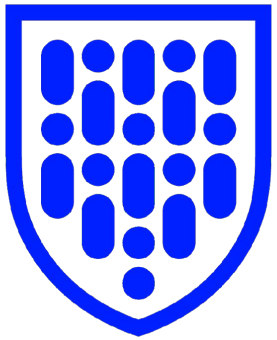


# Institute of Data

2024

# Data Science and AI

Module 5

## Supervised ML: Classification

Institute of Data

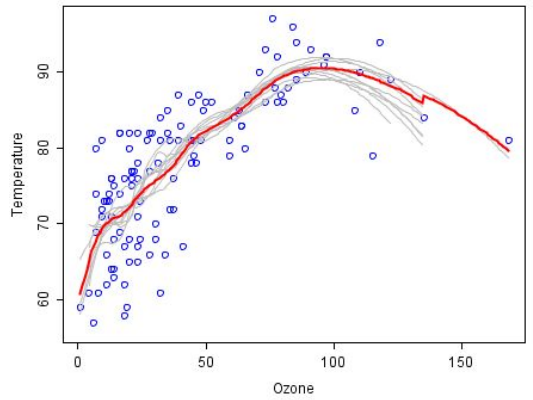
# Agenda: Module 5

- Introduction to **Classification**
- **Logistic Regression**
- **Evaluating** Classification Results
- **Neural Networks**
- **Support Vector Machines**
- **Bayesian Inference**
- **Applications**

# Classification

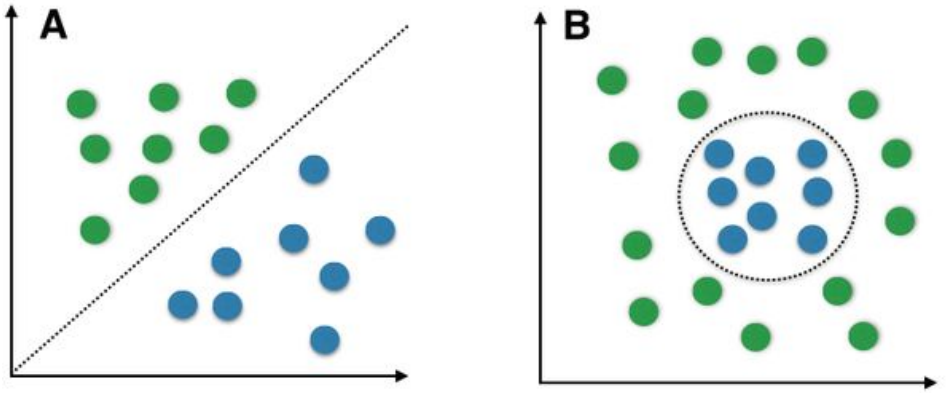
***Regression***

- train a model by fitting data to a **continuous** response
- *predict **continuous** numbers*




***Classification***

- train a model by fitting data to a **discrete** response
- predict **class membership**



Institute of Data

# Logistic Regression

- Introduction to **classification**
- Logistic regression **algorithm**
- **Evaluating** classification results
- Measuring the **quality** of classification models
- **Dummy variables**

# Classification

examples

- fraud detection
  - True / False
- customer segmentation:
  - frequent, high-value / regular, medium-value / occasional / transient
- credit risk
  - low / medium / high
- disease status
  - NYHA Class I / II / III / IV

# Predicting Class Membership by Supervised Machine Learning

training data

- **features**
  - for now, assume these are continuous
- **response**
  - for now, assume this is binary (True/False)
  - Could be multiple classes in some case

goal

- predict $p(y = 1 \mid X)$

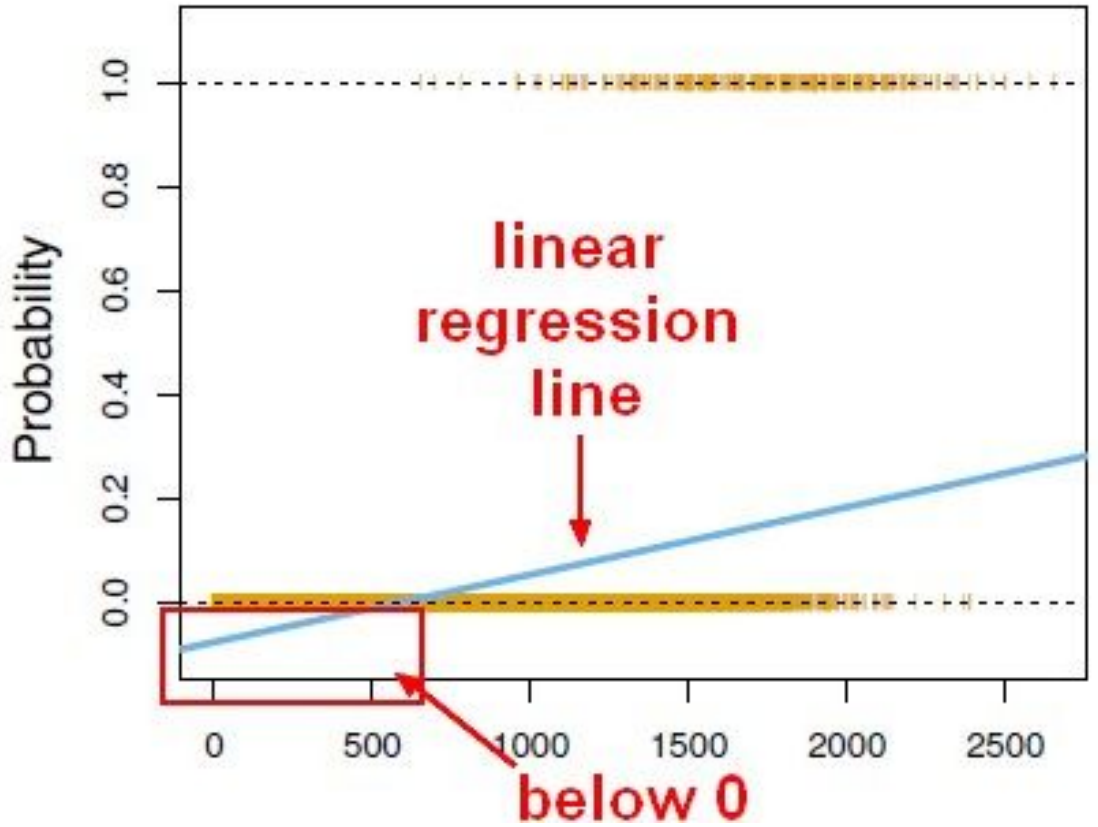
  the probability of $y$ being True given the predictor(s) $X$

# Binary Class Prediction

- **can we use linear regression?**

$y \in \{\text{False, True}\}$ ➡ $y = \beta X + \varepsilon$

- binary response variable results in large residuals
- predictions can be outside [0, 1]



# Binary Class Prediction – cont'd

- need to transform the response so that ***y*** becomes discrete
  - > model the ***probability*** of class membership!
- how about this:

$$\boldsymbol{p}\,(\boldsymbol{y} = 1 \mid \boldsymbol{X}) = \beta_0 + \beta_1 \boldsymbol{X}$$

  - > still gives $\boldsymbol{y} < 0$, $\boldsymbol{y} > 1$
  - > need an approximating function that ensures $\boldsymbol{y} \in [0,1\ ]$

# Binary Class Prediction – cont'd

- $\frac{p}{1-p}$ = odds ratio

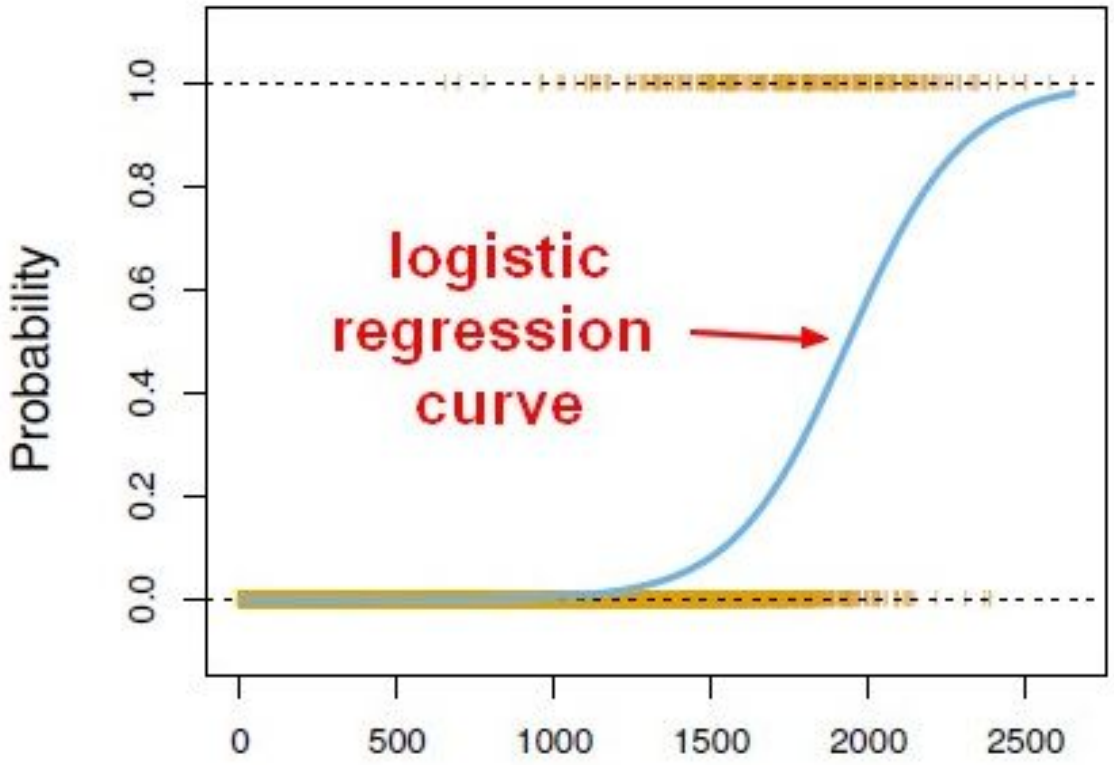
  ($p$ = probability of True)

$$\log\left(\frac{p}{1-p}\right) = \text{logit} \quad (\textit{aka} \text{ log odds})$$

***solve:***

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \boldsymbol{X}$$

logit function



# Logistic Regression

- $\beta_0$, $\beta_1$ are known from regression results:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \boldsymbol{X}$$

let $\boldsymbol{X}$ be a new data point for prediction,

calculate $\hat{y}$:

$$\hat{y} = \beta_0 + \beta_1 \boldsymbol{X}$$

then calculate $p$:

$$p = \frac{e^{\hat{y}}}{e^{\hat{y}} + 1} = \frac{1}{1 + e^{-\hat{y}}}$$

Conditions:

$p > 0$

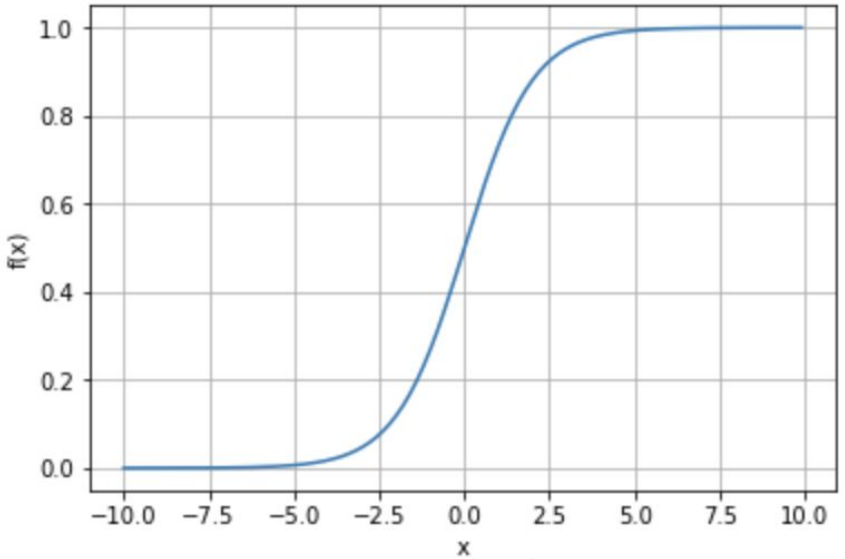
$p < 1$

# Logistic Regression

Appropriate when predicting a **binary categorical** outcome variable from a set of predictor variables (features) that may be **continuous and/or categorical**

**Features** should be independent with no missing data.

Logistic Regression typically requires **a relatively large sample size.** A general rule of thumb is that you need at least 10 cases of the least frequent outcome for each independent variable in your model.



Sigmoid or logistic curve

# Logistic Regression – Scikit-learn

from **sklearn.linear_model** import LogisticRegression

- Can deal with any number of features
- **Features must be numeric**
  - Categorical features should be converted to dummy features
- Can perform **multi-class classification**

# Evaluating Classification Results

There are a number of metrics that can be used to evaluate a classification model. Many of these metrics revolve around values drawn from the Confusion Matrix.

The **Confusion Matrix** is a table that contains counts of the predictions of the model versus the actuals.

**Note** that if we are interested in predicting the opposite class, the entries would be reversed (the positive becomes negative and vice versa).

| Actual | Positive | Negative |
|---|---|---|
| Prediction | | |
| Positive | True Positive (TP) | False Positive (FP) |
| Negative | False Negative (FN) | True Negative (TN) |

# Confusion Matrix

It is useful to develop an intuition of the meaning of each row and column and key combinations of the counts in the Confusion Matrix

| Actual | Positive | Negative | |
|---|---|---|---|
| Prediction | | | |
| Positive | True Positive (TP) | False Positive (FP) | Total predicted positive |
| Negative | False Negative (FN) | True Negative (TN) | Total predicted negative |
| | Total actual positive | Total actual negative | Total Sample count |

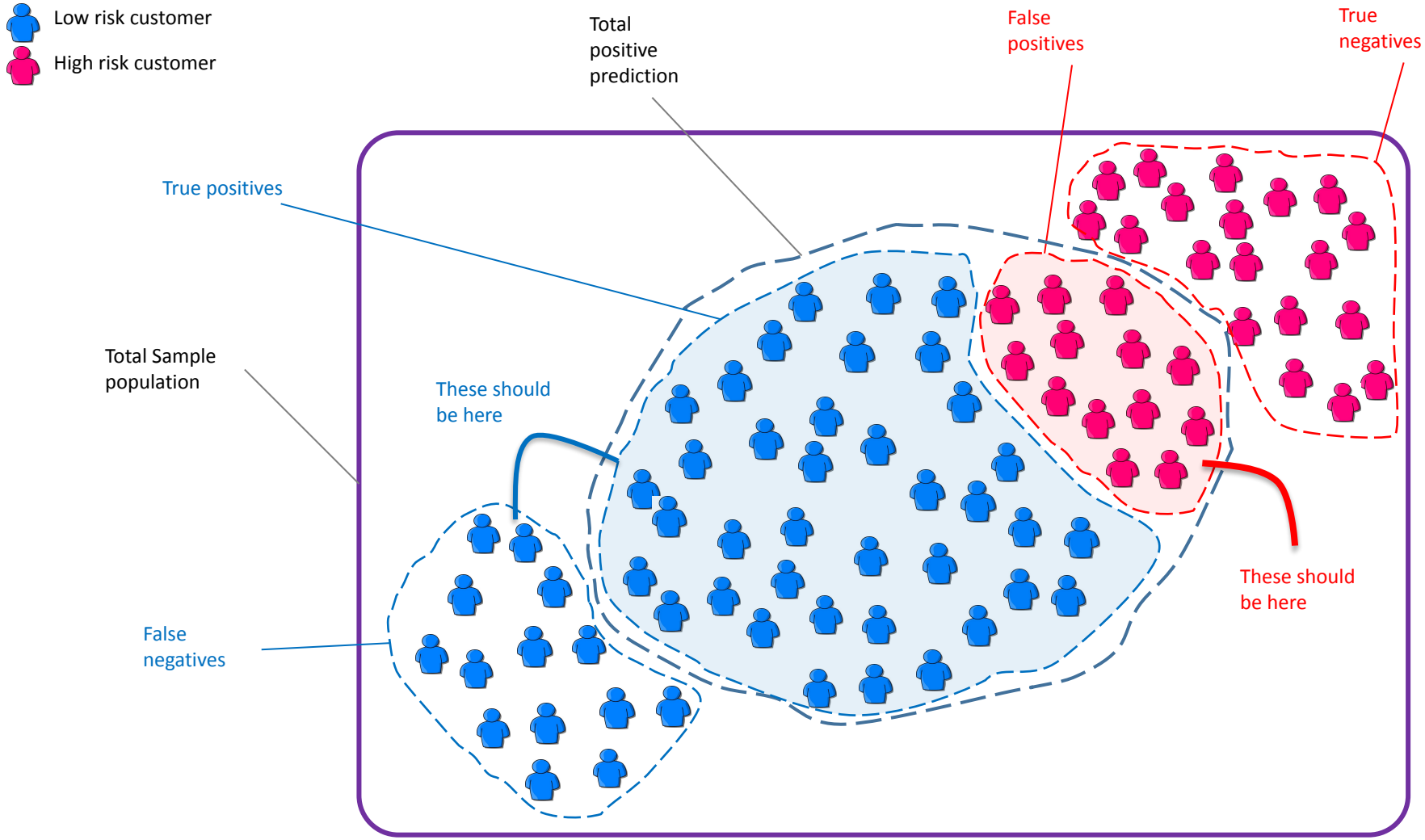
Total true prediction

# Evaluating Classification Results – cont'd

- **Accuracy**: the fraction of predictions that the model got right. i.e. Number of correct prediction / total prediction
  - Accuracy = Total true prediction / Total sample count
- **Precision**: how many of the prediction were correct. It is a measure of **exactness**.
  - Precision = TP / Total positive prediction (TP + FP)
- **Recall**: how many of the actual positive did the model predict. It is a measure of **completeness**.
  - Recall = TP / Total actual positive (TP + FN)

| Actual | Positive | Negative | |
|---|---|---|---|
| Prediction | | | |
| Positive | True Positive (TP) | False Positive (FP) | Total predicted positive |
| Negative | False Negative (FN) | True Negative (TN) | Total predicted negative |
| | Total actual positive | Total actual negative | Total Sample count |

Total true prediction




| Actual | Positive | Negative | |
|---|---|---|---|
| Prediction | | | |
| Positive | True Positive (TP) | False Positive (FP) | Total predicted positive |
| Negative | False Negative (FN) | True Negative (TN) | Total predicted negative |
| | Total actual positive | Total actual negative | Total Sample count |

Total true prediction

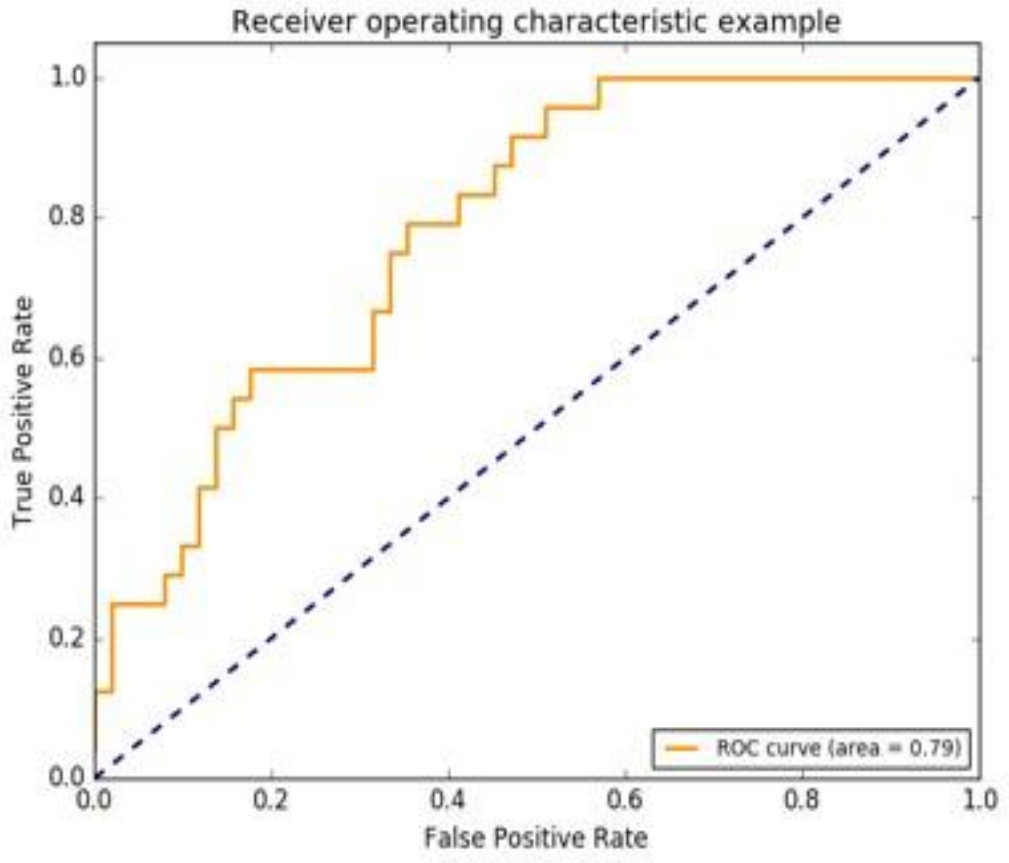
Note: icons are positioned where they are for illustration purpose only.

**Accuracy = Total true prediction / Total sample count**
**Precision = TP / Total positive prediction (TP + FP)**
**Recall = TP / Total actual positive (TP + FN)**

# Evaluating Classification Results – cont'd

***Receiver operating characteristics (ROC) curve***

- Compares True Positive Rate and False Positive Rate
- **True Positive Rate (TPR)** = Recall
- **False Positive Rate (FPR)** is FP/ Total negative count (FP+ TN)
- ROC plots **TPR** vs **FPR** by varying threshold over the entire range of threshold settings. It depicts relative **trade-offs between true positive (benefits) and false positive (costs)**
- **Area Under Curve (AUC)** is equal to the probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative one.



# Evaluating Classification Results – cont'd

- There are many other metrics and many other names for the same metrics
- It is better to stick with a small number of metrics that make sense in your domain before using other metrics
- **Accuracy, precision, recall** and **AUC** are the most common metrics

# Dummy Variables

How can we use categorical variables in an algorithm that requires numerical predictors?

- ***ordinal categoricals***
  - can be converted to a sequence of integers, if it makes sense to do so

| cold | cool | moderate | warm | hot |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

√

  - the above implies hot < warm < moderate < cool < cold
    *... which makes sense*

# Dummy Variables

How can we use categorical variables in an algorithm that requires numerical predictors?

- ***cardinal categoricals***

| apples | bananas | peaches | oranges | pears |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

> *this implies pears > oranges > peaches > bananas > apples ... does not make sense!*

> must convert to dummy variables instead

# Cardinal Dummy Variables

- full definition (number of variables = number of categories):
  - fruit_apples: 1 = apples, 0 = no apples
  - fruit_bananas: 1 = bananas, 0 = no bananas
  - fruit_peaches: 1 = peaches, 0 = no peaches
  - fruit_oranges; 1 = oranges, 0 = no oranges
  - fruit_pears; 1 = pears, 0 = no pears
- compact definition (number of variables is one less than number of categories):
  - fruit_bananas: 1 = bananas, 0 = apples
  - fruit_peaches: 1 = peaches, 0 = no peaches
  - fruit_oranges; 1 = oranges, 0 = no oranges
  - fruit_pears; 1 = pears, 0 = no pears

Python:

pandas.get_dummies

# Lab 5.1: Logistic Regression

- Purpose:
  - To predict survival amongst Titanic passengers using the LogisticRegression() method of Scikit-Learn
- Materials:
  - ‘Lab 5.1.ipynb’

# Discussion

- Strengths & weaknesses of logistic regression

Institute of Data

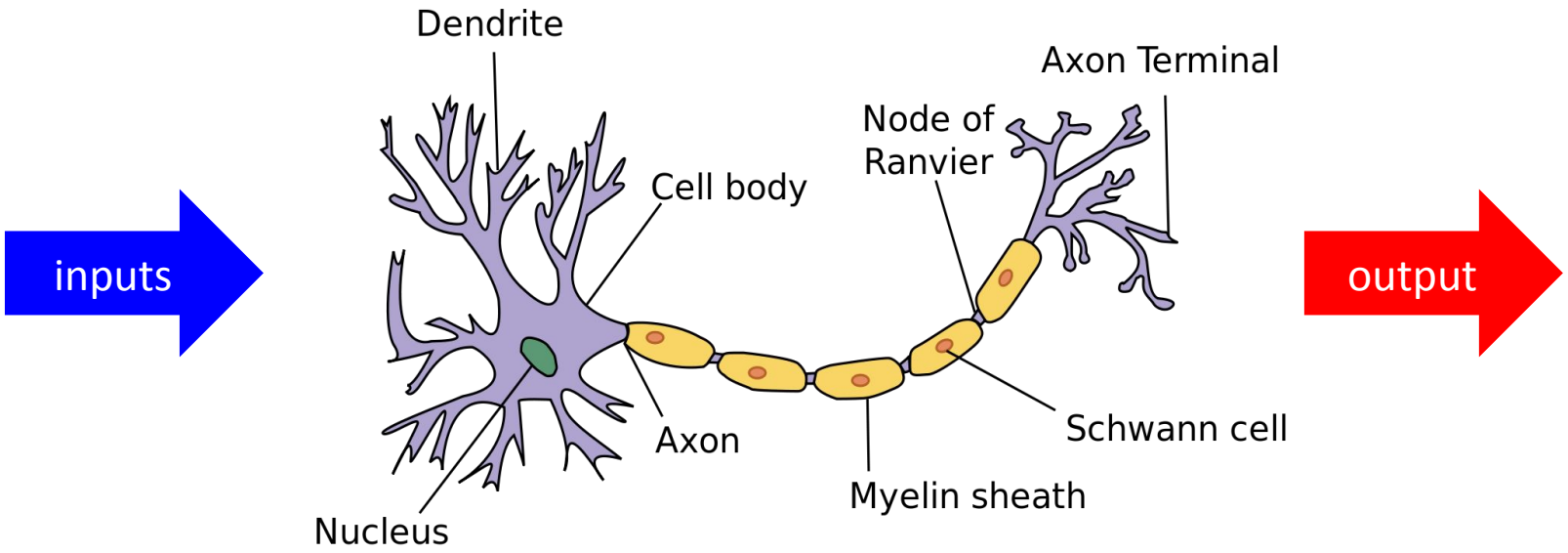
# Perceptron (Neural Networks)

- **Biological** and **artificial** neurons
- **Activation** functions
- **Linear regression** with a perceptron
- **Linear classification** with a perceptron
- **Gradient descent**
- Practical implementations

# How does a nerve cell make a decision?

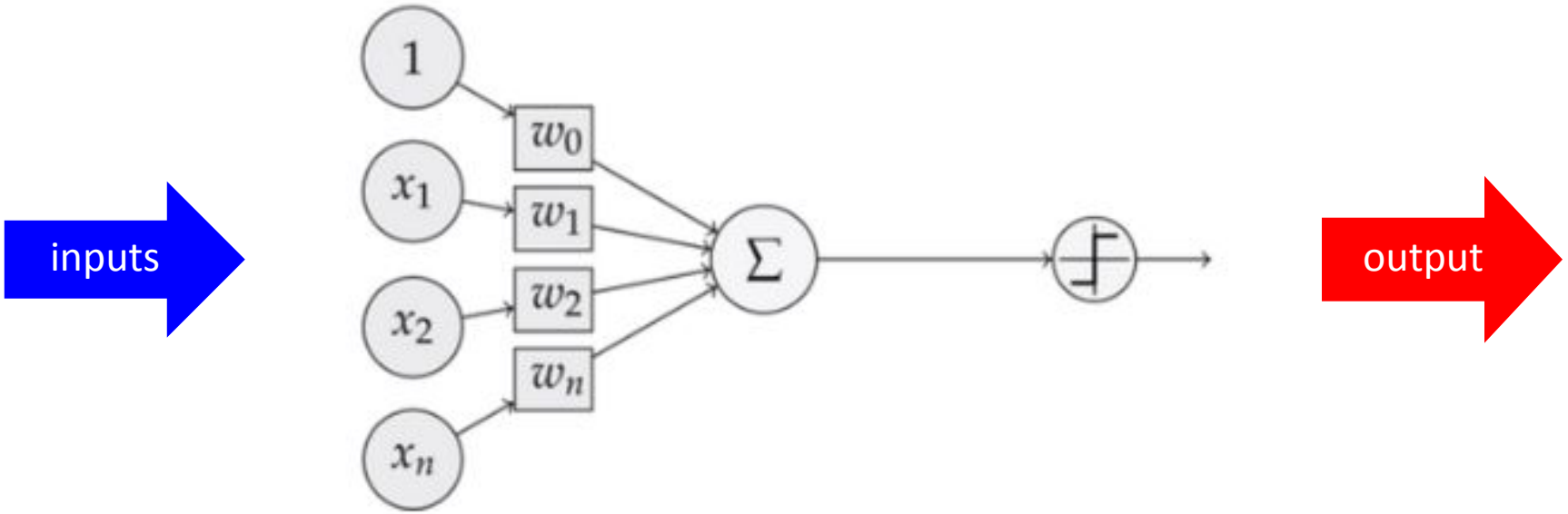
- Neuron receives inputs at antennae-like structures (‘**dendrites**’)
- Each incoming connection is **dynamically strengthened or weakened** by:
  - frequency of use (‘weighting’)
  - neurotransmitters

- **Weighted inputs** are summed in the cell body (transformed into a new signal)
- New signal is **propagated** along the cell’s axon to be detected by other neurons



# The Perceptron

- A **basic imitation** of the natural neuron
- Inputs are given **weights**
- **Weighted** signals are summed
- summed signal is transformed by an **activation function** to produce an output



# Perceptron Activation Function

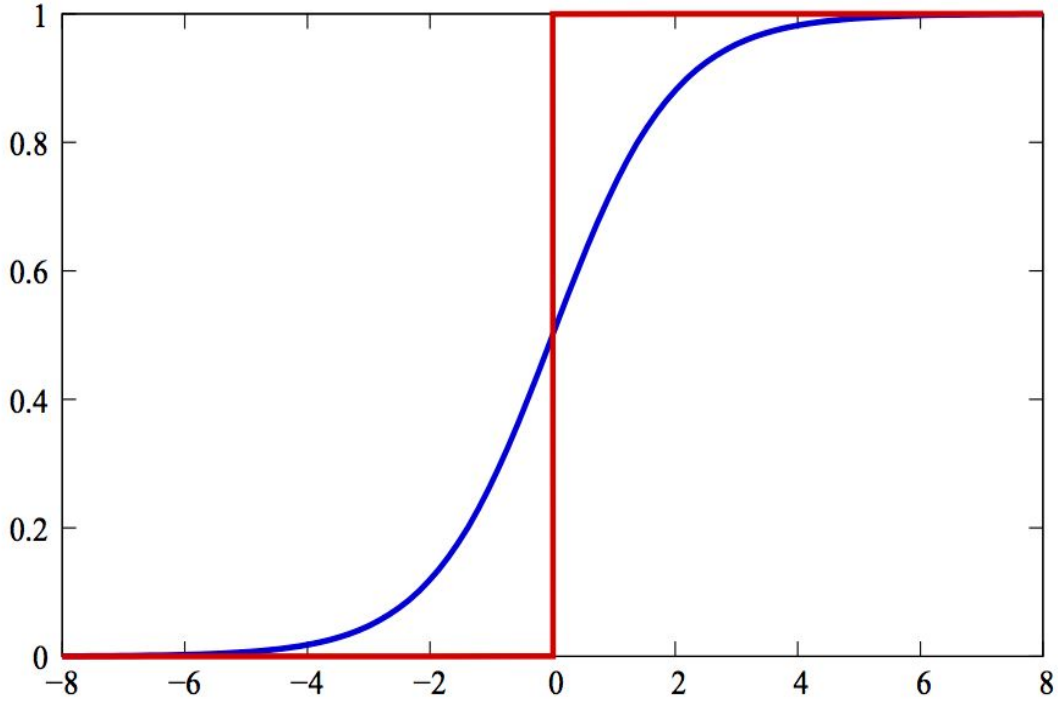
- z is the **weighted sum of inputs** (similar to Logistic Regression):

$$z = \sum_{j=0}^{n} w_j \, x_j$$

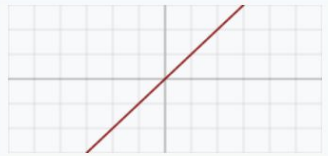
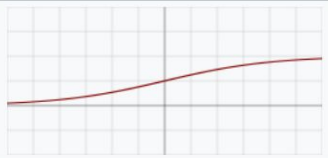
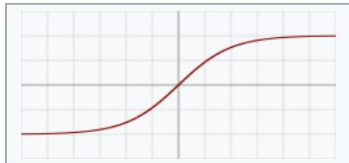
- A transfer function f(z) converts z to the output of the node
- f(z) is called the **activation function**

*example:*

$$f_{log}(z) = \frac{1}{1 + e^{-z}}$$



# Perceptron Activation Functions

| | |
|---|---|
|  | $f(x) = x$ |
|  | $f(x) = \frac{1}{1+e^{-x}}$ |
|  | $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ |
|  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |

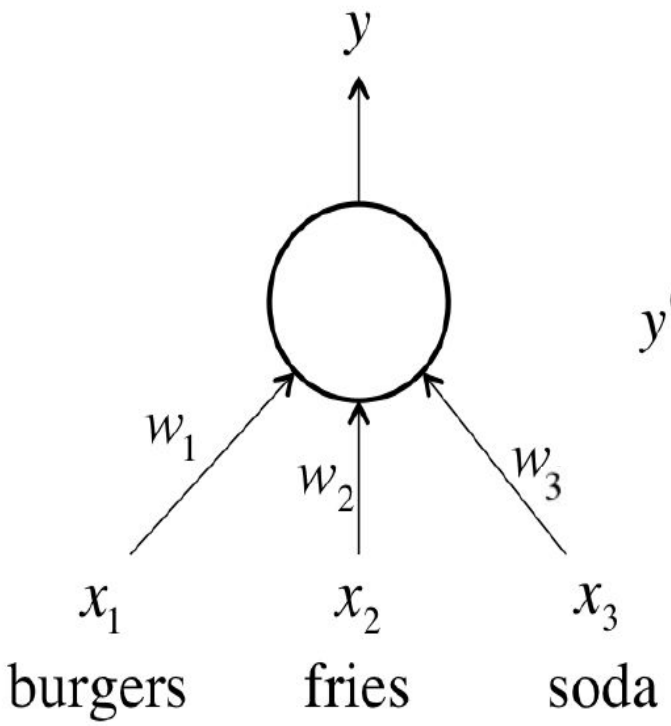
- Linear
- Logistic
- Hyperbolic tangent
- **Rectified linear unit (ReLU)**

# Linear Regression Perceptron

- Example: given a set of fast food orders and total prices, can we predict the unit prices?
- let

  $x_1$ = number of burgers
  $x_2$ = number of fries
  $x_3$ = number of sodas
  $y$ = total price
  for any order
- object:

  compute the weights
  $w_1, w_2, w_3$
  that minimise the residuals of $y$




$$y^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + w_3 x_3^{(i)}$$

# Linear Classification Perceptron

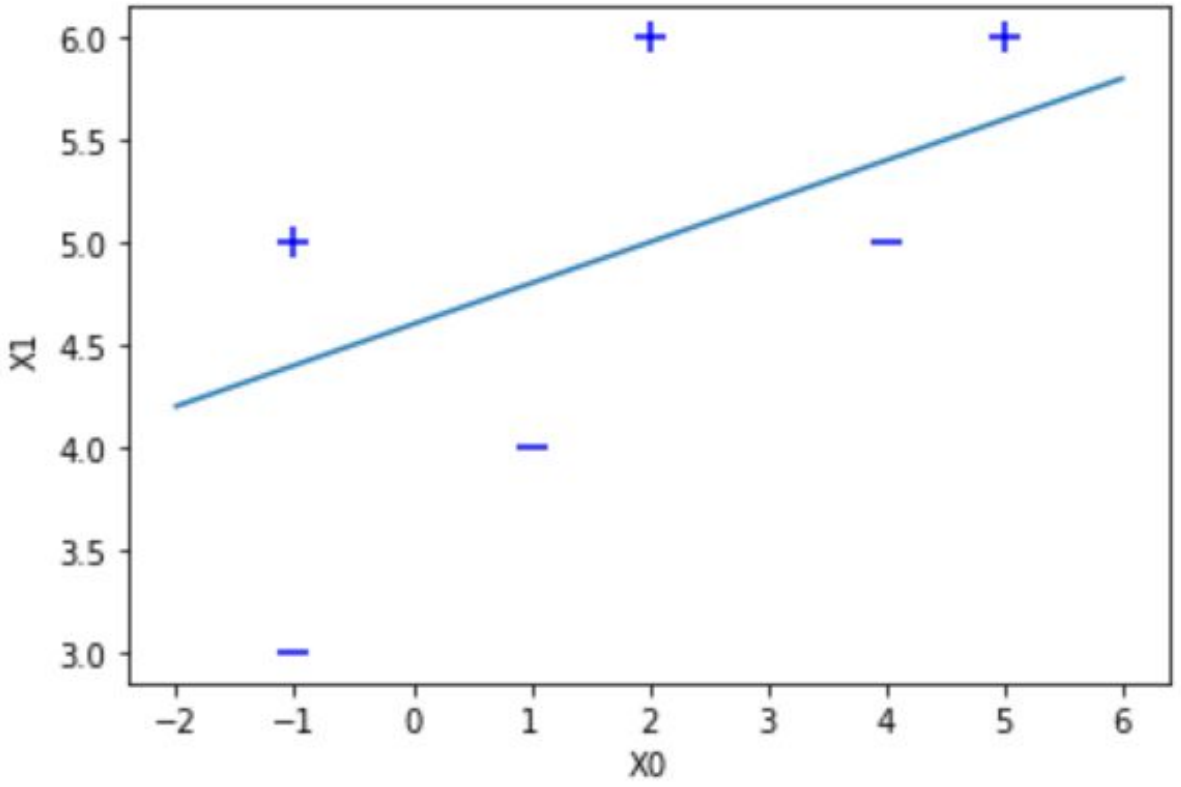
- Given
  - a set of *n*-D **features** $X$
  - a set of **labels** (response var) $y$
- Objective:

  compute the weights $w_1, w_2, \ldots w_n$

  that describe the ***hyperplane*** that separates points $X$ by class $y$

  *example*: labels $y \in \{-1, 1\}$

***SEE LAB 5.2***



# Learning the weights

How to **compute the weights**?

- Iteratively **adjust the weights** over the entire dataset until the algorithm converges to a solution:

  error (at output) < *tolerance*

# How to implement this?

## Gradient descent

- Error at a node is a function of the weights on the inputs flowing into the node
- Use gradient of error function to step in direction of decreasing error until minimum is found
- Update weight with a Learning Rate times the error and direction of the error (which is the derivative of the cost function)
- w = w + Learning Rate * (expected - predicted) * x

# The Perceptron in Practice

- in the *burgers/fries/sodas* example there should be an exact solution
  - (unless the unit process vary over the course of the dataset, in which case we would end up predicting the average price of each item)

- in general ...
  - the data will not permit an exact solution
  - presence of local minima may cause convergence to fail

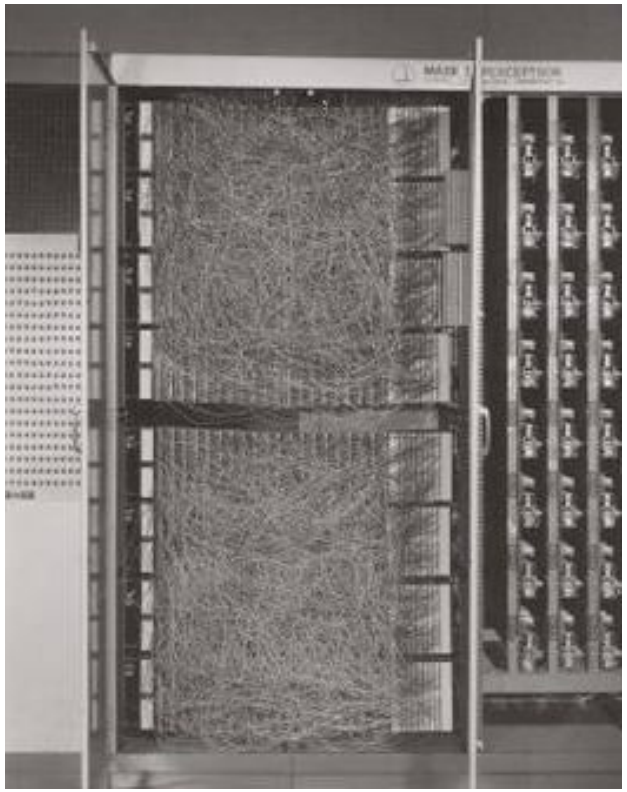
- in practice ...
  - use many perceptrons (***nodes***), organised into ***layers*** to learn more complex problems

    > neural network and Deep Learning

# Lab 5.2: Classification with a Perceptron

- Purpose:
  - To evaluate a simple perceptron for predicting classes from numeric features.

- Materials:
  - ‘Lab 5.2.ipynb’



*Mark 1 Perceptron*

By Source (WP:NFCC#4), Fair use, https://en.wikipedia.org/w/index.php?curid=47541432

# Discussion

- Strengths & weaknesses of perceptron-based prediction

Institute of Data

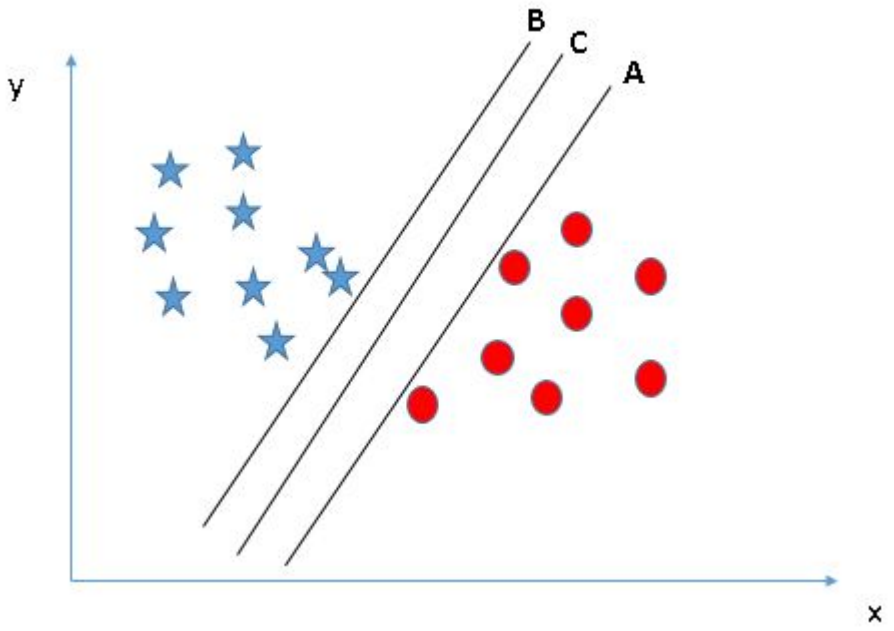
# Support Vector Machines

- Concepts
- Linear SVMs
- Nonlinear SVMs
- Limitations
- Applications

# Support Vector Machines

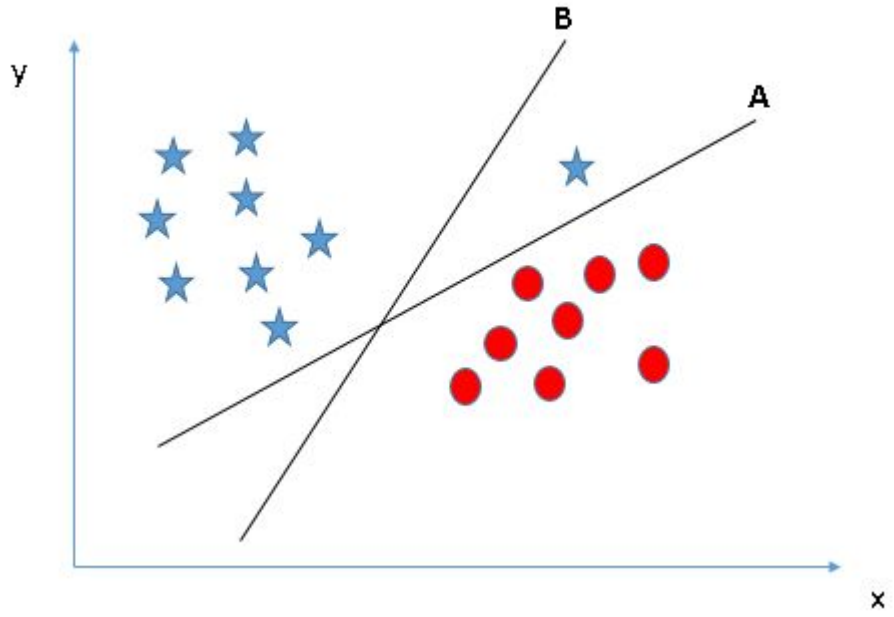
- A **linear** algebraic method for separating ***n*-dimensional** data into classes
  - Data points are separated by a ***hyperplane*** (i.e. a boundary that has dimensionality $n-1$)

*2D example:*




- lines A, B, C are **hyperplanes** in a 2D space
- each line correctly separates the two classes
- line C is preferred, because it **maximises** the average squared distance (***margin***) between the boundary and the points

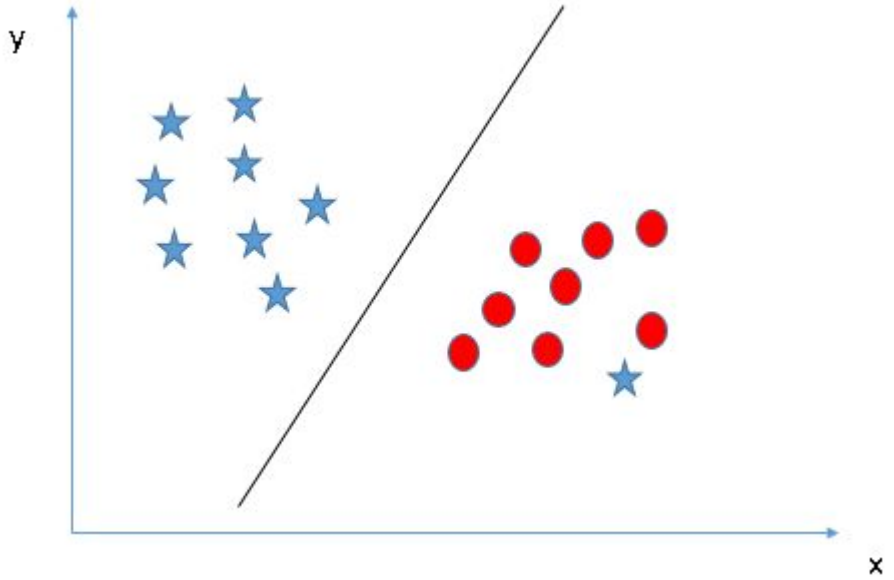
# Support Vector Machines – cont'd




- line B gives a larger margin
- line A correctly separates the classes

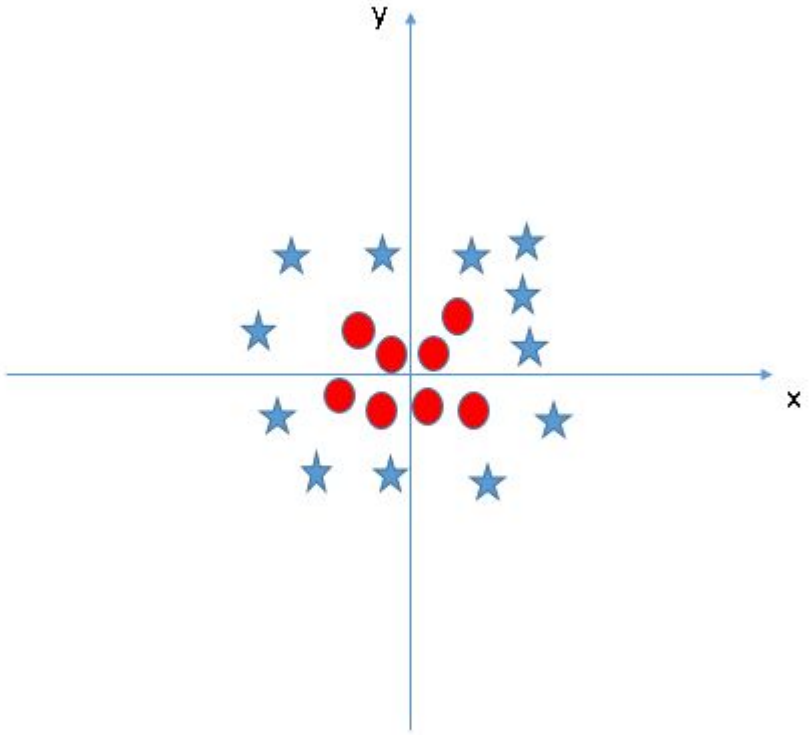
- line B will be chosen
  - **maximum separation** of classes is the first priority

# Support Vector Machines – cont'd




- the line separates the classes except for one outlier
- the presence of the outlier does not shift the line to the right
  - **SVMs are robust against outliers**

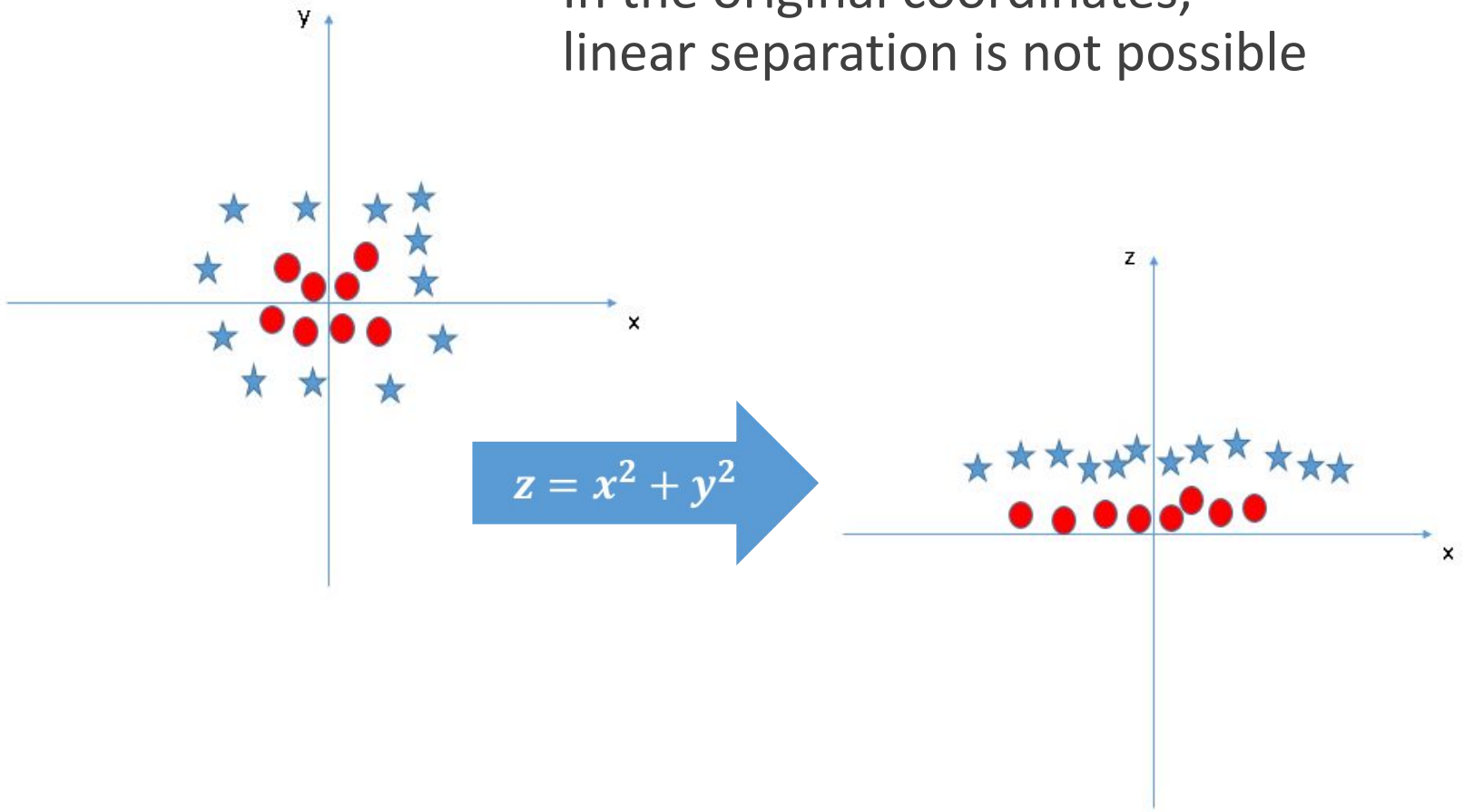
# Support Vector Machines – cont'd




- there is no line that can separate these classes
  - will an SVM fail, here?
  - *Hint: we could **transform** the coordinates or create a new feature*

# Support Vector Machines – cont'd

- In the original coordinates, linear separation is not possible




- in these **coordinate**, there *is* a line that can separate the classes

  **>** the **kernel trick**:

  - an inseparable problem can be transformed into a separable problem in a higher dimension
  - the transform function is called the ***kernel***

# Linear SVMs

Hard margin

- classes are separable by the margin around the solution hyperplane

Soft margin

- classes are not separable
- the best-fit hyperplane is computed by minimising the *hinge loss* function
  - incorporates a parameter that defines the trade-off between maximising the margin and minimising the number of misclassified points

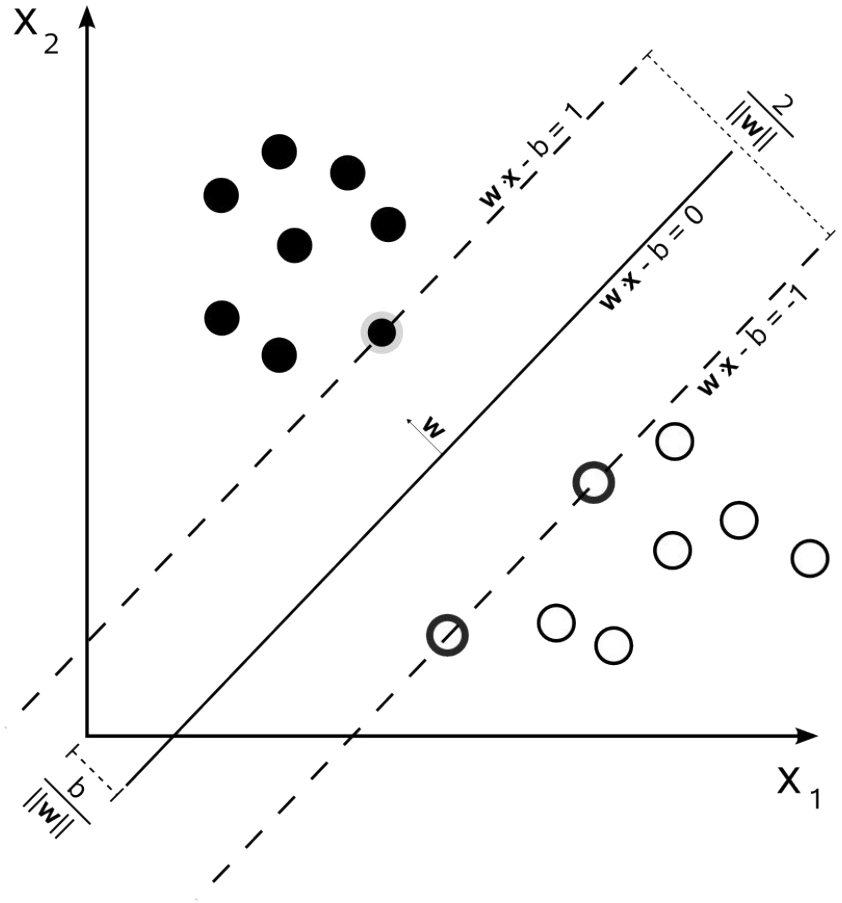
# Linear SVM with Hard Margin

- equation of hyperplane is

$$\vec{w} \cdot \vec{x} - b = 0$$

- compute $\vec{w}$, $b$ so as to maximise width of margin

$$2/\|\vec{w}\|$$



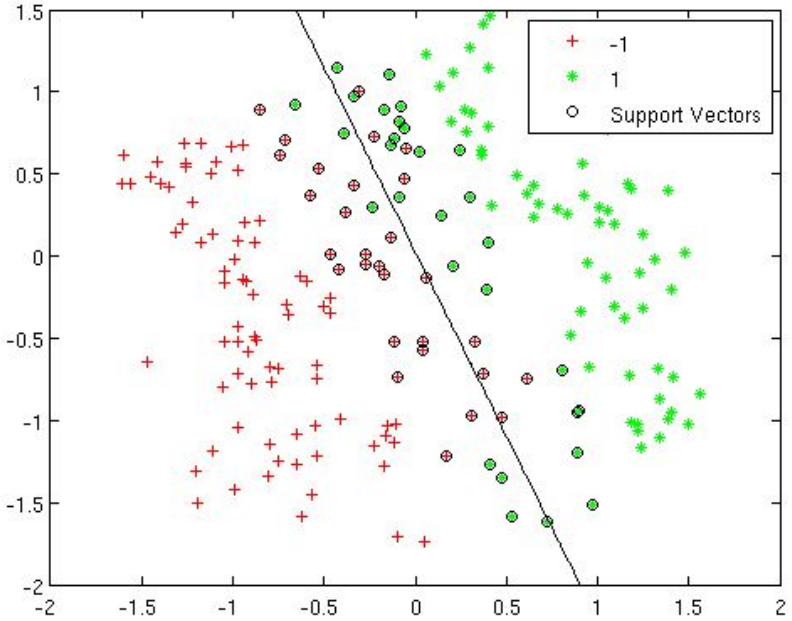
# Linear SVM with Soft Margin

- equation of hyperplane is

$$\vec{w} \cdot \vec{x} - b = 0$$

- compute $\vec{w}$, $b$ so as to minimise

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w}\cdot\vec{x}_i - b)\right)\right] + \lambda\|\vec{w}\|^2$$




Nb. This plot also shows that only the nearest points are used to compute the boundary. These points are called the ***support vectors***.

# Nonlinear SVMs

- Polynomial kernel
  - with or without constant term
- Gaussian radial basis function (RBF)
- Hyperbolic tangent

# Limitations of SVMs

- Generalisation error
  - margin is too small (data too close together) to accurately separate all classes
- Requires full labelling of input data
- Class membership probabilities are uncalibrated
- Does not natively handle more than 2 classes
  - multiclass problems must be reduced to a series of binary problems before applying SVMs
- Solution parameters are difficult to interpret

# Applications of SVMs

- Text categorisation
- Image classification
- Handwriting interpretation
- Protein classification
- Customer segmentation

# SVMs in Python

- Predictors **X**
- Response **y**
- Test data **Xtest**

- c = **regularisation** parameter: controls sensitivity to outliers
- gamma = kernel coefficient ('rbf', 'poly' and 'sigmoid' kernels): controls influence of nearby points

```
from sklearn import svm

# Create SVM classification object:
model = svm.svc(kernel = 'linear', c = 1, gamma = 1)

# Train model:
model.fit(X, y)

# Evaluate quality of fit:
model.score(X, y)

#Predict output for test data:
predicted = model.predict(Xtest)
```

# Lab 5.3: Support Vector Machines

- Purpose:
  - To apply the SVM method to linear and nonlinear classification problems.

- Materials:
  - ‘Lab 5.3.ipynb’

# Discussion

- Strengths & weaknesses of SVMs

Institute of Data

# Bayesian Inference

- **Frequentist** vs **Bayesian** probability
- **Bayes' theorem**
- Example: Disease detection
- **Bayesian modelling**
- **Naïve Bayes Classification**

# Frequentist Probability

- Probability:
  - a proportion of outcomes
- In the frequentist approach, we have some **sample results** from which we calculate the **frequency** of the *positive* result
  - we estimate the future probability of a positive result based entirely on this sample

# Bayesian Probability

- Probability:
  - a degree of belief

- In the Bayesian approach, we also have pre-existing information (a sample or distribution) about something that has a causal connection to the thing we are sampling
  - we use this **prior** distribution in addition to the current sample to estimate the future probability of a positive result

# Bayes' Theorem – cont'd

- Suppose event A leads to event B, or that B is seen to be associated with A
- Let

$$p(B|A)$$

be the probability of B given A

- Then

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)}$$

is the probability of A given B

- the probability that A happened given that the event B happened

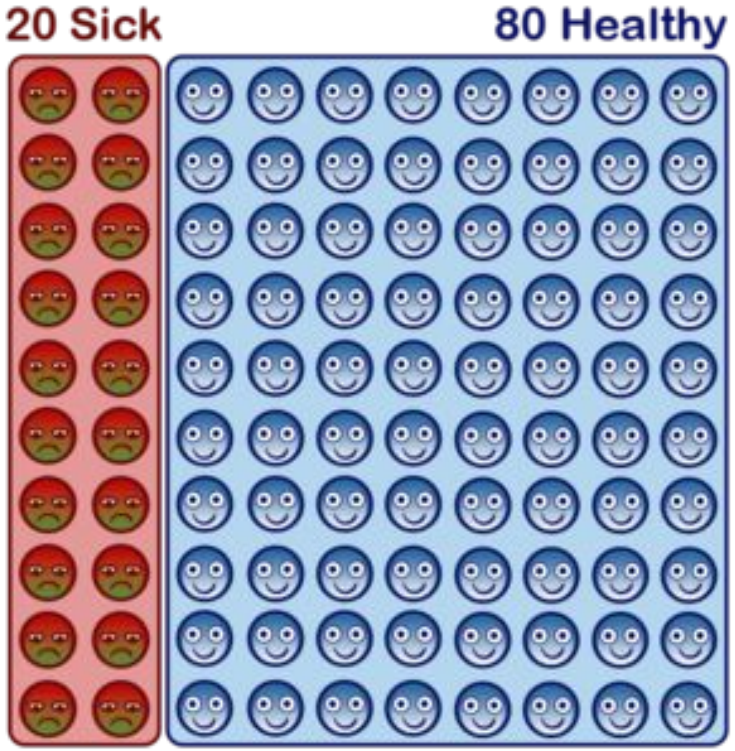
# Bayes' Theorem – cont'd

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)}$$

- $p(A), p(B)$ are **marginal** probabilities
- $p(A|B), p(B|A)$ are **conditional** probabilities
- $p(A)$ is the **prior** probability
- $p(B)$ is the **evidence**

# Example: Disease Detection

- prior knowledge:
  - 20% of population has the disease
    $P / (P + N) = 0.20$
  - in a random sample of 100 people,
    20 will have the disease (on average)




Frequency diagram - Arbital

# Example: Disease Detection

- sensitivity of test:
  - 90% of subjects with disease are correctly detected

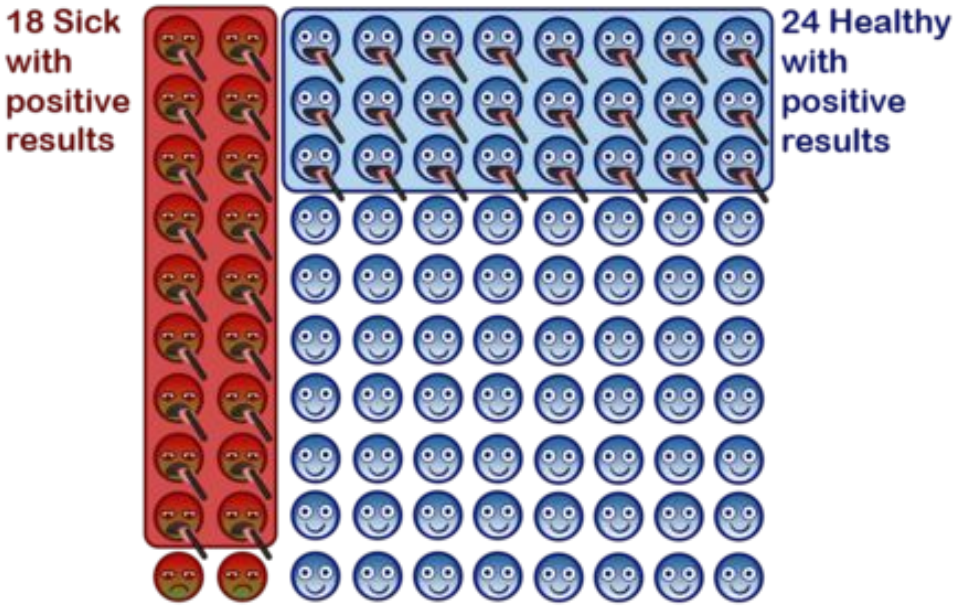
  TPR = 0.90 = TP / (TP + FN)

  TP = 0.90 * 20 = 18

- FPR (1 – specificity) of test:
  - 30% of disease-free subjects are erroneously detected

  FPR = 0.30 = FP / (FP + TN)

  FP = 0.30 * 80 = 24




> *if a person tests positive, what is the probability that they are sick?*

# Example: Disease Detection

- Probability that a subject with a positive test result is actually sick:

$p(\text{sick}|\text{test}^+) = TP / (TP + FP)$

$= 18 / (18 + 24)$

$= 18 / 42$

$\approx 0.43$

Alternatively, using Bayes' Theorem,

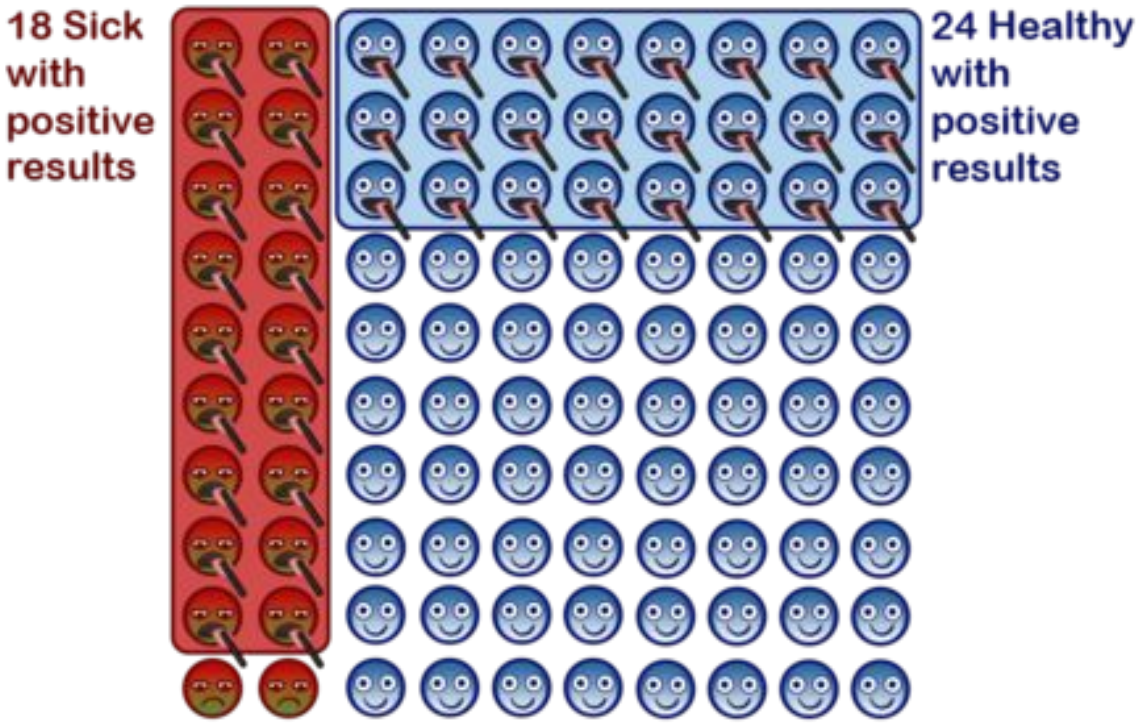
$p(\text{sick}|\text{test}^+)$

$= p(\text{sick})*p(\text{test}^+|\text{sick}) / p(\text{test}^+)$

$= 0.2*0.9 / (0.2*0.9 + 0.3*0.8)$

$= 0.18 / 0.42$

$\approx 0.43$




Frequency diagram - Arbital

# Bayesian Modelling

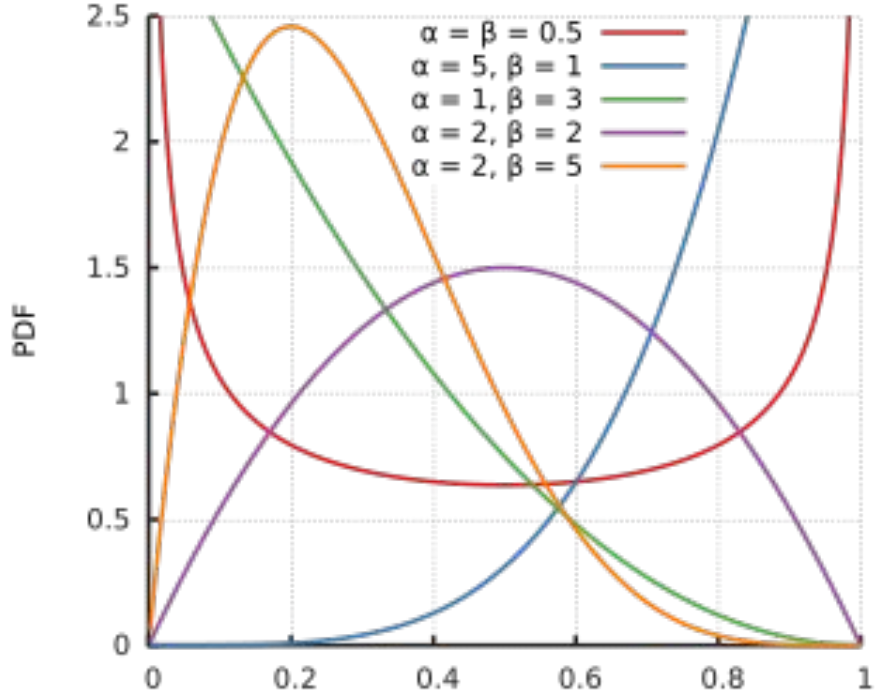
- **Model-based** approach
  - results not dependent on arbitrary ***p*-value, confidence interval**
- **Naïve Bayes** assumption
  - **Predictors are independent**
    - variables come from distributions that do not interact
- prior distribution is not usually known explicitly
  - type of distribution must be assumed:
    - from domain knowledge
    - by choosing a mathematically sound distribution
  - parameters of distribution are fitted to the prior data

# How to Choose the Prior Distribution?

- Informative, **empirical**:
  - Data from **related experiments** informs our prior beliefs
  - Prior beliefs will influence final predictions
- Informative, **non-empirical**:
  - Prefer certain values over others (e.g. need to **regularise coefficients**)
  - Prior beliefs will influence final predictions
- Informative, **domain-knowledge**:
  - No supporting data, but certain facts are known to be more true than others
  - Prior beliefs will influence final predictions
- **Non-informative**:
  - Prior beliefs will have little to no effect on our final assessment
  - Current data alone will determine final predictions

# Beta Distribution

- a probability density function suitable for modelling the random behaviour of percentages and proportions




$$\text{PDF} = \frac{x^{\alpha-1}\,(1-x)^{\beta-1}}{B(\alpha,\beta)}$$

$\alpha, \beta > 0$

$B$ is the beta function

Beta distribution - Wikipedia

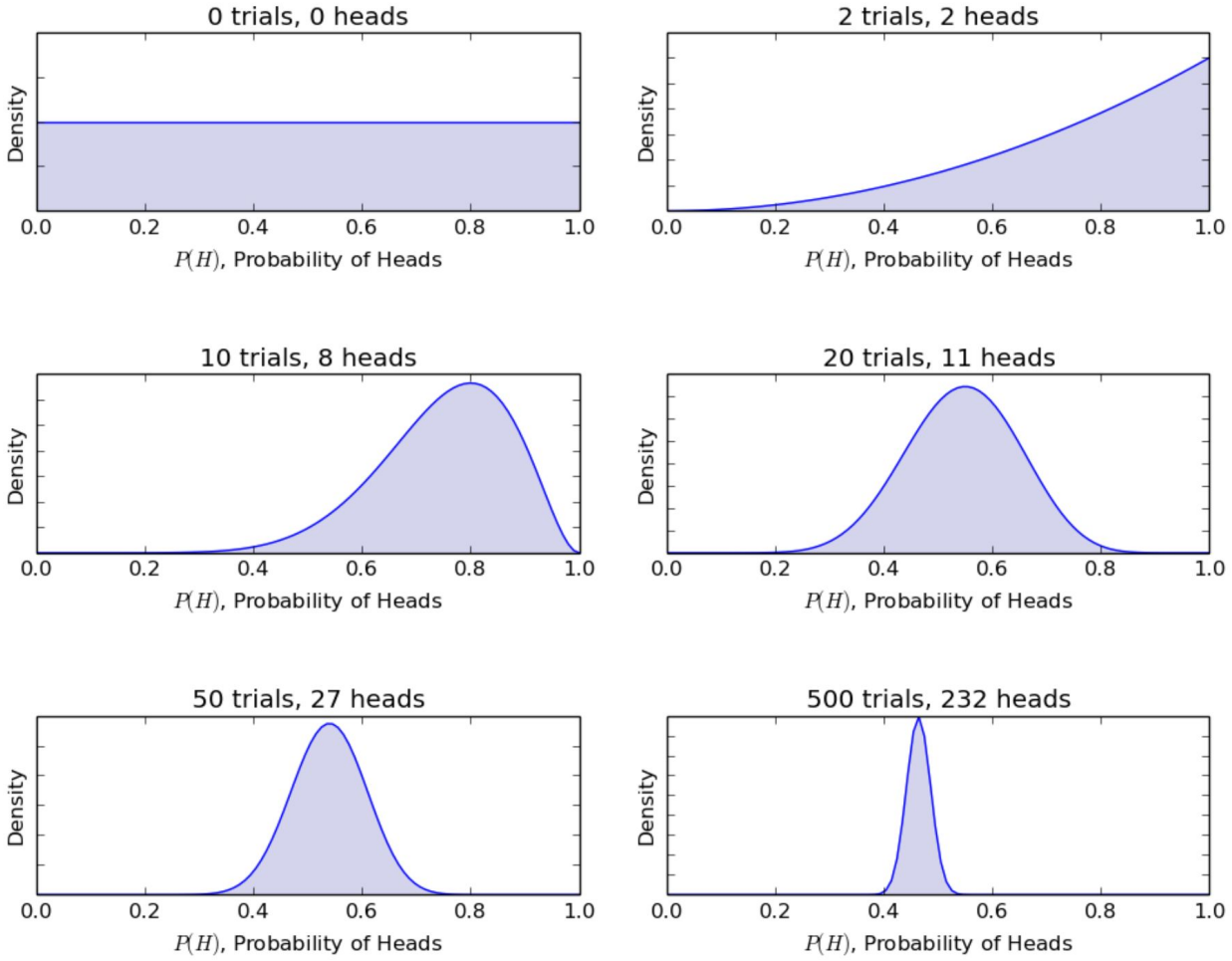
# Bayesian Modelling – cont'd

What can we use for the prior probability if we have no information?

> **a beta distribution**

$$\mu = \frac{\alpha}{\alpha + \beta}$$

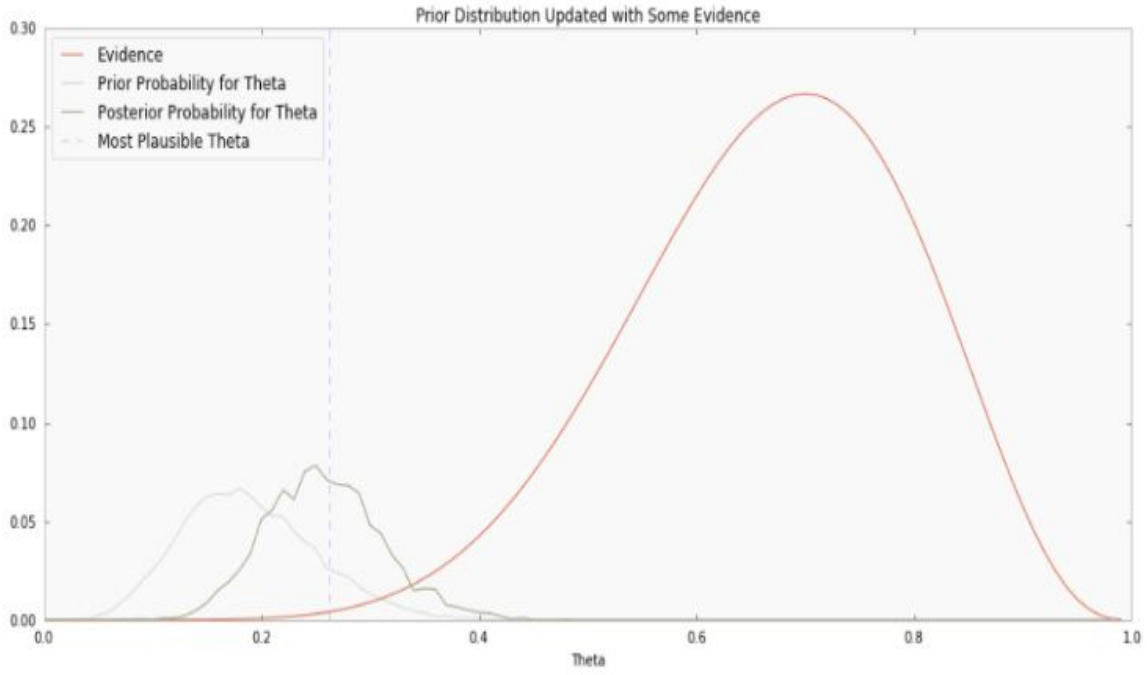
$$\sigma = \sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}$$



# Bayesian Analysis of a Sample

Example:

Campaign to boost click rates on ‘ads’




- new ad yielded 7 clicks from 10 users tested (red line)
  - beta distribution of $\theta$ peaks at 0.7 (most likely value)
- previous 100 campaigns yielded much lower click rates (light grey line)
  - prior $\theta$ peaks $\lesssim 0.17$
- we don’t believe the new result, so we modify it based on our prior information (result = dark grey line)
  - posterior $\theta$ peaks $\lesssim 0.25$

# Naïve Bayes Classification

- Probabilistic classification methods

- Assumptions
  - predictors are independent (hence ‘naïve’)
  - predictors are normally distributed

- Applications
  - text classification (spam, topic)
  - medical diagnosis
  - fraud detection
  - insurance risk category

# Naïve Bayes Classification – cont'd

- Advantages
  - Algorithms **scale linearly** with number of variables
  - Uses marginal distributions of variables

- Disadvantages
  - **Correlated features** bias the model
  - Absolute probabilities cannot be relied upon
    - only the **probability rankings** should be used
  - Assigns zero probability if a new category appears in test data
    - training data must span all possible levels of categorical features
      or
    - apply a smoothing technique (Sklearn uses Laplace estimation)

# Naïve Bayes Classification in Python

**sklearn.naive_bayes**

GaussianNB
- assumes normally distributed features

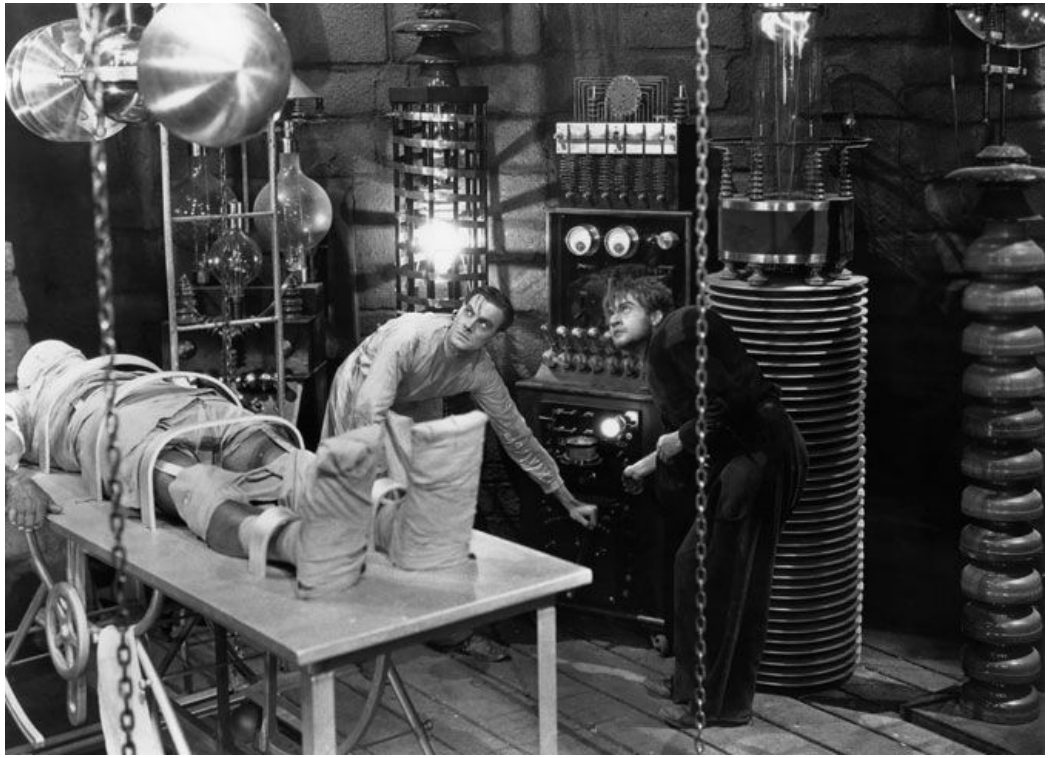
BernoulliNB
- binary/boolean features

MultinomialNB
- discrete features (multinomial distribution)
  - e.g. word counts for text classification
- uses smoothing to account for features not present in the training set

# Lab 5.3.1: Comparison of Classification Methods

- Purpose:
  - To compare classification methods in Python, including Naïve Bayes
  - To apply grid search to find optimal parameters

- Materials:
  - ‘Lab 5_3_1.ipynb’



# Discussion

- more on Bayesian inference:
  - Power of Bayesian Statistics & Probability | Data Analysis (Updated 2023)

- to be covered in a future module:
  - decision trees (nonlinear classification methods)

# Questions?

# Appendices

# End of Presentation!