UNIT II

Ashok Kumar Yadav

Feb 2018

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".

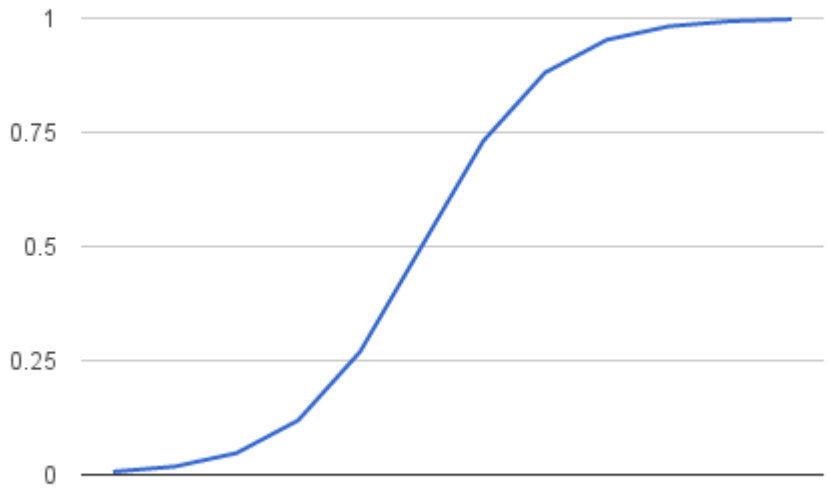

Fig. 1 Logistic Function

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.

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, 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. **??** is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.

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:

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

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).

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.

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.

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 **??** 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.

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).

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 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. **??** :

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

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

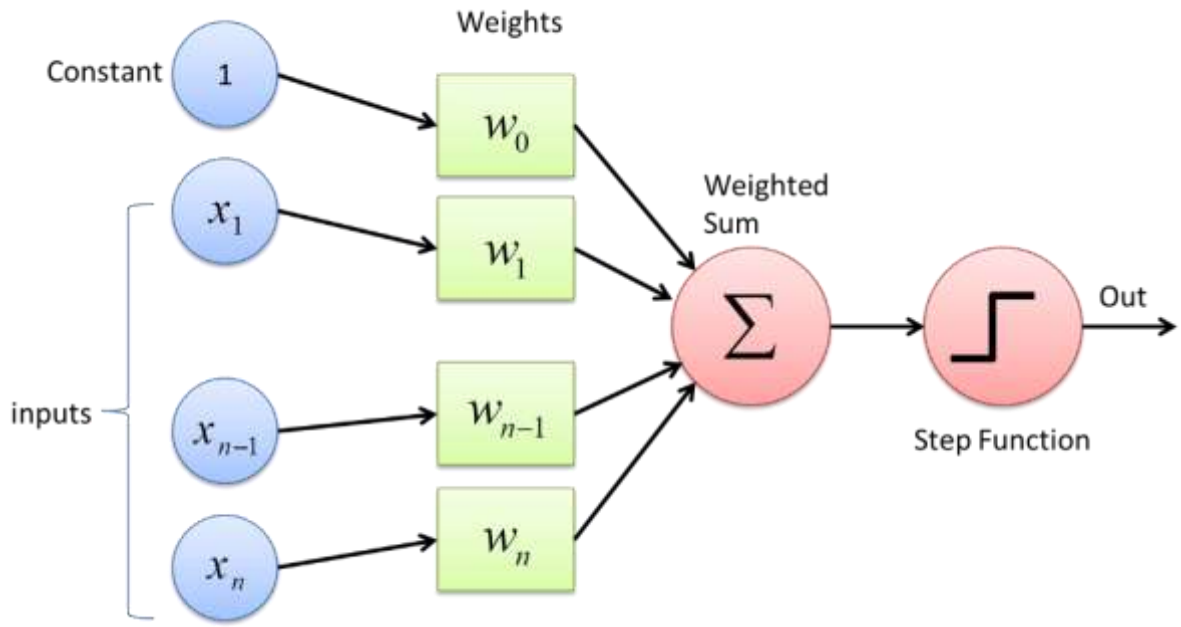

Fig. 2 Parts of Perceptron

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 $\sum$.
- 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. **??**.

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

2.2 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:

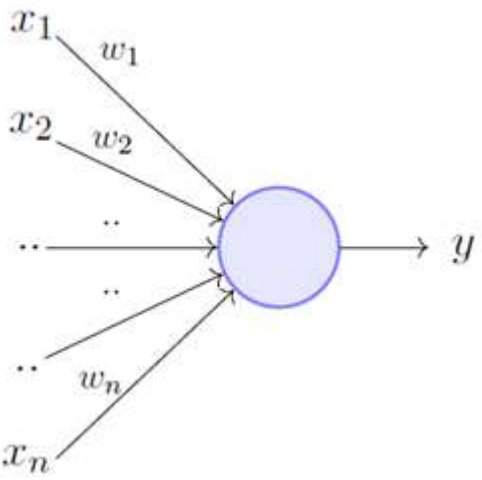

Fig. 3 Working of Perceptron

Step 1. P $\leftarrow$ input with label 1

Step 2. N $\leftarrow$ 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.

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.

3.1 Examples of exponential family

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

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.

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.

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 $X \sim Exp(\lambda)$.

3.1.4 Bernoulli distribution

3.1.5 Binomial distribution

3.1.6 Multinomial distribution

3.1.7 Gamma distribution

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.*

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:

$$P(x, y) = P(x|y).P(y) \tag{2}$$

Also, using Bayes' Rule we can write:

$$P(y|x) = \frac{P(x|y).P(y)}{P(x)} \tag{3}$$

Since, to predict a class label y, we are only interested in the argmax , the denominator can be removed from Eq. **??**. Hence to predict the label y from the training example x, generative models evaluate:

$$f(x) = \arg\max_y P(y|x) = \arg\max_y P(x|y).P(y) \tag{4}$$

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. **??**, 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)

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:

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

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. **??** as follows: Shown in the Fig. **??** 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 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 $y=0$.

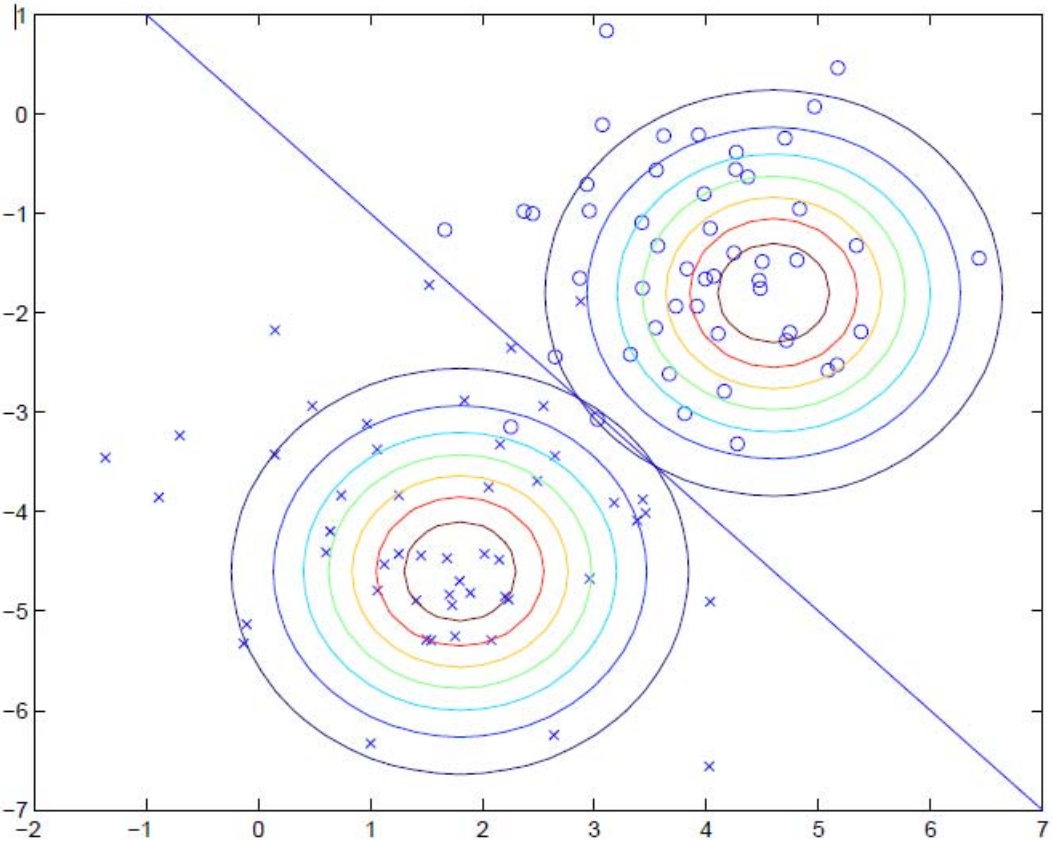

Fig. 4 GDA Model

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."

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

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

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

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

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

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

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 **??** for all the features against the different classes. What can we conclude from the above table?

Table 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 **??** for the features against the classes.

Table 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.

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.

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.

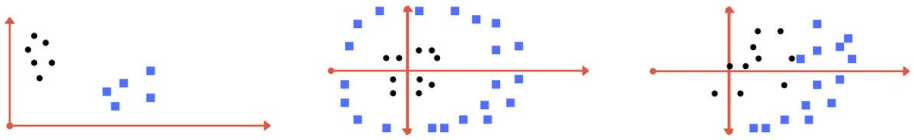

Fig. 5 Three simple graphs

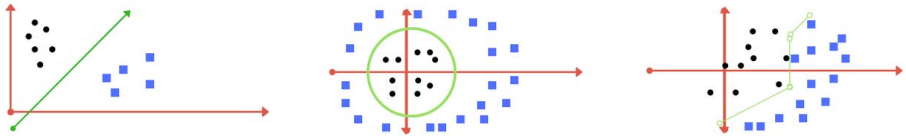

Fig. 6 Classification of Fig. ??

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 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.

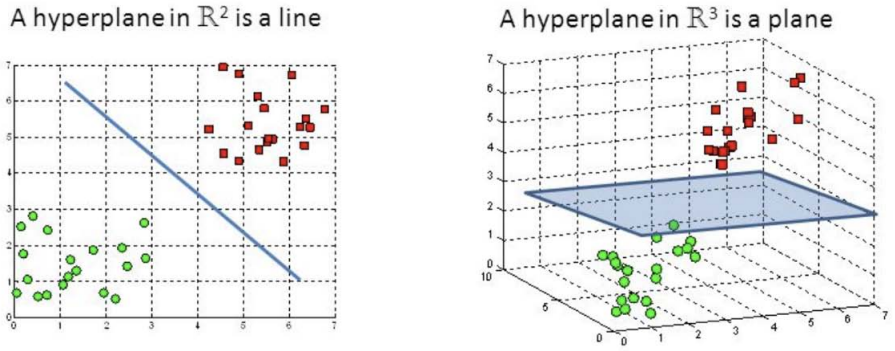

Fig. 7 Hyperplanes in 2D and 3D feature space

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. **??**. If the number of input features is 3, then the hyperplane becomes a three-dimensional plane Fig. **??**. It becomes difficult to imagine when the number of features exceeds 3. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane Fig. **??**. 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. **??**

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

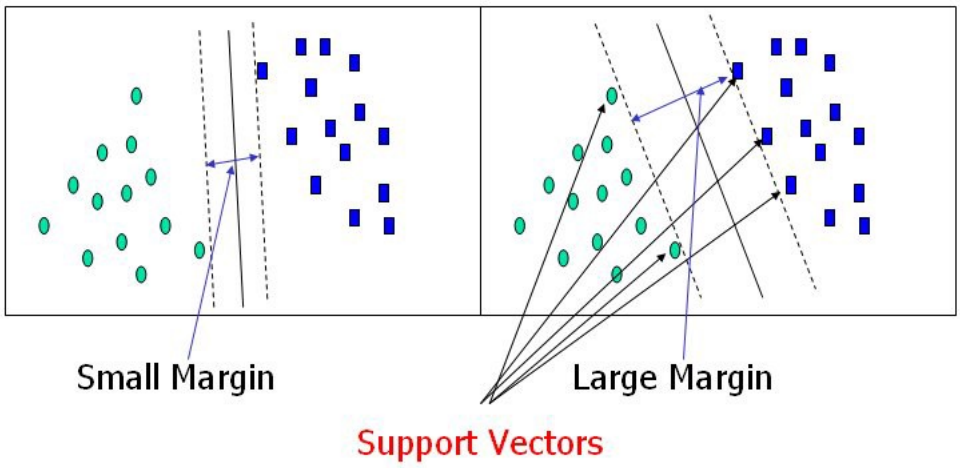

Fig. 8 Hyperplanes in 2D and 3D feature space

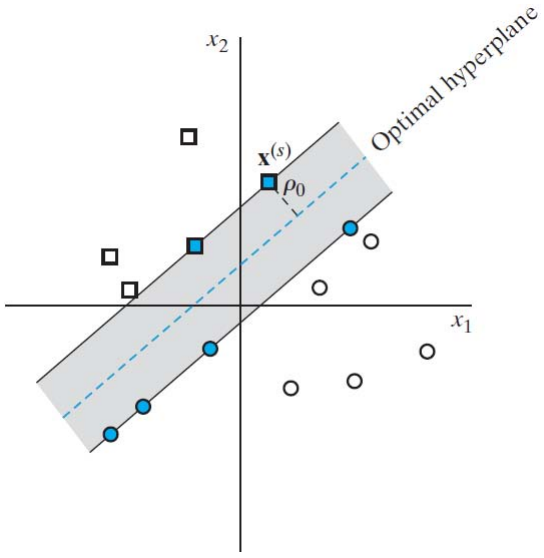

Fig. 9 Example of Optimal hyper planes

$2 * 5 + 3 * 6$ 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.

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.

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

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.

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.

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.

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.

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

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.

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

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.

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

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

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

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.

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.

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.

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 $f_{t-1}(x_n) = \sum_{r-1}^{t-1} \alpha_r h_r(x_n)$.

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.

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.