Massachvsetts Institvte of Technology
Department of Electrical Engineering and Computer Science
6.034 Recitation, Thursday, November 10

SVMs & Boosting Prof. Bob Berwick, 32D-728

Part I. SVMs: the Basic Ideas

Another method for classifying unseen data that avoids over-fitting. We train a system on a set of known (*x, y*) points, with known output + or –, then give it an unknown sample (as with neural nets, or k-Nearest Neighbors, or ID-trees…)

We are trying to find the decision boundary that maximizes the "margin" *m* or the "width of the street" separating the positive from the negative training data points (the "positive gutter" plus the "negative gutter"). Referring to Winston's notes, the equation for the width of the margin (aka the "street" or "road") is:

$$m = \frac{2}{\|\vec{w}\|} \text{ where } \|\vec{w}\| = \sqrt{\sum_i w_i^2} \text{ (i.e., the length of } \vec{w})$$

Thus to **maximize** the street width, we must **minimize** $\|\vec{w}\|$**,** subject to the constraint that the decision boundary classifies the training points correctly as either positive or negative, i.e.:

$$y_i(\vec{w} \bullet \vec{x}_i + b) \geq 1$$

The resulting Lagrange multiplier equation *L* that will solve this optimization problem is the following:

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

Solving the optimization problem above, we take partials with respect to *b* and then *w* (as shown below) to solve for *w, b*, and the values of alpha, α_i, parameters that determine a **unique** maximal margin (road/street) boundary line solution. On this maximum margin "street" the positive and negative data points that are on the road, with **nonzero alpha values**, are called **support vectors.** The decision boundary lies in the middle of the road. The definition of the road depends **only on the support vectors** so changing (adding or deleting) non-support vector points will not change the solution. (Note that in higher dimensions, we want the widest region separated by two planes, hyperplanes, etc.) That is, support vectors have **weights** α associated with them = amount of influence on the surrounding region = Langrangian multipliers found from the optimization problem. If a training data point receives a weight α_i of zero, this means that the data point **does not affect** the location of the decision boundary or the 'street'.

- The closer the support vectors of the opposite classes, the narrower the street, and the **greater** the weights (the alphas) for the corresponding support vectors. The support vectors have to supply more 'pressure' to push the margin tighter.
- The farther apart the support vectors of the opposite classes, the wider the street, and the **smaller** the alphas for the corresponding support vectors. The wider street needs less pressure on the supports to hold it in place.

The kernel trick.

Note that the equation *L* will in fact only require us to be able to compute the "dot product" of *w* and *x*. This amounts to being able to compute the "distance" between *w* and *x*. In ordinary Euclidean space, with an ordinary metric, this just *is* the dot product as written. However, we can always map the system to a different dimension (either higher or lower, but typically higher because this "separates" the data better), via a transform that is called ϕ. We will give examples below. The beautiful part is, we **do not need to compute this mapping explicitly**. We need only know what the corresponding **dot product** (= distance measure) is in the new space, which is called the **kernel function, *K*.** (So in a usual, normal, untransformed

Euclidean space, we have a linear Kernel function.) That is, for any two vectors *u, v*, we have that $\boldsymbol{K(u, v) = \phi(u) \cdot \phi(v)}$.

Solving for the Lagrange multiplier alphas in general requires numerical optimization methods beyond the scope of this course. In practice, one uses quadratic programming methods. On quizzes, we generally just ask you to solve for the values using algebra and/or geometry.

1. Useful equations for solving SVM questions, followed by an explicit example.

A. Equations derived from optimizing the Lagrangian:

1. Partial of the Lagrangian *L* wrt to *b*, i.e., $\frac{\partial L}{\partial b} = 0$

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

Important: The SUM of all the alphas (support vector weights) with their signs should add to 0 !!!!

2. Partial of the Lagrangian *L* wrt *w*, i.e., $\frac{\partial L}{\partial w} = 0$

There are 2 cases.

Case 1. When using a linear kernel: positive terms of the sum involve support vectors. Support vectors are training data points with alphas > 0

$\sum_i \alpha_i y_i \vec{x}_i = \vec{w}$

Case 2. The general case, where we have kernels that define the 'distance' between pts differently.

$\sum_i \alpha_i y_i \phi(\vec{x}_i) = \vec{w}$

B. Equations derived from the +/– boundaries and the constraints:

3. The decision boundary (used to classify a new data point as either + or –):

Case 1. When using a linear kernel, $K(\vec{x}_i, \vec{x}) = \vec{x}_i \bullet \vec{x}$

$h(\vec{x}) = \sum_i [(\alpha_i y_i \cdot \vec{x}_i) \cdot \vec{x}] + b \geq 0$ or $h(\vec{x}) = \vec{w} \cdot \vec{x} + b \geq 0$

Case 2. When using a general kernel function. We compute the kernel function $K(\vec{x}_i, \vec{x}) = \vec{x}_i \bullet \vec{x}$ **against each of the support vectors** x_i**. Support vectors are training data points with $\alpha_i > 0$.**

$h(\vec{x}) = \sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) + b \geq 0$

4. The positive gutter:

Case 1. When using a linear kernel:

$h(\vec{x}) = \sum_i [(\alpha_i y_i \cdot \vec{x}_i) \cdot \vec{x}] + b = 1$ or $h(\vec{x}) = \vec{w} \cdot \vec{x} + b = 1$

Case 2. When using a general kernel:

$h(\vec{x}) = \sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) + b = 1$

5. The negative gutter:

$h(\vec{x}) = \sum_i [(\alpha_i y_i \cdot \vec{x}_i) \cdot \vec{x}] + b = -1$ or $h(\vec{x}) = \vec{w} \cdot \vec{x} + b = -1$; $h(\vec{x}) = \sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) + b = -1$

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

$m = \frac{2}{\|\vec{w}\|}$ where $\|\vec{w}\| = \sqrt{\sum_i w_i^2}$ (i.e., the length of $\vec{w}$)

if just the two support vector case:

$m = \frac{2}{\|\vec{w}\|} \cdot (\vec{x}_+ - \vec{x}_-)$

(The last equation is useful in the 1-D or 2-D case where the width of the margin can be visually determined.)

As noted above, one common SVM kernel is just the **linear** one, where the kernel $K(u, v)$ is just the ordinary dot product of u and v. This is often used in document classification, with binary-valued feature vectors, and we output a +1 if the document is one class, and –1 if in another. Before looking at other kernels, let's first try an SVM problem where we have to arrive at the SVM solution visually (from the 2010 exam). Let's take a look, on your other handout,.

Let's first determine by eye where the boundary line goes. (Surprise – this has already been done for you.) What about the width of the margin m, i.e., the width of the 'street'? What is that width?

OK, the next part of the problem will ask us to find the actual equation of the SVM decision boundary. We can do this by inspection in such cases, in two steps.

Step 1. Find a (preliminary) decision boundary line, by visual inspection.

1. The decision boundary is on the line: $y = x–2$.
2. We have a positive support vector at the point (5, 1), with the associated line equation, $y = x–4$
3. We have a negative support vector at the point (3, 3) with the associated line equation, $y=x$

Step 2. We manipulate the boundary line equation to get it into the canonical form, $h(x)=w_1x + w_2y+b \geq 0$

1. $y \leq x–2$ (Why? Because positive points + are **below** the line.)
2. $x–y –2 \geq 0$

From this it *seems* like we can "read off" the 'solution' directly of: $w_1 = +1$; $w_2 = –1$; $b = –2$.
But is this correct? If we plug in these values for w into our equation for the road/street width m we get:

$$m = \frac{2}{\|\vec{w}\|} \text{ where } \|\vec{w}\| = \sqrt{\sum_i w_i^2} = \frac{2}{\sqrt{\sum_i w_i^2}} = \frac{2}{\sqrt{1+1}} = \frac{2}{\sqrt{2}}$$

Oops! This is **not** what the result for what the width is supposed to be – it is too small (by half). The street width, m, from visual inspection is $2\sqrt{2}$. The margin must be wider!

Step 3. So, one must realize that **any multiple c of the boundary equation is still the same decision boundary**. So, the general equation form below gives us the decision boundary, where we must now solve for c by using the other constraints of the problem (namely, the known street width):

$$cx - cy - 2c \geq 0$$

If we plug these values into the computation for the street/road width m, we get the following equation, because we can set it equal to the **known** street width $2\sqrt{2}$.:

$$\frac{2}{\sqrt{(c)^2 + (-c)^2}} = \frac{2}{\sqrt{2c^2}} = 2\sqrt{2}$$

$$\frac{4}{2c^2} = 8$$

$$\frac{2}{c^2} = 8$$

$$c^2 = \frac{1}{4}$$

$$c = \frac{1}{2}$$

Now we can read off the proper values for the *w* vector from the general equation:

$$cx - cy - 2c \geq 0$$

$$\frac{1}{2}x - \frac{1}{2}y - 1 \geq 0$$

$$\text{so: } \vec{w} = \begin{bmatrix} +1/2 \\ -1/2 \end{bmatrix} \text{ and } b = -1$$

The next page of the problem asks us to compute the alpha (α) values for **each** of the training points. Recall that we can do this by using Equation (2), Case 1 above (since we are using a linear kernel):

$$\sum_i \alpha_i y_i \vec{x}_i = \vec{w}$$

But before plunging ahead, **what is it we already know about the alpha values associated with NON-SUPPORT VECTORS? Answer: their alphas are 0. That is, for non-support vectors (all the points EXCEPT for the two we used) we already know that their alpha values are 0.**

As for the precisely two other data points, to find their associated α_3 and α_4 values we merely have to plug in values for (3, 3), which has an associated output of –1 (this becomes the '*y*' value in the equation above – don't get confused by the problem's notation), and for (5, 1), which has an associated output value of +1:

$$\sum_i \alpha_i y_i \vec{x}_i = \vec{w} = \text{ so } \alpha_3(-1)\begin{bmatrix} 3 \\ 3 \end{bmatrix} + \alpha_4(+1)\begin{bmatrix} 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix}$$

$$-3\alpha_3 + 5\alpha_4 = 1/2$$

$$-3\alpha_3 + \alpha_4 = -1/2 \ \therefore$$

$$4\alpha_4 = 1,\ \alpha_4 = 1/4; \quad -3\alpha_3 = -3/4; \quad \alpha_3 = 1/4$$

Now, to cement our understanding of what the support vectors *mean*, the question goes on to ask what the alpha values would be for some new points. First, what about a new data point (0, 6) classified as negative? The answer is that the alpha would be 0, because it **cannot** be a support vector – it lies behind the negative gutter line and so has no impact on the decision boundary. So what about a new data point (0, 0) also classified as negative?

More generally, we can consider what happens when we move one of the support vectors, as the last question does. What happens to its corresponding alpha value? (If it is still a support vector!). The *general* rule is that alpha will change *inversely* with the road width *m*. Since moving the support vector from (3, 3) to (4, 2) **decreases *m*,** it will **increase the alpha associated with the support vector. You can think of this intuitively as the vector having to 'push' harder on the gutter line to force it in.**

3. Other spaces, other kernels

OK, now the other big win with SVMs has to do with the ease with which you can transform from one space to another, where the data may be more easily separable. The remaining questions all ask you about that, for **different** kinds of kernels (= different ways to compute inner products, or 'distance', aka, 'similarity' of two points). That is, we need to define $K(u,v) = \varphi(u)\cdot\varphi(v)$, the dot product in the transformed space. (You should try these out in Winston's demo program to see how the decision boundaries change.)

The basic kernels we consider are these:

1. Single linear kernel. These are just straight lines in the plane (or in higher dimensions). You should remember what perceptrons can and cannot 'separate' via cuts, and this tells you what linear kernels can do. (But see below under linear combination of kernels!!!)

$K(\vec{u},\vec{v}) = (\vec{u}\cdot\vec{v}) + b$, e.g., $K(\vec{u},\vec{v}) = (\vec{u}\cdot\vec{v})$ (ordinary dot product)

2. Polynomial kernel.

$K(\vec{u},\vec{v}) = (\vec{u}\cdot\vec{v} + b)^n$, $n > 1$

eg., Quadratic kernel: $K(\vec{u},\vec{v}) = (\vec{u}\cdot\vec{v} + b)^2$

In 2-D the resulting decision boundary can look parabolic, linear, or hyperbolic depending on which terms in the expansion dominate.

3. Radial basis function (RBF) or Gaussian kernel.

$$K(\vec{u},\vec{v}) = -\exp\left(-\frac{\|\vec{u}-\vec{v}\|^2}{2\sigma^2}\right)$$

In 2-D, the decision boundaries for RBFs resemble contour circles around clusters of positive and negative points Support vectors are generally positive or negative points that are closest to the opposing cluster. The contour space that results is drawn from the sum of support vector Gaussians. Try the demo to see.

When the variance or spread of the Gaussian curve σ^2 ('sigma-squared') is large, you get 'wider' or 'flatter' Gaussians. When it is small, you get sharper Gaussians. Hence, when using a small sigma-squared, the contour density will appear closer, or tighter, around the support vector points. In 2-D, as a point gets closer to a support vector, it will approach *exp*(0)=1, and as it gets farther away, it approaches *exp*(–infinity)= 0.

Note that you can **combine** several radial basis function kernels to get a perfect fit around **any** set of data points, but this will usually amount to a typical case of over-fitting – there are 2 free parameters for every RBF kernel function.

4. Sigmoidal (tanh) kernel. This allows for a combination of linear decision boundaries, like neural nets.

$K(\vec{u},\vec{v}) = \tanh(k\vec{u}\cdot\vec{v} + b)$

$$K(\vec{u},\vec{v}) = \frac{e^{k\vec{u}\cdot\vec{v}+b} + 1}{e^{k\vec{u}\cdot\vec{v}+b} - 1}$$

The properties of this kernel function: it is similar to the sigmoid function; it ranges from –1 to +1; it approaches +1 when $x >> 0$; and it approaches –1 when $x << 0$. The resulting decision boundaries are logical combinations of linear boundaries, not that different from second-layer neurons in neural nets.

5. Linear combinations of kernels (scaling or general linear combination).

Kernel functions are closed under addition and scaling by a positive factor.

Let's practice on one more sample problem. There are also 2 appendices at the end of this handout for folks that are hard-core, one working out the math, the second, solving the XOR problem using SVMs, by using a polynomial kernel function.

Part II. Boosting and the Adaboost algorithm

0. The idea behind **boosting** is to find a weighted combination of s "weak" classifiers (classifiers that underfit the data and still make mistakes, though as we will see they make mistakes on less than ½ the data), h_1, h_2...,h_s, into a **single strong** classifer, $H(x)$. This will be in the form:

$$H(\vec{x}) = sign(\alpha_1 h_1(\vec{x}) + \alpha_2 h_2(\vec{x}) + \cdots + \alpha_s h_s(\vec{x})$$

$$H(\vec{x}) = sign\left(\sum_{i=1}^{s} a_i h_i(\vec{x})\right)$$

where: $H(\vec{x}) \in \{-1,+1\}, h_i(\vec{x}) \in \{-1,+1\}$

Recall that the *sign* function simply returns +1 if weighted sum is positive, and –1 if the weighted sum is negative (i.e., it classifies the data point as + or –).

Each training data point is **weighted.** These weights are denoted w_i for i=1, ..., n. **Weights** are *like* probabilities, from the interval (0, 1], with their **sum equal to 1. BUT** weights are **never 0**. This implies that **all data points have some vote** on what the classification shuld be, at all times. (You might contrast that with SVMs.)

The general idea will be to pick a single 'best' classifier h (one that has the lowest error rate when acting all alone), as an initial 'stump' to use. Then, we will **boost** the weights of the data points that this classifier **mis-classifies (makes mistakes on),** so as to focus on the next classifier h that does best on the re-weighted data points. This will have the effect of trying to fix up the errors that the first classifier made. Then, using this next classifier, we repeat to see if we can now do better than in the first round, and so on. In computational practice, we use the same sort of entropy-lowering function we used with ID/classifier trees: the one to pick is the one that lowers entropy the most. But usually we will give you a set of classifiers that is easier to 'see', or will specify the order.

In Boosting we always pick these initial 'stump' classifiers so that the error rate is strictly < ½. Note that if a stump gives an error rate greater than ½, this can always be 'flipped' by reversing the + and – classification outputs. (If the stump said –, we make it +, and vice-versa.) Classifiers with error exactly equal to ½ are useless because they are no better than flipping a fair coin.

1. Here are the definitions we will use.

Errors:

The error rate of a classifier s, E^s, is simply the sum of all the *weights* of the training points classifier h_s gets **wrong**.

$(1–E^s)$ is 1 minus this sum, the sum of all the *weights* of the training points classifier h_s gets **correct.**

By assumption, we have that:

$E^s < ½$ and $(1– E^s) > ½$, so $E^s < (1– E^s)$, which implies that $(1– E^s)/E^s > 1$

Weights:

α_s is **defined** to be ½ ln[$(1– E^s)/ E^s$)], so from the definition of weights, the quantity inside the ln term is > 1, so all alphas must be positive numbers.

Let's write out the Adaboost algorithm and then run through a few iterations of an example problem.

2. Adaboost algorithm

Input: training data, $(\vec{x}_1, y_1),\ldots,(\vec{x}_n, y_n)$

1. **Initialize data point weights.**

   Set $w_i^1 = \frac{1}{n} \ \forall i \in (1,\ldots,n)$

2. **Iterate over all 'stumps':** for s=1, ..., T

   a. **Train base** learner using distribution w^s on training data.
   Get a base (stump) classifier $h_s(x)$ that achieves the lowest error rate E^s .
   (In examples, these are picked from pre-defined stumps.)

   b. **Compute the stump weight**: $\alpha_s = \frac{1}{2}\ln\frac{(1-E^s)}{E^s}$

   c. **Update weights** (3 ways to do this; we pick Winston's method)

   For points that the classifier **gets correct,** $w_i^{s+1} = \left[\frac{1}{2}\cdot\frac{1}{1-E^s}\right]\cdot w_i^s$

   (Note from above that $1-E^s > ½$, so the fraction $1/(1-E^s)$ must be < 2, so the total factor scaling the old weight must be < 1, i.e., the **weight of correctly classified points must go DOWN in the next round)**

   For points that the classifier **gets incorrect,** $w_i^{s+1} = \left[\frac{1}{2}\cdot\frac{1}{E^s}\right]\cdot w_i^s$

   (Note from above that $E^s < ½$, so the fraction $1/E^s$) must be > 2, so the total factor scaling the old weight must be > 1, i.e., the **weight of incorrectly classified points must go UP in the next round**)

3. **Termination:**

   If $s > T$ or if $H(x)$ has error 0 on training data or < some error threshold, exit;

   If there are no more stumps h where the weighted error is < ½, exit (i.e., all stumps now have error exactly equal to ½)

4. **Output final classifier:**

$H(\vec{x}) = sign\left(\sum_{i=1}^{s} a_i h_i(\vec{x})\right)$ **[this is just the weighted sum of the original stump classifiers]**

Note that test stump classifiers 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 weight distributions we use. Therefore, we will **always** pick Y over X because it makes fewer errors. So X will **never** be used!

3. Let's try a boosting problem from an exam (on the other handout).

4. Food for thought questions.

1. How does the weight α^s given to classifier h_s relate to the performance of h_s as a function of the error E^s?
2. How does the error of the classifier E^s affect the new weights on the samples? (How does it raise or lower them?)
3. How does AdaBoost end up treating outliers?
4. Why is not the case that new classifiers "clash" with the old classifiers on the training data?
5. Draw a picture of the training error, theoretical bound on the true error, and the typical test error curve.
6. Do we expect the error of new weak classifiers to increase or decrease with the number of rounds of estimation and re-weighting? Why or why not?

Answers to these questions:

1. How does the weight α^s given to classifier h_s relate to the performance of h_s as a function of the error E^s?

Answer: The lower the error the better the classifier h is on the (weighted) training data, and the larger the weight α^t we give to the classifier output when classifying new examples.

2. How does the error of the classifier E^s affect the new weights on the samples? (How does it raise or lower them?)

Answer: The lower the error, the better the classifier h classifies the (weighted) training examples, hence the larger the increase on the weight of the samples that it classifies incorrectly and similarly the larger the decrease on those that it classifies correctly. More generally, the smaller the error, the more significant the change in the weights on the samples. Note that this dependence can be seen indirectly in the AdaBoost algorithm from the weight of the corresponding classifier α_t. The lower the error E^t, the larger α_t, the better h_t is on the (weighted) training data.

3. How does AdaBoost end up treating outliers?

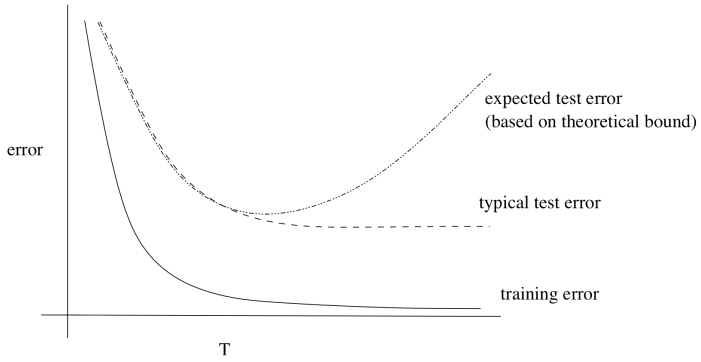
Answer: AdaBoost can help us identify outliers since those examples are the hardest to classify and therefore their weight is likely to keep increasing as we add more weak classifiers. At the same time, the theoretical bound on the training error implies that as we increase the number of base/weak classifiers, the final classifier produced by AdaBoost will classify all the training examples. This means that the outliers will eventually be "correctly" classified from the standpoint of the training data. Yet, as expected, this might lead to overfitting.

4. Why is not the case that new classifiers "clash" with the old classifiers on the training data?

Answer: The intuition is that, by varying the weight on the examples, the new weak classifiers are trained to perform well on different sets of examples than those for which the older weak classifiers were trained on. A similar intuition is that at the time of classifying new examples, those classifiers that are not trained to perform well in such examples will cancel each other out and only those that are well trained for such examples will prevail, so to speak, thus leading to a weighted majority for the correct label.

5. Draw a picture of the training error, theoretical bound on the true error, and the typical test error curve.

Answer:



6. Do we expect the error of new weak classifiers to increase or decrease with the number of rounds of estimation and re-weighting? Why?

Answer: We expect the error of the weak classifiers to increase in general since they have to perform well in those examples for which the weak classifiers found earlier did not perform well. In general, those examples will have a lot of weight yet they will also be the hardest to classify correctly.

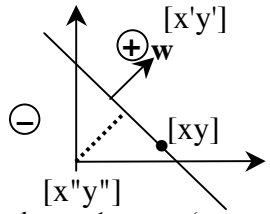
Appendix I. Gory details for SVMs [to be read at your non-leisure]

Equations

a. **Training**: maximize width of street by maximizing the equation of support vector weights (α_i) and the dot products of the support vectors: $L_{DUAL}=\Sigma\alpha_i - 1/2\ \Sigma_i\Sigma_j\alpha_i\alpha_j y_i y_j(\mathbf{x_i}\bullet\mathbf{x_j})$

b. **Classification:** assigns unknown point **u** to either one of 2 classes: + or – class:

w•u + b > 0 then +
w•u + b < 0 then –
o.w., ??? so assume –



b= –d
w•[xy] = d
w•[x'y'] > d
w•[x"y"] < d

c. Support vector constraints: $\mathbf{w}\bullet\mathbf{x}_+ + b = 1$; $\mathbf{w}\bullet\mathbf{x}_- + b = -1$ or $y_i(\mathbf{w}\bullet\mathbf{x_i} + b) = 1$ where y_i= +1 or –1

d. $\mathbf{w} = \Sigma\boldsymbol{\alpha}_i y_i \mathbf{x_i}$ is a function of weights on the support vectors, which are 0 for non-support vectors (Thus non-support vectors contribute nothing to where the decision boundary goes.)

e. $\Sigma\boldsymbol{\alpha}_i y_i = 0$ (The constraints that must be obeyed to correctly classify the training points.)

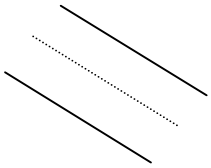
Now let's do the math to see where this all comes from

0. The two gutter lines are: $\bar{w}\cdot\mathbf{x}_1 + b = +1;\ \bar{w}\cdot\mathbf{x}_2 + b = -1$

1. From Winston's picture, the street width is found as:

$$\bar{w}\cdot(\mathbf{x}_1 - \mathbf{x}_2) = 2$$

$$\frac{\bar{w}}{\|\bar{w}\|}\cdot(\mathbf{x}_1 - \mathbf{x}_2) = \frac{2}{\|\bar{w}\|}$$



2. To maximize this, we minimize $\|\bar{w}\|$, which is the same as minimizing ½ $\|\bar{w}\|^2$ (Why this? We will be taking the derivative in just a bit, and the derivative of this is just $\bar{w}$ **.)**

3. So we want to:

A. *minimize* ½ $\|\bar{w}\|^2$ s.t. $y_i(\bar{w}\cdot\bar{x}_i + b) - 1 = 0$

4. To do the minimization, we use a Lagrangian formulation, to add a penalty function in order to ensure that the constraints on correctly classifying the training points are obeyed:

B. *minimize L* where $L=\frac{1}{2}\|\bar{w}\|^2 - \sum_{i=1}^{r}\alpha_i[y_i(\bar{w}\cdot\bar{x}_i + b) - 1]$

(See last page for cheat sheet on how Lagrange multipliers work.)

5. Take partial derivatives with respect to the 2 variables, $\bar{w}$ and b:

C.

$$\frac{\partial L}{\partial \bar{w}} = 0 \quad \frac{\partial L}{\partial b} = 0$$

So this means we have:

C.1 $\dfrac{\partial \frac{1}{2}\|\bar{w}\|^2}{\partial \bar{w}} = \bar{w}$

C.2 $\dfrac{\partial L}{\partial \bar{w}} = \bar{w} - \sum_{i=1}^{r}\alpha_i y_i \bar{x}_i \Rightarrow \bar{w} = \sum_{i=1}^{r}\alpha_i y_i \bar{x}_i$ **we substitute for $\bar{w}$ in (B) above to get C.4**

C.3 $\frac{\partial L}{\partial b} = \sum_{i=1}^{r} \alpha_i y_i = 0$ = **basic constraint on support vectors** α_i

C.4 Substituting for $\|\bar{w}\| = \sum_{i=1}^{r} \alpha_i y_i \bar{x}_i$:

$$L = \tfrac{1}{2}\|\bar{w}\|^2 - \sum_{i=1}^{r} \alpha_i [y_i(\bar{w}_i \cdot \bar{x}_i + b) - 1]$$

$$= \tfrac{1}{2}(\sum_{i=1}^{r} \alpha_i y_i \bar{x}_i) \cdot (\sum_{j=1}^{r} \alpha_j y_j \bar{x}_j) - (\sum_{i=1}^{r} \alpha_i y_i \bar{x}_i)(\sum_{j=1}^{r} \alpha_j y_j \bar{x}_j) - b\sum_{i=1}^{r} \alpha_i y_i + \sum_{i=1}^{r} \alpha_i$$

(0 from constraint **C.3**)

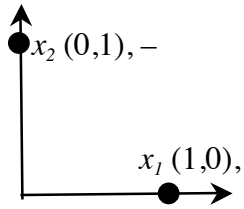
$$= \sum_{i=1}^{r} \alpha_i + \tfrac{1}{2}\sum_{i=1}^{r}\sum_{j=1}^{r} \alpha_i \alpha_j y_i y_j \bar{x}_i \cdot \bar{x}_j - \sum_{i=1}^{r}\sum_{j=1}^{r} \alpha_i \alpha_j y_i y_j \bar{x}_i \cdot \bar{x}_j$$

$$= \sum_{i=1}^{r} \alpha_i - \tfrac{1}{2}\sum_{i=1}^{r}\sum_{j=1}^{r} \alpha_i \alpha_j y_i y_j \bar{x}_i \cdot \bar{x}_j$$

C.5 *maximize* $L_{Dual} = \sum_{i=1}^{r} \alpha_i - \tfrac{1}{2}\sum_{i=1}^{r}\sum_{j=1}^{r} \alpha_i \alpha_j y_i y_j \bar{x}_i \cdot \bar{x}_j$

(IMPORTANT: Note <u>dot product</u> of training data, the last 2 terms! This is the "similarity" measure.)

<u>Worked Math example, linear SVM, 2 support vector "points" (one positive example, one negative example)</u>



$$y_i(\bar{w} \cdot \bar{x}_i + b) = 1; \quad y_i(\bar{w} \cdot \bar{x}_i + b) - 1 = 0$$

$$\bar{w} = \sum_{i=1}^{2} \alpha_i y_i \bar{x}_i$$

$$\sum_{i=1}^{2} \alpha_i y_i = 0$$

Find: $\alpha_1, \alpha_2, \bar{w}, b$:

$$L = \tfrac{1}{2}\|\bar{w}\|^2 - \sum_{i=1}^{r} \alpha_i [y_i(\bar{w} \cdot \bar{x}_i + b) - 1]$$

$$= \tfrac{1}{2}\|\bar{w}\|^2 - \alpha_1(+1 \cdot (\bar{w} \cdot \bar{x}_1 + b) - 1) - \alpha_2(-1 \cdot (\bar{w} \cdot \bar{x}_2 + b) - 1)$$

$$\frac{\partial L}{\partial \bar{w}} = \bar{w} = \alpha_1 \bar{x}_1 - \alpha_2 \bar{x}_2$$

$$\frac{\partial L}{\partial b} = -\alpha_1 + \alpha_2 = 0; \quad \therefore \ \alpha_1 = \alpha_2$$

Substituting for $\bar{w} = \alpha_1 \bar{x}_1 - \alpha_2 \bar{x}_2$ from above in the equation for *L*, we have:

$$L = \tfrac{1}{2}[\alpha_1 \bar{x}_1 - \alpha_2 \bar{x}_2]^2 - \alpha_1[+1 \cdot ((\alpha_1 \bar{x}_1 - \alpha_2 \bar{x}_2) \cdot \bar{x}_1 + b) - 1] - \alpha_2[-1 \cdot ((\alpha_1 \bar{x}_1 - \alpha_2 \bar{x}_2) \cdot \bar{x}_2 + b) - 1]$$

$$= \tfrac{1}{2}[\alpha_1^2 \bar{x}_1 \cdot \bar{x}_1 - \alpha_1 \alpha_2 \bar{x}_1 \cdot \bar{x}_2 + \alpha_2^2 \bar{x}_2 \cdot \bar{x}_2] - \alpha_1[\alpha_1 \bar{x}_1 \cdot \bar{x}_1 - \alpha_2 \bar{x}_2 \cdot \bar{x}_1 + b - 1] - \alpha_2[-\alpha_1 \bar{x}_1 \cdot \bar{x}_2 + \alpha_2 \bar{x}_2 \cdot \bar{x}_2 - b - 1$$

Since
$\bar{x}_1 \cdot \bar{x}_1 = 1;\ \bar{x}_2 \cdot \bar{x}_2 = 1;\ \bar{x}_1 \cdot \bar{x}_2 = 0$ (here is where we use the dot product!), we can simplify this to:

$$L = \tfrac{1}{2}[\alpha_1^2 + \alpha_2^2] - \alpha_1^2 - \alpha_1 b + \alpha_1 - \alpha_2^2 + \alpha_2 b + \alpha_2$$

Since $\alpha_1 = \alpha_2$

$$L = \tfrac{1}{2}[2\alpha_1^2] - 2\alpha_1^2 + 2\alpha_1 = -\alpha_1^2 + 2\alpha_1$$

Since we want to maximize (minimize) *L,*

$\frac{\partial L}{\alpha_1} = 0 = -2\alpha_1 + 2 \Rightarrow \alpha_1 = 1;$ since $\alpha_1 = \alpha_2,\ \alpha_2 = 1$ (Note that $\sum_{i=1}^{2} \alpha_i y_i = 1(+1) + 1(-1) = 0$ as required)

Finally, to find *b*:

$$y_i(\bar{w} \cdot \bar{x}_i + b) = 1$$

$$\bar{w} = \alpha_1 \bar{x}_1 - \alpha_2 \bar{x}_2 = \bar{x}_1 - \bar{x}_2$$

Substituting for $\bar{w}$,

$$1 \cdot (\bar{x}_1 - \bar{x}_2) \cdot \bar{x}_1 + b = 1$$

$\bar{x}_1 \cdot \bar{x}_1 - \bar{x}_2 \cdot \bar{x}_1 + b = 1,$ but $\bar{x}_1 \cdot \bar{x}_1 = 1$ and $\bar{x}_2 \cdot \bar{x}_1 = 0,$ so

$$b = 0$$

Alternatively, note that decision boundary goes through the origin, so *b*=0. Final decision boundary line is therefore just the line $x_1 = x_2$, i.e., line through origin at 45 degrees, with slope 1.

Lagrange Multipliers Cheat Sheet

1. A method for finding the maximum or minimum or a function, subject to constraints.
2. Key idea: define a new function, L, in terms of the original function, $f(x,y)$, the constraint equation $g(x,y)$, and a new variable, the "Langrange multiplier" λ, which is a penalty for when the constraint equation is violated (not equal to zero), i.e., we want to maximize $f(x,y)$ s.t. $g(x,y)=0$.

$$L = f(x,y) + \lambda g(x,y)$$

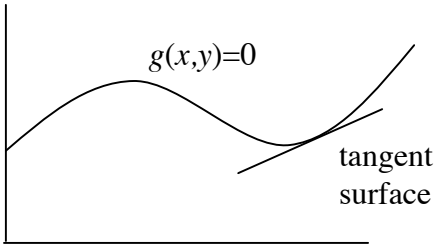
3. To do this we take partial derivatives, with respect to x, y, and λ:

$$\frac{\partial L}{\partial x} = \frac{\partial f(x,y)}{\partial x} + \lambda \frac{\partial g(x,y)}{\partial x} = 0$$
$$\frac{\partial L}{\partial y} = \frac{\partial f(x,y)}{\partial y} + \lambda \frac{\partial g(x,y)}{\partial y} = 0$$
$$\frac{\partial L}{\partial \lambda} = g(x,y) = 0$$

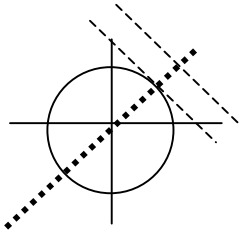
4. Graphical intuition. As we travel along the constraint surface $g(x,y)=0$, clearly $g(x,y)$ does not change, so the partials of g with respect to both x and y must be 0, i.e., $\frac{\partial g(x,y)}{\partial x} = 0 \quad \frac{\partial g(x,y)}{\partial y} = 0$

When we get to the maximum of $f(x,y)$, $\frac{\partial f(x,y)}{\partial x} = 0$ and $\frac{\partial f(x,y)}{\partial y} = 0$ because slope at max $=0$.

Picture:



tangent for $f(x,y)$ along g surface



Example: $f = x+y$; $g=x^2+y^2$ (constraint is a circle)

$L = (x+y) + \lambda(x^2+y^2-1)$

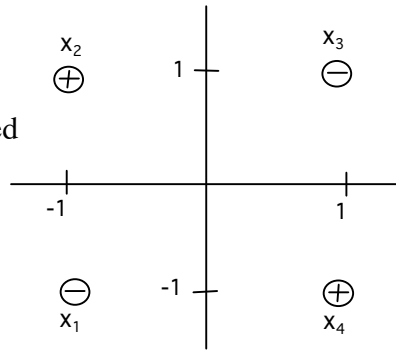
$$\frac{\partial L}{\partial x} = 1 + 2\lambda x = 0$$
$$\frac{\partial L}{\partial y} = 1 + 2\lambda y = 0$$
$$\lambda = \frac{-1}{2x}$$
$$1 + 2(\frac{-1}{2x})y = 0 \quad \Rightarrow y = x$$

From constraint as circle:

$x^2 + y^2 = 2x^2 = 1 \quad \therefore \; x=\sqrt{½} = y$

Appendix II. SVMs, another worked example, for a polynomial kernel function, XOR problem

The input data are shown in the graph in the original space.



For this problem, the polynomial kernel transform $K = (1 + v_1.v_2)^2$ is used

1. Note that $K(x_i\ x_j)=K(x_j\ x_i)$

$K(x_1,x_1) = (1 + [-1\ -1].[-1\ -1])^2$
$= (1 + \sqrt{2}\ \sqrt{2}\cos\theta)^2$ [or $(1 + (-1)(-1) + (-1)(-1))^2$]
$= (1 + 2)^2 = 9$

$K(x_1,x_2) = (1 + [-1\ -1].[-1\ +1])^2$
$= (1 + \sqrt{2}\ \sqrt{2}\cos\theta)^2$ [or alternatively: $(1+(-1)(-1)+(-1)(+1))^2$
$= (1 + 0)^2 = 1$

$K(x_1,x_3) = (1 + [-1\ -1].[+1\ +1])^2 = 1$; $K(x_1,x_4) = (1 + [-1\ -1].[+1\ -1])^2 = 1$
$K(x_2,x_2) = (1 + [-1\ +1].[-1\ +1])^2 = 9$; $K(x_2,x_3) = (1 + [-1\ +1].[+1\ +1])^2 = 1$
$K(x_2,x_4) = (1 + [-1\ -1].[+1\ -1])^2 = 1$
$K(x_3,x_3) = (1 + [-1\ -1].[-1\ -1])^2 = 9$; $K(x_3,x_4) = (1 + [-1\ -1].[+1\ -1])^2 = 1$;
$K(x_4,x_4) = (1 + [+1\ -1].[+1\ -1])^2 = 9$

2. Enter L_{Dual} as an algebraic expression of a_1, a_2, a_3, a_4.

Sum the alphas (a_1 to a_4), then sum over **all** dot products multiplied by their a's (alphas) and y_i multipliers, either +1 or –1 whether the data sample point is positive or negative (given in the data table).

$$L_{Dual} = \sum_i a_i - \tfrac{1}{2}\sum_i\sum_j a_i a_j y_i y_j x_i.x_j$$

$= (a_1 + a_2 + a_3 + a_4) - \tfrac{1}{2}((a_1a_1(-1)(-1)K(x_1\ x_1)+a_1a_2(-1)(1)K(x_1\ x_2) +a_1a_3(-1)(-1)K(x_1\ x_3) +a_1a_4(-1)(1)K(x_1\ x_4) + a_2a_1(1)(-1)K(x_2\ x_1)+a_2a_2(1)(1)K(x_2\ x_2)+ a_2a_3(1)(-1)K(x_2\ x_3)+ a_2a_4(1)(1)K(x_2\ x_4)$
$+ a_3a_1(-1)(-1)K(x_3\ x_1)+ a_3a_2(-1)(1)K(x_3\ x_2)+ a_3a_3(-1)(-1)K(x_3\ x_3)+ a_3a_4(-1)(1)K(x_3\ x_4)$
$+ a_4a_1(1)(-1)K(x_4\ x_1)+ a_4a_2(1)(1)K(x_4\ x_2)+ a_4a_3(1)(-1)K(x_4\ x_3)+ a_4a_4(1)(1)K(x_4\ x_4))$

$=(a_1+a_2+a_3+a_4)-\tfrac{1}{2}(9a_1^2-a_1a_2+a_1a_3-a_1a_4-a_2a_1+9a_2^2-a_2a_3+a_2a_4+a_3a_1-a_3a_2+9a_3^2-a_3a_4-a_4a_1+a_4a_2-a_4a_3+9a_4^2)$
$=(a_1+a_2+a_3+a_4)-\tfrac{1}{2}(9a_1^2-2a_1a_2+2a_1a_3-2a_1a_4+9a_2^2-2a_2a_3+2a_2a_4+9a_3^2-2a_3a_4+9a_4^2)$

3. You can optimize the above equation by setting to zero its partial derivatives with respect to each a_i.

$$dL/da_1 = 1 - 9a_1 + a_2 + a_3 - a_4$$
$$dL/da_2 = 1 + a_1 - 9a_2 + a_3 - a_4$$
$$dL/da_3 = 1 + a_1 - a_2 - 9a_3 + a_4$$
$$dL/da_4 = 1 + a_1 - a_2 + a_3 - 9a_4$$

4. Solving these simultaneous equations, yields $a_i = 1/8$. (As a check, $\Sigma a_i y_i = 0$, as it should.)

5. For a polynomial kernel, the degree of the polynomial and the original number of features determine the ultimate number of features in the transformed space (basically all combinations of input features up to the specified degree, with some terms scaled by the magnitude of the feature vector)

for $x_i = [a\ b]$ (a and b are just used here to designate first and second feature values, respectively)
Phi(x_i) = $[a^2\ \ b^2\ \ ab\sqrt{2}\ \ a\sqrt{2}\ \ b\sqrt{2}\ \ 1]$

Substituting values for the four data points:
x_1 [–1 –1]: Phi(x_1) = $[1\ \ 1\ \ \sqrt{2}\ \ -\sqrt{2}\ \ -\sqrt{2}\ \ 1]$
x_2 [–1 +1]: Phi(x_2) = $[1\ \ 1\ \ -\sqrt{2}\ \ -\sqrt{2}\ \ \sqrt{2}\ \ 1]$
x_3 [+1 –1]: Phi(x_3) = $[1\ \ 1\ \ -\sqrt{2}\ \ \sqrt{2}\ \ -\sqrt{2}\ \ 1]$
x_4 [+1 +1]: Phi(x_4) = $[1\ \ 1\ \ \sqrt{2}\ \ \sqrt{2}\ \ \sqrt{2}\ \ 1]$

6. The classification (aka weight) vector $\mathbf{w} = \Sigma a_i y_i x_i$:
x_1: $1/8(-1)[1\ \ 1\ \ \sqrt{2}\ \ -\sqrt{2}\ \ -\sqrt{2}\ \ 1]$
x_2: $1/8(1)[1\ \ 1\ \ -\sqrt{2}\ \ -\sqrt{2}\ \ \sqrt{2}\ \ 1]$
x_3: $1/8(1)[1\ \ 1\ \ -\sqrt{2}\ \ \sqrt{2}\ \ -\sqrt{2}\ \ 1]$
x_4: $1/8(-1)\ [1\ \ 1\ \ \sqrt{2}\ \ \sqrt{2}\ \ \sqrt{2}\ \ 1]$

Adding these vectors as per the above equation, $\mathbf{w} = [0\ \ 0\ \ -\frac{\sqrt{2}}{2}\ \ 0\ \ 0\ \ 0]$

7. To calculate b, substitute w and one of the data samples into any constraint equation:

$\mathbf{w} . x_1 + b = -1$

$\mathbf{w} . x_1$: $[0\ \ 0\ \ -\frac{\sqrt{2}}{2}\ \ 0\ \ 0\ \ 0] . [1\ \ 1\ \ \sqrt{2}\ \ -\sqrt{2}\ \ -\sqrt{2}\ \ 1] = 0 + 0 + -1 + 0 + 0 + 0 = -1$

so $-1 + b = -1$, so $b = 0$. Alternatively, you could notice that b passes through the origin, so it is 0.

8. To find an algebraic expression for the classifier function, $\mathbf{w} . \boldsymbol{x} + b$, substitute in Phi($x$) for $\boldsymbol{x}$:
($\mathbf{x} = [x_1\ x_2]$ instead of $[a\ b]$ as described above)

$\mathbf{w} .$ Phi($\boldsymbol{x}$) $+ b = (0*x_1 + 0*x_2 + -\frac{\sqrt{2}}{2}\sqrt{2}\,x_1x_2 + 0*\sqrt{2}\,x_1 + 0*\sqrt{2}\,x_2 + 0*1) + 0 = -x_1x_2$

9. For each test point, substitute the vector into the above equation and see if we get the correct output:
$f([-1, -1]) = -(-1)(-1) = 1$ (correct, a positive sample point)
$f([-1, 1]) = -(-1)(1) = -1$ (correct, a negative sample point)
etc.
$f([2, 2]) = -4$

10. A fifth data point at [2 2] with a y_i value of –1 (a negative example) would have an a_i (i.e. a_5) value of 0 because it would **not** be a support vector. You can see this two ways: Its classifier value, calculated above, is –4, and we know support vectors have values of 1 or –1; or you could plot it on the graph and notice that it does not change the decision boundary.