Introduction

Saturday, November 26, 2016 18:14

- An Artificial Neural Network (ANN) models the relationship between a set of input signals and an 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.

ANNs are versatile learners that can be applied to nearly any learning task: classification, numeric prediction, and even unsupervised pattern recognition.

- Speech and handwriting recognition programs
- The automation of smart devices like an office building's environmental controls or self-driving cars and self-piloting drones

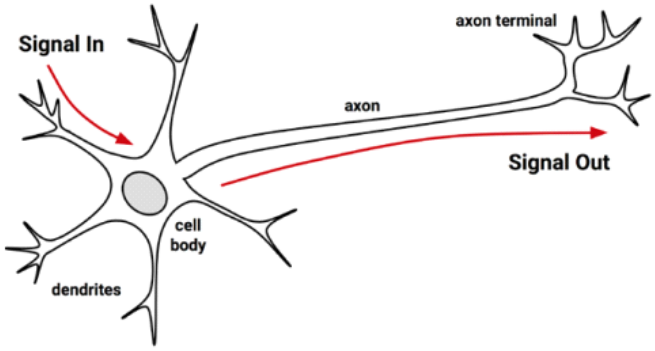
ANNs are best applied to problems where the input data and output data are well-defined or at least fairly simple, yet the process that relates the input to output is extremely complex.

From biological to artificial neurons

Tuesday, November 29, 2016 00:37

How does biological neurons function?

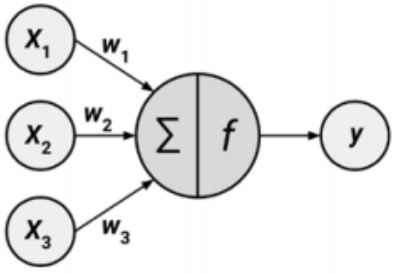
- Incoming signals are received by the cell's dendrites through a biochemical process. The process allows the impulse to be weighted according to its relative importance or frequency.



- 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.

The model of a single artificial neuron

- The directed network diagram below defines a relationship between the input signals received by the dendrites (x variables), and the output signal (y variable).



- 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.**
- A typical artificial neuron with n input dendrites can be represented by the formula

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

The w weights allow each of the n inputs (denoted by xi) 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:

Neural networks use neurons defined this way as building blocks to construct complex models of data.

Although there are numerous variants of neural networks, each 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 broadcasted further in the network
- A **network topology or architecture**, which describes the number of neurons in the model as well as the number of layers and manner in which they are connected
- The **training algorithm** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal

Characteristic features of ANNs

Wednesday, November 30, 2016 16:43

1. Activation functions

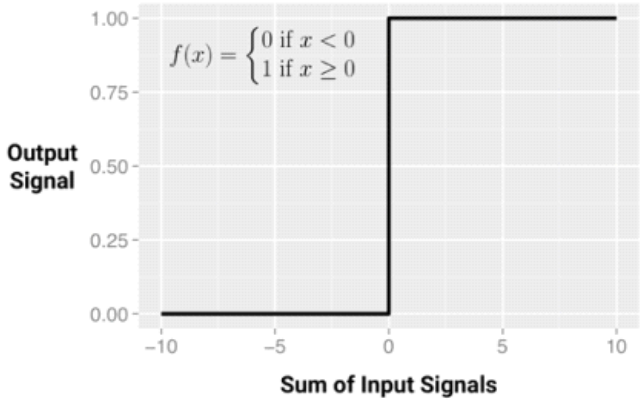
The activation function is the mechanism by which the artificial neuron processes incoming information and passes it throughout the network.

Kinds of Activation functions

a. Threshold Activation function

Activation function modeled after biological process that involves summing the total input signal and passing on the signal if it meets the firing threshold. In ANN, it is known as a threshold activation function, as it results in an output signal only once a specified input threshold has been attained.

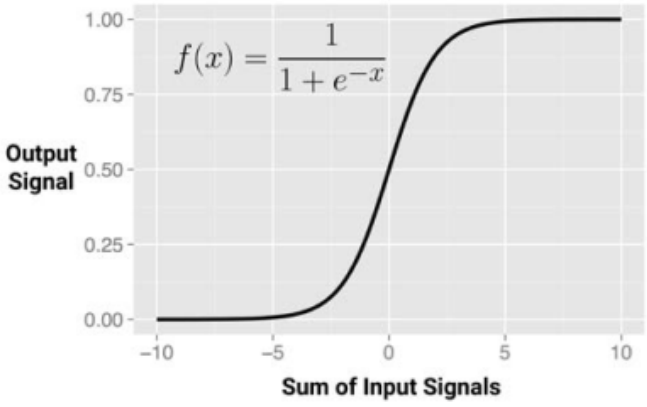
Example: Unit step activation function. Here, the neuron fires when the sum of input signals is at least zero.



Threshold activation function is rarely used in artificial neural networks.

The ANN activation functions can be chosen based on their ability to demonstrate desirable mathematical characteristics and accurately model relationships among data.

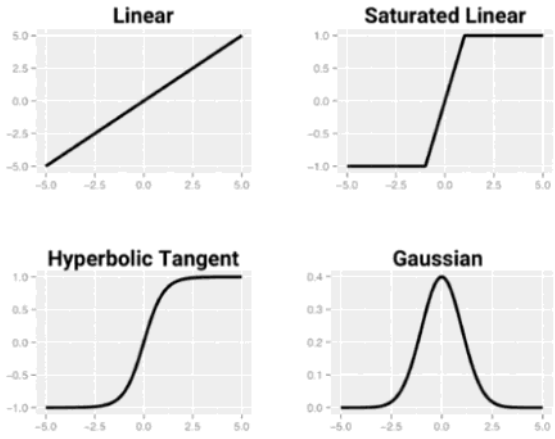
b. Sigmoid Activation function



- The output values can fall anywhere in the range from 0 to 1.
- Additionally, the sigmoid is differentiable, which is crucial to create efficient ANN optimization algorithms.

Although sigmoid is perhaps the most commonly used activation function, some neural network algorithms allow a choice of alternatives.

c. Other Activation functions



Primary details about activation functions:

- **How to choose the Activation function? See the Output signal range**
  - Typically, it is one of (0, 1), (-1, +1), or (-inf, +inf).
  - The choice of activation function biases the neural network such that it may fit certain types of data more appropriately, allowing the construction of specialized neural networks.
    - For instance, a linear activation function results in a neural network very similar to a linear regression model.

- **If the range of input values that affect the output signal is narrow?**
  - For example, in the case of sigmoid, the output signal is always nearly 0 or 1 for an input signal below -5 or above +5, respectively.
  - **Solution: Standardize/Normalize all features**
    - By restricting the range of input values, the activation function will have action across the entire range, preventing large valued features such as household income from dominating small-valued features such as the number of children in the household.
    - A side benefit is that the model may also be faster to train.

2. Network topology or Architecture

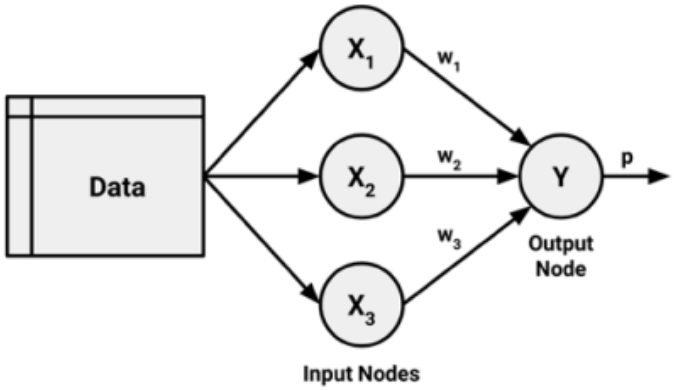
Although there are countless forms of network architecture, they can be differentiated by three key characteristics:

- The number of layers
- Whether information in the network is allowed to travel backward
- The number of nodes within each layer of the network

How does a single ANN work?

- Input nodes receives unprocessed signals directly from the input data. Each input node is responsible for processing a single feature in the dataset
- The feature's value will be transformed by the corresponding node's activation function. The signals sent by the input nodes are received by the output node, which uses its own activation function to generate a final prediction (denoted here as p).

The input and output nodes are arranged in groups known as layers.



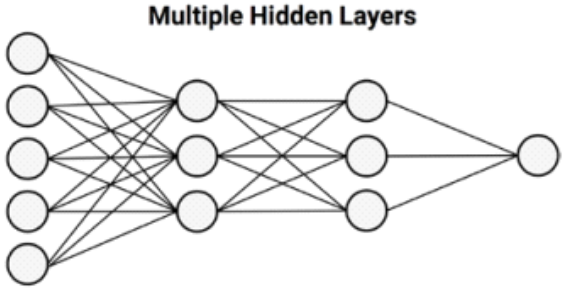
The topology as well as the arrangement determines the complexity of tasks that can be learned by the network.

a. The number of Layers

- **Single-layer network**
  - because the **input nodes process the incoming data exactly as it is received**, the network has only one set of connection weights (labeled here as w1, w2, and w3).
  - These can be used for basic pattern classification, particularly for patterns that are linearly separable.

- **Create more complex networks is by adding additional layers**
  - A 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**, which means that every node in one layer is connected to every node in the next layer, but this is not required.

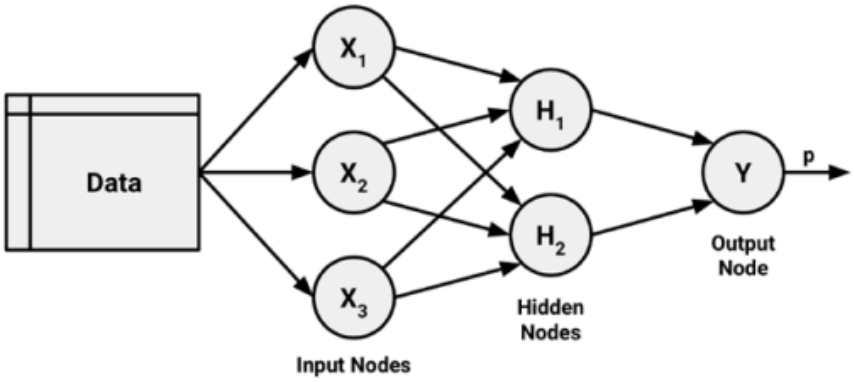
- **Multi-layer network/DNN**
  - A neural network with multiple hidden layers is called a Deep Neural Network (DNN) and the practice of training such network is sometimes referred to as deep learning.



b. The number of nodes in each layer

Neural networks can also vary in complexity by the number of nodes in each layer.

- **The number of input nodes** is predetermined by the number of features in the input data.
- **The number of output nodes** is predetermined by the number of outcomes to be modeled or the number of class levels in the outcome.
- **The number of hidden nodes**



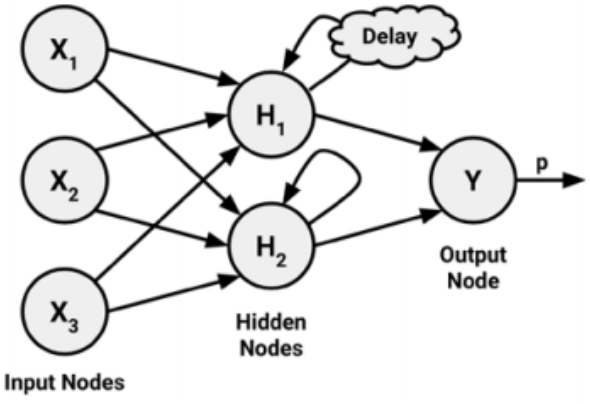
- **The appropriate number** depends on the number of input nodes, the amount of training data, the amount of noisy data, and the complexity of the learning task, among many other factors.
- **A greater number of neurons** will result in a model, capable of learning more complex problems but may closely mirror the training data. Overfitting i.e. it may generalize poorly to future data. Large neural networks can also be computationally expensive and slow to train.
- **The best practice** is to use the fewest nodes that result in adequate performance in a validation dataset.

b. The direction of information travel

Arrowheads are used to indicate the direction of signals.

- **Feedforward networks**
  - Networks in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer.
  - They are quite flexible. For instance, the number of nodes at each level can be varied, multiple outcomes can be modeled simultaneously, or multiple hidden layers can be applied.
  - The multilayer feedforward network is sometimes called the Multilayer Perceptron (MLP),

- **Feedback network or Recurrent network** allows signals to travel in both directions using loops.
  - This property closely mirrors how a biological neural network works and allows extremely complex patterns to be learned. The addition of a short-term memory, or delay, increases the power of recurrent networks immensely.



  - Notably, this includes the capability to understand the sequences of events over a period of time. This could be used for stock market prediction, speech comprehension, or weather forecasting.
  - In spite of their potential, **recurrent networks are still largely theoretical** and are rarely used in

practice.

3. Training neural networks with backpropagation

Neural Networks are trained by adjusting connection weights to reflect the patterns observed over time. This process is very computationally intensive. **The backpropagation algorithm** uses a strategy of back-propagating errors to do this. Although **it is still notoriously slow** relative to many other machine learning algorithms.

Strengths	Weaknesses
• 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 network 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 (existing) 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 function along the way. Upon reaching the final layer, an output signal is produced.
- A **backward phase** in which the network's output signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's output signal and the true value results in an error that is propagated backwards in the network **to modify the connection weights between neurons and reduce future errors.**

Over time, the network uses the information sent backward to reduce the total error of the network.

Because the relationship between each neuron's inputs and outputs is complex, **How does the algorithm determine how much a weight should be changed?**

The answer to this question involves a technique called **gradient descent** (Finds the value of weights that would minimize the error)

- The backpropagation algorithm uses the derivative of each neuron's activation function to identify the gradient in the direction of each of the incoming weights (hence the importance of having a differentiable activation function.)
- The gradient (slope) suggests how steeply the error will be reduced or increased for a change in the weight.

- The algorithm will attempt to change the weights that result in the greatest reduction in error by an amount known as the learning rate (α).
- The greater the learning rate, the faster the algorithm will attempt to descend down the gradients, which could reduce the training time at the risk of overshooting the valley.

Introduction

Sunday, November 27, 2016 13:00

Support vector machine (SVM) is an approach for classification and are often considered one of the best "out of the box" classifiers.

- The support vector machine is a generalization of a simple and intuitive classifier called the maximal margin classifier. Though it is elegant and simple, we will see that this classifier unfortunately cannot be applied to most data sets, since it requires that the classes be separable by a linear boundary.
- Support vector classifier is an extension of the maximal margin classifier that can be applied in a broader range of cases.
- Support vector machine is an extension of the support vector classifier that can accommodate non-linear class boundaries.

Maximal margin classifier, the support vector classifier, and the support vector machine are loosely referred to as "support vector machines". We will carefully distinguish between these three notions in this chapter.

Maximal Margin Classifier

Sunday, November 27, 2016 16:35

In this section,

- we define a hyperplane and
- introduce the concept of an optimal separating hyperplane.

What Is a Hyperplane?

In a p-dimensional space, a hyperplane is a flat affine subspace of dimension p − 1. (The word affine indicates that the subspace need not pass through the origin.)
For instance,

- In two dimensions, a hyperplane is a flat one-dimensional subspace — a line.
- In three dimensions, a hyperplane is a flat two-dimensional subspace—that is, a plane.

The mathematical definition of a hyperplane

- In two dimensions, a hyperplane is defined by the equation of a line

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

- In p-dimensional setting:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

  defines a p-dimensional hyperplane, such that if a point X = (X1, X2, . . . , Xp)T in p-dimensional space (i.e. a vector of length p) satisfies the equation, then X lies on the hyperplane.

- Suppose that X does not satisfy the above equation rather,

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0.$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0,$$

  Then this tells us that X lies to one or the other side of the hyperplane.

So, A hyperplane is dividing p-dimensional space into two halves. One can easily determine on which side of the hyperplane a point lies by simply calculating the sign of the left hand side..

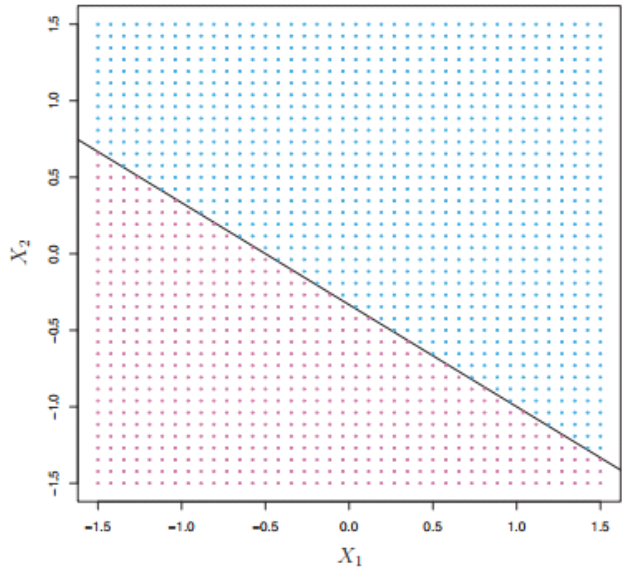


FIGURE 9.1. *The hyperplane* $1 + 2X_1 + 3X_2 = 0$ *is shown. The blue region is the set of points for which* $1 + 2X_1 + 3X_2 > 0$*, and the purple region is the set of points for which* $1 + 2X_1 + 3X_2 < 0$*.*

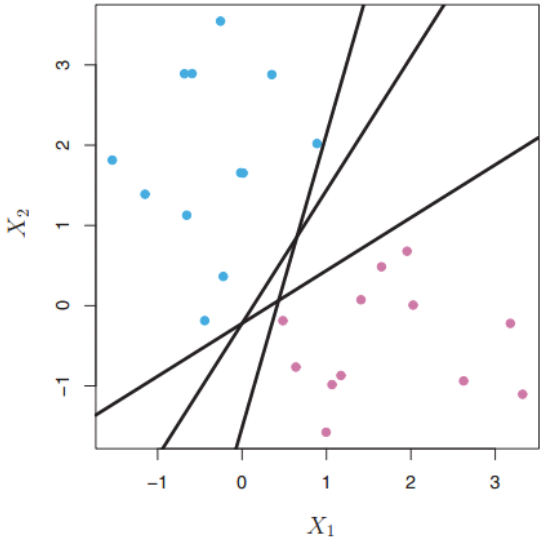
Classification Using a Separating Hyperplane

- **Data:** A n×p dimensional training data set with observations falling into two classes, purple and blue along with a test observation.
- **Goal:** to develop a classifier based on the training data that will correctly classify the test observation using its feature measurements <u>using the concept of a separating hyperplane.</u>

<u>Separating Hyperplane</u>

A hyperplane that separates the training observations perfectly according to their class labels.

- Suppose that it exists. Examples of three such separating hyperplanes:



<u>It will have the following property</u>

And if we label the observations from the blue class as yi = 1 and those from the purple class as yi = −1.

- Then a separating hyperplane has the property that

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0 \text{ if } y_i = 1,$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0 \text{ if } y_i = -1.$$

- Equivalently, the separating hyperplane has the property that

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0$$

If a separating hyperplane exists, we can use it to construct a very natural classifier.

Classification

- We classify the test observation x∗ to a class depending on which side of the hyperplane it is located i.e. based on the sign of f(x∗).

$$f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^*.$$

  - If f(x∗) is positive, then we assign the test observation to class 1, and if f(x∗) is negative, then we assign it to class -1.
- Using the **magnitude** of f(x∗), we can tell about the confidence in our assignment.
  - If f(x∗) is far from zero, then this means that x∗ lies far from the hyperplane, and so we can be **confident** about our class assignment for x∗.
  - If f(x∗) is close to zero, then x∗ is located near the hyperplane, and so we are less certain about the class assignment for x∗.

A classifier that is based on a separating hyperplane leads to a linear decision boundary.

The Maximal Margin Classifier

- If our data can be perfectly separated using a hyperplane, then there will in fact exist an **infinite number of such hyperplanes.** (because a given separating hyperplane can usually be shifted a tiny bit up or down, or rotated, without coming into contact with any of the observations.)
- In order to construct a classifier based upon a separating hyperplane, we must have a reasonable way to decide **which of the infinite possible separating hyperplanes to use.**

Which one to use?

Maximal margin hyperplane (also known as the optimal separating hyperplane), which is the separating hyperplane that is farthest from the training observations.

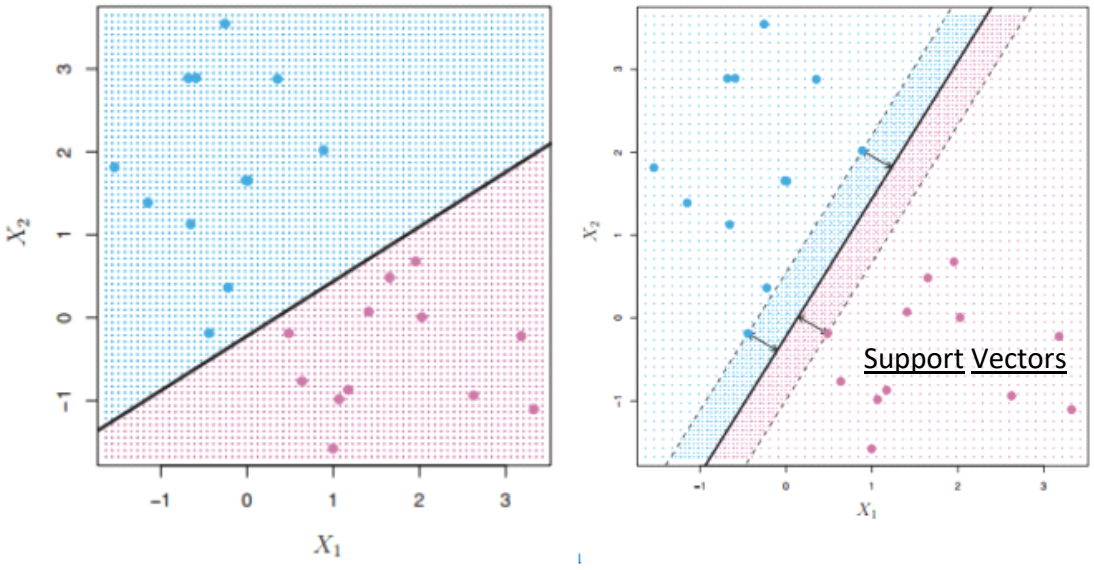
How?

- Compute the perpendicular distance (min. distance) from each training observation to a given separating hyperplane.
- The smallest such distance from the observations to the hyperplane is the **margin**.
- The maximal margin hyperplane is the **separating hyperplane that has the largest margin or the largest minimum distance to the training observations.**

Maximal margin classifier

- We can use it to classify a test observation based on which side of the maximal margin hyperplane it lies.
- If β0, β1, . . . , βp are the coefficients of the maximal margin hyperplane, then the maximal margin classifier classifies the test observation x∗ based on the sign of
  f(x∗) = β0 + β1x∗ 1 + β2x∗ 2 + . . . + βpx∗ p
- We hope that a classifier that has a large margin on the training data will also have a large margin on the test data, and hence will classify the test observations correctly.
- Although the maximal margin classifier is often successful, it **can also lead to overfitting when p is large**.

The RHS of the panel shows the maximal margin hyperplane on the data set of LHS of the panel. The figure on the right has a **larger margin.**



- In the RHS of the panel, we see that three training observations are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin.
- These three observations are known as **Support vectors**, since they are **vectors** in p-dimensional space (p = 2) and they "**support**" the maximal margin hyperplane in the sense that if these points were moved slightly then the maximal margin hyperplane would move as well.
- The maximal margin hyperplane depends directly on the support vectors, but not on the other observations: a movement to any of the other observations would not affect the separating hyperplane, provided that the observation's movement does not cause it to cross the boundary set by the margin.

Construction of the Maximal Margin Classifier

Based on a set of n training observations x1, . . . , xn ∈ Rp and associated class labels y1, . . . , yn ∈ {−1, 1}.

- Briefly, the maximal margin hyperplane is the solution to the optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M \tag{9.9}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.10}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \;\; \forall\, i = 1, \ldots, n. \tag{9.11}$$

1) **9.11:** For each observation to be on the correct side of the hyperplane we would simply need yi(β0 + β1xi1 + β2xi2+. . .+βpxip) > 0,
   so the constraint in (9.11) in fact requires that each observation be on the correct side of the hyperplane, with some **cushion**, provided that M is positive.

2) **9.10:**

- (9.10) is not really a constraint on the hyperplane, since

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} = 0$$

  defines a hyperplane, then so does

$$k(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) = 0 \text{ for any } k \neq 0.$$

- However, (9.10) adds meaning to (9.11): one can show that with this constraint the perpendicular distance from the ith observation to the hyperplane is given by
  yi(β0 + β1xi1 + β2xi2 + . . . + βpxip).

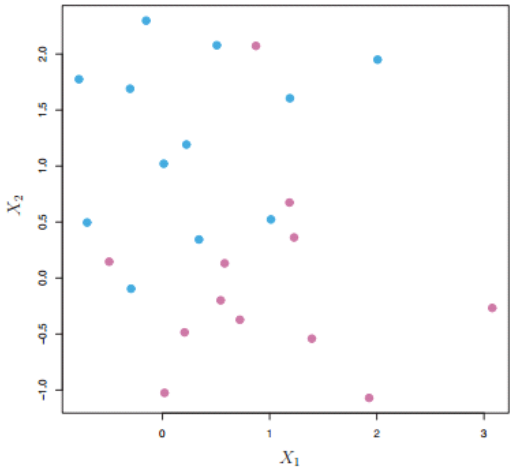
Therefore, the constraints (9.10) and (9.11) ensure that **each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane. Hence, M represents the margin of our hyperplane, and the optimization problem chooses β0, β1, . . ., βp to maximize M.**

The Non-separable Case

The maximal margin classifier is a very natural way to perform classification, if a separating hyperplane exists. However, in many cases no separating hyperplane exists, and so there is no maximal margin classifier. In this case, the optimization problem (9.9)–(9.11) has no solution with M > 0.

Example:

In this case, we cannot exactly separate the two classes.



- However, we can **extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called soft margin.**
- The generalization of the maximal margin classifier to the non-separable case is known as the **support vector classifier.**

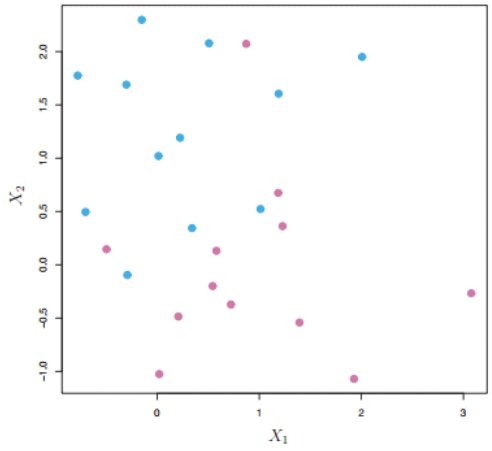
Support Vector Classifier

Sunday, November 27, 2016 16:38

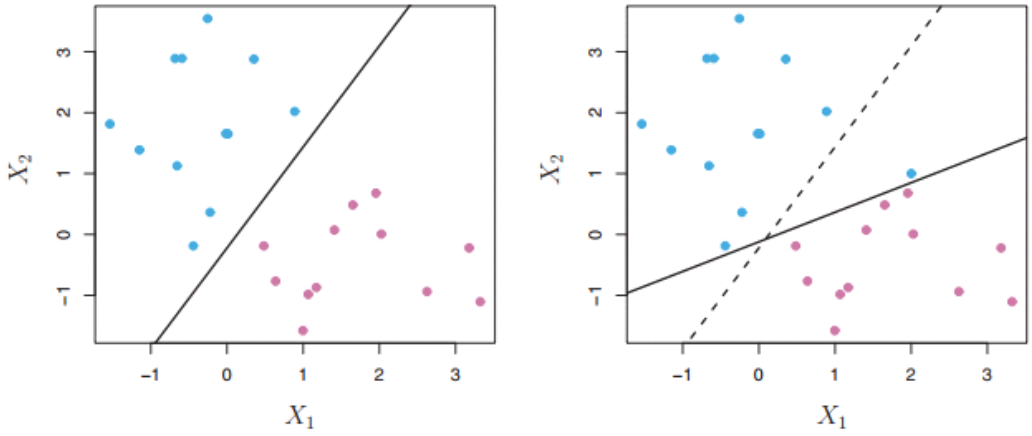
Overview

Maximal margin classifier cannot be safely used when

1. A Separating Classifier may not exist.



2. A classifier based on a separating hyperplane might not be desirable. Since classifier based on a separating hyperplane will necessarily perfectly classify all of the training observations; **this can lead to sensitivity to individual observations.**
   - The addition of a single observation in the right-hand panel leads to a dramatic change in the maximal margin hyperplane. The resulting maximal margin hyperplane is not satisfactory—for one thing, it has only a tiny margin.



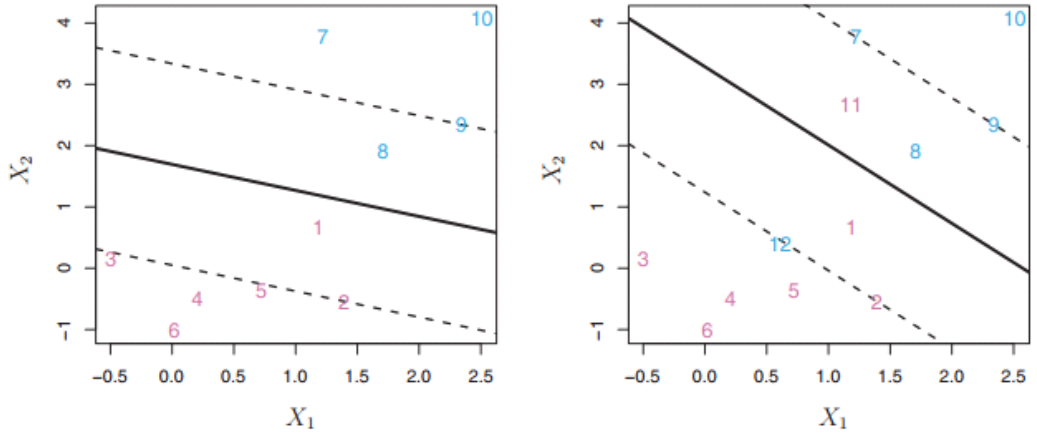
   - This is problematic because
     - the distance of an observation from the hyperplane can be seen as a measure of our **confidence** that the observation was correctly classified.
     - Moreover, the fact that the maximal margin hyperplane is extremely sensitive to a change in a single observation suggests that it may have **overfit** the training data.

Solution:

- A classifier based on a hyperplane that does not perfectly separate the two classes, in the interest of
  - Greater robustness to individual observations, and
  - Better classification of most of the training observations.

- **Soft margin classifier called Support Vector Classifier**
  Rather than seeking the largest possible margin so that every observation is not only on the correct side of the hyperplane but also on the correct side of the margin, we instead allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane.
  (The margin is soft because it can be violated by some of the training observations.)

  - **Example LHS:** Most of the observations are on the correct side of the margin. However, a small subset of the observations are on the wrong side of the margin.



  - **Example 2:** When there is no true separating hyperplane, an observation can be on the wrong side of the hyperplane. These correspond to training observations that are misclassified by the support vector classifier.

Details of the Support Vector Classifier

- The support vector classifier classifies a test observation depending on which side of a hyperplane it lies.
- The hyperplane is chosen to correctly separate most of the training observations into the two classes, but may misclassify a few observations.

It is the solution to the optimization problem:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \tag{9.12}$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.13}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \tag{9.14}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{9.15}$$

- C is a non-negative tuning parameter.
- M is the width of the margin; we seek to make this quantity as large as possible.
- $\epsilon_1, \ldots, \epsilon_n$

are **slack variables** that allow individual observations to be on the wrong side of the margin or the hyperplane.

Once we have solved (9.12)–(9.15), we **classify** a test observation x∗, by determining on which side of the hyperplane it lies. That is, we classify the test observation based on the sign of

f(x∗) = β0 + β1x∗ 1 + . . . + βpx∗ p

Elements of the optimization problem

1. **The slack variable ϵ_i** tells us where the ith observation is located, relative to the hyperplane and relative to the margin.

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

   - If ϵ_i = 0 then the ith observation is on the correct side of the margin.
   - If ϵ_i > 0 then the ith observation is on the wrong side of the margin, and we say that the ith observation has violated the margin.
   - If ϵ_i > 1 then it is on the wrong side of the hyperplane.

2. **Tuning parameter C**

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

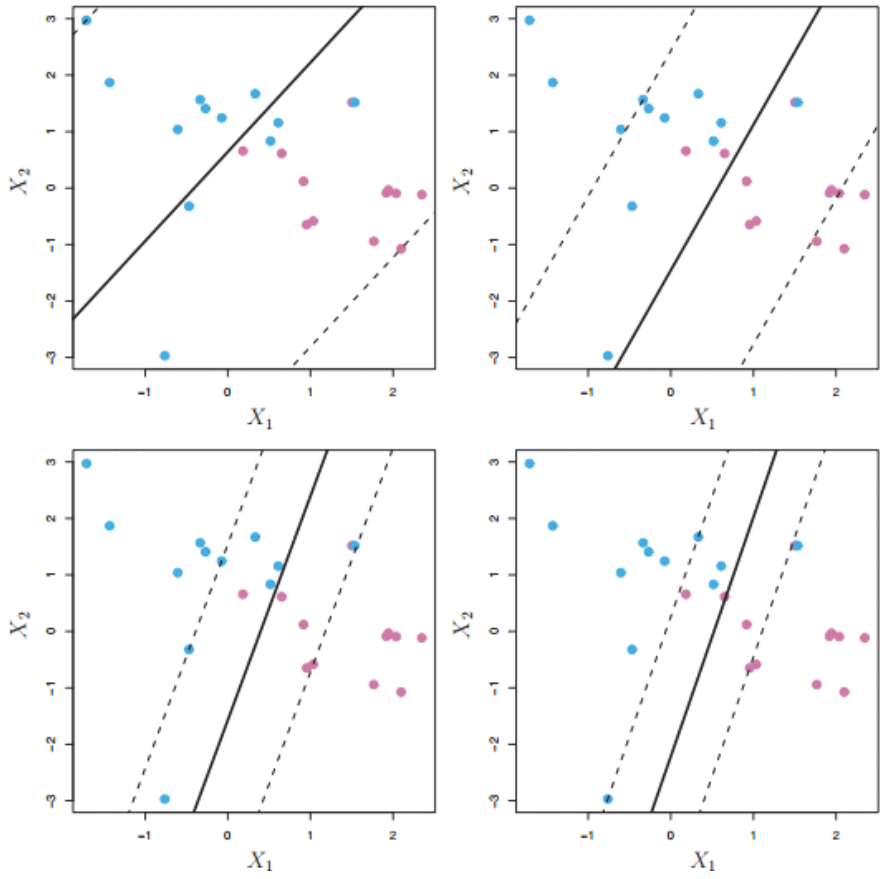
- In 9.14, C bounds the sum of the ϵ_i's, and so it determines the number and severity of the violations to the margin and to the hyperplane that we will tolerate.
- So, C is kind of a **budget for the amount that the margin can be violated by the n observations.**

  - If C = 0 then there is no budget for violations to the margin
    - It must be the case that all slack variables are zero,
    - So, (9.12)–(9.15) simply amounts to the maximal margin hyperplane optimization problem (9.9)–(9.11). (Of course, a maximal margin hyperplane exists only if the two classes are separable.)

  - For C > 0 no more than C observations can be on the wrong side of the hyperplane, because if an observation is on the wrong side of the hyperplane then ϵ_i > 1, and (9.14) requires that

$$\sum_{i=1}^{n} \epsilon_i \leq C.$$

  - As the budget C increases, we become more tolerant of violations to the margin, and so the margin will widen. Conversely, as C decreases, we become less tolerant of violations to the margin and so the margin narrows.



- C is a tuning parameter that is chosen via cross-validation and controls the bias-variance trade-off:
  - When C is small, we seek narrow margins that are rarely violated; this amounts to a classifier that is highly fit to the data, which may have low bias but high variance.
  - When C is larger, the margin is wider and we allow more violations to it; this amounts to fitting the data less hard and obtaining a classifier that is potentially more biased but may have lower variance.

(try analogy with linear regression)

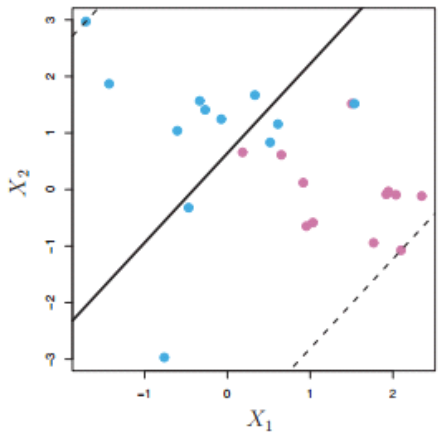
Support Vectors

- It turns out that only observations that either lie on the margin or that violate the margin will affect the hyperplane, and hence the classifier obtained.
- In other words, an observation that lies strictly on the correct side of the margin does not affect the classifier i.e. changing their position would not affect the classifier at all, provided that its position remains on the correct side of the margin.

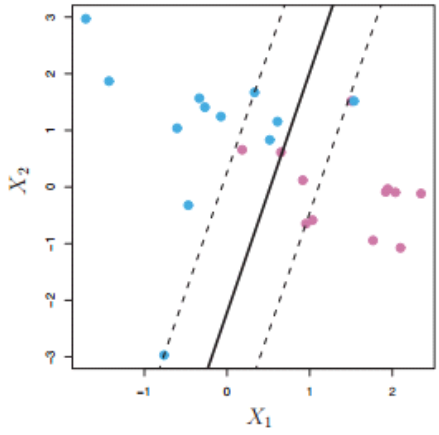
Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as **support vectors.** These observations do affect the support vector classifier.

Bias and Variance

- When the tuning parameter C is large, then the margin is wide, many observations violate the margin, and so there are many support vectors. In this case, many observations are involved in determining the hyperplane. So, this classifier has low variance but potentially high bias.



- In contrast, if C is small, then there will be fewer support vectors and hence the resulting classifier will have low bias but high variance.



Conclusion

- The fact that the support vector classifier's decision rule is based only on a potentially small subset of the training observations (the support vectors) means **that it is quite robust to the behavior of observations that are far away from the hyperplane.**

- This property is distinct from some of the other classification methods, such as linear discriminant analysis.
  - The LDA classification rule depends on the mean of all of the observations within each class, as well as the within-class covariance matrix computed using all of the observations.
  - In contrast, logistic regression, unlike LDA, has very low sensitivity to observations far from the decision boundary.

Support Vector Machines

Monday, November 28, 2016 19:58

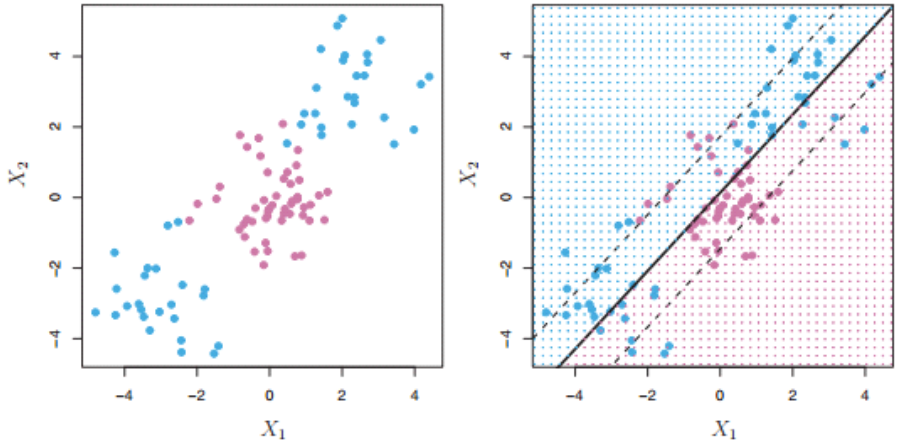
Contents

- A general mechanism for converting a linear classifier into one that produces non-linear decision boundaries.
- Support vector machine, which does this in an automatic way.

Classification with Non-linear Decision Boundaries

- The support vector classifier is a natural approach for classification in the two-class setting, if the boundary between the two classes is linear. However, in practice there are many instances of non-linear class boundaries.

For example, a support vector classifier or any linear classifier will perform poorly on the data below as shown in the right-hand panel.



- Analogously, the performance of linear regression can suffer when there is a nonlinear relationship between the predictors and the outcome. In that case, we consider enlarging the feature space using **functions of the predictors**, such as quadratic and cubic terms, in order to address this non-linearity.

- In the case of the support vector classifier, we could address the problem of non-linear boundaries between classes **by enlarging the feature space using quadratic, cubic, and even higher-order polynomial functions of the predictors.**

For instance, rather than fitting a support vector classifier using p features.

$$X_1, X_2, \ldots, X_p,$$

we could instead fit a support vector classifier using 2p features

$$X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2.$$

Then the classifier would be the solution of the following optimization problem:

$$\underset{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \qquad (9.16)$$

$$\underset{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \tag{9.16}$$

$$\text{subject to } y_i\left(\beta_0+\sum_{j=1}^{p}\beta_{j1}x_{ij}+\sum_{j=1}^{p}\beta_{j2}x_{ij}^2\right)\geq M(1-\epsilon_i),$$

$$\sum_{i=1}^{n}\epsilon_i\leq C,\ \ \epsilon_i\geq 0,\ \ \sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2=1.$$

Why does this lead to a non-linear decision boundary?

- In the enlarged feature space, the decision boundary that results from (9.16) is in fact linear. But in the original feature space, the decision boundary is of the form q(x) = 0, where q is a quadratic polynomial, and its solutions are generally non-linear.
- We can enlarge the feature space with higher-order polynomial terms, or with interaction terms, or other functions of the predictors rather than polynomials.
- We could end up with a huge number of features and then computations would become unmanageable.

The Support Vector Machine

The main idea: We may want to enlarge our feature space in order to accommodate a non-linear boundary between the classes. The kernel approach is simply an efficient computational approach for enacting this idea.

Support vector machine (SVM) is an extension of the support vector classifier that results from enlarging the feature space using kernels.

How the support vector classifier is computed?

Solution to the support vector classifier problem (9.12)–(9.15) involves only the **inner products of the observations** (as opposed to the observations themselves).

The inner product (dot product) of two r-vectors a and b is defined as

$$\langle a,b\rangle=\sum_{i=1}^{r}a_ib_i$$

Thus the inner product of two observations

$$x_i,\ x_{i'}$$

is given by

$$\langle x_i,x_{i'}\rangle=\sum_{j=1}^{p}x_{ij}x_{i'j}. \tag{9.17}$$

So,

The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle, \qquad (9.18)$$

where there are n parameters α_i, i = 1, . . . , n, one per training observation.

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.
  (The notation $\binom{n}{2}$ means n(n − 1)/2, and gives the number of pairs among a set of n items.)
- In order to evaluate the function f(x), we need to compute the inner product between the new point x and each of the training points x_i.

However, it turns out that α_i **is nonzero only for the support vectors in the solution**—that is, if a training observation is not a support vector, then its α_i equals zero.

So if **S is the collection of indices of these support points**, we can rewrite 9.18 as

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle, \qquad (9.19)$$

which typically involves far fewer terms.

To summarize, in representing the linear classifier f(x), and in computing its coefficients, all we need are inner products.

Kernel

- Replace the inner product in the calculation of the solution for the support vector classifier with **its generalization** of the form

  $$K(x_i, x_{i'}), \qquad (9.20)$$

  where K is some function called kernel.
- **A kernel is a function that quantifies the similarity of two observations.**

Kinds of Kernel

1. **Linear kernel**

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}, \qquad (9.21)$$

   - This gives us the inner product in the support vector classifier.
   - It is known as a **linear kernel** because it is linear in the features.
   - The linear kernel essentially quantifies the similarity of a pair of observations using Pearson (standard) correlation.

But one could instead choose another form for (9.20). For instance, one could replace every instance of

$$\sum_{j=1}^{p} x_{ij}x_{i'j}$$

with the quantity

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij}x_{i'j})^d. \qquad (9.22)$$

2. This is known as a **polynomial kernel** of degree d, where d is a positive integer.

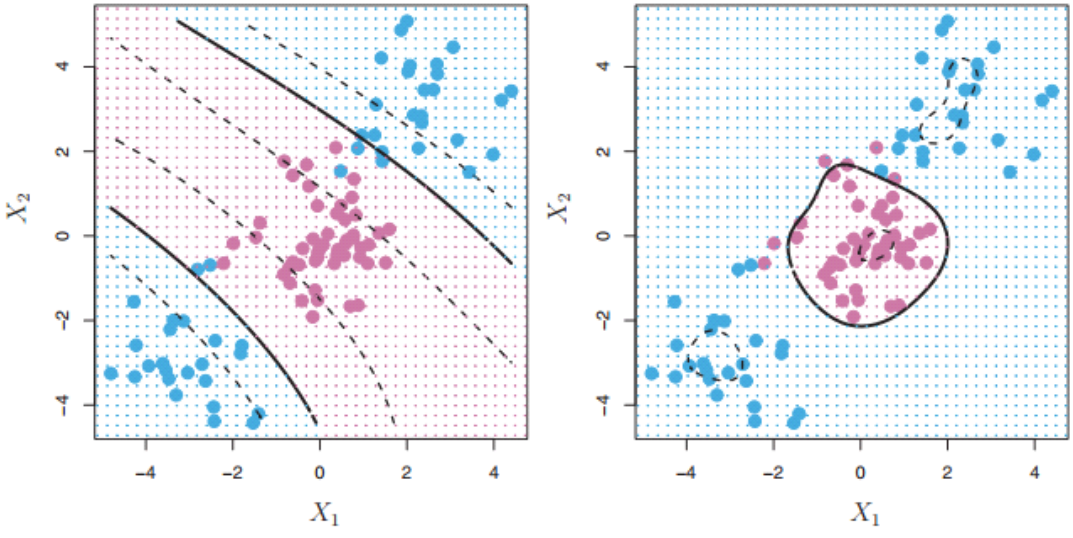
- Using such a kernel with d > 1, instead of the standard linear kernel (9.21), in the support vector classifier algorithm leads to a much more flexible decision boundary.
- It essentially amounts to fitting a support vector classifier in a higher-dimensional space involving polynomials of degree d, rather than in the original feature space.

When the support vector classifier is combined with a non-linear kernel such as (9.22), the resulting classifier is known as a support vector machine.

Non-linear function has the form

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i). \qquad (9.23)$$

The left-hand panel shows an example of an SVM with a polynomial kernel applied to the non-linear data.



When d = 1, then the SVM reduces to the support vector classifier

3. **Radial Kernel**

This kernel takes the form

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2). \qquad (9.24)$$

γ is a positive constant.

How does the radial kernel actually work?

If a given test observation

$$x^* = (x_1^* \dots x_p^*)^T$$

is far from a training observation x_i in terms of Euclidean distance, then

$$\sum_{j=1}^{p}(x_j^* - x_{ij})^2$$

will be large, and so

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p}(x_j^* - x_{ij})^2)$$

will be very tiny.

- This means that in (9.23), x_i will play virtually no role in f(x∗).
- Also, the predicted class label for the test observation is based on the sign of f(x∗).

So, training observations that are far from x∗ will play essentially no role in the predicted class label for x∗.

This means that the radial kernel has very local behavior, in the sense that only nearby training observations have an effect on the class label of a test observation.

What is the advantage of using a kernel rather than simply enlarging the feature space using functions of the original features, as in (9.16)?

Computational Advantage

It amounts to the fact that using kernels, one need only compute

$$K(x_i, x_{i'}) \text{ for all } \binom{n}{2} \text{ distinct pairs } i,\ i'.$$

This can be done without explicitly working in the enlarged feature space.

This is important because in many applications of SVMs, the enlarged feature space is so large that computations are intractable. For some kernels, such as the radial kernel, the feature space is implicit and infinite-dimensional, so we could never do the computations there anyway!

SVMs with More than Two Classes

Wednesday, November 30, 2016 07:45

The concept of separating hyperplanes upon which SVMs are based does not lend itself naturally to more than two classes.
The two most popular are the one-versus-one and one-versus-all approaches.

One-Versus-One Classification

- This approach constructs $\binom{k}{2}$ SVMs, each of which compares a pair of classes. For example, one such SVM might compare the kth class, coded as +1, to the k'th class, coded as -1.
- We classify a test observation using each of the $\binom{k}{2}$ classifiers, and we tally the number of times that the test observation is assigned to each of the K classes.
- The final classification is performed by assigning the test observation to the class to which it was **most frequently assigned** in these $\binom{k}{2}$ pairwise classifications.

One-Versus-All Classification

- We fit K SVMs, each time comparing one of the K classes to the remaining K $-$ 1 classes.
- Let
  $\beta_{0k}, \beta_{1k}, \ldots, \beta_{pk}$
  denote the parameters that result from fitting an SVM comparing the kth class (coded as +1) to the others (coded as -1).
- Let x$*$ denote a test observation. We assign the observation to the class for which
  $\beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* + \ldots + \beta_{pk}x_p^*$
  is largest, as this amounts to a high level of confidence that the test observation belongs to the kth class rather than to any of the other classes.

Implementation of svm using e1071 package

Wednesday, November 30, 2016 12:44

The svm() function from e1071 library can be used to fit a support vector classifier when the argument kernel="linear" is used.
This function uses a slightly different formulation from (9.14) and (9.25) for the support vector classifier.

1. **Cost Argument**

A cost argument allows us to specify the cost of a violation to the margin. This is opposite of the budget.

- When the cost argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin.
- When the cost argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin.

2. In order for the svm() function to perform classification (as opposed to SVM-based regression), we must encode the response as a factor variable

Contents

Monday, November 28, 2016 18:08

Convolutional Neural Networks

Friday, December 2, 2016 07:17

Use KDnuggets