Pattern Recognition and Machine Learning

James L. Crowley

ENSIMAG 3 - MMIS | Fall Semester 2016
Lessons 6 | 10 Jan 2017

Perceptrons and Support Vector machines

Outline

Notation

x_d	A feature. An observed or measured value.
$\vec{X}$	A vector of D features.
D	The number of dimensions for the vector $\vec{X}$
$\{\vec{X}_m\}$	Training samples for learning.
$\{y_m\}$	The indicator variable for each training sample,

$y_m =$ +1 for examples of the target pattern (class 1)

$y_m =$ –1 for all other examples (class 2)

M	The number of training samples.

$\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$ The coefficients for a linear model the model.

b bias so that $\vec{w}^T\vec{X}_m + b \geq 0$ for Class 1 and $\vec{w}^T\vec{X}_m + w_0 < 0$ for Class 2.

IF $\vec{w}^T\vec{X}_m + b \geq 0$ Then P ELSE N

IF $y_m\left(\vec{w}^T\vec{X}_m + b\right) \geq 0$ THEN T ELSE F

$\gamma = \min\left\{y_m \cdot \left(\vec{w}^T\vec{X}_m + b\right)\right\}$ Margin for the classifier

Perceptrons

History

During the 1950's, Frank Rosenblatt developed the idea to provide a trainable machine for pattern recognition, called a Perceptron. The perceptron is an incremental learning algorithm for linear classifiers invented by Frank Rosenblatt in 1956. The first Perceptron was a room-sized analog computer that implemented Rosenblatz learning recognition functions. Both the learning algorithm and the resulting recognition algorithm are easily implemented as computer programs.

In 1969, Marvin Minsky and Seymour Patert of MIT published a book entitled "Perceptrons", that claimed to document the fundamental limitations of the perceptron approach. Notably, they claimed that a linear classifier could not be constructed to perform an "exclusive OR". While this is true for a one layer perceptron, it is not true for multi-layer perceptrons.

Theory

The perceptron is an on-line learning method in which a linear classifier is improved by its own errors. A perceptron learns a linear decision boundary (hyper-plane) that separates training samples.

When the training data can be separated by a linear decision boundary, the data is said to be "separable". For a 2-Class classifier, the "margin", γ, is the smallest separation between the two classes.

The perceptron algorithm uses errors in classifying the training data to update the decision boundary plane until there are no more errors. The learning can be incremental. New training samples can be used to update the perceptron.

If the training data is non-separable, the method may not converge, and must be stopped after a certain number of iterations.

Assume a training set of M observations $\{\vec{X}_m\}$ $\{y_m\}$ where

$$\vec{X}_m = \begin{pmatrix} x_{1m} \\ x_{2m} \\ \vdots \\ x_{Dm} \end{pmatrix} \text{ and } y_m = \{-1, +1\}$$

The indicator variable, $\{y_m\}$, for each sample,

$y_m =$ +1 for examples of the target class (class 1)

$y_m =$ –1 for all others (class 2)

The Perceptron will learn the coefficients for a linear boundary

$$\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \text{ and } b$$

Such that for all training data, $\vec{X}_m$,

$\vec{w}^T\vec{X}_m + b \geq 0$ for Class 1 and $\vec{w}^T\vec{X}_m + w_0 < 0$ for Class 2.

Note that $\vec{w}^T\vec{X}_m + b \geq 0$ is the same as $\vec{w}^T\vec{X}_m \geq -b$.
Thus b can be considered as a threshold on the product : $\vec{w}^T\vec{X}_m$

The decision function is the sgn() function:

$$\operatorname{sgn}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

The algorithm requires a learning rate, α.

Note that a training sample is correctly classified if:

$$y_m \cdot \left(\vec{w}^T\vec{X}_m + b\right) \geq 0$$

The Perceptron Algorithm

The algorithm will continue to loop through the training data until it makes an entire pass without a single miss-classified training sample. If the training data are not separable then it will continue to loop forever.

Algorithm:

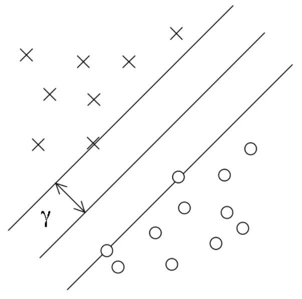
```
    $\vec{w}^{(0)} \leftarrow 0$; $b^{(i)} \leftarrow 0$, $i \leftarrow 0$;
    R $\leftarrow$ max { $\|\vec{X}_m\|$}
    WHILE update DO
        update $\leftarrow$ FALSE;
        FOR $m = 1$ TO M DO
            IF $y_m \cdot \left(\vec{w}^{(i)T}\vec{X}_m + b^{(i)}\right) < 0$ THEN
                update $\leftarrow$ TRUE
                $\vec{w}^{(i+1)} \leftarrow \vec{w}^{(i)} + \alpha \cdot y_m \cdot \vec{X}_m$
                $b^{(i+1)} \leftarrow b^{(i)} + \alpha \cdot y_m \cdot R^2$
                $i \leftarrow i + 1$
            END IF
        END FOR
    END WHILE.
```

The final classifier is: if $\text{sgn}\left(\vec{w}^{(i)T}\vec{X} + b^{(i)}\right) \geq 0$ then P else N.

The margin, γ_m, for each sample, m, is $\gamma_m = y_m \cdot \left(\vec{w}^{(i)T}\vec{X}_m + b^{(i)}\right)$

Thus the margin for a classifier and a set of training data is the minimum margin of the training data

$$\gamma = \min\left\{y_m \cdot \left(\vec{w}^{(i)T}\vec{X}_m + b^{(i)}\right)\right\}$$



The quality of a perceptron is given by the histogram of the margins, h(γ) for the training data.

If the data is not separable, then the Perceptron will not converge, and continue to loop. Thus it is necessary to have a limit the number of iterations.

Support Vector Machines

Support Vector Machines (SVM), also known as maximum margin classifiers are popular for problems of classification, regression and novelty detection. The solution of the model parameters corresponds to a convex optimization problem. SVM's use a minimal subset of the training data (the "support vectors) to define the "best" decision surface between two classes. We will use the two class problem, K=2, to illustrate the principle. Multi-class solutions are possible.

The simplest case, the hard margin SVM, require that the training data be completely separated by at least one hyper-plane. This is generally achieved by using a Kernel to map the features into a high dimensional space were the two classes are separable.

To illustrate the principle, we will first examine a simple linear SVM where the data are separable. We will then generalize with Kernels and with soft margins.

We will assume that the training data is a set of M training samples $\{\vec{X}_m\}$and their indicator variable, $\{y_m\}$, where , y_m is -1 or +1.

Hard-Margin SVMs - a simple linear classifier.

The simplest case is a linear classifier trained from separable data.

$$g(\vec{X}) = \vec{W}^T\vec{X} + b$$

Where the decision rule is: IF $\vec{W}^T\vec{X} + b > 0$ THEN P else N

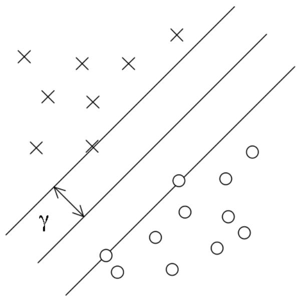
For a hard margin SVM we assume that the two classes are separable for all of the training data:

$$\forall m: \ y_m(\vec{W}^T\vec{X}_m + b) > 0$$

We will use a subset S of the training samples, $\{\vec{X}_s\} \subset \{\vec{X}_m\}$ composed of M_s training samples to define the "best" decision surface $g(\vec{X}) = \vec{W}^T\vec{X} + b$. The minimum number of support vectors depends on the number of features (M_s = D+1). The M_s selected training samples $\{\vec{X}_s\}$ are called the support vectors. For example, in a 2D feature space we need only 3 training samples to serve as support vectors.

Thus to define the classifier we will look for the subset if S training samples that maximizes the separation between the two classes.

The separation is defined using the margin: , γ.



Finding for the support vectors.

Assume that we have M training samples $\{\vec{X}_m\}$and their indicator variable $\{y_m\}$, where, y_m is -1 or +1.

Assume that we have normalized the coefficients of the hyperplane such at

$$\|\vec{W}\| = 1$$

Then the distance of any sample point $\vec{X}_m$ from the hyper-plane, $\vec{W}$.

$$d = y_m(\vec{W}^T\vec{X}_m + b)$$

The margin is the minimum distance

$$\gamma = \min\{y_m(\vec{W}^T\vec{X}_m + b)\}$$

A D dimensional decision surface is defined by at least D points. At least one additional point is required to define the margin. Thus we seek a subset of $M_s=D+1$ training samples, $\{\vec{X}_s\} \subset \{\vec{X}_m\}$ to define a decision surface and is margin.

Our algorithm must choose D+1 training samples $\{\vec{X}_s\}$ from the M Training samples in $\{\vec{X}_m\}$ such that the margin is a large as possible. This is equivalent to a search for a pair of parallel surfaces a distance γ from the decision surface.

We will use these samples as support vectors.

For the D+1 support vectors $\{\vec{X}_s\}$

$$d_s = \gamma$$

For all other training samples:

$$d_m \geq \gamma$$

To find the support vectors, we can arbitrarily define the margin as $\gamma = 1$ and then renormalize $\|\vec{W}\|$ once the support vectors have been discovered. We will look for two separate hyper-planes that "bound" the decision surface, such that for points on these surfaces:

$$\vec{W}^T\vec{X} + b = 1 \text{ and } \vec{W}^T\vec{X} + b = -1$$

The distance between these two planes is $\frac{2}{\|\vec{W}\|}$

We will add the constraint that for all training samples that are support vectors

$$\vec{W}^T\vec{X}_m + b \geq 1$$

while for all other samples:

$$\vec{W}^T\vec{X}_m + b \leq -1$$

This can be written as: $y_m(\vec{W}^T\vec{X}_m + b) \geq 1$

This gives we have an optimization algorithm that minimizes $\|\vec{W}\|$ subject to

$$y_m(\vec{W}^T\vec{X}_m + b) \geq 1.$$

If we note that minimizing $\|\vec{W}\|$ is equivalent minimizing $\frac{1}{2}\|W\|^2$, we can set this up as a quadratic optimization problem, and use Lagrange Multipliers.

Our problem is then to find $\arg\text{-}\min_{W,b}\{\|\vec{W}\|^2\}$ such that $y_m(\vec{W}^T\vec{X}_m + b) \geq 1$

We look for a surface, ($\vec{W}$, b) such that

$$\arg-\min_{\vec{W},b}\left\{\frac{1}{2}\|\vec{W}\|^2\right\} \quad \text{subject to} \quad y_m(\vec{W}^T\vec{X}_m+b)\geq 1$$

by searching for :

$$\arg-\min_{\vec{W},b}\{\max_{\alpha_m}\{\frac{1}{2}\|W\|^2-\sum_{n=1}^{N}a_m[y_m(\vec{W}^T\vec{X}_m+b)-1]\}\}$$

for a subset of $M_s \geq D+1$ samples, $a_m \geq 0$. These are the samples on the margins.

For all other samples where $(a_m < 0)$ we set $a_m = 0$.

The normal of the decision surface is then:

$$\vec{W}=\sum_{m=1}^{M}a_m y_m\vec{X}_m$$

and the offset can be found by solving for:

$$b=\frac{1}{M_S}\sum_{m\in S}\vec{W}^T\vec{X}_m-y_m$$

The solution can be generalized for use with non-linear decision surfaces using kernels.

Support Vector Machines with Kernel Functions

Kernel Functions

A Kernel function transforms the training data so that a non-linear decision surface is transformed to a linear equation in a higher number of dimensions.

Linear discriminant functions can provide very efficient 2-class classifiers, provided that the class features can be separated by a linear decision surface.

For many domains, it is easier to separate the classes with a linear function if you can transform your feature data into a space with a higher number of dimensions.

One way to do this is to transform the features with a "kernel" function.

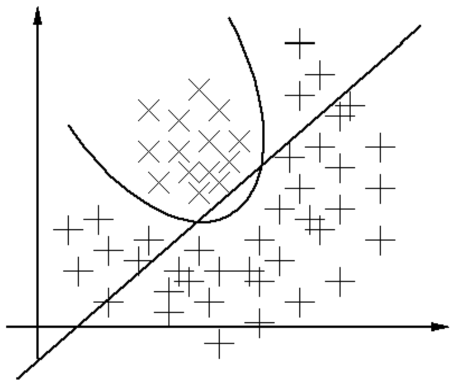
Instead of a decision surface: $g(\vec{X}) = \vec{W}^T \vec{X} + b$

We use a decision surface $g(\vec{X}) = \vec{W} \cdot \vec{f}(\vec{X}) + b$

where $\vec{W} = f(\vec{Z}) = \sum_{m=1}^{M} a_m y_m f(\vec{X}_m)$ is learned from the transformed training data.

and a_m is a binary variable learned from the training data that is $a_m \geq 0$ for support vectors and 0 for all others.

The function $\vec{f}(\vec{X})$ provides an implicit non-linear decision surfaces for the original data.



Formally, a "kernel function" is any function that satisfies the Mercer condition.

A function, *k(x, y)*, satisfies Mercer's condition if for all square, integrable functions *f(x)*,

$$\iint k(x,y)f(x)f(y)dxdy \geq 0$$

This condition is satisfied by inner products (dot products) $\vec{W}^T\vec{X} = \sum_{d=1}^{D} w_d x_d$

Thus $k(\vec{W},\vec{X}) = \vec{W}^T\vec{X}$ is a valid kernel function. (Known as the linear kernel).

as is $k(\vec{Z},\vec{X}) = f(\vec{Z})^T f(\vec{X})$

We can learn the discriminant in an inner product space $k(\vec{Z},\vec{X}) = f(\vec{Z})^T f(\vec{X})$ where $\vec{W}$ will be learned from the training data.

This will give us $g(\vec{X}) = \vec{W}^T f(\vec{X}) + b$

The Mercer function can be satisfied by many other functions.
Popular kernel functions include:

- Polynomial Kernels
- Radial Basis Functions
- Fisher Kernels
- Text intersection
- Bayesian Kernels

Kernel functions provide an implicit feature space. We will see that we can learn in the kernel space, and then recognize without explicitly computing the position in this implicit space!

This will allow us to use Kernels for infinite dimensional spaces as well as non-numerical and symbolic data!

Radial Basis Function (RBF)

Radial basis function (RBF) are popular for use a kernel function.
In this case, each support vector acts as a dimension in the new feature space.

The RBF Kernel function has the form:

$$g(\vec{X}) = \sum_{n=1}^{N} \vec{W}_n^T f\left(\left\|\vec{X} - \vec{X}_n\right\|\right)$$

Where the points N samples $\vec{X}_n$ can be derived from the training data.

The term $\left\|\vec{X} - \vec{X}_n\right\|$ is the Euclidean distance from the set of points $\left\{\vec{X}_n\right\}$.

The distance can be normalized by dividing by σ

$$g(\vec{X}) = \sum_{n=1}^{N} \vec{W}_n^T f\left(\frac{\left\|\vec{X}-\vec{X}_n\right\|}{\sigma}\right)$$

When used in this way, each center point, $\vec{c}$, is one of the support vectors.
The sigma parameter acts as a smoothing parameter that determines the influence of each of the points, $\vec{X}_n$. The zero-crossings in the distances define the decision surface.

The Gaussian function

$$f(\|\vec{x} - \vec{c}\|) = e^{-\frac{\|\vec{x}-\vec{c}\|^2}{2\sigma^2}}$$

is a popular Radial Basis Function, and is often used as a kernel for support vector machines.

We can use a sum of N radial basis functions to define a discriminant function, where the support vectors are drawn from the M training samples. This gives a discriminant function

$$g(\vec{X},\vec{w}) = \sum_{m=1}^{M} a_m y_m f(\| \vec{X} - \vec{X}_m \|) + w_0,$$

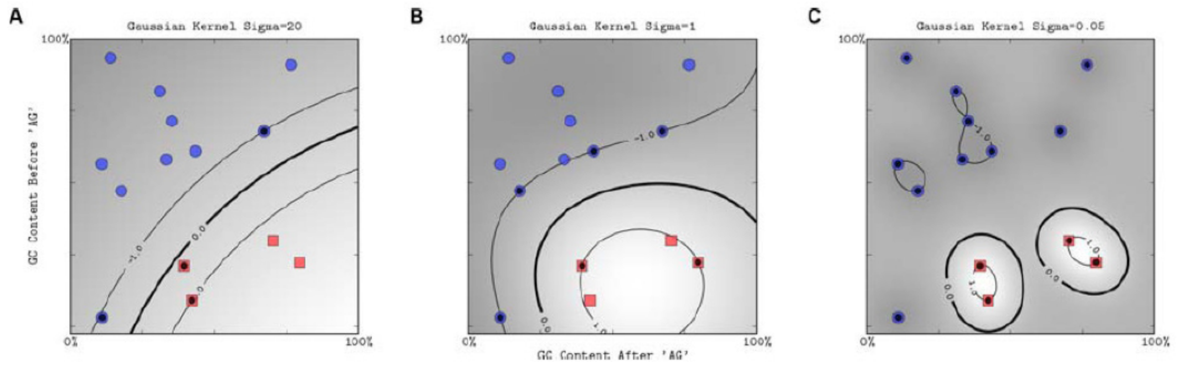
The training samples $\vec{X}_m$ for which $a_m \neq 0$ are the support vectors.

The distance can be normalized by dividing by σ

$$g(\vec{X}) = \sum_{n=1}^{N} \vec{W}_n^T f\left(\frac{\left\|\vec{X}-\vec{X}_n\right\|}{\sigma}\right)$$

Depending on σ, this can provide a good fit or an over fit to the data. If σ is large compared to the distance between the classes, this can give an overly flat discriminant surface. If σ is small compared to the distance between classes, this will over-fit the samples.

A good choice for σ will be comparable to the distance between the closest members of the two classes.



(images from "A Computational Biology Example using Support Vector Machines", Suzy Fei, 2009)

Each Radial Basis Function is a dimension in a high dimensional basis space.

Kernel Functions for Symbolic Data

Kernel functions can be defined over graphs, sets, strings and text!

Consider for example, a non-vector space composed of a set of words {W}.
We can select a subset of discriminant words {S} ⊂ {W}

Now given a set of words (a probe), {A} ⊂ {W}

We can define a kernel function of A and S using the intersection operation.

$$k(A,S) = 2^{|A \cap S|}$$

where | . | denotes the cardinality (the number of elements) of a set.