Advanced Machine Learning **Fall 2020**

Lecture 8: Soft Margin and Structured SVMs

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

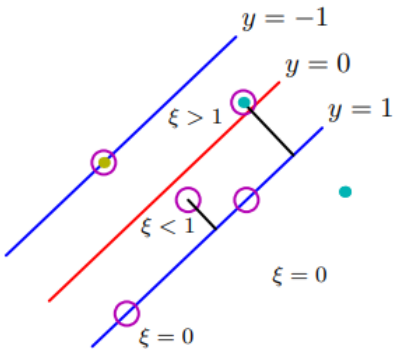
Disclaimer: *These notes are adapted from ETH's Advanced Machine Learning Course, Bishop's "Pattern Recognition and Machine Learning" book and Joachims et al.'s paper: "Predicting Structured Objects with Support Vector Machines"*

8.1 Soft Margin SVM

So far, we have assumed that the training data points are linearly separable in the feature space. In practice, however, the class conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization. We therefore need a way to modify support vector machines so as to allow some of the training points to be misclassified.

To do this, we introduce slack variables $\xi_i \geq 0$ where $i = 1, \ldots, N$. These are defined by $\xi_i = 0$ for data points that are on or inside the correct margin boundary and $\xi_i = |t_i - y(x_i)|$ for other points. Thus a data point x_i that is on the decision boundary $y(x_i) = 0$ will have $\xi_i = 1$, and points with $\xi_i > 1$ will be misclassified.

Figure 8.1: Illustration of the slack variables.



The exact classification constraints are then replaced with:

$$t_i y(x_i) \geq 1 - \xi_i \quad i = 1, \ldots, N$$
$$\xi_i \geq 0 \quad i = 1, \ldots, N$$

Our goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. We therefore minimize:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & 1 - \xi_i - t_i(w^T\phi(x_i) + b) \leq 0,\ i = 1,\ldots,N \\ & -\xi_i \leq 0,\ i = 1,\ldots,N \end{aligned}$$

The parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin (training error vs model complexity). In the limit $C \to \infty$ we will recover the "hard" margin SVM.

8.1.1 Dual Problem

It is trivial to show that Slater's condition holds since the constraints are affine functions. Therefore, strong duality holds and we can move to the Lagrangian:

$$\mathcal{L}(w,b,\xi,a,u) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}a_i\{1 - \xi_i - t_i(w^T\phi(x_i) + b)\} - \sum_{i=1}^{N}u_i\xi_i$$

Furthermore, the corresponding KKT conditions are:

$$a_n \geq 0 \tag{8.1}$$
$$t_n y(x_n) - 1 + \xi_n \geq 0 \tag{8.2}$$
$$a_n(t_n y(x_n) - 1 + \xi_n) = 0 \tag{8.3}$$
$$u_n \geq 0 \tag{8.4}$$
$$\xi_n \geq 0 \tag{8.5}$$
$$u_n\xi_n = 0 \tag{8.6}$$

We now optimize out w, b, ξ_i:

$$\frac{\partial\mathcal{L}}{\partial w} = 0 \implies w = \sum_{i=1}^{N}a_i t_i \phi(x_i)$$
$$\frac{\partial\mathcal{L}}{\partial b} = 0 \implies \sum_{i=1}^{N}a_i t_i = 0$$
$$\frac{\partial\mathcal{L}}{\partial \xi_i} = 0 \implies a_i = C - u_i$$

Using the previous results from "hard" margin SVM together with the above results, we can eliminate w, b, ξ_i from the Lagrangian. We obtain the dual Lagrangian function which is identical to the "hard" margin one, except that the constraints are somewhat different:

$$\mathcal{L}(\xi, a, u) = \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j t_i t_j \phi(x_i)^T \phi(x_j) + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i\xi_i - \sum_{i=1}^{N} u_i\xi_i$$

$$\mathcal{L}(\xi, a, u) = \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j t_i t_j \phi(x_i)^T \phi(x_j) + \overbrace{\sum_{i=1}^{N}\xi_i(a_i + u_i) - \sum_{i=1}^{N} a_i\xi_i - \sum_{i=1}^{N} u_i\xi_i}^{=0}$$

This gives the dual representation of the soft margin SVM problem:

$$\begin{aligned} \max_{a} \quad & \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j t_i t_j \phi(x_i)^T \phi(x_j) \\ \text{subject to} \quad & 0 \le a_i \le C,\ i = 1, \ldots, N \\ & \sum_{i=1}^{N} a_i t_i = 0 \end{aligned}$$

Where the first constraint comes from the fact that $a_i = C - u_i$ and $a_i \ge 0, u_i \ge 0$ must be satisfied (KKT).

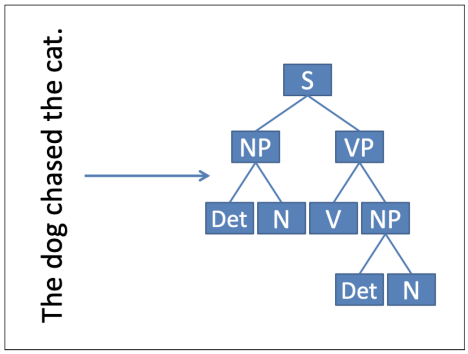
We can now interpret the resulting solution:

- data points for which $a_i = 0$: they do not contribute to the predictive model.
- data points for which $0 < a_i < C$: lie on the margin since $a_i < C \implies u_i > 0 \implies \xi_i = 0$ where the last implication comes from complementary slackness (8.6)
- data points for which $a_i = C$: lie inside the margin and can either be correctly classified if $\xi_i \le 1$ or misclassifed if $\xi_i > 1$

8.2 Structured SVMs

Consider the problem of natural language parsing illustrated in Figure 8.2. A parser takes as input a natural language sentence, and the desired output is the parse tree decomposing the sentence into its constituents. How can we take, say, an SVM and learn a rule for predicting trees?

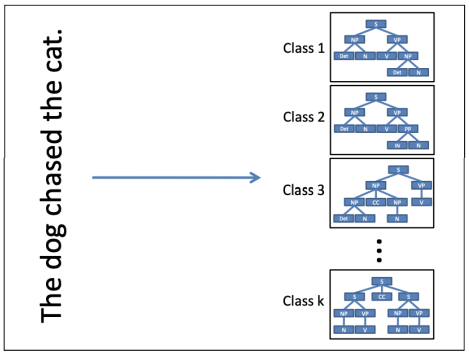
Figure 8.2: Predicting trees in natural language parsing.



Obviously, this question arises not only for learning to predict trees, but similarly for a variety of other structured and complex outputs. Structured output prediction is the name for such learning tasks, where one aims at learning a function $h : X \to Y$ mapping inputs $x \in X$ to complex and structured outputs $y \in Y$.

On an abstract level, a structured prediction task is much like a multi-class learning task. Each possible structure $y \in Y$ (e.g. parse tree) corresponds to one class (see Figure 8.3), and classifying a new example x amounts to predicting its correct "class".

Figure 8.3: Structured output prediction as a multiclass problem.



While the following derivation of structural SVMs starts from multi-class SVMs, there are three key problems that need to be overcome. **All of these problems arise from the huge number $|\mathbf{Y}|$ of classes**.

8.2.1 Problem 1: Structural SVM Formulation

We start the derivation of the structural SVM from the multi-class SVM. These multi-class SVMs use one weight vector $\mathbf{w_y}$ for each class $\mathbf{y}$. Each input $\mathbf{x}$ now has a score for each class $\mathbf{y}$ via $\mathbf{f(x, y)} = \mathbf{w_y}\phi(\mathbf{x})$. Here $\phi(\mathbf{x})$ is a vector of binary or numeric features extracted from $\mathbf{x}$. Thus, every feature will have an additively weighted influence in the modeled compatibility between inputs $\mathbf{x}$ and classes $\mathbf{y}$. To classify $\mathbf{x}$, the prediction rule $\mathbf{h(x)}$ then simply chooses the highest-scoring class:

$$h(x) = \max_{y \in Y} f(x, y) \tag{8.7}$$

as the predicted output. This will result in the correct prediction $\mathbf{y}$ for input $\mathbf{x}$ provided the weights $\mathbf{w} = (\mathbf{w_1}, ..., \mathbf{w_k})$ have been chosen such that the inequalities $\mathbf{f(x, y_i)} < \mathbf{f(x, y)}$ hold for all incorrect outputs $\mathbf{y_i} \neq \mathbf{y}$.

The first challenge in using (2) for structured outputs is that, while there is generalization across inputs $\mathbf{x}$, there is no generalization across outputs. This is due to having a separate weight vector $\mathbf{w_y}$ for each class $\mathbf{y}$. Furthermore, since the number of possible outputs can become very large (or infinite), naively reducing structured output prediction to multi-class classification leads to an undesirable blow-up in the overall number of parameters and in the overall number of inequalities, which is $\mathbf{n(k - 1)}$.

The key idea in overcoming these problems is to extract features from input-output pairs using a so-called joint feature map $\mathbf{\Psi(x, y)}$ instead of $\mathbf{\Psi(x)}$. These joint features will allow us to generalize across outputs and to define meaningful scores even for outputs that were never actually observed in the training data. At the same time, since we will define compatibility functions via $\mathbf{f(x, y)} = \mathbf{w} \cdot \mathbf{\Psi(x, y)}$, the number of parameters will simply equal the number of features extracted via $\mathbf{\Psi}$, which may not depend on $|\mathbf{Y}|$. One can then use the formulation in (2) with the more flexible definition of $\mathbf{f}$ via $\mathbf{\Psi}$ to arrive at the following (hard-margin) optimization problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2} \|w\|^2 \\ \text{subject to} \quad & w \cdot \Psi(x_i, y_i) - w \cdot \Psi(x_i, y) \geq 1, \ \forall i, y \neq y_i \end{aligned}$$

In other words, find a weight vector $\mathbf{w}$ of an input-output compatibility function $\mathbf{f}$ that is linear in some joint feature map $\mathbf{\Psi}$ so that on each training example it scores the correct output higher by a fixed margin than every alternative output, while having low complexity (i.e. small norm $\|w\|$). Note that the number of linear constraints is still $n(|Y| - 1)$. The design of the features $\mathbf{\Psi}$ is problem-specific, and it is a strength of the developed methods to allow for a great deal of flexibility in how to choose it.

8.2.2 Problem 2: Inconsistent Training Data

So far we have tacitly assumed that the optimization problem has a solution, i.e. there exists a weight vector that simultaneously fulfills all margin constraints. In practice this may not be the case, either because the training data is inconsistent or because our model class is not powerful enough. If we allow for mistakes, though, we must be able to quantify the degree of mismatch between a prediction and the correct output, since usually different incorrect predictions vary in quality. This is exactly the role played by a loss function, formally $\Delta : Y \times Y \to \mathbb{R}$, where $\mathbf{\Delta(y_i, y)}$ is the loss (or cost) for predicting $\mathbf{y}$, when the correct output is $\mathbf{y_i}$.

Like the choice of $\mathbf{\Psi}$, defining $\mathbf{\Delta}$ is problem-specific. One can convert this back into a quadratic program as follows:

$$\begin{aligned} \min_{\mathbf{w},\xi_i \geq 0} \quad & \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & w \cdot \Psi(x_i, y_i) - w \cdot \Psi(x_i, y) \geq \Delta(y_i, y) - \xi_i, \ \forall i, y \neq y_i \end{aligned}$$

Note that we added the $\mathbf{\frac{1}{n}}$ term, this is a guarantee that we are minimizing the empirical risk:

Theorem 8.1 *If w^*, ξ^* are optimal, then the empirical risk of w^* with respect to $\mathbf{\Delta}$:*

$$\frac{1}{n}\sum_{i=1}^{n}\Delta(y_i, h_{w^*}(x_i)) \leq \frac{1}{n}\sum_{i=1}^{n}\xi_i^*$$

Proof: Suffices to prove that $\Delta(y_i, h_{w^*}(x_i)) \leq \xi_i^*, \ \forall i$

If $h_{w^*}(x_i) = y_i$, then $\Delta(y_i, h_{w^*}(x_i)) = 0 \leq \xi_i^*$

If $h_{w^*}(x_i) = y \neq y_i$, then:

$$\begin{aligned} w^{*T} \cdot \Psi(x_i, y_i) < w^{*T} \cdot \Psi(x_i, y) \implies w^{*T} \cdot \Psi(x_i, y_i) - w^{*T} \cdot \Psi(x_i, y) = \delta < 0 \\ \delta + \xi_i^* \geq \Delta(y_i, y) \implies \xi_i^* \geq \Delta(y_i, y), \ \forall i \text{ (from the optimization constraint)} \end{aligned}$$

■

8.2.3 Problem 3: Efficient Training

Last, but not least, we need a training algorithm that finds the optimal $\mathbf{w}$ solving the quadratic program. Since there is a constraint for every incorrect label $\mathbf{y}$, we cannot enumerate all constraints and simply give the optimization problem to a standard QP solver. Instead, we propose to use the cutting-plane Algorithm 1. The key idea is to iteratively construct a working set of constraints $\mathbf{W}$ that is equivalent to the full set of constraints up to a specified precision ϵ.

Algorithm 1 for training structural SVMs (margin-rescaling).

Input: $S = ((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))$, C, ϵ
$\mathcal{W} \leftarrow \emptyset$, $\boldsymbol{w} = \mathbf{0}$, $\xi_i \leftarrow 0$ for all $i = 1, ..., n$
repeat
  for i=1,...,n **do**
    $\hat{\mathbf{y}} \leftarrow \text{argmax}_{\hat{\mathbf{y}} \in \mathcal{Y}}\{\Delta(\mathbf{y}_i, \hat{\mathbf{y}}) + \boldsymbol{w} \cdot \Psi(\mathbf{x}_i, \hat{\mathbf{y}})\}$
    if $\boldsymbol{w} \cdot [\Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \hat{\mathbf{y}})] < \Delta(\mathbf{y}_i, \hat{\mathbf{y}}) - \xi_i - \epsilon$ **then**
      $\mathcal{W} \leftarrow \mathcal{W} \cup \{\boldsymbol{w} \cdot [\Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \hat{\mathbf{y}})] \geq \Delta(\mathbf{y}_i, \hat{\mathbf{y}}) - \xi_i\}$
      $(\boldsymbol{w}, \boldsymbol{\xi}) \leftarrow \underset{\boldsymbol{w}, \boldsymbol{\xi} \geq \mathbf{0}}{\text{argmin}} \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} + \frac{C}{n}\sum_{i=1}^{n}\xi_i$ s.t. $\mathcal{W}$
    end if
  end for
until $\mathcal{W}$ has not changed during iteration
return($\boldsymbol{w}$,$\boldsymbol{\xi}$)

Starting with an empty $\mathbf{W}$ and $\mathbf{w} = \mathbf{0}$, Algorithm 1 iterates through the training examples. For each example, the **argmax** in Line 5 finds the most violated constraint of the quadratic program. If this constraint is violated by more than ϵ (Line 6), it is added to the working set $\mathbf{W}$ in Line 7 and a new $\mathbf{w}$ is computed by solving the quadratic program over the new $\mathbf{W}$ (Line 8). The algorithm stops and returns the current $\mathbf{w}$ if $\mathbf{W}$ did not change between iterations.

But how long does it take to terminate? It can be shown that Algorithm 1 always terminates in a polynomial number of iterations that is independent of the cardinality of the output space $|Y|$. In fact, a refined version of Algorithm 1 always terminates after adding at most $O(C\epsilon^{-1})$ constraints to W. Note that the number of constraints is not only independent of $|Y|$, but also independent of the number of training examples n, which makes it an attractive training algorithm even for conventional SVMs.
While the number of iterations is small, the **argmax** in Line 5 might be expensive to compute. In general, this is true, but note that this **argmax** is closely related to the **argmax** for computing a prediction $\mathbf{h}(\mathbf{x})$. It is therefore called the "loss-augmented"inference problem, and often the prediction algorithm can be adapted to efficiently solve the loss-augmented inference problem as well.