Queen Mary
University of London

DR ADRIAN BEVAN

PRACTICAL MACHINE LEARNING

LINEAR REGRESSION AND LINEAR DISCRIMINANTS

LECTURE PLAN

- Introduction
- Linear regression
- Fisher Discriminant

QMUL Summer School: https://www.qmul.ac.uk/summer-school/
Practical Machine Learning QMplus Page: https://qmplus.qmul.ac.uk/course/view.php?id=10006

INTRODUCTION

- Machine learning is function approximation
- Some set of parameters are fit to data in order to produce a concrete function definition that is used to approximate the data.
- There are parallels between machine learning training for model fitting and parameter estimation using least squares and likelihood approaches.
  - So we start by looking at these simpler fitting problems, and will focus on least squares linear regression*.
- Linear discriminants don't require machine learning to determine the functional form.

* If people wish to read up on the subject of likelihood fitting a good starting point is the book by Edwards entitled Likelihood.

LINEAR REGRESSION

- Consider the equation:

$$y = f(x)$$
$$mx + c$$

- This describes a straight line where
  - m is the slope of the line
  - c is the constant offset (y value at x=0).
- The problem is how to select the values of m and c in order to obtain the best possible model of the data.
  - To do this we need to make some assumptions.

* If people wish to read up on the subject of likelihood fitting a good starting point is the book by Edwards entitled Likelihood.

LINEAR REGRESSION

- 1) Assume that the data have a linear relationship so that we can describe the relationship between x and y with this model.
- 2) Define some figure of merit that can be optimised in order to determine the values of the parameters m and c.
- Define some method that can be used in order to extract the optimal values of m and c.
  - In scientific applications we also want to know the uncertainty (or error) on m and c.

* If people wish to read up on the subject of likelihood fitting a good starting point is the book by Edwards entitled Likelihood.

LINEAR REGRESSION

- 1) Assume that the data have a linear relationship so that we can describe the relationship between x and y with this model.

  ✓ Let's assume that this function is valid for the problem

- 2) Define some figure of merit that can be optimised in order to determine the values of the parameters m and c.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - \widehat{y}}{\sigma_i} \right)^2$$

y_i = y value for i^{th} example

σ_i = error on y value of i^{th} example

$\hat{y}$ = estimate of y given x using the model

χ^2 = Sum over all examples of the normalised squared residual.

* If people wish to read up on the subject of likelihood fitting a good starting point is the book by Edwards entitled Likelihood.

LINEAR REGRESSION

- 1) Assume that the data have a linear relationship so that we can describe the relationship between x and y with this model.

✓ Let's assume that this function is valid for the problem

- 2) Define some figure of merit that can be optimised in order to determine the values of the parameters m and c.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx + c)}{\sigma_i} \right)^2$$

y_i = y value for i^{th} example

σ_i = error on y value of i^{th} example

m and c are model parameters to be determined

χ^2 = Sum over all examples of the normalised squared residual.

* If people wish to read up on the subject of likelihood fitting a good starting point is the book by Edwards entitled Likelihood.

LINEAR REGRESSION

- 1) Assume that the data have a linear relationship so that we can describe the relationship between x and y with this model.

✓ Let's assume that this function is valid for the problem

- 2) Define some figure of merit that can be optimised in order to determine the values of the parameters m and c.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx + c)}{\sigma_i} \right)^2$$

y_i = y value for i^{th} example

σ_i = error on y value of i^{th} example

m and c are model parameters to be determined

χ^2 = Sum over all examples of the normalised squared residual.

Note - if we make a simplification that the error is similar for all data points, then we can simplify the problem by neglecting σ_i [effectively we set these values to unity].

* If people wish to read up on the subject of likelihood fitting a good starting point is the book by Edwards entitled Likelihood.

LINEAR REGRESSION

- 3) To move forward we need a set of data examples (N pairs of y and x values) to compute the χ^2 sum.
- We also need to be able to systematically vary m and c to optimise this figure of merit:
  - The optimal value of these parameters corresponds to the pair that result in the smallest χ^2 value. This will result in a model that matches the data the best.
  - This does not guarantee that the optimal choice of m and c will result in a good model (overfitting/overtraining will be discussed later in the course).

LINEAR REGRESSION

- 3 contd.) We will use a gradient descent parameter optimisation algorithm (See the optimisation lecture notes later in the course).
  - For now you can treat this optimisation process as a black box.
    - Visualise systematically choosing pairs of m and c, and for each point in this 2D space compute X^2. From the ensemble of points in this hyperspace, one can then select the minimum.
    - Algorithmically this is expensive so we use algorithms that approximate the search for the minimum that is computationally more efficient (and adaptable to higher dimensional parameter spaces).
  - The analytic solution for this problem is given at the end of these slides.

LINEAR REGRESSION

- Illustration of the optimisation process for a 1D problem.
- Take an ensemble of measurements of some quantity S*

i =	1	2	3	4	5	6	7
S	0.662	0.625	0.897	0.614	0.925	0.694	0.601
σ_S	0.039	0.091	0.100	0.160	0.160	0.061	0.239

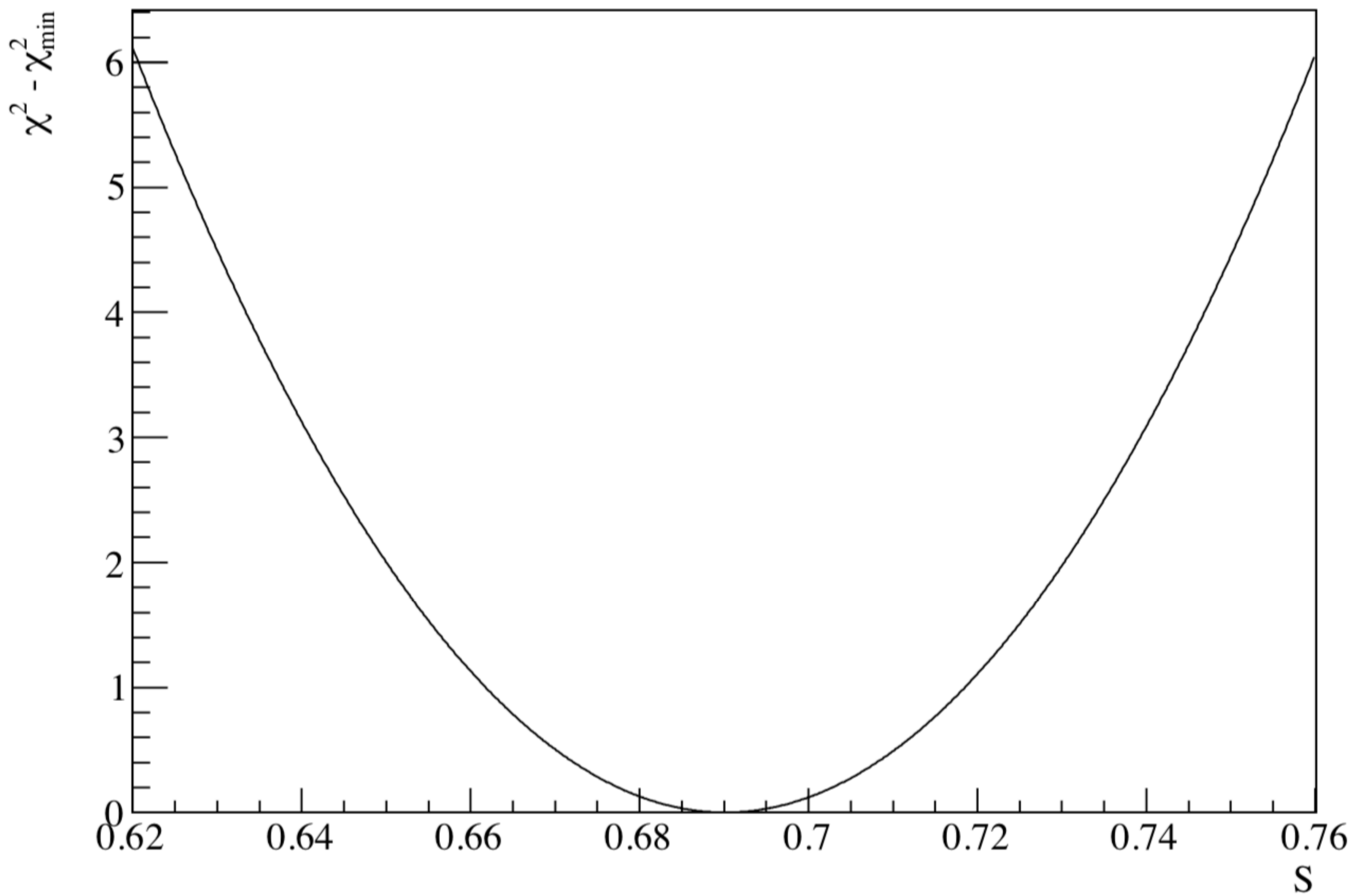
- We want to extract the average value of S from these data, which can be done by scanning through assumed values (over some sensible range) and computing:

$$\chi^2 = \sum_{i=1}^{7} \left(\frac{S_i - S}{\sigma_{S_i}} \right)^2$$

*S is a parameter that is related to matter-antimatter asymmetry in sub-atomic quantum systems. S=sin2β, where β is a manifestation of a phase difference between matter and antimatter decays in certain decays of neutral B mesons. The background behind this measurement is discussed in this Symmetry magazine article and from a technical perspective in this book: The Physics of the B Factories.

LINEAR REGRESSION

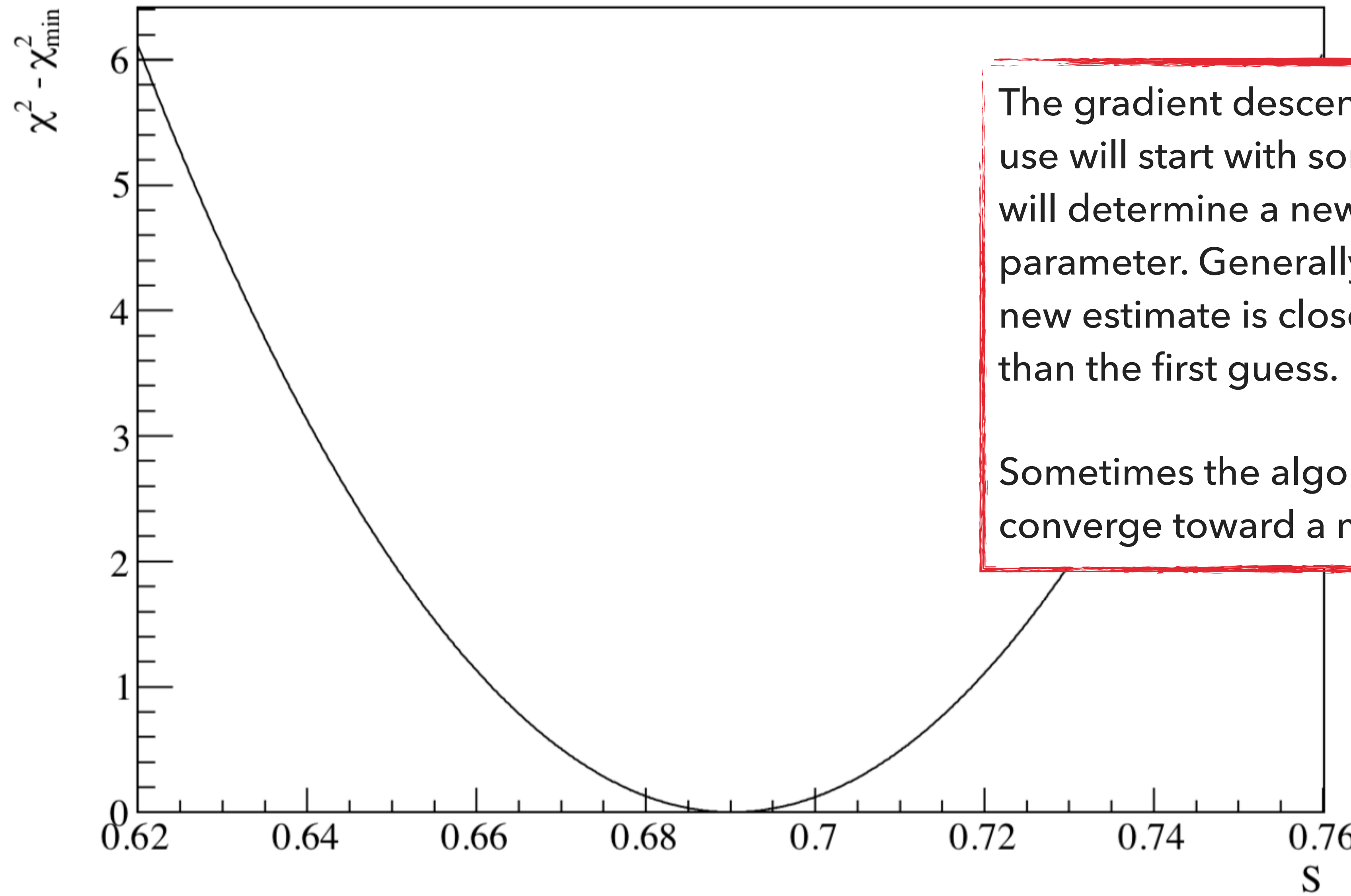
- On doing this we obtain a parabolic curve, where a change in one unit from the minimum corresponds to a change of 1σ (the error) in S.



*S is a parameter that is related to matter-antimatter asymmetry in sub-atomic quantum systems. S=sin2β, where β is a manifestation of a phase difference between matter and antimatter decays in certain decays of neutral B mesons. The background behind this measurement is discussed in this Symmetry magazine article and from a technical perspective in this book: The Physics of the B Factories.

LINEAR REGRESSION

- On doing this we obtain a parabolic curve, where a change in one unit from the minimum corresponds to a change of 1σ (the error) in S.



The gradient descent algorithm we will use will start with some initial value and will determine a new estimate of the parameter. Generally we expect that this new estimate is closer to the minimum than the first guess.

Sometimes the algorithm will fail to converge toward a minimum.

*S is a parameter that is related to matter-antimatter asymmetry in sub-atomic quantum systems. S=sin2β, where β is a manifestation of a phase difference between matter and antimatter decays in certain decays of neutral B mesons. The background behind this measurement is discussed in this Symmetry magazine article and from a technical perspective in this book: The Physics of the B Factories.

LINEAR REGRESSION

- We can read the minimum off of a 1D X^2 scan.
- However for a 2D problem we have to scan through points in a grid, and from that array of results choose the optimal.
  - e.g. see the logarithmic grid search performed by R's libsvm package, which performs a 2D parameter scan.
- For more dimensions than 2 it is computationally expensive to perform a parameter scane, and difficult to visualise this approach.

*S is a parameter that is related to matter-antimatter asymmetry in sub-atomic quantum systems. S=sin2β, where β is a manifestation of a phase difference between matter and antimatter decays in certain decays of neutral B mesons. The background behind this measurement is discussed in this Symmetry magazine article and from a technical perspective in this book: The Physics of the B Factories.

LINEAR REGRESSION

- Example_LinearRegression.py

- We need to select some parameters for the optimisation process:

```
learning_rate    = 0.005
training_epochs  = 10
min_x = 0
max_x = 1
Ngen  = 100
gradient  = 2.0
intercept = 0.5
noise     = 0.1  # data are inherently noisy, so we generate noise
```

learning_rate:

step size for the optimisation process

training_epochs:

number of times the optimisation is run

Ngen:

number of simulated examples to use (=N in the X^2 sum)

gradient and intercept:

the parameters m and c, respectively.

noise:

Data are inherently noisy, this parameter introduces an element of randomness to the value of y for a given x.

LINEAR REGRESSION

- Example_LinearRegression.py

- Implementing linear regression function for optimisation:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx + c)}{\sigma_i} \right)^2$$

```
x_ = tf.placeholder(tf.float32, [None, 1], name="x")
y_ = tf.placeholder(tf.float32, [None, 1], name="y_")

# parameters of the model are m (gradient) and c (constant offset)
#   - pick random starting values for fit convergence
c = tf.Variable(tf.random_uniform([1]), name="c")
m = tf.Variable(tf.random_uniform([1]), name="m")
y = m * x_ + c
```

LINEAR REGRESSION

- `Example_LinearRegression.py`

- Implementing linear regression function for optimisation:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx + c)}{\sigma_i} \right)^2$$

x and y_i are the data for the i^{th} example, represented by the placeholders **x_** and **y_**

```
x_ = tf.placeholder(tf.float32, [None, 1], name="x")
y_ = tf.placeholder(tf.float32, [None, 1], name="y_")

# parameters of the model are m (gradient) and c (constant offset)
#   - pick random starting values for fit convergence
c = tf.Variable(tf.random_uniform([1]), name="c")
m = tf.Variable(tf.random_uniform([1]), name="m")
y = m * x_ + c
```

LINEAR REGRESSION

- `Example_LinearRegression.py`

- Implementing linear regression function for optimisation:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx + c)}{\sigma_i} \right)^2$$

m and c are the model parameters, represented by the Variables **m** and **c**. The variable **y** corresponds to the model that is our estimator of the data given by the placeholder **y_**.

```
x_ = tf.placeholder(tf.float32, [None, 1], name="x")
y_ = tf.placeholder(tf.float32, [None, 1], name="y_")

# parameters of the model are m (gradient) and c (constant offset)
#   - pick random starting values for fit convergence
c = tf.Variable(tf.random_uniform([1]), name="c")
m = tf.Variable(tf.random_uniform([1]), name="m")
y = m * x_ + c
```

LINEAR REGRESSION

- `Example_LinearRegression.py`

- We need to define a loss function that will be optimised. For the problem at hand the loss function is χ^2, where we let $\sigma=1$.

```
# assume all data have equal uncertainties to avoid having to define an example by
# example uncertainty, and define the loss function as a simplified chi^2 sum
loss = tf.reduce_sum((y - y_) * (y - y_))
```

- An optimiser* is required in order to minimise the loss function and determine the “optimal” parameter values for m and c.

```
# use a gradient descent optimiser
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
```

*See the optimisation notes for more details regarding this algorithm.

LINEAR REGRESSION

- `Example_LinearRegression.py`

- The minimisation is performed by running the training step (computing the loss function and updating parameters) the specified number of training epochs:

```
for step in range(training_epochs):
    # run the minimiser
    sess.run(train_step, feed_dict={x_: data_x, y_: data_y})
```

- As the loss function depends on `x_` and `y_` placeholders, we need to feed the data (these are NumPy arrays) to the optimiser `train_step` for each training epoch.

*See the optimisation notes for more details regarding this algorithm.

LINEAR REGRESSION

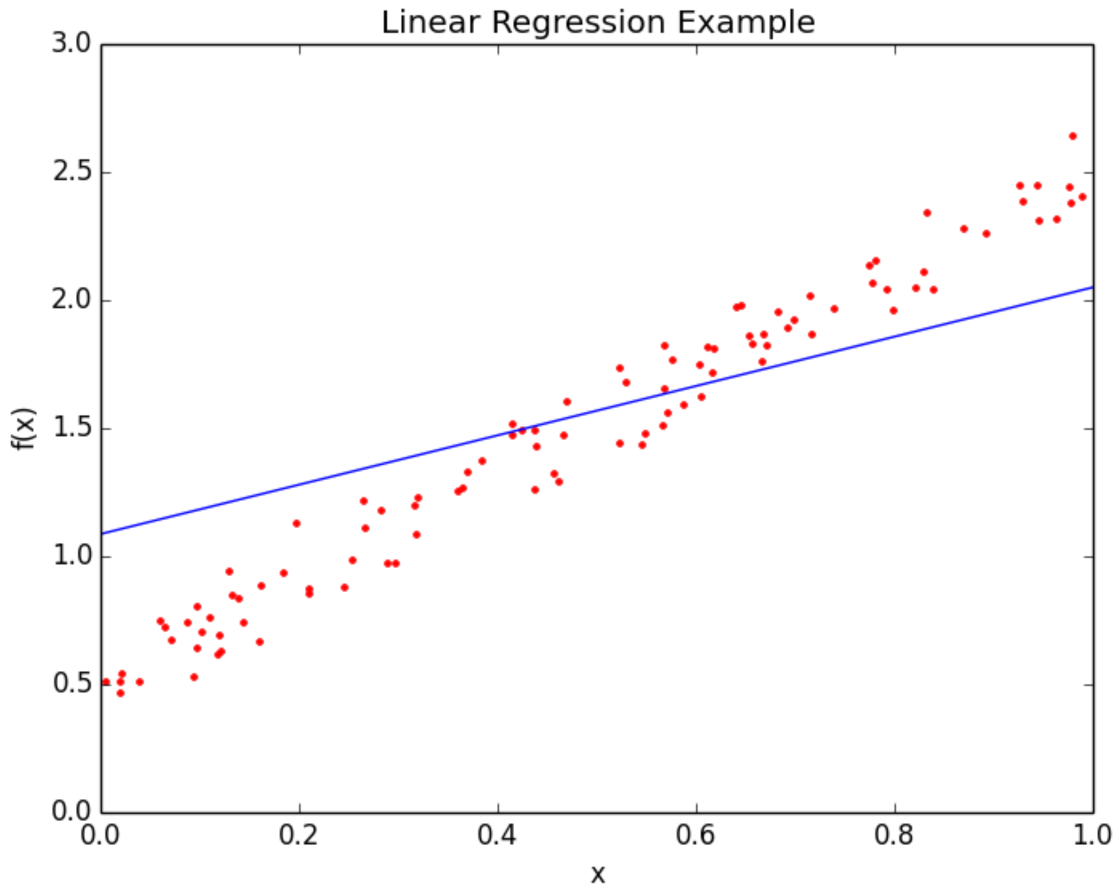
- `Example_LinearRegression.py`

- The output of this script can be seen below:

Input Values:
  m = 2.0
  c = 0.5

Training Parameters*:
  N = 100
  N_{Epochs} = 1
  Learning rate = 0.005

Output Values:
  m = 0.963…
  c = 1.087…



Recall that the parameters m and c are initialised using a random number.

The optimisation algorithm will iteratively search through the (m, c) space to determine the optimal set of parameters.

The solution found depends on:
- the starting point
- learning rate
- number of training epochs

* These will be explained in more detail when we discuss optimisation.

LINEAR REGRESSION

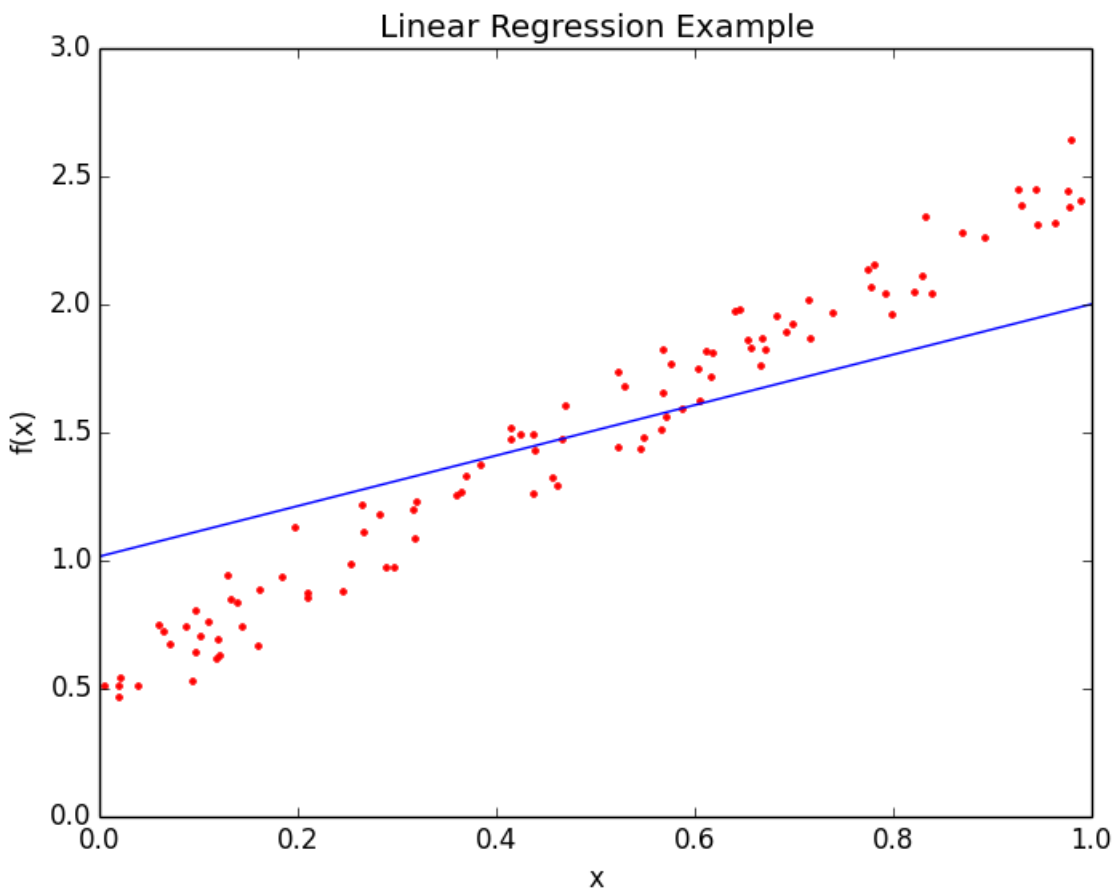
- `Example_LinearRegression.py`

- The output of this script can be seen below:

Input Values:
m = 2.0
c = 0.5

Training Parameters*:
N = 100
N_{Epochs} = 2
Learning rate = 0.005

Output Values:
m = 0.985 …
c = 1.016 …



* These will be explained in more detail when we discuss optimisation.

LINEAR REGRESSION

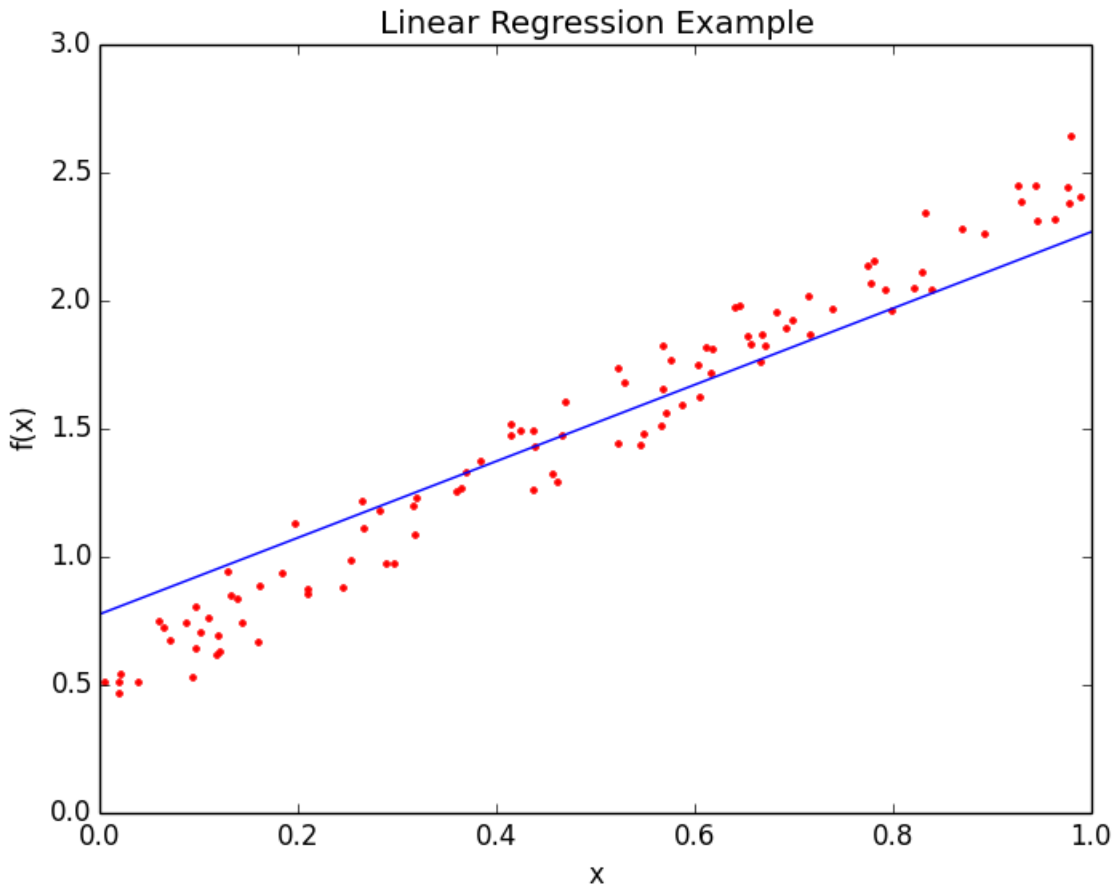
- `Example_LinearRegression.py`

- The output of this script can be seen below:

Input Values:
 m = 2.0
 c = 0.5

Training Parameters*:
 N = 100
 N_{Epochs} = 10
 Learning rate = 0.005

Output Values:
 m = 1.493…
 c = 0.776…



* These will be explained in more detail when we discuss optimisation.

LINEAR REGRESSION

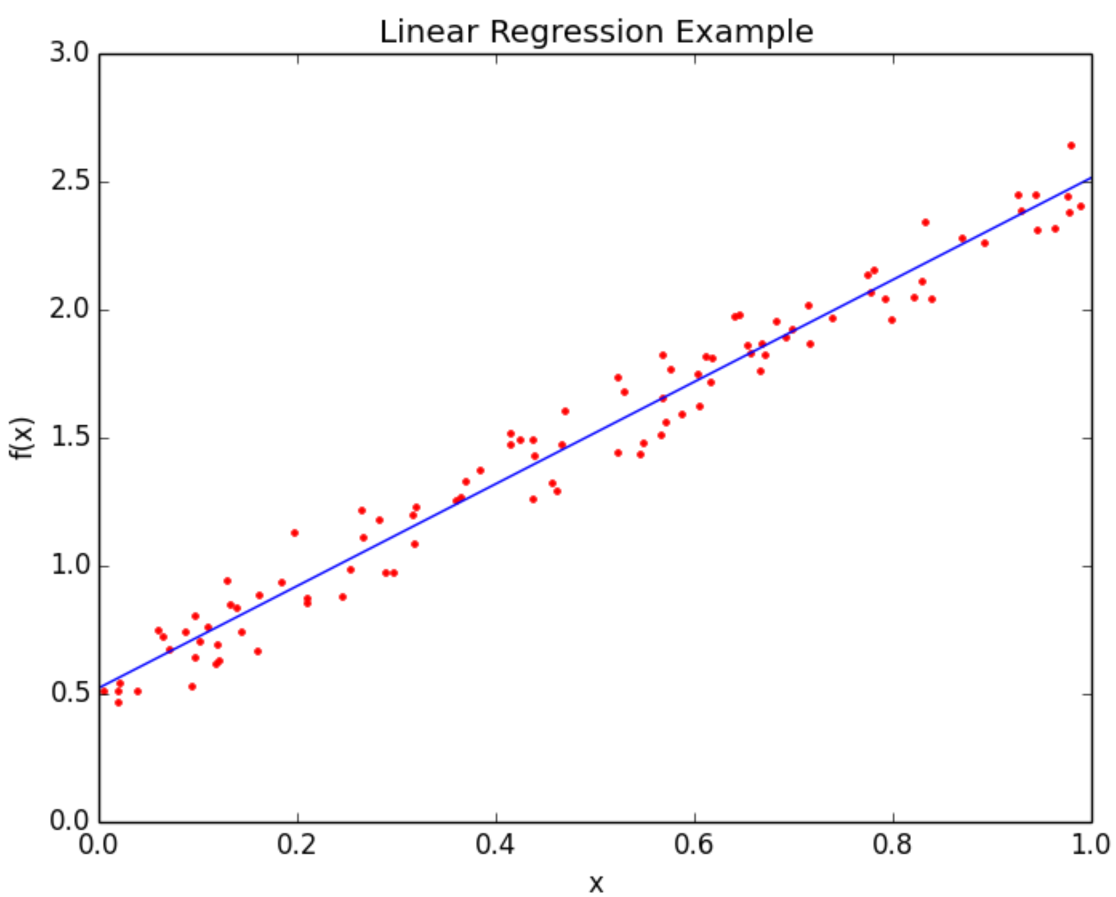
- `Example_LinearRegression.py`

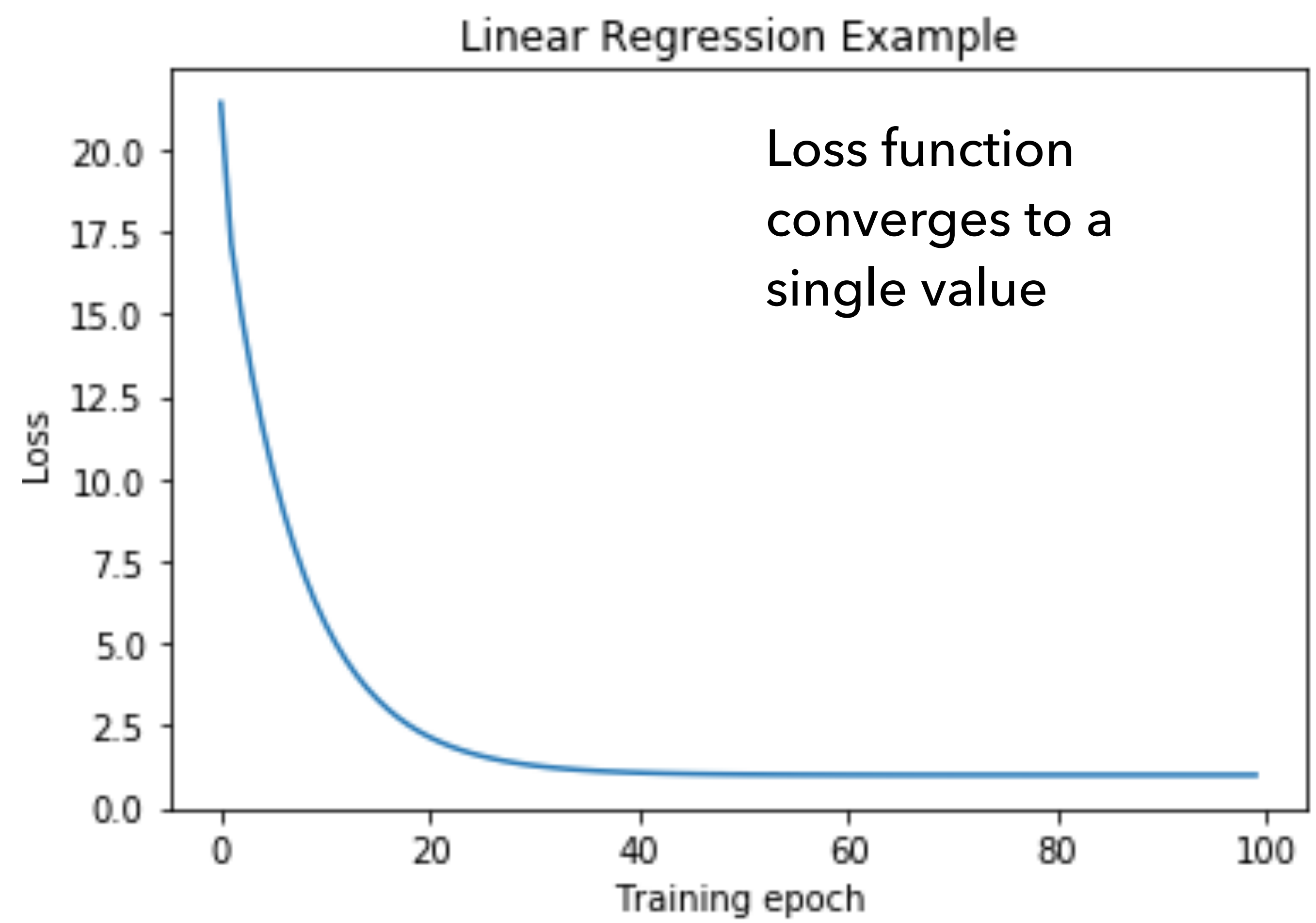
- The output of this script can be seen below:

Input Values:
m = 2.0
c = 0.5

Training Parameters*:
N = 100
N_{Epochs} = 100
Learning rate = 0.005

Output Values:
m = 1.993…
c = 0.523…





* These will be explained in more detail when we discuss optimisation.

LINEAR REGRESSION

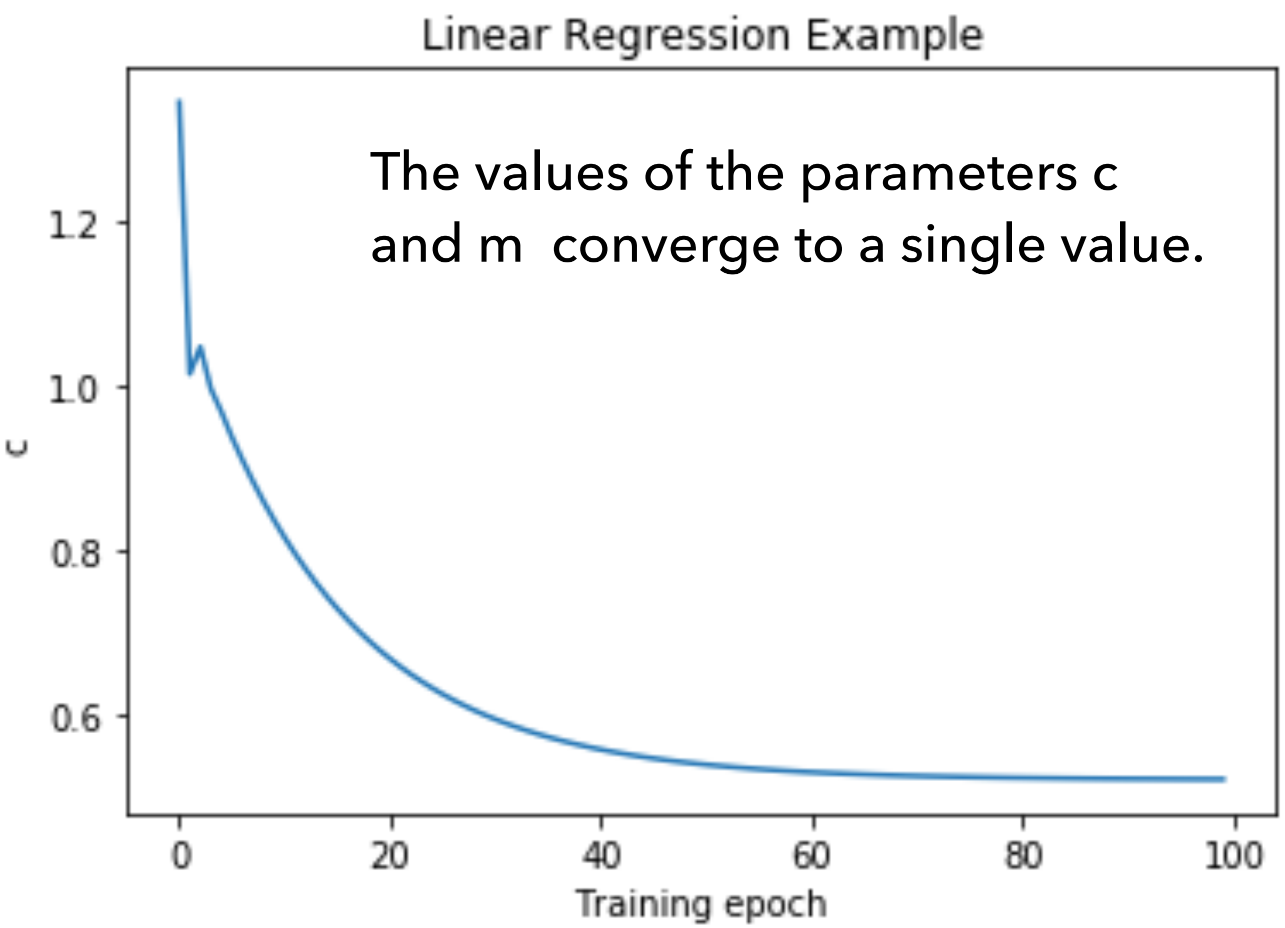
- `Example_LinearRegression.py`

- The output of this script can be seen below:

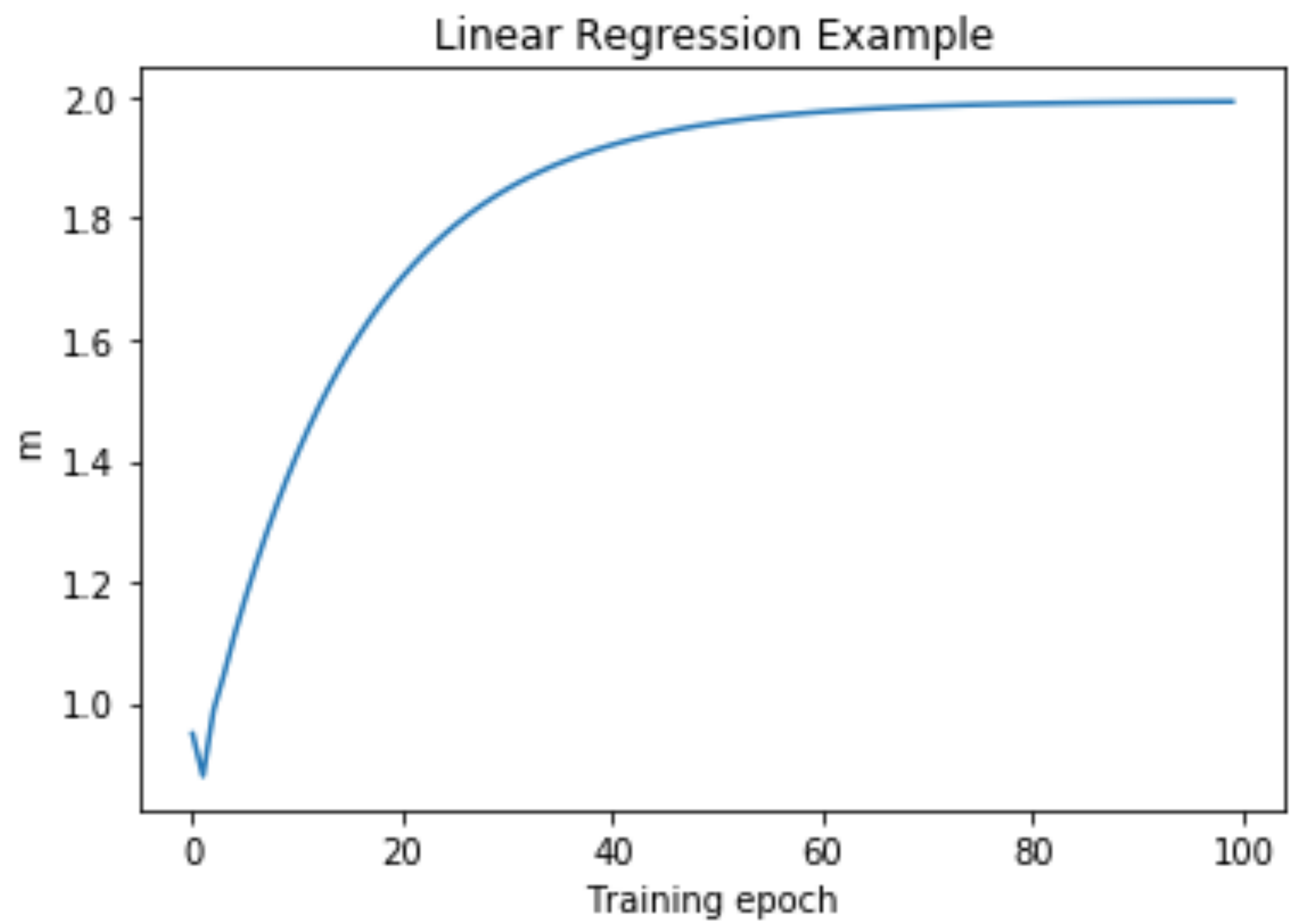
Input Values:
m = 2.0
c = 0.5

Training Parameters*:
N = 100
N_{Epochs} = 100
Learning rate = 0.005

Output Values:
m = 1.993…
c = 0.523…



The values of the parameters c and m converge to a single value.



* These will be explained in more detail when we discuss optimisation.

LINEAR REGRESSION

- **Summary**:
  - The linear regression problem discussed here uses a loss function that is based on the square residuals of data vs some prediction.
  - This allows us to model a simple linear relationship between some input x and some output y, where the functional form is just:
    - $y = mx+c$
  - where m and c are parameters to determine.
- If we addressed the issue of uncertainties in the data, we could also extract uncertainties on the optimal values of m and c obtained, and this would be fitting.
- This approach and will be generalised when we consider neural networks and extensions to non-linear problems that can not be solved analytically.
  - For machine learning problems we don't care about the uncertainty on the optimal parameters determined(*).

(*) Bayesian networks do allow for a probability distribution for weights, and so this remark is really method dependent. That detail is beyond the scope of this course.

FISHER DISCRIMINANT

- This is an analytic algorithm that was inspired by the classification problem for species of iris in the 1930's[1].
- Starting point is the assumption that data are distributed according to a multi-Gaussian probability (e.g. random sampling of data), and that one wishes to maximise the separation between different classes (types) of iris.
  - Maximise the separation of the mean distributions.
  - Minimise the sum of the covariances.

[1] R. A. Fisher, Ann. Eug., 7, 179188 (1936).

FISHER DISCRIMINANT

- The Fisher discriminant is given by

$$F = \alpha^T x + \beta$$

$$\alpha = W^{-1}(\mu_A - \mu_B)$$

  - α: a vector of weight parameters
  - x: data, with the shape [Nexample, dim(example)]
  - W: Sum of covariance matrices for classes A and B
  - $\mu_{A,B}$: Mean value of class A or B

NOTE: I like this algorithm as an example as it can help us understand the data and how to separate classes before looking in more detail at a neural network as the underlying equations appear later in the course.

It is also a simple algorithm that can be used as a benchmark to check that a more sophisticated algorithm performs at least as well as this one. Such a sanity check can be useful in low dimensional physics problems and it may or may not be useful for you machine learning work in the future.

[1] R. A. Fisher, Ann. Eug., 7, 179188 (1936).

FISHER DISCRIMINANT

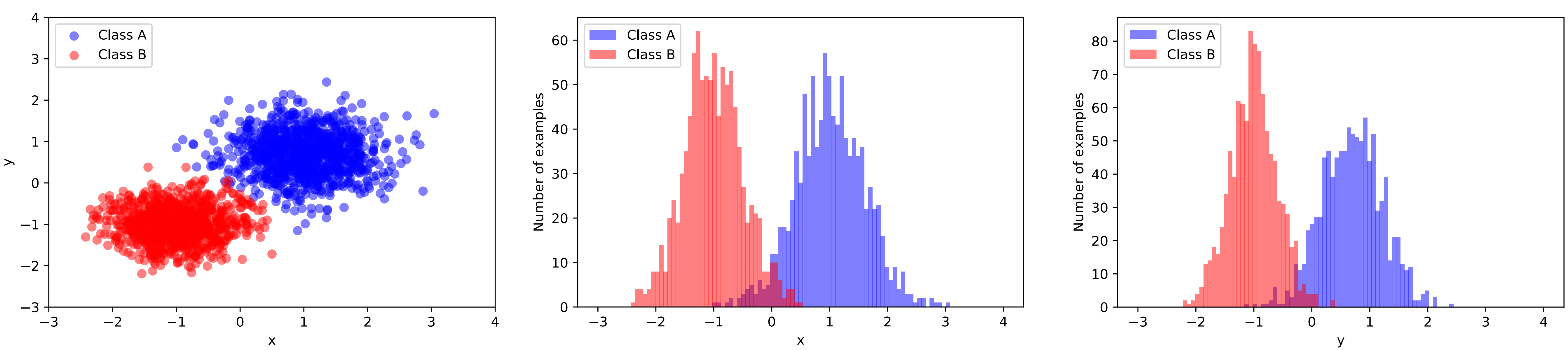
- 3 scripts are required for this example:
  - `Example_Fisher.py`
  - `Fisher.py`
  - `PracticalMachineLearning.py`

- The first of these three is the one that a user needs to work with, the others provide an implementation of the algorithm.
  - Fisher.py is a class that implements the computation of α and processing of the data according to the equation for $\mathcal{F}$.
  - PracticalMachineLearning.py includes a number of helper functions for this course.

[1] R. A. Fisher, Ann. Eug., 7, 179188 (1936).

FISHER DISCRIMINANT

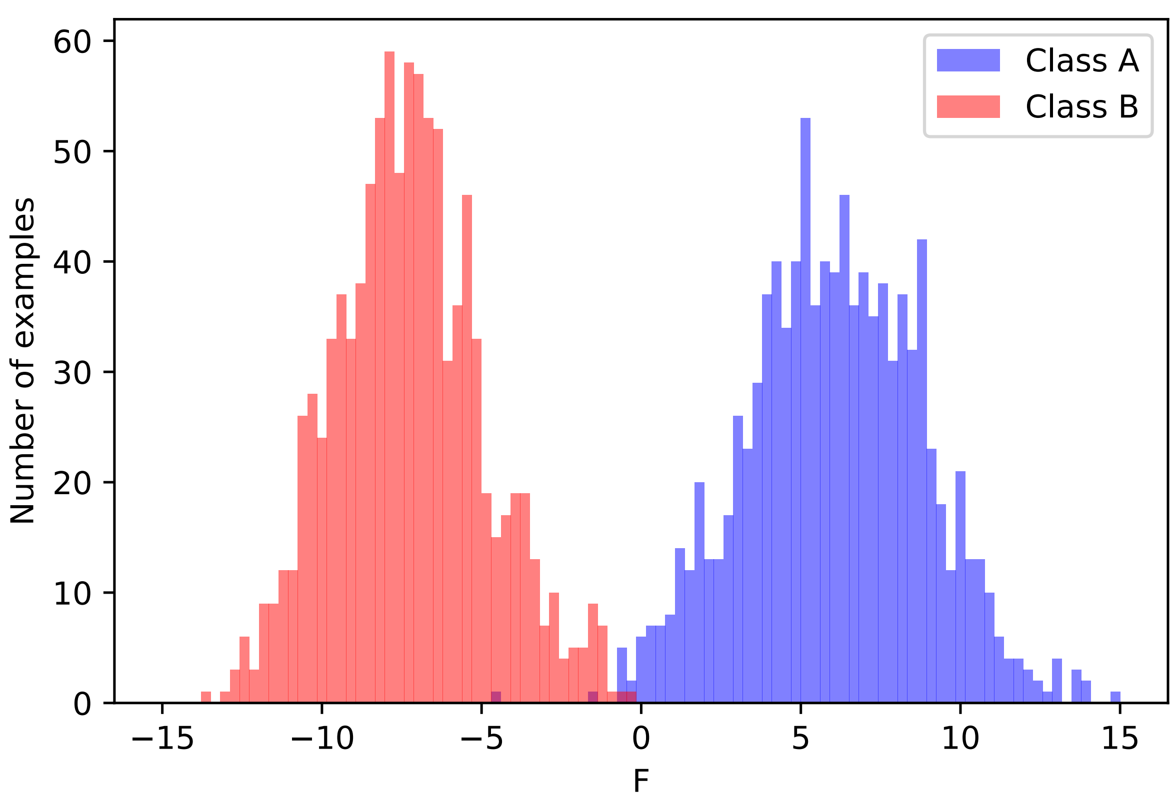
- Consider the problem where we have two classes of event. Some events of type A (signal) and some events of type B (background).
- These events are described by a 2D feature space, consisting of the dimensions x and y.
- We want to compute $\mathcal{F}$ in order for us to be able to distinguish between types A and B.



[1] R. A. Fisher, Ann. Eug., 7, 179188 (1936).

FISHER DISCRIMINANT

- Consider the problem where we have two classes of event. Some events of type A (signal) and some events of type B (background).
- These events are described by a 2D feature space, consisting of the dimensions x and y.
- We want to compute $\mathcal{F}$ in order for us to be able to distinguish between types A and B.



$$F = \alpha^T x + \beta$$

- For this example we can see that $\mathcal{F}$ is a rotated axis in the (x, y) plane along which we can project the data in order to obtain a smaller overlap than in either of the individual features (x or y)
- The offset β is arbitrary and is ignored in this calculation as it just changes the position of an example on the $\mathcal{F}$ axis without changing the separation between the classes.

[1] R. A. Fisher, Ann. Eug., 7, 179188 (1936).

SUMMARY

- We have discussed least squares regression as a simple numerical optimisation problem.
  - This has a large number of applications scientifically, but is limited. We will explore neural networks as a generalisation to this algorithm.
- We have also discussed Fisher discriminants as an algorithmic way to increase separation between two classes of events by mapping an N dimensional input feature space into a 1 dimensional output feature space.

SUGGESTED READING

- C. Bishop: *Neural Networks for Pattern Recognition*
  - Chapters: 2
- C. Bishop: *Pattern Recognition and Machine Learning*
  - Chapters: 4
- T. Hastie, R. Tibshirani, J. Friedman, *Elements of statistical learning*
  - Chapter: 4

APPENDIX: LINEAR REGRESSION

- We can return to the linear regression problem and solve this analytically. The following notes follow the convention that the slope of the line m=a, and that the constant offset c=b.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - ax_i - b}{\sigma(y_i)} \right)^2 = \frac{1}{\sigma^2} \sum_{i=1}^{N} (y_i - ax_i - b)^2$$

- This is minimised* when

$$\frac{\partial \chi^2}{\partial a} = 0, \text{ and } \frac{\partial \chi^2}{\partial b} = 0.$$

* For this parabolic problem there is a single turning point and that is a minimum. For arbitrary functions we would need to use the second derivative to distinguish between maxima, minima and points of inflection.

APPENDIX: LINEAR REGRESSION

- The derivatives of χ^2 with respect to a and b are:

$$\begin{aligned}
\sigma^2 \frac{\partial \chi^2}{\partial a} &= \sum_{i=1}^{N} \frac{\partial}{\partial a} (y_i - a x_i - b)^2, \\
&= \sum_{i=1}^{N} -2x_i (y_i - a x_i - b), \\
&= -2 \sum_{i=1}^{N} x_i y_i - a x_i^2 - b x_i, \\
&= -2N(\overline{xy} - a\overline{x^2} - b\overline{x}).
\end{aligned}$$

$$\sigma^2 \frac{\partial \chi^2}{\partial b} = -2N(\overline{y} - a\overline{x} - b)$$

If we assume that the uncertainties are equal for all i, then this becomes a constant factor that drops out of the problem.

The bar over x, y, xy, etc. denotes the average value of that quantity.

- Leading to the two simultaneous equations:

$$\begin{aligned}
\overline{xy} - a\overline{x^2} - b\overline{x} &= 0 \\
\overline{y} - a\overline{x} - b &= 0
\end{aligned}$$

APPENDIX: LINEAR REGRESSION

- From these we can solve for a and b, giving:

$$a = \frac{\overline{xy} - b\overline{x}}{\overline{x^2}} = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}$$

$$b = \overline{y} - a\overline{x}.$$

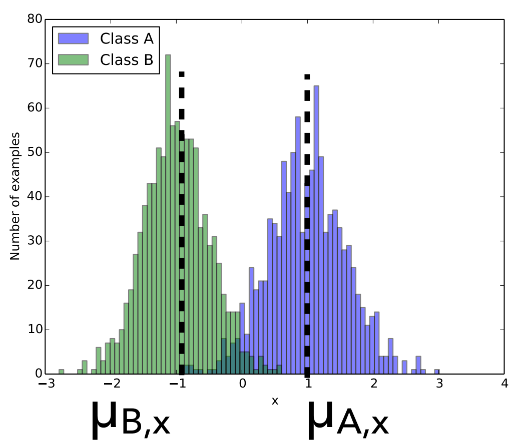
- We can also consider the uncertainty on a and b from the above equations, and considering error propagation (noting the error σ is on y and not x) we obtain:

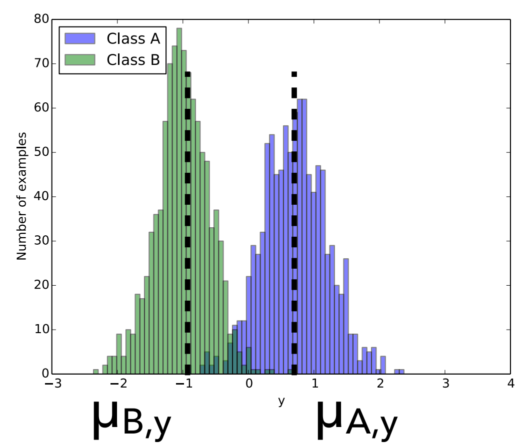
$$\sigma^2(a) = \frac{\sigma^2}{N(\overline{x^2} - \overline{x}^2)}$$

$$\sigma^2(b) = \frac{\sigma^2\overline{x^2}}{N(\overline{x^2} - \overline{x}^2)}$$

APPENDIX: FISHER DISCRIMINANT

- Consider this 2D problem:





$\mu_{A,B}$ are the mean values of A and B, respectively, given by

$$\mu_{A,B;u} = \frac{1}{N}\sum_{i=1}^{N} u_i, \quad u = x, y$$

i.e. $(\mu_{B,y})^T = (\mu_{B,x}, \mu_{B,y})$ and $(\mu_{A,y})^T = (\mu_{A,x}, \mu_{A,y})$.

- The means (μ) and standard deviations (σ) describe the distribution of data; where $\sigma_{A,B}$ are 2D covariance matrices.

- We can compute the mean (M) and variance (Σ) of the Fisher distribution using

$$M_{A,B} = \alpha^T \mu_{A,B} = \sum_i \alpha_i \mu_{A,B},$$

$$\Sigma^2_{A,B} = \alpha^T \sigma^2_{A,B} \alpha = \sum_i \sum_j \alpha_i \sigma_{ij\ A,B} \alpha_j$$

APPENDIX: FISHER DISCRIMINANT

- Optimise J, where

$$J(\alpha) = \frac{[M_A - M_B]^2}{\Sigma_A^2 + \Sigma_B^2}$$

$$\begin{aligned} [M_A - M_B]^2 &= \left[\sum_{i=1}^{n} \alpha_i(\mu_A - \mu_B)_i\right]\left[\sum_{j=1}^{n} \alpha_j(\mu_A - \mu_B)_j\right] \\ &= \sum_{i,j=1}^{n} \alpha_i(\mu_A - \mu_B)_i(\mu_A - \mu_B)_j\alpha_j, \\ &= \alpha^T B\alpha, \end{aligned}$$

$$\begin{aligned} \Sigma_A^2 + \Sigma_B^2 &= \alpha^T \sigma_A^2 \alpha + \alpha^T \sigma_B^2 \alpha, \\ &= \alpha^T W\alpha, \end{aligned}$$

- Thus:

$$J(\alpha) = \frac{\alpha^T B\alpha}{\alpha^T W\alpha}$$ which is optimised for $$\frac{\partial J(\alpha)}{\partial \alpha} = 0$$

- resulting in: $\alpha \propto W^{-1}(\underline{\mu}_A - \underline{\mu}_B)$.
    - The α are given up to an arbitrary scale factor.
    - The mean value of $\mathcal{F}$ can be offset by β arbitrarily.