Support Vector Machines (a brief introduction)

Adrian Bevan

email: a.j.bevan@qmul.ac.uk

Queen Mary University of London

Outline

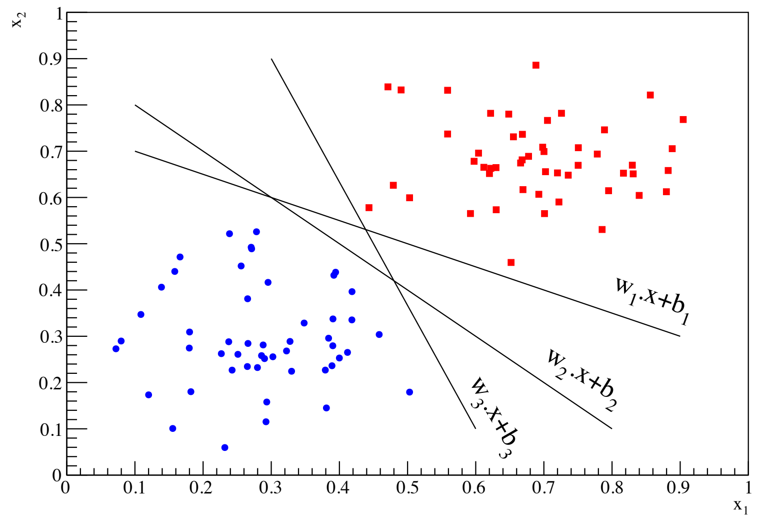
- Overview:
  - Introduce the problem and review the various aspects that underpin the SVM concept.
- Hard margin SVM
  - No mis-classification allowed
- Soft margin SVM
  - Misclassification permitted, but incurs a penalty.
- Kernel functions
  - Discuss kernel functions and their features before moving to SVMs.
- Checker board example
- References and further reading

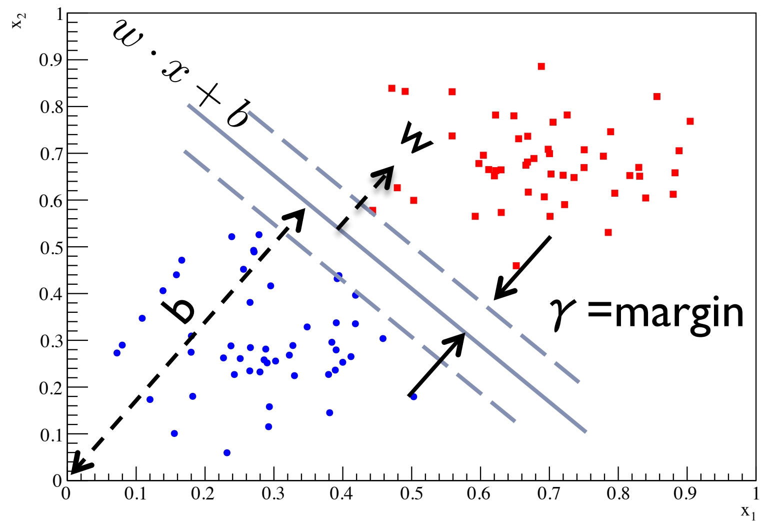
Overview

- Aim is to classify events into one of two types; signal (+1) and background (-1).
- Linearly separable problems can be treated using a hard margin (absolute classification with no classification error for the optimal SVM).
- Overlapping samples have some level of mis-classification, we use a soft margin approach and introduce parameters to describe the penalty of mis-classifcation: slack (ξ) and cost (C).
- We can use Kernel functions, K, to map data from our problem space (X) to a higher dimensional feature space (F) and solve the problem in this dual space.

Hard Margin SVM

- Identify the support vectors (SVs) : these are the points nearest the decision boundary.
- Use these to define the hyperplane that maximises the margin (distance) between the optimal plane and the SVs.





- Clearly if we can do this with a SVM – we would just cut on the data to get rid of our background!

Hard Margin SVM: Primal form

- Optimise the parameters for the maximal margin hyperplane via:

$$\arg\min_{w,b} \frac{1}{2}||w||^2$$

- such that

$$y_i(w \cdot x_i - b) \geq 1$$ (y$_i$ is the functional margin; γ_i)

- This is equivalent to solving the following optimisation:

$$\arg\min_{w,b}\max_{\alpha\geq 0}\left[\frac{1}{2}||w||^2 - \sum_{i=1}^{n}\alpha_i[y_i(w\cdot x_i - b) - 1]\right]$$

- Where: $w = \sum_{i=1}^{n}\alpha_i y_i x_i$ and $b = \frac{1}{N_{SV}}\sum_{i=1}^{n}(w\cdot x_i - y_i)$

Hard Margin SVM: Dual form

- The problem can be solved in the dual space by minimising the Lagrangian for the parameters α_i (Lagrange Multipliers):

$$\widetilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

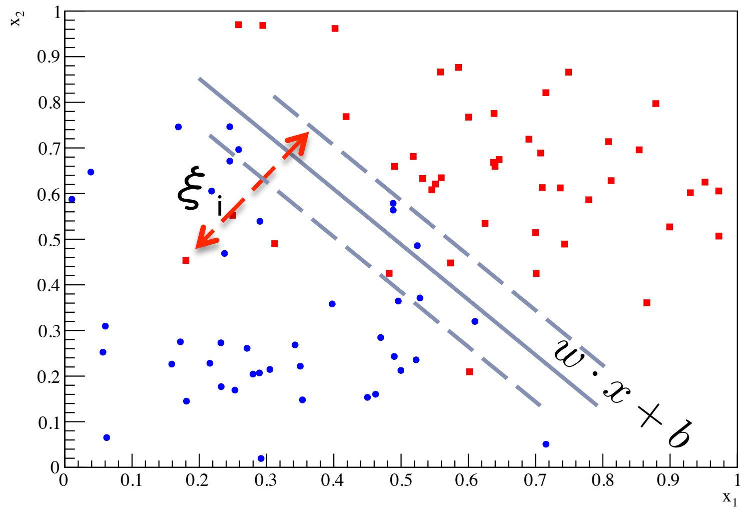
$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j).$$

- Such that: $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

- α_i are non-zero for support vectors only, and the last sum provides a constraint equation for optimisation.

Soft Margin SVM

- Relax the hard margin constraint by introducing mis-classification:
  - Describe by slack (ξ_i) and cost (C) parameters.
  - Alternatively describe mis-classification in terms of loss functions.
  - These are just ways to describe the error rate.



ξ_i = distance between the hyper-plane defined by the margin and the ith SV (i.e. now this is a mis-classified event).

Cost multiplies the sum of slack parameters in the optimisation.

We can use kernel functions to implicitly map from problem space to a higher dimensional dual feature space, to simplify optimisation.

The MVA architecture complexity is embedded in the kernel function.

- These are much more useful.

Soft Margin SVM: Dual form

- The Lagrangian to optimise simplifies when we introduce the slack parameters:

$$\widetilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

- Where

$$0 \leq \alpha_i \leq C$$

- and as before we constrain:

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

Kernel functions

- The kernel function[#] K(**x**,**y**) extends the use of inner products on data in a vector space to a transformed space where

$$K(x,y) = \langle \phi(x) \cdot \phi(y) \rangle$$

- The book
  - by Nello Cristianini and John Shawe-Taylor, called *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000

  discusses a number of KFs and the conditions required for these to be valid in the geometrical representation that SVMs are constructed from.

[#] abbrev. KF = kernel function

Kernel functions: Radial Basis Function

- This is a commonly used KF that maps the data from X space to F space using a single parameter. The distance between two support vectors is computed and used as an input to a Gaussian KF.
- For two data x and y in X space we can compute K(x, y) as

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

- There is one tuneable parameter in the mapping from X to F; this is given by $\Gamma=1/\sigma^2$.

Kernel functions: Multi-Gaussian

- This is an extension of the RBF function that recognises that the bandwidth of data in problem space can differ, and so the norm of the distance between two support vectors can result in loss of information.
- To overcome this limitation we can introduce a σ_i for each dimension in X.

$$K(x, y) = \prod_{i=1}^{\dim(X)} e^{-||x_i - y_i||^2/\sigma_i^2}$$

- The down side is that in doing so we have increased the number of parameters that need to be determined to optimally map from X to F.

Kernel functions: Multi-Gaussian

- The KF given by
$$K(x,y) = \prod_{i=1}^{\dim(X)} e^{-||x_i - y_i||^2/\sigma_i^2}$$
allows for a different σ for each dimension in the problem.
- However it explicitly neglects correlations between dimensions in the data. i.e. there is no parameterisation of covariance between x_i and x_j here. This can be accommodated by a further generalisation:
$$K(x,y) = e^{-(x-y)^T \Sigma^{-1} (x-y)}$$
- Here Σ is an n×n matrix, where n=dim(x). It is the covariance matrix for the problem.
- Often Σ is assumed to be diagonal for simplicity.
  - Typically too many parameters to optimise for the covariance matrix

Polynomial

- There are many different types of polynomial kernel functions that one can study.
- A common variant is of the form:

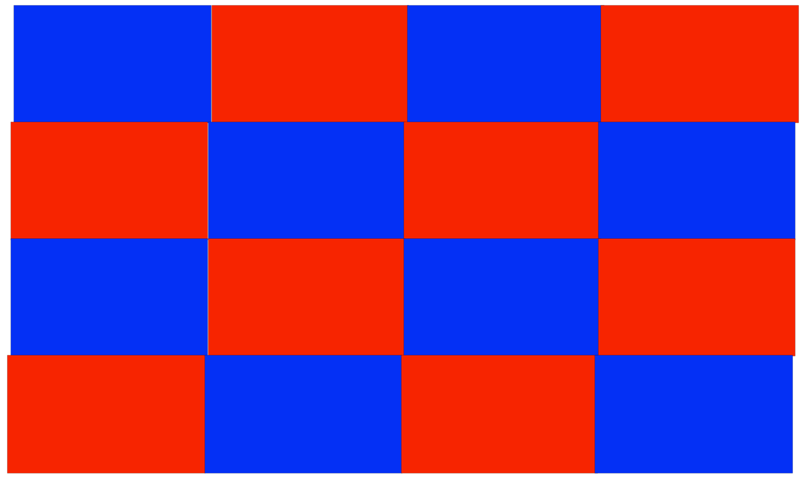
$$K(x,z) = (\langle x \cdot z \rangle + c)^d = \left(\sum_{i=1}^{\ell} x_i z_i + c \right)^d$$

- where c and d are tuneable parameters. The sum is over support vectors (i.e. events in the data set for a soft margin SVM).

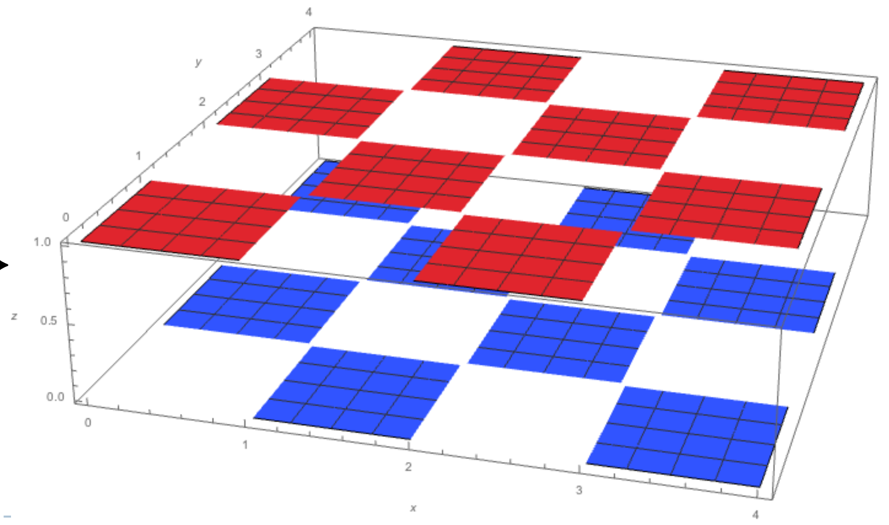
Example: the checker board

- Generate squares of different colour, and use an SVM to classify the pattern into +1 and -1 targets.
  - This problem is a hard margin SVM type of problem as the data do not overlap.
  - It's not easy to solve in 2D (x, y) with a linear discriminant, but it is clear to see that a 3D (x, y, colour) space would allow us to separate the squares.

$X \mapsto F$

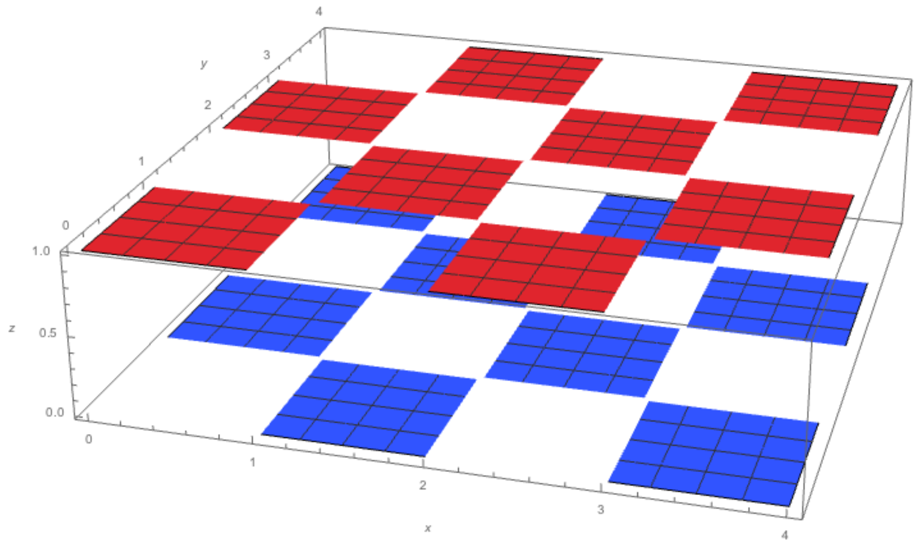


$\mapsto$



Example: the checker board

- Generate 1000 events in the blue and red squares and give each event x and y values.



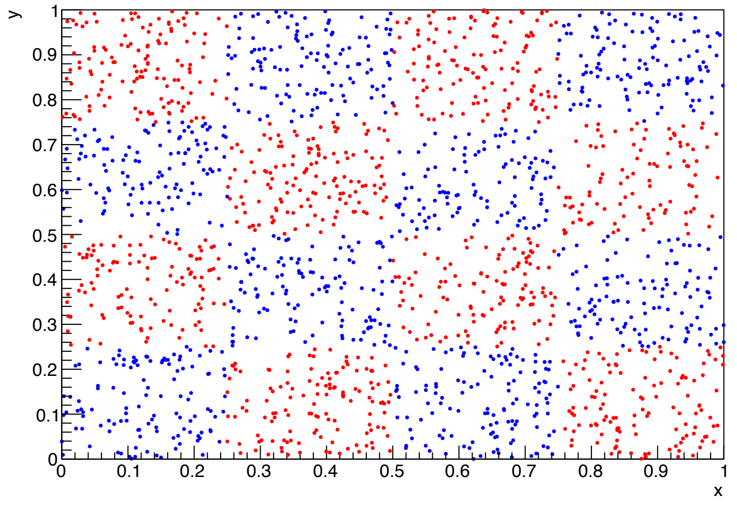
This is the ideal feature space that we would like to implicitly map into.

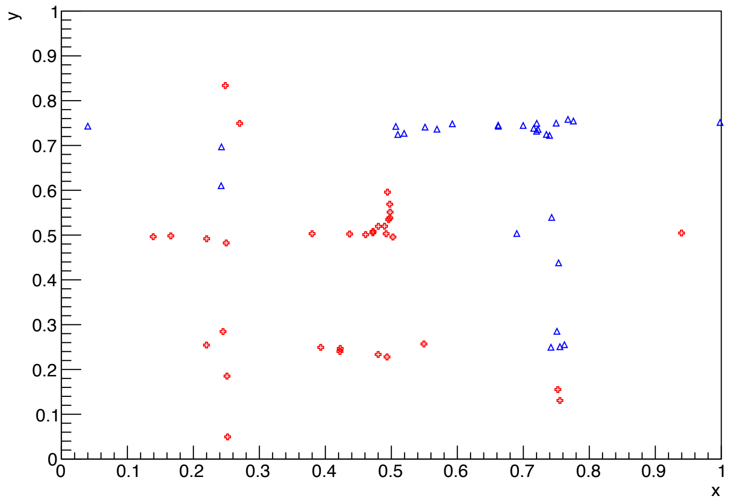
Because we implicitly do the mapping via the choice of KF, in practice we don't map into this space; but we do map into another space that allows us to try and separate signal from background (i.e. blue from red).

- Use a multi-Gaussian kernel function with Γ_1=1, Γ_2=2 and cost of 10^4 (not optimised) to see what separation we can obtain.

Example: the checker board

- Correctly classified events
- Incorrectly classified

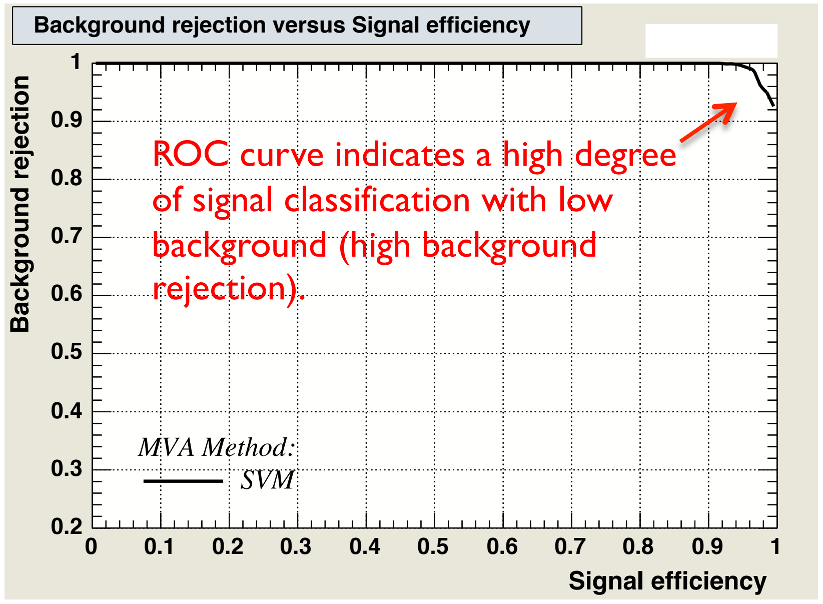


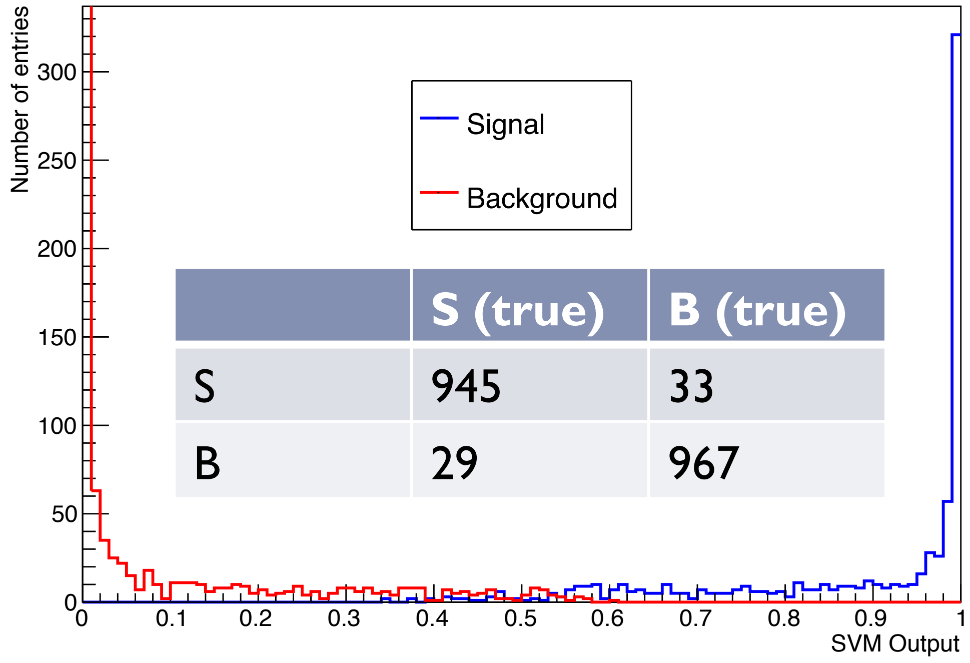


- Signal mis-classification rate ~3.3%
- Background mis-classification rate ~3.7%

Example: the checker board

- The confusion matrix ([in-]correctly classified events) for this example shows a high level of correct classification:





	S (true)	B (true)
S	945	33
B	29	967

- This SVM does a good job of separating signal from background.
- An optimised output would provide a better solution.

Summary

- SVMs are a modern MVAs based on linear decision boundaries.
- They have the advantage that they can be fitted using a small number of data (support vectors).
- They have the dis-advantage that the computational power required to compute an SVM scales with the number of support vectors (i.e. the number of data).
  - There are data redaction methods (e.g. chunking/dissection/cutting) that can help alleviate this issue.
  - There are algorithmic methods that can help alleviate this issue.

References

- Books:
  - An Introduction to Support Vector Machines and other kernel-based learning methods, Cristianini and Shawe-Taylor (CUP, 2014).
  - Statistical Analysis Techniques in Particle Physcis, Narsky and Porter (Wiley-Vch, 2014).

- Tools:
  - Matlab and R have interfaces to SVM libraries. libsvm is a popular implementation that is described in detail at:
    - https://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - ROOT has an interface to R (and hence the R svm packages). It also has an SVM implemented within TMVA:
    - https://root.cern.ch

Sequential Minimal Optimisation (SMO)

- The dual form of the Lagrangian minimised for SVMs depends on Lagrange multipliers (α_i) that satisfy a constraint equation $\sum_{i=1}^{n} \alpha_i y_i = 0$ as opposed to the weight and bias parameters.

- Rather than take a brute force approach to optimising for the α_i, SVMs use the constraint equation to select pairs of SVs with the largest slack values and change the α_i's in pairs to retain the overall constraint.

- This iterative process occurs for all SVs a number of times; so while the number of steps is larger than a brute force approach, the overall computing cost is smaller.