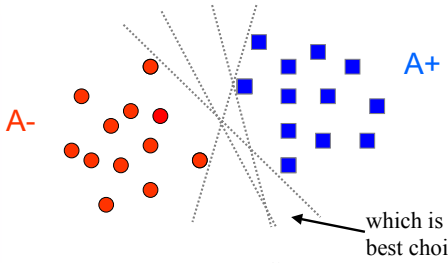
Support Vector Machines (SVMs)

- Learning mechanism based on linear programming
- Chooses a separating plane based on maximizing the notion of a margin
  - Based on PAC learning
- Has mechanisms for
  - Noise
  - Non-linear separating surfaces (kernel functions)
- Notes based on those of Prof. Jude Shavlik

Support Vector Machines

Find the best separating plane in feature space
- many possibilities to choose from



SVMs – The General Idea

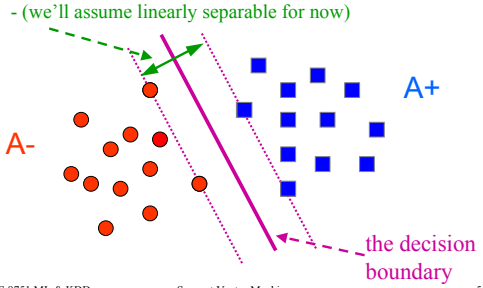
- How to pick the best separating plane?
- Idea:
  - Define a set of inequalities we want to satisfy
  - Use advanced optimization methods (e.g., linear programming) to find satisfying solutions
- Key issues:
  - Dealing with noise
  - What if no good linear separating surface?

Linear Programming

- Subset of Math Programming
- Problem has the following form:
  function $f(x_1,x_2,x_3,\ldots,x_n)$ to be maximized
  subject to a set of constraints of the form:
  $g(x_1,x_2,x_3,\ldots,x_n) > b$
- Math programming - find a set of values for the variables $x_1,x_2,x_3,\ldots,x_n$ that meets all of the constraints and maximizes the function f
- Linear programming - solving math programs where the constraint functions and function to be maximized use **linear combinations** of the variables
  - Generally easier than general Math Programming problem
  - Well studied problem

Maximizing the Margin

The margin between categories
- want this distance to be maximal
- (we'll assume linearly separable for now)



PAC Learning

- PAC – Probably Approximately Correct learning
- Theorems that can be used to define bounds for the risk (error) of a family of learning functions
- Basic formula, with probability $(1 - \eta)$:

$$R(\alpha) \le R_{emp}(\alpha) + \sqrt{\frac{h(\log(2N/h) - \log(\eta/4)}{N}}$$

- R – risk function, α is the parameters chosen by the learner, N is the number of data points, and h is the VC dimension (something like an estimate of the complexity of the class of functions)

Margins and PAC Learning

- Theorems connect PAC theory to the size of the ***margin***
- Basically, the ***larger*** the margin, the better the expected accuracy
- See, for example, Chapter 4 of *Support Vector Machines* by Christianini and Shawe-Taylor, Cambridge University Press, 2002

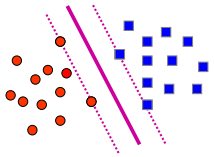
Some Equations

Separating Plane

$\vec{w} \cdot \vec{x} = \gamma$

$\vec{w}$ - weights, $\vec{x}$ - input features,

γ - threshold



For all positive examples

$\vec{w} \cdot \vec{x}_{pos} = \gamma + 1$

For all negative examples

$\vec{w} \cdot \vec{x}_{neg} = \gamma - 1$

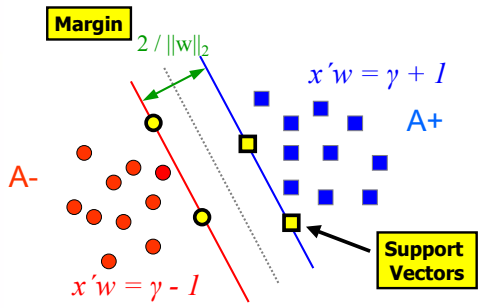
1s result from dividing through by a constant for convenience

Distance between blue and red planes (the margin)

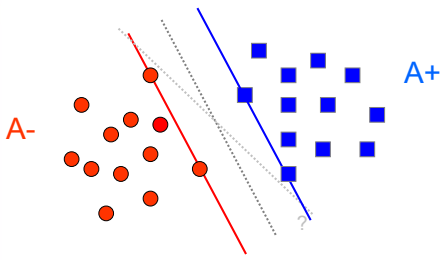
$$\text{margin} = \frac{2}{\|w\|}$$

Euclidean length ("2 norm") of the weight vector

What the Equations Mean



Choosing a Separating Plane



Our "Mathematical Program" (so far)

$$\min_{\vec{w},\gamma} \|\vec{w}\|^2$$

for technical reasons easier to optimize this "quadratic program"

such that

$\vec{w} \cdot \vec{x}_{pos} \geq \gamma + 1$ (for + examples)

$\vec{w} \cdot \vec{x}_{neg} \leq \gamma - 1$ (for – examples)

Note : $\vec{w}, \gamma$ are our adjustable parameters (we could, of course, use the ANN "trick" and move γ to the left side of our inequalities)

We can now use existing math programming optimization software to find a solution to the above (a global optimal soln)

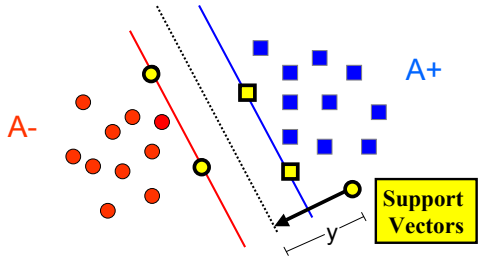
Dealing with Non-Separable Data

We can add what is called a "slack" variable to each example

This variable can be viewed as:

- 0 if the example is correctly separated
- y "distance" we need to move example to make it correct (i.e., the distance from its surface)

"Slack" Variables



The Math Program with Slack Variables

$$\min_{\vec{w},\vec{s},\gamma} \|\vec{w}\|^2 + \mu\|\vec{s}\|_1$$

$\vec{w}$ – one for each input feature

$\vec{s}$ – one for each example

μ – scaling constant

$\|\vec{s}\|_1$ – "one norm" - sum of components (all positive)

such that

$$\vec{w}\cdot\vec{x}_{pos_i} + s_i \geq \gamma + 1$$

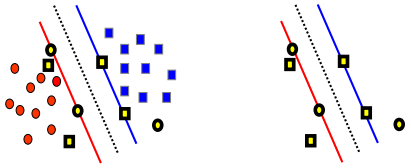
$$\vec{w}\cdot\vec{x}_{neg_j} - s_j \leq \gamma - 1$$

$$\forall_k s_k \geq 0$$

This is the "traditional" Support Vector Machine

Why the word "Support"?

- All those examples **on** or on the **wrong side** of the two separating planes are the support vectors
  - We'd get the same answer if we deleted all the non-support vectors!
  - i.e., the "support vectors [examples]" support the solution



PAC and the Number of Support Vectors

- The fewer the support vectors, the better the generalization will be
- Recall, non-support vectors are
  - Correctly classified
  - Don't change the learned model if left out of the training set
- So

$$\text{leave-one-out error rate} \leq \frac{\#\text{ support vectors}}{\#\text{ training examples}}$$

Finding Non-Linear Separating Surfaces

- Map inputs into new space

Example: features x_1 x_2

5 4

Example: features x_1 x_2 x_1^2 x_2^2 x_1*x_2

5 4 25 16 20

- Solve SVM program in this new space
  - Computationally complex if many features
  - But a clever trick exists

The Kernel Trick

- Optimization problems often/always have a "primal" and a "dual" representation
  - So far we've looked at the primal formulation
  - The dual formulation is better for the case of a non-linear separating surface

Perceptrons Re-Visited

In perceptrons, if T $\Rightarrow +1$ and F $\Rightarrow$ -1,

$\vec{w}_{k+1} = \vec{w}_{k+1} + \eta\, y_i \vec{x}_i$

if the example x_i is currently misclassified

So

$$\vec{w}_{final} = \sum_{i=1}^{\#examples} \alpha_i y_i \vec{x}_i$$

where α_i is some number of times we get $\vec{x}_i$ wrong and change weights

This assumes $\vec{w}_{initial} = \vec{0}$ (all zero)

Dual Form of the Perceptron Learning Rule

output of perceptron $\equiv h(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x})$

$$\text{sgn}(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

So $h(\vec{x}) = \text{sgn}\left(\left[\sum_{i=1}^{\#examples} \alpha_i y_i \vec{x}_i\right] \cdot \vec{x}\right)$

$= \text{sgn}\left(\sum \alpha_i y_i [\vec{x}_i \cdot \vec{x}]\right)$

New (i.e., dual) perceptron algorithm:

For each example i

if $y_i * \left(\sum_{j=1}^{\#examples} \alpha_j y_j [\vec{x}_j \cdot \vec{x}_i]\right) \leq 0$ (i.e., predicted$_i \neq$ teacher$_i$)

then $\alpha_i = \alpha_i + 1$ (counts errors)

Primal versus Dual Space

- Primal – "weight space"
  - Weight features to make output decision

$$h(\vec{x}_{new}) = \text{sgn}(\vec{w} \cdot \vec{x}_{new})$$

- Dual – "training-examples space"
  - Weight distance (which is based on the features) to training examples

$$h(\vec{x}_{new}) = \text{sgn}\left(\sum_{j=1}^{\#examples} \alpha_j y_j [\vec{x}_j \cdot \vec{x}_i]\right)$$

The Dual SVM

Let n = *#training examples*

$$\min_{\vec{\alpha}} \left(\left(\tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j)\right) - \sum_{i=1}^{n} \alpha_i\right)$$

such that

$\sum_{i=1}^{n} y_i \alpha_i$

$\forall \alpha_i \geq 0$

Can convert back to primal form:

$$\vec{w} = \sum_{i=1}^{n} y_i \alpha_i \vec{x}_i$$

$$\gamma = -\frac{\max_{y_i=-1}\left(\langle \vec{w} \cdot \vec{x}_i \rangle\right) + \min_{y_i=1}\left(\langle \vec{w} \cdot \vec{x}_i \rangle\right)}{2}$$

Non-Zero α_i's

Those examples with $\alpha_i \neq 0$ are the support vectors

Recall

$\vec{w} = \sum_{i=1}^{n} y_i \alpha_i \vec{x}_i$

- only the support vectors (i.e., $\alpha_i \neq 0$) contribute to the weights

Generalizing the Dot Product

We can generalize

Dot_Product$(\vec{x}_i, \vec{x}_j) \equiv \vec{x}_i \cdot \vec{x}_j$

to other "kernel functions"

e.g., $K(\vec{x}_i, \vec{x}_j) \equiv (\vec{x}_i \cdot \vec{x}_j)^s$

LAn acceptable kernel (usually non-linear) maps the original features into a new space implicitly

- in this new space we're computing a dot product
- we don't need to explicitly know the features in the new space
- usually more efficient than directly converting to new space

The New Space for a Sample Kernel

Let $K(\vec{x},\vec{z}) = (\vec{x}\cdot\vec{z})^2$

and let *#features* = 2

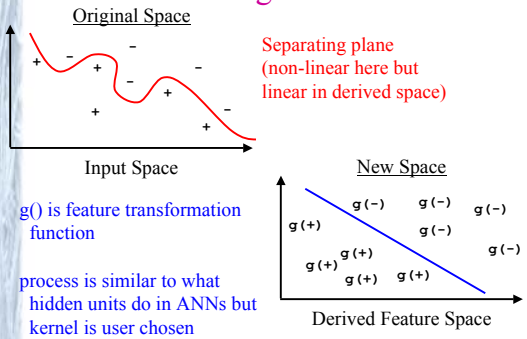
$$(\vec{x}\cdot\vec{z})^2 = (x_1z_1 + x_2z_2)^2$$

$$= x_1x_1z_1z_1 + x_1x_2z_1z_2 + x_2x_1z_2z_1 + x_2x_2z_2z_2$$

$$= \langle x_1x_1, x_1x_2, x_2x_1, x_2x_2\rangle \cdot \langle z_1z_1, z_1z_2, z_2z_1, z_2z_2\rangle$$

Our new feature space (with 4 dimensions)
- we're doing a dot product in it

Visualizing the Kernel



g() is feature transformation function

process is similar to what hidden units do in ANNs but kernel is user chosen

More Sample Kernels

1) $K(\vec{x},\vec{z}) = \left((\vec{x}\cdot\vec{z}) + const\right)^d$

2) $K(\vec{x},\vec{z}) = e^{-\|\vec{x}-\vec{z}\|^2/\sigma^2}$

- Gaussian kernel, leads to RBF network

3) $K(\vec{x},\vec{z}) = \tanh\left(c*(\vec{x}\cdot\vec{z}) + d\right)$

- Related to sigmoid of ANN's

- plus many more, including many designed for specific tasks (text, DNA, etc.)

What Makes a Kernel

Mercer's theoremn characterizes when a function $f(\vec{x},\vec{z})$ is a kernel

If $K_1()$ and $K_2()$ are kernels, then so are

1) $K_1() + K_2()$
2) $c*K_1()$ where c is constant
3) $K_1() * K_2()$
4) $f(\vec{x}) * f(\vec{z})$ where $f()$ returns a real

...

Key SVM Ideas

- Maximize the margin between positive and negative examples (connects to PAC theory)
- Penalize errors in non-separable case
- Only the support vectors contribute to the solution
- Kernels map examples into a new, usually non-linear space
  - We implicitly do dot products in this new space (in the "dual" form of the SVM program)