Module 4 (Data Science & ML)

Neural network learning: Artificial neurons, Activation functions, Network topology, Training neural networks with backpropogation.

Support Vector machines: Hyperplanes, classification using hyperplanes, Maximum margin hyperplanes in linearly separable data, Using kernels for non-linear spaces.

UNIT - IV

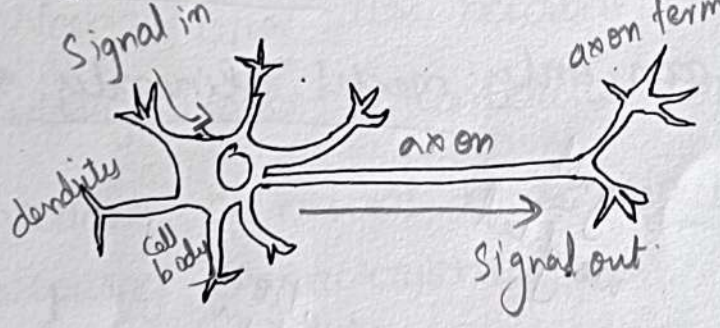
Neural Networks - Biological Motivation - perceptron - Activation functions - Networks models - cost Function - Back propagation algorithm - Introduction to deep learning.

Neural Networks

→ An Artificial Neural N/w (ANN) models the relationship between a set of input signal and output signal using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs. Just as a brain uses a network of interconnected cells called neurons to create a massive parallel processor, ANN uses a network of artificial neurons or nodes to solve learning problems

Biological Motivation

→ ANN were intentionally designed as conceptual models of human brain activity, it is helpful to first understand how biological neurons function



→ incoming signals are received by the cells dendrites through a biochemical process. The process allows the impulse to be weighted according to its relative importance or frequency.

→ As the cell body begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the axon. At the axon's terminals, the electric signal is again processed as a chemical signal to be passed to the neighboring neurons across a tiny gap known as a synapse.

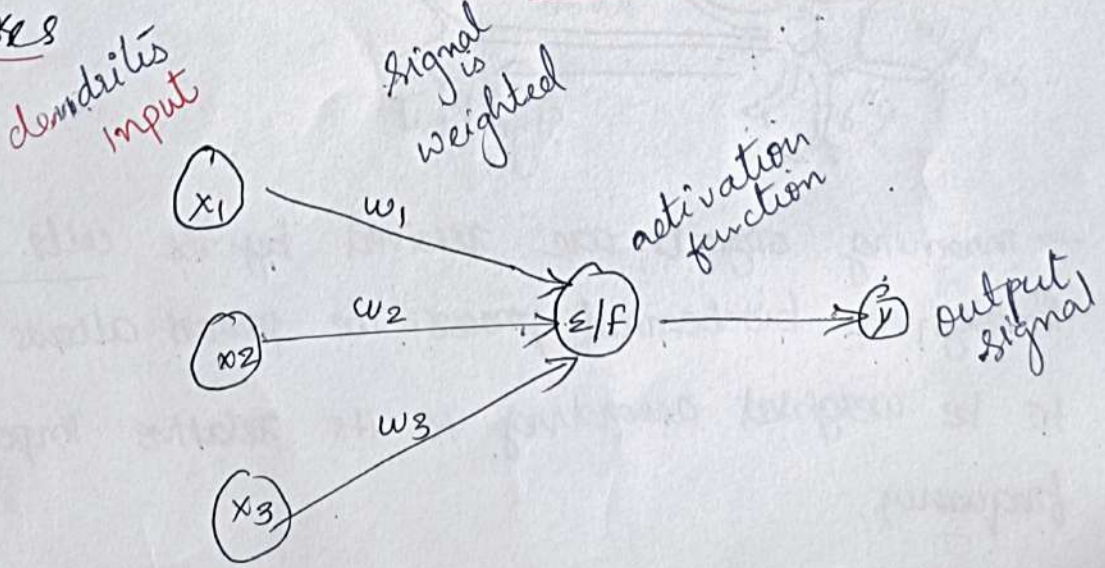
perceptron

→ perceptron can be defined as a single artificial neuron that computes its weighted input with the help of the threshold activation function or step function.

→ In machine learning, the perceptron is an algorithm for supervised classification of an input into one of several possible non-binary outputs.

→ the perceptron is used for binary classification.

→ the perceptron can only model linearly separable classes



→ a directed network diagram defines a relationship between the input signals received by the dendrites (x variables), and the output signal (y variable). Just as with the biological neuron each dendrite's signal is weighted (w values) according to its importance.

→ The input signals are summed by the cell body and the signal is passed on according to an activation function denoted by f.

→ the w weights allow each of the n inputs (denoted by x_i) to contribute a greater or lesser amount to the sum of input signals. the net total is used by the activation function f(x) and the resulting signal, y(x) is the output axon.

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

→ Neural n/ws use neurons defined this way as building blocks to construct complex models of data.

→ variants of neural n/ws can be defined in terms of the following characteristics:

* An activation function, which transforms a neuron's combined input signals into a single output signal to be broadcast further in the network.
* A n/w topology (or architecture), which describes the no. of neurons in the model as well as the no. of layers and manner in which they are connected.

* the training algorithm that specifies how weights are set in order to inhibit or excite, in proportion to the input signal.

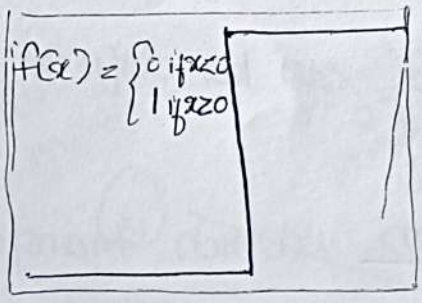
Activation functions

→ the activation function is the mechanism by which the artificial neuron processes incoming information and passes it throughout the n/w.

→ In the biological case, the activation fn. could be imagined as a process that involves summing the total input signal & determining whether it meets the firing threshold. If so, the neuron passes on the signal; otherwise it does nothing.

→ In ANN terms this is known as a threshold activation fn, as it results in an output signal only once a specified i/p threshold has been attained.

→ the following fig. depicts typical threshold fn. unit step activation fn.

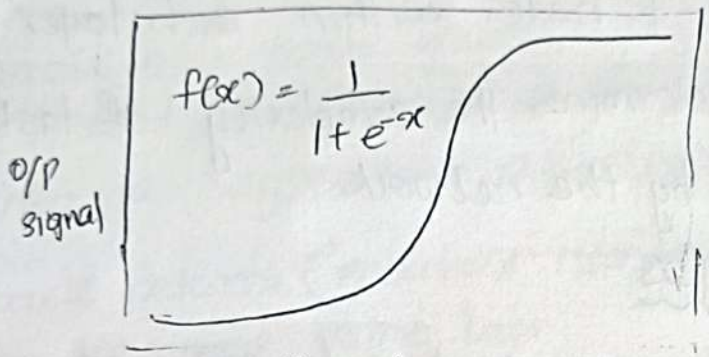


→ the ANN activation fn. can be chosen based on their ability to demonstrate desirable mathematical characteristics and accurate model relationships among data.

→ most commonly used alternative is the sigmoid activation fn.

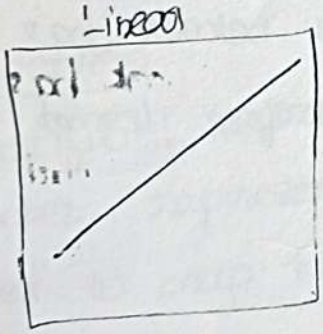
→ It shows a similar step or 'S' shape with the threshold activation fn, the output signal is no longer binary; output values can fall anywhere in the range from 0 to 1.

→ the sigmoid is differentiable, which means that it is possible to calculate the derivative across the entire range of i/p.

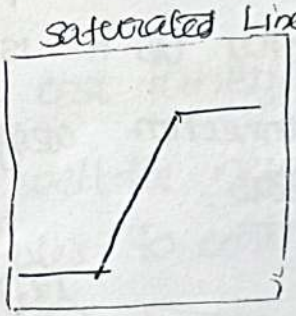


→ choice of alternative activation fn.

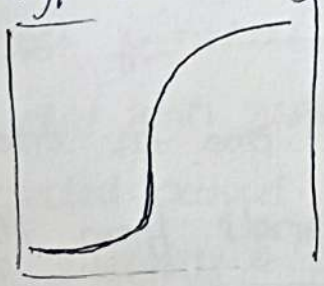
Linear



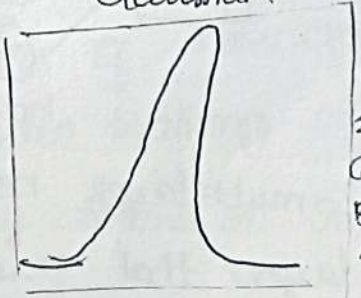
Saturated Linear



Hyperbolic Tangent



Gaussian.



• Gaussian AF results in a model called a Radial Basis Function (RBF) n/w

activation fns like the sigmoid are sometimes called squashing fn

Network topology

→ the ability of a neural n/w to learn is related to its topology, or the patterns and structures of interconnections.

→ n/w architecture can be differentiated by 3 key characteristics.

* The no. of layers
* whether information is allowed to travel backward
* the number of nodes within each layer of the network

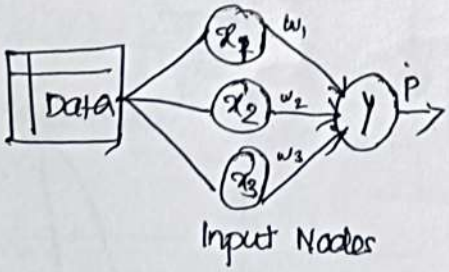
→ topology determines the complexity of tasks that can be learned by the network

The no of layers

→ the input and output nodes are arranged in groups known as layers

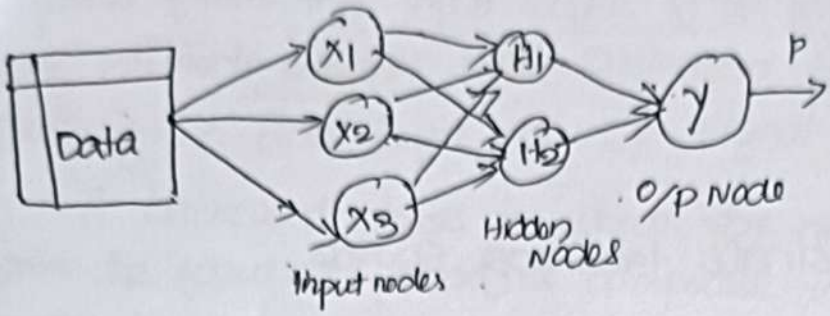
→ Because the input nodes process the incoming data exactly as it is received, the network has only one set of connection weights. It is therefore termed a single-layer network.

→



→ multilayer network adds one or more hidden layers that process the signals from the input nodes prior to it reaching the output node.

most multilayer networks are fully connected



the direction of information travel

→ N/w in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer are called feed forward n/ws

→ A neural n/w with multiple hidden layers is called a Deep Neural N/w (DNN) and the practice of training such n/w is referred to as deep learning.

→ feedback networks (recurrent n/w) allows signals to travel in both directions using loops

→ multilayer feedforward network, sometimes called multi-layer Perceptron (MLP)

cost function

→ In artificial neural n/w, the cost function is a number representing how well the neural network performed to map training examples to correct output.

→ less cost represent good model

→ A cost function is a measure of how good a neural network did with respect to it's given training sample and the expected output. It also may depend on variables such as weights & biases.

→ A cost function is of the form

$$C(W, B, S^n, E^n)$$

W → weights

B → biases

S^n → input of single training sample

E^n → desired o/p of that training sample

Training neural networks with backpropagation.

→ The algorithm which used a strategy of back-propagating errors is now known 'backpropagation'.

Strengths	Weakness
• Can be adapted to classification or numeric prediction problems • Capable of modeling more complex patterns than nearly any algorithm • Makes few assumptions about the data's underlying relationships.	• Extremely computationally intensive and slow to train, particularly if the n/w topology is complex. • Very prone to overfitting training data • Results in a complex black box model that is difficult, if not impossible to interpret

→ The backpropagation algorithm iterates through many cycles of two processes. Each cycle is known as an epoch. Because the network contains no a priori knowledge, the starting weights are typically set at random. Then, the algorithm iterates through the processes, until a stopping criterion is reached.

→ Each epoch in the backpropagation algorithm includes.

* A forward phase in which the neurons are activated in sequence from the input layer to the output layer, applying each neuron's weights and activation fn. along the way. Upon reaching the final layer, an o/p signal is produced.

* A backward phase in which the n/w's o/p signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's o/p signal and the true value results in an error that is propagated backwards in the n/w to modify the connection weights b/w neurons and reduce future errors.

→ the backpropagation algorithm uses the derivative of each neuron's activation fn. to identify the gradient in the direction of each of the incoming weights - hence the importance of having a differentiable activation fun.

→ The algorithm will attempt to change the weights that result in greatest reduction in error by an amount known as the learning rate.

Introduction to deep learning

→ Deep learning is a more appropriate name for an ANN.

→ The "deep" in deep learning refers to the depth of the n/w.

→ An ANN can be very shallow.

→ deep learning utilizes multiple layers of abstraction to solve pattern recognition problems.

→ Applications
- identifying faces in photographs
- speech recognition

→ probabilistic graphical model is a statistical model that uses graph to express dependence b/w probabilistic events.

UNIT-5

Support Vector Machines

-> SVM can be imagined as a surface that creates a boundary between points of data plotted in multidimensional that represent examples & their feature values.

-> the goal of a SVM is to create a flat boundary called a hyperplane, which divides the space to create fairly homogeneneous partitions on either side.

->SVM can be adapted for use with nearly any type of learning task, including both classification & numeric prediction

->applications include

* classification of microarray gene expression data in the field of bioinformatics to identify cancer or other genetic diseases.

* Text categorization such as identification of the language used in a document or the classification of documents by subject matter.

* the detection of rare yet important events like combustion engine failure security breaches or earthquakes.

Classification with hyperplanes

-> SVM use a boundary called a hyperplane to partition data into groups of similar class values.

Eg: hyperplane that separates groups of circles & squares in two and three dimensions. Because the circles and squares can be separated perfectly by the straight line or flat surface, they are said to be linearly separable.

UNIT - V
SVM

ML

Notice that the reported error (measured again by SSE) has been reduced from 5.08 in the previous model to 1.63 here. Additionally, the number of training steps rose from 4,882 to 86,849, which should come as no surprise given how much more complex the model has become. More complex networks take many more iterations to find the optimal weights.

Applying the same steps to compare the predicted values to the true values, we now obtain a correlation around 0.92, which is a considerable improvement over the previous result of 0.80 with a single hidden node:

```
> model_results2 <- compute(concrete_model2, concrete_test[1:8])
> predicted_strength2 <- model_results2$net.result
> cor(predicted_strength2, concrete_test$strength)
             [,1]
[1,] 0.9244533426
```

Interestingly, in the original publication, Yeh reported a mean correlation of 0.885 using a very similar neural network. This means that with relatively little effort, we were able to match the performance of a subject-matter expert. If you'd like more practice with neural networks, you might try applying the principles learned earlier in this chapter to see how it impacts model performance. Perhaps try using different numbers of hidden nodes, applying different activation functions, and so on. The `?neuralnet` help page provides more information on the various parameters that can be adjusted.

Understanding Support Vector Machines

A **Support Vector Machine** (SVM) can be imagined as a surface that creates a boundary between points of data plotted in multidimensional that represent examples and their feature values. The goal of a SVM is to create a flat boundary called a **hyperplane**, which divides the space to create fairly homogeneous partitions on either side. In this way, the SVM learning combines aspects of both the instance-based nearest neighbor learning presented in *Chapter 3, Lazy Learning – Classification Using Nearest Neighbors*, and the linear regression modeling described in *Chapter 6, Forecasting Numeric Data – Regression Methods*. The combination is extremely powerful, allowing SVMs to model highly complex relationships.

Hyperplanes.
Classification using hyperplanes
Maximum margin hyperplanes in linearly separable data.
Using kernels for non linear

Although the basic mathematics that drive SVMs have been around for decades, they have recently exploded in popularity. This is, of course, rooted in their state-of-the-art performance, but perhaps also due to the fact that award winning SVM algorithms have been implemented in several popular and well-supported libraries across many programming languages, including R. SVMs have thus been adopted by a much wider audience, might have otherwise been unable to apply the somewhat complex math needed to implement a SVM. The good news is that although the math may be difficult, the basic concepts are understandable.

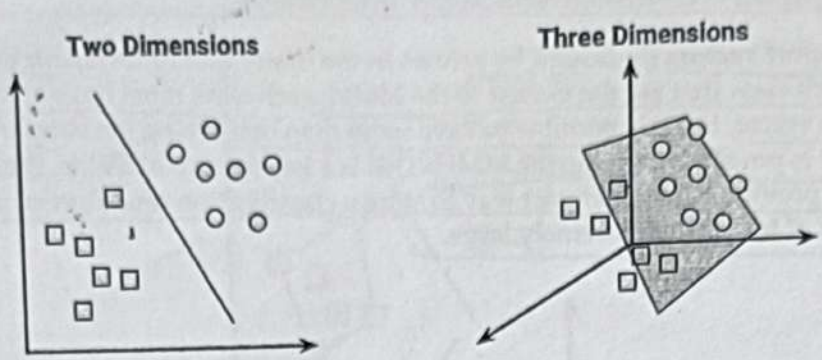
SVMs can be adapted for use with nearly any type of learning task, including both classification and numeric prediction. Many of the algorithm's key successes have come in pattern recognition. Notable applications include:

- Classification of microarray gene expression data in the field of bioinformatics to identify cancer or other genetic diseases
- Text categorization such as identification of the language used in a document or the classification of documents by subject matter
- The detection of rare yet important events like combustion engine failure, security breaches, or earthquakes

SVMs are most easily understood when used for binary classification, which is how the method has been traditionally applied. Therefore, in the remaining sections, we will focus only on SVM classifiers. Don't worry, however, as the same principles you learn here will apply while adapting SVMs to other learning tasks such as numeric prediction.

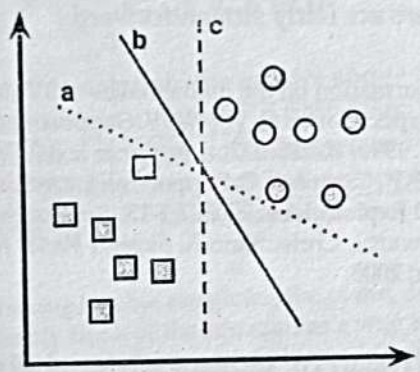
Classification with hyperplanes

As noted previously, SVMs use a boundary called a hyperplane to partition data into groups of similar class values. For example, the following figure depicts hyperplanes that separate groups of circles and squares in two and three dimensions. Because the circles and squares can be separated perfectly by the straight line or flat surface, they are said to be **linearly separable**. At first, we'll consider only the simple case where this is true, but SVMs can also be extended to problems where the points are not linearly separable.



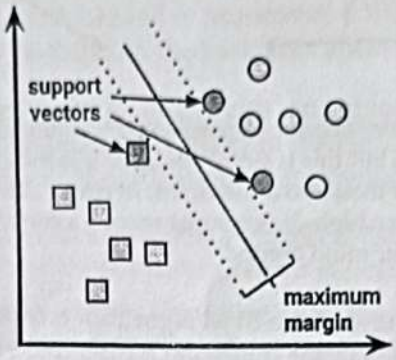
For convenience, the hyperplane is traditionally depicted as a line in 2D space, but this is simply because it is difficult to illustrate space in greater than two dimensions. In reality, the hyperplane is a flat surface in a high-dimensional space—a concept that can be difficult to get your mind around.

In two dimensions, the task of the SVM algorithm is to identify a line that separates the two classes. As shown in the following figure, there is more than one choice of dividing line between the groups of circles and squares. Three such possibilities are labeled a, b, and c. How does the algorithm choose?



The answer to that question involves a search for the **Maximum Margin Hyperplane** (MMH) that creates the greatest separation between the two classes. Although any of the three lines separating the circles and squares would correctly classify all the data points, it is likely that the line that leads to the greatest separation will generalize the best to the future data. The maximum margin will improve the chance that, in spite of random noise, the points will remain on the correct side of the boundary.

The **support vectors** (indicated by arrows in the figure that follows) are the points from each class that are the closest to the MMH; each class must have at least one support vector, but it is possible to have more than one. Using the support vectors alone, it is possible to define the MMH. This is a key feature of SVMs; the support vectors provide a very compact way to store a classification model, even if the number of features is extremely large.

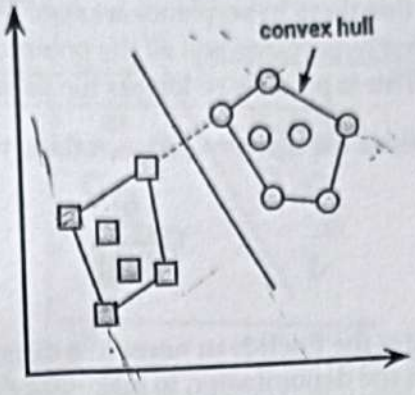


The algorithm to identify the support vectors relies on vector geometry and involves some fairly tricky math that is outside the scope of this book. However, the basic principles of the process are fairly straightforward.

More information on the mathematics of SVMs can be found in the classic paper: Cortes C, Vapnik V. Support-vector network. *Machine Learning*. 1995; 20:273-297. A beginner level discussion can be found in: Bennett KP, Campbell C. Support vector machines: hype or hallelujah. SIGKDD Explorations. 2003; 2:1-13. A more in-depth look can be found in: Steinwart I, Christmann A. *Support Vector Machines*. New York: Springer; 2008.

The case of linearly separable data

It is easiest to understand how to find the maximum margin under the assumption that the classes are linearly separable. In this case, the MMH is as far away as possible from the outer boundaries of the two groups of data points. These outer boundaries are known as the **convex hull**. The MMH is then the perpendicular bisector of the shortest line between the two convex hulls. Sophisticated computer algorithms that use a technique known as **quadratic optimization** are capable of finding the maximum margin in this way.



An alternative (but equivalent) approach involves a search through the space of every possible hyperplane in order to find a set of two parallel planes that divide the points into homogeneous groups yet themselves are as far apart as possible. To use a metaphor, one can imagine this process as similar to trying to find the thickest mattress that can fit up a stairwell to your bedroom.

To understand this search process, we'll need to define exactly what we mean by a hyperplane. In *n*-dimensional space, the following equation is used:

$$\vec{w} \cdot \vec{x} + b = 0$$

If you aren't familiar with this notation, the arrows above the letters indicate that they are vectors rather than single numbers. In particular, w is a vector of n weights, that is, $\{w_1, w_2, \ldots, w_n\}$, and b is a single number known as the **bias**. The bias is conceptually equivalent to the intercept term in the slope-intercept form discussed in *Chapter 6, Forecasting Numeric Data – Regression Methods*.

If you're having trouble imagining the plane, don't worry about the details. Simply think of the equation as a way to specify a surface, much like when the slope-intercept form ($y = mx + b$) is used to specify lines in 2D space.

Using this formula, the goal of the process is to find a set of weights that specify two hyperplanes, as follows:

$$\vec{w} \cdot \vec{x} + b \geq +1$$
$$\vec{w} \cdot \vec{x} + b \leq -1$$

We will also require that these hyperplanes are specified such that all the points of one class fall above the first hyperplane and all the points of the other class fall beneath the second hyperplane. This is possible so long as the data are linearly separable.

Vector geometry defines the distance between these two planes as:

$$\frac{2}{||\vec{w}||}$$

Here, $||w||$ indicates the **Euclidean norm** (the distance from the origin to vector w). Because $||w||$ is in the denominator, to maximize distance, we need to minimize $||w||$. The task is typically reexpressed as a set of constraints, as follows:

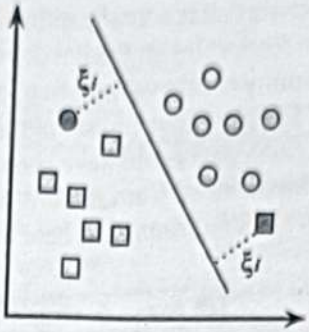
$$\min \frac{1}{2} \|\vec{w}\|^2$$
$$s.t.\ \ y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall \vec{x}_i$$

Although this looks messy, it's really not too complicated to understand conceptually. Basically, the first line implies that we need to minimize the Euclidean norm (squared and divided by two to make the calculation easier). The second line notes that this is subject to (*s.t.*), the condition that each of the y_i data points is correctly classified. Note that y indicates the class value (transformed to either +1 or -1) and the upside down "A" is shorthand for "for all."

As with the other method for finding the maximum margin, finding a solution to this problem is a task best left for quadratic optimization software. Although it can be processor-intensive, specialized algorithms are capable of solving these problems quickly even on fairly large datasets.

The case of nonlinearly separable data

As we've worked through the theory behind SVMs, you may be wondering about the elephant in the room: what happens if the data are not linearly separable? The solution to this problem is the use of a **slack variable**, which creates a soft margin that allows some points to fall on the incorrect side of the margin. The figure that follows illustrates two points falling on the wrong side of the line with the corresponding slack terms (denoted with the Greek letter Xi):



A cost value (denoted as C) is applied to all points that violate the constraints, and rather than finding the maximum margin, the algorithm attempts to minimize the total cost. We can therefore revise the optimization problem to:

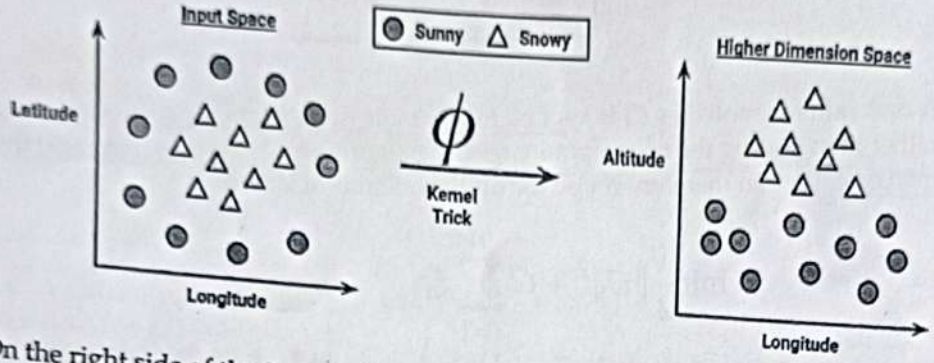
$$\min \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$s.t.\ \ y_i(\vec{w}\cdot\vec{x}_i - b) \geq 1 - \xi_i, \forall\vec{x}_i, \xi_i \geq 0$$

If you're still confused, don't worry, you're not alone. Luckily, SVM packages will happily optimize this for you without you having to understand the technical details. The important piece to understand is the addition of the cost parameter C. Modifying this value will adjust the penalty, for example, the fall on the wrong side of the hyperplane. The greater the cost parameter, the harder the optimization will try to achieve 100 percent separation. On the other hand, a lower cost parameter will place the emphasis on a wider overall margin. It is important to strike a balance between these two in order to create a model that generalizes well to future data.

Using kernels for non-linear spaces

In many real-world applications, the relationships between variables are nonlinear. As we just discovered, a SVM can still be trained on such data through the addition of a slack variable, which allows some examples to be misclassified. However, this is not the only way to approach the problem of nonlinearity. A key feature of SVMs is their ability to map the problem into a higher dimension space using a process known as the **kernel trick**. In doing so, a nonlinear relationship may suddenly appear to be quite linear.

Though this seems like nonsense, it is actually quite easy to illustrate by example. In the following figure, the scatterplot on the left depicts a nonlinear relationship between a weather class (sunny or snowy) and two features: latitude and longitude. The points at the center of the plot are members of the snowy class, while the points at the margins are all sunny. Such data could have been generated from a set of weather reports, some of which were obtained from stations near the top of a mountain, while others were obtained from stations around the base of the mountain.



On the right side of the figure, after the kernel trick has been applied, we look at the data through the lens of a new dimension: altitude. With the addition of this feature, the classes are now perfectly linearly separable. This is possible because we have obtained a new perspective on the data. In the left figure, we are viewing the mountain from a bird's eye view, while in the right one, we are viewing the mountain from a distance at the ground level. Here, the trend is obvious: snowy weather is found at higher altitudes.

SVMs with nonlinear kernels add additional dimensions to the data in order to create separation in this way. Essentially, the kernel trick involves a process of constructing new features that express mathematical relationships between measured characteristics. For instance, the altitude feature can be expressed mathematically as an interaction between latitude and longitude—the closer the point is to the center of each of these scales, the greater the altitude. This allows SVM to learn concepts that were not explicitly measured in the original data.

SVMs with nonlinear kernels are extremely powerful classifiers, although they do have some downsides, as shown in the following table:

Strengths	Weaknesses
• Can be used for classification or numeric prediction problems • Not overly influenced by noisy data and not very prone to overfitting • May be easier to use than neural networks, particularly due to the existence of several well-supported SVM algorithms • Gaining popularity due to its high accuracy and high-profile wins in data mining competitions	• Finding the best model requires testing of various combinations of kernels and model parameters • Can be slow to train, particularly if the input dataset has a large number of features or examples • Results in a complex black box model that is difficult, if not impossible, to interpret

Kernel functions, in general, are of the following form. The function denoted by the Greek letter phi, that is, $\varphi(x)$, is a mapping of the data into another space. Therefore, the general kernel function applies some transformation to the feature vectors x_i and x_j and combines them using the **dot product**, which takes two vectors and returns a single number.

$$K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j})$$

Using this form, kernel functions have been developed for many different domains of data. A few of the most commonly used kernel functions are listed as follows. Nearly all SVM software packages will include these kernels, among many others.

The linear kernel does not transform the data at all. Therefore, it can be expressed simply as the dot product of the features:

$$K(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$$

The **polynomial kernel** of degree d adds a simple nonlinear transformation of the data:

$$K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^d$$

The sigmoid kernel results in a SVM model somewhat analogous to a neural network using a sigmoid activation function. The Greek letters kappa and delta are used as kernel parameters:

$$K(\vec{x}_i, \vec{x}_j) = \tanh(\kappa \, \vec{x}_i \cdot \vec{x}_j - \delta)$$

radial basis fn.

The Gaussian RBF kernel is similar to a RBF neural network. The RBF kernel performs well on many types of data and is thought to be a reasonable starting point for many learning tasks:

$$K(\vec{x}_i, \vec{x}_j) = e^{\frac{-||\vec{x}_i - \vec{x}_j||^2}{2\sigma^2}}$$

There is no reliable rule to match a kernel to a particular learning task. The fit depends heavily on the concept to be learned as well as the amount of training data and the relationships among the features. Often, a bit of trial and error is required by training and evaluating several SVMs on a validation dataset. This said, in many cases, the choice of kernel is arbitrary, as the performance may vary slightly. To see how this works in practice, let's apply our understanding of SVM classification to a real-world problem.

Example – performing OCR with SVMs

Image processing is a difficult task for many types of machine learning algorithms. The relationships linking patterns of pixels to higher concepts are extremely complex and hard to define. For instance, it's easy for a human being to recognize a face, a cat, or the letter "A", but defining these patterns in strict rules is difficult. Furthermore, image data is often noisy. There can be many slight variations in how the image was captured, depending on the lighting, orientation, and positioning of the subject.

SVMs are well-suited to tackle the challenges of image data. Capable of learning complex patterns without being overly sensitive to noise, they are able to recognize visual patterns with a high degree of accuracy. Moreover, the key weakness of SVMs—the black box model representation—is less critical for image processing. If an SVM can differentiate a cat from a dog, it does not matter much how it is doing so.

• In ML, the (Gaussian) radial fn. kernel, or RBF kernel, is a popular kernel fn. used in various kernelized learning algorithms. It is commonly used in SVM classification.

• The RBF kernel on two samples x & x', represented as feature vectors in some input space, is defined as,

$$k(x, x') = \exp\left(\frac{\|x - x'\|^2}{2\sigma^2}\right), \quad \|x - x'\|^2 \text{ may be recognised}$$

as the squared Euclidean distance b/w two feature vectors. σ is a free parameter. Since the value of the RBF kernel decreases with distance and ranges between 0 & 1 it has a readily interpretation