Support Vector Machines (send corrections to yks at csail.mit.edu)

In SVMs we are trying to find a decision boundary that maximizes the "margin" or the "width of the road" separating the positives from the negative training data points.

To find this we **minimize**: $\frac{1}{2}\|\vec{w}\|^2$ subject to the constraints $y_i(\vec{w}\cdot\vec{x_i}+b)\geq 1$

The resulting Lagrange multiplier equation we try to optimize is:

$$L=\frac{1}{2}\|w\|^2-\sum_i\alpha_i(y_i(\vec{w}\cdot\vec{x_i}+b)-1)$$

Solving the above Lagrangian optimization problem will give us w, b, and alphas, parameters that determines a **unique** maximal margin (road) solution. On the maximum margin "road", the +ve, and -ve points that stride the "gutter" lines are called ***support vectors***. The decision boundary lies at the middle of the road. The definition of the "road" is dependent only on the support vectors, so changing (adding deleting) non-support vector points will not change the solution. Note, that widest "road" is a 2D concept. If the problem is in 3D we want the widest region bounded by two planes; in even higher dimensions, a subspace bounded by two hyperplanes.

Solving for the Lagrange multiplier α_is in general requires numerical optimization methods that are beyond the scope of this class. In practice, you use Quadratic Programming solvers. A popular algorithm for solving SVMs is Platt's SMO (Sequential Minimal Optimization) algorithm. For SVM problems on quizzes, we generally just ask you to solve for the values of w, b and alphas using algebra and/or geometry.

Useful Equations for solving SVM questions

A. Equations derived from optimizing the Lagrangian:

1. **Partial of the Lagrangian wrt to b**: From $\frac{\partial L}{\partial b}=0$

$\sum_i\alpha_i y_i=0$	Note that $y_i\in\{-1,+1\}$ and $\alpha_i=0$ for non-support vectors.

Sum of all alphas (support vector weights) with their signs should add to 0.

2. **Partial of the Lagrangian wrt to w**: From $\frac{\partial L}{\partial w}=0$

$\sum_i\alpha_i y_i\vec{x_i}=\vec{w}$	For when using a linear kernel. The summation only contains support vectors. Support vectors are training data points with $\alpha_i>0$
$\sum_i\alpha_i y_i\phi(\vec{x_i})=\vec{w}$	For when using a decomposable kernel (see definition below).

Sum of alphas, ys of support vectors wrt to vector w.

B. Equations from the boundaries and constraints:

3. **The Decision boundary**:

$h(\vec{x})=\sum_i\alpha_i y_i K(\vec{x_i},\vec{x})+b\geq 0$	General form, for any kernel. To classify an unknown $\vec{x}$, we compute the kernel function $K(\vec{x_i},\vec{x})$ against each of the **support vectors** $\vec{x_i}$. Support vectors are training data points with $\alpha_i>0$
$h(\vec{x})=\sum_i[(\alpha_i y_i\vec{x_i})\cdot\vec{x}]+b\geq 0$ $h(\vec{x})=\vec{w}\cdot\vec{x}+b\geq 0$	For when using a linear kernel $K(\vec{x_i},\vec{x})=\vec{x_i}\cdot\vec{x}$

4. **Positive gutter:**

$h(\vec{x})=\sum_i\alpha_i y_i K(\vec{x_i},\vec{x})+b=1$	General form, for any kernel.
$h(\vec{x})=\sum_i[(\alpha_i y_i\vec{x_i})\cdot\vec{x}]+b=1$	

$h(\vec{x}) = \vec{w} \cdot \vec{x} + b = 1$	For use when the Kernel is linear.

5. **Negative gutter:**

$$h(\vec{x}) = \sum_i \alpha_i y_i K(\vec{x_i}, \vec{x}) + b = -1 \qquad h(\vec{x}) = \vec{w} \cdot \vec{x} + b = -1$$

6. **The width of the margin (or road):**

$$\text{width of road} \equiv m = \frac{2}{\|\vec{w}\|} \quad \text{where,} \quad \|\vec{w}\| = \sqrt{\sum_i w_i^2}$$

Alternate formula for the two support vector case:

$$\text{width of road} \equiv m = \frac{\vec{w}}{\|\vec{w}\|} \cdot (\vec{x}_+ - \vec{x}\_)$$

This equation is useful when solving SVM problems in 1D or 2D, where the width of the road can be ***visually determined****.*

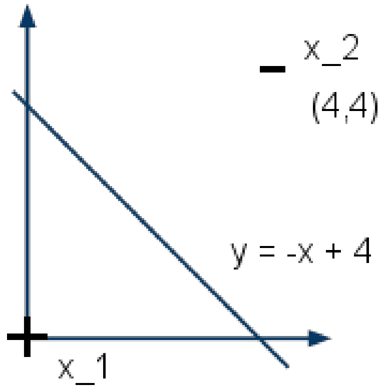
Common SVM Kernels:

Linear Kernel	$K(\vec{u}, \vec{v}) = \vec{u} \cdot \vec{v}$ In document classification, feature vectors are composed of binary word features: I(word=foo) outputs 1 if the word "foo" appears in the document 0 if it does not. Each document is represented as |vocabulary| length feature vectors. Support vectors found are generally particularly salient documents (documents best at discriminating topics being classified).
Decomposable Kernels Idea: Define $\phi(\vec{u})$ that transforms input vectors into a different (usually higher) dimensional space where the data is (more easily) linearly separable.	$K(\vec{u}, \vec{v}) = \phi(\vec{u}) \cdot \phi(\vec{v})$ Example: $\phi(\vec{u}) = \begin{bmatrix} \cos(u_1) \\ \sin(u_2) \end{bmatrix}$ $K(\vec{u}, \vec{v}) = \cos(\vec{u_1})\cos(\vec{v_1}) + \sin(\vec{u_2})\sin(\vec{v_2})$
Polynomial Kernel	$K(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v} + b)^n$ n > 1 Example: Quadratic Kernel: $K(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v} + b)^2$ - In 2D resulting decision boundary can look parabolic, linear or hyperbolic depending on which terms in the expansion dominate. - Here is an expansion of the quadratic kernel, with u = [x, y] $K(\vec{u}, \vec{v}) = \left(\begin{bmatrix} x \\ y \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + b \right)^2$ $= (v_1 x + v_2 y + b)^2$ $= [(v_1^2)x^2 + (v_2^2)y^2] + [b^2 + (2v_1 b)x + (2v_2 b)y] + [(2v_1 v_2)xy]$ HW: Try this Kernel using Professor Winston's demo
Radial Basis Function (RBF) or Gaussian	$K(\vec{u}, \vec{v}) = \exp\left(-\frac{\|\vec{u} - \vec{v}\|^2}{2\sigma^2} \right)$ In 2D generated decision boundaries resemble contour circles around clusters of +ve and -ve points. Support vectors are generally +ve or -ve points that

Kernel • Will fit almost any data. May exhibit overfitting when used improperly. • Similar to KNN but with all points having a vote; weight of each vote determined by Gaussian ∘ Points farther away get less of a vote than points nearby	are closest to the opposing cluster. The contour space drawn results from sum of support vector Gaussians. HW: Try this Kernel using Professor Winston's demo When $\sigma 2$ is large you get flatter Gaussians. When $\sigma 2$ is small you get sharper Gaussians. (Hence when using a small $\sigma 2$ contour density will appear closer / denser around support vector points). Here is the Kernel in-2D expanded out, with u = [x, y] $K(\vec{u},\vec{v}) = \exp\left(-\frac{(x-v_1)^2+(y-v_2)^2}{2\sigma^2}\right)$ As a point gets closer to a support vector it approaches exp(0) = 1. As a point moves far away from a support vector it approaches exp(-infinity) = 0.
Sigmoidal (tanh) Kernel • Allows for combination of linear decision boundaries	$K(\vec{u},\vec{v}) = \tanh(k\vec{u}\cdot\vec{v}+b)$ $K(\vec{u},\vec{v}) = \frac{e^{(k\vec{u}\cdot\vec{v}+b)}+1}{e^{(k\vec{u}\cdot\vec{v}+b)}-1}$ Properties of tanh: • Similar to the sigmoid function $s(x) = \frac{1}{1+e^{-x}}$ • Ranges from -1 to +1. • tahn(x) => +1 when x >> 0 • tahn(x) => -1 when x << 0 Resulting decision boundaries are logical combinations of linear boundaries. Not too different from second layer neurons in Neural Nets. Like RBF, may exhibit overfitting when improperly used.
Linear combination of Kernels Idea: Kernel functions are closed under addition and scaling (by a positive number).	Scaling: $aK(\vec{u},\vec{v})$ for a > 0 or Linear combination: $K(\vec{u},\vec{v}) = aK_1(\vec{u},\vec{v}) + bK_2(\vec{u},\vec{v})$ a,b>0

Method 1 of Solving SVM parameters by inspection:

This is a step-by-step solution to Problem 2.A from 2006 quiz 4:
We are given the following graph with x_1 and x_2 points on the x-y axis;
+ve point at x_1 (0, 0) and a -ve point x_2 at (4, 4).



Can a SVM separate this? i.e. is it linearly separable? Heck Yeah! using the line above.
Part 2A: Provide a decision boundary:

We can find the decision boundary by graphical inspection.

1. The decision boundary lies on the line: y = -x + 4
2. We have a +ve support vector at (0, 0) with line equation y = -x
3. We have a -ve support vector at (4, 4) with line equation y = - x + 8

Given the equation for the decision boundary, we next massage the algebra to get the decision boundary to conform with the desired form, namely:

$$h(\vec{x})=w_1x+w_2y+b\geq 0$$

1. $y<-x+4$ (< because +ve is below the line)
2. $x+y-4<0$
3. $-x-y+4\geq 0$ (multiplied by -1)
4. $-1x-1y+4\geq 0$ (writing out the coefficients explicitly)

Now we can read the solution from the equation coefficients:

w_1 = -1 w_2 = -1 b = 4

Next, using our formula for width of road, we check that these weights gives a road width of: $\frac{2}{\sqrt{-1^2+-1^2}}=\sqrt{2}$.

WAIT! This is clearly not the width of the "widest" road/margin.

We remember that any multiple c (c>0) of the boundary equation is still the same decision boundary. So all equations of the form:

$$-cx_1-cx_2+4c\geq 0$$

Strides this decision boundary. So here is a more general solution:

w_1 = -c w_2 = -c b = 4c

or $\vec{w}=\begin{bmatrix}-c\\-c\end{bmatrix}$ and $b=4c$

Using The Width of the Road Constraint

Graphically we see that the widest width margin should be: $4\sqrt{2}$

The solution weight vector $\vec{w}$ and intercept b can be solved by solving for c constrained by the known width-of-the-road.

Length of $\vec{w}$ in terms of c:

$$\|\vec{w}\|=\sqrt{(-c)^2+(-c)^2}=\sqrt{2}c$$

Now plugin all this into the margin width equation and solving for c, we get:

$$\frac{2}{\|\vec{w}\|}=4\sqrt{2} \quad => \quad \frac{2}{\sqrt{2}c}=4\sqrt{2} \quad => \quad \frac{2}{c}=4\cdot 2 \quad => \quad c=\frac{1}{4}$$

This means the true weight vector and intercept for the **SVM solution** should be:

$$\vec{w}=\begin{bmatrix}-\frac{1}{4}\\-\frac{1}{4}\end{bmatrix} \text{ and } b=4\cdot\frac{1}{4}=1$$

Next we **solve for alphas**, using the w vector and equation 1.

$$\sum_i \alpha_i y_i \vec{x_i}=\vec{w}$$

Plugin in the vector values of support vectors and w:

$$\alpha_1(+1)\vec{x_1}+\alpha_2(-1)\vec{x_2}=\alpha_1(+1)\begin{bmatrix}0\\0\end{bmatrix}+\alpha_2(-1)\begin{bmatrix}4\\4\end{bmatrix}=\begin{bmatrix}-\frac{1}{4}\\-\frac{1}{4}\end{bmatrix}$$

We get two identical equations:

$$-\frac{1}{4}=-4\alpha_2 \text{ or } \alpha_2=\frac{1}{16}$$

Using Equation 1, now we can solve for the other alpha:

$$(+1)\alpha_1+(-1)\alpha_2=0$$
$$\alpha_1=\alpha_2=\frac{1}{16}$$

Part 2B: Does the boundary change if a +ve point x_3 is added at (-1, -1)?
No. Support vectors are still at 1, and 2. Decision boundary stays the same.
Part 2C: What if point x_2 (-ve) is moved to coordinate (k, k)?
How will α values change, increase, decrease or stay same? When k = 2? and k = 8?
Answer: Go back to how we solved for alphas:

$$\alpha_1(+1)x_1+\alpha_2(-1)x_2=\begin{bmatrix}-c\\-c\end{bmatrix}$$

Plugin in x_2

$$\alpha_2(-1)\begin{bmatrix}k\\k\end{bmatrix}=\begin{bmatrix}-c\\-c\end{bmatrix}$$

Solving for α_2

$$c=k\alpha_2 \text{ or } \alpha_2=c\,/\,k$$

Using the fact that $|\vec{w}|=\sqrt{2}\cdot c$, $c=|\vec{w}|/\sqrt{2}$
and width-of-road/margin $m=2/|\vec{w}|$.
We express alpha in terms of the margin *m*:

$$\alpha_2=\frac{\sqrt{2}\|w\|}{k}=\frac{2\sqrt{2}}{mk}$$

Answer:

- When k changes from 4 to 2. The margin (road width) m is halved and k is also halved. So alpha must **increase** by a factor of 4.
- When k changes from 4 to 8. The margin m is doubled, k is also doubled. So alpha must **decrease** by a factor of 4.

Though we do not provide a full proof here. Alpha in generally changes *inversely* with m.
Widen road -> lower alpha. Narrowed road -> higher alpha

Method 2: Solving for alpha, b, and w without visual inspection (By computing Kernels and solving Constraint equations)

Example from 2005 Final Exam.
In this problem you are told that you have the following points.

-ve points: A at (0, 0) B at (1, 1)
+ve points: C at (2, 0)

and that these points lie on the gutter in the SVM max-margin solution.

Step 1. Compute all kernels function values, which in this case, these are all dot products.

K(A, A) = 0*0+0*0 = 0	K(A, B) = 0*1+0*1 = 0	K(A, C) = 0*2+0*0 = 0
K(B, A) = 1*0+1*0 = 0	K(B, B) = 1*1+1*1 = 2	K(B, C) = 1*2+1*0 = 2
K(C, A) = 2*0+0*0 = 2	K(C, B) = 2*1+0*1 = 2	K(C, C) = 2*2+0*0 = 4

Step 2: Write out the system of equations, using SVM constraints:
Constraint 1: $\sum_i \alpha_i y_i = 0$,
Constraint 2: $\sum_i \alpha_i y_i K(x_i, x) + b = +1$ positive gutter.
Constraint 3: $\sum_i \alpha_i y_i K(x_i, x) + b = -1$ negative gutter.
This will yield 4 equations.

C1	-1	$\alpha_A +$	-1	$\alpha_B +$	1	$\alpha_C +$	0	$b =$	0

C3.A	y_AK(A,A)=-1*0=0	$\alpha_A +$	y_BK(B,A)=-1*0=0	$\alpha_B +$	y_cK(C,A)=+1*2=2	α_C+	1	$b=$	-1
C3.B	y_AK(A,B)=-1*0=0	$\alpha_A +$	y_BK(B,B)=-1*2=-2	$\alpha_B +$	y_cK(C,B)=+1*2=2	α_C+	1	$b=$	-1
C2.C	y_AK(A,C)=-1*0=0	$\alpha_A +$	y_BK(B,C)=-1*2=-2	$\alpha_B +$	y_cK(C,C)=+1*4=4	α_C+	1	$b=$	+1

For clarity here are the four equations:

C1	$(-1)\alpha_A+(-1)\alpha_B+(+1)\alpha_C+(0)b=0$
C3.A	$(0)\alpha_A+(0)\alpha_B+(2)\alpha_C+(1)b=-1$
C3.B	$(0)\alpha_A+(-2)\alpha_B+(2)\alpha_C+(1)b=-1$
C2.C	$(0)\alpha_A+(-2)\alpha_B+(4)\alpha_C+(1)b=+1$

Step 3: Use your favorite method of solving linear equations to solve for the 4 unknowns.
Answer:

$\alpha_A=0$	$\alpha_B=1$	$\alpha_C=1$	$b=-1$

This is a more general way to solve SVM parameters, without the help of geometry. This method can be applied to problems where "margin" width or boundary equation can not be derived by inspection. (e.g. > 2D)

NOTE: We used the gutter constraints as equalities above because we are told that the given points lie on the "gutter". More realistically, if we were given more points, and not all points lay on the gutters, then we would be solving a system of *inequalities* (because the gutter equations are really constraints on >= 1 or <= -1).

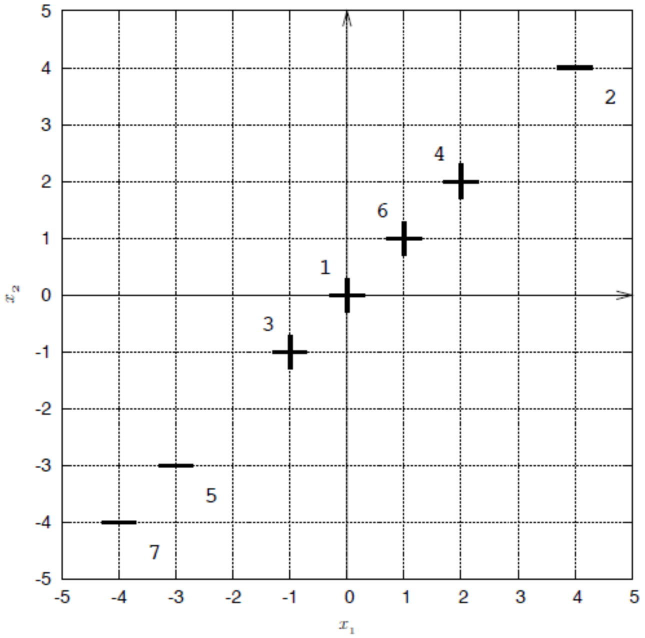
In the quadratic programming solvers used to solve SVMs, we are in fact doing just that, we are minimizing a target function by subjecting it to a system of linear inequality constraints.

Example of SVMs with a Non-Linear Kernel

From Part 2E of 2006 Q4. You are given the graph below and the following kernel:

$$K(\vec{u},\vec{v})=2\|\vec{u}\|\|\vec{v}\|$$

and you are asked to solve for equation for the decision boundary.



Step 1: First, decompose the kernel into a dot product of $\phi(\cdot)$ functions: $K(\vec{u},\vec{v})=\phi(\vec{u})\cdot\phi(\vec{v})$

Answer: $\phi(\vec{x}) = \sqrt{2}\|\vec{x}\|$

Step 2: Convert all our original points into the new space using the transform. (We are going from 2D to 1D).

Positive points are at:

$$\phi(p_1) = \sqrt{2} \cdot 0$$
$$\phi(p_3) = \sqrt{2} \cdot 1\sqrt{2} = 2$$
$$\phi(p_4) = \sqrt{2} \cdot 2\sqrt{2} = 4$$
$$\phi(p_6) = \sqrt{2} \cdot 1\sqrt{2} = 2$$

Negative points are at:

$$\phi(p_5) = \sqrt{2} \cdot 3\sqrt{2} = 6$$
$$\phi(p_2) = \phi(p_7) = \sqrt{2} \cdot 4\sqrt{2} = 8$$

Step 3: Plot the points in the new space, this appears as a line from 0 to 8.
With positive points at 0, 2, 4 and negative points at 6, 8.

The support vectors lie between $\phi(p_4)$ and $\phi(p_5)$ (between values of 4 and 6)
Hence the decision boundary (maximum margin) should be: $\phi(x) < 5$
The < due to the positive points being all less than 5.

Expanding the determined decision boundary in terms of components of x, we get:

$$\phi(x) = \sqrt{2} \cdot \sqrt{x_1^2 + x_2^2} < 5$$

Square both sides:

$$2(x_1^2 + x_2^2) < 25$$

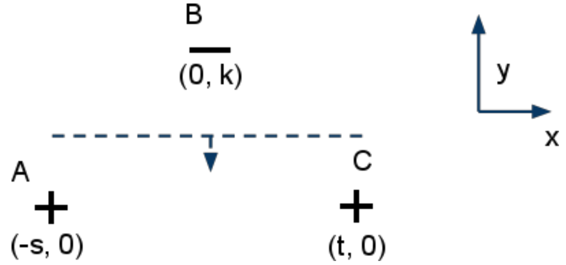
Convert to $\geq$ (standard form):

$$-2x_1^2 - 2x_2^2 + 25 \geq 0$$

$$(x_1^2 + x_2^2) < \frac{25}{2}$$

This is a circle with radius $5 / \sqrt{2} = \frac{5}{2}\sqrt{2} = 2.5\,\text{diagonals} \approx 3.5$

An Abstract Lesson on Support Vector Behavior



Suppose you have the above set of points. Let's solve the SVM parameters by inspection.

1. Boundary equation:

$$y \le \frac{k}{2} \;\text{=>}\; 0x + y - \frac{k}{2} \le 0 \;\text{=>}\; 0x - y + \frac{k}{2} \ge 0$$

2. Read off the $\vec{w}$ and b and multiply by c (c>0):

$$\vec{w} = \begin{bmatrix} 0 \\ -c \end{bmatrix} \quad b = \frac{ck}{2}$$

3. Now apply the width of the road/margin constraint:

$$\text{width of road} = \frac{2}{\|\vec{w}\|} = k$$

plugging in in length of w, and solving for c:

$$\frac{2}{\sqrt{(-c)^2}} = k \;\text{=>}\; c = \frac{2}{k}$$

4. Now we have the SVM optimal solutions to w and b:

$$\vec{w} = \begin{bmatrix} 0 \\ -\frac{2}{k} \end{bmatrix} \quad b = \frac{k2}{2k} = 1$$

5. Next, solve for the α_i using the two lagrangian equations:

$$\sum_i \alpha_i y_i \vec{x_i} = \vec{w} \text{ and } \sum_i \alpha_i y_i = 0$$

a) From expanding the first equation, we get:

$$(+1)\alpha_A \begin{bmatrix} -s \\ 0 \end{bmatrix} + (-1)\alpha_B \begin{bmatrix} 0 \\ k \end{bmatrix} + (+1)\alpha_C \begin{bmatrix} t \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{2}{k} \end{bmatrix}$$

which leads to two equations:

$$-\alpha_B k = \frac{-2}{k} \text{ or } \boxed{\alpha_B = \frac{2}{k^2}} \quad \text{and} \quad -\alpha_A s + \alpha_C t = 0 \text{ or } \alpha_C = \left(\frac{s}{t}\right)\alpha_A$$

b) From expanding the second equation $\sum_i \alpha_i y_i = 0$, we get:

$$\alpha_A(+1) + \alpha_B(-1) + \alpha_C(+1) = 0 \text{ or } \alpha_A + \alpha_C = \alpha_B \alpha_A(+1) + \alpha_B(-1) + \alpha_C(+1) = 0$$

c) Putting the equations from a) and b) together we can solve for the other two alphas.

$$\alpha_A + \left(\frac{s}{t}\right)\alpha_A = \frac{2}{k^2} \text{ or } \boxed{\alpha_A = \left(\frac{t}{s+t}\right)\frac{2}{k^2}} \text{ and similarly for } \alpha_c \; \boxed{\alpha_C = \left(\frac{s}{s+t}\right)\frac{2}{k^2}}$$

We see that the two +ve support vector alphas are split based on the ratio of distances determined by s and t. If t = s were equal, then α_A = α_C = $\frac{1}{k^2} = \frac{1}{2}\alpha_B$

Observation A:

Q: Suppose we moved point A to the origin at (0, 0). What happens to α_A and α_C?

A: This configuration basically implies s = 0; so we get: $\alpha_c = 0$ and $\alpha_A = \frac{2}{k^2}$.

Conceptually, α_A now becomes the **sole primary support vector** because point A sits directly across from point B. Point

A takes up all the share of the "pressure" in holding up the margin; point C, though still on the gutter, effectively becomes a non-support vector. So this implies that points on the gutter may not always serve the role of being a support vector.

Observation B:

Q: Suppose we changed k, by moving point B up/or down the y-axis what happens to the alphas?

A: All the alphas are proportional to $\frac{1}{k^2}$

If **k decreases**, the road **narrows**, the alphas **increases**. Analogy, *the supports need to apply more "pressure" to push the margin tighter.*

If **k increases**, the road **widens**, the alphas **decrease**. Analogy: *wider road needs less "pressure" on the supports to hold it in place*.

Boosting

The main idea in Boosting is that we are trying to combine (or ensemble) of "weak" classifiers (classifiers that underfit the data) h(x) into a single strong classifier H(x).

$$H(\vec{x}) = sign(\alpha_1 h_1(\vec{x}) + \alpha_2 h_2(\vec{x}) + ... + \alpha_s h_s(\vec{x}))$$
$$H(\vec{x}) = sign(\textstyle\sum_i^s \alpha_i h_i(\vec{x}))$$

where:

$$H(\vec{x}) \in \{-1,+1\} \quad h_i(\vec{x}) = \{-1,+1\}$$

Each data point is weighed. w_i for $i \in 1...n$. **Weights** are like probabilities, (0, 1], with $\sum_i^n w_i = 1$. But weights are never 0; this implies that all data points will have some vote at all times.

Decision stump weights:

$$\alpha_s = \ln\left(\frac{1-E^s}{E^s}\right)^{\frac{1}{2}} = \frac{1}{2}\ln\left(\frac{1-E^s}{E^s}\right)$$

Definition of Errors:

$$E^s = \sum_{incorrect} w_i \qquad (1-E^s) = \textstyle\sum_{correct} w_i$$

In Boosting we always pick stumps with errors < 1/2. Because stumps with errors > 1/2 can always be flipped. Stumps with error = 1/2 are useless because they are no better than flipping a fair coin.

$E^s < \frac{1}{2}$ and $1 - E^s > \frac{1}{2}$ so $E < (1-E)$ => $\frac{(1-E)}{E} > 1$ Therefore: $\alpha_s > 0$.

Adaboost Algorithm

Input: Training data $(\vec{x_1}, y_1) ... (\vec{x_n}, y_n)$

1. Initialize $w_i^1 = \frac{1}{n} \quad \forall\, i \in (1...n)$ a weight for each data point.
2. For s = 1 ... T:
   a. Train base learner using distribution w^s on training data. Get a base (stump) classifier $h_s(\vec{x})$ that achieves the lowest E_s (error). [Note in examples that we do in class, $h_s(\vec{x})$ are picked from a set of predefined stumps, this procedure of "picking" the best stump is the same as "training".]
   b. Compute the stump weight: $\alpha_s = \frac{1}{2}\ln\frac{(1-E_s)}{E_s}$
   c. Update weights (there are three ways to do this):
      - Original: $w_i^{s+1} = \frac{e^{-\alpha s}}{N^s} \cdot w_i^s$ (correct pts.) $w_i^{s+1} = \frac{e^{-\alpha s}}{N^s} \cdot w_i^s$ (incorrect pts.)
      - OR more human-friendly: $w_i^{s+1} = \left[\frac{1}{2} \cdot \frac{1}{1-E^s}\right] \cdot w_i^s$ (correct pts.) $w_i^{s+1} = \left[\frac{1}{2} \cdot \frac{1}{E^s}\right] \cdot w_i^s$ (incorrect pts.) (see derivation below)
      - OR use **numerator-denominator method** (see below)
1. Output the final classifier: $H(\vec{x}) = sign(\sum_s \alpha_s h_s(\vec{x}))$

Possible Termination conditions:
1. Stop after T rounds (we manually set some T.)
2. Stop after H(x) (final classifier) has error = 0 on training data or < some error threshold.
3. Stop when you can't find any more stumps h(x) where weighted error is < 0.5. (i.e. All stumps have E = 0.5).

The Numerator-Denominator method

A calculator-free method for finding weight updates quickly

Replace the Weight Update Step 1c above with these steps.

1. **Write** all weights in the form of $w_i = \frac{n_i}{d}$
where the denominator *d* is the same to all weights.

2. **Circle** the data points that are incorrectly classified.

3. Compute the **new denominator** for (the circled) incorrectly classified points:

$$d'_{incorrect} = 2\cdot\left(\sum\nolimits_{incorrect} n_i\right)$$

which is sum of all the incorrect numerators times two.
Compute the new denominator for (uncircled) correct points:

$$d'_{correct} = 2\cdot\left(\sum_{correct} n_i\right)$$

sum of all the correct numerators times two.

4. **New weights** are the old numerator divided by the updated denominators found in step 3.

$$w'_i = \frac{n_i}{d'_{incorrect}} \text{ if incorrect}$$
$$w'_i = \frac{n_i}{d'_{correct}} \text{ if correct.}$$

5. Adjust all the numerators and denominators such that the denominator is again the same for all weights. Optional: Check and make sure correct weights add up to 1/2, and incorrect weights also add up to 1/2.

A Shortcut on computing the output of H(x).

Quizzes often ask you for the Error of the final **H(x)** ensemble classifier on the training data.
Here is a quick way to compute the output of H(x) without calculating logarithms.
Step 1: compute the sign of each of stump h(x) on the given data point.
Step 2: compute products of the log arguments of the +ve stumps and -ve stump.

If $\prod_{+}\frac{1-E_s}{E_s} > \prod_{-}\frac{1-E_s}{E} \rightarrow +$ If $\prod_{+}\frac{1-E_s}{E_s} < \prod_{-}\frac{1-E_s}{E} \rightarrow -$

Example: suppose $H(\vec{x}) = \frac{1}{2} sign(\ln(5)\cdot h_1(\vec{x})) + \ln(2)\cdot h_2(\vec{x})) + \ln(2)\cdot h_3(\vec{x}))$

if $h_1(x)$ is + and $h_2(x)$ is + and $h_3(x)$ is -ve
(5 * 2) > 2 H(x) should output +ve
if $h_1(x)$ is + and $h_2(x)$ is - and $h_3(x)$ is -ve
5 > (2 * 2) H(x) should output +ve.

Step 3: Once you've computed all of the H(x) output values on the training data points, count the number of case where H(x) disagrees with the true output. That is the error.

FAQ

Dear TA, how do I determine if a stump will "never" be used (such as for part 1.A of 2006 Q4)?

Test stumps that are never used are ones that make more errors than some pre-existing test stump. In other words, if the set of mistakes stump X makes is a **superset** of errors stump Y makes, then Error(X) > Error(Y) is always true, no matter what weight distributions we use. Hence we will always chose Y over X because it makes less errors. So X will never be used!

Here is the answer to problem 1A from the 2006 Q4 with explanation. Setup: We are given the tests and the mistakes they make on the training examples, and we are asked to cross out the tests that are **never** used.

Test	Misclassified examples	**Never** used? Reason?
TRUE	1,2,3,5	Yes, superset of G=Y or U!=N
FALSE	4,6	Yes, superset of U=M
C=Y	1,6	No,
C=N	2,3,4,5	Yes, superset of G=Y or U=M
U=Y	1,2,3,6	Yes, superset of U!=N
U!=Y	4,5	Yes, superset of G=Y or U=M
U=N	4,5,6	Yes, superset of G=Y or U=M
U!=N	1,2,3	No,
U=M	4	No,
U!=M	1,2,3,5,6	Yes, superset of G=Y, C=Y or U!=N
G=Y	5	No,
G=N	1,2,3,4,6	Yes, superset of U=M, C=Y or U!=N

Food For thought:
Suppose we were to come up with a strong classifier that is a uniform combination of stumps (equal weights).
Q: How many mis-classifications would the following classifier commit?

H(x) = h(FALSE) + h(C=Y) + h(U!=Y)

A: Combining the misclassification sets of the stumps: {4, 6}, {1, 6}, {4, 5}
Points 1, 5 will be misclassified by 1 stump and correctly classified by 2 stumps.
So H(x) will be correct on 1, 5.
Points 4, 6 will be misclassified by 2 stumps and correctly classified by 1 stump.
So H(x) will misclassify 4, 6.
Therefore the points H(x) will mis-classify will be {4, 6}
How many misclassifications would H(x) = h(U=M) + h(G=Y) + h(C=Y) make?

(Optional 1) Derivation of the human-friendly weight update equations

Here is how the original weight update equations for Adaboost was derived into the more human friendly version. The original Adaboost weight update equations were

$$w_i^{s+1} = \frac{w_i^s}{N^s} \cdot e^{-\alpha_s}$$ For correctly classified samples (we ***reduce*** their weight)

$$w_i^{s+1} = \frac{w_i^s}{N^s} \cdot e^{+\alpha_s}$$ For incorrectly classified samples (we ***increase*** their weight)

Plug in alphas and redefine the exponential terms in terms of errors E:

$$w_i^{s+1} = \frac{w_i^s}{N^s} \sqrt{\frac{E^s}{1-E^s}}$$ For correctly classified examples.

$$w_i^{s+1} = \frac{w_i^s}{N^s} \sqrt{\frac{1-E^s}{E^s}}$$ For incorrectly classified examples.

Next, plug in the normalization factor (derived in Prof. Winston's handout)

$$N^s = 2\sqrt{E^s(1-E^s)}$$

Then simplifying gives us the:

$w_i^{s+1} = \left[\frac{1}{2} \cdot \frac{1}{1-E^s}\right] \cdot w_i^s$ for correctly classified examples.

$w_i^{s+1} = \left[\frac{1}{2} \cdot \frac{1}{E^s}\right] \cdot w_i^s$ for incorrectly classified samples.

To check your answers. Always sum up weights (for correct or wrong weights), they must each add up to 1/2!

$$\sum_{correct} w_i^{s+1} = \frac{1}{2} \cdot \frac{\sum_{correct} w_i^s}{1-E^s} = \frac{1}{2}$$

$$\sum_{incorrect} w_i^{s+1} = \frac{1}{2} \cdot \frac{\sum_{incorrect} w_i^s}{E^s} = \frac{1}{2}$$

(Optional 2) Proof of correctness of numerator-denominator method

In this proof we use the short hand: $w'_i \equiv w_i^{s+1}$

$$w_i = \frac{n_i}{d} \qquad w'_i = \frac{n_i}{d'}$$

In the procedure, we keep the numerator constant, only the denominator is updated during the weight update step from d to d'. Starting from the weight update equations (for correct points):

1. $w'_i = \frac{1}{2} \cdot w_i \cdot \frac{1}{1-E^s}$	2. $\frac{n_i}{d'} = \frac{1}{2} \cdot \frac{n_i}{d} \cdot \frac{1}{1-E^s}$
3. $\frac{1}{d'} = \frac{1}{2} \cdot \frac{1}{d} \cdot \frac{1}{\sum_{j \in correct} \frac{n_j}{d}}$	4. $\frac{1}{d'} = \frac{1}{2} \cdot \frac{1}{d} \cdot \frac{d}{\sum_{j \in correct} n_j}$
5. $d'_{correct} = 2 \sum_{j \in correct} n_j$	

The proof for incorrect points will yield the same result. This shows that the denominator update rule used in step 3 can be derived directly from the weight update equations so it is correct.