

PH3010 / MSci Skills

Statistical Data Analysis

Week 3: Introduction to Machine Learning

Autumn term 2017

Glen D. Cowan
RHUL Physics



Outline – week 3

What Machine Learning is and how it can be applied

Classification of two types of “events”

Linear classifiers: Fisher Discriminant

Nonlinear classifiers: Neural Networks

Software for Machine Learning: scikit-learn

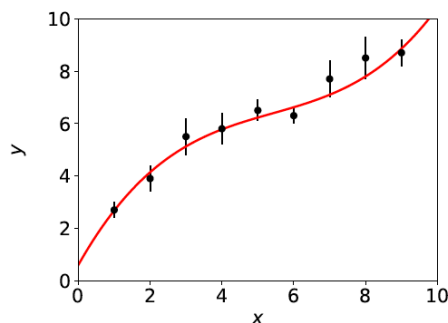
Exercises

What Machine Learning is

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

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:

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

Values of parameters are “learnt” 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 set $\mathbf{x} = (x_1, \dots, x_n)$.

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

Artificial Intelligence

Pattern Recognition

Statistical Learning

Multivariate Analysis

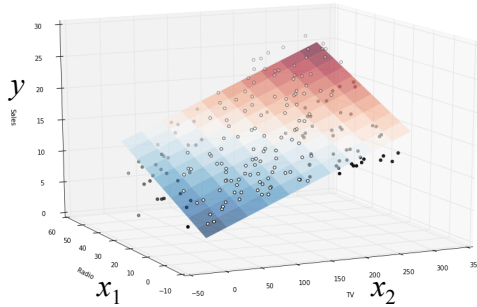
Development from (mainly) Computer Science, (also) Statistics; sometimes “Data Science” used to refer to all of above.

Curve fitting → regression

The generalisation of curve fitting with a multidimensional control variable $x \rightarrow x = (x_1, \dots, x_n)$ is called multivariate regression.

The data consist of sets of points (x_i, y_i) , where now 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 x .



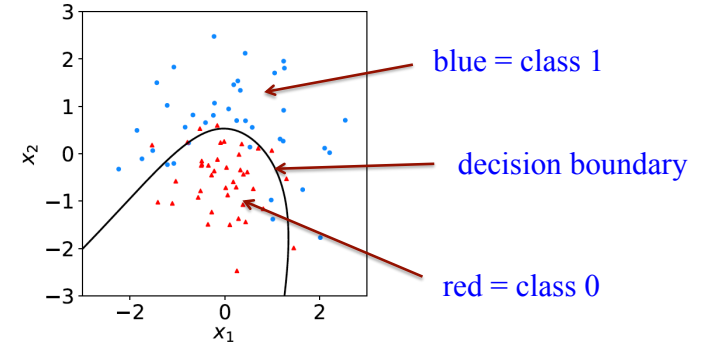
E.g. in two dimensions, multivariate regression means fitting a surface $f(x_1, x_2)$ to data points $(x_{1,i}, x_{2,i}, y_i)$:

Classification

A related type of ML algorithm is called **classification**: the data consist of points (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 x -space so as to predict the class y of a new instance of x .



Example of classification: Industrial Fishing

You scoop up fish which are of two types:



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

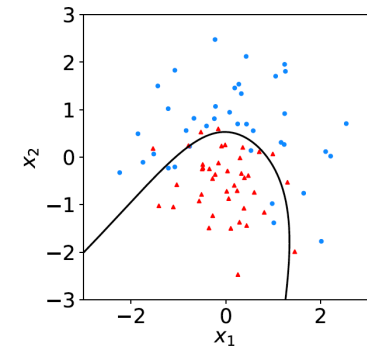
- $x_1 =$ length
- $x_2 =$ width
- $x_3 =$ weight
- $x_4 =$ area of fins
- $x_5 =$ mean spectral reflectance
- $x_6 =$...

These constitute the “feature vector” $x = (x_1, \dots, x_n)$.

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 $x = (x_1, x_2)$, 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



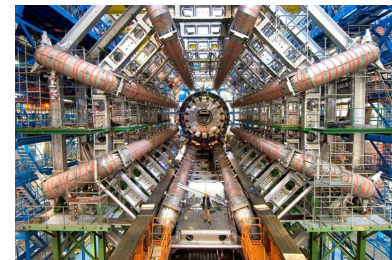
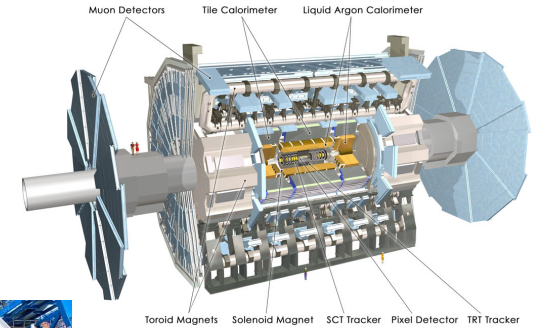
Counter-rotating proton beams
in 27 km circumference ring
pp centre-of-mass energy 14 TeV

- Detectors at 4 pp collision points:
- ATLAS
 - CMS ← general purpose
 - LHCb (b physics)
 - ALICE (heavy ion physics)



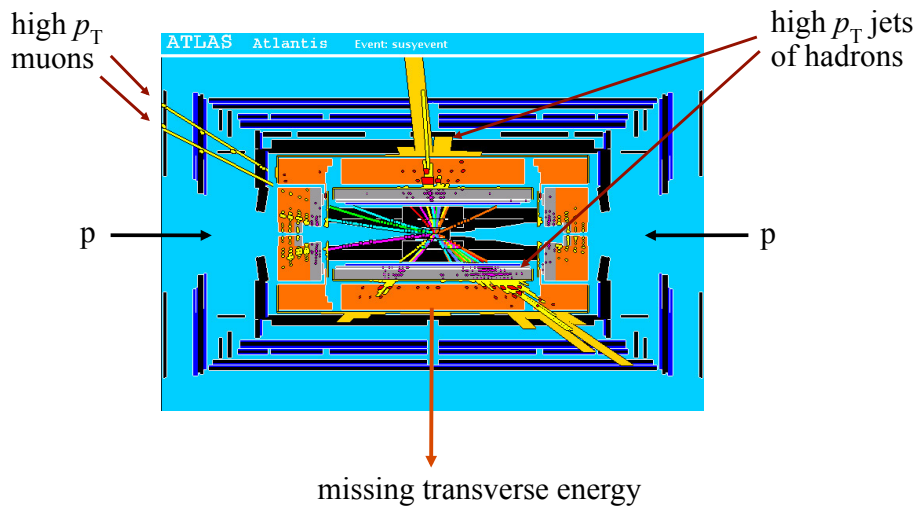
The ATLAS Detector at the LHC

3000 physicists
37 countries
167 universities/labs

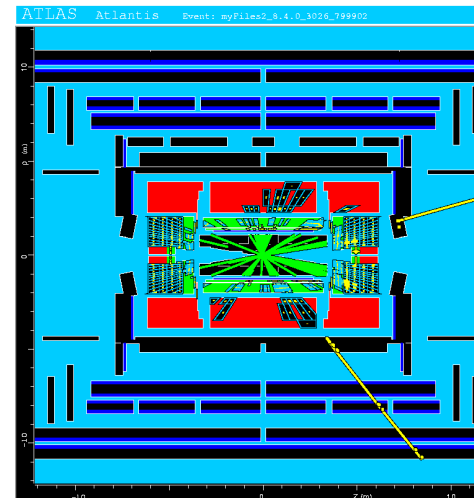


25 m diameter
46 m length
7000 tonnes
~10⁸ electronic channels

A simulated proton-proton collision from supersymmetry (“signal”)



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:

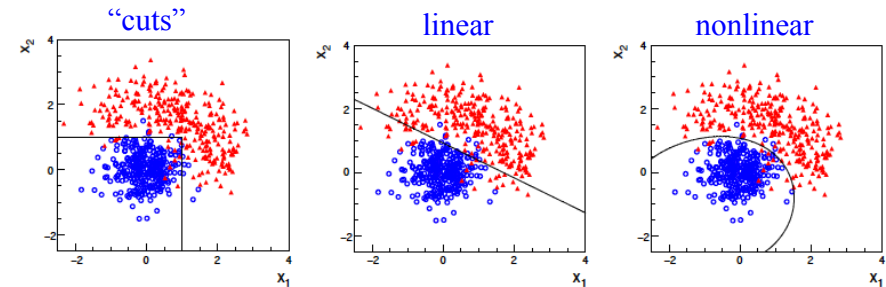
- x_1 = energy of muon
- x_2 = angle between jets
- x_3 = total jet energy
- x_4 = missing transverse energy
- x_5 = invariant mass of muon pair
- x_6 = ...

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 \mathbf{x} and true class label y (0 for background, 1 for signal):

$$(\mathbf{x}, y)_1, (\mathbf{x}, y)_2, \dots, (\mathbf{x}, y)_N$$

The simulated events are called "training data".

What is the best decision boundary?



What is the best decision function?

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

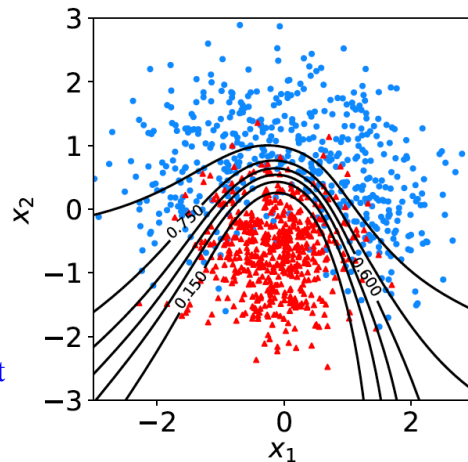
$$t(x_1, \dots, x_n) = t_c$$

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 $t(x)$.



Linear decision boundary

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

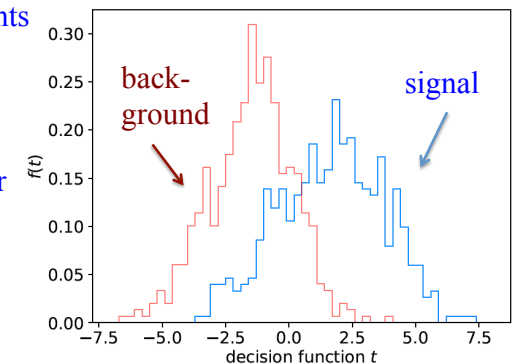
$$t(\mathbf{x}) = \sum_{i=1}^n w_i x_i$$

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

A given choice of the weights fixes the function $t(x)$.

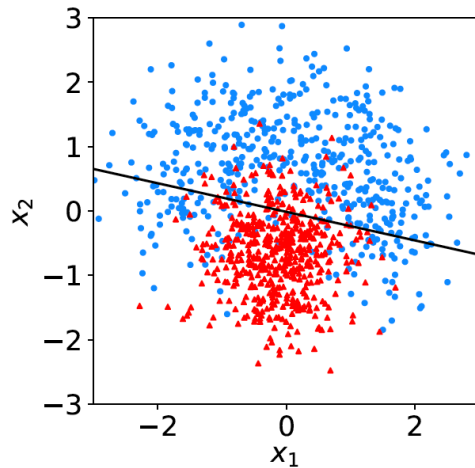
Look at the training events from the two classes, for each evaluate $t(x)$ and enter into a histogram.

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



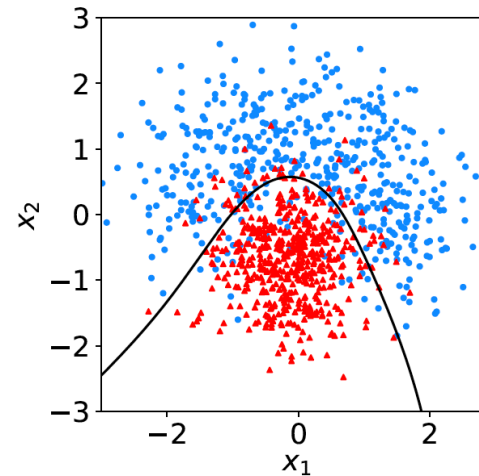
Fisher discriminant

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



Nonlinear decision boundaries

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



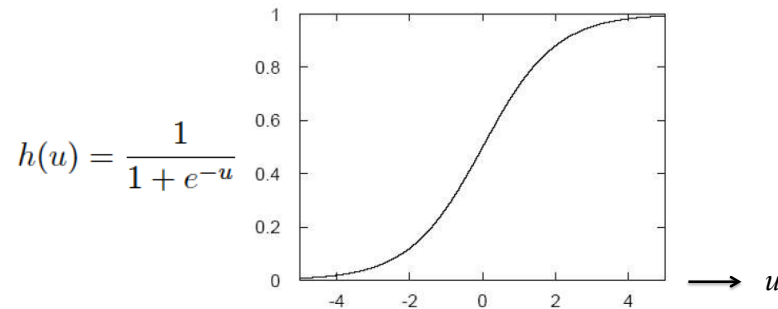
And to have a nonlinear boundary, the decision function $t(\mathbf{x})$ must be nonlinear in \mathbf{x} .

Neural Networks

A simple nonlinear decision function can be constructed as

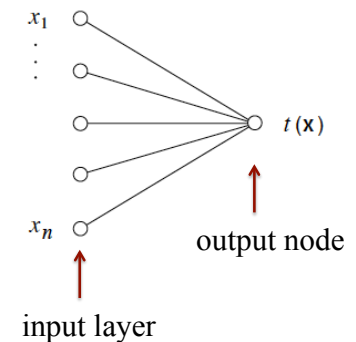
$$t(\mathbf{x}) = h \left(w_0 + \sum_{i=1}^n w_i x_i \right)$$

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



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 $t(\mathbf{x}) = t_c$ 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

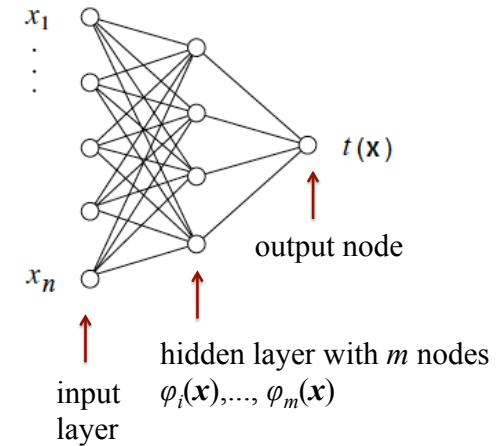
The Single Layer Perceptron can be generalized by defining first a set of functions $\varphi_i(\mathbf{x})$, with $i = 1, \dots, m$:

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

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

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

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 \mathbf{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).

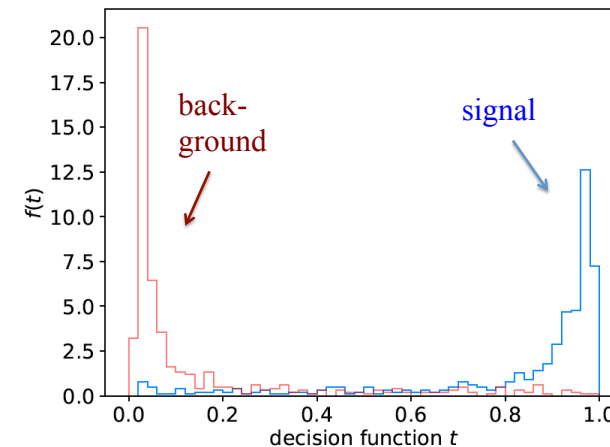
The optimal values of the weights are found by minimizing $E(\mathbf{w})$ with respect to the weights (non-trivial algorithms: backpropagation, stochastic gradient descent,...).

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

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

Distribution of neural net output

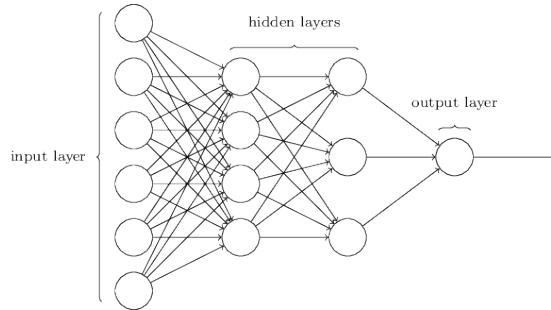
Degree of separation between classes now much better than with linear decision function:



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:

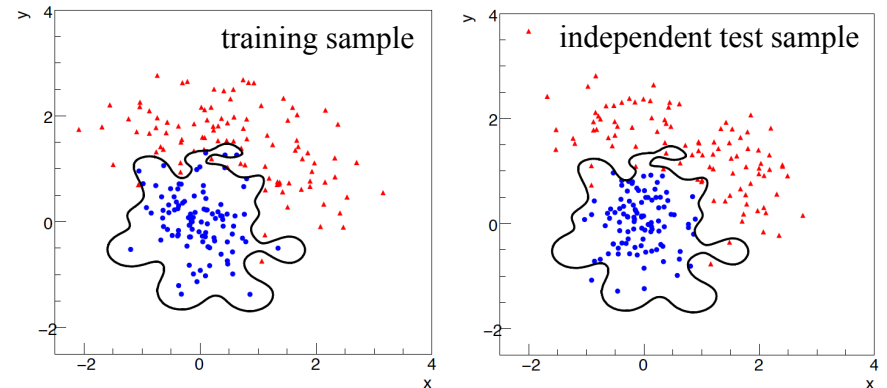


“Deep Learning” is a very recent and active field of research.

Overtraining

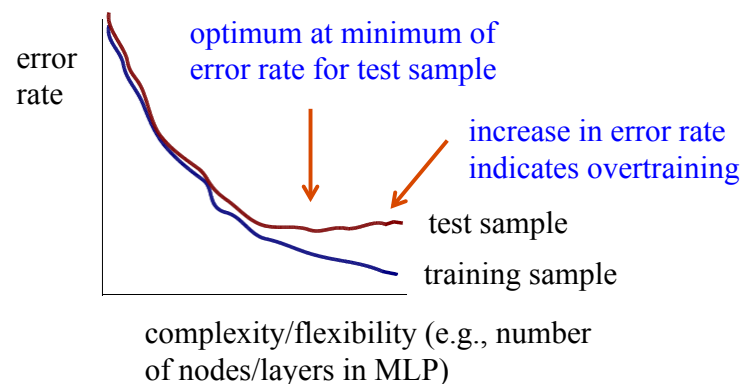
Including more parameters in a classifier makes its decision boundary increasingly flexible, e.g., more nodes/layers for a neural network.

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., error function $E(w)$) 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.

→ face/handwriting recognition, driverless cars...

Machine Learning for handwriting recognition

Initial feature vector = set of pixels of an image

```
[2+2, 5+5, 4+8, 0+0, 2+2, 7+7, 5+5, 1+1,
3+3, 0+0, 3+3, 9+9, 6+6, 2+2, 8+8, 2+2,
0+0, 6+6, 6+6, 1+1, 1+1, 7+7, 8+8, 5+5,
0+0, 4+4, 7+7, 6+6, 0+0, 2+2, 5+5,
3+3, 1+1, 5+5, 6+6, 7+7, 5+5, 4+4, 1+1,
9+9, 3+3, 6+6, 8+8, 0+0, 9+9, 3+3,
0+0, 3+3, 7+7, 4+4, 4+4, 7+7, 8+8, 0+0,
4+4, 1+1, 3+3, 7+7, 6+6, 4+4, 7+7, 2+2,
7+7, 2+2, 5+5, 2+2, 0+0, 9+9, 8+8, 9+9,
8+8, 1+1, 6+6, 4+4, 8+8, 5+5, 8+8,
0+0, 6+6, 7+7, 4+4, 5+5, 8+8, 4+4,
3+3, 1+1, 5+5, 1+1, 9+9, 9+9, 9+9, 2+2,
4+4, 7+7, 3+3, 1+1, 9+9, 2+2, 9+9, 6+6]]
```



[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.,

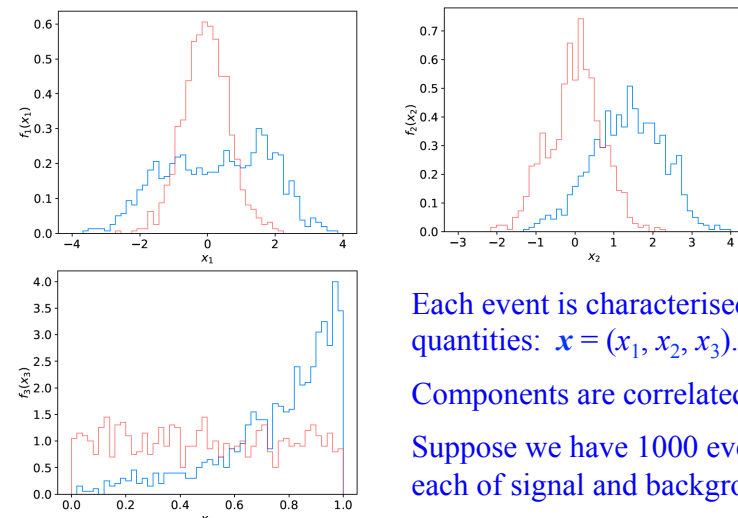
```
from sklearn.neural_network import MLPClassifier
```

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: $\mathbf{x} = (x_1, x_2, x_3)$.

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)[:,: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 any point (x_1, x_2) in the feature space, we can evaluate the decision:

```
xt = np.array([0.37, 2.46]).reshape((1L, 2L)) # 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`.

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://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos/>

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]` ; also 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 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.

Optional: by consulting the documentation and sample program, implement any/all 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.