# TOP CROSS SECTION
# WITH EVENT CLASSIFIER BASED ON SUPPORT VECTOR MACHINES


Ben Whitehouse*, Krzysztof Sliwa
Tufts University
Department of Physics and Astronomy
Medford, Massachusetts 02155, USA

ATLAS Top Mass Meeting, CERN, 22 October, 2010

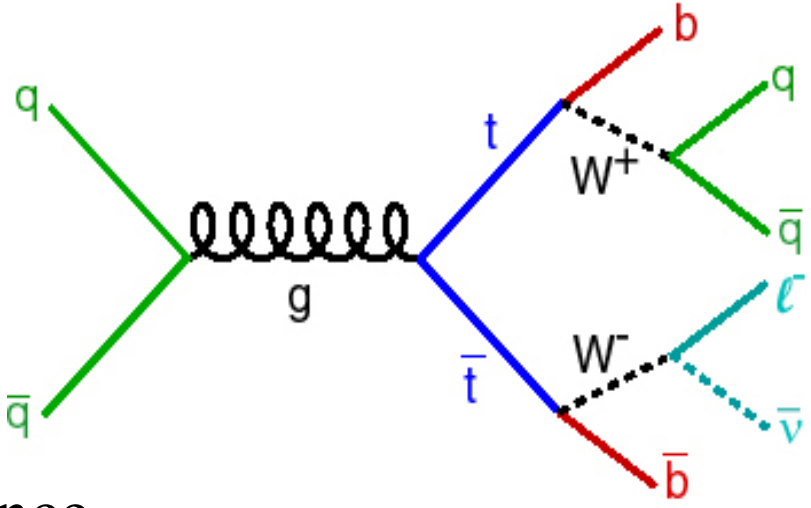
*Ben has just defended his Ph.D. and is looking for a job

# Task

Identify $t\bar{t}$ decays in the Lepton + Jets Channel

$$N = \sigma \mathcal{L}$$

$$\sigma_{t\bar{t}} = \frac{N_{t\bar{t}}}{\mathcal{L}} = \frac{1}{\mathcal{L}}\left(\frac{\theta_{t\bar{t}} N}{\varepsilon}\right)$$




$\varepsilon$ adjusts for selection cut acceptance

$\theta_{t\bar{t}}$ is the fractional amount of $t\bar{t}$ in the sample
and is what we will determine with the SVM

# Main Idea

Measure ttbar cross section in the L+J channel

Use a MultiClass SVM procedure to tell Signal from Backgrounds
3 SVMs are used together as a system

Include btag info in pretag L+J sample to improve discrimination

Results should at the same provide information about
heavy flavor content of BG (W+bb vs W+lf, etc)

# Support Vector Machines

Machine Learning via Hyper-plane Separation

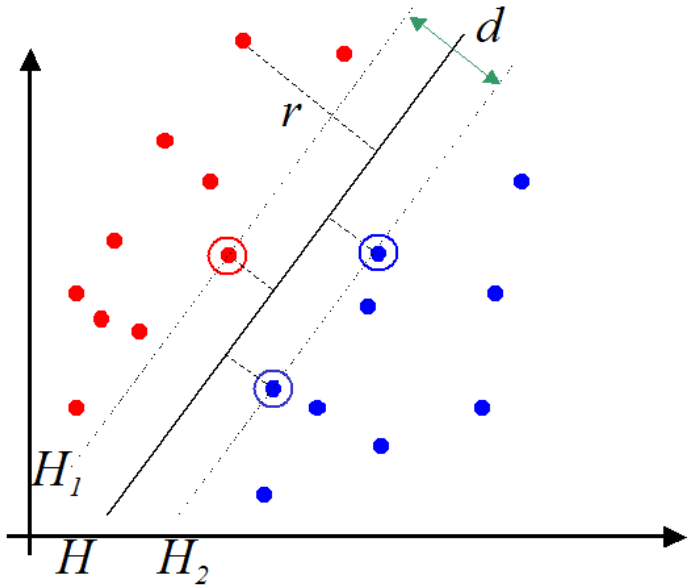
Take N training points
$\{(x_1, y_1),(x_2, y_2),\ldots,(x_N, y_N)\}$

$x_i$ = vector that describes "features"
$y_i$ = {-1,+1} describes class

We want to create a classifying function with the form:

$$f(\vec{x}) = \vec{w} \cdot \vec{x} - b$$

that maximizes the margin d between classes



# Support Vector Machines

Support Vector Machine (SVM) is a learning algorithm developed a part of Vapnik-Chervonenkis theory in 1989. The soft-margin SVM variation used currently was proposed by Vapnik and Cortes in 1995.

A data point can be viewed as a N-dimensional vector (a list of N numbers), and we can separate such points with a $N-1$-dimensional hyperplane we have defined a linear classifier.

A support vector machine is a binary, linear, classifier which decides which of the two possible classes of events the input event belongs to. Given two sets of learning samples, SVM constructs a hyperplane in a high or infinite dimensional space which can be used for classification of the input events into two possible classes.

# Support Vector Machines

N=1 case

in a linear case a separating hyperplane can be found, it is a point *
(1-1=0-D "hyperplane")

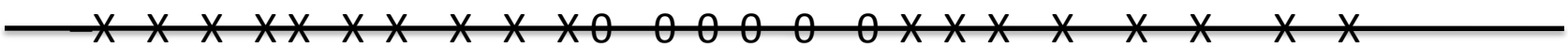
# Support Vector Machines

If the two sets to be discriminated are not linearly separable in the original space, the original finite dimensional space is mapped into a much higher dimensional (or even infinite dimensional) space in which the problem is linearly separable

# Support Vector Machines

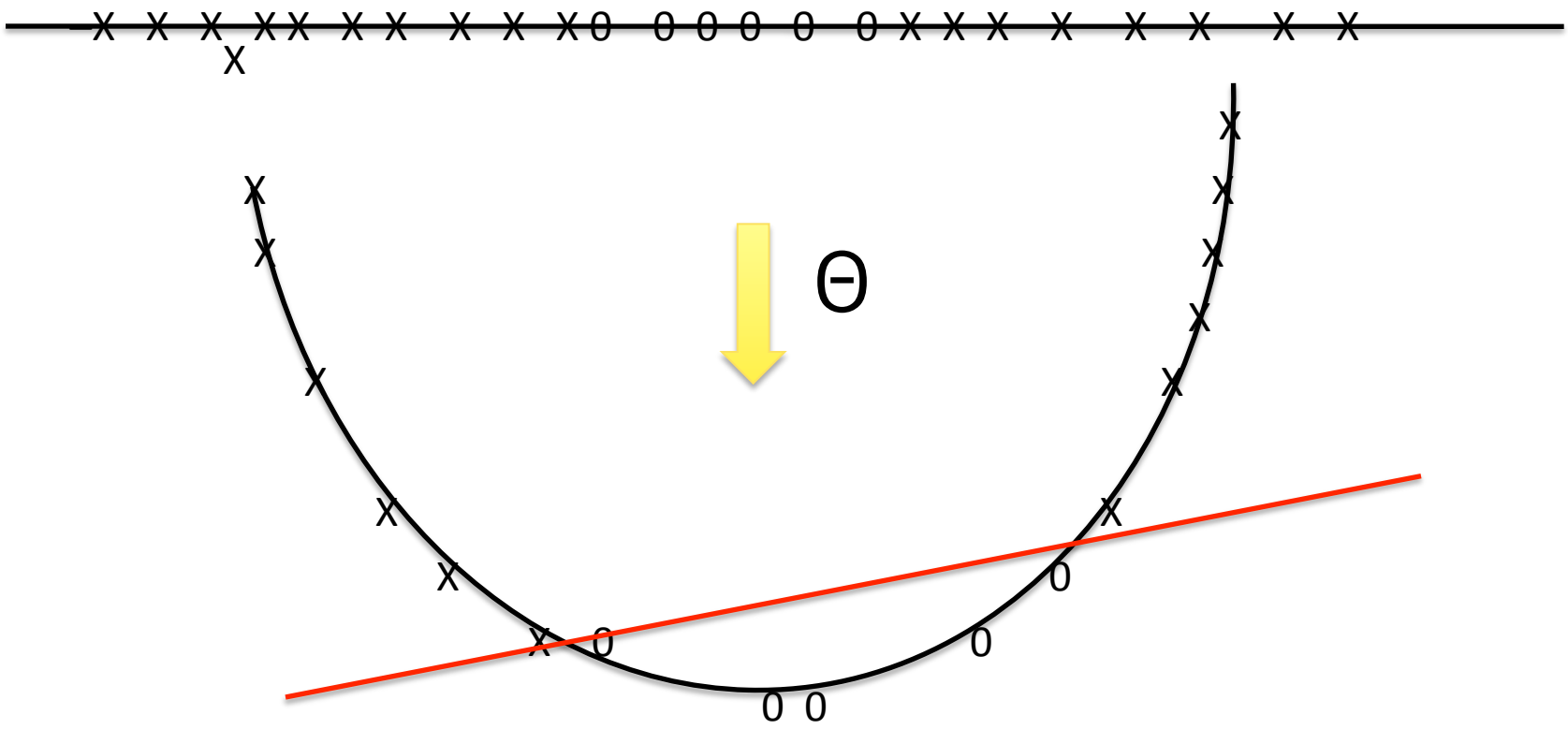
N=1 case

in a non-linear case a separating hyperplane cannot be found



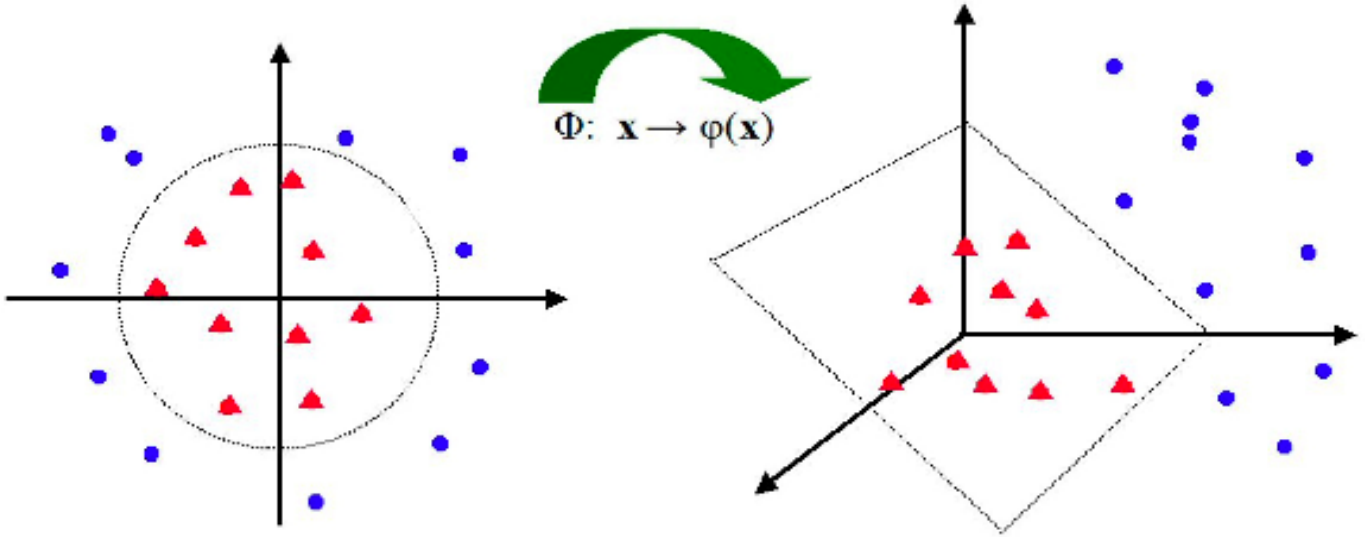
# Support Vector Machines

One can map Θ from 1-D space to 2-D space and then a separating hyperplane can be found, a line in this case (2-1=1-D "hyperplane")



# Support Vector Machines

Here, one can map $\Phi$ from 2-D space to 3-D space and then a separating hyperplane can be found, in this case is a plane (3-1=2-D "hyperplane")



# Support Vector Machines

Machine Learning via Hyper-plane Separation

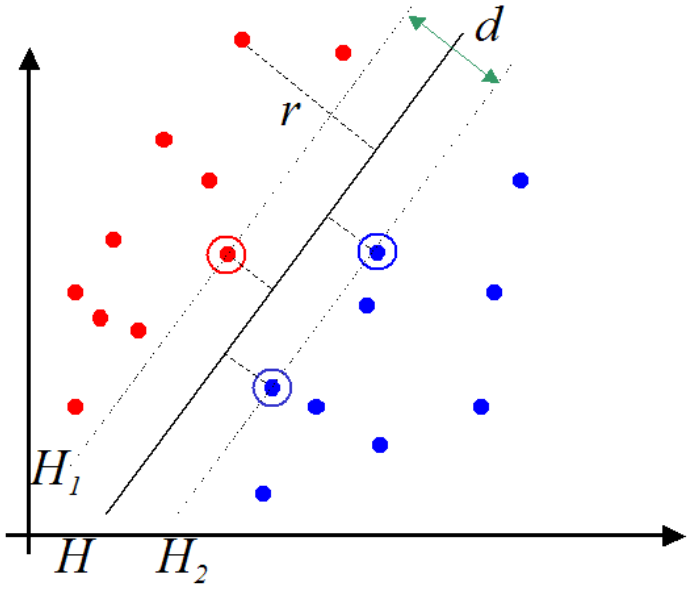
Take N training points
$\{(x_1, y_1),(x_2, y_2),\ldots,(x_N, y_N)\}$

$x_i$ = vector that describes "features"
$y_i$ = {-1,+1} describes class

We want to create a classifying function with the form:

$$f(\vec{x}) = \vec{w} \cdot \vec{x} - b$$

that maximizes the margin d between the two classes of points



# Support Vector Machines (dual representation)

Machine Learning via Hyper-plane Separation

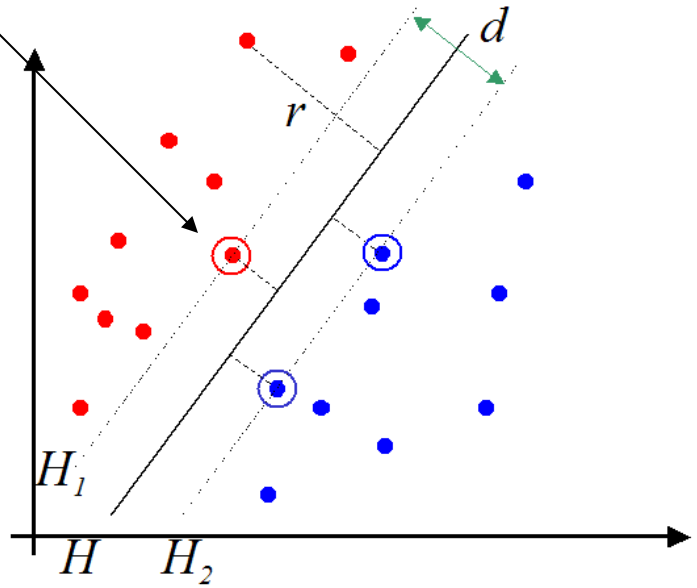
$$f(\vec{x}) = \vec{w} \cdot \vec{x} - b$$

Only vectors at the margin get non-zero alpha

These are the "support vectors" they hold up the hyper-plane

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

Solution is a linear combination of training points



# Support Vector Machines (dual representation)

One can re-write the decision function as:

$$f(\vec{x}) = \vec{w} \bullet \vec{x} - b = \sum \alpha_i y_i (\vec{x}_i \bullet \vec{x}) - b$$

In the dual representation data appears ONLY through inner products (scalar products) both in training and in the classifying function

# Support Vector Machines

To define a separating hyperplane one needs to be able to calculate the inner (scalar) products. It turns out that one DOES NOT need to know the details of the mapping from the original space (in which the problem is nonlinear) into the higher dimensional space (in which the problem becomes linear).

The "Kernel trick" (based on Mercer's theorem 1909) allows to express the needed inner products in the higher dimensional space through the real valued kernel function which is evaluated in the original (lower dimensional) space.

$$K(\vec{x}_1, \vec{x}_2) = \Theta(\vec{x}_1) \bullet \Theta(\vec{x}_2)$$

# Support Vector Machines

Machine Learning via Hyper-plane Separation

$$f(\vec{x}) = \vec{w} \cdot \vec{x} - b$$

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

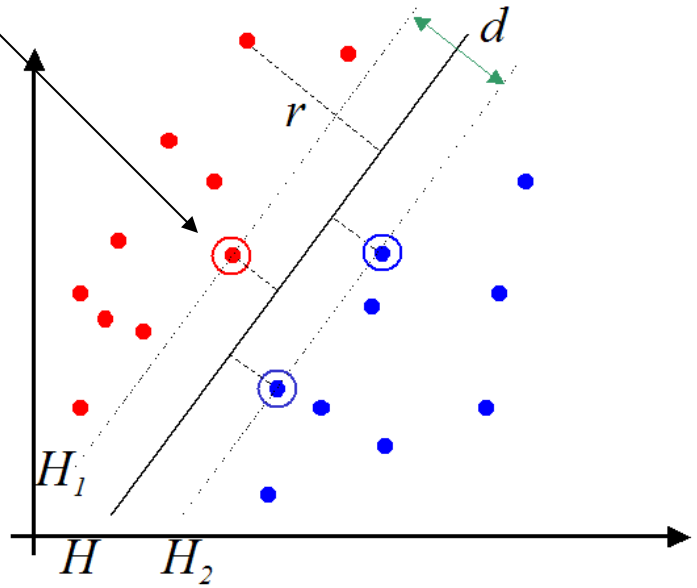
Only vectors at the margin get non-zero alpha

These are the "support vectors" they hold up the hyper-plane

We used a Gaussian kernel function.

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

$$f(\vec{z}) = \sum_{i=1}^{N} \alpha_i y_i e^{\frac{-|\vec{x}_i - \vec{z}|^2}{2\sigma^2}} - b$$



# Method

Use data through P13 (2.2 fb-1)

3 Classes (equals 3 SVMs used in conjunction):

- Signal (ttbar)
- Light BG (W+lf, W+c, W+cc)
- Heavy BG (W+bb)

Gradient to investigate what features matter

Updated Systematics

- update ISR/FSR
- update Herwig
- W+Jets $Q^2$
- b-tag SF

QCD Templates

# 20 Features

Kinematic
b-tagging

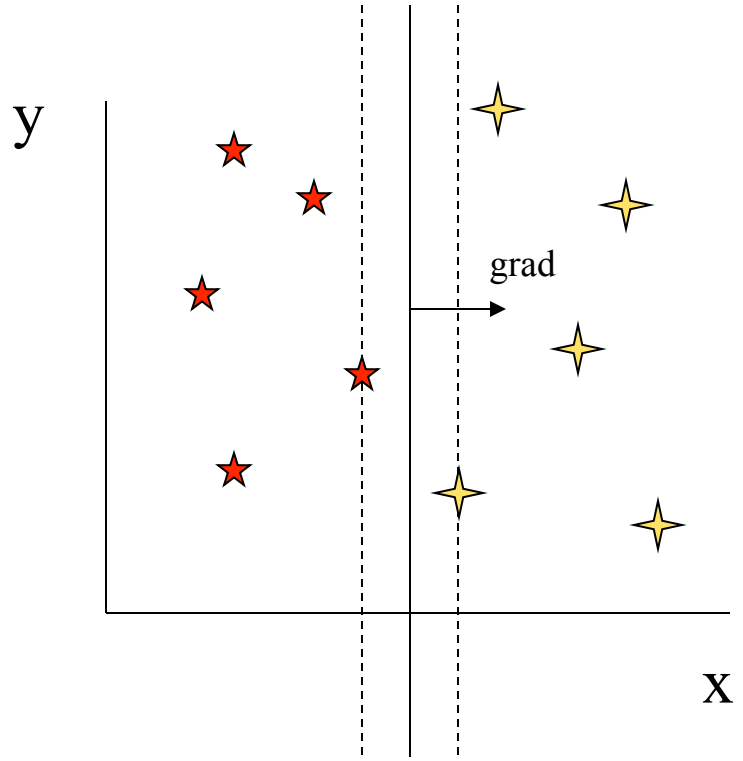
Lepton E, Px, Py, Pz
Missing Et and Phi
Fox-Wolfram Moments (first 5)
Sum Jets $E_t$, E, Px, Py, Pz
HT
Two Eigenvalues from Normalized Momentum Tensor

Number Positive SecVtx tags

# Gradient for Feature Ranking

An evaluation of the Gradient can be used to get a feel for what features of the SVM are important in discrimination.

Here, a grad of (1,0) shows x is important, while y is not so much.



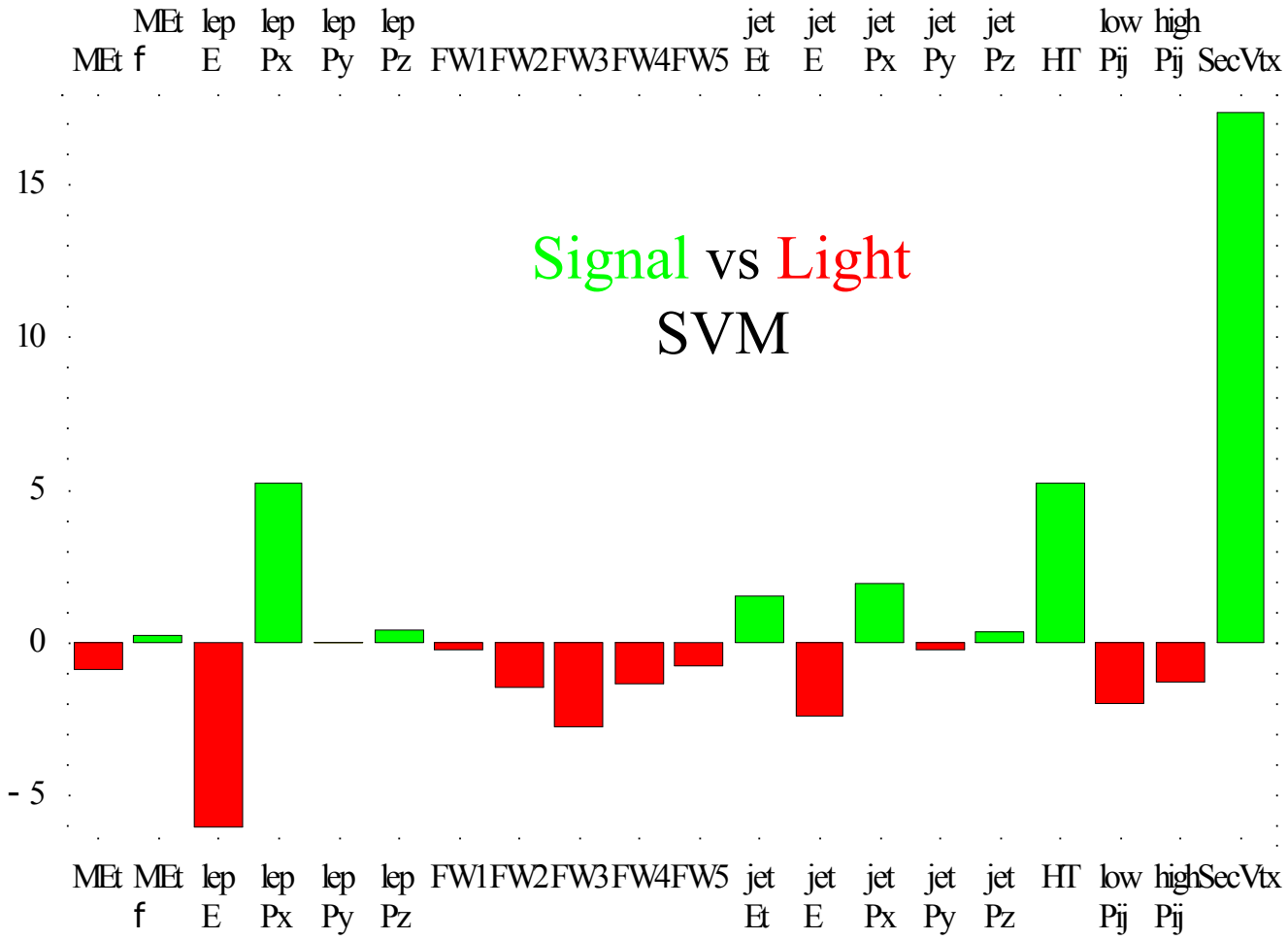
# Gradient for Feature Ranking

$$f(\vec{z}) = \sum_{i=1}^{N} \alpha_i y_i e^{\frac{-|\vec{x_i} - \vec{z}|^2}{2\sigma^2}} - b$$

The gradient of the SVM hyper-plane is given by

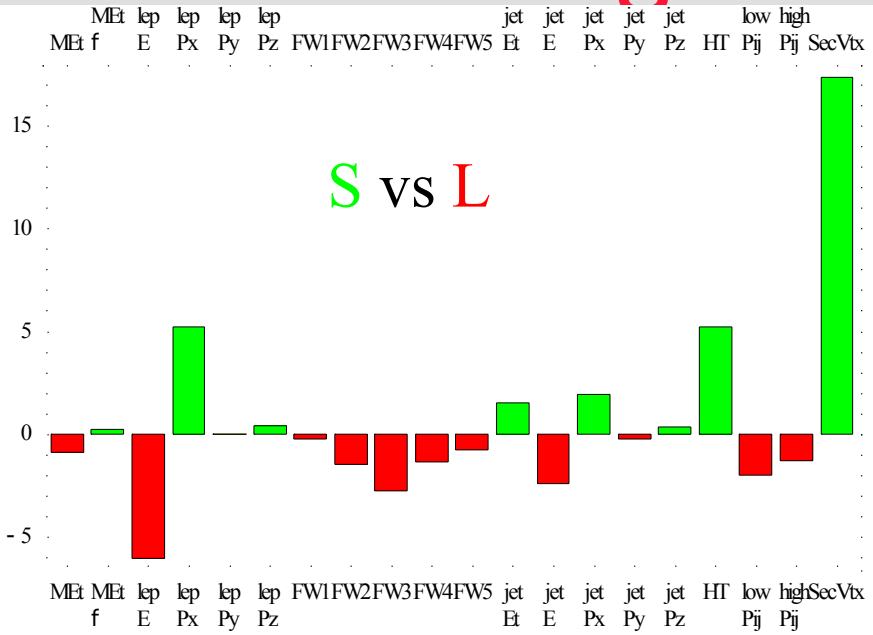
$$(\vec{\nabla} f)_j = \sum_{i=1}^{N} \frac{\alpha_i y_i}{\sigma^2} (\vec{x_{ij}} - z_j) e^{\frac{-|\vec{x_i} - \vec{z}|^2}{2\sigma^2}}$$

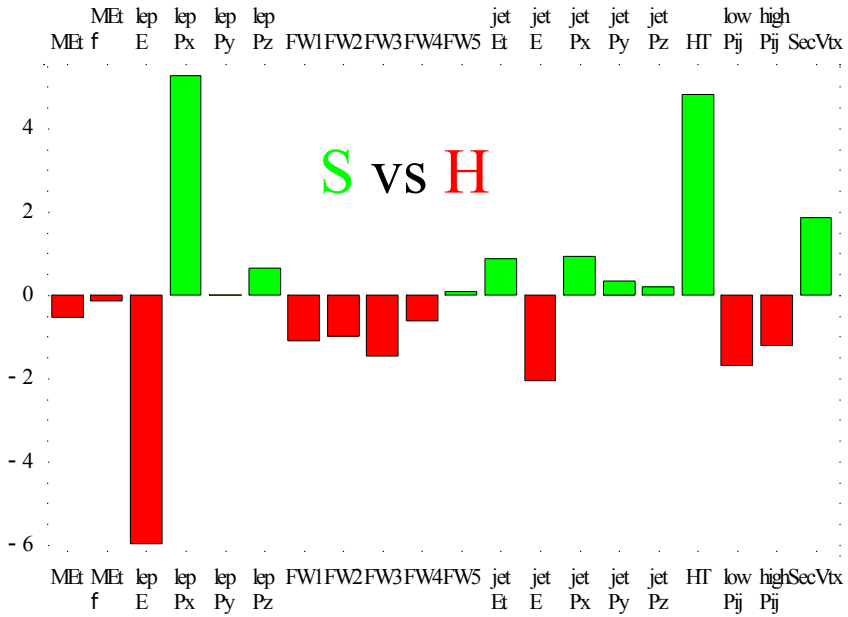
# Gradient for Feature Ranking

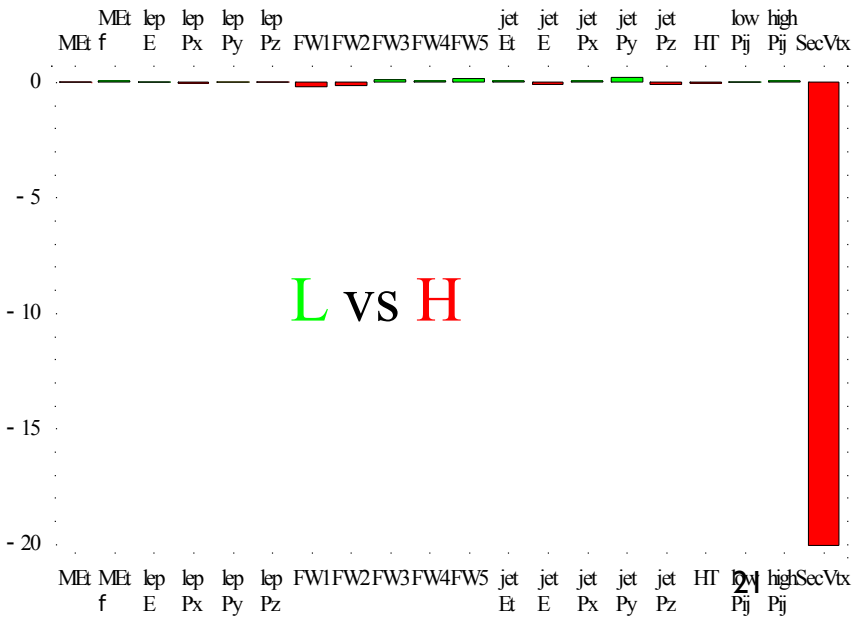


# Gradient for Feature Ranking

This is an average of the gradient, sampled in the ttbar region of the feature space.

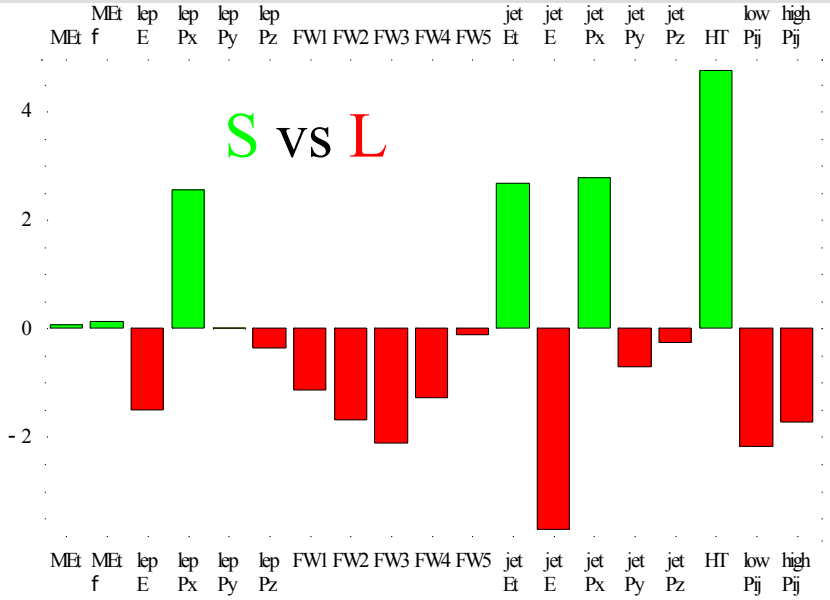


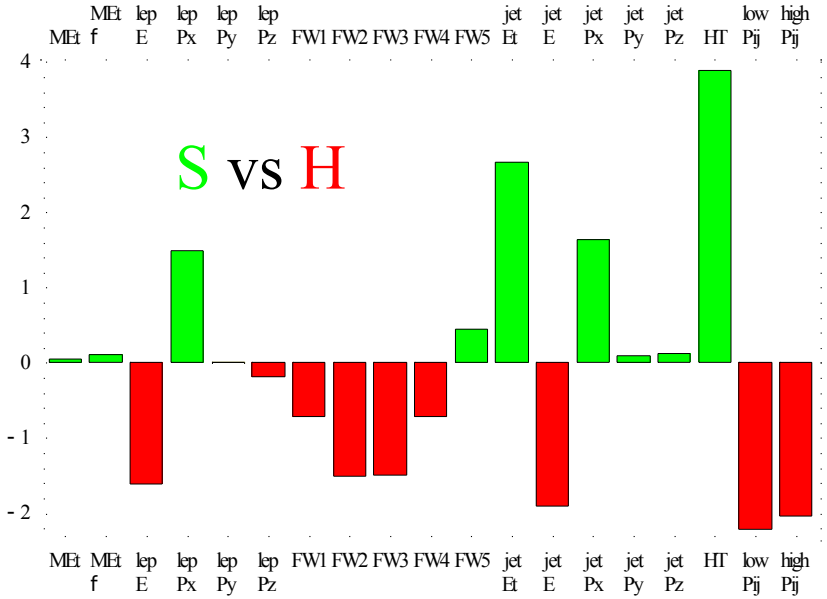



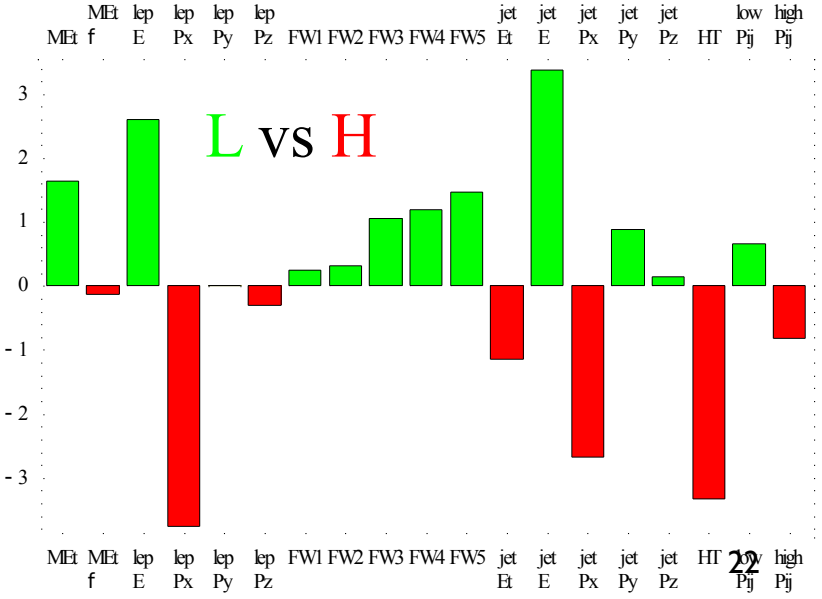



# Gradient for Feature Ranking

This is the same done on the Kinematic only SVMs



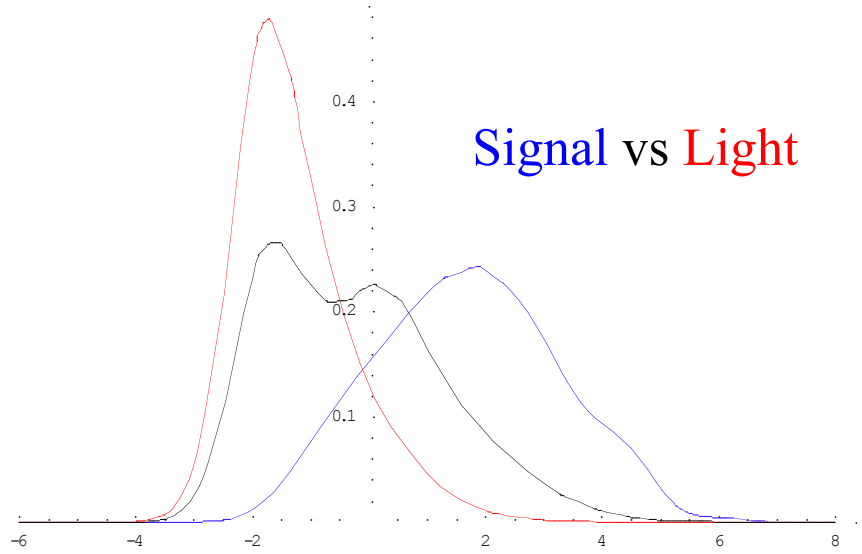



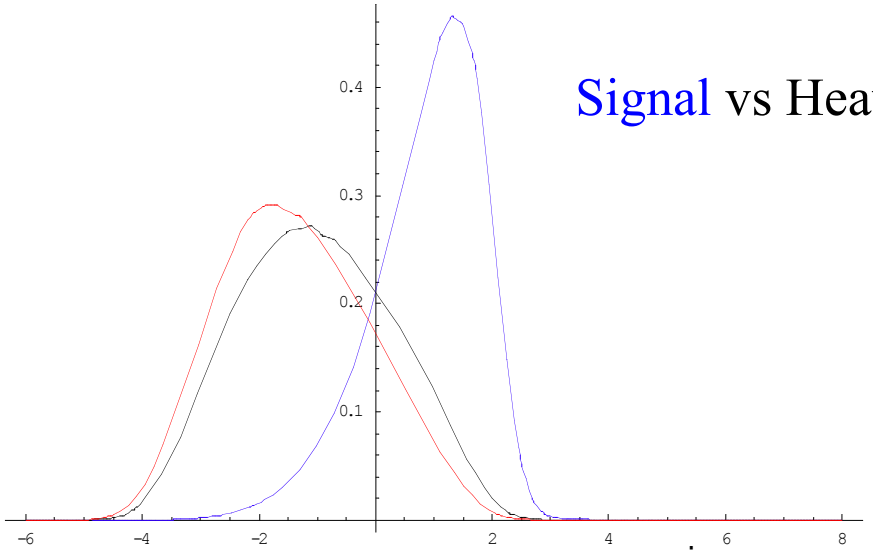
# MultiClass Response

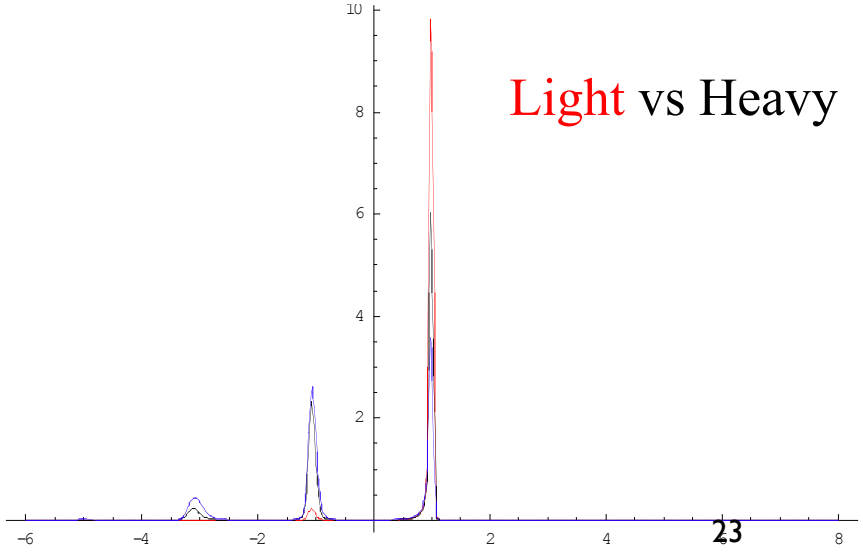
Signal
Light
Heavy

Shows overall output from the 3 SVMs

Light and Heavy are composed of sub-processes added by cross section




g, CE

# HOW TO SEPARATE MORE THAN 2 CLASSES?

Signal
Light
Heavy

- USE PAIR-WISE COUPLING (needs K=(N-1)N/2 (3 SVM in our case)

- FEATURE SPACE

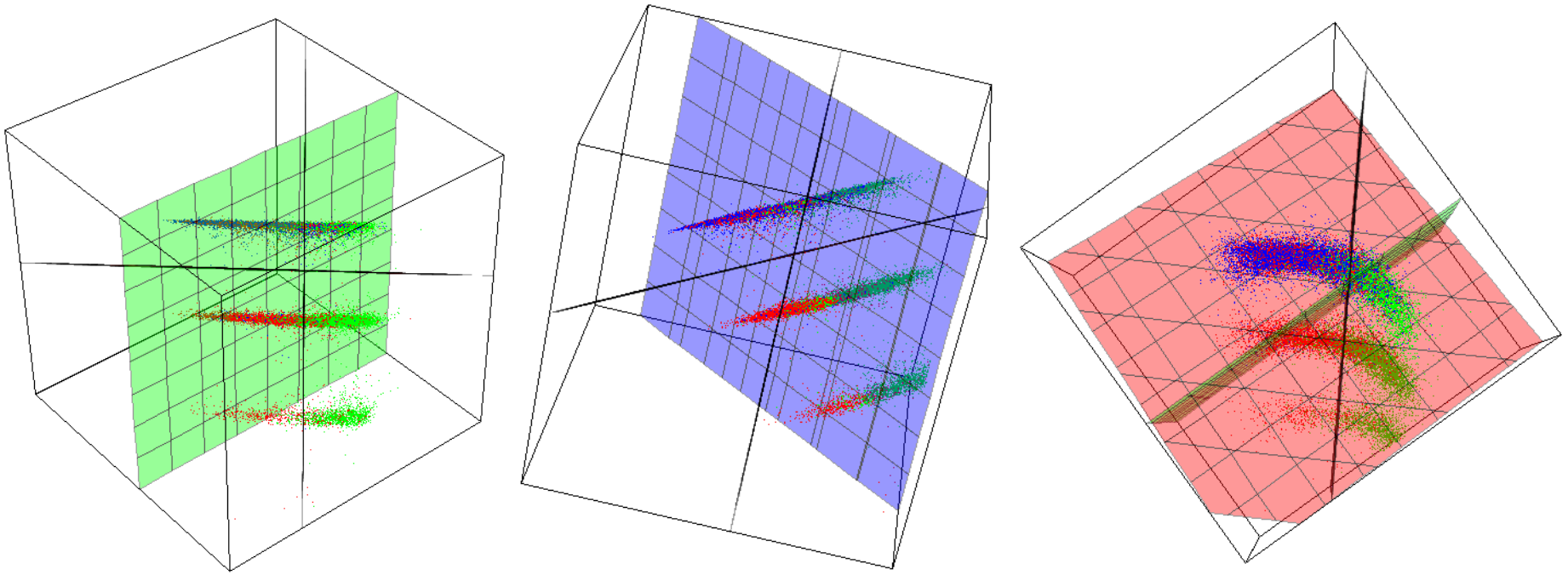
A direct way of determining the separation ability of the SVM system is to project data points in the problem's feature space, and then examine how the different data classes are distributed in this space. After finding an orthonormal basis in the feature space, one can map the classifiying (decision) functions for all thee SVM in the feature space. One can easily see how well the separation between the 3 classes of events is.

# MultiClass PDFs

Signal
Light
Heavy

3D Histograms used to create templates

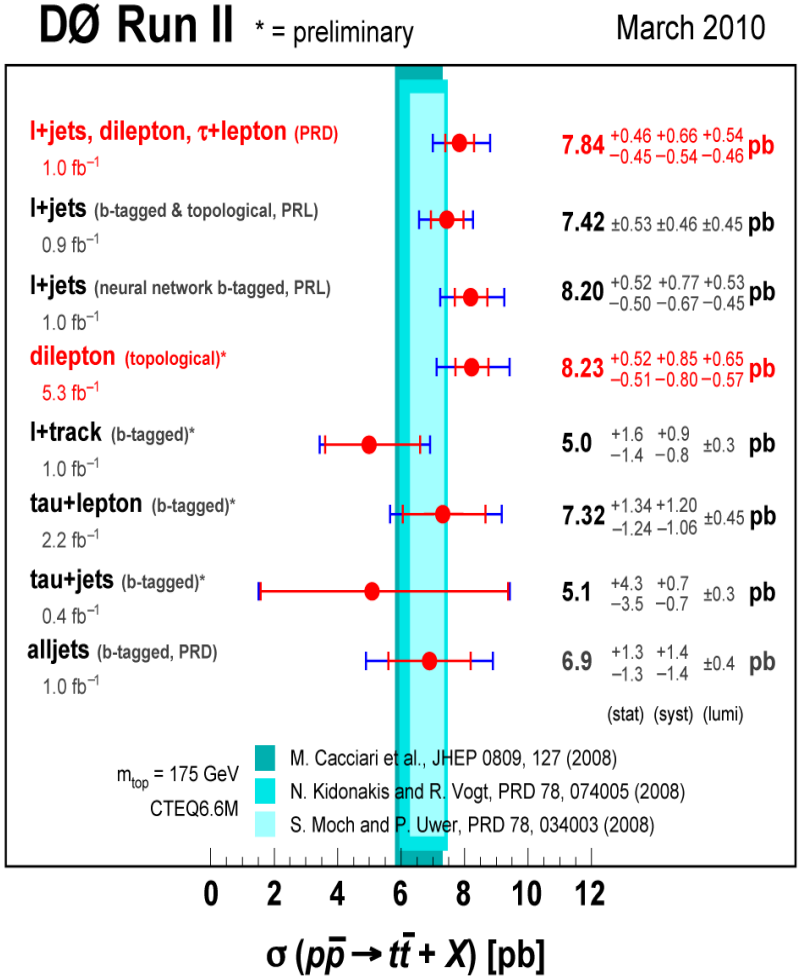
http://tuhept.phy.tufts.edu/~ben/talk1.html

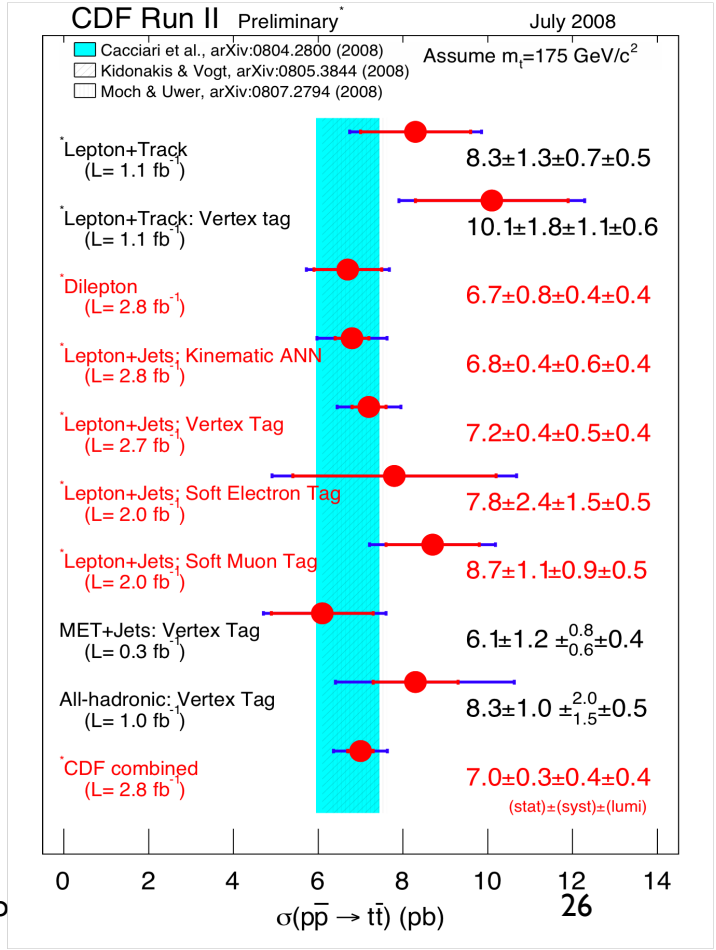


# Results: excellent classifier

$$\sigma_{t\bar{t}} = 7.14 \pm 0.25(\text{stat})^{+0.41}_{-0.76}(\text{sys})^{+0.45}_{-0.40}(\text{lumi})\ \text{pb}$$

$$m_t = 175\ \text{GeV/c}^2$$



# plans for svm analyses in ATLAS

The entire machinery can be applied in a straightforward fashion to the lepton+jets in ATLAS, the only question is the availability of MC background samples which describe the real data well, this can wait a bit for tuning and validation

The same method could be also applied to the di-lepton final state, of course a set of features would have to modified

Chapter 2 on SVM from Ben's Thesis as attached in the end, the entire thesis is available at:

http://tuhept.phy.tufts.edu/~sliwa/cdf10280_svmThesis.pdf

# Chapter 2

# The Support Vector Machine

The overall idea for the Support Vector Machine was put forth by Vladimir Vapnik[20] in the 1990s. Using labeled training data as a basis, the SVM methodology attempts to calculate the optimal separating hyperplane between the two classes of data under consideration. In order to tell apart apples from oranges, one might consider any number of descriptive, quantifiable features of these two objects. Color, texture, size, lifetime, ripening time, seed content, skin width, shape, density, mass, *etc.* can all be assembled in a vector of values used to describe an apple or an orange. Given a labeled handful of these vectors, the SVM formalism computes a separating hyperplane that attempts to place all the apples on one side and all oranges on the other. The purpose of this chapter is to set forth the theory behind Support Vector Machines, point out some of their interesting properties, and show how they can be applied to larger classification problems.

## 2.1 The Basics

Suppose that we define a vector of real numbers, where each dimension represents some characteristic of the subject we wish to study. The $N$ dimensions of this vector are known as *features*. Let us define

$$\vec{x} \equiv \{x_1, x_2, \ldots, x_N\} \tag{2.1}$$

$$y \equiv \pm 1$$

We also need a way to distinguish between the two classes of data. We will use the convention that every data point is specified by its vector $\vec{x}$, and its class $y$ when this information is available.

Imagine we are given some number of events whose class is known. Some of the points belong to the $y = +1$ class, and others to the $y = -1$ class. Suppose we plot these vectors in real $N$ dimensional space in the normal way. Our task is to then use these event vectors to find a separating hyper-plane between the points with $y = +1$ and $y = -1$. For vectors with only 2 features, consider Figure 2.1. As can be seen, it is necessary to establish a way to uniquely define the plane of separation. The method that is chosen is to find the plane that separates the two classes of points with the widest *margin*. Figure 2.2 defines the geometry of the situation in a two feature problem.

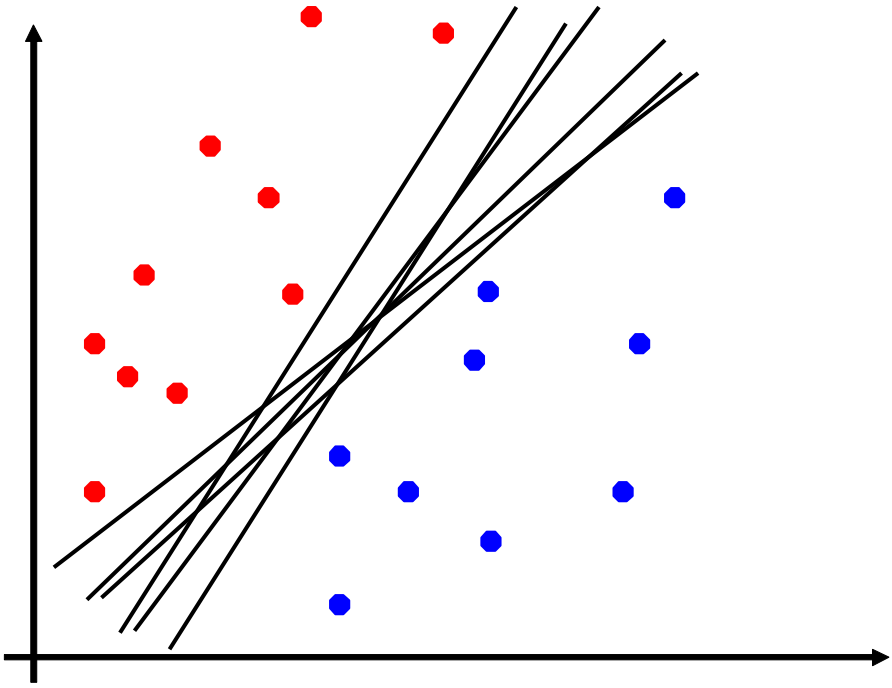


Figure 2.1: Many planes could potentially separate our data.

For an $N$ dimensional space, the equation of a hyper-plane in that space is given by:

$$f(\vec{x}) = \vec{w} \cdot \vec{x} - b \tag{2.2}$$

where $\vec{w}$ is a vector normal to the plane, and $b$ is a real number that offsets the plane's

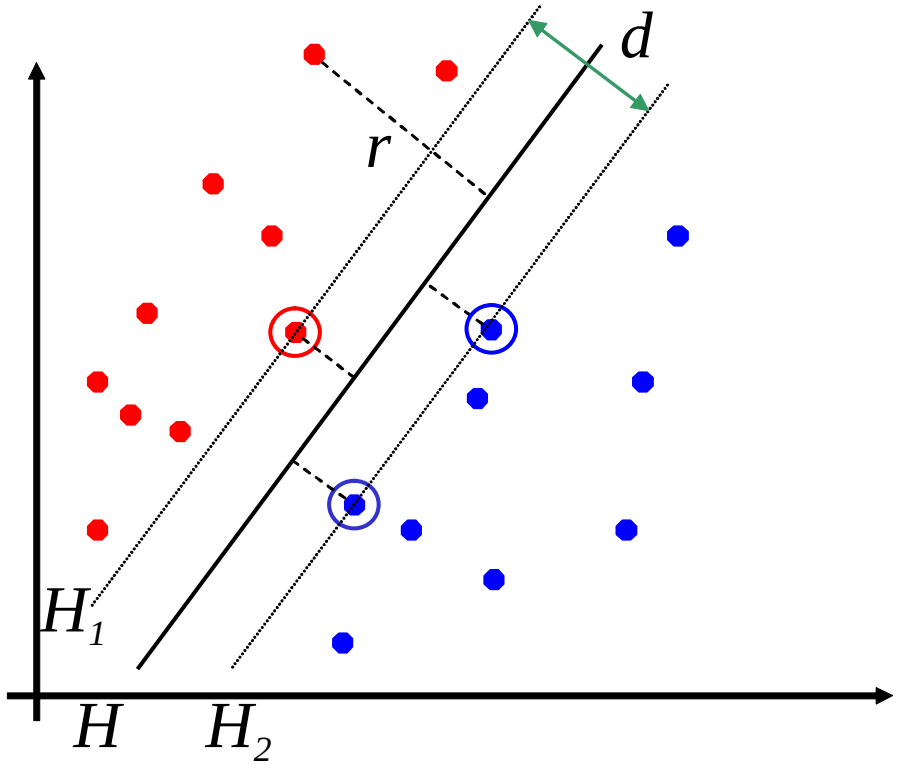



Figure 2.2: 2D SVM Geometry

position from the origin. Once $\vec{w}$ and $b$ have been determined, this expression becomes the *learned function*. After computing $f(\vec{a})$ on some vector of unknown class, the sign of $f(\vec{a})$ can be our guess as to which class $\vec{a}$ belongs.

Looking more closely at the learned function, we see that the perpendicular distance $r$ from the plane to a point $\vec{z}$ is given by

$$r = \frac{\vec{w} \cdot \vec{z} - b}{|\vec{w}|} \tag{2.3}$$

A further step needs to be taken to set the overall scale for the problem. This is because there are an infinite number of vectors $\vec{w}$ that give the same orientation for the plane. We could simply constrain $\vec{w}$ to be a unit vector. However, a more convenient method for our future calculations is simply to compute $\vec{w}$ such that the value of the learned function is $\pm 1$ at the margin. This fixes the scale, and the overall width of the margin is then

$$d = \frac{2}{|\vec{w}|} \tag{2.4}$$

This quantity is clearly maximized if we minimize the length of $\vec{w}$. To be clear, we have now totally specified our geometry. $\vec{w}$'s direction gives the orientation of the separating hyperplane, $b$ displaces it from the origin to its proper absolute position, and $\vec{w}$'s magnitude sets the width of the margin (*i.e.* the density of the contours parallel to the hyper-plane).

In order to ensure that no data points invade the margin, we also require the following linear constraints on each of the $N$ training points under consideration.

$$y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1 \tag{2.5}$$

Putting everything together, we can assemble a Lagrangian to minimize for this problem.

$$\mathscr{L} = \frac{1}{2}\vec{w} \cdot \vec{w} - \sum_{i=1}^{N} \alpha_i y_i(\vec{w} \cdot \vec{x_i} - b) + \sum_{i=1}^{N} \alpha_i \tag{2.6}$$

The first term maximizes the margin, and the subsequent $N$ terms ensure the separation of the two classes in our training set. Notice the use of Lagrange multipliers ($\alpha_i' s \geq 0$) to enforce these linear constraints. This approach is common in solving quadratic optimization problems such as we have here. A dual Lagrangian can be constructed by taking the partial derivatives with respect to the primal variables $\vec{w}$ and $b$ and noting they are zero at optimality. This leads to the following relations

$$\frac{\partial \mathscr{L}}{\partial \vec{w}} = 0 \Rightarrow \vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x_i} \tag{2.7}$$

$$\frac{\partial \mathscr{L}}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.8}$$

Substituting these back into our original Lagrangian, we get the dual Lagrangian

$$\mathscr{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \tag{2.9}$$

This formulates our problem entirely in terms of the $\alpha$'s. Once the $\alpha$'s are known, we can recover $\vec{w}$ through the relation (2.7) above, and $b$ from the fact that

$$b = \vec{w} \cdot \vec{x_k} - y_k \tag{2.10}$$

for any training vector $k$, as long as its corresponding $\alpha_k \neq 0$. Using our new formulation we can re-express the learned function in terms of the $\alpha$'s.

$$f(\vec{z}) = \sum_{i=1}^{N} \alpha_i y_i \vec{x_i} \cdot \vec{z} - b \tag{2.11}$$

The important thing to note here is that our learned function depends only on those training points $\vec{x_i}$ which have an $\alpha_i \neq 0$. Training points that meet this requirement are known as the *Support Vectors*, as they are the vectors that hold up the decision plane. Only support vectors, along with their associated Lagrange multipliers, are needed to reconstruct the decision plane. All other training points are ignorable. Also, notice that the learned function only depends on the inner product between training points and the test point $\vec{z}$. We will return to this later when considering non-linear decision surfaces.

## 2.2 Mechanical Analogy

An interesting interpretation of the formulation of the Support Vector Machine reveals it to have an exact physical analog. The support vectors can be viewed as exerting a positive or negative force on the decision boundary, and the solution in which we find the best separating hyper-plane is exactly that which provides mechanical equilibrium. In other words, all we are requiring is that the sum of all the forces and torques net zero at optimality. This view can be helpful in getting a handle on what actually is happening when a SVM is optimized. It can be useful to think about how certain points "push around" the decision boundary.

To illustrate this point, let's associate a force $\vec{F}_i$ with each support vector. We will take

$$\vec{F}_i = -y_i \alpha_i \hat{w} \tag{2.12}$$

Here, $\hat{w}$ is simply a unit vector in the direction of $\vec{w}$, defined as $\hat{w} = \frac{\vec{w}}{|\vec{w}|}$. Recall from Equation (2.2) that $\hat{w}$ gives the orientation of the hyper-plane. In fact, $\hat{w}$ is normal to the hyper-plane's surface. Therefore, Equation (2.12) associates a force normal to the hyper-plane which points inwards toward the zero contour of the decision surface. The

magnitude of the force is proportional to the size of that vector's Lagrange multiplier $\alpha_i$. The requirements for mechanical equilibrium are as follows:

$$\sum_{i}^{SVs} \vec{F}_i = 0 \tag{2.13}$$

$$\sum_{i}^{SVs} \vec{x}_i \times \vec{F}_i = 0 \tag{2.14}$$

Using Equations (2.7) and (2.8), we see that these conditions hold.

$$\sum_{i}^{SVs} \vec{F}_i = \sum_{i}^{SVs} -y_i \alpha_i \hat{w} = -\hat{w} \sum_{i}^{SVs} y_i \alpha_i = 0 \tag{2.15}$$

$$\sum_{i}^{SVs} \vec{x}_i \times \vec{F}_i = \sum_{i}^{SVs} \vec{x}_i \times (-y_i \alpha_i)\hat{w} = \sum_{i}^{SVs} (-y_i \alpha_i)\vec{x}_i \times \hat{w} = -\vec{w} \times \hat{w} = 0 \tag{2.16}$$

## 2.3 Soft Margin

Sometimes, it happens that a problem is not perfectly separable. Imagine there is some noise in the problem, or that some points in the training sample are mislabeled. To accommodate this kind of situation, our problem can be reformulated to allow some points to invade the margin. However, we will penalize these points in order to discourage their occurrence. To relax the margin constraints, we modify Equation (2.5) to read

$$\begin{aligned} y_i(\vec{w} \cdot \vec{x_i} - b) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \end{aligned} \tag{2.17}$$

These free parameters $\xi_i$ allow individual points to enter the margin. However, if we don't suppress this behavior in some manner, then the trivial solution where all the points fall into the margin will result. So, we modify the Lagrangian as follows:

$$\mathscr{L} = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i y_i (\vec{w} \cdot \vec{x_i} - b) + \sum_{i=1}^{N} \alpha_i (1 - \xi_i) \tag{2.18}$$

The second term in (2.18) involving $C$ is the penalty term. Since we are minimizing $\mathscr{L}$, and both $C$ and the $\xi_i$'s are taken to be non-negative, this term will tend to cap the error. It is instructive to view the Lagrangian rearranged like this:

$$\mathscr{L} = \frac{1}{2}\vec{w} \cdot \vec{w} - \sum_{i=1}^{N} \alpha_i y_i (\vec{w} \cdot \vec{x_i} - b) + \sum_{i=1}^{N} \alpha_i + \sum_{i=1}^{N} \xi_i (C - \alpha_i) \tag{2.19}$$

Equation (2.19) has exactly the same form as (2.6), except for the last term that isolates $\xi$. In order to constrain $\mathscr{L}$, we note what happens to the objective function when $\alpha$'s are allowed to grow larger then $C$. In that case, the sum in the last term can be driven to $-\infty$ by letting the $\xi$'s become arbitrarily large. To avoid this situation, we require that the $\alpha$'s are bound within the range $0 \leq \alpha_i \leq C$. This result follows naturally by taking

$$\frac{\partial \mathscr{L}}{\partial \xi_i} = 0 \Rightarrow \alpha_i = C \tag{2.20}$$

for any $\xi_i$ that exists (that is $\xi_i > 0$). A very convenient by-product of applying this constraint is that it eliminates the last term in (2.19)! We are left with exactly Equation (2.6), and the rest of the solution follows as before. Equations (2.7) and (2.8) are the same, as is the dual $\mathscr{L}_D$ in (2.9). We are solving exactly the same problem as previously, except we have restricted the range of the Lagrange multipliers. We have a new free parameter $C$ that controls the rigidity of the margin. To summarize, the problem we are solving is

$$\begin{aligned} \mathscr{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \\ 0 \leq \alpha_i \leq C \\ \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{2.21}$$

Note that there is also a formulation that fixes the overall invasion "length" by placing an upper bound on the sum of the $\xi$ variables. This is in contrast to the method just outlined which limits the individual penetration of the given training points, but places no bound on the number of points which invade the margin.

## 2.4 Non-linear Solutions

Up to this point, we have shown how to find the best separating hyper-plane for a given set of training data. However, the hyper-plane we have constructed exists in the same space as the data points, or what is known as the *feature space*. One drawback to this situation is that it may not be possible to find a separating plane for the problem we would like to solve. It would be nice if we were able to carry our support vector formalism over to such problems that only have non-linear solutions. This can be accomplished if we map our original space into some other feature space of higher dimensionality. The hyper-plane can then be constructed in this higher dimensional space where the problem is linearly separable.

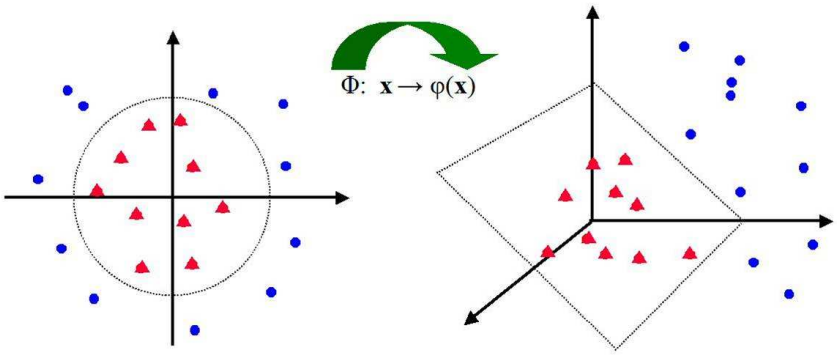



Figure 2.3: Mapping problem into higher dimensional space

Notice that in Equations (2.21) and (2.11) the training vectors $\vec{x_i}$ only enter the calculations when being dotted into another $\vec{x_j}$. If we were to actually do a mapping on the original vectors $\vec{x_i} \Rightarrow \varphi(\vec{x_i})$, the only expressions that show up in the training procedure are of the form $\varphi(\vec{x_i}) \cdot \varphi(\vec{x_j})$. Therefore, as long as we can carry out the dot products in the higher dimensional space, *we need not know the exact details of the mapping function* $\varphi$.

It turns out that there is a whole class of functions, known as *kernel functions*, which are dot products in some multi-dimensional space. In fact, any positive semi-definite function that can be shown to have the form given in (2.22) is a kernel function. Equations (2.23) through (2.26) are some example kernel functions:

$$K(\vec{x_i}, \vec{x_j}) \equiv \varphi(\vec{x_i}) \cdot \varphi(\vec{x_j}) \tag{2.22}$$

$$K(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j} \tag{2.23}$$

$$K(\vec{x_i}, \vec{x_j}) = (1 + \vec{x_i} \cdot \vec{x_j})^p \tag{2.24}$$

$$K(\vec{x_i}, \vec{x_j}) = \tanh(\beta_0 \vec{x_i} \cdot \vec{x_j} + \beta_1) \tag{2.25}$$

$$K(\vec{x_i}, \vec{x_j}) = e^{\frac{-|\vec{x_i} - \vec{x_j}|^2}{2\sigma^2}} \tag{2.26}$$

All we have to do in order to apply SVMs to non-linear problems is to replace everywhere we see $\vec{x_i} \cdot \vec{x_j}$ with some suitable kernel function $K(\vec{x_i}, \vec{x_j})$. The function given in (2.26) will be the one we employ. It's known as the *Gaussian kernel*, and has been shown to give good performance over a variety of different problems. It has one free parameter, $\sigma$, which controls the width of the Gaussian function. Its value should be chosen to be of roughly the same magnitude as the length of your training vectors. The actual value should be determined through an empirical study of different trainings on your specific problem in order to optimize the SVM's performance. As a general rule, $\sigma$ determines how flexible the learned function's contours can become in the problem's unmapped feature space. The larger the magnitude of $\sigma$, the more rigid (less bendy) the contours. In fact, the Gaussian kernel function tends toward the linear kernel ( *i.e.* Equation (2.23) ) as $\sigma$ tends toward $\infty$. If $\sigma$ is taken to be too small, the SVM will simply "memorize" your training set and not generalize well.

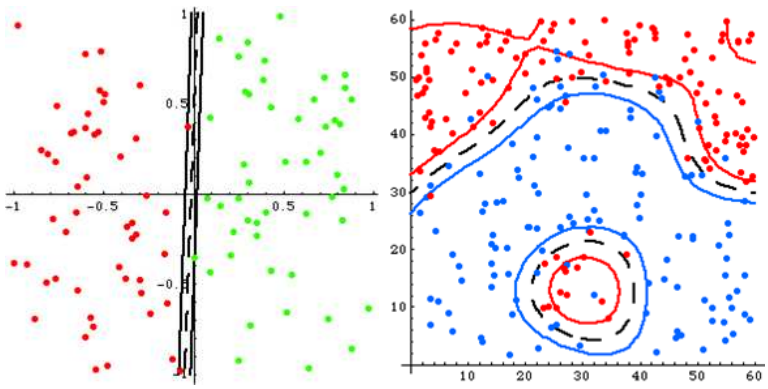



Figure 2.4: Linear and non-linear problems

## 2.5 Feature Ranking & Reduction

Another interesting property of the SVM formalism is that the learned function describes a geometrical space which, in and of itself, can be analyzed. This provides additional information above and beyond the simple binary answers derived from the learned function's sign. One piece of information which can be very insightful in analyzing a classification problem is a ranking of the features from most to least relevant.

In a SVM, each of the features is represented by one of the dimensions in the space defined by the learned function. The most straightforward way to achieve this ranking is by taking the gradient of the learned function (in its $d$ dimensions). By then sampling the gradient at different points, a numerical ordering of the most relevant features can be found. Recall, the learned function is defined as

$$f(\vec{z}) = \sum_{i=1}^{N} \alpha_i y_i K(\vec{x_i}, \vec{z}) - b \tag{2.27}$$

For a Gaussian kernel function, this becomes

$$f(\vec{z}) = \sum_{i=1}^{N} \alpha_i y_i e^{\frac{-|\vec{x_i} - \vec{z}|^2}{2\sigma^2}} - b \tag{2.28}$$

Note that the sum index $i$ in this context runs over the $N$ support vectors. Thus $\vec{x}_i$ refers to the $i^{th}$ support vector, and not the $i^{th}$ dimension of $\vec{x}$. The $j^{th}$ component of the gradient $(\vec{\nabla} f)_j$ is then given by

$$(\vec{\nabla} f)_j = \frac{\partial}{\partial z_j} f$$

$$\sum_{i=1}^{N} \alpha_i y_i e^{\frac{-|\vec{x_i} - \vec{z}|^2}{2\sigma^2}} = \sum_{i=1}^{N} \alpha_i y_i e^{\frac{-(\vec{x_i}1 - z_1)^2 - (\vec{x_i}2 - z_2)^2 - \ldots - (\vec{x_i}j - z_j)^2 - \ldots - (\vec{x_i}d - z_d)^2}{2\sigma^2}}$$

$$\Rightarrow (\vec{\nabla} f)_j = \sum_{i=1}^{N} \frac{\alpha_i y_i}{\sigma^2} (\vec{x_{ij}} - z_j) e^{\frac{-|\vec{x_i} - \vec{z}|^2}{2\sigma^2}} \tag{2.29}$$

where by $\vec{x_{ij}}$ it is meant the $j^{th}$ component of the $i^{th}$ support vector.

For completeness, the gradient for a linear kernel is

$$f(\vec{z}) = \sum_{i=1}^{N} \alpha_i y_i \vec{x_i} \cdot \vec{z} - b$$

$$(\vec{\nabla} f)_j = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_{ij} \tag{2.30}$$

To see why this method works, consider a simple two dimensional problem and a linear SVM. The decision plane and the margin define a surface of constant slope and orientation. The gradient in this situation is then a constant vector in some direction which is normal to the decision plane. Now, if one of the dimensions (features) turns out to be completely irrelevant to our decision, then our problem was really one dimensional. In this case we would find a decision plane parallel to the ignorable dimension, and a gradient pointing along the dimension that matters. Extended over many dimensions, the same general behavior still holds. The gradient points in the direction of the most relevant features, and is weighted in proportion to the pertinence each feature has with respect to the decision. See Figure 2.5 below.

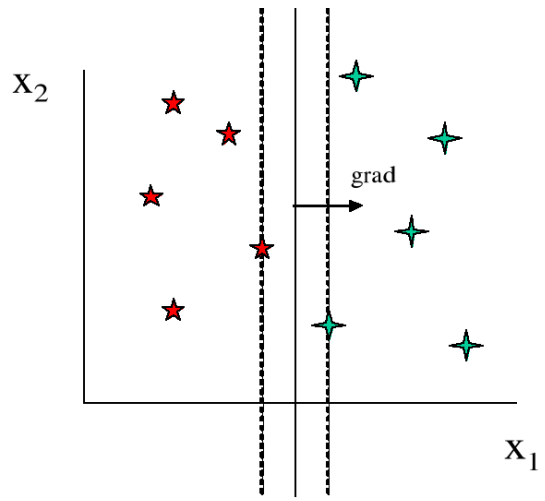



Figure 2.5: Using the gradient to rank features

In the non-linear case, the gradient varies from point to point in the decision space. To cope with this situation, we take the average of the gradient for some appropriate number of test points in the area of interest. This may have some drawbacks, as the averaging might wash out over a feature that is positively and negatively correlated over the test sample in different regimes. However, as an intuitive tool, the gradient is

a very nice way to gain confidence in your classifier's behavior and to better understand the dataset.

## 2.6 Implementation

To create a functional Support Vector Machine, it is required to solve the quadratic optimization problem given in Equation (2.21) with the appropriate kernel function included. The SVMs used in this thesis were implemented by the author and Jacob Borgman in C++. The implementation we used is based upon the improvements to Platt's Sequential Minimal Optimization (SMO) [30] method proposed by Keerthi, *et al.* [29]. This implementation was found to be faster than general quadratic optimization software, and other existing SVM implementations.