

## Lecture 11 — Feb 13

Scribe: Dennis Park, Pornpat Nikamanon

Lecturer: Deva Ramanan

**Note:** These lecture notes are still rough, and have only have been mildly proofread.

## 11.1 Soft Margin SVM

### 11.1.1 Recall

To control the sensitivity of SVM to possible outliers, we introduced *slack* variables  $\xi$ 's, and now we have slightly different form of max-margin problem.

$$\begin{aligned} \min \quad & \frac{1}{2}\|w\|^2 + C \sum \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad i \end{aligned} \tag{11.1}$$

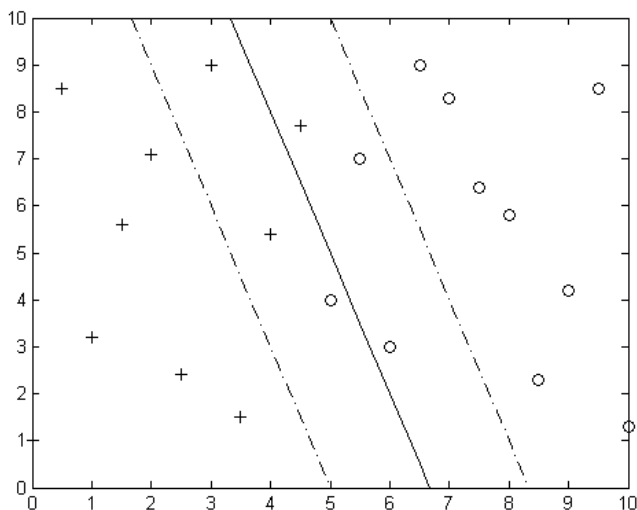
Whereas the margin was defined by hard constraint before, now we allow some amount of slackness in constraint which is represented by  $\xi$ 's. By definition,  $\xi$ 's have to be larger or equal to 0, so that it really means *slackness*. Also, we can observe that if  $0 < \xi \leq 1$ , it means the data point lies somewhere between the the margin and the correct side of hyperplane, and if  $\xi > 1$ , it means the data point is misclassified.

Note the problem can also be written as a single minimization without constraints:

$$\min_w \quad \frac{1}{2}\|w\|^2 + C \sum_i \max(0, 1 - y^{(i)}(w^T x^{(i)} + b)) \tag{11.2}$$

Each given data point  $x^{(i)}$  falls in one of three categories. It either lies beyond the margin  $y^{(i)}(w^T x^{(i)} + b) > 1$ , in which case it does not contribute to the loss in (11.2). It could lie directly on the margin  $y^{(i)}(w^T x^{(i)} + b) = 1$ , in which case it doesn't directly add to the loss, but participates in the optimization as a support vector - similar to the hard margin case we previously described. If the point lies within the margin, we add a penalty to the cost function that is proportional to the amount by which each data point is violating the hard constraint.

We are given a free parameter  $C$  that controls the relative importance of minimizing the norm of  $w$  (which is equivalent to maximizing the margin) and satisfying the margin constraint for each data point.



If  $C$  is close to 0, then we don't pay that much for points violating the margin constraint. We can minimize the cost function by setting  $w$  to be a small vector - this is equivalent to creating a very wide "tube" or safety margin around the decision boundary (but having many points violate this safety margin - see Figure 11.1.1). If  $C$  is close to  $\inf$ , then we pay a lot for points that violate the margin constraint, and we are close the hard-margin formulation we previously described - the difficulty here is that we may be sensitive to outlier points in the training data.

### 11.1.2 Lagrangian Dual

As before, we can construct the Lagrangian of (11.1) and solve for the dual. Now we have additional constraints,  $\xi_i \geq 0$ . This means we will require additional dual variables  $r_i$  corresponding to those constraints when constructing the full Lagrangian.

#### Duality - notes

We will now go over some additional terms and clarifications for Lagrangian duality - these will be needed for the homework. Recall that we are trying to minimize  $f(w)$  such that for all  $i$ ,  $g_i(w) \leq 0$ . The original formulation is equivalent to solving  $p^* = \min_w \max_{\alpha \geq 0} L(w, \alpha)$ , where  $L(w, \alpha) = f(w) + \sum \alpha_i g_i(w)$ . We define the dual as  $d^* = \max_{\alpha \geq 0} \min_w L(w, \alpha)$ . For any  $f(w)$  and  $g_i(w)$ ,  $d^* \leq p^*$ . We call this property *weak duality* - this was proven in the lecture 9 notes.

In certain cases,  $d^* = p^*$ . This is called strong duality. This holds if certain conditions are met. For example, strong duality holds if  $f(w)$  is convex and  $g_i(w)$  are affine. This is the case for both hard and soft-margin SVMs. Strong duality also holds if  $f(w)$  is convex and  $g_i(w)$  is convex, and there exists a feasible  $w$  (that satisfies all the  $g_i(w)$  constraints).

This condition for strong duality will be used in the homework.

### Soft-margin Lagrangian

Let us define the Lagrangian for a soft-margin SVM as

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}\|w\|^2 + C \sum_i \xi_i - \sum_i \alpha_i \{y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i\} - \sum_i r_i \xi_i$$

We now can say that solving the original constrained optimization (11.1) is equivalent to computing  $p^*$

$$p^* = \min_{w, b, \xi} \max_{\substack{\alpha \geq 0 \\ r \geq 0}} \mathcal{L}(w, b, \xi, \alpha, r) \quad (11.3)$$

$$d^* = \max_{\substack{\alpha \geq 0 \\ r \geq 0}} \min_{w, b, \xi} \mathcal{L}(w, b, \xi, \alpha, r) \quad (11.4)$$

We strong duality,  $d^* = p^*$ , and so we can solve the original problem by solving the dual (11.4). We first solve the inner minimization problem of  $d^*$  by taking the partial derivative of  $\mathcal{L}$  w.r.t.  $w, b, \xi$ .

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial \xi} = 0$$

which result in :

$$\begin{aligned} w &= \sum \alpha_i y^{(i)} x^{(i)} \\ \sum \alpha_i y^{(i)} &= 0 \\ \alpha_i &= c - r_i \quad \forall i \end{aligned} \quad (11.5)$$

As we have seen in the previous lecture, the dual solution also have to satisfy KKT condition. In this case the KKT condition becomes :

$$\begin{aligned} \alpha_i \{y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i\} &= 0 \\ r_i \xi_i &= 0 \end{aligned} \quad (11.6)$$

Plugging (11.2) back into original problem (11.1), we obtain

$$\max_{\alpha \geq 0} \mathcal{L}(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} \quad (11.7)$$

$$\text{s.t.} \quad \sum_i \alpha_i y^{(i)} = 0 \quad (11.8)$$

$$0 \leq \alpha_i \leq c \quad \forall i \quad (11.9)$$

The dual soft-margin SVM has a quadratic objective that is equivalent to the hard-margin case. The only difference is that there is now a *box constraint* on  $\alpha$ 's. The influence of any point  $x^{(i)}$  on the the weight vector  $w$  is capped by  $C$ , while in the hard-margin case, it was unbounded. This reduces the influence of outliers in the final solution.

The KKT condition (11.6) can be rewritten as

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1 && \text{"easy"} \\ \alpha_i = C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1 && \text{"hard"} \\ 0 < \alpha_i < C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1 && \text{"marginally hard"} \end{aligned}$$

Again,  $\alpha_i$  is non-zero only for the support vectors. However, support vectors now include not only the data points on the margin but also the data points within the margin and those on the wrong side of boundary.

## 11.2 SMO (Sequential Minimal Optimization)

Now the dual problem we have to solve is represented by (11.7). This is a quadratic equation in terms of all  $\alpha_i$ 's, and we might want to solve the problem using some constraint-based form of gradient-descent or Newton Rhapsod. Instead, a common approach for SVMs is to use coordinate descent. The idea here is to fix a whole bunch of variables, and only optimize a subset. In the extreme case, we only optimize a single variable, freezing all others to their current value. We then pick a different variable to optimize, and repeat the procedure until we can no longer improve the objective. The advantage here is that optimizing a single variable, while holding the others fixed, could be computationally convenient.

The SMO algorithm attempts to optimize the dual objective by choosing a single  $\alpha_i$  to optimize at a time. However, because we need to satisfy the balance constraint (11.8), we actually need to select two  $\alpha$ s at a time.

The algorithm is following :

1. Start with feasible set of  $\alpha$ 's that satisfy the dual constraints (i.e.  $\alpha_i = 0$ ).
2. Pick an  $\alpha_i$  and  $\alpha_j$  that we think will improve the objective (11.7).
  - Many heuristics have been proposed for picking these points. Such heuristics try to exploit the sparseness property of SVMs. Ideally, we should spend most of the time optimizing over points near the boundary, so these heuristics tend to pick "hard" or "marginally hard" points.
  - A reasonable heuristic is to pick points that violate the KKT conditions. This is reasonable because we know that at the final solution, all the KKT conditions must be satisfied. (i.e. if  $y^{(i)}(w^T x^{(i)} + b) < 1$  and  $\alpha_i = 0$ , we're in violation.)
3. We now optimize (11.7), subject to the constraints, with respect to  $\alpha_i$  and  $\alpha_j$ . We can do this in closed form. Assume that  $i = 1, j = 2$ . Recall the balance constraint

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} + \sum_{i \geq 3} \alpha_i y^{(i)} = 0 \quad (11.10)$$

$$\alpha_1 = \frac{1}{y^{(1)}} (S - y^{(2)} \alpha_2) \text{ where } S = \sum_{i \geq 3} \alpha_i y^{(i)}. \quad (11.11)$$

Hence the balance constraint is just a linear constraint on  $\alpha_1$  and  $\alpha_2$ . Plugging (11.11) into (11.7) and fixing all other  $\alpha$ 's, we obtain a quadratic expression in  $\alpha_2$ . In order to enforce the box constraints for both  $\alpha_1$  and  $\alpha_2$ , we have to introduce additional bounds on  $\alpha_2$  - they will be of the form  $0 \leq L \leq \alpha_2 \leq H \leq C$  - see the paper by Pratt (note  $L$  is just some lower bound, not the Lagrangian). However, its easy to minimize a quadratic function of one variable subject to such bound constraints. Apply the quadratic formula to compute the global minimum, and then clip the answer (if necessary) to fall between the bounds  $L$  and  $H$ .

## 11.3 Decision Trees

### 11.3.1 Motivation

Given a a set of training points  $\{x^{(i)}, y^{(i)}\}$ , we may want to learn a non-linear predictor of  $y$  given  $x$ . For example, given boolean points in two dimensions  $x \in \{0, 1\}^2$ , we are given training data consistent with  $h(x) = x_1 \oplus x_2$  (see Figure 11.1). This hypothesis cannot be represented with a linear decision boundary.

To fit the data set, we can think of a sequential decision process. We start by looking at one feature ( $x_2$ ) and ask if  $x_2$  **is zero or one**. If it is zero, we go to one branch of a decision tree, and if it is one, we go to another branch. We next ask a question  $x_1$  **is zero or one**. We will finally reach the *leaf* of the tree, which will be labelled with a  $y$  value of zero or one. For example, if  $x_2$  and  $x_1$  are zero we have output as zero. Each branch decision is made with a “mini-classifier”, or decision-stump.

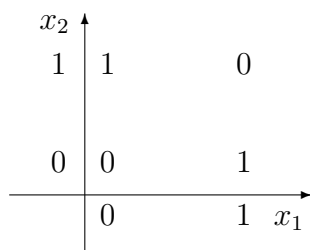
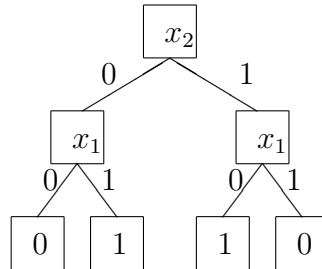


Figure 11.1.

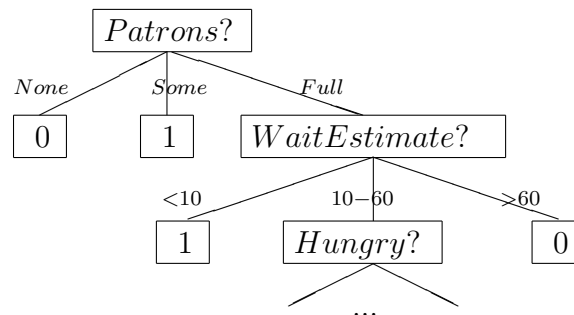
**Note:**

- 1) The same attribute ( $x_i$ ) can reappear in a tree (i.e.  $x_1$  )
- 2) Each node in a tree is called a **decision stump** (simple classifier for one task)

Example of decision tree: You want to decide **should we eat at a restaurant?**

The first question is are there patrons in this restaurant? If no one there, we should not eat there. If someone there, we will step in this restaurant, but if it is full, we will decide on how long it take to wait? If it is less than ten minute, we will eat there. If it is greater than an hour, we will not eat there, but if it is between ten minute and one hour, we will decide on if we are hungry or not.

You can think of input feature in this case as three dimensional vector  $x^{(i)}$  which contains a boolean values representing if patrons are there, the state of our hunger, and a continuous value representing the wait estimate.



### 11.3.2 Expressiveness

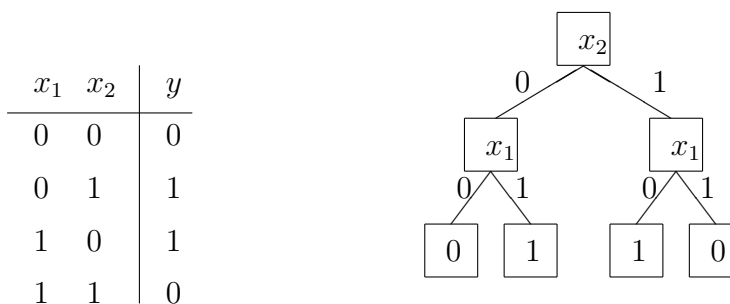
The decision trees are expressive. They can represent arbitrary boolean functions. We can write out the truth table which tabulates all possible inputs of the function and the output of the function for each possible input. We can construct a decision tree such that there exists a leaf node for every entry in this table.

For example  $h(x) = x_1 \oplus x_2$  can express in the table below. Each row can make this its own path from root to "leaf"

One natural question is what is the size of a tree necessary to represent an arbitrary boolean function of  $n$  variables. We know the number of leaves must be equal to the number

of entries in the table, or  $2^n$ . We can also ask, how many such arbitrary boolean functions of  $n$  variables exist. This is  $2^{2^n}$ . This is the number of decision trees one could build given a  $n$ -dimensional boolean input  $x^{(i)}$ .

Imagine given a dataset, we construct 2 trees that correctly label all the training data  $\{x^{(i)}, y^{(i)}\}$ , but one tree has significantly fewer nodes. We expect its decision rules are more general. Hence, one criteria for good trees is to find small trees that correctly label the training data.

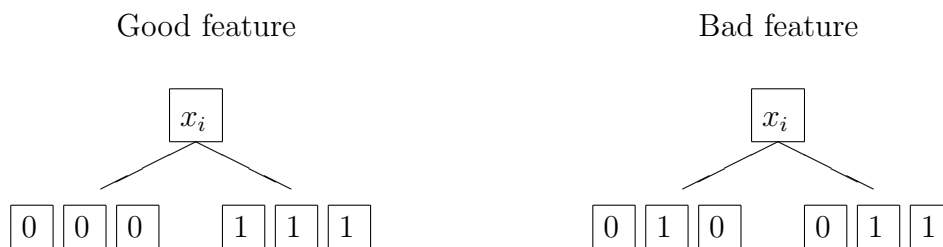


### 11.3.3 Tree-building algorithm

**Goal:** Given  $\{x^{(i)}, y^{(i)}\}$ , find a small tree that correctly classifies data

**Idea:** Recursively choose & expand most significant feature

**A good feature** is one that, if we split the training data according to this feature, all the data in one side have  $y = 0$  and all data in other side have  $y = 1$ . Intuitively, the split results in a good classifier. **A bad feature** has mixed output labels on both side.



### 11.3.4 Entropy

To put a mathematical expression for the goodness intuition above, we define entropy  $H$  of a random variable  $X$ .

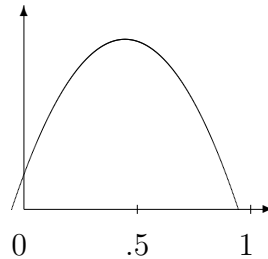
$$H(X) = - \sum_x P(X = x) \log P(X = x) \quad (11.12)$$

**Entropy** is the measure of uncertainty of a random variable (related to variance).

**Entropy of a Bernoulli distribution:** To evaluate the goodness of a proposed feature split, we fit a Bernoulli model to the  $y$  labels of each group after the split. We assume  $y \sim \text{Bernoulli}(\pi)$ . We will then calculate the entropy of the Bernoulli random variable  $y$ . For any split, let  $p = \#pos$ ,  $n = \#neg$ ,  $Y \sim \text{Bernoulli}(\frac{p}{p+n})$

$$H(Y) = -\left(\frac{p}{p+n} \log\left(\frac{p}{p+n}\right) + \frac{n}{n+p} \log\left(\frac{n}{n+p}\right)\right) \quad (11.13)$$

Consider tossing an un-biased coin. If you toss the coin very often, the frequency of heads is  $\pi$  and hence the frequency of tail is  $1 - \pi$ . We can calculate  $H(Y)$  as a function of  $\pi$ . We obtain a maximum entropy, or randomness, when the coin is unbiased  $\pi = .5$ .

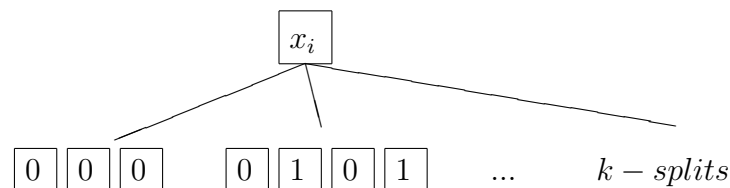


s

**Entropy of a gaussian distribution:** For a gaussian distribution, the entropy is.

$$H(X) = \frac{1}{2} n [1 + \log(2\pi)] + \frac{1}{2} \log|\Sigma| \quad (11.14)$$

**Entropy of splitting  $x_i$ :**



It is possible that by splitting a feature, one branch results in a group whose  $y$  labels have low entropy (they are mostly 1 or 0), but another branch has high entropy. In this case, we will evaluate the goodness of splitting the feature as a (weighted) average of entropies

that would result from falling into each of the branches. We weight the entropies by the probability of taking that branch, which is just the fraction of training points  $x^{(i)}$  that fall under the specified branch.

In general, assume that the feature  $x_i$  takes on  $k$  discrete values. Let  $p_j$  be the number of positives in the  $j^{\text{th}}$  group, where  $j = 1 \dots k$ . Let  $n_j$  be the number of negatives in the  $j^{\text{th}}$  group.

$$p_j = \#pos \text{ in group } j \quad (11.15)$$

$$n_j = \#neg \text{ in group } j \quad (11.16)$$

$$p = \sum p_j \quad (11.17)$$

$$n = \sum n_j \quad (11.18)$$

$$H[\text{split}(x_i)] = \sum_{j=1}^k \frac{p_j + n_j}{p + n} \text{Entropy}(\text{Bernoulli}(\frac{p_j}{p_j + n_j})) \quad (11.19)$$

The above equation tells us how to calculate the entropy of  $y$  labels that results from splitting particular feature  $x_i$ . We finally rank a feature by the decrease in entropy we obtain by splitting the training data along that feature - we call this the information gain of a feature.

$$\text{information gain} = \text{entropy before} - \text{entropy after} \quad (11.20)$$

### 11.3.5 Greedy Algorithm

In greedy algorithm, we can construct a decision tree by

- 1) Initialize root of a tree with all training data  $\{x^{(i)}, y^{(i)}\}$ .
- 2) Choose a feature to split that maximizes the information gain. Assign the training data to the corresponding branch it falls under and repeat (2).

### 11.3.6 Continuous Version

For the continuous version, we choose the feature  $x_i$  and a threshold  $x_i > \theta$ . Ideally, we would choose the feature and threshold that maximizes the information gain. In certain cases, we can do this by brute-force search over all possible thresholds  $\theta$ . Alternatively, we could choose a threshold for a given feature  $x_i$  by training a 1-d linear classifier such as logistic regression. For example, given the following two dimensional features that lie between 0 and 1, here is the corresponding decision tree:

