Classification

Unit 3: Classification

Concept, Evaluation, Rule Based Classifiers, Decision Tree, Baysian Classifier

–

School of Mathematical Sciences
Institute of Science and Technology (IoST), TU

Feb, 2025

Table of contents

- Concepts
- Evaluation
- Rule-Based Approach
- Decision Tree
- Bayesian Classification

Classification

What is classification?

Classification is a supervised learning task where a model assigns discrete labels (classes) to input data based on learned patterns from labeled training examples

Input Features (X)

Variables describing the data (e.g., pixel values, text embeddings)

Output Labels (y)

Categorical targets (e.g., spam/not spam, positive/negative)

Model

A function $f(X) \rightarrow y$ (e.g., logistic regression, decision trees)

Classification - Defination

Given a collection of records (training dataset), each records are characterized by a *tuple*(x, y), where x is the **attribute set** and y is the **class label**

x: attribute, predictor, independent variable, input
y: class, response, dependent variable, output

Task

Learn a model that maps each attribute set x into one of the predefined class labels y

Model

A classification model is an **abstract representation** of the relationship between the attribute set and the class label

Classification - Examples

Email message categorization

Dataset: Features extracted from email message header and content
Class Label: spam or non-spam

Identifying tumor cells

Dataset: Features extracted from x-rays or MRI scans
Class Label: malignant or benign cells

Cataloging galaxies

Dataset: Features extracted from telescope images
Class Label: Elliptical, spiral, or irregular-shaped galaxies

Classification Models - Predictive, Descriptive

Predictive Model

- Used to classify the previously unlabeled instances
- A good classification model must provide accurate predictions with a fast response time

Descriptive Model

- Used to identify the characteristics that distinguish instances from different classes
- This is particularly useful for critical applications, such as medical diagnosis, where it is insufficient to have a model that makes a prediction without justifying how it reaches such a decision

Classification Model (cont.)

Predictive or Descriptive?

Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

Predictive Model

Whole dataset can be used to predict the class label of the given vertebrate

Descriptive Model

It can be used as a descriptive model to help determine characteristics that define a vertebrate as a mammal

General Framework for Classification Model

Training Set

The model is created using a given a set of instances, known as the training set, which contains attribute values as well as class labels for each instance

Learning Algorithm

The systematic approach for learning a classification model given a training set is known as a learning algorithm.

- Rule Based Models
- Decision Tree based Models
- Probabilistics Models - Naive Bayes, Bayesian
- Generic Models - SVM, ANN
- Ensemble Classifier - Boosting, Bagging, Random Forests

General Framework for Classification Model (cont.)

Induction

The process of using a learning algorithm to build a classification model from the training data is known as induction

Classifier / Model

It is typically described in terms of a model

Deduction

This process of applying a classification model on unseen test instances to predict their class labels is known as deduction

Evaluation

Why Evaluate a Classification Model?

- To measure the model's performance on unseen data.
- To compare different models and select the best one.
- To understand the types of errors made by the model.
- To ensure reliability and robustness in real-world applications.

Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model in machine learning. It provides a detailed breakdown of the model's predictions compared to the actual labels, allowing for a clear understanding of where the model is performing well and where it is making errors.

For a binary classification problem, the confusion matrix is a 2x2 table with the following components:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Confusion Matrix (cont.)

Benefits

- Provides a detailed error analysis
- Helps in computing various performance metrics

Weaknesses

- Does not account for class imbalance
- Can be misleading if only accuracy is considered

Accuracy (cont.)

Measures the proportion of correct predictions out of the total predictions

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Measures overall correctness
- Can be misleading if classes are imbalanced

Example If TP = 50, TN = 40, FP = 5, FN = 5, then:

$$\text{Accuracy} = \frac{50 + 40}{50 + 40 + 5 + 5} = 90\%$$

Accuracy (cont.)

Consider a 2-class problem

- Number of Class 0 examples = 9990
- Number of Class 1 examples = 10

What is the accuracy if model predicts everything to be class 0

$$\textit{accuracy} = 9990/10000 = 99.9\%$$

Accuracy is misleading because model does not detect any class 1 example

Precision and Recall

Precision

Measures the proportion of correctly predicted positive instances out of all predicted positive instances.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (Sensitivity or True Positive Rate)

Measures the proportion of correctly predicted positive instances out of all actual positive instances.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision and Recall

Interpretation

- Precision: How many predicted positives are actually correct?
- Recall: How many actual positives are correctly identified?

Example If TP = 50, FP = 10, FN = 20:

$$\text{Precision} = \frac{50}{50 + 10} = 0.83, \quad \text{Recall} = \frac{50}{50 + 20} = 0.71$$

F1-Score

The harmonic mean of precision and recall, providing a balance between the two.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Harmonic mean of Precision and Recall.
- Balances both metrics, useful for imbalanced datasets.

Example

$$\text{F1-Score} = 2 \times \frac{0.83 \times 0.71}{0.83 + 0.71} = 0.77$$

Interpretation: Precision=0.83 and Recall = 0.71

Precision

- A precision of 0.83 means that 83% of the instances predicted as positive by the model are actually positive.
- This indicates that the model has a relatively low rate of false positives (FP). In other words, when the model predicts a positive outcome, it is correct 83% of the time.

Recall

- A recall of 0.71 means that the model correctly identifies 71% of all actual positive instances.
- This indicates that the model misses 29% of the actual positive instances (false negatives, FN). In other words, the model fails to capture some of the positive cases.

Trade-off Between Precision and Recall

Precision and recall often have an **inverse relationship**, meaning improving one may lead to a decrease in the other. This trade-off depends on the model's threshold for classifying positive instances:

- **High Precision, Lower Recall:** The model is conservative in predicting positives, resulting in fewer false positives but potentially missing some actual positives (higher false negatives).
- **High Recall, Lower Precision:** The model is more aggressive in predicting positives, capturing more actual positives but also increasing false positives.

Trade-off Between Precision and Recall

(cont.)

The interpretation of precision and recall depends on the specific problem and the cost of errors:

High Precision is Important

- In scenarios where false positives are costly, high precision is desirable.
- **Example** Spam detection (incorrectly marking a non-spam email as spam is more costly than missing some spam emails).

Trade-off Between Precision and Recall

(cont.)

The interpretation of precision and recall depends on the specific problem and the cost of errors:

High Recall is Important

- In scenarios where false negatives are costly, high recall is desirable.
- **Example** Medical diagnosis (missing a positive case, such as a disease, is more costly than false alarms).

Balanced Precision and Recall

In some cases, a balance between precision and recall is needed, which can be measured using the F1-score.
The F1-score for your model is:

$$F1{-}score = 2\times\frac{Precision \times Recall}{(Precision + Recall)} = 2\times(0.83\times0.71)/(0.83+0.71) = 0.76$$

This indicates a reasonably balanced performance.

Rule-Based Approach

Problem: Classifying Emails as Spam or Not Spam (Ham) Using Rule-Based Approach

We will define a simple rule-based classifier based on common spam email characteristics.

- Define the dataset
- Define the rules
- Implement the classifier
- Evaluate

Step 1: Define the Dataset

The dataset consists of email samples with features such as:

- Contains "buy now" (Boolean)
- Contains "free" (Boolean)
- Contains "limited offer" (Boolean)
- Contains "urgent" (Boolean)
- Number of words (Integer)
- Label: Spam or Ham (Non-Spam)

Step 2: Define Rule-Based Classifier

We'll classify an email as Spam if it matches any of these rules:

- If an email **contains** "buy now" AND "free", classify it as Spam.
- If an email **contains** "limited offer" AND "urgent", classify it as Spam.
- If an email **contains** any three of the spam words, classify it as Spam.
- If an email has **less than 40 words** AND contains any two spam words, classify it as Spam.
- Otherwise, classify it as Ham.

Step 3: Implementing the Rule-Based Classifier

It is mostly if else statements
Use any programming language to interprate those rules

Sample dataset

```
data = [
    {"buy_now": 1, "free": 1, "limited_offer": 0,
      "urgent": 1, "word_count": 50, "label": "Spam"},
    {"buy_now": 0, "free": 1, "limited_offer": 0,....
]
```

Classifier

```
def classify_email(email):
    spam_words = sum([email["buy_now"], email["free"],
         email["limited_offer"], email["urgent"]])

    if email["buy_now"] and email["free"]:
        return "Spam"
    if email["limited_offer"] and email["urgent"]:
        return "Spam"
    if spam_words >= 3:
        return "Spam"
    if email["word_count"] < 40 and spam_words >= 2:
        return "Spam"
    return "Ham"
```

Inference

```
import pandas as pd
df = pd.DataFrame(data)

df["predicted_label"] = df.apply(classify_email, axis=1)

# Evaluate
accuracy = (df["label"] == df["predicted_label"]).mean()
print("Rule-Based Classifier Accuracy:", accuracy)
print(df)
```

Step 4: Evaluate

The rule-based model is evaluated by checking how many labels it correctly predicts. Accuracy is computed as:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Samples}$$

Example Problem

Classifying Patients as Diabetic or Non-Diabetic Using Rule-Based Approach
A patient is classified as Diabetic if they meet any of the following conditions:

- Blood Sugar Level 180 mg/dL = Diabetic
- BMI $\geq$ 30 AND Blood Sugar Level $\geq$ 140 mg/dL = Diabetic
- Age $\geq$ 50 AND Family History = 1 = Diabetic
- Age $\geq$ 40 AND Blood Sugar Level $\geq$ 150 mg/dL Diabetic
- Otherwise, classify as Non-Diabetic.

Example Problem

Classifier

```
def classify_patient(patient):
    if patient["blood_sugar"] >= 180:
        return "Diabetic"
    if patient["bmi"] >= 30 and patient["blood_sugar"] >= 140:
        return "Diabetic"
    if patient["age"] >= 50 and patient["family_history"] == 1:
        return "Diabetic"
    if patient["age"] >= 40 and patient["blood_sugar"] >= 150:
        return "Diabetic"
    return "Non-Diabetic"
```

Rule-Based Classification

Advantages of Rule-Based Classification

- Interpretable: The decision-making process is clear.
- Fast Execution: No need for training data.
- Works with Small Datasets: Unlike deep learning models, which require large labeled datasets

Disadvantages

- Limited Generalization: May not work well on unseen cases.
- Rule Engineering Required: Needs domain-specific rules.

Decison Tree

Decision Tree is a supervised learning algorithm used for classification tasks

It represents decisions and their possible consequences in a tree-like structure.

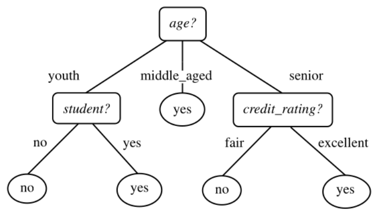
Each **internal node** represents a decision based on a feature, **branches** represent outcomes, and **leaves** represent final output

Classification by Decision Tree Induction

Class: **buys computer**

Each **internal (nonleaf) node** represents a test on an attribute.

Each leaf node represents a class (either buys *computer* = *yes* or *buyscomputer* = *no*)



Decision Tree (cont.)

How are decision trees used for classification?

Given a tuple, X, for which the associated class label is *unknown*, the attribute values of the tuple are tested against the decision tree.

A **path** is traced from the **root to a leaf node**, which holds the class prediction for that tuple.
Decision trees can easily be converted to classification rules.

Decision Tree (cont.)

Why are decision tree classifiers so popular?

- Easy to Understand and Interpret
- Handles numerical and categorical data
- Non-linear relationships - Unlike linear models, decision trees can capture complex, nonlinear patterns without needing feature engineering
- Minimal Assumptions - don't assume the data follows a specific distribution (e.g., normality), making them robust for messy, real-world datasets
- Foundation for Advanced Models - building blocks for powerful ensemble methods like Random Forests and Gradient Boosting (e.g., XGBoost), which amplify their strengths

Attribute Selection Measures

Decision Trees **divide data into subsets based on feature values**.

The best split is determined using a splitting criterion.

- Information gain
- Gain ratio
- Gini index

Entropy

Entropy (H) measures the uncertainty or impurity in a dataset.

- A dataset with **pure classes** (only one class) has **low entropy** (H = 0).
- A dataset with **equal distribution** of classes has **high entropy**.

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

where, p_i is the probability of class i in the dataset, m is distinct class

Entropy Interpretation

If all samples belong to the same class: $H = 0$ *(Pure dataset, no uncertainty, best case)*

If classes are evenly distributed (50-50 split in binary classification): $H = 1$ *(Maximum uncertainty, worst case)*

Entropy (cont.)

Dataset

Instances	Label
A	YES
B	YES
C	NO
D	NO

Class Probabilities

$$p_{YES} = 2/4 = 0.5 \qquad p_{NO} = 2/4 = 0.5$$

Entropy Calculation

$$H = -\sum p_i \log_2(p_i) = -(0.5 \times \log_2 0.5 + 0.5 \times log_2 0.5) = 1.0$$

Interpretation

- The dataset is highly impure
- We need to split the data to reduce entropy

Information Gain

Information Gain (IG) measures the reduction in entropy after a split. It helps decide which feature to split on.

$$IG = H(Parent) - \sum\left(\frac{|S_v|}{|S|} H(S_v)\right)$$

Where,
$H(Parent)$ = Entropy of the original dataset
S_v = Subset after splitting by a feature
$|S_v|/|S|$ = Proportion fo subset in dataset

Use Case

A self-driving car needs to decide whether to turn left or right at an intersection based on the surrounding road conditions. The decision should consider various features such as

- traffic signals
- pedestrian movement
- oncoming traffic
- lane blockage

Use Case (cont.)- Do youtself

The dataset used for decision-making are as follows. Make a decision tree.

Traffic Signal	Pedestrian Nearby	Oncoming Traffic	Left Lane Blocked	Right Lane Blocked	Decision
Green	No	No	No	No	Left
Green	No	Yes	No	Yes	Left
Red	No	No	No	No	No Turn
Green	Yes	No	No	Yes	Left
Green	No	Yes	Yes	No	Right
Green	Yes	Yes	No	No	Right
Green	Yes	No	No	No	Left
Green	No	Yes	Yes	Yes	No Turn
Red	Yes	No	No	No	No Turn
Green	No	No	Yes	No	Right
Green	No	No	No	Yes	Left
Green	Yes	Yes	Yes	No	Right

Table: Decision Dataset for Turning Left or Right

Where Information Gain fails?

The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values.

For example, consider an attribute that **acts as a unique identifier**, such as **product ID**. A split on product ID would result in a large number of partitions (as many as there are values), each one containing just one tuple.

Because **each partition is pure**, the information required to classify data set D based on this partitioning would be $IG_{productID}(D) = 0$.

Therefore, the information gained by partitioning on this attribute is maximal. Clearly, **such a partitioning is useless for classification**.

Gain Ratio

Attributes with many distinct values (e.g., unique IDs) may have high Information Gain but are not meaningful for decision-making.

Gain Ratio **penalizes splits that produce many small subsets**, making it more balance

It applies a normalization to information gain using a "split information" value defined analogously with $IG(D)$ as

$$SplitInfo(SI) = -\sum \frac{|S_i|}{|S|} \log_2\left(\frac{|S_i|}{|S|}\right)$$

Where, S is total dataset and S_i represent the subset created by splitting on an attribute

Gain Ratio (cont.)

Split Information (SI) is a normalization factor that accounts for how broadly an attribute divides the dataset.

$$GainRatio = \frac{\text{Information Gain}}{\text{Split Information}}$$

The attribute with the maximum gain ratio is selected as the splitting attribute.

C4.5 Algorithm selects the attribute with the highest Gain Ratio rather than just Information Gain.

Gini Index

The **Gini Index** (or **Gini Impurity**) measures how often a randomly chosen element would be incorrectly classified.

It is used in **CART (Classification and Regression Trees)** for decision tree splits.

$$Gini(S) = 1 - \sum p_i^2$$

Where, p_i is the probability of class i in the dataset S.

Gini Index(cont.)

Suppose we have a dataset with three classes:

- Left Turn: 5 samples
- Right Turn: 4 samples
- No Turn: 3 samples

$$Gini(S) = 1 - \left(\left(\frac{5}{12}\right)^2 + \left(\frac{4}{12}\right)^2 + \left(\frac{3}{12}\right)^2\right)$$
$$Gini(S) = 1 - (0.1736 + 0.1111 + 0.0625) = 0.6528$$

Gini Index(cont.)

Gini Index for Splitting

To determine the best split, we calculate the weighted Gini Index for each attribute and choose the one with the lowest value.

$$Gini_{split} = \sum \frac{|S_i|}{|S|} Gini(S_i)$$

Where, S_i are the subsets after splitting

Gini vs. Entropy

- **Gini Index:** Faster computation (no logarithms), used in CART.
- **Entropy (Information Gain):** Used in ID3, C4.5, requires logarithm calculation.
- **Key Difference:** Gini prefers balanced splits, while Information Gain may favor attributes with many unique values.

Decision Tree Induction

Algorithms

- Hunt's Algorithm
- ID3
- C4.5
- ID3
- SLIQ
- SPRINT

Read Yourself

- Go through the algorithms
- Note their weakness and strengths

Overfitting in Decision Tree

The problem of overfitting is considered when the algorithm continues to go deeper and deeper to reduce the training-set error but results with an increased test-set error.

So, accuracy of prediction for our model goes down.

It generally happens when we build many branches due to outliers and irregularities in data.

Tree Pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data.

Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

Tree Pruning (cont.)

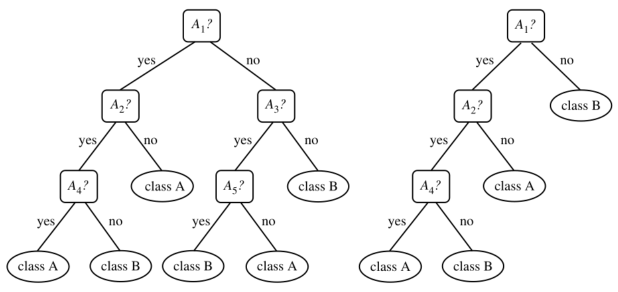


Figure: An unpruned decision tree and a pruned version of it

Tree Pruning (cont.)

How does tree pruning work?
There are two common approaches to tree pruning:

1. prepruning
2. postpruning

Prepruning

In the prepruning approach, a tree is "pruned" by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node)

Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.

If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. Choosing an appropriate threshold is challenging

- High thresholds could result in oversimplified trees
- Low thresholds could result in very little simplification.

Postpruning

The more common approach is postpruning, which removes subtrees from a "fully grown" tree.

A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.

For example, notice the subtree at node "A3?" in the unpruned tree of previous figure. Suppose that the most common class within this subtree is class B. In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf "class B".

Other Pruning:

- Cost complexity - used by CART,
- Pessimistic pruning - used by C4.5

Bayesian classifiers

What are Bayesian classifiers?" Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class.

Studies comparing classification algorithms have found a simple Bayesian classifier known as the naive Bayesian classifier to be comparable in performance with decision tree and selected neural network classifiers.

Bayesian classifiers (cont.)

Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence.

It is made to simplify the computations involved and, in this sense, is considered "naïve."

Bayes' Theorem

Let X be a data tuple. In Bayesian terms, X is considered **"evidence**. As usual, it is described by measurements made on a set of n attributes.

Let H be some hypothesis, such as that the data tuple X belongs to a specified class C.

For classification problems, we want to determine $P(H|X)$, **the probability that the hypothesis H holds given the "evidence"** or observed data tuple X.
In other words, we are looking for the probability that tuple X belongs to class C, given that we know the attribute description of X.

Bayes' Theorem (cont.)

Posterior probability

$P(H|X)$ is a posteriori probability of H conditioned on X.

For example, suppose our world of data tuples is confined to customers described by the attributes *age* and *income*, respectively, and that X is a 35-year-old customer with an income of $40,000.

Suppose that H is the hypothesis that our customer will buy a computer. Then $P(H|X)$ reflects the probability that customer X will buy a computer given that we know the customer's age and income.

Bayes' Theorem (cont.)

Prior probability

$P(H)$ is the prior probability, or a priori probability of H.
For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter.

The posterior probability, $P(H|X)$, is based on more information (e.g., customer information) than the prior probability, $P(H)$, which is **independent of X**.

Similarly, $P(X|H)$ is the posterior probability of X conditioned on H. That is, it is the probability that a customer, X, is 35 years old and earns $40,000, given that we know the customer will buy a computer.

Bayes' Theorem (cont.)

How are these probabilities estimated

$P(H)$, $P(X|H)$, and $P(X)$ may be estimated from the given data.

Bayes' theorem is useful in that it provides a way of calculating the posterior probability, $P(H|X)$, from $P(H)$, $P(X|H)$, and $P(X)$.

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

Naïve Bayesian Classification

Features

Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, $X = (x_1, x_2, ..., x_n)$, depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, ..., A_n$.

Classes

Suppose that there are m classes, $C_1, C_2, ..., C_m$

Naïve Bayesian Classification (cont.)

Given a tuple, X, the **classifier will predict that X belongs to the class having the highest posterior probability**, conditioned on X.

That is, the naïve Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

$$P(C_i|X) > P(C_j|X) \qquad for 1 \leq j \leq m, j \neq i$$

Thus we maximize $P(C_i|X)$.

The class C_i for which $P(C_i|X)$ is maximized is called the maximum posteriori hypothesis. By Bayes' theorem,

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

Naïve Bayesian Classification (cont.)

As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ need be maximized.

If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is,
$P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize $P(X|C_i)$.
Otherwise, we maximize $P(X|C_i)P(C_i)$.

Note that the class prior probabilities may be estimated by

$$P(C_i) = \frac{|C_{i,D}|}{|D|}$$

where $|C_{i,D}|$ is the number of training tuples of class C_i in D

Naïve Bayesian Classification - naive assumption

Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$. In order to reduce computation in evaluating $P(X|C_i)$, the **naive assumption of class conditional independence** is made.

This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

$$\begin{aligned} P(X|C_i) &= \prod_{k=1}^{n} P(x_k|C_i) \\ &= P(x_1|C_i) \times P(x_2|C_i) \cdots \times P(x_n|C_i) \end{aligned}$$

We can easily estimate the probabilities $P(x_1|C_i), P(x_2|C_i), ..., P(x_n|C_i)$ from the training tuples. x_k refers to the value of attribute A_k for tuple X.

Naïve Bayesian Classification - data types

For each attribute, we look at whether the attribute is **categorical** or **continuous-valued**.
For instance, to compute $P(X|C_i)$, we consider the following:
If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k , divided by $|C_{i,D}|$ - the number of tuples of class C_i in D.
If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward.
A continuous-valued attribute is typically **assumed to have a Gaussian distribution** with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

So that for classification,

$$P(x_k|C_i) = g(x_k, \mu_{Ci}, \sigma_{Ci})$$

Naïve Bayesian Classification (cont.)

A continuous-valued attribute is typically **assumed to have a Gaussian distribution** with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

So that,

$$P(x_k|C_i) = g(x_k, \mu_{Ci}, \sigma_{Ci})$$

We need to compute μ_{Ci} and σ_{Ci} , which are the mean (i.e., average) and standard deviation respectively, of the values of attribute A_k for training tuples of class C_i

Naïve Bayesian Classification (cont.)

In order to predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i .

The classifier predicts that the class label of tuple X is the class C_i if and only if

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \qquad for 1 \le j \le m, j \ne i$$

In other words, the predicted class label is the class C_i for which $P(X|C_i)P(C_i)$ is the maximum.

Read yourself
Book: Example 6.4 (Page No: 313)