Machine Learning: Support Vector Machines: Linear Kernel

Support Vector Machines

Extending Perceptron Classifiers. There are two ways to extend perceptron classifiers. Each of the two ways corresponds to noticing a significant drawback in how perceptrons operate, and attempting to "fix" it.

Perceptron weakness #1: binary classification using a single hyperplane. The core problem with a perceptron is that it is a simple binary classifier that separates the classes using a *single* hyperplane. It cannot work well in one of the following cases:

- Separation boundary is non-linear[1].
- There is no clear separability of data points.
- More than one hyperplane is required for true separation of data points (the so called XOR scenario), i.e., the situation, when the function to learn is non-monotonic in all its inputs.

This weakness is addressed by **Neural Networks**, which are constructed as follows:

- Individual perceptrons are modified into *neurons* by replacing their activation function from a θ-threshold, to some differentiable function (e.g., a hyperbolic tangent function).
- Neurons are organized into networks, but putting the output of the activation function of one neuron as input to another neuron. Typical networks consist

[1] There are some space transformation techniques that can still allow one to use a single perceptron for some non-linearly separable cases.

of *layers* of neurons, with the input layer having one neuron per input data dimension, and *hidden layer*(s) being given as inputs the outputs of previous layers.

- The output layer may contain multiple neurons, which allows to extend binary classification into multi-class classification.

Perceptron weakness $\#2$: choosing the right hyperplane. This is actually two separate weaknesses:

- inability to deal with linearly inseparable datasets, and
- inability to properly determine the best separating hyperplane.

This weakness is addressed by Support Vector Machines.

Essentially, a perceptron is converted into a Support Vector Machine (SVM) by making the *error function* more complex.

Support Vector Machines with Linear Kernels

Note: The key issue making perceptrons not distinguish between "good" and "bad" separating hyperplanes is the *weak* error function. Support Vector Machines (SVMs) change the error function. An SVM attempts to select a hyperplane

$$\mathbf{w} \cdot \bar{x} + b = 0,$$

such that this hyperplane lies *as far as possible* from any point in the training set (while classifying properly as many points as possible).

Idea: Points that are far away from the decision boundary (or from any separating hyperplane) are *easy to classify* and give us more certainty about the class they belong to. Points that are near the decision boundary/separating hyperplane are harder to classify. The further away a separating hyperplane is from the nearest points, the easier it is to classify those nearest points, and the less uncertainty we have about their class.

Support Vectors. Given a separating hyperplane $\mathbf{w} \cdot \bar{x} + b = 0$, the points $\{\bar{z_1}, \ldots, \bar{z_k}\} \subseteq X$ which have the *shortest distance to the hyperplane*, i.e., all the points with the smallest absolute values $e = e_i = \mathbf{w} \cdot \bar{z_i} + b$[2] are called **support vectors** of the hyperplane.

Attempt 1. Given a training set $(X, Y) = \{(\bar{x_i}, y_i)\}, y_i \in \{-1, +1\}$, $\bar{x_i} = (a_1, \ldots, a_d)$ (for some d - number of features/dimensions), determine the vector of weights $\mathbf{w} = (w_1, \ldots, w_d)$ and the intercept b that **maximize the value** γ, such that for all $i = 1, \ldots, n$:

$$y_i \cdot (\mathbf{w} \cdot \bar{x_i} + b) \geq \gamma.$$

Intuitively, we want the largest value of γ, such that for all data points $\bar{x_i} \in X$, where $y_i = +1$, $(\mathbf{w} \cdot \bar{x_i} + b) >= \gamma$, and for all data points $\bar{x_i} \in X$, where $y_i = -1$, $(\mathbf{w} \cdot \bar{x_i} + b) <= -\gamma$.

[2] That is, the distance from **any** point $\bar{z_1}, \ldots, \bar{z_k}$ to the hyperplane is the same.

Problem: too many degrees of freedom. We can always increase all values of $\mathbf{w}$ and b and thus increase γ: if $y_i \cdot (\mathbf{w} \cdot \bar{x_i} + b) \geq \gamma$ then $y_i \cdot (\mathbf{2} \cdot \mathbf{w} \cdot \bar{x_i} + 2b) \geq 2\gamma$.

Changing the formulation of the problem. Consider the training set $(X, Y) = \{(\bar{x_i}, y_i)\}, y_i \in \{-1, +1\}$, $\bar{x_i} = (a_1, \ldots, a_d)$ and a hyperplane

$$h(\bar{x}) = \mathbf{w} \cdot \bar{x} + b = 0$$

Given two points $\bar{v_1} \neq \bar{v_2}$, such that $h(\bar{v_1}) = 0$ and $h(\bar{v_2}) = 0$ (i.e., two points on the hyperplane $h(\bar{x})$, we notice:

$$h(v_1) - h(v_2) = (\mathbf{w} \cdot \bar{v_1} + b) - (\mathbf{w} \cdot \bar{v_2} + b) =$$
$$\mathbf{w} \cdot (\bar{v_1} - \bar{v_2}) + (b - b) = \mathbf{w} \cdot (\bar{v_1} - \bar{v_2}) = 0,$$

from which we observe that vector $\mathbf{w}$ is orthogonal to the hyperplane $h(\bar{x})$.

Now, suppose $\bar{x} = (a_1, \ldots, a_d)$ is an arbitrary d-dimensional point. We can represent this point as the sum

$$\bar{x} = \bar{x_h} + \mathbf{r},$$

where $\bar{x_h}$ is the orthogonal projection of $\bar{x}$ onto the hyperplane $h(\bar{x})$, and $\mathbf{r}$ is the vector $\bar{x} - \bar{x_h}$. Here,

$$\mathbf{r} = r \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} = r \cdot \frac{\mathbf{w}}{\sqrt{\sum_{j=1}^{d} w_j^2}},$$

where r is the *directed* distance from $\bar{x_h}$ to $\bar{x}$ in terms of the *unit weight vector* $\frac{\mathbf{w}}{\|\mathbf{w}\|}$.

Therefore,

$$h(\bar{x}) = h\left(\bar{x_h} + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) = \mathbf{w} \cdot \left(\bar{x_h} + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b =$$

$$= \mathbf{w} \cdot \bar{x_h} + b + r\frac{\mathbf{w} \cdot \mathbf{w}}{\|\mathbf{w}\|} = h(\bar{x_h}) + r \cdot \frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} = r\|\mathbf{w}\|.$$

(remember that since $\bar{x_h}$ is on the hyperplane $h(\bar{x})$, $h(\bar{x_h}) = 0$.)

Therefore, the directed distance from any point $\bar{x}$ to the hyperplane $h(\bar{x})$ is:

$$r = \frac{h(\bar{x})}{\|\mathbf{w}\|}.$$

The *distance* δ from the point $\bar{x_i} in X$ to the hyperplane $h(\bar{x})$ is:

$$\delta_i = y_i \cdot r_i = y \cdot \frac{h(\bar{x_i})}{\|\mathbf{w}\|}.$$

Margin. Let $h(\bar{x})$ be a hyperplane and $(X, Y) = \{(\bar{x_i}, y_i)\}$ be a training set of size n. The **margin** δ^* of $h(\bar{x})$ is defined as

$$\delta^* = \min_i \left(y_i \frac{h(\bar{x_i})}{\|\mathbf{w}\|} \right) = \min_i(\delta_i),$$

where

$$\delta_i = y_i \frac{\mathbf{w} \cdot \bar{x_i} + b}{\|\mathbf{w}\|},$$

i.e., the **margin** of $h(\bar{x})$ is the **smallest relative distance** (in terms of the length of vector $\mathbf{w}$) from some training set point to the hyperplane.

Canonical planes. To address the issue with **Attempt 1**, we fix the scale of the hyperplane $h(\bar{x})$. Since the hyperplane equation can be multiplied by any scalar $s \neq 0$ and preserve the equality:

$$h(\bar{x}) = 0 \Rightarrow s \cdot h(\bar{x}) = s \cdot \mathbf{w} \cdot \bar{x} + sb = (\mathbf{s} \cdot \mathbf{w}) \cdot x + (sb) = 0,$$

we can limit ourselves to only considering the hyperplanes where s takes a restricted value. Specifically, **we want the absolute distance from the support vectors to the hyperplane** to be 1:

$$sy^*(\mathbf{w} \cdot \bar{x^*} + b) = 1,$$

for any support vector $\bar{x^*} \in X$ such that $y^* \frac{\mathbf{w} \cdot \bar{x} + b}{\|\mathbf{w}\|} = \delta^*$. This means that

$$s = \frac{1}{y^* h(x^*)}.$$

We will limit our search for $\mathbf{w}$ and b, i.e., for hyperplanes $h(\bar{x} = \mathbf{w} \cdot \bar{x} + b$ to those instances where the absolute distances to support vectors are equal to 1, that is:

$$\delta^* = \frac{y^* h(\bar{x^*}}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}.$$

We call such hyperplanes **canonical**.

Support Vector Machines: Linearly Separable Case

Optimization Problem for Linearly Separable SVMs. Given the training set $(X, Y) = \{(\bar{x_i}, y_i)\}, i = 1, \ldots, n$, where $\bar{x_i} \in \mathbb{R}^d$, and $y_i \in \{+1, -1\}$, find a *canonical hyperplane*

$$h(\bar{x}) = \mathbf{w} \cdot \bar{x} + b,$$

that maximizes the margin δ^*:

$$h^* = \arg\max_h(\delta_h^*) = \arg\max_{\mathbf{w}, b} \left(\frac{1}{\|\mathbf{w}\|} \right).$$

Note: Maximizing δ^* is the same as minimizing $\|\mathbf{w}\|$, which, in turn is the same as minimizing $\|\mathbf{w}\|^2 = \mathbf{w} \cdot \mathbf{w}$. Therefore, we can represent our problem as the following **optimization problem**:

Objective Function:

$$\min_{\mathbf{w},b} \left(\frac{\|\mathbf{w}\|^2}{2} \right)$$

Subject to constraints:

$$y_i(\mathbf{w} \cdot \bar{x_i} + b) \geq 1, \forall \bar{x_i} \in X$$

Soft Margin SVMs: Linear and Non-Separable Cases

If either

- Our training set contains points that make it linearly inseparable, or
- We suspect that some data points we will be asked to classify later will fall inside the margins of the support vector hyperplane,

we can correct our objective function to account for that. Essentially, in this case, we need to change the definition of support vectors. In a linearly non-separable cases, **support vectors are**:

- all support vectors from the linearly separable problem (i.e., all points that lie on the right side of the classification hyperplane at the distance 1 from it);
- all points from each class that lie on the wrong side of the hyperplane.

Because we added another group of *support vectors*, we can no longer use

$$y_i(\mathbf{w} \cdot \bar{x_i} + b) = 1$$

as the condition that determines which points are *support vectors*.

Instead, we introduce *slack variables* $\xi_i \geq 0$ to capture the fact that some support vectors may be closer or further away from the hyperplane:

$$y_i(\mathbf{w} \cdot \bar{x_i} + b) \geq 1 - \xi_i$$

We consider three cases:

1. $\xi_i = 0$. This turns the distance inequality into

   $$y_i(\mathbf{w} \cdot \bar{x_i} + b) \geq 1.$$

   This means that $\bar{x_i}$ is either our old support vector, or any point that lies on the correct side of the hyperplane *further away from the hyperplane* than the "old" support vectors.

2. $0 < \xi_i < 1$. In this case $1 - \xi_i > 0$, and therefore, $\bar{x_i}$ lies on the *correct side* of the hyperplane, but *it is closer to the hyperplane* than the "old-style" support vectors.

3. $\xi_i ge 1$. In this case, the point $\bar{x_i}$ either lies on the hyperplane (in which case we cannot really classify it) or is on the other side of the hyperplane, in which case it will be misclassified.

Hinge Loss. We define the *soft margin* SVM classifier as a canonical hyperplane $h(\bar{x})$ subject to the following conditions:

Objective Function:

$$\min_{\mathbf{w},b,\{\xi_i\}} \left(\frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{n} \xi_i \right)$$

Subject to constraints:

$$y_i(\mathbf{w} \cdot \bar{x_i} + b) \geq 1 - \xi_i, \forall \bar{x_i} \in X$$

The function

$$f(d) = \max(0, 1 - d)$$

is called the hinge loss function. This function represents the penalty assessed by an SVM construct for a point that is at the directed distance d from a given hyperplane.

- If the point is at the directed distance of 1 or more, it is on the "correct" side of the hyperplane and therefore, there is no penalty.
- If the point is at the distance between 0 and 1, it lies in the "neutral zone" between the hyperplane and the support vector from the same class as the point. We give this point a small penalty, because we do not like to see points closer to the hyperplane than the support vector.
- If the point is on the other side of the hyperplane, we penalize it by the distance from this point to the hyperplane that is parallel to ours, but that passes through the support vectors from the point's class. We do this because for us the penalty is not how far the point is from the hyperplane, but how far it is from "its" support vectors.

Training SVM Classifiers

Step 1. Get rid of the intercept. This can be accomplished by replacing all vectors $\bar{x} = (a_1, \ldots, a_d) \in X$ with the vectors $\bar{x'} = (a_1, \ldots, a_d, 1)$, and replacing the vector of weights $\mathbf{w} = (w_1, \ldots, w_d)$ with the vector $\mathbf{w}' = (w_1, \ldots, w_d, b)$. (This is a standard procedure that we have seen multiple times already).

Without loss of generality, we assume that all vectors $\mathbf{w}$ and $\bar{x}$ mentioned below have gone through this transformation.

Step 2. Pick the problem to optimize. There are two SVM problems that can be solved: *primal* and *dual*.

The dual problem is solved using Stochastic Gradient Descent, and it is the more commonly used technique. We discuss it in a separate handout.

The primal problem is one of the optimization problems described above (for linearly separable or soft-margin cases). It can also be solved using Stochastic Gradient Descent, but it requires some care.

Step 3. Optimizing the primal problem. Minimize objective function:[3]

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i)$$

subject to linear constraints

$$y_i(\mathbf{w}\cdot\bar{x_i}) \geq 1 - \xi_i; \xi_i \geq 0 \text{ for all } i = 1, \ldots, n$$

Let us eliminate ξ_is from the objective function.

$\xi_i \geq 1 - y_i(\mathbf{w}\cdot\bar{x_i})$, and
$\xi_i \geq 0$

imply

$\xi_i = \max(0, 1 - y_i(\mathbf{w}\cdot\bar{x_i})$ (i.e., ξ_is are computed via hinge loss function!)

We can now substitute ξ_is for the hinge loss expression in the objective function:

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}\cdot\bar{x_i})$$

When $y_i(\mathbf{w}\cdot\bar{x_i} \geq 1$, the hinge loss is 0, and the penalty is not assessed, therefore, we only need to assess the penalty when $y_i(\mathbf{w}\cdot\bar{x_i} < 1)$:

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{y_i(\mathbf{w}\cdot\bar{x_i})<1}(1 - y_i(\mathbf{w}\cdot\bar{x_i}))$$

To solve this problem using stochastic gradient descent, we need to compute partial derivatives. This requires some care, because the hinge loss function is not differentiable at $x = 1$. We need to write the partial derivative out as follows:

$$\frac{\partial\frac{1}{2}\|\mathbf{w}\|^2}{\partial w_j} = \frac{\partial\frac{1}{2}\sum_{i=1} dw_i^2}{\partial w_j} = w_j$$

Let $\bar{x} = (a_1, \ldots, a_d)$ For the hinge loss function $L(\bar{x}, y) = \max(0, 1-y(\mathbf{w}\cdot\bar{x}))$, the partial derivative can be built as follows:

[3] In general, there are two types of objective functions to optimize, the *hinge loss* function, which uses the hinge-loss penalties ξ_i, and the *quadratic loss* function, which uses the squares of hinge-loss penalties. We concentrate on the hinge loss optimization here.

$$\frac{\partial L}{\partial w_j} = \begin{cases} 0 & \text{if } y_i \mathbf{w} \cdot \bar{x} \geq 1 \\ -ya_j & \text{otherwise} \end{cases}$$

The overall partial derivatives are:

$$\frac{\partial J}{\partial w_j} = w_j + C \sum_{y_i \mathbf{w} \bar{x}_i < 1} (-y_i \cdot a_{ij})$$

For Stochastic Gradient Descent we must pick

- C: the misclassification penalty multiplier. Large values of C minimize the number of misclassifications, but also cause the margin δ^* to be small. Smaller values of C will yield more misclassifications, but will also allow for the margin δ^* to be larger. Larger margins means most points are relatively far away from the hyperplane, therefore most points are easier to classify.
- η: the learning rate.
- initial values for $\mathbf{w}$, including the intercept (bias) b.

Run Gradient Descent:

- Compute $\frac{\partial J(\mathbf{w})}{\partial w_j}$
- adjust $w_j \longleftarrow w_j - \eta \frac{\partial J(\mathbf{w})}{\partial w_j}$

Alternatively, when n is very large, perform Stochastic Gradient Descent.

References

[1] Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman, *Mining of Massive Datasets*, 2nd Edition, Cambridge University Press, 2014.

[2] Mohammed J. Zaki, Wagner Meira Jr., *Data Mining and Analysis: Fundamental Concepts and Algorithms*, Cambridge University Press, 2014.