Introduction to Multivariate Methods

Classification and Function Approximation

Harrison B. Prosper

Florida State University

Bari Lectures

30, 31 May, 1 June 2016

Outline

- Lecture 1
  - Introduction
  - Classification
  - Grid Searches
  - Decision Trees
- Lecture 2
  - Boosted Decision Trees
- Lecture 3
  - Neural Networks
  - Bayesian Neural Networks

MVA Twiki

https://support.ba.infn.it/redmine/projects/statistics/wiki/MVA

Examples

Today, we shall apply neural networks to two problems:

- Wine Tasting
- Approximating a 19-parameter function.

Example: Wine Tasting

Recall that our goal is to classify wines as good or bad using the variables highlighted in blue, below:

variables	description
acetic	acetic acid
citric	citric acid
sugar	residual sugar
salt	NaCl
SO2free	free sulfur dioxide
SO2tota	total sulfur dioxide
pH	pH
sulfate	potassium sulfate
alcohol	alcohol content
quality	(between 0 and 1)

http://www.vinhoverde.pt/en/history-of-vinho-verde

Neural Networks
aka
Multi-Layer Perceptrons

A Bit of History: Hilbert's 13th Problem

(One version of Problem 13): Prove

that it is *possible* to do the following:

$$f(x_1,\ldots,x_n) = F(\, g_1(x_{(1)},\ldots,\, x_{(m)}),\ldots,\, g_k(x_{(1)},\ldots,\, x_{(m)}))$$

$m < n$ for all n

In 1957, Kolmogorov proved that it was possible with $m = 3$.

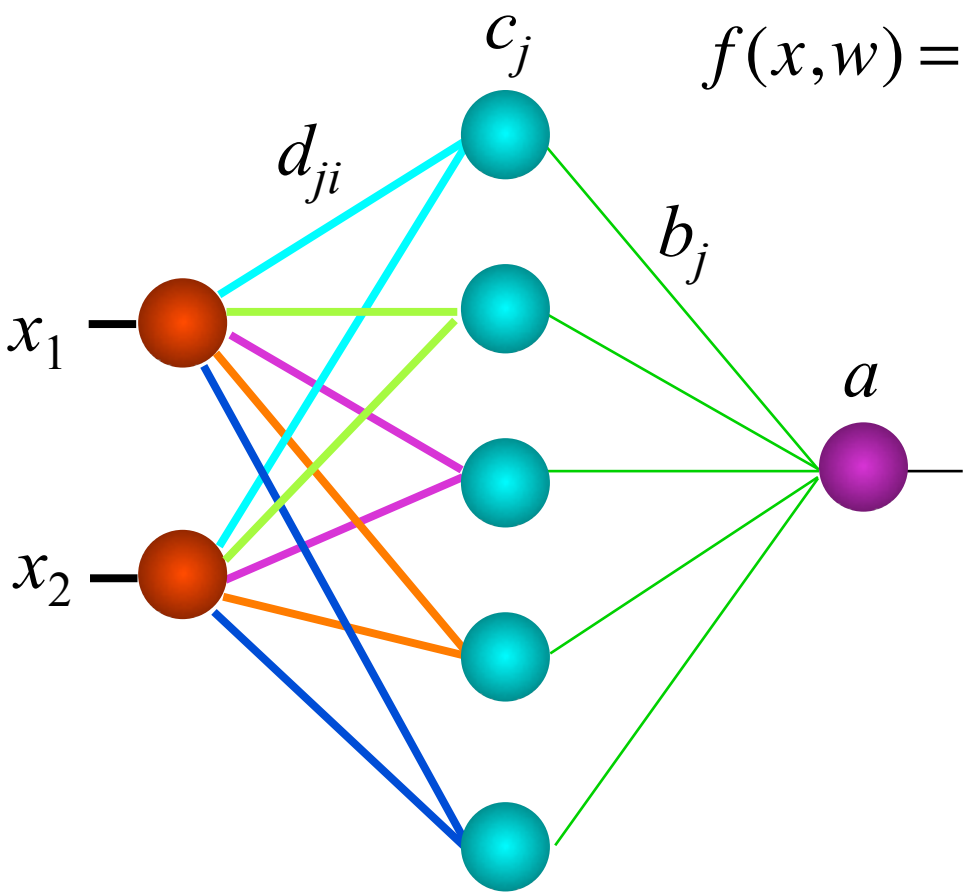
Today, we know that functions of the form

$$f(x_1,\cdots,x_I) = a + \sum_{j=1}^{H} b_j \tanh\left[c_j + \sum_{i=1}^{I} d_{ji} x_i \right]$$

can provide arbitrarily accurate approximations of real functions of I real variables.

(Hornik, Stinchcombe, and White, *Neural Networks* **2**, 359-366 (1989))

Feed-Forward Neural Networks



$$f(x,w) = a + \sum_{j=1}^{H} b_j \tanh\left[c_j + \sum_{i=1}^{I} d_{ji} x_i \right]$$

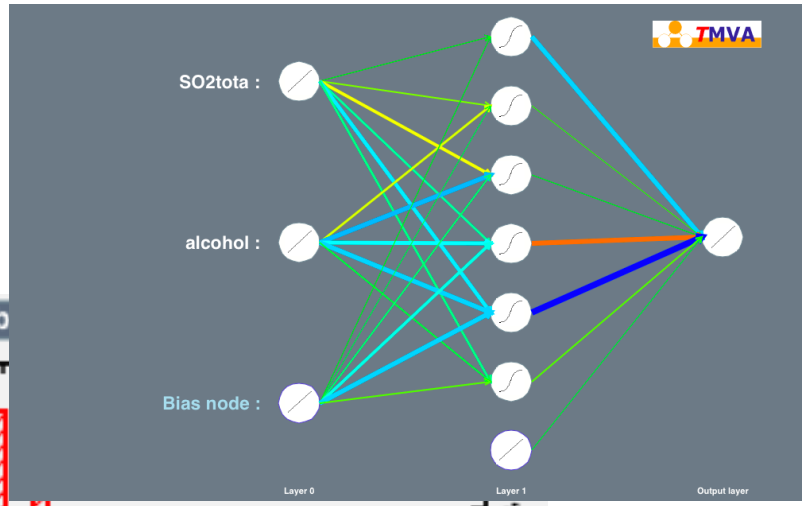
$$n(x,w) = \frac{1}{1+\exp[-f(x,w)]}$$

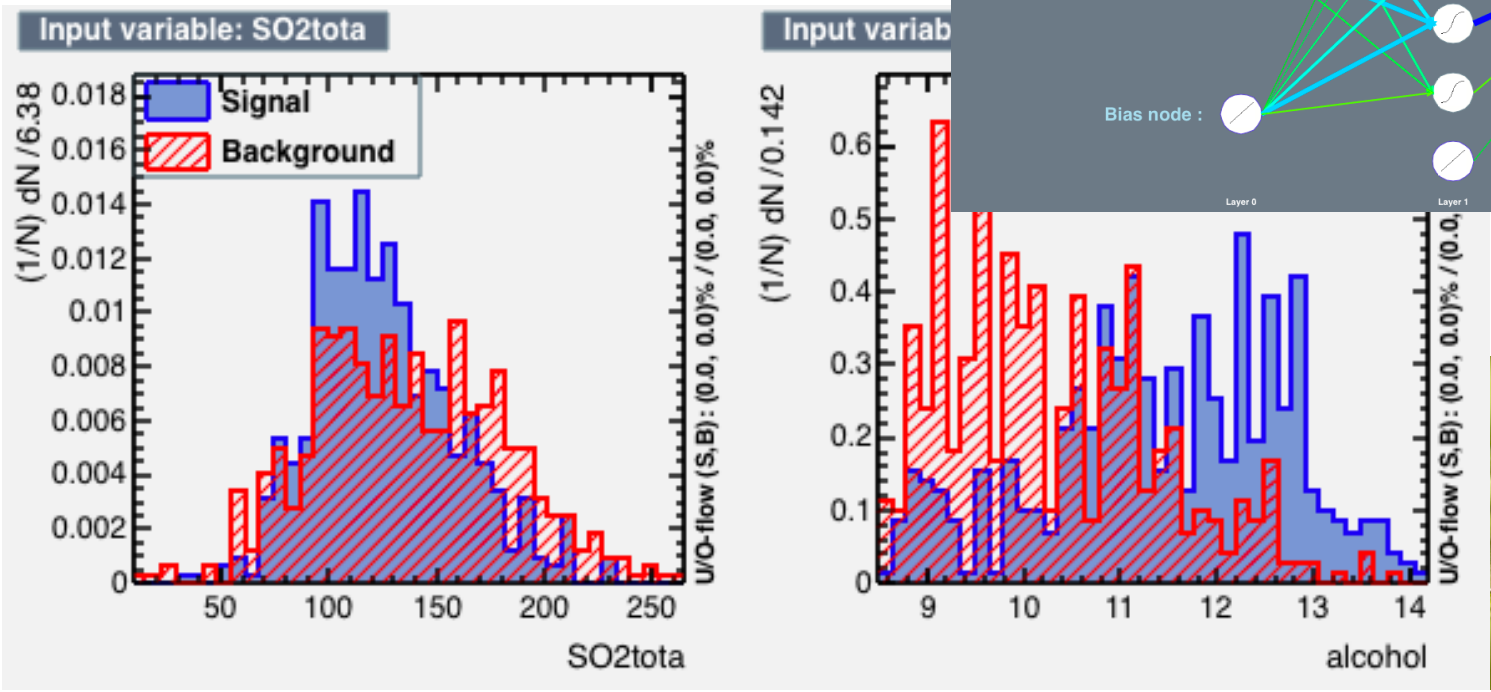
f is used for regression
n is used for classification
w = a, b, c, d are free parameters to be found by the training.

Example: Wine Tasting

We consider the same two variables SO2tota and alcohol and the NN function shown here:

We use data for 500 good wines and 500 "bad" wines.





Example: Wine Tasting

To construct the NN, we minimize the empirical risk function

$$R(w) = \frac{1}{2}\sum_{i=1}^{1000}\left[y_i - n(x_i, w)\right]^2$$

where

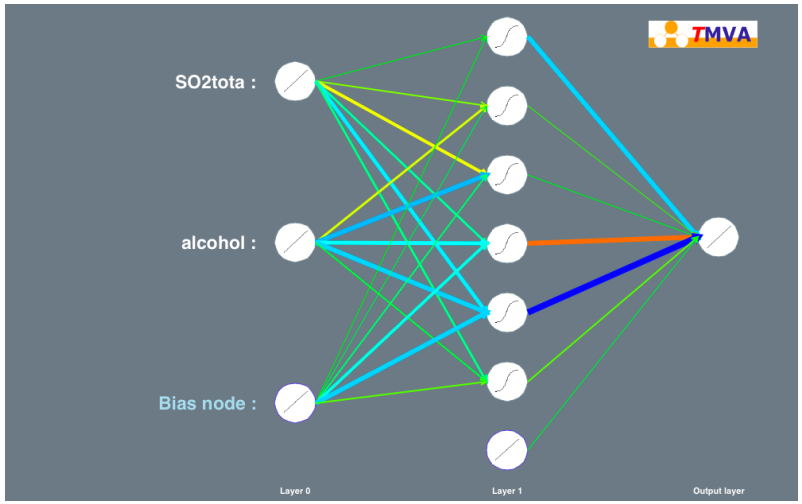
$y = 1$ for good wines and

$y = 0$ for bad wines,

which yields an approximation to

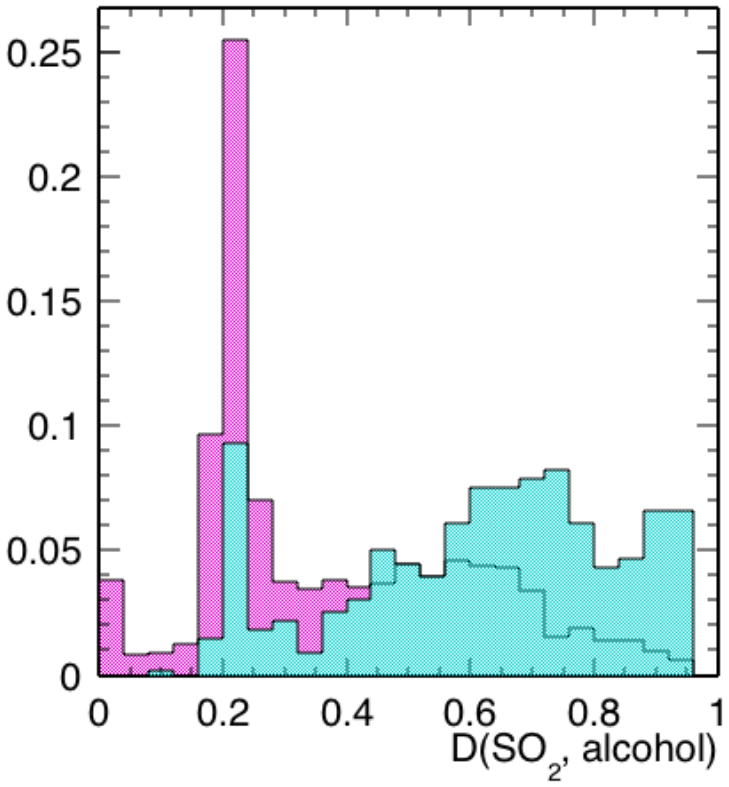
$$D(x) = \frac{p(x \mid good)}{p(x \mid good) + p(x \mid bad)}$$

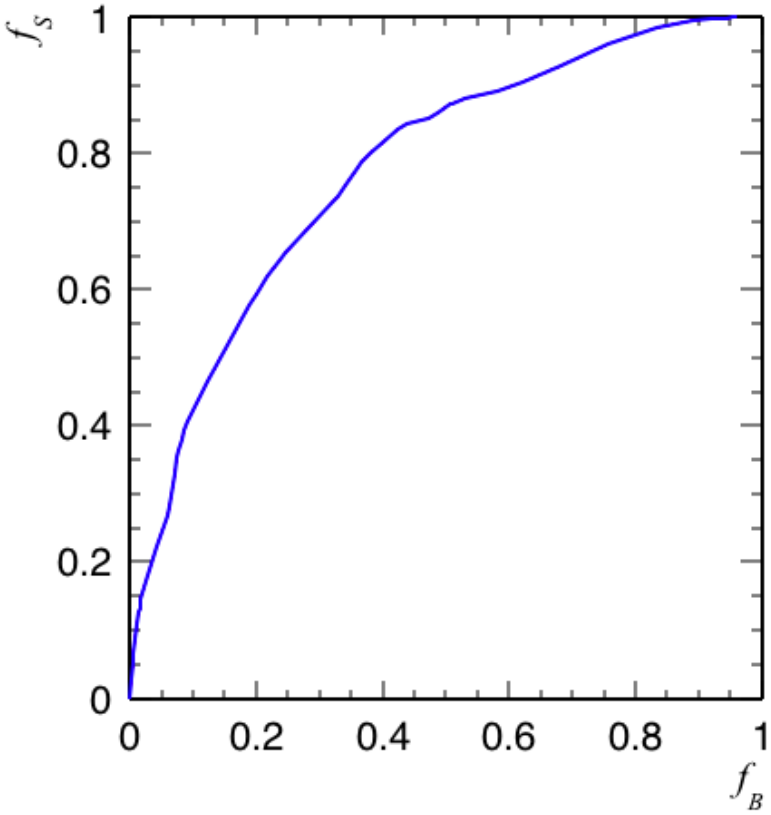
$x = (\text{SO2tota}, \text{ alcohol})$

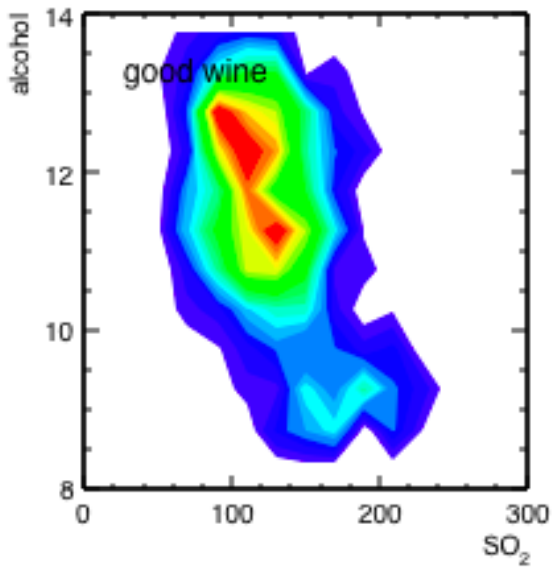


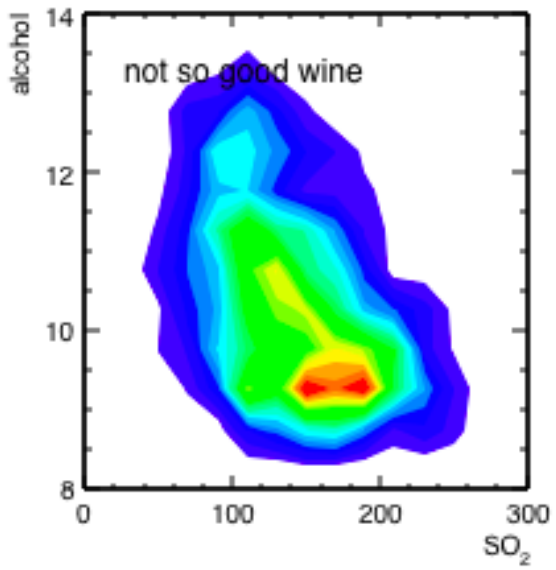
Results (NN)

The distribution of $D(x)$ and the Receiver Operating Characteristic (ROC) curve. In this example, the ROC curve shows the fraction f of each class of wine *accepted*.

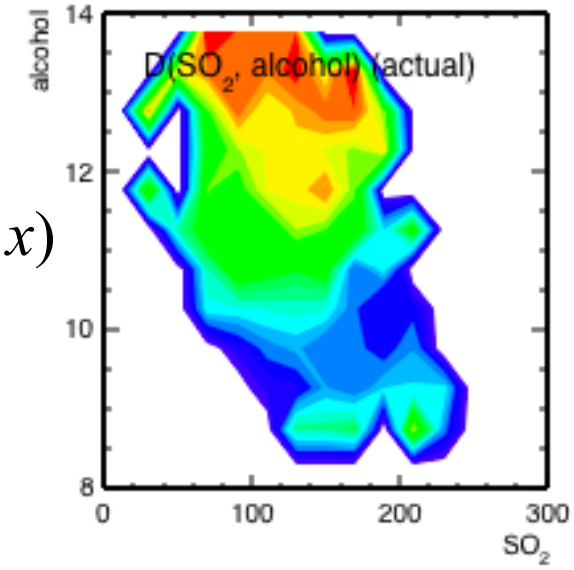


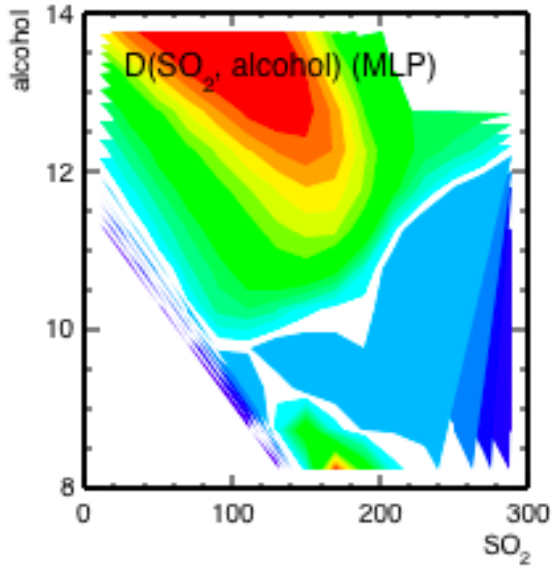






Actual
$p(good \mid x)$





Best fit
NN

Bayesian Neural Networks

Bayesian Learning

Choose

Function space $\boldsymbol{F} = \{ f(x, \boldsymbol{w}) \}$

Likelihood $p(\boldsymbol{T} \mid \boldsymbol{w})$, training set $\boldsymbol{T} = \{(\boldsymbol{y}, \boldsymbol{x})\}$

Loss function $\boldsymbol{L}$

Prior $p(\boldsymbol{w})$

Method

Use Bayes' theorem to assign a probability (density)

$$p(\boldsymbol{w} \mid \boldsymbol{T}) = p(\boldsymbol{T} \mid \boldsymbol{w})\, p(\boldsymbol{w}) / p(\boldsymbol{T})$$
$$= p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w})\, p(\boldsymbol{x} \mid \boldsymbol{w})\, p(\boldsymbol{w}) / p(\boldsymbol{y} \mid \boldsymbol{x})\, p(\boldsymbol{x})$$
$$\sim p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w})\, p(\boldsymbol{w}) \qquad \text{(assuming } p(\boldsymbol{x} \mid \boldsymbol{w}) = p(\boldsymbol{x})\text{)}$$

to *every* function in the function space, where $p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w})$ is the product of the likelihoods $p(y_i \mid x_i, \boldsymbol{w})$ for each (y_i, x_i).

Bayesian Learning

Given, the posterior density $p(\boldsymbol{w} \mid T)$, and some *new* data x one can compute the predictive distribution of y

$$p(y \mid x, T) = \int p(y \mid x, w)\, p(w \mid T)\, dw$$

If a definite estimate of y is needed for every x, this can be obtained by minimizing a suitable risk function,

$$R[f_w] = \int L(y, f)\, p(y \mid x, T)\, dy$$

Using, for example, $L = (y - f)^2$ yields the following estimate

$$f(x) \simeq \bar{y}(x, T) \equiv \int y\, p(y \mid x, T)\, dy$$

In general, a different L, will yield a different estimate of $f(x)$.

Bayesian Learning

For function approximation, the likelihood $p(y \mid x, w)$ for a given (y, x) is typically chosen to be Gaussian

$$p(y \mid x, w) = \exp\left[-\frac{1}{2}\left[y_i - f(x_i, w)\right]^2 / \sigma^2\right]$$

For classification one uses

$$p(y \mid x, w) = [n(x, w)]^y [1 - n(x, w)]^{1-y}, \quad y = 0, \text{ or } 1$$

Setting $y = 1$, in $p(y \mid x, T) = \int p(y \mid x, w)\, p(w \mid T) dw$ gives

$$p(1 \mid x, T) = \int n(x, w)\, p(w \mid T)\, dw$$

which is an estimate of the probability that x belongs to the class for which $y = 1$.

1-D Example: H to ZZ to 4Leptons

Dots

$p(s \mid x) \sim H_s / (H_s+H_b)$

H_s, H_b, are 1-D histograms of the transverse momentum of the Higgs boson.

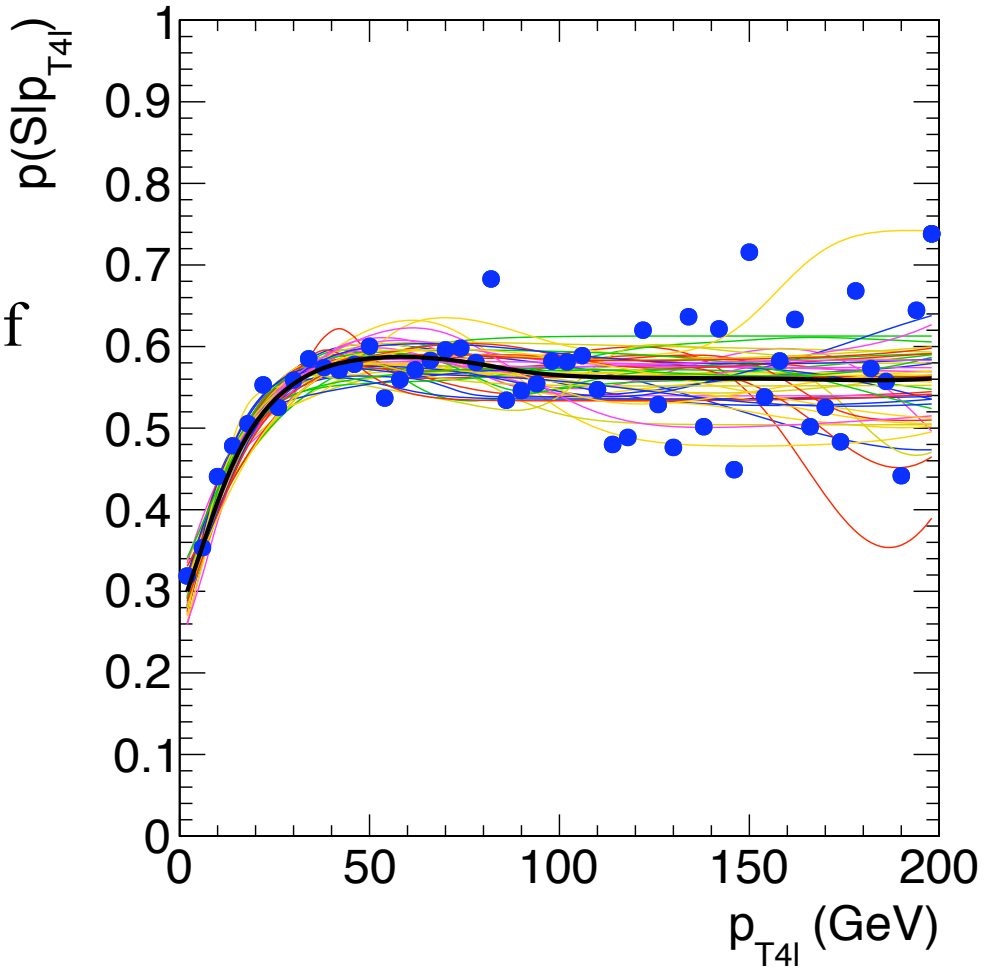
Curves

Individual NNs

$n(x, w_k)$

Black curve

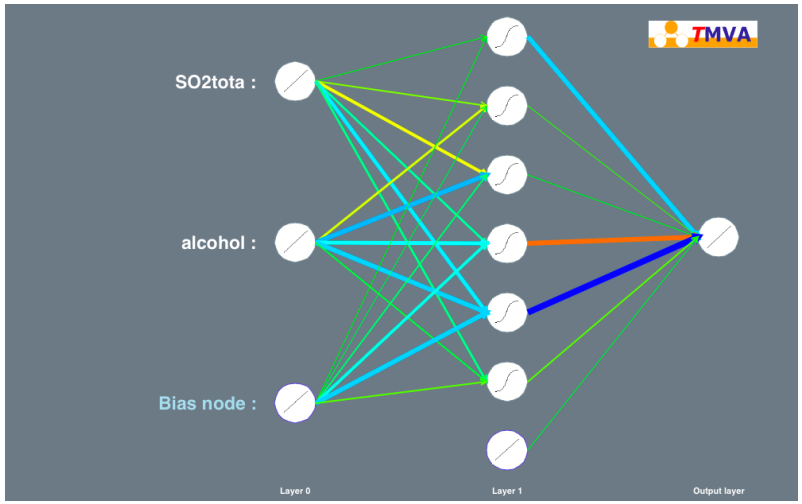
$n(x) = < n(x, w) >$



Example: Wine Tasting

To construct the BNN, we *sample* points w from the posterior density

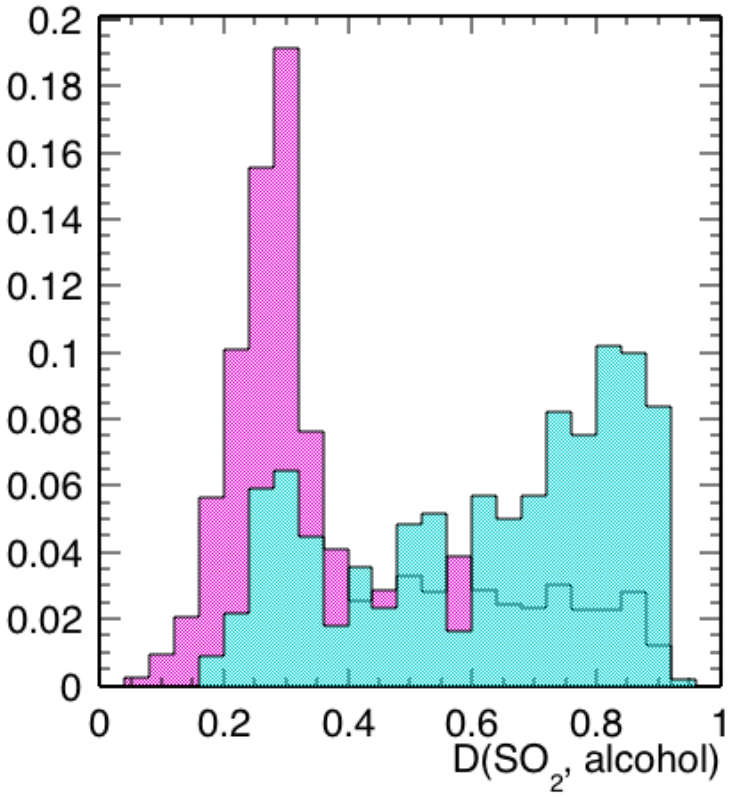
$$p(w \mid T) = \frac{p(T \mid w)p(w)}{p(T)}$$

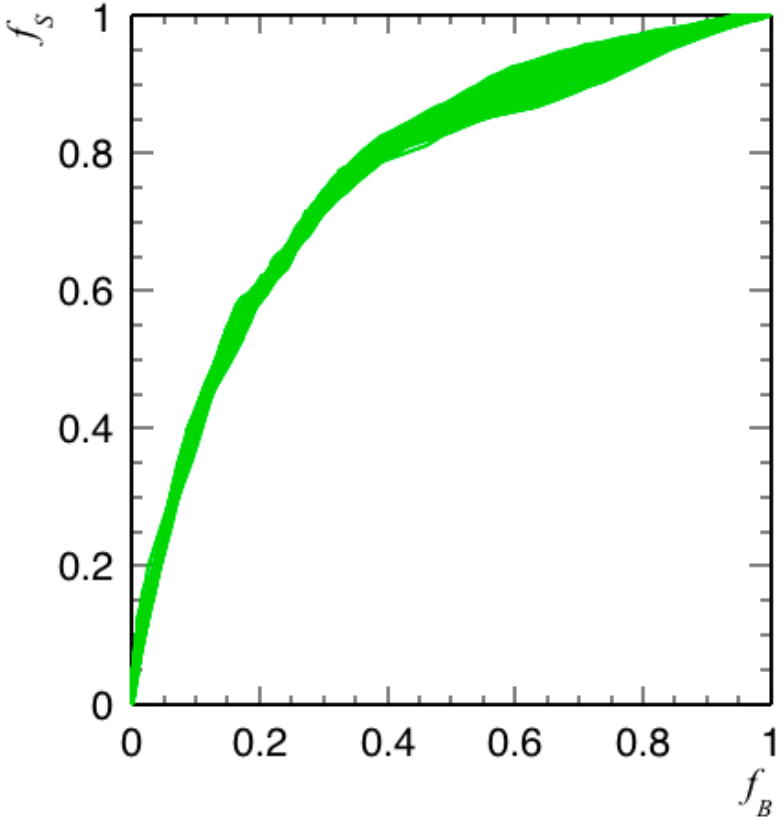


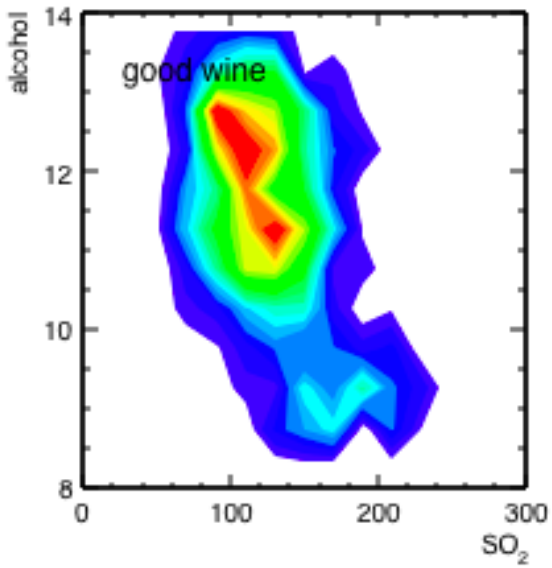
using a Markov Chain Monte Carlo method. This generates an *ensemble* of neural networks, just as in the 1-D example on the previous slide.

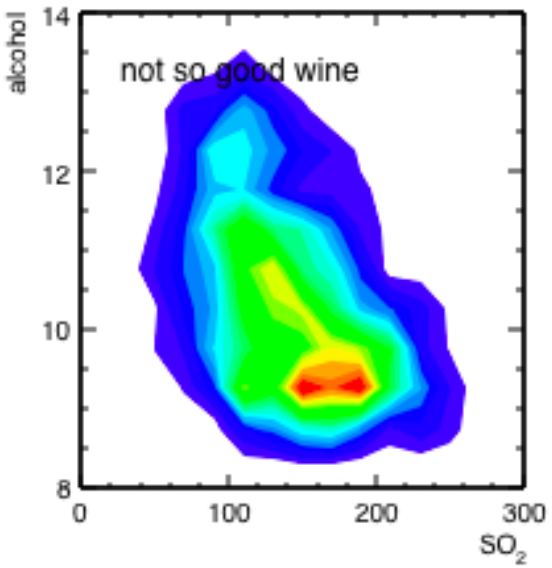
Results (BNN)

The distribution of $D(x)$ (using the *average* of 100 NNs) and the *ensemble* of ROC curves, one for each NN.

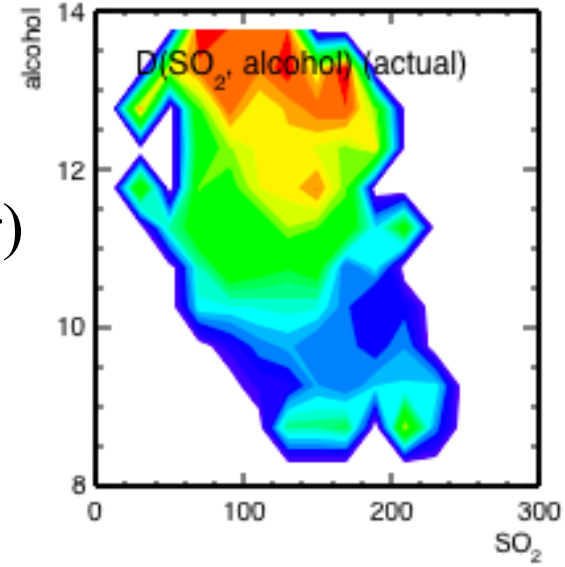


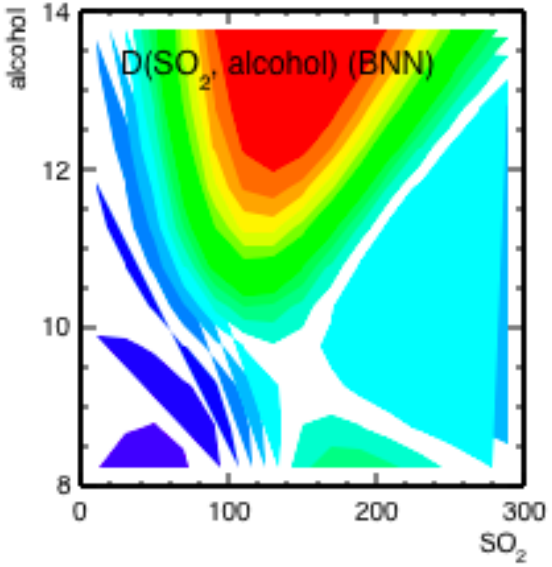






Actual
$p(good \mid x)$



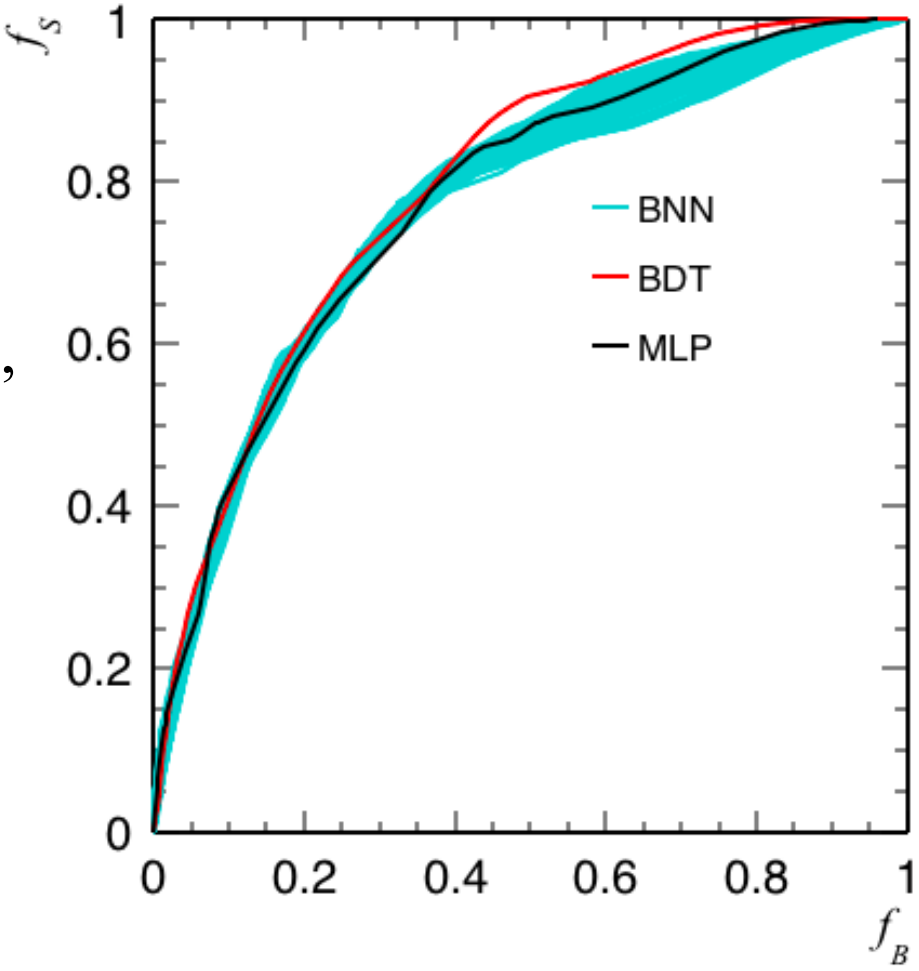


<NN>

Comparing ROCs!

We see that if good and bad wines are equally likely and we are prepared to accept 40% of bad wines, while accepting 80% of good ones, then the BDT does slightly better.

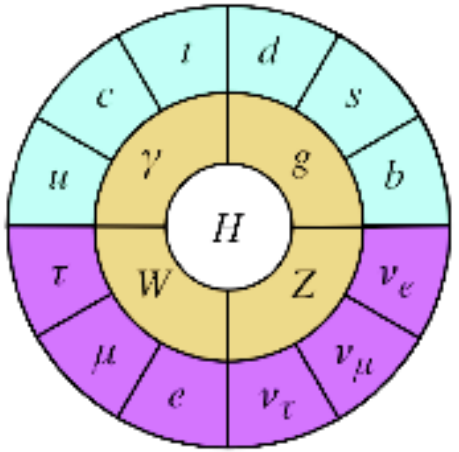
But, if we lower the acceptance rate of bad wines, all three methods work equally well.



Function Approximation

Function Approximation

Now that the Higgs boson has been found, the primary goal of researchers at the LHC is to look for, and we hope, discover new physics.



There are two basic strategies:

1. Look for any deviation from the predictions of our current theory of particles (the Standard Model).

2. Look for the specific deviations predicted by theories of new physics, such as those based on a proposed symmetry of Nature called supersymmetry.

Function Approximation

While finishing a paper related to his PhD dissertation, my student Sam Bein had to approximate a function of 19 parameters!

Targeting the Minimal Supersymmetric Standard Model with the Compact Muon Solenoid Experiment

by

Samuel Louis Bein

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Physics

at the

FLORIDA STATE UNIVERSITY

June 2016

Function Approximation

This is what I suggested he try.

1. Write the function to be approximated as

$$f(\theta) = C(\theta)\prod_{i=1}^{19} g_i(\theta_i)$$

2. Use NNs to approximate each $g_i(\theta_i)$ by setting $y(\theta) = g$.

3. Use another neural network to approximate the ratio

$$D(\theta) = \frac{p(\theta)}{p(\theta) + \prod_{i=1}^{19} g_i(\theta_i)}$$

where the "background" is sampled from the g functions and the "signal" is the original distribution of points.

Function Approximation

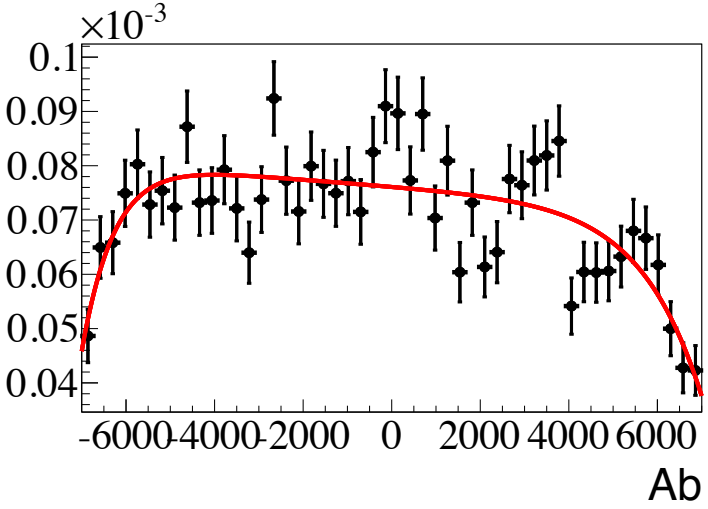
4. Finally, approximate the function f using

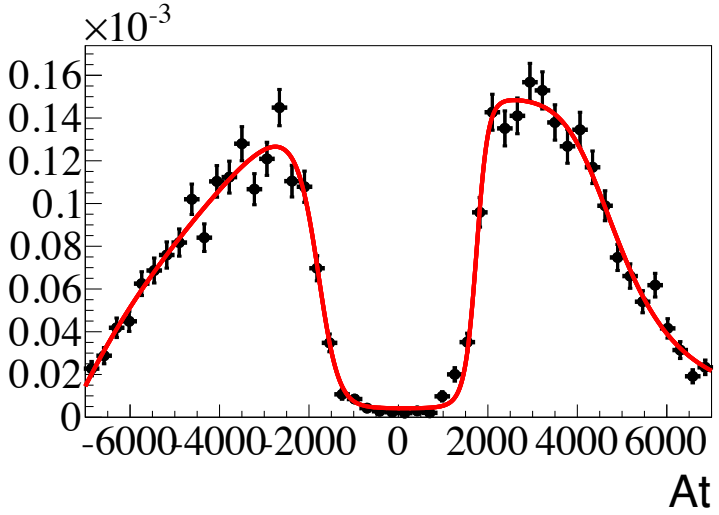
$$f(\theta) = \frac{D(\theta)}{1 - D(\theta)} \prod_{i=1}^{19} g_i(\theta_i)$$

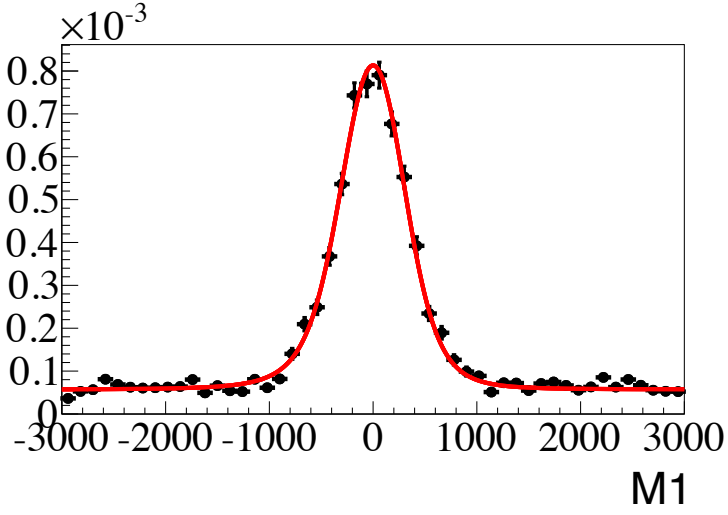
The intuition here is that much of the dependence of f on θ is captured by the product function, leaving (we hoped!) a somewhat gentler 19-dimensional function to be approximated.

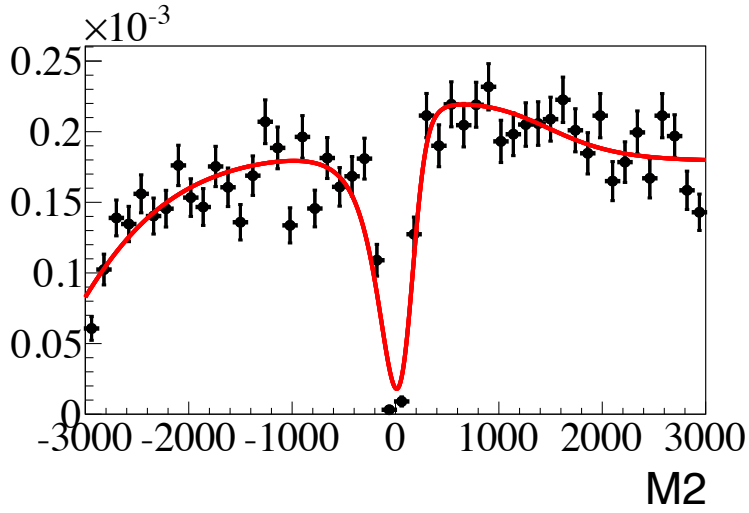
The next slide shows the neural network approximations of a few of the g functions. Of course, 1-D functions can be approximated using a wide variety of methods. The point here is to show the flexibility of a *single* class of NNs.

$g_i(\theta_i)$


×10^-3
0.1
0.09
0.08
0.07
0.06
0.05
0.04
-6000 -4000 -2000 0 2000 4000 6000
Ab


×10^-3
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-6000 -4000 -2000 0 2000 4000 6000
At


×10^-3
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-3000 -2000 -1000 0 1000 2000 3000
M1


×10^-3
0.25
0.2
0.15
0.1
0.05
0
-3000 -2000 -1000 0 1000 2000 3000
M2

A Final Cool Thing!

Summary

- Multivariate methods can be applied to many aspects of data analysis, including classification and function approximation. Even though there are hundreds of methods, almost all are *numerical approximations* of the *same* mathematical quantity.

- We considered random grid searches, decision trees, boosted decision trees, neural networks, and the Bayesian variety.
- Since *no* method is best for *all* problems, it is good practice to try a few of them, say BDTs and NNs, and check that they give approximately the same results.