Intelligent Systems: Reasoning and Recognition

James L. Crowley

MoSIG M1 Winter Semester 2018
Lesson 6 27 February 2018

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 P and $\vec{w}^T\vec{x}_m + w_0 < 0$ for N.
	
	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

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

Linear Models

Lines, Planes and Hyper-planes

(a quick review of basic geometry)
In a 2-D space, a line is a set of points that obeys the relation:

$$w_1 x_1 + w_2 x_2 + b = 0$$

In vector notation: $\begin{pmatrix} w_1 & w_2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + b = 0$ which is written : $\vec{w}^T \vec{x} + b = 0$

If $\|\vec{w}\| = 1$ then b is the perpendicular distance to the origin.
For points that are not on the hyperplane, the linear equation gives the signed perpendicular distance to the hyperplane.

$$d = w_1 x_1 + w_2 x_2 + b$$

For points on one side of the line, $d > 0$. On the other side $d < 0$. On the line $d=0$.
if $\vec{w}^T \vec{x} + b = 0$ is a boundary then the sign of *d* can be used as a decision function.
Lines, planes and hyperplanes can act as boundaries for regions of a feature space.

This is called a "first order homogeneous" equation because the all terms are first order.

This is easily generalized to any number of dimensions (features).
In a 3D space, a plane is represented by $w_1 x_1 + w_2 x_2 + w_3 x_3 + b = 0$
In a D dimensional space, linear homogeneous equation is called a hyper-plane.
The equation $\vec{w}^T \vec{x} + b = 0$ is a hyper-plane in a D-dimensional space,

$\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$ is the normal to the hyper-plane and b is a constant term.

The signed distance is : $d = \vec{w}^T \vec{X} + b = w_1 x_1 + w_2 x_2 + ... + w_D x_D + b$
indicates which side of the boundary a point lies.

Note that it is sometimes convenient to include the bias in the vector:

$$\begin{pmatrix} w_1 & w_2 & b \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} = 0$$

Perceptrons

History

During the 1950's, Frank Rosenblatt developed the idea to provide a trainable linar classifier for pattern recognition, called a Perceptron. The first Perceptron was a room-sized analog computer that implemented Rosenblatz's learning algorithm. 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 limits of the perceptron approach. For example, they showed 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".

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 (P)

$y_m = -1$ for all others classes (N)

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 P and $\vec{w}^T\vec{x}_m + b < 0$ for N.

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 d(z) function:

$$d(z) = \begin{cases} P & \text{if } z \geq 0 \\ N & \text{if } z < 0 \end{cases}$$

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 uses misclassified training data as a correction.
The algorithm will continue to loop thugh 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.
Update rule:

```
FOR $m = 1$ TO  M  DO
    IF  $y_m \cdot \left(\vec{w}^T\vec{x}_m + b\right) < 0$ THEN
        $\vec{w} \leftarrow \vec{w} + \alpha \cdot y_m \cdot \vec{x}_m$
        $b \leftarrow b + \alpha \cdot y_m \cdot R$
```

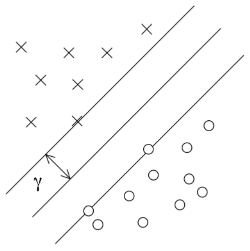
where R is a scale factor for the data $R = \max\|\vec{x}_m\|$
If the data is not separable, then the Perceptron will not converge, and continue to loop. Thus it is necessary to have a limit the number of iterations.

The final classifier is: if $\vec{w}^T\vec{x}_m + b \geq 0$ then P else N.

Perceptron Margin

For a 2-Class classifier, the "margin", γ, is the smallest separation between the two classes. The margin, γ_m, for each sample, m, is $\gamma_m = y_m \cdot \left(\vec{w}^T \vec{x}_m + b\right)$

The margin for a classifier and a set of training data is the minimum margin of the training data $\gamma = \min\left\{y_m \cdot \left(\vec{w}^T \vec{x}_m + b\right)\right\}$



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

Full perceptron algorithm (Optional)

For reference, here is the full perceptron algorithm.

Algorithm:

```
    $\vec{w}^{(0)} \leftarrow 0$; $b^{(0)} \leftarrow 0$, $i \leftarrow 0$; $R = \max\|\vec{x}_m\|$
    WHILE update DO
        update ← 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 ← TRUE
                $\vec{w}^{(i+1)} \leftarrow \vec{w}^{(i)} + \alpha \cdot y_m \cdot \vec{x}_m$
                $b^{(i+1)} \leftarrow b^{(i)} + \alpha \cdot y_m \cdot R$
                $i \leftarrow i + 1$
            END IF
        END FOR
    END WHILE.
```

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 $\{\vec{x}_m\}$ with an 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.

$g(\vec{x}) = \vec{w}^T\vec{x} + b$ with the decision rule is: IF $g(\vec{x}) \geq 0$ THEN P else N

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

$$\forall m: \; y_m(\vec{w}^T\vec{x}_m + b) \geq 0$$

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 $g(\vec{x}) = \vec{w}^T\vec{x} + b$.

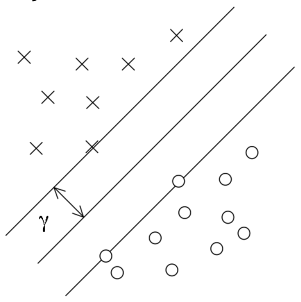
The minimum number of support vectors depends on the number of features, D:

$M_s = D+1$

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.

To define the classifier we will look for the subset if S training samples that maximizes the margin (γ) between the two classes.

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



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 the margin. We will use these samples as support vectors.

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.

Finding the Support Vectors

For any hyperplane with normalized the coefficients: $\|\vec{w}\| = 1$
the distance of any sample point $\vec{x}_m$ from the hyper-plane, $\vec{w}$ is

$$d = y_m \cdot \left(\vec{w}^T\vec{x}_m + b\right)$$

The margin is the minimum distance

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

For the D+1 support vectors $\vec{x}_m \in \{\vec{x}_s\}$ $\quad d_m = \gamma$
For all other training samples $\vec{x}_m \notin \{\vec{x}_s\}$: $d_m \geq \gamma$

The scale of the margin is determined by $\|\vec{w}\|$.
To find the support vectors, we can arbitrarily define the margin as $\gamma = 1$ and then renormalize $\|\vec{w}\| = 1$ once the support vectors have been discovered.

With $\gamma = 1$, we will look for two separate hyper-planes that "bound" the decision surface, such that for points on these surfaces:

$\vec{w}^T\vec{x}_m + b = 1$ and $\vec{w}^T\vec{x}_m + b = -1$

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:

$\vec{w}^T\vec{x}_m + b < -1$

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

This is known as the Canonical Representation for the decision hyperplane.

This gives we have an optimization algorithm that
minimizes $\|\vec{w}\|$ subject to $y_m(\vec{w}^T\vec{x}_m + b) \geq 1$.

For points on the margin: $y_m(\vec{w}^T\vec{x}_m + b) = 1$
for all other points $y_m(\vec{w}^T\vec{x}_m + b) > 1$

The training sample where $y_m(\vec{w}^T\vec{x}_m + b) = 1$ are said to be "active".
All other training samples are "inactive".

Thus the optimization problem is to maximize $\arg\min_{\vec{w},b}\left\{\frac{1}{2}\|\vec{w}\|^2\right\}$ while minimizing the number of active points, S. (The factor of ½ is a convenience for analysis.)

This can be solved as a quadratic optimization problem using Lagrange Multipliers, with one multiplier, $a_m \geq 0$, for each constraint.

The Lagrangian function is: $L(\vec{w},b,\vec{a}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{m=1}^{M} a_m\left\{y_m(\vec{w}^T\vec{x}_m + b) - 1\right\}$

Setting the derivatives to zero, we obtain:

$$\frac{\partial L}{\partial \vec{w}} = 0 \Rightarrow \vec{w} = \sum_{m=1}^{M} a_m y_m \vec{x}_m \qquad \frac{\partial L}{\partial b} = 0 \Rightarrow b = \frac{1}{M_s}\sum_{x_m \in S} \vec{w}^T x_m - y_m$$

Soft Margin SVM

We can extend the SVM to the case where the training data are not separable by introducing the “hinge loss” function.

hinge loss: $L_m = \max\left\{0, 1 - y_m(\vec{w}^T\vec{x}_m + b)\right\}$

This loss is 0 is the data is separable, and non-zero if the data is non-separable.

Using the hinge loss, we can minimize

$$\left[\frac{1}{M}\sum_{m=1}^{M}\max\left\{0, 1 - y_m(\vec{w}^T\vec{x}_m + b)\right\}\right] + \lambda\|\vec{w}\|^2$$

To set this up as an optimization problem, we note that
$L_m = \max\left\{0, 1 - y_m(\vec{w}^T\vec{x}_m + b)\right\}$ is the smallest non-negative that satisfies

$$y_m(\vec{w}^T\vec{x}_m + b) \geq 1 - L_m$$

we rewrite the optimization as

minimize $\frac{1}{M}\sum_{m=1}^{M} L_m + \lambda\|\vec{w}\|^2$ subject to $y_m(\vec{w}^T\vec{x}_m + b) \geq 1 - L_m$ and $L_m \geq 0$ for all *m*.

This was classically solved as a quadratic optimization problem with Lagrange multipliers.

Maximize $\sum_{m=1}^{M} a_m - \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} y_i a_i (\vec{x}_i^T\vec{x}_j) y_j a_j$
subject to $\sum_{m=1}^{M} a_m y_m = 0$ and $0 \leq a_m \leq \frac{1}{2M\lambda}$ for all m

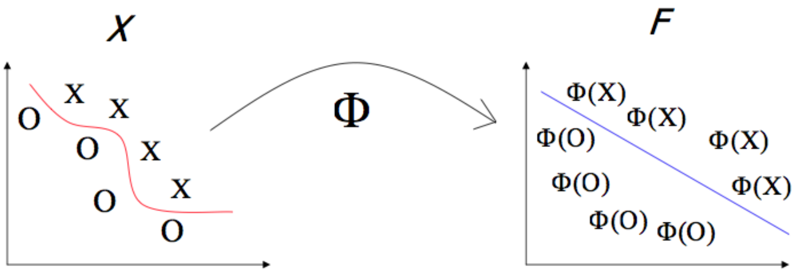
The resulting decision surface is : $\vec{w} = \sum_{m=1}^{M} a_m y_m \vec{x}_m$
and the bias is found by solving for: $b = \vec{w}^T\vec{x}_m - y_m$ for any support vector: $\vec{x}_m \in \{\vec{x}_s\}$

Modern approaches use Gradient Descent to solve for $\vec{w}$ and b, offering important advantages for large data sets.

SVM with Kernels

Support Vector Machines can be generalized for use with non-linear decision surfaces using kernel functions. This provides a very general and efficient classifier.

A "kernel" function, f(z), transforms the data into a space where the two classes are separate.



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

We use a decision surface $g(\vec{x}) = \vec{w}^T \cdot \vec{f}(\vec{x}) + b$

the coefficients $\vec{w}$ and the bias b are provided by the D+1 support vectors.
For this we need to include the

where $\vec{w} = \sum_{m=1}^{M} a_m y_m \vec{f}(\vec{x}_m)$

is learned from the transformed training data.

As before, $S = \{\vec{x}_s\}$ are the support vectors, and a_m is a variable learned from the training data that is $a_m \geq 0$ for support vectors and 0 for all others.

Radial Basis Function Kernels:

Radial Basis functions are a popular kernel function: A radial basis function (RBF) is a real-valued function whose value depends only on the distance from the origin.

For Example, the Gaussian function $f(z) = e^{-\frac{z^2}{2\sigma^2}}$

is a popular Radial Basis Function, and is often used as a kernel for support vector machines, with radial basis functions: $z = \| \vec{X} - \vec{x}_s \|$ for each support vectors.

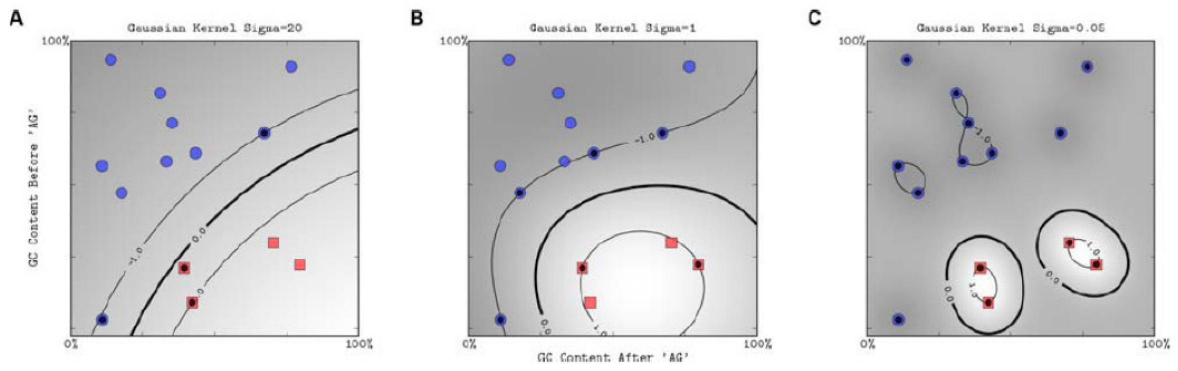
We can use a sum of M_s radial basis functions to define a discriminant function, where the support vectors are drawn from the M training samples.
This gives a discriminant function

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

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

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 $\{S\} \subset \{W\}$

Now given a set of words (a probe), $\{A\} \subset \{W\}$

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.