Webinar Series, 7 Nopember 2020
PROGRAM PASCASARJANA TERAPAN
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

Workshop & Tutorial
Data Mining with Python

Classification & Validation Model

Ali Ridho Barakbah

Knowledge Engineering Laboratory
Department of Information and Computer Engineering
Politeknik Elektronika Negeri Surabaya

Classification: Definition

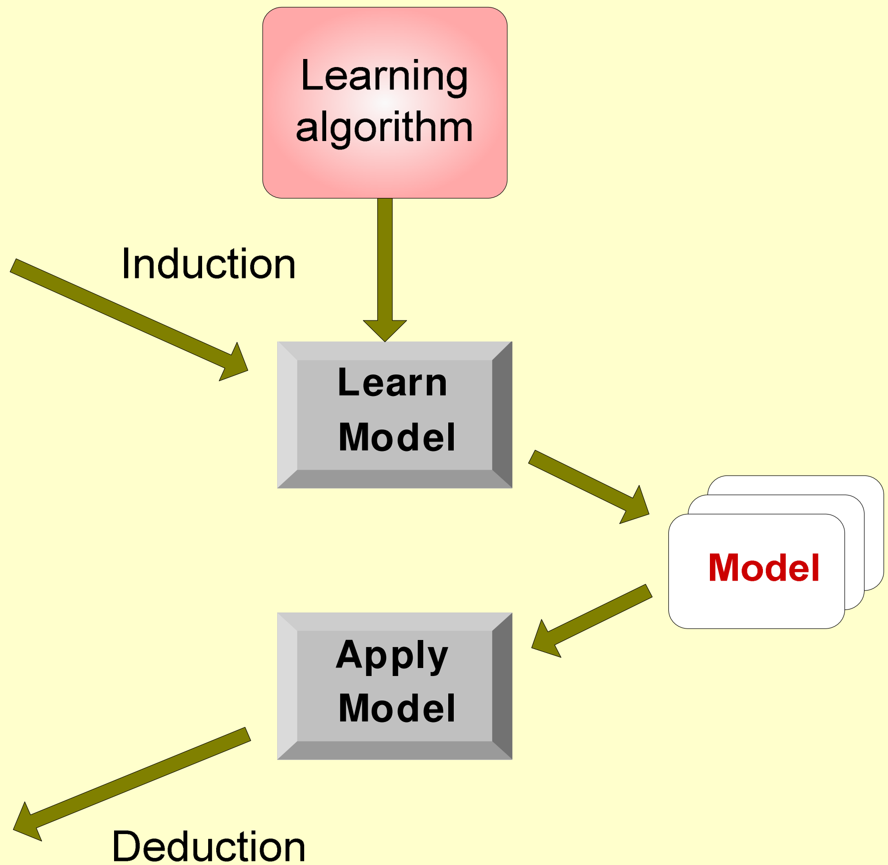
- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the *class*.
- Find a *model* for class attribute as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.
  - A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

Tan, Steinbach, Kumar, *Introduction to Data Mining*

Illustrating Classification Task

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



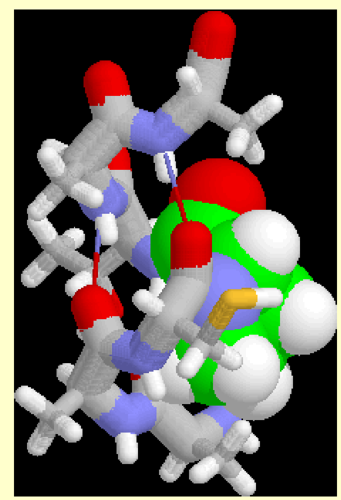
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

Examples of Classification Task

- Predicting tumor cells as benign or malignant

- Classifying credit card transactions as legitimate or fraudulent
- Classifying secondary structures of protein as alpha-helix, beta-sheet, or random coil



- Categorizing news stories as finance, weather, entertainment, sports, etc

Classification using Nearest Neighbors (NN)

- A simple method to classify a new data based on similarity with labeled data
- Similarity usually uses the distance metric
- The unit of distance generally uses the Euclidian
- Has several names: lazy algorithm, memory-based, instance-based, exemplar-based, case-based, experience-based

1-NN Algorithm

- Calculate the distance between a new data to training data
- Determine 1 nearest training data
- Classify a new data into the label of the 1 nearest training data

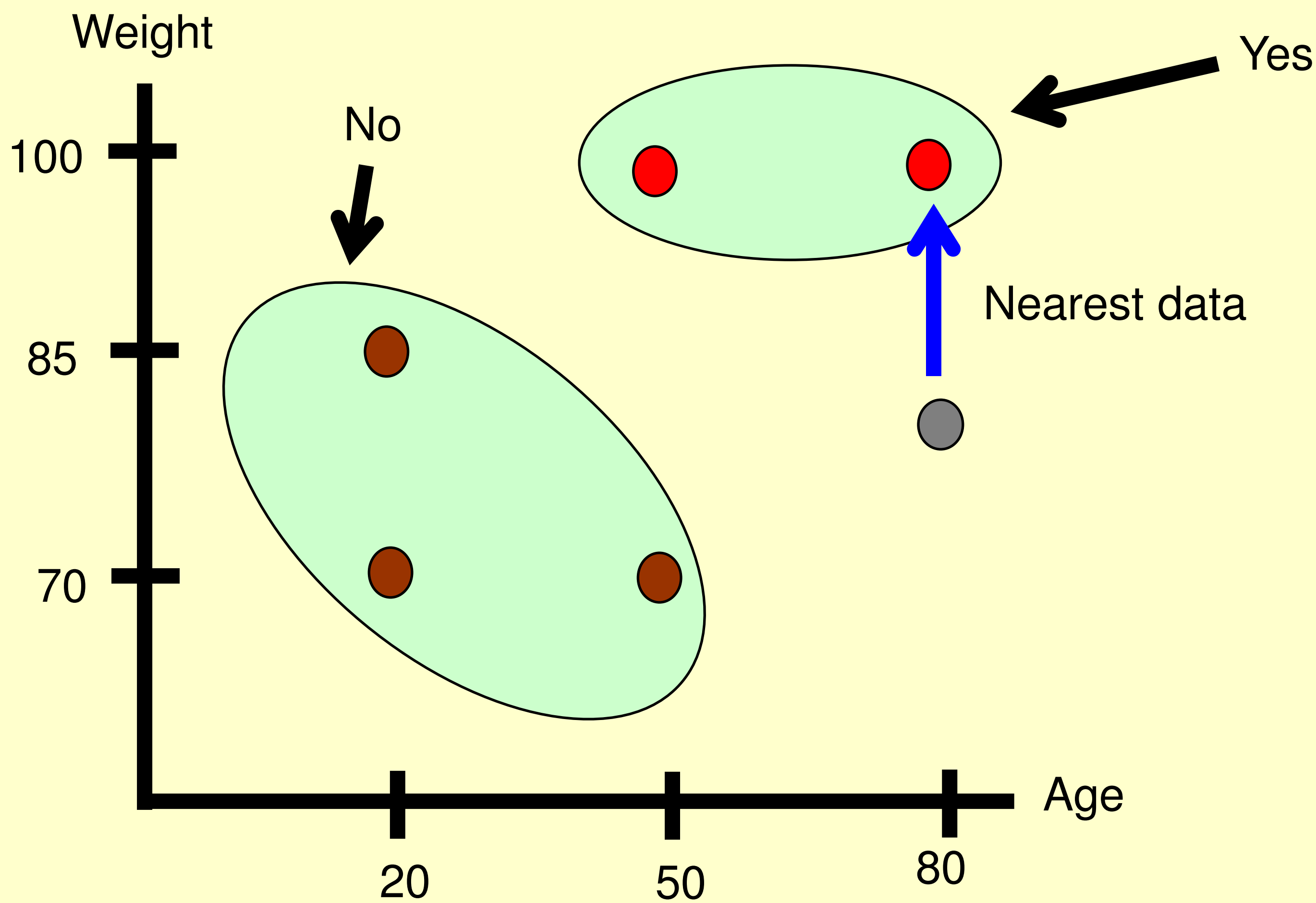
Example

Age	Weight	Hypertension
20	70	No
20	85	No
50	70	No
50	100	Yes
80	100	Yes

80	85	?

a new data

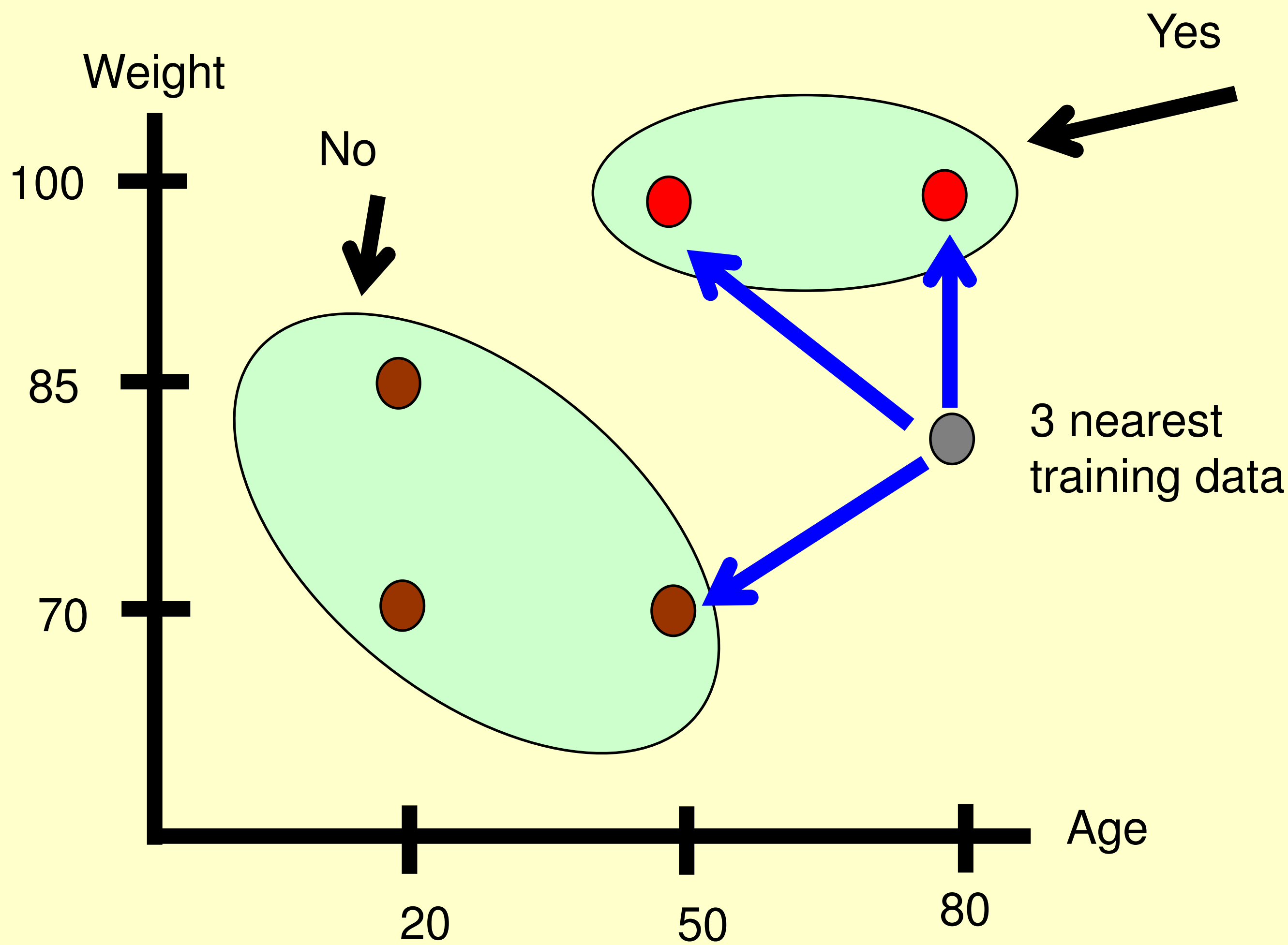
Classification with 1-NN



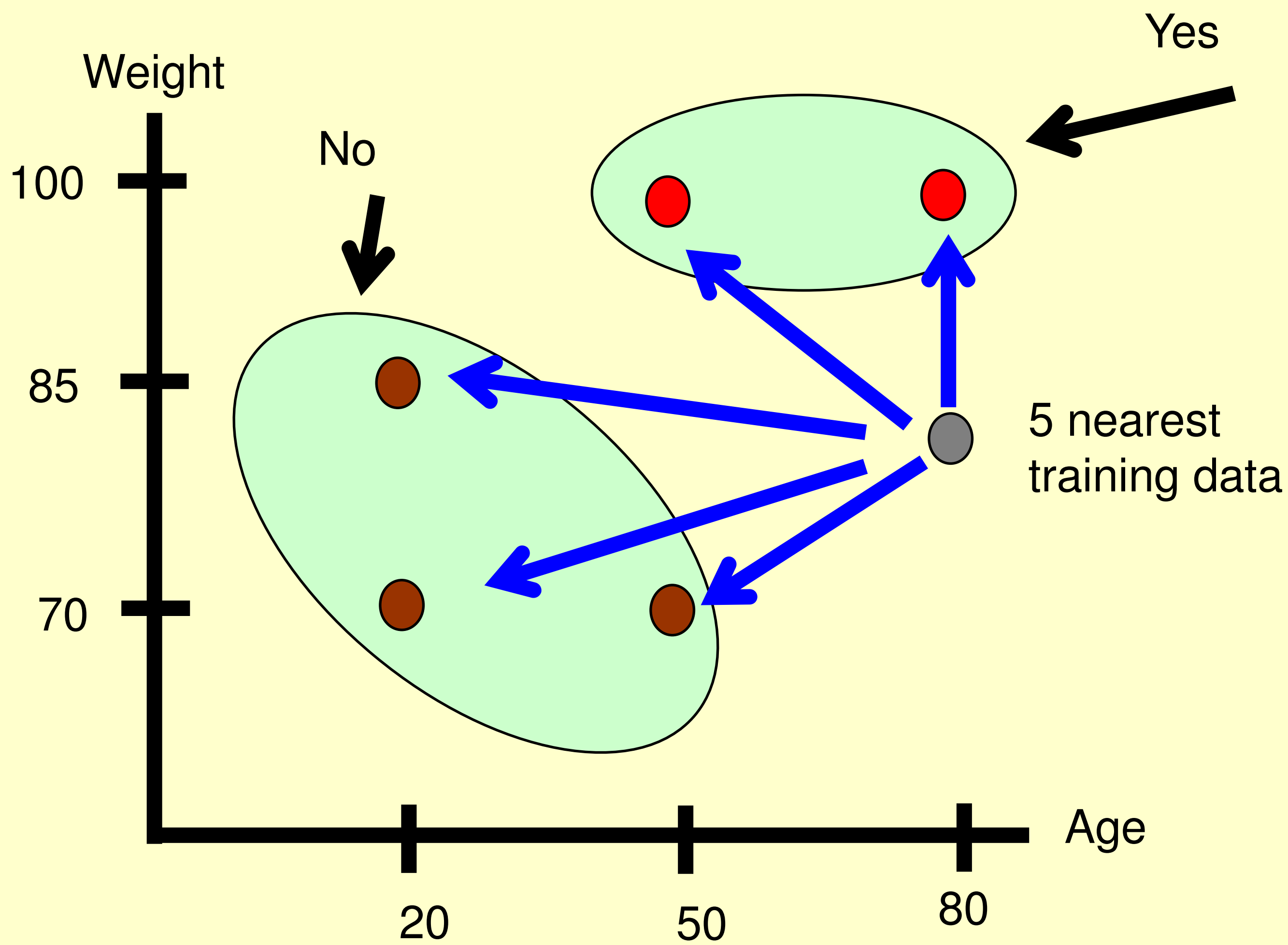
k-NN Algorithm

- Determine k
- Calculate the distance between a new data to all training data
- Find k nearest training data
- Vote class labels of the k nearest training data and classify a new data into the winning vote class label

For example, k=3

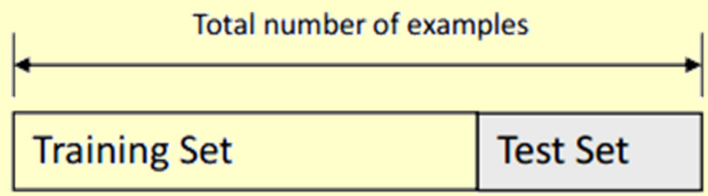


For example, k=5



Validation Model - Holdout Method

- Split dataset into two groups
  - Training set: used to train the classifier
  - Test set: used to estimate the error rate of the trained classifier

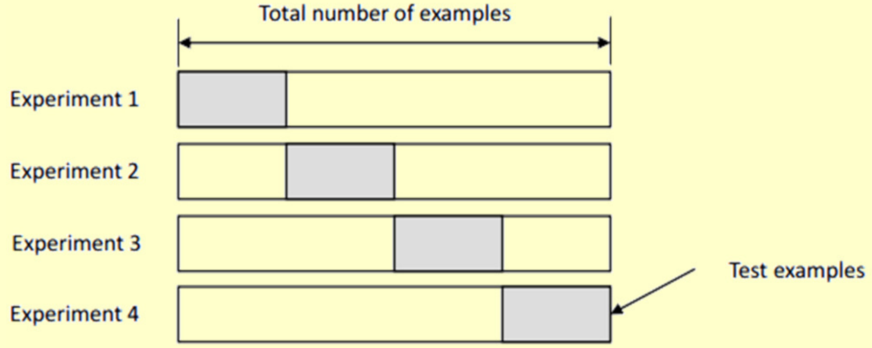


- The holdout method has two basic drawbacks
  - In problems where we have a sparse dataset we may not be able to afford the "luxury" of setting aside a portion of the dataset for testing
  - Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an "unfortunate" split

Ricardo Gutierrez-Osuna, *Pattern Analysis*, CSE@TAMU

Validation Model - K-fold Cross Validation

- Create a K-fold partition of the dataset
  - For each of K experiments, use K - 1 folds for training and a different fold for testing
  - This procedure is illustrated in the following figure for K = 4



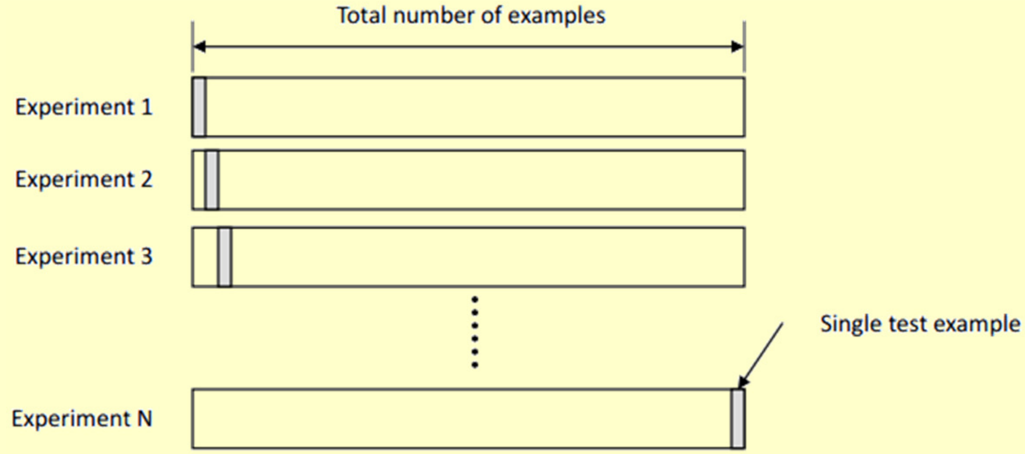
- K-Fold cross validation is similar to random subsampling
  - The advantage of KFCV is that all the examples in the dataset are eventually used for both training and testing
  - As before, the true error is estimated as the average error rate on test examples

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

Ricardo Gutierrez-Osuna, *Pattern Analysis*, CSE@TAMU

Validation Model - Leave-one-out Cross Validation

- LOO is the degenerate case of KFCV, where K is chosen as the total number of examples
  - For a dataset with *N* examples, perform ?? Experiments
  - For each experiment use *N* - 1 examples for training and the remaining example for testing



  - As usual, the true error is estimated as the average error rate on test examples

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i$$

Ricardo Gutierrez-Osuna, *Pattern Analysis*, CSE@TAMU

Klasifikasi – (Nearest Neighbors)

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

dataset = pd.read_csv('titanic_all.csv')
data = dataset[['Age', 'Fare', 'Survived']]
data=data.dropna()

train_data=data[['Age', 'Fare']]
train_label=data[['Survived']]

kNN=KNeighborsClassifier(n_neighbors=3, weights='distance')
test_data=[[35, 50]]
kNN.fit(train_data, np.ravel(train_label))
class_result=kNN.predict(test_data)

print('Hasil klasifikasi = ', class_result.item())
```

```
Hasil klasifikasi =  1
```

sklearn (https://scikit-learn.org/):

- conda install scikit-learn
- pip install -U scikit-learn

Klasifikasi dengan Validasi

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

dataset = pd.read_csv('titanic_all.csv')
data = dataset[['Age', 'Fare', 'Survived']]

train, test = train_test_split(data, test_size=0.2)

train_data=train[['Age','Fare']]
train_label=train[['Survived']]
test_data=test[['Age','Fare']]
test_label=test[['Survived']]

pos_null = train_data.index[train_data.isnull().any(axis=1)].tolist()
train_data = train_data.drop(pos_null)
train_label = train_label.drop(pos_null)

pos_null = test_data.index[test_data.isnull().any(axis=1)].tolist()
test_data = test_data.drop(pos_null)
test_label = test_label.drop(pos_null)

newmin=0
newmax=1
mindata=train_data.min()
maxdata=train_data.max()
train_data = ((train_data-mindata) * (newmax-newmin) / (maxdata-mindata)) + newmin
test_data = ((test_data-mindata) * (newmax-newmin) / (maxdata-mindata)) + newmin

kNN=KNeighborsClassifier(n_neighbors=3, weights='distance')
kNN.fit(train_data, np.ravel(train_label))
class_result=kNN.predict(test_data)

print('Hasil klasifikasi\n', class_result)

error=test_label.loc[:]
error['Class_Result']=class_result
error['Output']=(error['Survived'] == error['Class_Result'])

print('\n\nPerbandingan dengan class label asli:\n', error)

precision_ratio=kNN.score(test_data, test_label)
error_ratio=1-precision_ratio

print('\n\nError ratio = ', error_ratio)
```

```
Hasil klasifikasi
 [1 0 0 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0
1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1
0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 1 1
 0 0 1 1 1 0 0 0 1 1 0 0 0 0 1 1 0 1 0 1
1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0
 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0
1 1 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0
 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1
0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 0 0
 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1 1]
```

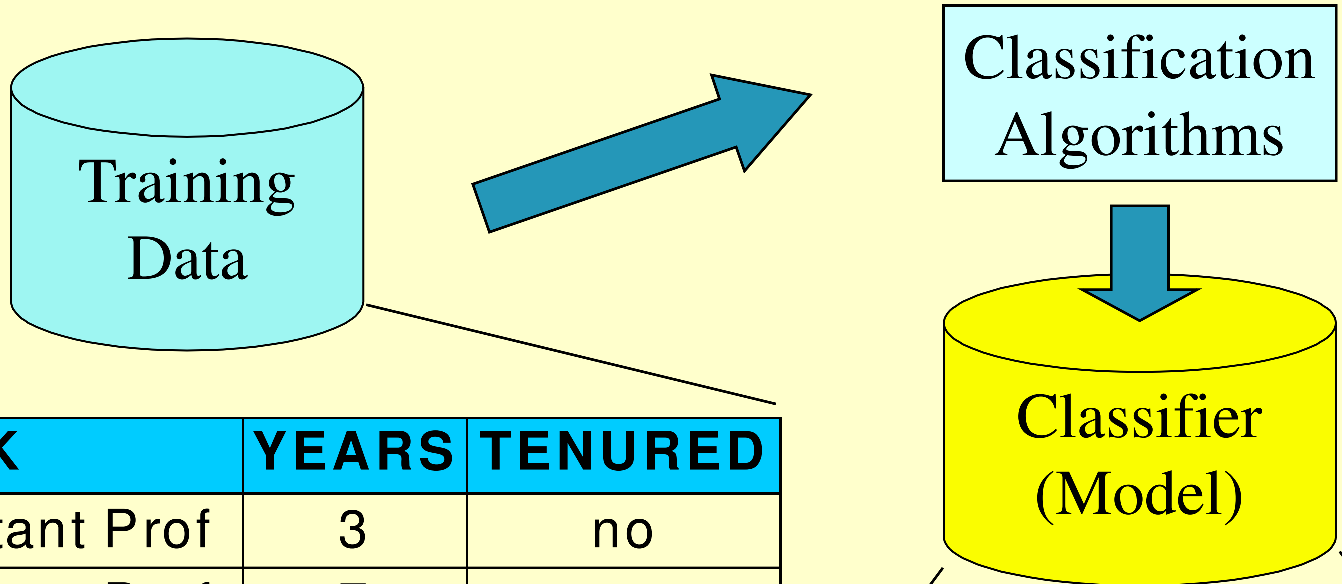
```
Perbandingan dengan class label asli:
      Survived  Class_Result  Output
1283         0             1   False
1279         0             0    True
1202         0             0    True
585          1             0   False
628          0             0    True
...        ...           ...     ...
1265         1             0   False
39           1             0   False
380          1             1    True
871          1             1    True
119          0             1   False

[202 rows x 3 columns]
```

```
Error ratio =  0.38613861386138615
```

What is a Decision Tree?

- An *inductive learning task*
  - Use particular facts to make more generalized conclusions
- A predictive model based on a branching series of Boolean tests
  - These smaller Boolean tests are less complex than a one-stage classifier

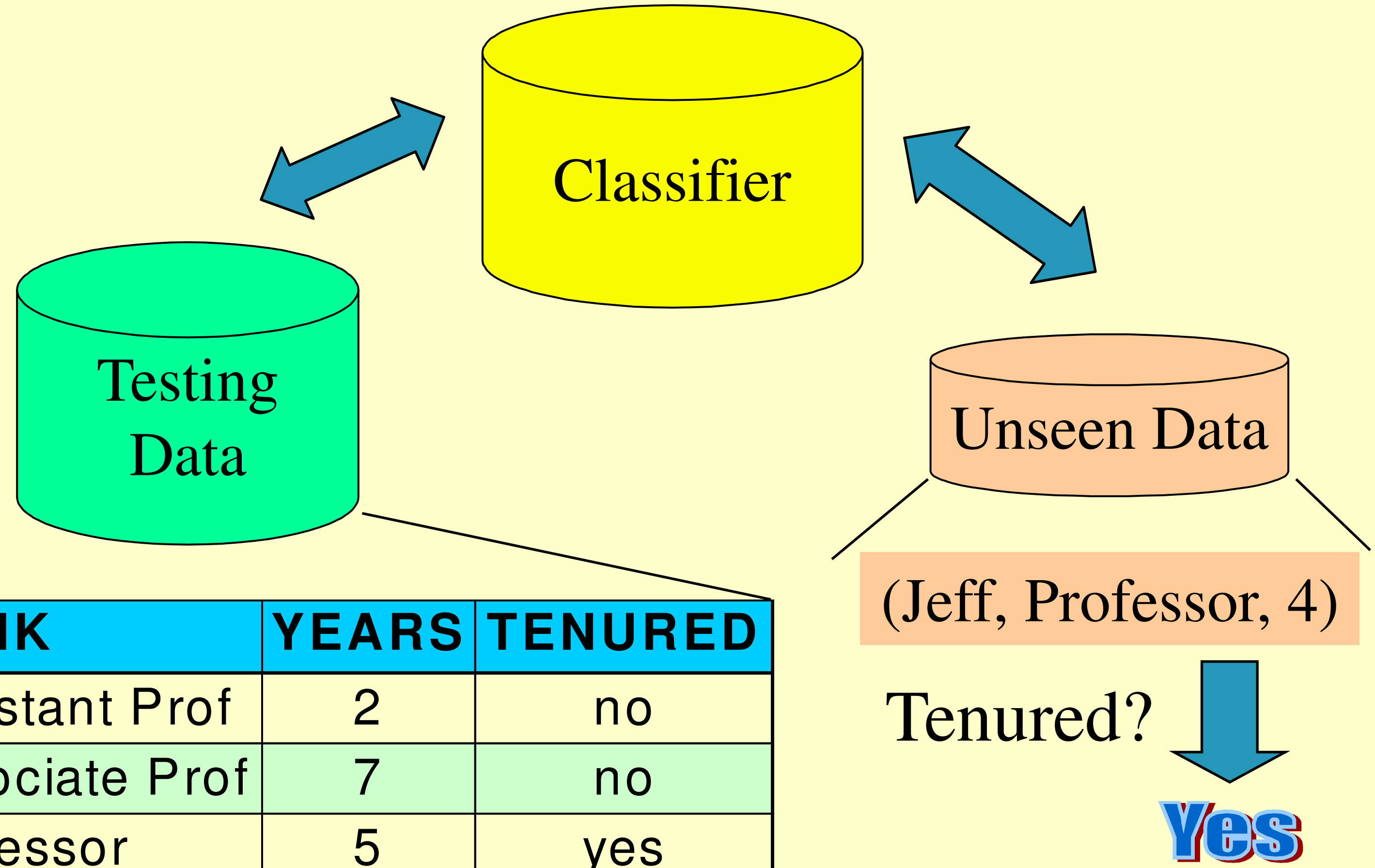
Process (1): Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

IF rank = 'professor'
OR years > 6
THEN tenured = 'yes'

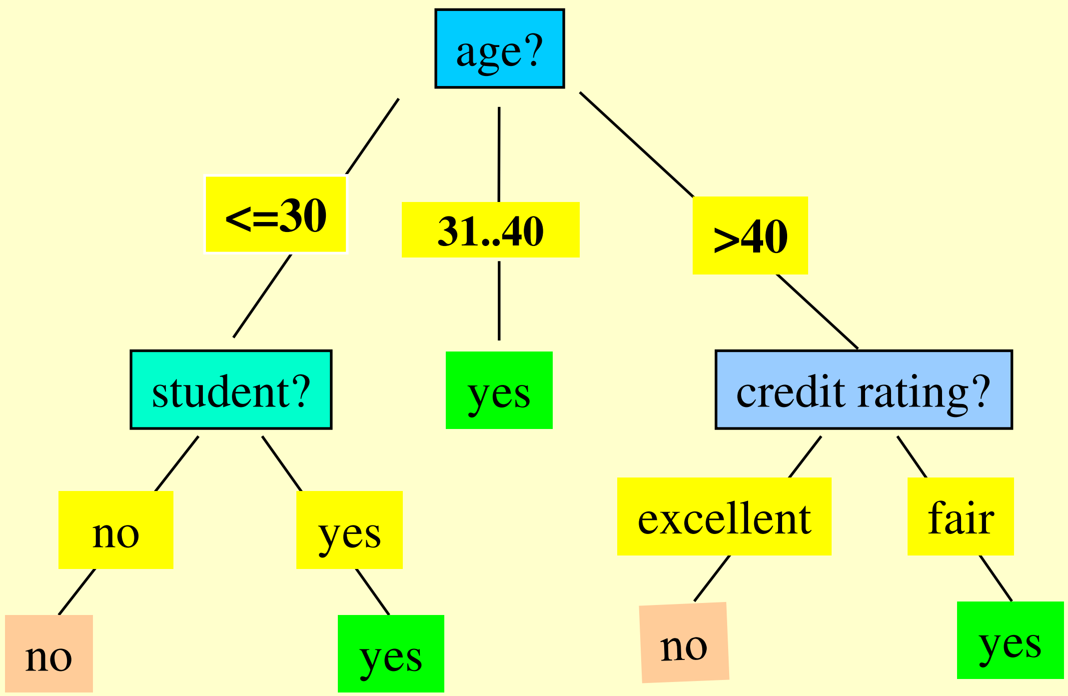
Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Decision Tree Induction: An Example

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:

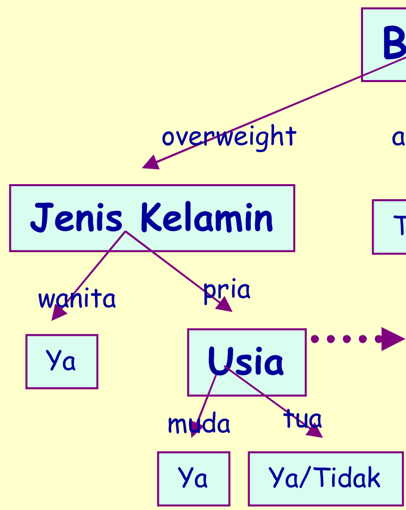


age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Concept of Decision Tree

Data → **Decision Tree** → **Rule**

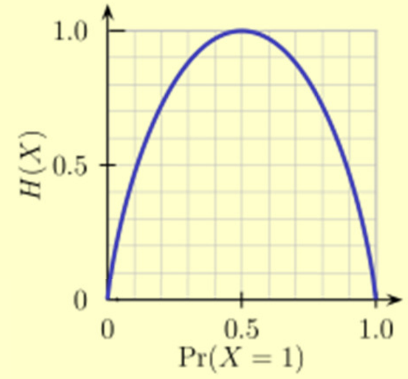
Nama	Usia	Berat	Kelamin	Hipertensi
Ali	muda	overweight	pria	ya
Edi	muda	underweight	pria	tidak
Annie	muda	average	wanita	tidak
Budiman	tua	overweight	pria	tidak
Herman	tua	overweight	pria	ya
Didi	muda	underweight	pria	tidak
Rina	tua	overweight	wanita	ya
Gatot	tua	average	pria	tidak



R1: IF berat=average v berat=underweight THEN hipertensi=tidak
R2: IF berat=overweight^kelamin=wanita THEN hipertensi=ya
R3: IF berat=overweigt^kelamin=pria^ usia=muda THEN hipertensi=ya
R4: IF berat=overweigt^kelamin=pria^ usia=tua THEN hipertensi=tidak

Brief Review of Entropy

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable *Y* taking *m* distinct values $\{y_1, \ldots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$



m = 2

Konversi Numerical Attribute ke Categorical Attibute (dengan Gini Index)

- If a data set *D* contains examples from *n* classes, gini index, *gini*(*D*) is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class *j* in *D*

- If a data set *D* is split on A into two subsets D_1 and D_2, the *gini* index *gini*(*D*) is defined as

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

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

Klasifikasi – (Decision Tree)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

dataset = pd.read_csv('titanic_all.csv')
data = dataset[['Age', 'Fare', 'Survived']]

train, test = train_test_split(data, test_size=0.2)

train_data=train[['Age','Fare']]
train_label=train[['Survived']]
test_data=test[['Age','Fare']]
test_label=test[['Survived']]

pos_null = train_data.index[train_data.isnull().any(axis=1)].tolist()
train_data = train_data.drop(pos_null)
train_label = train_label.drop(pos_null)

pos_null = test_data.index[test_data.isnull().any(axis=1)].tolist()
test_data = test_data.drop(pos_null)
test_label = test_label.drop(pos_null)

print('\nTest Data:\n', test_data)

dtc=DecisionTreeClassifier(criterion='entropy', max_depth=3)
dtc.fit(train_data, train_label)

class_result=dtc.predict(test_data)
print('\nClass = \n', class_result)

acc=dtc.score(test_data, test_label)
err=round((1-acc)*100, 2)
print('\nError ratio = ', err, '%')
```

```
Test Data:
        Age      Fare
450   36.0   27.7500
597   49.0    0.0000
618    4.0   39.0000
366   60.0   75.2500
693   25.0    7.2250
...    ...       ...
779   43.0  211.3375
1303  28.0    7.7750
1121  14.0   65.0000
942   27.0   15.0333
236   44.0   26.0000

[204 rows x 2 columns]

Class =
 [0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 0 1
 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0
 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 1 0 0]

Error ratio =  30.88 %
```

Klasifikasi – (Decision Tree dengan Gambar Hirarki)

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import graphviz

dataset = pd.read_csv('titanic_all.csv')
data = dataset[['Sex', 'Age', 'Pclass', 'Fare', 'Survived']]

train, test = train_test_split(data, test_size=0.2)

train_data=train.loc[:,['Sex', 'Age', 'Pclass', 'Fare']]
train_label=train.loc[:,['Survived']]
test_data=test.loc[:,['Sex', 'Age', 'Pclass', 'Fare']]
test_label=test.loc[:,['Survived']]

train_data['Sex'] = train_data['Sex'].astype('category')
train_data['Sex']=train_data['Sex'].cat.codes
test_data['Sex'] = test_data['Sex'].astype('category')
test_data['Sex']=test_data['Sex'].cat.codes

pos_null = train_data.index[train_data.isnull().any(axis=1)].tolist()
train_data = train_data.drop(pos_null)
train_label = train_label.drop(pos_null)

pos_null = test_data.index[test_data.isnull().any(axis=1)].tolist()
test_data = test_data.drop(pos_null)
test_label = test_label.drop(pos_null)

print('\nTest Data:\n', test_data)

dtc=DecisionTreeClassifier(criterion='entropy', max_depth=3)
dtc.fit(train_data, train_label)

class_result=dtc.predict(test_data)
print('\nClass = \n', class_result)

acc=dtc.score(test_data, test_label)
err=round((1-acc)*100, 2)
print('\nError ratio = ', err, '%')

dot_data = tree.export_graphviz(dtc, out_file=None, feature_names=train_data.columns.values)
graph = graphviz.Source(dot_data, format="png")
graph.render(view=True)
```

```
Test Data:
       Sex   Age  Pclass     Fare
1253    0  31.0       2  21.0000
1199    1  55.0       1  93.5000
6       1  54.0       1  51.8625
807     0  18.0       3   7.7750
232     1  59.0       2  13.5000
...   ...   ...     ...      ...
228     1  18.0       2  13.0000
1142    1  20.0       3   7.9250
175     1  18.0       3   7.8542
636     1  32.0       3   7.9250
759     0  33.0       1  86.5000

[210 rows x 4 columns]

Class =
 [1 0 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0
 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0
 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0
 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 1 0 0
 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 1
 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1]

Error ratio =  11.43 %
```

graphviz (https://graphviz.org):

- pip install graphviz
- `conda install graphviv`
- `conda install python-graphviz`

Kedalaman level=3