



# Data Intensive Systems (DIS) KBH-SW7 E25

## 7. Classification

# PYTHON FOR DATA SCIENCE CHEAT SHEET

## Python Scikit-Learn

### Introduction

Scikit-learn: "sklearn" is a machine learning library for the Python programming language. Simple and efficient tool for data mining, Data analysis and Machine Learning.

Importing Convention - import sklearn

### Preprocessing

#### Data Loading

- **Using NumPy:**  
>>> import numpy as np  
>>> a = np.array([(1,2,3,4),(7,8,9,10)], dtype=int)  
>>> data = np.loadtxt('file\_name.csv', delimiter=',')
- **Using Pandas:**  
>>> import pandas as pd  
>>> df = pd.read\_csv('file\_name.csv', header=0)

#### Train-Test Data

```
>>> from sklearn.model_selection import train_test_split  
  
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

### Data Preparation

#### Standardization

```
>>> from sklearn.preprocessing import StandardScaler  
>>> get_names = df.columns  
>>> scaler = preprocessing.StandardScaler()  
>>> scaled_df = scaler.fit_transform(df)  
>>> scaled_df = pd.DataFrame(scaled_df, columns=get_names)
```

#### Normalization

```
>>> from sklearn.preprocessing import Normalizer  
  
>>> pd.read_csv('File_name.csv')  
>>> x_array = np.array(df['Column1'])  
# Normalize Column1  
>>> normalized_X = preprocessing.normalize([x_array])
```

### Working On Model

#### Model Choosing

##### Supervised Learning Estimator:

- **Linear Regression:**  
>>> from sklearn.linear\_model import LinearRegression  
>>> new\_lr = LinearRegression(normalize=True)
- **Support Vector Machine:**  
>>> from sklearn.svm import SVC  
>>> new\_svc = SVC(kernel='linear')

##### Naive Bayes:

```
>>> from sklearn.naive_bayes import GaussianNB  
>>> new_gnb = GaussianNB()  
  
• KNN:  
>>> from sklearn import neighbors  
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=1)
```

##### Unsupervised Learning Estimator:

- **Principal Component Analysis (PCA):**  
>>> from sklearn.decomposition import PCA  
>>> new\_pca = PCA(n\_components=0.95)
- **K Means:**  
>>> from sklearn.cluster import KMeans  
>>> k\_means = KMeans(n\_clusters=5, random\_state=0)

#### Train-Test Data

##### Supervised:

```
>>> new_lr.fit(X, y)  
>>> knn.fit(X_train, y_train)  
>>> new_svc.fit(X_train, y_train)
```

##### Unsupervised:

```
>>> k_means.fit(X_train)  
>>> pca_model_fit = new_pca.fit_transform(X_train)
```

### Post-Processing

#### Prediction

##### Supervised:

```
>>> y_predict = new_svc.predict(np.random.random((3,5)))  
>>> y_predict = new_lr.predict(X_test)  
>>> y_predict = knn.predict_proba(X_test)
```

##### Unsupervised:

```
>>> y_pred = k_means.predict(X_test)
```

#### Model Tuning

##### Grid Search:

```
>>> from sklearn.grid_search import GridSearchCV  
>>> params = {"n_neighbors": np.arange(1,5), "metric": ["euclidean", "cityblock"]}  
>>> grid = GridSearchCV(estimator=knn, param_grid=params)  
>>> grid.fit(X_train, y_train)  
>>> print(grid.best_score_)  
>>> print(grid.best_estimator_.n_neighbors)
```

##### Randomized Parameter Optimization:

```
>>> from sklearn.grid_search import RandomizedSearchCV  
>>> params = {"n_neighbors": range(1,5), "weights": ["uniform", "distance"]}  
>>> rsearch = RandomizedSearchCV(estimator=knn, param_distributions=params, cv=4, n_iter=8, random_state=5)  
>>> rsearch.fit(X_train, y_train)  
>>> print(rsearch.best_score_)
```

### Evaluate Performance

#### Classification:

##### 1. Confusion Matrix:

```
>>> from sklearn.metrics import confusion_matrix  
>>> print(confusion_matrix(y_test, y_pred))
```

##### 2. Accuracy Score:

```
>>> knn.score(X_test, y_test)  
>>> from sklearn.metrics import accuracy_score  
>>> accuracy_score(y_test, y_pred)
```

#### Regression:

##### 1. Mean Absolute Error:

```
>>> from sklearn.metrics import mean_absolute_error  
  
>>> y_true = [3, -0.5, 2]  
>>> mean_absolute_error(y_true, y_predict)
```

##### 2. Mean Squared Error:

```
>>> from sklearn.metrics import mean_squared_error  
>>> mean_squared_error(y_test, y_predict)
```

##### 3. R<sup>2</sup> Score:

```
>>> from sklearn.metrics import r2_score  
>>> r2_score(y_true, y_predict)
```

#### Clustering:

##### 1. Homogeneity:

```
>>> from sklearn.metrics import homogeneity_score  
>>> homogeneity_score(y_true, y_predict)
```

##### 2. V-measure:

```
>>> from sklearn.metrics import v_measure_score  
>>> metrics.v_measure_score(y_true, y_predict)
```

#### Cross-validation:

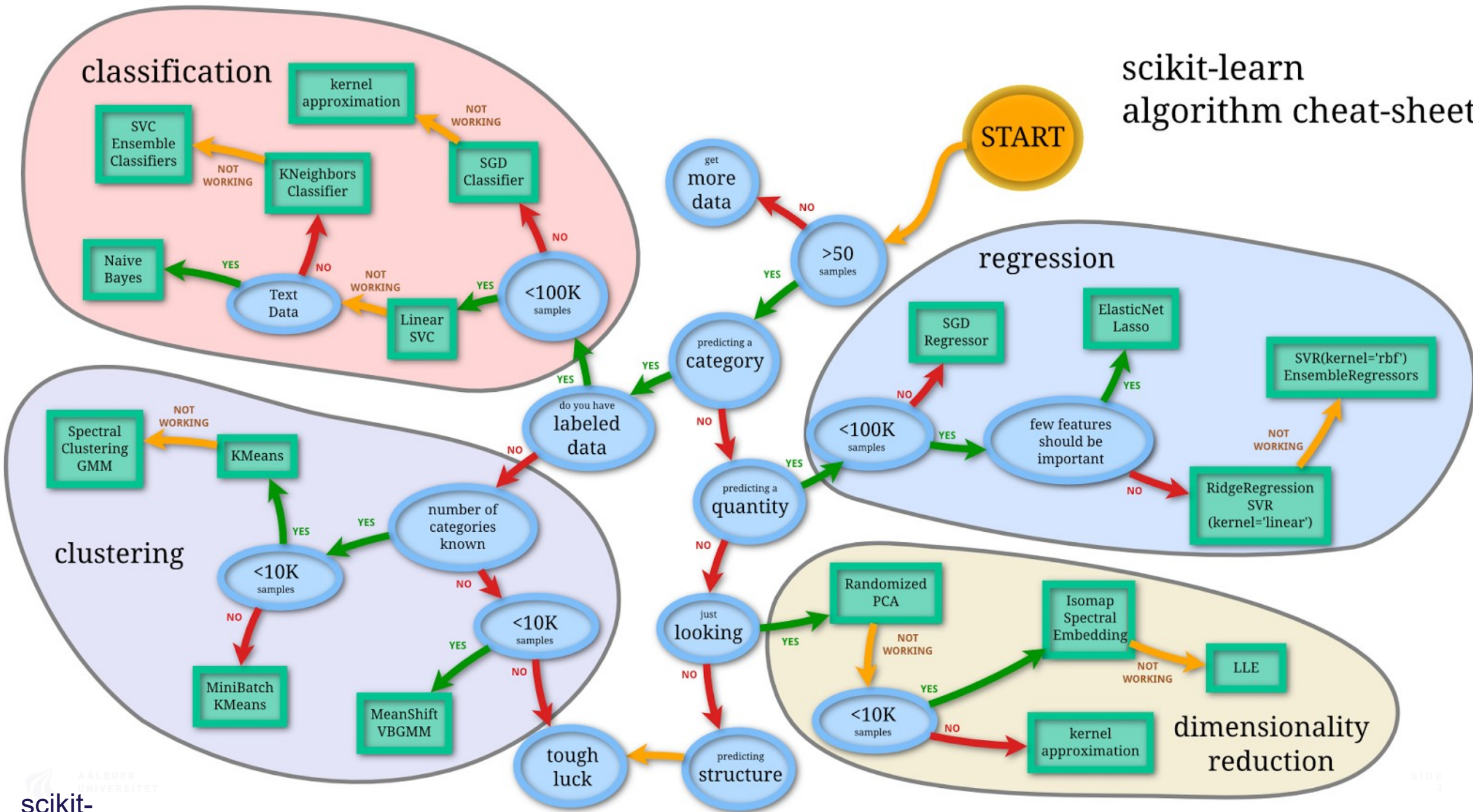
```
>>> from sklearn.cross_validation import cross_val_score  
>>> print(cross_val_score(knn, X_train, y_train, cv=4))  
>>> print(cross_val_score(new_lr, X, y, cv=2))
```

FURTHERMORE:

Python for Data Science Certification Training Course



# scikit-learn algorithm cheat-sheet



# Agenda

- Classification and model evaluation
  - Classification steps
  - Classification performance
- Typical classification models
- Data scaling
- Model evaluation and selection

# Supervised vs. Unsupervised Learning

- ▶ **Supervised learning** generalizes from *known examples* to automate decision-making processes.
  - ▶ **Classification**: Predict a **discrete** value from a *pre-defined* set of class labels
    - › E.g., given a loan applicant, predict if she/he is a *good* or *bad* client. (*Approval* or *rejection*)
    - › More examples: digital recognition from handwritings, fraud detection in banking, spam filtering.
  - ▶ **Regression**: Predict a **continuous** value from a continuous range
    - › E.g., predict the price of a stock
- ▶ **Unsupervised learning** does *not* need any known examples. It works on input data directly. (*Future lectures*)
  - ▶ E.g., similarity-based client grouping, outlier detection for website access patterns

# Classification: Three Major Steps

## 1. **Model construction**: describing a set of *predetermined* classes

- Each tuple/sample is assumed to belong to a predefined class, as determined by the **class label** column
- The set of tuples used for model construction is **training set**
- A model is created using an algorithm on selected features.
  - › Simply speaking, features are columns or generated based on columns.
  - › Not all columns are used for creating a model.
  - › **Feature selection** or **engineering** decides which features (columns) to be used.
- The model is represented as classification rules, a decision tree, or mathematical formulae.
- The model can predict the class label for a given (unseen) tuple

## 2. **Model validation**

## 3. **Model application/test**

# Classification: Three Steps (cont.)

2. **Model validation**: to *evaluate* how good your model is for the given validation data set; to tune the parameters of a model (*parameter tuning*).

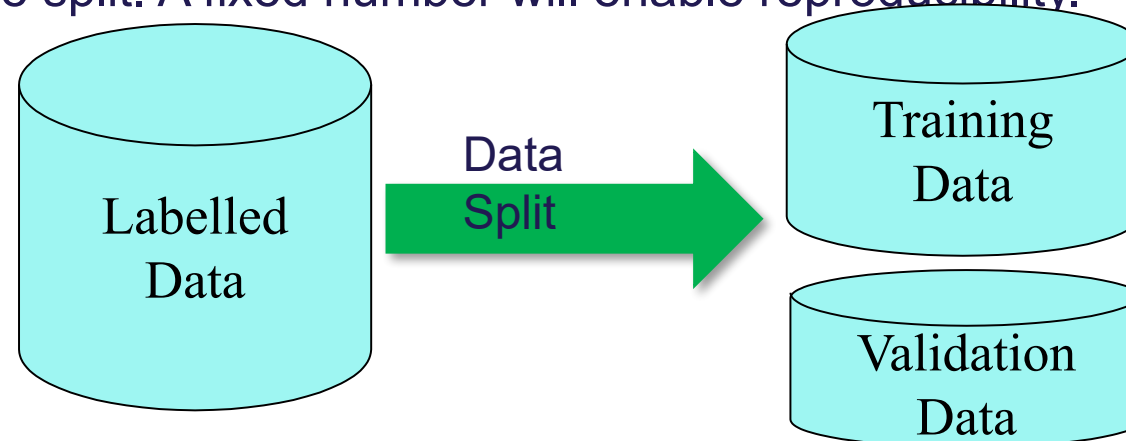
- **Estimate accuracy** of the model using **validation set** (data set for validation)
  - › The known label of test sample is compared with the classified result from the model
  - › **Accuracy** rate is the percentage of test set samples that are correctly classified by the model
  - › **Validation set** should be independent of training set (otherwise **overfitting**)
- If the accuracy is *acceptable*, the model can be used to *classify new/unseen data* (model application/test)
- Different models may be compared for selection of the best
- **NB**: Sometimes validation is also called test (e.g., in sklearn)

3. **Model application/test**: for classifying future or *unseen* objects

- For those objects, you don't know their classes!

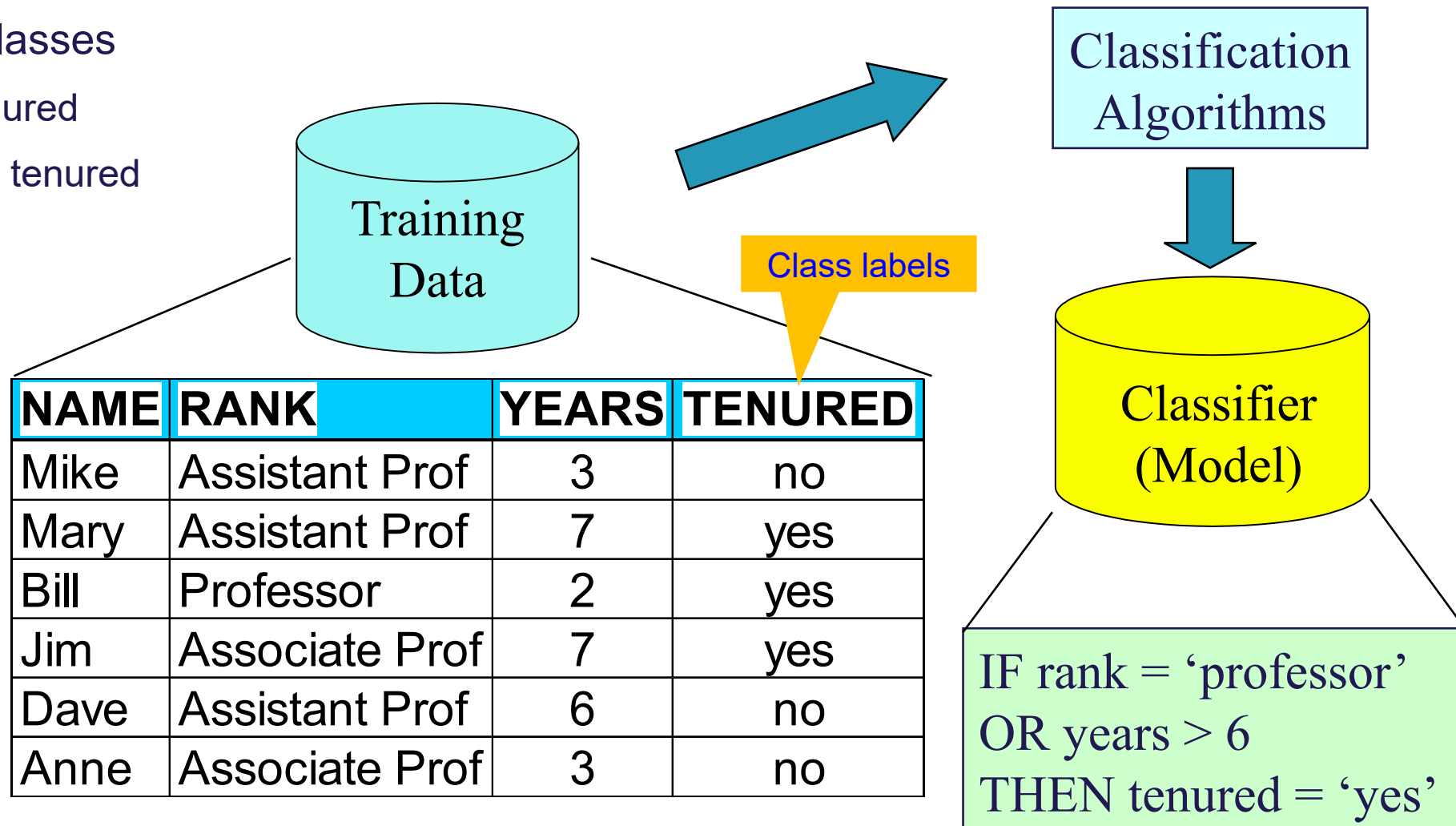
# Split Labelled Data

- `sklearn.model_selection.train_test_split`
- `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)`
  - `X_train`: features of training data; `y_train`: class labels of training data
  - `X_test`: features of validation data; `y_test`: class labels of validation data
  - `test_size`: percentage of validation data
  - `random_state`: randomization of the split. A fixed number will enable reproducibility.
- Different ways of split can result in different models and performance
  - We will see more next week



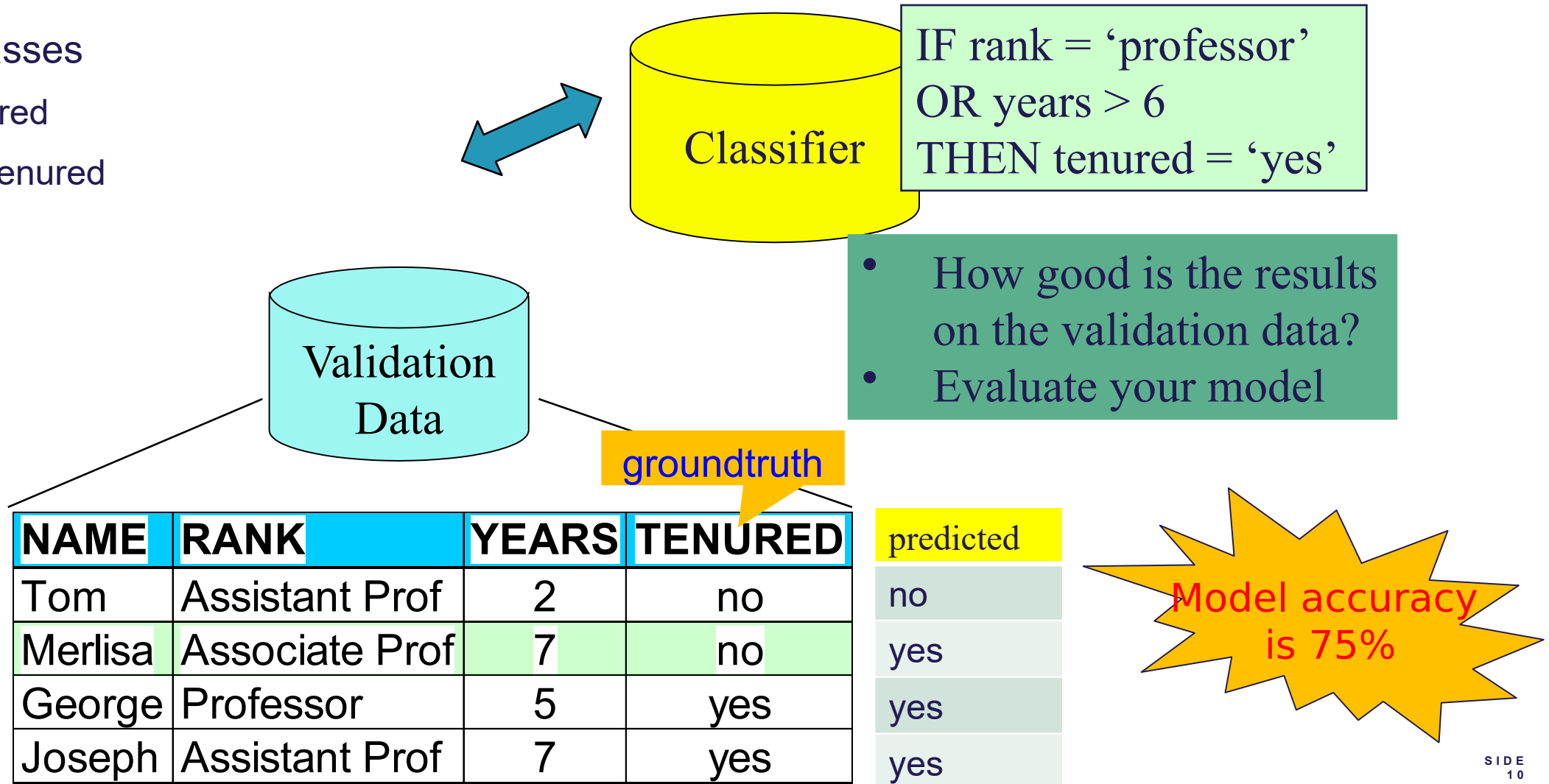
# Step 1: Model Construction

- ▶ Two classes
  - ▶ Tenured
  - ▶ Not tenured



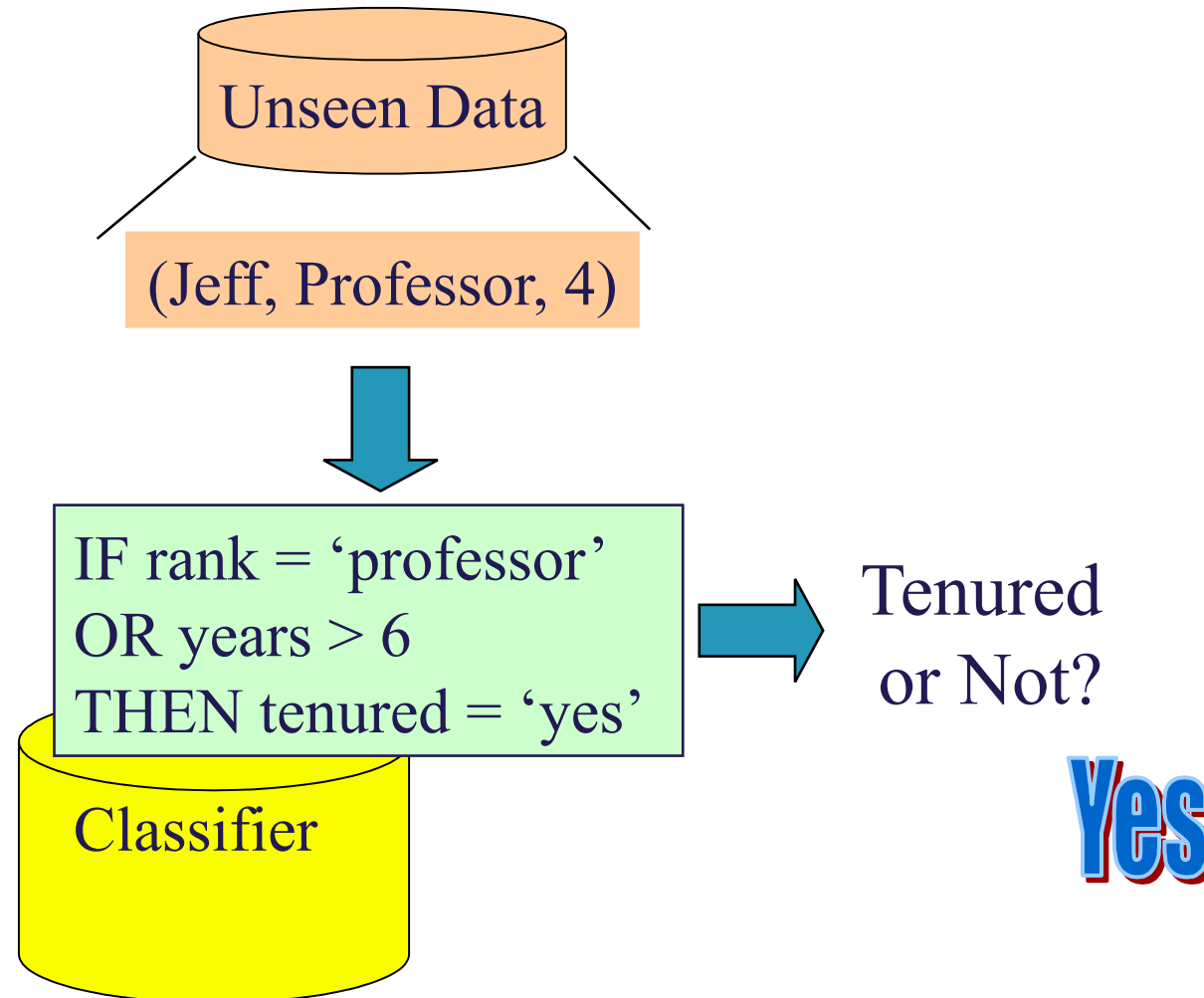
# Step 2: Model Validation

- Two classes
  - Tenured
  - Not tenured



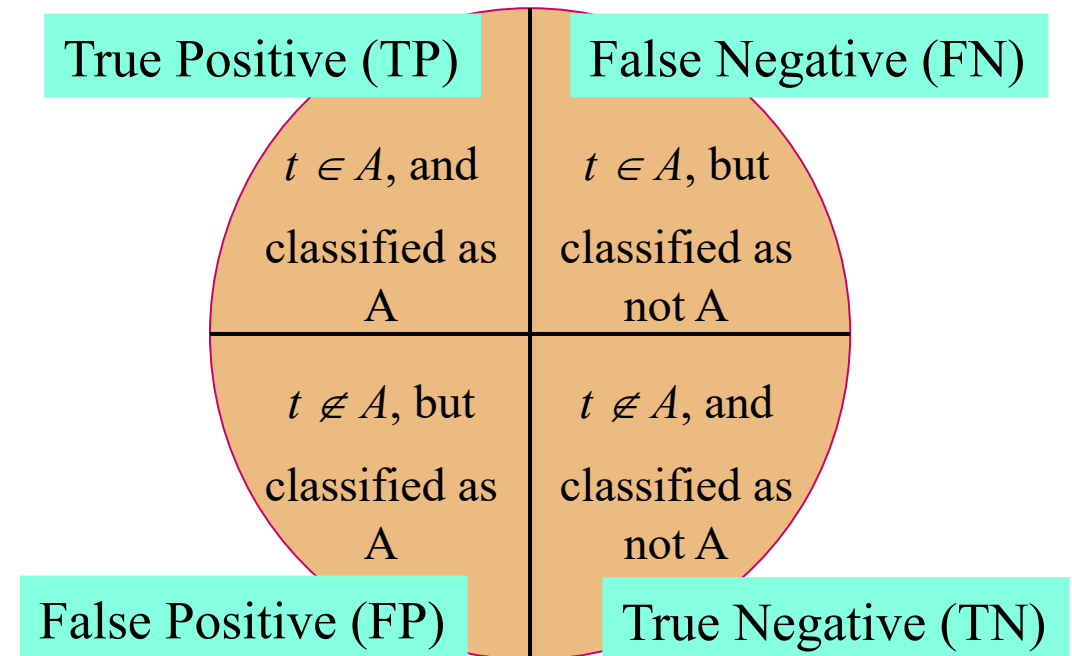
# Step 3: Model Application/Test

- ▶ Two classes
  - ▶ Tenured
  - ▶ Not tenured



# Classifier Performance

- ▶ Consider a binary classifier
  - ▶ Target class A: *positive*
  - ▶ Target class NOT A: *negative*
- ▶ Consider the *ground truth* and classification result. There are four cases.
  - ▶ Put it simply, ground truth is the 'fact' we know.



# Performance Metrics (for One Class A)

## ► **Precision** (exactness)

- How often the positive classification is correct.
- $TP / (TP+FP)$

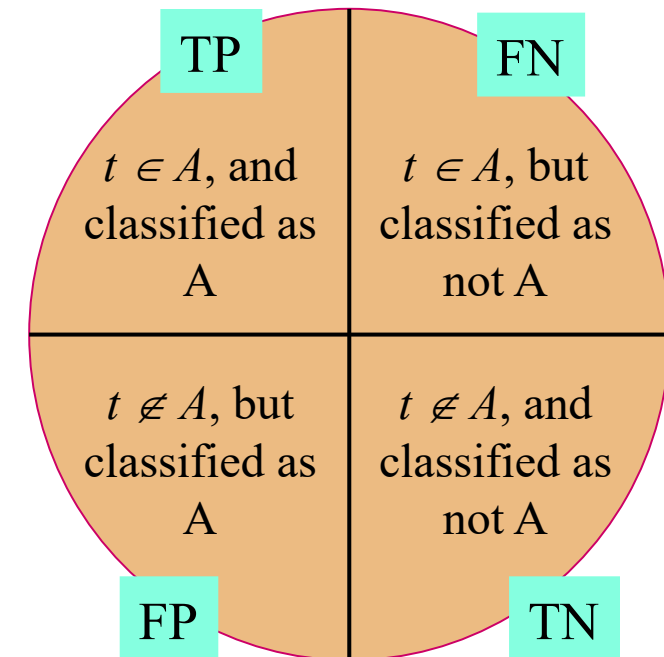
## ► **Recall** (completeness)

- How many of the actual positive cases are classified as positive.
- $TP / (TP+FN)$

## ► **Accuracy**

- The fraction of the time when the classifier gives the correct classification.
- $(TP+TN) / (TP+FP+TN+FN)$

This is easy to understand  
for *binary classification*.



# Precision, Recall and F-measures

- Perfect score for Precision and Recall is 1
- Inverse relationship exists between Precision and Recall
- **F measure** ( $F_1$  or **F-score**): harmonic mean of Precision and Recall

$$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- $F_\beta$ : weighted measure of precision and recall
  - assigns  $\beta$  times as much weight to recall as to precision

$$F = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}$$

What if more than two class labels?

# Confusion Matrix

► Three pre-defined classes A, B, C

► Ground truth and classification result

Object	Ground Truth	Classification Result
object-1	A	A
object-2	B	A
object-3	C	C
object-4	C	C
object-5	B	B
object-6	A	B
object-7	B	B

► Confusion Matrix (N x N for N classes)

► The cell  $M[i, j]$  counts the case that groundtruth  $i$  is classified as  $j$

result

Classification result

	A	B	C	Total	
Ground truth	A	1	1	2	
	B	1	2	3	
	C	0	0	2	
	Total	2	3	2	7

FN for A

FP for A

TP for one class, and TN for others

► **Accuracy** = (TP+TN) / All

► In this example, Accuracy =  $(1+2+2)/7 = 71.4\%$

# Precision/Recall in Confusion Matrix

- ▶ Calculate the precision  $p_i$  for each class  $C_i$ 
  - ▶ Overall precision is the *average* of all  $p_i$ s
- ▶ Calculate the recall  $r_i$  for each class  $C_i$ 
  - ▶ Overall recall is the *average* of all  $r_i$ s
- ▶ Example
  - ▶  $p_A = 30/60 = 1/2$ ,  $r_A = 30/100 = 3/10$
  - ▶  $p_B = 60/120 = 1/2$ ,  $r_B = 60/100 = 3/5$
  - ▶  $p_C = 80/120 = 2/3$ ,  $r_C = 80/100 = 4/5$
  - ▶ Overall precision =  $5/9$ ,  
overall recall =  $17/30$

**Confusion matrix**

Classification result

Ground truth		A	B	C	Total
	A	30	50	20	100
	B	20	60	20	100
	C	10	10	80	100
	Total	60	120	120	300

Recall for A:  $r_A$

Precision for A:  
 $p_A$

# Analyze Your Confusion Matrix

- Essentially, the more zeroes or smaller the numbers on all cells but the diagonal, the better a classifier is. So you may analyze your confusion matrix and tweak your features accordingly.
- Confusion matrix gives strong clues as to where your classifier is going wrong.
  - E.g., if for Class A you can see that the classifier incorrectly predicts Class B for majority of the mislabeled cases, it indicates the classifier is somehow confused between classes A and B.
  - One way to fix this is to add biasing features to improve classification of class A, e.g., more training data of A.

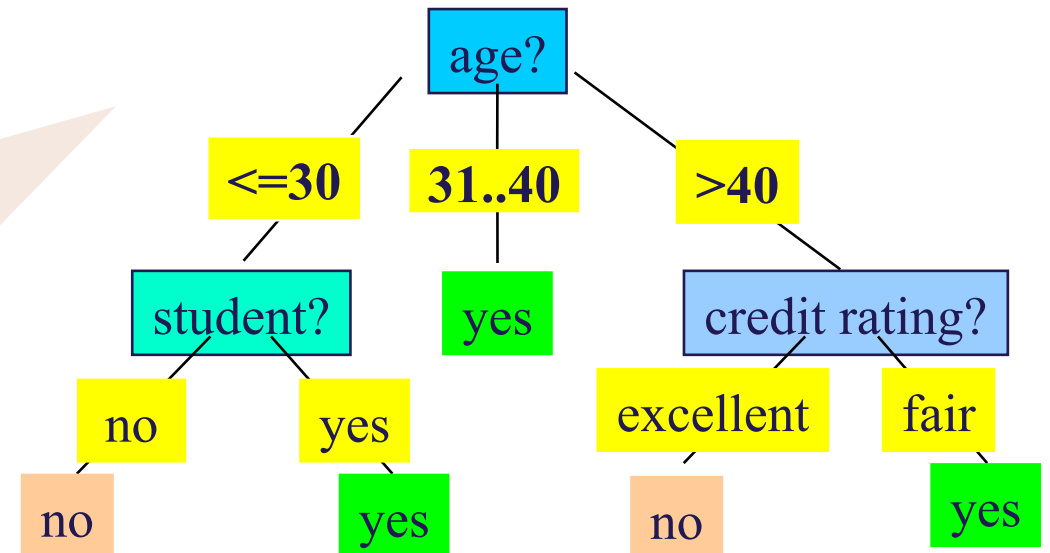
# Agenda

- ▶ Classification and model evaluation
- ▶ **Typical classification models**
  - ▶ Rule based classifier (the previous tenure example)
  - ▶ Decision tree
  - ▶ Random forest
  - ▶ K nearest neighbors (KNN)
- ▶ Data scaling
- ▶ Model evaluation and selection

# Classification Using Decision Trees

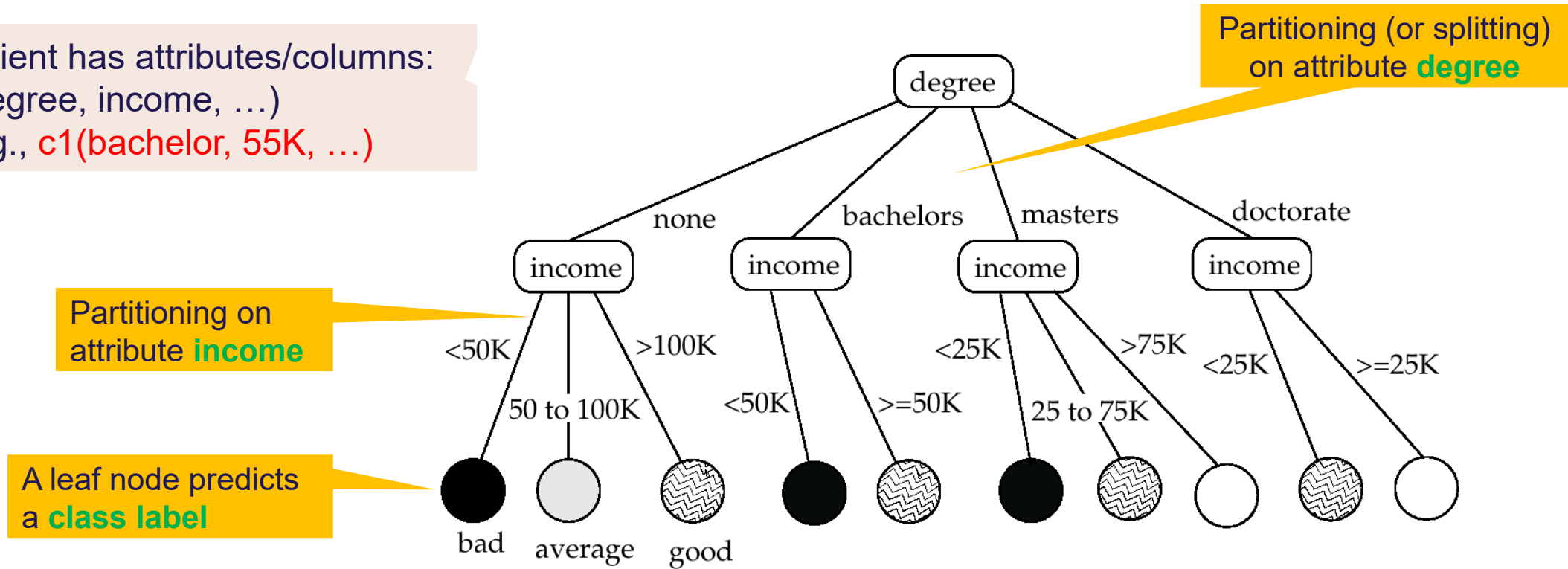
- ▶ The classification model (classifier) is organized as a tree for decision making.
  - ▶ It's thus called a **decision tree**.
- ▶ Internal nodes are associated with an *attribute/column* and arcs with *values* for that attribute.
- ▶ A leaf node tells the predicted class label.

- Each person has attributes/columns:
  - (age, student [yes/no], credit rating)
  - p1(18, yes, fair)
  - p2(55, no, excellent)
- Two classe labels
  - Buy computer
  - Not buy computer



# Another Decision Tree Example

- Each client has attributes/columns:
  - (degree, income, ...)
  - E.g., **c1(bachelor, 55K, ...)**



- Four class labels (credit levels)



# Construction/Optimization of Decision Trees

- Input:

- Training data: a set of data in which the classes are already known.

- Output:

- A decision tree that can be used for classification.

- Basic idea:

- Generating a decision tree *top-down* using the training data.
  - Each internal node of the tree partitions the training data into groups based on a **splitting attribute** and a **splitting condition** for the node

- › Measure of the quality of a split: **gini** index and **entropy**.

- › **Best** split vs. **Random** split (in sklearn)

- In a leaf node, all the items at the node belong to the same class, or all attributes have been considered and no further partitioning is possible.

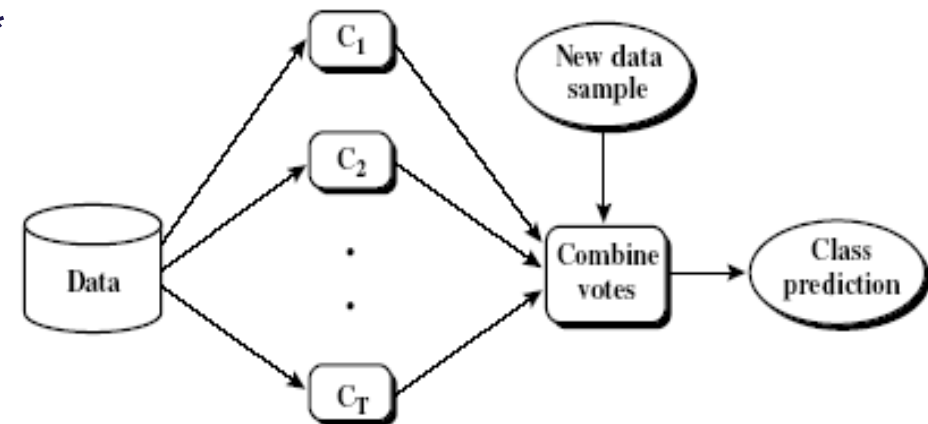
**NB:** You can specify how a DT is constructed.

# Random Forest (of Decision Trees)

- ▶ Decision trees' main drawback: tendency to **overfit** the training data.
  - ▶ Overfitting: A model focuses so much on training data that it does not generalize well to unseen data in predication.
- ▶ A random forest
  - ▶ A collection of decision trees (DT)
  - ▶ Each DT is slightly different from the others
  - ▶ With many DTs, we can maintain good prediction and reduce overfitting by using all trees' **majority vote** as the classification result.

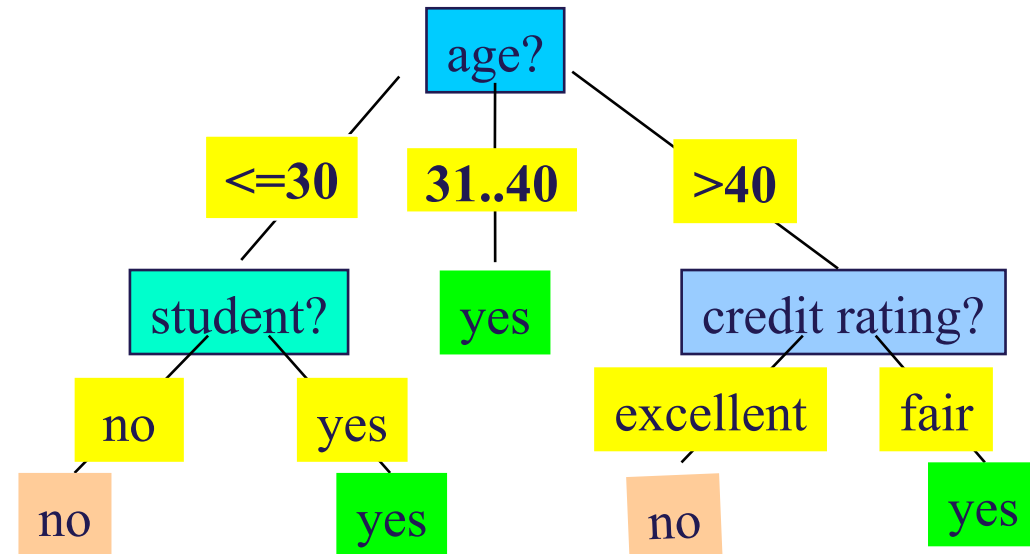
# Random Forest, *cont.*

- ▶ Two ways of introducing randomness to the tree building
  - ▶ Randomly selecting the training data points
  - ▶ Randomly selecting the features in each tree node split
- ▶ Random forest is an example of **ensemble learning**
  - ▶ Use a combination of models to increase prediction accuracy
  - ▶ Combine a series of  $T$  learned models/classifiers,  $C_1, C_2, \dots, C_T$ , with the aim of creating an improved model  $C^*$ 
    - › A simple combination: **majority vote**



# Comments on Decision Trees

- ▶ Given the same training data set, different decision trees may be constructed by different methods (with different parameters)
- ▶ A decision tree may be unbalanced
- ▶ At the same level, different nodes in a decision tree may split on different attributes.
- ▶ Decision trees belong to *eager learners*



# Eager vs Lazy Learning

- ▶ **Eager learning** (model-based methods): Given a set of training samples, constructs a classification model before receiving new (e.g., test) data to classify.
  - ▶ More time in training but less time in predicting/classification
  - ▶ E.g., we need to construct a decision tree before using it.
- ▶ **Lazy learning** (e.g., instance-based learning): Simply stores training data as instances (or only minor processing) and waits until a new instance must be classified
  - ▶ Less time in training but more time in predicting/classification
  - ▶ E.g., k nearest neighbors: Instances represented as points for which *distances* can be measured.

# K Nearest Neighbors (KNN)

## 8 Instance data set (training data)

- ▶ A set  $D$  contains  $|D|$  ( $\geq K$ ) items, each is labeled with a class.
- ▶  $D = \{(item, class)\}$
- ▶  $D$  should cover *all* pre-defined class:  $|D| \geq C$  (totally  $C$  classes)

## ▶ Classification

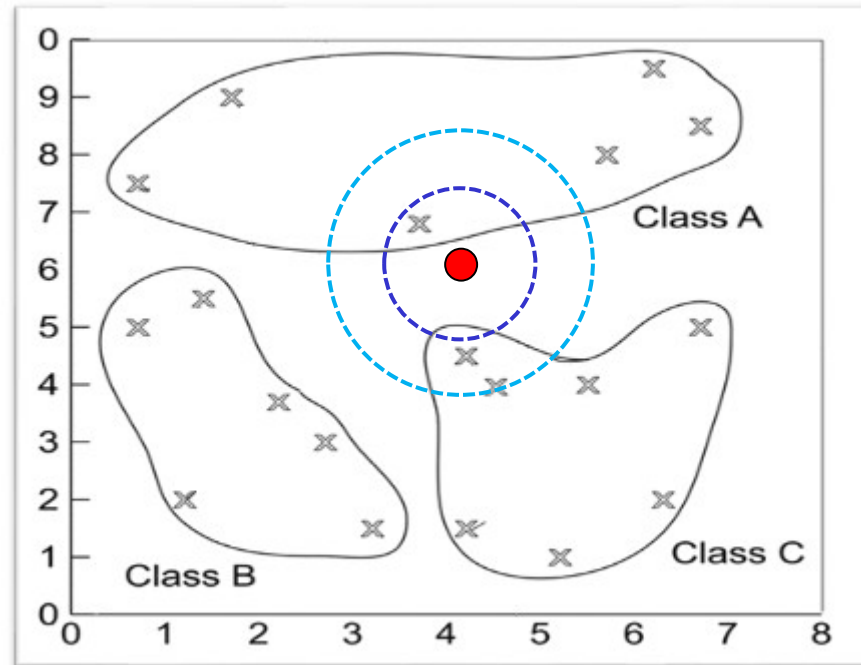
- ▶ For a given item  $t$  to be classified, we find its  **$K$**  nearest neighbor items (**decision set**) from  $D$ .
  - › Distance measurement: Usually Euclidean distance
- ▶ Count the class labels in the KNNs and give  $t$  the most frequent class label.
  - › In other words, item  $t$  is placed in the class with the highest number of NNs.
  - › NB: the **decision rule** can be different.

# KNN Example

• Different  $K$ s may lead to different classification results.

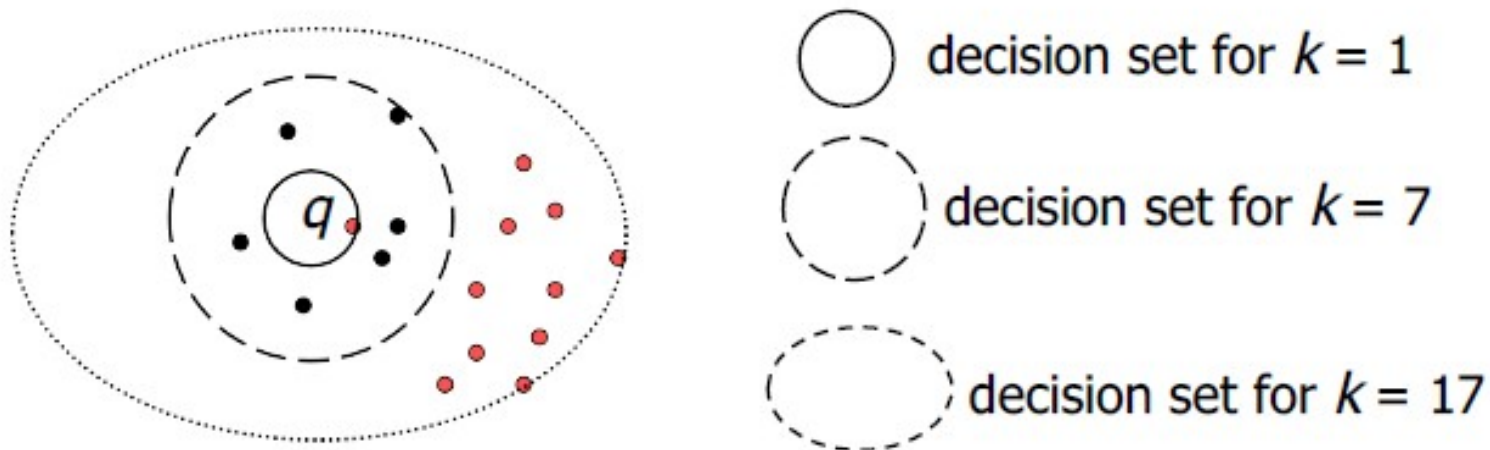
•  $K = 1$ : Class A

•  $K = 3$ : Class C



# Appropriate Value for K

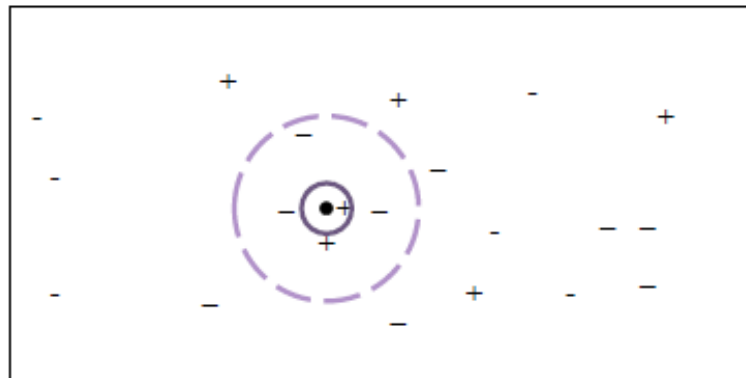
- Different  $K$ s may lead to different classification results.
- Too small  $K$ : High sensitivity to outliers
- Too large  $K$ : Decision set contains many items from other classes.
- Empirically,  $1 \ll K < 10$  yields a high classification accuracy in many cases.





# Decision Rules of KNN

- ▶ Using unit weights (i.e., no weights) for the decision set
  - ▶ Simply “majority vote” or **standard rule**
  - ▶ For  $k=5$  in the example, the rule yields class “-”
- ▶ Using the *reciprocal square of the distances* as weights
  - ▶ For  $k=5$  in the example, the rule yields class “+”
- ▶ Using *a-priori probability (frequency)* of classes as weight
  - ▶ For  $k=5$  in the example, the rule yields class “+”

- ▶ “-”:  $3/15 = 1/5$
- ▶ “+”:  $2/6 = 1/3$



Classes + and -

-  decision set for  $k = 1$
-  decision set for  $k = 5$

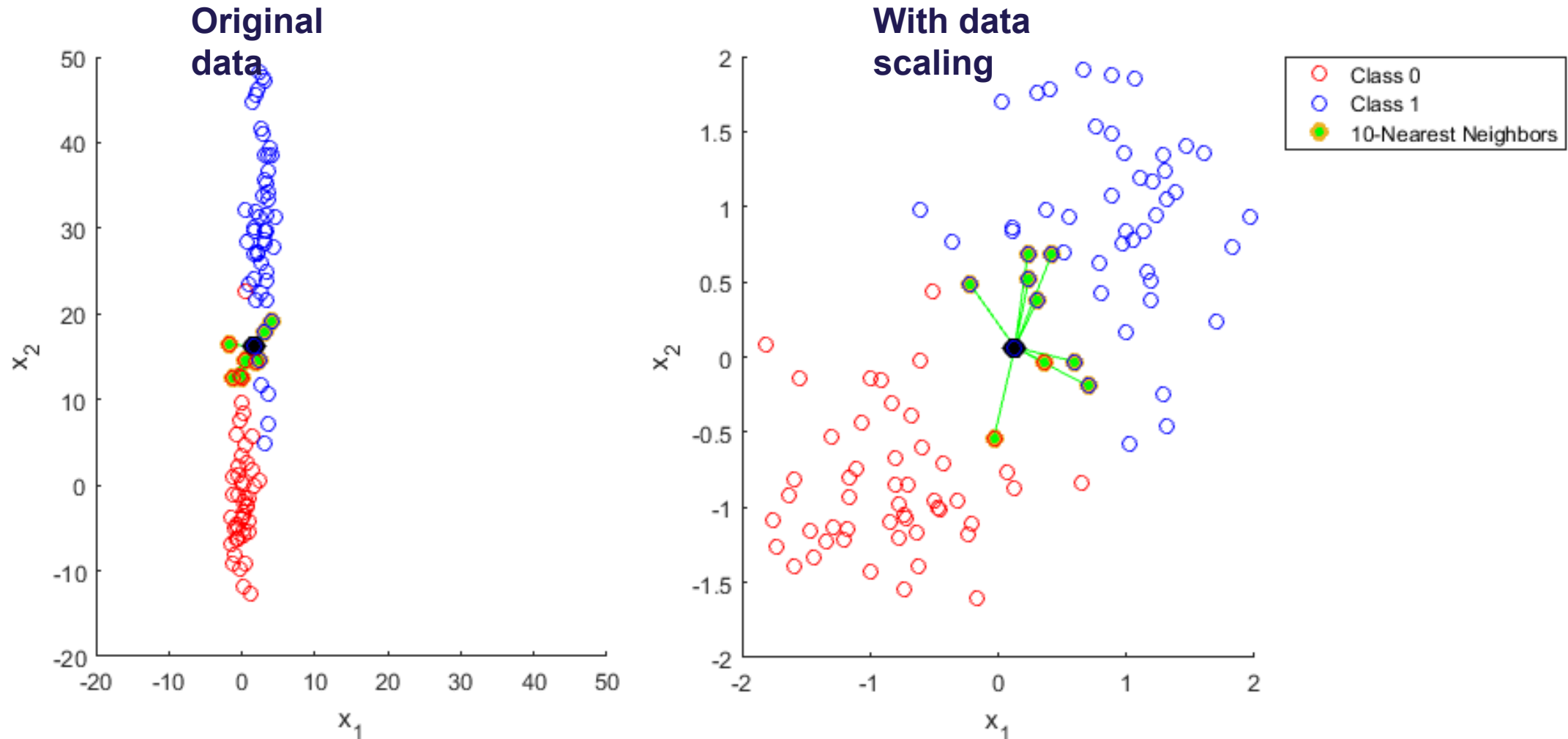
# Pros and Cons of KNN

- **Applicability**: sample (training) data required only without training
- High **classification accuracy** in many applications
- Easy **incremental adaptation** to new sample objects
- Also useful for **prediction**
- Robust to noisy data by averaging K nearest neighbors
- Naïve implementation is inefficient
  - KNN search is not straightforward. Support by database in query processing may help.
- Does not produce explicit knowledge about classes but some explanation information.
- **Curse of dimensionality**: distance could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

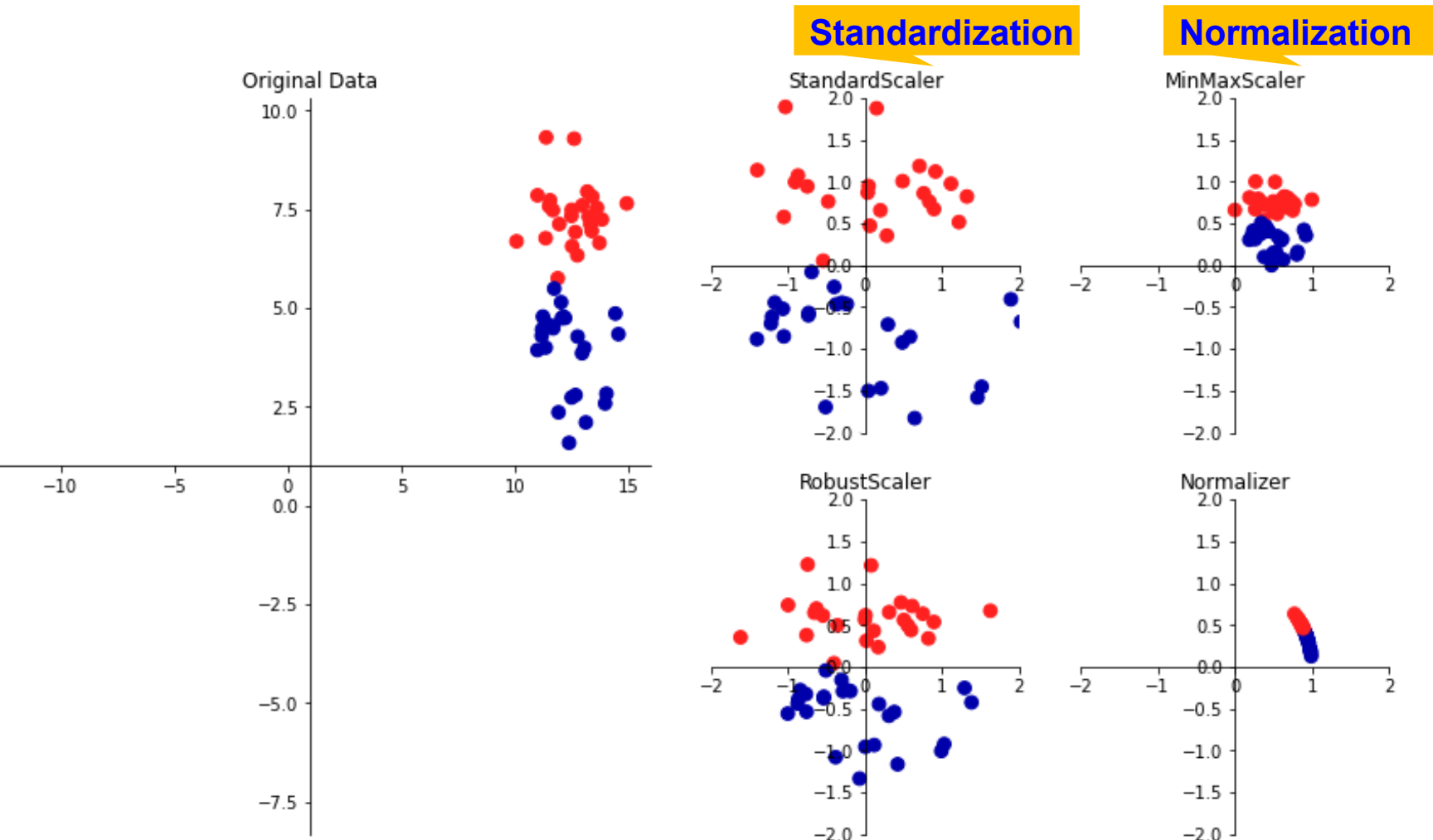
# Agenda

- Classification and model evaluation
- Typical classification models
- Data scaling
  - Why, what and how
- Model evaluation and selection

# A Motivation Example



# Data Scaling



- **StandardScaler**
  - For each feature:  
mean=0 and  
variance=1
- **MinMaxScaler**
  - Shifts the data, each feature falls in [0..1]
- **RobustScaler**
  - Similar to SS but uses  
median and quartile to avoid outliers
- **Normalizer**
  - Scales each data point s.t. its Euclidean distance to (0, 0) is 1
  - Used when *only* the direction matters

# Notes on Data Scaling

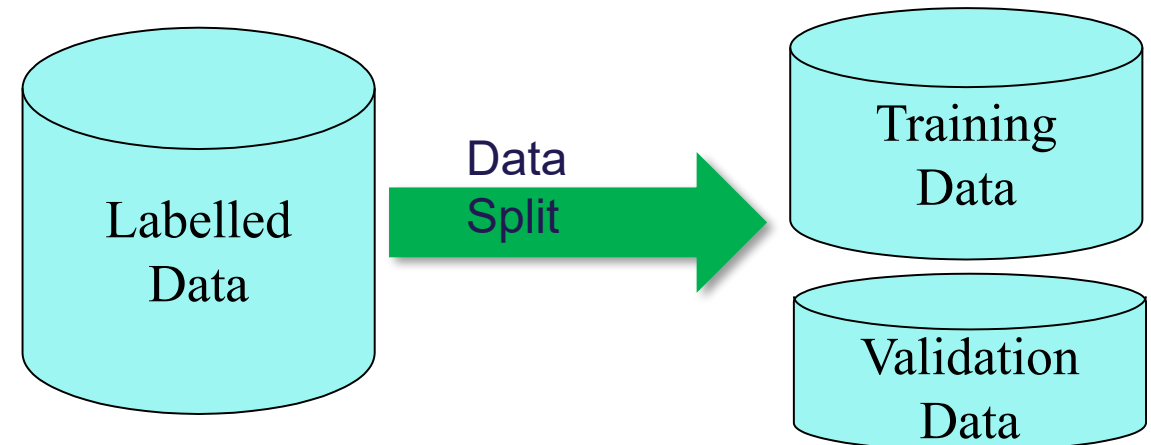
- Observe and/or plot your data to see how it skews
- Choose the right scaler you want to use
- Apply the scaler to *both* training and testing data
  - Apply the scaling on the whole original dataset
  - Then split the scaled dataset
- Standardization or Normalization? (Rule of thumb)
  - Normal data distribution: standardization; otherwise normalization
  - If uncertain: normalization; or standardization followed by normalization
  - Try different ways and decide the option with the best model performance

# Agenda

- Classification and model evaluation
- Typical classification models
- Data scaling
- **Model evaluation and selection**
  - Data split and model evaluation
  - ROC and AUC for binary classification

# Split Labelled Data

- `sklearn.model_selection.train_test_split`
- `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)`
  - `test_size`: validation/test data percentage
  - `random_state`: randomization of the split
- Different ways of split can result in different models and performance



# Model Evaluation and Selection

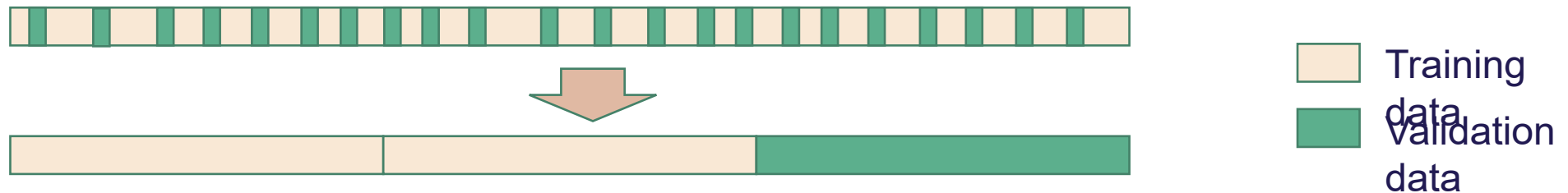
- Use **validation set** of labeled data samples instead of training set when assessing model accuracy
  - Otherwise, **overfitting!**
    - A model focuses so much on the training data that it does not generalize well to unseen data in predication.
- All labeled samples form  $D$ . How to split  $D$  into training and validation sets?
  - Holdout method, random subsampling
  - Cross-validation (k-fold)
  - Bootstrap (use it only when your data is not sufficient)
- These methods differ in how you partition/split all your labelled sample data into training set and validation set

# Holdout

`sklearn.model_selection.train_test_split()`

► Split the given labelled data *randomly* into two *independent* sets

- Training set (e.g., 2/3) for model construction
- Validation set (e.g., 1/3) for accuracy assessment

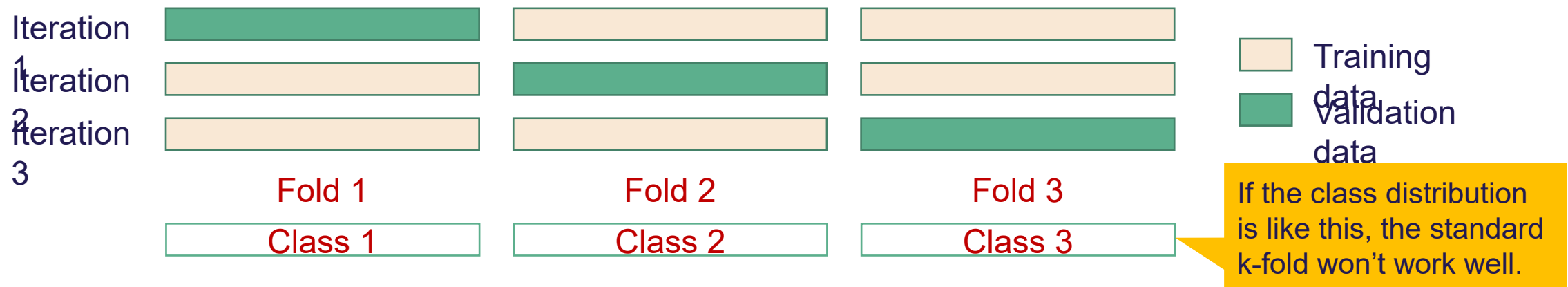


► **Random sampling**: a variant of holdout

- Repeat holdout  $k$  times, accuracy = *average* of the accuracies obtained

# Standard Cross-Validation (CV)

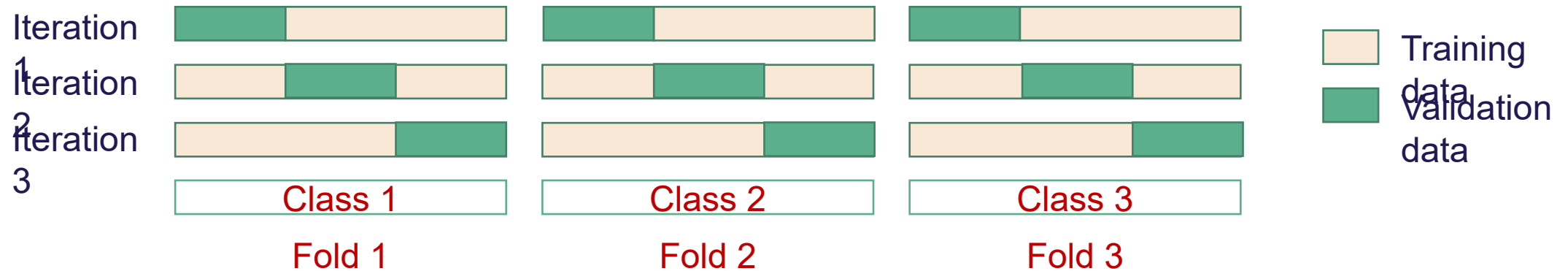
- Aka **k-fold** ( $k = 10$  is most popular)
  - Split the sample data  $D$  into  $k$  mutually exclusive subsets, each of approximately equal size:  $D_1 \dots D_k$ . Each  $D_i$  is called a *fold*.
  - Do model construction and evaluation for  $k$  time. Use the *average* accuracy.
    - › At the  $i$ -th iteration, use fold  $D_i$  as the validation set and the others as the training set.
- Example of standard 3-fold cross validation



# Variants of k-fold

## ● Stratified cross- validation

- folds are stratified so that *class distribution* in each fold is approximately the same as that in the initial given data.



## ● **Leave-one-out:** $k$ -fold where $k = \#$ of sample points

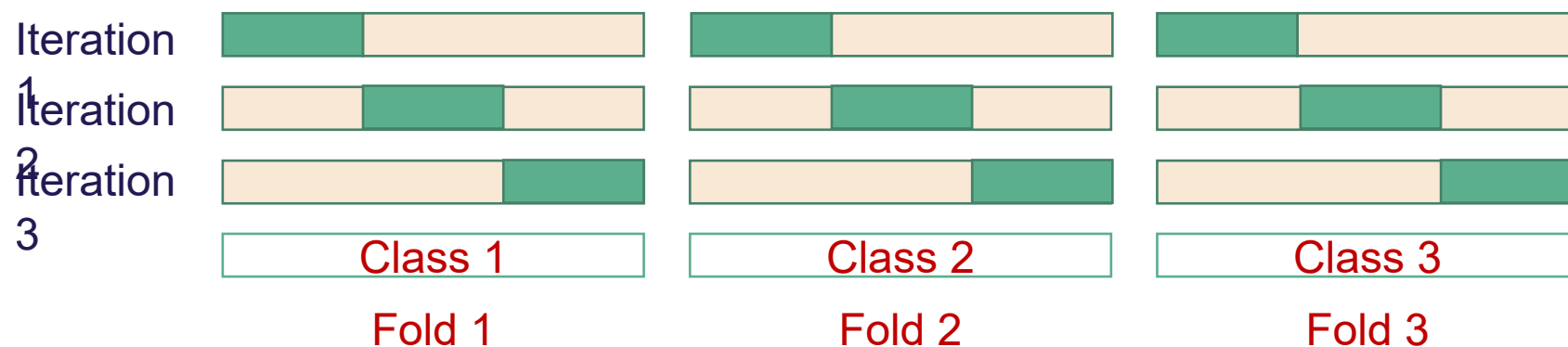
- Use it only for small sized data; otherwise too many models to construct.

# k-fold Cross Validation

## ► Standard 3-fold cross validation



## ► Stratified 3-fold cross validation



Default in scikit-learn:  
Stratified 5-fold

# Notes on Cross-Validation

- CV is not a way to construct an applicable model.
- The function `cross_val_score(.)` builds multiple models *internally*, but these models are not returned.
- The purpose of CV is to evaluate how well a *type* of model will generalize when it is trained on a specific dataset.
  - Model type: decision tree, random forest, KNN, SVM, ...
- By using CV, we can decide what type of model to use, and tune hyperparameters for constructing a model
  - **Hyperparameters**: algorithm parameters that can be set by the user before training a model. E.g., `gini` or `entropy` for a DT, `K` for KNN, `test_size` and `random_state` for `train_test_split(.)` ...
  - In contrast, **model parameters** are learned internally from training data
    - › E.g., how many levels actually in a DT?

# Bootstrap

## ► Bootstrap

Advanc  
ed

- Given a data set  $D$  with  $m$  tuples, sample uniformly *with replacement*
  - › Select one tuple randomly, put it in a set  $D'$ , and put it back to  $D$ .
  - › Repeat  $m$  times
- Training set:  $D'$  (with  $m$  tuples that may repeat)
- Validation set:  $D \setminus D'$  ( $D$  is not changed)

## ► Remarks

- No overfitting
  - › It can be proved that about 36.8% tuples in  $D$  do not enter  $D'$
  - › When  $m$  is infinite,  $(1 - 1/m)^m \approx e^{-1} = 0.368$
  - › This is a.k.a. **.632 bootstrap**
- Works well with a small data set  $D$
- But the original data distribution is distorted. So don't use bootstrap when your training data is sufficient.

# Issues Affecting Model Selection

## ➤ Accuracy

- Classifier accuracy: predicting class label

## ➤ Speed

- Time to construct the model (training time)
- Time to use the model (classification/prediction time)

## ➤ Robustness

- How well to handle noise and missing values

## ➤ Scalability

- Efficiency in disk-resident databases

## ➤ Interpretability

- Understanding and insight provided by the model

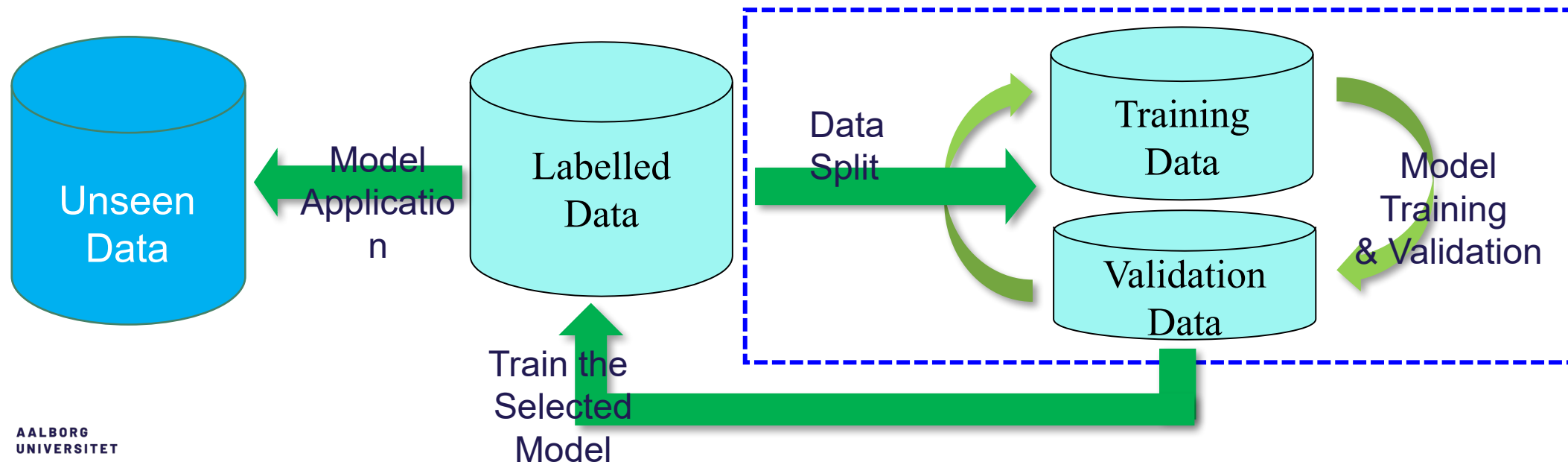
## ➤ Other measures, e.g., decision tree size

# Classification of Class-Imbalanced Data Sets

- ▶ **Class-imbalance problem**: Rare positive example but numerous negative ones, e.g., COVID-19 tests, fraud, oil-spill, fault, etc.
- ▶ Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
- ▶ Typical methods for imbalance data in binary class classification:
  - ▶ **Oversampling**: re-sampling of data from positive class
  - ▶ **Under-sampling**: randomly eliminate tuples from negative class
  - ▶ **Threshold-moving**: moves the decision threshold,  $t$ , so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
  - ▶ **Ensemble techniques**: Ensemble multiple classifiers to be introduced next
- ▶ Still difficult for class imbalance problem on multiclass tasks

# After Validation: Making Use of All Labelled Data

- CV enables us to select the type of model (including hyperparameters) with the *best expected generalization ability* (to unseen data)
- We train the selected model using all labelled data
- We apply the final model to unseen data (test in applications)



# Binary Classification Performance Metrics

- ▶ **Sensitivity** / True Positive Rate / Recall

- ▶  $TPR = TP / (TP + FN)$

- ▶ False Negative Rate

- ▶  $FNR = FN / (TP + FN) = 1 - TPR$

- ▶ **Specificity** / True Negative Rate

- ▶  $TNR = TN / (TN + FP)$

- ▶ False Positive Rate

- ▶  $FPR = FP / (TN + FP) = 1 - TNR$

		Prediction	
		POSITIVE	NEGATIVE
Groundtruth	POSITIVE	TP	FN
	NEGATIVE	FP	TN

# Prediction Probability

- To predict a data object's *probability* of belonging to different classes
  - A **threshold** can be used to control how to decide the predicted class label
  - E.g., KNN's decision rule can be changed to do so

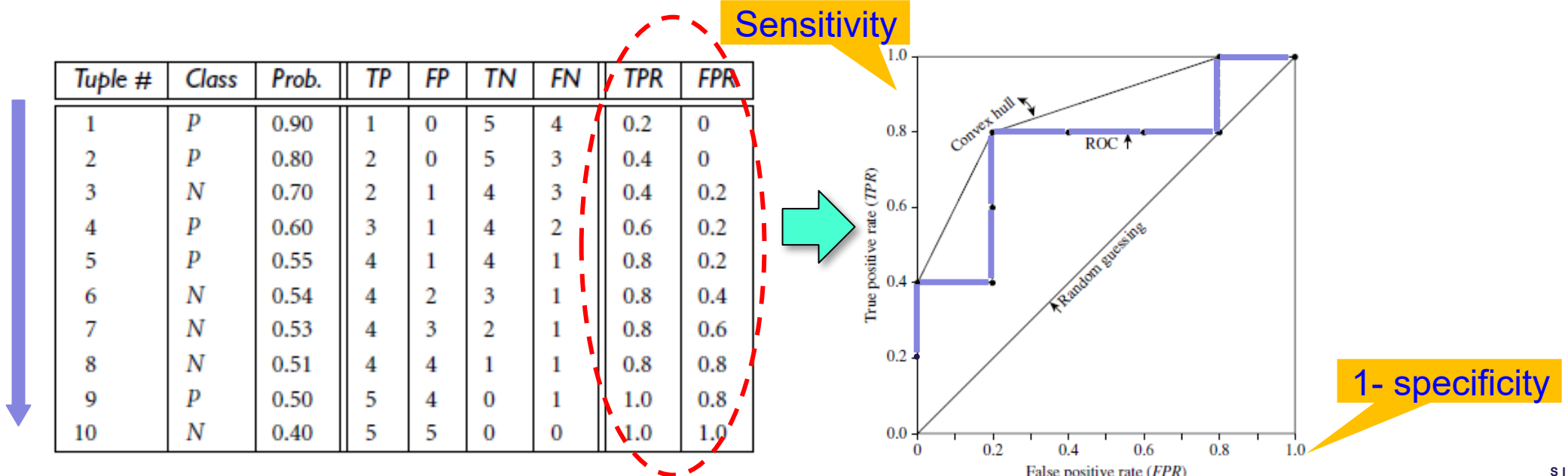
- Different thresholds lead to different metric values.
- This requires us to generate different confusion matrixes 😞

ID	Actual	Prediction Probability	>0.6	>0.7	>0.8	Metric
1	0	0.98	1	1	1	
2	1	0.67	1	0	0	
3	1	0.58	0	0	0	
4	0	0.78	1	1	0	
5	1	0.85	1	1	1	
6	0	0.86	1	1	1	
7	0	0.79	1	1	0	
8	0	0.89	1	1	1	
9	1	0.82	1	1	1	
10	0	0.86	1	1	1	
			0.75	0.5	0.5	TPR
			1	1	0.66	FPR
			0	0	0.33	TNR
			0.25	0.5	0.5	FNR

For positive label

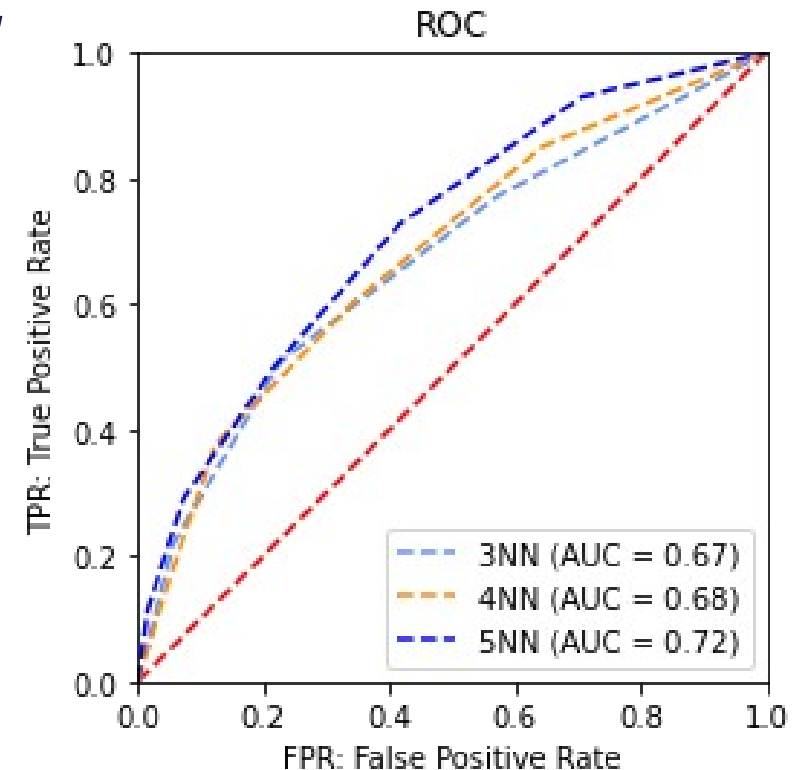
# ROC Curves

- **Receiver Operating Characteristics** curves: for visual comparison of binary classifiers
  - Rank your classification results in *descending* order of prediction probabilities
  - Calculate TPR and FPR for each current tuple in the ranked order
  - Mark each (FPR, TPR) point on the graph.
  - Connect all such points using a *convex hull*
- **NB:** TP, FP, TN, FN (and **TPR** and **FPR**) change as you seen more tuples in classification result



# ROC Curves and AUC

- ▶ A ROC curve shows the trade-off between the **True Positive Rate** and the **False Positive Rate**
- ▶ The diagonal represents *random guessing*
- ▶ The *area under the ROC curve* (**AUC**) is a measure of the accuracy of the model
- ▶ The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model
- ▶ A model with perfect accuracy will have an area of 1.0



# Summary

- Classification process
  - Model construction/training, validation, test
  - Labelled data splitting
- Classification models
  - Decision tree
  - Random forest
  - KNN
- Classification performance evaluation and model selection
  - Precision, recall, accuracy
  - Confusion matrix
  - Holdout, Cross-validation (k-fold)
  - ROC and AUC
- Data scaling
  - Necessary when distances are involved in modelling and columns are of different scales

# Readings and References

## ➤ Mandatory reading

- Jiawei Han, Micheline Kamber, and Jian Pei (Data Mining: Concepts and Techniques, 3rd Edition 2011): Chapter 1, 8.1, 8.2, and 8.5

## ➤ Further reading

### ➤ Decision tree

- › **Tutorial:** <https://www.datacamp.com/community/tutorials/decision-tree-classification-python>
- › **Doc:** <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

### ➤ Random forest

- › **Tutorial:** <https://www.datacamp.com/community/tutorials/random-forests-classifier-python>
- › **Doc:** <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

### ➤ ROC and AUC

- › <https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/>
- › [https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_roc.html](https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html)

# Exercises (1)

1. Given two classes:  $A = \{(1, 3), (2, 2), (3.5, 1), (5, 4), (1.5, 4), (4, 2)\}$  and  $B = \{(2, 3), (3, 0.5), (4, 3), (3.5, 2), (1, 2.5), (2, 1)\}$  and three unclassified points  $(4, 1)$ ,  $(1.5, 2.5)$  and  $(3, 4)$ .
  1. Use the kNN classification approach and Euclidean distance to decide the classes for the three points, for  $k = 1, 2, 3$ .
  2. Repeat 1.1 but use Manhattan distance instead. The Manhattan distance between  $(x_1, y_1)$  and  $(x_2, y_2)$  is defined as  $|x_1 - x_2| + |y_1 - y_2|$ .
  3. Compare the results of 1.1 and 1.2.
2. Suppose that labelled data is distributed in  $M$  sites and we need to run kNN to decide the class label for a new point at site  $S_0$ .
  1. Describe a parallel algorithm for it. How much data do you need to transfer?
  2. Can it make a difference whether you run local kNN at  $S_0$  or not?

# Exercises (2, hands-on, optional)

Using the Diabetes dataset (available in Moodle), do the following

1. Split the dataset  $D$  into two parts: 80% for training ( $D_T$ ) and 20% for validation/test  $D_v$ .
  - › *NB*: You may use different ratios, do the subsequent steps, and see the effect
  - 1. Build a decision tree for predicting if a person has diabetes or not. Use  $D_T$  to train the model, apply the model to  $D_v$ , and evaluate the classification accuracy.
  - › Try to build a number of different trees using different parameters, see their accuracy
  - 2. Build a random forest on the same training/test datasets, obtain its accuracy, and plot the important features for it.
2. Apply KNN and cross validation
  1. Use a default KNN ( $K=5$ ) to see the effect of data scaling (with vs. without).
  2. Try different  $K$ 's (2 to 8) for KNN, validate each classifier using stratified 3-fold, and plot the ROC with AUC for each model.