Lecture Notes on Support Vector Machines
Advanced Machine Learning

Prof: Yannet Interian

February 15, 2017

Required Reading: Bishop 6.2 (p294-297), Bishop Appendix E. Bishop 7.1 (p335-338)

Optional Reading: Andrew Ng notes
`http://cs229.stanford.edu/notes/cs229-notes3.pdf`

1 Introduction

Based on Bishop (Chapter 7), ESL Hastie, Tibshirani and Friedman, SVM lecture by Pedro Domingos, lecture notes by Andrew Ng.

In this lecture, we discuss support vector machines (SVMs), an approach for classification that was developed in the theoretical computer science community in the 1990s. SVMs have been shown to perform well for a variety of problems, and are often considered one of the best "out of the box" classifiers.

The training dataset comprises of N input vectors $x_1, \ldots, x_N$ with the corresponding target values $y_1, \ldots, y_N$ where $y_i \in \{-1, 1\}$.

What is a support vector machine?

- A subset of the training examples $\{(x_i, y_i)\}$ where $i \in S$ (support vectors)

- A vector of weights w_i for $i \in S$ and w_0
- A similarity function $K(x, x')$ (The kernel)

Given a new example x, we predict the class for x by

$$\hat{y}(x) = \text{sign}\left(w_0 + \sum_{i \in S} \alpha_i y_i K(x, x_i)\right)$$

where $\text{sign}(x) = 1$ if $x > 0$ and -1 otherwise.

You may have a few questions:

- How do we find the support vectors?
- How do we define the weights α_i and w_0
- How do we define K?
- How we fit the data to this model?
- What is the loss/error function?
- What is linear SVM and Kernel SVM?
- What is the relationship between SVM and other techniques such us logistic regression?

In the rest of the lecture we will discuss these questions.

1.1 Review of logistics regression

For $y \in \{-1, 1\}$

$$P(y = 1|x, w, w_0) = \frac{1}{1 + e^{-(w_0 + w \cdot x)}}$$

$$P(y=-1|x,w,w_0) = 1 - \frac{1}{1+e^{-(w_0+w\cdot x}} = \frac{1}{1+e^{w_0+w\cdot x}}$$

To find the optimal w, w_0 we minimize the negative log likelihood. How do we get the likelihood?

$$\mathcal{L}(w,w_0) = \prod_{i=1}^{N} \left(\frac{1}{1+e^{-(w_0+w\cdot x_i)}}\right)^{\mathbb{1}_{y_i=1}} \left(\frac{1}{1+e^{w_0+w\cdot x_i}}\right)^{\mathbb{1}_{y_i=-1}} = \prod_{i=1}^{N} \frac{1}{1+e^{-y_i(w_0+w\cdot x_i)}}$$

Therefore, the negative log likelihood has the following form

$$\mathcal{NLL}(w,w_0) = \sum_{i=1}^{N} \log(1+e^{-y_i(w_0+w\cdot x_i)})$$

To estimate w, w_0 for logistic regression with regularization we solve the following problem

$$\min_{w,w_0} \mathcal{NLL}(w,w_0) + \lambda||w||^2$$

New examples are classified by $\text{sign}(w_0 + w \cdot x)$

Let's define the log-loss as

$$L_{nll}(y,\eta) = log(1+e^{-y\eta})$$

The intuition is the following. You want to find the best line $(w_0 + w \cdot x)$ that separates your positives and negatives (daw a diagram). Suppose you have one line: $w_0 + w \cdot x$. If $w_0 + w \cdot x > 0$ you predict $+1$ and otherwise you predict -1. If you want to quantify the errors that this line if making if $y(w_0 + w \cdot x) \geq 0$ the prediction is correct so your error is 0 on the other hand when $y_i(w_0 + w \cdot x) < 0$ you are making an error. You want a loss function that penalizes $y(w_0 + w \cdot x) < 0$ and you want that loss function to be convex to make it easier to optimize. Logistic regression is using L_{nll} as the loss function where $\eta = w_0 + w \cdot x$.

1.2 Relationship between SVM and logistic regression

Claim: (Linear) SVM are solving a very similar problem than logistic regression. Instead of using the log-loss L_{nll} they use the hinge loss L_{hinge} defined below.

$$L_{hinge}(y, \eta) = \max(0, 1 - y\eta)$$

TODO: add figure 7.5 from Bishop. add more discussion here on the differences between different loss functions.

What does it mean? In linear SVMs we are trying to find a hyperplane that separates the positive from the negative examples like in logistic regression but we are going to be using a different optimization function to fit the parameters.

$$\min_{w,w_0} C \sum_{i=1}^{N} \max(0, 1 - y_i(w_0 + w \cdot x_i)) + \frac{1}{2}||w||^2 \tag{1}$$

A new example x is going to be classified using the equation $\text{sign}(w_0 + w \cdot x)$ Note that C is equivalent to $\frac{1}{\lambda}$, they are both regularization parameters. What happens as C increases / decreases?

1.3 SVM as a Constrain Optimization

The SVM optimization function is non-differentiable because of the max term. We can transform the problem into a constrain optimization problem.

Let $\epsilon_i = max(0, 1 - y_i(w_0 + w \cdot x_i))$ then the following holds

$$\begin{aligned} \epsilon_i &\geq 0 \\ \epsilon_i &\geq 1 - y_i(w_0 + w \cdot x_i) \\ y_i(w_0 + w \cdot x_i) &\geq 1 - \epsilon_i \end{aligned}$$

So we can rewrite equation (**??** as the following optimization.

$$\min_{w,w_0} C\sum_{i=1}^{N}\epsilon_i + \frac{1}{2}||w||^2 \tag{2}$$
$$\text{subject to the constrains (st)}$$
$$\epsilon_i \geq 0$$
$$y_i(w_0 + w\cdot x_i) \geq 1-\epsilon_i \text{ for } i = 1,\ldots,N$$

This is a quadratic program with $N + D + 1$ variables subject to $O(N)$ constrains. Here D is the dimension of the input variables x.

1.4 Review of Constrain Optimization

Let's look at a simpler example to get some intuition.

$$\min_{x_1,x_2} x_1^2 + x_2^2$$
$$\text{st}$$
$$x_1 + x_2 = 1$$

In order to solve this problem we have to write a "Lagrangian".

$$\mathcal{L}(x_1, x_2, \alpha) = x_1^2 + x_2^2 + \alpha(x_1 + x_2 - 1)$$

a necessary condition is that

$$\nabla\mathcal{L}(x_1, x_2, \alpha) = 0$$

so we get

$$2x_1 + \alpha = 0$$
$$2x_2 + \alpha = 0$$
$$x_1 + x_2 = 1$$

TODO: add figure

In this case the equations are easy to solve and we get $x_1 = x_2 = \frac{1}{2}$ and $\alpha = -1$.

x_1 and x_2 are called primal variables, α is called a dual variable. In general we get as many dual variables as the number of constrains. If you eliminate the primal variables you end up with a dual problem which is often easier to solve.

Now, let's consider a more general optimization problem.

$$\begin{aligned} &\min_{x \in X} F(x) \\ &st \\ &g_i(x) \leq 0 \text{ for } i = 1, \ldots, m \end{aligned}$$

We can define the Lagrangian as

$$\mathcal{L}(x, \alpha) = F(x) + \sum_{i=1}^{m} \alpha_i g_i(x)$$

where α_i are known as lagrange or dual variables.

Theorem 1.1 *(Karush-Kuhn-Tucker's theorem.) Assume F, g_i are convex and differentiable and that the constrains are qualified. Then $\bar{x}$ is a solution of the constrained program if and only if there exits $\bar{\alpha} \geq 0$ such that,*

$$\begin{aligned} &\nabla_x \mathcal{L}(\bar{x}, \bar{\alpha}) = \nabla_x F(\bar{x}) + \bar{\alpha} \cdot \nabla_x g(\bar{x}) = 0 \\ &\nabla_\alpha \mathcal{L}(\bar{x}, \bar{\alpha}) = g(\bar{x}) \leq 0 \\ &\bar{\alpha} \cdot g(\bar{x}) = \sum_{i=1}^{m} \bar{\alpha}_i g(\bar{x}_i) = 0 \end{aligned}$$

TODO: define constrains are qualified.

1.5 Equivalence of the two formulations of SVMs

Let's show that equations (2) are give us solution to the form of the one in the introduction to the class. In particular let's prove the following:

1. $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
2. Predictions have the formula
$$y(x) = \text{sign}(w_0 + \sum_{i=1}^{n} \alpha_i y_i x_i \cdot x)$$
3. $\alpha_i > 0$ if and only if $y_i(x_i \cdot w + w_0) = 1 - \epsilon_i$.

Note that the last claim means that if $y_i(x_i \cdot w + w_0) > 1 - \epsilon_i$ then $\alpha_i = 0$ and therefore that vector is not going to be a support vector. Therefore points away from the boundary are not going to be support vectors.

We can write our primal problem as:

$$\begin{aligned}
&\min_{w,w_0} C \sum_{i=1}^{N} \epsilon_i + \frac{1}{2} \sum_{j=1}^{D} w_j^2 \\
&\text{st} \\
&-\epsilon_i \leq 0 \\
&1 - \epsilon_i - y_i(w_0 + w \cdot x_i) \leq 0 \text{ for } i = 1, \ldots, N
\end{aligned}$$

We can write our Lagrangian

$$\begin{aligned}
\mathcal{L}(\epsilon, w, w_0, \alpha, \widetilde{\alpha}) = C \sum_{i=1}^{N} \epsilon_i + \frac{1}{2} \sum_{j=1}^{D} w_j^2 - \sum_{i=1}^{N} \widetilde{\alpha}_i \epsilon_i \\
+ \sum_{i=1}^{N} \alpha_i (1 - \epsilon_i - y_i(w_0 + w \cdot x_i))
\end{aligned}$$

2 Kernels

This part come mostly from Andrew Ng's lecture notes. `http://cs229.stanford.edu/notes/cs229-notes3.pdf`

One way to obtain non-linear boundaries using linear classifiers such us logistic regression is using original features together with powers of that feature, say x, x^2, x^3. To distinguish between these two sets of variables, we'll call the "original" input value the input attributes of a problem (in this case, x). When that is mapped to some new set of quantities that are then passed to the learning algorithm, we will call those new quantities the input features.

Let denote ϕ the feature mapping, which maps from the attributes to the features. For instance, in our example, we had

$$\phi(x) = (x, x^2, x^3)$$

Rather than applying SVMs using the original input attributes x, we may instead want to learn using some features $\phi(x)$. To do so, we simply need to go over our previous algorithm, and replace x everywhere in it with $\phi(x)$. Since the algorithm can be written entirely in terms of the inner products $x \cdot x'$ this means that we would replace all those inner products with $\phi(x) \cdot \phi(x')$. Given a feature mapping ϕ, we define the corresponding Kernel to be

$$K(x, x') = \phi(x) \cdot \phi(x')$$

Then, everywhere we previously had $x \cdot x'$ in our algorithm, we could simply replace it with $K(x, x')$, and our algorithm would now be learning using the features ϕ. Given ϕ, we could easily compute $K(x, x')$ by finding $\phi(x)$ and $\phi(x')$ and taking their inner product. But what's more interesting is that often, $K(x, x')$ may be very inexpensive to calculate, even though $\phi(x)$ itself may be very expensive to calculate (perhaps because it is an extremely high dimensional vector). In such settings, by using in our algorithm an efficient way to calculate $K(x, x')$ we can get SVMs to learn in the high dimensional feature space space given by ϕ , but without ever having to explicitly find or represent vectors $\phi(x)$.

Let's see an example. Suppose $x, z \in \mathbb{R}^n$, and consider

$$K(x,z) = (x \cdot z)^2.$$

We can also write this as

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{i=1}^{n} x_i z_i\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i z_i x_j z_j \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

Thus, we can see that $K(x,z) = \phi(x) \cdot \phi(z)$, where the feature mapping ϕ is given (shown for the case $n = 3$)by

$$\phi(x) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$$

Note that calculation $\phi(x)$ in this example requires $O(n^2)$, while calculation $K(x,z) = x \cdot z$ requires $O(n)$,

A Kernel that is very of often used in practice is the **Gaussian kernel**.

$$K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$$

Note that when x is close to z $K(x,z)$ is close to 1 and when x and z are far apart $K(x,z)$ is close to 0. This Kernel defines a reasonable measure of similarity. How do we show that this is a valid Kernel for some function ϕ? Note that σ is a parameter for this Kernel. How do we choose σ? Suppose your SVM is overfitting, do you increase or decrease σ? How is σ affecting the decision boundary?

Here is an example of Kernel application. Consider the digit recognition problem, in which given an image (16x16 pixels) of a handwritten digit (0-9), we have to figure out which digit it was. Using either a simple polynomial

kernel $K(x, z) = (x \cdot z)^d$ or the Gaussian kernel, SVMs are able to obtain extremely good performance on this problem. This was particularly surprising since the input attributes x were just a 256-dimensional vector of the image pixel intensity values, and the system had no prior knowledge about vision, or even about which pixels are adjacent to which other ones.

2.1 Scaling issues using the Gaussian Kernels

Because the Gaussian Kernel is a function of the euclidean distance feature vectors should be scaled / normalized before using the Gaussian Kernel. To see what this is a problem, suppose that we what to predict whether a house is going to sell within a week at a particular price and we have the following features: number of rooms, size in square foots, price in thousands of dollars etc ... Note that since features are in very different scales.

$$house_1 : \text{ 2 rooms, 900sf, 300,000}$$
$$house_2 : \text{3 rooms, 1200sf, 700,000}$$

$||house_1 - house_2||^2 = 1 + 300^2 + 400000^2$ As you can see in this example, without scaling some features are going to have a disproportional influence with respect to other features.

There are many ways to normalize a vector. One way to do it it to compute the sample mean and sample variance from the training set and normalize each feature x with $\frac{x-\bar{x}}{\bar{\sigma}}$. You should perform the exact same transformation on your test set features as you do on features from your training set.

3 Maximal Margin Classifier

From Foundations of Machine Learning Chapter 4.

The support vector machine is a generalization of the *maximal margin classifier*. As we will see the maximal margin classifier is a simple and elegant approach that can be applied to datasets that are separable by a linear boundary. The support vector machines are an extension of maximal margin classifiers that can be applied to non-separable datasets.

3.1 SVMs – separable case

In this section, we assume that the training sample can be linearly separable. That is, we assume the existence of a hyperplane that perfectly separates the positive and negative labeled points. Which hyperplane should the algorithm select? The SVM solution is the hyperplane with the maximum *margin* (or the distance to the closest points in the training set).

TODO (attach a figure here)

The general equation of a hyperplane is

$$w \cdot x + w_0 = 0$$

where w is a non-zero vector normal to the hyperplane and w_0 is a scalar.

We can scale w and w_0 such that $min_{(x,y)\in T}|w \cdot x + w_0| = 1$, where T is the set of training points. We can define this representation of the hyperplane as the *canonical hyperplane* with respect to T.

Lemma 3.1 *The distance from any point x_0 to the hyperplane is given by*

$$\frac{|w \cdot x_0 + w_0|}{||w||}$$

For a proof see `https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line`

To see this first assume $w_0 = 0$ and note that the distance between x_0 and the hyperplane is the projection of x_0 into w (seeing them as vector). To prove it for w_0 different than 0 is just a change of coordinates.

Thus, for a canonical hyperplane, the margin ρ is given by

$$\rho = min_{(x,y)\in T}\frac{|w \cdot x + w_0|}{||w||} = \frac{1}{||w||}$$

TODO (attach a figure of here with marginal hyperplanes)

Marginal hyperplanes are hyperplanes parallel to the separating hyperplane and passing through the closest points on the negative or positive side. The equations of the maximal hyperplane are $w \cdot x + w_0 = \pm 1$.

A hyperplane defined by w, w_0 correctly classifies a training point $(x_i, y_i) \in T$ when $w \cdot x_i + w_0$ has the same sign as y_i. For canonical hyperplane, by definition we have $|w \cdot x_i + w_0| \geq 1$ for all $(x_i, y_i) \in T$. Thus x_i is correctly classified if $y_i(w \cdot x_i + w_0) \geq 1$. In view of equation ..., maximizing the margin while correctly classifying all training points, can be expressed as the solution of the following convex optimization problem:

$$\begin{aligned} &\min_{w,w_0} \frac{1}{2}||w||^2 \\ &\text{subject to: } y_i(w \cdot x_i + w_0) \geq 1, \forall (x_i, y_i) \in T \end{aligned}$$

This is a quadratic programming problem that admits a unique solution.

3.2 SVMs – non separable case

In most practical settings, the training data is not linearly separable, that is for any hyperplane $w \cdot x + w_0 = 0$, there exists $x_i \in T$ such that

$$y_i(w \cdot x_i + w_0) < 1$$

The constrains imposed in the linearly separable case cannot all hold simultaneously. However, a relaxed version of these constrains can hold. For each point in the training set (x_i, y_i), let's define $\xi_i \geq 0$ such that

$$y_i(w \cdot x_i + w_0) \geq 1 - \xi_i$$

The variables ξ_i are known as *slack variables* and are commonly used in optimization to define relaxed versions of some constrains. Here, a slack variable measures the distance by which vector x_i violates the desired inequality, $y_i(w \cdot x_i + w_0) \geq 1$. For a hyperplane $w \cdot x + w_0 = 0$, a vector x_i with $\xi_i > 0$ can be viewed as an *outlier*.

This leads to the following general optimization problem defining SVMs in the non-separable case

$$\min_{w,w_0} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i$$
subject to:
$y_i(w \cdot x_i + w_0) \geq 1 - \xi_i$ for $i \in [1, \cdots N]$
$\xi_i \geq 0$ for $i \in [1, \cdots N]$

4 Using SVMs

This is based on Advised from Andrew Ng (Stanford, Coursera).

When you decide to use SVMs to solve your classification problems. You need to decide at least two things:

- Type of Kernel
- Value of the regularization parameter C.

Which Kernel should I use?

- SVM with no Kernel: When you have a huge number of features and a low number of points in your training set you should use a linear SVM (SVM with no Kernel).
- Gaussian Kernel: Here we have to choose the value of σ^2. If your feature set are numerical, the number of features is not too large and you have a large training set. Also you problem is clearly not linearly separable. You may want to use a Gaussian Kernel to learn a non-linear decision boundary.

Logistics versus SVMs?
Let's D be the number of features and N the number of training examples

- If D is large relative to N, use logistic regression or SVM without a Kernel. For example a text classification problem with 10k features (words), 1k examples. Here you don't have enough data to fit a non-linear boundary.
- If D is small and N is intermediate, use SVM with a Gaussian Kernel (D between 1-1k , N up to 50k).
- If the number of examples is large N larger than 50k or 100k, the SVM packages are going to struggle. Here use linear SVM or logistic regression. If D is small you can create manual features.
- A well designed Neural Networks is likely to work well for many of these settings but they are slower to train than SVMs. (This may have changed)

5 Example

Suppose you have the following training set.

$$
\begin{aligned}
x_1 &= (-1, 0), y_1 = +1 \\
x_2 &= (-1, 1), y_2 = +1 \\
x_3 &= (2, 0), y_3 = -1 \\
x_4 &= (3, 0), y_4 = -1 \\
x_5 &= (4, 0), y_5 = -1
\end{aligned}
$$

(a) Find the lagrangian coefficient for all the points.
(b) Find w and w_0.

To solve this problem it is useful to draw the points in a two dimensional plot. First note that this dataset is separable and that the minimum distance between any two points with opposite labels is 3. Therefore the margin is smaller or equal to 1.5 (why?). Denote (x^1, x^2) the coordinates of the plane. The line $x^1 = 0.5$ has margin 1.5, so this line is our SVM solution and therefore $\alpha_4 = 0$ and $\alpha_5 = 0$. Also the equations for the marginal hyperplane are $x^1 = -1$ and $x^1 = 2$.

Using equations for the separating hyperplane $w \cdot x + w_0 = 0$ we get that there exits a non-zero constant c such that if $w = (w_1, w_2)$ then $w_2 = 0$, $w_1 = c$ and $w_0 = -0.5c$.

Using the equations from the marginal hyperplane $y_i(w \cdot x + w_0) = 1$ we get that $c = 2$.

To get the alphas we need to solved these equations.

$w_1 = \sum_{i=1}^{N} \alpha_i y_i x_i^1$ and

$w_2 = \sum_{i=1}^{N} \alpha_i y_i x_i^2$