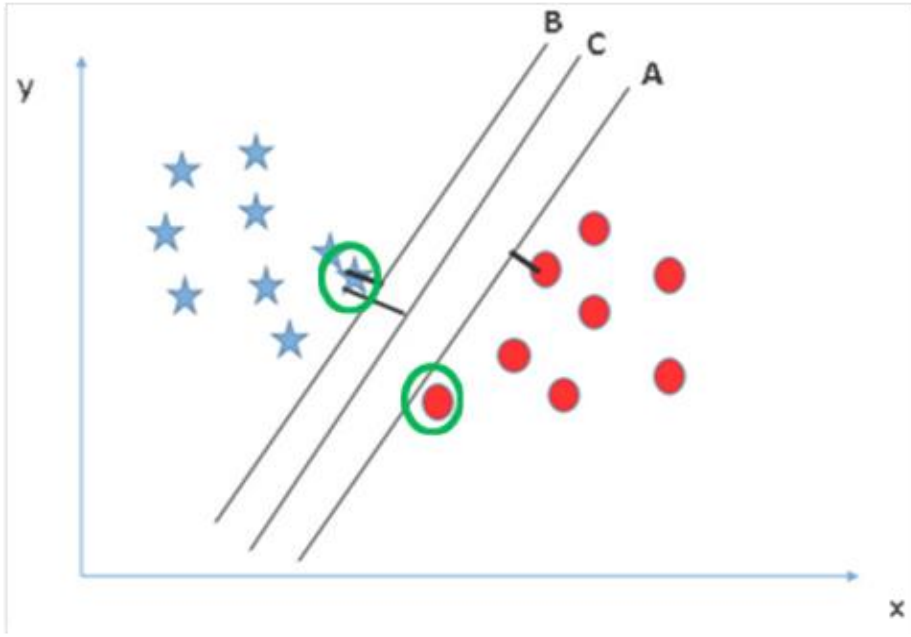


Linear SVM

- **Four Selection Rules for the Optimal Decision Hyperplane by SVM**

RULE 2: SELECT THE MAXIMAL MARGIN DECISION HYPERPLANE THAT MAKES THE MARGINAL DISTANCE BETWEEN THE SUPPORT VECTORS MAXIMAL.

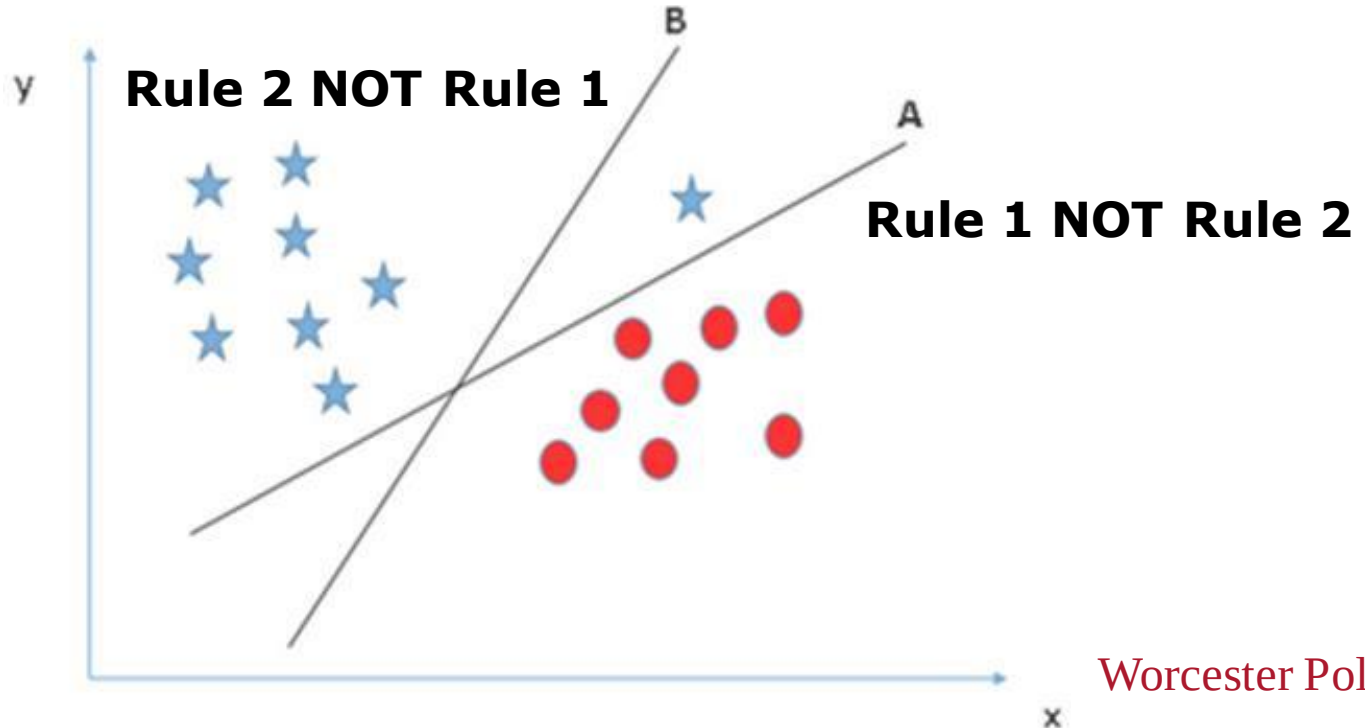




Linear SVM

- **Four Selection Rules for the Optimal Decision Hyperplane by SVM**

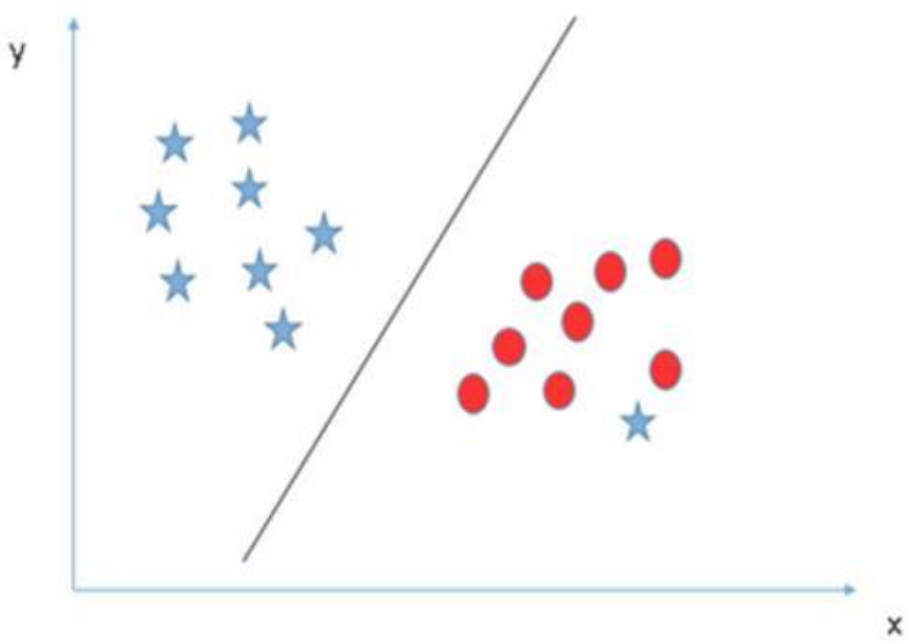
RULE 3: SELECT THE DECISION PLANE WHICH CLASSIFIES THE CLASSES ACCURATELY (RULE 1) PRIOR TO MAXIMIZING THE MARGINAL DISTANCE (RULE 2).



Linear SVM

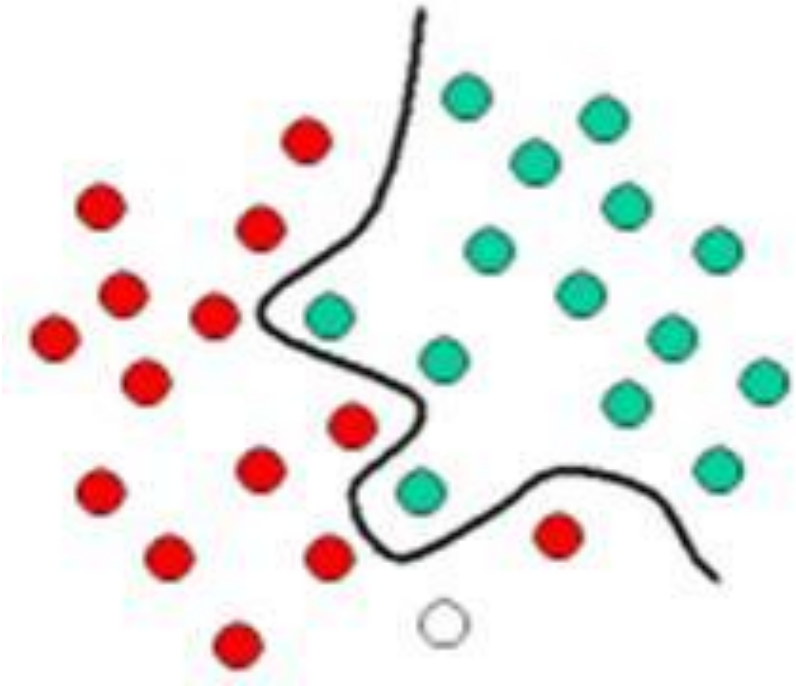
- **Four Selection Rules for the Optimal Decision Hyperplane by SVM**

RULE 4: IGNORE OUTLIERS AND FIND THE DECISION HYPERPLANE THAT HAS THE MAXIMAL MARGINAL DISTANCE BETWEEN THE SUPPORT VECTORS.



Non-Linear SVM

- **Problem: It is usually quite rare that we can separate the data with a linear decision hyperplane because the data is not usually distributed in such a way that it is linearly separable.**
  - A linear decision hyperplane cannot be applied to separate the data instances in two different classes.



  - **How can SVM classify these two classes of circles?**

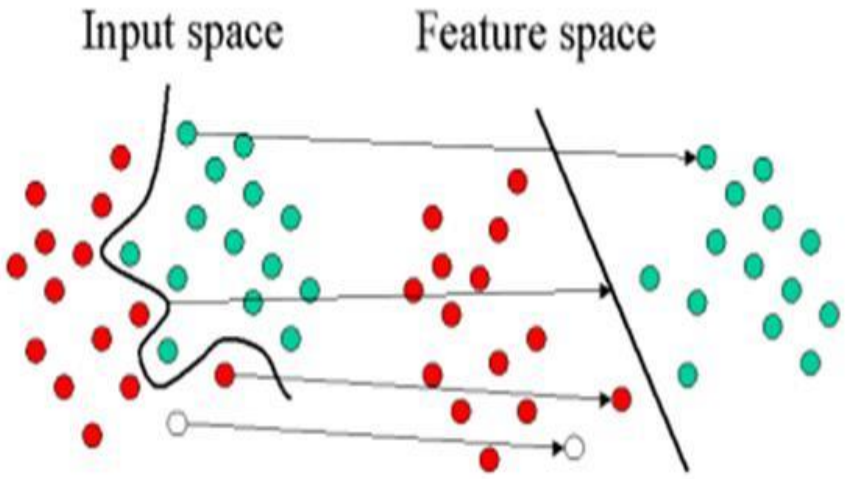
Non-Linear SVM

- **Solution: When this non-linear case happens, a kernel function is used to remap the original data in a different vector space, that is, creating new variables so that the classes are then more likely to become linearly separable by a decision hyperplane (i.e., so that with the new dimensional data, there is a gap between data instances in the two classes).**

  - The original data instances of the input space are rearranged using a set of mathematical functions, known as kernels.

  - The process of rearranging the data instances is known as **mapping**.

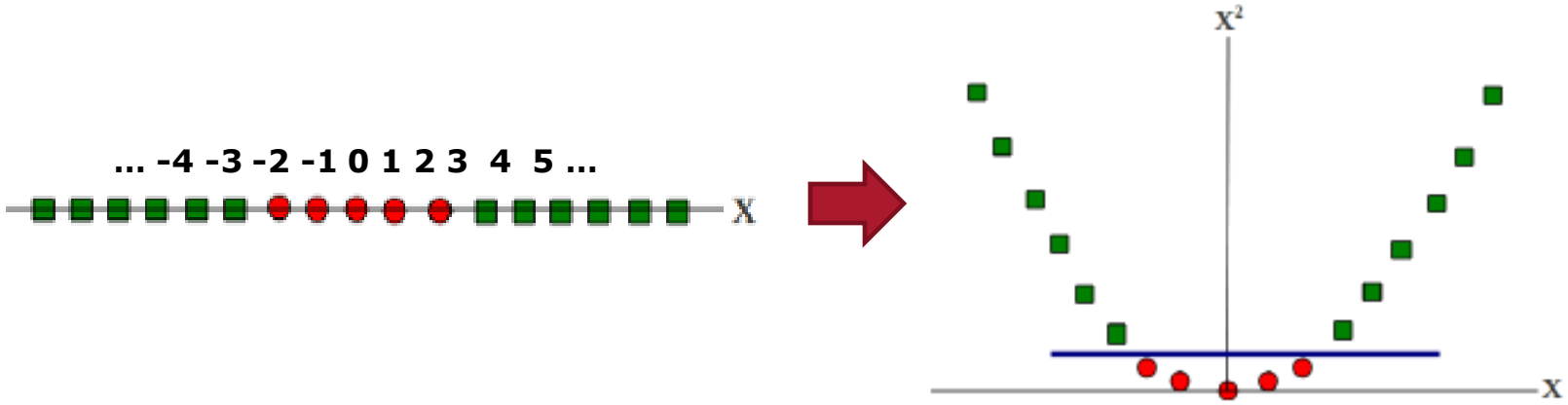
  - In this new feature space, the mapped data instances of the feature space is linearly separable and, thus, all we have to do is to find an optimal decision hyperplane that can separate the **green** and the **red** circles.



Non-Linear SVM

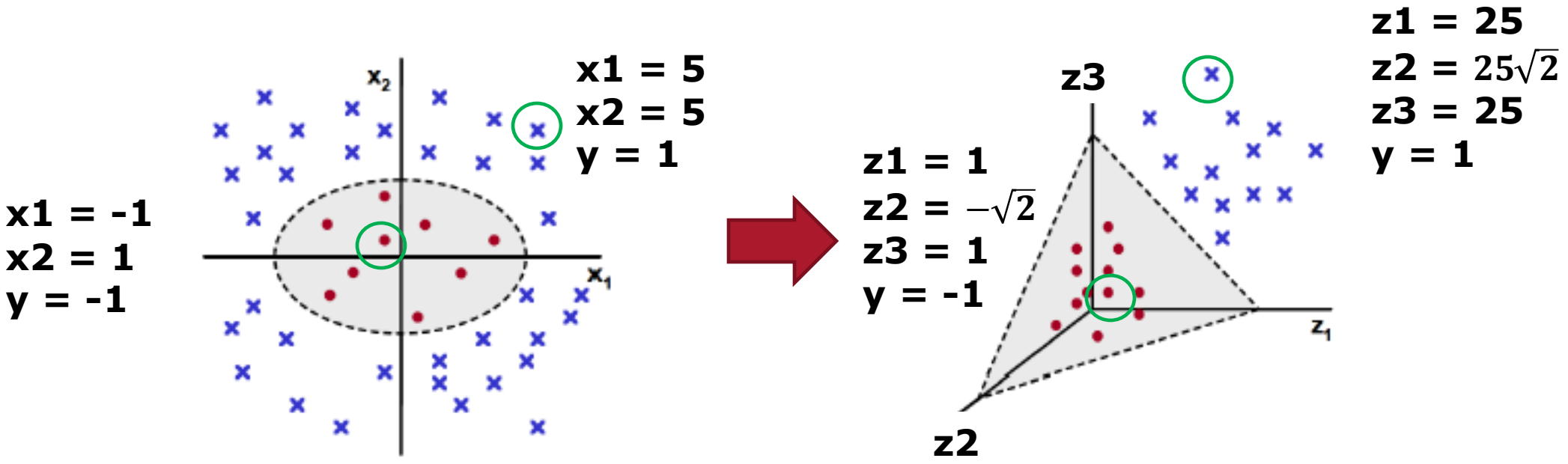
- **Example 1:**
  - Each data instance is represented by a single attribute X.
  - Can we use a linear plane to separate them?
  - A **kernel function** maps each data instance as follows:

    $\mathbf{x \rightarrow z = \{x, x^2\}}$



Non-Linear SVM

- **Example 2:**
  - Each data instance is represented by a two attributes x = {x_1, x_2}
  - Can we use a linear plane to separate them?
  - A kernel function maps each data instance as follows:
  - **x = {x_1, x_2} → z = {z_1, z_2, z_3} → z = {x_1^2, $\sqrt{2}x_1x_2$, x_2^2}**

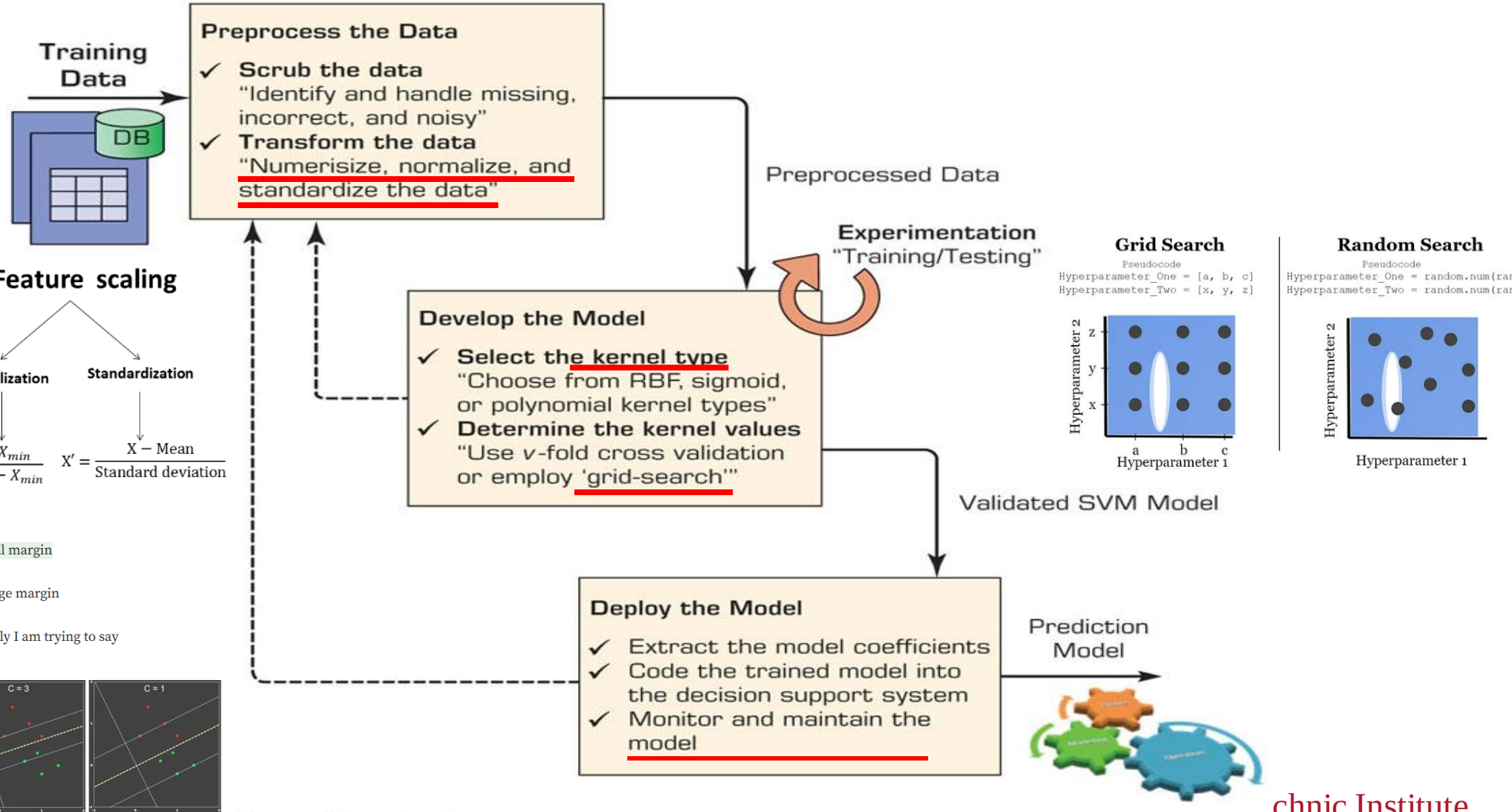


Non-Linear SVM

- **Kernel Functions**
  - Convert non-linear separate problems into linear separate problems.
  - Do some complex data transformations and then find out the process to separate data instances based on the class labels of the target variable.
  - There are a number of kernel functions that can be used in SVM models. Some functions include **linear**, **polynomial**, **radial basis**, etc.

- **Question: How can we choose the right kernel function for SVM to solve non-linear separate problems?**
  - There is **NO** standard answer how to help us determine a proper kernel function to be used.
  - In general, we can try the **Linear kernel** first because it is much faster and can yield great results in many cases (specifically high dimensional problems).
  - If the Linear kernel fails, our second choice is the **Radial Basis Function (RBF) kernel**. They are known to perform very well on a large variety of problems.
  - In the worst case, we can **try all the kernels available** and pick the one with the best classification result on the validation dataset.

Process for Developing SVM Models



Paramerter C:

Large Value of parameter C => small margin

Small Value of paramerter C => Large margin

Diagram below will give what exactly I am trying to say

Change in margin with change in C

Pearson Education, Inc.

chnic Institute

Hands-on Example: SVM

```python
import pandas as pd
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

def main():
    # Read in a small version of the skin data set
    skin_df = pd.read_csv('Skin_NonSkin_small.csv')
    X = skin_df.drop(columns='skin')
    y = skin_df['skin']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    # Creating a logistic model will take much less time
    # but will ultimately be less accurate for this data
    print('starting logistic fit')
    log_model = LogisticRegression()
    log_model.fit(X_train, y_train)
    print('finished logistic fit\n')

    log_preds = log_model.predict(X_test)
    print('Logistic Regression Results:')
    print(accuracy_score(y_test, log_preds))
    print(confusion_matrix(y_test, log_preds))
    print()

    # Train an SVM model
    # note that if the full skin data set is used, the fit time
    # becomes impractical
    print('starting SVM fit')
    svm_model = SVC()
    svm_model.fit(X_train, y_train)
    print('finished SVM fit')

    print()
    svm_preds = svm_model.predict(X_test)
    print('SVM Results:')
    print(accuracy_score(y_test, svm_preds))
    print(confusion_matrix(y_test, svm_preds))

if __name__ == "__main__":
    main()
```

r	g	b	skin
74	85	123	1
73	84	122	1
72	83	121	1
70	81	119	1
70	81	119	1
69	80	118	1
70	81	119	1
70	81	119	1
76	87	125	1
76	87	125	1

C : *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

Support Vector Machine

https://scikit-learn.org/stable/modules/svm.html#classification

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Naïve Bayesian

- **What is *the Naïve Bayesian algorithm*?**
  - The naïve Bayesian algorithm is built on the Bayes' theorem.
  - Assume **X** is a set of input attributes $\{X_1, X_2, \ldots, X_n\}$ and **Y** is the output attribute, i.e., target class.
  - P(**X**) is the probability of **X** that shows the likelihood of the value of **X** happening, i.e., $\{X_1, X_2, \ldots, X_n\}$, in a given dataset.
  - P(**Y**) is the *probability* of **Y** that shows the likelihood of an outcome **Y** happening in a given dataset.
  - P(**Y**|**X**) is called the *conditional probability* of **Y** that provides the probability of an outcome **Y** given the value of **X**, i.e., $\{X_1, X_2, \ldots, X_n\}$, is known.
  - P(**X**|**Y**) is called the *conditional probability* of **X** that provides the probability of the value of **X**, i.e., $\{X_1, X_2, \ldots, X_n\}$ given an outcome **Y** is known.

- ***Bayes' Theorem*** states that:

$$P(Y|\mathbf{X}) = \frac{P(Y) \times P(\mathbf{X}|Y)}{P(\mathbf{X})} \qquad P(Y|\mathbf{X}) = \frac{P(Y) \times \prod_{i=1}^{n} P(\mathbf{X}_i|Y)}{P(\mathbf{X})}$$

The objective is to predict if the Golf player will Play (yes or no) tomorrow, given the weather condition

- **STEP 1: Calculate Probability P(Y)**
  - P(Y = no) = 5 / 14
  - P(Y = yes) = 9 / 14

- **STEP 2: Calculate Conditional Probability $P(X_i|Y)$**

Temperature (X_1)	$P(X_1\|Y=\text{no})$	$P(X_1\|Y=\text{yes})$
High	2/5	2/9
Med	1/5	3/9
Low	2/5	4/9
Humidity (X_2)	**$P(X_2\|Y=\text{no})$**	**$P(X_2\|Y=\text{yes})$**
High	2/5	2/9
Low	1/5	4/9
Med	2/5	3/9
Outlook (X_3)	**$P(X_3\|Y=\text{no})$**	**$P(X_3\|Y=\text{yes})$**
Overcast	0/5	4/9
Rain	2/5	3/9
Sunny	3/5	2/9
Wind (X_4)	**$P(X_4\|Y=\text{no})$**	**$P(X_4\|Y=\text{yes})$**
False	2/5	6/9
True	3/5	3/9

No.	Temperature X_1	Humidity X_2	Outlook X_3	Wind X_4	Play (Class Label) Y
1	High	Med	Sunny	false	no
2	High	High	Sunny	true	no
3	Low	Low	Rain	true	no
4	Med	High	Sunny	false	no
5	Low	Med	Rain	true	no
6	High	Med	Overcast	false	yes
7	Low	High	Rain	false	yes
8	Low	Med	Rain	false	yes
9	Low	Low	Overcast	true	yes
10	Low	Low	Sunny	false	yes
11	Med	Med	Rain	false	yes
12	Med	Low	Sunny	true	yes
13	Med	High	Overcast	true	yes
14	High	Low	Overcast	false	yes

The objective is to predict if the Golf player will Play (yes or no) tomorrow, given the weather condition

- **STEP 1: Calculate Probability P(Y)**
  - P(Y = no) = 5 / 14
  - P(Y = yes) = 9 / 14

Unknown Record

No.	Temperature X_1	Humidity X_2	Outlook X_3	Wind X_4	Play (Class Label) Y
Unlabeled test	high	Low	Sunny	False	?

- **STEP 2: Calculate Conditional Probability $P(X_i|Y)$**

Temperature (X_1)	$P(X_1\|Y=\text{no})$	$P(X_1\|Y=\text{yes})$
High	2/5	2/9
Med	1/5	3/9
Low	2/5	4/9
Humidity (X_2)	**$P(X_2\|Y=\text{no})$**	**$P(X_2\|Y=\text{yes})$**
High	2/5	2/9
Low	1/5	4/9
Med	2/5	3/9
Outlook (X_3)	**$P(X_3\|Y=\text{no})$**	**$P(X_3\|Y=\text{yes})$**
Overcast	0/5	4/9
Rain	2/5	3/9
Sunny	3/5	2/9
Wind (X_4)	**$P(X_4\|Y=\text{no})$**	**$P(X_4\|Y=\text{yes})$**
False	2/5	6/9
True	3/5	3/9

$$P(Y|X) = \frac{P(Y) \times \prod_{i=1}^{n} P(X_i|Y)}{P(X)}$$

$$P(Y=\text{yes}|X) = \frac{P(Y) * \prod_{i=1}^{n} p(X_i|Y)}{P(X)}$$

$$= P(Y=\text{yes}) * \{P(\text{Temp}=\text{high}|Y=\text{yes}) * P(\text{Humidity}=\text{low}|Y=\text{yes}) * P(\text{Outlook}=\text{sunny}|Y=\text{yes}) * P(\text{Wind}=\text{false}|Y=\text{yes})\} / P(X)$$

$$= \frac{9/14 * \{2/9 * 4/9 * 2/9 * 6/9\}}{P(X)}$$

$$= \frac{0.0094}{P(X)}$$

P(Y=no|X) = [5 / 14 * (2 / 5 * 1 / 5 * 3 / 5 * 2 / 5)] / P(X)
= 0.00686 / P(X)

P(Y=yes|X) + P(Y=no|X) = 1
0.0094 / P(X) + 0.00686 / P(X) = 1
P(X) = 0.0094 + 0.00686 = 0.01626

P(Y=yes|X) = 0.0094 / 0.01626 = 0.58
P(Y=no|X) = 0.00686 / 0.01626 = 0.42

Hands-on Example: Naïve Bayesian

```python
import pandas as pd
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.model_selection import train_test_split

def main():
    bank_df = pd.read_csv('bank.csv', delimiter=';')
    for col in bank_df.columns:
        labels, uniques = pd.factorize(bank_df[col])
        bank_df[col] = labels

    y = bank_df['y']
    X = bank_df.drop(columns='y')
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    # Multinomial Naïve Bayes assumes categories follow multinomial distribu
    nb_multi = MultinomialNB()
    nb_multi.fit(X, y)

    # Gaussian Naïve Bayes assumes categories follow normal distribution
    nb_gauss = GaussianNB()
    nb_gauss.fit(X, y)

    from sklearn.metrics import accuracy_score, confusion_matrix
    multi_preds = nb_multi.predict(X_test)
    print("Results for multinomial distribution assumption:")
    print(accuracy_score(y_test, multi_preds))
    print(confusion_matrix(y_test, multi_preds))

    print("Results for Gaussian distribution assumption:")
    gauss_preds = nb_gauss.predict(X_test)
    print(accuracy_score(y_test, gauss_preds))
    print(confusion_matrix(y_test, gauss_preds))

if __name__ == "__main__":
    main()
```

age;"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutcome";"y"
30;"unemployed";"married";"primary";"no";1787;"no";"no";"cellular";19;"oct";79;1;-1;0;"unknown";"no"
33;"services";"married";"secondary";"no";4789;"yes";"yes";"cellular";11;"may";220;1;339;4;"failure";"no"
35;"management";"single";"tertiary";"no";1350;"yes";"no";"cellular";16;"apr";185;1;330;1;"failure";"no"
30;"management";"married";"tertiary";"no";1476;"yes";"yes";"unknown";3;"jun";199;4;-1;0;"unknown";"no"
59;"blue-collar";"married";"secondary";"no";0;"yes";"no";"unknown";5;"may";226;1;-1;0;"unknown";"no"
35;"management";"single";"tertiary";"no";747;"no";"no";"cellular";23;"feb";141;2;176;3;"failure";"no"
36;"self-employed";"married";"tertiary";"no";307;"yes";"no";"cellular";14;"may";341;1;330;2;"other";"no"
39;"technician";"married";"secondary";"no";147;"yes";"no";"cellular";6;"may";151;2;-1;0;"unknown";"no"
41;"entrepreneur";"married";"tertiary";"no";221;"yes";"no";"unknown";14;"may";57;2;-1;0;"unknown";"no"
43;"services";"married";"primary";"no";-88;"yes";"yes";"cellular";17;"apr";313;1;147;2;"failure";"no"

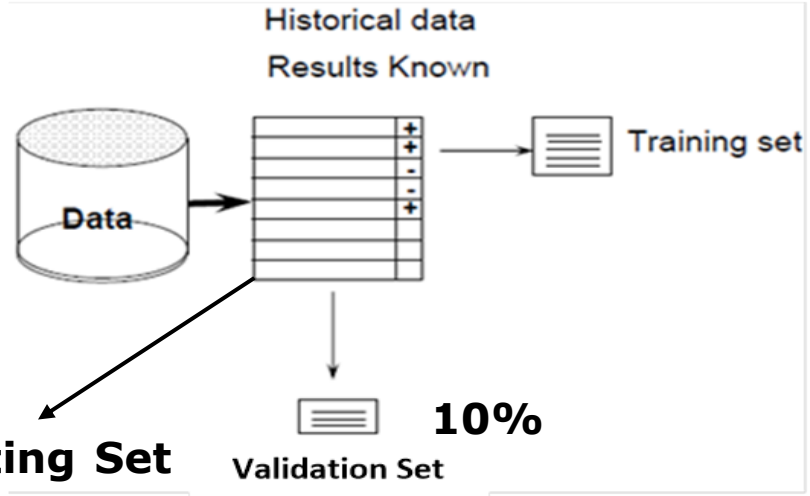
Naïve Bayesian

https://scikit-learn.org/stable/modules/naive_bayes.html

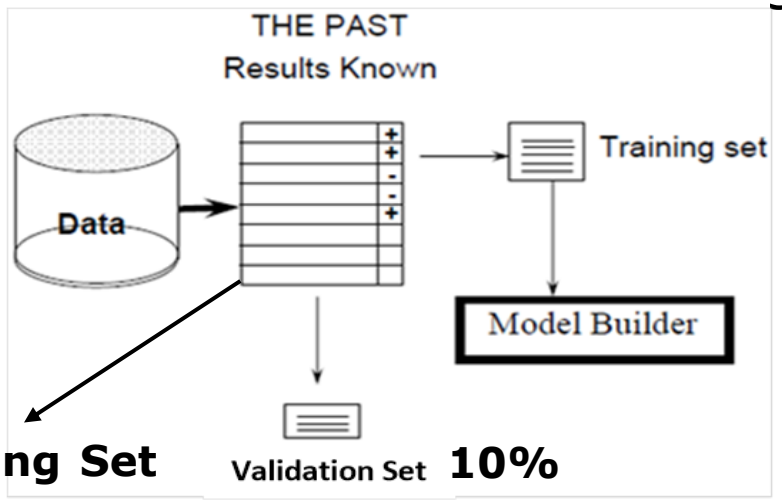
Training and Validation/Testing Process

- **STEP 1:** Split historical data into the training, validation, and testing datasets

Approach 1:



- **STEP 2:** Build a series of models on the training dataset



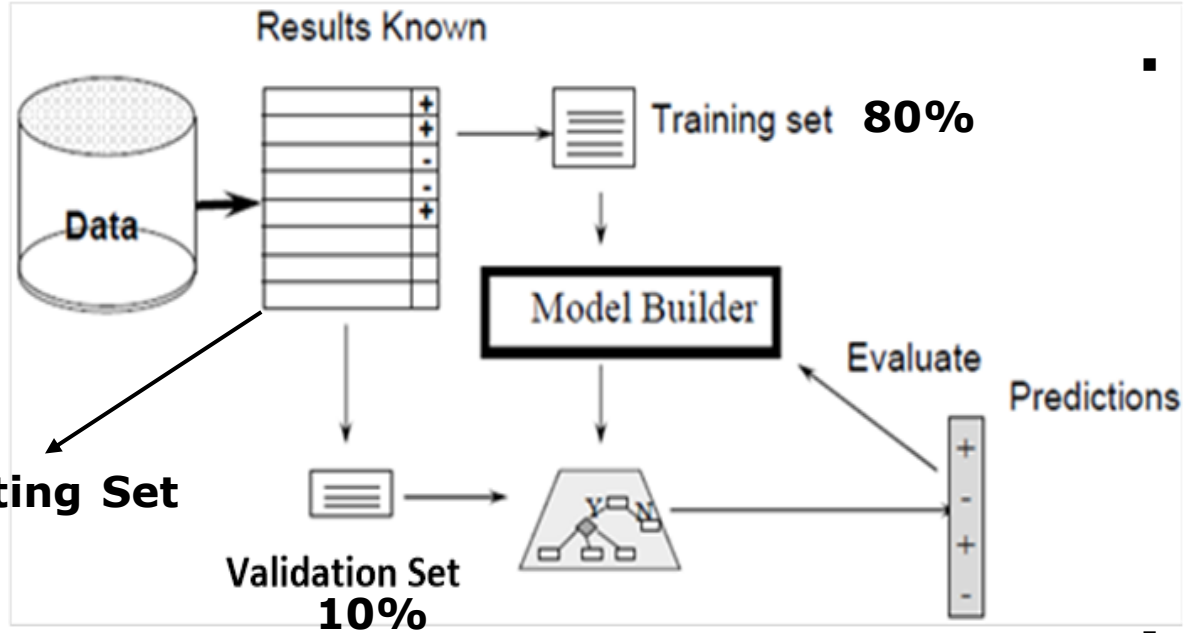
Approach 2:

- If the ***K-fold cross-validation*** is used, you only need to split the data into **training (90%) and testing (10%) dataset**, as the K-fold cross-validation has done the performance evaluation and returned you the model performance from which you can find the best model among all the possible candidates.
- Ave Perf = $\frac{1}{k}\sum_{i=1}^{k} P_i$, where P_i can be errors, accuracy, etc.
- And then you can assess the final selected model on the testing dataset.

Training and Validation/Testing Process

- **STEP 3:** Evaluate the models among all the possible candidates on the validation dataset and select the best one

Approach 1:



- **STEP 4:** Assess the final selected model on the testing dataset
  - If satisfied, go to the Deployment phase
  - If not, go back to the Problem Definition phase

Approach 2:

- If the ***K-fold cross-validation*** is used, you only need to split the data into **training (90%) and testing (10%) dataset**, as the K-fold cross-validation has done the performance evaluation and returned you the model performance from which you can find the best model among all the possible candidates.
- Ave Perf $= \frac{1}{k}\sum_{i=1}^{k} P_i$, where P_i can be errors, accuracy, etc.
- And then you can assess the final selected model on the testing dataset.

Confusion Matrix – Accuracy

- A confusion matrix is a table that is used to describe the performance of a ***binary classification model*** on a set of data (usually **validation**/**testing**) for which the true values of a target variable are known.

Confusion Matrix		Predicted	
		Positive (Yes)	Negative (No)
Actual	Positive (Yes, 1)	True Positive (TP)	False Negative (FN)
	Negative (No, 0)	False Positive (FP)	True Negative (TN)

Accuracy:

0.90-1.0 = excellent (A)
0.80-0.90 = good (B)
0.70-0.80 = fair (C)
0.60-0.70 = poor (D)
0.50-0.60 = fail (F)

  - TP: These are the cases in which we predicted YES, and actually they DO.
  - TN: These are the cases in which we predicted NO, and actually they DO.
  - FP: These are the cases in which we predicted YES, but actually they DON'T.
  - FN: These are the cases in which we predicted NO, but actually they DON'T.

- $Accuracy = \frac{TP+TN}{TP+FN+FP+TN}$ ***It measures the percentage of data instances correctly classified, regardless of which class they belong to.***

Confusion Matrix – Sensitivity/Recall

- A confusion matrix is a table that is used to describe the performance of a ***binary classification model*** on a set of data (usually **validation**/**testing**) for which the true values of a target variable are known.

Confusion Matrix		Predicted	
		Positive (Yes)	Negative (No)
Actual	Positive (Yes, 1)	True Positive (TP)	False Negative (FN)
	Negative (No, 0)	False Positive (FP)	True Negative (TN)

Sensitivity/Recall:

0.90-1.0 = excellent (A)
0.80-0.90 = good (B)
0.70-0.80 = fair (C)
0.60-0.70 = poor (D)
0.50-0.60 = fail (F)

  - TP: These are the cases in which we predicted YES, and actually they DO.
  - TN: These are the cases in which we predicted NO, and actually they DO.
  - FP: These are the cases in which we predicted YES, but actually they DON'T.
  - FN: These are the cases in which we predicted NO, but actually they DON'T.

- $Sensitivity/Recall = \frac{TP}{TP+FN}$ ***It measures the number of correct positive predictions divided by the total number of positives.***

Confusion Matrix – Specificity

- A confusion matrix is a table that is used to describe the performance of a ***binary classification model*** on a set of data (usually **validation**/**testing**) for which the true values of a target variable are known.

Confusion Matrix		Predicted	
		Positive (Yes)	Negative (No)
Actual	Positive (Yes, 1)	True Positive (TP)	False Negative (FN)
	Negative (No, 0)	False Positive (FP)	True Negative (TN)

Specificity:

0.90-1.0 = excellent (A)
0.80-0.90 = good (B)
0.70-0.80 = fair (C)
0.60-0.70 = poor (D)
0.50-0.60 = fail (F)

  - TP: These are the cases in which we predicted YES, and actually they DO.
  - TN: These are the cases in which we predicted NO, and actually they DO.
  - FP: These are the cases in which we predicted YES, but actually they DON'T.
  - FN: These are the cases in which we predicted NO, but actually they DON'T.

- $Specificity = \frac{TN}{TN+FP}$ ***It measures the number of correct negative predictions divided by the total number of negatives.***

Confusion Matrix – Precision

- A confusion matrix is a table that is used to describe the performance of a ***binary classification model*** on a set of data (usually **validation**/**testing**) for which the true values of a target variable are known.

Confusion Matrix		Predicted	
		Positive (Yes)	Negative (No)
Actual	Positive (Yes, 1)	True Positive (TP)	False Negative (FN)
	Negative (No, 0)	False Positive (FP)	True Negative (TN)

Precision:

0.90-1.0 = excellent (A)
0.80-0.90 = good (B)
0.70-0.80 = fair (C)
0.60-0.70 = poor (D)
0.50-0.60 = fail (F)

  - TP: These are the cases in which we predicted YES, and actually they DO.
  - TN: These are the cases in which we predicted NO, and actually they DO.
  - FP: These are the cases in which we predicted YES, but actually they DON'T.
  - FN: These are the cases in which we predicted NO, but actually they DON'T.

- $Precision = \frac{TP}{TP+FP}$ ***It measures the number of correct positive predictions divided by the total number of positive predictions.***

Confusion Matrix - Negative Predictive Value (NPV)

- A confusion matrix is a table that is used to describe the performance of a ***binary classification model*** on a set of data (usually **validation**/**testing**) for which the true values of a target variable are known.

Confusion Matrix		Predicted	
		Positive (Yes)	Negative (No)
Actual	Positive (Yes, 1)	True Positive (TP)	False Negative (FN)
	Negative (No, 0)	False Positive (FP)	True Negative (TN)

NPV:

0.90-1.0 = excellent (A)
0.80-0.90 = good (B)
0.70-0.80 = fair (C)
0.60-0.70 = poor (D)
0.50-0.60 = fail (F)

  - TP: These are the cases in which we predicted YES, and actually they DO.
  - TN: These are the cases in which we predicted NO, and actually they DO.
  - FP: These are the cases in which we predicted YES, but actually they DON'T.
  - FN: These are the cases in which we predicted NO, but actually they DON'T.

- $NPV = \frac{TN}{TN+FN}$ ***It measures the number of correct negative predictions divided by the total number of negative predictions.***

Confusion Matrix – F1 score

- F1 score is the harmonic mean of precision and recall, i.e., the reciprocal of the arithmetic mean of the reciprocals of the given set of observations

- $$\boldsymbol{F1\ Score} = \left(\frac{1}{\frac{\frac{1}{Recall}+\frac{1}{Precision}}{2}}\right) = \frac{2}{\frac{1}{Recall}+\frac{1}{Precision}} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- F1 Score might be a better measure to use if ***there is an uneven class distribution*** (a large number of Actual Negatives, i.e., 0; a small number of Actual Positives, i.e., 1).

- F1 score reaches its best value at ***1*** and worst at ***0***.

- The higher F1-score, the high value for both recall and precision.

Confusion Matrix – Matthew's Correlation Coefficient

Matthew's correlation coefficient is calculated as follows:

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$

The MCC takes values between -1 and 1. A score of 1 indicates perfect agreement.

The F1-score is a very popular metric for imbalanced class problems. However, MCC is a more complete measure, compared to F1, as MCC involves, TP, TN, FP, and FN.

Highly Recommended for Your Model Evaluations

ROC Curves and ROC AUC

- A binary classifier can have a high accuracy on a dataset but have poor class recall ($Sensitivity = \frac{TP}{TP+FN}$) and precision ($Precision = \frac{TP}{TP+FP}$). ***What does it means in your dataset?***

- A Receiver Operating Characteristic (**ROC**) curve is created by plotting the fraction of TPs (TP rate) versus the fraction of FPs (FP rate).

- The FP rate (X-axis) can be plotted on the horizontal axis and the TP rate (Y-axis).

- Consider a binary classifier that could predict if a website visitor is likely to click on a banner ad:

ROC Curves and ROC AUC

- Consider a binary classifier that could predict if a website visitor is likely to click on a banner ad:

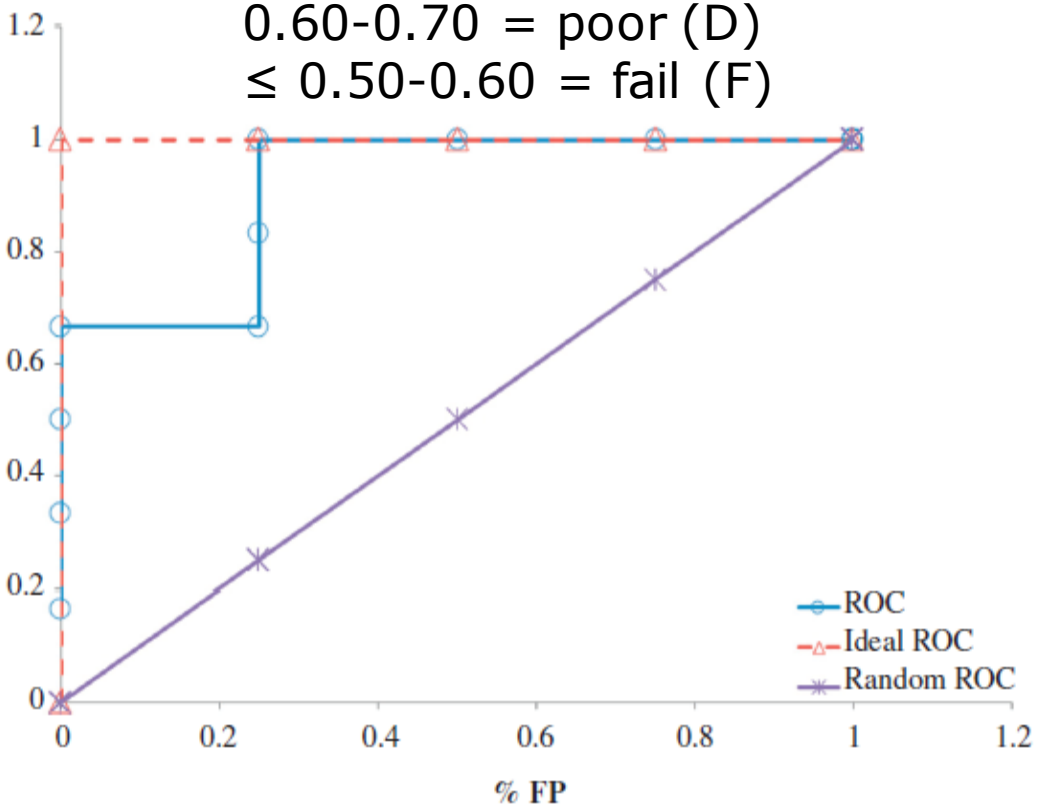
Actual Class	Predicted Class	Type (TP, FP, TN, FN)	Number of FP	Number of TP	FP Rate (X) = FP %	TP Rate (Y) = TP %
Response	Response	TP	0	1	0	0.166666667
Response	Response	TP	0	2	0	0.333333333
Response	Response	TP	0	3	0	0.5
Response	Response	TP	0	4	0	0.666666667
No Response	Response	FP	1	4	0.25	0.666666667
Response	Response	TP	1	5	0.25	0.833333333
Response	Response	TP	1	6	0.25	1
No Response	Response	FP	2	6	0.5	1
No Response	Response	FP	3	6	0.75	1
No Response	Response	FP	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1

ROC Curves and ROC AUC

Actual Class	Predicted Class	Type (TP, FP, TN, FN)	Number of FP	Number of TP	FP Rate (X) = FP %	TP Rate (Y) = TP %
Response	Response	TP	0	1	0	0.166666667
Response	Response	TP	0	2	0	0.333333333
Response	Response	TP	0	3	0	0.5
Response	Response	TP	0	4	0	0.666666667
No Response	Response	FP	1	4	0.25	0.666666667
Response	Response	TP	1	5	0.25	0.833333333
Response	Response	TP	1	6	0.25	1
No Response	Response	FP	2	6	0.5	1
No Response	Response	FP	3	6	0.75	1
No Response	Response	FP	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1
No Response	No Response	TN	4	6	1	1

Area Under Curve (AUC):

0.90-1.0 = excellent (A)
0.80-0.90 = good (B)
0.70-0.80 = fair (C)
0.60-0.70 = poor (D)
≤ 0.50-0.60 = fail (F)



Classification metrics

See the Classification metrics section of the user guide for further details.

metrics.accuracy_score(y_true, y_pred, *[, ...])	Accuracy classification score.
metrics.auc(x, y)	Compute Area Under the Curve (AUC) using the trapezoidal rule.
metrics.average_precision_score(y_true, ...)	Compute average precision (AP) from prediction scores.
metrics.balanced_accuracy_score(y_true, ...)	Compute the balanced accuracy.
metrics.brier_score_loss(y_true, y_prob, *)	Compute the Brier score loss.
metrics.classification_report(y_true, y_pred, *)	Build a text report showing the main classification metrics.
metrics.cohen_kappa_score(y1, y2, *[, ...])	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix(y_true, y_pred, *)	Compute confusion matrix to evaluate the accuracy of a classification.
metrics.dcg_score(y_true, y_score, *[, k, ...])	Compute Discounted Cumulative Gain.
metrics.det_curve(y_true, y_score[, ...])	Compute error rates for different probability thresholds.
metrics.f1_score(y_true, y_pred, *[, ...])	Compute the F1 score, also known as balanced F-score or F-measure.
metrics.fbeta_score(y_true, y_pred, *, beta)	Compute the F-beta score.
metrics.hamming_loss(y_true, y_pred, *[, ...])	Compute the average Hamming loss.
metrics.hinge_loss(y_true, pred_decision, *)	Average hinge loss (non-regularized).
metrics.jaccard_score(y_true, y_pred, *[, ...])	Jaccard similarity coefficient score.
metrics.log_loss(y_true, y_pred, *[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef(y_true, y_pred, *)	Compute the Matthews correlation coefficient (MCC).
metrics.multilabel_confusion_matrix(y_true, ...)	Compute a confusion matrix for each class or sample.
metrics.ndcg_score(y_true, y_score, *[, k, ...])	Compute Normalized Discounted Cumulative Gain.
metrics.precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds.
metrics.precision_recall_fscore_support(...)	Compute precision, recall, F-measure and support for each class.
metrics.precision_score(y_true, y_pred, *[, ...])	Compute the precision.
metrics.recall_score(y_true, y_pred, *[, ...])	Compute the recall.
metrics.roc_auc_score(y_true, y_score, *[, ...])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
metrics.roc_curve(y_true, y_score, *[, ...])	Compute Receiver operating characteristic (ROC).
metrics.top_k_accuracy_score(y_true, y_score, *)	Top-k Accuracy classification score.
metrics.zero_one_loss(y_true, y_pred, *[, ...])	Zero-one classification loss.