CS142: Machine Learning Spring 2017

Lecture 11

Instructor: Pedro Felzenszwalb *Scribes: Dan Xiang, Tyler Dae Devlin*

Linear Soft Margin Support Vector Machines

We continue our discussion of linear soft margin support vector machines. But first let's break down that phrase a little bit.

- *Linear* because our classifier takes a linear combination of the input features and outputs the sign of the result.
- *Soft margin* because we allow for some points to be located within the margins or even on the wrong side of the decision boundary. This was the purpose of introducing the ϵ_i terms.
- *Support vector machines* because the final decision boundary depends only on a subset of the training data known as the support vectors.

Suppose we have a training set $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_i \in \mathbb{R}^D$ and $y_i \in \{-1, +1\}$. Recall that the goal is to choose w so as to minimize the objective function $F : \mathbb{R}^D \to \mathbb{R}$ defined by

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

The first term above is motivated by regularization considerations and the second term is the hinge loss training error. Speaking of which, recall that the hinge loss H is given by

$$H(w, x, y) = \max\{0,\, 1 - y w^T x\}.$$

To minimize this loss, we would like to make $yw^T x$ large, which happens when points are far off on the correct side of the decision boundary.

Definition: A function F is called *convex* if for any $w_1, w_2 \in \mathbb{R}^D$ and $\alpha \in (0, 1)$, we have

$$F(\alpha w_1 + (1 - \alpha)w_2) \leq \alpha F(w_1) + (1 - \alpha)F(w_2).$$

If F is convex, then any local minimum of F is a global minimum. The sum of two convex functions is also convex. The maximum of two convex functions is also convex. From these facts it follows that the objective function F given by (1) is convex in w.

Gradient descent

Gradient descent is a procedure that allows one to move from some starting point on a function to a nearby local minimum. This algorithm formalizes the notion of what it means to "roll downhill." For convex functions, gradient descent will find the global minimum, since in this case a local minimum is a global minimum.

Algorithm 1 Gradient Descent

Precondition: A function $F(w)$, a step size r, and a stopping threshold ϵ.

1: **function** GRADIENTDESCENT(F, r, ϵ)
2: Initialize $w \leftarrow 0$
3: **while** the difference in the values of F on successive iterations is $> \epsilon$ **do**
4: Set $g \leftarrow \nabla F(w)$
5: Update $w \leftarrow w - rg$
6: **return** w

Although the algorithm above initializes w to 0, in general it is often better to choose the first w randomly (say multivariate normal with mean 0) so as to avoid complications produced by any symmetries the function might have at the origin.

Line search procedure

If our step size r is too large, there is a possibility that we might overshoot the minimum of F. Consequently, it makes sense to adjust r whenever this may be the case. The following line search procedure selects r in such a way that we are guaranteed to move lower down on the function in the next step.

Algorithm 2 Line Search

Precondition: A function $F(w)$, a step direction g, and the current location w_0.

1: **function** LINESEARCH(F, g, w_0)
2: Initialize $r \leftarrow \frac{1}{2}$
3: **while** $F(w_0 - rg) > F(w_0)$ **do**
4: Update $r \leftarrow \frac{r}{2}$
5: **return** r

The line search subroutine would be called between lines 4 and 5 of the gradient descent algorithm, i.e. after the step direction is chosen but before the step size is chosen.

Computing the gradient of F

In order to compute the gradient of F, we need to compute the partial derivatives of the hinge loss $H(w, x, y) = \max\{0, 1 - yw^T x\}$ with respect to each w_j. We have

$$\frac{\partial}{\partial w_j} H(w, x, y) = \begin{cases} 0 & yw^T x \geq 1 \\ -yx_j & yw^T x < 1. \end{cases}$$

Note that the derivative is not defined when $yw^T x = 1$, so we adopt the convention that the derivative is 0 in this case.

The gradient of H is then

$$\nabla H(w, x, y) = \begin{cases} 0 & yw^T x \geq 1 \\ -yx & yw^T x < 1. \end{cases} \tag{2}$$

Next, observe that the gradient of the regularization term is

$$\nabla \left(\frac{1}{2} \|w\|^2 \right) = w, \tag{3}$$

since $\|w\|^2 = \sum_{i=1}^{D} w_i^2$.

Combining equations (2) and (3) with the linearity of the gradient operator, we have

$$\nabla F(w) = w + C \sum_{i \in E} -y_i x_i,$$

where $E = \{i : y_i w^T x_i < 1\}$ is the set of indices of the training examples that are misclassified or within the margin.

Gradient descent update

Returning to the gradient descent algorithm, we see that the update rule for the weight vector w is (algorithm 1, line 5)

$$\begin{aligned} w &\leftarrow w - rw + rC \sum_{i \in E} y_i x_i \\ &= (1 - r)w + rC \sum_{i \in E} y_i x_i. \end{aligned}$$

Thus, the effect of the update step is to shrink (i.e. scale down) the vector w and add the misclassified example vectors to w.

Multiclass support vector machines

Up until now we have restricted our attention to binary classifiers, i.e. $Y = \{-1, +1\}$. Now we explore how support vector machines can be extended to handle K-class classification, where $K \geq 2$.

Let $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ be a training set with each $x_i \in \mathbb{R}^D$ and $y_i \in \{1, \ldots, K\}$. For each class $j = 1, \ldots, K$, we define a weight vector w_j. We can think of the inner product $w_j^T x$ to be the *score* of class j for example x. We define our classifier $y : \mathbb{R}^D \to \{1, \ldots, K\}$ by picking the class that has the maximum score for a given point, i.e.

$$y(x) = \operatorname*{argmax}_j w_j^T x.$$

As in the 2-class setting, we would like to pick the weight vectors in such a way that the decision boundaries are accompanied by margins. This can be accomplished by imposing the constraint

$$w_{y_i}^T x_i > \max_{j \neq y_i} (w_j^T x_i) + 1$$

for each training point $(x_i, y_i) \in T$. In words, this says that the score of the correct class should be bigger than the score of any other class by a margin of 1.

We can now define an objective F that maps a set of weight vectors to a non-negative real number,

$$F(w_1, \ldots, w_k) = \frac{1}{2} \sum_{i=1}^{K} \|w_i\|^2 + \sum_{i=1}^{N} \max \left\{ 0, \max_{j \neq y_i} \left(w_j^T x_i \right) + 1 - w_{y_i}^T x_i \right\}$$

The terms in the second sum are called the *multiclass hinge loss*. The set of weights that minimize this objective function define the **multiclass soft margin linear SVM**.

How naïve Bayes can fail

What makes the naïve Bayes approach to classification so naïve is its strong (often unwarranted) assumption of conditional independence. Recall that a naïve Bayes classifier assumes that features (such as the length and weight of a fish) are independent conditioned on the class (the type of fish). It is not difficult to come up with examples for which a naïve Bayes classifer fails badly.

Indeed, let (x_1, x_2) be the length and weight of a fish. The assumption of conditional independence gives $P(x_1, x_2 \mid y) = P(x_1 \mid y)P(x_2 \mid y)$. It's not unreasonable to expect naïve Bayes to perform decently well in this scenario. But now

suppose we redefine our feature vector by duplicating the weight x_2 999 times over. So our new feature vector is

$$(x_1, x_2, x_3, x_4, \ldots, x_{1000}),$$

where x_1 is the length and $x_2 = x_3 = \cdots = x_{1000}$ are all equal to the weight. Of course, this feature vector contains no additional information compared to the original vector (x_1, x_2), and in principle we would like that our classifier remain largely unaffected. Nevertheless, the naïve Bayes model is now

$$P(x \mid y) = P(x_1 \mid y)P(s_2 \mid y)^{999}.$$

Thus, our classification decision will be skewed in a way that almost entirely neglects the value of x_1, since the term $P(x_2 \mid y)^{999}$ now dominates the posterior probability.