MACHINE LEARNING

[COURSE MATERIAL FOR B.TECH[CSE] 8TH SEMESTER]

by
Dr. Ashok Kumar Yadav
Department of Computer Science and Engineering
Amity School of Engineering and Technology
GGSIP University, New Delhi

2018

This Course Material

is dedicated to

my Wife

Kalpna Yadav,

whose deepest love and

best wishes always support me.

"If you salute your Duty, You no need to Salute Anybody,

But if you pollute your Duty, you have to Salute Everybody."

A.P.J. Abdul Kalam

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

About Author

Dr. Ashok Kumar Yadav is working as Assistant Professor in the Department of Computer Science and Engineering at Amity School of Engineering and Technology, New Delhi since 2004. He has completed his Ph.D. from Department of Computer Science and Engineering, University Institute of Engineering and Technology, Maharishi Dayanand University, Rohtak, India. He received his B.Sc. in Computer Science in 1999, M.Sc. in Computer Science (Software) in 2001 and M.Tech. in Computer Science and Engineering in 2003 from Kurukshetra University. His research interest in Artificial Neural Network, Genetic Algorithm, Machine Learning Algorithms, Fuzzy Logic and PSO for different applications like Image Processing, Digital Watermarking and Information Hiding. He has published more than 15 papers in International Journal like Multimedia Tools and Application, International Journal of Computer Electrical Automation Control and Information Engineering, International Journal of Machine Learning & Cybernetics and many International Conferences.

Contents

List of Figures

List of Tables

UNIT I

1.1 Concept of learning system

Machine learning is a subfield of computer science, but is often also referred to as predictive analytics, or predictive modeling. Its goal and usage is to build new and/or leverage existing algorithms to learn from data, in order to build generalizable models that give accurate predictions, or to find patterns, particularly with new and unseen similar data.
A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.
A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percentage of games won against opponents
- Training experience E: playing practice games against itself.

A handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P:percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

A robot driving learning problem:

- Task T: driving on public four-lane highway using vision sensors
- Performance measure P: average distance travelled before an error
- Training experience E: a sequence of image and steering commands recorded while observing a human driver.

1.2 Goals of Machine Learning

The primary goal of machine learning research is to develop general purpose algorithms of practical value. Such algorithms should be efficient. As usual, as computer scientists, we care about time and space efficiency. But in the context of learning, we also care a great deal about another precious resource, namely, the amount of data that is required by the learning algorithm. Learning algorithms should also be as general purpose as possible. We are looking for algorithms that can be easily applied to a broad class of learning problems, such as those listed above.

Of primary importance, we want the result of learning to be a prediction rule that is as accurate as possible in the predictions that it makes. Occasionally, we may also be interested in the interpretability of the prediction rules produced by

learning. In other words, in some contexts (such as medical diagnosis), we want the computer to find prediction rules that are easily understandable by human experts.

Machine learning can be thought of as "programming by example." What is the advantage of machine learning over direct programming? First, the results of using machine learning are often more accurate than what can be created through direct programming. The reason is that machine learning algorithms are data driven, and are able to examine large amounts of data. On the other hand, a human expert is likely to be guided by imprecise impressions or perhaps an examination of only a relatively small number of examples. Imagine a dataset as a table, where the rows are each observation (aka measurement, data point, etc), and the columns for each observation represent the features of that observation and their values. At the outset of a machine learning project, a dataset is usually split into two or three subsets. The minimum subsets are the training and test datasets, and often an optional third validation dataset is created as well. Once these data subsets are created from the primary dataset, a predictive model or classifier is trained using the training data, and then the model's predictive accuracy is determined using the test data. As mentioned, machine learning leverages algorithms to automatically model and find patterns in data, usually with the goal of predicting some target output or response. These algorithms are heavily based on statistics and mathematical optimization. Optimization is the process of finding the smallest or largest value (minima or maxima) of a function, often referred to as a loss, or cost function in the minimization case. One of the most popular optimization algorithms used in machine learning is called gradient descent, and another is known as the the normal equation. In a nutshell, machine learning is all about automatically learning a highly accurate predictive or classifier model, or finding unknown patterns in data, by leveraging learning algorithms and optimization techniques.

1.3 Applications of Machine Learning

Machine learning algorithms are used primarily for the following types of output:

- Clustering (Unsupervised)
- Two-class and multi-class classification (Supervised)
- Regression: Univariate, Multivariate, etc. (Supervised)
- Anomaly detection (Unsupervised and Supervised)
- Recommendation systems (aka recommendation engine)

Specific algorithms that are used for each output type are discussed in the next section, but first, let's give a general overview of each of the above output, or problem types. As discussed, clustering is an unsupervised technique for discovering the composition and structure of a given set of data. It is a process of clumping data into clusters to see what groupings emerge, if any. Each cluster is characterized by a contained set of data points, and a cluster centroid. The cluster centroid is basically the mean (average) of all of the data points that the cluster contains, across all features. Classification problems involve placing a data point (aka observation) into a pre-defined class or category. Sometimes classification problems simply assign a class to an observation, and in other cases the goal is to estimate the probabilities that an observation belongs to each of the given classes. A great example of a two-class classification is assigning the class of Spam or Ham to an incoming email, where ham just means 'not spam'. Multi-class classification just means more than two possible classes. So in the spam example, perhaps a third class would be 'Unknown'. Regression is just a fancy word for saying that a model

will assign a continuous value (response) to a data observation, as opposed to a discrete class. A great example of this would be predicting the closing price of the Dow Jones Industrial Average on any given day. This value could be any number, and would therefore be a perfect candidate for regression. Note that sometimes the word regression is used in the name of an algorithm that is actually used for classification problems, or to predict a discrete categorical response (e.g., spam or ham). A good example is logistic regression, which predicts probabilities of a given discrete value. Another problem type is anomaly detection. While we'd love to think that data is well behaved and sensible, unfortunately this is often not the case. Sometimes there are erroneous data points due to malfunctions or errors in measurement, or sometimes due to fraud. Other times it could be that anomalous measurements are indicative of a failing piece of hardware or electronics. Sometimes anomalies are indicative of a real problem and are not easily explained, such as a manufacturing defect, and in this case, detecting anomalies provides a measure of quality control, as well as insight into whether steps taken to reduce defects have worked or not. In either case, there are times where it is beneficial to find these anomalous values, and certain machine learning algorithms can be used to do just that. The final type of problem is addressed with a recommendation system, or also called recommendation engine. Recommendation systems are a type of information filtering system, and are intended to make recommendations in many applications, including movies, music, books, restaurants, articles, products, and so on. The two most common approaches are content-based and collaborative filtering. Two great examples of popular recommendation engines are those offered by Netflix and Amazon. Netflix makes recommendations in order to keep viewers engaged and supplied with plenty of content to watch. In other words, to keep people using Netflix. They do this with their "Because you watched ... ", "Top Picks for Alex", and "Suggestions for you" recommendations. Amazon does a

similar thing in order to increase sales through up-selling, maintain sales through user engagement, and so on. They do this through their "Customers Who Bought This Item Also Bought", "Recommendations for You, Alex", "Related to Items You Viewed", and "More Items to Consider"recommendations.

The value of machine learning technology has been recognized by companies across several industries that deal with huge volumes of data. By leveraging insights obtained from this data, companies are able work in an efficient manner to control costs as well as get an edge over their competitors. This is how some sectors / domains are implementing machine learning :-

- **Financial Services:** Companies in the financial sector are able to identify key insights in financial data as well as prevent any occurrences of financial fraud, with the help of machine learning technology. The technology is also used to identify opportunities for investments and trade. Usage of cyber surveillance helps in identifying those individuals or institutions which are prone to financial risk, and take necessary actions in time to prevent fraud.
- **Marketing and Sales:** Companies are using machine learning technology to analyze the purchase history of their customers and make personalized product recommendations for their next purchase. This ability to capture, analyze, and use customer data to provide a personalized shopping experience is the future of sales and marketing.
- **Government:** Government agencies like utilities and public safety have a specific need FOR Ml, as they have multiple data sources, which can be mined for identifying useful patterns and insights. For example sensor data can be analyzed to identify ways to minimize costs and increase efficiency.

Furthermore, ML can also be used to minimize identity thefts and detect fraud.

- **Healthcare:** With the advent of wearable sensors and devices that use data to access health of a patient in real time, ML is becoming a fast-growing trend in healthcare. Sensors in wearable provide real-time patient information, such as overall health condition, heartbeat, blood pressure and other vital parameters. Doctors and medical experts can use this information to analyze the health condition of an individual, draw a pattern from the patient history, and predict the occurrence of any ailments in the future. The technology also empowers medical experts to analyze data to identify trends that facilitate better diagnoses and treatment.

- **Transportation:** Based on the travel history and pattern of traveling across various routes, machine learning can help transportation companies predict potential problems that could arise on certain routes, and accordingly advise their customers to opt for a different route. Transportation firms and delivery organizations are increasingly using machine learning technology to carry out data analysis and data modeling to make informed decisions and help their customers make smart decisions when they travel.

- **Oil and Gas:** This is perhaps the industry that needs the application of machine learning the most. Right from analyzing underground minerals and finding new energy sources to streaming oil distribution, ML applications for this industry are vast and are still expanding.

1.4 Aspects of Training Data

Data pre-processing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we preprocess our data before feeding it into our model. The process for getting data ready for a machine learning algorithm can be summarized in these steps:

1 Select Data

2 Pre-process Data

3 Transform Data

1.4.1 Select Data

Selecting the right dataset for Machine learning is very important to make the AI model functional with right approach. Though selecting the right quality and amount of data is challenging task but there are few rules needs to be followed for machine learning on big data. There is always a strong desire for including all data that is available, that the maxim "more is better" will hold. This may or may not be true.

We need to consider what data we actually need to address the question or problem we are working on. Make some assumptions about the data we require and be careful to record those assumptions so that we can test them later if needed.

Below are some questions to help we think through this process:

a What is the extent of the data we have available? For example, through time, database tables, connected systems. Ensure we have a clear picture of everything that we can use.

b What data is not available that we wish we had available? For example, data that is not recorded or cannot be recorded. We may be able to derive or simulate this data.

c What data don't we need to address the problem? Excluding data is almost always easier than including data. Note down which data we excluded and why.

It is only in small problems, like competition or toy datasets where the data has already been selected for us. Unstructured data such as images, text or video will eventually need to be converted into a data frame before applying predictive methods, so these metrics listed below apply to all types of data during the machine learning process.

n: Usually the first characteristic of interest in a dataset is its size, N measured as the number of rows or examples.

d: the next descriptor of the data is its dimension, d measured by the number of columns or attributes

k: this descriptor applies only to classification problems. k represents the number of classes. Today most classification problems are binary or k=2. But it is not difficult to envision situations where $k >> 2$.

m: is the ratio of number of samples of the minority class to number of samples of the majority class in a 2-class problem. In a multiclass problem ($k > 2$), this could be the ratio of the number of samples of a given class to the rest of the samples.

1.4.2 Pre-Process Data

The collected data cannot be used directly for performing analysis process. Whenever the data is gathered from different sources, it is collected in raw format which is not feasible for the analysis. Therefore, certain steps are executed to convert the data into a small clean data set. These are the common data pre-processing methods:

1 **Formatting:** The data you have selected may not be in a format that is suitable for you to work with. The data may be in a relational database and you would like it in a flat file, or the data may be in a proprietary file format and you would like it in a relational database or a text file.

2 **Cleaning:** Cleaning data is the removal or fixing of missing data. There may be data instances that are incomplete and do not carry the data you believe you need to address the problem. These instances may need to be removed. Additionally, there may be sensitive information in some of the attributes and these attributes may need to be anonymized or removed from the data entirely.

3 **Sampling:** There may be far more selected data available than you need to work with. More data can result in much longer running times for algorithms and larger computational and memory requirements. You can take a smaller representative sample of the selected data that may be much faster for exploring and prototyping solutions before considering the whole dataset.

4 **Binarize Data (Make Binary):** We can transform our data using a binary threshold. All values above the threshold are marked 1 and all equal to or below are marked as 0. This is called binarizing your data or threshold your

data. It can be useful when you have probabilities that you want to make crisp values. It is also useful when feature engineering and you want to add new features that indicate something meaningful.

5 **Standardize Data:** Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

6 **Dimensionality Reduction:** When data sets become large in the number of predictor variables or the number of instances, data mining algorithms face the curse of dimensionality problem. It is a serious problem as it will impede the operation of most data mining algorithms as the computational cost rise. This section will underline the most influential dimensionality reduction algorithms according to the division established into Feature Selection (FS) and space transformation-based methods.

1.4.3 Transform Data

The final step is to transform the process data. This step is used to convert the raw data into a specified format according to the need of the model. The methods used for transformation of data are given below :–

1 **Rescale Data:** When our data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale. This is useful for optimization algorithms in used in the core of machine learning algorithms like gradient descent. It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like K-Nearest Neighbours.

2 **Decomposition:** There may be features that represent a complex concept that may be more useful to a machine learning method when split into the constituent parts. An example is a date that may have day and time components that in turn could be split out further. Perhaps only the hour of day is relevant to the problem being solved. consider what feature decompositions you can perform.

3 **Aggregation:** There may be features that can be aggregated into a single feature that would be more meaningful to the problem you are trying to solve. For example, there may be a data instances for each time a customer logged into a system that could be aggregated into a count for the number of logins allowing the additional instances to be discarded. Consider what type of feature aggregations could perform.

Data preparation is a large subject that can involve a lot of iterations, exploration and analysis. Getting good at data preparation will make you a master at machine learning. For now, just consider the questions raised in this post when preparing data and always be looking for clearer ways of representing the problem you are trying to solve.

1.5 Concept Learning and Concept Representation

Conceptual learning is an educational method that centers on big-picture ideas and learning how to organize and categorize information. Unlike more traditional learning models which concentrate on the ability to recall specific facts (such as

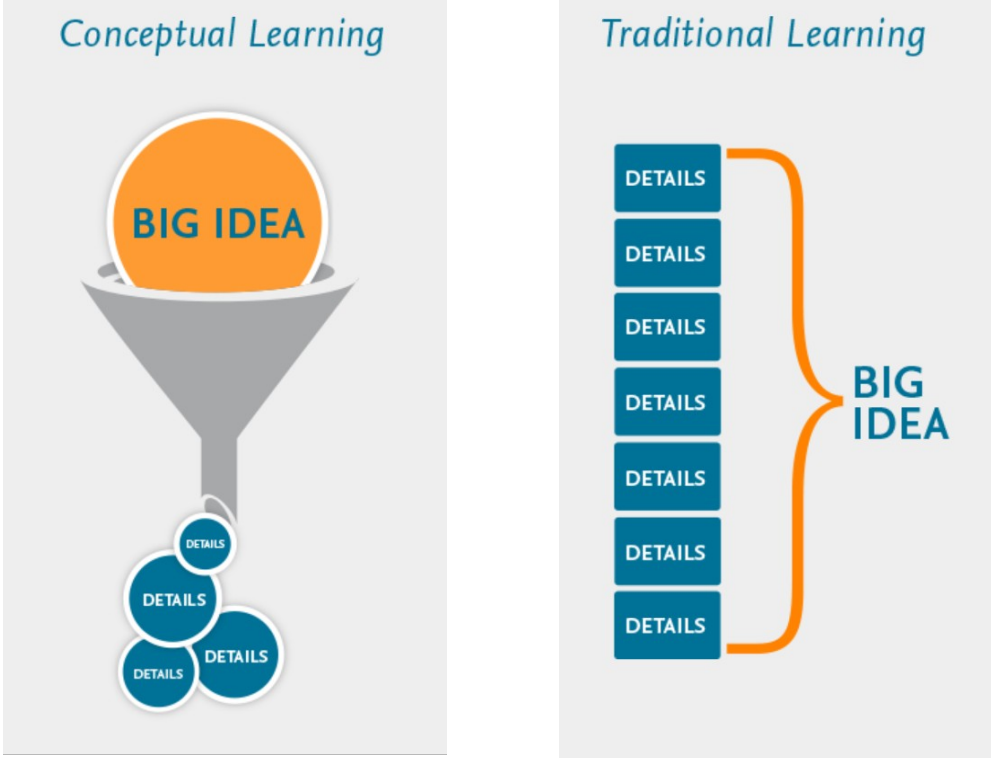

Fig. 1.1 Conceptual Learning and Traditional Learning

the dates of an event or the twenty possible causes of a particular illness), conceptual learning focuses on understanding broader principles or ideas (what we call "concepts") that can later be applied to a variety of specific examples.

To some, conceptual learning can be seen as more of a top-down approach versus the bottom- up model used in more traditional learning (Fig. 1.1). To others who view traditional learning as rote memorization of facts and figures, conceptual learning is seen as a means for getting students to think more critically about the new subjects and situations they encounter. The main goal of representation learning or feature learning is to find an appropriate representation of data in order to perform a machine learning task.

In particular, deep learning exploits this concept by its very nature. In a neural network, each hidden layer maps its input data to an inner representation that

tends to capture a higher level of abstraction. These learnt features are increasingly more informative through layers towards the machine learning task that we intend to perform (e.g. classification).

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.

1.5.1 Concepts and Exemplars

Concepts are mental categories for facts, objects, events, people, ideas — even skills and competencies — that have a common set of features across multiple situations and contexts. Concepts can range from simple to complex according to how easily they can be defined.

Examples of concepts :-

Concrete Concepts have aspects or dimensions that are easily seen, heard, or touched. Fruit would be an example of a concrete concept due to its tangible characteristics of being seed-associated, fleshy, and plant-derived.

Semi-concrete Concepts have some combination of concrete and non-concrete characteristics. Take the semi-concrete concept of a politician, for instance. Some characteristics of a politician could be concrete, such as a holder or candidate for an elected office. However, other characteristics may not be as concrete, such as one who serves the public.

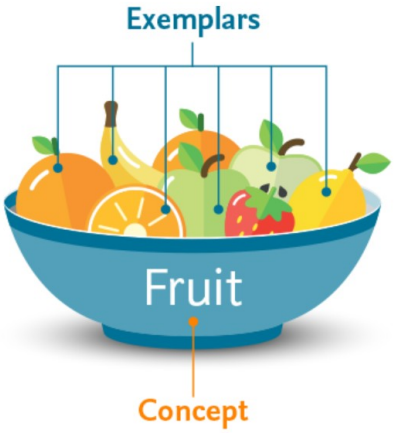

Fig. 1.2 Concept of Fruit

Abstract Concepts do not have many (if any) absolute characteristics that are easy to comprehend with the senses. Unlike concrete and semi-concrete concepts, abstract concepts are not explained by a list of well-defined rules or characteristics. More often, they are understood by mental images or beliefs about its characteristics. Love would be a good example of an abstract concept, as the characteristics of love might differ from one person to the next.

So if concepts are the broad principles or classifications, **Exemplars** then, are the "typical examples" or "excellent models" of that principle. For example, if you are teaching about the concept of fruit, then some good exemplars would be apples, oranges, and bananas (Fig. 1.2). If love is the concept at hand, depending on the type of course you are teaching, some exemplars to use could be the relationship of a mother and daughter, or a group of friends.

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)

Inductive Learning Hypothesis

As we discussed earlier, the ultimate goal of concept learning is to identify a hypothesis H identical to target concept C over data set X with the only available information about C being its value over X. Our algorithm can guarantee that it best fits the training data. In other words:

"Any hypothesis found approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples."

For example, whether a person goes to a movie is based on four binary features with two values (true or false):

1. Has money
2. Has free time
3. It's a holiday
4. Has pending work

With the training data, we have with two data objects as positive samples and one as negative:

1. $x1 :< true, true, false, false >: +ve$

2. $x2 :< true, false, false, true >: +ve$

3. $x3 :< true, false, false, true >: -ve$

Hypothesis Notations

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. Ø(represents a hypothesis that rejects all)

2. $<?, ?, ?, ? >$ (accepts all) 3. $< true, false, ?, ? >$ (accepts some) The hypothesis Ø 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 (Ø).

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 $h <- < \emptyset, \emptyset, \emptyset, \emptyset >$.

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 $< \emptyset, \emptyset, \emptyset, \emptyset >$, 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:

$< true, true, false, true > \emptyset < true, true, false, false >=< true, true, false, ? >$

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.

Example of working with conceptual learning:-

Consider you are tasked to classify a set of shapes. The way you do that is to find unique characteristics of each of your input shape. An example is number of corners (or vertices). Circle has 0, Triangle has 3 and a square has 4 Fig. 1.4.

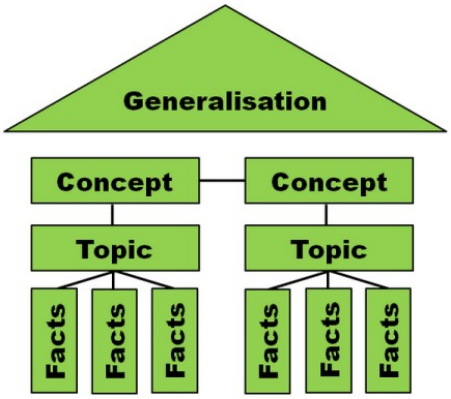

Fig. 1.3 Generalisation and Concept

Fig. 1.4 Example of conceptual learning

Your system may work something like this:

Input - An image

Representation - No of corners in the image (you might use tools like openCV)

Model - Gets an input representation or feature (e.g. no of corners) and applies rules to detect the shape. (Like if feature input is 0 then circle).

Output - We have a working system.

But what happens when you start getting inputs as cuboid, trapezium or all sort of shapes. You would realize that designing features gets not just difficult, time consuming and requires a deep domain expertise as you start working with real world use-cases. E.g. consider you need to recognize equilateral versus obtuse triangle (both have same no of corners). It is observed that designing features is a complex process and the way to solve that is how our brain is able to design these

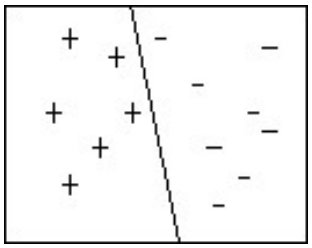

Fig. 1.5 A linear discrimination between two classes

features. In this example, you looked at shapes and decided that no of corners seems like a good way to uniquely classify images. It is this task of brain that is performed by feature or representation learning algorithms. Deep learning is just one of such methods. Deep Learning learns tries to learn features on its own. All you would like to do is pass an image and let the system learn features like we do.

1.6 Function Approximation

Statistical and connectionist approaches to machine learning are related to function approximation methods in mathematics. For the purposes of illustration let us assume that the learning task is one of classification. That is, we wish to find ways of grouping objects in a universe. In Fig. 1.5 we have a universe of objects that belong to either of two classes '+' or '-'. By function approximation, we describe a surface that separates the objects into different regions. The simplest function is that of a line and linear regression methods and perceptrons are used to find linear discriminant functions. A perceptron is a simple pattern classifier. Given a binary input vector, x, a weight vector, w, and a threshold value, T, if,

$$\sum_i w_i \times x_i > T$$

Then, the output is 1, indicating membership of a class, otherwise it is 0, indicating exclusion from the class. Clearly, $w.x - T$ describes a hyper plane and the goal of perceptron learning is to find a weight vector, w, that results in correct classification for all training examples. The perceptron learning algorithm is quite straight forward. All the elements of the weight vector are initially set to 0. For each training example, if the perceptron outputs 0 when it should output 1 then add the input vector to the weight vector; if the perceptron outputs 1 when it should output 0 then subtract the input vector to the weight vector; otherwise, do nothing. This is repeated until the perceptron yields the correct result for each training example. The algorithm has the effect of reducing the error between the actual and desired output.

The perceptron is an example of a linear threshold unit (LTU). A single LTU can only recognize one kind of pattern, provided that the input space is linearly separable. If we wish to recognize more than one pattern, several LTU's can be combined. In this case, instead of having a vector of weights, we have an array. The output will now be a vector:

$$u = Wx$$

Where, each element of u indicates membership of a class and each row in W is the set of weights for one LTU. This architecture is called a pattern associators. LTU's can only produce linear discriminant functions and consequently, they are limited in the kinds of classes that can be learned. However, it was found that by cascading pattern associators, it is possible to approximate decision surfaces that are of a higher order than simple hyper planes. In cascaded system, the outputs of one pattern associators are fed into the inputs of another, thus:

$$u = W(Vx)$$

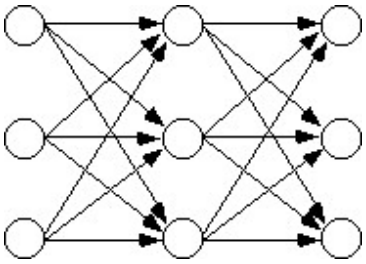

Fig. 1.6 A multi-layer network

To facilitate learning, a further modification must be made. Rather than using a simple threshold, as in the perceptron, multi-layer networks Fig. 1.6 usually use a non-linear threshold such a sigmoid function, such as

$$\frac{1}{1+e^{-x}}$$

Like perceptron learning, back-propagation attempts to reduce the errors between the output of the network and the desired result. However, assigning blame for errors to hidden units (ie. nodes in the intermediate layers), is not so straightforward. The error of the output units must be propagated back through the hidden units. The contribution that a hidden unit makes to an output unit is related strength of the weight on the link between the two units and the level of activation of the hidden unit when the output unit was given the wrong level of activation. This can be used to estimate the error value for a hidden unit in the penultimate layer, and that can, in turn, be used in make error estimates for earlier layers. Despite the non-linear threshold, multi-layer networks can still be thought of as describing a complex collection of hyper planes that approximate the required decision surface.

Version Space

A version space is a hierarchical representation of knowledge that enables you to keep track of all the useful information supplied by a sequence of learning examples

without remembering any of the examples. The version space method is a concept learning process accomplished by managing multiple models within a version space.

Version Space Characteristics

Tentative heuristics are represented using version spaces. A version space represents all the alternative plausible descriptions of a heuristic. A plausible description is one that is applicable to all known positive examples and no known negative example. A version space description consists of two complementary trees:

1. One that contains nodes connected to overly general models, and
2. One that contains nodes connected to overly specific models.

Node values/attributes are discrete.

Fundamental Assumptions

1. The data is correct; there are no erroneous instances.
2. A correct description is a conjunction of some of the attributes with values.

Diagrammatical Guidelines

There is a generalization tree and a specialization tree. Each node is connected to a model. Nodes in the generalization tree are connected to a model that matches everything in its subtree. Nodes in the specialization tree are connected to a model that matches only one thing in its subtree. Links between nodes and their models denote:

- generalization relations in a generalization tree, and
- specialization relations in a specialization tree.

1.7 Types of Learning

1.7.1 Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). Supervised learning, in the context of artificial intelligence (AI) and machine learning, is a type of system in which both input and desired output data are provided. Input and output data are labelled for classification to provide a learning basis for future data processing. Supervised machine learning systems provide the learning algorithms with known quantities to support future judgements, Chatbots, self-driving cars, facial recognition programs, systems and robots are among the systems that may use either supervised or unsupervised learning. Supervised learning systems are mostly associated with retrieval-based AI, but they may also be capable of using a generative learning model. In supervised learning for image processing, for example, an AI system might be provided with labelled pictures of vehicles in categories such as cars and trucks. After an enough observation, the system should be able to distinguish between and categorize unlabelled images, at which time training can be said to be complete. Supervised learning models have some advantages over the unsupervised approach, but they also have limitations. The systems are more likely to make judgements that humans can relate to, for example, because humans have provided the basis for decisions. However, in the case of a retrieval-based method, supervised learning systems have trouble dealing with new information. If a system with categories for cars and trucks is presented with

a bicycle, for example, it would have to be incorrectly lumped in one category or the other. If the AI system was generative, however, it may not know what the bicycle is but would be able to recognize it as belonging to a separate category. Supervised learning is the Data mining task of inferring a function from labelled training data. The training data consist of a set of training example. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instance . This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. There are some of the points:-

- You already learn from your previous work about the physical characters of fruits.
- So, arranging the same type of fruits at one place is easy now.
- Your previous work is called as training data in data mining.
- so, you already learn the things from your train data, this is because of response variable.
- Response variable mean just a decision variable.
- You can observe response variable below (FRUIT NAME) .
- Suppose you have taken a new fruit from the basket then you will see the size , colour and shape of that fruit.

- If size is Big , colour is Red , shape is rounded shape with a depression at the top, you will conform the fruit name as apple, and you will put in apple group.
- Likewise, for other fruits also.
- Job of groping fruits was done and happy ending.
- You can observe in the table that a column was labelled as "FRUIT NAME" this is called as response variable.
- If you learn the thing before from training data and then applying that knowledge to the test data(for new fruit), This type of learning is called as Supervised Learning.
- Classification comes under Supervised learning.

1.7.1.1 Supervised Learning Algorithms

- Support Vector Machines
- Linear regression
- Logistic regression
- Naive Bayes
- Linear discriminant analysis
- Decision trees
- K-Nearest neighbor algorithm
- Neural Networks (Multilayer perceptron)

- Similarity learning

1.7.1.2 Steps taken to implement supervised algorithm

In order to solve a given problem of supervised learning, one has to perform the following steps:

- Determine the type of training examples. Before doing anything else, the user should decide what kind of data is to be used as a training set. In case of Handwriting Analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
- Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.
- Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.
- Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use support vector machines or decision trees.

- Complete the design. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation.

- Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set

1.7.1.3 Major issues in supervised learning

Bias-variance tradeoff

A first issue is the tradeoff between bias and variance. Imagine that we have available several different, but equally good, training data sets. A learning algorithm is biased for a particular input . The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance.

Function complexity and amount of training data

The second issue is the amount of training data available relative to the complexity of the "true" function (classifier or regression function). If the true function is simple, then an "inflexible" learning algorithm with high bias and low variance will be able to learn it from a small amount of data. But if the true function is highly complex (e.g., because it involves complex interactions among many different input features and behaves differently in different parts of the input space), then the function will only be learn able from a very large amount of training data and using a "flexible" learning algorithm with low bias and high variance.

Dimensionality of the input space

A third issue is the dimensionality of the input space. If the input feature vectors have very high dimension, the learning problem can be difficult even if the true function only depends on a small number of those features. This is because the many "extra" dimensions can confuse the learning algorithm and cause it to have high variance. Hence, high input dimensional typically requires tuning the classifier to have low variance and high bias This is an instance of the more general strategy of dimensionality reduction, which seeks to map the input data into a lower-dimensional space prior to running the supervised learning algorithm.

Noise in the output values

A fourth issue is the degree of noise in the desired output values (the supervisory target variables). If the desired output values are often incorrect (because of human error or sensor errors), then the learning algorithm should not attempt to find a function that exactly matches the training examples.In such a situation, the part of the target function that cannot be modeled "corrupts" your training data - this phenomenon has been called deterministic noise. When either type of noise is present, it is better to go with a higher bias, lower variance estimator.

1.7.2 Unsupervised Learning

Unsupervised Learning is a class of Machine Learning techniques to find the patterns in data. The data given to unsupervised algorithm are not labeled, which means only the input variables(X) are given with no corresponding output variables. In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data. Yan Lecun, director of AI research, explains that unsupervised learning-teaching machines to learn for themselves without having to be explicitly told if everything they do is right or wrong-is the key to "true"

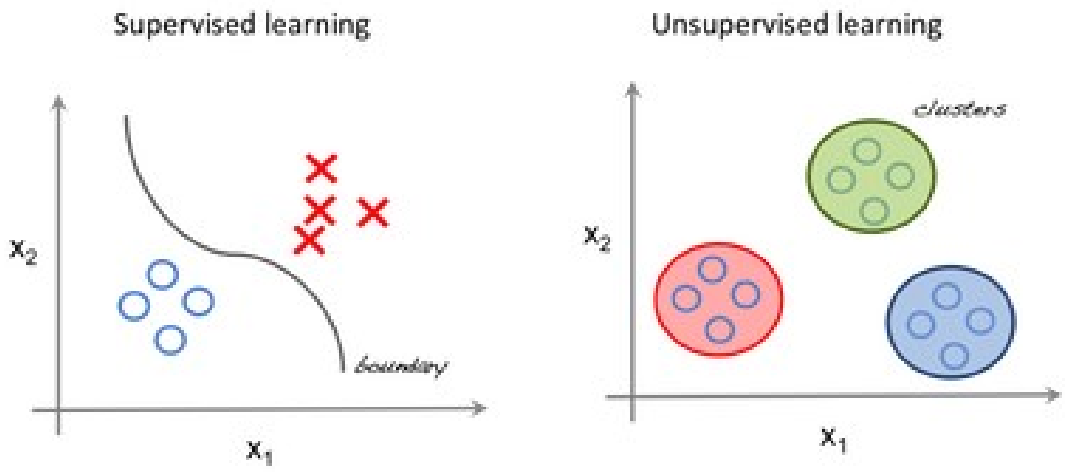

Fig. 1.7 Supervised and Unsupervised Learning

AI. The image to the left in Fig. 1.7 is an example of supervised learning; we use regression techniques to find the best fit line between the features. While in unsupervised learning the inputs are segregated based on features and the prediction is based on which cluster it belonged. Unsupervised Learning mainly divided into two categories:- Clustering and Classification.

1.7.2.1 Clustering

The cluster analysis as a branch of machine learning that groups the data that has not been labelled, classified or categorized. Instead of responding to feedback, cluster analysis identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data.

Common clustering algorithms:

k-Means clustering: partitions data into k distinct clusters based on distance to the centroid of a cluster

Gaussian mixture models: models clusters as a mixture of multivariate normal

density components

Self-organizing maps: uses neural network that learn the topology and distribution of the data

Hidden Markov models: uses observed data to recover the sequence of states.

1.7.2.2 Classification

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y). For example, spam detection in email service providers can be identified as a classification problem. This is s binary classification since there are only 2 classes as spam and not spam. A classifier utilizes some training data to understand how given input variables relate to the class. In this case, known spam and non-spam emails have to be used as the training data. When the classifier is trained accurately, it can be used to detect an unknown email. Classification belongs to the category of supervised learning where the targets also provided with the input data. There are many applications in classification in many domains such as in credit approval, medical diagnosis, target marketing etc.

Classification algorithms:

Decision Tree

Decision tree builds classification or regression models in the form of a tree structure. It utilizes an if-then rule set which is mutually exclusive and exhaustive for classification. The rules are learned sequentially using the training data one at a time. Each time a rule is learned, the tuples covered by the rules are removed. This process is continued on the training set until meeting a termination condition. The

tree is constructed in a top-down recursive divide-and-conquer manner. All the attributes should be categorical. Otherwise, they should be discretized in advance. Attributes in the top of the tree have more impact towards in the classification and they are identified using the information gain concept. A decision tree can be easily over-fitted generating too many branches and may reflect anomalies due to noise or outliers. An over-fitted model has a very poor performance on the unseen data even though it gives an impressive performance on training data. This can be avoided by pre-pruning which halts tree construction early or post-pruning which removes branches from the fully grown tree.

Naive Bayes

Naive Bayes is a probabilistic classifier inspired by the Bayes theorem under a simple assumption which is the attributes are conditionally independent. The classification is conducted by deriving the maximum posterior which is the maximal with the above assumption applying to Bayes theorem. This assumption greatly reduces the computational cost by only counting the class distribution. Even though the assumption is not valid in most cases since the attributes are dependent, surprisingly Naive Bayes has able to perform impressively. Naive Bayes is a very simple algorithm to implement and good results have obtained in most cases. It can be easily scalable to larger datasets since it takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Artificial Neural Networks

Artificial Neural Network is a set of connected input/output units where each connection has a weight associated with it started by psychologists and neurobiologists to develop and test computational analogs of neurons. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples. There are many network architectures available

now like Feed-forward, Convolutional, Recurrent etc. The appropriate architecture depends on the application of the model. For most cases feed-forward models give reasonably accurate results and especially for image processing applications, convolutional networks perform better.

k-Nearest Neighbor (KNN)

k-Nearest Neighbor is a lazy learning algorithm which stores all instances correspond to training data points in n-dimensional space. When an unknown discrete data is received, it analyzes the closest k number of instances saved (nearest neighbors)and returns the most common class as the prediction and for real-valued data it returns the mean of k nearest neighbors. In the distance-weighted nearest neighbor algorithm, it weights the contribution of each of the k neighbors according to their distance using the following query giving greater weight to the closest neighbors.

1.7.2.3 Challenges in Implementing Unsupervised Learning

a) In addition to the regular issues of finding the right algorithms and hardware, unsupervised learning presents a unique challenge: it's difficult to figure out if you're getting the job done or not.

b) In supervised learning, we define metrics that drive decision making around model tuning. Measures like precision and recall give a sense of how accurate your model is, and parameters of that model are tweaked to increase those accuracy scores. Low accuracy scores mean you need to improve, and so on.

c) Since there are no labels in unsupervised learning, it's near impossible to get a reasonably objective measure of how accurate your algorithm is. In clustering for example, how can you know if K-Means found the right clusters? Are you using the right number of clusters in the first place? In supervised learning we can look

to an accuracy score; here you need to get a bit more creative.
d) A big part of the "will unsupervised learning work for me?" question is totally dependent on your business context. In our example of customer segmentation, clustering will only work well if your customers actually do fit into natural groups. One of the best (but most risky) ways to test your unsupervised learning model is by implementing it in the real world and seeing what happens! Designing an A/B test–with and without the clusters your algorithm outputted–can be an effective way to see if it's useful information or totally incorrect.
e) Researchers have also been working on algorithms that might give a more objective measure of performance in unsupervised learning. Check out the below section for some examples.

1.8 Training Dataset

The actual dataset that we use to train the model (weights and biases in the case of Neural Network). The model sees and learns from this data. Machine learning is a subfield of computer science that gives the computer the ability to learn without being explicitly programmed. For this to happen, a machine needs to be "trained" by explicitly feeding it data that has the correct answers attached. This training data will help the machine to connect the patterns in the data to the right answer. Once trained in this way, a machine can now be given test data that has no answers. The machine will then predict the answers based on the training it received. Most data scientists divide their data (with answers, that is historical data) into three portions: training data, cross-validation data and testing data. The training data is used to make sure the machine recognizes patterns in the data, the cross-validation data is used to ensure better accuracy and efficiency of

the algorithm used to train the machine, and the test data is used to see how well the machine can predict new answers based on its training.

1.8.1 How to create training data?

Machine Learning models are trained using data with specific features. The way in which the data is structured helps the models to learn and develop relationship between these features. A well-processed training set is required to build a robust model which in turn generates accurate results. In this article we shall look at some of the ways in which one can build a structured dataset for training. To build a robust model, one has to keep in mind the flow of operations involved in building a quality dataset. The data should be accurate with respect to the problem statement. For example, while trying to determine the height of a person, feature such as age, sex, weight, or the size of the clothes, among others, are to be considered. Here, the person's clothes will account for his/her height, whereas the colour of the clothes and the material will not add any value in this case. Hence these features have very low weightage for predicting the height of a person. A golden rule of machine learning is: Larger the data better the results. There are several steps included in this process:

1 **Data Selection** In this step, one should be concerned about opting the right number of features for the particular dataset. The data should be consistent and should have least number of missing values. If a feature has more than 25 to 30 percent missing values then it is usually considered not fit to be a part of the training set. But there are instances where the relationship between this feature and the Y feature is high. In that case, one has to impute and handle the missing values for better results. For example, let us say an

institution has borrowed a loan from a bank. A feature containing the GDP value of the particular country is available with 30 percent missing values. If one infers that the particular feature has a very high importance to predict whether the institution is able to repay the loan or not, then this feature has to be considered. If the feature does not hold high importance for developing the AI model, one should exclude the data. At the end of this particular step, one should have an idea about how to deal with the preprocessing data.

2 **Data Preprocessing** Once the right data is selected, preprocessing includes selection of the right data from the complete dataset and building a training set. Here, some of the common steps are:

Organise and Format: The data might be scattered in different files, for example, classroom datasets of various grades in a school which needs to be clubbed together to form a dataset. One has to find the relation between these datasets and preprocess to form a dataset of required dimensions. Also if the datasets are in different language they have to be transformed into a universal language before proceeding.

Data Cleaning: This is one of the major steps in data preprocessing. Cleaning refers to mainly dealing with the missing values and removal of unwanted characters from the data. For example, if a feature consists of age of a person, with 4 percent missing values, it can either deleted or replaced. Here, is an in-depth article of how to handle missing in machine learning.

Feature Extraction: This step involves analysis and optimisation of the number of features. One has to find out which features are important for prediction and select them for faster computations and low memory consumption. For example, while dealing with an image classification problem, images with noise (irrelevant images with respect to the dataset) should be

removed.

3 **Data Conversion**

Scaling: This is necessary when the dataset is placed. Considering a linear dataset - bank data. If the feature containing the transaction amount is important, then the data has to be scaled in order to build a robust model. By default in correlation matrix, the Pearson method is used to find the relationship. This might lead to a misunderstanding of the data if it is not scaled by a definite value.

Disintegration and Composition: This step is considered when one needs to split a particular feature to build a better training data for the model to understand. One of the best examples of the data disintegration is splitting up the time-series feature. Where one can extract the days, months, year, hour, minutes, seconds, etc. from a particular sample. And also let us say, the Project ID is IND0002. Here the first three characters refer to the country code and 0002 refer to a categorical value. Separating and processing may result in better accuracy.

Composition: This process involves combining different features to a single feature for more meaningful data. For example, in the Titanic dataset, the prefix of the passengers with Dr, Mr, Miss. etc can be clubbed into a particular age groups of categorical data which adds more weight in predicting the passengers' survival.

one can understand how processed training set helps a machine learning to develop the relationship between the features. This process involves a lot of time, analysis and examination of the data. With a well - structured data, machine learning model can train faster and give robust results.

1.9 Test Dataset

The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets). The test set is generally what is used to evaluate competing models (For example on many Kaggle competitions, the validation set is released initially along with the training set and the actual test set is only released when the competition is about to close, and it is the result of the the model on the Test set that decides the winner). Many a times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled data that spans the various classes that the model would face, when used in the real world. This corresponds to the final evaluation that the model goes through after the training phase (utilizing training and validation sets) has been completed. This step is critical to test the generalizability of the model (Step 3). By using this set, we can get the working accuracy of our model. It is worth mentioning that we need to be subjective and honest by not exposing the model to the test set until the training phase is over. This way, we can consider the final accuracy measure to be reliable. Training a model involves looking at training examples and learning from how off the model is by frequently evaluating it on the validation set. However, the last and most valuable pointer on the accuracy of a model is a result of running the model on the testing set when the training is complete.

1.10 Validation Dataset

Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. We as machine learning engineers use this data to fine-tune the model hyperparameters. Hence the model occasionally sees this data, but never does it "Learn" from this. We use the validation set results and update higher level hyperparameters. So the validation set in a way affects a model, but indirectly.

1.11 Dataset split ratio

Now that you know what these datasets do, you might be looking for recommendations on how to split your dataset into Train, Validation and Test sets. This mainly depends on 2 things. First, the total number of samples in your data and second, on the actual model you are training. Some models need substantial data to train upon, so in this case you would optimize for the larger training sets. Models with very few hyperparameters will be easy to validate and tune, so you can probably reduce the size of your validation set, but if your model has many hyperparameters, you would want to have a large validation set as well(although you should also consider cross validation). Also, if you happen to have a model with no hyperparameters or ones that cannot be easily tuned, you probably don't need a validation set too! All in all, like many other things in machine learning, the train-test-validation split ratio is also quite specific to your use case and it gets

easier to make judgement as you train and build more and more models. Many a times, people first split their dataset into two - Train and Test. After this, they keep aside the Test set, and randomly choose $X\%$ of their Train dataset to be the actual Train set and the remaining $(100 - X)\%$ to be the Validation set, where X is a fixed number(say 80%), the model is then iteratively trained and validated on these different sets. There are multiple ways to do this, and is commonly known as Cross Validation. Basically you use your training set to generate multiple splits of the Train and Validation sets. Cross validation avoids over fitting and is getting more and more popular, with K-fold Cross Validation being the most popular method of cross validation.

1.12 Over fitting

Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data Fig. 1.8. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize. Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns. For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up. The cause

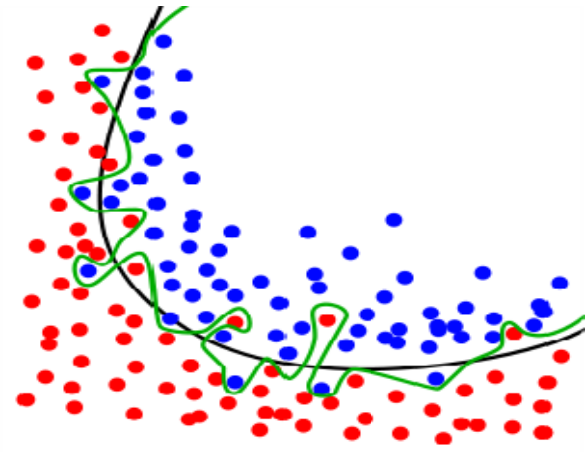

Fig. 1.8 Overfitting

of poor performance in machine learning is either overfitting or underfitting the data. Supervised machine learning is best understood as approximating a target function (f) that maps input variables (X) to an output variable (Y). Y = f(X) This characterization describes the range of classification and prediction problems and the machine algorithms that can be used to address them. An important consideration in learning the target function from the training data is how well the model generalizes to new data. Generalization is important because the data we collect is only a sample, it is incomplete and noisy.

1.12.1 Generalization

In machine learning we describe the learning of the target function from training data as inductive learning. Induction refers to learning general concepts from specific examples which is exactly the problem that supervised machine learning problems aim to solve. This is different from deduction that is the other way around and seeks to learn specific concepts from general rules. Generalization

refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning. The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen. There is a terminology used in machine learning when we talk about how well a machine learning model learns and generalizes to new data, namely overfitting and underfitting. Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms.

1.12.2 Statistical Fit

In statistics, a fit refers to how well you approximate a target function. This is good terminology to use in machine learning, because supervised machine learning algorithms seek to approximate the unknown underlying mapping function for the output variables given the input variables. Statistics often describe the goodness of fit which refers to measures used to estimate how well the approximation of the function matches the target function. Some of these methods are useful in machine learning (e.g. calculating the residual errors), but some of these techniques assume we know the form of the target function we are approximating, which is not the case in machine learning. If we knew the form of the target function, we would use it directly to make predictions, rather than trying to learn an approximation from samples of noisy training data. Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models

ability to generalize. Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns. For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up.

1.12.3 A Good Fit in Machine Learning

Ideally, we want to select a model at the sweet spot between underfitting and overfitting. This is the goal, but is very difficult to do in practice. To understand this goal, we can look at the performance of a machine learning algorithm over time as it is learning a training data. We can plot both the skill on the training data and the skill on a test dataset we have held back from the training process. Over time, as the algorithm learns, the error for the model on the training data goes down and so does the error on the test dataset. If we train for too long, the performance on the training dataset may continue to decrease because the model is overfitting and learning the irrelevant detail and noise in the training dataset. At the same time the error for the test set starts to rise again as the model's ability to generalize decreases. The sweet spot is the point just before the error on the test dataset starts to increase where the model has good skill on both the training dataset and the unseen test dataset. We can perform this experiment with your favorite machine learning algorithms. This is often not useful technique in practice, because by choosing the stopping point for training using the skill on the test dataset it means that the testset is no longer "unseen" or a standalone objective measure. Some knowledge (a lot of useful knowledge) about that data has

leaked into the training procedure. There are two additional techniques you can use to help find the sweet spot in practice: resampling methods and a validation dataset.

1.12.4 Detection of Overfitting

A key challenge with overfitting, and with machine learning in general, is that we can't know how well our model will perform on new data until we actually test it. To address this, we can split our initial dataset into separate training and test subsets. This method can approximate of how well our model will perform on new data. If our model does much better on the training set than on the test set, then we're likely overfitting.

1.12.5 Prevention of Overfitting

Detecting overfitting is useful, but it doesn't solve the problem. Fortunately, you have several options to try. Popular solutions for overfitting:

Cross-validation

Cross-validation is a powerful preventative measure against overfitting. The idea is clever: Use your initial training data to generate multiple mini train-test splits. Use these splits to tune your model. Cross-validation allows you to tune hyper-parameters with only your original training set. This allows you to keep your test set as a truly unseen dataset for selecting your final model.

Train with more data

It won't work every time, but training with more data can help algorithms detect the signal better. In the earlier example of modeling height vs. age in children, it's clear how sampling more schools will help your model. Of course, that's not

always the case. If we just add more noisy data, this technique won't help. That's why you should always ensure your data is clean and relevant.

Remove features

Some algorithms have built-in feature selection. For those that don't, you can manually improve their generalizability by removing irrelevant input features. An interesting way to do so is to tell a story about how each feature fits into the model. This is like the data scientist's spin on software engineer's rubber duck debugging technique, where they debug their code by explaining it, line-by-line, to a rubber duck. If anything doesn't make sense, or if it's hard to justify certain features, this is a good way to identify them. In addition, there are several feature selection heuristics you can use for a good starting point.

Early stopping

When you're training a learning algorithm iteratively, you can measure how well each iteration of the model performs. Up until a certain number of iterations, new iterations improve the model. After that point, however, the model's ability to generalize can weaken as it begins to overfit the training data. Early stopping refers stopping the training process before the learner passes that point. Today, this technique is mostly used in deep learning while other techniques (e.g. regularization) are preferred for classical machine learning.

1.13 Classification families

In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like

identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, biometric identification, document classification etc. Here we have the types of classification algorithms in Machine Learning:

1 Linear Classifiers: Logistic Regression, Naive Bayes Classifier

2 Support Vector Machines

3 Decision Trees

4 Boosted Trees

5 Random Forest

6 Neural Networks

7 Nearest Neighbor

1.13.1 Linear discriminative

Linear Discriminant Analysis (LDA) is a classification method originally developed in 1936 by R. A. Fisher. Linear Discriminant Analysis is a dimensionality reduction technique used as a preprocessing step in Machine Learning and pattern classification applications.

1.13.2 Non-linear discriminative

1.13.3 Decision trees

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and a leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data. Decision Tree Analysis is a general, predictive modelling tool that has applications spanning a number of different areas. In general, decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The decision rules are generally in form of if-then-else statements. The deeper the tree, the more complex the rules and fitter the model.

- Instances: Refer to the vector of features or attributes that define the input space
- Attribute: A quantity describing an instance
- Concept: The function that maps input to output

- Target Concept: The function that we are trying to find, i.e., the actual answer
- Hypothesis Class: Set of all the possible functions
- Sample: A set of inputs paired with a label, which is the correct output (also known as the Training Set)
- Candidate Concept: A concept which we think is the target concept
- Testing Set: Similar to the training set and is used to test the candidate concept and determine its performance

A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers to the question, and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface. Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive in nature and is repeated for every sub tree rooted at the new nodes. A general algorithm for a decision tree can be described as follows:

1 Pick the best attribute/feature. The best attribute is one which best splits or separates the data.

2 Ask the relevant question.

3 Follow the answer path.

4 Go to step 1 until you arrive to the answer.

The best split is one which separates two different labels into two sets.

Calculating information gain

As stated earlier, information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification. In the figure below, we can see that an attribute with low information gain (right) splits the data relatively evenly and as a result doesn't bring us any closer to a decision. Whereas, an attribute with high information gain (left) splits the data into groups with an uneven number of positives and negatives and as a result helps in separating the two from each other. To define information gain precisely, we need to define a measure commonly used in information theory called entropy that measures the level of impurity in a group of examples.

1.13.3.1 Advantages and Disadvantages

Following are the advantages of decision trees: -

1 Easy to use and understand.

2 Can handle both categorical and numerical data.

3 Resistant to outliers, hence require little data preprocessing.

4 New features can be easily added.

5 Can be used to build larger classifiers by using ensemble methods.

Following are the disadvantages of decision trees: -

1 Prone to over fitting.

2 Require some kind of measurement as to how well they are doing.

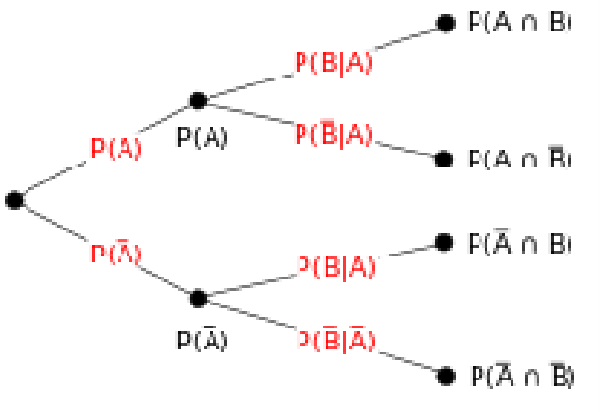

Fig. 1.9 Conditional Model

3 Need to be careful with parameter tuning.

4 Can create biased learned trees if some classes dominate.

1.13.4 Conditional Model

Conditional Probabilistic models is a class of statistical models in which sample data are divided into input and output data and the relation between the two kind of data is studied by modelling the conditional probability distribution of the outputs given the inputs Fig. 1.9. This is in contrast to unconditional models (sometimes also called generative models) where the data is studied by modelling the joint distribution of inputs and outputs. Before introducing conditional models, let us review the main elements of a statistical model:

1 There is a sample ξ, which can be regarded as a realization of a random vector Ξ. (for example, could be a vector collecting the realizations of some independent random variables);

2 The joint distribution function of the sample, denoted by $F_{\Xi}(\xi)$, is not known exactly;

3 The sample ξ is used to infer some characteristics of $F_{\Xi}(\xi)$;

4 A model for Ξ is used to make inferences, where a model is simply a set of joint distribution functions to which $F_{\Xi}(\xi)$ is assumed to belong.

In a conditional model, the sample ξ is partitioned into inputs and outputs: Where, y denotes the vector of outputs and x the vector of inputs.

$$\xi = [y \; x]$$

$$F_{Y|X=x}(y)$$

The object of interest is the conditional distribution function of the outputs given the inputs and specifying a conditional model means specifying a set of conditional distribution functions to which $F_{Y|X=x}(y)$ is assumed to belong. In other words, in a conditional model, the problem of model specification is simplified by narrowing the focus of the statistician's attention on the conditional distribution of the outputs and by ignoring the distribution of the inputs. This can be seen, for example, in the case in which both inputs and outputs are continuous random variables. In such a case, specifying an unconditional model is equivalent to specifying a joint probability density function $f_{X,Y}(x, y)$ for the inputs and the outputs. But a joint density can be seen as the product of a marginal and a conditional density: $f_{X,Y}(x, y) = F_{Y|X=x}(y)f_X(x)$. So, in an unconditional model we explicitly or implicitly specify both the marginal probability density function $f_X(x)$ and the conditional probability density function $f_{X,Y}(x, y)$. On the other hand, in a conditional model, we specify only the conditional $f_{X,Y}(x, y)$ and we leave the marginal $f_X(x)$ unspecified.

Regression and classification

The following distinction is often made, especially in the field of machine learning:

1 If the output is a continuous random variable, then a conditional model is called a regression model;

2 If the output is a discrete random variable, taking finitely many values (typically few), then a conditional model is called a classification model.

1.13.4.1 Linear regression model

The linear regression model is probably the oldest, best understood and most widely used conditional model. In the linear regression model, the response variables y are assumed to be a linear function of the inputs x: $y_i = x_i\beta + \varepsilon_i$. A linear regression model is specified by making assumptions about the error term ε_i. For example, ε_i is often assumed to have a normal distribution with zero mean and to be independent of x_i. In such a case, we have that, conditional on the inputs x_i, the output y_i has a normal distribution with mean $x_i\beta$. As a consequence, the conditional density of y_i is:

$$f_{Y_i|X_i=x_i}(y_i) = \frac{1}{\sqrt{2\pi}}\frac{1}{\sigma}e^{\left(-\frac{1}{2}\frac{(y_i-x_i\beta)^2}{\sigma^2}\right)}$$

where σ^2 is the variance of ε_i. The parameters β and σ^2 are usually unknown and need to be estimated. So, we have a different conditional distribution for each of the values of β and σ^2 that are deemed plausible by the statistician before observing the sample. The set of all these conditional distributions (associated to the different parameters) constitutes the conditional model for (y_i, x_i).

1.13.4.2 Logistic classification model

In the logistic classification model, the response variable is a Bernoulli random variable. It can take only two values, either 1 or 0. It is assumed that the conditional probability mass function of y_i is a non-linear function of the inputs x_i:

$$P_{Y_i|X_i=x_i}(y_i) = \begin{cases} sigm(x_i\beta) & \text{if } y_i = 1 \\ 1 - sigm(x_i\beta) & \text{if } y_i = 0 \\ 0 & otherwise \end{cases}$$

where x_i is a $1 \times K$ vector of inputs, β is a $K \times 1$ vector of constants and $sigm(t)$ is the logistic function defined by

$$sigm(t) = \frac{1}{1 - e^{-t}}$$

1.13.5 Generative Model

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

In probability and statistics, a generative model is a model for randomly generating observable data values, typically given some hidden parameters. It specifies

a joint probability distribution over observation and label sequences. Generative models are used in machine learning for either modeling data directly (i.e., modeling observations drawn from a probability density function, or as an intermediate step to forming a conditional probability density function. A conditional distribution can be formed from a generative model through Bayes' rule.

Generative models contrast with discriminative models, in that a generative model is a full probabilistic model of all variables, whereas a discriminative model provides a model only for the target variable(s) conditional on the observed variables. Thus a generative model can be used, for example, to simulate (i.e. generate) values of any variable in the model, whereas a discriminative model allows only sampling of the target variables conditional on the observed quantities. Despite the fact that discriminative models do not need to model the distribution of the observed variables, they cannot generally express more complex relationships between the observed and target variables. They don't necessarily perform better than generative models at classification and regression tasks. In modern applications the two classes are seen as complementary or as different views of the same procedure. Examples of generative models include:

- Gaussian mixture model
- Hidden Markov model
- Probabilistic context free grammar
- Naive Bayes
- Averaged one dependence estimators
- Latent Dirichlet allocation
- Restricted Boltzmann machine

- Generative adversarial networks

1.13.6 Nearest Neighbor

The k-nearest-neighbors algorithm is a classification algorithm, and it is supervised: it takes a bunch of labelled points and uses them to learn how to label other points. To label a new point, it looks at the labelled points closest to that new point (those are its nearest neighbors), and has those neighbors vote, so whichever label the most of the neighbors have is the label for the new point (the "k" is the number of neighbors it checks). K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). In pattern recognition, the k-nearest neighbors' algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor. The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

Algorithm

Let m be the number of training data samples. Let p be an unknown point.

1 Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).

2 for i=0 to m:

3 Calculate Euclidean distance d(arr[i], p).

4 Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.

5 Return the majority label among S.

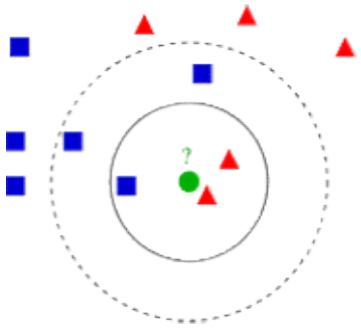

Fig. 1.10 k-NN classification

Example of k-NN classification Fig. 1.10. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 it is classified to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 it is classified to first class (3 squares vs. 2 triangles inside the outer circle). The training examples are vectors in a multidimensional feature space. The space is partitioned into regions by locations and labels of the training samples. A point in the space is assigned to the class c if it is the most frequent class label among the k nearest training samples. Usually Euclidean distance is used. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the actual classification phase, the test sample (whose class is not known) is represented as a vector in the feature space. Distances from the new vector to all stored vectors are computed and k closest samples are selected. There are a number of ways to classify the new vector to a particular class, one of the most used technique is to predict the new vector to the most common class amongst the K nearest neighbors. A major drawback to use this technique to classify a new vector to a class is that the classes with the more frequent examples tend to dominate the prediction of the new vector, as they tend to come up in the K nearest neighbors when the neighbors are computed due to their large number. One of the ways to overcome

this problem is to take into account the distance of each K nearest neighbors with the new vector that is to be classified and predict the class of the new vector based on these distances.

Parameter selection

The best choice of k depends upon the data; generally, larger values of k reduces effect of the noise on the classification, but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques (see hyperparameter optimization). The special case where the class is predicted to be the class of the closest training sample (i.e. when k = 1) is called the nearest neighbor algorithm. The accuracy of the k-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve classification. A particularly popular approach is the use of evolutionary algorithms to optimize feature scaling. Another popular approach is to scale features by the mutual information of the training data with the training classes. In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids tied votes. One popular way of choosing the empirically optimal k in this setting is via bootstrap method.

The 1-nearest neighbor classifier

The most intuitive nearest neighbour type classifier is the one nearest neighbour classifier that assigns a point x to the class of its closest neighbour in the feature space. As the size of training data set approaches infinity, the one nearest neighbour classifier guarantees an error rate of no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data).

A few Applications and Examples of KNN

- Credit ratings-collecting financial characteristics vs. comparing people with

similar financial features to a database. By the very nature of a credit rating, people who have similar financial details would be given similar credit ratings. Therefore, they would like to be able to use this existing database to predict a new customer's credit rating, without having to perform all the calculations.

- Should the bank give a loan to an individual? Would an individual default on his or her loan? Is that person closer in characteristics to people who defaulted or did not default on their loans?
- In political science-classing a potential voter to a "will vote" or "will not vote", or to "vote Democrat" or "vote Republican".
- More advance examples could include handwriting detection (like OCR), image recognition and even video recognition.

Some pros and cons of KNN :

Pros:

- No assumptions about data - useful, for example, for non-linear data
- Simple algorithm - to explain and understand/interpret
- High accuracy (relatively) - it is pretty high but not competitive to better supervised learning models
- Versatile - useful for classification or regression

Cons:

- Computationally expensive - because the algorithm stores all of the training data
- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data

UNIT II

2.1 Logistic regression

The logistic model (or logit model) is a widely used statistical model that, in its basic form, uses a logistic function to model a binary dependent variable; many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model; it is a form of binomial regression. Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail, win/lose, alive/dead or healthy/sick; these are represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a logit, from logistic unit, hence the

alternative names. Analogous models with a different sigmoid function instead of the logistic function can also be used, such as the probit model; the defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at a constant rate, with each dependent variable having its own parameter; for a binary independent variable this generalizes the odds ratio. We can also say that the target variable is categorical. Based on the number of categories, Logistic regression can be classified as:

1 **binomial:**

Target variable can have only 2 possible types: "0" or "1" which may represent "win" vs "loss", "pass" vs "fails", "dead" vs "alive", etc.

2 **Multinomial:**

Target variable can have 3 or more possible types which are not ordeR (i.e. types have no quantitative significance) like "disease A" vs "disease B" vs "disease C".

3 **Ordinal:**

It deals with target variables with ordeR categories. For example, a test score can be categorized as: "very poor", "poor", "good", "very good". Here, each category can be given a score like 0, 1, 2, 3.

2.1.1 Logistic Function

Logistic regression is named for the function used at the core of the method, the logistic function. The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology,

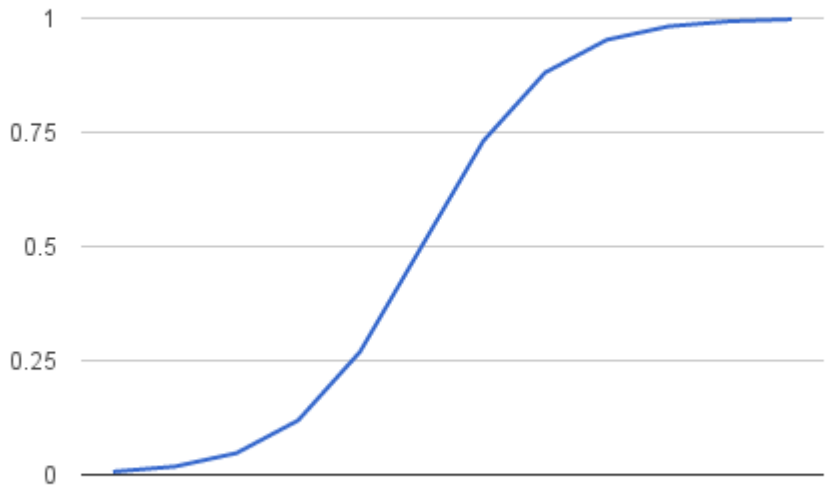

Fig. 2.1 Logistic Function

rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits given below.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Where e is the base of the natural logarithms and value is the actual numerical value that you want to transform. Below Fig. 2.1 is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.

2.1.2 Representation of Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referR to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is

a binary value (0 or 1) rather than a numeric value. Below is an example logistic regression equation:

$$y = \frac{e^{(bo+b1\times x)}}{1 + e^{(bo+b1\times x)}} \tag{2.1}$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data. The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b's).

2.1.3 Logistic Regression Predicts Probabilities

Logistic regression models the probability of the default class (e.g. the first class). For example, if we are modeling people's sex as male or female from their height, then the first class could be male and the logistic regression model could be written as the probability of male given a person's height, or more formally:

$$P(sex = male|height)$$

Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

$$P(X) = P(Y = 1|X)$$

We're predicting probabilities. I thought logistic regression was a classification algorithm. Note that the probability prediction must be transformed into a binary value (0 or 1) in order to actually make a probability prediction. More on this later

when we talk about making predictions. Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that we can no longer understand the predictions as a linear combination of the inputs as we can with linear regression, for example, continuing on from above, the model can be stated as:

$$p(X) = \frac{e^{(b0+b1\times X)}}{1 + e^{(b0+b1\times X)}}$$

We don't want to dive into the math too much, but we can turn around the above equation as follows (remember we can remove the e from one side by adding a natural logarithm (ln) to the other):

$$ln(\frac{p(X)}{1–p(X)}) = b0 + b1 \times X$$

This is useful because we can see that the calculation of the output on the right is linear again (just like linear regression), and the input on the left is a log of the probability of the default class. This ratio on the left is called the odds of the default class (it's historical that we use odds, for example, odds are used in horse racing rather than probabilities). Odds are calculated as a ratio of the probability of the event divided by the probability of not the event, e.g. 0.8/(1-0.8) which has the odds of 4. So we could instead write:

$$ln(odds) = b0 + b1 \times X$$

Because the odds are log transformed, we call this left hand side the log-odds or the probit. It is possible to use other types of functions for the transform (which is out of scope), but as such it is common to refer to the transform that relates the linear regression equation to the probabilities as the link function, e.g. the probit

link function. We can move the exponent back to the right and write it as:

$$odds = e^{(b0+b1\times X)}$$

All of this helps us understand that indeed the model is still a linear combination of the inputs, but that this linear combination relates to the log-odds of the default class.

2.1.4 Learning the Logistic Regression Model

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation. Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data). The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

We are not going to go into the math of maximum likelihood. It is enough to say that a minimization algorithm is used to optimize the best values for the coefficients for your training data. This is often implemented in practice using efficient numerical optimization algorithm (like the Quasi-newton method). When you

are learning logistic, you can implement it yourself from scratch using the much simpler gradient descent algorithm.

2.1.5 Making Predictions with Logistic Regression

Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result. Let's make this concrete with a specific example. Let's say we have a model that can predict whether a person is male or female based on their height (completely fictitious). Given a height of 150cm is the person male or female.

We have learned the coefficients of b0 = -100 and b1 = 0.6. Using the equation 2.1 above we can calculate the probability of male given a height of 150cm or more formally $P(male|height = 150)$

$$y = \frac{e^{(-100+0.6\times150)}}{1 + e^{(-100+0.6\times150)}} = 0.0000453978687$$

$y = 0.0000453978687$ means probability that the person is a male is nearly zero. In practice we can use the probabilities directly. Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value, for example:

$$\begin{cases} 0 & \text{if } p(male) < 0.5 \\ 1 & \text{if } p(male) >= 0.5 \end{cases}$$

Now that we know how to make predictions using logistic regression, let's look at how we can prepare our data to get the most from the technique.

2.1.6 Prepare Data for Logistic Regression

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression. Much study has gone into defining these assumptions and precise probabilistic and statistical language is used. My advice is to use these as guidelines or rules of thumb and experiment with different data preparation schemes. Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

- **Binary Output Variable:** This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
- **Remove Noise:** Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
- **Gaussian Distribution:** Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.

- **Remove Correlated Inputs:** Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge:** It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

2.1.7 Pros and Cons of Logistic Regression

Pros:

Logistic regression is designed for this purpose! The dependent variable must be categorical, and the explanatory variables can take any form; both of which are satisfied by your problem.

Linear combination of parameters $\beta\beta$ and the input vector will be incRibly easy to compute. Given that your explanatory variables are also binary, you should be able to partition your input space by outcome quite well.

Cons:

You say several binary predictors. Going off the dictionary definition of, "More than two, but not many." - logistic regression might be overkill.

2.2 Perceptron

Perceptron is a single layer neural network. It is a linear classifier. It is used in supervised learning. It helps to classify the given input data. A perceptron is a simple model of a biological neuron in an artificial neural network. It is also the

name of an early algorithm for supervised learning of binary classifiers. Machine learning algorithms find and classify patterns by many different means. perception is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.
The perceptron consists of 4 parts as shown in Fig. 2.2 :

1. Input values or One input layer
2. Weights and Bias
3. Net sum
4. Activation Function

2.2.1 How does a Perceptron work?

Perception is not the Sigmoid neuron we use in ANNs or any deep learning networks today. The perceptron model is a more general computational model than McCulloch-Pitts neuron. It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0. Rewriting the threshold as shown above and making it a constant input with a variable weight, we would end up with something like the following:

- All the inputs x are multiplied with their weights w.
- Then Add all the multiplied values and call them Weighted Sum $\sum$.

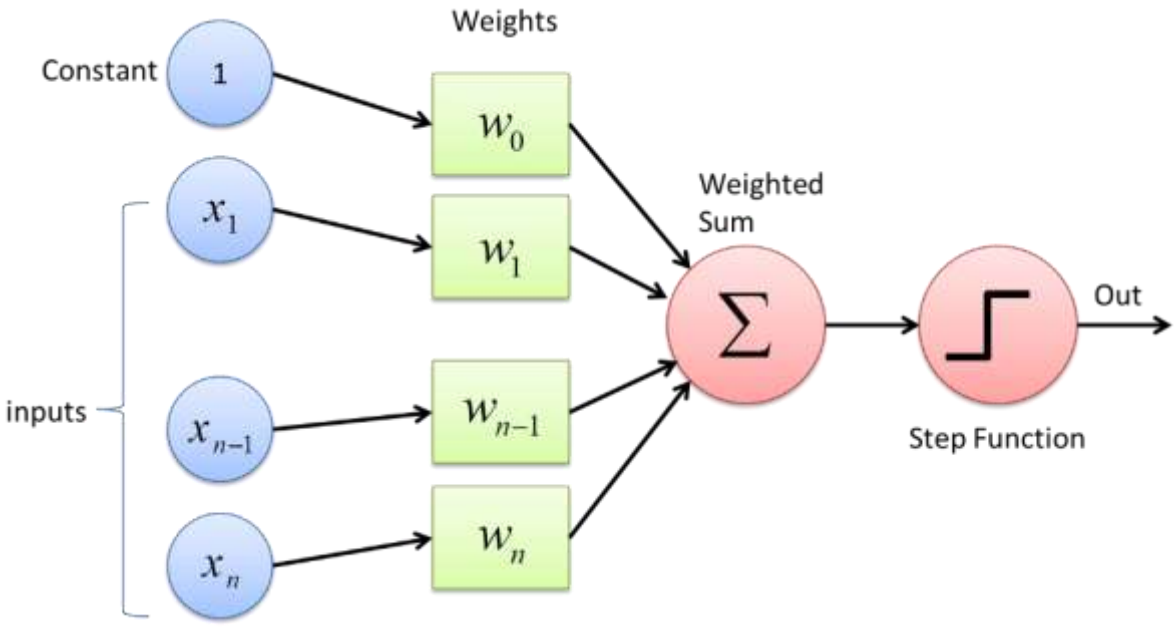

Fig. 2.2 Parts of Perceptron

- Apply that weighted sum to the correct Activation Function.

A single perceptron can only be used to implement linearly separable functions. It takes both real and boolean inputs and associates a set of weights to them, along with a bias (threshold). We learn the weights, we get the function Fig. 2.3.

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i \times x_i - \theta \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i \times x_i - \theta < 0 \end{cases}$$

2.2.2 Perceptron Learning Algorithm

Our goal is to find the w vector that can perfectly classify positive inputs and negative inputs in our data. straight to the. Here is the algorithm:

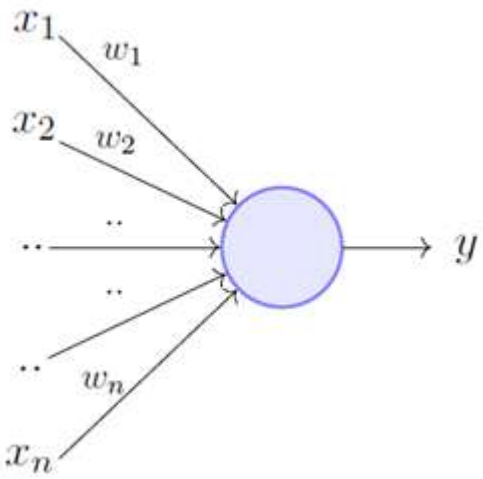

Fig. 2.3 Working of Perceptron

Step 1. P ← input with label 1

Step 2. N ← input with label 0

Step 3. Initialize w randomly

Step 4. While !convergence do

Pick random $x \in P \cup N$

if $x \in P$ and $w.x < 0$ then

$w = w + x$

end

if $x \in N$ and $w.x \geq 0$ then

$w = w - x$

end

end

The algorithm converges when all the inputs are classified correctly. We initialize w with some random vector. We then iterate over all the examples in the data, $P \cup N$ both positive and negative examples. Now if an input x belongs to P, ideally what should the dot product w.x be? It should be greater than or equal to 0 because that's the only thing what our perceptron wants at the end of the day so let's give it that. And if x belongs to N, the dot product must be less than 0.

Case 1: When x belongs to P and its dot product $w.x < 0$

Case 2: When x belongs to N and its dot product $w.x \geq 0$

Only for these cases, we are updating our randomly initialized w. Otherwise, we don't touch w at all because Case 1 and Case 2 are violating the very rule of a perceptron. So we are adding x to w in Case 1 and subtracting x from w in Case 2.

2.3 Exponential family

Exponential families are a broad class of probability distributions which includes many basic distributions such as Bernoulli's and Gaussians, as well as Markov random fields. In all of these distributions can be represented in terms of log-linear functions of sufficient statistics. A distribution over a random variable X is in the exponential family if we can write it as :

$$P(x|\eta) = h(x)exp(\eta^T T(x) - A(\eta))$$

Here, η is the vector of natural parameters, T is the vector of sufficient statistics, and A is the log partition function.
Exponential families are useful in many fields such as:

- They unify many of the most important, widely-used statistical models such as the Normal, Binomial, Poisson, and Gamma into one framework.
- No matter how massive the data set is, there is a sufficient statistic of a fixed dimensionality. Under some regularity conditions (such as that the support does not depend on the parameter), this is only true for exponential families.
- You can easily see what the minimal sufficient statistic for the model is, and better yet it will be a complete sufficient statistic (under some regularity conditions). Usually completeness of a statistic is hard to prove, but in an exponential family you get it almost for free. This paves the way to be able to apply Basu's theorem, for example. Moreover, the complete sufficient statistic itself comes from an exponential family.
- Exponential families maximize entropy, among distributions satisfying certain natural constraints.
- Conjugate distributions are easy to write down, and the conjugate distributions come from an exponential family.
- Maximum likelihood estimation (MLE) behaves nicely in this setting, and has a very simple intuitive interpretation: set the observed value of the natural sufficient statistic equal to its expected value. The log-likelihood function will be concave, so we don't get nasty multimodal situations such as can occur in a Cauchy location problem.

2.3.1 Examples of exponential family

Here are some examples of distributions that are in the exponential family:-

2.3.1.1 Normal/Gaussian distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where

μ is the mean or expectation of the distribution (and also its median and mode),

σ is the standard deviation, and

σ^2 is the variance.

2.3.1.2 Poisson distribution

An event can occur 0, 1, 2, ... times in an interval. The average number of events in an interval is designated λ. λ is the event rate, also called the rate parameter. The probability of observing k events in an interval is given by the equation:

$$P(k \text{ events in interval}) = e^{-\lambda} \frac{\lambda^k}{k!}$$

where

λ is the average number of events per interval

e is the number 2.71828 ... (Euler's number) the base of the natural logarithms

k takes values 0, 1, 2, ...

$k! = k \times (k-1) \times (k-2) \times ... \times 2 \times 1$ is a factorial of k.

This equation is the probability mass function (PMF) for a Poisson distribution.

2.3.1.3 Exponential distribution

The probability density function (pdf) of an exponential distribution is

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

Where
$\lambda > 0$ is the parameter of the distribution, often called the rate parameter. The distribution is supported on the interval $[0, \infty)$. If a random variable X has this distribution, we write $X \sim Exp(\lambda)$.

2.3.1.4 Bernoulli distribution

2.3.1.5 Binomial distribution

2.3.1.6 Multinomial distribution

2.3.1.7 Gamma distribution

2.3.2 Properties

The exponential family has the following property (called the moment generating property):

1 The d'th derivative of the log partition equals the d'th centeR moment of the suffcient statistic (if you have a vector of sufficient statistics, then $\partial^d A/\partial\eta_i^d = E[T(x)_i^d]$). E.g., the First derivative of the log partition function is the mean of T(X); the 2nd is its variance.

2 This implies that the log partition function is convex, because its second derivative must be positive, since variance is always non-negative.

3 This further implies that: we can write the first derivative of the log partition function as a function of the natural parameter (aka the canonical parameter), set it equal to the mean, and then invert to solve for the natural parameter in terms of the mean (aka the moment parameter). In symbols: $\eta = \Psi(\mu)$.

4 Doing Maximum likelihood estimation (MLE) on the exponential family is the same as doing moment matching. This follows by:

   a Writing down the log likelihood of a generic exponential family member: $const + \eta^T(\sum_{i=1}^n T(x_i)) - nA(\eta)$.

   b Taking the gradient w.r.t. $\eta : \sum_{i=1}^n T(x_i) - n\nabla_\eta A(\eta)$.

   c Setting equal to zero and solving for $\nabla_\eta A$:
   $\nabla_\eta A = \frac{1}{n}\sum_{i=1}^n T(x_i) \Rightarrow \mu = \frac{1}{n}\sum_{i=1}^n T(x_i) \Rightarrow$ *estimated moment* = *sample moment*.

2.4 Generative learning algorithms

Consider a classification problem in which we want to learn to distinguish between elephants ($y = 1$) and dogs ($y = 0$), based on some features of an animal. Given a training set, an algorithm like logistic regression or the perceptron algorithm (basically) tries to find a straight line - that is, a decision boundary that separates the elephants and dogs. Then, to classify a new animal as either an elephant or a dog, it checks on which side of the decision boundary it falls, and makes its prediction accordingly.

Here's a different approach. First, looking at elephants, we can build a model of what elephants look like. Then, looking at dogs, we can build a separate model of what dogs look like. Finally, to classify a new animal, we can match the new animal against the elephant model, and match it against the dog model, to see whether the new animal looks more like the elephants or more like the dogs we had seen in the training set.

Algorithms that try to learn $p(y|x)$ directly (such as logistic regression), or algorithms that try to learn mappings directly from the space of inputs X to the labels $\{0, 1\}$, (such as the perceptron algorithm) are called discriminative learning algorithms. Here, we'll talk about algorithms that instead try to model $p(x|y)$ (and $p(y)$). These algorithms are called generative learning algorithms. Generative approaches try to build a model of the positives and a model of the negatives. You can think of a model as a "blueprint" for a class. A decision boundary is formed where one model becomes more likely. As these create models of each class they can be used for generation .To create these models, a generative learning algorithm learns the joint probability distribution $P(x, y)$.
The joint probability can be written as:

$$P(x, y) = P(x|y).P(y) \tag{2.2}$$

Also, using Bayes' Rule we can write:

$$P(y|x) = \frac{P(x|y).P(y)}{P(x)} \tag{2.3}$$

Since, to predict a class label y, we are only interested in the argmax , the denominator can be removed from Eq. 2.3. Hence to predict the label y from the training

example x, generative models evaluate:

$$f(x) = \arg\max_y P(y|x) = \arg\max_y P(x|y).P(y) \tag{2.4}$$

The most important part in the above is $P(x|y)$. This is what allows the model to be generative. $P(x|y)$ means – what x (features) are there given class y. Hence, with the joint probability distribution function Eq. 2.2, given a y, you can calculate ("generate") its corresponding x. For this reason they are called generative models. Generative learning algorithms make strong assumptions on the data. To explain this let's look at a generative learning algorithm called Gaussian Discriminant Analysis (GDA)

2.5 Gaussian discriminant analysis

The first generative learning algorithm that we'll look at is Gaussian discriminant analysis (GDA). In this model, we'll assume that $p(x|y)$ is distributed according to a multivariate normal distribution. When we have a classification problem in which the input features x are continuous-valued random variables, we can then use the Gaussian Discriminant Analysis (GDA) model, which models $p(x|y)$ using a multivariate normal distribution. The model is:

$$\begin{aligned} y &\sim Bernoulli(\phi) \\ x|y=0 &\sim N\left(\mu_0, \sum\right) \\ x|y=1 &\sim N\left(\mu_1, \sum\right) \end{aligned}$$

Writing out the distributions, this is:

$$
\begin{aligned}
p(y) &= \phi^{y}(1-\phi)^{1-y} \\
p(x|y=0) &= \frac{1}{(2\pi)^{n/2}\,|\sum|^{1/2}} exp\left(-\frac{1}{2}(x-\mu_0)^{T}\sum^{-1}(x-\mu_0)\right) \\
p(x|y=1) &= \frac{1}{(2\pi)^{n/2}\,|\sum|^{1/2}} exp\left(-\frac{1}{2}(x-\mu_1)^{T}\sum^{-1}(x-\mu_1)\right)
\end{aligned}
$$

Here, the parameters of our model are $\phi, \sum, \mu_0$ and μ_1. (Note that while there are two different mean vectors μ_0 and μ_1, this model is usually applied using only one covariance matrix $\sum$). The log-likelihood of the data is given by

$$
\begin{aligned}
\ell\left(\phi, \mu_0, \mu_1, \sum\right) &= log\Pi_{i=1}^{m}\left(x^i, y^i; \phi, \mu_0, \mu_1, \sum\right) \\
&= log\Pi_{i=1}^{m}\left(x^i|y^i; \phi, \mu_0, \mu_1, \sum\right) p\left(x^i; \phi\right)
\end{aligned}
$$

By maximizing ℓ with respect to the parameters, we find the maximum likelihood estimate of the parameters to be:

$$
\begin{aligned}
\phi &= \frac{1}{m}\sum_{i=1}^{m} 1\{y^i = 1\} \\
\mu_0 &= \frac{\sum_{i=1}^{m} 1\{y^i = 0\}x^i}{\sum_{i=1}^{m} 1\{y^i = 0\}} \\
\mu_1 &= \frac{\sum_{i=1}^{m} 1\{y^i = 1\}x^i}{\sum_{i=1}^{m} 1\{y^i = 1\}} \\
\sum &= \frac{1}{m}\sum_{i=1}^{m}\left(x^i - \mu_{y^i}\right)\left(x^i - \mu_{y^i}\right)^{T}
\end{aligned}
$$

Pictorially, what the algorithm is doing can be seen in Fig. 2.4 as follows: Shown in the Fig. 2.4 are the training set, as well as the contours of the two Gaussian distributions that have been fit to the data in each of the two classes. Note that

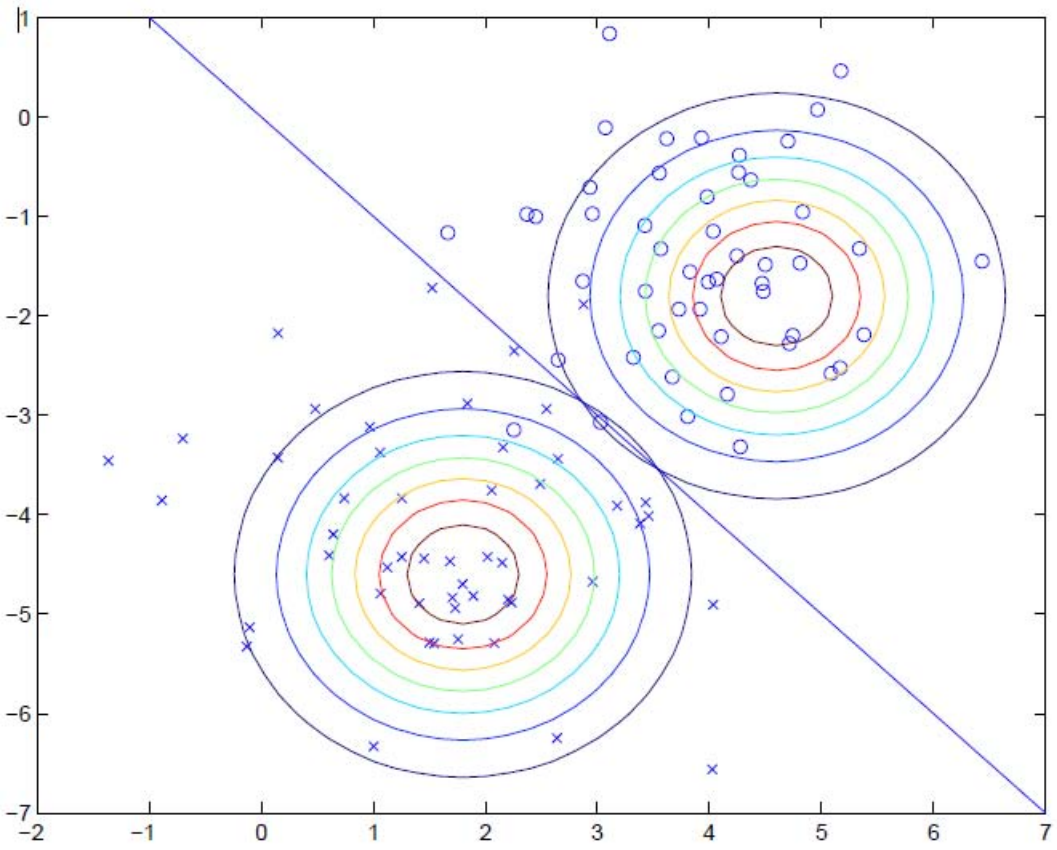

Fig. 2.4 GDA Model

the two Gaussians have contours that are the same shape and orientation, since they share a covariance matrix $\sum$, but they have different means μ_0 and μ_1. Also shown in the figure is the straight line giving the decision boundary at which $p(y = 1|x) = 0.5$. On one side of the boundary, we'll predict $y = 1$ to be the most likely outcome, and on the other side, we'll predict $y = 0$.

2.6 Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm based on the Bayes Theorem, used in a wide variety of classification tasks. They are probabilistic, which means that they calculate the probability of each tag for a given text, and then output the tag with the highest one. Typical applications include filtering spam, classifying documents, sentiment prediction etc. It is based on the works of Rev. Thomas Bayes (1702–61).

Naive Bayes algorithm is the algorithm that learns the probability of an object with certain features belonging to a particular group/class. In short, it is a probabilistic classifier. The Naive Bayes algorithm is called "naive" because it makes the assumption that the occurrence of a certain feature is independent of the occurrence of other features. For instance, if you are trying to identify a fruit based on its color, shape, and taste, then an orange coloR, spherical, and tangy fruit would most likely be an orange. Even if these features depend on each other or on the presence of the other features, all of these properties individually contribute to the probability that this fruit is an orange and that is why it is known as "naive."

2.6.1 Bayes' theorem

The basis of Naive Bayes algorithm is Bayes' theorem or alternatively known as Bayes' rule or Bayes' law. It gives us a method to calculate the conditional probability, i.e., the probability of an event based on previous knowledge available on

the events. More formally, Bayes' Theorem is stated as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

The components of the above statement are:

- $P(A|B)$: Probability (conditional probability) of occurrence of event A given the event B is true.
- $P(A)$ and $P(B)$: Probabilities of the occurrence of event A and B respectively.
- $P(B|A)$: Probability of the occurrence of event B given the event A is true.

The terminology in the Bayesian method of probability (more commonly used) is as follows:

- A is called the proposition and B is called the evidence.
- $P(A)$ is called the prior probability of proposition and $P(B)$ is called the prior probability of evidence.
- $P(A|B)$ is called the posterior.
- $P(B|A)$ is the likelihood.

This sums the Bayes' theorem as

$$Posterior = \frac{(Likelihood)(Proposition\ prior\ probability)}{(Evidence\ prior\ probability)}$$

2.6.2 Example Bayes' theorem

Suppose you have to draw a single card from a standard deck of 52 cards. Now the probability that the card is a $Queen$ is $P(Queen) = \frac{4}{52} = \frac{1}{13}$. If you are given evidence that the card that you have picked is a face card, the posterior probability $P(Queen|Face)$ can be calculated using Bayes' Theorem as follows:

$$P(Queen|Face) = \frac{P(Face|Queen)P(Queen)}{P(Face)}$$

Now $P(Face|Queen) = 1$ because given the card is $Queen$, it is definitely a face card. We have already calculated $P(Queen)$. The only value left to calculate is $P(Face)$, which is equal to $\frac{3}{13}$ as there are three face cards for every suit in a deck. Therefore,
Now

$$P(Face|Queen) = 1$$
$$P(Queen) = \frac{1}{13}$$
$$P(Face) = \frac{3}{13}$$

So

$$P(Queen|Face) = \frac{1 \times \frac{1}{13}}{\frac{3}{13}} = \frac{1}{3}$$

2.6.3 Bayes' Theorem for Naive Bayes Algorithm

In a machine learning classification problem, there are multiple features and classes, say, $C_1, C_2, \ldots, C_k$. The main aim in the Naive Bayes algorithm is to calculate the conditional probability of an object with a feature vector $x_1, x_2, \ldots, x_n$ belongs to a particular class C_i,

$$P(C_i|x_1, x_2, \ldots, x_n) = \frac{P(x_1, x_2, \ldots, x_n|C_i)P(C_i)}{P(x_1, x_2, \ldots, x_n)} \text{ for } 1 \leq i \leq k$$

Now, the numerator of the fraction on right-hand side of the equation above is $P(x_1, x_2, \ldots, x_n|C_i)P(C_i) = P(x_1, x_2, \ldots, x_n, C_i)$

$$\begin{aligned} P(x_1, x_2, \ldots, x_n, C_i) &= P(x_1|x_2, \ldots, x_n, C_i).P(x_2, \ldots, x_n, C_i) \\ &= P(x_1|x_2, \ldots, x_n, C_i).P(x_2|x_3 \ldots, x_n, C_i).P(x_3, \ldots, x_n, C_i) \\ &= \ldots \\ &= P(x_1|x_2, \ldots, x_n, C_i).P(x_2|x_3, \ldots, x_n, C_i) \ldots P(x_{n-1}|x_n, C_i).P(x_n|C_i).P(C_i) \end{aligned}$$

The conditional probability term $P(x_j|x_{j+1}, \ldots, x_n, C_i)$ becomes $P(x_j|C_i)$ because of the assumption that features are independent. From the calculation above and the independence assumption, the Bayes theorem boils down to the following easy expression:

$$P(C_i|x_1, x_2, \ldots, x_n) = \frac{(\Pi_{j=1}^{n} P(x_j|C_i)).P(C_i)}{P(x_1, x_2, \ldots, x_n)} \text{ for } 1 \leq i \leq k$$

The expression $P(x_1, x_2, \ldots, x_n)$ is constant for all the classes, we can simply say that

$$P(C_i|x_1, x_2, \ldots, x_n) \propto (\Pi_{j=1}^{n} P(x_j|C_i)).P(C_i) \text{ for } 1 \leq i \leq k$$

2.6.4 Example of the algorithm

Let us take a simple example to understand the functionality of the algorithm. Suppose, we have a training data set of 1200 fruits. The features in the data set are these: is the fruit Red(R) or not, is the fruit long(L) or not, and is the fruit Sweet(S) or not. There are three different classes: Mango(M), Banana(B), and Others.

- **Step 1** Create a frequency table 2.1 for all the features against the different classes. What can we conclude from the above table?

Table 2.1 Frequency table for all the features

Name	Red(R)	Sweet(S)	Long(L)	Total
Mango(M)	350	450	0	650
Banana(B)	400	300	350	400
Others	50	100	50	150
Total	800	850	400	1200

  - Out of 1200 fruits, 650 are mangoes, 400 are bananas, and 150 are others.
  - 350 of the total 650 mangoes are Red and the rest are not and so on.
  - 800 fruits are Red, 850 are sweet and 400 are long from a total of 1200 fruits.

  Let's say you are given with a fruit which is Red, sweet, and long and you have to check the class to which it belongs.

- **Step 2** Draw the likelihood table 2.2 for the features against the classes.

Table 2.2 Likelihood table for the feature

Name	Red(R)	Sweet(S)	Long(L)	Total
Mango(M)	$\frac{350}{800} = P(M\|R)$	$\frac{450}{850}$	$\frac{0}{400}$	$\frac{650}{1200} = P(M)$
Banana(B)	$\frac{400}{800}$	$\frac{300}{850}$	$\frac{350}{400}$	$\frac{400}{1200}$
Others	$\frac{50}{800}$	$\frac{100}{850}$	$\frac{50}{400}$	$\frac{150}{1200}$
Total	$800 = P(R)$	850	400	1200

- **Step 3** Calculate the conditional probabilities for all the classes, i.e., the following in our example:

$$
\begin{aligned}
P(M|R,S,L) &= \frac{P(R|M).P(S|M).P(L|M).P(M)}{P(R,S,L)} \\
&= 0 \\
P(B|R,S,L) &= \frac{P(R|B).P(S|B).P(L|B).P(B)}{P(R,S,L)} \\
&= \frac{400 \times 300 \times 350 \times 400}{400 \times 400 \times 400 \times 1200 \times P(Evidence)} \\
&= \frac{0.21875}{P(Evidence)} \\
P(Others|R,S,L) &= \frac{P(R|Others).P(S|Others).P(L|Others).P(Others)}{P(R,S,L)} \\
&= \frac{50 \times 100 \times 50 \times 150}{150 \times 150 \times 150 \times 1200 \times P(Evidence)} \\
&= \frac{0.00926}{P(Evidence)}
\end{aligned}
$$

- **Step 4** Calculate $max_i P(C_i|x_1, x_2, \ldots, x_n)$. In our example, the maximum probability is for the class banana, therefore, the fruit which is long, sweet

and R is a banana by Naive Bayes Algorithm. In a nutshell, we say that a new element will belong to the class which will have the maximum conditional probability described above.

2.6.5 Variations of the algorithm

There are multiple variations of the Naive Bayes algorithm depending on the distribution of $P(x_j|C_i)$. Three of the commonly used variations are:

- Gaussian: The Gaussian Naive Bayes algorithm assumes distribution of features to be Gaussian or normal, i.e.

$$P(x_j|C_i) = \frac{1}{\sqrt{2\pi\sigma^2 C_i}} exp\left(-\frac{(x_j - \mu C_j)^2}{2\sigma^2 C_i}\right)$$

- Multinomial: The Multinomial Naive Bayes algorithm is used when the data is distributed multinomially, i.e. multiple occurrences matter a lot.

- Bernoulli: The Bernoulli algorithm is used when the features in the data set are binary-valued. It is helpful in spam filtration and adult content detection techniques.

2.6.6 Pros and Cons of the algorithm

Every coin has two sides. So does the Naive Bayes algorithm. It has advantages as well as disadvantages, and they are listed below:

Pros

- It is a relatively easy algorithm to build and understand.
- It is faster to predict classes using this algorithm than many other classification algorithms.
- It can be easily trained using a small data set.

Cons

- If a given class and a feature have 0 frequency, then the conditional probability estimate for that category will come out as 0. This problem is known as the "Zero Conditional Probability Problem." This is a problem because it wipes out all the information in other probabilities too. There are several sample correction techniques to fix this problem such as "Laplacian Correction."
- Another disadvantage is the very strong assumption of independence class features that it makes. It is near to impossible to find such data sets in real life.

2.7 Support vector machine

Support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling

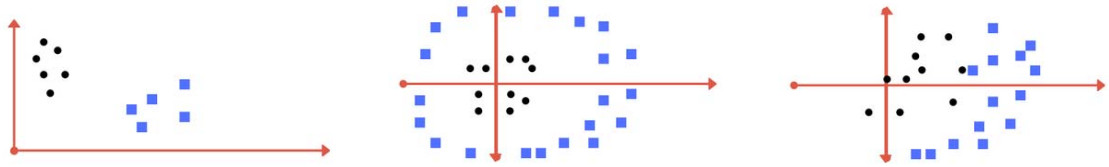

Fig. 2.5 Three simple graphs

exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data is unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support-vector clustering algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabelled data, and is one of the most widely used clustering algorithms in industrial applications.

Given a training sample, the support vector machine constructs a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized.

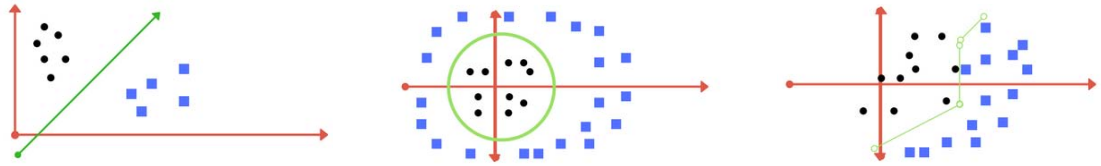

Fig. 2.6 Classification of Fig. 2.6

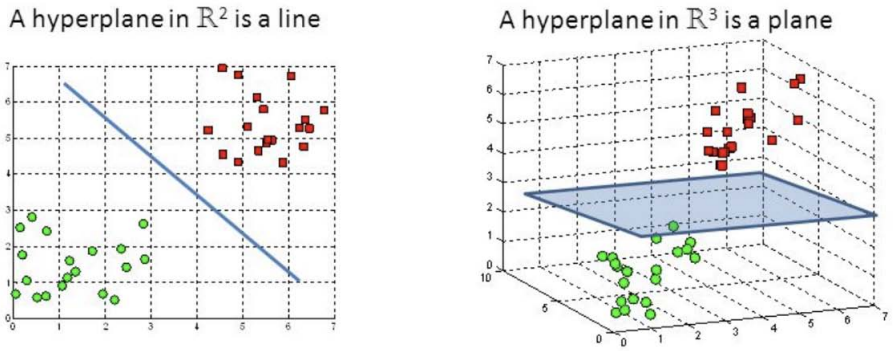

Fig. 2.7 Hyperplanes in 2D and 3D feature space

2.7.1 Optimal hyper planes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line Fig. 2.7. If the number of input features is 3, then the hyperplane becomes a three-dimensional plane Fig. 2.7. It becomes difficult to imagine when the number of features exceeds 3. Support vectors are data points that are closer

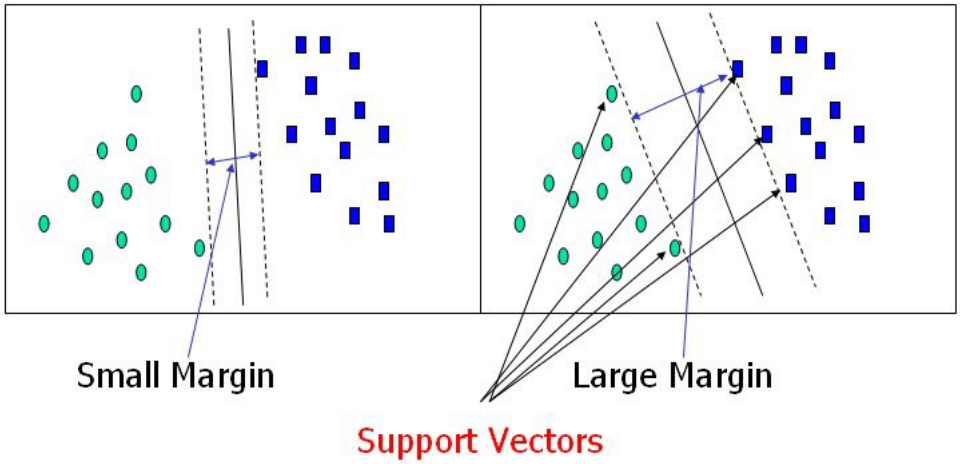

Fig. 2.8 Hyperplanes in 2D and 3D feature space

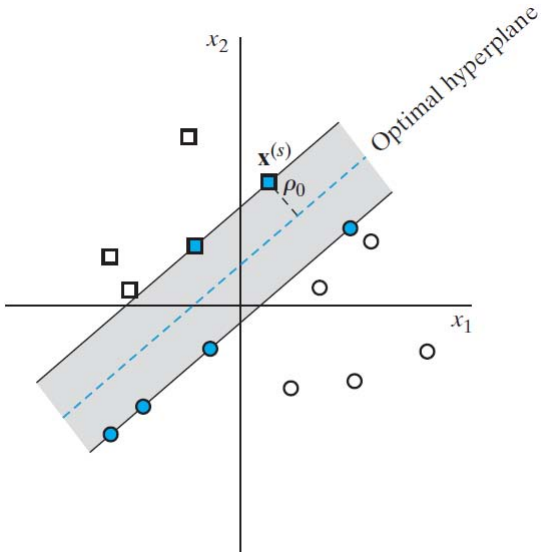

Fig. 2.9 Example of Optimal hyper planes

to the hyperplane and influence the position and orientation of the hyperplane Fig. 2.8. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM Fig. 2.9

2.7.2 Kernels

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role. The SVM algorithm is implemented in practice using a kernel. The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra, which is out of the scope of this introduction to SVM. A powerful insight is that the linear SVM can be rephrased using the inner product of any two given observations, rather than the observations themselves. The inner product between two vectors is the sum of the multiplication of each pair of input values. For example, the inner product of the vectors $[2, 3]$ and $[5, 6]$ is $2*5+3*6$ or 28. The equation for making a prediction for a new input using the dot product between the input x and each support vector x_i is calculated as follows:

$$f(x) = B_0 + \sum(a_i \times (x, x_i))$$

This is an equation that involves calculating the inner products of a new input vector x with all support vectors in training data. The coefficients B_0 and a_i (for each input) must be estimated from the training data by the learning algorithm.

2.7.2.1 Linear Kernel

The Linear kernel is the simplest kernel function. It is given by the inner product $< x, y >$ plus an optional constant c. Kernel algorithms using a linear kernel are often equivalent to their non-kernel counterparts, i.e. KPCA with linear kernel is

the same as standard PCA.

$$k(x, y) = x^T y + c$$

2.7.2.2 Polynomial Kernel

The Polynomial kernel is a non-stationary kernel. Polynomial kernels are well suited for problems where all the training data is normalized.

$$k(x, y) = (\alpha x^T y + c)^d$$

Adjustable parameters are the slope α, the constant term c and the polynomial degree d.

2.7.2.3 Radial Kernel

Finally, we can also have a more complex radial kernel. For example:

$$k(x, y) = exp\left(-\gamma \| x – y \|^2\right)$$

Where γ is a parameter that must be specified to the learning algorithm. A good default value for γ is 0.1, where γ is often $0 < \gamma < 1$. The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.

2.7.2.4 Gaussian Kernel

The Gaussian kernel is an example of radial basis function kernel.

$$k(x, y) = exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

Alternatively, it could also be implemented using

$$k(x, y) = exp\left(-\gamma\|x - y\|^2\right)$$

The adjustable parameter σ plays a major role in the performance of the kernel, and should be carefully tuned to the problem at hand. If overestimated, the exponential will behave almost linearly and the higher-dimensional projection will start to lose its non-linear power. In the other hand, if underestimated, the function will lack regularization and the decision boundary will be highly sensitive to noise in training data.

2.7.2.5 Exponential Kernel

The exponential kernel is closely related to the Gaussian kernel, with only the square of the norm left out. It is also a radial basis function kernel.

$$k(x, y) = exp\left(-\frac{\|x - y\|}{2\sigma^2}\right)$$

2.7.2.6 Laplacian Kernel

The Laplace Kernel is completely equivalent to the exponential kernel, except for being less sensitive for changes in the σ parameter. Being equivalent, it is also a

radial basis function kernel.

$$k(x, y) = exp\left(-\frac{\|x - y\|}{\sigma}\right)$$

It is important to note that the observations made about the σ parameter for the Gaussian kernel also apply to the Exponential and Laplacian kernels.

2.7.2.7 Sigmoid Kernel

The Hyperbolic Tangent Kernel is also known as the Sigmoid Kernel and as the Multilayer Perceptron (MLP) kernel. The Sigmoid Kernel comes from the Neural Networks field, where the bipolar sigmoid function is often used as an activation function for artificial neurons.

$$k(x, y) = \tanh(\alpha x^T y + c)$$

It is interesting to note that a SVM model using a sigmoid kernel function is equivalent to a two-layer, perceptron neural network. This kernel was quite popular for support vector machines due to its origin from neural network theory. Also, despite being only conditionally positive definite, it has been found to perform well in practice. There are two adjustable parameters in the sigmoid kernel, the slope α and the intercept constant c. A common value for alpha is $\frac{1}{N}$, where N is the data dimension

2.7.3 Model selection

Choosing the most appropriate kernel highly depends on the problem at hand – and fine tuning its parameters can easily become a tedious and cumbersome task

because it depends on what we are trying to model. A polynomial kernel, for example, allows us to model feature conjunctions up to the order of the polynomial. Radial basis functions allows to pick out circles (or hyperspheres) – in constrast with the Linear kernel, which allows only to pick out lines (or hyperplanes). The motivation behind the choice of a particular kernel can be very intuitive and straightforward depending on what kind of information we are expecting to extract about the data.

2.7.4 Feature selection

Feature selection is the method of reducing data dimension while doing predictive analysis. One major reason is that machine learning follows the rule of "garbage in-garbage out" and that is why one needs to be very concerned about the data that is being fed to the model. The feature selection techniques simplify the machine learning models in order to make it easier to interpret by the researchers. It mainly eliminates the effects of the curse of dimensionality. Besides, this technique reduces the problem of overfitting by enhancing the generalisation in the model. Thus it helps in better understanding of data, improves prediction performance, reducing the computational time as well as space which is required to run the algorithm The feature selection problem can be addressed in the following two ways:
(1) given a fixed $m << n$, find the m features that give the smallest expected generalization error γ; or
(2) given a maximum allowable generalization error γ, find the smallest m.
In both of these problems the expected generalization error γ is of course unknown, and thus must be estimated. Note that choices of m in problem (1) can usually can be reparameterized as choices of γ in problem (2). Different feature selection methods are :

- Filter Method
- Wrapper Method
- Embedded Method

2.7.5 Applications

SVMs can be used to solve various real-world problems:

- SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings. Some methods for shallow semantic parsing are based on support vector machines.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. This is also true for image segmentation systems, including those using a modified version SVM that uses the privileged approach as suggested by Vapnik.
- Hand-written characters can be recognized using SVM.
- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models. Support-vector machine weights have also been used to interpret SVM models in the past. Posthoc interpretation of support-vector machine models in order to

identify features used by the model to make predictions is a relatively new area of research with special significance in the biological sciences.

- Face detection – SVMc classify parts of the image as a face and non-face and create a square boundary around the face.
- Text and hypertext categorization – SVMs allow Text and hypertext categorization for both inductive and transductive models. They use training data to classify documents into different categories. It categorizes on the basis of the score generated and then compares with the threshold value.
- Classification of images – Use of SVMs provides better search accuracy for image classification. It provides better accuracy in comparison to the traditional query-based searching techniques.
- Bioinformatics – It includes protein classification and cancer classification. We use SVM for identifying the classification of genes, patients on the basis of genes and other biological problems. Protein fold and remote homology detection – Apply SVM algorithms for protein remote homology detection.
- Generalized predictive control(GPC) – Use SVM based GPC to control chaotic dynamics with useful parameters

2.7.6 Pros and Cons

Pros

- Based on nice theory
- Excellent generalization properties

- Objective function has no local minima
- Can be used to find non linear discriminant functions
- Complexity of the classifier is characterized by the number of support vectors rather than the dimensionality of the transformed space

Cons

- It's not clear how to select a kernel function in a principled manner
- Tends to be slower than other methods

2.8 Combining classifier

Experimental observations confirm that a given learning algorithm outperforms all others for a specific problem or for a exact subset of the input data, but it is abnormal to find a single expert achieving the best results on the overall problem domain. As a consequence the multiple learner systems try to exploit the locally different behaviour of the base classifiers to improve the accuracy and the reliability of the overall inductive learning system. There are also hopes that if some learner fails, the overall system can recover.

The aim of ensemble generation is a set of classifiers such that they are at the same time as different to each other as possible while remaining as accurate as possible when viewed individually. Independence (or diversity) is important because ensemble learning can only get better on individual classifiers when their errors are not correlated. Obviously these two aims (maximum accuracy of the individual

predictors and minimum correlation of incorrect predictions) conflict with each other, as two perfect classifiers would be rather alike, and two maximally different classifiers could not at the same time both be very accurate.

The final goal of classifier combination is to create a classifier which operates on the same type of input as the base classifiers and separates the same types of classes. Classifier combination techniques operate on the outputs of individual classifiers and usually fall into one of two categories. In the first approach the outputs are treated as inputs to a generic classifier, and the combination algorithm is created by training this, sometimes called 'secondary', classifier.

2.8.1 Types of Combined Classifiers

- **Type I (abstract level)**: This is the lowest level since a classifier provides the least amount of information on this level. Classifier output is merely a single class label or an unordered set of candidate classes.
- **Type II (rank level)**: Classifier output on the rank level is an ordered sequence of candidate classes, the so-called n-best list. The candidate class at the first position is the most likely class, while the class positioned at the end of the list is the most unlikely. Note that there are no confidence values attached to the class labels on rank level. Only their position in the n-best list indicates their relative likelihood.
- **Type III (measurement level)**: In addition to the ordered n-best lists of candidate classes on the rank level, classifier output on the measurement level has confidence values assigned to each entry of the n-best list. These

confidences, or scores, can be arbitrary real numbers, depending on the classification architecture used. The measurement levels.

2.8.2 Bagging

A lot of research has been concentrated on improving single-classifier systems mainly because of their lack in sufficient resources for simultaneously developing several different classifiers. A simple method for generating multiple classifiers in those cases is to run several training sessions with the same single-classifier system and different subsets of the training set, or slightly modified classifier parameters. Each training session then creates an individual classifier. The first systematic approach to this idea was proposed by Leo Breiman back in the 90s and became popular under the name "Bagging." This method draws the training sets with replacement from the original training set, each set resulting in a slightly different classifier after training. This technique is one of the several bootstrap techniques used for generating individual training sets and aims at reducing the error of statistical estimators. In practice, bagging has shown good results. However, the performance gains are usually small when bagging is applied to weak classifiers. In these cases, boosting which is another technique can be applied.

2.8.3 Boosting - Ada Boost algorithm

Boosting deals with the question whether an almost randomly guessing classifier can be boosted into an arbitrarily accurate learning algorithm. Boosting attaches a weight to each instance in the training set and these weights are updated after each training cycle according to the performance of the classifier on the corresponding training samples. Initially, all weights are set equally, but on each round, the

weights of incorrectly classified samples are increased so that the classifier is forced to focus on the hard examples in the training set.

A very popular type of boosting is AdaBoost (Adaptive Boosting), which was introduced by Freund and Schapire in 1995 to expand the boosting approach introduced by Schapire. The AdaBoost algorithm generates a set of classifiers and votes them. It changes the weights of the training samples based on classifiers previously built (trials). The goal is to force the final classifiers to minimize expected error over different input distributions. The final classifier is formed using a weighted voting scheme.

AdaBoost is best used to boost the performance of decision trees on binary classification problems. AdaBoost was originally called AdaBoost.M1 by the authors of the technique Freund and Schapire. More recently it may be referred to as discrete AdaBoost because it is used for classification rather than regression. AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem. The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps. Each instance in the training dataset is weighted. The initial weight is set to:

$$weight(x_i) = \frac{1}{n}$$

Where x_i is the i^{th} training instance and n is the number of training instances.

The AdaBoost algorithm

1 **Input:** $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, Number of Iterations T

2 **Initialize:** $d_n^{(1)} = \frac{1}{N}$ for all $n = 1, \ldots, N$

3 **Do for** $t = 1, \ldots, T$

a Train classifier with respect to the weighted sample set $\{S, d_t\}$ and obtain hypothesis $h_t : x \mapsto \{-1, +1\}$ i.e. $h_t = \mathcal{L}(S, d_t)$

b Calculate the weighted training error ε_t of h_t:

$$\varepsilon_t = \sum_{n=1}^{N} d_n^{(t)} I(y_n \neq h_t(x_n))$$

c Set:

$$\alpha_t = \frac{1}{2} log \frac{1 - \varepsilon_t}{\varepsilon_t}$$

d Update weights:

$$d_n^{(t+1)} = d_n^{(t)} exp\{-\alpha_t y_n h_t(x_n)\}/Z_t$$

where Z_t is a normalization constant, such that $\sum_{n=1}^{N} d_n^{(t+1)} = 1$.

4 **Break if** $\varepsilon_t = 0$ or $\varepsilon_t \geq \frac{1}{2}$ and set $T = t - 1$.

5 **Output:** $f_T(x) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{r=1}^{T} \alpha_r} h_t(x)$

After selecting the hypothesis h_t in $\varepsilon_t(h_t, d^{(t)}) = \sum_{n=1}^{N} d_n^{(t)} I(y_n \neq h_t(x_n))$ its weight α_t is computed such that it minimizes a certain loss function (Step 3c). In AdaBoost we minimizes $G^{AB}(\alpha) = \sum_{n=1}^{N} exp\{-y_n(\alpha h_t(x_n) + f_{t-1}(x_n))\}$. where f_{t-1} is the combined hypothesis of the previous iteration given by $f_{t-1}(x_n) = \sum_{r-1}^{t-1} \alpha_r h_r(x_n)$.

2.8.4 Evaluating and debugging learning algorithms

Once you have defined your problem and prepared your data you need to apply machine learning algorithms to the data in order to solve your problem. You can spend a lot of time choosing, running and tuning algorithms. You want to make sure you are using your time effectively to get closer to your goal. We will step through a process to rapidly test algorithms and discover whether or not there is structure in your problem for the algorithms to learn and which algorithms are effective.

Test Harness

You need to define a test harness. The test harness is the data you will train and test an algorithm against and the performance measure you will use to assess its performance. It is important to define your test harness well so that you can focus on evaluating different algorithms and thinking deeply about the problem. The goal of the test harness is to be able to quickly and consistently test algorithms against a fair representation of the problem being solved. The outcome of testing multiple algorithms against the harness will be an estimation of how a variety of algorithms perform on the problem against a chosen performance measure. You will know which algorithms might be worth tuning on the problem and which should not be considered further. The results will also give you an indication of how learnable the problem is. If a variety of different learning algorithms university perform poorly on the problem, it may be an indication of a lack of structure available to algorithms to learn. This may be because there actually is a lack of learnable structure in the selected data or it may be an opportunity to try different transforms to expose the structure to the learning algorithms.

Performance Measure

The performance measure is the way you want to evaluate a solution to the problem. It is the measurement you will make of the predictions made by a trained model on the test dataset. Performance measures are typically specialized to the class of problem you are working with, for example classification, regression, and clustering. Many standard performance measures will give you a score that is meaningful to your problem domain. For example, classification accuracy for classification (total correct correction divided by the total predictions made multiple by 100 to turn it into a percentage). You may also want a more detailed breakdown of performance, for example, you may want to know about the false positives on a spam classification problem because good email will be marked as spam and cannot be read. There are many standard performance measures to choose from. You rarely have to devise a new performance measure yourself as you can generally find or adapt one that best captures the requirements of the problem being solved. Look to similar problems you uncovered and at the performance measures used to see if any can be adopted.

Test and Train Datasets

From the transformed data, you will need to select a test set and a training set. An algorithm will be trained on the training dataset and will be evaluated against the test set. This may be as simple as selecting a random split of data (66% for training, 34% for testing) or may involve more complicated sampling methods. A trained model is not exposed to the test dataset during training and any predictions made on that dataset are designed to be indicative of the performance of the model in general. As such you want to make sure the selection of your datasets are representative of the problem you are solving.

Cross Validation

A more sophisticated approach than using a test and train dataset is to use the entire transformed dataset to train and test a given algorithm. A method you could

use in your test harness that does this is called cross validation. It first involves separating the dataset into a number of equally sized groups of instances (called folds). The model is then trained on all folds exception one that was left out and the prepared model is tested on that left out fold. The process is repeated so that each fold get's an opportunity at being left out and acting as the test dataset. Finally, the performance measures are averaged across all folds to estimate the capability of the algorithm on the problem. For example, a 3-fold cross validation would involve training and testing a model 3 times:

1: Train on folds 1+2, test on fold 3

2: Train on folds 1+3, test on fold 2

3: Train on folds 2+3, test on fold 1

The number of folds can vary based on the size of your dataset, but common numbers are 3, 5, 7 and 10 folds. The goal is to have a good balance between the size and representation of data in your train and test sets. When you're just getting started, stick with a simple split of train and test data (such as 66%/34%) and move onto cross validation once you have more confidence.

Testing Algorithms

When starting with a problem and having defined a test harness you are happy with, it is time to spot check a variety of machine learning algorithms. Spot checking is useful because it allows you to very quickly see if there is any learnable structures in the data and estimate which algorithms may be effective on the problem. Spot checking also helps you work out any issues in your test harness and make sure the chosen performance measure is appropriate. The best first algorithm to spot check is a random. Plug in a random number generator to generate predictions in the appropriate range. This should be the worst "algorithm result" you achieve and will be the measure by which all improvements can be

assessed. Select 5-10 standard algorithms that are appropriate for your problem and run them through your test harness. By standard algorithms, I mean popular methods no special configurations. Appropriate for your problem means that the algorithms can handle regression if you have a regression problem. Choose methods from the groupings of algorithms we have already reviewed. I like to include a diverse mix and have 10-20 different algorithms drawn from a diverse range of algorithm types. Depending on the library I am using, I may spot check up to a 50+ popular methods to flush out promising methods quickly. If you want to run a lot of methods, you may have to revisit data preparation and reduce the size of your selected dataset. This may reduce your confidence in the results, so test with various data set sizes. You may like to use a smaller size dataset for algorithm spot checking and a fuller dataset for algorithm tuning.

2.8.5 Classification errors

In this section, we develop a categorization for prediction errors considering both training set and generalization errors. We also demonstrate that our categorization is exhaustive, that is, we provide a characterization of prediction errors. Our categorization is relative to a particular training set T, feature set F, and learning algorithm $\mathcal{L}$. We describe four categories of errors: mislabelling errors, representation errors, learner errors, and boundary errors. Generalization errors are of a different nature than training set prediction errors due to the fact that they are not in the training set. This difference is important because the teacher can only see a generalization error when they provide a label for an object not in the training set. We classify the types of generalization errors relative to a particular training set T, feature set F, and learning algorithm $\mathcal{L}$ by considering the result of adding a correctly labelled version of the object to the training set.

- **Mislabelling Errors**

  A mislabeling error is a labeled object such that the label does not agree with the target classification function. At first glance it is not clear that mislabelling errors have anything to do with a prediction error, however, mislabelling errors can give rise to prediction errors.

- **Learner Errors**

  A learner error is a prediction error that arises due to the fact that the learner does not find a classification function that correctly predict the training set when such a learnable classifier exists.

- **Representation Errors**

  A representation error is a prediction error that arises due to the fact that there is no learnable classification function that correctly predicts the training set.

- **Boundary Errors**

  Our final type of prediction error is a type of generalization error. A boundary error is a prediction error for an object if adding to the training set yields a classification function that correctly predicts the augmented training set.

UNIT III

3.1 Unsupervised learning

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labelled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabelled data by our-self. Unsupervised learning classified into two categories of algorithms:

Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
Association: An association rule learning problem is where you want to discover

rules that describe large portions of your data, such as people that buy X also tend to buy Y.

3.2 Clustering K means

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).
K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian mixture modeling. They both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means due to the name. Applying the 1-nearest neighbor classifier to the cluster centers obtained by k-means classifies new data into the existing clusters. This is known

as nearest centroid classifier or Rocchio algorithm.

3.2.1 K means Algorithm

k-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

where

$\|x_i - v_j\|$ is the Euclidean distance between x_i and v_j

c_i is the number of data points in i^{th} cluster.

c is the number of cluster centers.

Let $X = \{x_1, x_2, x_3, \ldots, x_n\}$ be the set of data points and $V = \{v_1, v_2, \ldots, v_c\}$ be the set of centers.

1 Randomly select c cluster centers.

2 Calculate the distance between each data point and cluster centers.

3 Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers

4 Recalculate the new cluster center using:

$$v = \frac{1}{c_i} \sum_{j=1}^{c_i} x_i$$

Where c_i represents the number of data points in i^{th} cluster.

5 Recalculate the distance between each data point and new obtained cluster centers.

6 If no data point was reassigned then stop, otherwise repeat from step 3)

3.2.2 Pros and Cons

Pros

- Fast, robust and easier to understand

- Relatively efficient: $O(tknd)$, where n is number of objects, k is number of clusters, d is number of dimension of each object, and t is number of iterations. Normally, $k, t, d << n$.
- Gives best result when data set are distinct or well separated from each other.

Cons

- The learning algorithm requires apriori specification of the number of cluster centers.
- The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.
- The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).
- Euclidean distance measures can unequally weight underlying factors.
- The learning algorithm provides the local optima of the squared error function.
- Randomly choosing of the cluster center cannot lead us to the fruitful result.
- Applicable only when mean is defined i.e. fails for categorical data.
- Unable to handle noisy data and outliers.
- Algorithm fails for non-linear data set.

3.3 Expectation-Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm is a way to find maximum-likelihood estimates for model parameters when your data is incomplete, has missing data points, or has unobserved (hidden) latent variables. It is an iterative way to approximate the maximum likelihood function. While maximum likelihood estimation can find the "best fit" model for a set of data, it doesn't work particularly well for incomplete data sets. The more complex EM algorithm can find model parameters even if you have missing data. It works by choosing random values for the missing data points, and using those guesses to estimate a second set of data. The new values are used to create a better guess for the first set, and the process continues until the algorithm converges on a fixed point. The Expectation-Maximization (EM) Algorithm is an iterative method to find the Maximum Likelihood Estimation (MLE) or Maximum a-posteriori estimates (MAP) estimate for models with latent variables. This is a description of how the algorithm works:

1 **Initialization:** Initialization: Get an initial estimate for parameters θ^0 (e.g. all the μ_k, μ_k^2 and π variables). In many cases, this can just be a random initialization.

2 **Expectation Step:** Assume the parameters $(\theta^t - 1)$ from the previous step are fixed, compute the expected values of the latent variables (or more often a function of the expected values of the latent variables)

3 **Maximization Step:** Given the values you computed in the last step (essentially known values for the latent variables), estimate new values for θ^t that maximize a variant of the likelihood function.

4 **Exit Condition:** If likelihood of the observations have not changed much, exit; otherwise, go back to Step 1.

3.3.1 Limitations

The EM algorithm can very very slow, even on the fastest computer. It works best when you only have a small percentage of missing data and the dimensionality of the data isn't too big. The higher the dimensionality, the slower the E-step; for data with larger dimensionaloty, you may find the E-step runs extremely slow as the procedure approaches a local maximum.

3.4 Mixture of Gaussians

Gaussian mixture models are a probabilistic model for representing normally distributed subpopulations within an overall population. Mixture models in general don't require knowing which subpopulation a data point belongs to, allowing the model to learn the subpopulations automatically. Since subpopulation assignment is not known, this constitutes a form of unsupervised learning.

For example, in modeling human height data, height is typically modeled as a normal distribution for each gender with a mean of approximately 5'10" for males and 5'5" for females. Given only the height data and not the gender assignments for each data point, the distribution of all heights would follow the sum of two scaled (different variance) and shifted (different mean) normal distributions. A model making this assumption is an example of a Gaussian mixture model (GMM), though in general a GMM may have more than two components. Estimating the

parameters of the individual normal distribution components is a canonical problem in modeling data with GMMs.

GMMs have been used for feature extraction from speech data, and have also been used extensively in object tracking of multiple objects, where the number of mixture components and their means predict object locations at each frame in a video sequence.

A Gaussian mixture model is parameterized by two types of values, the mixture component weights and the component means and variances/covariances. For a Gaussian mixture model with K components, the component has a mean of μ_k and variance of σ_k for the univariate case and a mean of $\vec{\mu}_k$ and covariance matrix of $\sum_k$ for the multivariate case. The mixture component weights are defined as ϕ_k for component C_k, with the constraint that $\sum_{i=1}^{K} \phi_i = 1$ so that the total probability distribution normalizes to 1. If the component weights aren't learned, they can be viewed as an a-priori distribution over components such that $p(x$ generated by component $C_k) = \phi_k$. If they are instead learned, they are the a-posteriori estimates of the component probabilities given the data.

One-dimensional Model

$$p(x) = \sum_{i=1}^{K} \phi_i N(x|\mu_i, \sigma_i)$$

$$N(x|\mu_i, \sigma_i) = \frac{1}{\sigma_i\sqrt{2\pi}} exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

Multi-dimensional Model

$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i N(\vec{x}|\vec{\mu}_i, \sum_i)$$

$$N(\vec{x}|\vec{\mu}_i, \sum_i) = \frac{1}{\sqrt{(2\pi)^K |\sum_i|}} exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_i)^T \sum_i^{-1} (\vec{x} - \vec{\mu}_i)\right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

3.4.1 EM for Gaussian Mixture Models

Expectation maximization for mixture models consists of two steps. The first step, known as the expectation step or E step, consists of calculating the expectation of the component assignments C_k for each data point $x_i \in X$ given the model parameters ϕ_k, μ_k and σ_k.

The second step is known as the maximization step or M step, which consists of maximizing the expectations calculated in the E step with respect to the model parameters. This step consists of updating the values ϕ_k, μ_k and σ_k.

The entire iterative process repeats until the algorithm converges, giving a maximum likelihood estimate. Intuitively, the algorithm works because knowing the component assignment C_k for each x_i makes solving for ϕ_k, μ_k and σ_k easy, while knowing ϕ_k, μ_k and σ_k makes inferring $P(C_k|x_i)$ easy. The expectation step corresponds to the latter case while the maximization step corresponds to the former.

Thus, by alternating between which values are assumed fixed, or known, maximum likelihood estimates of the non-fixed values can be calculated in an efficient manner.

3.4.2 Algorithm for Univariate Gaussian Mixture Models

The expectation maximization algorithm for Gaussian mixture models starts with an initialization step, which assigns model parameters to reasonable values based on the data. Then, the model iterates over the expectation (E) and maximization (M) steps until the parameters' estimates converge, i.e. for all parameters θ_t at iteration $t, |\theta_t - \theta_{t-1}| \leq \epsilon$ for some user-defined tolerance ϵ. The EM algorithm for a univariate Gaussian mixture model with K components is described below. A variable denoted $\hat{\theta}$ denotes an estimate for the value θ. All equations can be derived algebraically by solving for each parameter as outlined in the section above titled EM for Gaussian Mixture Models.

Initialization Step:

- Randomly assign samples without replacement from the dataset $X = \{x_1, \ldots, x_N\}$ to the component mean estimates $\hat{\mu}_1, \ldots, \hat{\mu}_K$ E.g. for $K = 3$ and $N = 100$, set $\hat{\mu}_1 = x_{45}, \hat{\mu}_2 = x_{32}, \hat{\mu}_3 = x_{10}$
- Set all component variance estimates to the sample variance $\hat{\sigma}_1^2, \ldots, \hat{\sigma}_K^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$ where $\bar{x}$ is the sample mean $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$.
- Set all component distribution prior estimates to the uniform distribution $\hat{\phi}_1, \ldots, \hat{\phi}_K = \frac{1}{K}$

Expectation (E) Step:

- Calculate $\forall i, k$

$$\hat{\gamma}_{ik} = \frac{\hat{\phi}_k N(x_i|\hat{\mu}_k, \hat{\sigma}_k)}{\sum_{j=1}^{K} \hat{\phi}_j N(x_i|\hat{\mu}_j, \hat{\sigma}_j}$$

  where $\hat{\gamma}_{ik}$ is the probability that x_i is generated by component C_k. Thus, $\hat{\gamma}_{ik} = p(C_k|x_i, \hat{\phi}, \hat{\mu}, \hat{\sigma})$.

Maximization (M) Step:
Using the $\hat{\gamma}_{ik}$ calculated in the expectation step, calculate the following in that order $\forall k$:

- $\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$
- $\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} x_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$
- $\hat{\sigma}_k^2 = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik}(x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$

When the number of components K is not known a priori, it is typical to guess the number of components and fit that model to the data using the EM algorithm. This is done for many different values of K. Usually, the model with the best trade-off between fit and number of components (simpler models have fewer components) is kept. The EM algorithm for the multivariate case is analogous, though it is more complicated and thus is not expounded here.

3.5 Factor Analysis

Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables

called factors. For example, it is possible that variations in six observed variables mainly reflect the variations in two unobserved (underlying) variables. Factor analysis searches for such joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors, plus "error" terms. Factor analysis aims to find independent latent variables.

It is a theory used in machine learning and related to data mining. The theory behind factor analytic methods is that the information gained about the interdependencies between observed variables can be used later to reduce the set of variables in a dataset. Factor analysis is commonly used in biology, psychometrics, personality theories, marketing, product management, operations research, and finance. It may help to deal with data sets where there are large numbers of observed variables that are thought to reflect a smaller number of underlying/latent variables. It is one of the most commonly used inter-dependency techniques and is used when the relevant set of variables shows a systematic inter-dependence and the objective is to find out the latent factors that create a commonality

3.5.1 Types of factoring:

Principal component analysis:

This is the most common method used by researchers. PCA starts extracting the maximum variance and puts them into the first factor. After that, it removes that variance explained by the first factors and then starts extracting maximum variance for the second factor. This process goes to the last factor.

Common factor analysis:

The second most preferred method by researchers, it extracts the common variance and puts them into factors. This method does not include the unique variance of

all variables. This method is used in SEM.

Image factoring:

This method is based on correlation matrix. OLS Regression method is used to predict the factor in image factoring.

Maximum likelihood method:

This method also works on correlation metric but it uses maximum likelihood method to factor.

Other methods of factor analysis:

Alfa factoring outweighs least squares. Weight square is another regression based method which is used for factoring.

3.6 Principal Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. If there are n observations with p variables, then the number of distinct principal components is min(n-1,p). This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing n observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualised as a set of coordinates in a high-dimensional data space, PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced. PCA is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix.

3.6.1 Properties of Principal Component

Technically, a principal component can be defined as a linear combination of optimally-weighted observed variables. The output of PCA are these principal components, the number of which is less than or equal to the number of original variables. Less, in case when we wish to discard or reduce the dimensions in our dataset. The PCs possess some useful properties which are listed below:

1. The PCs are essentially the linear combinations of the original variables, the weights vector in this combination is actually the eigenvector found which in turn satisfies the principle of least squares.
2. The PCs are orthogonal.
3. The variation present in the PCs decrease as we move from the 1st PC to the last one, hence the importance.

The least important PCs are also sometimes useful in regression, outlier detection, etc.

3.6.2 Implementing PCA on a 2-D Dataset

1 **Normalize the data**

First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become x' and all Y become y'. This produces a dataset whose mean is zero.

2 **Calculate the covariance matrix**

Since the dataset we took is 2-dimensional, this will result in a 2×2 Covariance matrix

$$Matrix(Covariance) = \begin{bmatrix} Var[X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Var[X_2] \end{bmatrix}$$

Please note that $Var[X_1] = Cov[X_1, X_1]$ and $Var[X_2] = Cov[X_2, X_2]$.

3 **Calculate the eigenvalues and eigenvectors**

Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. λ is an eigenvalue for a matrix A if it is a solution of the characteristic equation:

$|\lambda I - A| = 0$.

Where, I is the identity matrix of the same dimension as A which is a required condition for the matrix subtraction as well in this case and $||$ is the determinant of the matrix. For each eigenvalue λ, a corresponding eigenvector v, can be found by solving:

$(\lambda I - A)v = 0$

4 **Choosing components and forming a feature vector**

We order the eigenvalues from largest to smallest so that it gives us the

components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater eigenvalue or simply take both.

$FeatureVector = (\lambda_1, \lambda_2)$

5 **Forming Principal Components**

This is the final step where we actually form the principal components using all the math we did till here. For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.

$NewData = FeatureVector^T \times ScaledData^T$

Here, $NewData$ is the Matrix consisting of the principal components, $FeatureVector$ is the matrix we formed using the eigenvectors we chose to keep, and $ScaledData$ is the scaled version of original dataset ('T' in the superscript denotes transpose of a matrix which is formed by interchanging the rows to columns and vice versa. In particular, a 2×3 matrix has a transpose of size 3×2) If we go back to the theory of eigenvalues and eigenvectors, we see that, essentially, eigenvectors provide us with information about the patterns in the data. In particular, in the running example of 2-D set, if we plot the eigenvectors on

the scatter plot of data, we find that the principal eigenvector (corresponding to the largest eigenvalue) actually fits well with the data. The other one, being perpendicular to it, does not carry much information and hence, we are at not much loss when deprecating it, hence reducing the dimension.
All the eigenvectors of a matrix are perpendicular to each other. So, in PCA, what we do is represent or transform the original dataset using these orthogonal (perpendicular) eigenvectors instead of representing on normal x and y axes. We have now classified our data points as a combination of contributions from both x and y. The difference lies when we actually disregard one or many eigenvectors, hence, reducing the dimension of the dataset. Otherwise, in case, we take all the eigenvectors in account, we are just transforming the co-ordinates and hence, not serving the purpose.

3.7 Independent Component Analysis

Independent component analysis (ICA) is a method for finding underlying factors or components from multivariate (multi-dimensional) statistical data. What distinguishes ICA from other methods is that it looks for components that are both statistically independent, and non-Gaussian. Independent component analysis (ICA) is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals.

ICA defines a generative model for the observed multivariate data, which is typically given as a large database of samples. In the model, the data variables are assumed to be linear mixtures of some unknown latent variables, and the mixing

system is also unknown. The latent variables are assumed nongaussian and mutually independent, and they are called the independent components of the observed data. These independent components, also called sources or factors, can be found by ICA.

The data analyzed by ICA could originate from many different kinds of application fields, including digital images, document databases, economic indicators and psychometric measurements. In many cases, the measurements are given as a set of parallel signals or time series; the term blind source separation is used to characterize this problem. Typical examples are mixtures of simultaneous speech signals that have been picked up by several microphones, brain waves recorded by multiple sensors, interfering radio signals arriving at a mobile phone, or parallel time series obtained from some industrial process.

3.7.1 Definition of ICA

In practical situations, we cannot in general find a representation where the components are really independent, but we can at least find components that are as independent as possible. This leads us to the following definition of ICA.
Given a set of observations of random variables $(x_1(t), (x_2(t), \ldots, (x_n(t))$, where t is the time or sample index, assume that they are generated as a linear mixture of independent components:

$$\begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{pmatrix} = A \begin{pmatrix} s_1(t) \\ s_2(t) \\ \vdots \\ s_n(t) \end{pmatrix} \tag{3.1}$$

where A is some unknown matrix. Independent component analysis now consists of estimating both the matrix A and the $si(t)$, when we only observe the $xi(t)$. Note that we assumed here that the number of independent components s_i is equal to the number of observed variables; this is a simplifying assumption that is not completely necessary.

Alternatively,we could define ICA as follows: find a linear transformation given by a matrix W, so that the random variables $y_i, i = 1, \ldots, n$ are as independent as possible. Since after estimating A its inverse gives W.

It can be shown that the problem is well defined, that is, the model in eq. 3.1 can be estimated if and only if the components s_i are nongaussian. This is a fundamental requirement that also explains the main difference between ICA and factor analysis, in which the nongaussianity of the data is not taken into account. In fact, ICA could be considered as nongaussian factor analysis, since in factor analysis, we are also modeling the data as linear mixtures of some underlying factors.

3.7.2 Applications

Due to its generality the ICA model has applications in many different areas. Some examples are:

- In brain imaging, we often have different sources in the brain emit signals that are mixed up in the sensors outside of the head, just like in the basic blind source separation model.
- In econometrics, we often have parallel time series, and ICA could decompose them into independent components that would give an insight to the structure of the data set.

- A somewhat different application is in image feature extraction, where we want to find features that are as independent as possible

3.7.3 ICA estimation principles

- Nonlinear decorrelation: Find the matrix W so that for any $i \neq j$, the components y_i and y_j are uncorrelated, and the transformed components $g(y_i)$ and $h(y_j)$ are uncorrelated, where g and h are some suitable nonlinear functions.
- Maximum nongaussianity: Find the local maxima of nongaussianity of a linear combination $y = Wx$ under the constraint that the variance of x is constant.
- Each local maximum gives one independent component.
- Assumptions have many like Source signals are statistically independent knowing the value of one of the components does not give any information about the others ICs have non-gaussian distributions. There are initial distributions unknown. Currently at most one Gaussian source only the recovered sources can be permutated and scaled.

3.8 Latent Semantic Indexing

Latent semantic indexing, sometimes referred to as latent semantic analysis, is a mathematical method developed in the late 1980s to improve the accuracy of information retrieval. It uses a technique called singular value decomposition to scan unstructured data within documents and identify relationships between the

concepts contained therein. In essence, it finds the hidden (latent) relationships between words (semantics) in order to improve information understanding (indexing). It provided a significant step forward for the field of text comprehension as it accounted for the contextual nature of language. Earlier technologies struggled with the use of synonyms that characterizes natural language use, and also the changes in meanings that come with new surroundings.

For example, the words 'hot' and 'dog' may seem easy to understand, but both have multiple definitions based on how they are used. Put both of them together and you have a whole new concept altogether.

So how can we train a machine to adapt to these nuances? This is a problem that has troubled great minds for centuries and LSI has helped computers to start understanding language in use. It works best on static content and on small sets of documents, which was great for its initial purposes. LSI also allows documents to be clustered together based on their thematic commonalities, which was a very useful capability for early search engines.

3.8.1 Basic concepts

Latent Semantic Indexing is a technique that projects queries and documents into a space with "latent" semantic dimensions. In the latent semantic space, a query and a document can have high cosine similarity even if they do not share any terms - as long as their terms are semantically similar in a sense to be described later.

We can look at LSI as a similarity metric that is an alternative to word overlap measures like tf.idf.

The latent semantic space that we project into has fewer dimensions than the original space (which has as many dimensions as terms). LSI is thus a method for dimensionality reduction. A dimensionality reduction technique takes a set of objects that exist in a high-dimensional space and represents them in a low dimensional space, often in a two-dimensional or three-dimensional space for the purpose of visualization.

Latent semantic indexing is the application of a particular mathematical technique, called Singular Value Decomposition or SVD, to a word-by-document matrix. SVD (and hence LSI) is a least-squares method. The projection into the latent semantic space is chosen such that the representations in the original space are changed as little as possible when measured by the sum of the squares of the differences.

SVD takes a matrix A and represents it as $\hat{A}$ in a lower dimensional space such that the "distance" between the two matrices as measured by the 2-norm is minimized:

$$\Delta = \|A - \hat{A}\|_2 \tag{3.2}$$

The 2-norm for matrices is the equivalent of Euclidean distance for vectors. SVD project an n-dimensional space onto a k-dimensional space where $n >> k$. In our application (word-document matrices), n is the number of word types in the collection. Values of k that are frequently chosen are 100 and 150. The projection transforms a document's vector in n-dimensional word space into a vector in the k-dimensional reduced space.

There are many different mappings from high dimensional to low-dimensional spaces. Latent Semantic Indexing chooses the mapping that is optimal in the

sense that it minimizes the distance Δ. This setup has the consequence that the dimensions of the reduced space correspond to the axes of greatest variation. The SVD projection is computed by decomposing the document-by-term matrix $A_{t\times d}$ into the product of three matrices, $T_{t\times n}, S_{n\times n}, D_{d\times n}$:

$$A_{t\times d} = T_{t\times n} S_{n\times n} \left(D_{d\times n}\right)^T$$

where t is the number of terms, d is the number of documents, $n = min(t, d)$, T and D have orthonormal columns, i.e. $TT^T = DD^T = I$, $rank(A) = r$, $S = diag(\sigma_1, \sigma_2, \ldots, \sigma_n)$, $\sigma_i > 0$ for $1 \leq i \leq r, \sigma_j = 0$ for $j \geq r + 1$.

We can view SVD as a method for rotating the axes of the n-dimensional space such that the first axis runs along the direction of largest variation among the documents, the second dimension runs along the direction with the second largest variation and so forth. The matrices T and D represent terms and documents in this new space. The diagonal matrix S contains the singular values of A in descending order. The i^{th} singular value indicates the amount of variation along the i^{th} axis. By restricting the matrices T, S and D to their first $k < n$ rows one obtains the matrices $T_{t\times n}, S_{n\times n}, D_{d\times n}$. Their product $\hat{A}$ defined in Eq. 3.3 is the best square approximation of A by a matrix of rank k in the sense defined in the equation Eq. 3.2.

$$\hat{A}_{t\times d} = T_{t\times k} S_{k\times k} \left(D_{d\times k}\right)^T \tag{3.3}$$

Choosing the number of dimensions k for $\hat{A}$ is an interesting problem. While a reduction in k can remove much of the noise, keeping too few dimensions or factors may loose important information. LSI performance can improve considerably after

10 or 20 dimensions, peaks between 70 and 100 dimensions, and then begins to diminish slowly. This pattern of performance (initial large increase and slow decrease to word-based performance) is observed with other datasets as well. Eventually performance must approach the level of performance attained by standard vector methods, since with $k = n$ factors $\hat{A}$ will exactly reconstruct the original term by document matrix A. That LSI works well with a relatively small (compared to the number of unique terms) number of dimensions or factors k shows that these dimensions are, in fact, capturing a major portion of the meaningful structure.

One can also prove that SVD is unique, that is, there is only one possible decomposition of a given matrix. That SVD finds the optimal projection to a low dimensional space is the key property for exploiting word co-occurrence patterns. It is important for the LSI method that the derived $\hat{A}$ matrix does not reconstruct the original term document matrix A exactly. The truncated SVD, in one sense, captures most of the important underlying structure in the association of terms and documents, yet at the same time removes the noise or variability in word usage that plagues word-based retrieval methods. Intuitively, since the number of dimensions, k is much smaller than the number of unique terms t, minor differences in terminology will be ignored. Terms which occur in similar documents, for example, will be near each other in the k-dimensional factor space even if they never co-occur in the same document. This means that some documents, which do not share any words with a user's query, may nonetheless be near it in k-space. This derived representation, which captures term-term associations, is used for retrieval.

3.8.2 Advantages and Disadvantages

Advantages

1 **True (latent) dimensions**

The assumption in LSI (and similarly for other forms of dimensionality reduction like principal component analysis) is that the new dimensions are a better representation of documents and queries. The metaphor underlying the term "latent" is that these new dimensions are the true representation. This true representation was then obscured by a generation process that expressed a particular dimension with one set of words in some documents and a different set of words in another document. LSI analysis recovers the original semantic structure of the space and its original dimensions.

2 **Synonymy**

Synonymy refers to the fact that the same underlying concept can be described using different terms. Traditional retrieval strategies have trouble discovering documents on the same topic that use a different vocabulary. In LSI, the concept in question as well as all documents that are related to it are all likely to be represented by a similar weighted combination of indexing variables.

3 **Polysemy**

Polysemy describes words that have more than one meaning, which is common property of language. Large numbers of polysemous words in the query can reduce the precision of a search significantly. By using a reduced representation in LSI, one hopes to remove some "noise" from the data, which could be described as rare and less important usages of certain terms. (Note however that this would work only when the real meaning is close to the average meaning. Since the LSI term vector is just a weighted average of the different meanings of the term, when the real meaning differs from the average meaning, LSI may actually reduce the quality of the search).

4 **Term Dependence**

The traditional vector space model assumes term independence and terms serve as the orthogonal basis vectors of the vector space. Since there are strong associations between terms in language, this assumption is never satisfied. While term independence represents the most reasonable first-order approximation, it should be possible to obtain improved performance by using term associations in the retrieval process. Adding common phrases as search items is a simple application of this approach. On the other hand, the LSI factors are orthogonal by definition, and terms are positioned in the reduced space in a way that reflects the correlations in their use across documents. It is very difficult to take advantage of term associations without dramatically increasing the computational requirements of the retrieval problem. While the LSI solution is difficult to compute for large collections, it need only be constructed once for the entire collection and performance at retrieval time is not affected.

Disadvantages

1 **Storage**

One could also argue that the SVD representation is more compact. Many documents have more than 150 unique terms. So the sparse vector representation will take up more storage space than the compact SVD representation if we reduce to 150 dimensions. In reality, the opposite is actually true. For example, the document by term matrix for the Cranfield collection used in Hull's experiments had 90,441 non-zero entries (after stemming and stop word removal). Retaining only 100 of the possible 1399 LSI vectors requires storing 139,900 values for the documents alone. The term vectors require the storage of roughly 400,000 additional values. In addition, the LSI values are

real numbers while the original term frequencies are integers, adding to the storage costs. Using LSI vectors, we can no longer take advantage of the fact that each term occurs in a limited number of documents, which accounts for the sparse nature of the term by document matrix. With recent advances in electronic storage media, the storage requirements of LSI are not a critical problem, but the loss of sparseness has other, more serious implications.

2 **Efficiency**

One of the most important speed-ups in vector space search comes from using an inverted index. As a consequence, only documents that have some terms in common with the query must be examined during the search. With LSI, however, the query must be compared to every document in the collection. There are, however, several factors that can reduce or eliminate this drawback. If the query has more terms than its representation in the LSI vector space, then inner product similarity scores will take more time to compute in term space. For example, if relevance feedback is conducted using the full text of the relevant documents, the number of terms in the query is likely to grow to be many times the number of LSI vectors, leading to a corresponding increase in search time. In addition, using a data structure such as the k-d tree in conjunction with LSI would greatly speed the search for nearest neighbors, provided only a partial ordering of the documents is required. Most of the additional costs come in the pre-processing stage when the SVD and the k-d tree are computed, and actual search time should not be significantly degraded. Other query expansion techniques suffer even more heavily from the difficulties described above, and LSI performs relatively well for long documents due to the small number of context vectors used to describe each document. However, implementation of LSI does require an additional investment of storage and computing time.

3 **LSI and normally-distributed data**

Another obection to SVD is that, along with all other least-squares methods, it is really designed for normally-distributed data, but such a distribution is inappropriate for count data, and count data is what a term-by-document matrix consists of. The link between least squares and normal distribution can be easily seen by looking at the definition of the normal distribution.

4 **Toward a theoretical foundation**

Although (little) empirical improved performance has been observed, there is very little in the literature in the way of a mathematical theory that predicts this improved performance. In this session I briefly describe one paper that is an attempt at using mathematical techniques to rigorously explain the empirically observed improved performance of LSI, Papadimitriou starts citating an interesting mathematical fact due to Eckart and Young, often cited as an explanation of the improved performance of LSI, that states, informally, that LSI retains as much as possible the relative position (and distances) of the document vectors while projecting it to a lower-dimensional space. This may only provide an explanation of why LSI does not deteriorate too much in performance over conventional vector-space methods; it fails to justify the observed improvement in precision and recall.

3.8.3 Applications of LSI

1 **Information retrieval**

The application of Singular Value Decomposition to information retrieval was originally proposed by a group of researchers at Bellcore and called Latent Semantic Indexing in this context. At this point it should be clear how to use LSI for IR. Regarding the performances, reports that for several

information science test collections, the average precision using LSI ranged from comparable to 30% better than that obtained using standard keyword vector methods. The LSI method performs best relative to standard vector methods when the queries and relevant documents do not share many words, and at high levels of recall.

2 **Relevance Feedback**

Most of the tests of Relevance Feedback using LSI have involved a method in which the initial query is replaced with the vector sum of the documents the users has selected as relevant. The use of negative information has not yet been exploited in LSI; for example, by moving the query away from documents which the user has indicated are irrelevant. Replacing the users' query with the first relevant document improves performance by an average of 33% and replacing it with the average of the first three relevant documents improves performance by an average of 67%. Relevance feedback provides sizeable and consistent retrieval advantages. One way of thinking about the success of these methods is that many words (those from relevant documents) augment the initial query that is usually quite impoverished. LSI does some of this kind of query expansion or enhancement even without relevance information, but can be augmented with relevance information.

3 **Information Filtering**

Applying LSI to information filtering applications is straightforward. An initial sample of documents is analyzed using standard LSI/SVD tools. A users' interest is represented as one (or more) vectors in this reduced-dimension LSI space. Each new document is matched against the vector and if it is similar enough to the interest vector it is recommended to the user. Learning

methods like relevance feedback can be used to improve the representation of interest vectors over time. Performances studies are encouraging.

4 **Cross-Language Retrieval**

It is important to note that the LSI analysis makes no use of English syntax or semantics. This means that LSI is applicable to any language. In addition, it can be used for cross-language retrieval - documents are in several languages and user queries (again in several languages) can match documents in any language. What is required for cross-language applications is a common space in which words from many languages are represented.

5 **Matching People Instead of Documents**

In a couple of applications, LSI has been used to return the best matching people instead of documents. In these applications, people were represented by articles they had written. In one application, known as the Bellcore Advisor, a system was developed to find local experts relevant to users' queries. A query was matched to the nearest documents and project descriptions and the authors' organization was returned as the most relevant internal group. In another application, LSI was used to automate the assignment of reviewers to submitted conference papers. Several hundred reviewers were described by means of texts they had written, and this formed the basis of the LSI analysis. Hundreds of submitted papers were represented by their abstracts, and matched to the closest reviewers. These LSI similarities were used to assign papers to reviewers for a major human-computer interaction conference. Subsequent analyses suggested that these completely automatic assignments (which took less than 1 hour) were as good as those of human experts.

6 **Noisy Input**

Because LSI does not depend on literal keyword matching, it is especially usefulwhen the text input is noisy, as in OCR (Optical Character Reader), open input, or spelling errors. If there are scanning errors and a word (Dumais) is misspelled (as Duniais), many of the other words in the document will be spelled correctly. If these correctly spelled context words also occur in documents that contained a correctly spelled version of Dumais, then Dumais will probably be near Dunials in the k-dimensional space determined by $\hat{A}$.

3.9 Spectral clustering

Clustering is a widely used unsupervised learning method. The grouping is such that points in a cluster are similar to each other, and less similar to points in other clusters. Thus, it is up to the algorithm to find patterns in the data and group it for us and, depending on the algorithm used, we may end up with different clusters. There are 2 broad approaches for clustering Fig. 3.1:

1. **Compactness Points** that lie close to each other fall in the same cluster and are compact around the cluster center. The closeness can be measured by the distance between the observations. E.g. K-Means Clustering.

2. **Connectivity Points** that are connected or immediately next to each other are put in the same cluster. Even if the distance between 2 points is less, if they are not connected, they are not clustered together. Spectral clustering is a technique that follows this approach.

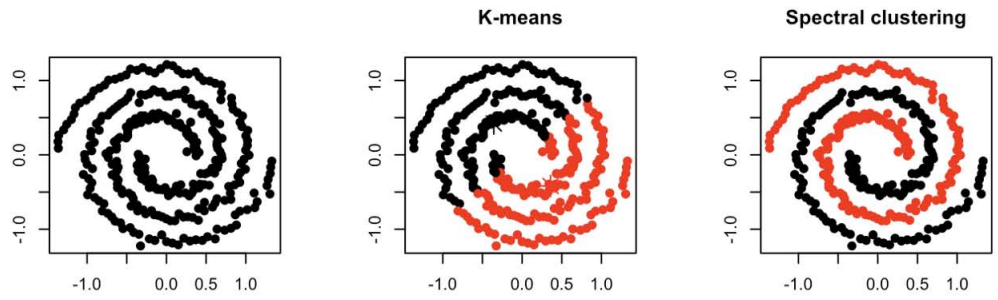

Fig. 3.1 Difference between K-Means and Spectral Clustering

3.9.1 Spectral Clustering Algorithm

In spectral clustering, the data points are treated as nodes of a graph. Thus, clustering is treated as a graph partitioning problem. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters. An important point to note is that no assumption is made about the shape/form of the clusters. Spectral clustering involves 3 steps:

1. Compute a similarity graph:
   We first create an undirected graph $G = (V, E)$ with vertex set $V = v1, v2, \ldots, vn = 1, 2, \ldots, n$ observations in the data. This can be represented by an adjacency matrix which has the similarity between each vertex as its elements . To do this, we can either compute: The ϵ-neighborhood graph, KNN Graph or Fully connected graph.

2. Project the data onto a low-dimensional space
   As we can see in Fig. 3.1, data points in the same cluster may also be far away–even farther away than points in different clusters. Our goal then is to transform the space so that when the 2 points are close, they are always in same cluster, and when they are far apart, they are in different clusters. We

need to project our observations into a low-dimensional space. For this, we compute the Graph Laplacian, which is just another matrix representation of a graph and can be useful in finding interesting properties of a graph.

3. Create clusters

   We use the eigenvector corresponding to the 2nd eigenvalue to assign values to each node. To get bipartite clustering (2 distinct clusters), we first assign each element of v2 to the nodes. We then split the nodes such that all nodes with value ¿ 0 are in one cluster, and all other nodes are in the other cluster. It is important to note that the 2nd eigenvalue indicates how tightly connected the nodes are in the graph. For good, clean partitioning, lower the 2nd eigenvalue, better the clusters. For k clusters, we have to modify our Laplacian to normalize it.

Normalized Spectral Clustering Algorithm

1. Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.
2. Construct a similarity graph by one of the ways described above. Let W be its weighted adjacency matrix.
3. Compute the normalized Laplacian L_{sym}.
4. Compute the first k eigenvectors $u_1, u_2, \ldots, u_k$ of L_{sym}.
5. Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, u_2, \ldots, u_k$ as columns.
6. Form the matrix $T \in \mathbb{R}^{n \times k}$ from U by normalizing the rows to norm 1, that is set $t_{ij} = \frac{u_{ij}}{\sqrt{\sum_k u_{ik}^2}}$
7. For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i^{th} row of T.

8. Cluster the points (y_i) $i = 1, ..., n$ with the k-means algorithm into clusters $C_1, C_2, \ldots, C_k$.

9. Output: Clusters $A_1, A_2, \ldots, A_k$ with $A_i = \{j | y_j \in C_i\}$.

3.9.2 Advantages and Disadvantages

Advantages:

- Does not make strong assumptions on the statistics of the clusters. Clustering techniques like K-Means Clustering assume that the points assigned to a cluster are spherical about the cluster center. This is a strong assumption to make, and may not always be relevant. In such cases, spectral clustering helps create more accurate clusters.
- Easy to implement and gives good clustering results. It can correctly cluster observations that actually belong to the same cluster but are farther off than observations in other clusters due to dimension reduction.
- Reasonably fast for sparse data sets of several thousand elements.

Disadvantages:

- Use of K-Means clustering in the final step implies that the clusters are not always the same. They may vary depending on the choice of initial centroids.
- Computationally expensive for large datasets. This is because eigenvalues and eigenvectors need to be computed and then we have to do clustering on these vectors. For large, dense datasets, this may increase time complexity quite a bit.

3.10 Markov Model

In probability theory, a Markov model is a stochastic model used to model randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it (that is, it assumes the Markov property). Generally, this assumption enables reasoning and computation with the model that would otherwise be intractable. For this reason, in the fields of predictive modelling and probabilistic forecasting, it is desirable for a given model to exhibit the Markov property.

3.10.1 Markov Process

A Markov process is a process that is capable of being in more than one state, can make transitions among those states, and in which the states available and transition probabilities depend only upon what state the system is currently in. In other words, there is no memory in a Markov process.

3.10.2 Markov Chain

A Markov Chain is a statistical model of a system that moves sequentially from one state to another Fig. 3.2. The probabilities of transition from one state to another are dependent only on the current state (not on previous states). Generally modeled as a stochastic process. A Markov chain can be described by a transition matrix.

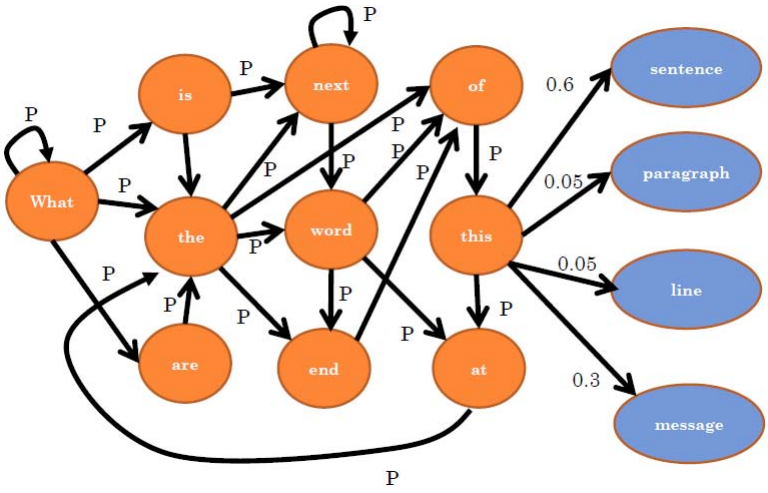

Fig. 3.2 Markov Chain

3.10.3 Example of Markov Chain

Design a Markov Chain to predict the weather of tomorrow using previous information of the past days.

- Our model has only 3 states: $\{S_1, S_2, S_3\}$ and the name of each state is $S_1 = Sunny, S_2 = Rainy, S_3 = Cloudy$.
- To establish the transition probabilities relationship between states we will need to collect data.
- Assume the data produces the following transition probabilities Fig. 3.3:
- Let's say we have a sequence: $Sunny, Rainy, Cloudy, Cloudy, Sunny, Sunny, Sunny, Rainy, \ldots$; so, in a day we can be in any of the three states.
- We can use the following state sequence notation: $q_1, q_2, q_3, q_4, \ldots$ where $q_i \in \{Sunny, Rainy, Cloudy\}$.

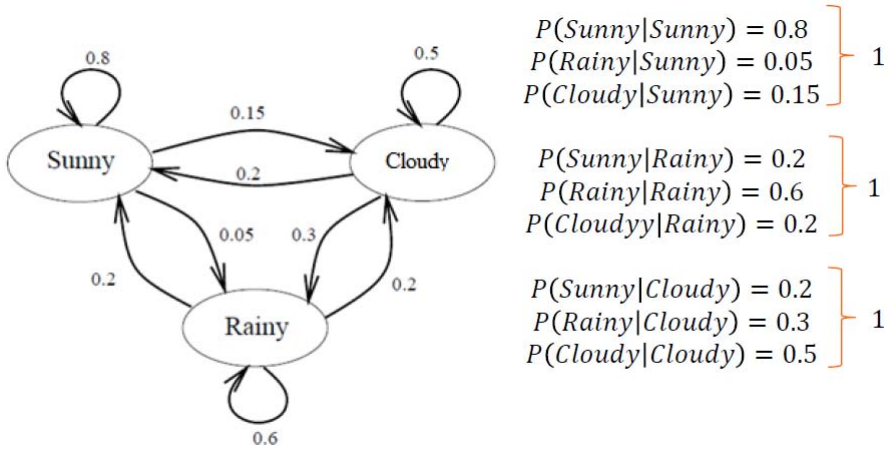

Fig. 3.3 Markov Chain for weather

- In order to compute the probability of tomorrow's weather we can use the Markov property: $p(q_1, \ldots, q_n) = \Pi_{i=1}^{n} P(q_i|q_{i-1})$

Exercise 1: Given that today is Sunny, what's the probability that tomorrow is Sunny and the next day Rainy?

Solution:

$$\begin{aligned} P(q_2, q_3|q_1) &= P(q_2|q_1)P(q_3|q_1, |q_2) \\ &= P(q_2|q_1)P(q_3|q_2) \\ &= P(Sunny|Sunny)|P(Rainy|Sunny) \\ &= 0.8 \times 0.05 \\ &= 0.04 \end{aligned}$$

Exercise 2: Assume that yesterday's weather was Rainy, and today is Cloudy, what is the probability that tomorrow will be Sunny?

Solution:

$$\begin{aligned} P(q_3|q_1, q_2) &= P(q_3|q_2) \\ &= P(Sunny|Cloudy) \\ &= 0.2 \end{aligned}$$

3.10.4 Hidden Markov Model

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modelled is assumed to be a Markov process with unobservable (i.e. hidden) states. The hidden Markov model can be represented as the simplest dynamic Bayesian network. The mathematics behind the HMM were developed by L. E. Baum and co-workers. In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters, while in the hidden Markov model, the state is not directly visible, but the output (in the form of data or "token" in the following), dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states; this is also known as pattern theory, a topic of grammar induction.

Hidden Markov models are especially known for their application in reinforcement learning and temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bio-informatics. A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a

Markov process rather than independent of each other. Recently, hidden Markov models have been generalized to pairwise Markov models and triplet Markov models which allow consideration of more complex data structures and the modelling of non-stationary data.

3.10.5 HMM Terminology

A HMM Model is specified by:

- The set of states $S = \{s_1, s_2, \ldots, s_N\}$ and a set of parameters $\Theta = \{\pi, A, B\}$
- The prior probabilities $\pi_i = P(q_1 = s_i)$ are the probabilities of s_i being the first state of a state sequence collected in a vector π.
- The transition probabilities are the probabilities to go from state i to state j: $a_{i,j} = P(q_{n+1} = s_j | q_n = s_i)$. They are collected in the matrix A.
- The emission probabilities characterize the likelihood of a certain observation x, if the model is in state s_i. Depending on the kind of observation x we have:
  - for discrete observations, $x_n \in \{v_1, \ldots, v_K\} : b_{i,k} = P(x_n = v_k | q_n = s_i)$, the probabilities to observe v_k if the current state is $q_n = s_i$. The numbers $b_{i,k}$ can be collected in a matrix B.
  - for continuous valued observations, e.g., $x_n \in \mathbb{R}^D$: A set of functions $b_i(x_n) = p(x_n | q_n = s_i)$ describing the probability densities functions over the observation space for the system being in state s_i. Collected in the vector $B(x)$ of functions.

The operation of a HMM is characterized by

- The (hidden) state sequence $Q = \{q_1, q_2, \ldots, q_N\}$, $q_n \in S$.

- The observation sequence $X = \{x_1, x_2, \ldots, x_N\}$.

A HMM allowing for transitions from any emitting state to any other emitting state is called an ergodic HMM. The other extreme, a HMM where the transitions only go from one state to itself or to a unique follower is called a left-right HMM.
Useful formula:

- **Probability of a state sequence**: the probability of a state sequence $Q = \{q_1, q_2, \ldots, q_N\}$ coming from a HMM with parameters Θ corresponds to the product of the transition probabilities from one state to the following:

$$P(Q|\Theta) = \pi_{q_1} \Pi_{n=1}^{N-1} a_{q_n, q_{n+1}} = \pi_{q_1}.a_{q_1,q_2}.a_{q_2,q_3}.\ldots.a_{q_{N-1},q_N}.$$

- **Likelihood of an observation sequence given a state sequence, or likelihood of an observation sequence along a single path**: given an observation sequence $X = \{x_1, x_2, \ldots, x_N\}$ and a state sequence $Q = \{q_1, q_2, \ldots, q_N\}$ (of the same length) determined from a HMM with parameters Θ, the likelihood of X along the path Q is equal to:

$$P(X|Q, \Theta) = \Pi_{n=1}^{N} P(x_n|q_n, \Theta) = b_{q_1,x_1}.b_{q_2,x_2}.\ldots.b_{q_N,x_N}$$

  i.e., it is the product of the emission probabilities computed along the considered path.

- **Joint likelihood of an observation sequence X and a path Q**: it is the probability that X and Q occur simultaneously, $P(X, Q|\Theta)$, and decomposes

into a product of the two quantities defined previously:

$$P(X, Q|\Theta) = P(X|Q, \Theta).P(Q|\Theta) \text{ (Bayes)}$$

- **Likelihood of a sequence with respect to a HMM**: the likelihood of an observation sequence X = x1, x2, . . . , xN with respect to a Hidden Markov Model with parameters Θ expands as follows:

$$P(X|\Theta) = \sum_{all\ Q} P(X, Q|\Theta)$$

i.e., it is the sum of the joint likelihoods of the sequence over all possible state sequences Q allowed by the model.

3.10.6 Applications of HMM

- Speech recognition
- Handwriting recognition
- Gesture recognition
- Speech tagging
- Musical score following
- Bioinformatics
- Data compression
- Computer vision applications

UNIT IV

4.1 Reinforcement Learning and Control

4.2 MDP

4.3 Bellman equations

4.4 Value iteration and policy iteration

4.5 Linear quadratic regularization (LQR)

4.6 LQG Q-learning

4.7 Value function approximation

4.8 Policy search

4.9 Reinforce

4.10 POMDPs