Week 7

Data Science Laboratory 1

Dr John Evans
`j.evans8@herts.ac.uk`

University of Hertfordshire UH

Plan for today

Normalisation and Standardisation

- Minmax
- Logarithmic Scaling
- Z-normalisation
- Unit Vector Normalisation
- Decimal Scaling

Fitting data with a model

- Goodness of Fit

Optimisation Methods

- Least Squares
- Maximum Likelihood Estimation

Normalisation

- Normalisation in data science is simply the process of transforming the values of a given dataset (e.g. a column from a spreadsheet) to a new set of values.
- **Question:** Why normalise data in the first place?

Normalisation

- Normalisation in data science is simply the process of transforming the values of a given dataset (e.g. a column from a spreadsheet) to a new set of values.
- **Question:** Why normalise data in the first place?
- The most common reason is to make the data easier to handle, particularly in machine learning applications.
- For example, we could put different samples on the same scale, making comparisons between them easier, whilst retaining the relative differences between individual values.
- Perhaps the most common form of normalisation is in the display of images. A two dimensional array of values can be stored as an 8-bit greyscale image, which means that each pixel takes a value between 0 (white) and 255 (black, since $2^8 = 256$). Therefore, each value in the input array is normalised to lie between these two extremes, often on a linear scale, but sometimes not.

You must take care

- When making any such changes, care must obviously be taken.
- For example, changing measurement units from metres to inches for height, or from kilograms to pounds for weight, may lead to very different results. bijective
- For an extreme example of this, read about the Mars Climate Orbiter. This is a lesson for ensuring all measurement data in a data set is written in the same unit.

You must take care

- When making any such changes, care must obviously be taken.
- For example, changing measurement units from metres to inches for height, or from kilograms to pounds for weight, may lead to very different results.
- For an extreme example of this, read about the Mars Climate Orbiter. This is a lesson for ensuring all measurement data in a data set is written in the same unit.
- In general, expressing an attribute in smaller units will lead to a larger range for that attribute, and thus tend to give such an attribute greater effect, or ‘weight’.
- To help avoid dependence on the choice of measurements, the data should be *normalised* (or *standardised*). This involves transforming the data to fall within a common range (usually $[0, 1]$, or sometimes $[-n, n]$).
- In this way, normalising attempts to give all attributes equal weight. It is particularly useful for classification algorithms involving neural networks or distance measurements such as nearest-neighbour classification and clustering.

Minmax

- The simplest method of normalisation is called *minmax*, which is a linear transformation on the original data.
- Suppose that min_A and max_A are the minimum and maximum values of our numerical attribute (call it A).
- Then min-max normalisation maps a value v_i of A to v_i' in the range $[new\_min_A, new\_max_A]$ by computing,

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A.$$

- Note that this will preserve the relationships among the original data values and will encounter an ‘out-of-bounds’ error if a future input case for normalisation falls outside of the original data range for A.
- It is typical for the new range to be $[0, 1]$.

Example

- In Python, to get our value to fall between 0 and 1, we do the following:

```
X_norm = (X - np.min(X)) / (np.max(X) - np.min(X))
```

Example

- In Python, to get our value to fall between 0 and 1, we do the following:

```
X_norm = (X - np.min(X)) / (np.max(X) - np.min(X))
```

- Suppose $A = \textit{income}$ and the values we have are (in order),

$$£14,000,\ £22,500,\ £33,700,\ £40,900,\ £51,000,\ £124,000.$$

  Now suppose we want to map *income* to the range $[0,\ 1]$.

- Then, for each v_i, we compute

$$v_i' = \frac{v_i - 14000}{110000} \times 1 + 0.$$

Results summarised

Old Values	New Values
14000	0
22500	0.077
33700	0.179
40900	0.245
51000	0.336
124000	1

Logarithmic scaling

- If our data has a huge dynamic range, with very large and/or very small numbers, we can take the logarithm (normally base 10) of the values. This compresses the dynamic range and can make the data easier to deal with (and certainly visualise – that is why we often see graphs with ‘log’ axes). For example:

x	$\log_{10}(x)$
0.0001	-4
0.001	-3
0.01	-2
0.1	-1
1	0
10	1
100	2
1000	3
10000	4

invertible operation

log e = ln
e^(ln(x)) = x
ln(e^x) = x

Z-normalisation

- In *Z*-normalisation (also called zero-mean normalisation) we scale based on the mean and standard deviation, i.e. the same as the *Z*-score last week.
- A value v_i is normalised to v_i' by computing

$$v_i' = \frac{v_i - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation, respectively.

Z-normalisation

- In Z-normalisation (also called zero-mean normalisation) we scale based on the mean and standard deviation, i.e. the same as the Z-score last week.
- A value v_i is normalised to v_i' by computing

sample standard deviation (n-1)

$$v_i' = \frac{v_i - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation, respectively.

- Note that this will not be in the range [0, 1]. If v_i is below the mean, we will get a negative number, and if v_i is above the mean, we get a positive number.
- If the unnormalised data had a large standard deviation, the normalised values will be closer to 0.
- This method of normalisation is useful when the actual minimum and maximum of A are unknown, or when there are outliers that dominate the min-max normalisation.

Example

Use the same values as in the earlier example. Then,

$$\mu = \frac{1}{6}(14000 + 22500 + 33700 + 40900 + 51000 + 124000) = 47683.33$$

and

$$\sigma = \sqrt{\frac{1}{5}(7845548333)} = 39611.99.$$

This means that

$$v_i' = \frac{v_i - 47683.33}{39611.99}.$$

Example (results)

Old Values	New Values
14000	-0.85
22500	-0.64
33700	-0.35
40900	-0.17
51000	0.08
124000	1.92

A variation

- In Python, this is as follows: `Z = (X - np.mean(X)) / np.std(X)`

University of Hertfordshire UH

A variation

- In Python, this is as follows: `Z = (X - np.mean(X)) / np.std(X)`
- A variation of this normalisation method replaces the standard deviation by the mean absolute deviation of A. This is defined as,

$$s_A = \frac{1}{n}(|v_1 - \mu| + |v_2 - \mu| + \cdots + |v_n - \mu|).$$

- Everything else is the same, i.e.

$$v_i' = \frac{v_i - \mu}{s_A}.$$

- The mean absolute deviation, s_A is more robust to outliers than the standard deviation since we are taking absolute values instead of squaring. It is left as an exercise to apply this to the above example.

Unit vector normalisation

- For a vector **v** we can normalise to the 'unit vector' simply

$$\mathbf{v}' = \frac{\mathbf{v}}{|\mathbf{v}|} \tag{1}$$

- The way to think of this is easiest in 3D space (although of course your vector could have many dimensions). If **v** is an arrow pointing somewhere in 3D space, the length of the vector is described by $|\mathbf{v}|$. Therefore, if we divide by $|\mathbf{v}|$, we retain the direction of the vector, but force its length to one unit.

Unit vector normalisation

- For a vector $\mathbf{v}$ we can normalise to the ‘unit vector’ simply

$$\mathbf{v}' = \frac{\mathbf{v}}{|\mathbf{v}|} \tag{1}$$

- The way to think of this is easiest in 3D space (although of course your vector could have many dimensions). If $\mathbf{v}$ is an arrow pointing somewhere in 3D space, the length of the vector is described by $|\mathbf{v}|$. Therefore, if we divide by $|\mathbf{v}|$, we retain the direction of the vector, but force its length to one unit.
- In Python we can achieve that as follows: `v_norm = np.sqrt(v.dot(v))`
- Assuming `v` is a Numpy array, allowing us to use the `v.dot()` method. This works because $|\mathbf{v}| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$.

Decimal scaling

- Normalisation by decimal scaling normalises by moving the decimal point of values of our attribute A.
- The number of decimal points moved depends on the maximum absolute value of A.
- A value v_i is normalised to v_i' by computing

$$v_i' = \frac{v_i}{10^j},$$

where j is the smallest integer such that $max(|v_i'|) < 1$.

Example

Again, return to the values of the earlier two examples. In this case, $j = 6$ since that is the smallest j such that $124000/10^j < 1$. The table will be as follows:

Old Values	New Values
14000	0.014
22500	0.0225
33700	0.0337
40900	0.0409
51000	0.051
124000	0.124

Fitting data with a model

- Often, we will want to understand data by fitting it to a model.
- The idea is that the model represents the underlying process that gives rise to the observations.
- The model is often some simple mathematical function or combination of functions, and in general will have a number of parameters.
- For example, in the case of a simple linear relationship between two variables, we have two parameters describing the gradient and intercept of a straight line $y = mx + c$. Here, m and c are ‘free parameters’.

Goodness of fit

- Before we look at how to construct a model, suppose we have already found a suitable model. We then have to decide if this model is a good one.

Goodness of fit

- Before we look at how to construct a model, suppose we have already found a suitable model. We then have to decide if this model is a good one.
- A common way to answer this question is by calculating the χ^2 (chi-square) value:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{O_i - E_i}{\sigma_i} \right)^2 .$$

- This represents a sum over the squared difference between the observation O_i and the expected value (given the model) E_i for a set of N observations, where σ_i represents the uncertainty on O_i.

Goodness of fit

- Before we look at how to construct a model, suppose we have already found a suitable model. We then have to decide if this model is a good one.
- A common way to answer this question is by calculating the χ^2 (chi-square) value:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{O_i - E_i}{\sigma_i} \right)^2 .$$

- This represents a sum over the squared difference between the observation O_i and the expected value (given the model) E_i for a set of N observations, where σ_i represents the uncertainty on O_i.
- Note that in a counting experiment, O_i will be driven by Poisson statistics, and recall that for a Poisson distribution the variance is equal to the expected value:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{(O_i - E_i)^2}{E_i} \right) .$$

How this helps us

- Having calculated χ^2, we can use its probability distribution to assess ‘how good’ the fit is:

$$P(\chi^2) \propto (\chi^2)^{\frac{\nu-2}{2}} e^{-\chi^2/2}.$$

- Here, ν is the number of degrees of freedom in the model, which is $\nu = N - p$, where p is the number of parameters in the model.

How this helps us

- Having calculated χ^2, we can use its probability distribution to assess 'how good' the fit is:

$$P(\chi^2) \propto (\chi^2)^{\frac{\nu-2}{2}} e^{-\chi^2/2}.$$

- Here, ν is the number of degrees of freedom in the model, which is $\nu = N - p$, where p is the number of parameters in the model.
- In general, it is important not to try to fit a model where $p > N$. While a negative degree of freedom is valid, it suggests that we have more statistics than we have values that can change. This can lead to a risk of overfitting the training dataset.
- The probability distribution has the following properties:

$$\begin{aligned} \langle\chi^2\rangle &= \nu \\ Var(\chi^2) &= 2\nu \end{aligned}$$

How this helps us

- If the model is good, we expect $\chi^2 \sim \nu \pm \sqrt{2\nu}$.
- Often when quoting a χ^2 value, we use the 'reduced χ^2', which is χ^2/ν, where a good fit is $\chi^2/\nu \sim 1$.
- The confidence bounds on the best fit values can be found by using the $\Delta\chi^2$ interval, with $\Delta\chi^2 = 1$ equivalent to the 1σ confidence interval for $\nu = 1$.

Optimisation methods: Least squares

- The method of finding the best fitting values in a model is called optimization, and there are few key techniques, two of which we will explore here.

Optimisation methods: Least squares

- The method of finding the best fitting values in a model is called optimization, and there are few key techniques, two of which we will explore here.
- In the method of least squares fitting we are trying to minimise the difference between the observation and the model prediction by adjusting the values of the parameters.
- If we have a set of pairs of data $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$ and a model for the dependence of y on x represented by a function, $f(x, \beta)$ where β is a vector of free parameters, we need to find the set of β that minimises

$$r = \sum_{i}^{N} (y_i - f(x_i, \beta))^2$$

f(x,B) = ax+b
where B = (a b)

where r is called the residual. In the machine learning world, this is also known as a 'loss function'.

What are we trying to do?

- We are trying to reduce the loss.

What are we trying to do?

- We are trying to reduce the loss.
- To incorporate the concepts of χ^2 statistics, it is common to simply define the residual as the χ^2, such that we are trying to minimize the χ^2 value.
- In the old days, one would loop over a grid of possible candidate values for β and find the combination of parameters that result in the minimum r – these would be the 'best fit' values.
- This 'parameter search' is very costly computationally, especially as the number of parameters increases, so more efficient means of searching a parameter space have been devised. One of these is called gradient descent which is a very nice use of multivariable calculus (so, the loss function must be differentiable) which makes use of a very natural property of the gradient.

Using Scipy and Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import optimize

# Define some noisy data
err = 2
x = np.random.uniform(0.,10,10)
y = 2*x - 3
y += np.random.normal(0,err,len(x))
errs = np.array([err]*len(x))

# Plot the points
plt.plot(x,y,marker='o', linestyle='none',markerfacecolor='black',markeredgecolor='black')

# And error bars
plt.errorbar(x,y,yerr=err, linestyle='none', color='black')

plt.xlabel('x')
plt.ylabel('y')
plt.axis((0,10,-4,20))
```

Using Scipy and Python

```
# Define our model with two parameters
def model(x,m,c):
    return m*x + c

# Use the optimize curve_fit() function to fit. We also supply the estimate
# of the uncertainty on each point. Our first guess of the parameters is m=0, c=0
popt,pcov = optimize.curve_fit(model,x,y, sigma=errs, absolute_sigma=True, p0=(0,0))

# Plot the ground truth input and the best fit
xs = np.linspace(0,10,10)

plt.plot(xs,model(xs,2,-3), linestyle='solid', label='Ground truth',color='black')
```

model

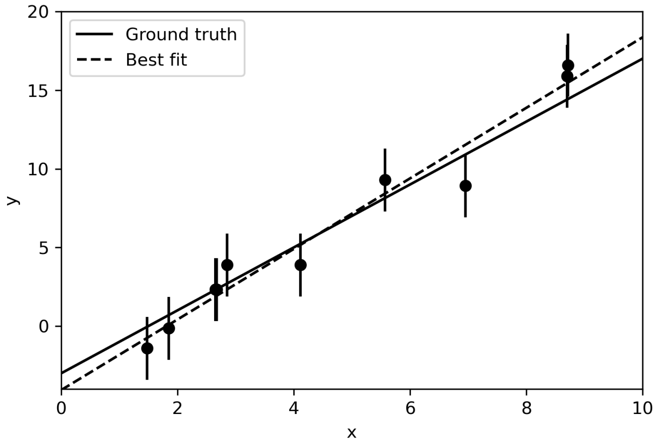
```
plt.plot(xs,model(xs,*popt), linestyle='dashed', label='Best fit', color='black')

plt.legend()

# Print the best fit parameters and their uncertainties
print(popt,np.diag(pcov)**0.5)
```

uncertainty

The output



What have we done?

- Here, we generate some noisy data that follows the ‘ground truth’ relationship $y = 2x - 3$.
- We define our model as $y = mx + c$ where m and c are our free parameters.
- This model goes into `curve_fit()` as the first argument, followed by our observations `x` and `y`.
- We supply some keyword arguments: `sigma` is an array of our uncertainties on each point, so we can take into account how much we ‘trust’ each point in the fit (or rather, how much it contributes to the fit).
- We should always include these in the fitting if we know them, but the optimization will work even if we do not know them.

What have we done?

- Next, `absolute_sigma` states that these are absolute, not relative uncertainties.
- Finally, we supply an initial first guess as to the parameters.
- The return values `popt` and `pcov` contain the best fit parameters and the covariance matrix respectively.
- The latter is an estimate of the covariance of the parameters, and we can take the square root of the diagonal terms to estimate the 1σ confidence bounds of the best fit values in `popt`, although we should investigate their covariance!
- In this case we find the best fit parameters $m = 2.2 \pm 0.3$ and $c = -4.1 \pm 1.5$, so within 1σ of the true values. The χ^2/ν (where $\nu = 10 - 2$ here) is $\chi^2/\nu = 0.81$, which indicates a reasonable fit.

Optimisation methods: Maximum Likelihood Estimation

- Consider the same idea of a model $f(x, \beta)$ as above. Maximum Likelihood Estimation (MLE) is a method of finding the best fit parameters β by maximising the likelihood that the given model and parameters produce the observations y.
- We can write this in the language of probability as follows:

$$p(y_1, y_2, ..., y_N|\beta).$$

  This is the conditional probability of observing the set of N observations given the model parameters β.

[1] From a precision and stability point of view.

Optimisation methods: Maximum Likelihood Estimation

- Consider the same idea of a model $f(x, \beta)$ as above. Maximum Likelihood Estimation (MLE) is a method of finding the best fit parameters β by maximising the likelihood that the given model and parameters produce the observations y.
- We can write this in the language of probability as follows:

$$p(y_1, y_2, ..., y_N|\beta).$$

  This is the conditional probability of observing the set of N observations given the model parameters β.
- The joint probability of producing all of the observations would be given by the product of the individual $p(x_i|\beta)$ for $i = 1$ to N, but for very small values of p, this is hard to deal with computationally[1]. Instead we can *sum* the (natural) log likelihoods to define the 'negative log-likelihood'

similar to entropy (how surprised we are with the result we get)

$$-\ln(\mathcal{L}) = -\sum_{i=1}^{N} \ln(p(y_i|\beta)).$$

[1] From a precision and stability point of view.

University of Hertfordshire UH

Some questions...

Question: Why negative?

Answer: Because although we are talking about maximising the likelihood, often it is preferable to perform minimization. Hence, if we minimize the negative log-likelihood, this is the same as finding the maximum likelihood.

Some questions...

Question: Why negative?

Answer: Because although we are talking about maximising the likelihood, often it is preferable to perform minimization. Hence, if we minimize the negative log-likelihood, this is the same as finding the maximum likelihood.

Follow-up Question: What is the form of the probability distribution?

Answer: Going back to the notion of the residual, if the data points are independent and their uncertainties Gaussian (as is often the case) we expect the distribution of r to follow a Gaussian or Normal distribution.

Negative log likelihood function

- This allows us to define the negative log likelihood function as follows:

$$-\ln(\mathcal{L}) = -\frac{1}{2}\sum_{i=1}^{N}\left(\frac{y_i - f(x_i, \beta)}{\sigma_i}\right)^2 .$$

- Now our goal is to find the best fit parameters β that minimize $-\ln(\mathcal{L})$.
- To estimate uncertainties we can think of the likelihood surface (or, more generally, volume) for all combinations of model parameters.
- Note that for linear models, MLE is equivalent to least squares optimization.

Summary

- We have seen how to perform normalisation and standardisation:
  - minmax
  - logarithmic scaling
  - Z-normalisation
  - unit vector normalisation
  - decimal scaling
- We discussed how to fit a data to a model (least squares and MLE), and how to judge the suitability of the model.