**IV Year – II Semester**

| L | T | P | C |
|---|---|---|---|
| 4 | 0 | 0 | 3 |

# MACHINE LEARNING

**OBJECTIVES:**

- Familiarity with a set of well-known supervised, unsupervised and semi-supervised
- learning algorithms.
- The ability to implement some basic machine learning algorithms
- Understanding of how machine learning algorithms are evaluated

**UNIT -I:The ingredients of machine learning, Tasks:** the problems that can be solved with machine learning, Models: the output of machine learning, Features, the workhorses of machine learning. **Binary classification and related tasks:** Classification, Scoring and ranking, Class probability estimation

**UNIT- II:Beyond binary classification:**Handling more than two classes, Regression, Unsupervised and descriptive learning. **Concept learning**: The hypothesis space, Paths through the hypothesis space, Beyond conjunctive concepts

**UNIT- III: Tree models:** Decision trees, Ranking and probability estimation trees, Tree learning as variance reduction. **Rule models:**Learning ordered rule lists, Learning unordered rule sets, Descriptive rule learning, First-order rule learning

**UNIT -IV:Linear models:** The least-squares method, The perceptron: a heuristic learning algorithm for linear classifiers, Support vector machines, obtaining probabilities from linear classifiers, Going beyond linearity with kernel methods.**Distance Based Models:** Introduction, Neighbours and exemplars, Nearest Neighbours classification, Distance Based Clustering, Hierarchical Clustering.

**UNIT- V:Probabilistic models:** The normal distribution and its geometric interpretations, Probabilistic models for categorical data, Discriminative learning by optimising conditional likelihoodProbabilistic models with hidden variables.**Features:** Kinds of feature, Feature transformations, Feature construction and selection. Model ensembles: Bagging and random forests, Boosting

**UNIT- VI: Dimensionality Reduction:** Principal Component Analysis (PCA), Implementation and demonstration. **Artificial Neural Networks:**Introduction, Neural network representation, appropriate problems for neural network learning, Multilayer networks and the back propagation algorithm.

**OUTCOMES:**

- Recognize the characteristics of machine learning that make it useful to real-world
- Problems.
- Characterize machine learning algorithms as supervised, semi-supervised, and
- Unsupervised.
- Have heard of a few machine learning toolboxes.
- Be able to use support vector machines.
- Be able to use regularized regression algorithms.
- Understand the concept behind neural networks for learning non-linear functions.

**TEXT BOOKS:**

1. Machine Learning: The art and science of algorithms that make sense of data, Peter Flach, Cambridge.
2. Machine Learning, Tom M. Mitchell, MGH.

**REFERENCE BOOKS:**

1. UnderstandingMachine Learning: From Theory toAlgorithms, Shai Shalev-Shwartz, Shai Ben-
David, Cambridge.
2. Machine Learning in Action, Peter Harington, 2012, Cengage.

**MACHINE LEARNING**

**UNIT -I:The ingredients of machine learning, Tasks:** the problems that can be solved with Machine learning, Models: the output of machine learning, Features, the workhorses of machine learning.
**Binary classification and related tasks:** Classification, Scoring and ranking, Class probability estimation

# A) Machine Learning

**Machine Learning** is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that which makes it more similar to humans: ***The ability to learn***. Machine learning is actively being used today, perhaps in many more places than one would expect

## I- THE PROBLEMS THAT CAN BE SOLVED WITH MACHINE LEARNING:

The most common machine learning tasks are predictive, in the sense that they concern predicting a target variable from features.
**Binary and multi-class classification**: categorical target
**Regression:** numerical target
**Clustering:** hidden target
Descriptive tasks are concerned with exploiting underlying structure in the data

8 PROBLEMS SOLVED BY MACHINE LEARNING

- Manual data entry.
- Detecting Spam.
- Product recommendation.
- Medical Diagnosis.
- Customer segmentation and Lifetime value prediction.
- Financial analysis.
- Predictive maintenance.
- Image recognition (Computer Vision).

### 1. MANUAL DATA ENTRY

Inaccuracy and duplication of data are major business problems for an organization wanting to automate its processes. Machines learning (ML) algorithms and predictive modelling algorithms can significantly improve the situation. ML programs use the discovered data to improve the process as more calculations are made. Thus machines can learn to perform time-intensive documentation and data entry tasks. Also, knowledge workers can now spend more time on higher-value problem-solving tasks. Arria, an AI based firm has developed a natural language processing technology which scans texts and determines the relationship between concepts to write reports.

## 2. DETECTING SPAM

Spam detection is the earliest problem solved by ML. Four years ago, email service providers used pre-existing rule-based techniques to remove spam. But now the spam filters create new rules themselves using ML. Thanks to 'neural networks' in its spam filters, Google now boasts of 0.1 percent of spam rate. Brain-like "neural networks" in its spam filters can learn to recognize junk mail and phishing messages by analyzing rules across an enormous collection of computers. In addition to spam detection, social media websites are using ML as a way to identify and filter abuse.

## 3. PRODUCT RECOMMENDATION

Unsupervised learning enables a product based recommendation system. Given a purchase history for a customer and a large inventory of products, ML models can identify those products in which that customer will be interested and likely to purchase. The algorithm identifies hidden pattern among items and focuses on grouping similar products into clusters. A model of this decision process would allow a program to make recommendations to a customer and motivate product purchases. E-Commerce businesses such as Amazon has this capability. Unsupervised learning along with location detail is used by Facebook to recommend users to connect with others users.

## 4. MEDICAL DIAGNOSIS

Machine Learning in the medical field will improve patient's health with minimum costs. Use cases of ML are making near perfect diagnoses, recommend best medicines, predict readmissions and identify high-risk patients. These predictions are based on the dataset of anonymized patient records and symptoms exhibited by a patient. Adoption of ML is happening at a rapid pace despite many hurdles, which can be overcome by practitioners and consultants who know the legal, technical, and medical obstacles.

## 5. CUSTOMER SEGMENTATION AND LIFETIME VALUE PREDICTION

Customer segmentation, churn prediction and customer lifetime value (LTV) prediction are the main challenges faced by any marketer. Businesses have a huge amount of marketing relevant data from various sources such as email campaign, website visitors and lead data. Using data mining and machine learning, an accurate prediction for individual marketing offers and incentives can be achieved. Using ML, savvy marketers can eliminate guesswork involved in data-driven marketing. For example, given the pattern of behavior by a user during a trial period and the past behaviors of all users, identifying chances of conversion to paid version can be predicted. A model of this decision problem would allow a program to trigger customer interventions to persuade the customer to convert early or better engage in the trial.

## 6. FINANCIAL ANALYSIS

Due to large volume of data, quantitative nature and accurate historical data, machine learning can be used in financial analysis. Present use cases of ML in finance includes algorithmic trading, portfolio management, fraud detection and loan underwriting. According to Ernst and Young report on 'The future of underwriting' – Machine learning will enable continual assessments of data for detection and analysis of anomalies and nuances to improve

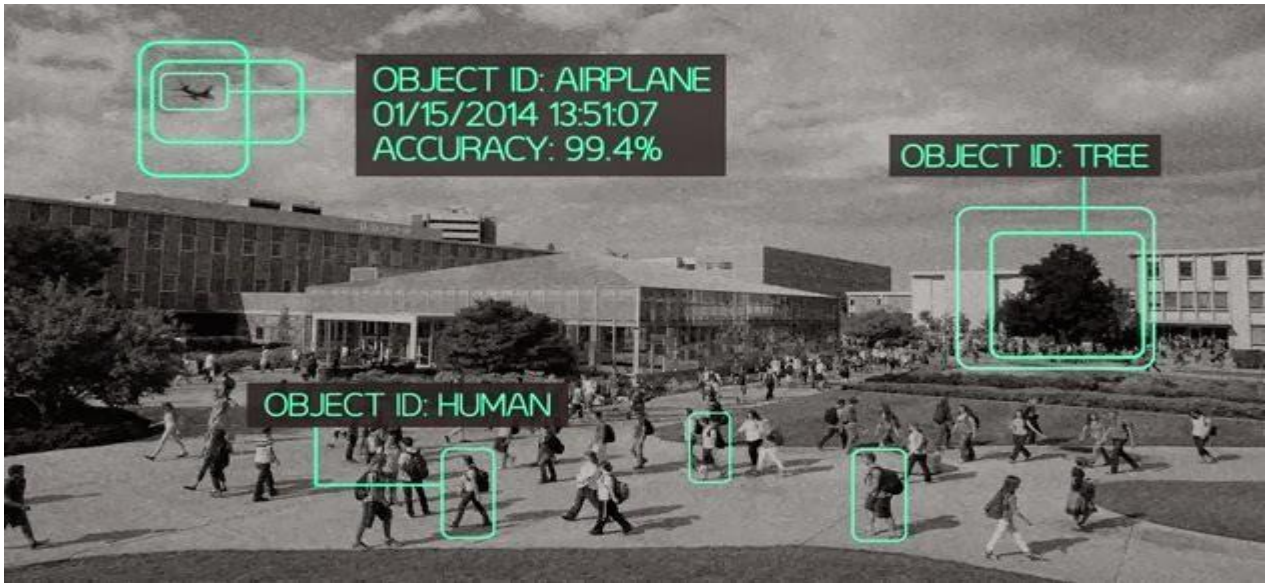
the precision of models and rules. And machines will replace a large no. of underwriting positions. Future applications of ML in finance include_chatbots and conversational interfaces for customer service, security and sentiment analysis.

## 7. PREDICTIVE MAINTENANCE

Manufacturing industry can use artificial intelligence (AI) and ML to discover meaningful patterns in factory data. Corrective and preventive maintenance practices are costly and inefficient. Whereas predictive maintenance minimizes the risk of unexpected failures and reduces the amount of unnecessary preventive maintenance activities.


## 8. IMAGE RECOGNITION (COMPUTER VISION)

Computer vision produces numerical or symbolic information from images and high-dimensional data. It involves machine learning, data mining, database knowledge discovery and pattern recognition. Potential business uses of image recognition technology are found in healthcare, automobiles – driverless cars, marketing campaigns, etc. Baidu has developed a prototype of DuLight for visually impaired which incorporates computer vision technology to capture surrounding and narrate the interpretation through an earpiece. Image recognition based marketing campaigns such as Makeup Genius by L'Oreal drive social sharing and user engagement.

- 

## II-MODELS: THE OUTPUT OF MACHINE LEARNING, FEATURES, THE WORKHORSES OF MACHINE LEARNING.

## MODELS: THE OUTPUT OF MACHINE LEARNING

1.Geometric models

2.Probabilistic models

3.Logical models Grouping and grading

Machine learning models can be distinguished according to their main intuition:

**Geometric models** :use intuitions from geometry such as separating (hyper-)planes, linear transformations and distance metrics.

**Probabilistic models**: view learning as a process of reducing uncertainty, modelled by means of probability distributions.

**Logical models** are defined in terms of easily interpretable logical expressions.

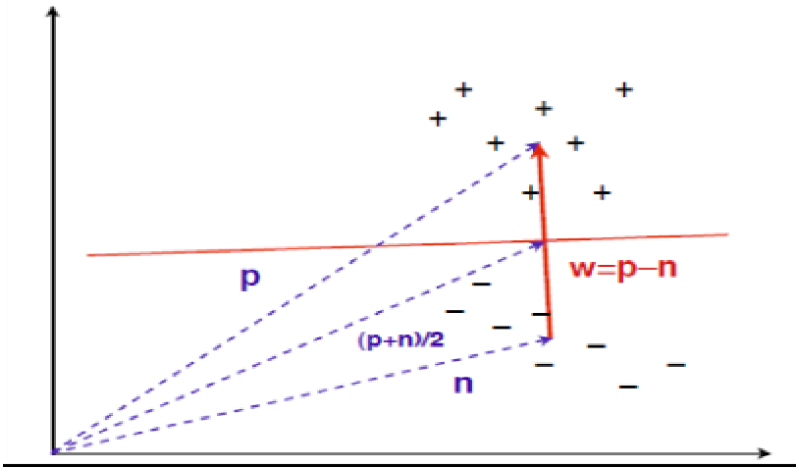
Alternatively, they can be characterised by their modus operandi:

**Grouping models** divide the instance space into segments; in each segment a very simple (e.g., constant) model is learned.

**Grading models** learning a single, global model over the instance space

### 1.Geometric models

Basic linear classifier



The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass. It is described by the equation $\mathbf{w} \cdot \mathbf{x} = t$, with $\mathbf{w} = \mathbf{p} - \mathbf{n}$; the decision threshold can be found by noting that (**p**+**n**)/2 is on the decision boundary, and $t = (\mathbf{p} - \mathbf{n}) \cdot (\mathbf{p} + \mathbf{n})/2 = (\|\mathbf{p}\|^2 - \|\mathbf{n}\|^2)/2$, where ||**x**|| denotes the length of vector **x**.

**2.Probabilistic models**

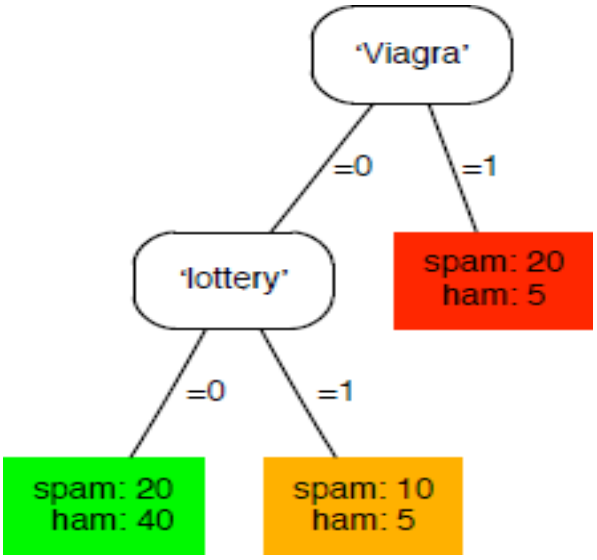
- A model describes data that one could observe from a system
- If we use the mathematics of probability theory to express all forms of uncertainty and Noise associated with our model...
- Then inverse probability (i.e. Bayes rule) allows us to infer unknown quantities, adapt our models, Make predictions and learn from data

Bayes Rule:

$$P(\text{hypothesis}|\text{data}) = \frac{P(\text{data}|\text{hypothesis})P(\text{hypothesis})}{P(\text{data})}$$

- Bayes rule tells us how to do inference about hypotheses from data.
- Learning and prediction can be seen as forms of inference.

**3.Logical models**



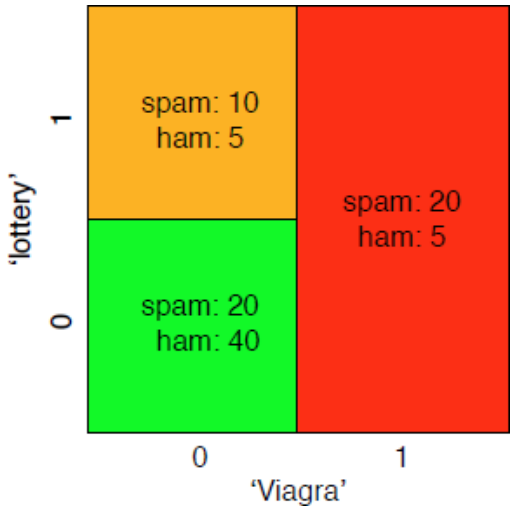



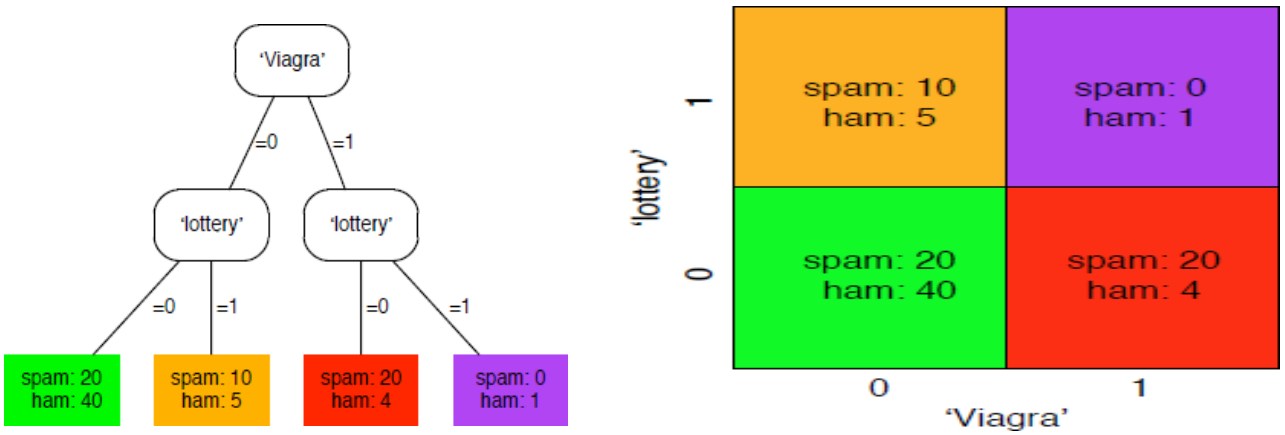
Fig:1:4-Feature Tree for Logical models

## Labelling a feature tree

(left) A feature tree combining two Boolean features. Each internal node or split is labelled with a feature, and each edge emanating from a split is labelled with a feature value. Each leaf therefore corresponds to a unique combination of feature values. Also indicated in each leaf is the class distribution derived from the training set. (right) A feature tree partitions the instance space into rectangular regions, one for each leaf. We can clearly see that the majority of ham lives in the lower left-hand corner.

- The leaves of the tree in Figure 1.4 could be labelled, from left to right, as ham – spam – spam, employing a simple decision rule called majority class.
- Alternatively, we could label them with the proportion of spam e-mail occurring in each leaf: from left to right, 1/3, 2/3, and 4/5.
- Or, if our task was a regression task, we could label the leaves with predicted real values or even linear functions of some other, real-valued features.

## A complete feature tree




Consider the following rules:

- **if** lottery = 1 **then** Class = Y = spam
- **if** Peter = 1 **then** Class = Y = ham

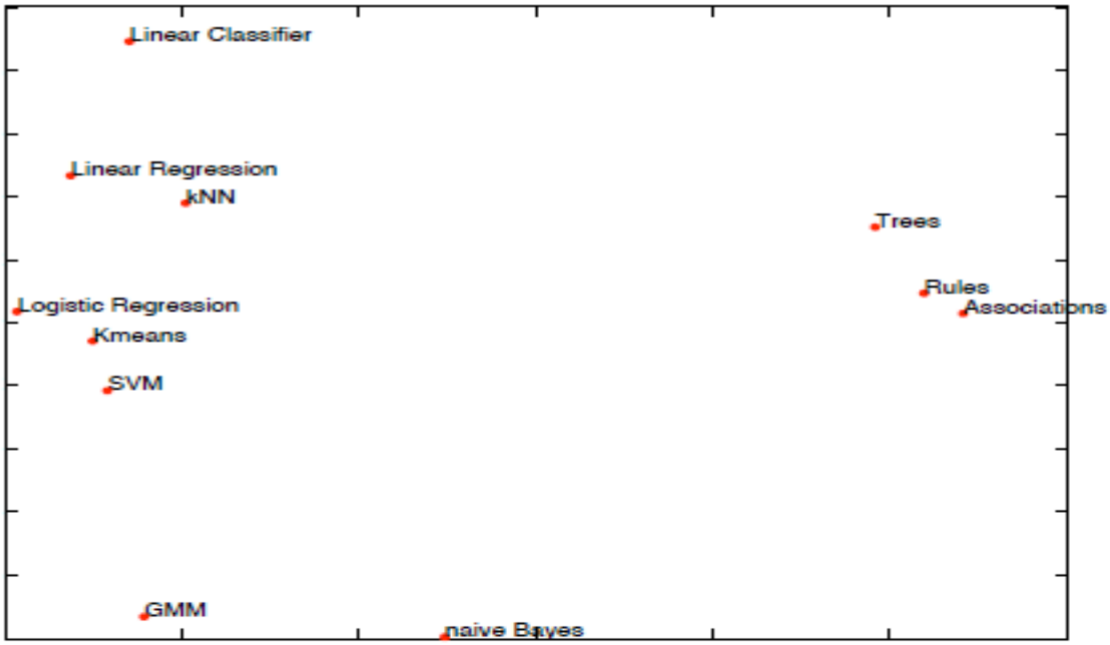
**4.Grouping and Grading models**

## Mapping machine learning models:

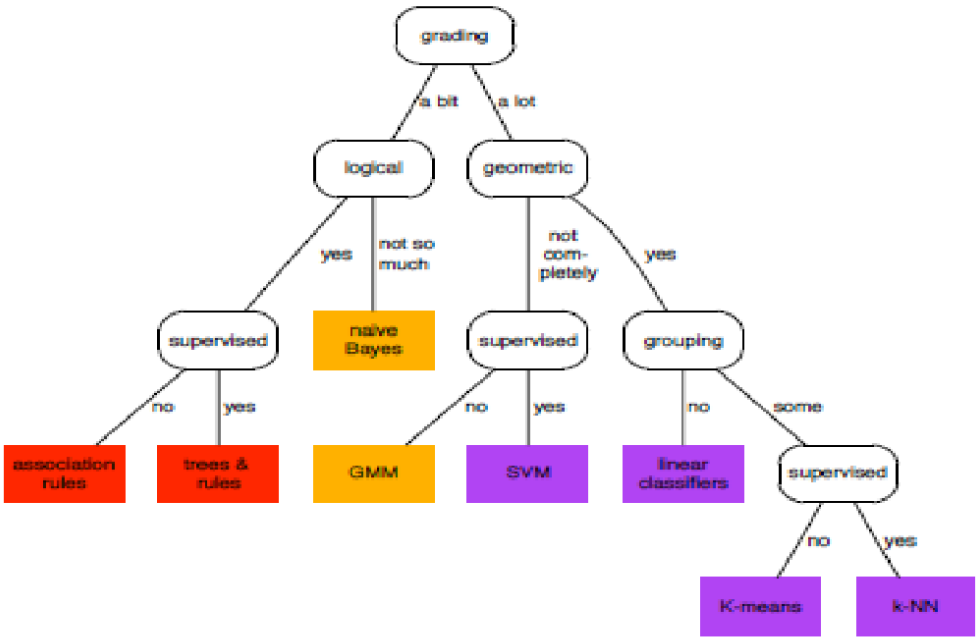
A 'map' of some of the models that will be considered in this book. Models that share

characteristics are plotted closer together: logical models to the right, geometric models on the top left and probabilistic models on the bottom left. The horizontal dimensiroughly ranges from grading models on the left to grouping models on the right.

## Mapping machine learning model diagram




## ML taxonomy diagram



A taxonomy describing machine learning methods in terms of the extent to which they are grading or grouping models, logical, geometric or a combination, and supervised or unsupervised. The colours indicate the type of model, from left to right: logical (red), probabilistic (orange) and geometric (purple).

## III- FEATURES :-THE WORKHORSES OF MACHINE LEARNING

Suppose we have a number of learning models that we want to describe in terms of a number of properties:

→The extent to which the models are geometric, probabilistic or logical;

→Whether they are grouping or grading models;

→The extent to which they can handle discrete and/or real-valued features;

→Whether they are used in supervised or unsupervised learning; and

→The extent to which they can handle multi-class problems.

The first two properties could be expressed by discrete features with three and two values, respectively; or if the distinctions are more gradual, each aspect could be rated on some numerical scale. A simple approach would be to measure each property on an integer scale from 0 to 3, as in Table 1.4. This table establishes a data set in which each row represents an instance and each column a feature

Table 1.4. **THE MLM DATA SET**

| Model | geom | stats | logic | group | grad | disc | real | sup | unsup | multi |
|---|---|---|---|---|---|---|---|---|---|---|
| Trees | 1 | 0 | 3 | 3 | 0 | 3 | 2 | 3 | 2 | 3 |
| Rules | 0 | 0 | 3 | 3 | 1 | 3 | 2 | 3 | 0 | 2 |
| naive Bayes | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 0 | 3 |
| kNN | 3 | 1 | 0 | 2 | 2 | 1 | 3 | 3 | 0 | 3 |
| Linear Classifier | 3 | 0 | 0 | 0 | 3 | 1 | 3 | 3 | 0 | 0 |
| Linear Regression | 3 | 1 | 0 | 0 | 3 | 0 | 3 | 3 | 0 | 1 |
| Logistic Regression | 3 | 2 | 0 | 0 | 3 | 1 | 3 | 3 | 0 | 0 |
| SVM | 2 | 2 | 0 | 0 | 3 | 2 | 3 | 3 | 0 | 0 |
| Kmeans | 3 | 2 | 0 | 1 | 2 | 1 | 3 | 0 | 3 | 1 |
| GMM | 1 | 3 | 0 | 0 | 3 | 1 | 3 | 0 | 3 | 1 |
| Associations | 0 | 0 | 3 | 3 | 0 | 3 | 1 | 0 | 3 | 1 |

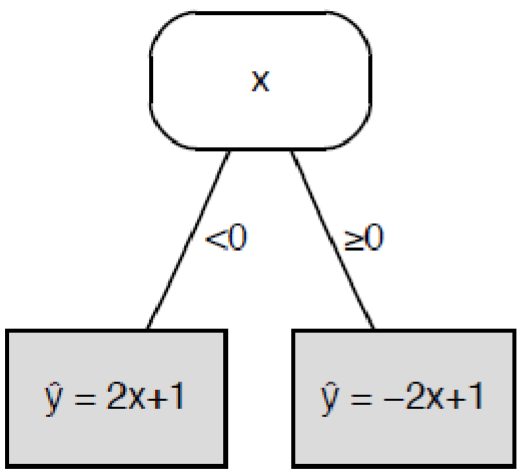
**THE MANY USES OF FEATURES:**

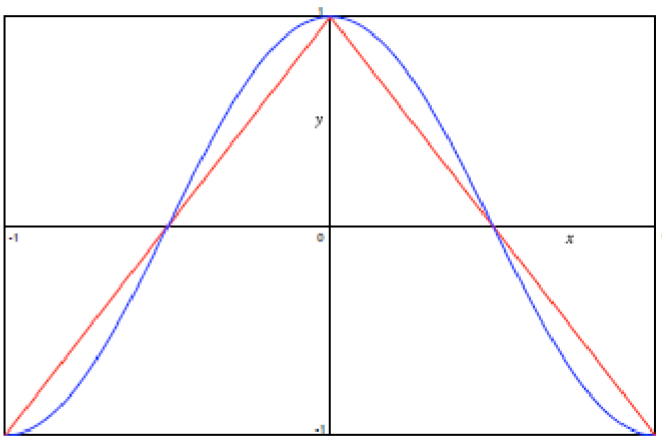
Suppose we want to approximate $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$. A linear approximation is not much use here, since the best fit would be $y = 0$. However, if we split the $x$-axis in two intervals $-1 \leq x < 0$ and $0 \leq x \leq 1$, we could find reasonable linear approximations on each interval. We can achieve this by using $x$ both as a *splitting feature* and as a *regression variable* (Figure 1.9).

---

## A small regression tree

**FIG:DIAGRAM FORREGRESSION TREE**



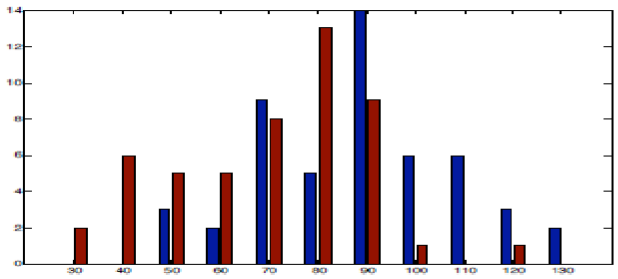

---

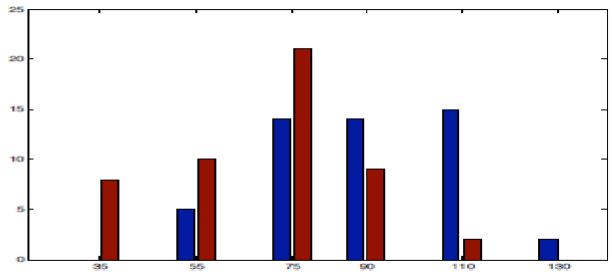
**(left)** A regression tree combining a one-split feature tree with linear regression models in the leaves. Notice how *x* is used as both a splitting feature and a regression variable.
**(right)** The function *y=COSπx* on the interval -1≤X≤+1, and the piecewise linear approximation achieved by the regression tree.

**Feature construction and transformation**

### Class-sensitive discretisation:

**(left)** Artificial data depicting a histogram of body weight measurements of people with (blue) and without (red) diabetes, with eleven fixed intervals of 10 kilograms width each. **(right)** By joining the first and second, third and fourth, fifth and sixth, and the eighth,ninth and tenth intervals, we obtain a discretisation such that the proportion of diabetes cases increases from left to right. This discretisation makes the feature more useful in predicting diabetes.




# B) BINARY CLASSIFICATION AND RELATED TASKS

Binary classification and related tasks are

- **Classification**
  -Assessing classification performance
  -Visualising classification performance

- **Scoring and ranking**
  -Assessing and visualising ranking performance
  -Tuning rankers
- **Class probability estimation**
  -Assessing class probability estimates

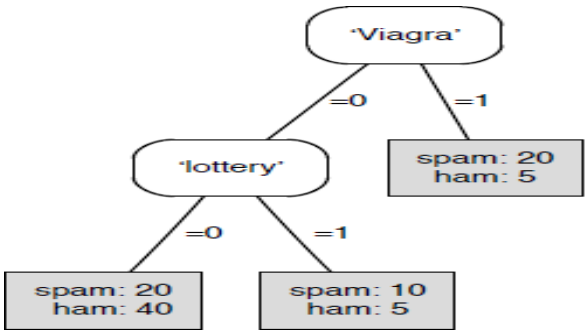
## 1) CLASSIFICATION

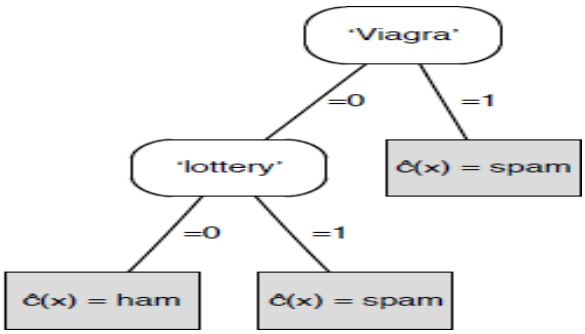
A *classifier* is a mapping $\hat{c} : \mathscr{X} \rightarrow \mathscr{C}$, where $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ is a finite and usually small set of *class labels*. We will sometimes also use $C_i$ to indicate the set of examples of that class.

We use the 'hat' to indicate that $\hat{c}(x)$ is an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathscr{X}$ is an instance and $c(x)$ is the true class of the instance (sometimes contaminated by noise).

Learning a classifier involves constructing the function $\hat{c}$ such that it matches $c$ as closely as possible (and not just on the training set, but ideally on the entire instance space $\mathscr{X}$).

**Decision Tree Diagram For Classification:**




**(left)** A feature tree with training set class distribution in the leaves. **(right)** A decision tree obtained using the majority class decision rule.

## →Assessing classification performance

**(left)** A two-class contingency table or confusion matrix depicting the performance of the decision tree in Figure 2.1. Numbers on the descending diagonal indicate correct predictions, while the ascending diagonal concerns prediction errors. **(right)** Acontingency table with the same marginals but independent rows and columns.

### Contingency table

| | *Predicted* ⊕ | *Predicted* ⊖ | |
|---|---|---|---|
| *Actual* ⊕ | **30** | **20** | 50 |
| *Actual* ⊖ | **10** | **40** | 50 |
| | 40 | 60 | 100 |

| | ⊕ | ⊖ | |
|---|---|---|---|
| ⊕ | **20** | **30** | 50 |
| ⊖ | **20** | **30** | 50 |
| | 40 | 60 | 100 |

### →Visualising classification performance:

### Degrees of freedom for above topic

contains 9 values, however some of them depend on others: e.g., marginal sums depend on rows and columns, respectively. Actually, we need only 4 values to determine the rest of them. Thus, we say that this table has **4 degrees of freedom**. In general table having (*k* Å1)2 entries has *k*2 degrees of freedom.In the following, we assume that *Pos*, *Neg* , **TP** and **FP** are enough to reconstruct whole table.

The following contingency table:

| | *Predicted* ⊕ | *Predicted* ⊖ | |
|---|---|---|---|
| *Actual* ⊕ | TP | FN | $Pos$ |
| *Actual* ⊖ | FP | TN | $Neg$ |
| | 0 | 0 | 0 |

**2) SCORING AND RANKING**

A *scoring classifier* is a mapping $\hat{\mathbf{s}} : \mathcal{X} \rightarrow \mathbb{R}^k$, i.e., a mapping from the instance space to a $k$-vector of real numbers.

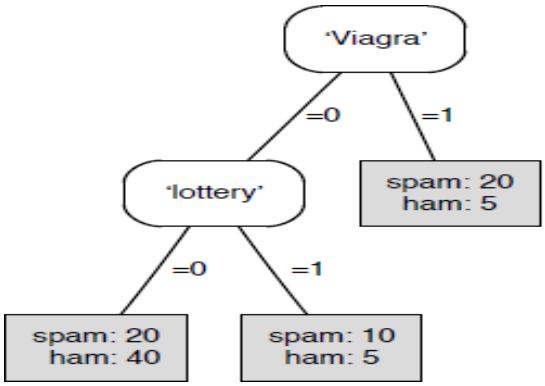
The boldface notation indicates that a scoring classifier outputs a vector $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class $C_i$ for instance $x$.

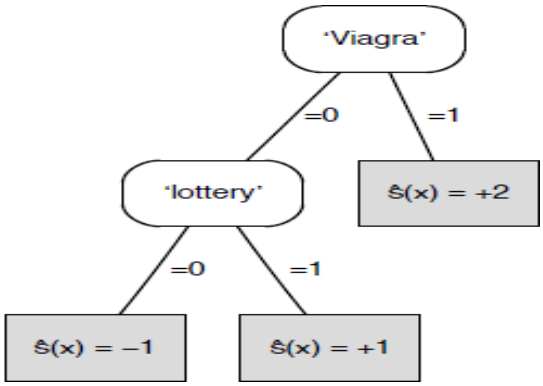
This score indicates how likely it is that class label $C_i$ applies.

If we only have two classes, it usually suffices to consider the score for only one of the classes; in that case, we use $\hat{s}(x)$ to denote the score of the positive class for instance $x$.

---

**SCORING TREE**



**(left)** A feature tree with training set class distribution in the leaves.
**(right)** A scoring tree using the logarithm of the class ratio as scores; spam is taken as the positive class.

**→ Assessing and visualising ranking performance:**

☞ By selecting a split point in the ranking we can turn the ranking into a classification. In this case there are four possibilities:

(A) setting the split point before the first segment, and thus assigning all segments to the negative class;
(B) assigning the first segment to the positive class, and the other two to the negative class;
(C) assigning the first two segments to the positive class; and
(D) assigning all segments to the positive class.

The *ranking error rate* is defined as

$$rank\text{-}err = \frac{\sum_{x \in Te^{+}, x' \in Te^{-}} I[\hat{s}(x) < \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{Pos \cdot Neg}$$

**→Tuning rankers:**

You have carefully trained your Bayesian spam filter, and all that remains is setting the decision threshold. You select a set of six spam and four ham e-mails and collect the scores assigned by the spam filter. Sorted on decreasing score these are 0.89 (spam), 0.80 (spam), 0.74 (ham), 0.71 (spam), 0.63 (spam), 0.49 (ham), 0.42 (spam), 0.32 (spam), 0.24 (ham), and 0.13 (ham).

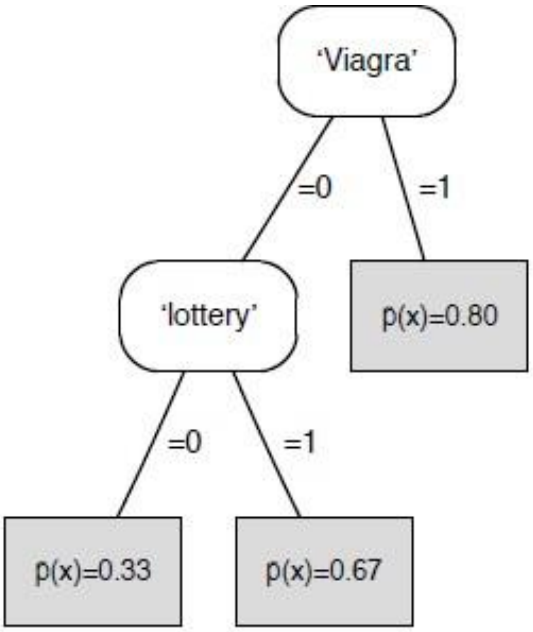
**3.CLASS PROBABILITY ESTIMATION**

A *class probability estimator* – or probability estimator in short – is a scoring classifier that outputs probability vectors over classes, i.e., a mapping $\hat{\mathbf{p}} : \mathcal{X} \rightarrow [0,1]^k$. We write $\hat{\mathbf{p}}(x) = (\hat{p}_1(x), \ldots, \hat{p}_k(x))$, where $\hat{p}_i(x)$ is the probability assigned to class $C_i$ for instance $x$, and $\sum_{i=1}^{k} \hat{p}_i(x) = 1$.

If we have only two classes, the probability associated with one class is 1 minus the probability of the other class; in that case, we use $\hat{p}(x)$ to denote the estimated probability of the positive class for instance $x$.

As with scoring classifiers, we usually do not have direct access to the true probabilities $p_i(x)$.

PROBABILTIY ESTIMATION TREE:




A probability estimation tree derived from the feature tree in Figure 1.4.

**→Assessing class probability estimates:**

**It requires mean and squared probability form is**

We can define the *squared error* (*SE*) of the predicted probability vector $\hat{\mathbf{p}}(x) = \left(\hat{p}_1(x), \ldots, \hat{p}_k(x)\right)$ as

$$\mathrm{SE}(x) = \frac{1}{2}\sum_{i=1}^{k}\left(\hat{p}_i(x) - I[c(x) = C_i]\right)^2$$

and the *mean squared error* (*MSE*) as the average squared error over all instances in the test set:

$$\mathrm{MSE}(Te) = \frac{1}{|Te|}\sum_{x \in Te}\mathrm{SE}(x)$$

# UNIT- II

## Chapter-3

## Beyond binary classification

### Handling more than two classes

How to evaluate multi-class performance and how to build multi-class models out of binary models.

→Multi-class classification

→Multi-class scores and probabilities

Multi-class classification: **multiclass** or **multinomial classification** is the problem of classifying instances into one of three or more classes. (Classifying instances into one of two classes is called binary classification.)

The existing multi-class classification techniques can be categorized into

(i) Transformation to binary

(ii) Extension from binary(Multi-class scores and probabilities)

(iii) Hierarchical classification.

### 1. Transformation to binary

This section discusses strategies for reducing the problem of multiclass classification to multiple binary classification problems. It can be categorized into One vs Rest and One vs One. The techniques developed based on reducing the multi-class problem into multiple binary problems can also be called problem transformation techniques.

**One-vs.-rest**

One-vs.-rest (or *one-vs.-all*, OvA or OvR, *one-against-all*, OAA) strategy involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample.

In pseudocode, the training algorithm for an OvA learner constructed from a binary classification learner $L$ is as follows:

Inputs:

- $L$, a learner (training algorithm for binary classifiers)
- samples $X$
- labels $y$ where $y_i \in \{1, \dots K\}$ is the label for the sample $X_i$

Output:

- a list of classifiers $f_k$ for $k \in \{1, \dots, K\}$

Procedure:

- For each $k$ in $\{1, \dots, K\}$
  - Construct a new label vector $z$ where $z_i = y_i$ if $y_i = k$ and $z_i = 0$ otherwise

- Apply $L$ to $X$, $z$ to obtain $f_k$

Making decisions means applying all classifiers to an unseen sample $x$ and predicting the label $k$ for which the corresponding classifier reports the highest confidence score:

> Although this strategy is popular, it is a heuristic that suffers from several problems. Firstly, the scale of the confidence values may differ between the binary classifiers. Second, even if the class distribution is balanced in the training set, the binary classification learners see unbalanced distributions because typically the set of negatives they see is much larger than the set of positives.

**One-vs.-one**

In the *one-vs.-one* (OvO) reduction, one trains $K\ (K-1)\ /\ 2$ binary classifiers for a $K$-way multiclass problem; each receives the samples of a pair of classes from the original training set, and must learn to distinguish these two classes. At prediction time, a voting scheme is applied: all $K\ (K-1)\ /\ 2$ classifiers are applied to an unseen sample and the class that got the highest number of "+1" predictions gets predicted by the combined classifier.

Like OvR, OvO suffers from ambiguities in that some regions of its input space may receive the same number of votes.

## 2. Multi-class scores and probabilities

**Extension from binary** This section discusses strategies of extending the existing binary classifiers to solve multi-class classification problems. Several algorithms have been developed based on neural networks, decision trees, k-nearest neighbors, naive Bayes, support vector machines and Extreme Learning Machines to address multi-class classification problems. These types of techniques can also be called algorithm adaptation techniques.

**Neural networks**

Multiclass perceptrons provide a natural extension to the multi-class problem. Instead of just having one neuron in the output layer, with binary output, one could have N binary neurons leading to multi-class classification. In practice, the last layer of a neural network is usually a softmax function layer, which is the algebraic simplification of N logistic classifiers, normalized per class by the sum of the N-1 other logistic classifiers.

Extreme learning machines

Extreme Learning Machines (ELM) is a special case of single hidden layer feed-forward neural networks (SLFNs) where in the input weights and the hidden node biases can be chosen at random. Many variants and developments are made to the ELM for multiclass classification.

**k-nearest neighbours**

k-nearest neighbors kNN is considered among the oldest non-parametric classification algorithms. To classify an unknown example, the distance from that example to every other

training example is measured. The k smallest distances are identified, and the most represented class by these k nearest neighbours is considered the output class label.

**Naive Bayes**

Naive Bayes is a successful classifier based upon the principle of maximum a posteriori (MAP). This approach is naturally extensible to the case of having more than two classes, and was shown to perform well in spite of the underlying simplifying assumption of conditional independence.

**Decision trees**

Decision tree learning is a powerful classification technique. The tree tries to infer a split of the training data based on the values of the available features to produce a good generalization. The algorithm can naturally handle binary or multiclass classification problems. The leaf nodes can refer to either of the K classes concerned.

**Support vector machines**

Support vector machines are based upon the idea of maximizing the margin i.e. maximizing the minimum distance from the separating hyperplane to the nearest example. The basic SVM supports only binary classification, but extensions have been proposed to handle the multiclass classification case as well. In these extensions, additional parameters and constraints are added to the optimization problem to handle the separation of the different classes.

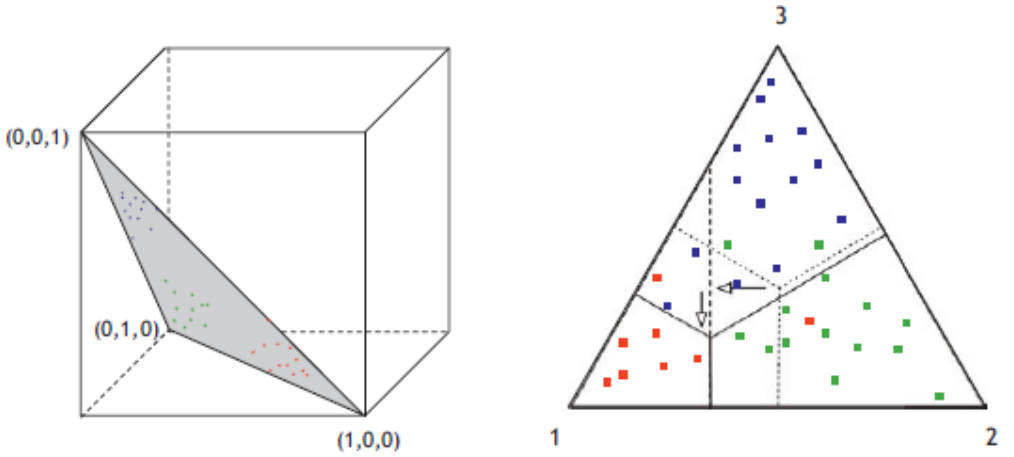



**Figure 3.1.** **(left)** Triples of probabilistic scores represented as points in an equilateral triangle connecting three corners of the unit cube. **(right)** The arrows show how the weights are adjusted from the initial equal weights (dotted lines), first by optimising the separation of $C_2$ against $C_1$ (dashed line), then by optimising the separation of $C_3$ against the other two classes (solid lines). The end result is that the weight of $C_1$ is considerably decreased, to the benefit of the other two classes.

$w_1 = 1$. Unfortunately, finding a globally optimal weight vector is computationally intractable. A heuristic approach that works well in practice is to first learn $w_2$ to optimally separate $C_2$ from $C_1$ as in the two-class case; then learn $w_3$ to separate $C_3$ from $C_1 \cup C_2$, and so on.

## 3.Hierarchical classification

Hierarchical classification tackles the multi-class classification problem by dividing the output space i.e. into a tree. Each parent node is divided into multiple child nodes and the process is continued until each child node represents only one class. Several methods have been proposed based on hierarchical classification.

**Regression**

A *function estimator*, also called a *regressor*, is a mapping $\hat{} f : X \rightarrow$R. The regression learning problem is to learn a function estimator from examples $(xi , f (xi ))$

Regression models are used to predict a continuous value. Predicting prices of a house given the features of house like size, price etc is one of the common examples of Regression. It is a supervised technique.

**Types of Regression**

1. Simple Linear Regression
2. Polynomial Regression
3. Support Vector Regression
4. Decision Tree Regression
5. Random Forest Regression

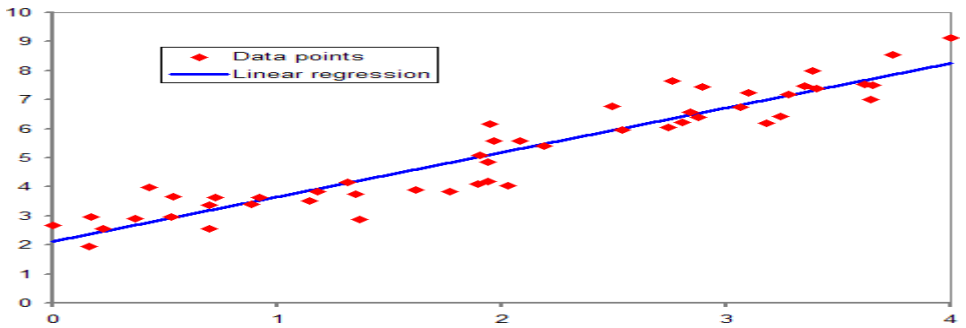
**1.Simple Linear Regression**

This is one of the most common and interesting type of Regression technique. Here we predict a target variable Y based on the input variable X. A linear relationship should exist between target variable and predictor and so comes the name Linear Regression.

Consider predicting the salary of an employee based on his/her age. We can easily identify that there seems to be a correlation between employee's age and salary (more the age more is the salary). The hypothesis of linear regression is

$$Y = a + bX$$

Y represents salary, X is employee's age and a and b are the coefficients of equation. So in order to predict Y (salary) given X (age), we need to know the values of a and b (the model's coefficients).




*While training and building a regression model, it is these coefficients which are learned and fitted to training data. The aim of training is to find a best fit line such that cost function is minimized. The cost function helps in measuring the error. During training process we try to minimize the error between actual and predicted values and thus minimizing cost function.*

In the figure, the red points are the data points and the blue line is the predicted line for the training data. To get the predicted value, these data points are projected on to the line.

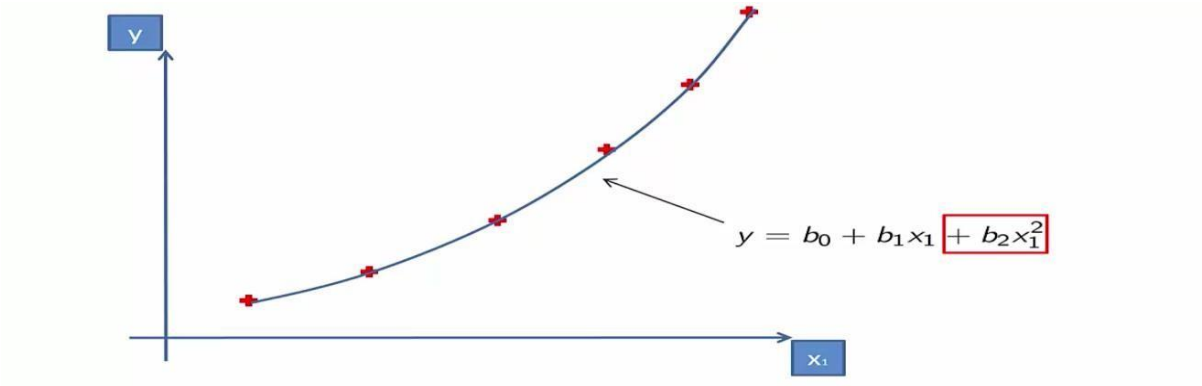
To summarize, our aim is to find such values of coefficients which will minimize the cost function. The most common cost function is **Mean Squared Error** (MSE) which is equal to average squared difference between an observation's actual and predicted values. The coefficient values can be calculated using **Gradient Descent** approach which will be discussed in detail in later articles. To give a brief understanding, in Gradient descent we start with some random values of coefficients, compute gradient of cost function on these values, update the coefficients and calculate the cost function again. This process is repeated until we find a minimum value of cost function.

**2.Polynomial Regression**

In polynomial regression, we transform the original features into polynomial features of a given degree and then apply Linear Regression on it. Consider the above linear model Y = a+bX is transformed to something like

$$Y = a+bX+cX^2$$

It is still a linear model but the curve is now quadratic rather than a line. Scikit-Learn provides PolynomialFeatures class to transform the features.

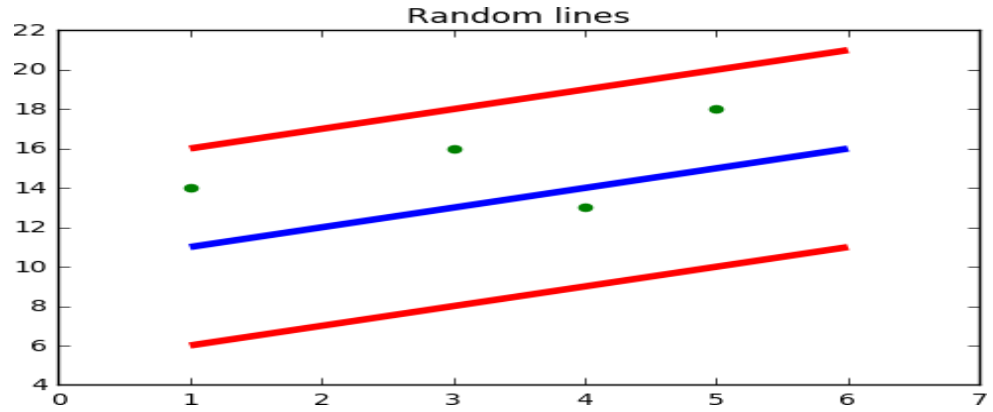



If we increase the degree to a very high value, the curve becomes overfitted as it learns the noise in the data as well.

**3.Support Vector Regression**

In SVR, we identify a hyperplane with maximum margin such that maximum number of data points are within that margin. SVRs are almost similar to SVM classification algorithm. We will discuss SVM algorithm in detail in my next article.

Instead of minimizing the error rate as in simple linear regression, we try to fit the error within a certain threshold. Our objective in SVR is to basically consider the points that are within the margin. **Our best fit line is the hyperplane that has maximum number of points.**



**4.Decision Tree Regression**

Decision trees can be used for classification as well as regression. In decision trees, at each level we need to identify the splitting attribute. In case of regression, the ID3 algorithm can be used to identify the splitting node by *reducing standard deviation (in classification information gain is used).*

A decision tree is built by partitioning the data into subsets containing instances with similar values (homogenous). Standard deviation is used to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous, its standard deviation is zero.

The steps for finding splitting node is briefly described as below:

1. Calculate standard deviation of target variable using below formula.

$$\textbf{Standard Deviation} = S = \sqrt{\frac{\Sigma(x-\bar{x})^2}{n}}$$

2. Split the dataset on different attributes and calculate standard deviation for each branch (standard deviation for target and predictor). This value is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

$$SDR(T,X) = S(T) - S(T,X)$$

3. The attribute with the largest standard deviation reduction is chosen as the splitting node.

4. The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until all data is processed.

To avoid overfitting, Coefficient of Deviation (**CV**) is used which decides when to stop branching. **Finally the average of each branch is assigned to the related leaf node (in regression mean is taken where as in classification mode of leaf nodes is taken).**

**5.Random Forest Regression**

Random forest is an ensemble approach where we take into account the predictions of several decision regression trees.

1. Select K random points
2. Identify n where n is the number of decision tree regressors to be created. Repeat step 1 and 2 to create several regression trees.
3. The average of each branch is assigned to leaf node in each decision tree.
4. To predict output for a variable, the average of all the predictions of all decision trees are taken into consideration.

Random Forest prevents overfitting (which is common in decision trees) by creating random subsets of the features and building smaller trees using these subsets.

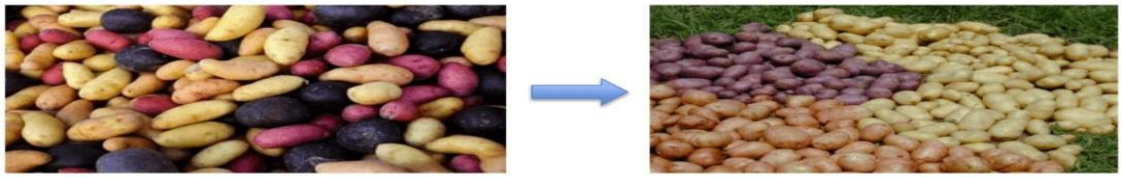
The above explanation is a brief overview of each regression type.

**Unsupervised and descriptive learning**

- Unsupervised machine learning finds all kind of unknown patterns in data.
- Unsupervised methods help you to find features which can be useful for categorization.
- It is taken place in real time, so all the input data to be analyzed and labeled in the presence of learners.
- It is easier to get unlabeled data from a computer than labeled data, which needs manual intervention.

**Types of Unsupervised Learning**

Unsupervised learning problems further grouped into clustering and association problems. Clustering




Clustering is an important concept when it comes to unsupervised learning. It mainly deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process your data and find natural clusters(groups) if they exist in the data. You can also modify how many clusters your algorithms should identify. It allows you to adjust the granularity of these groups.

There are different types of clustering you can utilize:

**Exclusive (partitioning)**

In this clustering method, Data are grouped in such a way that one data can belong to one cluster only.

Example: K-means

**Agglomerative**

In this clustering technique, every data is a cluster. The iterative unions between the two nearest clusters reduce the number of clusters.

Example: Hierarchical clustering

**Overlapping**

In this technique, fuzzy sets is used to cluster data. Each point may belong to two or more clusters with separate degrees of membership.

**Descriptive Learning** : Using descriptive analysis you came up with the idea that, two products A (Burger) and B (french fries) are brought together with very high frequency. Now you want that if user buys A then machine should automatically give him a suggestion to buy B. So by seeing past data and deducing what could be the possible factors influencing this situation can be achieved using ML.

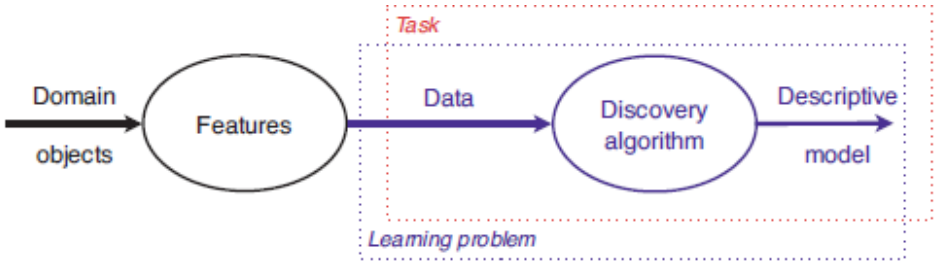



Figure 3.4. In descriptive learning the task and learning problem coincide: we do not have a separate training set, and the task is to produce a descriptive model of the data.

**Predictive Learning** : We want to increase our sales, using descriptive learning we came to know about what could be the possible factors influencing sales. By tuning the parameters in such a way so that sales should be maximized in the next quarter, and therefore predicting what sales we could generate and hence making investments accordingly. This task can be handled using ML also.

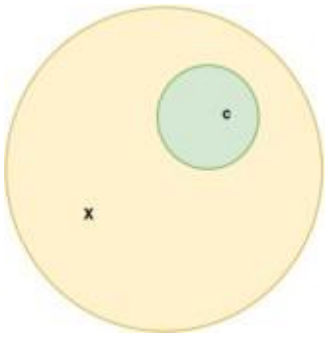
## Chapter-4
## Concept learning

**Concept learning**, also known as **category learning.** "The search for and listing of attributes that can be used to distinguish exemplars from non exemplars of various categories". It is Acquiring the definition of a general category from given sample positive and negative training examples of the category.

Much of human learning involves acquiring general concepts from past experiences. For example, humans identify different vehicles among all the vehicles based on specific sets of features defined over a large set of features. This special set of features differentiates the subset of cars in a set of vehicles. This set of features that differentiate cars can be called a concept.

Similarly, machines can learn from concepts to identify whether an object belongs to a specific category by processing past/training data to find a hypothesis that best fits the training examples.

Target concept:




The set of items/objects over which the concept is defined is called the set of instances and denoted by X. The concept or function to be learned is called the target concept and denoted by c. It can be seen as a boolean valued function defined over X and can be represented as **c: X -> {0, 1}**.

If we have a set of training examples with specific features of target concept C, the problem faced by the learner is to estimate C that can be defined on training data.

H is used to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. The goal of a learner is to find a hypothesis H that can identify all the objects in X so that **h(x) = c(x) for all x in X**.

An algorithm that supports concept learning requires:

1. **Training data** (past experiences to train our models)
2. **Target concept** (hypothesis to identify data objects)
3. **Actual data objects** (for testing the models)

**The hypothesis space**

Each of the data objects represents a concept and hypotheses. Considering a hypothesis **<true, true, false, false>** is more specific because it can cover only one sample. Generally, we can add some notations into this hypothesis. We have the following notations:

1. □ (represents a hypothesis that rejects all)
2. **< ? , ? , ? , ? >** (accepts all)
3. **<true, false, ? , ? >** (accepts some)

The hypothesis □ will reject all the data samples. The hypothesis **<? , ? , ? , ? >** will accept all the data samples. The **?** notation indicates that the values of this specific feature do not affect the result.

The total number of the possible hypothesis is **(3 * 3 * 3 * 3) + 1** — 3 because one feature can have either true, false, or **?** and one hypothesis for rejects all (□).

**General to Specific**

Many machine learning algorithms rely on the concept of general-to-specific ordering of hypothesis.

1. h1 = < true, true, ?, ? >
2. h2 = < true, ? , ? , ? >

Any instance classified by h1 will also be classified by h2. We can say that h2 is more general than h1. Using this concept, we can find a general hypothesis that can be defined over the entire dataset X.

To find a single hypothesis defined on X, we can use the concept of being more general than partial ordering. One way to do this is start with the most specific hypothesis from H and generalize this hypothesis each time it fails to classify and observe positive training data object as positive.

1. The first step in the Find-S algorithm is to start with the most specific hypothesis, which can be denoted by **h <- <□, □, □, □>**.
2. This step involves picking up next training sample and applying Step 3 on the sample.
3. The next step involves observing the data sample. If the sample is negative, the hypothesis remains unchanged and we pick the next training sample by processing Step 2 again. Otherwise, we process Step 4.
4. If the sample is positive and we find that our initial hypothesis is too specific because it does not cover the current training sample, then we need to update our current hypothesis. This can be done by the pairwise conjunction (logical *and* operation) of the current hypothesis and training sample.

   If the next training sample is **<true, true, false, false>** and the current hypothesis is **<□, □, □, □>**, then we can directly replace our existing hypothesis with the new one.

If the next positive training sample is **<true, true, false, true>** and current hypothesis is **<true, true, false, false>**, then we can perform a pairwise conjunctive. With the current hypothesis and next training sample, we can find a new hypothesis by putting **?** in the place where the result of conjunction is false:

**<true, true, false, true>** □ **<true, true, false, false> = <true, true, false, ?>**

Now, we can replace our existing hypothesis with the new one: **h <-<true, true, false, ?>**

5. This step involves repetition of Step 2 until we have more training samples.
6. Once there are no training samples, the current hypothesis is the one we wanted to find. We can use the final hypothesis to classify the real objects.

Paths through the hypothesis space

As we can clearly see in Figure 4.4, in this example we have not one but two most general hypotheses. What we can also notice is that *every concept between the least general one and one of the most general ones is also a possible hypothesis*, i.e., covers all the positives and none of the negatives. Mathematically speaking we say that the set of **Algorithm 4.3:** LGG-Conj-ID(*x*, *y*) – find least general conjunctive generalisation of two conjunctions, employing internal disjunction.

**Input** : conjunctions *x*, *y*.
**Output** : conjunction *z*.
**1** *z* ←true;
**2 for** each feature *f* **do**
**3 if** *f* = *vx* is a conjunct in *x* and *f* = *vy* is a conjunct in *y* **then**
**4** add *f* = Combine-ID(*vx* , *vy* ) to *z*; // Combine-ID: see text
**5 end**
**6 end**
***7* return** *z*



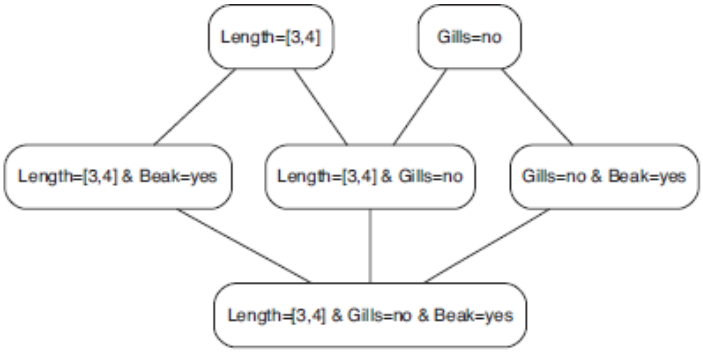



**Figure 4.4.** **(top)** A snapshot of the expanded hypothesis space that arises when internal disjunction is used for the 'Length' feature. We now need one more generalisation step to travel upwards from a completely specified example to the empty conjunction. **(bottom)** The version space consists of one least general hypothesis, two most general hypotheses, and three in between.

hypotheses that agree with the data is a *convex set*, which basically means that we can interpolate between any two members of the set, and if we find a concept that is less general than one and more general than the other then that concept is also a member of the set. This in turn means that we can describe the set of all possible hypotheses by its least and most general members. This is summed up in the following definition.

**Definition 4.1 (Version space).** *A concept is* complete *if it covers all positive examples. A concept is* consistent *if it covers none of the negative examples. The* version space *is the set of all complete and consistent concepts. This set is convex and is fully defined by its least and most general elements.*

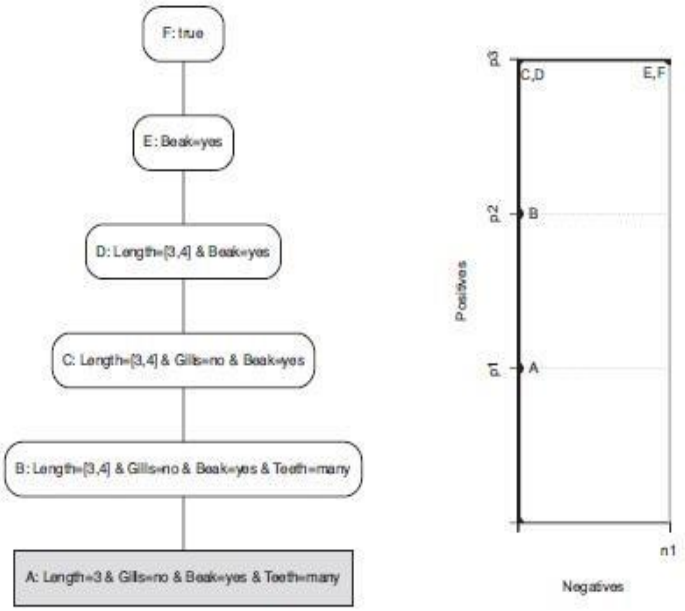



**Figure 4.5.** **(left)** A path in the hypothesis space of Figure 4.3 from one of the positive examples (p1, see Example 4.2 on p.110) all the way up to the empty concept. Concept A covers a single example; B covers one additional example; C and D are in the version space, and so cover all three positives; E and F also cover the negative. **(right)** The corresponding coverage curve, with ranking p1 – p2 – p3 – n1.



## Beyond conjunctive concepts

Recall from Background 4.1 that a conjunctive normal form expression (CNF) is a conjunction of disjunctions of literals, or equivalently, a conjunction of clauses. The conjunctions of literals we have looked at until now are trivially in CNF where each disjunction consists of a single literal. CNF expressions are much more expressive, particularly since literals can occur in several clauses. We will look at an algorithm for learning Horn theories, where each clause $A \rightarrow B$ is a Horn clause, i.e., $A$ is a conjunction of literals and $B$ is a single literal. For ease of notation we will restrict attention to Boolean features, and write F for F = true and ¬F for F = false. In the example below we adapt the dolphins example to Boolean variables ManyTeeth (standing for Teeth = many), Gills, Short (standing for Length = 3) and Beak.

When we looked at learning conjunctive concepts, the main intuition was that uncovered positive examples led us to generalise by dropping literals from the conjunction, while covered negative examples require specialisation by adding literals. This intuition still holds if we are learning Horn theories, but now we need to think 'clauses' rather than 'literals'. Thus, if a Horn theory doesn't cover a positive we need to drop all clauses that violate the positive, where a clause $A \rightarrow B$ violates a positive if all literals in the conjunction $A$ are true in the example, and $B$ is false.

Things get more interesting if we consider covered negatives, since then we need to find one or more clauses to add to the theory in order to exclude the negative. For example, suppose that our current hypothesis covers the negative

$$\text{ManyTeeth} \wedge \text{Gills} \wedge \text{Short} \wedge \neg\text{Beak}$$

To exclude it, we can add the following Horn clause to our theory:

$$\text{ManyTeeth} \wedge \text{Gills} \wedge \text{Short} \rightarrow \text{Beak}$$

While there are other clauses that can exclude the negative (e.g., ManyTeeth → Beak) this is the most specific one, and hence least at risk of also excluding covered positives. However, the most specific clause excluding a negative is only unique if the negative has exactly one literal set to false. For example, if our covered negative is

$$\text{ManyTeeth} \wedge \text{Gills} \wedge \neg\text{Short} \wedge \neg\text{Beak}$$

then we have a choice between the following two Horn clauses:

$$\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Short}$$
$$\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Beak}$$

Notice that, the fewer literals are set to true in the negative example, the more general the clauses excluding the negative are.

The approach of Algorithm 4.5 is to add *all* of these clauses to the hypothesis. However, the algorithm applies two clever tricks. The first is that it maintains a list $S$ of negative examples, from which it periodically rebuilds the hypothesis. The second is that, rather than simply adding new negative examples to the list, it tries to find negatives with fewer literals set to true, since this will result in more general clauses. This is possible if we assume we have access to a *membership oracle* $Mb$ which can tell us whether a particular example is a member of the concept we're learning or not. So in line 7 of the algorithm we form the *intersection* of a new negative $x$ and an existing one $s \in S$ – i.e., an example with only those literals set to true which are true in both $x$ and $s$ – and pass the result $z$ to the membership oracle to check whether it belongs to the target concept. The algorithm also assumes access to an *equivalence oracle* $Eq$ which either tells us that our current hypothesis $h$ is logically equivalent to the target formula $f$, or else produces a *counter-example* that can be either a false positive (it is covered by $h$ but not by $f$) or a false negative (it is covered by $f$ but not by $h$).

**Algorithm 4.5:** Horn($Mb, Eq$) – learn a conjunction of Horn clauses from membership and equivalence oracles.

**Input** : equivalence oracle $Eq$; membership oracle $Mb$.
**Output** : Horn theory $h$ equivalent to target formula $f$.

```
1  h ←true;                                   // conjunction of Horn clauses, initially empty
2  S ←∅ ;                                     // a list of negative examples, initially empty
3  while Eq(h) returns counter-example x do
4      if x violates at least one clause of h then        // x is a false negative
5          specialise h by removing every clause that x violates
6      else                                               // x is a false positive
7          find the first negative example s ∈ S such that (i) z = s ∩ x has fewer true
           literals than s, and (ii) Mb(z) labels it as a negative;
8          if such an example exists then replace s in S with z, else append x to the
           end of S;
9          h ←true;
10         for all s ∈ S do                               // rebuild h from S
11             p ←the conjunction of literals true in s;
12             Q ←the set of literals false in s;
13             for all q ∈ Q do h ←h ∧ (p → q);
14         end
15     end
16 end
17 return h
```

___XXX___

**UNIT- III**

**Tree models: Decision trees, Ranking and probability estimation trees, Tree learning as variance reduction. Rule models: Learning ordered rule lists, Learning unordered rule sets, Descriptive rule learning, First-order rule learning**

## 1)Tree models:

- A tree model is a hierarchical structure of conditions, where leafs contain tree outcome.
- They represent recursive divide-and-conquer strategies.
- Tree models are among the most popular models in machine learning, because they are easy to understand and interpret:
- E.g., Kinect uses them to detect character pose.

**a) Decision trees**

**b) Ranking and probability estimation trees**

**c) Tree learning as variance reduction**

## a) Decision trees

A **decision tree** is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (**Decision** taken after computing all attributes). The paths from root to leaf represent classification rules.

**Decision Analysis**, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of three types of nodes:

1. Decision nodes – typically represented by squares
2. Chance nodes – typically represented by circles
3. End nodes – typically represented by triangles

Decision trees are commonly used in operations research and operations management. If, in practice, decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

Decision trees, influence diagrams, utility functions, and other decision analysis tools and methods are taught to undergraduate students in schools of business, health economics, and public health, and are examples of operations research or management science methods.

**Decision tree rules**

The decision tree can be linearized into decision rules, where the outcome is the contents of the leaf node, and the conditions along the path form a conjunction in the if clause. In general, the rules have the form: if condition1 and condition2 and condition3 then outcome.

Decision rules can be generated by constructing association rules with the target variable on the right. They can also denote temporal or causal relations

## Decision tree

How to define BestSplit$(D, F)$ for classification?

Assume that we have binary features and two classes only. Let $D^{\oplus}$ denote set of instances from positive class and $D^{\ominus}$ from negative class, $D = D^{\oplus} \cup D^{\ominus}$.

Let split $D$ into $D_1$ and $D_2$ using an attribute. The best situation is where $D_1^{\oplus} = D^{\oplus}$ and $D_1^{\ominus} = \emptyset$ or $D_1^{\oplus} = \emptyset$ and $D_1^{\ominus} = D^{\ominus}$. In this cases the child node is said to be *pure*.

This, however, is unlikely in practice, thus we have to measure *impurity* of children nodes somehow.

## Impurity measures

- Minority class $min(\dot{p}, 1 - \dot{p})$ – proportion of misclassified examples if labeling leaf using majority class
- Gini index $2\dot{p}(1 - \dot{p})$ – expected error if labeling examples in leaf randomly with prorability $\dot{p}$for positive class and $1 - \dot{p}$ for negative class
- Entropy $-\dot{p}\log_2 \dot{p} - (1 - \dot{p})\log_2(1 - \dot{p})$ – expected amount of information in bits required to classify an example in leaf

## Decision tree example

Suppose you come across a number of sea animals that you suspect belong to the same species. You observe their length in metres, whether they have gills, whether they have a prominent beak, and whether they have few or many teeth. Let the following be dolphins (positive class):

p1: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p2: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p3: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few
p4: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p5: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

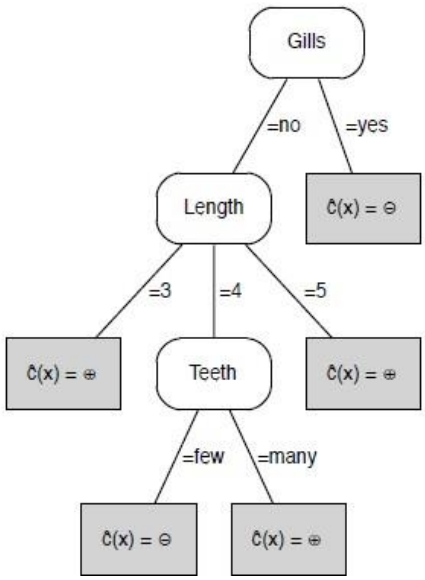
and the following be not dolphins (negative class):

n1: Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n2: Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n3: Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n4: Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

Figure 5.1, p.130

Decision tree example

A decision tree learned on this data separates the positives and negatives perfectly.



 Algorithm 5.2, p.137 **Finding the best split for a decision tree**

**Algorithm** BestSplit-Class($D, F$) – find the best split for a decision tree.

**Input** : data $D$; set of features $F$.
**Output** : feature $f$ to split on.

```
1 I_min ← 1;
2 for each f ∈ F do
3     split D into subsets D_1, ..., D_l according to the values v_j of f;
4     if Imp({D_1, ..., D_l}) < I_min then
5         I_min ← Imp({D_1, ..., D_l});
6         f_best ← f;
7     end
8 end
9 return f_best
```

**b) Ranking and probability estimation trees**

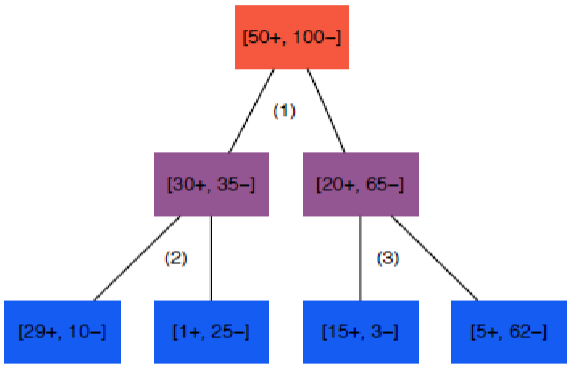
Decision trees divide the instance space into segments, by learning ordering on those segments the decision trees can be turned into rankers.

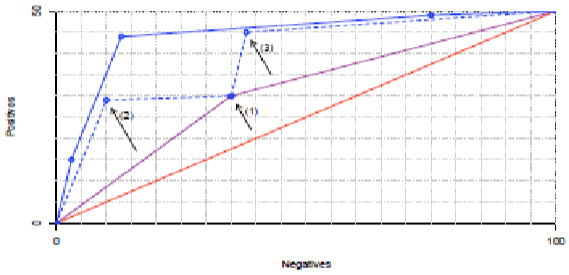
Thanks to access to class distribution in each leaf the optimal orderdering for the training data can be obtained from empirical probabilities $\dot{p}$ (of positive class).

The ranking obtained from the empirical probabilities in the leaves of a decision tree yields a convex ROC curve on the training data.

Figure 5.4, p.140

## Growing a tree



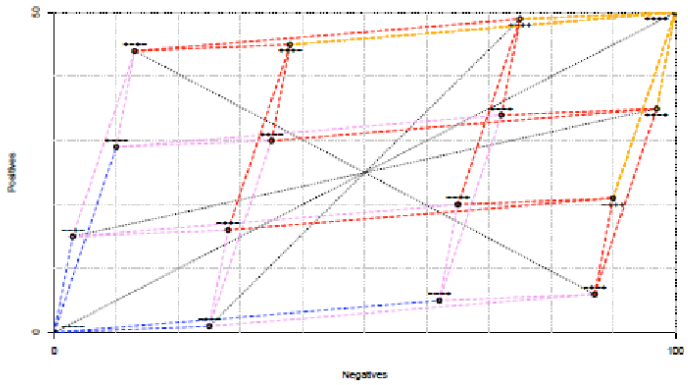




**(left)** Abstract representation of a tree with numbers of positive and negative examples covered in each node. Binary splits are added to the tree in the order indicated. **(right)** Adding a split to the tree will add new segments to the coverage curve as indicated by the arrows. After a split is added the segments may need reordering, and so only the solid lines represent actual coverage curves.

---

Figure 5.5, p.141

## Labelling a tree

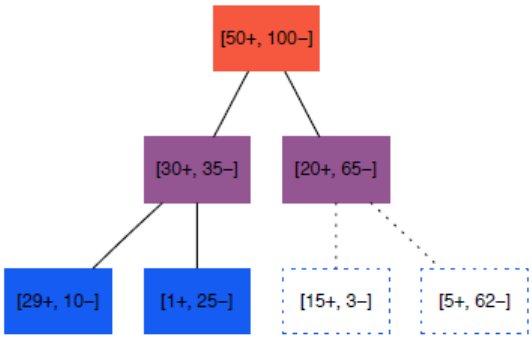


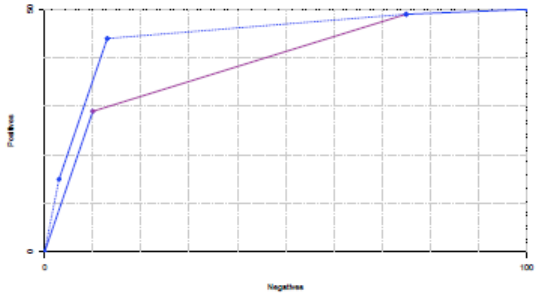

Graphical depiction of all possible labellings and all possible rankings that can be obtained with the four-leaf decision tree in Figure 5.4. There are $2^4 = 16$ possible leaf labellings; e.g., '$+-+-$' denotes labelling the first and third leaf from the left as $+$ and the second and fourth leaf as $-$. There are $4! = 24$ possible blue-violet-red-orange paths through these points which start in $----$ and switch each leaf to $+$ in some order; these represent all possible four-segment coverage curves or rankings.

---

Figure 5.6, p.143

## Pruning a tree



**(left)** To achieve the labelling +− ++ we don't need the right-most split, which can therefore be pruned away. **(right)** Pruning doesn't affect the chosen operating point, but it does decrease the ranking performance of the tree.

- Prunning must not improve classification accuracy on training set
- However may improve generalization accuracy on test set
- A popular algorithm for pruning decision trees is *reduced-error pruning* that employs a separate *prunning set* of labelled data not seen during training.

Algorithm 5.3, p.144

## Reduced-error pruning

**Algorithm** PruneTree($T, D$) – reduced-error pruning of a decision tree.

**Input** : decision tree $T$; labelled data $D$.
**Output** : pruned tree $T'$.

**for** every internal node $N$ of $T$, starting from the bottom **do**
  $T_N \leftarrow$ subtree of $T$ rooted at $N$;
  $D_N \leftarrow \{x \in D | x$ is covered by $N\}$;
  **if** accuracy of $T_N$ over $D_N$ is worse than majority class in $D_N$ **then**
    replace $T_N$ in $T$ by a leaf labelled with the majority class in $D_N$;
  **end**
**end**
**return** pruned version of $T$

Example 5.3, p.144

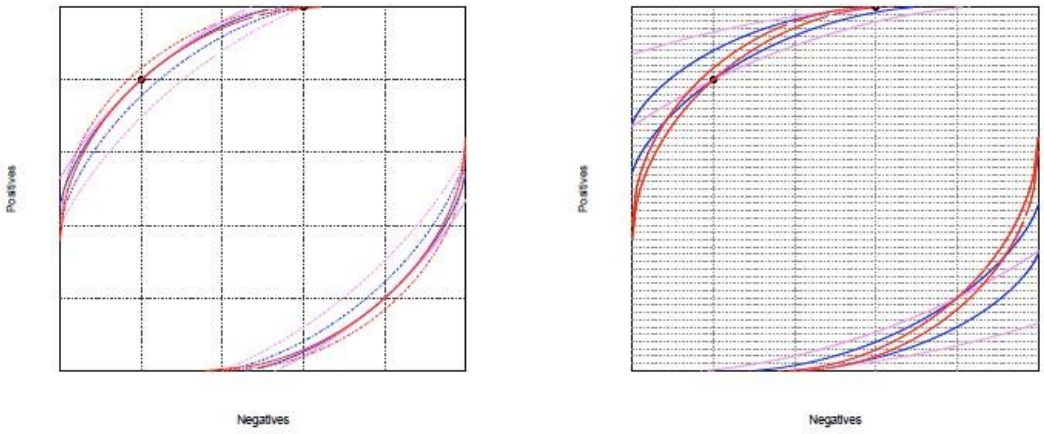
## Skew sensitivity of splitting criteria II

You then remember that mistakes on the positives are about ten times as costly as mistakes on the negatives.

- You're not quite sure how to work out the maths, and so you decide to simply have ten copies of every positive: the splits are now $[80+,2-][20+,8-]$ and $[100+,6-][0+,4-]$.
- You recalculate the three splitting criteria and now all three favour the second split.
- Even though you're slightly bemused by all this, you settle for the second split since all three splitting criteria are now unanimous in their recommendation.

Figure 5.7, p.146

## Skew sensitivity of splitting criteria




**(left)** ROC isometrics for entropy in blue, Gini index in violet and $\sqrt{\text{Gini}}$ in red through the splits $[8+,2-][2+,8-]$ (solid lines) and $[10+,6-][0+,4-]$ (dotted lines). Only $\sqrt{\text{Gini}}$ prefers the second split. **(right)** The same isometrics after inflating the positives with a factor 10. All splitting criteria now favour the second split; the $\sqrt{\text{Gini}}$ isometrics are the only ones that haven't moved.

## c) Tree learning as variance reduction

### (i)Regression Tree

## Tree learning as variance reduction

- The variance of a Boolean (i.e., Bernoulli) variable with success probability $\dot{p}$ is $\dot{p}(1-\dot{p})$, which is half the Gini index. So we could interpret the goal of tree learning as minimising the class variance (or standard deviation, in case of $\sqrt{\text{Gini}}$) in the leaves.
- In regression problems we can define the variance in the usual way:

$$\text{Var}(Y) = \frac{1}{|Y|}\sum_{y\in Y}(y-\overline{y})^2$$

If a split partitions the set of target values $Y$ into mutually exclusive sets $\{Y_1,\ldots,Y_l\}$, the weighted average variance is then

$$\text{Var}(\{Y_1,\ldots,Y_l\}) = \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\text{Var}(Y_j) = \ldots = \frac{1}{|Y|}\sum_{y\in Y}y^2 - \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\overline{y}_j^2$$

The first term is constant for a given set $Y$ and so we want to maximise the weighted average of squared means in the children.

---

Example 5.4, p.150

## Learning a regression tree I

Imagine you are a collector of vintage Hammond tonewheel organs. You have been monitoring an online auction site, from which you collected some data about interesting transactions:

| # | Model | Condition | Leslie | Price |
|---|---|---|---|---|
| 1. | B3 | excellent | no | 4513 |
| 2. | T202 | fair | yes | 625 |
| 3. | A100 | good | no | 1051 |
| 4. | T202 | good | no | 270 |
| 5. | M102 | good | yes | 870 |
| 6. | A100 | excellent | no | 1770 |
| 7. | T202 | fair | no | 99 |
| 8. | A100 | good | yes | 1900 |
| 9. | E112 | fair | no | 77 |

Example 5.4, p.150

## Learning a regression tree II

From this data, you want to construct a regression tree that will help you determine a reasonable price for your next purchase.
There are three features, hence three possible splits:

| | |
|---|---|
| Model = [A100, B3, E112, M102, T202] | [1051, 1770, 1900][4513][77][870][99, 270, 625] |
| Condition = [excellent, good, fair] | [1770, 4513][270, 870, 1051, 1900][77, 99, 625] |
| Leslie = [yes, no] | [625, 870, 1900][77, 99, 270, 1051, 1770, 4513] |

The means of the first split are 1574, 4513, 77, 870 and 331, and the weighted average of squared means is $3.21 \cdot 10^6$.
The means of the second split are 3142, 1023 and 267, with weighted average of squared means $2.68 \cdot 10^6$;
for the third split the means are 1132 and 1297, with weighted average of squared means $1.55 \cdot 10^6$.
We therefore branch on Model at the top level. This gives us three single-instance leaves, as well as three A100s and three T202s.

Example 5.4, p.150

## Learning a regression tree III

For the A100s we obtain the following splits:

| | |
|---|---|
| Condition = [excellent, good, fair] | [1770][1051, 1900][] |
| Leslie = [yes, no] | [1900][1051, 1770] |

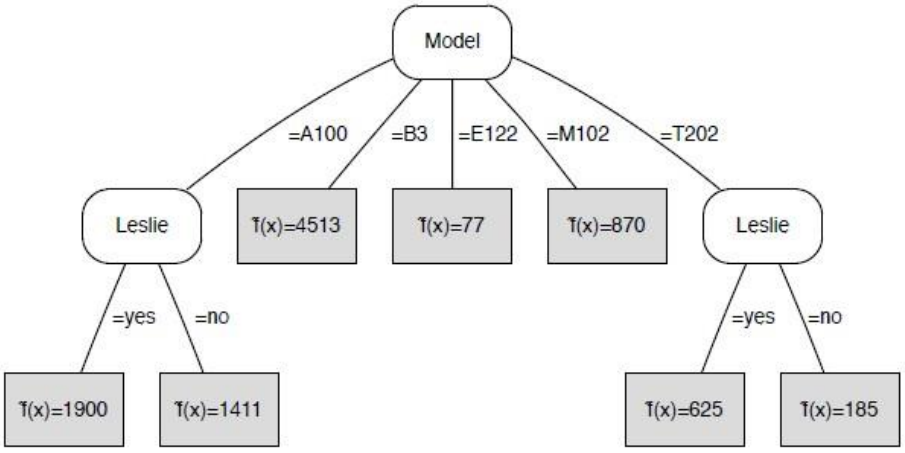
Without going through the calculations we can see that the second split results in less variance (to handle the empty child, it is customary to set its variance equal to that of the parent). For the T202s the splits are as follows:

| | |
|---|---|
| Condition = [excellent, good, fair] | [][270][99, 625] |
| Leslie = [yes, no] | [625][99, 270] |

Again we see that splitting on Leslie gives tighter clusters of values. The learned regression tree is depicted in Figure 5.8.

Figure 5.8, p.150

## A regression tree




A regression tree learned from the data in Example 5.4.

**(ii)Clustering Tree**

Example 5.5, p.152

## Learning a clustering tree I

Assessing the nine transactions on the online auction site from Example 5.4, using some additional features such as reserve price and number of bids, you come up with the following dissimilarity matrix:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 11 | 6 | 13 | 10 | 3 | 13 | 3 | 12 |
| 11 | **0** | 1 | **1** | 1 | 3 | **0** | 4 | 0 |
| 6 | 1 | **0** | 2 | 1 | **1** | 2 | **2** | 1 |
| 13 | **1** | 2 | **0** | 0 | 4 | **0** | 4 | 0 |
| 10 | 1 | 1 | 0 | 0 | 3 | 0 | 2 | 0 |
| 3 | 3 | **1** | 4 | 3 | **0** | 4 | **1** | 3 |
| 13 | **0** | 2 | **0** | 0 | 4 | **0** | 4 | 0 |
| 3 | 4 | **2** | 4 | 2 | **1** | 4 | **0** | 4 |
| 12 | 0 | 1 | 0 | 0 | 3 | 0 | 4 | 0 |

This shows, for instance, that the first transaction is very different from the other eight. The average pairwise dissimilarity over all nine transactions is 2.94.

Example 5.5, p.152

## Learning a clustering tree II

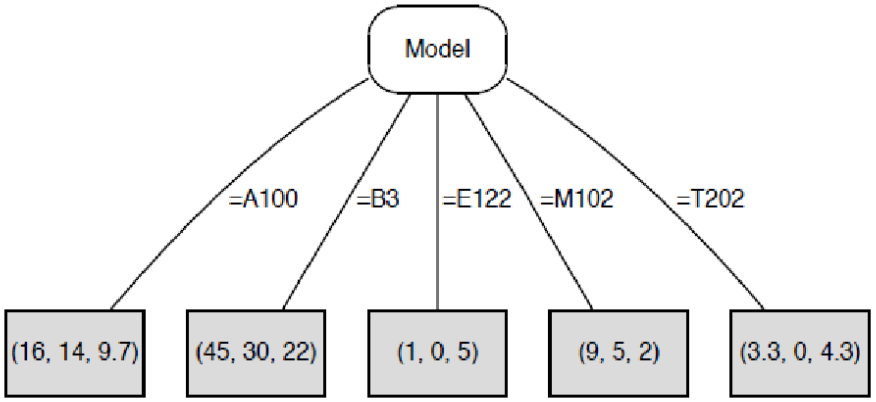
Using the same features from Example 5.4, the three possible splits are (now with transaction number rather than price):

| | |
|---|---|
| Model = [A100, B3, E112, M102, T202] | [3,6,8][1][9][5][2,4,7] |
| Condition = [excellent, good, fair] | [1,6][3,4,5,8][2,7,9] |
| Leslie = [yes, no] | [2,5,8][1,3,4,6,7,9] |

The cluster dissimilarity among transactions 3, 6 and 8 is $\frac{1}{3^2}(\mathbf{0}+\mathbf{1}+\mathbf{2}+\mathbf{1}+\mathbf{0}+\mathbf{1}+\mathbf{2}+\mathbf{1}+\mathbf{0}) = 0.89$; and among transactions 2, 4 and 7 it is $\frac{1}{3^2}(\mathbf{0}+\mathbf{1}+\mathbf{0}+\mathbf{1}+\mathbf{0}+\mathbf{0}+\mathbf{0}+\mathbf{0}+\mathbf{0}) = 0.22$. The other three children of the first split contain only a single element and so have zero cluster dissimilarity. The weighted average cluster dissimilarity of the split is then $3/9 \cdot 0.89 + 1/9 \cdot 0 + 1/9 \cdot 0 + 1/9 \cdot 0 + 3/9 \cdot 0.22 = 0.37$. For the second split, similar calculations result in a split dissimilarity of $2/9 \cdot 1.5 + 4/9 \cdot 1.19 + 3/9 \cdot 0 = 0.86$, and the third split yields $3/9 \cdot 1.56 + 6/9 \cdot 3.56 = 2.89$. The Model feature thus captures most of the given dissimilarities, while the Leslie feature is virtually unrelated.

Figure 5.9, p.154

## A clustering tree




A clustering tree learned from the data in Example 5.6 using Euclidean distance on the numerical features.

## 2) Rule models:

They are four types of rule models

a) Learning ordered rule lists, b) Learning unordered rule sets,

c) Descriptive rule learning  d) First-order rule learning

## a) Learning ordered rule lists

Algorithm 6.1, p.163 — Learning an ordered list of rules

**Algorithm** LearnRuleList($D$) – learn an ordered list of rules.

**Input** : labelled training data $D$.
**Output** : rule list $R$.

```
1 R ← ∅;
2 while D ≠ ∅ do
3     r ← LearnRule(D) ;          // LearnRule: see Algorithm 6.2
4     append r to the end of R;
5     D ← D \ {x ∈ D | x is covered by r};
6 end
7 return R
```

Example 6.1, p.159 — Learning a rule list I

Consider again our small dolphins data set with positive examples

p1: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p2: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p3: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few
p4: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p5: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

and negatives

n1: Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n2: Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n3: Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n4: Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

Example 6.1, p.159

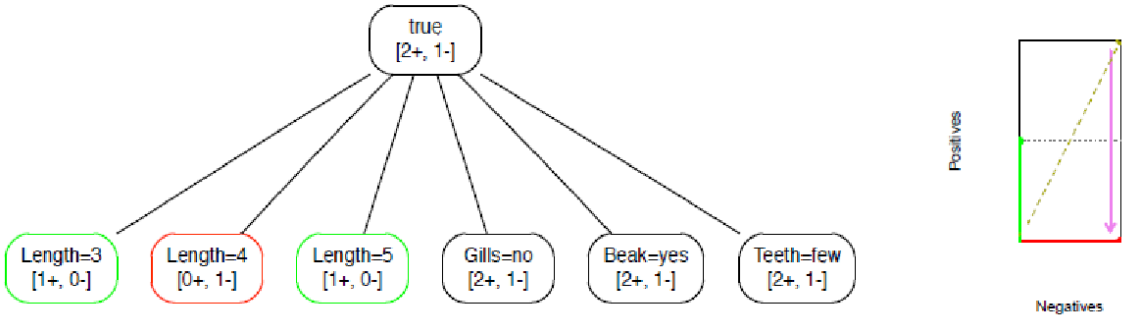
## Learning a rule list II

- The nine possible literals are shown with their coverage counts in Figure 6.2 (left).
- Three of these are pure; in the impurity isometrics plot in Figure 6.2 (right) they end up on the $x$-axis and $y$-axis.
- One of the literals covers two positives and two negatives, and therefore has the same impurity as the overall data set; this literal ends up on the ascending diagonal in the coverage plot.

---

Figure 6.4, p.162

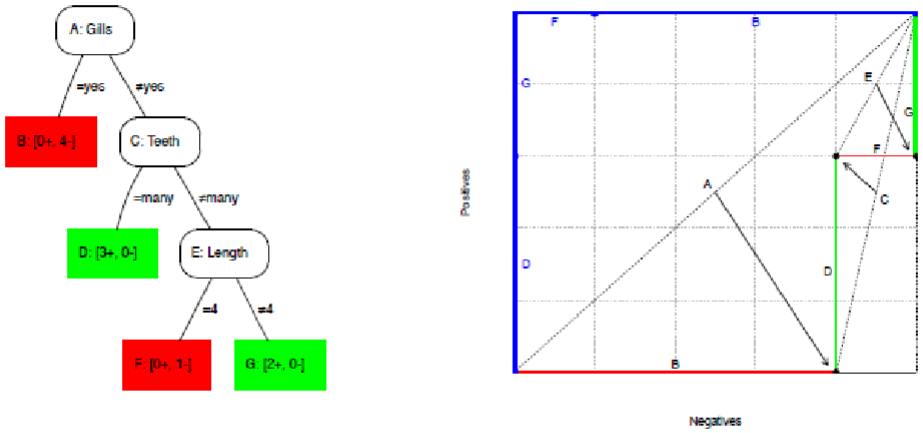
## Constructing the third rule




**(left)** The third rule covers the one remaining negative example, so that the remaining positives can be swept up by a default rule. **(right)** This will collapse the coverage space.

---

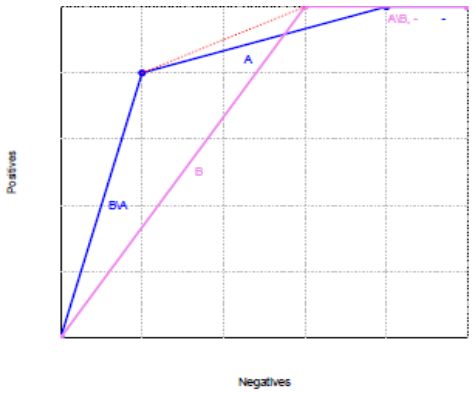
Figure 6.5, p.164 **Rule list as a tree**




**(left)** A right-branching feature tree corresponding to a list of single-literal rules. **(right)** The construction of this feature tree depicted in coverage space. The leaves of the tree are either purely positive (in green) or purely negative (in red). Reordering these leaves on their empirical probability results in the blue coverage curve. As the rule list separates the classes this is a perfect coverage curve.

## i)Rules List For Ranking and Probability Estimations

Rule lists inherit the property of decision trees that their training set coverage curve is always convex




Coverage curves of two rule lists consisting of the rules from Example 6.2, in different order (AB in blue and BA in violet). B \ A corresponds to the coverage of rule B once the coverage of rule A is taken away, and '-' denotes the default rule. The dotted segment in red connecting the two curves corresponds to the overlap of the two rules A ∧ B, which is not accessible by either rule list.

## b)Learning ordered rule lists

Algorithm 6.3, p.171 — Learning an unordered set of rules

**Algorithm** LearnRuleSet($D$) – learn an unordered set of rules.

**Input** : labelled training data $D$.
**Output** : rule set $R$.

```
1  R ← ∅;
2  for every class C_i do
3  |   D_i ← D;
4  |   while D_i contains examples of class C_i do
5  |   |   r ← LearnRuleForClass(D_i, C_i) ;  // LearnRuleForClass: see Algorithm 6.4
6  |   |   R ← R ∪ {r};
7  |   |   D_i ← D_i \ {x ∈ C_i | x is covered by r} ;   // remove only positives
8  |   end
9  end
10 return R
```

Algorithm 6.4, p.171 — Learning a single rule for a given class

**Algorithm** LearnRuleForClass($D, C_i$) – learn a single rule for a given class.

**Input** : labelled training data $D$; class $C_i$.
**Output** : rule $r$.

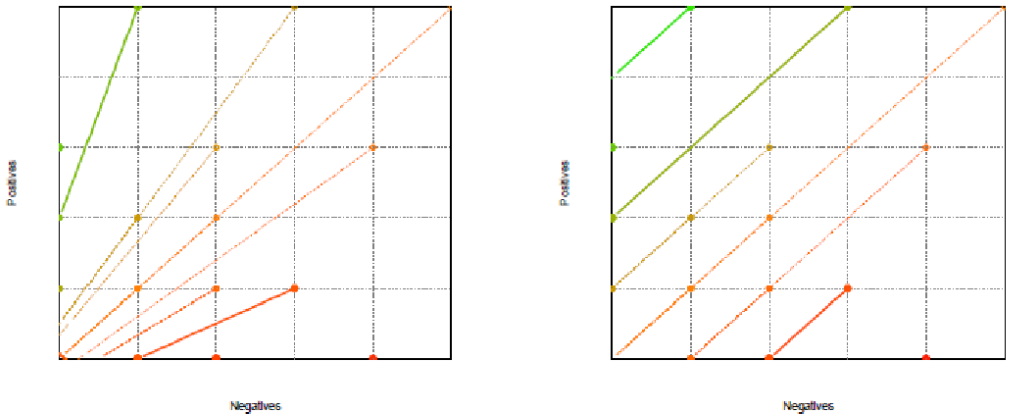
```
1  b ← true;
2  L ← set of available literals ;          // can be initialised by seed example
3  while not Homogeneous(D) do
4  |   l ← BestLiteral(D, L, C_i) ;        // e.g. maximising precision on class C_i
5  |   b ← b ∧ l;
6  |   D ← {x ∈ D | x is covered by b};
7  |   L ← L \ {l' ∈ L | l' uses same feature as l};
8  end
9  r ← ·if b then Class = C_i·;
10 return r
```

## c)Descriptive rule learning

## i)Rule Learning for subgroup discovery

*Subgroups* are subsets of the instance space – or alternatively, mappings $\hat{g} : \mathscr{X} \to \{\text{true}, \text{false}\}$ – that are learned from a set of labelled examples $(x_i, l(x_i))$, where $l : \mathscr{X} \to \mathscr{C}$ is the true labelling function.

- A good subgroup is one whose class distribution is significantly different from the overall population. This is by definition true for pure subgroups, but these are not the only interesting ones.
- For instance, one could argue that the complement of a subgroup is as interesting as the subgroup itself: in our dolphin example, the concept Gills = yes, which covers four negatives and no positives, could be considered as interesting as its complement Gills = no, which covers one negative and all positives.
- This means that we need to move away from impurity-based evaluation measures.




**(left)** Subgroups and their isometrics according to Laplace-corrected precision. The solid, outermost isometrics indicate the best subgroups. **(right)** The ranking changes if we order the subgroups on average recall. For example, $[5+, 1-]$ is now better than $[3+, 0-]$ and as good as $[0+, 4-]$.

## ii) Association rule mining:

### Rule-1

Frequent item sets can be used to build *association rules*, which are rules of the form ·**if** $B$ **then** $H$· where both body $B$ and head $H$ are item sets that frequently appear in transactions together.

- ☞ Pick any edge in Figure 6.17, say the edge between {beer} and {nappies, beer}. We know that the support of the former is 3 and of the latter, 2: that is, three transactions involve beer and two of those involve nappies as well. We say that the *confidence* of the association rule ·**if** beer **then** nappies· is 2/3.
- ☞ Likewise, the edge between {nappies} and {nappies, beer} demonstrates that the confidence of the rule ·**if** nappies **then** beer· is 2/4.
- ☞ There are also rules with confidence 1, such as ·**if** beer **then** crisps·; and rules with empty bodies, such as ·**if** true **then** crisps·, which has confidence 5/8 (i.e., five out of eight transactions involve crisps).

---

### Rule-2

But we only want to construct association rules that involve frequent items.

- ☞ The rule ·**if** beer ∧ apples **then** crisps· has confidence 1, but there is only one transaction involving all three and so this rule is not strongly supported by the data.
- ☞ So we first use Algorithm 6.6 to mine for frequent item sets; we then select bodies $B$ and heads $H$ from each frequent set $m$, discarding rules whose confidence is below a given confidence threshold.
- ☞ Notice that we are free to discard some of the items in the maximal frequent sets (i.e., $H \cup B$ may be a proper subset of $m$), because any subset of a frequent item set is frequent as well.

---

**Algorithm** AssociationRules$(D, f_0, c_0)$ – find all association rules exceeding given support and confidence thresholds.

**Input** : data $D \subseteq \mathscr{X}$; support threshold $f_0$; confidence threshold $c_0$.
**Output** : set of association rules $R$.

```
1 R ← ∅;
2 M ← FrequentItems(D, f0) ;                    // FrequentItems: see Algorithm 6.6
3 for each m ∈ M do
4     for each H ⊆ m and B ⊆ m such that H ∩ B = ∅ do
5         if Supp(B ∪ H)/Supp(B) ≥ c0 then R ← R ∪ {·if B then H·};
6     end
7 end
8 return R
```

## Association rule example

A run of the algorithm with support threshold 3 and confidence threshold 0.6 gives the following association rules:

| | |
|---|---|
| ·if beer **then** crisps· | support 3, confidence 3/3 |
| ·if crisps **then** beer· | support 3, confidence 3/5 |
| ·if true **then** crisps· | support 5, confidence 5/8 |

Association rule mining often includes a *post-processing* stage in which superfluous rules are filtered out, e.g., special cases which don't have higher confidence than the general case.

## d) First-order rule learning

→One of the most *expressive* and *human readable* representations for learned hypotheses is sets of *production rules* (*if-then* rules).

→Rules can be derived from other representations (e.g., decision trees) or they can be learned *directly*. Here, we are concentrating on the direct method.

→An important aspect of direct rule-learning algorithms is that they can learn sets of *first-order rules* which have much more representational power than the *propositional* rules that can be derived from decision trees. Learning first-order rules can also be seen as automatically inferring *PROLOG programs* from example.

**Propositional versus First-Order Logic:**

→*Propositional Logic* does not include variables and thus cannot express general relations among the values of the attributes.

*Example 1:* in Propositional logic, you can write: *IF* $(Father_1=Bob) \wedge (Name_2=Bob) \wedge (Female_1=True)$ *THEN* $Daughter_{1,2}=True$.

This rule applies only to a specific family!

*Example 2:* In First-Order logic, you can write

*IF Father(y,x) ^ Female(y), THEN Daughter(x,y)*

This rule (which you cannot write in Propositional Logic) applies to any family!

**Learning Propositional versus First-Order Rules:**

→Both approaches to learning are useful as they address different types of learning problems.

→Like Decision Trees, Feed forward Neural Nets and IBL systems, Propositional Rule Learning systems are suited for problems in which no substantial relationship between the values of the different attributes needs to be represented.

→In First-Order Learning Problems, the hypotheses that must be represented involve relational assertions that can be conveniently expressed using first-order representations such as ***horn clauses*** (***H*** <- $L_1 \wedge \ldots \wedge L_n$



# Linear Models

1. Definition: If x1 and x2 are two scalars or vectors of the same dimension and α and β are arbitrary scalars, then αx1+βx2 is called a linear combination of x1 and x2. If f is a linear function of x, then f (αx1 +βx2) = αf (x1)+βf (x2)

The function value of a linear combination of some inputs is a linear combination of their function values. As a special case, if $\beta = 1-\alpha$ we are taking a weighted average of $x1$ and $x2$, and the linearity of $f$ then means that the function value of the weighted average is the weighted average of the function values. Linear functions take particular forms, depending on the domain and codomain of $f$. If $x$ and $f(x)$ are scalars, it follows that $f$ is of the form $f(x) = a+bx$ for some constants $a$ and $b$; $a$ is called the *intercept* and $b$ the *slope*. If $\mathbf{x} = (x1, \ldots, xd)$ is a vector and $f(\mathbf{x})$ is a scalar, then $f$ is of the form

$$f(\mathbf{x}) = a + b1x1 + \ldots + bd\,xd = a + \mathbf{b}\cdot\mathbf{x}$$

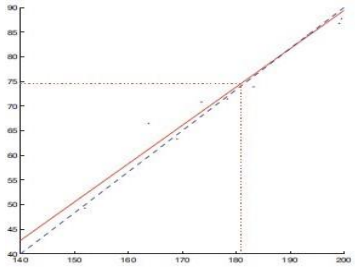
with $\mathbf{b} = (b1, \ldots, bd)$. The equation $f(\mathbf{x}) = 0$ defines a plane in R$d$ perpendicular to the *normal vector* **b**.

### 1.1 Characteristics of Linear Model:

a. Linear models are *parametric*, meaning that they have a fixed form with a small number of numeric parameters that need to be learned from data. This is different from tree or rule models, where the structure of the model (e.g., which features to use in the tree, and where) is not fixed in advance.
b. Linear models are stable, which is to say that small variations in the training data have only limited impact on the learned model. Tree models tend to vary more with the training data, as the choice of a different split at the root of the tree typically means that the rest of the tree is different as well.
c. Linear models are less likely to overfit the training data than some other models, largely because they have relatively few parameters. The flipside of this is that they sometimes lead to underfitting: e.g., imagine you are learning where the border runs between two countries from labelled samples, then a linear model is unlikely to give a good approximation.

## 2. The least-squares method:

a. The method of least squares is about estimating parameters by minimizing the squared discrepancies between observed data and their expected values on the other.
b. Considering an arbitrary straight line, $y=b_0+b_1x$ is to be fitted through these data points.



c. The Least Squares (LS) criterion states that the sum of the squares of errors is minimum. The least squares yields y(x), whose elements sum to 1, but do not ensure the outputs to be in the range [0,1].
d. Draw a line based on the data points and assume the imaginary line of y=a+bx and imagine a vertical distance between the line and a data point and calculate E=Y-E(Y), this error is the deviation of the data point from the regression line.
e. Let us get a and b that can minimizes the sum of squared deviations. This method is called "Least Squares". The process of getting parameter estimators is called "estimations". Least Square method is the estimation method of Ordinary Least Squares (OLS)

## 2.1 Univariate Analysis:

1. The simplest form of regression analysis is a univariate regression or a model with one independent variable.
2. Univariate analysis is the simplest form of data analysis where the data being analyzed contains only one variable. Since it's a single variable it does not deal with causes or relationships.
3. The main purpose of univariate analysis is to describe the data and find patterns that exist with in it.
4. Assuming a linear relationship between the independent and dependent variables, the general equation can be written as:

$$W_i=\alpha+\beta_i+€$$

We can write univariate linear regression in matrix form as

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} a + \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} b + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

5. The parameter **α** is called "intercept" or the value of W, when x=0
6. Since there is only one independent variable, regression analysis estimates parameters that provide the "best fitting" line when the dependent variable is graphed on the vertical axis and independent variable is on horizontal axis.

**Ex:** For a given data having 100 examples, if squared errors SE1,SE2, and SE3 are 13.33,3.33 and 4.00 respectively, calculate Mean Squared Error (MSE).

Sol: Mean Squared Error= $\frac{\sum SEi}{n}$

$$MSE=\frac{13.33+3.33+4}{3}=0.2066$$

## 2.2 Multivariate Regression:

1. In order to deal with an arbitrary number of features it will be useful to employ matrix Notation
2. The *multivariate regression* problem is

$$\hat{\mathbf{w}} = \mathbf{S}^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

3. Assume that the features are uncorrelated (meaning the covariance between every pair of different features is 0) in addition to being zero-centred. The covariance matrix $\boldsymbol{\Sigma}$ is diagonal with entries $\sigma_{jj}$. Since $\mathbf{X}^{\mathrm{T}}\mathbf{X} = n(\boldsymbol{\Sigma}+\mathbf{M})$, and since the entries of $\mathbf{M}$ are 0 because the columns of $\mathbf{X}$ are zero-centred, this matrix is also diagonal with entries $n\sigma_{jj}$ – in fact, it is the matrix $\mathbf{S}$ referred to above.
4. $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ *acts as a transformation that decorrelates, centres and normalises the features*.
5. If multiple independent variables affect the response variable, then the analysis calls for a model different from that used for the single predictor variable.

## 2.3 Bivariate linear regression in matrix notation:

1. we derive the basic expressions.

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{pmatrix} = n \begin{pmatrix} \sigma_{11} + \overline{x_1}^2 & \sigma_{12} + \overline{x_1}\,\overline{x_2} \\ \sigma_{12} + \overline{x_1}\,\overline{x_2} & \sigma_{22} + \overline{x_2}^2 \end{pmatrix}$$

2.

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} = \frac{1}{nD} \begin{pmatrix} \sigma_{22} + \overline{x_2}^2 & -\sigma_{12} - \overline{x_1}\,\overline{x_2} \\ -\sigma_{12} - \overline{x_1}\,\overline{x_2} & \sigma_{11} + \overline{x_1}^2 \end{pmatrix}$$

3.

$$\mathbf{X}^{\mathrm{T}}\mathbf{y} = \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = n \begin{pmatrix} \sigma_{1y} + \overline{x_1}\,\overline{y} \\ \sigma_{2y} + \overline{x_2}\,\overline{y} \end{pmatrix}$$

## 2.4 Regularized Expression:

1. *Regularisation* is a general method to avoid such overfitting by applying additional constraints to the weight vector
2. A common approach is to make sure the weights are, on average, small in magnitude: this is referred to as *shrinkage*.
3. Process to achieve:
   a. write down the least-squares regression problem as an optimisation problem:
   $$\mathbf{w}^* = \arg\min_{\mathbf{w}} (\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\mathbf{w})$$
   b. write the sum of squared residuals as a dot product. The regularised version of this optimisation is then as follows:
   $$\mathbf{w}^* = \arg\min_{\mathbf{w}} (\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\mathbf{w}) + \lambda\|\mathbf{w}\|^2$$
   where $\|\mathbf{w}\|^2 = \sum_i w_i^2$ is the squared normof the vector w, or, equivalently, the dot product $\mathbf{w}^{\mathrm{T}}\mathbf{w}$; λ is a scalar determining the amount of regularisation.
4. This regularised problem still has a closed-form solution:
   $$\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

where **I** denotes the identity matrix with 1s on the diagonal and 0s everywhere else. regularisation amounts to adding $\lambda$ to the diagonal of $\mathbf{X}^T\mathbf{X}$, a well-known trick to improve the numerical stability of matrix inversion. This form of least-squares regression is known as *ridge regression*.

5. alternative form of regularised regression is provided by the lasso, which stands for 'least absolute shrinkage and selection operator'. It replaces the ridge regularisation term $\sum_i w_i^2$ with the sum of absolute weights $\sum_i |w_i|$.
6. The result is that some weights are shrunk, but others are set to 0, and so the lasso regression favours *sparse solutions*.
7. It should be added that lasso regression is quite sensitive to the regularisation parameter $\lambda$, which is usually set on hold-out data or in cross-validation.
8. Also, there is no closed-form solution and so some numerical optimisation technique must be applied.

***Using least-squares regression for classification***

1. we can also use linear regression to learn a binary classifier by encoding the two classes as real numbers.
2. For instance, we can label the *Pos* positive examples with $y\oplus = +1$ and the *Neg* negative examples with $y\_ = -1$. It then follows that $\mathbf{X}^T\mathbf{y} = Pos\ \boldsymbol{\mu}\oplus - Neg\ \boldsymbol{\mu}\_$, where $\boldsymbol{\mu}\oplus$ and $\boldsymbol{\mu}\_$ are *d*-vectors containing each feature's mean values for the positive and negative examples, respectively.
3. In the general case, the *least-squares classifier* learns the decision boundary $\mathbf{w}\cdot\mathbf{x} = t$ with

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}(Pos\,\mu^{\oplus} - Neg\,\mu^{\ominus})$$

4. We would hence assign class $\hat{y} = \text{sign}(w\cdot x - t)$ to instance x, where

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Various simplifying assumptions can be made, including zero-centred features, equal variance features, uncorrelated features and equal class prevalences.

5. A general way of constructing a linear classifier with decision boundary $w\cdot x = t$ is by constructing w as $M^{-1}(n\oplus\mu\oplus - n\,\mu)$, with different possible choices of M, $n\oplus$ and n.
6. The full covariance approach with $M = \mathbf{X}^T\mathbf{X}$ has time complexity $O(n^2d)$ for construction of M and $O(d^3)$ for inverting it,1 so this approach becomes unfeasible with large numbers of features.

## 3. The Perceptron:

1. A linear classifier that will achieve perfect separation on linearly separable data is the *perceptron*, originally proposed as a simple neural network.
2. The perceptron iterates over the training set, updating the weight vector every time it encounters an incorrectly classified example.

3. For example, let $\mathbf{x}i$ be a misclassified positive example, then we have $yi = +1$ and $\mathbf{w}\cdot\mathbf{x}i < t$ . We therefore want to find $\mathbf{w}$ such that $\mathbf{w}\cdot\mathbf{x}i > \mathbf{w}\cdot\mathbf{x}i$ , which moves the decision boundary towards and hopefully past $xi$ . This can be achieved by calculating the new weight vector as $\mathbf{w} = \mathbf{w}+\eta\mathbf{x}i$ , where $0 < \eta \leq 1$ is the *learning rate*.
4. It iterates through the training examples until all examples are correctly classified. The algorithm can easily be turned into an *online* algorithm that processes a stream of examples, updating the weight vector only if the last received example is misclassified.
5. The perceptron is guaranteed to converge to a solution if the training data is linearly separable, but it won't converge otherwise.
6. The key point of the perceptron algorithm is that, every time an example $\mathbf{x}i$ is misclassified, we add $yi\ \mathbf{x}i$ to the weight vector.
7. After training has completed, each example has been misclassified zero or more times – denote this number $\alpha i$ for example $\mathbf{x}i$ .

**Algorithm: Perceptron($D,\eta$) – train a perceptron for linear classification.**

**Input** : labelled training data $D$ in homogeneous coordinates; learning rate $\eta$.
**Output** : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w}\cdot\mathbf{x})$.

```
w←0 ; // Other initialisations of the weight vector are possible
converged←false;
while converged = false do
    converged←true;
    for i = 1 to |D| do
        if yiw· xi ≤ 0 // i.e., ŷi = yi
        then
            w←w+η yi xi ;
            converged←false; //We changed w so haven't converged yet
        end
    end
end
```

## 4. Support vector machines:

- The training examples nearest to the decision boundary are called support vectors: the decision boundary of a support vector machine (SVM) is defined as a linear combination of the support vectors.
- The margin is thus defined as m/||w||, where m is the distance between the decision boundary and the nearest training instances (at least one of each class) as measured along w. Since we are free to rescale t , ||w|| and m, it is customary to choose m = 1.
- Maximising the margin then corresponds to minimising ||w|| or, more conveniently, ½ $||w||^2$, provided of course that none of the training points fall inside the margin. This leads to a quadratic, constrained optimisation problem:

$$\mathbf{w}^*, t^* = \arg\min_{\mathbf{w},t} \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1, 1 \leq i \leq n$$

- Adding the constraints with multipliers$\alpha i$ for each training example gives the Lagrange Function

$$\begin{aligned}\Lambda(\mathbf{w},t,\alpha_1,\ldots,\alpha_n) &= \frac{1}{2}||\mathbf{w}||^2-\sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i-t)-1)\\ &= \frac{1}{2}||\mathbf{w}||^2-\sum_{i=1}^{n}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i)+\sum_{i=1}^{n}\alpha_i y_i t+\sum_{i=1}^{n}\alpha_i\\ &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w}-\mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)+t\left(\sum_{i=1}^{n}\alpha_i y_i\right)+\sum_{i=1}^{n}\alpha_i\end{aligned}$$

- By taking the partial derivative of the Lagrange function with respect to *t* and setting it to 0 we find that for the optimal threshold *t* we have $\sum_{i=1}^{n}\alpha_i y_i = 0.$ .
- Similarly, by taking the partial derivative of the Lagrange function with respect to **w** we see that the Lagrange multipliers define the weight vector as a linear combination of the training examples:

$$\frac{\partial}{\partial\mathbf{w}}\Lambda(\mathbf{w},t,\alpha_1,\ldots,\alpha_n)=\frac{\partial}{\partial\mathbf{w}}\frac{1}{2}\mathbf{w}\cdot\mathbf{w}-\frac{\partial}{\partial\mathbf{w}}\mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)=\mathbf{w}-\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i$$

- Now, by plugging the expressions $\{\sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } \mathbf{w}=\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i$ back into the Lagrangian we are able to eliminate **w** and *t* , and hence obtain the dual optimisation problem, which is entirely formulated in terms of the Lagrange multipliers:

$$\begin{aligned}\Lambda(\alpha_1,\ldots,\alpha_n) &= -\frac{1}{2}\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)+\sum_{i=1}^{n}\alpha_i\\ &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j+\sum_{i=1}^{n}\alpha_i\end{aligned}$$

- The dual problem is to maximise this function under positivity constraints and one equality constraint:

$$\alpha_1^*,\ldots,\alpha_n^*=\underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}}-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j+\sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } \alpha_i\geq 0, 1\leq i\leq n \text{ and } \sum_{i=1}^{n}\alpha_i y_i=0$$

The following example makes these issues a bit more concrete by showing detailed calculations on some toy data.

**4.1Two maximum-margin classifiers and their support vectors:** Let the data points and labels be as follows:

$$\mathbf{X}=\begin{pmatrix}1 & 2\\ -1 & 2\\ -1 & -2\end{pmatrix}\qquad \mathbf{y}=\begin{pmatrix}-1\\ -1\\ +1\end{pmatrix}\qquad \mathbf{X}'=\begin{pmatrix}-1 & -2\\ 1 & -2\\ -1 & -2\end{pmatrix}$$

The matrix **X** on the right incorporates the class labels; i.e., the rows are *yi* **x***i* . The Gram matrix is (without and with class labels):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}}=\begin{pmatrix}5 & 3 & -5\\ 3 & 5 & -3\\ -5 & -3 & 5\end{pmatrix}\qquad \mathbf{X}'\mathbf{X}'^{\mathrm{T}}=\begin{pmatrix}5 & 3 & 5\\ 3 & 5 & 3\\ 5 & 3 & 5\end{pmatrix}$$

The dual optimisation problem is thus

$$\begin{aligned}\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(5\alpha_1^2+3\alpha_1\alpha_2+5\alpha_1\alpha_3+3\alpha_2\alpha_1+5\alpha_2^2+3\alpha_2\alpha_3+5\alpha_3\alpha_1\right.\\ \left.+3\alpha_3\alpha_2+5\alpha_3^2\right)+\alpha_1+\alpha_2+\alpha_3\\ =\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(5\alpha_1^2+6\alpha_1\alpha_2+10\alpha_1\alpha_3+5\alpha_2^2+6\alpha_2\alpha_3+5\alpha_3^2\right)+\alpha_1+\alpha_2+\alpha_3\end{aligned}$$

subject to $\alpha 1 \geq 0, \alpha 2 \geq 0, \alpha 3 \geq 0$ and $-\alpha 1 - \alpha 2 + \alpha 3 = 0$.

Using the equality constraint we can eliminate one of the variables, say $\alpha 3$, and simplify the objective function to

$$\begin{aligned}\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(5\alpha_1^2+6\alpha_1\alpha_2+10\alpha_1(\alpha_1+\alpha_2)+5\alpha_2^2+6\alpha_2(\alpha_1+\alpha_2)+5(\alpha_1+\alpha_2)^2\right)\\ +2\alpha_1+2\alpha_2\\ =\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(20\alpha_1^2+32\alpha_1\alpha_2+16\alpha_2^2\right)+2\alpha_1+2\alpha_2\end{aligned}$$

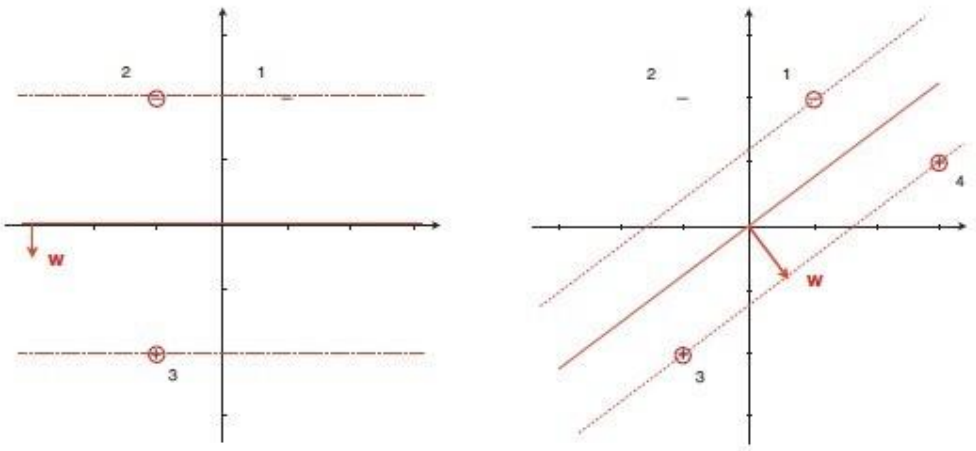



**Figure 7.8.** **(left)** A maximum-margin classifier built from three examples, with $\mathbf{w} = (0, -1/2)$ and margin 2. The circled examples are the support vectors: they receive non-zero Lagrange multipliers and define the decision boundary. **(right)** By adding a second positive the decision boundary is rotated to $\mathbf{w} = (3/5, -4/5)$ and the margin decreases to 1.

**Algorithm 7.3:** PerceptronRegression($D, T$) – train a perceptron for regression.

**Input** : labelled training data $D$ in homogeneous coordinates;
maximum number of training epochs $T$.
**Output** : weight vector $\mathbf{w}$ defining function approximator $\hat{y} = \mathbf{w} \cdot \mathbf{x}$.

```
w ← 0; t ← 0;
while t < T do
    for i = 1 to |D| do
        w ← w + (y_i − ŷ_i)² x_i;
    end
    t ← t + 1;
end
```

## 5. Soft margin SVM

1. If the data is not linearly separable, then the constraints $\mathbf{w} \cdot \mathbf{x}i - t \geq 1$ posed by the examples are not jointly satisfiable. However, there is a very elegant way of adapting the optimisation problem such that it admits a solution even in this case
2. The idea is to introduce slack variables ξi , one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary – we will call these margin errors. Thus, we change the constraints to w· xi −t ≥ 1−ξi and add the sum of all slack variables to the objective function to be minimised, resulting in the following *soft margin* optimisation problem:

$$\mathbf{w}^*, t^*, \xi_i^* = \arg\min_{\mathbf{w},t,\xi_i} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1-\xi_i \text{ and } \xi_i \geq 0, 1 \leq i \leq n$$

*C* is a user-defined parameter trading off margin maximisation against slack variable minimisation: a high value of *C* means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin.

3. If we allow more margin errors we need fewer support vectors, hence *C* controls to some extent the 'complexity' of the SVM and hence is often referred to as the *complexity parameter*. It can be seen as a form of regularisation similar to that discussed in the context of least-squares regression.
4. The Lagrange function is then as follows:

$$\begin{aligned}\Lambda(\mathbf{w}, t, \xi_i, \alpha_i, \beta_i) &= \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - t) - (1-\xi_i)) - \sum_{i=1}^{n}\beta_i\xi_i \\ &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i\right) + t\left(\sum_{i=1}^{n}\alpha_i y_i\right) + \sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}(C-\alpha_i-\beta_i)\xi_i \\ &= \Lambda(\mathbf{w}, t, \alpha_i) + \sum_{i=1}^{n}(C-\alpha_i-\beta_i)\xi_i\end{aligned}$$

5. For an optimal solution every partial derivative with respect to *ξi* should be 0, from which it follows that $C - \alpha i - \beta i = 0$ for all $i$ , and hence the added term vanishes from the dual problem

## 6. Obtaining probabilities from linear classifiers:

1. A linear classifier produces scores $\hat{s}(x_i) = w \cdot x_i - t$ that are thresholded at 0 in order to classify examples. Owing to the geometric nature of linear classifiers, such scores can be used to obtain the (signed) distance of $x_i$ from the decision boundary
2. The length of the projection of $\mathbf{x}_i$ onto $\mathbf{w}$ is $||\mathbf{x}_i|| \cos\theta$, where $\theta$ is the angle between $\mathbf{x}_i$ and $\mathbf{w}$. Since $\mathbf{w} \cdot \mathbf{x}_i = ||\mathbf{w}|| \, ||\mathbf{x}_i|| \cos\theta$, we can write this length as $(\mathbf{w} \cdot \mathbf{x}_i)/||\mathbf{w}||$. This gives the following signed distance:

$$d(\mathbf{x}_i) = \frac{\hat{s}(\mathbf{x}_i)}{||\mathbf{w}||} = \frac{\mathbf{w} \cdot \mathbf{x}_i - t}{||\mathbf{w}||} = \mathbf{w}' \cdot \mathbf{x}_i - t'$$

   with $\mathbf{w} = \mathbf{w}/||\mathbf{w}||$ rescaled to unit length and $t = t/||\mathbf{w}||$ the corresponding rescaled intercept. The sign of this quantity tells us which side of the decision boundary we are on: positive distances for points on the 'positive' side of the decision boundary
3. This geometric interpretation of the scores produced by linear classifiers offers an interesting possibility for turning them into probabilities, a process that was called *calibration*

Example: Suppose we now observe a point $\mathbf{x}$ with distance $d(\mathbf{x})$. We classify this point as positive if $d(\mathbf{x}) > 0$ and as negative if $d(\mathbf{x}) < 0$, but we want to attach a probability $\hat{p}(\mathbf{x}) = P(\oplus|d(\mathbf{x}))$ to these predictions. Using Bayes' rule we obtain

$$P(\oplus|d(\mathbf{x})) = \frac{P(d(\mathbf{x})|\oplus)P(\oplus)}{P(d(\mathbf{x})|\oplus)P(\oplus) + P(d(\mathbf{x})|\ominus)P(\ominus)} = \frac{LR}{LR + 1/clr}$$

where $LR$ is the likelihood ratio obtained from the normal score distributions, and $clr$ is the class ratio. We will assume for simplicity that $clr = 1$ in the derivation below. Furthermore, assume for now that $\sigma^2 = 1$ and $\bar{d}^{\oplus} = -\bar{d}^{\ominus} = 1/2$ (we will relax this in a moment). We then have

$$LR = \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \frac{\exp\left(-(d(\mathbf{x}) - 1/2)^2/2\right)}{\exp\left(-(d(\mathbf{x}) + 1/2)^2/2\right)}$$
$$= \exp\left(-(d(\mathbf{x}) - 1/2)^2/2 + (d(\mathbf{x}) + 1/2)^2/2\right) = \exp(d(\mathbf{x}))$$

and so

$$P(\oplus|d(\mathbf{x})) = \frac{\exp(d(\mathbf{x}))}{\exp(d(\mathbf{x})) + 1} = \frac{\exp(\mathbf{w} \cdot \mathbf{x} - t)}{\exp(\mathbf{w} \cdot \mathbf{x} - t) + 1}$$

So, in order to obtain probability estimates from a linear classifier outputting distance scores d, we convert d into a probability by means of the mapping $d \mapsto \exp(d)/\exp(d)+1$ (or, equivalently, $d \mapsto 1/1+\exp(-d)$ ). This S-shaped or sigmoid function is called the logistic function; it finds applications in a wide range of areas

Suppose now that $\bar{d}^{\oplus} = -\bar{d}^{\ominus}$ as before, but we do not assume anything about the magnitude of these mean distances or of $\sigma 2$. In this case we have

$$LR = \exp\left(\frac{-(d(\mathbf{x}) - \overline{d}^{\oplus})^2 + (d(\mathbf{x}) - \overline{d}^{\ominus})^2}{2\sigma^2}\right)$$
$$= \exp\left(\frac{2\overline{d}^{\oplus} d(\mathbf{x}) - \left(\overline{d}^{\oplus}\right)^2 - 2\overline{d}^{\ominus} d(\mathbf{x}) + \left(\overline{d}^{\ominus}\right)^2}{2\sigma^2}\right) = \exp\left(\gamma d(\mathbf{x})\right)$$

with $a = (\overline{d}^{\oplus} - \overline{d}^{\ominus})/\sigma^2$ a scaling factor that rescales the weight vector so that the mean distances per class are one unit of variance apart. In other words, by taking the scaling factor $\gamma$ into account, we can drop our assumption that $\mathbf{w}$ is a unit vector.

If we also drop the assumption $\overline{d}^{\oplus}$ and $\overline{d}^{\ominus}$ that are symmetric around the decision boundary, then we obtain the most general form

$$LR = \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \exp\left(\gamma(d(\mathbf{x}) - d_0)\right)$$
$$\gamma = \frac{\overline{d}^{\oplus} - \overline{d}^{\ominus}}{\sigma^2} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{\sigma^2}, \quad d_0 = \frac{\overline{d}^{\oplus} + \overline{d}^{\ominus}}{2} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})}{2} - t$$

$d_0$ has the effect of moving the decision boundary from $\mathbf{w}\cdot\mathbf{x} = t$ to $\mathbf{x} = (\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$, that is, halfway between the two class means. The logistic mapping thus becomes $d \mapsto \frac{1}{1+\exp(-\gamma(d-d_0))}$, and the effect of the two parameters is visualised in Figure 7.11.

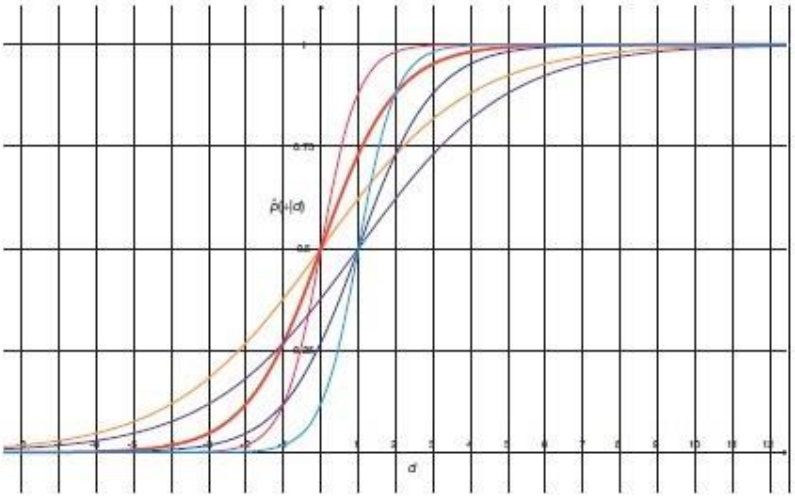


**Figure 7.11.** The logistic function, a useful function for mapping distances from a linear decision boundary into an estimate of the positive posterior probability. The fat red line indicates the standard logistic function $\hat{p}(d) = \frac{1}{1+\exp(-d)}$; this function can be used to obtain probability estimates if the two classes are equally prevalent and the class means are equidistant from the decision boundary and one unit of variance apart. The steeper and flatter red lines show how the function changes if the class means are 2 and 1/2 units of variance apart, respectively. The three blue lines show how these curves change if $d_0 = 1$, which means that the positives are on average further away from the decision boundary.

### 5.1 Example: Logistic calibration of a linear classifier:

Logistic calibration has a particularly simple form for the basic linear classifier, which has $\mathbf{w} = \boldsymbol{\mu}\oplus - \boldsymbol{\mu}\_$. It follows that

$$\overline{d}^{\oplus} - \overline{d}^{\ominus} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{\|\mathbf{w}\|} = \frac{\|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|^2}{\|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|} = \|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|$$

and hence $\gamma = \|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|/\sigma^2$. Furthermore, $d_0 = 0$ as $(\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$ is already on the decision boundary. So in this case logistic calibration does not move the decision boundary, and only adjusts the steepness of the sigmoid according to the separation of the classes.

## 7. Going beyond linearity with kernel methods:

It is customary to call the transformed space the *feature space* and the original space the *input space*. The approach thus appears to be to transform the training data to feature space and learn a model there. In order to classify new data we transform that to feature space as well and apply the model.

**Process**: Take the perceptron algorithm in dual form. The algorithm is a simple counting algorithm – the only operation that is somewhat involved is testing whether example **x*i*** is correctly classified by evaluating $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot$

$\mathbf{x}_j$. The key component of this calculation is the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$. Assuming bivariate examples $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}_j = (x_j, y_j)$ for notational simplicity, the dot product can be written as $\mathbf{x}_i \cdot \mathbf{x}_j = x_i x_j + y_i y_j$. The corresponding instances in the quadratic feature space are $(x_i^2, y_i^2)$ and $(x_j^2, y_j^2)$, and their dot product is

$$(x_i^2, y_i^2) \cdot (x_j^2, y_j^2) = x_i^2 x_j^2 + y_i^2 y_j^2$$

This is almost equal to

$$(\mathbf{x}_i \cdot \mathbf{x}_j)^2 = (x_i x_j + y_i y_j)^2 = (x_i x_j)^2 + (y_i y_j)^2 + 2 x_i x_j y_i y_j$$

but not quite because of the third term of cross-products. We can capture this term by extending the feature vector with a third feature $\sqrt{2}xy$. This gives the following feature space:

$$\phi(\mathbf{x}_i) = (x_i^2, y_i^2, \sqrt{2}x_i y_i) \qquad \phi(\mathbf{x}_j) = (x_j^2, y_j^2, \sqrt{2}x_j y_j)$$
$$\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = x_i^2 x_j^2 + y_i^2 y_j^2 + 2 x_i x_j y_i y_j = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

We now define $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$, and replace $\mathbf{x}_i \cdot \mathbf{x}_j$ with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ in the dual perceptron algorithm to obtain the *kernel perceptron* (Algorithm 7.4), which is able to learn the kind of non-linear decision boundaries illustrated in Example 7.8.

The introduction of kernels opens up a whole range of possibilities. Clearly we can define a polynomial kernel of any degree $p$ as $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p$. This transforms

a *d*-dimensional input space into a high-dimensional feature space, such that each new feature is a product of *p* terms (possibly repeated). If we include a constant, say $\kappa(\mathbf{x}i, \mathbf{x}j) = (\mathbf{x}i \cdot \mathbf{x}j + 1)p$, we would get all lower-order terms as well. So, for example, in a bivariate input space and setting $p = 2$ the resulting feature space is

$$\phi(\mathbf{x}) = \left(x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1\right)$$

with linear as well as quadratic features. But we are not restricted to polynomial kernels. An often-used kernel is the *Gaussian kernel*, defined as

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right)$$

where $\sigma$ is a parameter known as the *bandwidth*. To understand the Gaussian kernel a bit better, notice that $\kappa(\mathbf{x},\mathbf{x}) = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}) = ||\varphi(\mathbf{x})||2$ for any kernel obeying a number of standard properties referred to as 'positive semi-definiteness'.

**Algorithm 7.4:** KernelPerceptron($D, \kappa$) – perceptron training algorithm using a kernel.

**Input** : labelled training data $D$ in homogeneous coordinates;
kernel function $\kappa$.
**Output** : coefficients $\alpha_i$ defining non-linear decision boundary.

```
1  α_i ← 0 for 1 ≤ i ≤ |D|;
2  converged ← false;
3  while converged = false do
4      converged ← true;
5      for i = 1 to |D| do
6          if y_i Σ_{j=1}^{|D|} α_j y_j κ(x_i, x_j) ≤ 0 then
7              α_i ← α_i + 1;
8              converged ← false;
9          end
10     end
11 end
```

Kernel methods are best known in combination with support vector machines. Notice that the soft margin optimisation problem is defined in terms of dot products between training instances and hence the 'kernel trick' can be applied:

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1,\ldots,\alpha_n}{\arg\max} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } 0 \le \alpha_i \le C \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

# Probabilistic Models

## 1. Introduction:

1. Probability estimation tree attaches a class probability distribution to each leaf of the tree, and each instance that gets filtered down to a particular leaf in a tree model is labelled with that particular class distribution.
2. Similarly, a calibrated linear model translates the distance from the decision boundary into a class probability. These are examples of what are called discriminative probabilistic models. They model the posterior probability distribution P(Y |X), where Y is the target variable and X are the features. That is, given X they return a probability distribution over Y.
3. A. The other main class of probabilistic models are called generative models. They model the joint distribution P(Y,X) of the target Y and the feature vector X. Once we have access to this joint distribution we can derive any conditional or marginal distribution involving the same variables.

   B. In particular, since $P(X) = \sum_y P(Y = y, X)$ follows that the posterior distribution can be obtained as

$$P(Y|X) = \frac{P(Y,X)}{\sum_y P(Y = y, X)}$$

   C. Alternatively, generative models can be described by the likelihood function P(X|Y),since P(Y ,X) = P(X|Y )P(Y ) and the target or prior distribution can be easily estimated or postulated. Such models are called 'generative' because we can sample from the joint distribution to obtain new data points together with their labels.

   D. Alternatively, we can use P(Y ) to sample a class and P(X|Y ) to sample an instance for that class.

   E. In contrast, a discriminative model such as a probability estimation tree or a linear classifier models P(Y |X) but not P(X), and hence can be used to label data but not generate it.

4. One of the most attractive features of the probabilistic perspective is that it allow us to view learning as a process of reducing uncertainty. For instance, a uniform class prior tells us that, before knowing anything about the instance to be classified, we are maximally uncertain about which class to assign. If the posterior distribution after observing the instance is less uniform, we have reduced our uncertainty in favour of one class or the other.

5. The key point is that probabilities do not have to be interpreted as estimates of relative frequencies, but can carry the more general meaning of (possibly subjective) degrees of belief. we can attach a probability distribution to almost anything: not just features and targets, but also model parameters and even models.

6. An important concept related to probabilistic models is Bayes-optimality. A classifier is Bayes-optimal if it always assigns $\text{argmax}_y P*(Y = y|X = x)$ to an instance x, where $P*$ denotes the true posterior distribution.

Ex: we can perform experiments with artificially generated data for which we have chosen the true distribution ourselves: this allows us to experimentally evaluate how close the performance of a model is to being Bayes-optimal.

**Note:** The derivation of a probabilistic learning method usually makes certain assumptions about the true distribution, which allows us to prove theoretically that the model will be Bayes-optimal provided these assumptions are met.

## 2. The normal distribution and its geometric interpretations:

**1.** We can draw a connection between probabilistic and geometric models by considering probability distributions defined over Euclidean spaces. The most common such distributions are normal distributions, also called **"Gaussians".**

2. We start by considering the univariate, two-class case. Suppose the values of $x \in R$ follow a mixture model: i.e., each class has its own probability distribution (a component of the mixture model).

3. We will assume a Gaussian mixture model, which means that the components of the mixture are both Gaussians. We thus have

$$P(x|\oplus) = \frac{1}{\sqrt{2\pi}\sigma^{\oplus}} \exp\left(-\frac{1}{2}\left[\frac{x-\mu^{\oplus}}{\sigma^{\oplus}}\right]^2\right) \qquad P(x|\ominus) = \frac{1}{\sqrt{2\pi}\sigma^{\ominus}} \exp\left(-\frac{1}{2}\left[\frac{x-\mu^{\ominus}}{\sigma^{\ominus}}\right]^2\right)$$

Where,

$\mu\oplus$ and $\sigma\oplus$ are the mean and standard deviation for the positive class,

$\mu$ - and $\sigma$- are the mean and standard deviation for the negative class.

4. This gives the following likelihood ratio:

$$\mathrm{LR}(x) = \frac{P(x|\oplus)}{P(x|\ominus)} = \frac{\sigma^{\ominus}}{\sigma^{\oplus}} \exp\left(-\frac{1}{2}\left[\left(\frac{x-\mu^{\oplus}}{\sigma^{\oplus}}\right)^2 - \left(\frac{x-\mu^{\ominus}}{\sigma^{\ominus}}\right)^2\right]\right)$$

**Points to Remember:**

a. The normal distribution describes a special class of such distribution that are symmetric and can be described by the distribution mean, $\mu$ and the standard deviation, $\sigma$

b. A continuous random variable is said to be normally distributed with mean and variance if its probability density function is:

$$P(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) = \frac{1}{E} \exp\left(-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2\right) = \frac{1}{E} \exp\left(-z^2/2\right), \quad E = \sqrt{2\pi}\sigma$$

c. The distribution has two parameters: $\mu$, which is the mean or expected value, as well as the median (i.e., the point where the area under the density function is split in half) and the mode (i.e., the point where the density function reaches its maximum); and $\sigma$, which is the standard deviation and determines the width of the bell-shaped curve.

d. $z = (x - \mu)/\sigma$ is the z-score associated with x; it measures the number of standard deviations between x and the mean (it has itself mean 0 and standard deviation 1). It follows that $P(x|\mu,\sigma) = 1\sigma P(z|0, 1)$, where $P(z|0,1)$ denotes the standard normal distribution.

e. The multivariate normal distribution over d-vectors $x = (x1, \ldots, xd)^T \in R^d$ is

$$P(\mathbf{x}|\mu,\Sigma) = \frac{1}{E_d}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right), \quad E_d = (2\pi)^{d/2}\sqrt{|\Sigma|}$$

f. Lets first consider the case that both components have the same standard deviation, i.e., $\sigma \oplus = \sigma\_ = \sigma$.

$$\begin{aligned} -\frac{1}{2\sigma^2}\left[(x-\mu^{\oplus})^2-(x-\mu^{\ominus})^2\right] &= -\frac{1}{2\sigma^2}\left[x^2-2\mu^{\oplus}x+\mu^{\oplus 2}-(x^2-2\mu^{\ominus}x+\mu^{\ominus 2})\right] \\ &= -\frac{1}{2\sigma^2}\left[-2(\mu^{\oplus}-\mu^{\ominus})x+(\mu^{\oplus 2}-\mu^{\ominus 2})\right] \\ &= \frac{\mu^{\oplus}-\mu^{\ominus}}{\sigma^2}\left[x-\frac{\mu^{\oplus}+\mu^{\ominus}}{2}\right] \end{aligned}$$

g. The general form of the likelihood ratio can be derived from

$$\mathrm{LR}(\mathbf{x}) = \sqrt{\frac{|\Sigma^{\ominus}|}{|\Sigma^{\oplus}|}}\exp\left(-\frac{1}{2}\left[(\mathbf{x}-\mu^{\oplus})^T(\Sigma^{\oplus})^{-1}(\mathbf{x}-\mu^{\oplus})-(\mathbf{x}-\mu^{\ominus})^T(\Sigma^{\ominus})^{-1}(\mathbf{x}-\mu^{\ominus})\right]\right)$$

where $\mu\oplus$ and $\mu\_$ are the class means, and $\Sigma\oplus$ and $\Sigma\_$ are the covariance matrices for each class.

The ML decision boundary is a straight line at equal distances from the class means – in which we recognise our old friend, the basic linear classifier! In other words, for uncorrelated, unit-variance Gaussian features, the basic linear classifier is Bayes-optimal.

4. The multivariate normal distribution essentially translates distances into probabilities. This becomes obvious when we plug the definition of Mahalanobis distance

$$P(\mathbf{x}|\mu,\Sigma) = \frac{1}{E_d}\exp\left(-\frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\mu|\Sigma)\right)^2\right)$$

5. Similarly, the standard normal distribution translates Euclidean distances into probabilities:

$$P(\mathbf{x}|\mathbf{0},\mathbf{I}) = \frac{1}{(2\pi)^{d/2}}\exp\left(-\frac{1}{2}\left(\mathrm{Dis}_2(\mathbf{x},\mathbf{0})\right)^2\right)$$

6. Conversely, we see that the negative logarithm of the Gaussian likelihood can be interpreted as a squared distance:

$$-\ln P(\mathbf{x}|\mu,\Sigma) = \ln E_d + \frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\mu|\Sigma)\right)^2$$

**Case Study1:** suppose we want to estimate the mean μ of a multivariate Gaussian distribution with given covariance matrix Σ from a set of data points X. The principle of maximum-likelihood estimation states that we should find the value of μ that maximises the joint likelihood of X.

1. Assuming that the elements of X were independently sampled, the joint likelihood decomposes into a product over the individual data points in X, and the maximum-likelihood estimate can be found as follows:

$$\begin{aligned}\hat{\mu} &= \arg\max_{\mu} \prod_{\mathbf{x}\in X} P(\mathbf{x}|\mu,\boldsymbol{\Sigma}) \\ &= \arg\max_{\mu} \prod_{\mathbf{x}\in X} \frac{1}{E_d} \exp\left(-\frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\mu|\boldsymbol{\Sigma})\right)^2\right) \\ &= \arg\min_{\mu} \sum_{\mathbf{x}\in X} \left[\ln E_d + \frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\mu|\boldsymbol{\Sigma})\right)^2\right] \\ &= \arg\min_{\mu} \sum_{\mathbf{x}\in X} \left(\mathrm{Dis}_M(\mathbf{x},\mu|\boldsymbol{\Sigma})\right)^2\end{aligned}$$

2. We thus find that the maximum-likelihood estimate of the mean of a multivariate distribution is the point that minimises the total squared Mahalanobis distance to all points in X.

**Case Study2: (least-squares solution to a linear regression problem):**

1. The starting point is the assumption that our training examples ($h_i$,$y_i$) are noisy measurements of true function points ($x_i$ , f ($x_i$ )): i.e., $y_i$ = f ($x_i$ )+€$_i$ , where the €i are independently and identically distributed errors.

2. We want to derive the maximum-likelihood estimates ˆ $y_i$ of f ($x_i$). We can derive this if we assume a particular noise distribution, for example Gaussian with variance $\sigma^2$. It then follows that each $y_i$ is normally distributed with mean a +b$x_i$ and variance $\sigma^2$, and thus

$$P(y_i|a,b,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i-(a+bx_i)\right)^2}{2\sigma^2}\right)$$

3. Taking the partial derivatives with respect to a, b and σ2 and setting to zero in order to maximise the negative log likelihood gives the following three equations:

$$\begin{aligned}&\sum_{i=1}^{n} y_i-(a+bx_i)=0 \\ &\sum_{i=1}^{n} \left(y_i-(a+bx_i)\right)x_i=0 \\ &\frac{n}{2}\frac{1}{\sigma^2}-\frac{\sum_{i=1}^{n}\left(y_i-(a+bx_i)\right)^2}{2(\sigma^2)^2}=0\end{aligned}$$

A good probabilistic treatment of a machine learning problem achieves a balance between solid theoretical foundations and the pragmatism required to obtain a workable solution.

### 3. Probabilistic models for categorical data:

1. Categorical variables or features (also called discrete or nominal) are ubiquitous in machine learning.

2. The most common form of the Bernoulli distribution models whether or not a word occurs in a document. That is, for the i -th word in our vocabulary we have a random variable Xi governed by a Bernoulli distribution. The joint distribution over the bit vector X = (X1, . . . ,Xk ) is called a multivariate Bernoulli distribution.

3. Variables with more than two outcomes are also common: for example, every word position in an e-mail corresponds to a categorical variable with k outcomes, where k is the size of the vocabulary.

4. The multinomial distribution manifests itself as a count vector: a histogram of the number of occurrences of all vocabulary words in a document.

5. This establishes an alternative way of modelling text documents that allows the number of occurrences of a word to influence the classification of a document.

6. In the multinomial document model, this follows from the very use of the multinomial distribution, which assumes that words at different word positions are drawn independently from the same categorical distribution.

7. In the multivariate Bernoulli model we assume that the bits in a bit vector are statistically independent, which allows us to compute the joint probability of a particular bit vector (x1, . . . ,xk ) as the product of the probabilities of each component P(Xi = xi ).

### Using a naive Bayes model for classification:

The more different these two distributions are, the more useful the features X are for classification. Thus, for a specific e-mail x we calculate both P(X = x|Y = spam) and P(X = x|Y = ham), and apply one of several possible decision rules:

| | |
|---|---|
| maximum likelihood (ML) | – predict $\operatorname{argmax}_y P(X = x \mid Y = y)$; |
| maximum a posteriori (MAP) | – predict $\operatorname{argmax}_y P(X = x \mid Y = y)P(Y = y)$; |
| recalibrated likelihood | – predict $\operatorname{argmax}_y w_y P(X = x \mid Y = y)$. |

**Example 9.4 (Prediction using a naive Bayes model).** Suppose our vocabulary contains three words $a$, $b$ and $c$, and we use a multivariate Bernoulli model for our e-mails, with parameters

$$\theta^{\oplus} = (0.5, 0.67, 0.33) \qquad \theta^{\ominus} = (0.67, 0.33, 0.33)$$

This means, for example, that the presence of $b$ is twice as likely in spam (+), compared with ham.

The e-mail to be classified contains words $a$ and $b$ but not $c$, and hence is described by the bit vector $\mathbf{x} = (1, 1, 0)$. We obtain likelihoods

$$P(\mathbf{x}|\oplus) = 0.5 \cdot 0.67 \cdot (1 - 0.33) = 0.222 \qquad P(\mathbf{x}|\ominus) = 0.67 \cdot 0.33 \cdot (1 - 0.33) = 0.148$$

The ML classification of $\mathbf{x}$ is thus spam. In the case of two classes it is often convenient to work with likelihood ratios and odds. The likelihood ratio can be calculated as $\frac{P(\mathbf{x}|\oplus)}{P(\mathbf{x}|\ominus)} = \frac{0.5}{0.67}\frac{0.67}{0.33}\frac{1-0.33}{1-0.33} = 3/2 > 1$. This means that the MAP classification of $\mathbf{x}$ is also spam if the prior odds are more than 2/3, but ham if they are less than that. For example, with 33% spam and 67% ham the prior odds are $\frac{P(\oplus)}{P(\ominus)} = \frac{0.33}{0.67} = 1/2$, resulting in a posterior odds of $\frac{P(\oplus|\mathbf{x})}{P(\ominus|\mathbf{x})} = \frac{P(\mathbf{x}|\oplus)}{P(\mathbf{x}|\ominus)}\frac{P(\oplus)}{P(\ominus)} = 3/2 \cdot 1/2 = 3/4 < 1$. In this case the likelihood ratio for $\mathbf{x}$ is not strong enough to push the decision away from the prior.

Alternatively, we can employ a multinomial model. The parameters of a multinomial establish a distribution over the words in the vocabulary, say

$$\theta^{\oplus} = (0.3, 0.5, 0.2) \qquad \theta^{\ominus} = (0.6, 0.2, 0.2)$$

The e-mail to be classified contains three occurrences of word $a$, one single occurrence of word $b$ and no occurrences of word $c$, and hence is described by the count vector $\mathbf{x} = (3, 1, 0)$. The total number of vocabulary word occurrences is $n = 4$. We obtain likelihoods

$$P(\mathbf{x}|\oplus) = 4!\frac{0.3^3}{3!}\frac{0.5^1}{1!}\frac{0.2^0}{0!} = 0.054 \qquad P(\mathbf{x}|\ominus) = 4!\frac{0.6^3}{3!}\frac{0.2^1}{1!}\frac{0.2^0}{0!} = 0.1728$$

The likelihood ratio is $\left(\frac{0.3}{0.6}\right)^3\left(\frac{0.5}{0.2}\right)^1\left(\frac{0.2}{0.2}\right)^0 = 5/16$. The ML classification of $\mathbf{x}$ is thus ham, the opposite of the multivariate Bernoulli model. This is mainly because of the three occurrences of word $a$, which provide strong evidence for ham.

Observation:

Notice how the likelihood ratio for the multivariate Bernoulli model is a product of factors $\theta_i^{\oplus}/\theta_i^{\ominus}$ if $x_i = 1$ in the bit vector to be classified, and $(1-\theta_i^{\oplus})/(1-\theta_i^{\ominus})$ if $x_i = 0$.

- One consequence of this is that the multinomial model only takes the presence of words into account, whereas in the multivariate Bernoulli model absent words can make a difference. In the previous example, not containing word b corresponds to a factor of (1−0.67)/(1−0.33) = 1/2 in the likelihood ratio.
- The other main difference between the two models is that multiple occurrences of words are treated like duplicated features in the multinomial model, through the exponential 'weight' xi .

### 4. Training a naive Bayes model:

Training a probabilistic model usually involves estimating the parameters of the distributions used in the model. The parameter of a Bernoulli distribution can be estimated by counting the number of successes d in n trials and setting $\hat{\theta} = d/n$. In other words, we count, for each class, how many e-mails contain the word in question. Such relative frequency estimates are usually smoothed by including pseudocounts, representing the outcome of virtual trials according to some fixed distributions.

In the case of a Bernoulli distribution the most common smoothing operation is the Laplace correction, which involves two virtual trials, one of which results in success and the other in failure. Consequently, the relative frequency estimate is changed to (d +1)/(n +2).

From a Bayesian perspective this amounts to adopting a uniform prior, representing our initial belief that success and failure are equally likely. If appropriate, we can strengthen the influence of the prior by including a larger number of virtual trials, which means that more data is needed to move the estimate away from the prior.

For a categorical distribution smoothing adds one pseudo-count to each of the k categories, leading to the smoothed estimate (d +1)/(n +k). The m-estimate generalises this further by making both the total number of pseudo-counts m and the way they are distributed over the categories into parameters. The estimate for the i -th category is defined as $(d + p_i m)/(n+m)$, where pi is a distribution over the categories

**Example:**

We now show how the parameter vectors in the previous example might have been obtained. Consider the following e-mails consisting of five words a, b, c, d, e:

e1: b d e b b d e

e2: b c e b b d d e c c

e3: a d a d e a e e

e4: b a d b e d a b

e5: a b a b a b a e d

e6: a c a c a c a e d

e7: e a e d a e a

e8: d e d e d

We are told that the e-mails on the left are spam and those on the right are ham, and so we use them as a small training set to train our Bayesian classifier. First, we decide that d and e are so-called stop words that are too common to convey class information. The remaining words, a, b and c, constitute our vocabulary. For the multinomial model, we represent each e-mail as a count vector, as in Table

| E-mail | #a | #b | #c | Class |
|---|---|---|---|---|
| $e_1$ | 0 | 3 | 0 | + |
| $e_2$ | 0 | 3 | 3 | + |
| $e_3$ | 3 | 0 | 0 | + |
| $e_4$ | 2 | 3 | 0 | + |
| $e_5$ | 4 | 3 | 0 | − |
| $e_6$ | 4 | 0 | 3 | − |
| $e_7$ | 3 | 0 | 0 | − |
| $e_8$ | 0 | 0 | 0 | − |

In order to estimate the parameters of the multinomial, we sum up the count vectors for each class, which gives (5, 9,3) for spam and (11,3,3) for ham. To smooth these probability estimates we add one pseudo-count for each vocabulary word, which brings the total number of occurrences of vocabulary words to 20 for each class. The estimated parameter vectors are thus $\hat{\theta}\oplus$ = (6/20,10/20,4/20) = (0.3,0.5,0.2) for spam and $\hat{\theta}\_$ = (12/20,4/20,4/20) = (0.6, 0.2,0.2) for ham.

In the multivariate Bernoulli model e-mails are represented by bit vectors, as in Table

| E-mail | a? | b? | c? | Class |
|---|---|---|---|---|
| $e_1$ | 0 | 1 | 0 | + |
| $e_2$ | 0 | 1 | 1 | + |
| $e_3$ | 1 | 0 | 0 | + |
| $e_4$ | 1 | 1 | 0 | + |
| $e_5$ | 1 | 1 | 0 | − |
| $e_6$ | 1 | 0 | 1 | − |
| $e_7$ | 1 | 0 | 0 | − |
| $e_8$ | 0 | 0 | 0 | − |

Adding the bit vectors for each class results in (2, 3,1) for spam and (3, 1,1) for ham. Each count is to be divided by the number of documents in a class, in order to get an estimate of the probability of a document containing a particular vocabulary word. Probability smoothing now means adding two pseudo-documents, one containing each word and one containing none of them. This results in the estimated parameter vectors $\hat{\theta}\oplus$ = (3/6,4/6,2/6) = (0.5,0.67,0.33) for spam and $\hat{\theta}\_$ = (4/6,2/6,2/6) = (0.67,0.33,0.33) for ham.

**Observation**: 'the' naive Bayes classifier employs neither a multinomial nor a multivariate Bernoulli model, but rather a multivariate categorical model. This means that features are categorical, and the probability of the i -th feature taking on its l -th value for class c examples is given by $\theta^{(c)}_{i\,l}$, under the constraint that $\sum_{l=1}^{k_i} \theta_{il}^{(c)} = 1$, where $k_i$ is the number of values of the i -th feature. These parameters can be estimated by smoothed relative frequencies in the training set, as in the multivariate Bernoulli case. We again have that the joint probability of the feature vector is the product of the individual feature probabilities, and hence P(Fi ,Fj |C) = P(Fi |C)P(Fj |C) for all pairs of features and for all classes.

**Summary:** The naive Bayes model is a popular model for dealing with textual, categorical and mixed categorical/real-valued data. Its main shortcoming as a probabilistic model – poorly calibrated probability estimates – are outweighed by generally good ranking performance. Another apparent paradox with naive Bayes is that it isn't particularly Bayesian at all! For one thing, we have seen that the poor probability estimates necessitate the use of reweighted likelihoods, which avoids using Bayes' rule altogether. Secondly, in training a naive Bayes model we use maximum-likelihood parameter estimation, whereas a fully fledged Bayesian approach would not commit to a particular parameter value, but rather employ a full posterior distribution.

## 5. Discriminative learning by optimising conditional likelihood:

The easiest way to understand logistic regression is as a linear classifier whose probability estimates have been logistically calibrated, but with one crucial difference: calibration is an integral part of the training algorithm, rather than a post-processing step. While in generative models the decision boundary is a by-product of modelling the distributions of each class, logistic regression models the decision boundary directly.

For example, if the classes are overlapping then logistic regression will tend to locate the decision boundary in an area where classes are maximally overlapping, regardless of the 'shapes' of the samples of each class.

the likelihood ratio as $\exp\left(\gamma(d(\mathbf{x}) - d_0)\right)$ with $d(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - t$. Since we are learning the parameters all at once in discriminative learning, we can absorb γ and d0 into **w** and t . So the logistic regression model is simply given by

$$\hat{p}(\mathbf{x}) = \frac{\exp(\mathbf{w}\cdot\mathbf{x} - t)}{\exp(\mathbf{w}\cdot\mathbf{x} - t) + 1} = \frac{1}{1 + \exp(-(\mathbf{w}\cdot\mathbf{x} - t))}$$

Assuming the class labels are y = 1 for positives and y = 0 for negatives, this defines a Bernoulli distribution for each training example:

$$P(y_i|\mathbf{x}_i) = \hat{p}(\mathbf{x}_i)^{y_i}(1 - \hat{p}(\mathbf{x}_i))^{(1-y_i)}$$

It is important to note that the parameters of these Bernoulli distributions are linked through **w** and t , and consequently there is one parameter for every feature dimension, rather than for every training instance. The likelihood function is

$$\mathrm{CL}(\mathbf{w}, t) = \prod_i P(y_i|\mathbf{x}_i) = \prod_i \hat{p}(\mathbf{x}_i)^{y_i}(1 - \hat{p}(\mathbf{x}_i))^{(1-y_i)}$$

This is called conditional likelihood to stress that it gives us the conditional probability P(yi |**x**i ) rather than P(**x**i ) as in a generative model. Notice that our use of the product requires the assumption that the y-values are independent given **x**; but this is an entirely reasonable assumption and not nearly as strong as the naive Bayes assumption of **x** being independent within each class. the logarithm of the likelihood function is easier to work with:

$$\mathrm{LCL}(\mathbf{w},t)=\sum_i y_i \ln \hat{p}(\mathbf{x}_i)+(1-y_i)\ln(1-\hat{p}(\mathbf{x}_i))=\sum_{\mathbf{x}^{\oplus}\in Tr^{\oplus}}\ln \hat{p}(\mathbf{x}^{\oplus})+\sum_{\mathbf{x}^{\ominus}\in Tr^{\ominus}}\ln(1-\hat{p}(\mathbf{x}^{\ominus}))$$

## 6. Probabilistic models with hidden variables:

Suppose you are dealing with a four-class classification problem with classes A, B, C and D. If you have a sufficiently large and representative training sample of size n, you can use the relative frequencies in the sample nA, . . . ,nD to estimate the class prior pˆA = nA/n, . . . ,pˆD = $n_D$/n

### 6.1. Expectation-Maximisation:

- the expectation $\mathbb{E}[Z|X,\theta^t]$ of the hidden variables given the observed variables and the current estimate of the parameters (so in Equation 9.6 the expectations of $a$ and $b$ depend on $s$ and $\beta$);
- the likelihood $P(Y|\theta)$, which is used to find the maximising value of $\theta$.

This means that we really want to maximise $P(X\cup\mathbb{E}[Z|X,\theta^t]|\theta)$, or equivalently, the logarithm of that function. We now make the assumption that the logarithm of the likelihood function is linear in Y : notice that this assumption is valid in the example above. For any linear function f , $f(\mathbb{E}[Z])=\mathbb{E}[f(Z)]$ and thus we can bring the expectation outside in our objective function:

$$\ln P(X\cup\mathbb{E}[Z|X,\theta^t]|\theta)=\mathbb{E}[\ln P(X\cup Z|\theta)|X,\theta^t]=\mathbb{E}[\ln P(Y|\theta)|X,\theta^t]$$

This last expression is usually denoted as Q(θ|θt ), as it essentially tells us how to calculate the next value of θ from the current one:

$$\theta^{t+1}=\arg\max_{\theta} Q(\theta|\theta^t)=\arg\max_{\theta}\mathbb{E}[\ln P(Y|\theta)|X,\theta^t]$$

the general form of the celebrated Expectation-Maximisation (EM) algorithm, which is a powerful approach to probabilistic modelling with hidden variables or missing data. Similar to the example above, we iterate over assigning an expected value to the hidden variables given our current estimates of the parameters, and re-estimating the parameters from these updated expectations, until a stationary configuration is reached.

### 6.2. Gaussian mixture models

A common application of Expectation-Maximisation is to estimate the parameters of a Gaussian mixture model from data. In such a model the data points are generated by K

normal distributions, each with their own mean $\boldsymbol{\mu}$j and covariance matrix $\boldsymbol{\Sigma}$j , and the proportion of points coming from each Gaussian is governed by a prior $\boldsymbol{\tau} = (\tau 1, \ldots, \tau K)$. If each data point in a sample were labelled with the index of the Gaussian it came from this would be a straightforward classification problem, which could be solved easily by estimating each Gaussian's $\boldsymbol{\mu}$j and $\boldsymbol{\Sigma}$j separately from the data points belonging to class j .

A convenient way to model this is to have for each data point $\mathbf{x}$i a Boolean vector $\mathbf{z}$i = (zi1, . . . , ziK ) such that exactly one bit zi j is set to 1 and the rest set to 0, signalling that the i -th data point comes from the j -th Gaussian. Using this notation we can adapt the expression for the multivariate normal distribution to obtain a general expression for a Gaussian mixture model:

$$P(\mathbf{x}_i, \mathbf{z}_i|\theta) = \sum_{j=1}^{K} z_{ij}\tau_j \frac{1}{(2\pi)^{d/2}\sqrt{|\boldsymbol{\Sigma}_j|}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_j)\right)$$

Here, θ collects all the parameters τ, $\boldsymbol{\mu}$1, . . . ,$\boldsymbol{\mu}$K and $\boldsymbol{\Sigma}$1, . . . ,$\boldsymbol{\Sigma}$K . The interpretation as a generative model is as follows: we first randomly select a Gaussian using the prior $\boldsymbol{\tau}$ , and then we invoke the corresponding Gaussian using the indicator variables zi j . In order to apply Expectation-Maximisation we form the Q function:

$$\begin{aligned}
Q(\theta|\theta^t) &= \mathbb{E}\left[\ln P(\mathbf{X}\cup\mathbf{Z}|\theta)|\mathbf{X},\theta^t\right] \\
&= \mathbb{E}\left[\ln\prod_{i=1}^{n} P(\mathbf{x}_i\cup\mathbf{z}_i|\theta)\middle|\mathbf{X},\theta^t\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{n}\ln P(\mathbf{x}_i\cup\mathbf{z}_i|\theta)\middle|\mathbf{X},\theta^t\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{n}\ln\sum_{j=1}^{K} z_{ij}\tau_j\frac{1}{(2\pi)^{d/2}\sqrt{|\boldsymbol{\Sigma}_j|}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)\middle|\mathbf{X},\theta^t\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{K} z_{ij}\ln\left(\tau_j\frac{1}{(2\pi)^{d/2}\sqrt{|\boldsymbol{\Sigma}_j|}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)\right)\middle|\mathbf{X},\theta^t\right] \qquad (*) \\
&= \mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{K} z_{ij}\left(\ln\tau_j - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_j| - \frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)\middle|\mathbf{X},\theta^t\right] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{K}\mathbb{E}\left[z_{ij}|\mathbf{X},\theta^t\right]\left(\ln\tau_j - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_j| - \frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)
\end{aligned}$$

The last line shows the Q function in the desired form, involving on the one hand expectations over the hidden

variables conditioned on the observable data $\mathbf{X}$ and the previously estimated parameters θt , and on the other hand expressions in θ that allow us to find θt+1 by maximisation. In the general case these expectations are apportioned proportionally to the probability mass assigned to the point by each Gaussian:

$$\mathbb{E}[z_{ij}|\mathbf{X},\theta^t] = \frac{\tau_j^t f(\mathbf{x}_i|\mu_j^t,\Sigma_j^t)}{\sum_{k=1}^K \tau_k^t f(\mathbf{x}_i|\mu_k^t,\Sigma_k^t)}$$

where f ($\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}$) stands for the multivariate Gaussian density function.

## 6.3. Compression-based models

Consider the maximum a posteriori decision rule again:

$$y_{\text{MAP}} = \arg\max_y P(X = x|Y = y)P(Y = y)$$

Taking negative logarithms, we can turn this into an equivalent minimisation:

$$y_{\text{MAP}} = \arg\min_y -\log P(X = x|Y = y) - \log P(Y = y)$$

This follows because for any two probabilities 0 < p < p < 1 we have ∞ > −logp > −logp > 0. If an event has probability p of happening, the negative logarithm of p quantifies the information content of the message that the event has indeed happened. This makes intuitive sense, as the less expected an event is, the more information an announcement of the event contains. The unit of information depends on the base of the logarithm: it is customary to take logarit;:hms to the base 2, in which case information is measured in bits.

for a uniform distribution over k outcomes, each outcome has the same information content −log2 1/k = log2 k. For a non-uniform distribution these information contents differ, and hence it makes sense to compute the average information content or entropy $\sum_{i=1}^k -p_i \log_2 p_i$

**Minimum description length principle:**

Let L(m) denote the length in bits of a description of model m, and let L(D|m) denote the length in bits of a description of data D given model m. According to the minimum description length principle, the preferred model is the one minimising the description length of model and data given model:

$$m_{\text{MDL}} = \arg\min_{m \in M} (L(m) + L(D|m))$$

'description of data given model' refers to whatever information we need, in addition to the model and the feature values of the data, to infer the target labels. If the model is 100% accurate no further information is needed, so this term essentially quantifies the extent to which the model is incorrect.

The term L(m) quantifies the complexity of the model. For instance, if we are fitting a polynomial to the data we need to encode the degree of the polynomial as well as its roots, up to a certain resolution. MDL learning thus trades off accuracy and complexity of a model: the complexity term serves to avoid overfitting in a similar way to the regularisation term in ridge regression and the slack variable term in soft-margin SVMs

## 7. Features:

Features are 'the workhorses of machine learning' – it is therefore high time to consider them in more detail. Features, also called attributes, are defined as mappings fi :X →Fi from the instance spaceX to the feature domain, Fi.

### 7.1. Kinds of feature

Consider two features, one describing a person's age and the other their house number. Both features map into the integers, but the way we use those features can be quite different. Calculating the average age of a group of people is meaningful, but an average house number is probably not very useful! In other words, what matters is not just the domain of a feature, but also the range of permissible operations. These, in turn, depend on whether the feature values are expressed on a meaningful scale. Despite appearances, house numbers are not really integers but ordinals

### 7.2. Calculations on features

Three main categories are statistics of central tendency, statistics of dispersion and shape statistics. Each of these can be interpreted either as a theoretical property of an unknown population or a concrete property of a given sample, here we will concentrate on sample statistics.

A. Starting with statistics of central tendency, the most important ones are

- the mean or average value;
- the median, which is the middle value if we order the instances from lowest to highest feature value; and
- the mode, which is the majority value or values.

B. The second kind of calculation on features are statistics of dispersion or 'spread'. Two well-known statistics of dispersion are the variance or average squared deviation from the (arithmetic) mean, and its square root, the standard deviation. Variance and standard deviation essentially measure the same thing, but the latter has the advantage that it is expressed on the same scale as the feature itself.

> A simpler dispersion statistic is the difference between maximum and minimum value, which is called the range. A natural statistic of central tendency to be used with the range is the midrange point, which is the mean of the two extreme values. These definitions assume a linear scale but can be adapted to other scales using suitable transformations.

Other statistics of dispersion include percentiles. The p-th percentile is the value such that p per cent of the instances fall below it. If we have 100 instances, the 80th percentile is the value of the 81st instance in a list of increasing values.2 If p is a multiple of 25 the percentiles are also called quartiles, and if it is a multiple of 10 the percentiles are also called deciles. Note that the 50th percentile, the 5th decile and the second quartile are all the same as the median. Percentiles, deciles and quartiles are special cases of quantiles. Once we have quantiles we can measure dispersion as the distance between different quantiles. For instance, the interquartile range is the difference between the third and first quartile (i.e., the 75th and 25th percentile).

One advantage of drawing the plot this way is that, by interpreting the y-axis as probabilities, the plot can be read as a cumulative probability distribution: a plot of P(X ≤ x) against x for a random variable X.

C. The skew and 'peakedness' of a distribution can be measured by shape statistics such as skewness and kurtosis. The main idea is to calculate the third and fourth central moment of the sample. In general, the k-th central moment of a sample {xi , . . . ,xn} is defined as $m_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^k$, where μ is the sample mean.
Skewness is then defined as m3/σ3, where σ is the sample's standard deviation. A positive value of skewness means that the distribution is right-skewed, which means that the right tail is longer than the left tail. Negative skewness indicates the opposite, left-skewed case. Kurtosis is defined as m4/σ4.

| *Kind* | *Order* | *Scale* | *Tendency* | *Dispersion* | *Shape* |
|---|---|---|---|---|---|
| Categorical | × | × | mode | n/a | n/a |
| Ordinal | √ | × | median | quantiles | n/a |
| Quantitative | √ | √ | mean | range, interquartile range, variance, standard deviation | skewness, kurtosis |

Given these various statistics we can distinguish three main kinds of feature: those with a meaningful numerical scale, those without a scale but with an ordering, and those without either. We will call features of the first type quantitative; they most often involve a mapping into the reals (another term in common use is 'continuous'). Even if a feature maps into a subset of the reals, such as age expressed in years, the various statistics such as mean or standard deviation still require the full scale of the reals.

Features with an ordering but without scale are called ordinal features. The domain of an ordinal feature is some totally ordered set, such as the set of characters or strings. Even if the domain of a feature is the set of integers, denoting the feature as ordinal means that we have to dispense with the scale, as we did with house numbers. Another common example are features that express a rank order: first, second, third, and so on. Ordinal features allow the mode and median as central tendency statistics, and quantiles as dispersion statistics. Features without ordering or scale are called categorical features (or sometimes 'nominal' features). They do not allow any statistical summary except the mode. One subspecies of the categorical features is the Boolean feature, which maps into the truth values true and false

## 7.3. Structured features:

The instance space is a Cartesian product of d feature domains: X =F1 ×. . .×Fd . This means that there is no other information available about an instance apart from the information conveyed by its feature values. Identifying an instance with its vector of feature values is what computer scientists call an abstraction, which is the result of filtering out unnecessary information. Representing an e-mail as a vector of word frequencies is an example of an abstraction. However, sometimes it is necessary to avoid such abstractions, and to keep more information about an instance than can be captured by a finite vector of feature values. For example, we could represent an e-mail as a long string; or as a sequence of words and punctuation marks; or as a tree that captures the HTML mark-up; and so on. Features that operate on such structured instance spaces are called structured features.

Structured features can be constructed either prior to learning a model, or simultaneously with it. The first scenario is often called propositionalisation because the features can be seen as a translation from first-order logic to propositional logic without local variables.

## 8. Feature transformations

Feature transformations aim at improving the utility of a feature by removing, changing or adding information. We could order feature types by the amount of detail they convey: quantitative features are more detailed than ordinal ones, followed by categorical features, and finally Boolean features. The best-known feature transformations are those that turn a feature of one type into another of the next type down this list. But there are also transformations that change the scale of quantitative features, or add a scale (or order) to ordinal, categorical and Boolean features

| ↓ to, from → | *Quantitative* | *Ordinal* | *Categorical* | *Boolean* |
|---|---|---|---|---|
| *Quantitative* | **normalisation** | **calibration** | **calibration** | **calibration** |
| *Ordinal* | **discretisation** | **ordering** | **ordering** | **ordering** |
| *Categorical* | **discretisation** | **unordering** | **grouping** | |
| *Boolean* | **thresholding** | **thresholding** | **binarisation** | |

**Table 10.2.** An overview of possible feature transformations. **Normalisation and calibration** adapt the scale of quantitative features, or add a scale to features that don't have one. **Ordering** adds or adapts the order of feature values without reference to a scale. The other operations abstract away from unnecessary detail, either in a deductive way (**unordering**, **binarisation**) or by introducing new information (**thresholding**, **discretisation**).

The simplest feature transformations are entirely deductive, in the sense that they achieve a well-defined result that doesn't require making any choices. Binarisation transforms a categorical feature into a set of Boolean features, one for each value of the categorical feature. This loses information since the values of a single categorical feature are mutually exclusive, but is sometimes needed if a model cannot handle more than two feature values. Unordering trivially turns an ordinal feature into a categorical one by discarding the ordering of the feature values.

### 8.1. Thresholding and discretisation

Thresholding transforms a quantitative or an ordinal feature into a Boolean feature by finding a feature value to split on. Concretely, let $f : X \rightarrow R$ be a quantitative feature and let $t \square R$ be a threshold, then $f_t : X \rightarrow \{true, false\}$ is a Boolean feature defined by $f_t(x) = true$ if $f(x) \geq t$ and $f_t(x) = false$ if $f(x) < t$ .

unsupervised thresholding typically involves calculating some statistic over the data, whereas supervised thresholding requires sorting the data on the feature value and traversing down this ordering to optimise a particular objective function such as information gain. If we generalise thresholding to multiple thresholds we arrive at one of the most commonly used non-deductive feature transformations. Discretisation transforms a quantitative feature into an ordinal feature. Each ordinal value is referred to as a bin and corresponds to an interval of the original quantitative feature. Again, we can distinguish between supervised and unsupervised approaches. Unsupervised discretisation methods typically require one to decide the number of bins beforehand. A simple method that often works reasonably well is

to choose the bins so that each bin has approximately the same number of instances: this is referred to as equal-frequency discretisation.

supervised discretisation methods, we can distinguish between top–down or divisive discretisation methods on the one hand, and bottom–up or agglomerative discretisation methods on the other. Divisive methods work by progressively splitting bins, whereas agglomerative methods proceed by initially assigning each instance to its own bin and successively merging bins. In either case an important role is played by the stopping criterion, which decides whether a further split or merge is worthwhile. We give an example of each strategy. A natural generalisation of thresholding leads to a top–down recursive partitioning algorithm (Algorithm 10.1). This discretisation algorithm finds the best threshold according to some scoring function Q, and proceeds to recursively split the left and right bins. One scoring function that is often used is information gain.

**Algorithm 10.1:** RecPart($S, f, Q$) – supervised discretisation by means of recursive partitioning.

**Input** : set of labelled instances $S$ ranked on feature values $f(x)$; scoring function $Q$.
**Output** : sequence of thresholds $t_1, \ldots, t_{k-1}$.

```
if stopping criterion applies then return ∅;
Split S into S_l and S_r using threshold t that optimises Q;
T_l = RecPart(S_l, f, Q);
T_r = RecPart(S_r, f, Q);
return T_l ∪ {t} ∪ T_r;
```

An algorithm for bottom–up agglomerative merging is given in Algorithm 10.2. Again the algorithm can take various choices for the scoring function and the stopping criterion: a popular choice is to use the $\chi 2$ statistic for both.

**Algorithm 10.2:** AggloMerge($S, f, Q$) – supervised discretisation by means of agglomerative merging.

**Input** : set of labelled instances $S$ ranked on feature values $f(x)$; scoring function $Q$.
**Output** : sequence of thresholds.

```
initialise bins to data points with the same scores;
merge consecutive pure bins;                    // optional optimisation
repeat
    evaluate Q on consecutive bin pairs;
    merge the pairs with best Q (unless they invoke the stopping criterion);
until no further merges are possible;
return thresholds between bins;
```

In agglomerative discretisation the stopping criterion usually takes the form of a simple threshold on the scoring function. In the case of the χ2 statistic, the threshold can be derived from the p-value associated with the χ2 distribution, which is the probability of observing a χ2 value above the threshold if the two variables are actually independent.

## 8.3 Normalisation and calibration

Thresholding and discretisation are feature transformations that remove the scale of a quantitative feature. Feature normalisation is often required to neutralise the effect of different quantitative features being measured on different scales. If the features are approximately normally distributed, we can convert them into z-scores by centring on the mean and dividing by the standard deviation. In certain cases it is mathematically more convenient to divide by the variance. Sometimes feature normalisation is understood in the stricter sense of expressing the feature on a [0,1] scale. This can be achieved in various ways. If we know the feature's highest and lowest values h and l , then we can simply apply the linear scaling f _→ ( f −l )/(h −l ). We sometimes have to guess the value of h or l , and truncate any value outside [l ,h].

Feature calibration is understood as a supervised feature transformation adding a meaningful scale carrying class information to arbitrary features. This has a number of important advantages. For instance, it allows models that require scale, such as linear classifiers, to handle categorical and ordinal features. It also allows the learning algorithm to choose whether to treat a feature as categorical, ordinal or quantitative. This has the additional advantage that models that are based on such probabilities, such as naive Bayes, do not require any additional training once the features are calibrated. Ordinal and quantitative features can be discretised and then calibrated as categorical features. A calibrated weight feature attaches a probability to every weight, such that these probabilities are non-decreasing with weight. Assuming the feature is normally distributed within each class with the same variance, we can express the likelihood ratio of a feature value v as

$$LR(v) = \frac{P(v|\oplus)}{P(v|\ominus)} = \exp\left(\frac{-(v-\mu^{\oplus})^2 + (v-\mu^{\ominus})^2}{2\sigma^2}\right)$$
$$= \exp\left(\frac{\mu^{\oplus}-\mu^{\ominus}}{\sigma}\frac{v-(\mu^{\oplus}+\mu^{\ominus})/2}{\sigma}\right) = \exp\left(d'z\right)$$

where $d' = (\mu^{\oplus} - \mu^{\ominus})/\sigma$ is the difference between the means in proportion to the standard deviation, which is known as d-prime in signal detection theory; we obtain the calibrated feature value as

$$F^c(x) = \frac{LR(F(x))}{1+LR(F(x))} = \frac{\exp\left(d'z(x)\right)}{1+\exp\left(d'z(x)\right)}$$

In essence, logistic feature calibration performs the following steps.

1. Estimate the class means $\mu^{\oplus}$ and $\mu^{\ominus}$ and the standard deviation $\sigma$.
2. Transform $F(x)$ into $z$-scores $z(x)$, making sure to use $\mu = (\mu^{\oplus} + \mu^{\ominus}))/2$ as the feature mean.
3. Rescale the $z$-scores to $F^{\mathrm{d}}(x) = d'z(x)$ with $d' = (\mu^{\oplus} - \mu^{\ominus})/\sigma$.
4. Apply a sigmoidal transformation to $F^{\mathrm{d}}(x)$ to give calibrated probabilities $F^{\mathrm{c}}(x) = \frac{\exp(F^{\mathrm{d}}(x))}{1+\exp(F^{\mathrm{d}}(x))}$.

isotonic calibration, a method that requires order but ignores scale and can be applied to both ordinal and quantitative features. We essentially use the feature as a univariate ranker, and construct its ROC curve and convex hull to obtain a piecewise-constant calibration map. isotonic feature calibration performs the following steps.

1. Sort the training instances on feature value and construct the ROC curve. The sort order is chosen such that the ROC curve has AUC $\geq 1/2$.
2. Construct the convex hull of this curve, and count the number of positives $m_i$ and the total number of instances $n_i$ in each segment of the convex hull.
3. Discretise the feature according to the convex hull segments, and associate a calibrated feature value $v_i^{\mathrm{c}} = \frac{m_i+1}{m_i+1+c(n_i-m_i+1)}$ with each segment.
4. If an additive scale is required, use $v_i^{\mathrm{d}} = \ln \frac{v_i^{\mathrm{c}}}{1-v_i^{\mathrm{c}}} = \ln v_i^{\mathrm{c}} - \ln(1 - v_i^{\mathrm{c}})$.

## 8.4. Incomplete features

Probabilistic models handle this rather gracefully by taking a weighted average over all possible values of the feature:

$$P(Y|X) = \sum_z P(Y, Z = z|X) = \sum_z P(Y|X, Z = z)P(Z = z)$$

Here, Y is the target variable as usual, X stands for the features that are observed for the instance to be classified, while Z are the features that are unobserved at classification time. The distribution P(Z) can be obtained from the trained model, at least for a generative model – if our model is discriminative we need to estimate P(Z) separately.

Missing feature values at training time are trickier to handle. First of all, the very fact that a feature value is missing may be correlated with the target variable. For example, the range of medical tests carried out on a patient is likely to depend on their medical history. For such features it may be best to have a designated 'missing' value so that, for instance, a tree model can split on it. However, this would not work for, say, a linear model. In such cases we can complete the feature by 'filling in' the missing values, a process known as imputation.

## 9. Feature construction and selection

we can construct a new feature from two Boolean or categorical features by forming their Cartesian product. For example, if we have one feature Shape with values Circle, Triangle and Square, and another feature Colour with values Red, Green and Blue, then their Cartesian product would be the feature (Shape,Colour) with values (Circle,Red), (Circle,Green), (Circle,Blue), (Triangle,Red), and so on. The effect that this would have depends on the model being trained. Constructing Cartesian product features for a naive Bayes classifier means that the two original features are no longer treated as independent, and so this reduces the strong bias that naive Bayes models have. This is not the case for tree models, which can already distinguish between all possible pairs of feature values. On the other hand, a newly introduced Cartesian product feature may incur a high information gain, so it can possibly affect the model learned.

There are many other ways of combining features. One attractive possibility is to first apply concept learning or subgroup discovery, and then use these concepts or subgroups as new Boolean features. Once we have constructed new features it is often a good idea to select a suitable subset of them prior to learning. Not only will this speed up learning as fewer candidate features need to be considered, it also helps to guard against overfitting. There are two main approaches to feature selection. The filter approach scores features on a particular metric and the top-scoring features are selected. Many of the metrics we have seen so far can be used for feature scoring, including information gain, the $\chi 2$ statistic, the correlation coefficient, to name just a few. An interesting variation is provided by the Relief feature selection method, which repeatedly samples a random instance x and finds its nearest hit h (instance of the same class) as well as its nearest miss m (instance of opposite class). The i -th feature's score is then decreased by Dis(xi ,hi )2 and increased by Dis(xi ,mi )2, where Dis is some distance measure (e.g., Euclidean distance for quantitative features, Hamming distance for categorical features). The intuition is that we want to move closer to the nearest hit while differentiating from the nearest miss.

One of the best-known algebraic feature construction methods is principal component analysis (PCA). Principal components are new features constructed as linear combinations of the given features. The first principal component is given by the direction of maximum variance in the data; the second principal component is the direction of maximum variance orthogonal to the first component, and so on. PCA can be explained in a number of different ways: here, we will derive it by means of the singular value decomposition (SVD). Any n-by-d matrix can be uniquely written as a product of three matrices with special properties: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$

Here, **U** is an n-by-r matrix, **Σ** is an r -by-r matrix and **V** is an d-by-r matrix (for the moment we will assume r = d < n). Furthermore, **U** and **V** are orthogonal (hence rotations) and **Σ** is diagonal (hence a scaling). The columns of **U** and **V** are known as the left and right singular vectors, respectively; and the values on the diagonal of **Σ** are the corresponding singular values. It is customary to order the columns of **V** and **U** so that the singular values are decreasing from top-left to bottom-right.

Now, consider the n-by-r matrix **W**= **UΣ**, and notice that **XV** = **UΣVTV** = **UΣ** =**W** by the orthogonality of **V**. In other words, we can construct **W** from **X** by means of the transformation **V**: this is the reformulation of **X** in terms of its principal components. The newly constructed features are found in**UΣ**: the first rowis the first principal component, the second row is the second principal component, and so on. These principal components have a geometric interpretation as the directions in which **X** has largest, second-largest, . . . variance.

Assuming the data is zero-centred, these directions can be brought out by a combination of rotation and scaling, which is exactly what PCA does. We can also use SVD to rewrite the scatter matrix in a standard form:

$$\mathbf{S} = \mathbf{X}^T\mathbf{X} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^1 (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$$

This is known as the eigen decomposition of the matrix S: the columns of V are the eigenvectors of S, and the elements on the diagonal of $\Sigma^2$ – which is itself a diagonal matrix – are the eigen values. The right singular vectors of the data matrix X are the eigenvectors of the scatter matrix $S = X^TX$, and the singular values of X are the square root of the eigen values of S. We can derive a similar expression for the Gram matrix $G = XX^T = U\Sigma^2U^T$, from which we see that the eigenvectors of the Gram matrix are the left singular vectors of X. This demonstrates that in order to perform principal components analysis it is sufficient to perform an eigen decomposition of the scatter or Gram matrices, rather than a full singular value decomposition.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The matrix on the left expresses people's preferences for films (in columns). The right hand side decomposes or factorises this into film genres: the first matrix quantifies people's appreciation of genres; the last matrix associates films with genres; and the middle matrix tells us the weight of each genre in determining preferences.

## 10. Bagging and random forests

Bagging, short for 'bootstrap aggregating', is a simple but highly effective ensemble method that creates diverse models on different random samples of the original data set. These samples are taken uniformly with replacement and are known as bootstrap samples. Because samples are taken with replacement the bootstrap sample will in general contain duplicates, and hence some of the original data points will be missing even if the bootstrap sample is of the same size as the original data set.

Algorithm 11.1 gives the basic bagging algorithm, which returns the ensemble as a set of models. We can choose to combine the predictions from the different models by voting – the class predicted by the majority of models wins – or by averaging, which is more appropriate if the base classifiers output scores or probabilities.

**Algorithm 11.1:** Bagging($D, T, \mathscr{A}$) – train an ensemble of models from bootstrap samples.

**Input** : data set $D$; ensemble size $T$; learning algorithm $\mathscr{A}$.
**Output** : ensemble of models whose predictions are to be combined by voting or averaging.

```
for t = 1 to T do
    build a bootstrap sample D_t from D by sampling |D| data points with replacement;
    run A on D_t to produce a model M_t;
end
return {M_t | 1 ≤ t ≤ T}
```

Bagging is particularly useful in combination with tree models, which are quite sensitive to variations in the training data. When applied to tree models, bagging is often combined with another idea: to build each tree from a different random subset of the features, a process also referred to as subspace sampling. This encourages the diversity in the ensemble even more, and has the additional advantage that the training time of each tree is reduced. The resulting ensemble method is called random forests, and the algorithm is given in Algorithm 11.2.

**Algorithm 11.2:** RandomForest($D, T, d$) – train an ensemble of tree models from bootstrap samples and random subspaces.

**Input** : data set $D$; ensemble size $T$; subspace dimension $d$.
**Output** : ensemble of tree models whose predictions are to be combined by voting or averaging.

```
for t = 1 to T do
    build a bootstrap sample D_t from D by sampling |D| data points with replacement;
    select d features at random and reduce dimensionality of D_t accordingly;
    train a tree model M_t on D_t without pruning;
end
return {M_t | 1 ≤ t ≤ T}
```

## 11. Boosting

Boosting is an ensemble technique that is superficially similar to bagging, but uses a more sophisticated technique than bootstrap sampling to create diverse training sets. The basic idea is simple and appealing. Suppose we train a linear classifier on a data set and find that its training error rate is _. We want to add another classifier to the ensemble that does better on the misclassifications of the first classifier. One way to do that is to duplicate the misclassified instances: if our base model is the basic linear classifier, this will shift the class means towards the duplicated instances. A better way to achieve the same thing is to give the

misclassified instances a higher weight, and to modify the classifier to take these weights into account (e.g., the basic linear classifier
can calculate the class means as a weighted average).

**Algorithm 11.3:** Boosting($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

**Input** : data set $D$; ensemble size $T$; learning algorithm $\mathscr{A}$.
**Output** : weighted ensemble of models.

$w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$; // start with uniform weights
**for** $t = 1$ to $T$ **do**
  run $\mathscr{A}$ on $D$ with weights $w_{ti}$ to produce a model $M_t$;
  calculate weighted error $\epsilon_t$;
  **if** $\epsilon_t \geq 1/2$ **then**
    set $T \leftarrow t-1$ and break
  **end**
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$; // confidence for this model
  $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$; // increase weight
  $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$; // decrease weight
**end**
**return** $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$

## 12. Boosted rule learning

An interesting variant of boosting arises when our base models are partial classifiers that sometimes abstain from making a prediction. For example, suppose that our base classifiers are conjunctive rules whose head is fixed to predicting the positive class. An individual rule therefore either predicts the positive class for those instances it covers, or otherwise abstains from making a prediction. We can use boosting to learn an ensemble of such rules that takes a weighted vote among its members.

We need to make some small adjustments to the boosting equations, as follows. Notice that $\epsilon_t$ is the weighted error of the $t$-th base classifier. Since our rules always predict positive for covered instances, these errors only concern covered negatives, which we will indicate by $\epsilon_t^{\ominus}$. Similarly, we indicate the weighted sum of covered positives as $\epsilon_t^{\oplus}$, which will play the role of $1-\epsilon_t$. However, with abstaining rules there is a third component, indicated as $\epsilon_t^0$, which is the weighted sum of instances which the rule doesn't cover ($\epsilon_t^0+\epsilon_t^{\oplus}+\epsilon_t^{\ominus}=1$). We then have

$$Z_t = \epsilon_t^0 + \epsilon_t^{\ominus}\exp(\alpha_t) + \epsilon_t^{\oplus}\exp(-\alpha_t)$$

The value of $\alpha_t$ which maximises this is

$$\alpha_t = \frac{1}{2}\ln\frac{\epsilon_t^{\oplus}}{\epsilon_t^{\ominus}} = \ln\sqrt{\frac{\epsilon_t^{\oplus}}{\epsilon_t^{\ominus}}} \tag{11.4}$$

which gives

$$Z_t = \epsilon_t^0 + 2\sqrt{\epsilon_t^{\oplus}\epsilon_t^{\ominus}} = 1-\epsilon_t^{\oplus}-\epsilon_t^{\ominus}+2\sqrt{\epsilon_t^{\oplus}\epsilon_t^{\ominus}} = 1-\left(\sqrt{\epsilon_t^{\oplus}}-\sqrt{\epsilon_t^{\ominus}}\right)^2$$

This means that in each boosting round we construct a rule that maximises $\left|\sqrt{\epsilon_t^{\oplus}}-\sqrt{\epsilon_t^{\ominus}}\right|$, and set its confidence factor to $\alpha_t$ as in Equation 11.4. In order to obtain a prediction

# Dimensionality Reduction

## 1. Principal Component Analysis:

It is a statistical procedure that uses an orthogonal transformation which converts a set of correlated variables to a set of uncorrelated variables. PCA is a most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover, PCA is an unsupervised statistical technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.

### 1.1 Steps Involved in PCA

1. Standardize the data. (with mean =0 and variance = 1)
2. Compute the Covariance matrix of dimensions.
3. Obtain the Eigenvectors and Eigenvalues from the covariance matrix (we can also use correlation matrix or even Single value decomposition, however in this post will focus on covariance matrix).
4. Sort eigen values in descending order and choose the top k Eigenvectors that correspond to the k largest eigen values (k will become the number of dimensions of the new feature subspace k≤d, d is the number of original dimensions).
5. Construct the projection matrix W from the selected k Eigenvectors.
6. Transform the original data set X via W to obtain the new k-dimensional feature subspace Y.

### 1.2 Implementation and Demonstration:

#### 1. Standardization

When there are different scales used for the measurement of the values of the features, then it is advisable to do the standardization to bring all the feature spaces with mean = 0 and variance = 1.

The reason why standardization is very much needed before performing PCA is that PCA is very sensitive to variances. Meaning, if there are large differences between the scales (ranges) of the features, then those with larger scales will dominate over those with the small scales. So, transforming the data to the same scales will prevent this problem. That is where we use standardization to bring the features with mean value 0 and variance 1.

$$Standardized\ value\ of\ x_i = \frac{x_i - mean\ of\ x}{std\ Deviation\ of\ x}$$

#### 2. Eigen decomposition — Computing Eigenvectors and Eigenvalues

The eigenvectors and eigen values of a covariance (or correlation) matrix represent the “core” of a PCA:

- The Eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude.
- In other words, the eigenvalues explain the variance of the data along the new feature axes. It means the corresponding eigenvalue tells us that how much variance is included in that new transformed feature.
- To get eigenvalues and Eigenvectors we need to compute the covariance matrix. So in the next step let's compute it.

### 3. Selecting The Principal Components

- The typical goal of a PCA is to reduce the dimensionality of the original feature space by projecting it onto a smaller subspace, where the eigenvectors will form the axes.
- However, the eigenvectors only define the directions of the new axis, since they have all the same unit length 1.

### 4. Construct the projection matrix W from the selected k eigenvectors

- Projection matrix will be used to transform the Iris data onto the new feature subspace or we say newly transformed data set with reduced dimensions.
- It is a matrix of our concatenated top k Eigenvectors.

## 2. Artificial Neural Networks:

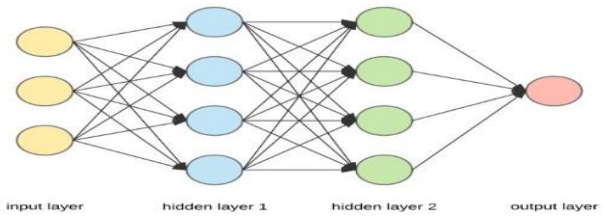
"Artificial Neural Networks or ANN is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements(neurons) working in unison to solve a specific problem."

## 2.1 NEURAL NETWORK REPRESENTATIONS

***Artificial Neural Network*** is computing system inspired by biological neural network that constitute animal brain. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

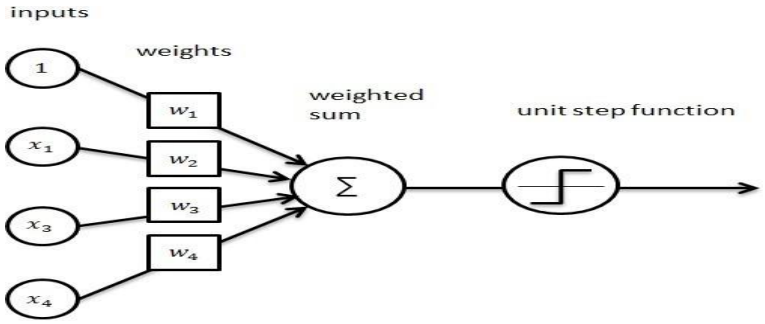
The Neural Network is constructed from 3 type of layers:

1. Input layer — initial data for the neural network.
2. Hidden layers — intermediate layer between input and output layer and place where all the computation is done.
3. Output layer — produce the result for given inputs.



There are 3 yellow circles on the image above. They represent the input layer and usually are noted as vector ***X.*** There are 4 blue and 4 green circles that represent the hidden layers. These circles represent the "activation" nodes and usually are noted as ***W*** or ***θ***. The red circle is the output layer or the predicted value (or values in case of multiple output classes/types).

Each node is connected with each node from the next layer and each connection (black arrow) has particular weight. Weight can be seen as impact that that node has on the node from the next layer. So if we take a look on one node it would look like this




## 2.2APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING:

ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones. It is also applicable to problems for which more symbolic representations are often used, such as the decision tree learning. It is appropriate for problems with the following characteristics:

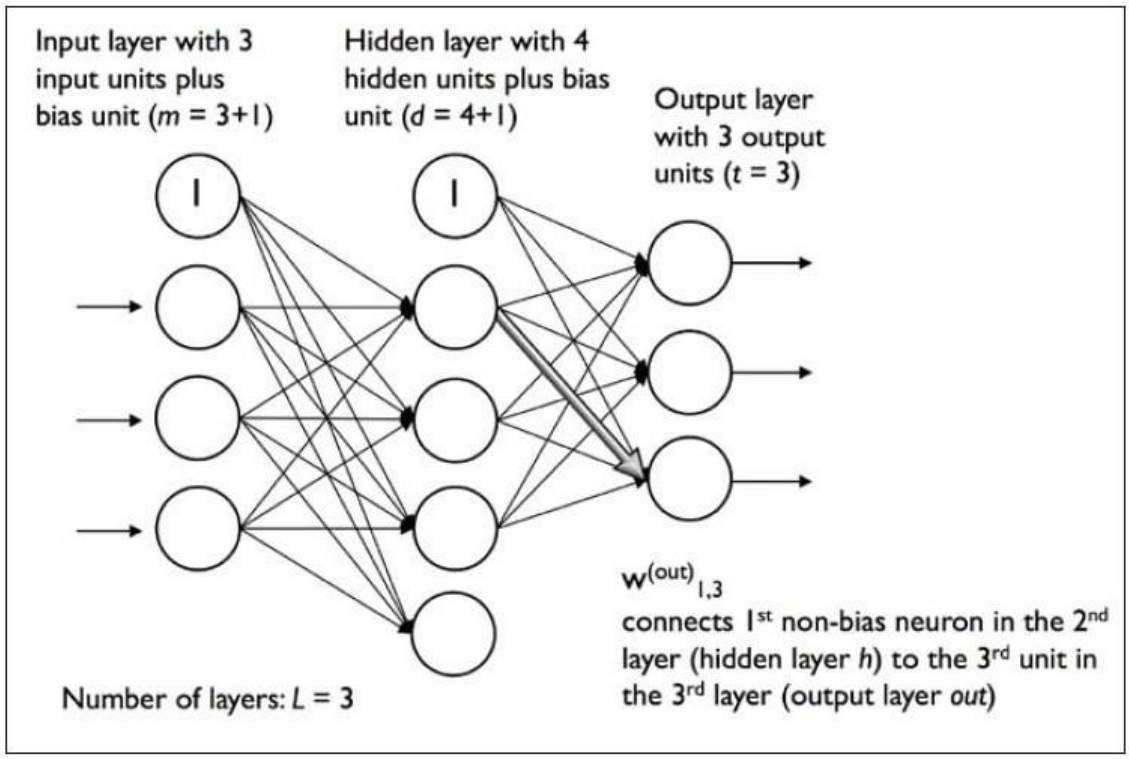
- ***Instances are represented by many attribute-value pairs.***
  The target function to be learned is defined over instances that can be described by a vector of predefined features, such as the pixel values in the ALVINN example. These input attributes may be highly correlated or independent of one another. Input values can be any real values.
- ***The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.***
  For example, in the ALVINN system the output is a vector of 30 attributes, each corresponding to a recommendation regarding the steering direction. The value of each output is some real number between 0 and 1, which in this case corresponds to the confidence in predicting the corresponding steering direction. We can also train a single network to output both the steering command and suggested acceleration, simply by concatenating the vectors that encode these two output predictions.
- ***The training examples may contain errors.***
  ANN learning methods are quite robust to noise in the training data.
- ***Long training times are acceptable.***
  Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.
- ***Fast evaluation of the learned target function may be required.***
  Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast. For example,

ALVINN applies its neural network several times per second to continually update its steering command as the vehicle drives forward.

- ***The ability of humans to understand the learned target function is not important.*** The weights learned by neural networks are often difficult for humans to interpret. Learned neural networks are less easily communicated to humans than learned rules.

## 2.3 Multi Layer ANN:

A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP).




It has 3 layers including one hidden layer. If it has more than 1 hidden layer, it is called a deep ANN. An MLP is a typical example of a feedforward artificial neural network. In this figure, the $i^{th}$ activation unit in the $l^{th}$ layer is denoted as $a_i^{(l)}$. The number of layers and the number of neurons are referred to as hyperparameters of a neural network, and these need tuning. Cross-validation techniques must be used to find ideal values for these. The weight adjustment training is done via backpropagation. Deeper neural networks are better at processing data. However, deeper layers can lead to vanishing gradient problem. Special algorithms are required to solve this issue.

Notations

In the representation below:

$$a^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix}$$

- $a_i^{(in)}$ refers to the $i^{th}$ value in the input layer
- $a_i^{(h)}$ refers to the $i^{th}$ unit in the hidden layer
- $a_i^{(out)}$ refers to the $i^{th}$ unit in the output layer
- $a_0^{(in)}$ is simply the bias unit and is equal to 1; it will have the corresponding weight $w_0$
- The weight coefficient from layer l to layer l+1 is represented by $w_{k,j}^{(l)}$

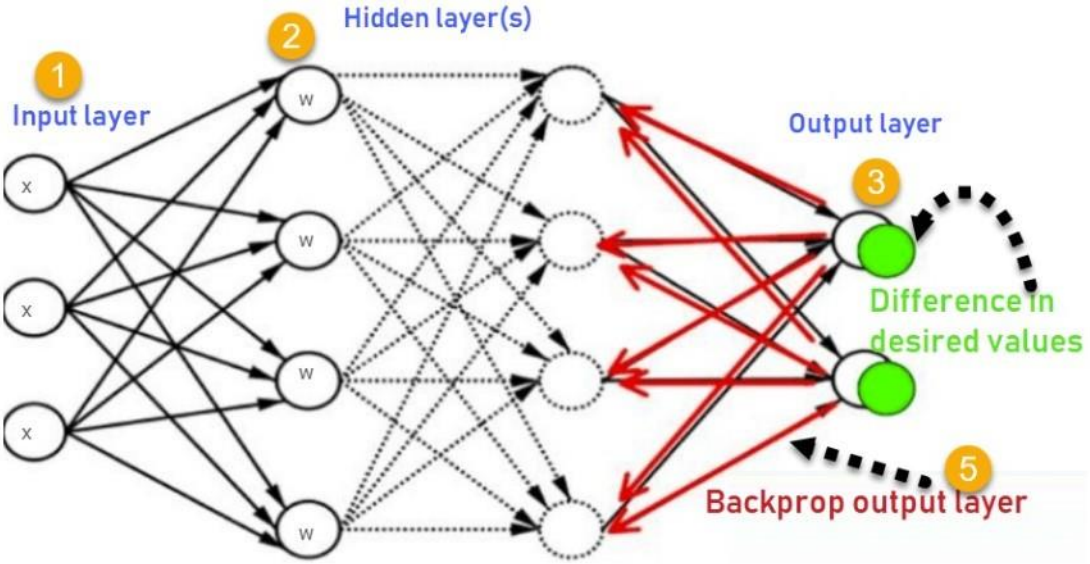
## 2.4Backpropagation Algorithm:

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

### How Backpropagation Works: Simple Algorithm

Consider the following diagram




How Backpropagation Works

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

   $Error_B$= Actual Output – Desired Output
5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

   Keep repeating the process until the desired output is achieved

**Advantages of Backpropagation are:**

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Code No: **R1642053** **R16** **Set No. 1**

**IV B.Tech II Semester Regular Examinations, September - 2020**

**MACHINE LEARNING**

**(Common to Computer Science and Engineering and Information Technology)**

**Time: 3 hours** **Max. Marks: 70**

***Question paper consists of Part-A and Part-B***
***Answer ALL sub questions from Part-A***
***Answer any FOUR questions from Part-B***
*****

## PART–A *(14 Marks)*

| | | | |
|---|---|---|---|
| 1. | a) | Define binary Classification. | [2] |
| | b) | Describe the performance of Multi-class classifier. | [3] |
| | c) | What is a decision tree? | [2] |
| | d) | What is Minkowski distance? | [2] |
| | e) | What is discriminative probabilistic model? | [2] |
| | f) | What is the representational power of perceptron? | [3] |

## PART–B *(4x14 = 56 Marks)*

| | | | |
|---|---|---|---|
| 2. | a) | What are the different types of a Machine Learning models? | [7] |
| | b) | Explain about Feature Construction and Transformation. | [7] |
| 3. | a) | How to handle more than two classes in beyond Binary Classification. | [7] |
| | b) | Explain the following<br>i. One-versus-one voting.<br>ii. Loss based decoding.<br>iii. Coverage counts as scores. | [7] |
| 4. | a) | Explain Rule set for Ranking and Probability estimation. | [7] |
| | b) | Discuss in detail about Learning Ordered Rule Lists. | [7] |
| 5. | a) | Discuss in detail about Soft Margin SVM. | [7] |
| | b) | Describe Nearest-Neighbor Classification in detail. | [7] |
| 6. | a) | Write detailed note on Feature Transformations. | [7] |
| | b) | Explain about normal distribution with the help of sample data. | [7] |
| 7. | a) | Explain about Principle Component Analysis in detail. | [7] |
| | b) | Discuss in detail about representation of Neural Networks. | [7] |

Code No: **R1642053** **R16** **Set No. 2**

**IV B.Tech II Semester Regular Examinations, September - 2020**

**MACHINE LEARNING**

**(Common to Computer Science and Engineering and Information Technology)**

**Time: 3 hours** **Max. Marks: 70**

***Question paper consists of Part-A and Part-B***
***Answer ALL sub questions from Part-A***
***Answer any FOUR questions from Part-B***
*****

## PART–A *(14 Marks)*

1. a) What is Scoring Classifier? [2]
   b) What is unsupervised learning? [3]
   c) Define Feature Tree. [2]
   d) What is Support Vector Regression? [3]
   e) Write a short note on random forests. [2]
   f) Write a short note on PCA? [2]

## PART–B *(4x14 = 56 Marks)*

2. a) Explain in detail about geometric model. [7]
   b) Explain the two uses of features in machine learning. [7]

3. a) Explain the following
      i. most general consistent hypothesis.
      ii. closed concepts in path through the hypothesis . [7]
   b) Write in detailed note on Regression. [7]

4. a) Explain in detail about ranking and probability estimation tree. [7]
   b) Discuss about First-Order rule learning in detail. [7]

5. a) Explain about the Least-Squares method? [7]
   b) Discuss in detail about Distance Based Clustering. Write its importance in machine learning. [7]

6. a) Write about Probabilistic models for categorical data. [7]
   b) Discuss about the Normal Distribution and its Geometric interpretations? [7]

7. a) Explain how dimensionality reduction takes place using PCA. [7]
   b) Describe in detail about neural networks role in machine learning. [7]

Code No: **R1642053** **R16** **Set No. 3**

**IV B.Tech II Semester Regular Examinations, September - 2020**

**MACHINE LEARNING**

**(Common to Computer Science and Engineering and Information Technology)**

**Time: 3 hours** **Max. Marks: 70**

***Question paper consists of Part-A and Part-B***
***Answer ALL sub questions from Part-A***
***Answer any FOUR questions from Part-B***
*****

## PART–A *(14 Marks)*

| | | | |
|---|---|---|---|
| 1. | a) | Write short notes on Geometric model. | [3] |
| | b) | What are the Descriptive models? | [2] |
| | c) | What is Ranking? | [2] |
| | d) | What is Univariate Linear Regression? | [2] |
| | e) | What is Cumulative Probability Distribution? | [2] |
| | f) | List the applications of Neural Networks in Machine Learning. | [3] |

## PART–B *(4x14 = 56 Marks)*

| | | | |
|---|---|---|---|
| 2. | a) | List the problems that can be solved with machine learning. | [7] |
| | b) | Explain about binary classification and related tasks. | [7] |
| 3. | a) | Find least general conjunctive generalization of two conjunctions, employing internal disjunction. | [7] |
| | b) | How to learn a conjunction of horn clauses from membership, equivalence and also explain algorithm for it? | [7] |
| 4. | a) | Distinguish between regression and clustering trees. | [7] |
| | b) | Explain in detail about descriptive rule learning. | [7] |
| 5. | a) | Explain about K-means algorithm with an example. | [7] |
| | b) | With an example explain Hierarchical clustering? | [7] |
| 6. | a) | Explain the probabilistic models with hidden variables. | [7] |
| | b) | What is Ensemble modeling? Discuss about Bagging and Boosting. | [7] |
| 7. | a) | List and explain in detail about appropriate problems for Neural Network learning. | [7] |
| | b) | Explain in detail about multilayer neural networks and back propagation algorithm. | [7] |

Code No: **R1642053** **R16** **Set No. 4**

**IV B.Tech II Semester Regular Examinations, September - 2020**

**MACHINE LEARNING**

**(Common to Computer Science and Engineering and Information Technology)**

**Time: 3 hours** **Max. Marks: 70**

***Question paper consists of Part-A and Part-B***
***Answer ALL sub questions from Part-A***
***Answer any FOUR questions from Part-B***
*****

## PART–A *(14 Marks)*

| | | | |
|---|---|---|---|
| 1. | a) | Compare Supervised and unsupervised learning. | [2] |
| | b) | What is descriptive learning? | [2] |
| | c) | What are the functions used in Decision Tree? | [2] |
| | d) | Write a short note on Distance based clustering. | [3] |
| | e) | What is boosting? | [2] |
| | f) | What is dimensionality reduction? | [3] |

## PART–B *(4x14 = 56 Marks)*

| | | | |
|---|---|---|---|
| 2. | a) | Explain about Grouping and Grading models. | [7] |
| | b) | Describe in detail about the role of features in Machine Learning. | [7] |
| 3. | a) | Discuss about beyond Conjunctive concepts using first-order logic. | [7] |
| | b) | Write in detailed note on multi class Probabilities from Coverage counts. | [7] |
| 4. | a) | Explain in detail about Decision Tree with an example. | [7] |
| | b) | Write in detailed note on Regression Trees. | [7] |
| 5. | a) | How to obtain the probabilities from Linear Classifiers? Explain. | [7] |
| | b) | Explain in detail about Kernel Perceptron. | [7] |
| 6. | a) | Write a note on Feature construction and selection. | [7] |
| | b) | Describe about Probabilistic models used for categorical data. | [7] |
| 7. | a) | Explain in detail about multilayer neural network? | [7] |
| | b) | Explain how dimensionality is reduced using PCA. | [7] |