PH3010 / Advanced Skills

Introduction to Machine Learning

Glen D. Cowan
RHUL Physics

Outline

Part 1:

What Machine Learning is and how it can be applied

Classification of two types of “events”

Linear classifiers: Fisher Discriminant

Part 2:

Nonlinear classifiers: Neural Networks

Software for Machine Learning: scikit-learn

Exercises

Part 3 (extension):

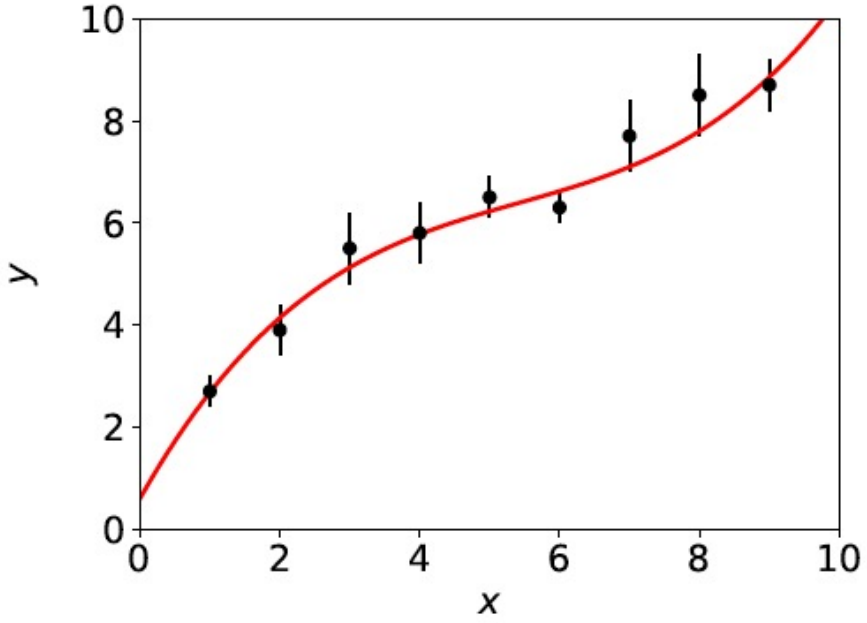
Multiple regression

What Machine Learning is

The term Machine Learning (ML) refers to algorithms that "learn from data" and make predictions based on what has been learned.

In its simplest sense, "learning" means the algorithm contains adjustable parameters whose values are estimated using data.

Formally, curve fitting can be seen as Machine Learning.



Hypothesized curve:

$$f(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

Values of parameters are "learned" from the data.

Fitted curve can make predictions at x values of future measurements.

More general Machine Learning

But generally ML refers to situations in which:

the hypothesized model is very general (e.g., not just a polynomial) and contains many adjustable parameters;

the quantity we want to predict could depend on many variables, e.g., not just x but a vector $\boldsymbol{x} = (x_1,\ldots,x_n)$.

ML can be seen as a part of or related to:

Artificial Intelligence

Pattern Recognition

Statistical Learning

Multivariate Analysis

Development from (mainly) Computer Science, (also) Statistics; sometimes "Data Science" used to refer to all of above.

Curve fitting ⟶ regression

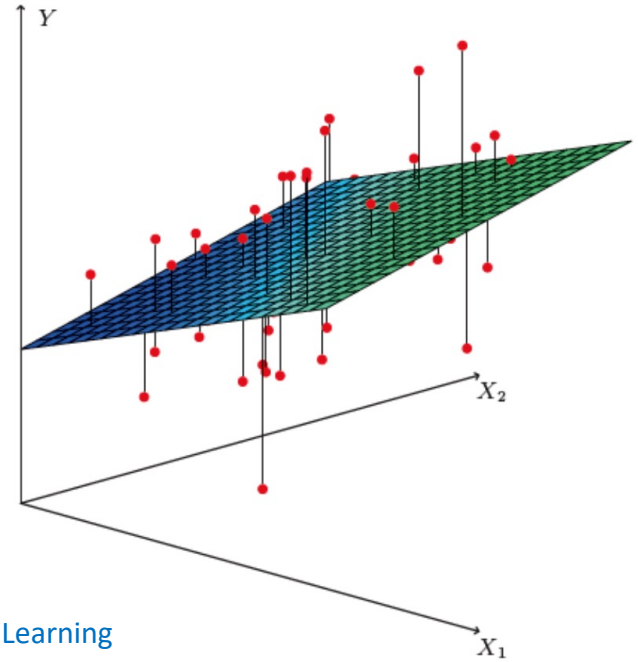
The generalisation of curve fitting with a multidimensional control variable $x \longrightarrow \boldsymbol{x} = (x_1, \ldots, x_n)$ is called multiple regression.

The data consist of sets of points $(\boldsymbol{x_i}, y_i)$, where now $\boldsymbol{x_i}$ is a multidimensional vector, and usually the y_i does not come with an error bar.

Goal is to predict the expected y value for a new $\boldsymbol{x}$.

https://web.stanford.edu/~hastie/ISLR2/ISLRv2_website.pdf

E.g. in two dimensions, multiple regression means fitting a surface $f(x_1, x_2)$ to data points $(x_{1,i}, x_{2,i}, y_i)$:

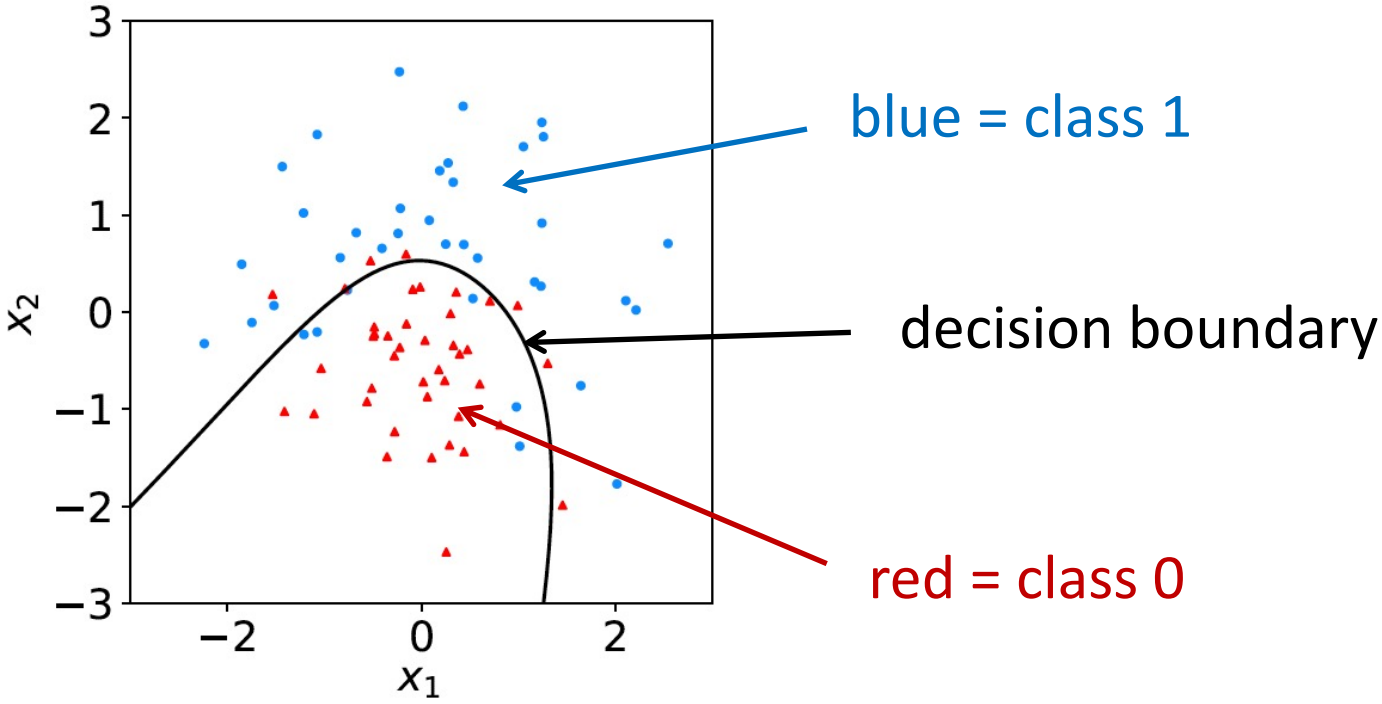


Classification

A related type of ML algorithm is called classification: the data consist of points $(\boldsymbol{x}_i, y_i)$, where now y_i is discrete class label, (e.g., 0 or 1, red or blue, etc.).

Learning based on events with known y_i = supervised learning.

Goal is to create a decision boundary in $\boldsymbol{x}$-space so as to predict the class y of a new instance of $\boldsymbol{x}$.



Example of classification: Industrial Fishing

You scoop up fish which are of two types:

Sea Bass

Cod

You examine the fish with automatic sensors and for each one you measure a set of features:

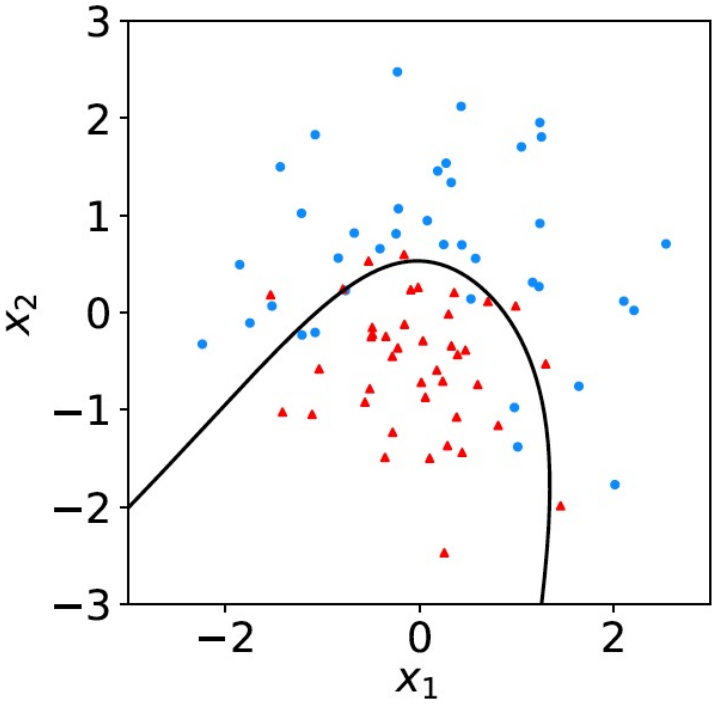
x_1 = length
x_2 = width
x_3 = weight
x_4 = area of fins
x_5 = mean spectral reflectance
x_6 = ...

These constitute the "feature vector" $\boldsymbol{x} = (x_1, \ldots, x_n)$.

In addition you hire a fish expert to identify the "true class label" $y = 0$ or 1 (i.e., 0 =sea bass, 1 = cod) for each fish.

Distributions of the features

If we consider only two features $\boldsymbol{x} = (x_1, x_2)$, we can display the results in a scatter plot (red: $y = 0$, blue: $y = 1$).



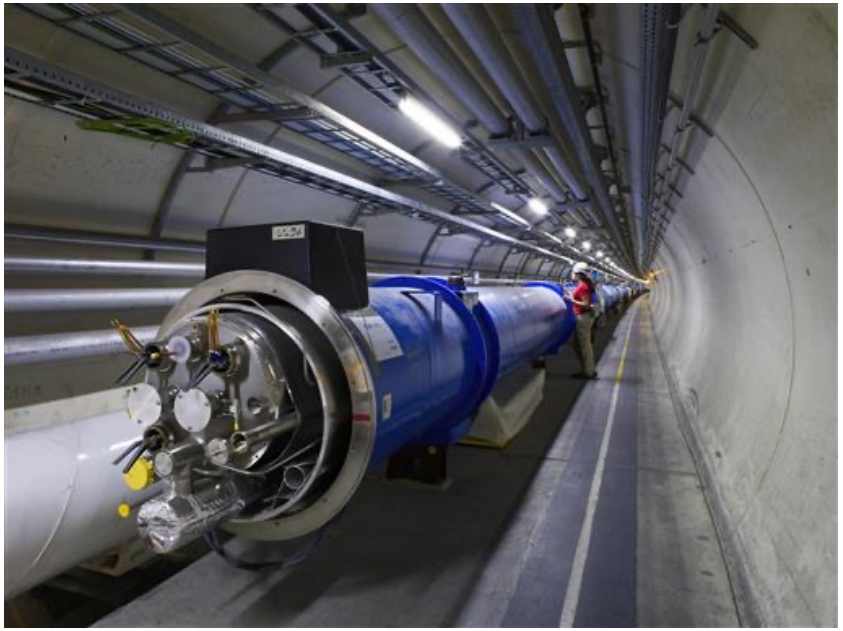
Goal is to determine a decision boundary, so that, without the help of the fish expert, we can classify new fish by seeing where their measured features lie relative to the boundary.

Same idea in multi-dimensional feature space, but cannot represent as 2-D plot. Decision boundary is n-dim. hypersurface.

Example use of Machine Learning: Particle Physics at the LHC

Counter-rotating proton beams in 27 km circumference ring

pp centre-of-mass energy 14 TeV

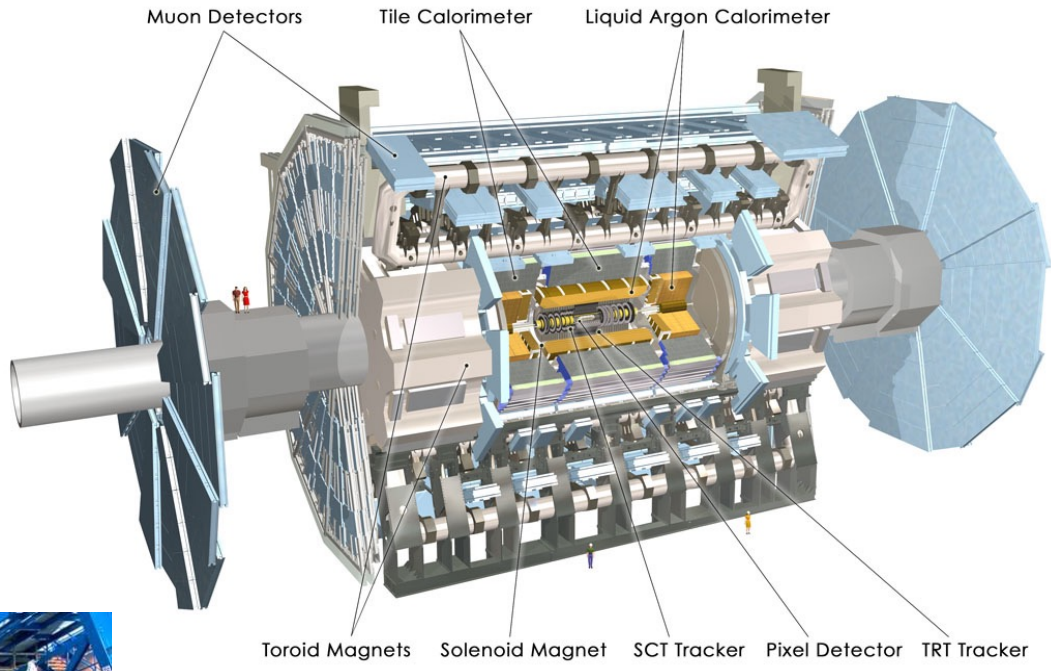


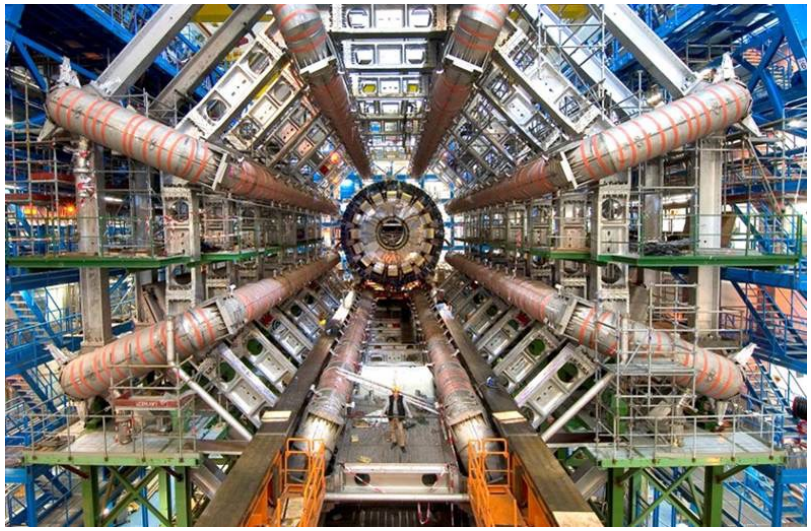
Detectors at 4 pp collision points:

ATLAS
CMS } general purpose
LHCb (b physics)
ALICE (heavy ion physics)

The ATLAS Detector at the LHC

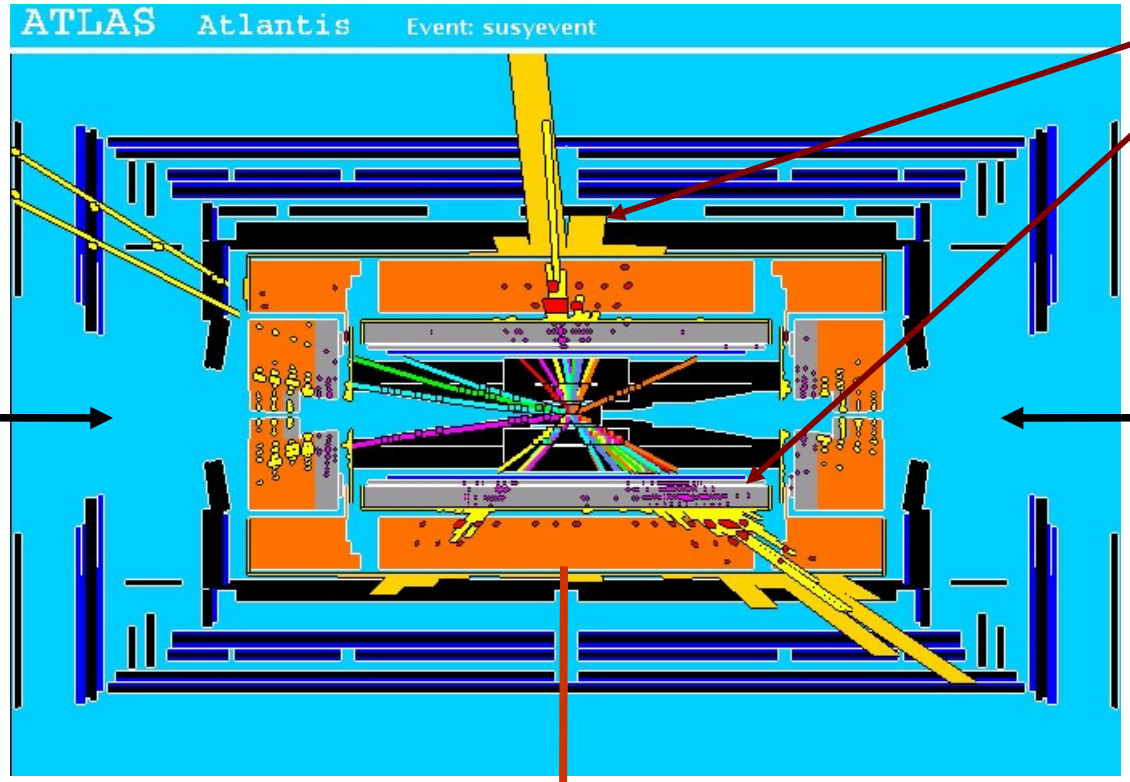
3000 physicists
37 countries
167 universities/labs





25 m diameter
46 m length
7000 tonnes
$\sim 10^8$ electronic channels

A simulated proton-proton collision from supersymmetry ("signal")



high p_T muons

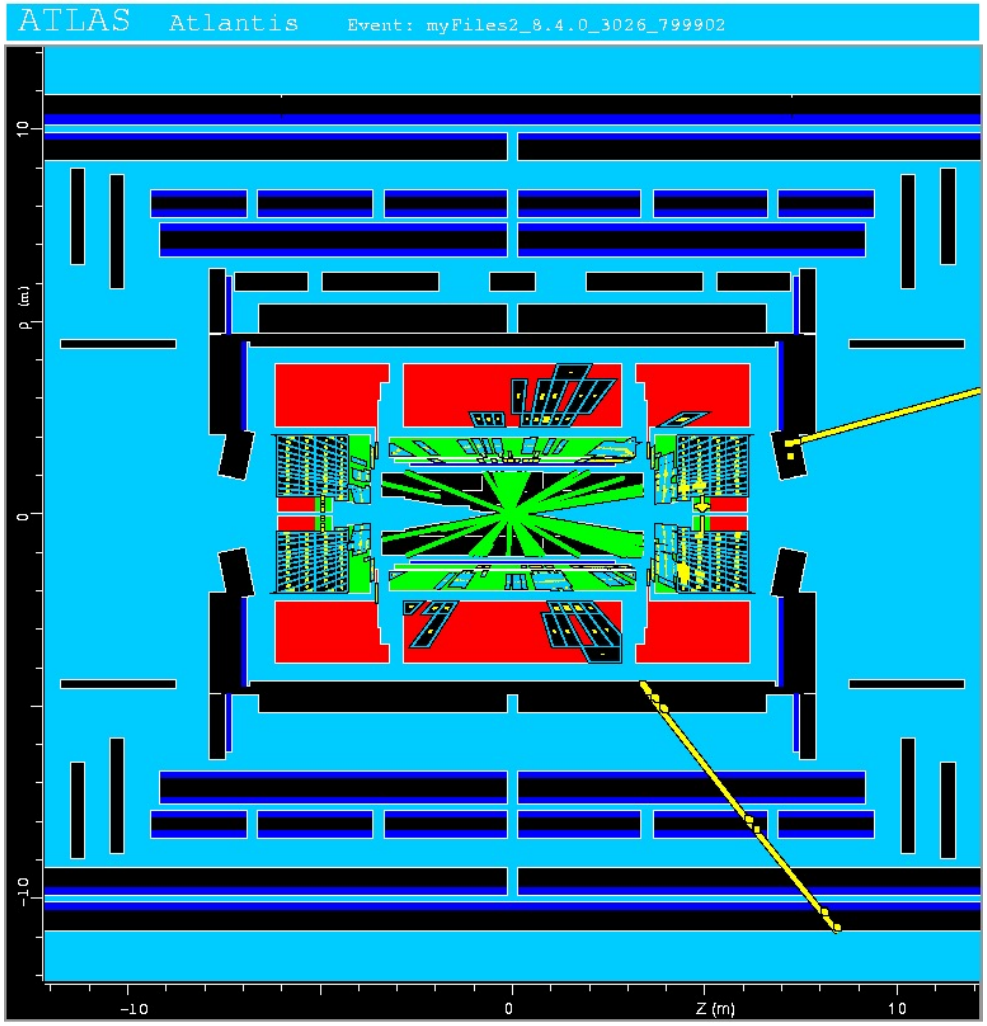
high p_T jets of hadrons

p

p

missing transverse energy

A simulated proton-proton collision from production of top quarks (“background”)



This event has features similar to what we hope to see in supersymmetric events, and thus constitutes a “background” that can mimic the “signal”.

Classification of proton-proton collisions

Proton-proton collisions can be considered to come in two classes:

signal (the kind of event we're looking for, $y = 1$)
background (the kind that mimics signal, $y = 0$)

For each collision (event), we measure a collection of features:

x_1 = energy of muon
x_2 = angle between jets
x_3 = total jet energy
x_4 = missing transverse energy
x_5 = invariant mass of muon pair
x_6 = ...

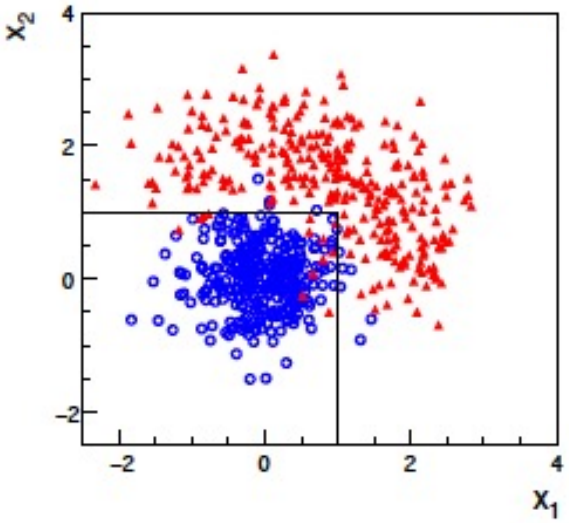
The real events don't come with true class labels, but computer-simulated events do. So we can have a set of simulated events that consist of a feature vector $\boldsymbol{x}$ and true class label y (0 for background, 1 for signal):

$$(\boldsymbol{x}, y)_1, (\boldsymbol{x}, y)_2, \ldots, (\boldsymbol{x}, y)_N$$

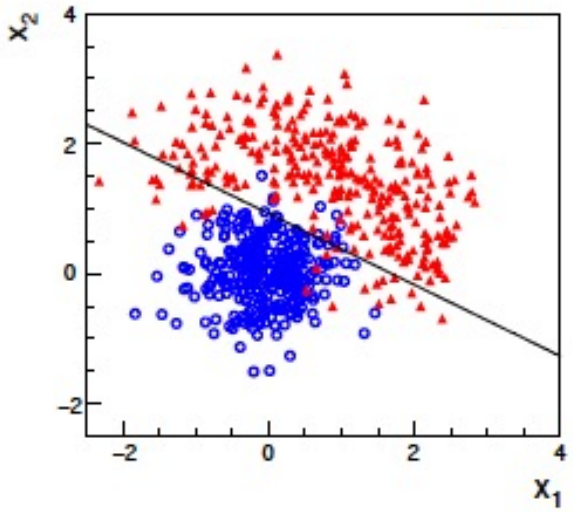
The simulated events are called "training data".

What is the best decision boundary?

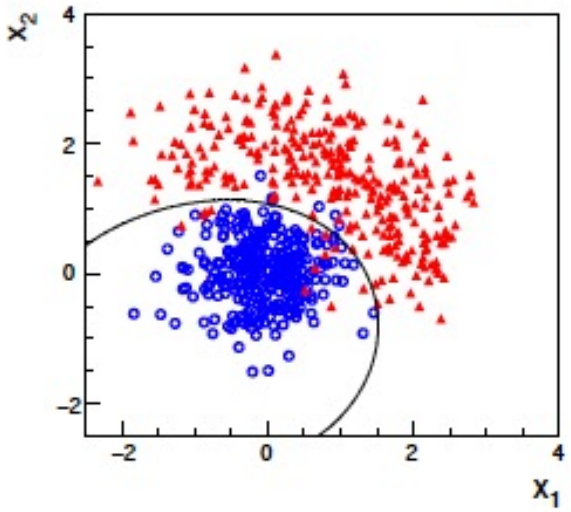
"cuts"



linear



nonlinear



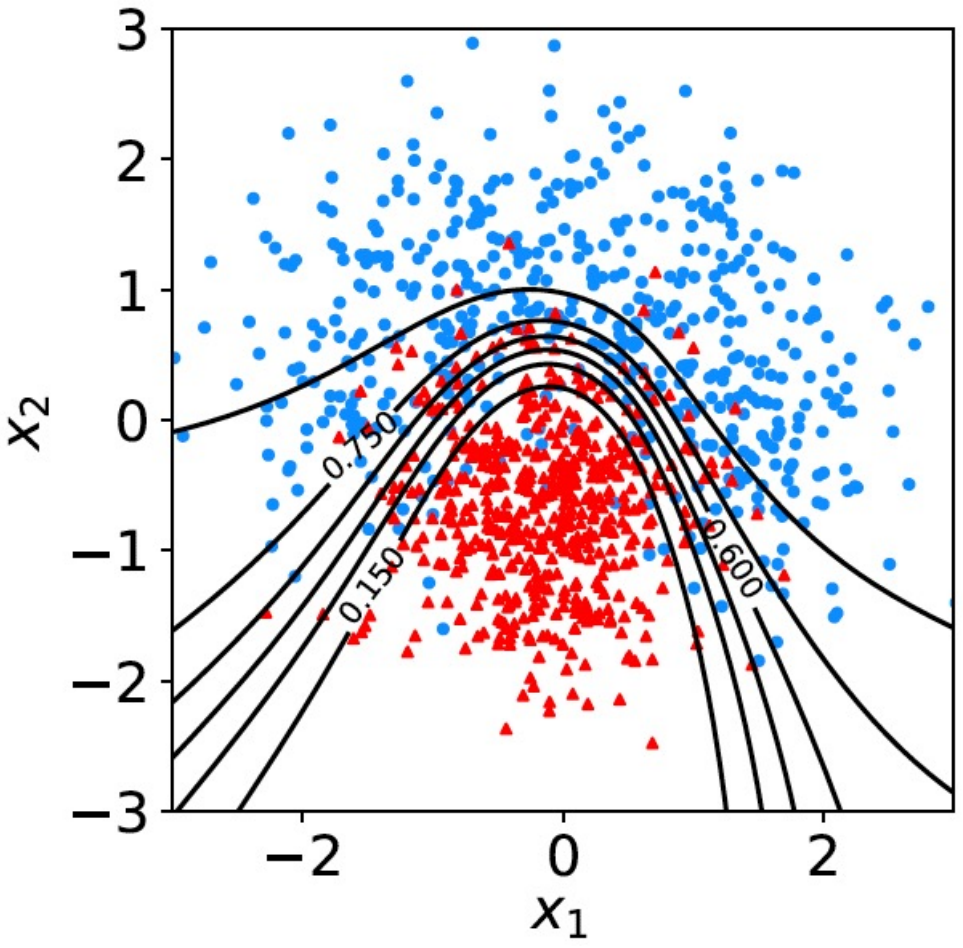
What is the best decision function?

A surface in an n-dimensional space can be described by

$$t(x_1, \ldots, x_n) = t_c$$

scalar function

constant



Different values of the constant t_c result in a family of surfaces.

Problem is reduced to finding the best decision function $t(\boldsymbol{x})$.

Linear decision boundary

A simple *Ansatz* is to try a decision function of the form

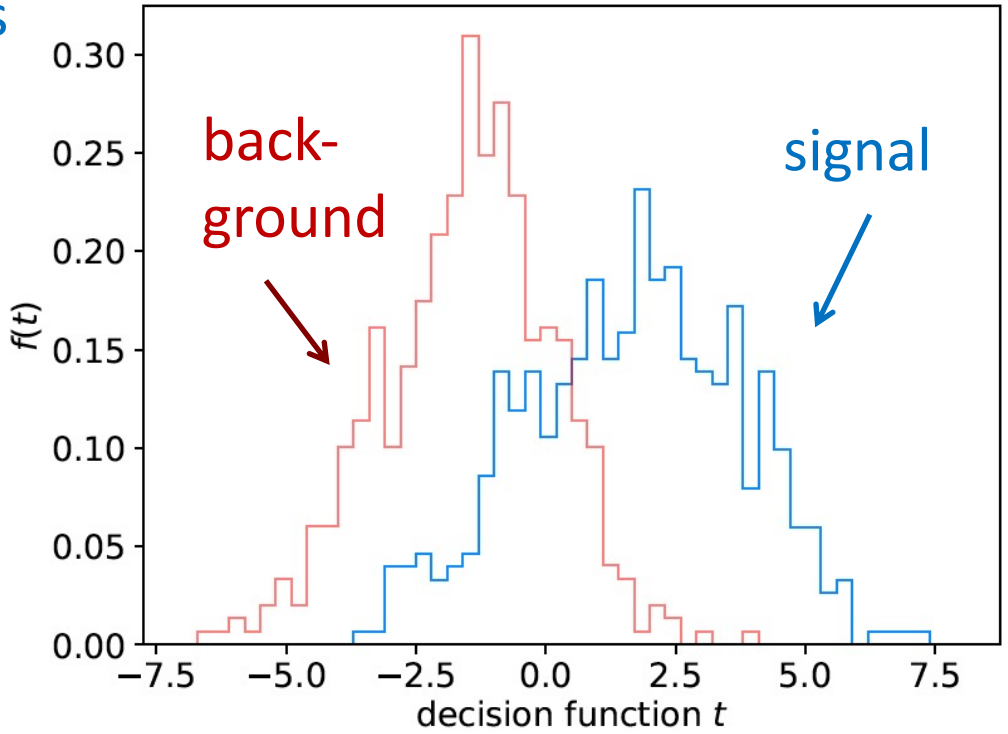
$$t(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

where the coefficients $w_1, \ldots, w_n$ are constants (or "weights") we need to determine using the training data.

A given choice of the weights fixes the function $t(\boldsymbol{x})$.

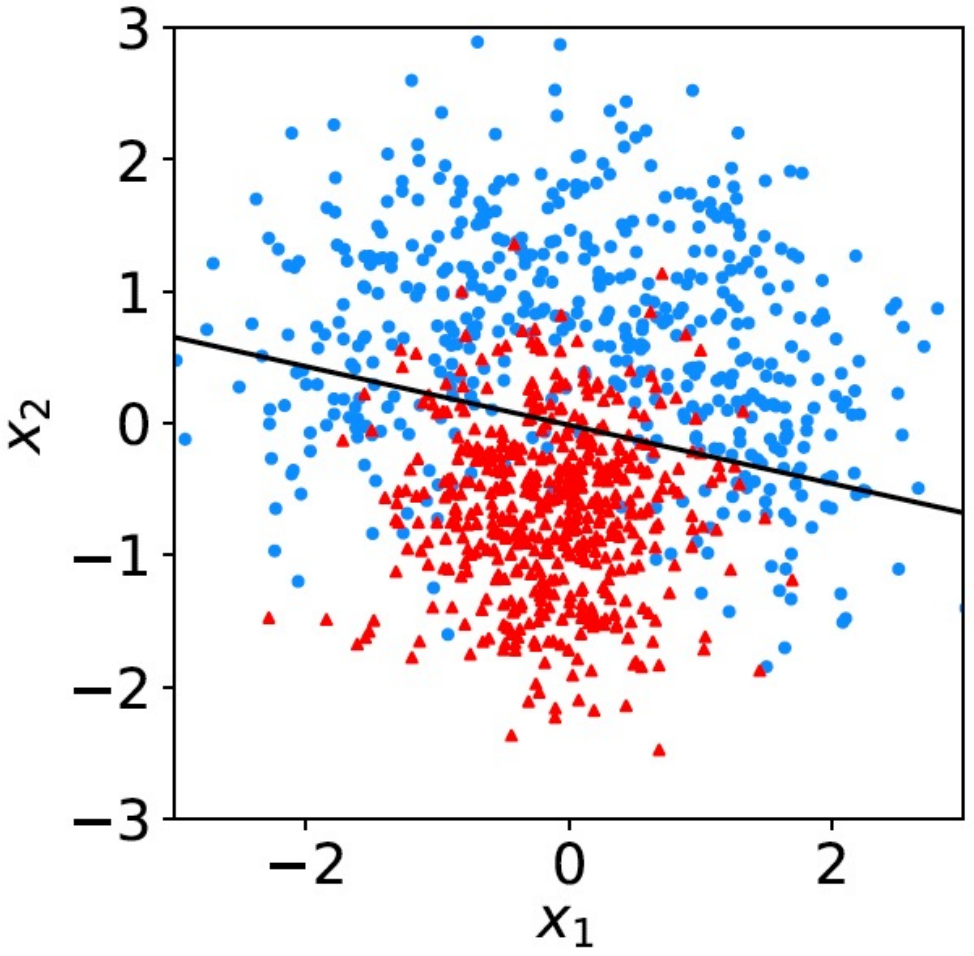
Look at the training events from the two classes, for each evaluate $t(\boldsymbol{x})$ and enter into a histogram.

Goal is to maximize the "separation" between the two distributions.



Fisher discriminant

Using a particular definition of what constitutes "best separation" due to R. Fisher one obtains a *Fisher discriminant*:



Outline

Part 1:

What Machine Learning is and how it can be applied

Classification of two types of “events”

Linear classifiers: Fisher Discriminant

→ Part 2:

Nonlinear classifiers: Neural Networks

Software for Machine Learning: scikit-learn

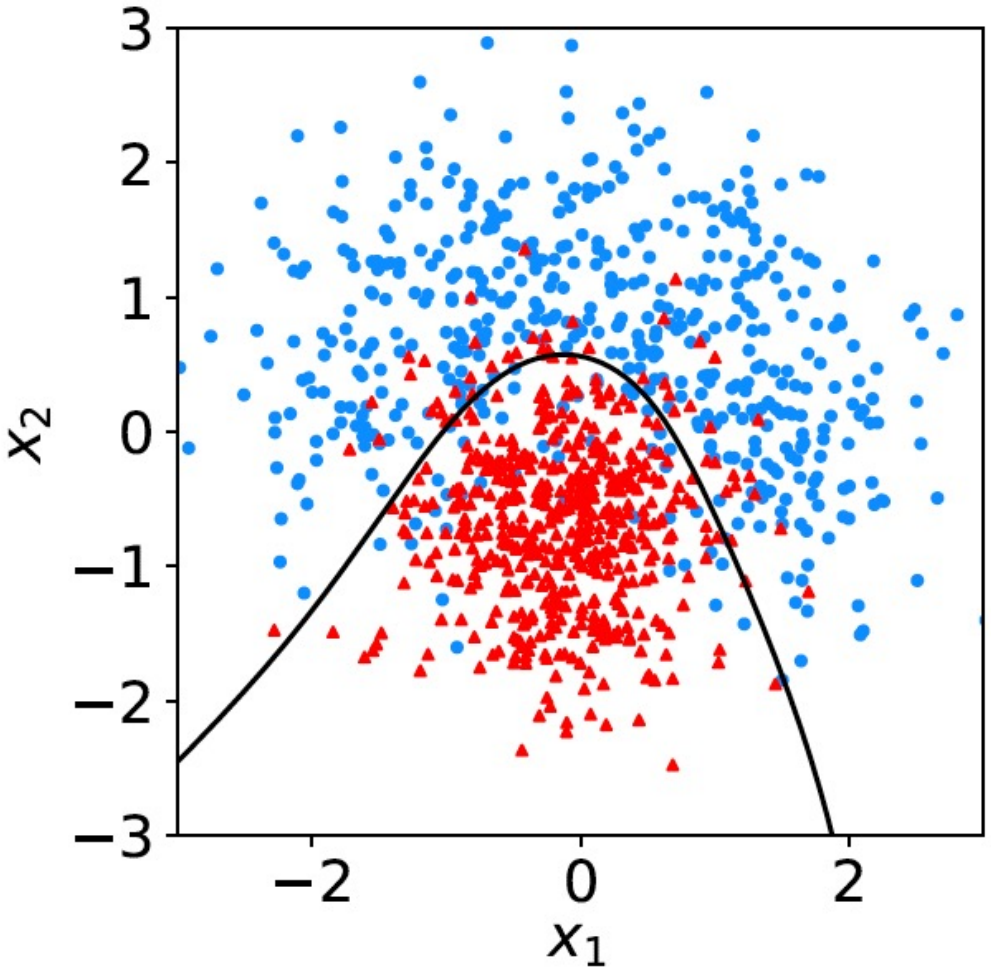
Exercises

Part 3 (extension):

Multiple regression

Nonlinear decision boundaries

From the scatter plot below it's clear that some nonlinear boundary would be better than a linear one:



And to have a nonlinear boundary, the decision function $t(\boldsymbol{x})$ must be nonlinear in $\boldsymbol{x}$.

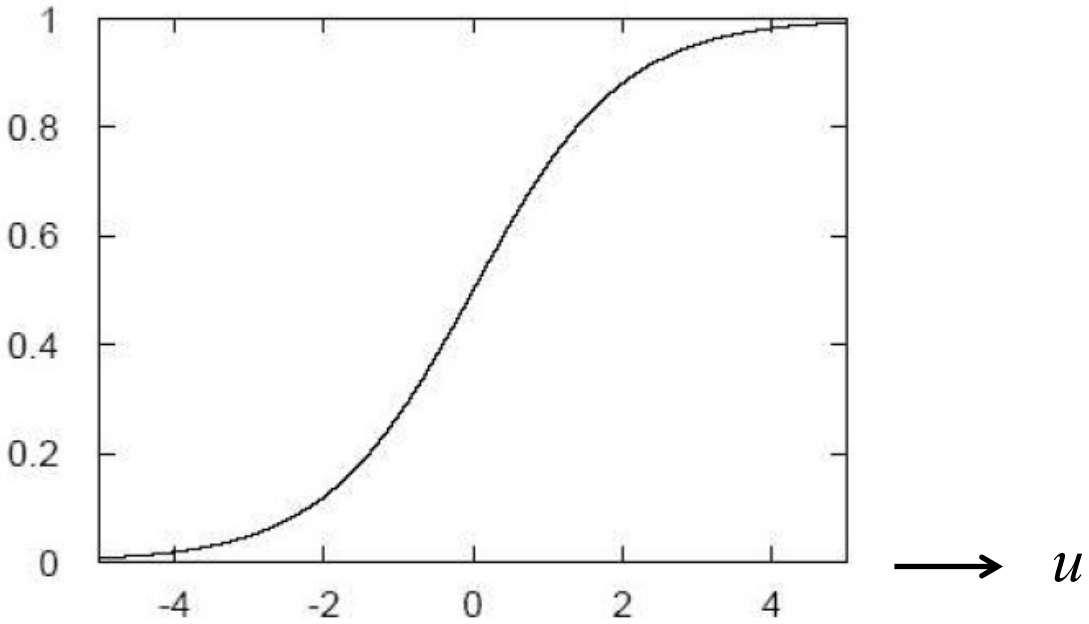
Neural Networks

A simple nonlinear decision function can be constructed as

$$t(\mathbf{x}) = h\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

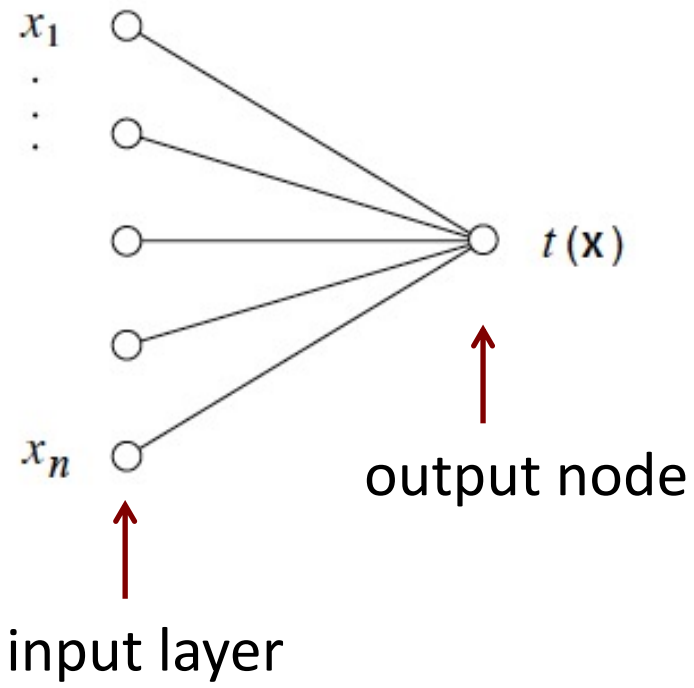
where h is called the "activation function". For this one can use, e.g., a logistic sigmoid function,

$$h(u) = \frac{1}{1 + e^{-u}}$$



Single Layer Perceptron

In this form, the decision function is called a Single Layer Perceptron – the simplest example of a Neural Network.



But the surface described by $t(\boldsymbol{x}) = t_c$ is the same as by

$$h^{-1}(t(\mathbf{x})) = w_0 + \sum_{i=1}^{n} w_i x_i = h^{-1}(t_c)$$

So here we still have a linear decision boundary.

Multilayer Perceptron

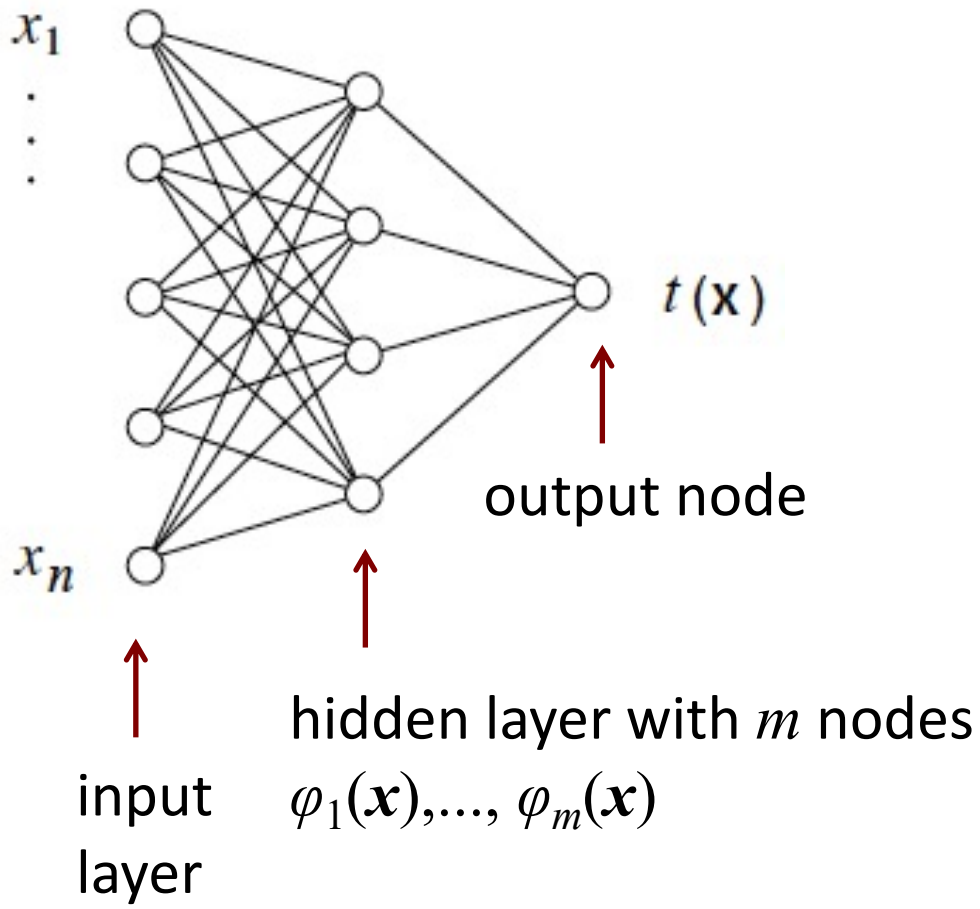
The Single Layer Perceptron can be generalized by defining first a set of functions $\varphi_i(\boldsymbol{x})$, with $i = 1,\ldots, m$:

$$\varphi_i(\mathbf{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j\right) \qquad i = 1, \ldots, m$$

The $\varphi_i(\boldsymbol{x})$ are then treated as if they were the input variables, in a perceptron, i.e., the decision function (output node) is

$$t(\mathbf{x}) = h\left(w_{10}^{(2)} + \sum_{j=1}^{n} w_{1j}^{(2)} \varphi_j(\mathbf{x})\right)$$

Multilayer Perceptron (2)



Each line in the graph represents one of the weights $w_{ij}^{(k)}$, which must be adjusted using the training data.

Training a Neural Network

To train the network (i.e., determine the best values for the weights), define a loss function, e.g.,

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} |t(\mathbf{x}_i) - y_i|^2$$

where $\boldsymbol{w}$ represents the set of all weights, the sum is over the set of training events, and y_i is the (numeric) true class label of each event (0 or 1).

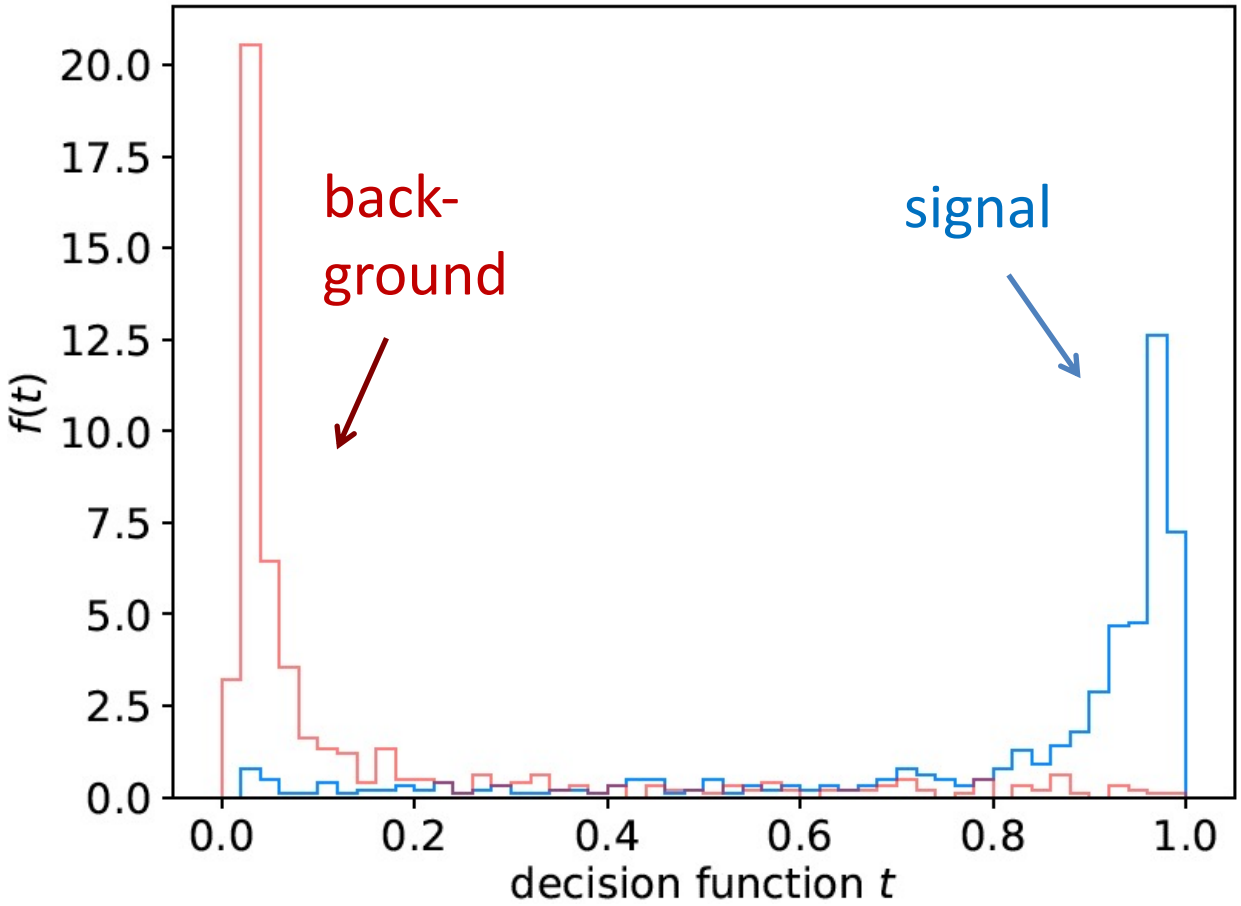
The optimal values of the weights are found by minimizing $E(\boldsymbol{w})$ with respect to the weights (non-trivial algorithms: backpropagation, stochastic gradient descent,...).

The desired result for an event with feature vector $\boldsymbol{x}$ is:

if the event is of type 0, want $t(\boldsymbol{x}) \sim 0$,
if the event is of type 1, want $t(\boldsymbol{x}) \sim 1$.

Distribution of neural net output

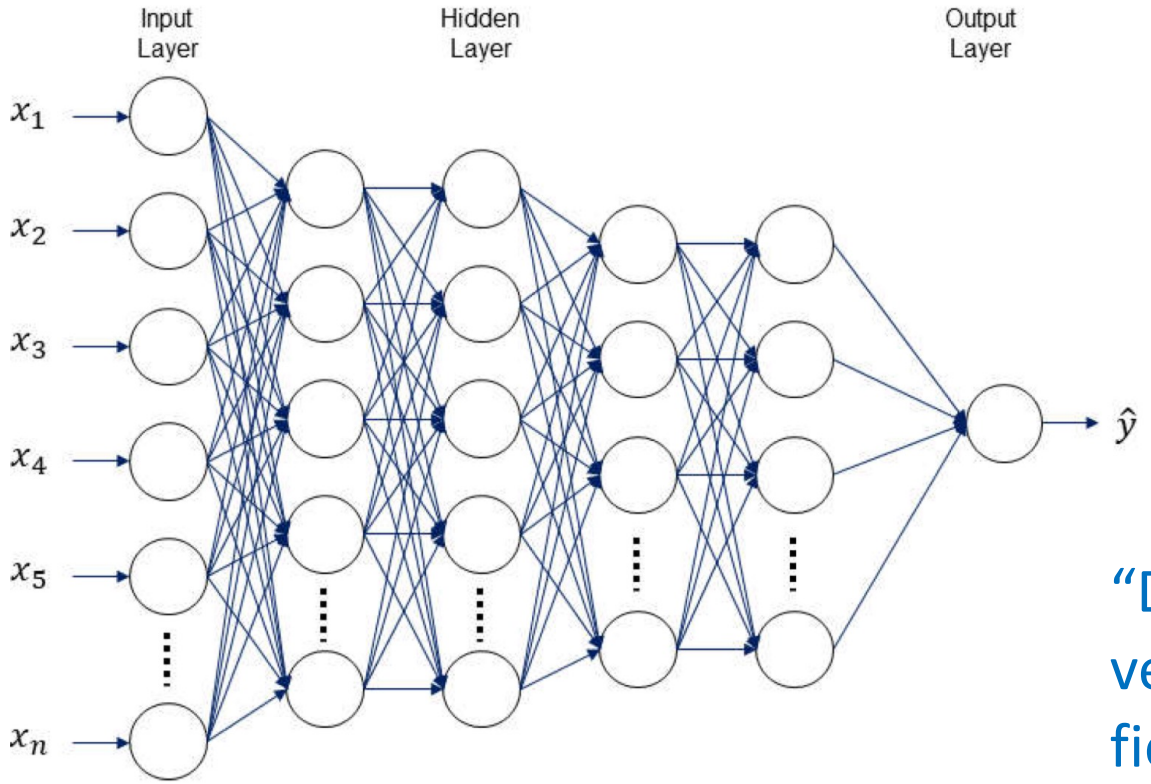
Degree of separation between classes now much better than with linear decision function:



Deep Neural Networks

The multilayer perceptron can be generalized to have an arbitrary number of hidden layers, with an arbitrary number of nodes in each (= "network architecture").

A "deep" network has several (or many) hidden layers:

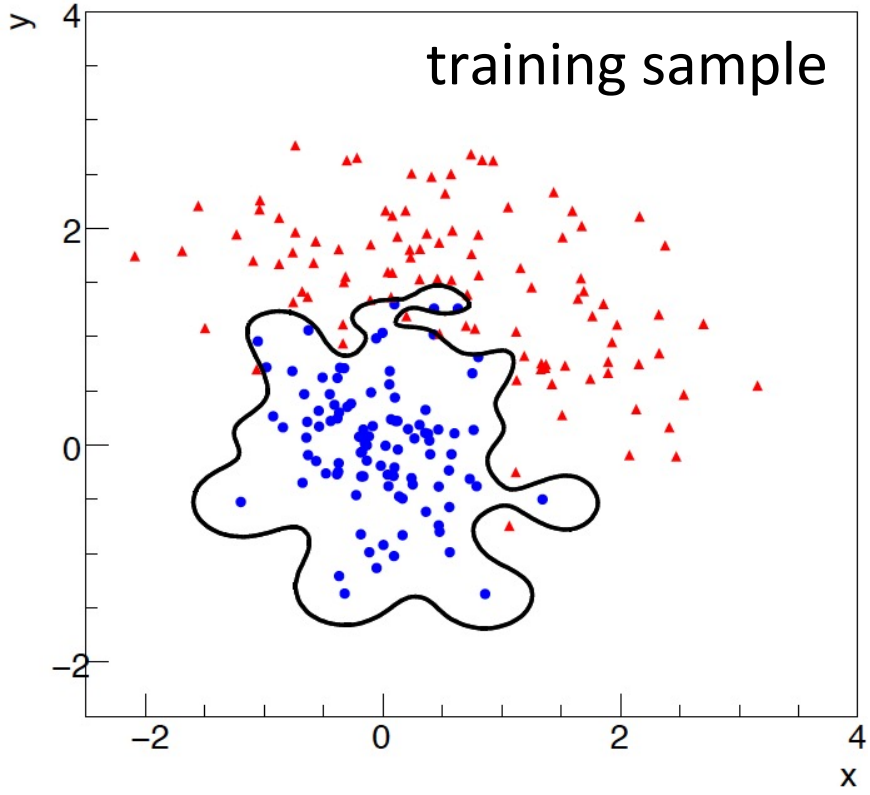


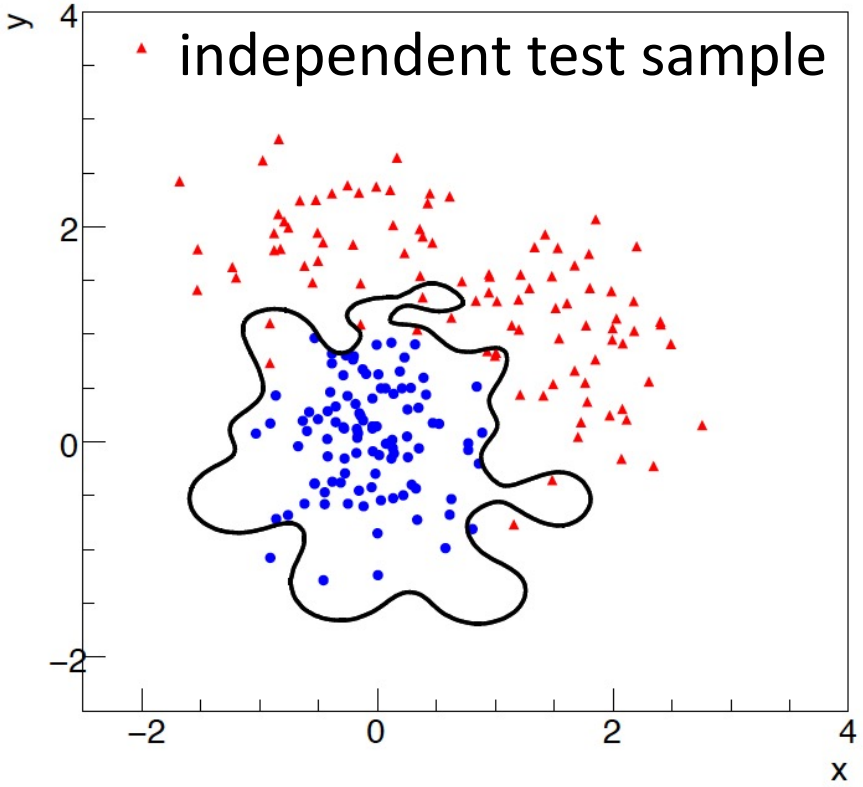
"Deep Learning" is a very recent and active field of research.

Overtraining

Including more parameters in a classifier makes its decision boundary increasingly flexible, e.g., more nodes/layers for a neural network.

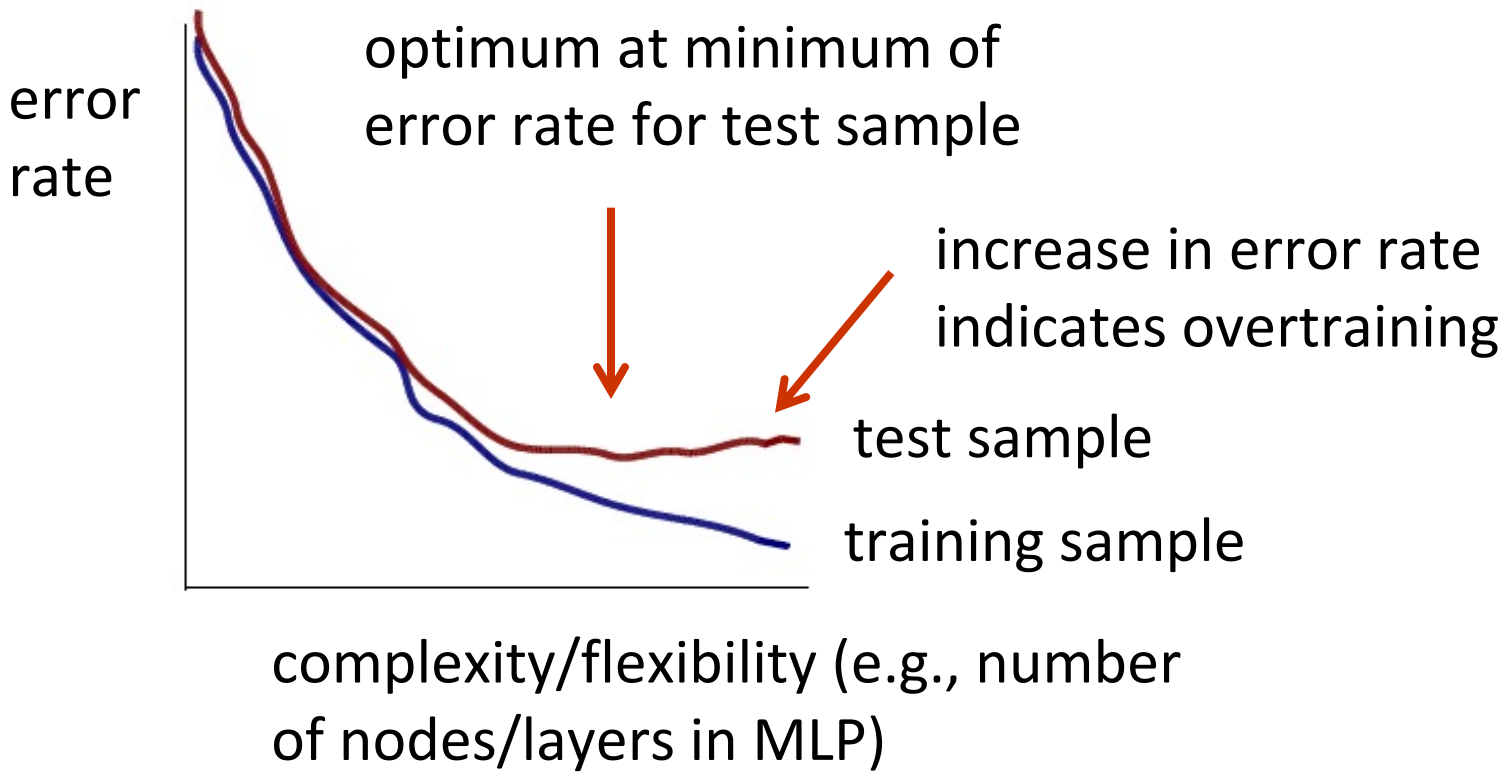
A “flexible” classifier may conform too closely to the training points; the same boundary will not perform well on an independent test data sample (→ “overtraining”).





Monitoring overtraining

If we monitor the fraction of misclassified events (or similar, e.g., loss function $E(\boldsymbol{w})$) for test and training samples, it will usually decrease for both as the boundary is made more flexible:



Other types of classifiers

We have seen only two types of classifiers:

Linear (Fisher discriminant)

Neural Network

There are many others:

Support Vector Machine

Boosted Decision Tree

K-Nearest Neighbour

...

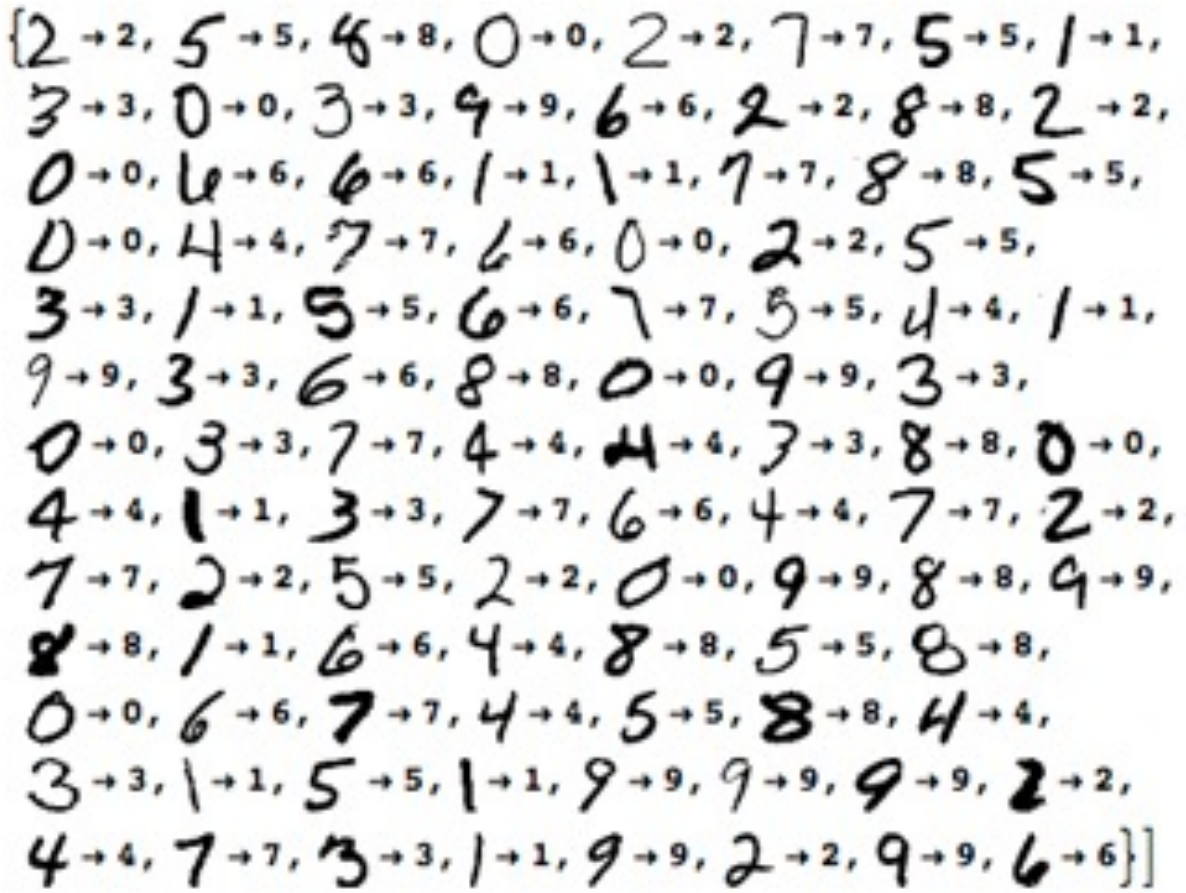
The field is rapidly developing with advances, e.g., that allow one to use feature vectors of very high dimension, such as the pixels of an image.

$\rightarrow$ face/handwriting recognition, driverless cars...

https://www.wolfram.com/mathematica/new-in-10/highly-automated-machine-learning/

Machine Learning for handwriting recognition

Initial feature vector = set of pixel values of an image



Y. LeCun

Scene parsing/labeling with Convolutional Neural Nets

[Farabet et al. ICML 2012, PAMI 2013]

See *Deep Learning and the Future of AI*, seminar at CERN by Yann LeCun: **https://indico.cern.ch/event/510372/**

Software for Machine Learning

We will practice ML with the Python package scikit-learn

scikit-learn.org ← software, docs, example code

scikit-learn built on NumPy, SciPy and matplotlib, so you need

```
import scipy as sp
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

and then you import the needed classifier(s), e.g.,

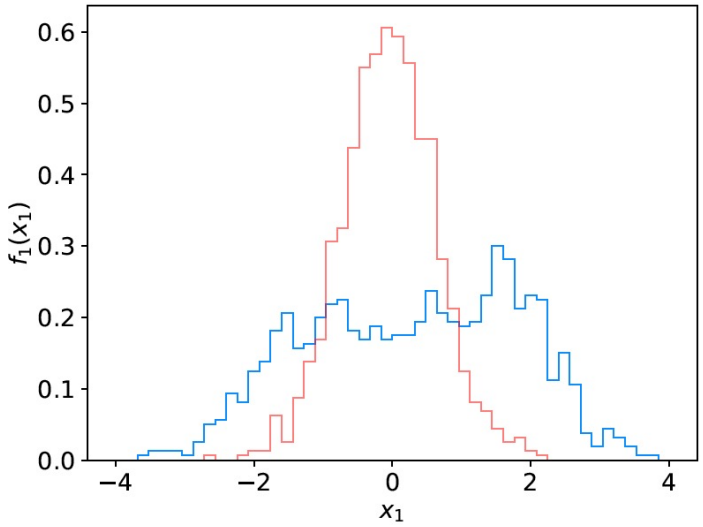
```
from sklearn.neural_network import MLPClassifier
```

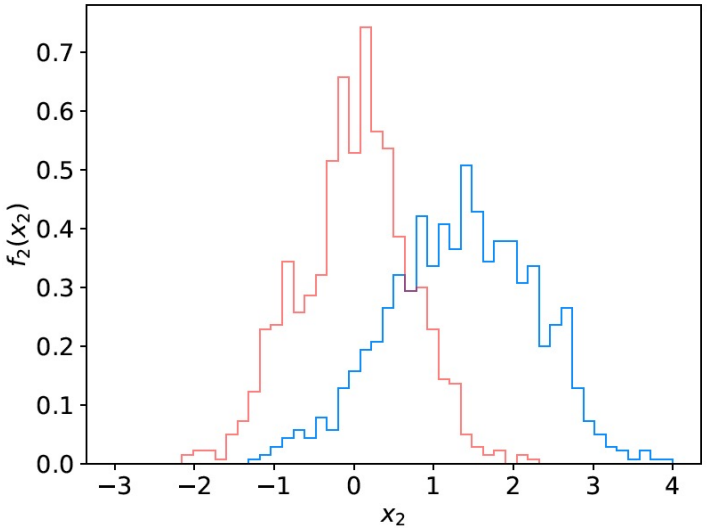
For a list of the various classifiers in scikit-learn see the docs on scikit-learn.org, also a very useful sample program:

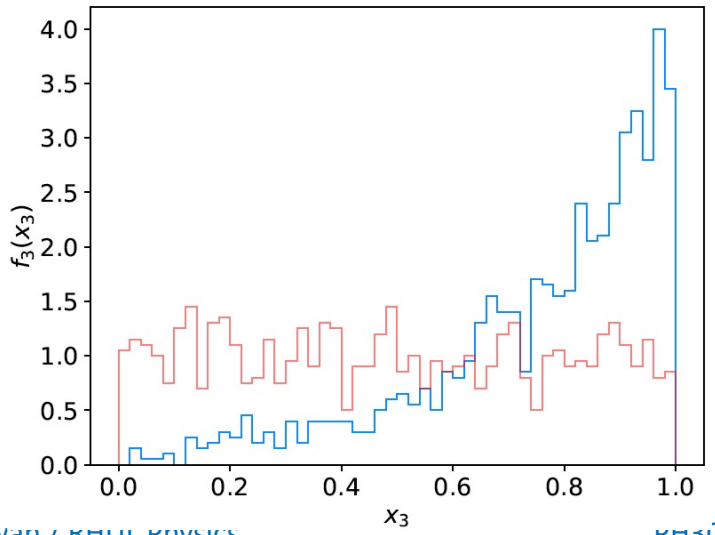
http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Example: the data

We will do an example with data corresponding to events of two types: signal ($y = 1$, blue) and background ($y = 0$, red).







Each event is characterised by 3 quantities: $\boldsymbol{x} = (x_1, x_2, x_3)$.

Components are correlated.

Suppose we have 1000 events each of signal and background.

Reading in the data

scikit-learn wants the data in the form of numpy arrays:

```
#  read the data in from files,
# assign target values 1 for signal, 0 for background
sigData = np.loadtxt('signal.txt')
nSig = sigData.shape[0]
sigTargets = np.ones(nSig)
bkgData = np.loadtxt('background.txt')
nBkg = bkgData.shape[0]
bkgTargets = np.zeros(nBkg)

# concatenate arrays into data X and targets y
# split into two parts:  use one for training, the other for testing
X = np.concatenate((sigData,bkgData),0)[:,0:2]       # for now, only use x1, x2
y = np.concatenate((sigTargets, bkgTargets))

# split data into training and testing samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
                                                    random_state=1)
```

Create, train, evaluate the classifier

Create an instance of the MLP (multilayer perceptron) class and "train", i.e., adjust the values of the weights to minimise the loss function.

Here we request 3 hidden layers with 10 nodes each:

```
# create classifier object and train
clf = MLPClassifier(hidden_layer_sizes=(10,10,10), activation='tanh',
                    max_iter=2000, random_state=6)
clf.fit(X_train, y_train)
```

Use test data to see what fraction of events are correctly classified (default takes threshold of 0.5 for decision function)

```
# evaluate its accuracy (= 1 – error rate) using the test data
y_pred = clf.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

Evaluating the decision function

So now for an arbitrary point (x_1, x_2) in the feature space, we can evaluate the decision:

```
xt = np.array([0.37, 2.46]).reshape((1,-1))      # input is numpy array
t = clf.predict_proba(xt)[0, 1]
```

Or we may have an array of points in ***x***-space, so we can get an array of probabilities:

```
t = clf.predict_proba(X_test)[:, 1]          # returns prob to be of type y=1
```

Can get this separately for the signal and background events and make histograms (see sample code).

Note for most other classifiers, the decision function is called decision_function – use this instead of predict_proba.

Exercises

Run the program simpleClassifier.py and describe the output. It is set up to use at first only the first two components, x_1 and x_2, so that the results can be displayed as a scatter plot.

Change program to use all three input variables by removing the line X = X[:,0:2] and you will have to side-step the code that makes the scatter plot.

Change numbers of hidden layers and nodes – try to find the maximum possible classification accuracy.

For 1 hidden layer e.g., with 10 nodes: hidden_layer_sizes=(10,) .

Note if the number of requested layers/nodes gets too large, it will not be possible to train (find the minimum of the loss function).

For the best architecture that you find, use the test sample to produce a histogram of the network output (see sample code).

Exercises (continued)

By looking at the scikit-learn documentation and the sample program plot_classifier_comparison mentioned above, implement a linear classifier (class LinearDiscriminantAnalysis) using all three input variables.

Make a histogram of the classifier output; compare its performance to your best neural network.

By consulting the documentation and sample program, implement at least one of:

- *K*-Nearest Neighbor Classifier (KNeighborsClassifier).
- Support Vector Machine (SVC)
- Boosted Decision Tree (AdaBoostClassifier)

Write up the ML exercises as a single section of your project report.

Resources on Machine Learning

Mini-intro to ML included in PH4515 Statistical Data Analysis

Course in our CS Dept., CS3920/CS4920 (also KCL Maths).

Online courses, e.g., K. Markham, Introduction to machine learning with scikit-learn (also on youtube):

www.dataschool.io/machine-learning-with-scikit-learn/

Many online courses, tutorials – worth a look but often very qualitative and/or approach from computer science angle.

More python software (very rapid development):

pandas, seaborn, tensorflow, keras, theano,..

Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, An Introduction to Statistical Learning with Applications in R

www-bcf.usc.edu/~gareth/ISL/

https://youtu.be/5N9V07EIfIg

Outline

Part 1:

What Machine Learning is and how it can be applied

Classification of two types of "events"

Linear classifiers: Fisher Discriminant

Part 2:

Nonlinear classifiers: Neural Networks

Software for Machine Learning: scikit-learn

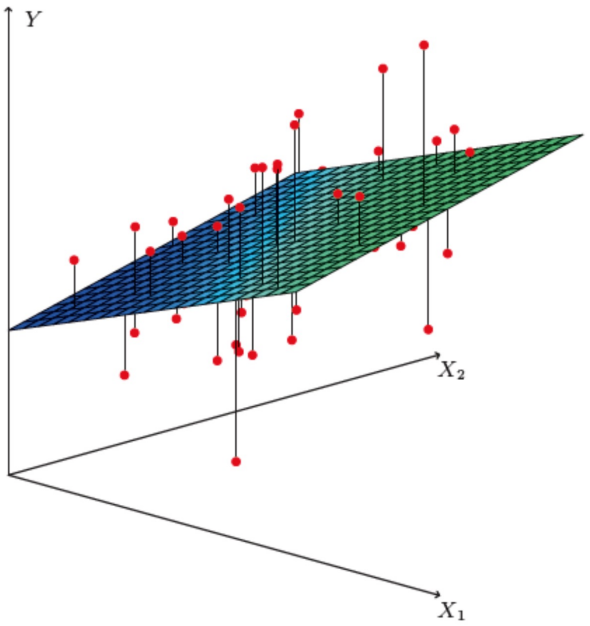
Exercises

→ Part 3 (extension):

Multiple regression

Brief intro to multiple regression

Multiple regression* can be seen as an extension of curve fitting to the case where the variable x is replaced by a multi-dimensional $\boldsymbol{x} = (x_1,...,x_n)$, e.g., fitting a surface. Here suppose the data are points $(\boldsymbol{x}_i, y_i)$, $i = 1,...,N$ (no error bars) and $\boldsymbol{x}$ is usually a random variable, often called the explanatory or predictor variable.



Equivalently, we can view it as an extension to classification with the discrete class label $y = 0, 1$ replaced by a continuous target y (and in this context $\boldsymbol{x}$ can also be called the feature vector).

*Note the term ”multivariate” regression refers to a vector target variable y; here we treat only scalar y.

Target (fit) function and loss function

As in the case of curve fitting, we assume some parametric function of $\boldsymbol{x}$ that represents the mean of the target variable

$$E[y] = f(\mathbf{x}; \mathbf{w})$$

where $\boldsymbol{w}$ is a vector of adjustable parameters ("weights").

Suppose we have training data consisting of $(\boldsymbol{x}_i, y_i)$, $i = 1,...,N$.

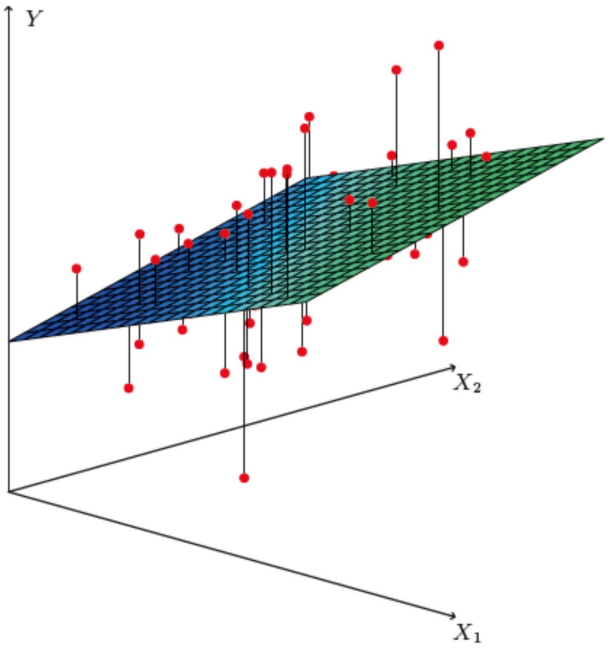
Use these to determine the weights by minimizing a loss function (analogous to the χ^2), e.g.,

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} |y_i - f(\mathbf{x}_i; \mathbf{w})|^2$$

Linear regression

In linear regression, the fit function is of the form

$$f(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{n} w_i x_i$$



i.e. the problem is equivalent to an unweighted least-squares fit of a (hyper-)plane:

Can be generalized to a nonlinear surface with higher order terms,

$$f(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i,j=1}^{n} w_{ij} x_i x_j + \sum_{i,j,k=1}^{n} w_{ijk} x_i x_j x_k + \ldots$$

Nonlinear regression

Other examples of nonlinear regression include:

MLP (multilayer perceptron) regression

Boosted decision tree regression

Support vector regression

For MLP regression, as with classification, regard the feature vector as the layer $k = 0$; i.e., $\varphi_i^{(0)} = x_i$.

The ith node of hidden layer k is

$$\varphi_i^{(k)} = h\left(w_{i0}^{(k)} + \sum_{j=1}^{n} w_{ij}^{(k)} \varphi_j^{(k-1)} \right)$$

where h is the activation function (tanh, relu, sigmoid,...).

MLP Regression (cont.)

For the final layer (k=K), in MLP regression (in contrast to classification), one omits the activation function, i.e.,

$$f(\mathbf{x};\mathbf{w}) = w_0^{(K)} + \sum_{j=1}^{n} w_j^{(K)} \varphi_j^{(K-1)}$$

where $\varphi_j^{(K-1)} =$ are the nodes of the last hidden layer ($k = K-1$).

For info on other types of multiple regression see, e.g.,

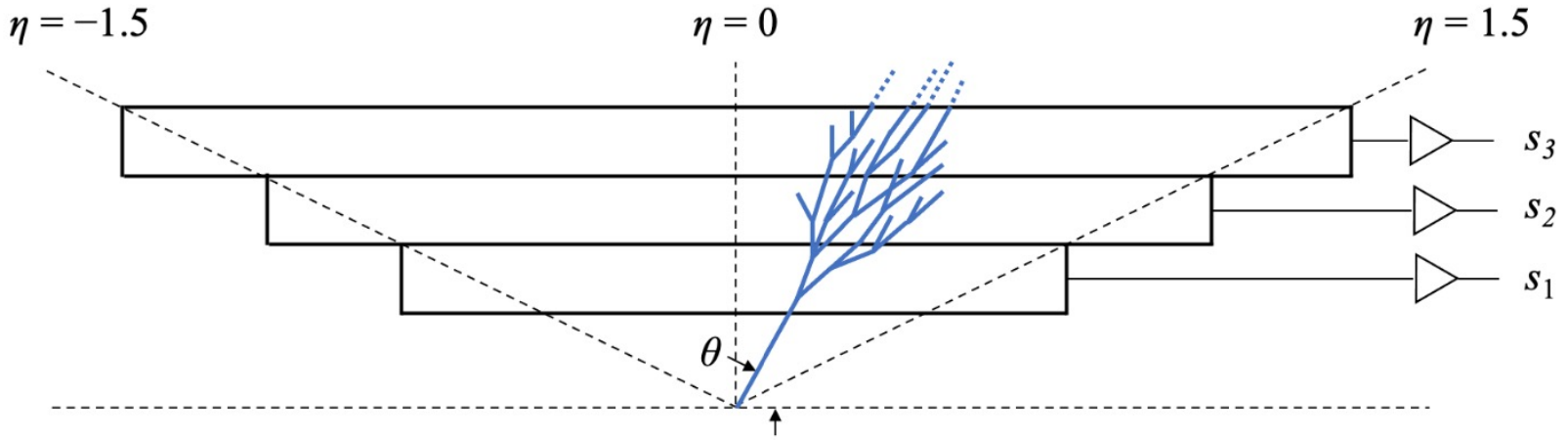
Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, 2013;
http://www-bcf.usc.edu/~gareth/ISL/

and the scikit-learn documentation.

Multiple regression example

Suppose particles with different energies E and angles θ (or equivalently $\eta = -\ln \tan(\theta/2)$) enter a calorimeter and create a particle showers that gives signals in three layers, s_1, s_2 and s_3, as well as an estimate of η.

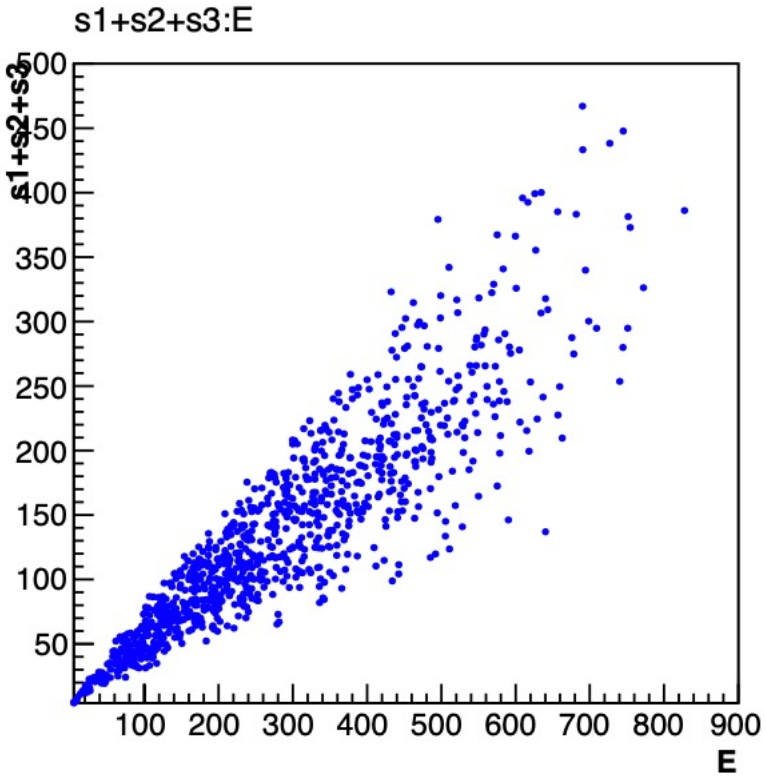
Some of the energy leaks through, with increased leakage for higher energy and more oblique angles (higher η).



The goal is to estimate the target $y_i = E_i$ given feature vectors $\boldsymbol{x}_i = (\eta, s_1, s_2, s_3)_i$ for $i = 1,...,N$ training events.

Energy estimate from sum of signals

Naively, one could try just summing the signals: $\hat{E} = s_1 + s_2 + s_3$

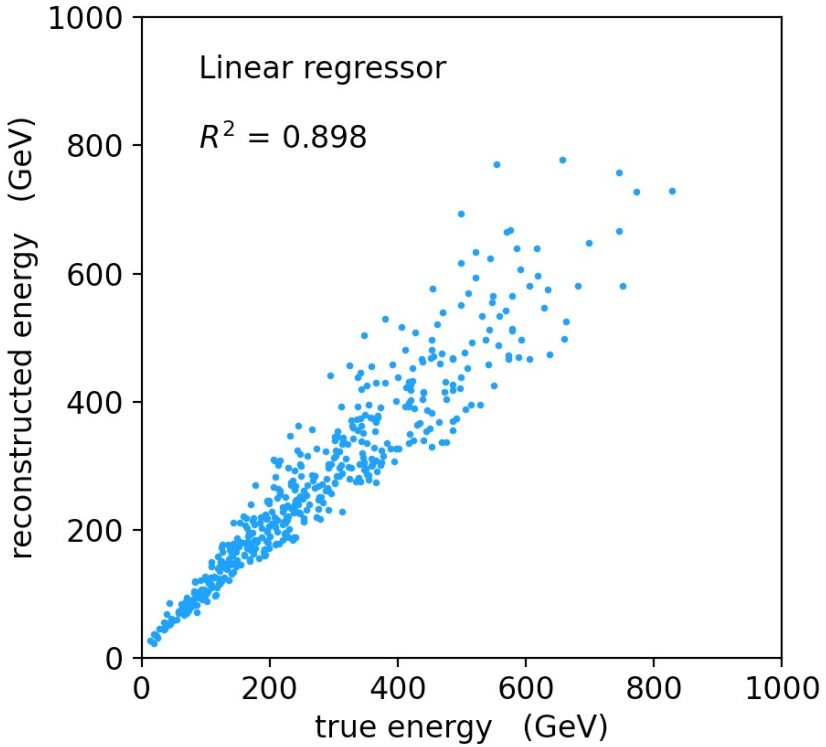


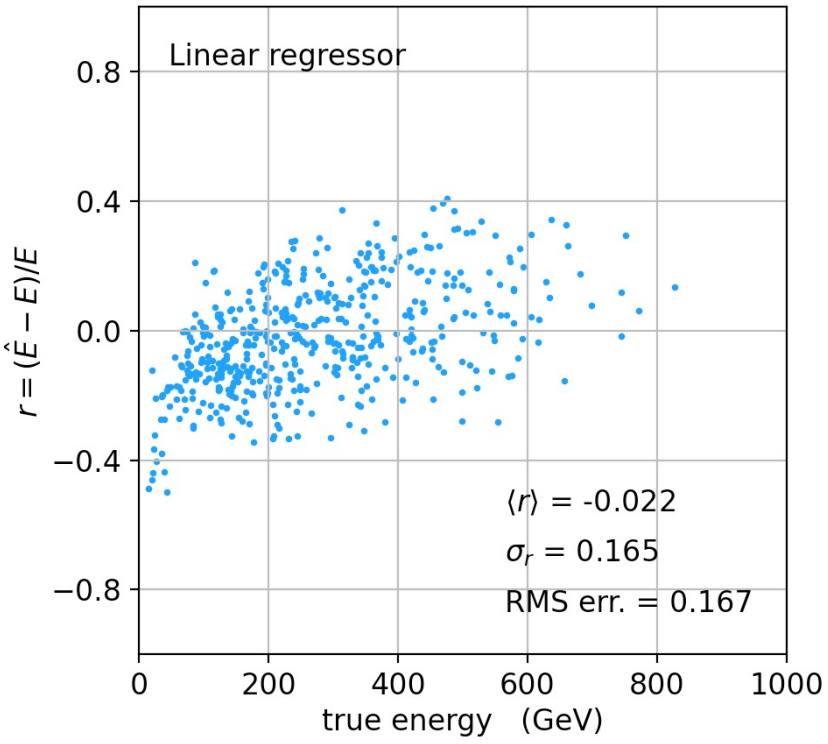
Gives very poor resolution because the particles have a distribution of energies and angles and hence differing amounts of the energy leak through undetected.

Linear regression

See MVRegressor.py, here using

```
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

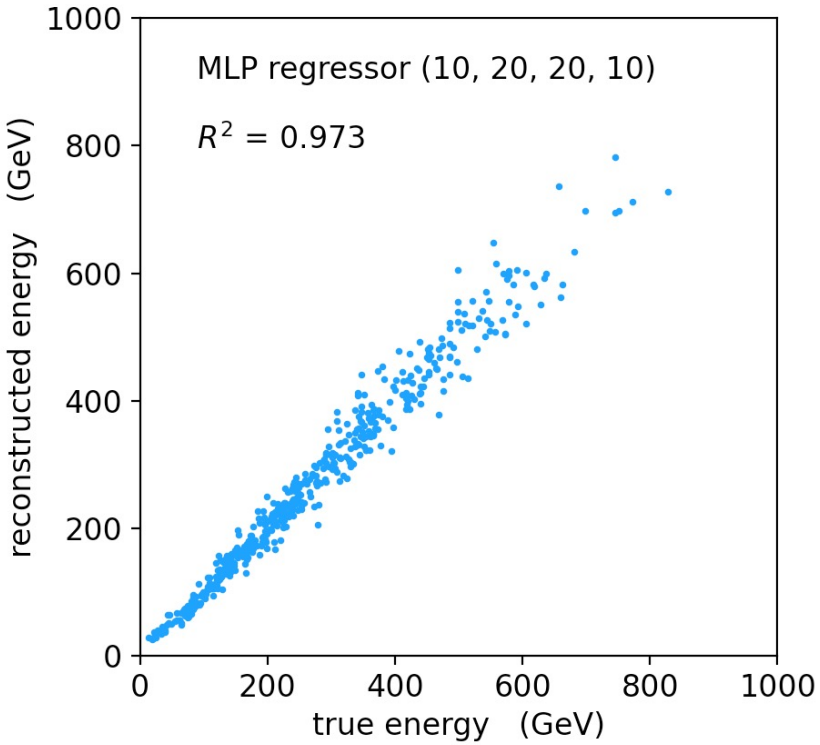


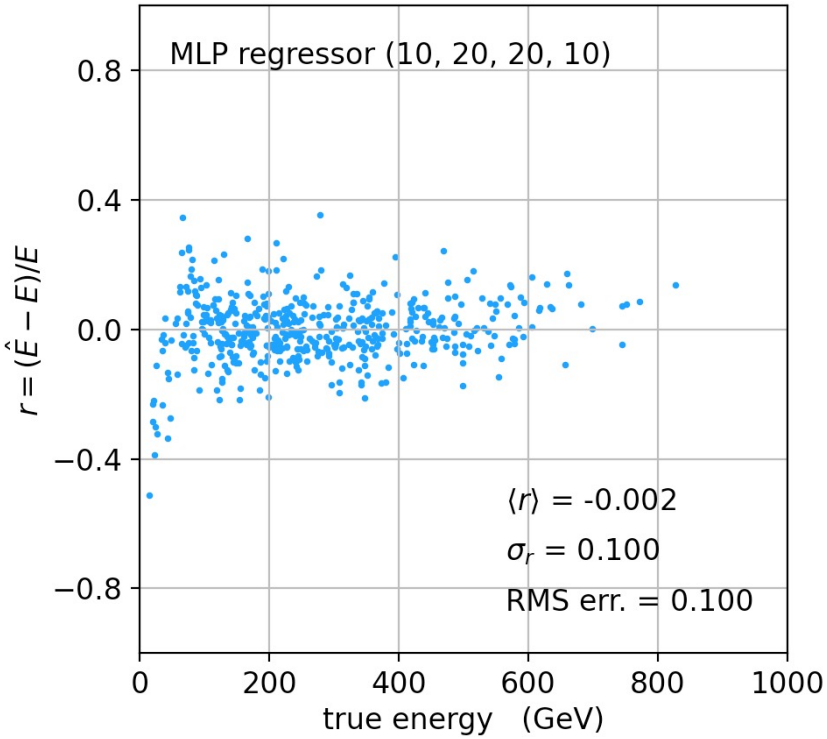


Average relative resolution 16.7%.

MLP Regression

```
regr = MLPRegressor(hidden_layer_sizes=(10,20,20,10), activation='relu'
regr.fit(X_train, y_train)
```





Better resolution (10%), here significant bias at low energies.

Refinements for multiple regression

One can try many improvements:

Scaling of predictor and target variables, e.g., standardize to zero mean and unit variance.

Use cross-validation to assess accuracy (and hence use entire sample of events for training.

Try different loss functions.

Try different regression algorithms (ridge regression, lasso, decision tree, support vector regression,...).

Some simple code using scikit-learn and a short write-up (from a year-3 project) is on the course webpage.