Week 7a:

Pre-processing

G6061: Fundamentals of Machine Learning [23/24]

Dr. Johanna Senk

UNIVERSITY
OF SUSSEX

Outline

A model or method is only as good as its features!

- General pre-processing principles
- Input normalisation
- Imputing (filling in missing values)
- Feature selection

General framework for model training

Input data
$\downarrow$
Pre-processing
$\downarrow$
Model optimisation
$\downarrow$
Post-processing
$\downarrow$
Output data

Although **in principle**, networks can uncover any patterns in any data, **in practice** it's easier if pre-processing is performed first.

Pre-processing does not need to be optimal, but applying some **human intelligence** in this step can enhance and speed up the performance of the **artificial intelligence** in model training.

Post-processing often only involves passively re-mapping the output into its raw form.

Pre-processing data

- **Input normalisation**: Normalise (rescale) features to lie on a sensible scale. Useful for network/model training, and for visual inspection of data.
- **Imputing**: I.e., filling in missing data.
- **Feature construction and selection**: Incorporate your **domain knowledge** to select features that are likely to be useful.
  Mindfully build features from the data based on **visual inspection and consideration of the task** to be done.
- **Dimensionality reduction**: Use maths to efficiently reduce the number of dimensions.
  Define a few new variables that are combinations of the raw variables, and account for most of the variance.

Input normalisation

For example, using height and weight as potential features amongst others in predicting something about patient health:
Results shouldn't depend on whether using centimetres, metres or inches, pounds, kilograms etc. – so normalise!

Table 10-1. Heights and Weights

Person	Height (inches)	Height (centimeters)	Weight
A	63 inches	160 cm	150 pounds
B	67 inches	170.2 cm	160 pounds
C	70 inches	177.8 cm	171 pounds

```
a_to_b = distance([63, 150], [67, 160])        # 10.77
a_to_c = distance([63, 150], [70, 171])        # 22.14
b_to_c = distance([67, 160], [70, 171])        # 11.40
```

```
a_to_b = distance([160, 150], [170.2, 160])    # 14.28
a_to_c = distance([160, 150], [177.8, 171])    # 27.53
b_to_c = distance([170.2, 160], [177.8, 171])  # 13.37
```


UNIVERSITY OF SUSSEX

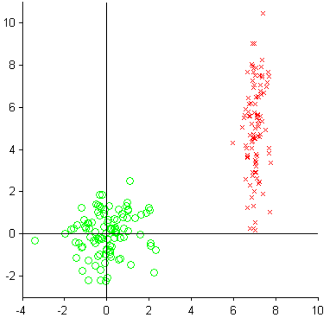
Input normalisation

If variation and scale of some variables is much greater than variation in others, this can mess up the rest of pre-processing, and hinder model-fitting.
Therefore, pre-process data to make each feature have mean 0 and standard deviation 1, via simple transformation on each feature:

$$x \to \frac{x - \mu}{\sigma}$$

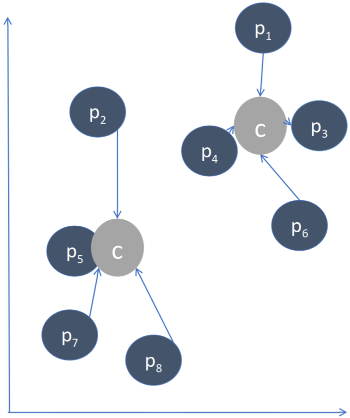
μ is the mean
σ is the standard deviation

(individually for each variable)



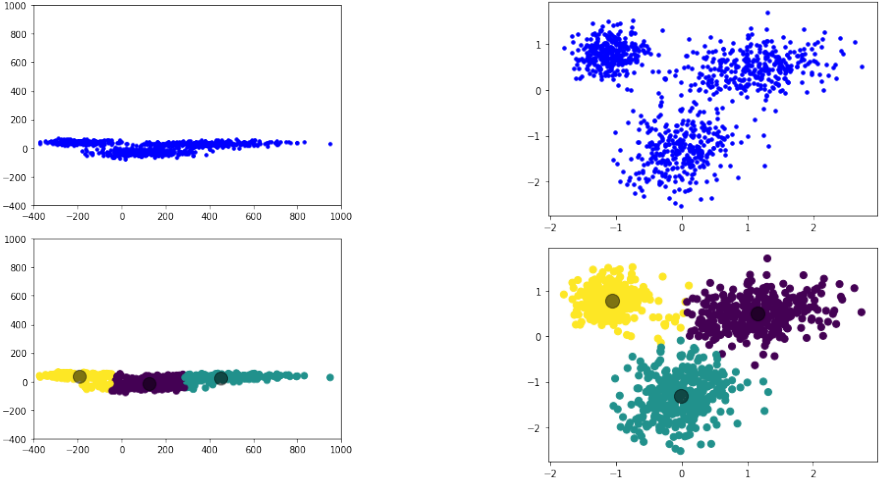
Recap: K-means clustering

- Select k points as initial centroids
- While centroids changing:
  - Form k clusters by assigning each point to its nearest centroid
  - Recompute centroid of the cluster

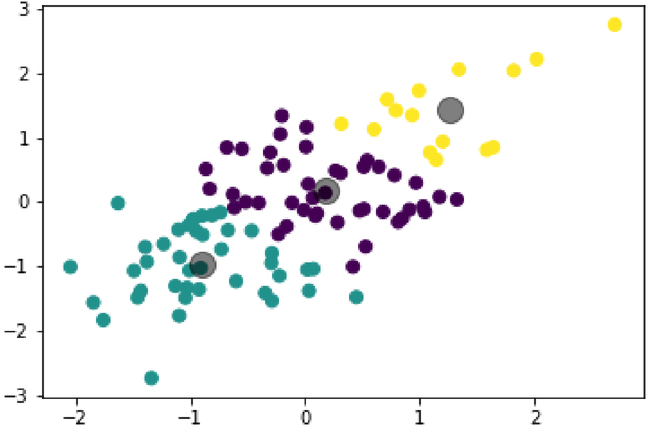


UNIVERSITY OF SUSSEX

K-means without and with normalisation



Example: T-shirts



But is this optimal to scale height and weight equally?

UNIVERSITY OF SUSSEX

Input normalisation maintains correlations

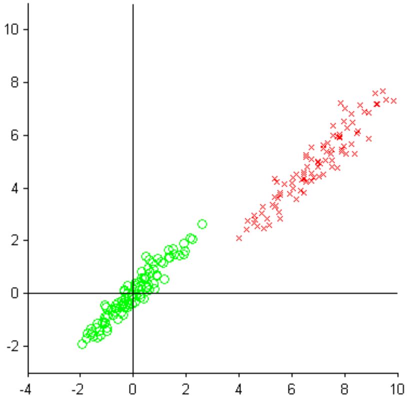
But will change the covariance, recall:

$$Cov(X, Y) = \langle (X - \mu_X)(Y - \mu_Y) \rangle$$

$$Corr(X, Y) = \frac{\langle (X - \mu_X)(Y - \mu_Y) \rangle}{\sigma_X \sigma_Y}$$

Red: before pre-processing

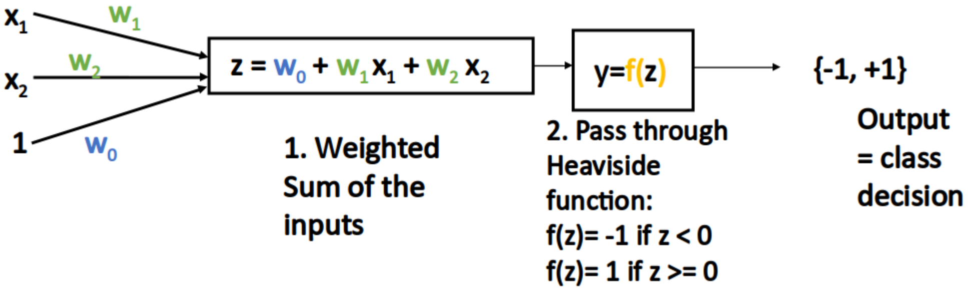
Green: after pre-processing



UNIVERSITY OF SUSSEX

Recap: The perceptron

For D-dimensional data, a perceptron consists of:
D weights, a bias and a thresholding activation function.
For 2D data we have:



Hint: View the bias as another weight from an input which is constantly on.

$z = \boldsymbol{w} \cdot \boldsymbol{x}$ where $\boldsymbol{x}$ is $[1, x_1, x_2, ..., x_D]$ and $\boldsymbol{w}$ is $[w_0, w_1, w_2, ..., w_D]$

Error for perceptron

$E(\boldsymbol{w}) = 0$ when classification is correct.
When wrong, error is how far the perceptron was from getting it right.

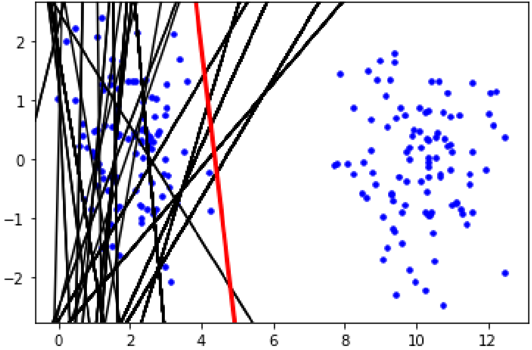
$$
\begin{aligned}
E(\boldsymbol{w}) &= \boldsymbol{w} \cdot \boldsymbol{x} && \text{if output is +1 and output shoud have been -1} \\
E(\boldsymbol{w}) &= -\boldsymbol{w} \cdot \boldsymbol{x} && \text{if output is -1 and output shoud have been +1}
\end{aligned}
$$

Recall $\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{i=1}^{D} w_i x_i + w_0$

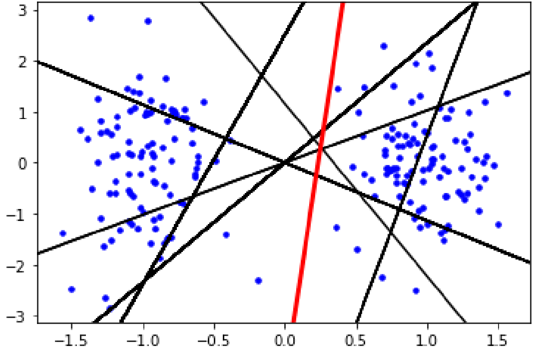
Input normalisation for perceptron

Works best when each input dimension (feature) varies on the same scale, so that the formula for the error behaves sensibly.

Not normalised



Normalised



UNIVERSITY OF SUSSEX

Normalisation enables regularisation

- Size of weights (model parameters) will depend on the scale of the features (inputs).
- So, for regularization schemes involving overall magnitude of weights to work well, inputs should be normalised.

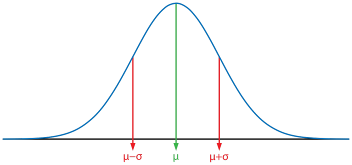
$$\underset{\boldsymbol{w}}{\text{minimise}}\, \mathcal{L}'(y, \hat{f}(x, \boldsymbol{w})) = \underset{\boldsymbol{w}}{\text{minimise}} \left\{ \mathcal{L}(y, \hat{f}(x, \boldsymbol{w})) + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w} \right\}$$

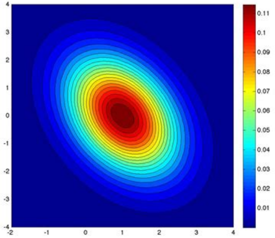
where $\boldsymbol{w}^T \boldsymbol{w} = \sum_{i=0}^{D} w_i^2$

UNIVERSITY OF SUSSEX

Input normalisation - some variations

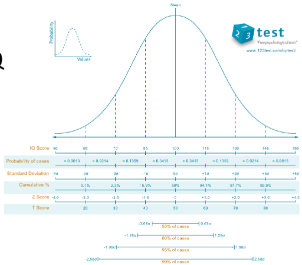
- Straightforward input normalization works best for Gaussian distributed data, i.e., when inputs have a normal (Gaussian) distribution.
- This assumption frequently holds, but obviously not always, and for some distributions other kinds of rescaling will work better than standard input normalisation.



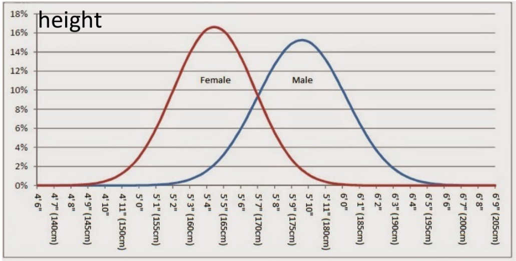


UNIVERSITY OF SUSSEX

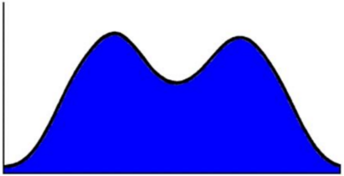
IQ



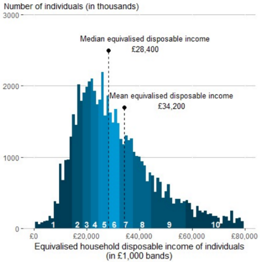
height



Book prices – bimodal, peak for paperbacks and peak for hardbacks.



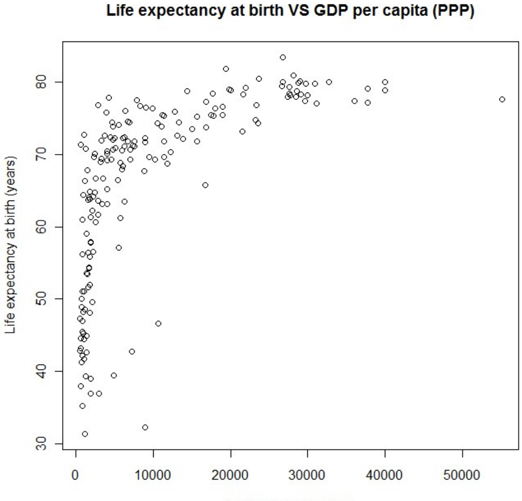
income



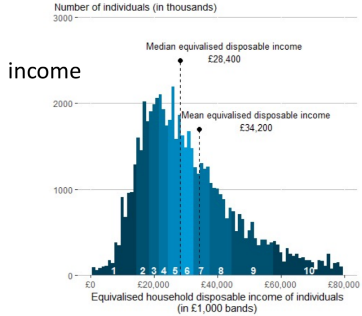
UNIVERSITY OF SUSSEX

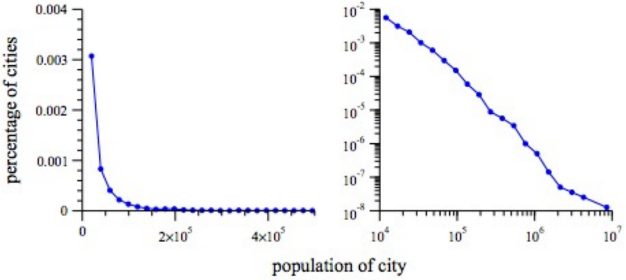
Input normalisation - some variations

If a variable is bounded, you might want to transform the range to a uniform scale, say 0 to 1, or simply binarise (although this throws away information).



Logarithmic transformations





Income distributions and city populations have “fat tails”.
Taking the logarithm can be useful.

Categorical data: One hot encoding

For example, colour (red, blue, green)

Name	Favourite colour
Adam	red
Bob	green
Charlie	blue
...	

→

Name	Favourite colour red	Favourite colour green	Favourite colour blue
Adam	1	0	0
Bob	0	1	0
Charlie	0	0	1
. . .			

UNIVERSITY OF SUSSEX

Imputing - filling in missing values

- If some instances in the dataset have no data for some of the features, they might need to be filled in for the model to run.
- Common to take mean, median or mode.
- But sometimes the fact the value is missing could be predictive of something! Sometimes take the fact the value is missing as a bonus feature!
- Sometimes a preliminary round of machine learning is done to fill in (impute) the missing values, e.g., **logistic regression**, or **k-nearest neighbours**.

UNIVERSITY OF SUSSEX

Logistic regression

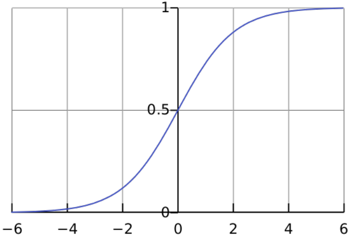
Estimating the probability that an instance belongs to a particular class - a regression algorithm used for classification.

Logistic (sigmoid) function:

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Prediction:

$$\hat{y} = \begin{cases} 0 \text{ if } \sigma(\boldsymbol{w} \cdot \boldsymbol{x}) < 0.5 \\ 1 \text{ if } \sigma(\boldsymbol{w} \cdot \boldsymbol{x}) \geq 0.5 \end{cases}$$



Wikipedia

UNIVERSITY OF SUSSEX

K-nearest neighbours

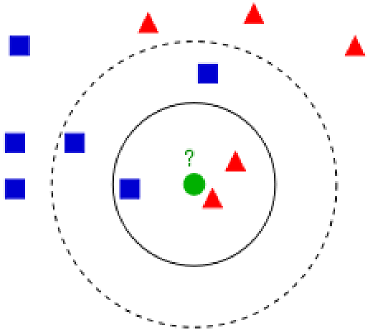
- Use all the other features to predict the missing value.
- Choose a k, and consider the values of the missing values for the k closest neighbours in the population.
- This requires having a measure of distance like for k-means clustering.
- For a binary feature, the most common value amongst k-nearest neighbours is taken as the missing value.
- For a continuous feature, take mean of values from k-nearest neighbours.

UNIVERSITY OF SUSSEX

K-nearest neighbours

E.g., here we have
two continuous features (x and y)
and a categorical feature (blue or red),
which is missing for the point coloured green.

If we use k=3, then choose red.
For k=5, get blue.



Wikipedia

This can be used for pre-processing (imputing) or sometimes for some (rough) actual machine learning!

Feature selection

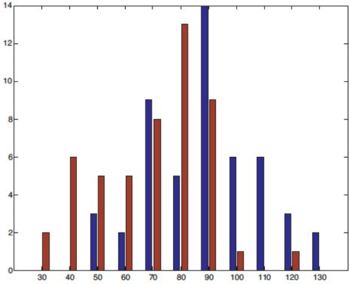
- Inputting too many features into a model can be bad:
  - Overfitting more likely.
  - Computational cost high.
- For small datasets, you want every single feature to count. Even on very large datasets, clever feature selection and extraction is a good idea.
- Anything you can do to boost the signal to noise ratio will be good.
- For methods like k-means clustering and k-nearest neighbours, lots of noise features can seriously mess up the algorithm:
  The Curse of Dimensionality

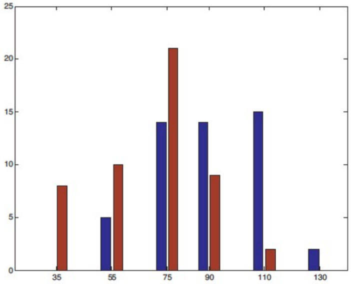
Feature selection - Example

Taking words as features could be good for classifying email as spam but might be useless for classifying sentences as grammatical vs. ungrammatical.

Feature selection - Data visualisation

Example: Predicting diabetes based on weight For a small sample, the histogram might be spiky $\rightarrow$ convert to histogram with fewer bins.





[Flach- Chapter 1]

Blue- diabetic

red- not diabetic

UNIVERSITY OF SUSSEX

Summary and outlook

- General pre-processing principles
- Input normalisation
- Imputing (filling in missing values)
    - Logistic regression
    - K-nearest neighbours
- Feature selection

Next lecture:

- Dimensionality reduction
    - Principal Components Analysis

UNIVERSITY OF SUSSEX