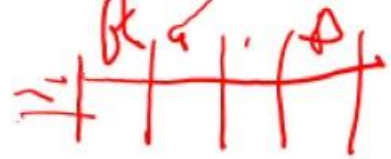
Supervised Learning

Prof. Douglas Kunda, PhD
Vice Chancellor – ZCAS University
vc@zcasu.edu.zm

Contents

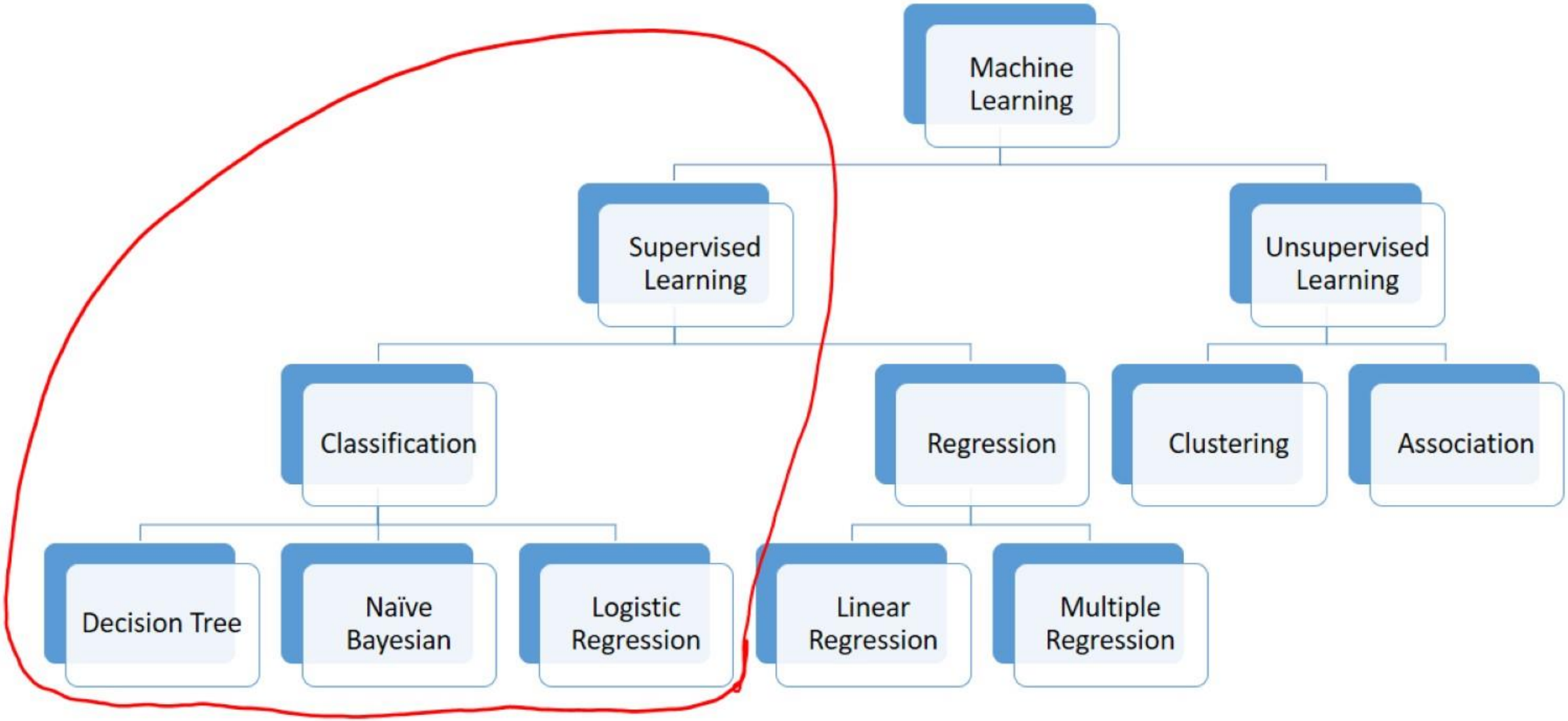
- Machine learning
- Supervised learning
- Classification
- Decision Tree Classifier
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- k-Nearest Neighbor Method
- Evaluating classification methods
- Exercise

Machine learning



- Machine Learning (ML) is a subset of artificial intelligence (AI) that involves the development of algorithms and models that enable computers to learn from and make predictions or decisions based on data.
- ML algorithms learn patterns from historical data and use these patterns to make predictions or decisions about new, unseen data.
- Supervised Learning: Learns from labeled data, predicting outcomes based on existing examples.
- Unsupervised Learning: Extracts patterns and relationships from unlabeled data, such as clustering similar data points.
- Reinforcement Learning: Involves training models to make sequential decisions through interaction with an environment.

Machine Learning Categories



Machine learning and our focus

- Like human learning from past experiences.
- A computer does not have "experiences".
- A computer system learns from data, which represent some "past experiences" of an application domain.
- Our focus: learn a target function that can be used to predict the values of a discrete class attribute, e.g., approve or not-approved, and high-risk or low risk.
- The task is commonly called: Supervised learning, classification, or inductive learning.

What do we mean by learning?

- Given
  - a data set *D*,
  - a task *T*, and
  - a performance measure *M*,

  a computer system is said to **learn** from *D* to perform the task *T* if after learning the system's performance on *T* improves as measured by *M*.
- In other words, the learned model helps the system to perform *T* better as compared to no learning.

The loan data example

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

An example

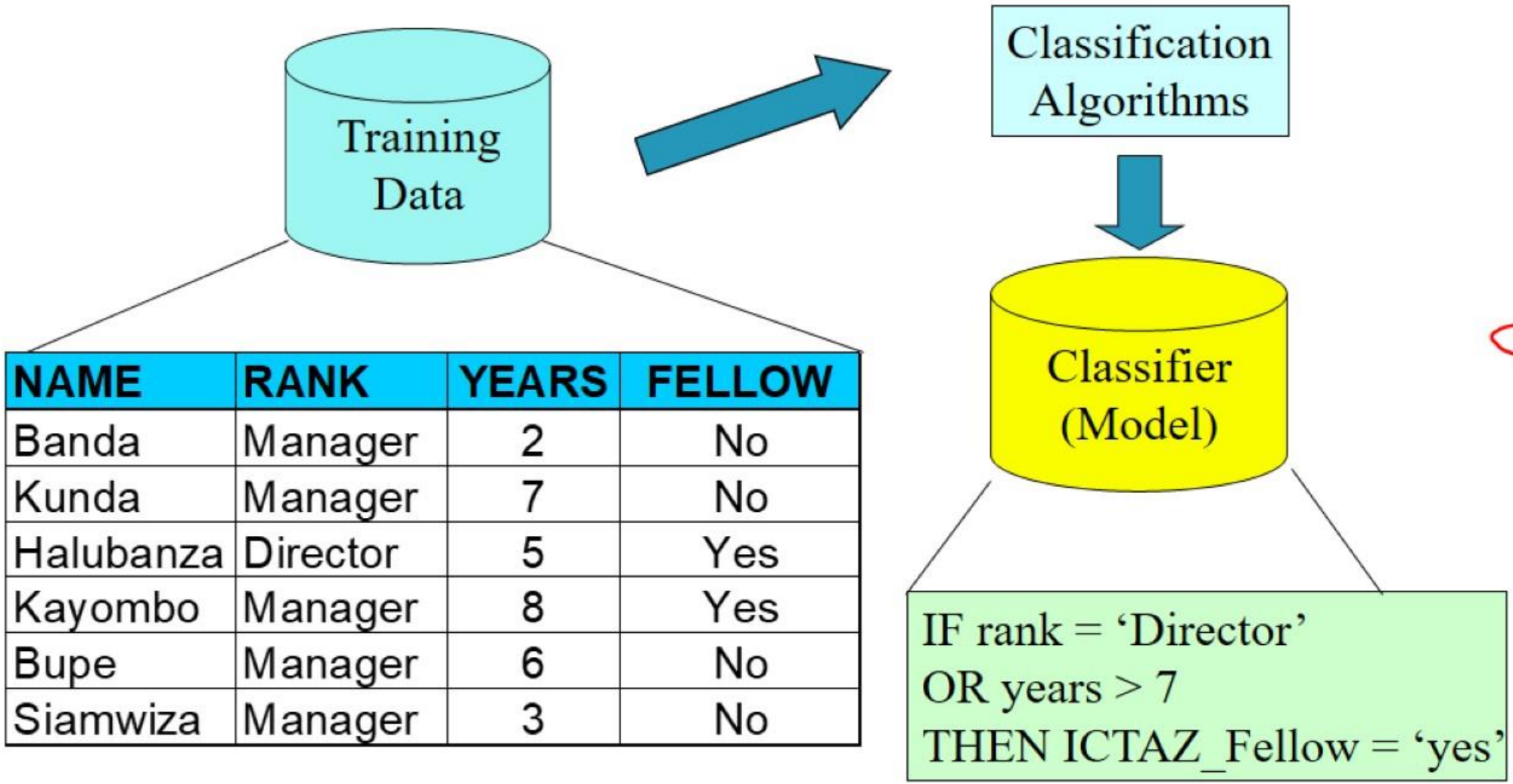
- Data: Loan application data
- Task: Predict whether a loan should be approved or not.
- Performance measure: accuracy.

No learning: classify all future applications (test data) to the majority class (i.e., Yes):

Accuracy = 9/15 = 60%.

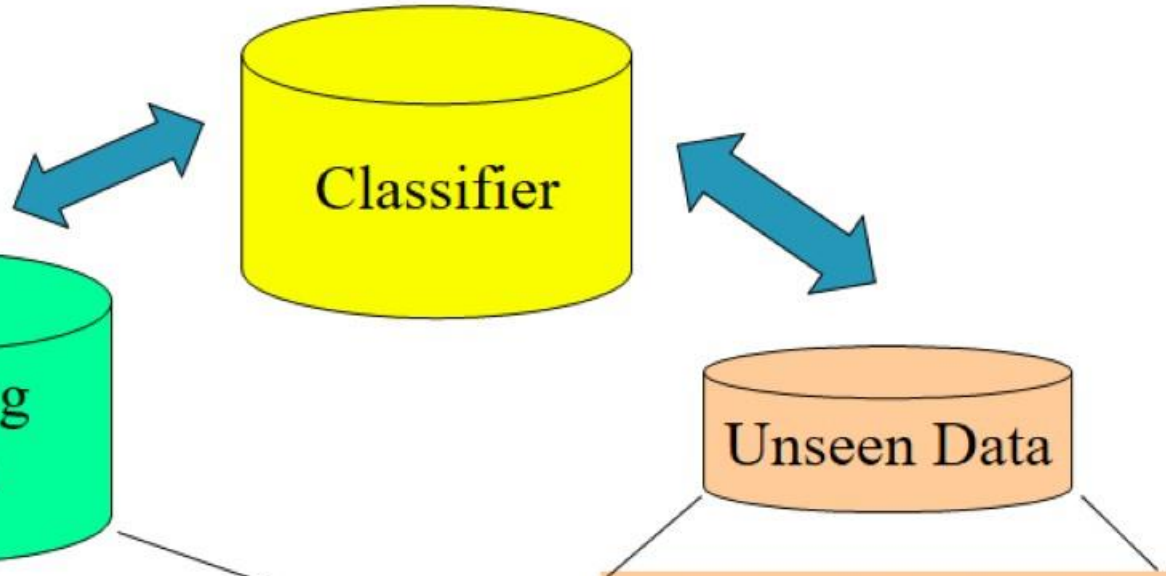
- We can do better than 60% with learning.

Learning - A Two-Step Process



NAME	RANK	YEARS	FELLOW
Banda	Manager	2	No
Kunda	Manager	7	No
Halubanza	Director	5	Yes
Kayombo	Manager	8	Yes
Bupe	Manager	6	No
Siamwiza	Manager	3	No

Learning - A Two-Step Process

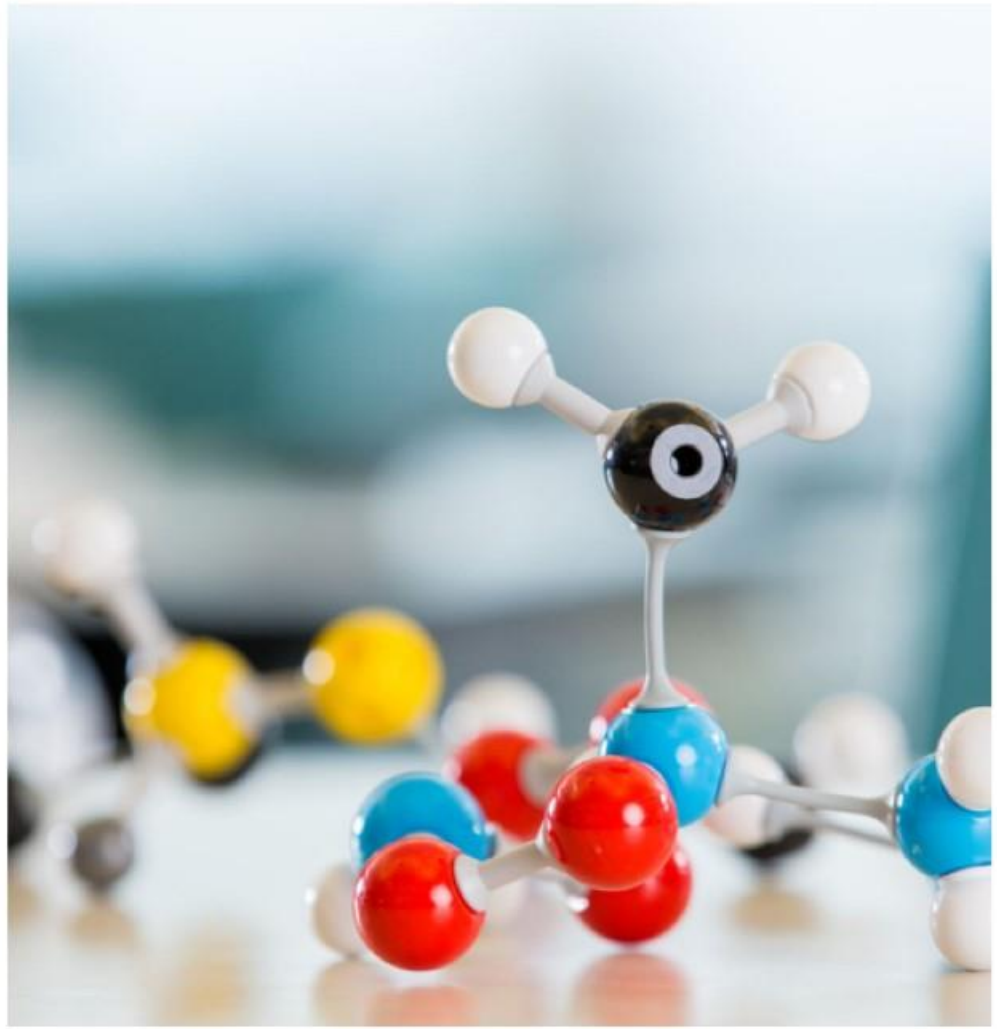


NAME	ICT RANK	YEARS	FELLOW
Tom	Manager	2	no
Merlisa	Manager	7	no
George	Director	5	yes
Joseph	Manager	7	yes

Fundamental assumption of learning

Assumption: The distribution of training examples is identical to the distribution of test examples (including future unseen examples).

- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.



Supervised Machine Learning

- Supervised machine learning falls into two categories—classification and regression.
- You train machine-learning models on datasets that consist of rows and columns.
- Each row represents a data sample.
- Each column represents a feature of that sample.
- In supervised machine learning, each sample has an associated label called a target.
- This is the value you're trying to predict for new data that you present to your models.

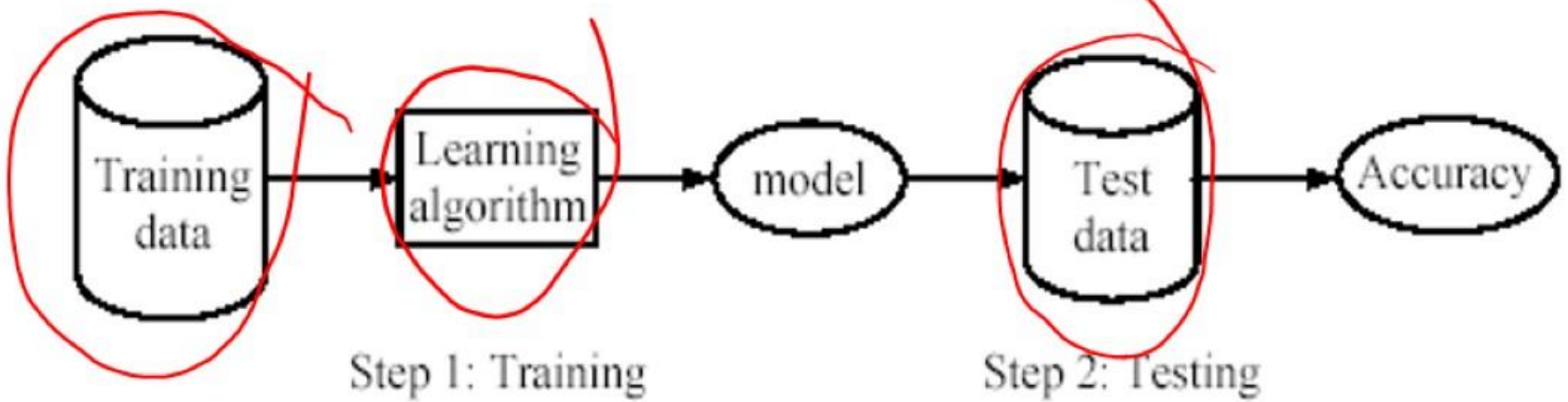
Supervised Learning

- Supervised learning: classification is seen as supervised learning from examples.
  - Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a "teacher" gives the classes (supervision).
  - Test data are classified into these classes too.
- Unsupervised learning (clustering)
  - Class labels of the data are unknown
  - Given a set of data, the task is to establish the existence of classes or clusters in the data

Supervised learning process: two steps

- Learning (training): Learn a model using the training data
- Testing: Test the model using unseen test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



100 record

80 data

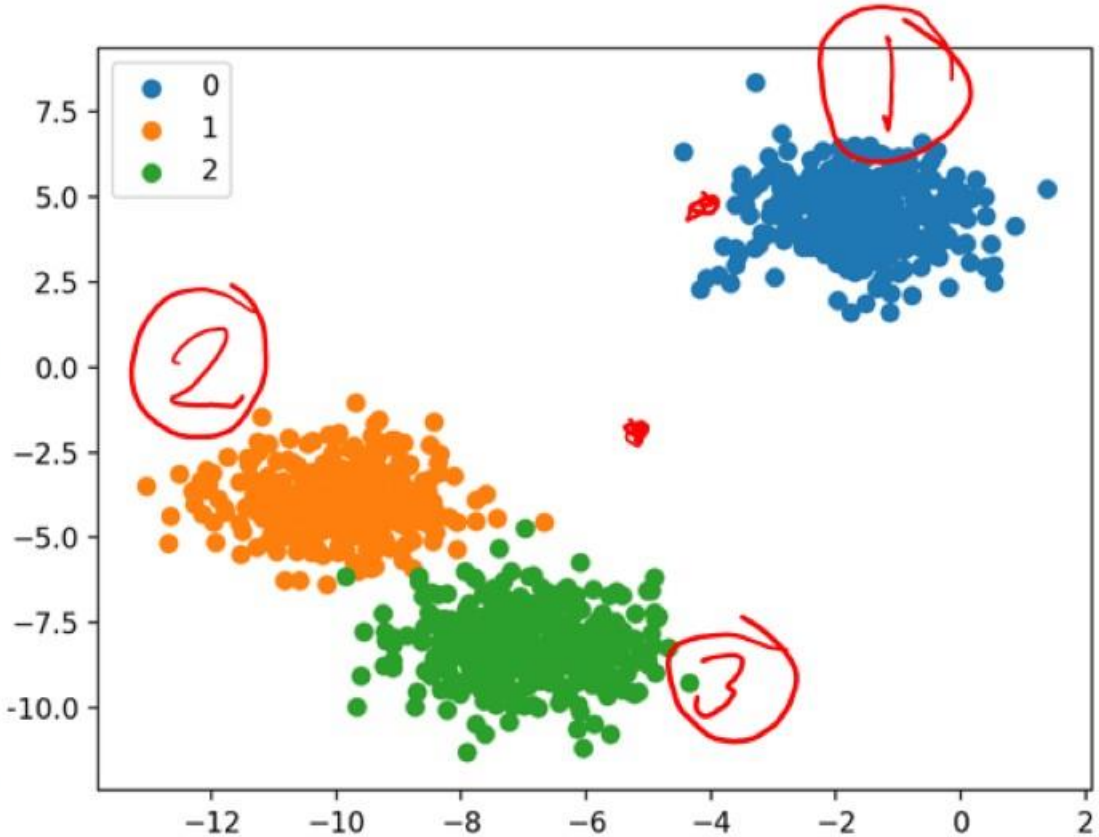
20 test

Contents

- Supervised learning
- Classification
- Decision Tree Classifier
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- k-Nearest Neighbor Method
- Evaluating classification methods
- Exercise

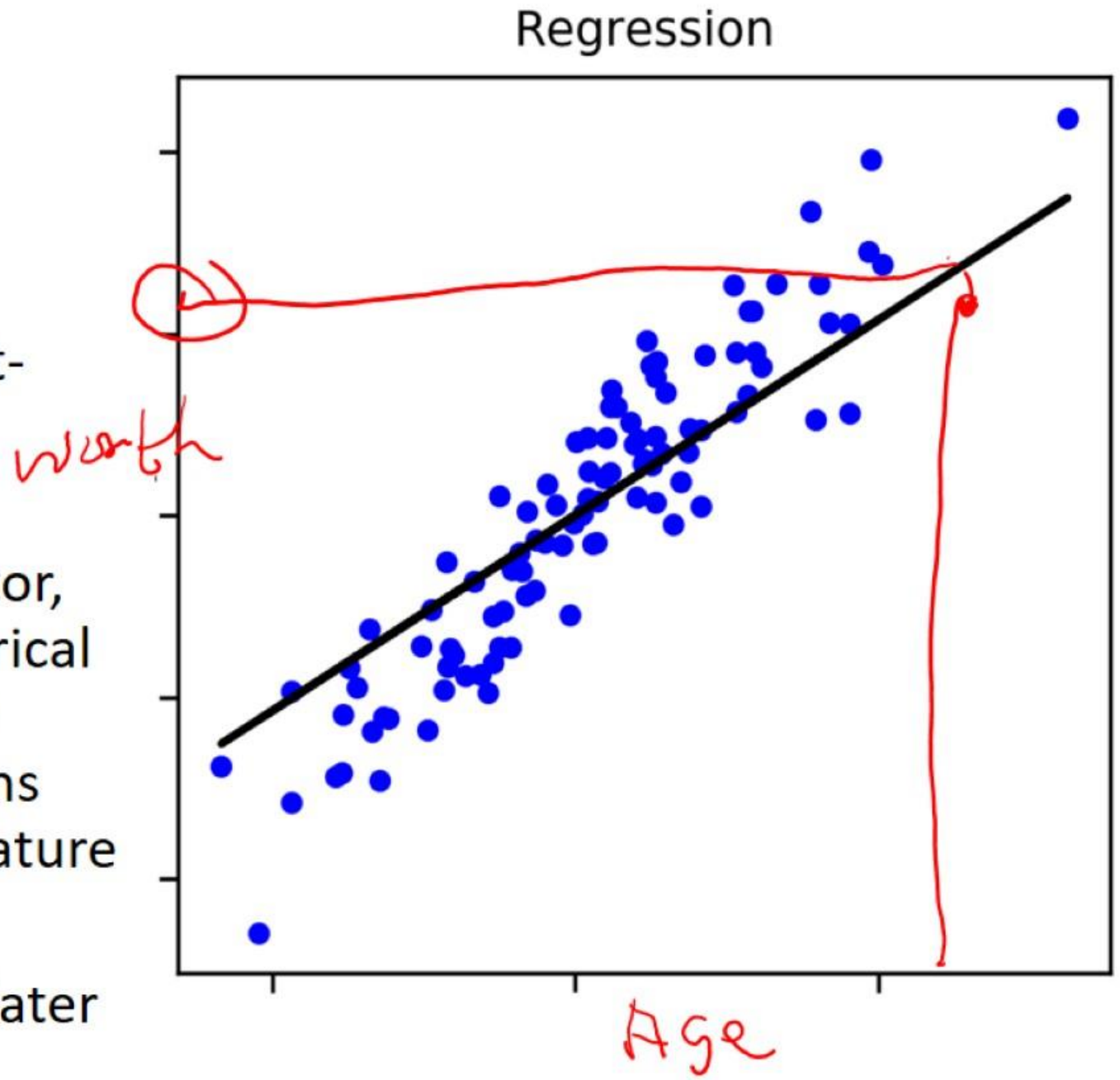
Classification

- Classification algorithms predict the discrete classes (categories) to which samples belong.
- Classification is a supervised machine learning method where the model tries to predict the correct label of a given input data.
- In classification, the model is fully trained using the training data, and then it is evaluated on test data before being used to perform prediction on new unseen data.



Regression

- Regression models predict a continuous output.
- We'll perform simple linear regression using python scikit-learn's LinearRegression estimator.
- The LinearRegression estimator, by default, uses all the numerical features in a dataset to make more sophisticated predictions than you can with a single-feature simple linear regression.
- We will deal with regression later



An example application

- An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.
- A decision is needed: whether to put a new patient in an intensive-care unit.
- Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.
- Problem: to predict high-risk patients and discriminate them from low-risk patients.

Another application

- A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,
  - age
  - Marital status
  - annual salary
  - outstanding debts
  - credit rating
  - etc.
- Problem: to decide whether an application should approved, or to classify applications into two categories, approved and not approved.

Contents

- Introduction to machine learning
- Supervised learning
- Classification
- Decision Tree Classifier
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- k-Nearest Neighbor Method
- Evaluating classification methods
- Exercise

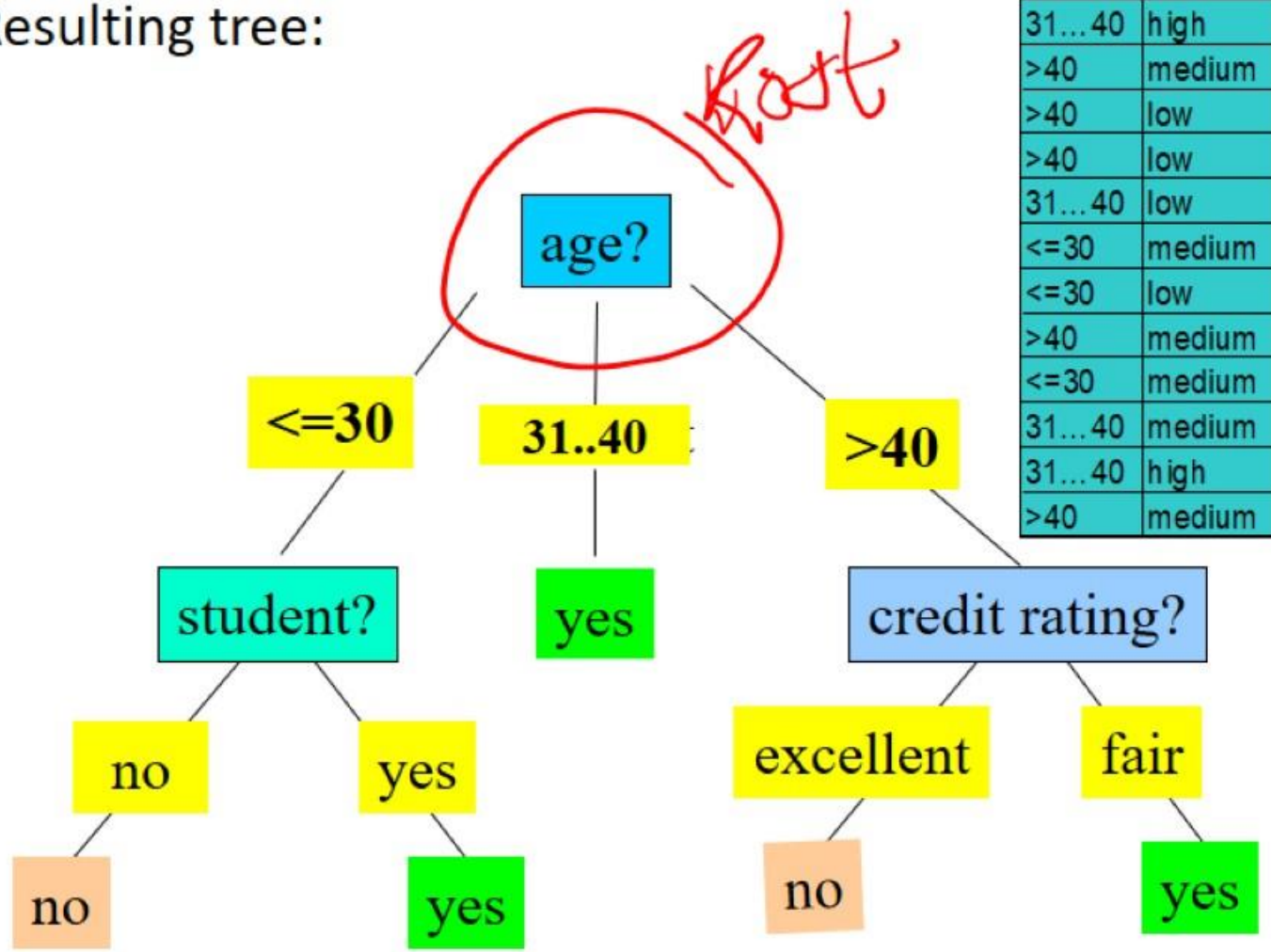
Decision tree

- Decision tree learning is one of the most widely used techniques for classification.
  - Its classification accuracy is competitive with other methods, and
  - it is very efficient.
- The classification model is a tree, called decision tree.
- C4.5 by Ross Quinlan is perhaps the best known system. It can be downloaded from the Web.

Decision Tree Induction: An Example

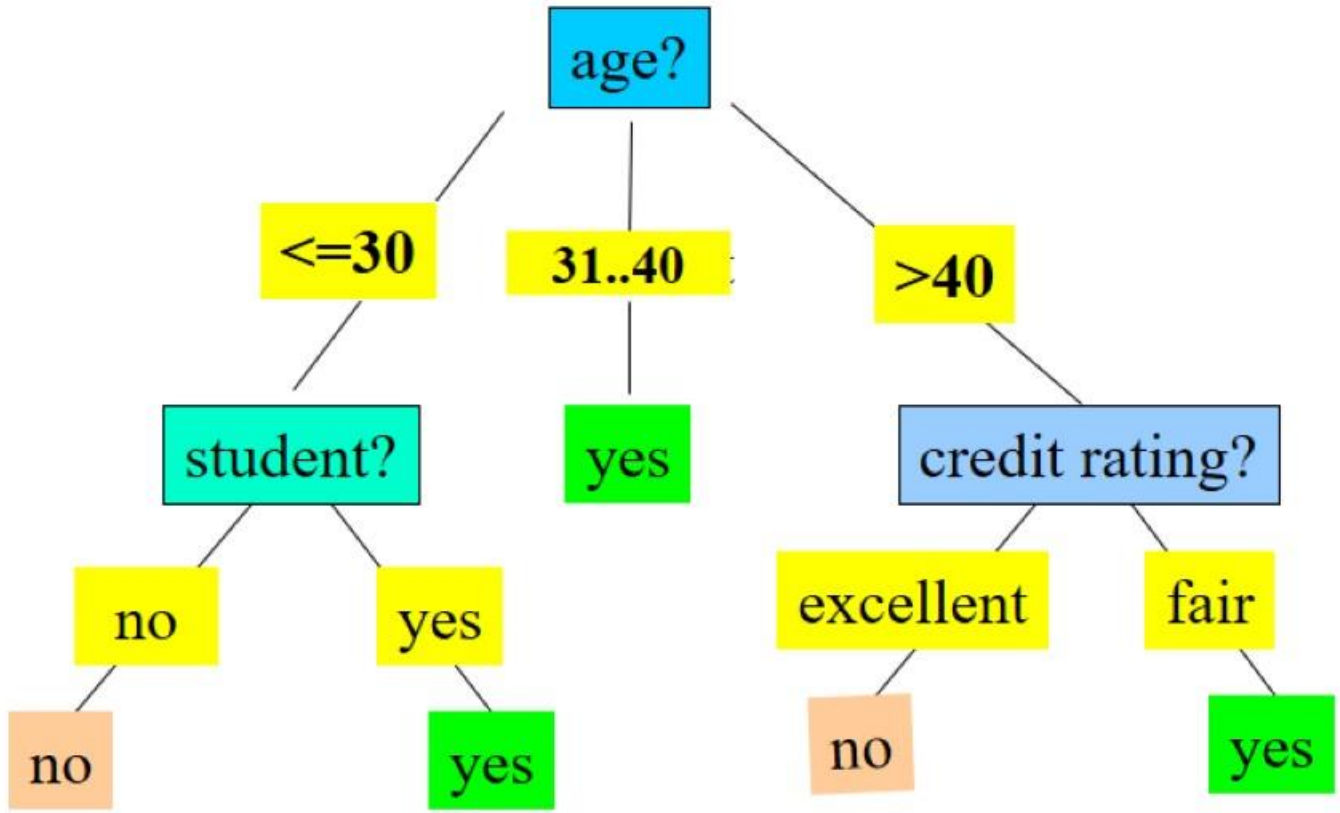
- Training data set: Buys_computer
- Resulting tree:

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



From a decision tree to a set of rules

- A decision tree can be converted to a set of rules
- Each path from the root to a leaf is a rule.
- Age = >40 -> buy_computer=yes [sup=3/5]
- Age= <30, student=yes -> buy_computer=yes [sup=2/5]
- Age= <30, student=no -> buy_computer=yes [sup=3/5]



Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Decision tree learning algorithm

. **Algorithm** decisionTree(D, A, T)

```
1   if D contains only training examples of the same class c_j ∈ C then
2       make T a leaf node labeled with class c_j;
3   elseif A = ∅ then
4       make T a leaf node labeled with c_j, which is the most frequent class in D
5   else  // D contains examples belonging to a mixture of classes. We select a single
6         // attribute to partition D into subsets so that each subset is purer
7       p_0 = impurityEval-1(D);
8       for each attribute A_i ∈ {A_1, A_2, ..., A_k} do
9           p_i = impurityEval-2(A_i, D)
10      end
11      Select A_g ∈ {A_1, A_2, ..., A_k} that gives the biggest impurity reduction,
            computed using p_0 − p_i;
12      if p_0 − p_g < threshold then    // A_g does not significantly reduce impurity p_0
13        make T a leaf node labeled with c_j, the most frequent class in D.
14      else                             // A_g is able to reduce impurity p_0
15          Make T a decision node on A_g;
16          Let the possible values of A_g be v_1, v_2, ..., v_m. Partition D into m
                disjoint subsets D_1, D_2, ..., D_m based on the m values of A_g.
17          for each D_j in {D_1, D_2, ..., D_m} do
18              if D_j ≠ ∅ then
19                  create a branch (edge) node T_j for v_j as a child node of T;
20                  decisionTree(D_j, A-{A_g}, T_j)// A_g is removed
21              end
22          end
23      end
24  end
```

Choose an attribute to partition data

- The *key* to building a decision tree - which attribute to choose in order to branch.
- The objective is to reduce impurity or uncertainty in data as much as possible.
  - A subset of data is pure if all instances belong to the same class.
- The *heuristic* in C4.5 is to choose the attribute with the maximum Information Gain or Gain Ratio based on information theory.

Comparing Attribute Selection Measures

The three measures, in general, return good results but

- **Information gain**:
  - biased towards multivalued attributes
- **Gain ratio**:
  - tends to prefer unbalanced splits in which one partition is much smaller than the others
- **Gini index**:
  - biased to multivalued attributes
  - has difficulty when # of classes is large
  - tends to favor tests that result in equal-sized partitions and purity in both partitions

Avoid overfitting in classification

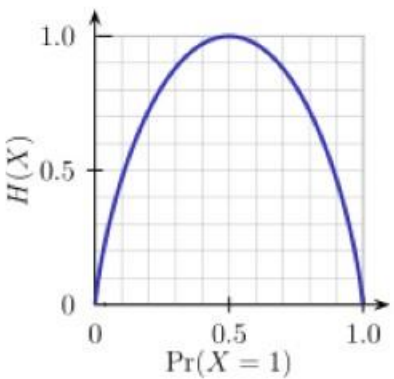
- Overfitting: A tree may overfit the training data
  - Good accuracy on training data but poor on test data
  - Symptoms: tree too deep and too many branches, some may reflect anomalies due to noise or outliers
- Two approaches to avoid overfitting
  - Pre-pruning: Halt tree construction early
    - Difficult to decide because we do not know what may happen subsequently if we keep growing the tree.
  - Post-pruning: Remove branches or sub-trees from a "fully grown" tree.
    - This method is commonly used. C4.5 uses a statistical method to estimates the errors at each node for pruning.
    - A validation set may be used for pruning as well.

Other issues in decision tree learning

- From tree to rules
- Handling of miss values
- Handing skewed distributions
- Handling attributes and classes with different costs.
- Attribute construction
- Etc.

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

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

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

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex.

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

  - gain_ratio(income) = 0.029/1.557 = 0.019
- The attribute with the maximum gain ratio is selected as the splitting attribute

Gini Index (CART, IBM IntelligentMiner)

- 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*)

Computation of Gini Index

- Ex. D has 9 tuples in buys_computer = "yes" and 5 in "no"

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2

$$gini_{income \in \{low,medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$

$$= 0.443$$

$$= Gini_{income \in \{high\}}(D).$$

$Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes

Python example of decision tree

Step 1 – Import Scikit-learn package into python

- `import sklearn`

Step 2 – load dataset or import Scikit-learn's dataset (sample)

- `from sklearn import datasets`
- `data = datasets.load_breast_cancer()`
- `dataset = pd.read_csv("diabetes.csv", header=None, names=col_names)`

Step 3 - Organizing data into sets

- `from sklearn.model_selection import train_test_split`
- `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)`

Step 4 – Building the model

- `clf = DecisionTreeClassifier()`
- `clf = clf.fit(X_train,y_train)`

Step 5 – Evaluating the model and its accuracy

Python example of decision tree

```
1. import pandas as pd
2. from sklearn import tree
3. from sklearn.tree import
   DecisionTreeClassifier # Import
   Decision Tree Classifier
4. from sklearn.model_selection
   import train_test_split # Import
   train_test_split function
5. from sklearn import metrics
   #Import scikit-learn metrics
   module for accuracy calculation
6. import matplotlib.pyplot as plt
7. data = pd.read_csv("diabetes.csv")
   # load dataset
8. print(pima.head()) # show some
   data
9. X = data[feature] #split dataset
   in features and target variable
10.y = data.label # Target variable
11.X_train, X_test, y_train, y_test
   = train_test_split(X, y,
   test_size=0.3, random_state=1) #
   Split dataset into 70% training
   set and 30% test set
12.clf = DecisionTreeClassifier() #
   Create Decision Tree classifer
   object
13.clf = clf.fit(X_train,y_train) #
   Train Decision Tree Classifer
14.y_pred = clf.predict(X_test)
   #Predict the response for test
   dataset
15.print("Accuracy:",metrics.accurac
   y_score(y_test, y_pred)) # Model
   Accuracy
16.tree.plot_tree(clf)  # draw tree
17.plt.show()
```

Contents

- Introduction to machine learning
- Supervised learning
- Classification
- Decision Tree Classifier
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- k-Nearest Neighbor Method
- Evaluating classification methods
- Exercise

Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem: Basics

- Total probability Theorem: $P(B) = \sum_{i=1}^{M} P(B|A_i)P(A_i)$

- Bayes' Theorem: $P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$

  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posteriori probability):* the probability that the hypothesis holds given the observed data sample **X**
  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed
  - P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds
    - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Prediction Based on Bayes' Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be viewed as

  posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification Is to Derive the Maximum Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifier: Training Dataset

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data to be classified:
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

age	income	student	credit_rating	com
<=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

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	comp
<=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

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643

  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class

  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222

  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6

  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444

  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667

  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667

  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028

P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

Pseudo code for naive bayes algorithm

```
# Training Phase
# Initialize class probabilities and feature conditional probabilities
for each class C:
    count(C) = 0
    for each feature F:
        count(F|C) = 0

# Count occurrences of classes and features in the training data
for each training example (X, C):
    count(C) += 1
    for each feature F in X:
        count(F|C) += 1
# Calculate class probabilities and feature conditional probabilities
for each class C:
    class_prob(C) = count(C) / total_training_examples
    for each feature F:
        feature_prob(F|C) = count(F|C)/ count(C)

# Testing Phase
# Given a new example X to classify:
for each class C:
    score(C) = log(class_prob(C))
    for each feature F in X:
        score(C) += log(feature_prob(F|C))

# Choose the class with the highest score as the predicted class for X
predicted_class = argmax(score(C) for each class C)
```

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)
- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

    Prob(income = high) = 11/1003
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

Naïve Bayes Classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.

      Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayes Classifier

Python example of Naïve Bayes Classifier

```
1. # import dataset
2. from sklearn import datasets
3. data = datasets.load_breast_cancer()
4. print(data.keys())
5. X = data.data # Features
6. y = data.target # Target variable

7. # Split dataset into training set and test set
8. from sklearn.model_selection import train_test_split
9. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
   random_state=1) # 70% training and 30% test

10. from sklearn.naive_bayes import GaussianNB
11. gnb = GaussianNB()
12. model = gnb.fit(X_train,y_train)
13. y_pred = gnb.predict(X_test)

14. print(y_pred)
```

Contents

- Machine learning
- Supervised learning
- Classification
- Decision Tree Classifier
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- k-Nearest Neighbor Method
- Evaluating classification methods
- Exercise

Support Vector Machines (SVM)

- SVM are among the most popular machine-learning techniques
- SVM belong to the family of generalized linear models (capable of representing non-linear relationships in a linear fashion)
- SVM achieve a classification or regression decision based on the value of the linear combination of input features
- Because of their architectural similarities, SVM are also closely associated with Artificial neural networks (ANN)

Support Vector Machines (SVM)

- Goal of SVM: to generate mathematical functions that map input variables to desired outputs for classification or regression type prediction problems
  - First, SVM uses nonlinear kernel functions to transform non-linear relationships among the variables into linearly separable feature spaces.
  - Then, the maximum-margin hyperplanes are constructed to optimally separate different classes from each other based on the training dataset.
- SVM has a solid mathematical foundation

Basic concepts

- Let the set of training examples D be

  $$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\},$$

  where $\mathbf{x}_i = (x_1, x_2, \ldots, x_n)$ is an **input vector** in a real-valued space $X \subseteq R^n$ and y_i is its **class label** (output value), $y_i \in \{1, -1\}$.

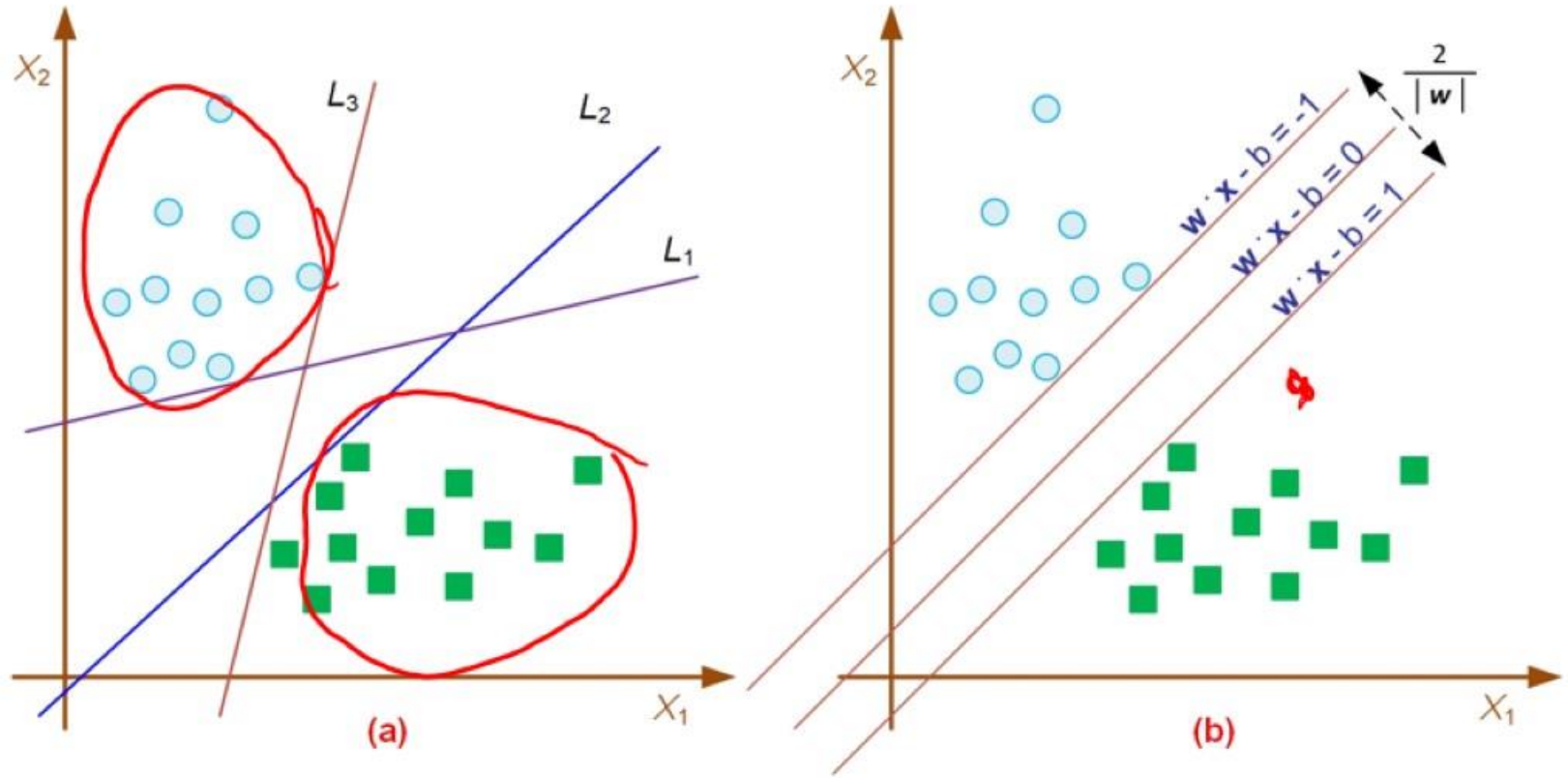
  1: positive class and -1: negative class.

- SVM finds a linear function of the form (**w**: weight vector)

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$$

$$y_i = \begin{cases} 1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

Support Vector Machines (SVM)

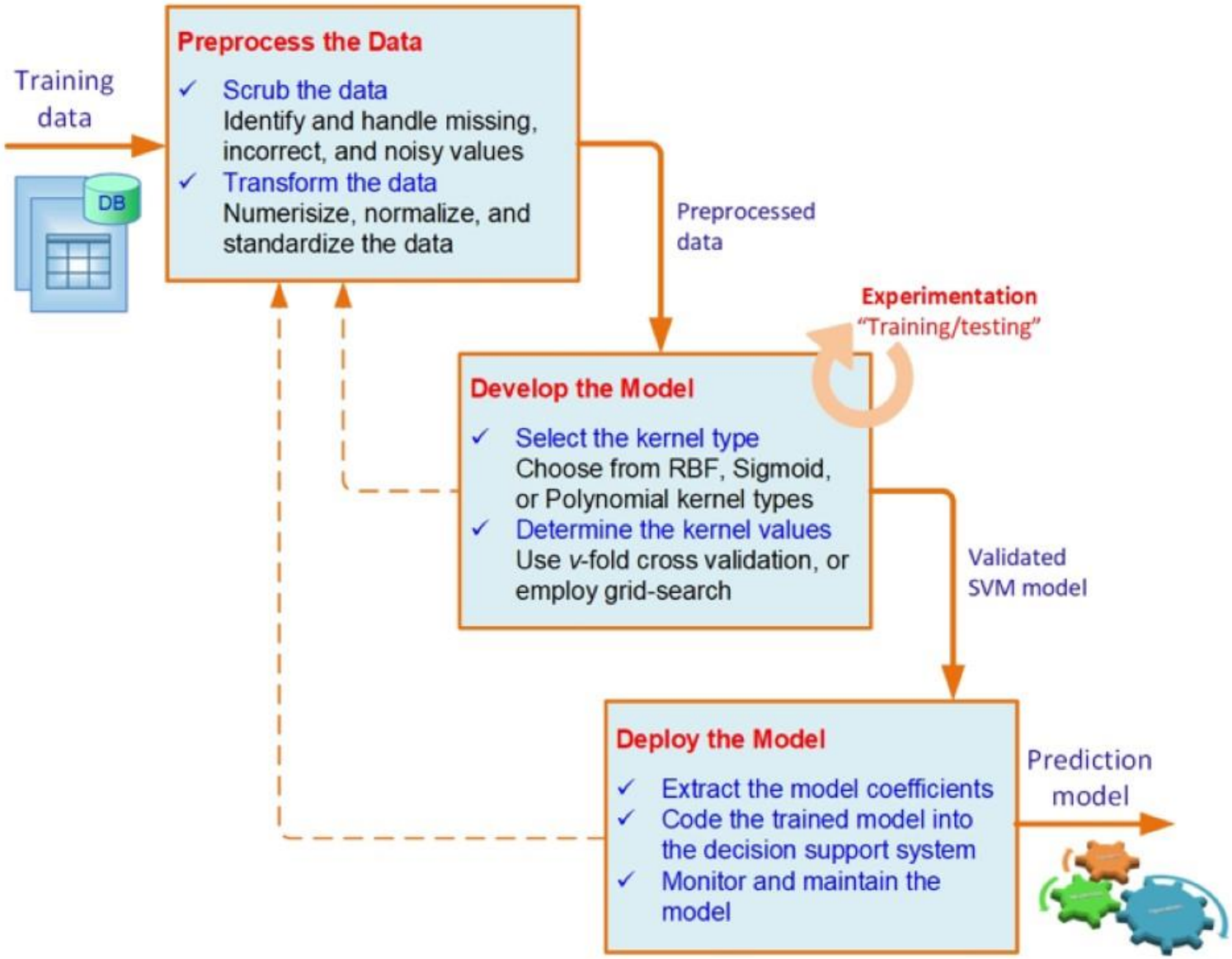


- Many linear classifiers may separate the data
- Separators can be lines, planes, hyperplanes

How Does a SVM Works?

- Following a machine-learning process, a SVM learns from the historic cases/samples
- The Process of Building SVM
  1. Preprocess the data
     - Scrub and transform the data.
  2. Develop the model
     - Select the kernel type (RBF is often a natural choice)
     - Determine the kernel parameters for the selected kernel
     - If the results are satisfactory, finalize the model, otherwise change the kernel type and/or kernel parameters to achieve the desired accuracy level
  3. Finalize and deploy the model

The Process of Building a SVM



Support vector machines (SVM)

- Support vector machines were invented by V. Vapnik and his co-workers in 1970s in Russia and became known to the West in 1992.
- SVMs are linear classifiers that find a hyperplane to separate two class of data, positive and negative.
- Kernel functions are used for nonlinear separation.
- SVM not only has a rigorous theoretical foundation, but also performs classification more accurately than most other methods in applications, especially for high dimensional data.
- It is perhaps the best classifier for text classification.

Some other issues in SVM

- SVM works only in a real-valued space. For a categorical attribute, we need to convert its categorical values to numeric values.
- SVM does only two-class classification. For multi-class problems, some strategies can be applied, e.g., one-against-rest, and error-correcting output coding.
- The hyperplane produced by SVM is hard to understand by human users. The matter is made worse by kernels. Thus, SVM is commonly used in applications that do not required human understanding.

SVM Python Example

```
1.  import numpy as np
2.  from sklearn.datasets import
    load_breast_cancer
3.  from sklearn.model_selection import
    train_test_split
4.  from sklearn.svm import SVC
5.  from sklearn.metrics import
    accuracy_score
6.  import matplotlib.pyplot as plt

7.  # Load the cancer dataset
8.  cancer = load_breast_cancer()

9.  # Split the data into features and
    labels
10. X = cancer.data
11. y = cancer.target

12. # Create a support vector classifier
13. clf = SVC()

14. # Fit the classifier to the data
15. clf.fit(X, y)

16. # Split the dataset into training
    and testing sets
17. X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.2,
    random_state=42)

18. # Initialize the Support Vector
    Classifier
19. svc_classifier = SVC()

20. # Train the classifier on the
    training data
21. svc_classifier.fit(X_train, y_train)

22. # Make predictions on the test data
23. y_pred =
    svc_classifier.predict(X_test)

24. # Calculate the accuracy of the
    classifier
25. accuracy = accuracy_score(y_test,
    y_pred)
26. print(f"Accuracy: {accuracy:.2f}")

27. # Plot the support vector machine
    boundaries
28. plt.scatter(X[:, 0], X[:, 1], c=y)
29. plt.show()
```

Contents

- Introduction to machine learning
- Supervised learning
- Classification
- Decision Tree Classifier
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- k-Nearest Neighbor Method (KNN)
- Evaluating classification methods
- Exercise

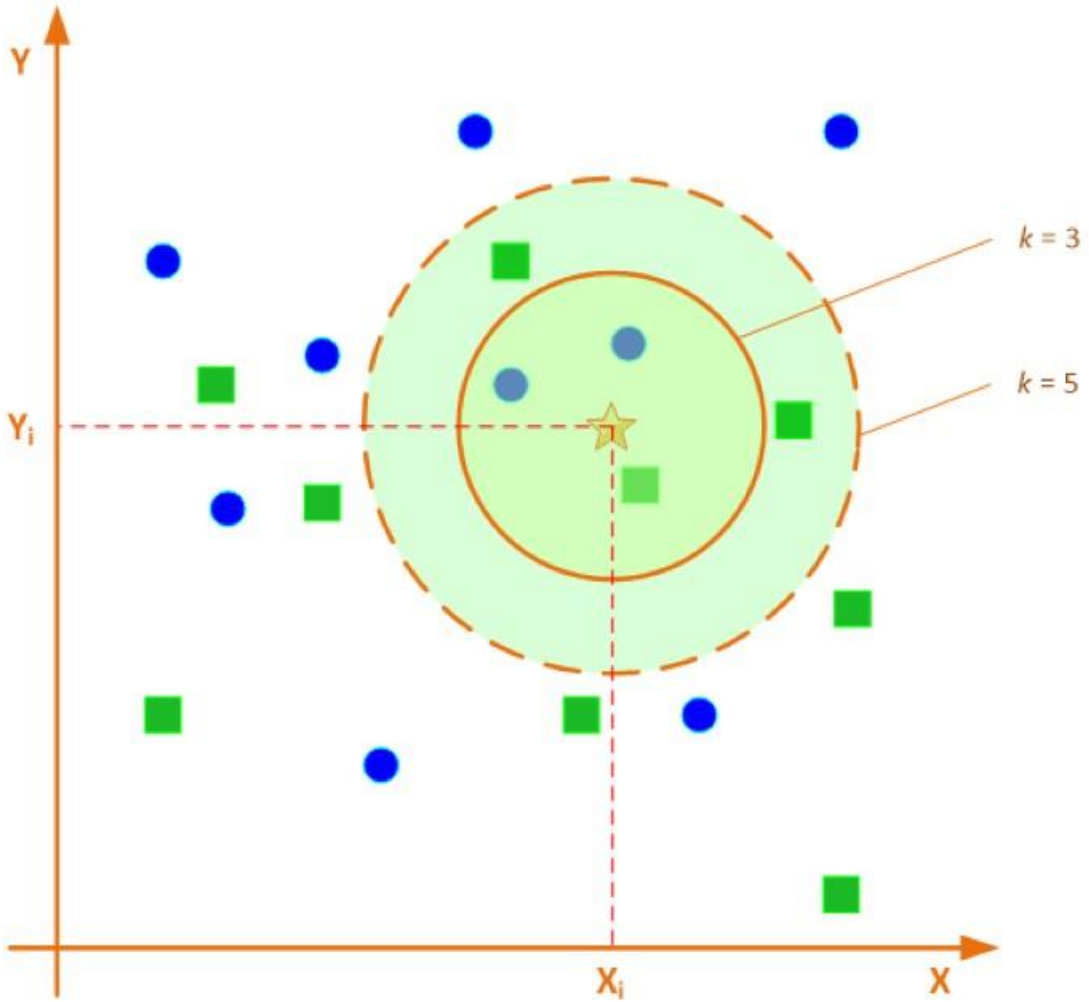
k-Nearest Neighbor Classification (kNN)

- Unlike all the previous learning methods, kNN does not build model from the training data.
- To classify a test instance d, define k-neighborhood P as k nearest neighbors of d
- Count number n of training instances in P that belong to class c_j
- Estimate $\Pr(c_j|d)$ as n/k
- No training is needed. Classification time is linear in training set size for each test case.

Nearest Neighbor

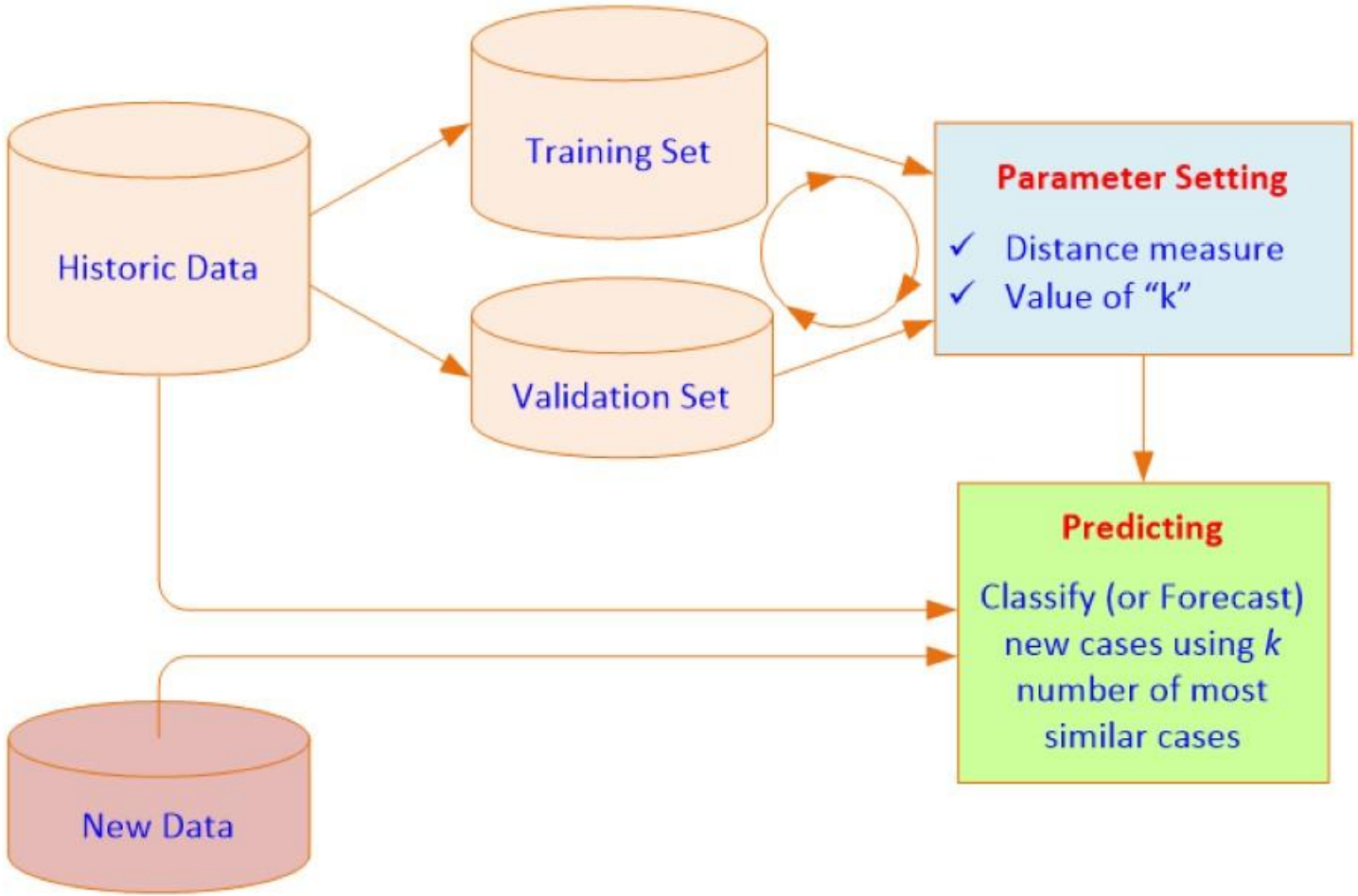
- Also called *k*-Nearest Neighbor Method (*k*NN)
  - *k* = the number of neighbors used in the model
- ANNs and SVMs → time-demanding, computationally intensive, iterative derivations
- *k*NN is a simplistic and logical prediction method, that produces very competitive results
- *k*NN is a prediction method for classification as well as regression types (as is the case with ANN & SVM)
- *k*NN is a type of instance-based learning (or lazy learning) – most of the work takes place at the time of prediction (not at modeling)

k-Nearest Neighbor Algorithm (*k*NN)



- ✓ The answer to the question of "which class a data point belongs to?" depends on the value of *k* (the number of neighbors)

The Process of *k*NN Method



*k*NN Model Parameter

1. Similarity Measure: The Distance Metric

Minkowski distance Most generic distance formulation

$$d(i, j) = \sqrt[q]{(|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \ldots + |x_{ip} - x_{jp}|^q)}$$

Two special cases of the generic formulation:

If $q = 1$, then d is called Manhattan distance

$$d(i, j) = \sqrt{|x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{ip} - x_{jp}|}$$

If $q = 2$, then d is called Euclidean distance

$$d(i, j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \ldots + |x_{ip} - x_{jp}|^2)}$$

- Numeric versus nominal values?
  - *k*NN can uses only the numeric input variables

*k*NN Model Parameter

2. Number of Neighbors (the value of *k*)
   - The best value depends on the data set
   - Larger values reduces the effect of noise but also make boundaries between classes less distinct
   - An "optimal" value can be found heuristically through experimentation
   - Cross validation is often used to determine the best value for *k* and the appropriate distance measure for a given data set
     - *Best parameters produce best prediction results*

kNNAlgorithm

Algorithm kNN(D, d, k)

1 Compute the distance between d and every example in D;
2 Choose the k examples in D that are nearest to d, denote the set by P ($\subseteq D$);
3 Assign d the class that is the most frequent class in P (or the majority class);

- k is usually chosen empirically via a validation set or cross-validation by trying a range of k values.
- Distance function is crucial, but depends on applications.

k-Nearest Neighbor Method

- kNN can deal with complex and arbitrary decision boundaries.
- Despite its simplicity, researchers have shown that the classification accuracy of kNN can be quite strong and in many cases as accurate as those elaborated methods.
- kNN is slow at the classification time
- kNN does not produce an understandable model

k-Nearest Neighbor Python Example

```
# Importing required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Loading the sklearn diabetes dataset
X, Y = load_diabetes(return_X_y=True)

# Taking only one feature to perform simple linear regression
X = X[:,8].reshape(-1,1)

# Splitting the dependent and independent features of the dataset into training and testing dataset
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.3, random_state = 10 )

# Creating an instance for the linear regression model of sklearn
lr = linear_model.LinearRegression()

# Training the model by passing the dependent and independent features of the training dataset
lr.fit( X_train, Y_train )

# Creating an array of predictions made by the model for the unseen or test dataset
Y_pred = lr.predict( X_test )

# The value of the coefficients for the independent feature through the multiple regression model
print("Value of the oefficients: \n", lr.coef_)

# The value of the mean squared error
print(f"Mean square error: {mean_squared_error( Y_test, Y_pred)}")

# The value of the coefficient of determination, i.e., R-square score of the model
print(f"Coefficient of determination: {r2_score( Y_test, Y_pred )}")

# Plotting the output
plt.scatter(X_test, Y_test, color = "black", label = "original data")
plt.plot(X_test, Y_pred, color = "blue", linewidth=3, label = "regression line")
plt.xlabel("Independent Feature")
plt.ylabel("Target Values")
plt.title("Simple Linear Regression")
plt.show()
```

Contents

- Machine learning
- Supervised learning
- Classification
- Decision Tree Classifier
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- k-Nearest Neighbor Method
- Evaluating classification methods
- Exercise

Evaluating classification methods

- **Predictive accuracy**

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

- Efficiency
  - time to construct the model
  - time to use the model
- Robustness: handling noise and missing values
- Scalability: efficiency in disk-resident databases
- Interpretability:
  - understandable and insight provided by the model
- Compactness of the model: size of the tree, or the number of rules.

Model Evaluation and Selection

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
  - Holdout method, random subsampling
  - Cross-validation
  - Bootstrap
- Comparing classifiers:
  - Confidence intervals
  - Cost-benefit analysis and ROC Curves

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 boostrap**
  - A data set with *d* tuples is sampled *d* times, with replacement, resulting in a training set of *d* samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data end up in the bootstrap, and the remaining 36.8% form the test set (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)
  - Repeat the sampling procedure *k* times, overall accuracy of the model:

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

Evaluation methods

- **Holdout set**: The available data set *D* is divided into two disjoint subsets,
  - the *training set* D_{train} (for learning a model)
  - the *test set* D_{test} (for testing the model)
- **Important:** training set should not be used in testing and the test set should not be used in learning.
  - Unseen test set provides a unbiased estimate of accuracy.
- The test set is also called the holdout set. (the examples in the original data set *D* are all labeled with classes.)
- This method is mainly used when the data set *D* is large.

Evaluation methods (cont...)

- **n-fold cross-validation**: The available data is partitioned into *n* equal-size disjoint subsets.
- Use each subset as the test set and combine the rest *n*-1 subsets as the training set to learn a classifier.
- The procedure is run *n* times, which give *n* accuracies.
- The final estimated accuracy of learning is the average of the *n* accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

Evaluation methods (cont...)

- **Leave-one-out cross-validation**: This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only a single test example and all the rest of the data is used in training.
- If the original data has *m* examples, this is *m*-fold cross-validation

Evaluation methods (cont...)

- **Validation set**: the available data is divided into three subsets,
  - a training set,
  - a validation set and
  - a test set.
- A validation set is used frequently for estimating parameters in learning algorithms.
- In such cases, the values that give the best accuracy on the validation set are used as the final parameter values.
- Cross-validation can be used for parameter estimating as well.

Classification measures

- Accuracy is only one measure (error = 1-accuracy).
- **Accuracy is not suitable in some applications**.
- In text mining, we may only be interested in the documents of a particular topic, which are only a small portion of a big document collection.
- In classification involving skewed or highly imbalanced data, e.g., network intrusion and financial fraud detections, we are interested only in the minority class.
  - High accuracy does not mean any intrusion is detected.
  - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing.
- The class of interest is commonly called the **positive class**, and the rest **negative classes.**

Precision and recall measures

- Used in information retrieval and text classification.
- We use a confusion matrix to introduce them.

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where

TP: the number of correct classifications of the positive examples (**true positive**),

FN: the number of incorrect classifications of positive examples (**false negative**),

FP: the number of incorrect classifications of negative examples (**false positive**), and

TN: the number of correct classifications of negative examples (**true negative**).

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?
- Perfect score is 1.0

$$recall = \frac{TP}{TP + FN}$$

- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

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

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

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

Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity)*
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

- Precision *p* is the number of correctly classified positive examples divided by the total number of examples that are classified as positive.
- Recall *r* is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set.

An example

	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

- This confusion matrix gives
  - precision p = 100% and
  - recall r = 1%

    because we only classified one positive example correctly and no negative examples wrongly.
- Note: precision and recall only measure classification on the positive class.

F_1-value (also called F_1-score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure

$$F_1 = \frac{2pr}{p + r}$$

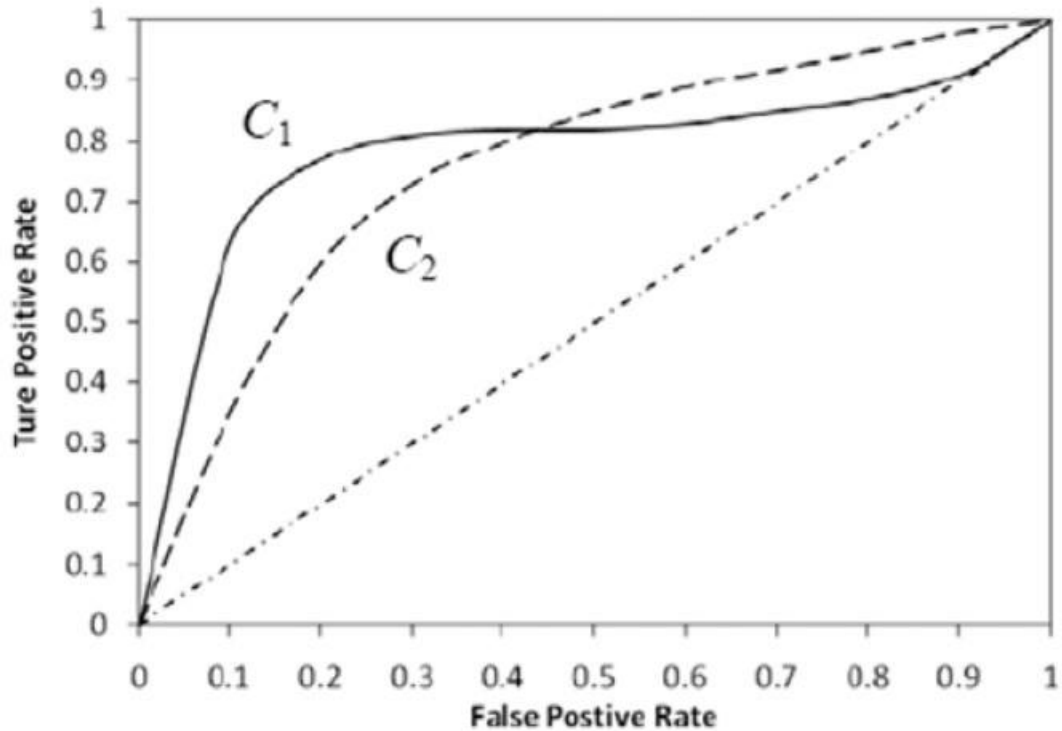
F_1-score is the harmonic mean of precision and recall.

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F_1-value to be large, both *p* and *r* much be large.

Receive operating characteristics (ROC) curve

- It is commonly called the ROC curve.
- It is a plot of the true positive rate (TPR) against the false positive rate (FPR).
- True positive rate: $$TPR = \frac{TP}{TP + FN}$$
- False positive rate: $$FPR = \frac{FP}{TN + FP}$$



Sensitivity and Specificity

- In statistics, there are two other evaluation measures:
  - Sensitivity: Same as true positive rate (TPR)
  - Specificity: Also called True Negative Rate (TNR)

- Then we have

$$TNR = \frac{TN}{TN + FP}$$

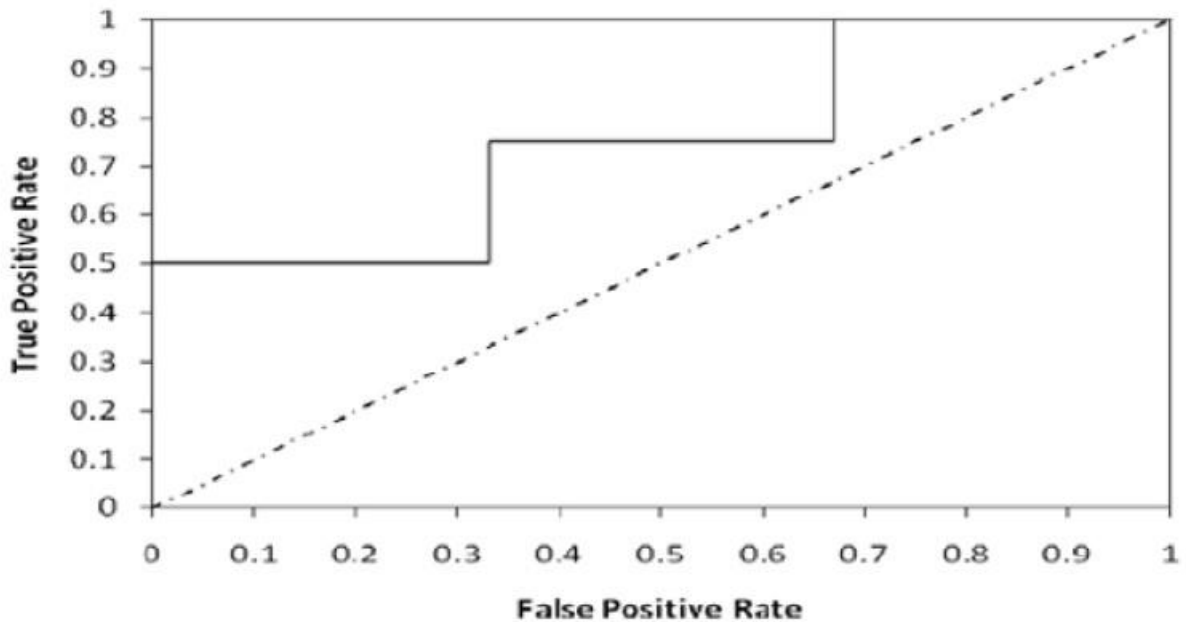
$$FPR = 1 - specificity$$

Area under the curve (AUC)

- Which classifier is better, C_1 or C_2?
  - It depends on which region you talk about.
- Can we have one measure?
  - Yes, we compute the area under the curve (AUC)
- If AUC for C_i is greater than that of C_j, it is said that C_i is better than C_j.
  - If a classifier is perfect, its AUC value is 1
  - If a classifier makes all random guesses, its AUC value is 0.5.

Drawing an ROC curve

Rank		1	2	3	4	5	6	7	8	9	10
Actual class		+	+	–	–	+	–	–	+	–	–
TP	0	1	2	2	2	3	3	3	4	4	4
FP	0	0	0	1	2	2	3	4	4	5	6
TN	6	6	6	5	4	4	3	2	2	1	0
FN	4	3	2	2	2	1	1	1	0	0	0
TPR	0	0.25	0.5	0.5	0.5	0.75	0.75	0.75	1	1	1
FPR	0	0	0	0.17	0.33	0.33	0.50	0.67	0.67	0.83	1



Another evaluation method: Scoring and ranking

- Scoring is related to classification.
- We are interested in a single class (positive class), e.g., buyers class in a marketing database.
- Instead of assigning each test instance a definite class, scoring assigns a probability estimate (PE) to indicate the likelihood that the example belongs to the positive class.

Ranking and lift analysis

- After each example is given a PE score, we can rank all examples according to their PEs.
- We then divide the data into n (say 10) bins. A lift curve can be drawn according how many positive examples are in each bin. This is called lift analysis.
- Classification systems can be used for scoring. Need to produce a probability estimate.
  - E.g., in decision trees, we can use the confidence value at each leaf node as the score.

An example

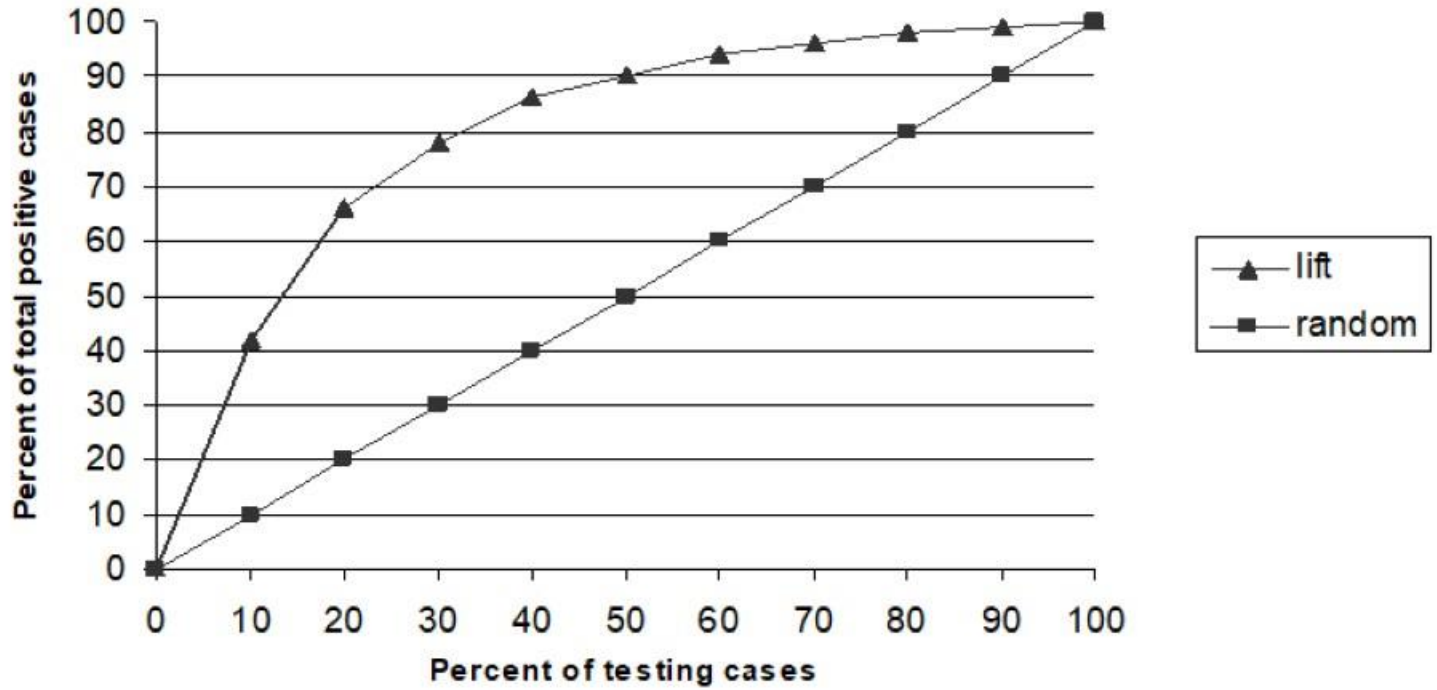
- We want to send promotion materials to potential customers to sell a watch.
- Each package cost $0.50 to send (material and postage).
- If a watch is sold, we make $5 profit.
- Suppose we have a large amount of past data for building a predictive/classification model. We also have a large list of potential customers.
- How many packages should we send and who should we send to?

An example

- Assume that the test set has 10000 instances. Out of this, 500 are positive cases.
- After the classifier is built, we score each test instance. We then rank the test set, and divide the ranked test set into 10 bins.
  - Each bin has 1000 test instances.
  - Bin 1 has 210 actual positive instances
  - Bin 2 has 120 actual positive instances
  - Bin 3 has 60 actual positive instances
  - …
  - Bin 10 has 5 actual positive instances

Lift curve

Bin	1	2	3	4	5	6	7	8	9	10
	210	120	60	40	22	18	12	7	6	5
	42%	24%	12%	8%	4.40%	3.60%	2.40%	1.40%	1.20%	1%
	42%	66%	78%	86%	90.40%	94%	96.40%	97.80%	99%	100%



Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Estimating Confidence Intervals: Classifier Models M_1 vs. M_2

- Suppose we have 2 classifiers, M_1 and M_2, which one is better?
- Use 10-fold cross-validation to obtain $\overline{err}(M_1)$ and $\overline{err}(M_2)$
- These mean error rates are just *estimates* of error on the true population of *future* data cases
- What if the difference between the 2 error rates is just attributed to *chance*?
  - Use a **test of statistical significance**
  - Obtain **confidence limits** for our error estimates

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**: handling noise and missing values
- **Scalability**: efficiency in disk-resident databases
- **Interpretability**
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Summary

- Applications of supervised learning are in almost any field or domain.
- We looked at 4 classification techniques.
- There are still many other methods, e.g.,
  - Neural networks
  - Genetic algorithms
  - Fuzzy classification

  This large number of methods also show the importance of classification and its wide applicability.
- It remains to be an active research area.

Model Evaluation python example

```
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, log_loss,
confusion_matrix, roc_auc_score, f1_score,
mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler

# Load the diabetes dataset
data = load_diabetes()
X = data.data
y = (data.target > 150).astype(int)  # Convert to binary
classification problem

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(),
    "Naive Bayes": GaussianNB(),
    "SVM": SVC(probability=True),
    "KNN": KNeighborsClassifier()
}

# Initialize metrics
metrics = {
    "Classification Accuracy": accuracy_score,
    "Logarithmic Loss": log_loss,
    "Confusion Matrix": confusion_matrix,
    "Area under Curve": roc_auc_score,
    "F1 Score": f1_score,
    "Mean Absolute Error": mean_absolute_error,
    "Mean Squared Error": mean_squared_error
}

# Iterate over classifiers
for name, clf in classifiers.items():
    print(f"Classifier: {name}")
    clf.fit(X_train, y_train)

    # Predict probabilities for Log Loss and AUC
    if name in ["Decision Tree", "KNN"]:
        y_pred_prob = clf.predict_proba(X_test)[:, 1]
    else:
        y_pred_prob = clf.predict_proba(X_test)[:, 1]

    # Iterate over metrics
    for metric_name, metric_func in metrics.items():
        if metric_name == "Confusion Matrix":
            print(f"{metric_name}:\n{metric_func(y_test,
clf.predict(X_test))}")
        else:
            print(f"{metric_name}: {metric_func(y_test,
y_pred_prob if 'AUC' in metric_name else
clf.predict(X_test)):.4f}")

    print("="*50)
```

Exercise 2

1. Develop new ML and AI algorithms,

- N/A

2. Develop new and efficient ML packages and implementation,

- N/A

3. Application of existing ML and AI algorithms to solve new or existing problems,

- Import the diabetes data and use at least 5 ML algorithms to evaluate the performance of the models. Ensure that you print the precision, recall, AUC score and f1-scores as well as plot ROC curve and plot the Confusion Matrix calculate

4. Use of existing ML and AI software to solve new or existing problems,

- List 3 AI software that uses classifiers. You can Google or use ChatGPT or BARD

References and further reading

- Stuart Russell (2021), Artificial Intelligence: A Modern Approach, 4th edition. Pearson publishing
- Dursun Delen (2021), Predictive Analytics: Data Mining, Machine Learning and Data Science for Practitioners, 2nd edition, Pearson FT Press
- Paul Deitel and Harvey M. Deitel (2020), Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and The Cloud, 1st edition, Pearson Publishing
- Mark E. Fenner (2020), Machine Learning with Python for Everyone, Addison-Wesley Publishing
- Pang-Ning Tan, Michael Steinbach, Vipin Kumar (2018), Introduction to Data Mining (2nd Edition), Pearson publishing
- Data mining: Concepts and Techniques, by Jiawei Han and Micheline Kamber, Morgan Kaufmann Publishers, ISBN 1-55860-489-8.
- Machine Learning, by Tom M. Mitchell, McGraw-Hill, ISBN 0-07-042807-7
- Modern Information Retrieval, by Ricardo Baeza-Yates and Berthier Ribeiro-Neto, Addison Wesley, ISBN 0-201-39829-X

References and further reading

- https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
- https://www.cs.uic.edu/~liub/teach/cs583-fall-23/cs583.html
- https://www.edlitera.com/en/blog/posts/evaluating-classification-models
- https://www.cs.uic.edu/~liub/teach/cs583-fall-23/cs583.html
- https://www.analyticsvidhya.com/blog/2021/12/evaluation-of-classification-model/
- https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
- https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/