Work done in collaboration with Tom Stevenson, Agni Bethani*, Rodrigo Gamboa-Goni, Jon Hays and Lewis Milward.

* now at LPSC Grenoble

Results shown here use the updated code in TMVA (in the git repository; will roll out into standard ROOT downloads).

Support Vector Machines and generalisation in HEP

Adrian Bevan

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

Queen Mary University of London

Outline

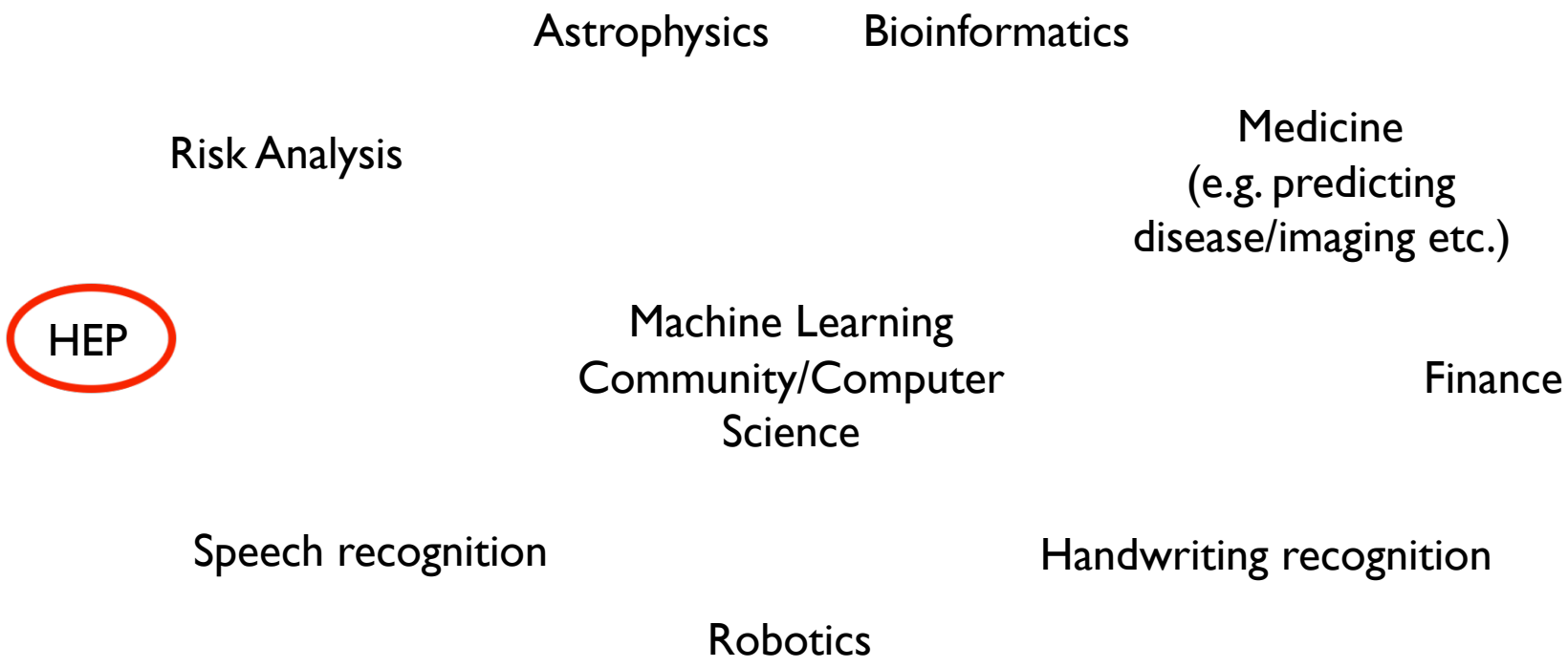
- Overview:
  - Introduce the problem and review the various aspects that underpin the SVM concept.
- Hard margin Support Vector Machine (SVM)
  - No mis-classification allowed.
- Soft margin SVM
  - Misclassification permitted, but incurs a penalty.
- Kernel functions
  - Discuss kernel functions and their features before moving to SVMs.
- Examples
- Generalisation
- Summary

Overview

- Aim: classify events into signal (+1) and background (-1).
- Linearly separable problems can be treated using a hard margin (absolute classification with no mis-classification).
- Overlapping samples have some level of mis-classification; use a soft margin approach and introduce parameters to describe the penalty of mis-classifcation: slack (ξ) and cost (C).
- Use Kernel functions (KF) to map data from the problem space (X) to a feature space (F) and solve the problem in this dual space.
- Need to be sure that we learn a general solution, rather than learn the fluctuations (over fit) to the training data.

Part I - SVMs

Widely used ML algorithm: some example fields



- HEP problems are low dimensional simple use cases compared with issues being addressed for some of the existing fields using these algorithms.

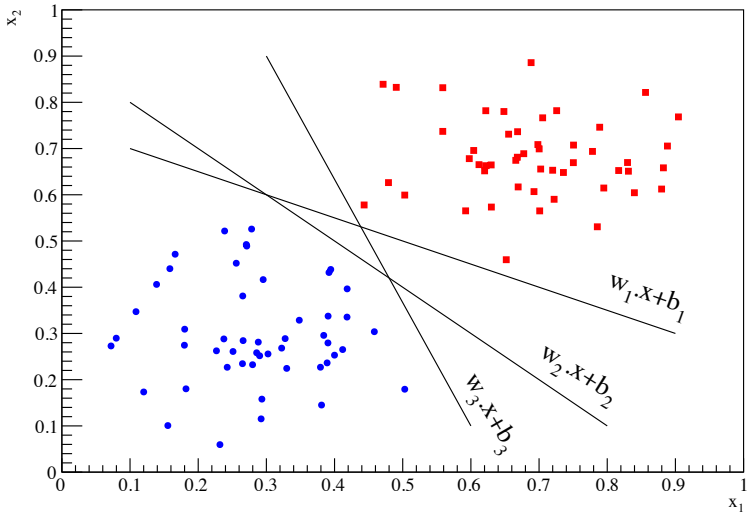
Widely used ML algorithm: HEP examples

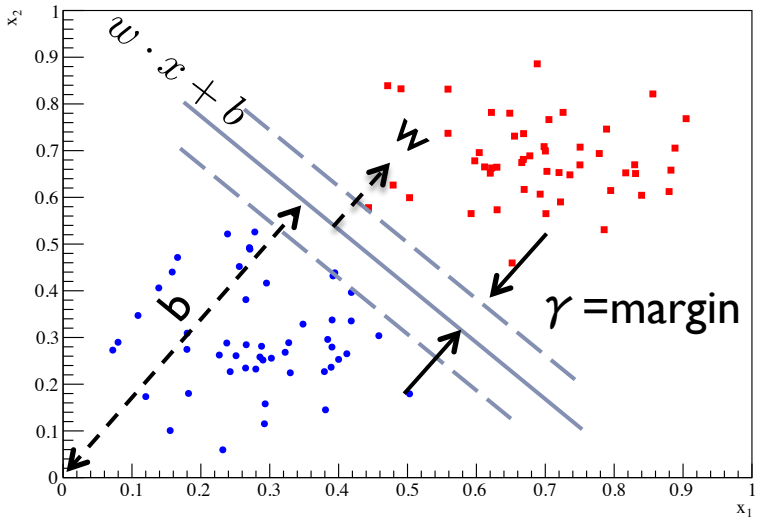
- **Background suppression (jets):**
  - F. Sforza, V. Lippi, Nucl. Inst. Meth. A**722**, (2013), p11-19 (arXiv:1407.0317).
- **Flavour Tagging:**
  - P. Vannerum et al., Freiburg EHEP-99-01 (hep-ex/9905027).
- **Machine Physics:**
  - Bijan Sayyar-Rodsari, C. Schweiger, SLAC-R-948.
- **Review:**
  - A. Vossen, Part of the proceedings of the Track 'Computational Intelligence for HEP Data Analysis' at iCSC 2006 arXiv:0803.2345.
- **Top:**
  - A. Vaiciulis, Nucl. Instrum. Meth. A502 (2003) 492-494 (hep-ex/0205069).
  - S. Ridella et al., IEEE Conf.Proc. (2004) no.3, 2059-2064.
  - B. Whitehouse, FERMILAB-THESIS-2010-61.

Studied at LEP/Tevatron/LHC

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.





- If we can do this with a SVM – we would simply cut on the data to separate classes of event.

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)

- Equivalent to solving the following optimisation problem:

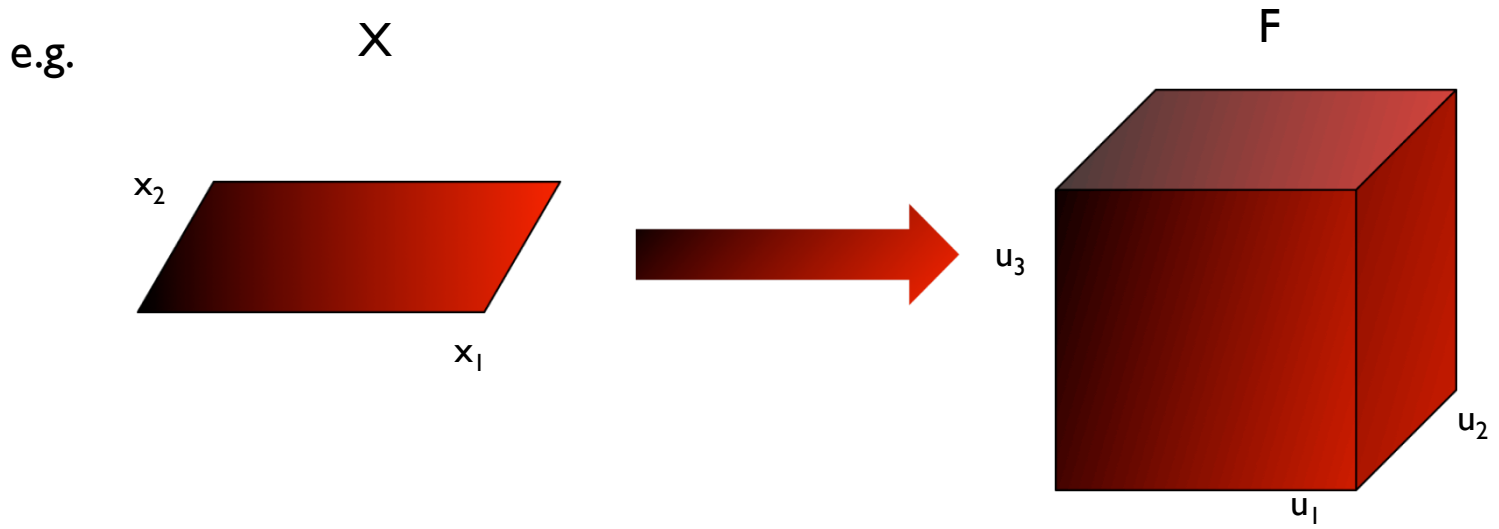
$$\arg\min_{w,b}\max_{\alpha \geq 0}\left[\frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i[y_i(w \cdot x_i - b) - 1]\right]$$

- 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: KFs

- We can introduce the use of a KF to implicitly map from our problem space X to some feature space F.
- Define the function:

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



- We don't need to know the details of the mapping; this is the "kernel trick".

B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond. MIT Press, 2002.*

Hard Margin SVM: KFs

- We can introduce the use of a KF to implicitly map from our problem space X to some feature space F.
- Define the function:

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

e.g.

$$x \in \mathbb{R}^n \quad \Rightarrow \quad F \in \{\phi(x) | x \in X\}$$

- We don't need to know the details of the mapping; this is the "kernel trick".

B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond. MIT Press, 2002.*

Hard Margin SVM: Dual form

- The problem can be solved in the dual space by minimising the Lagrangian for the Lagrange multipliers α_i :

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

dot product

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

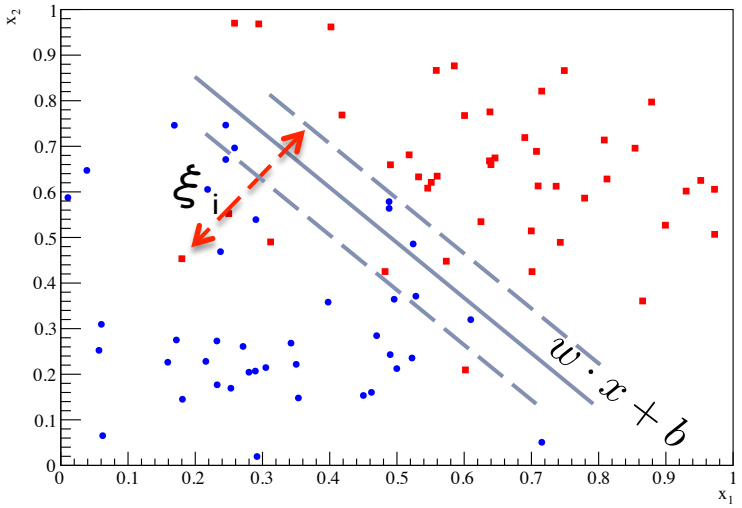
KF

- 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.
- The 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 i^{th} SV (i.e. now this is a mis-classified event).

Cost multiplies the sum of slack parameters in optimisation.

MVA architecture complexity is encoded by the KF.

- 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:

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

The optimisation problem in dual space is essentially the same for the hard and soft margin SVMs.

Kernel functions

- The KF, 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

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

Kernel functions: Radial Basis Function

- Commonly used KF that maps the data from X to F.
- 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

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

- One tuneable parameter in mapping from X to F; given by $\Gamma=1/\sigma^2$.

Previously Implemented in TMVA

Kernel functions: Multi-Gaussian

- Extend the RBF function to recognise that the bandwidth of data in problem space can differ for each input dimension; i.e. the norm of the distance between two support vectors can result in loss of information.

- Overcome this by introducing a $\Gamma_i = 1/\sigma_i$ for each dimension:

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

- Down side ... we increase the number of parameters that need to be optimally determined for the map from X to F.

New to TMVA

Kernel functions: Multi-Gaussian

- The KF

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

  allows for a different σ for each dimension in the problem.
- Neglects correlations between input dimensions in data. 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.

NOT in TMVA: computationally very expensive...

Polynomial

- There are many different types of polynomial kernel functions that one can study.

- A common variant is of the form:

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

- c and d are tuneable parameters.

- The sum is over support vectors (i.e. events in the data set for a soft margin SVM).

New to TMVA

Products and sums

- KFs that satisfy Mercer's conditions* can be combined to make more complicated KFs:
  - The sum of KFs is a valid KF.
  - The product of KFs is a valid KF.

J. Mercer. Phil.Trans.Roy.Soc.Lond.,A209:415, 1909.

* Mercer's conditions require that the Gramm matrix formed from SVs is positive semi-definite. This is a consequence of the geometric interpretation of SVMs given x is real. Modern extensions of the SVM construct allow for complex input spaces, and for example can be based on Clifford algebra to accommodate this extension.

Complex input spaces are of interest for electronic engineering problems.

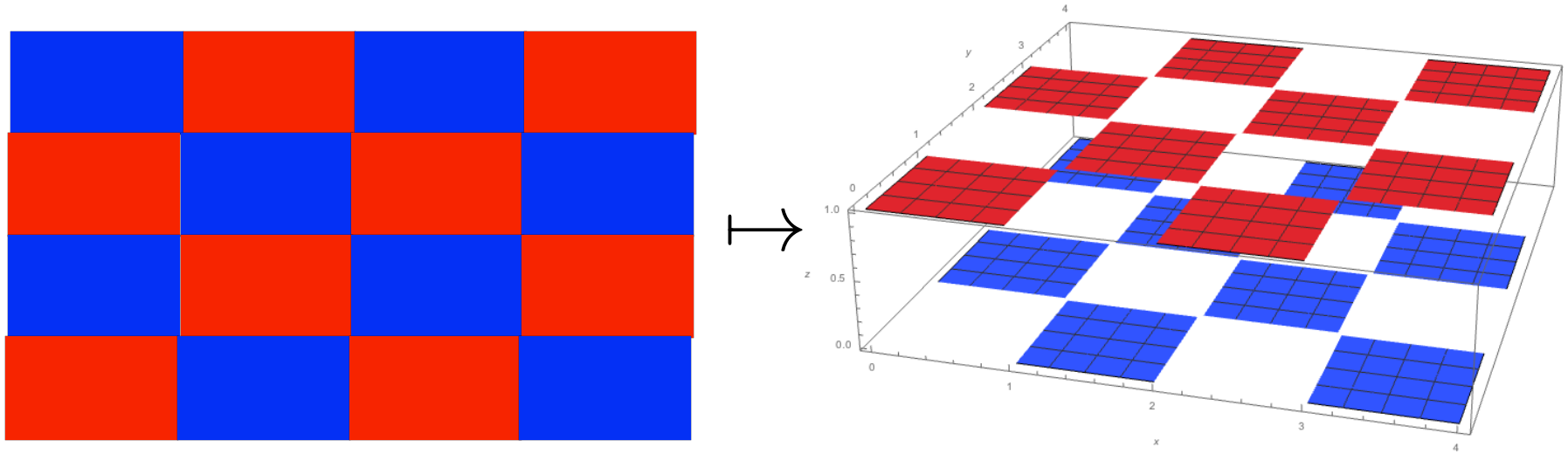
N.B. It is conceivable that one could be interested in using these if an amplitude analysis were to be written using SVMs to directly extract phase and magnitudes... but that could also be incorporated by mapping the complex feature space element into a doublet of reals.

New to TMVA

Example: the checker board

- Generate squares of different colour.
- Use SVM to classify the pattern into +1 and −1 targets.
  - Hard margin SVM problem; but can solved for using soft margin SVM.
  - Not easy to solve in 2D (x, y) with a linear discriminant, but a 3D space of (x, y, colour) allows us to separate the squares.

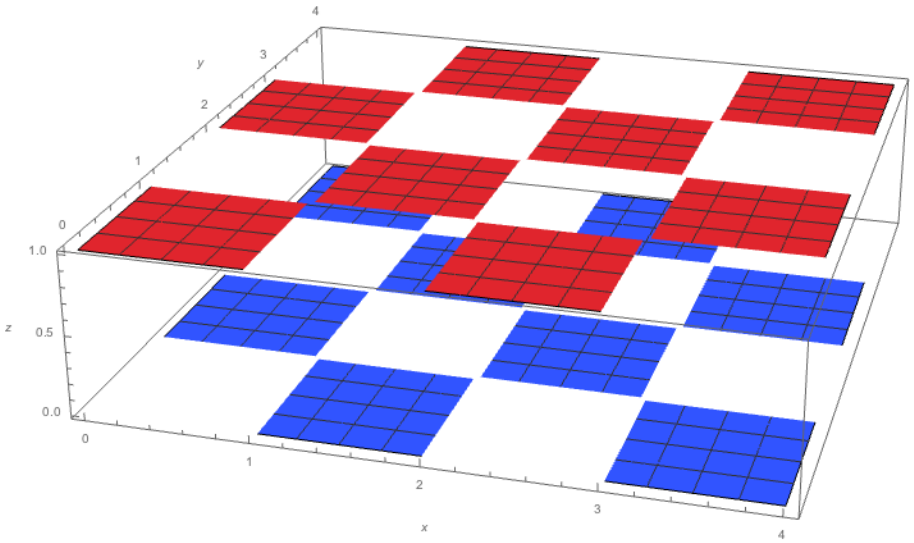
$$X \longmapsto F$$



- Want to find a KF that approximates this mapping.

Example: the checker board

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



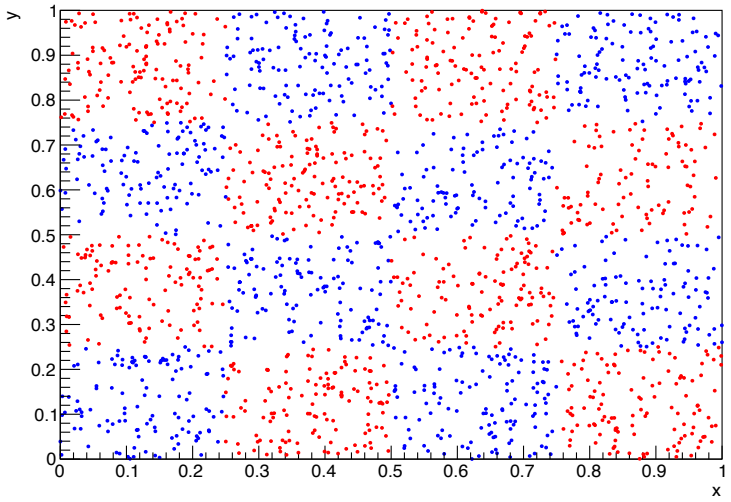
This is the ideal feature space that we would like to implicitly map into.

Because we implicitly do the mapping via choice of KF, in practice we don't explicitly map into this space; but we implicitly map into another space that we hope will be *approximately topologically equivalent*.

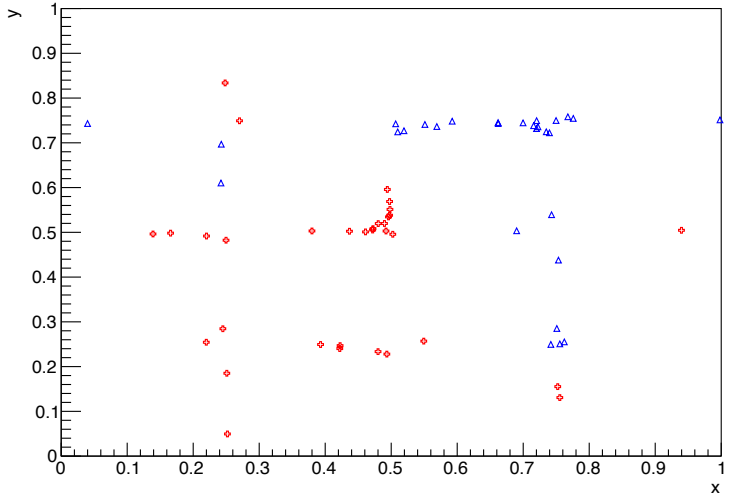
- e.g. Use a multi-Gaussian kernel function with $\Gamma_1=1$, $\Gamma_2=2$ and cost of 10^4 (not optimised) to see what separation we can obtain.

Example: the checker board

- Correctly classified events



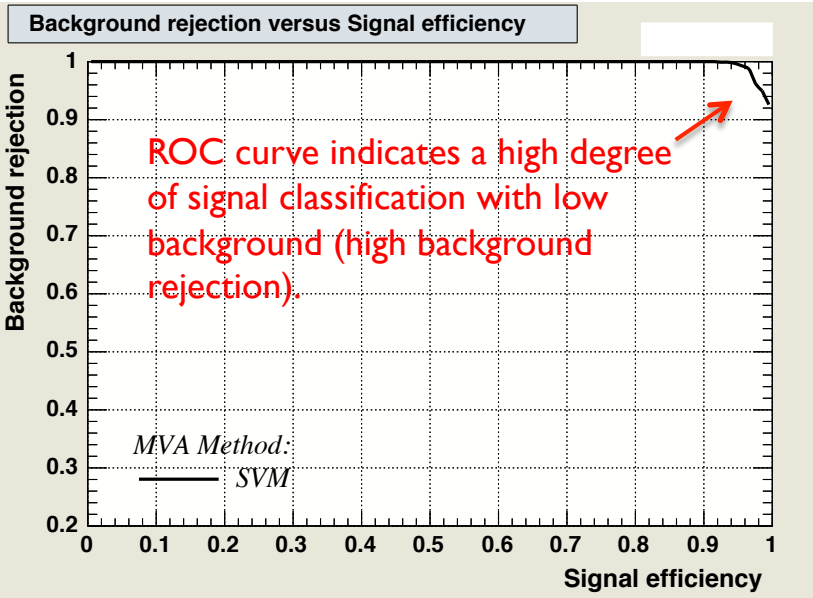
Incorrectly classified

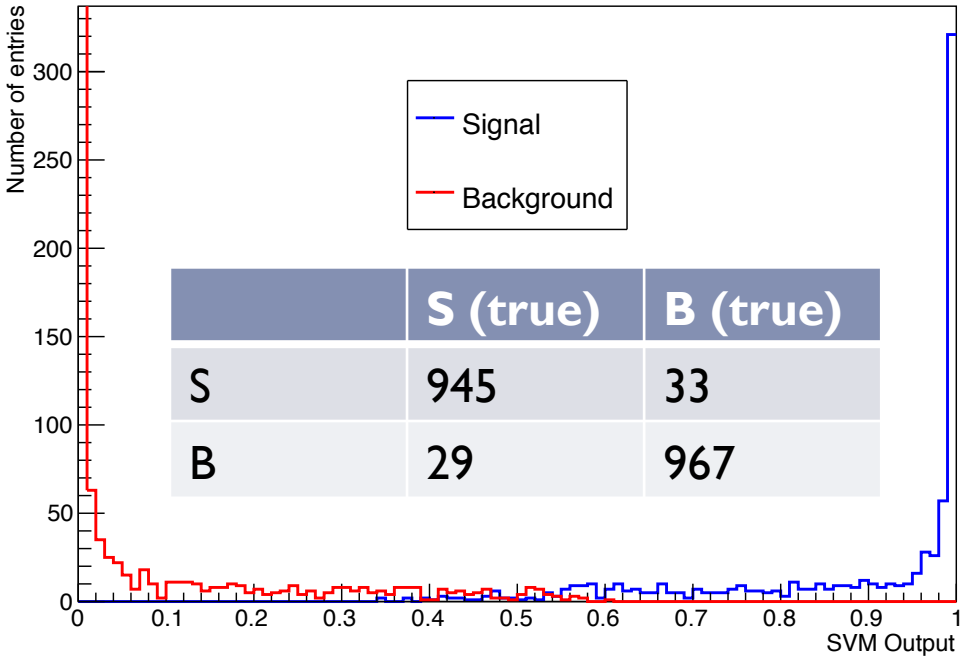


- Signal mis-classification rate ~3.3%.
- Background mis-classification rate ~3.7%.

Example: the checker board

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





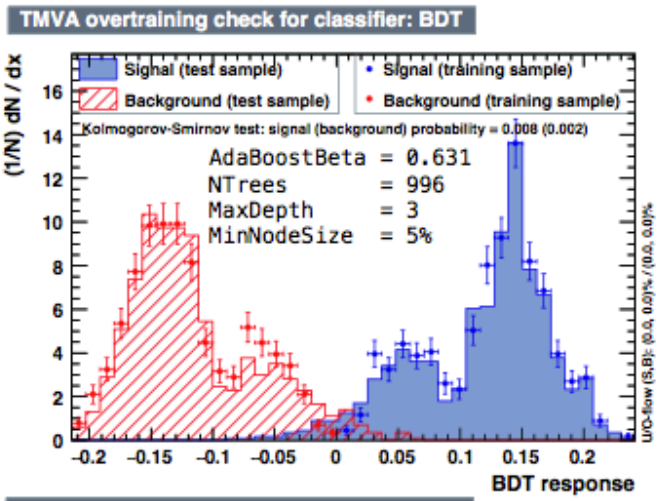
	S (true)	B (true)
S	945	33
B	29	967

- This SVM does a good job of separating signal from background.
- An optimised output would provide a better solution.
- BDTs and NNs work well with this kind of problem as well.

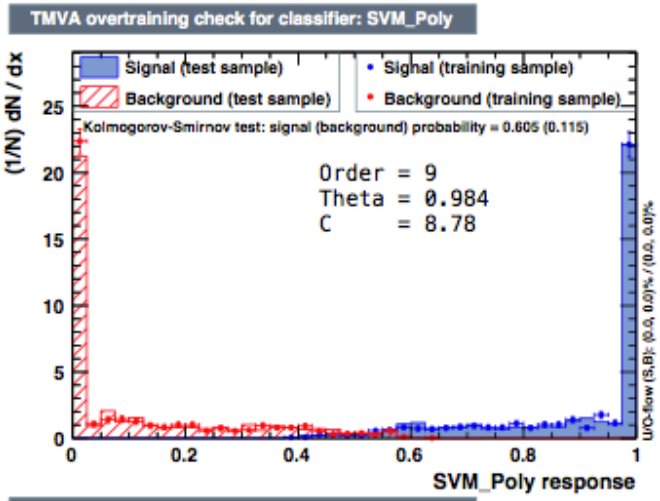
Optimised results

Trained using the hold out method of cross validation (what is normally done in TMVA), with optimised hyper-parameters.

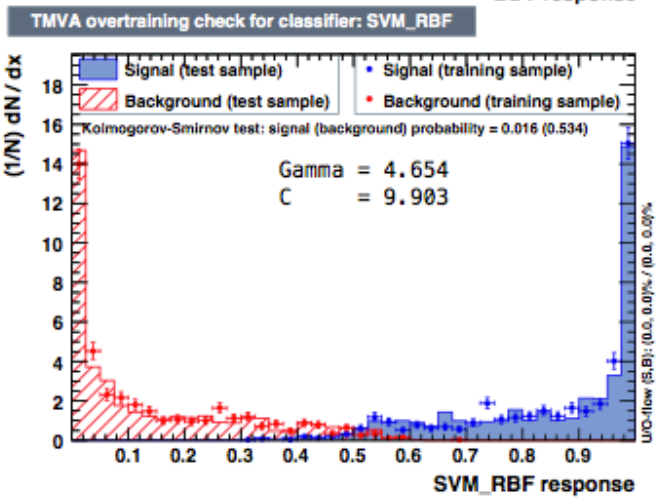
BDT



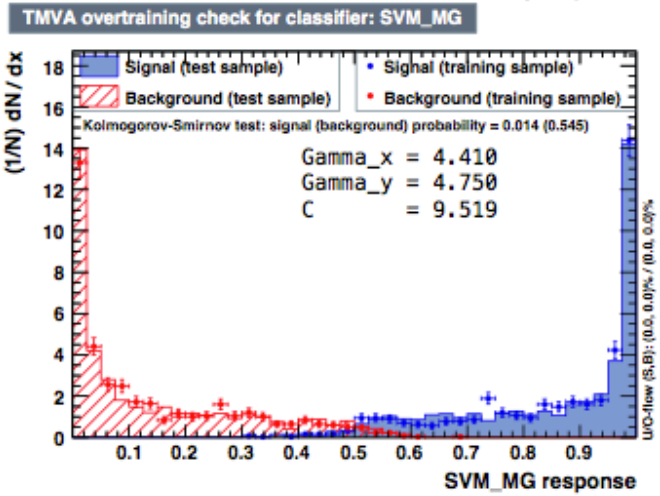
SVM Polynomial (1)



SVM RBF (2)



SVM Multi-Gaussian (3)



Now for a HEP example: H→ττ

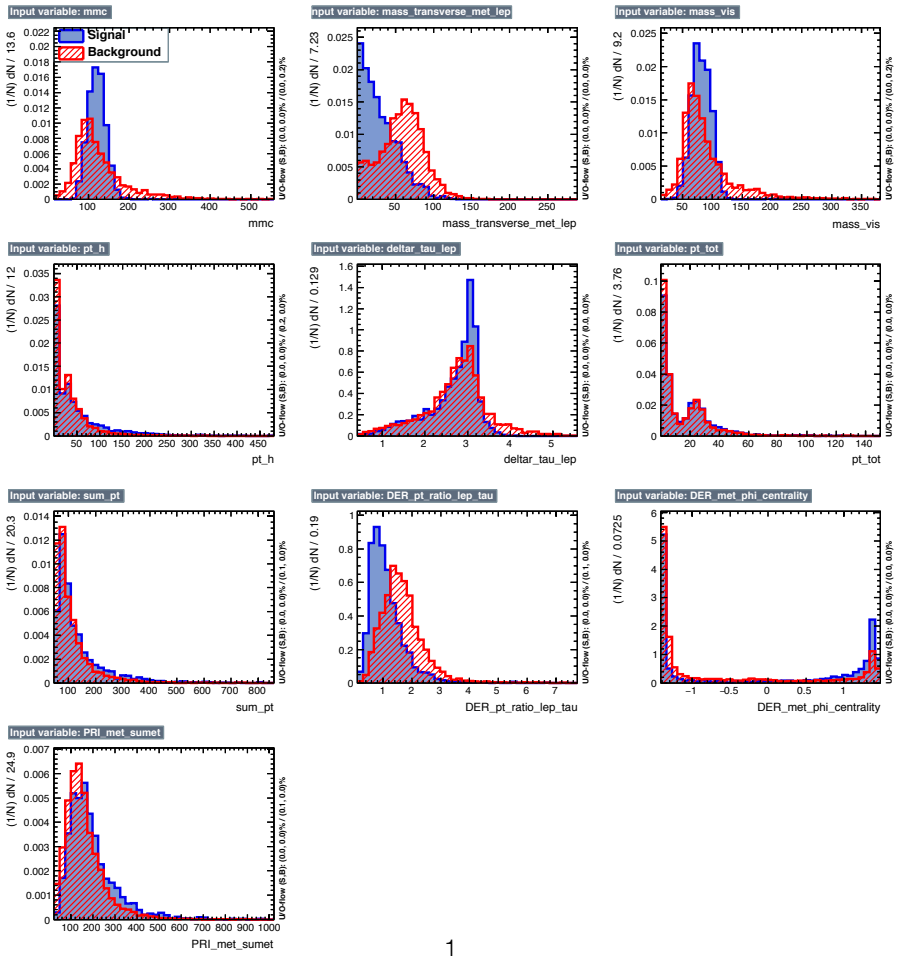
- Use the Kaggle data challenge sample of signal and background events.
  - LHC data (from ATLAS).
  - Packaged up in a convenient format (CSV file).
  - Sufficient description of variables provided for non-HEP users to apply machine learning (ML) techniques to HEP data.
  - Real application to compare performance for different KFs and different MVAs.

https://www.kaggle.com/c/higgs-boson

https://www.kaggle.com/c/higgs-boson

$H \rightarrow \tau^+\tau^-$ inputs

- Use 10 variables as inputs; 20K events.



MMC
transverse mass between MET and lep
Visible invariant mass of H
$p_T(H)$
R between τ_{had} and lepton
$p_T(tot)$
$\Sigma\, p_T$
$p_T(lepton)/p_T(had\ \tau)$
MET φ centrality
MET_{total}

This selection of variables is not optimised, and is selected in order to show a physics example for illustrative purposes.

https://www.kaggle.com/c/higgs-boson

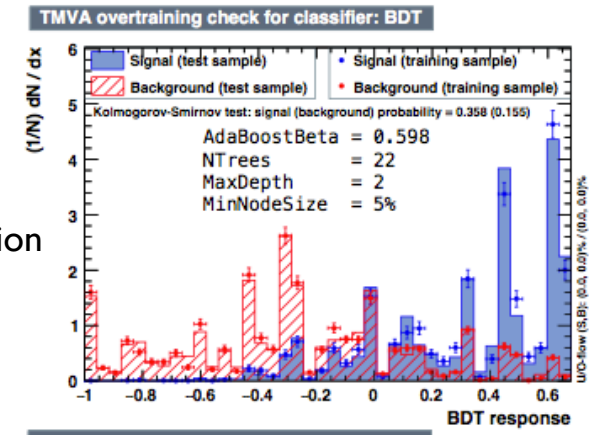
Trained using the hold out method of cross validation (what is normally done in TMVA), with optimised hyper-parameters.

H→ττ MVAs

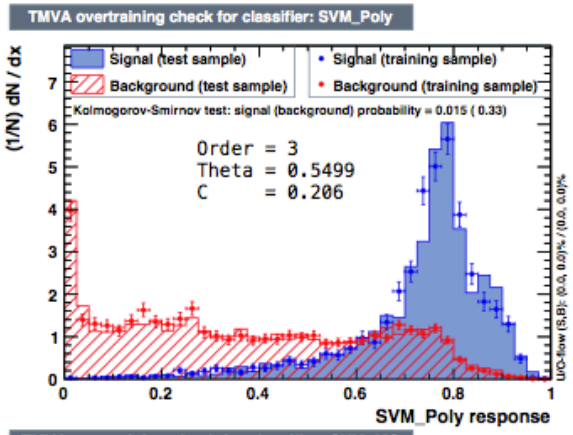
- NOTE: this is an illustrative example – not a fully optimised analysis of the sample; hyper-parameters are optimised.

BDT

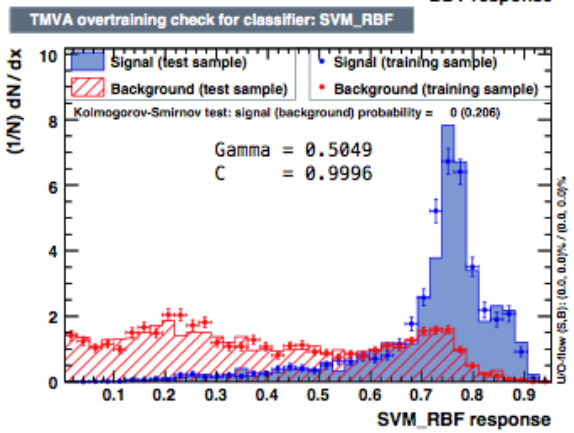
Spiky as optimisation chooses a low number of trees.



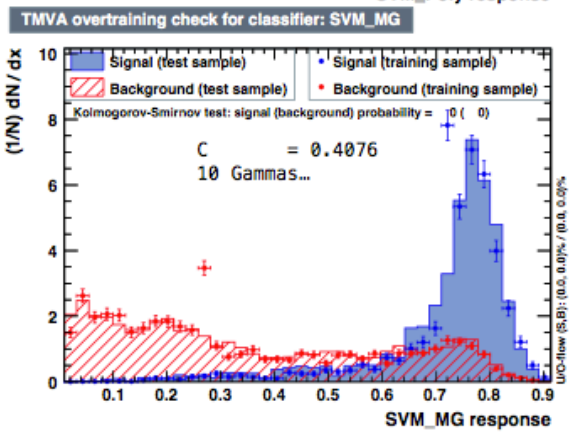
SVM Polynomial (1)



SVM RBF (2)



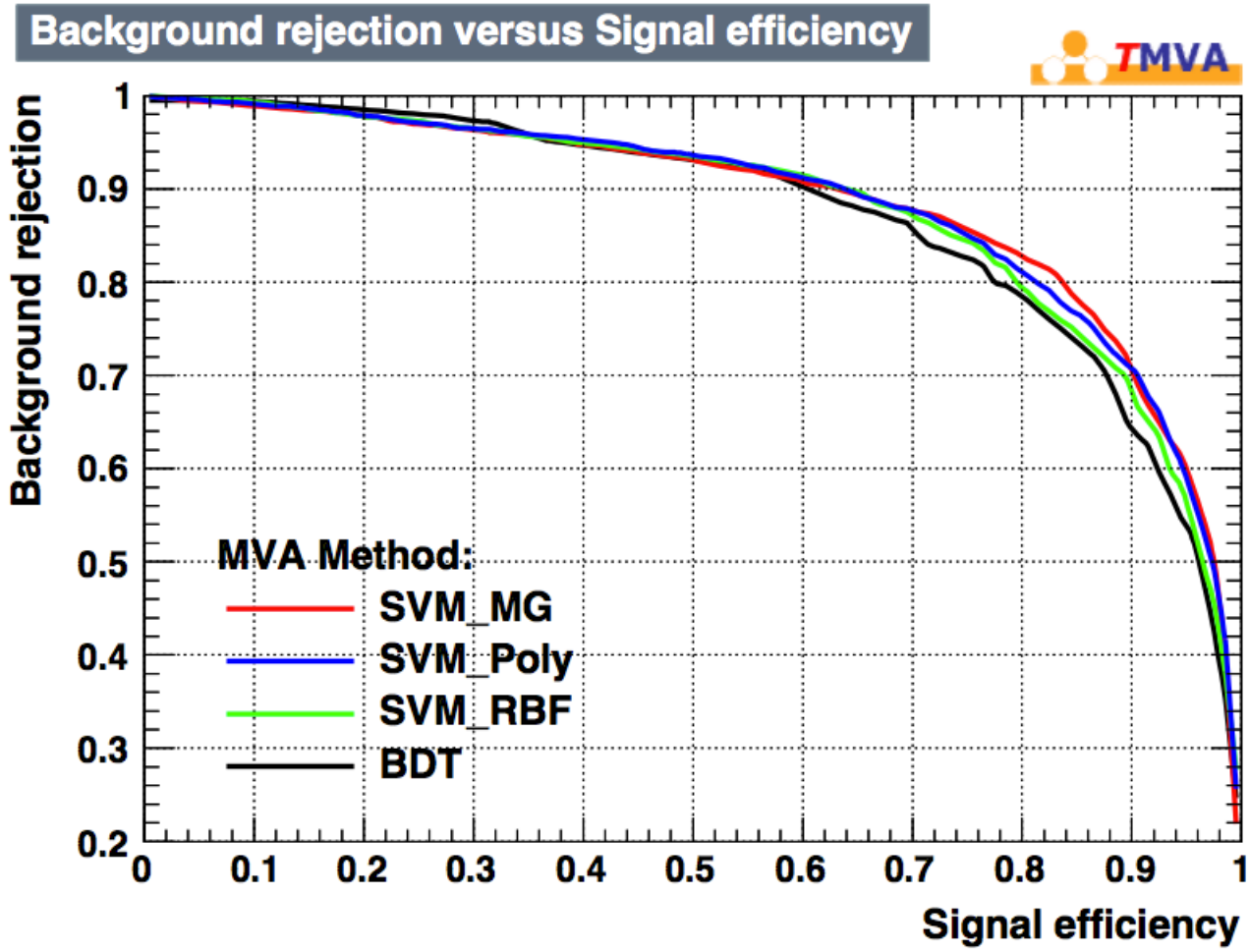
SVM Multi-Gaussian (3)



https://www.kaggle.com/c/higgs-boson

H→ττ performance

- SVM provides comparable performance to BDT.



SVM OPTIONS TABLE

TJS@IML

Changed to "GammaList"

Option	Default	Predefined Values	Description
Kernel	RBF	RBF, MultiGauss, Polynomial, Prod, Sum	Choice of kernel function. RBF, Multi-Gaussian and Polynomial are standard kernels. Prod or Sum require **MultiKernels** option to be specified with a list of kernels.
MultiKernels	—	—	Option for specifying products or sums of kernels. String with delimiter */+ for product/sum, e.g. `MultiKernels=RBF*Polynomial`.
Gamma	1	—	RBF kernel parameter. Related to the width, $\Gamma = 1/2\sigma^2$.
Theta	0	—	Polynomial kernel parameter.
Order	1	—	Polynomial kernel parameter.
~~Gammas~~	—	—	Multi-Gaussian kernel parameters. Requires same number of gammas as input variables. Separated by , delimiter e.g. `Gammas=0.8,0.7,0.4` for problem with 3 input variables.
Tune	All	—	Choice of kernel parameters and range to optimise. String separated by commas for parameters and range within square brackets separated by semicolon e.g. `Tune=Gamma[0.1;0.9;8]` will optimise Γ between 0.1 and 0.9 in 8 steps if using Scan to optimise.
C	1	—	Cost parameter.
Tol	0.01	—	Tolerance parameter.
MaxIter	1000	—	Maximum number of training loops.

SVM: Summary

- SVM algorithm functionality improved in TMVA.
- You can download the updated code from the ROOT git repository to start using this today.
- SVM performance comparable with BDTs for the Kaggle challenge data.
- Other ways to access SVM implementations:
  - TMVA has an R interface, so can access libsvm via that.
  - Matlab has libsvm interface.
  - Mathematica has an SVM interface.
  - etc.

Part II - Generalisation

- Is our MVA fine tuned or is it general?

What do we mean by fine tuned?

What do we mean by fine tuned?

Generalisation

- We have an MVA – but fitting/training does not guarantee that this is general.
  - Over fitting (SVMs) or over training (BDTs/ANNs) means that your classification/regression algorithms are honed to fluctuations in the data.
  - TMVA uses a binned KS probability computed using TH1 as part of an over training check.
    - Do not rely on this quantity to decide if your MVA is over-trained or not as the TH1::KolmogorovTest(...) function does not behave as you would expect.

Regularisation is another way to enforce generalisation (e.g. see C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995); not discussed here.

Cross validation

- Commonly used ML technique to ensure a more robust training for a classifier.

- Several types used in the literature:
  - Hold out method (simplest form – used by HEP).
  - k-fold cross validation (CV).
  - leave p-out CV / leave one out CV.
  - progressive validation.

For CV references, see the review: S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. Statistics Surveys, 4:4079, 2010.

For progressive validation: see A. Blum, A. Kalai, and J. Langford. Beating the hold-out: bounds for k-fold and progressive cross-validation. *99 Proceedings of the twelfth annual conference on Computational learning theory*, pages 203–208, 1999.

- Common theme:
  - Split data into subsets and train/validate on one part of the data, test performance against the residual sample.

Ideal training process

- Divide our data sample into three parts:
  - 1) test sample – only use this at the very end of the process;
  - 2) training sample;
  - 3) validation sample.

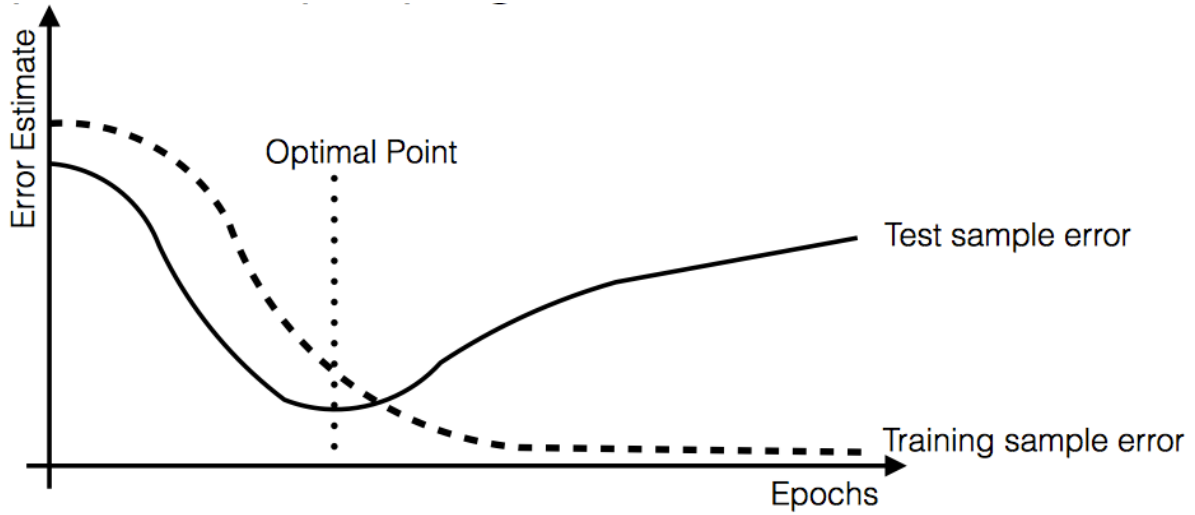
} Division of data depends on training method

- Cross validation refers to the use of training and validation samples to obtain a set of hyper parameters for an MVA.

- Once cross validated, we can then compare the MVA performance / output against the unseen test sample.

Hold out method

- Simplest form of cross validation.
- Typically encountered in HEP.
- Splits data into training and validation samples.
- e.g. MLP training; compare error rates of training and validation samples as a function of training epoch.



Devroye, L. and Wagner, T. J. (1979). Distribution-Free performance Bounds for Potential Function Rules. IEEE Transaction in Information Theory, 25(5):601–604.

k-fold CV

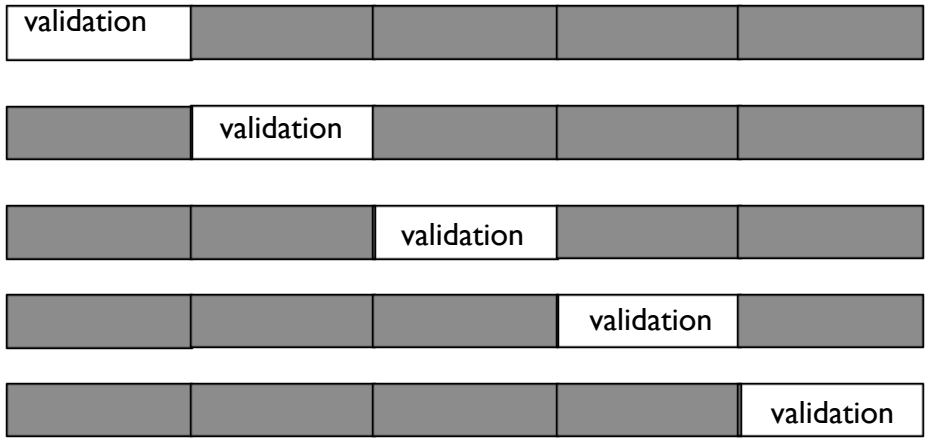
1) Reserve a test sample from the data $\Omega \rightarrow \Omega' \subset \Omega$ (if one wants to validate generalisation beyond the *k*-fold cross validation step).
2) Randomly split the remaining data into k sub samples:
   $\Omega' \rightarrow \Omega_i, i = 1, 2, \ldots k$.
3) Cycle through training *k* times, each time leaving one sub-sample out.

e.g. 5-fold cross validation: train 5 times dropping out one sub-sample at a time.

Use average MVA parameter configuration obtained from the *k*-folds.

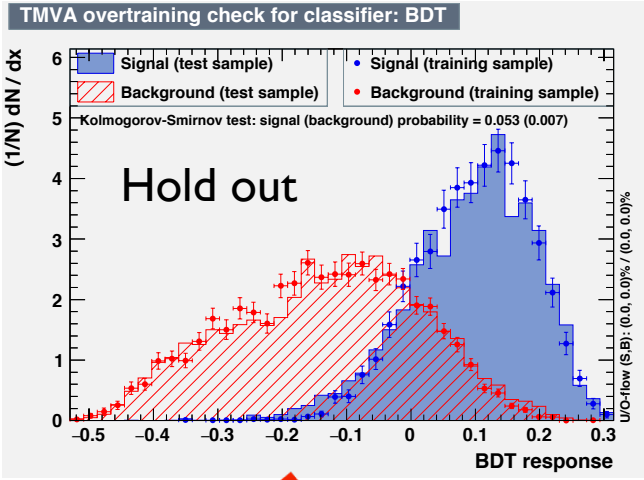
The optimal value of k needs to be determined.

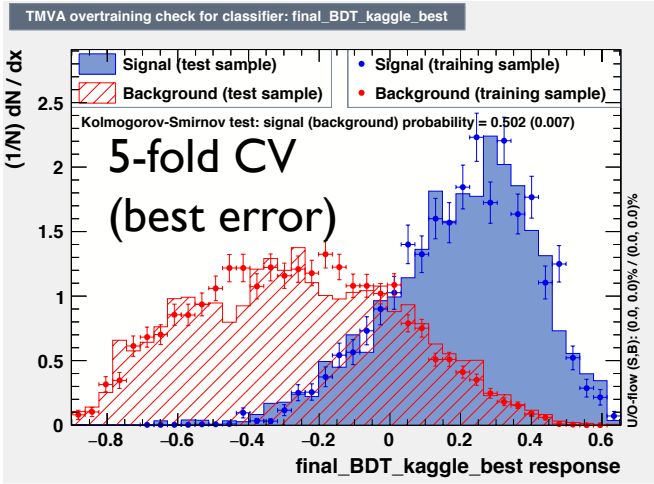
<u>limiting case:</u>
k=N(data): gives the leave-one-out cross validation method.



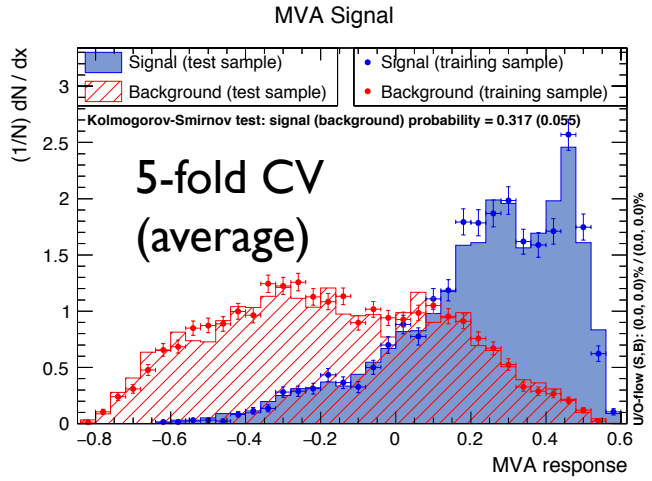
Geisser, S. (1975). The predictive sample reuse method with applications. J. Amer. Statist. Assoc., 70:320–328.

BDT training





k-fold cross validated BDT response has a better level of agreement between the training and validation samples than the hold out solution.



See backup slides for notes on configuration of TMVA example.

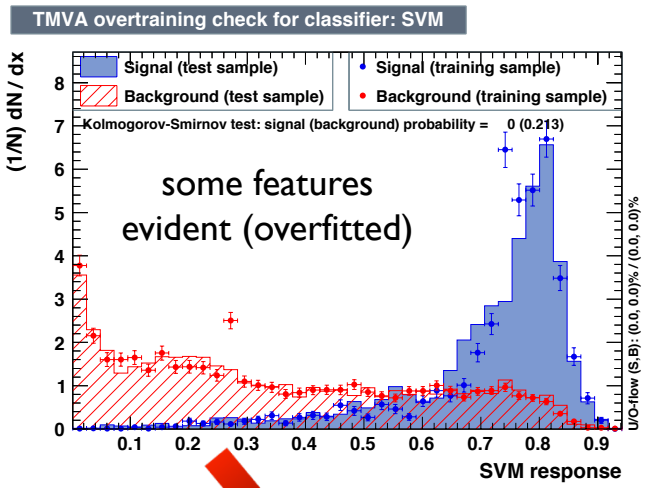
Push request to ROOT git repository has been made.

Work on more elegant solution for TMVA in progress.

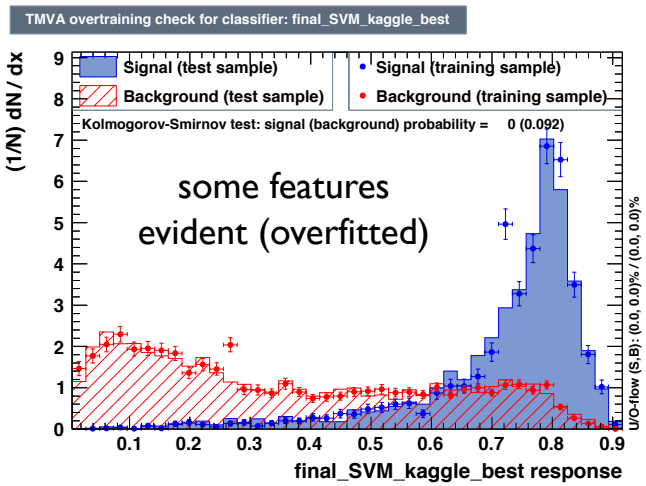
SVM training

This example uses an RBF Kernel function optimised using the ROC integral as the FOM

Hold-out

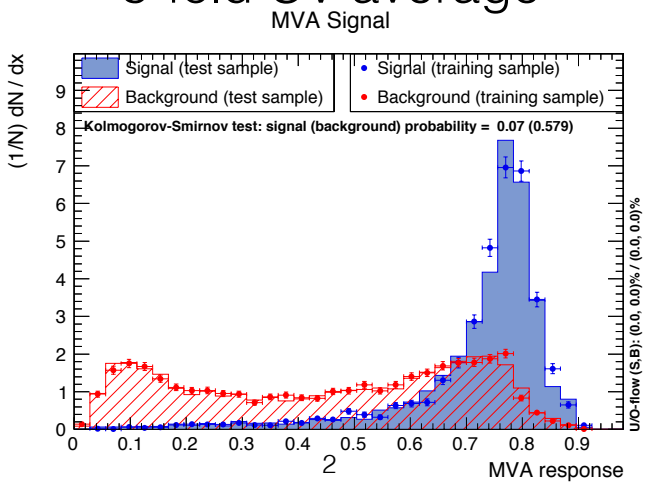


5-fold CV best



k-fold cross validated SVM response has a better level of agreement between the training and validation samples than the hold out solution.

5-fold CV average



See backup slides for notes on configuration of TMVA example.

Push request to ROOT git repository has been made.

Work on more elegant solution for TMVA in progress.

Summary: Generalisation

- Optimal MVA solution is not necessarily generalised.
- Cross validation helps improve generalisation.
- Does not guarantee a general solution – that requires a further step;
  - The quantity used by TMVA for the overtraining check is not the right quantity to use for this purpose.
  - We are looking into this issue in more detail within ATLAS.
  - If you can't wait for a solution – take a look at the following papers on the topic:

F. Porter: arXiv:0804.0380.
S. Bityukov: arXiv:1302.2651.
S.-H. Cha: , Int. J. Math. Models and Meth. in Applied Sciences, Issue 4, Volume 1, 2007, 300-307.

Overall summary

- Tools are available for using SVMs with HEP data.
  - Some are interfaced to ROOT (e.g. TMVA, RTMVA)
  - TMVA implementation has some necessary updates that going into ROOT to expand functionality and improve user friendliness.
- SVMs can perform well with HEP data problems.
  - You might wish to explore what they can do for your analysis.
- Generalisation is an issue that has largely been overlooked by HEP.
  - Should split data into test/training/validation samples and perform cross validation appropriately.
  - Cross validation example going into TMVA now; more elegant solution is under development.
- We can (and should) learn a lot from the machine learning (and other science) discipline(s) about the use of ML algorithms.

Backup

References

- Some of our recent (public) talks:
  - TJS recently presented our work at ACAT.
  - Recent updates at the LHC IML by TJS (Oct), TJS (Dec), TJS (Feb).

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

Contributors to the SVM algorithm in TMVA

- Original implementation by:
  - Marcin Wolter @ IFJ PAN, Krakow, Poland
  - Andrzej Zemla @ IFJ PAN, Krakow, Poland
- Regression added by:
  - Krzysztof Danielowski @ IFJ PAN & AGH, Krakow, Poland
  - Kamil Kraszewski @ IFJ PAN & UJ, Krakow, Poland
  - Maciej Kruk @ IFJ PAN & AGH, Krakow, Poland
- Latest updates (bug fix, optimisation and user friendly tweaks, additional KFs):
  - Adrian Bevan @ Queen Mary University of London
  - Tom Stevenson @ Queen Mary University of London

Sequential Minimal Optimisation (SMO)

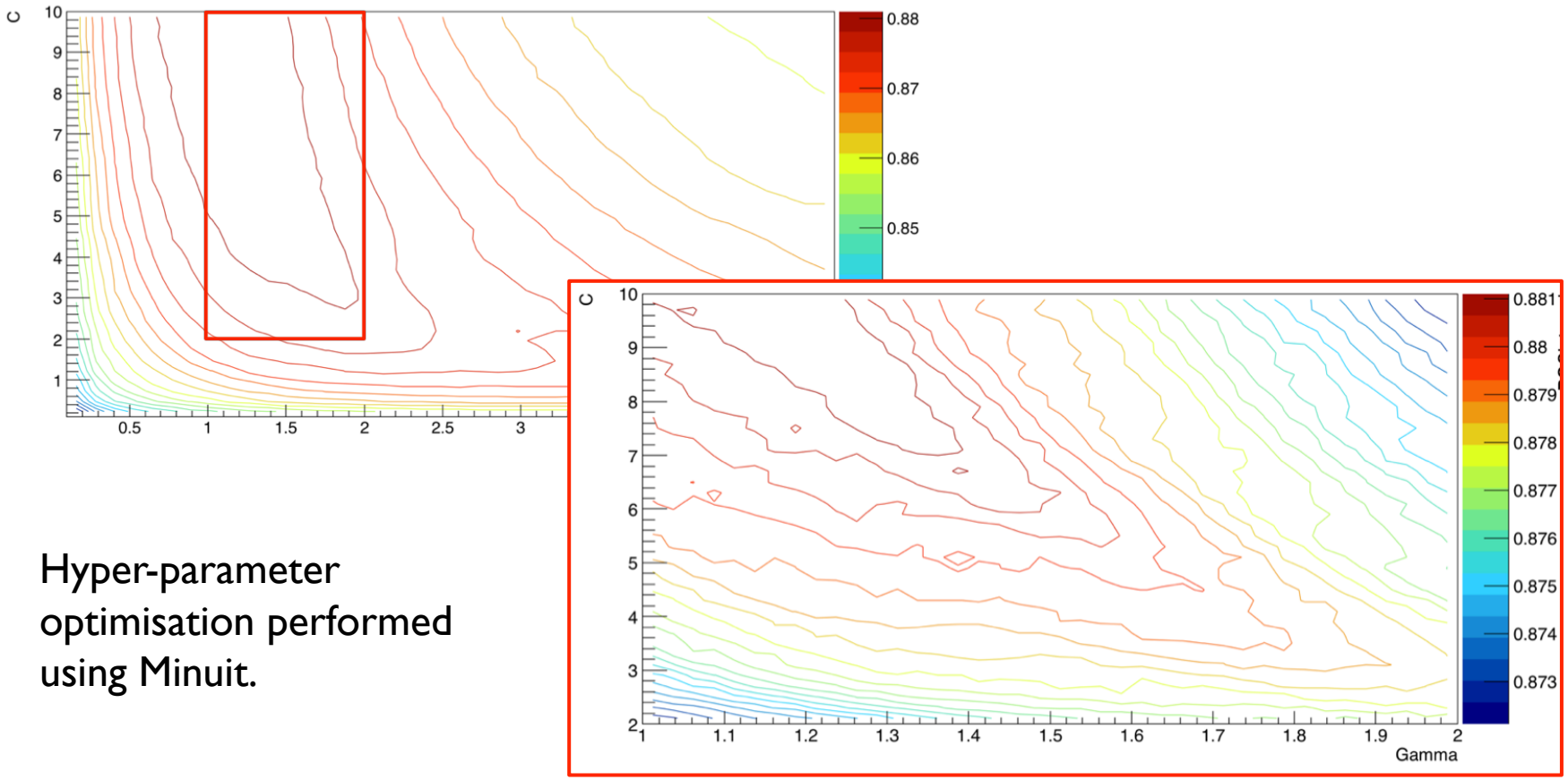
- The dual form of the Lagrangian minimised for SVMs depends on Lagrange multipliers (α_i) that satisfy a constraint equation $\sum_{i=1}^{n} \alpha_i y_i = 0$ as opposed to the weight and bias parameters.

- 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.

SVM optimisation

- Able to extract contours of Γ vs C for parameter optimisation for an RBF KF to view correlations.



Hyper-parameter optimisation performed using Minuit.

The Sigmoid KF

- This is conspicuous by its absence...
- The Sigmoid KF is not a valid Mercer kernel.
  - i.e. the Gramm matrix is not guaranteed to be positive semi-definite; hence the geometric interpretation of the SVM can fail with this KF.
  - A number of papers have reported sensible results with this KF.
  - The usage appears to be a hang over from the transition from using Neural Networks to SVMs (given that both originate from Rosenblatt's perceptron, and the Sigmoid function is a widely used activation function for NNs).
- We have consciously chosen not to implement this as a KF for TMVA.

k-fold CV in TMVA

- An example macro exists in TMVA: **`crossvalidate.cc`**
  - Push request was made a few weeks ago.
  - Lorenzo is working on this and it should appear in the git repository soon*.
- Macro is general and configuration follows TMVA syntax.

```
const int numFolds = 4;
const TString inputSigFile = "chess_sig.root";
const TString inputBkgFile = "chess_bkg.root";
const TString inputSigTree = "sig";
const TString inputBkgTree = "bkg";
const std::vector<TString> treeVars = {"x","y"};
const std::vector<TString> trainVars = {"x","y"};
const Float_t split[] = {0.1,0.4,0.5};
const Double_t sigWeight = 1.0;
const Double_t bkgWeight = 1.0;
const TCut sigCuts ="";
const TCut bkgCuts = sigCuts;
const TMVA::Types::EMVA MVA = TMVA::Types::kBDT;
const TString MVAname = "BDT_chess";
const TString baseOptionString = "BoostType=AdaBoost:SeparationType=GiniIndex:PruneMethod=NoPruning";
const Double_t SigEff = 0.8;
const Double_t MVAmin = -1.0;
const TString eventWeight = "";
```

Example is being pushed with a checker board data sample.

*Along with improvements to TMVA's SVM implementation

k-fold CV in TMVA

- CV steps:
  - Split data into training/validation and test samples for k-folds.
  - Optimise the hyper parameters on k-1 folds; validate with k^{th} fold.
    - Repeat for all k.
  - Compute error rate for samples for each permutation.
  - Combine training and validation samples for final trainings.
    - Evaluate performance for training with "best" error.

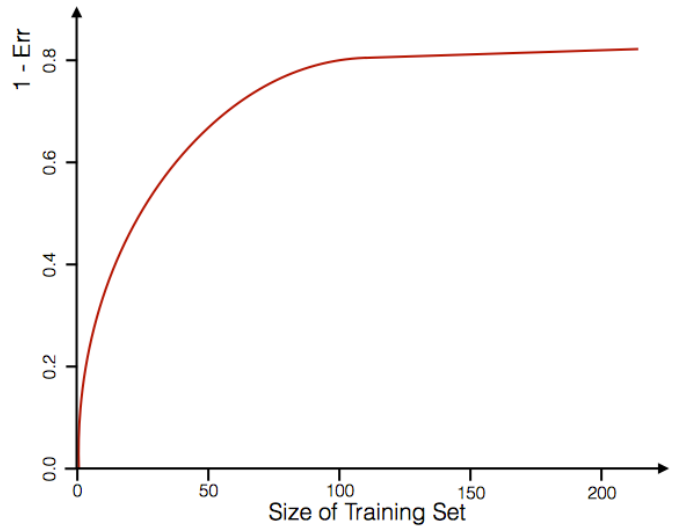
      Tendency to be over-trained
    - Evaluate performance for the k-fold MVAs on each event, weighting the contributions by (1-ε).

      Expect more generalised result than best error or hold-out method.

What value of k should be used?

- This question is problem dependent.
  - The answer is problem dependent!

- Determine E(k); from this distribution search for the asymptotic limit that is sufficiently good to minimise the error:



• A larger value of k will mean a given training will have an error rate similar to the asymptotic limit of $k=N_{data}$.

• As k increases the computational expense increases.

• Common values of k are 5-10; **but you should really determine this for your given problem.**

CV in R (so via the new interface)

- R has cross validation implemented in it; e.g. libsvm uses this by default (library e1071).

  https://www.csie.ntu.edu.tw/~cjlin/libsvm/

  [others are also available]

- The number of folds used for CV is set via tune control; default in R 3.2.3 is 10-fold CV.

- e.g. using an SVM with the cats data sample in R (a sample of male and female cats studied by R. A. Fisher):

  http://pprc.qmul.ac.uk/~bevan/statistics.html

Leave one out CV

- In the extreme limit of k-fold CV that $k \rightarrow N(data)$ we obtain the leave one out CV method.
- Requires N(data) trainings of an MVA.
- Average the result obtained from the N(data) MVAs to determine the output.
- Can provide useful results for small samples of data where training and validation examples are scarce.
- However, can be computationally expensive for large data samples.
- See: Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. J. Roy. Statist. Soc. Ser. B, 36:111–147. With discussion and a reply by the authors; Allen, D. M. (1974). The relationship between variable selection and data augmentation and a method for prediction. Technometrics, 16:125–127; Geisser, S. (1975). The predictive sample reuse method with applications. J. Amer. Statist. Assoc., 70:320–328.
- This can be extended to the leave p-out CV method, where one successively omits p examples from a training and cycles through the possible permutations.

Shao, J. (1993). Linear model selection by cross-validation. J. Amer. Statist. Assoc., **88** (422):486–494.