Language Analytics

Session 4: Text Classification 1 (Logistic Regression)

Ross Deans Kristensen-McLachlan

rdkm@cas.au.dk

Course outline

- 1. Introductions
- 2. String Processing with Python
- 3. NLP for linguistic analysis
- **4. Text Classification 1**
- 5. Text Classification 2
- 6. Word embeddings
- 7. Language modelling 1
- 8. Language modelling 2
- 9. BERT
- 10. More BERT
- 11. Project pitches
- 12. Generative models
- 13. Social impact

Plan for today

- Short catch-up
- Why classification?
- What are we classifying
- How does a machine learn?

Catch-up

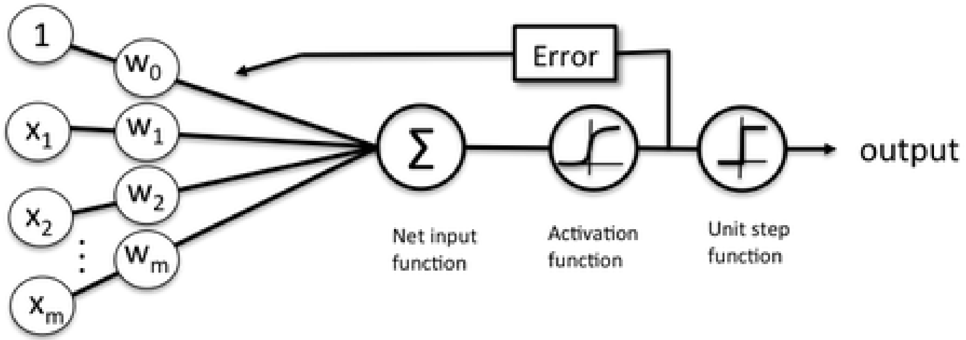
- How is everyone?
- Any problems or issues you want to flag?
- Any problems or issues with the assignment?

Why classification?

- Many problems can be framed as essentially a classification problem
  - Does object X belong to class A or class B?
- Part of speech tagging and NER
  - Which class does a word belong to? What kind of entity is it?
- Sentiment analysis
  - Does this document present a positive or negative perspective?
- Danielsen et al. (2019). Investigated whether mechanical restraint after 3 days of admission could be predicted from patient medical records
  - Restraint vs no-restraint
  - AUC 0.87 (95% CI 0.79–0.93). At 94% specificity, the sensitivity was 56%

Feature engineering

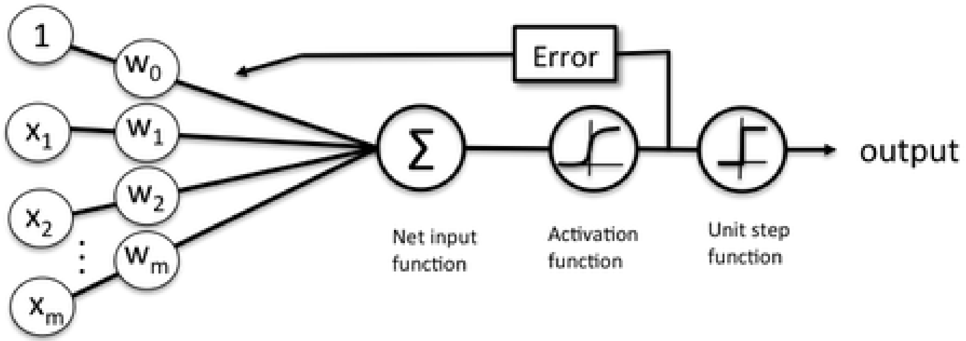
- A logistic regression classifier requires taking some kind of input (text) and some kind of output (class labels)
- The model learns the relationship between the input and output through minimising some kind of loss function
- But what actually *is* the input in this case?



Schematic of a logistic regression classifier.

Feature engineering

- One approach is to do *feature engineering*
- For your chosen domain and problem, select a particular range of features which may be relevant for your task
- This could be linguistic features such as distributions of PoS tags, dependency labels etc
- However, this is a time-consuming and technically challenging task



Schematic of a logistic regression classifier.

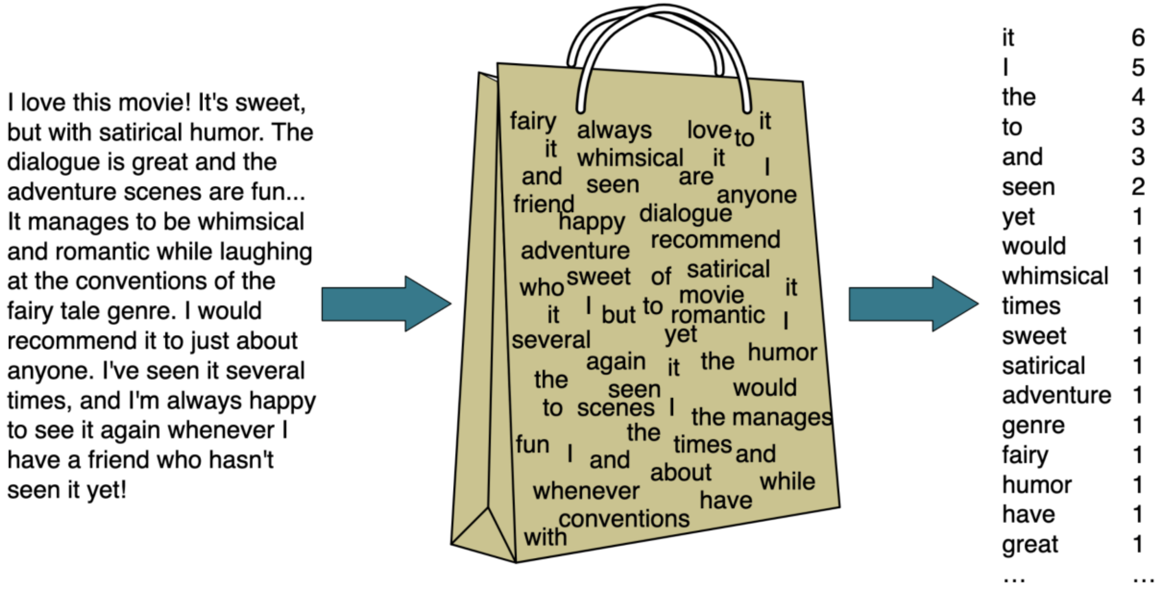
Vectorization

- Another simple and effective way of feature extraction is through what is often referred to as *document vectorization*
- Textual data is 'transformed' into some kind of numerical representation which captures something about the nature of the language in the text
- One of the easiest ways to do this is to simple *count* how often individual features (words) appear in each document in a corpus

Bag of words

- A bag of words representation of document is an unordered set of all words in that document
- Positional information is ignored – we only keep the overall frequency of that word in the document
- If V is the vocabulary of all words w_j in corpus C, then each document D_j can be represented as a numerical vector of how often each w appears:

$$D_1 = \{w_1=0,\ w_2=10,\ w_3=8:,\ \ldots,\ w_i=100\}$$
$$D_2 = \{w_1=20,\ w_2=5,\ w_3=0:,\ \ldots,\ w_i=10\}$$
…
$$D_i = \{w_1=50,\ w_2=10,\ w_3=5:,\ \ldots,\ w_i=20\}$$



Term-document matrix

- The table on the right shows a concrete example

$$C = \{D_1, D_2, D_3, D_4\}$$
$$V = \{battle, good, fool, wit\}$$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- We can represent the whole corpus as a *term-document matrix*

- Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column).

Term-document matrix

- We can then think of each document as a point in 4-dimensional space

As You Like It = (1, 114, 36, 20)
Twelfth Night = (0, 80, 58, 15)
Julius Caesar = (7, 62, 1, 2)
Henry V = (13, 89, 4, 3)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Term-document matrix

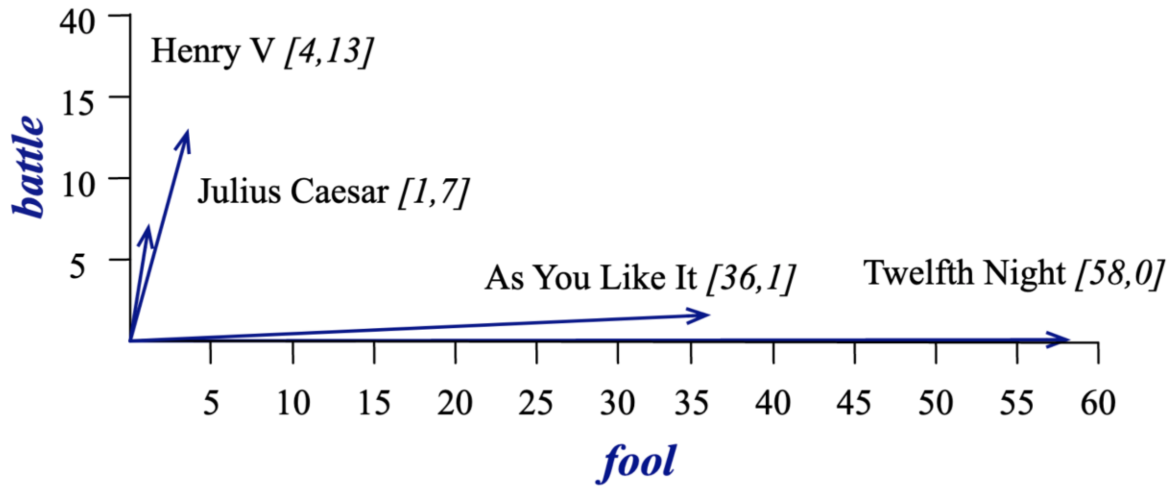
- We can then think of each document as a point in 4-dimensional space

  As You Like It = *(1, 114, 36, 20)*
  Twelfth Night = *(0, 80, 58, 15)*
  Julius Caesar = *(7, 62, 1, 2)*
  Henry V = *(13, 89, 4, 3)*

- Visualising 4-dimensions is tricky, so let's focus on only those dimensions corresponding to *battle* and *fool*

- *Documents with similar vectors contain similar words in similar distributions*

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3



TF-IDF weighting

- In this example, the word *good* is not particularly discriminative

- Words which occur frequently in a document are important but if they occur *too* frequently in the corpus, they become redundant

- We can address this through the use of a *weighting scheme*

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

TF-IDF weighting

- A commonly used weighting is known as *term frequency-inverse document frequency*

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

- For term *t* and document *d*

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d)+1)$$

$$\mathrm{tf}_{t,d} = \begin{cases} 1+\log_{10}\mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d)>0 \\ 0 & \text{otherwise} \end{cases}$$

- Where N is corpus size and df_t is number of documents containing t

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

- TF-IDF score $\quad w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$

A brief summary

- Bag-of-words and TF-IDF weighting are two quick and efficient ways to create numerical representations of documents
- Neither approach accounts for positional information – tokens are counted separately, disregarding how they might be related to one another grammatically
- Tokens need not necessarily be words but could also be (for example) bigrams or trigrams
- Both approaches – but specifically TF-IDF – provide a good, simple baseline for many classification tasks
- But how are they used for classification purposes?

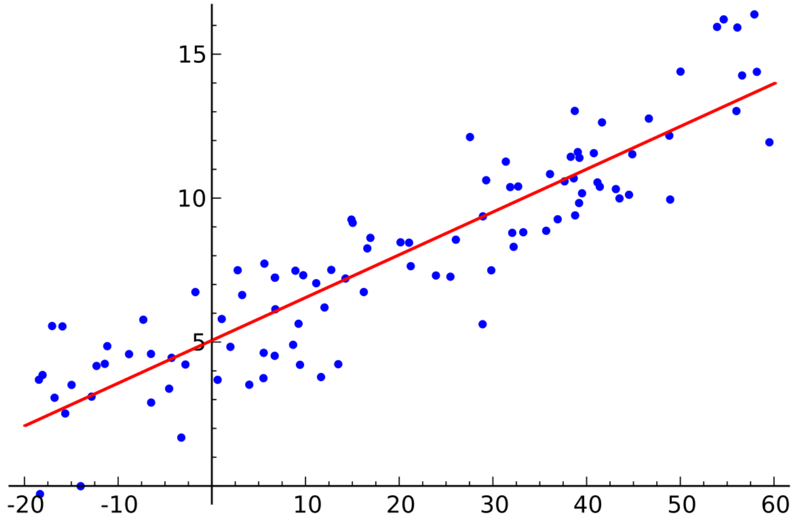
Thinking about regression

- Regression is all about looking for and modelling relationships between variables

- Specifically, we want to model the relationship between one or more *independent variables* and a *dependent variable*

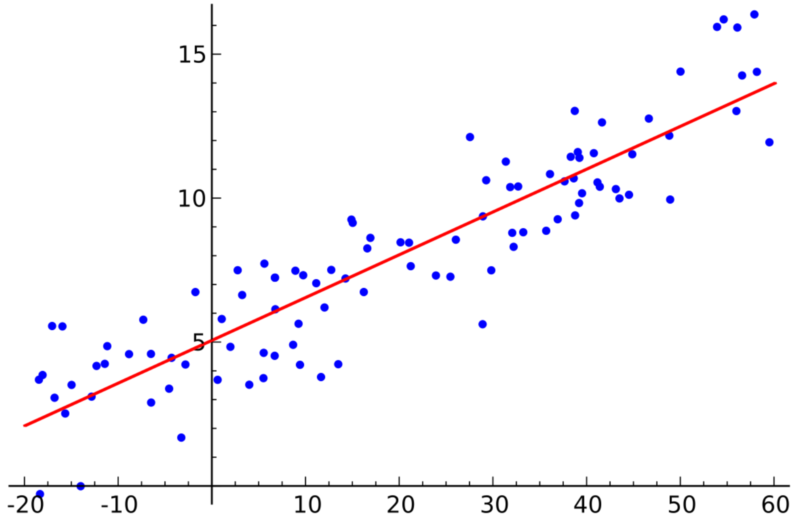
- The intuition is that a change in the independent variable corresponds to some kind of linear change in the dependent variable

- Think about the distribution of the blue dots to the right



Thinking about regression

- In this visualisation, each blue point is a measurement
- The x-axis is one independent variable; the y-axis is the dependent variable
- Generally speaking, an increase in the x-axis has a corresponding effect on the y-axis
- This relationship can be modelled by finding a 'line of best fit' – shown here by the red line



Thinking about regression

- Simple linear regression is the process of estimating the parameters to create this best fit for the data
- It does this by minimising the overall difference between the model and the observed data

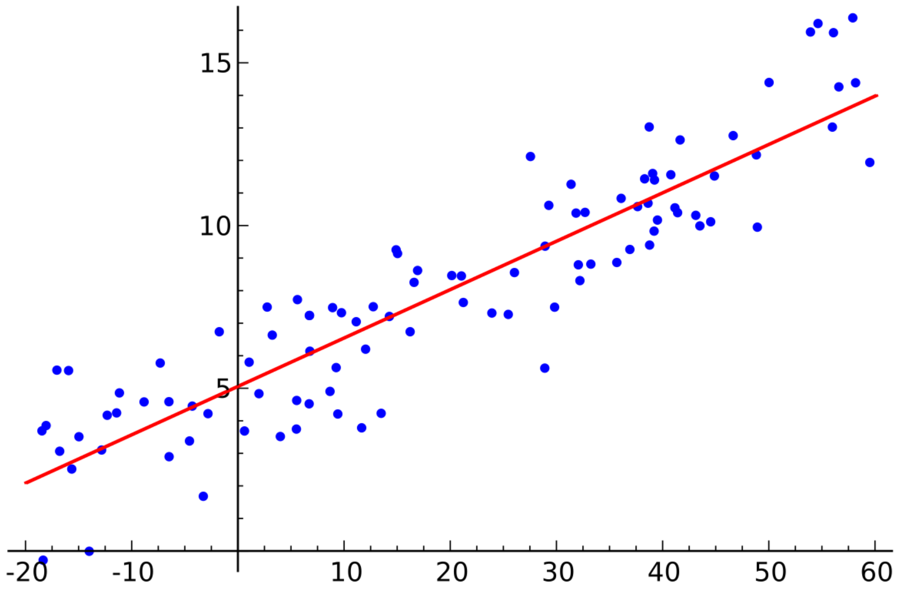
Constant/Intercept

Independent Variable

$$Y_i = \beta_0 + \beta_1 X_i$$

Dependent Variable

Slope/Coefficient



Thinking about regression

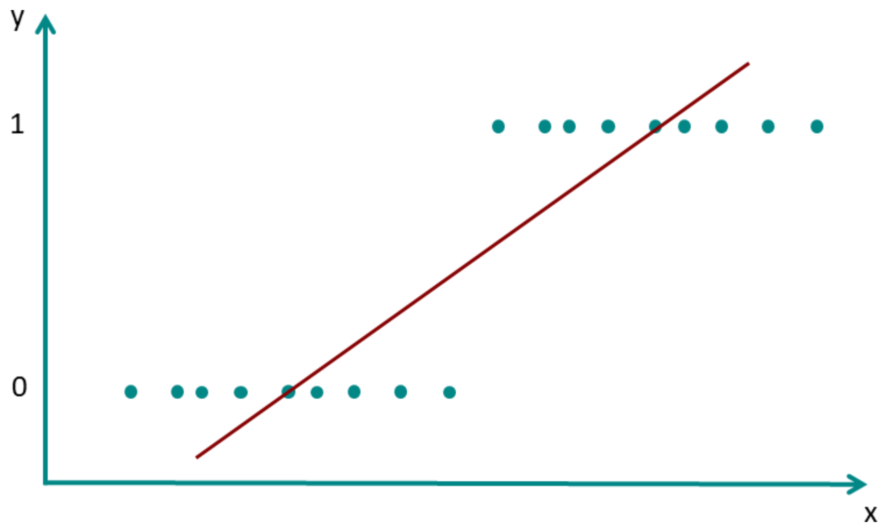
- Linear regression is great when we have continuous variables. But what about this example?

- Here the dependent variable has only two possible values, namely 0 or 1. So it is *categorical*

- A regression of the kind we just saw can have a wide range of values, including negative values, and so it is unsuited to modelling this kind of relationship

- So how can we model it?

- Essentially, we need to find some way of 'squashing' the regression into the space from 0 -> 1



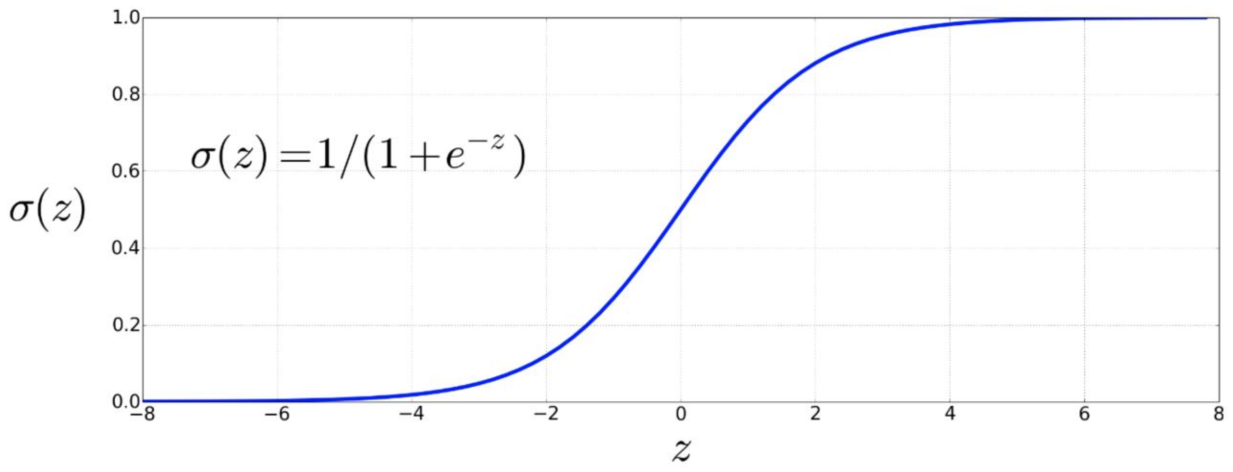
Thinking about regression

- This can be done using the *logistic function*
- The logistic function is a kind or **sigmoid curve** (S-shaped)

Linear model: $$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

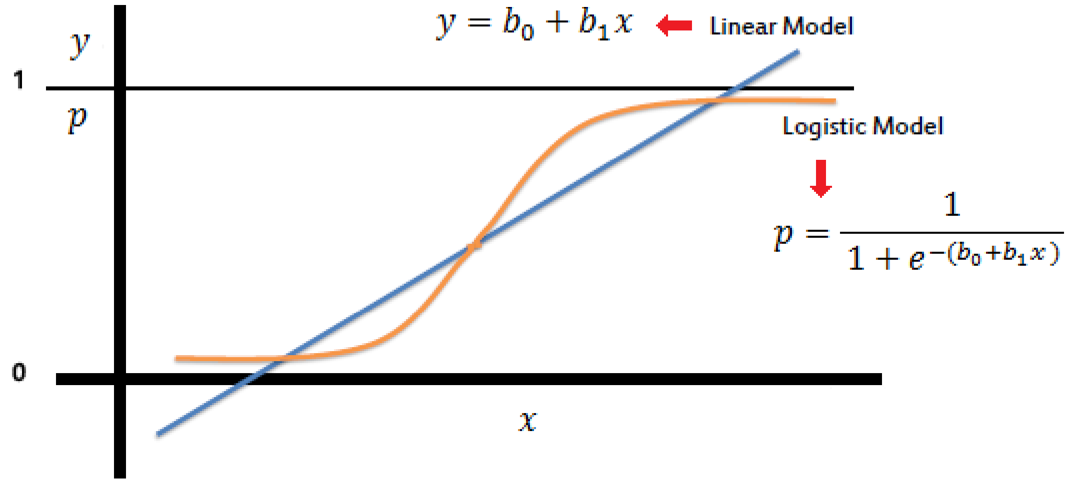
Matrix notation: $$z = w \cdot x + b$$

Then: $$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$$



Thinking about regression

- We can use the logistic function as a *link* function to ‘squash’ our regression
- This allows us to keep our regression values bounded within the limits of 0 -> 1
- It also has the added benefit of returning a *percentage value*



From regression to classification

- Linear regression can be used to model the relationship between continuous variables
- We also saw that we can use a logistic link function to model situation where we have binary, categorical dependent variables
- But how do we get from logistic regression to *classification*?

Break

From regression to classification

- Logistic regression models the probability of a certain class or event given the independent variables and some model parameters

- We need to set a *decision boundary* greater than which say it's likely to be 1; otherwise it's 0

- What do you think the most sensible decision boundary is?

From regression to classification

- Logistic regression models the probability of a certain class or event given the independent variables and some model parameters

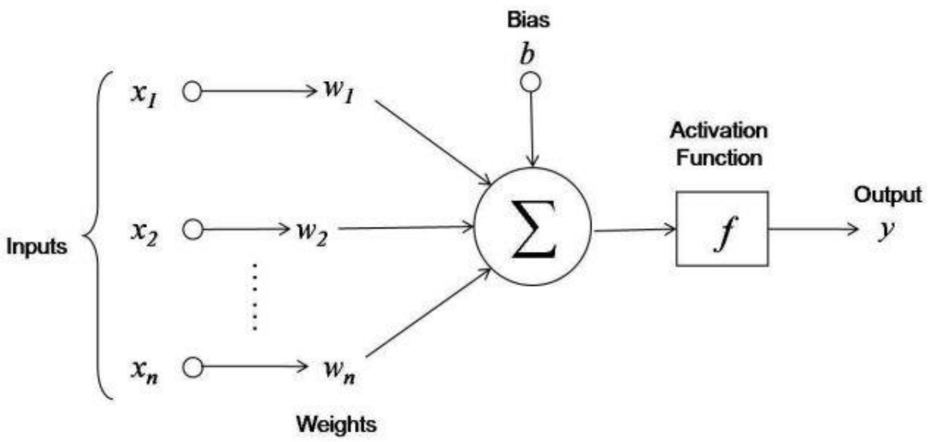
- We need to set a *decision boundary* greater than which say it's likely to be 1; otherwise it's 0

- What do you think the most sensible decision boundary is?

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Logistic regression classifier

- A logistic regression classifier does the following:
  - Takes some input features from the training data
    - BoW vectors, TF-IDF vectors
  - Estimates parameters to best fit the model
    - Loss function, optimization algorithms
  - Use probabilities to predict class membership, given some decision boundary



- How does the model actually *learn* the parameters?

How the classifier learns

- With linear regression, we saw that the model tried to minimise the difference between the observed data and the 'line of best fit'

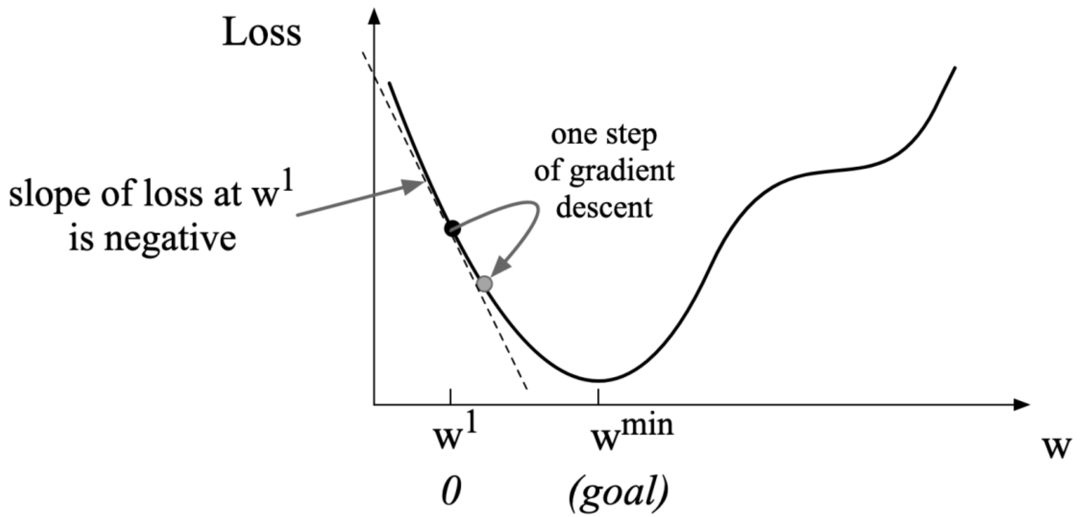
- In machine learning, we try to *minimise a loss function.* In other words, we want to find the parameters which return the smallest possible value for some given equation

- For some loss function L and some output prediction $\hat{y}$:

$$L(\hat{y},y) = \text{How much } \hat{y} \text{ differs from the true } y$$

Gradient descent

- The classifier doesn't test weights and biases in a completely random way
- Instead, it uses optimization methods in order to find the most suitable parameters efficiently
- One way to do this is through the use of *stochastic gradient descent*
- Essentially, we find the tangent of the loss function at a given point and move in the opposite direction
- If the slope is negative relative to the y-axis, we move along the x-axis and so forth



Take home points

- A logistic regression classifier is used to learn parameters to estimate the probability of some class or category, given some input features
- These parameters are weights and biases (a bit like slopes and intercepts) which are assigned to each of the features
- The algorithm learns the best parameters by finding the ones which return the smallest difference in the predicted and true labels in the training data
- The final calculated weights show the relative importance of each word for making a prediction
    - High +ve weight predictive of positive class
    - High –ve weight predictive of negative class

Summary

- Logistic regression is a probabilistic classifier using supervised machine learning which has four main components
- It takes a ***feature representation*** of the input rather than the raw data
- It uses the ***logistic (sigmoid) function*** to return the probability of an output class given the input features
- It learns the best parameters by minimising error on the training examples, calculated by a given **loss function** (e.g. cross-entropy loss)
- It uses an **optimization algorithm** to learn these parameters efficiently from the data (e.g. stochastic gradient descent)

Additional reading

- Danielsen, A. A., Fenger, M. H.J., Østergaard, S. D., Nielbo, K. L., Mors, O. (2019). “Predicting mechanical restraint of psychiatric inpatients by applying machine learning on electronic health data”, *Acta Psychiatrica Scandinavica,* 140, 147–157. doi: 10.1111/acps.13061

- Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition.* New York: Springer-Verlag.

Appendix – Some maths

Some extra slides for those interested in learning more about the mathematics underlying machine learning.

You don't have to learn this, it's entirely supplementary!

Appendix – Binary cross entropy loss

- For a logistic regression classifier, we want to find the *weights* and *biases* that maximize the *log probability* of the true *y* labels in the training data given the observations *x*

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^{y}(1-\hat{y})^{1-y}\right]\\ &= y\log\hat{y}+(1-y)\log(1-\hat{y})\end{aligned}$$

Where $\hat{y}=\sigma(w\cdot x+b)$

- We can turn this into a *loss function* (or something we want to minimise) by flipping the signs:

$$L_{\mathrm{CE}}(\hat{y},y)=-\log p(y|x) \;=\; -[y\log\hat{y}+(1-y)\log(1-\hat{y})]$$

Or

$$L_{\mathrm{CE}}(\hat{y},y) \;=\; -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

Appendix – Gradient descent

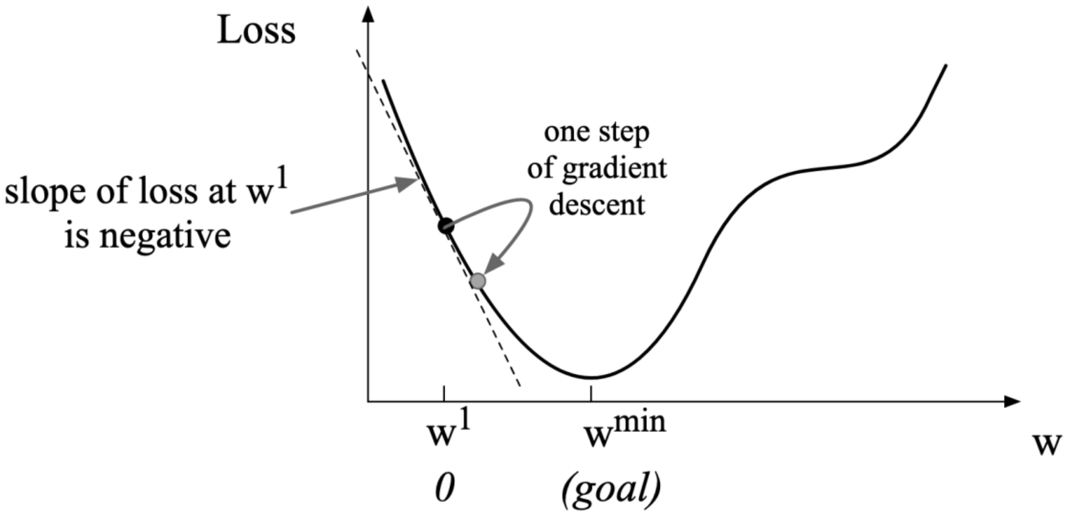
- The goal is to find the set of weights which minimizes the loss function, averaged over all examples
- For

$$\boldsymbol{\theta} = w, b$$

- The *objective function* is

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

- Or in natural language terms, "Find the parameters θ which return the lowest average loss value"



Appendix – Gradient descent

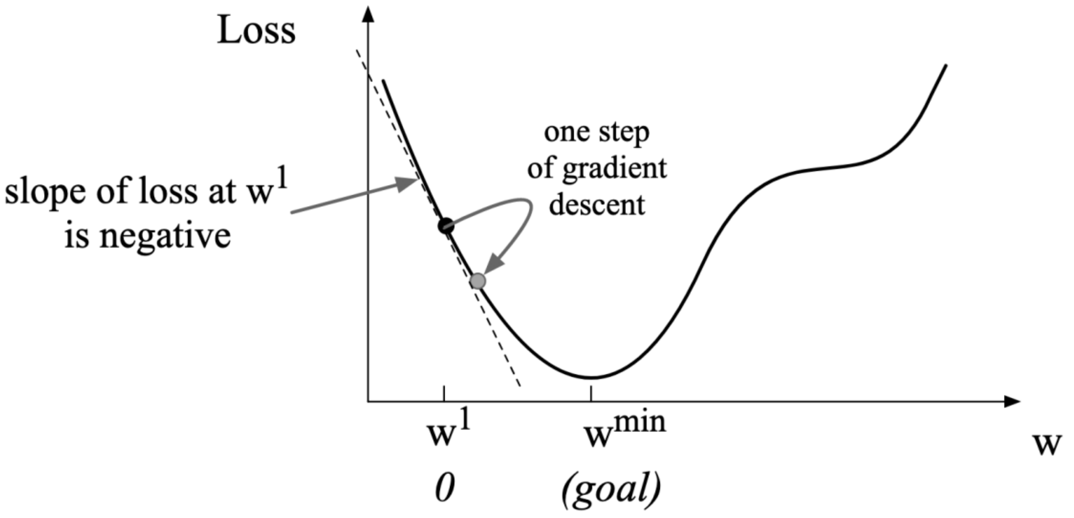
- The magnitude of the step is the value of the slope weighted at *w* by some learning rate *η*

- Slope at *w*

$$\frac{d}{dw}L(f(x;w),y)$$

- So

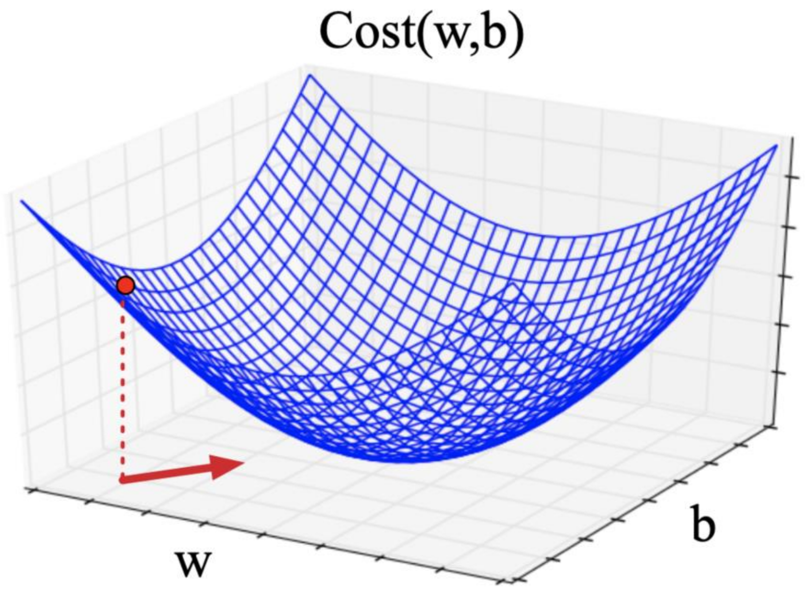
$$w^{t+1} = w^t - \eta \frac{d}{dw}L(f(x;w),y)$$



Appendix – Gradient descent

- In practice, we're not moving simply one-dimension along an x-axis
- Instead, we're working in an *N-dimensional space* where *N* is the number of parameters making up θ
- For each w_j in w, how much does a small change in w_i affect the total loss function?
- Through partial differentiation:

$$\nabla_{\theta} L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \\ \frac{\partial}{\partial b} L(f(x;\theta),y) \end{bmatrix}$$



Appendix – Gradient descent

- The final equation for updating parameters θ is thus:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

- For binary cross entropy loss:

$$L_{\text{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1-y)\log(1 - \sigma(w \cdot x + b))]$$

- For a single observation vector *x,* the gradient is can be derived to show that

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

- Hence, the gradient for w_j is just difference between the predicted $\hat{y}$ and the true y, multiplied by the input value x_j