# 6.867 Machine Learning

## Problem set 4 -solutions

Thursday, October 31

### What and how to turn in?

Turn in short written answers to the questions explicitly stated, and when requested to explain or prove. Do **not** turn in answers when requested to "think", "consider", "try" or "experiment" (except when specifically instructed). You may turn in answers to questions marked "optional"— they will be read and corrected, but a grade will not be recorded for them.

Turn in all MATLAB code explicitly requested, or that you used to calculate requested values. It should be clear exactly what command was used to get the answer to each question.

To help the graders (including yourself...), please be neat, answer the questions briefly, and in the order they are stated. Staple each "Problem" separately, and be sure to write your name on the top of every page.

# Problem 1: Boosting

In this problem, we slightly modify the AdaBoost algorithm seen in class to better explore some properties of the algorithm. Specifically, we no longer normalize the weights on the training examples after each iteration. The modified algorithm, which is set to run for $T$ iterations, is shown in Algorithm 1.

Note that in the modified version, the weights associated with the training examples are no longer guaranteed to sum to one after each iteration (and therefore can not be viewed as a "distribution"), but the algorithm is still valid. Let us denote the sum of weights at the start of iteration $t$ by $Z_t = \sum_{i=1}^{n} w_i^{(t)}$. At the start of the first iteration of boosting, $Z_1 = n$.

Let us now investigate the behavior of $Z_t$, as a function of $t$.

1. [**10pt**] At the $t^{th}$ iteration we found a weak classifier that achieves a weighted training error $\epsilon_t$. Show that the choice of "votes" given by $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$ is optimal in the

**Algorithm 1** AdaBoost: slightly modified version from the lecture (the weights are not normalized to sum to one).

**Input:** Set of $n$ labeled examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, $y_i = \pm 1$.

Set of associated weights $w_1^{(1)}, \ldots, w_n^{(1)}$, initially all $w_i^{(1)} = 1$.

Required number of iterations, $T$.

**for** $t = 1, \ldots, T$ **do**

Find a weak classifier $h_t$, which outputs binary predictions $h_t(\mathbf{x}) = \pm 1$, such that its weighted training error

$$\epsilon_t = \frac{1}{Z_t} \sum_{i=1}^{n} w_i^{(t)} \left(1 - \delta\left(h_t(\mathbf{x_i}), y_i\right)\right)$$

satisfies $\epsilon_t < 1/2$.

Set the vote $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$.

Update the weights : for each $i = 1, \ldots, n$ set

$$w_i^{(t+1)} = w_i^{(t)} e^{-y_i \alpha_t h_t(\mathbf{x}_i)}$$

**end for**

**Output:** the combined classifier defined by

$$\hat{h}_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$

sense that it minimizes $Z_{t+1}$.

*Advice: Look at $Z_{t+1}$ as a function of $\alpha$, and find the value for which the function achieves its minimum. You may also find the following notational shorthand useful:*

$$W_I = \sum_{i=1}^{n} w_i^{(t)}\big(1 - \delta(y_i, h_t(\mathbf{x}_i))\big), \quad W_C = \sum_{i=1}^{n} w_i^{(t)}\delta(y_i, h_t(\mathbf{x}_i)),$$

*where $W_C$ is the total weight of points classified by $h_t$ correctly, and $W_I$ the total weight of misclassified points. $\delta(y, h_t(\mathbf{x})) = 1$ whenever the label predicted by $h_t$ is correct and zero otherwise. The weights here are those available at the start of iteration $t$.*

**Answer:** For an example $\mathbf{i}$ which is misclassified by $h_t$, the weight becomes $w_i^{(t)}e^{\alpha_t}$, and for a correctly classified example $\mathbf{x}_j$ the new weight is $w_j^{(t)}e^{-\alpha_t}$. Grouping the new weights of all the misclassified examples in one term and the weights of the correctly classified examples in another term, we get

$$Z_{t+1}(\alpha_t) = W_\mathsf{I} e^{\alpha_t} + W_\mathsf{C} e^{-\alpha_t}.$$

Taking derivative w.r.t. $\alpha_t$ and finding its zero yields

$$\frac{\partial Z_{t+1}}{\partial \alpha_t} = W_\mathsf{I} e^{\alpha_t} - W_\mathsf{C} e^{-\alpha_t} = 0$$
$$\Rightarrow e^{2\alpha_t} = \frac{W_\mathsf{C}}{W_\mathsf{I}} \quad \Rightarrow \quad \alpha_t = \frac{1}{2}\log\frac{W_\mathsf{C}}{W_\mathsf{I}}.$$

Note that by definition, $W_\mathsf{C} = Z_t(1-\epsilon_t)$, $W_\mathsf{I} = Z_t\epsilon_t$, and we are done with the proof.

2. [**5pt**] Show that the sum of weights $Z_t$ is monotonically decreasing as a function of $t$.

**Answer:** If we use $Z_{t+1}(\alpha_t) = W_\mathsf{I} e^{\alpha_t} + W_\mathsf{C} e^{-\alpha_t}$ from the previous problem, we see that $Z_{t+1}(0) = Z_t$. Thus

$$Z_{t+1} = \min_{\alpha_t} Z_{t+1}(\alpha_t) < Z_{t+1}(0) = Z_t$$

whenever the minimizing $\alpha_t$ is non-zero. Since $\alpha_t = 0.5 * \log((1-\epsilon_t)/\epsilon_t)$, $\alpha_t$ is zero only when the error of the weak classifier is exactly chance.

We might also want to get an idea of how much $Z_t$ decreases as a function of $t$. Using the exression for $\alpha_t$ in terms of $W_\mathsf{C}, W_\mathsf{I}$ found in the previous question, we have

$$Z_{t+1} = W_\mathsf{C} e^{\frac{1}{2}\log W_\mathsf{C}/W_\mathsf{I}} + W_\mathsf{I} e^{-\frac{1}{2}\log W_\mathsf{C}/W_\mathsf{I}} = 2\sqrt{W_\mathsf{I} W_\mathsf{C}}$$

We also know that $Z_t = W_\mathsf{I} + W_\mathsf{C}$, and that $W_\mathsf{I} < W_\mathsf{C}$ (by our choice of $h_t$). Thus, taking the difference between the positive $Z_t$ and $Z_{t+1}$, we get

$$Z_t^2 - Z_{t+1}^2 = W_\mathsf{C}^2 + W_\mathsf{I}^2 + 2W_\mathsf{C}W_\mathsf{I} - 4W_\mathsf{C}W_\mathsf{I} = (W_\mathsf{C} - W_\mathsf{I})^2 > 0,$$

We know now that $Z_t$ decreases, and that our choice of $\alpha_t$ is optimal for minimizing $Z_{t+1}$; but is this quantity useful in any way? It is useful in bounding how successive boosting iterations reduce the training error.

3. [**10pt**] Show that the training error (the average number of misclassified training examples) of the combined classifier after $m$ iterations of boosting,

$$\hat{h}_m(\mathbf{x}) = \sum_{t=1}^{m} \alpha_t h_t(\mathbf{x}),$$

is bounded from above by $Z_{m+1}/n$.

*Advice: In the definition of the algorithm, the weights (and therefore $Z_t$) are defined recursively. You will find it helpful to "unroll" the definition, and write out the value of $Z_{t+1}$ in terms of the initial weights (which are all 1) and the subsequent updates, using $\alpha_1, \ldots, \alpha_m$. You may also find the following (simple) fact helpful: for any $x \geq 0$, $e^x \geq 1$.*

**Answer:** Following the advice, we find that

$$w_i^{(m+1)} = 1 \cdot e^{-y_i \alpha_1 h_1(\mathbf{x}_1)} \cdot \ldots e^{-y_i \alpha_m h_m(\mathbf{x}_1)} = e^{-y_i \sum_{t=1}^{m} \alpha_t h_t(\mathbf{x}_i)}$$

Let us look at an example $\mathbf{x}_i$ which is misclassified by the combined classifier $\hat{h}_m$. The value of $-y_i \sum_{t=1}^{m} \alpha_t h_t(\mathbf{x}_i)$ will be positive; thus, using the property in the advice, we have

$$w_i^{(m+1)} = e^{-y_i \sum_{t=1}^{m} \alpha_t h_t(\mathbf{x}_i)} > 1.$$

Suppose there are $N$ misclassified examples, out of total $n$ examples in the training set. Then,

$$\begin{aligned} Z_{m+1} = & \sum_{j:\, y_j = \hat{h}_m(\mathbf{x}_j)} w_j^{(m+1)} + \sum_{i:\, y_i = -\hat{h}_m(\mathbf{x}_i)} w_i^{(m+1)} \\ \geq & \sum_{i:\, y_i = -\hat{h}_m(\mathbf{x}_i)} w_i^{(m+1)} \\ \geq & N, \end{aligned}$$

where we have used the fact $w_i^{(m+1)} > 1$ for each of the $N$ misclassified examples. We get the result by dividing both sides of the inequality by $n$ and noting that $N/n$ is the training error of the combined classifier.

We have shown that AdaBoost tries in each iteration to minimize an upper bound on the training error of the combined classifier, and guarantees that this bound will monotonically decrease as the algorithm proceeds. How much the bound decreases at each iteration depends on the weighted training error of each weak classifier.

In the AdaBoost algorithm presented above, the vote $\alpha_t$ assigned to a weak classifier $h_t$ does not differentiate between the "directions" of mistakes it makes (misclassifying a positive example as negative or vice versa). We already know that the magnitude of the vote is related to the performance of the classifier - that is, a classifier which achieves lower weighted training error will get a higher $\alpha$. But what if $h_t$ makes almost all of its mistakes on, say, negative examples? That is, for a training example $\mathbf{x}$ if $h_t(\mathbf{x}) = -1$, with very high probability $\mathbf{x}$ is indeed negative, but if $h_t(\mathbf{x}) = +1$, it is quite likely that in fact $\mathbf{x}$ is negative.

Recall that the vote $\alpha_t$ measures the influence of $h_t$ in the final (combined) classifier. Thus, it would make sense to assign a high vote to its decisions in the "direction" in which $h_t$ makes fewer mistakes, and a lower vote to the opposite decisions. Algorithm 2 describes the modification in AdaBoost that allows for such a refinement.

**Algorithm 2** Modified boosting algorithm, allowing different votes for positive and negative decisions

**Input:** Set of $n$ labeled examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, $y_i = \pm 1$.
Set of associated weights $w_1^{(1)}, \ldots, w_n^{(1)}$, initially all $w_i^{(1)} = 1$.
Required number of iterations, $T$.
**for** $t = 1, \ldots, T$ **do**
Find a weak classifier $h_t$, which outputs binary predictions $h_t(\mathbf{x}) = \pm 1$, such that its weighted training error

$$\epsilon_t = \frac{1}{Z_t} \sum_{i=1}^{n} w_i^{(t)} \left(1 - \delta\left(h_t(\mathbf{x_i}), y_i\right)\right)$$

satisfies $\epsilon_t < 1/2$
Set the votes $\alpha_t$, $\beta_t$ (see Question 5).
Update the weights : for each $i = 1, \ldots, n$ set

$$w_i^{(t+1)} = w_i^{(t)} e^{-y_i v_t(\mathbf{x}_i)}$$

where

$$v_t(\mathbf{x}) = \begin{cases} \alpha_t & \text{if } h_t(\mathbf{x}) = 1, \\ -\beta_t & \text{if } h_t(\mathbf{x}) = -1. \end{cases}$$

**end for**
**Output:** the combined classifier defined by

$$\hat{h}_T(\mathbf{x}) = \sum_{t=1}^{T} v_t(\mathbf{x})$$

4. [**3pt**] Express $Z_{t+1}$ as a function of $\alpha_t$, $\beta_t$ and the weights at the start of $t$-th iteration.

**Answer:** Using reasoning identical to that in Question 1, we note that the new weight for a misclassified positive example $\mathbf{x}_i$ ($h_t$ predicts negative) would be

$$w_i^{(t+1)} = w_i^{(t)} e^{-y_i(-\beta_t)} = w_i^{(t)} e^{\beta_t},$$

the weight on a correctly classified negative example $\mathbf{x}_j$ ($h_t$ predicts positive) would be

$$w_j^{(t+1)} = w_j^{(t)} e^{-y_j(-\beta_t)} = w_j^{(t)} e^{\beta_t},$$

and so on. Let's define $W_{+-}$ as the weight of examples labeled $+$ (first subscript) for which $h_t$ predicts $-$ (second subscript). Defining $W_{++}$, $W_{-+}$, and $W_{--}$ analogously, we can group the new weights into four terms:

$$Z_{t+1} = W_{++}^{(t)} e^{-\alpha_t} + W_{+-}^{(t)} e^{\beta_t} + W_{--}^{(t)} e^{-\beta_t} + W_{-+}^{(t)} e^{\alpha_t}.$$

Note that whether the term has $\alpha_t$ or $\beta_t$ depends on the second subscript of the preceding weight.

5. [**12pt**] Fill in the missing part in the algorithm above. That is, find the rules for computing $\alpha_t$ and $\beta_t$ that would minimize $Z_{t+1}$. You may use the following notational shorthands: $W_{++}^{(t)}$ is the total weight (under the weights active at the beginning of the $t$-th iteration) of the positive examples that are classified as positive by $h_t$; $W_{+-}^{(t)}$ is the total weight of the positive examples misclassified by $h_t$; and similarly defined $W_{--}^{(t)}$ and $W_{-+}^{(t)}$.

   **Answer:** Let us first find the partial derivatives of $Z_{t+1}$ w.r.t. the votes:

$$\frac{\partial Z_{t+1}}{\partial \alpha_t} = -W_{++}^{(t)} e^{-\alpha_t} + W_{-+}^{(t)} e^{\alpha_t},$$
$$\frac{\partial Z_{t+1}}{\partial \beta_t} = -W_{--}^{(t)} e^{-\beta_t} + W_{+-}^{(t)} e^{\beta_t}.$$

   Setting these to zero and manipulating the equations in exactly same way as in Question 1, we get the rules

$$\alpha_t = \frac{1}{2} \log \frac{W_{++}^{(t)}}{W_{-+}^{(t)}},$$
$$\beta_t = \frac{1}{2} \log \frac{W_{--}^{(t)}}{W_{+-}^{(t)}},$$

   which corresponds to our intuitive objective: larger $\alpha_t$ means that out of all examples classified as $+1$ many are true positives.

# Problem 2: Complexity

Here we try to understand a bit better some implications of the fact that the feature vectors corresponding to the radial basis kernel are functions (living in an infinite dimensional space). It turns out that the VC-dimension of the classifier that uses a radial basis kernel is infinite. We can actually make a bit stronger statement: with a radial basis kernel we can perfectly separate *any* finite set of *distinct* training points. Why is this a stronger statement? To claim that the VC-dimension of a classifier is $n$ we only need to find *some* set of $n$ training points that we can shatter; this need not hold for all sets of $n$ points.

To get started we'll assume that we have a set of $n$ points in $\mathcal{R}^d$. The dimension $d$ is assumed to be finite and, perhaps surprisingly, plays no role in the argument. We also make the critical assumption that all the training points are distinct. The radial basis kernel we use has the following form:

$$K(\mathbf{x}, \mathbf{x}') = \exp\{ -\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}'\|^2 \}$$

where $\sigma^2$ is a width parameter specifying how quickly the kernel vanishes as the points move further away from each other. All of our results hold for any positive finite value of $\sigma^2$. In other words, whether or not we'll be able to perfectly separate the training points has nothing to do with the kernel width. The kernel width does affect generalization performance, however.

1. [**15pt**] Let's proceed in stages. To make things easier we are going to prove a bit stronger result than necessary. In particular, we'll show that

   $$\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } y_i \mathbf{w}^T \phi(\mathbf{x}_i) = 1,\ i = 1, \ldots, n$$

   has a solution regardless of how we set the $\pm 1$ training labels $y_i$. You should convince yourself first that this is consistent with our goal. Here $\phi(\mathbf{x}_i)$ is the feature vector (function actually) corresponding to the radial basis kernel. Our formulation here is a bit non-standard for two reasons. We try to find a solution where *all* the points are support vectors. This is not possible for all valid kernels but makes our life easier here. We also omit the bias term since it is not needed for the result.

   Introduce Lagrange multipliers for the constraints similarly to finding the SVM solution (see also the tutorial on Lagrange multipliers distributed along with the lecture slides) and show the form that the solution $\hat{\mathbf{w}}$ has to take. You can assume that $\mathbf{w}$ and $\phi(\mathbf{x}_i)$ are finite vectors for the purposes of these calculations. Note that the Lagrange multipliers here are no longer constrained to be positive. Since you are trying to satisfy equality constraints, the Lagrange multipliers can take any real value.

   We are after $\hat{\mathbf{w}}$ as a function of the Lagrange multipliers. (this should not involve lengthy calculations).

**Answer:** We introduce a Lagrange multiplier $\alpha_i$ for each constraint and add these to the minimization objective. As a result we will *minimize* the unconstraint objective

$$J(\mathbf{w}, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{j=1}^{n} \alpha_j (y_j \mathbf{w}^T \phi(\mathbf{x}_j) - 1)$$

with respect to $\mathbf{w}$ and try to *maximize* it with respect to the Lagrange multiplers (so as to enforce the classification constraints). Note that when the classification constraints are satisfied the terms that we added to the objective vanish. On the other hand, if we set $\mathbf{w}$ to a value that does not satisfy the constraints then will choose $\alpha_i$ (any real value) so that $J(\mathbf{w}, \alpha) = \infty$ since we are maximizing the objective with respect to $\alpha_i$.

We can now take the derivative of $J(\mathbf{w}, \alpha)$ with respect to $\mathbf{w}$ and set it to zero. This will give us $\hat{\mathbf{w}}$ as a function of $\alpha$'s:

$$\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}, \alpha) = \mathbf{w} - \sum_{j=1}^{n} \alpha_j y_j \phi(\mathbf{x}_j) = 0$$

which immediately gives $\hat{\mathbf{w}} = \sum_{j=1}^{n} \alpha_j y_j \phi(\mathbf{x}_j)$.

2. [**5pt**] Put the resulting solution back into the classification (margin) constraints and express the result in terms of a linear combination of the radial basis kernels.

**Answer:** Here we plug in the above solution to each constraint $y_i \mathbf{w}^T \phi(\mathbf{x}_i) = 1$ giving

$$\begin{aligned} y_i \left( \sum_{j=1}^{n} \alpha_j y_j \phi(\mathbf{x}_j) \right)^T \phi(\mathbf{x}_i) &= y_i \left( \sum_{j=1}^{n} \alpha_j y_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) \right) \quad (1) \\ &= y_i \left( \sum_{j=1}^{n} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) \right) = 1 \quad (2) \end{aligned}$$

3. [**5pt**] Indicate briefly how we can use the following Michelli theorem to show that any $n$ by $n$ kernel matrix $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ for $i, j = 1, \ldots, n$ is invertible.

Theorem: *If $\rho(t)$ is monotonic function in $t \in [0, \infty)$, then the matrix $\rho_{ij} = \rho(\|\mathbf{x}_i - \mathbf{x}_j\|)$ is invertible for any distinct set of points $\mathbf{x}_i$, $i = 1, \ldots, n$.*

**Answer:** We can express $K(\mathbf{x}_i, \mathbf{x}_j)$ as a monotonic function of the distance $\|\mathbf{x}_i - \mathbf{x}_j\|$:

$$K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \exp\{ -\frac{1}{2\sigma^2} \|\mathbf{x}_i - \mathbf{x}_j\|^2 \} = \rho(\|\mathbf{x}_i - \mathbf{x}_j\|)$$

where $\rho(z) = \exp(-z^2/2)$. Here $\rho(z)$ is a monotonically decreasing function of $z$ (right half of a Gaussian) and therefore qualifies for the theorem.

4. [**10pt**] Based on the above results put together the argument to show that we can indeed find a solution where all the points are support vectors.

**Answer:** Let's first write the classification constraints in terms of the kernel matrix $K_{ij}$ evaluated at the available points:

$$y_i \sum_{j=1}^{n} \alpha_j y_j K_{ji} = 1, \text{ for all } i$$

or, equivalently, as

$$y_i \sum_{j=1}^{n} K_{ij} y_j \alpha_j = 1, \text{ for all } i$$

since $K_{ij} = K_{ji}$. To put these constraints in a matrix form, we can define a diagonal matrix $Y$, where $Y_{ii} = y_i$, and $Y^{-1} = Y$. Thus, in a matrix form, the above constraints are $YK(Y\alpha) = \mathbf{1}$, where $\alpha = [\alpha_1, \ldots, \alpha_n]^T$ and $\mathbf{1} = [1, \ldots, 1]^T$. Since we have already established that $K$ is invertible, we get an explicit solution for $\alpha$. Specifically, $\hat{\alpha} = (YKY)^{-1}\mathbf{1} = YK^{-1}Y\mathbf{1}$.

Of course the fact that we can in principle separate any set of training examples does not mean that our classifier does well (on the contrary). So, why do we use the radial basis kernel? The answer is given by the more refined notions of VC-dimension such as the $V(\gamma)$ dimension. This refined notion of complexity takes into account the margin that we achieve for the linear classifier in the feature space. In other words, $V(\gamma)$ is defined as the number of points that we can shatter with *margin* $\gamma$. $\gamma$ here is the distance of the closest training point to the linear boundary in the feature space (the distance is measured in the feature space).

This is useful in our case here since we know that if the feature vectors lie within a sphere of radius $R$ then $V(\gamma) \leq R^2/\gamma^2$. The larger the margin requirement, the fewer points we can shatter. This is particularly convenient for radial basis kernels since

$$\phi(\mathbf{x})^T \phi(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}) = 1$$

In other words, the feature vectors have norm exactly one. If we now use a radial basis kernel and look for a classifier that separates the training points with margin $\gamma$ what is the $V(\gamma)$ dimension of our classifier as a function of $\gamma$?

**Answer:** For us to use the bound $V(\gamma) \leq R^2/\gamma^2$ we have to know the radius of the smallest sphere containing the points. This sphere needs to be in the feature space where our classifier is linear. For a radial basis kernel $\phi(\mathbf{x})^T\phi(\mathbf{x}) = 1$ and thus all the feature vectors are contained within a unit sphere. Therefore $R = 1$ in the bound and we get

$$V(\gamma) \leq 1/\gamma^2$$

5. [**10pt**] Show a one dimensional example where adjusting the kernel width makes a difference in terms of the margin that we can achieve for the points.

**Answer:** If we only have two training examples with opposite labels, then we would try to set the kernel width to the smallest possible value or zero. This is because any influence that the kernel permits between the points (the size of $K_{12} = K(\mathbf{x}_1, \mathbf{x}_2)$) is in this case necessarily destructive for classification.

Let's see how this works a bit more formally (not required from your perspective). Since the points are separable, the margin is given by $1/\|\hat{\mathbf{w}}\|$, where $\hat{\mathbf{w}}$ can be expressed in terms of the kernel and the Lagrange multipliers $\hat{\alpha}$. In this simple case we know that both points will become support vectors and therefore satisfy the margin constraints with equality. To get the solution $\hat{\alpha}$ we can use our earlier result: $\hat{\alpha} = YK^{-1}Y\mathbf{1}$, where $K$ is a 2 by 2 matrix

$$K = \begin{bmatrix} 1 & K_{12} \\ K_{12} & 1 \end{bmatrix}, \text{ and } K^{-1} = \frac{1}{1-K_{12}^2}\begin{bmatrix} 1 & -K_{12} \\ -K_{12} & 1 \end{bmatrix},$$

where $K_{11} = K_{22} = 1$ for the radial basis kernel and $K_{12} = K_{21}$ by symmetry. The kernel width only affects the value of $K_{12}$ (larger width, larger value). We can evaluate the margin exactly in this simple setting

$$\begin{aligned}
1/\|\hat{\mathbf{w}}\| &= (\|\hat{\mathbf{w}}\|^2)^{-1/2} = \left((Y\hat{\alpha})^T K (Y\hat{\alpha})\right)^{-1/2} \text{ (by definition of } \hat{\mathbf{w}}) && (3)\\
&= \left((K^{-1}Y\mathbf{1})^T K (K^{-1}Y\mathbf{1})\right)^{-1/2} \text{ (since } Y\hat{\alpha} = K^{-1}Y\mathbf{1}) && (4)\\
&= \left(\mathbf{1}^T Y K^{-1} Y \mathbf{1}\right)^{-1/2} \text{ (since } K^{-1}KK^{-1} = K^{-1}) && (5)\\
&= \left(\frac{1}{1-K_{12}^2}\mathbf{1}^T Y \begin{bmatrix} 1 & -K_{12} \\ -K_{12} & 1 \end{bmatrix} Y\mathbf{1}\right)^{-1/2} && (6)\\
&= \left(\frac{1}{1-K_{12}^2}\mathbf{1}^T \begin{bmatrix} y_1y_1 & -y_1y_2K_{12} \\ -y_2y_1K_{12} & y_2y_2 \end{bmatrix} \mathbf{1}\right)^{-1/2} && (7)\\
&= \left(\frac{1}{1-K_{12}^2}\mathbf{1}^T \begin{bmatrix} 1 & K_{12} \\ K_{12} & 1 \end{bmatrix} \mathbf{1}\right)^{-1/2} && (8)\\
&= \left(\frac{1}{1-K_{12}^2}(2+2K_{12})\right)^{-1/2} && (9)\\
&= \left(\frac{1}{(1+K_{12})(1-K_{12})}(2+2K_{12})\right)^{-1/2} && (10)\\
&= \left(\frac{2}{(1-K_{12})}\right)^{-1/2} = \sqrt{(1-K_{12})/2} && (11)
\end{aligned}$$

So, to maximize the margin we'd like $K_{12}$ to be as small as possible. This corresponds to setting the kernel width to zero.

It is actually more common that we can improve the margin by increasing the width. Points that are close with the same label can help each other's classification when $K(\mathbf{x}_i, \mathbf{x}_j)$ is reasonably large.