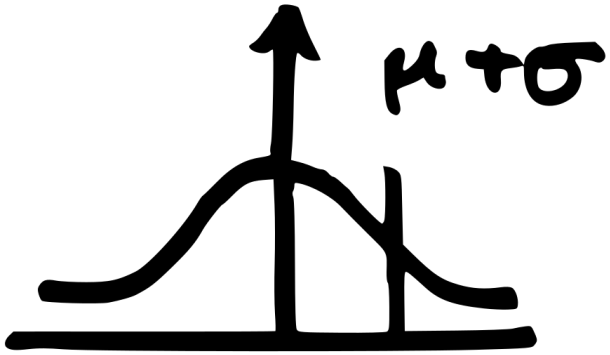
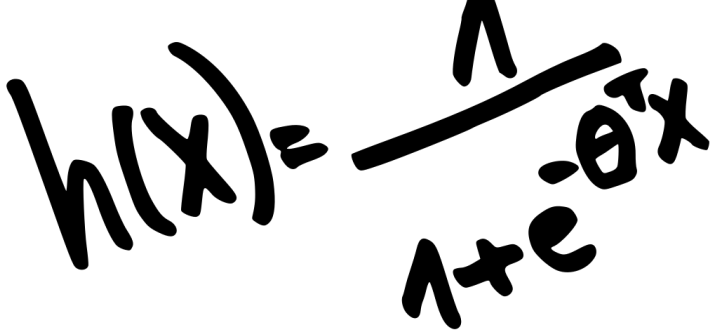
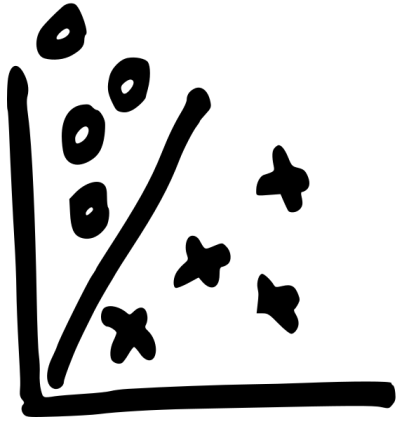


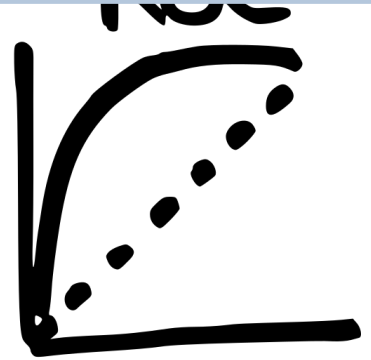


Sequence algorithms

DAG workshop, 2020

Cyril Matthey-Doret





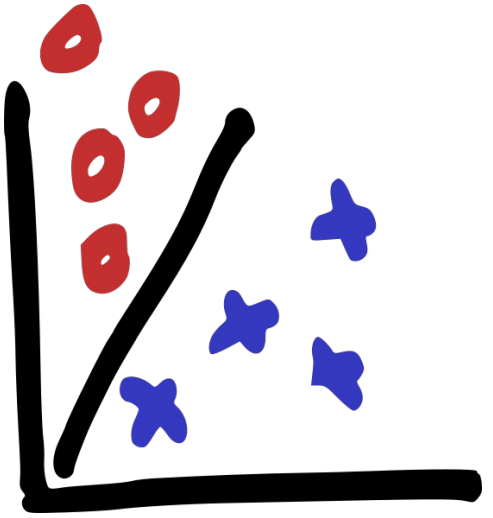
cyril.mattheydoret@gmail.com
cmdoret

Exercises from session 2

The classification problem

- We want to assign a label to values
- “Supervised”: We need to learn using known examples

ML Tasks *Broad Categories*	***Supervised***	***Unsupervised***
Discrete	Classification Computer vision | Image Classification Speech, handwriting recognition Drug discovery	Clustering K-means, mean-shift Large-scale clustering problem Hierarchical clustering, GMM
Continuous	Regression Computer vision | Object Detection Linear, logistic regression	Reduction of Dimensionality PCA, LDA (Kernel) Density Estimation



Classification in biology

- Many uses of classification:
  - Gene expression → Cancer / healthy
  - Protein sequence → localisation (nuclear/membrane/…)
  - Medical data (blood pressure, weight) → disease status
  - Histology pictures → cell type
  - ...

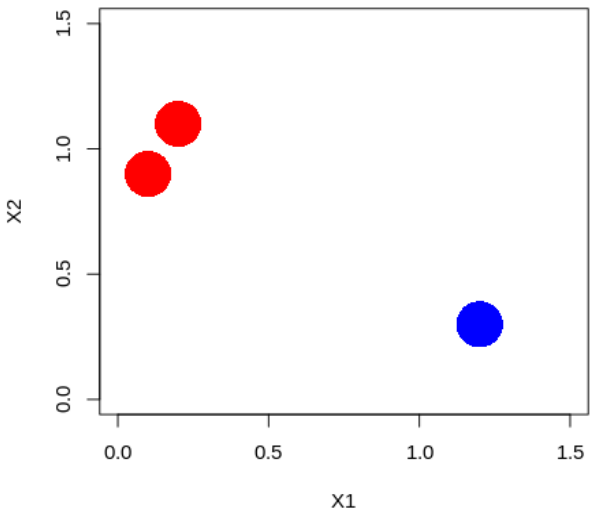
Classification: general formulation

- Given **p** input features (variables), X_1, …, X_p, of **N** values
- We want a function f(X) which accurately predicts labels y

p=2

N=3

X1	X2	y
1.2	0.3	0
0.1	0.9	0
0.2	1.1	1



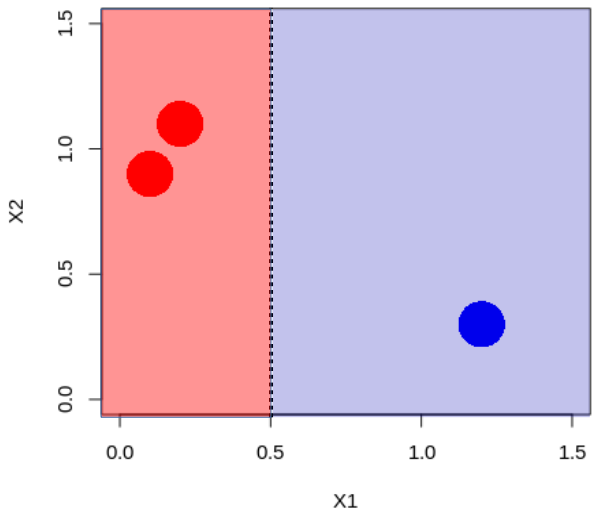
Classification: general formulation

- Given **p** input features (variables), X_1, …, X_p, of **N** values
- We want a function f(X) which accurately predicts labels y

p=2

N=3

X1	X2	y
1.2	0.3	0
0.1	0.9	0
0.2	1.1	1



$$f(x) = \begin{cases} 1 & if \quad x_1 > 0.5 \\ 0 & if \quad x_1 \leq 0.5 \end{cases}$$

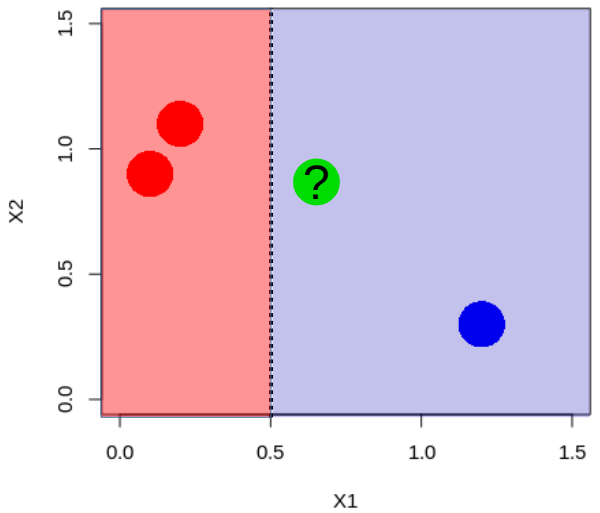
Classification: general formulation

- Our function is trained (fitted) on known values in X, y
- It should be able to predict new values robustly

p=2

N=3

X1	X2	y
1.2	0.3	0
0.1	0.9	0
0.2	1.1	1



$$f(x) = \begin{cases} 1 & if \quad x_1 > 0.5 \\ 0 & if \quad x_1 \leq 0.5 \end{cases}$$

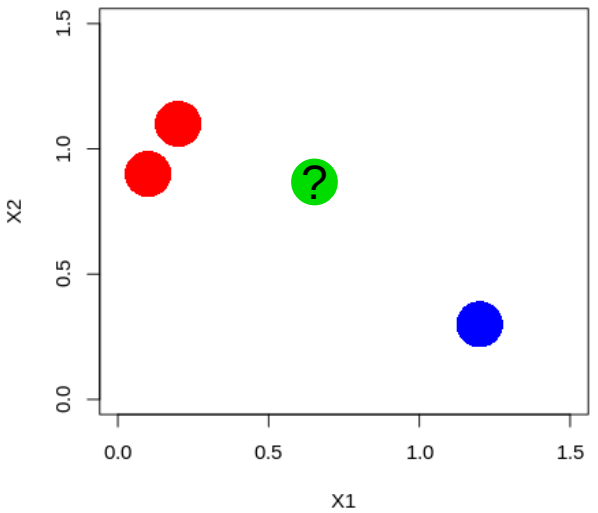
K-nearest neighbours (KNN)

- The predicted label of x is the average of its K closest neighbours
- For more than two labels (e.g. A, B, C), we use a majority vote

p=2

N=3

X1	X2	y
1.2	0.3	0
0.1	0.9	0
0.2	1.1	1



$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

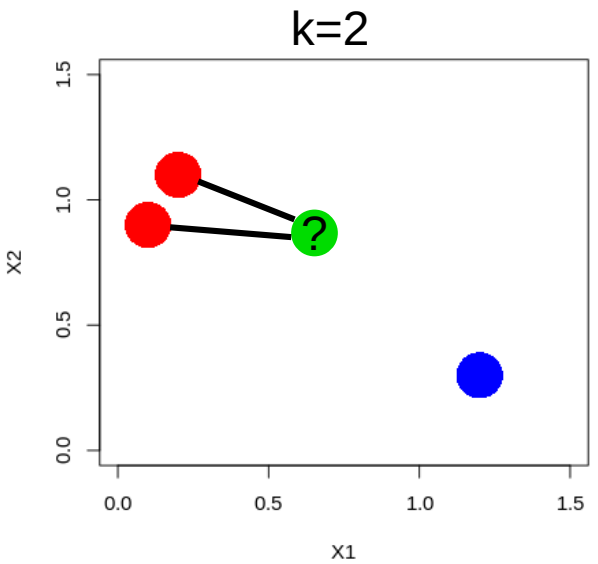
K-nearest neighbours (KNN)

- The predicted label of x is the average of its K closest neighbours
- For more than two labels (e.g. A, B, C), we use a majority vote

p=2

N=3

X1	X2	y
1.2	0.3	0
0.1	0.9	0
0.2	1.1	1



$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

K-nearest neighbours (KNN)

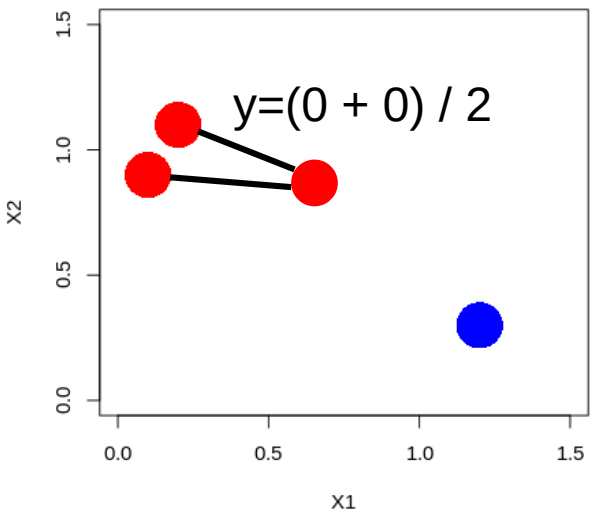
- The predicted label of x is the average of its K closest neighbours
- For more than two labels (e.g. A, B, C), we use a majority vote
- No explicit training step required

p=2

N=3

X1	X2	y
1.2	0.3	0
0.1	0.9	0
0.2	1.1	1

k=2



$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

Speed of KNN

- We have a dataset of N samples with p features
- Time complexity of predicting the label for one new value ?

Speed of KNN

```
from math import sqrt, inf
from statistics import mean
dist = [0 for i in range(N)]
neigh = [0 for i in range(N)]

for i in range(N):
    for j in range(p):
        dist[i] += (X[i, j] - P[j])**2
    dist[i] = sqrt(dist[i])

for k in K:
    min_d, min_d_idx = inf, -1
    for i in range(N):
        if dist[i] < min_d and neigh[i] == 0:
            min_d, min_d_idx = d[i], i
    neigh[min_d_idx] = 1

y_neigh = [y[i] for i in range(len(neigh)) if neigh[i]==1]
y_pred = mean(y_neigh)
```

Speed of KNN

```
1   from math import sqrt, inf
2   from statistics import mean
3   dist = [0 for i in range(N)]
4   neigh = [0 for i in range(N)]
5
6   for i in range(N):
7       for j in range(p):
8           dist[i] += (X[i, j] - P[j])**2
9       dist[i] = sqrt(dist[i])
10
11  for k in K:
12      min_d, min_d_idx = inf, -1
13      for i in range(N):
14          if dist[i] < min_d and neigh[i] == 0:
15              min_d, min_d_idx = d[i], i
16      neigh[min_d_idx] = 1
17
18  y_neigh = [y[i] for i in range(len(neigh)) if neigh[i]==1]
19  y_pred = mean(y_neigh)
```

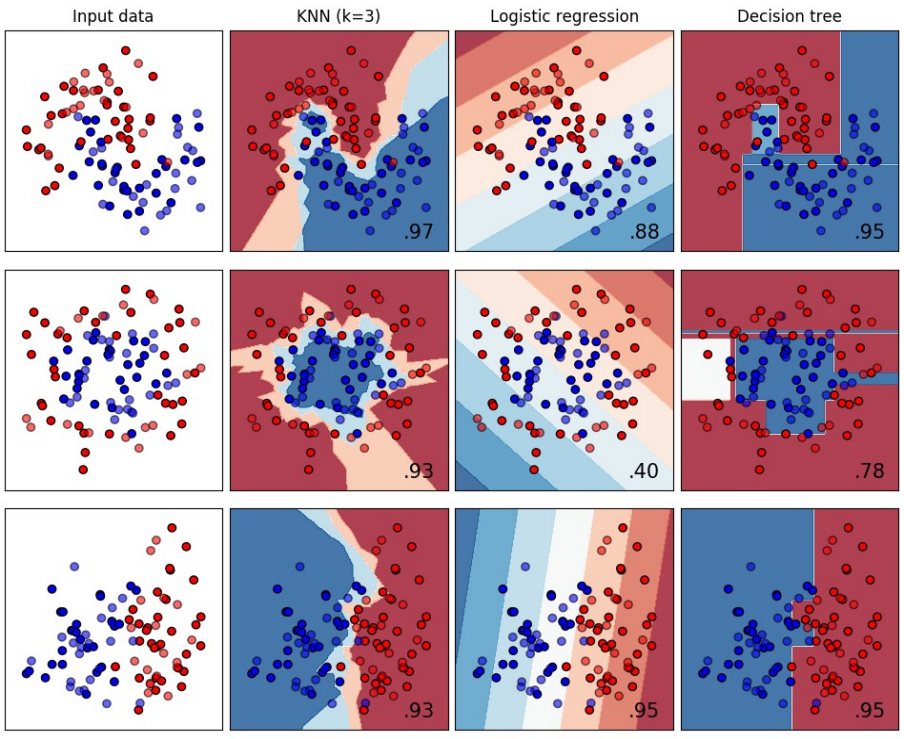
O(N*p)

O(K*N)

Predicting **a single** value takes **O(np + nk)**

Other types of classifiers

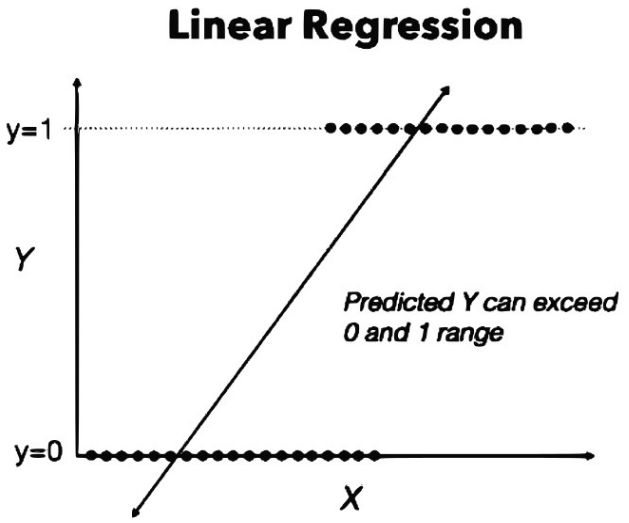
- Most classifiers can predict values in O(1) once they are trained
- Many types of models available
- Vary in complexity and training time

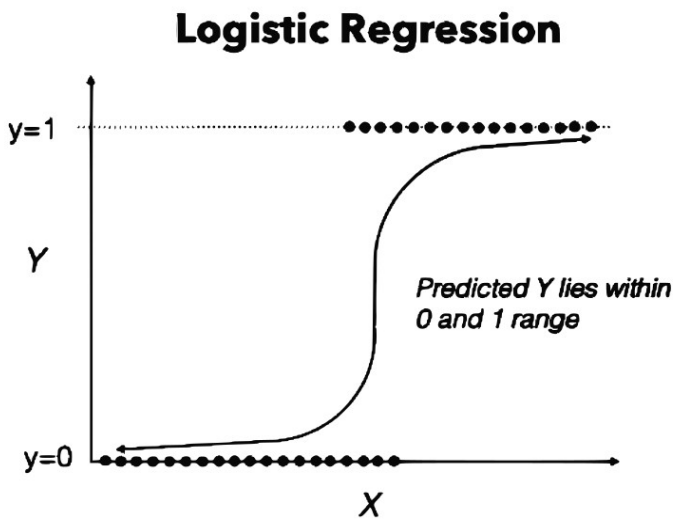


Logistic regression

Similar idea to linear regression, but output is a probability

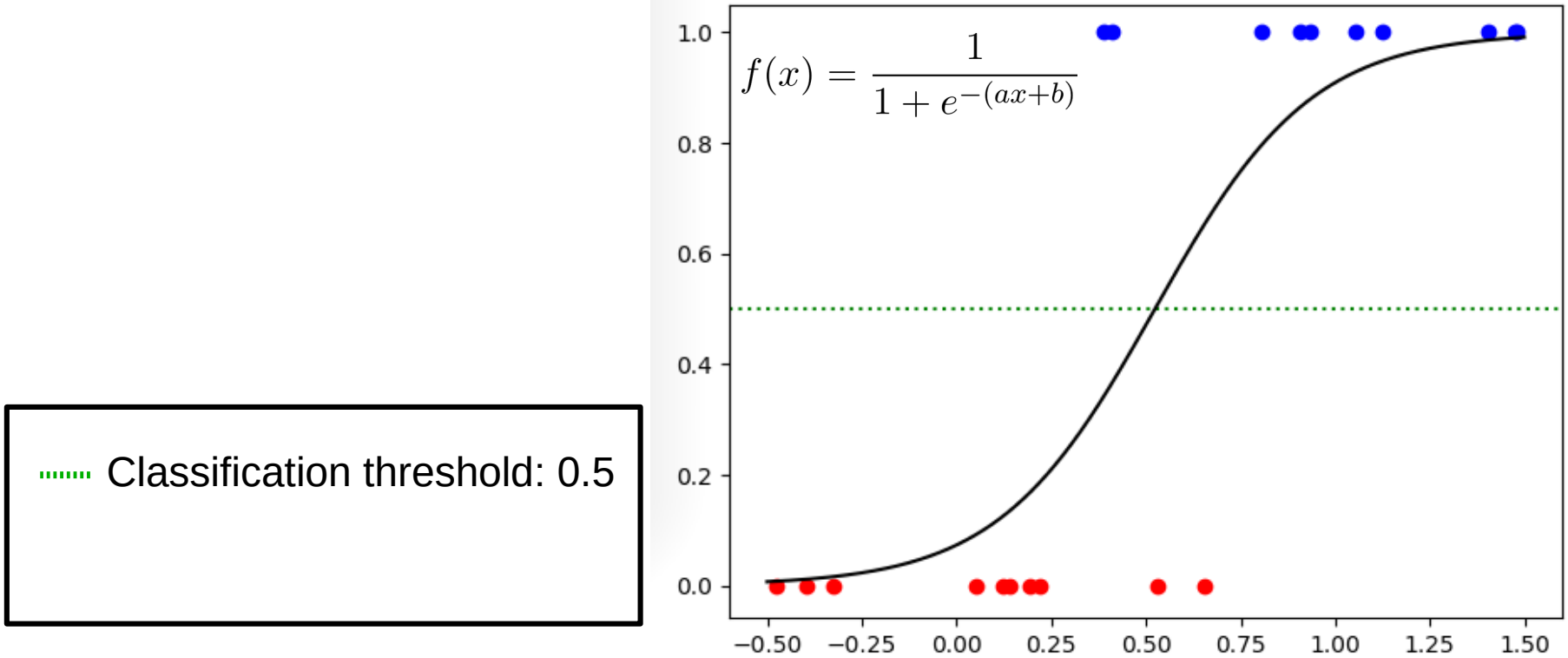
$$f(x) = ax + b \qquad f(x) = \frac{1}{1 + e^{-(ax+b)}}$$





Fitting

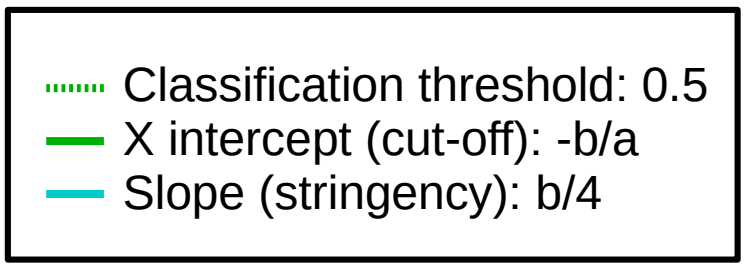
- We need to find optimal **a** and **b**

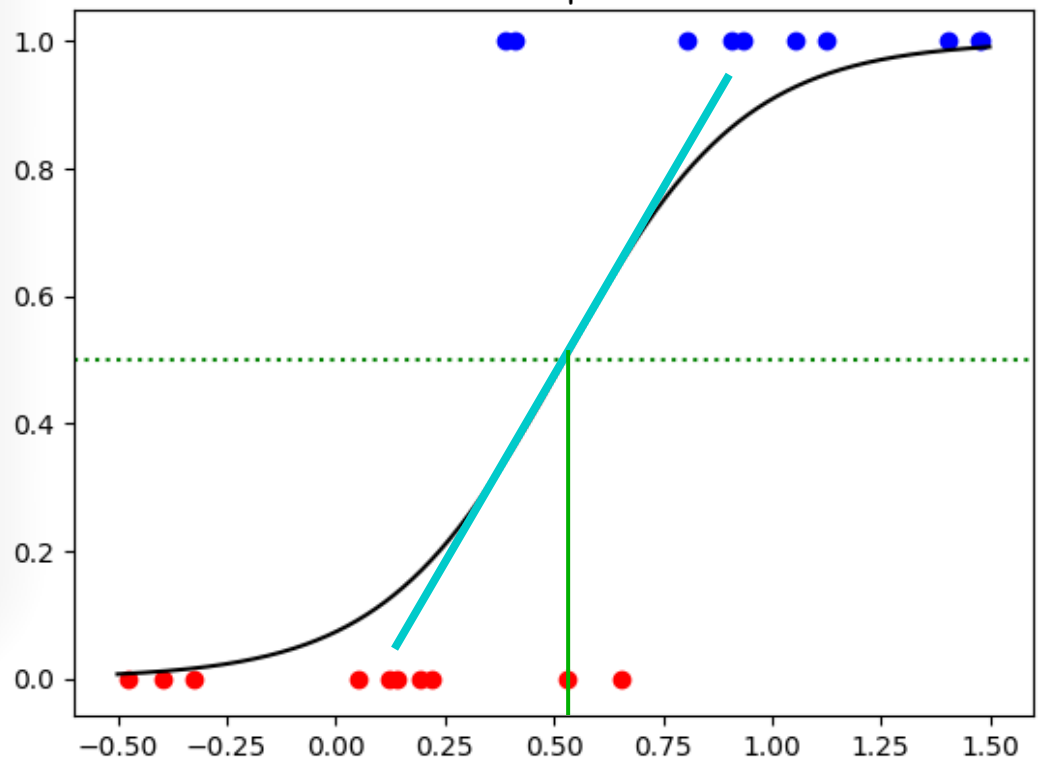


Fitting

- We need to find optimal **a** and **b**
- What is optimal ?

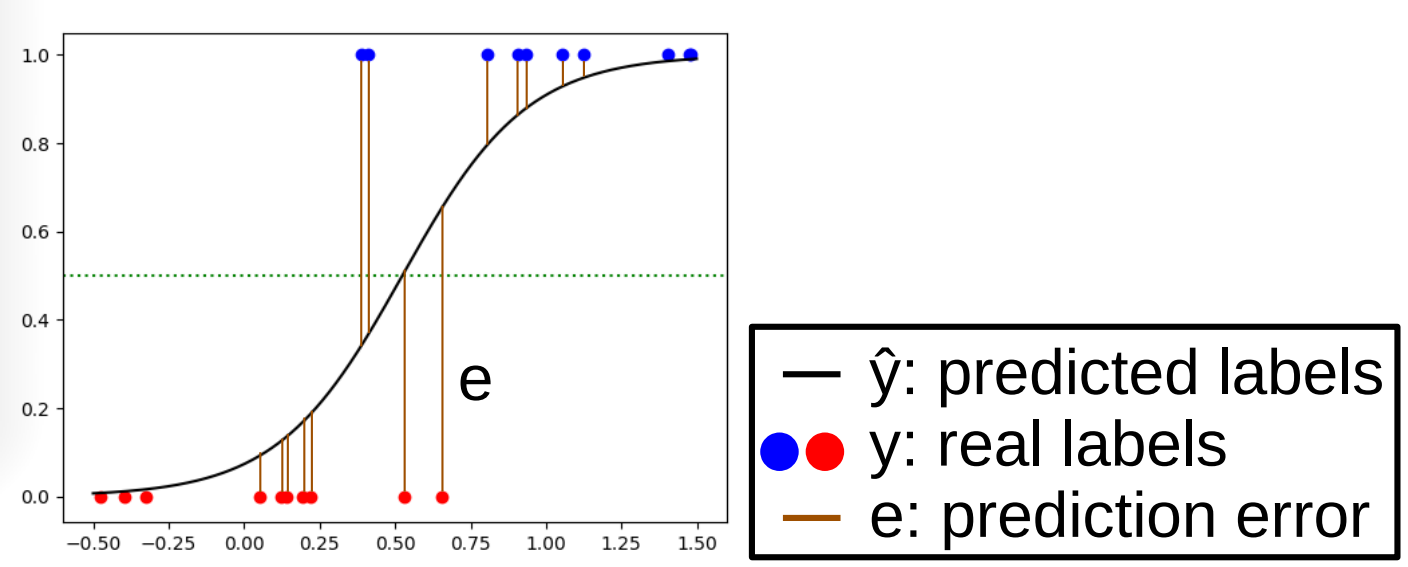
$$f(x) = \frac{1}{1 + e^{-(ax+b)}}$$





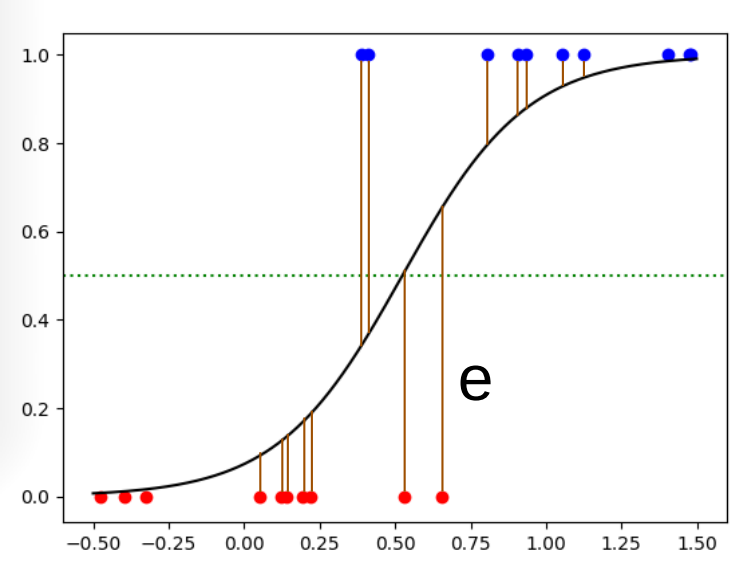
Cost function

- We need a cost function to assess the “goodness of fit”
- Classic cost function: Squared prediction error



Cost function

- We need a cost function to assess the “goodness of fit”
- Classic cost function: Squared prediction error

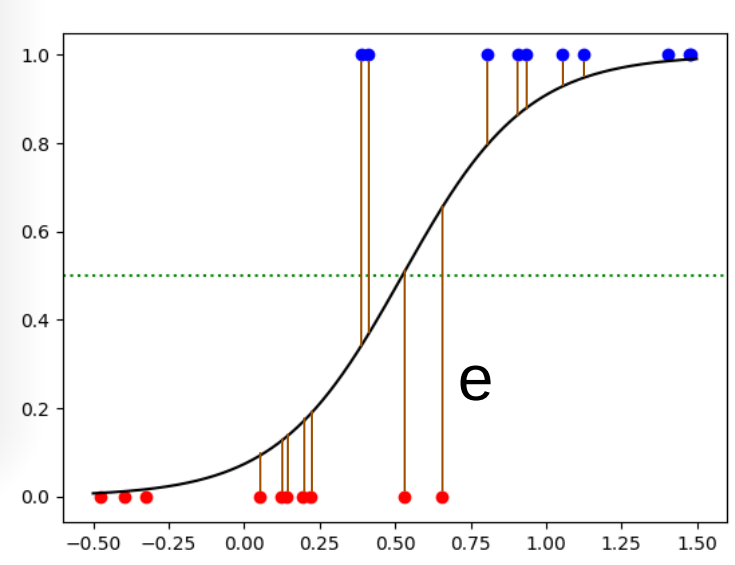


$$e = (\hat{y}_i - y_i)^2 = (y_i - f(x_i))^2$$

ŷ: predicted labels
y: real labels
e: prediction error

Cost function

- We need a cost function to assess the “goodness of fit”
- Classic cost function: Squared prediction error
- We need to find **a** and **b** which minimize cost function **E**. How ?



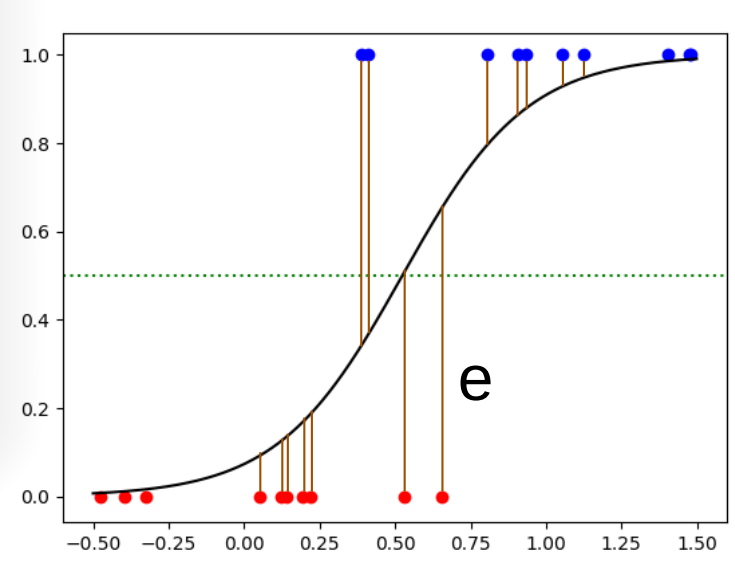
$$e = (\hat{y}_i - y_i)^2 = (y_i - f(x_i))^2$$

$$E(a,b) = \frac{1}{2}\sum_{i=1}^{N}(y_i - f(x_i,a,b))^2$$

— ŷ: predicted labels
●● y: real labels
— e: prediction error

Naive fitting ?

- Try every value of a and b until it works well



$$e = (\hat{y}_i - y_i)^2 = (y_i - f(x_i))^2$$

$$E(a,b) = \frac{1}{2}\sum_{i=1}^{N}(y_i - f(x_i,a,b))^2$$

— ŷ: predicted labels
●● y: real labels
— e: prediction error

Newton’s method

- Find the 0s of the derivative of the cost function
- These values are the a and b which minimize the cost function

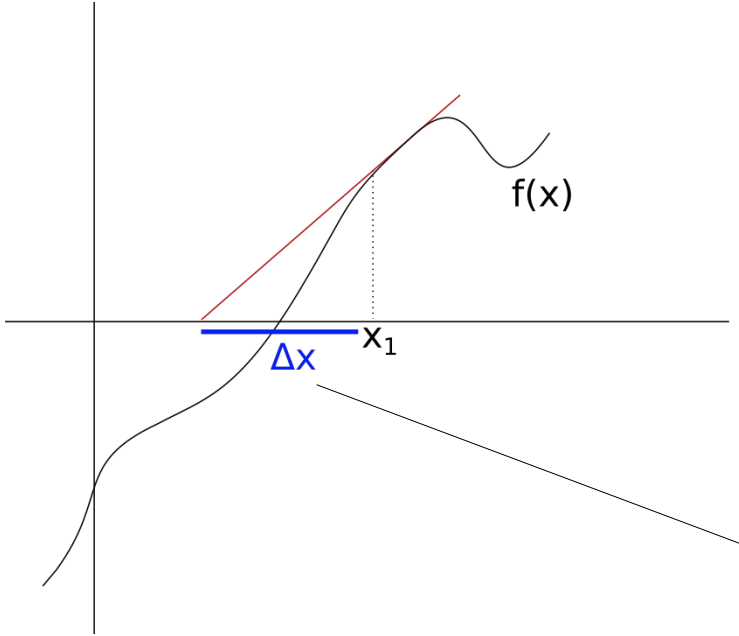
$$\frac{d}{da}E(a,b) = \frac{d}{da}\left(\frac{1}{2}\sum_{i=1}^{N}(y_i - f(x_j,a,b))^2\right) = F_o(a,b) = 0$$

$$\frac{d}{db}E(a,b) = \frac{d}{db}\left(\frac{1}{2}\sum_{i=1}^{N}(y_i - f(x_j,a,b))^2\right) = F_o(a,b) = 0$$

No analytical solution

Newton’s method

- Start with a random x and update it iteratively to find where f(x)=0
- We use the function gradient to find the next position of x

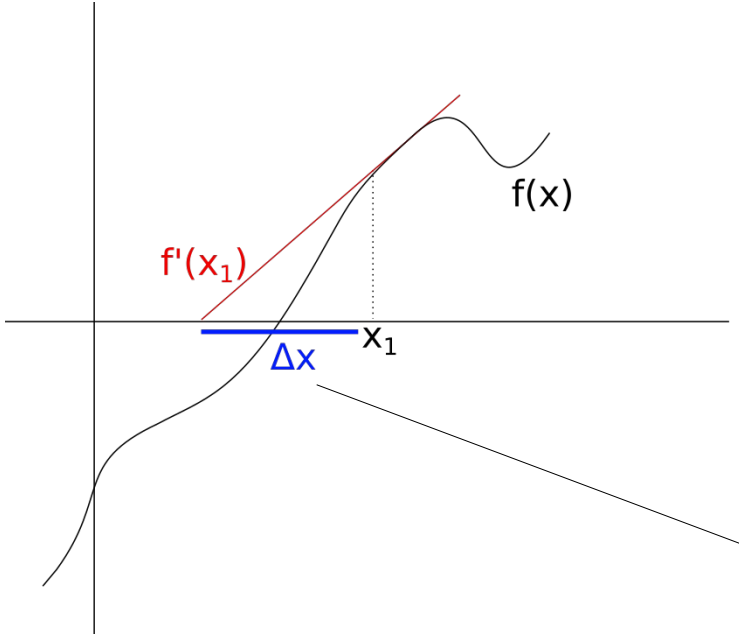


$$slope = \frac{\Delta y}{\Delta x}$$

$$\Delta x = \frac{\Delta y}{slope}$$

Newton's method

- Start with a random x and update it iteratively to find where f(x)=0
- We use the function gradient to find the next position of x



$$slope = \frac{\Delta y}{\Delta x}$$

$$\Delta x = \frac{\Delta y}{slope}$$

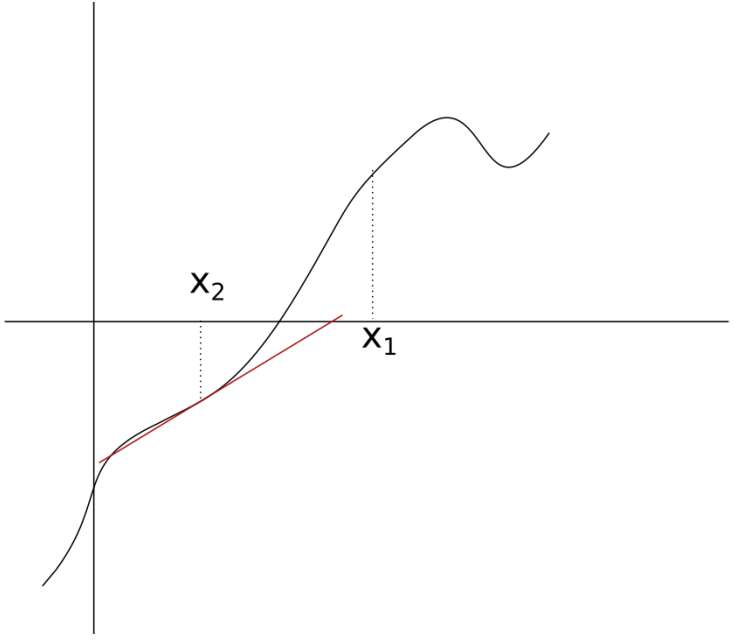
$$\Delta y = f(x_1)$$

$$slope = f'(x_1)$$

$$\Delta x = \frac{f(x_1)}{f'(x_1)}$$

Newton’s method

- Start with a random x and update it iteratively to find where f(x)=0
- We use the function gradient to find the next position of x

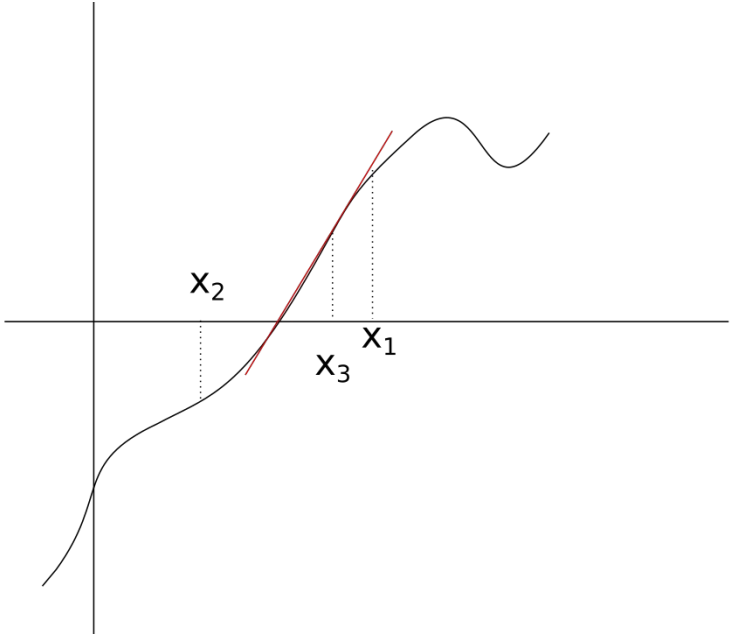


Repeat until f(x) is close enough to 0...

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Newton’s method

- Start with a random x and update it iteratively to find where f(x)=0
- We use the function gradient to find the next position of x

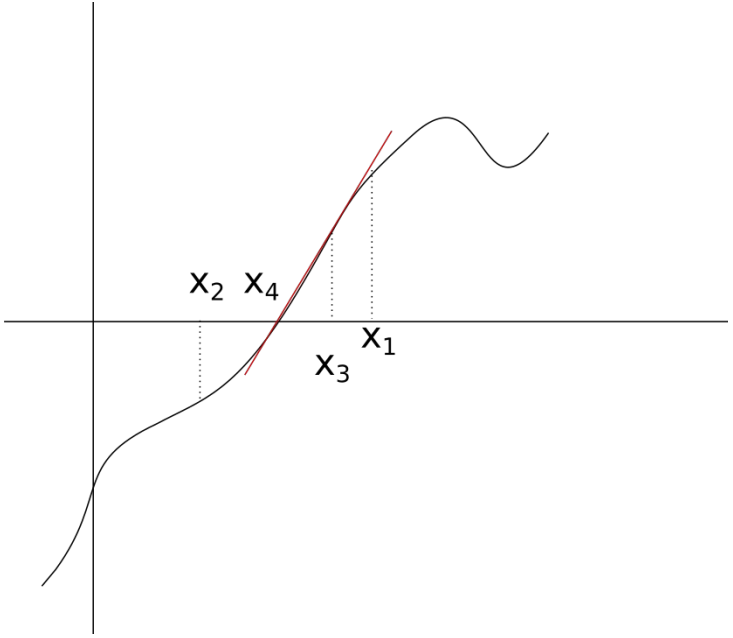


Repeat until f(x) is close enough to 0...

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Newton’s method

- Start with a random x and update it iteratively to find where f(x)=0
- We use the function gradient to find the next position of x

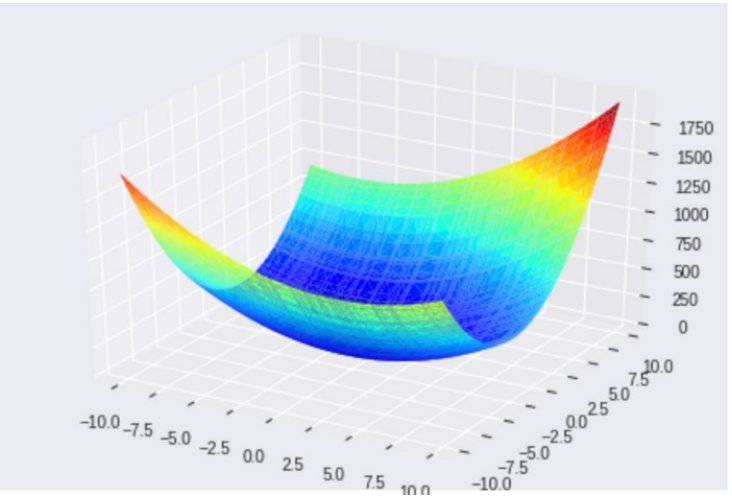


Repeat until f(x) is close enough to 0...

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Newton’s method

- Start with a random x and update it iteratively to find where f(x)=0
- We use the function gradient to find the next position of x



Repeat until f(x) is close enough to 0...

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

$$- \frac{E'(a_t, b_t)}{E''(a_t, b_t)}$$

Note: can be extended to 2D cases (a and b at the same time)

Fitting of a logistic regression

- Define a cost function E(a, b) (e.g. squared error)
- Find a and b minimizing E(a, b) using E’(a, b)=0.
- Use Newton’s method to approximate optimal a and b

ESL: p.120

Increasing dimensions

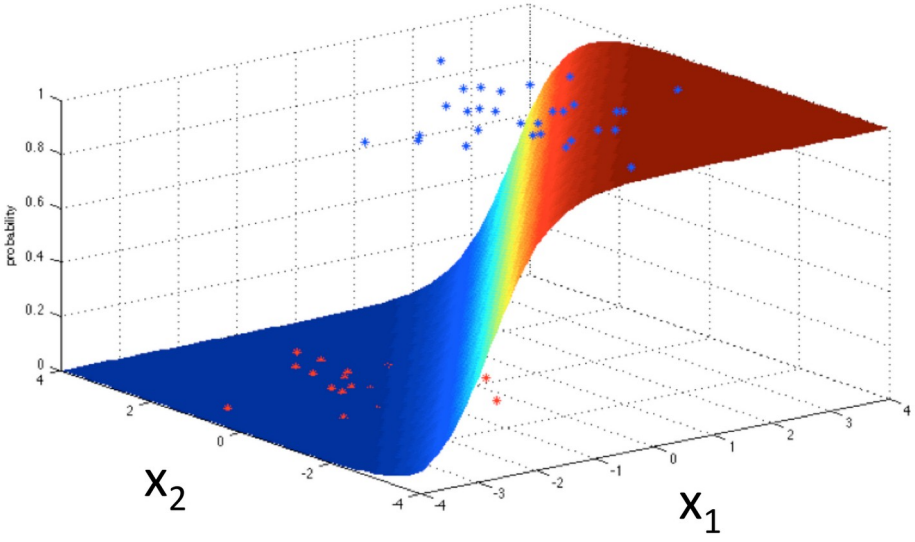
- In practice, we have more than 1 feature
- Matrix notation (and operations) are useful to generalize.

$$\theta_0 + \theta_1 * x_1 + \theta_2 * x_2 =$$

$$\begin{bmatrix}\theta_0 & \theta_1 & \theta_2\end{bmatrix} * \begin{bmatrix}1\\ x_1\\ x_2\end{bmatrix} =$$

$$\theta^t X$$

$$f(X) = \frac{1}{1 + e^{(-\theta^t X)}}$$



Increasing dimensions

- In practice, we have more than 1 feature
- Matrix notation (and operations) are useful to generalize.
- Computers are very fast at matrix multiplication !

```
import numpy as np

theta = [13, 0.4, 1, 3] # Weights (first is the bias)
x       = [1, 0.2, 0.1, 0.4] # Features of 1 sample

y = theta[0] + sum([x[i] * theta[i] for i in range(1, len(x))])  # Usual method

y = np.array(theta).T @ np.array(x)  # Matrix version (@ means dot product)
```

Increasing dimensions

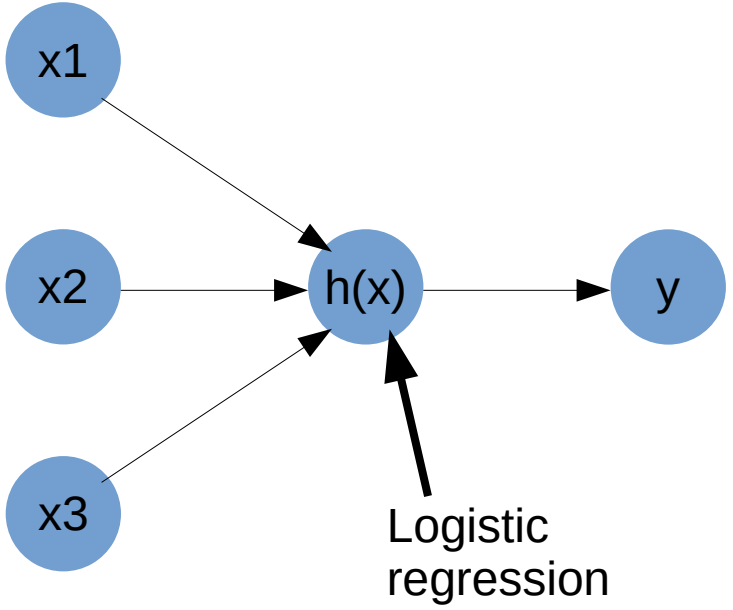
- In practice, we have more than 1 feature
- Matrix notation (and operations) are useful to generalize.
- Computers are very fast at matrix multiplication !

We can compute y for all samples
In a single matrix operation !

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix}$$

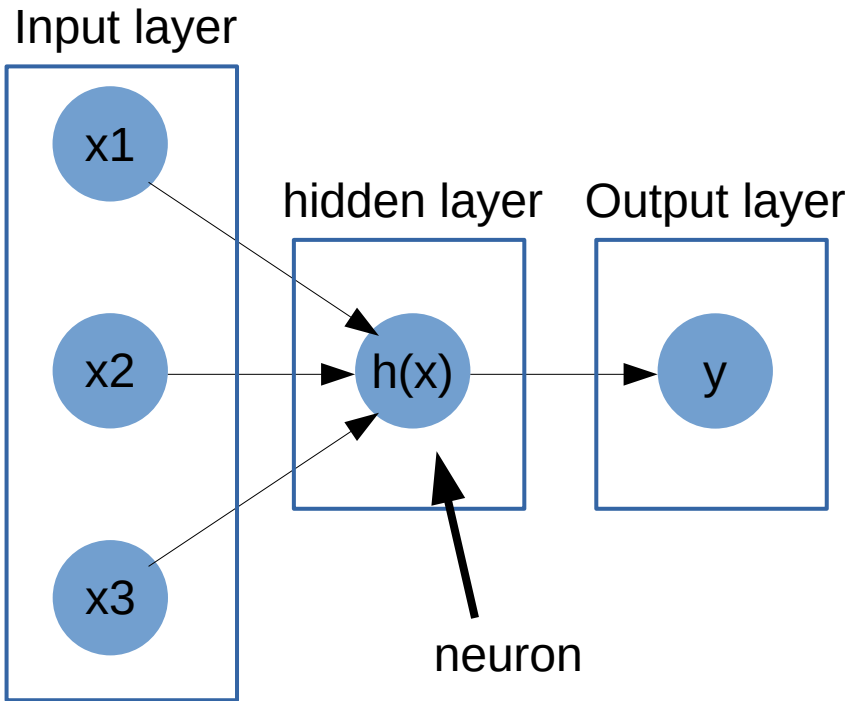
Stacking logistic regressions

- A single logistic regression can only separate data linearly
- Combine them into a neural network for more complex tasks



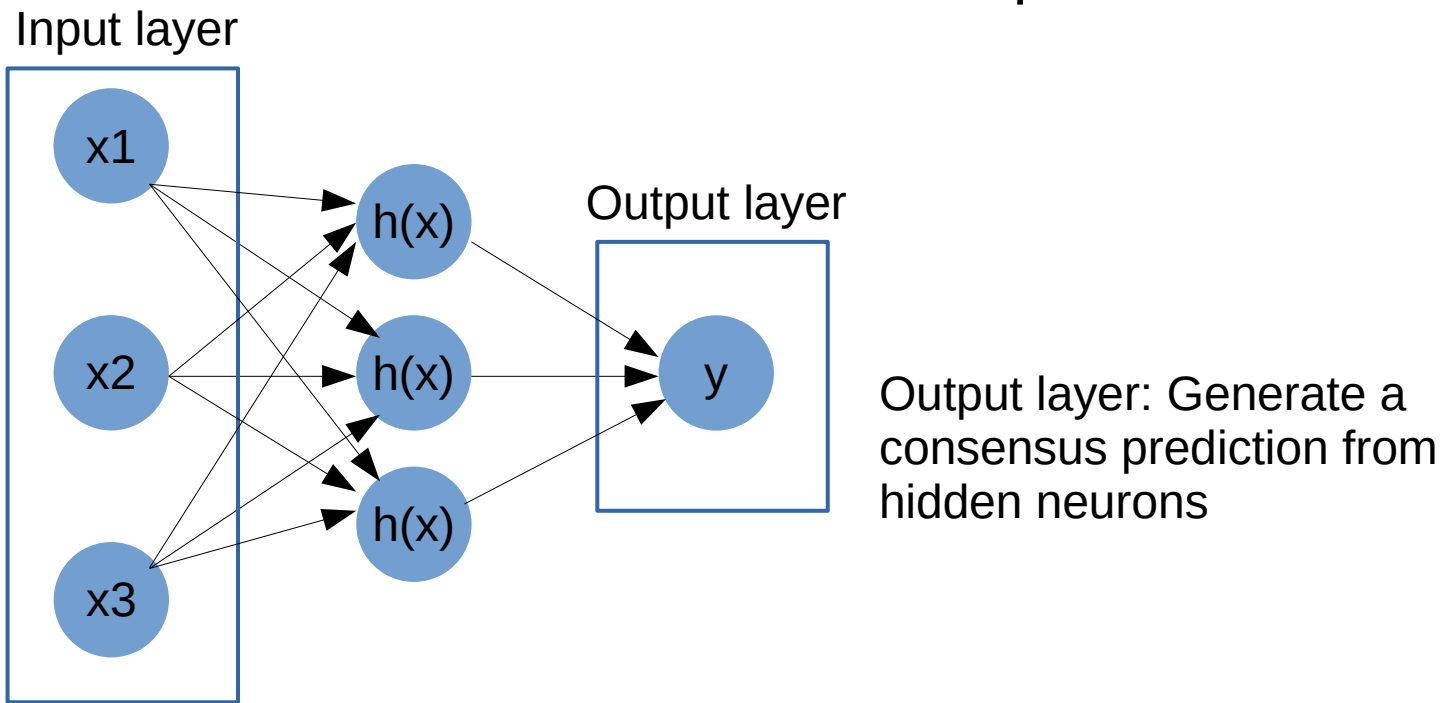
Stacking logistic regressions

- A single logistic regression can only separate data linearly
- Combine them into a neural network for more complex tasks



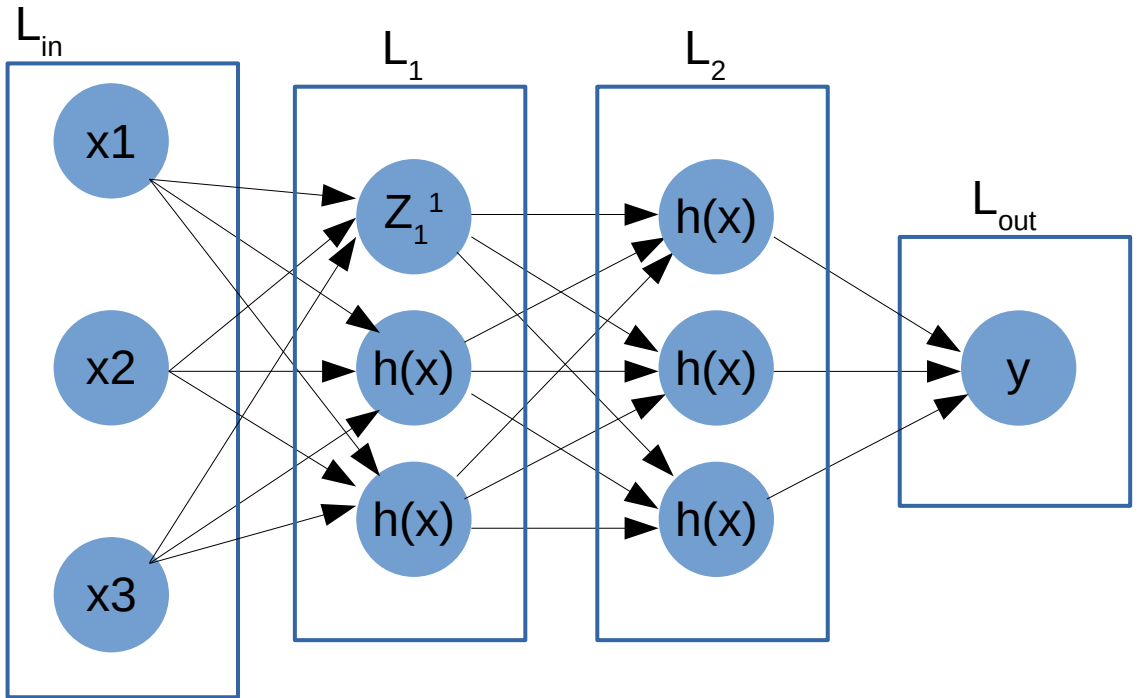
Stacking logistic regressions

- A single logistic regression can only separate data linearly
- Combine them into a neural network for more complex tasks



Neural networks: principle

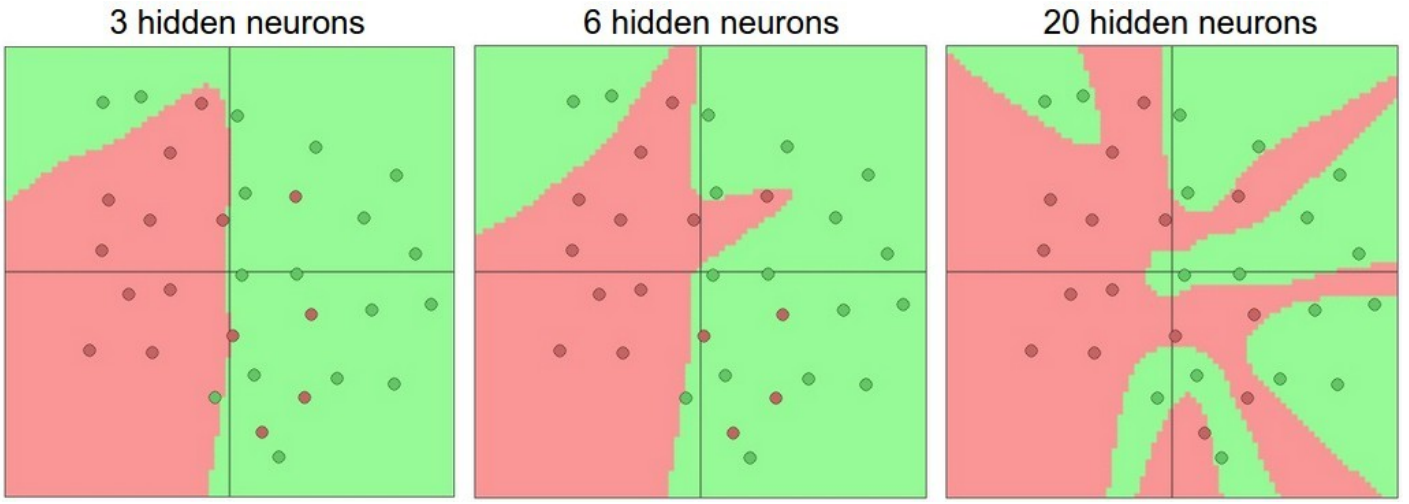
- One or multiple hidden layers, each with multiple neurons
- Input of each layer is the output of the previous one



ESL: p.392

Neural networks

- Fitting more difficult, but similar : Use derivatives to minimize E(θ)
- Also needs to iteratively approximate weights
- Gradient descent instead of Newton's method (E'' hard to compute)



Neural networks in genomics

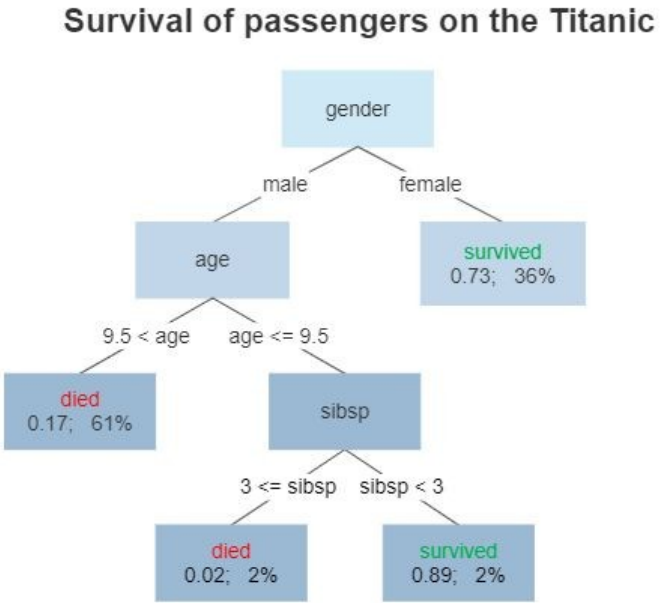
- DNA sequences can be used as input (one-hot encoding)
- Used to detect SNPs (DeepVariant)
- Improve resolution of Hi-C data (SRHiC)
- Predict gene expression from histone marks (DeepChrome)

Drawbacks:
- Requires very large training datasets
- “Black box”: very hard to understand what features are used by the model

	A	A	C	C	T	A	G	A	G
A	1	1	0	0	0	1	0	1	0
C	0	0	1	1	0	0	0	0	0
T	0	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	1	0	1

Decision trees

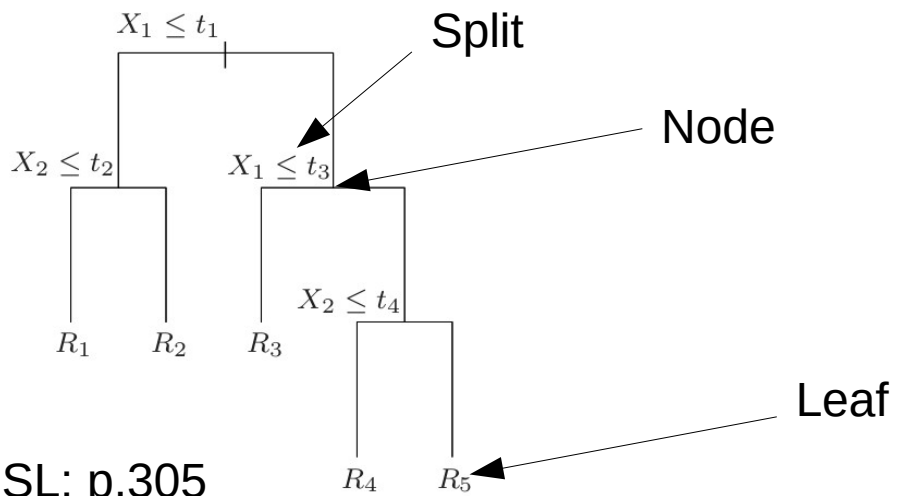
- Easily interpretable: Basically a series of if/else !
- Each node in the tree is a split rule
- Leaves are the output
- Works for regression and classification



ESL: p.305

Decision trees: General concept

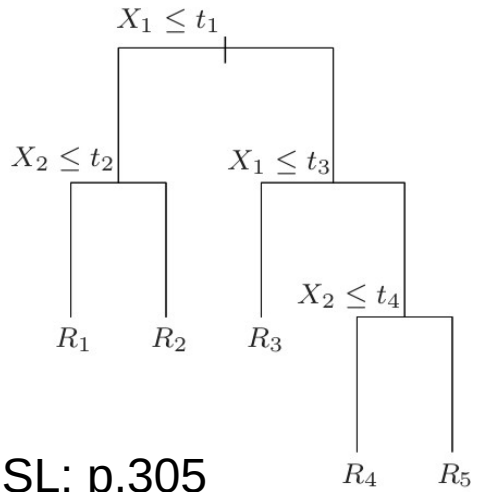
- We want a tree separating our dataset into regions R_m
- The separations (splits) are based on features (X)
- We need to find the best splits to predict y at the leaves. How ?

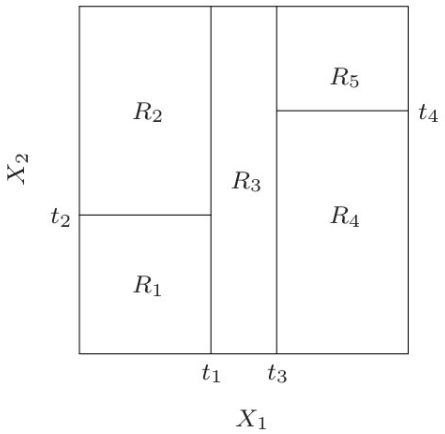


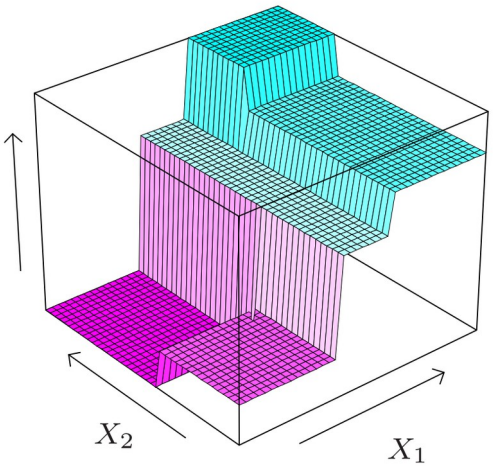
ESL: p.305

Decision trees: General concept

- We want a tree separating our dataset into regions R_m
- The separations (splits) are based on features (X)
- We need to find the best splits to predict y at the leaves. How ?



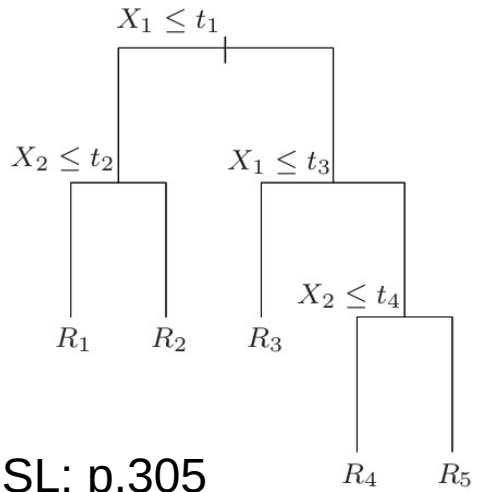


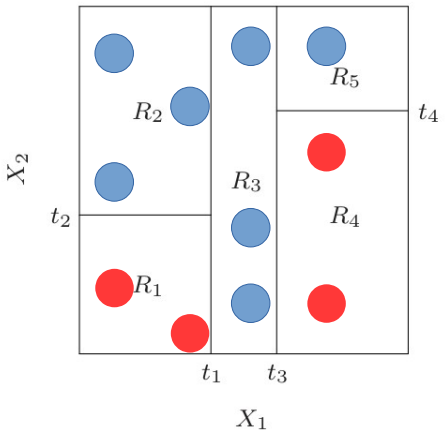


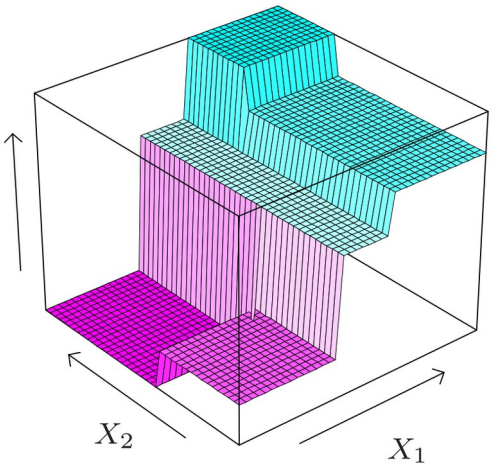
ESL: p.305

Decision trees: General concept

- We want a tree separating our dataset into regions R_m
- The separations (splits) are based on features (X)
- We need to find the best splits to predict y at the leaves. How ?







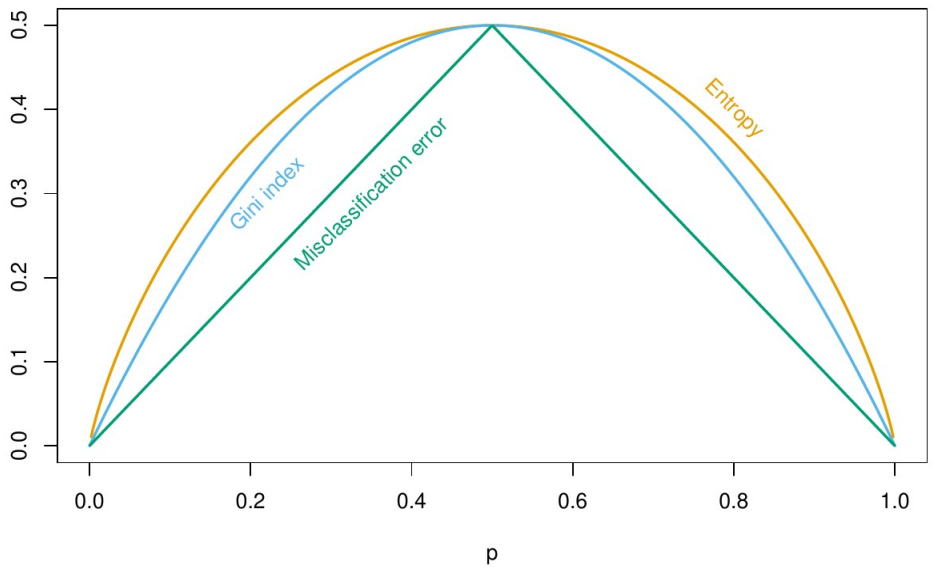
ESL: p.305

Fitting trees

- Recursive partitioning, general idea:
  - We use a metric to measure information content (impurity)

Entropy: Amount of "uncertainty" in the feature

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$



Fitting trees

- Recursive partitioning, general idea:
  - We use a metric to measure information content (impurity)
  - We want to find splits maximizing information gain
  - Recurse until node too small, or all samples in the group have the same y value

Entropy: Amount of "uncertainty" in the feature

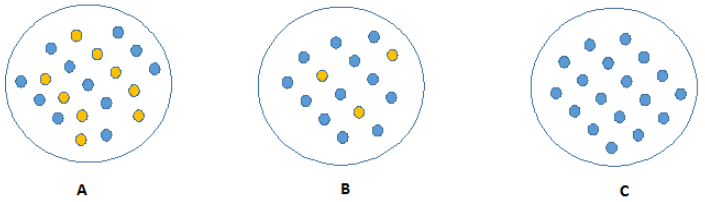
$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

Information gain: Loss of entropy after a split

$$IG(T, a) = \mathrm{H}(T) - \mathrm{H}(T|a)$$

Fitting trees: rationale

- We need a metric to compute “impurity” or discriminatory power



Entropy: Amount of “uncertainty” in the feature

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

$$H(A) = -\left(\frac{11}{20} log(\frac{11}{20}) + \frac{9}{20} log(\frac{9}{20})\right) = 0.69$$

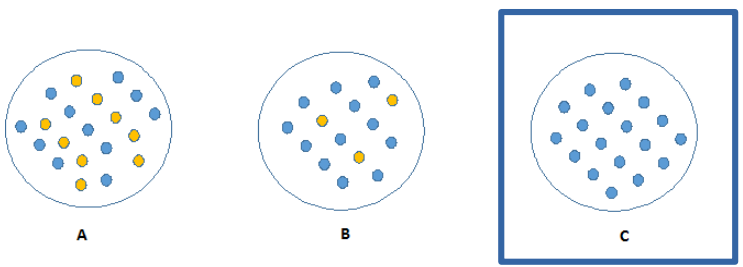
$$H(B) = -\left(\frac{12}{15} log(\frac{12}{15}) + \frac{3}{15} log(\frac{3}{15})\right) = 0.5$$

$$H(C) = -(1 log(1)) = 0$$

Fitting trees: rationale

- We need a metric to compute “impurity” or discriminatory power

“Pure” node → not informative



$$H(A) = -\left(\frac{11}{20}log(\frac{11}{20}) + \frac{9}{20}log(\frac{9}{20})\right) = 0.69$$

$$H(B) = -\left(\frac{12}{15}log(\frac{12}{15}) + \frac{3}{15}log(\frac{3}{15})\right) = 0.5$$

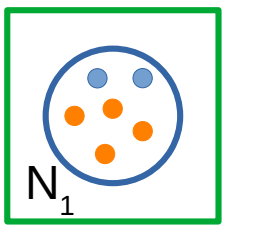
$$H(C) = -(1log(1)) = 0$$

Entropy: Amount of “uncertainty” in the feature

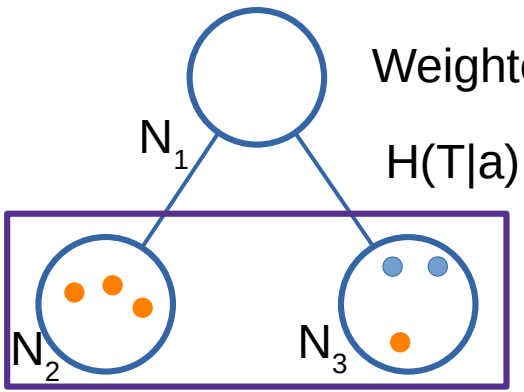
$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

Fitting trees: rationale

- We need a metric to compute “impurity” or discriminatory power
- Find split a minimizing resulting impurity



$H(T) = H(N_1)$



Weighted average of children

$H(T|a) = 3/6 * H(N_2) + 3/6 * H(N_3)$

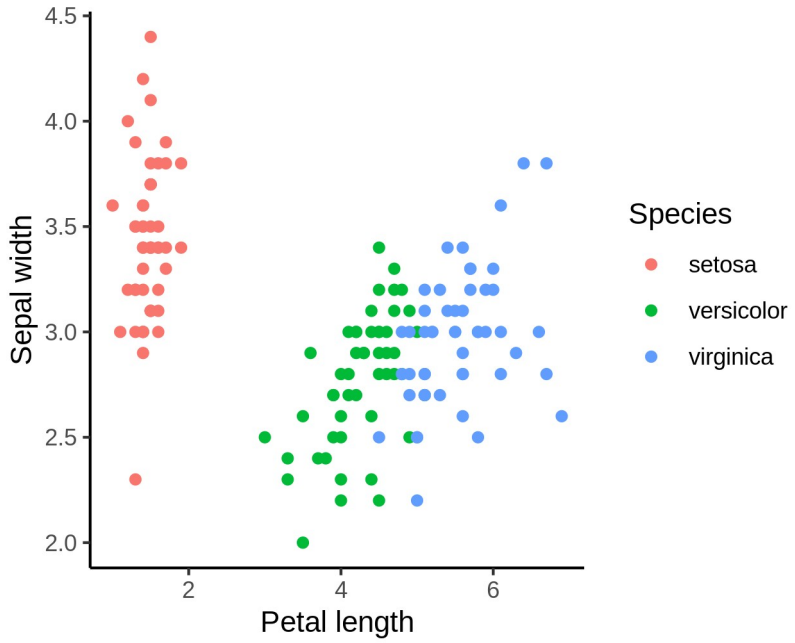
Information gain: Loss of entropy after a split

$$IG(T,a) = \mathrm{H}(T) - \mathrm{H}(T|a)$$

Fitting trees: Example

- We want to predict flower species from petal and sepal shape

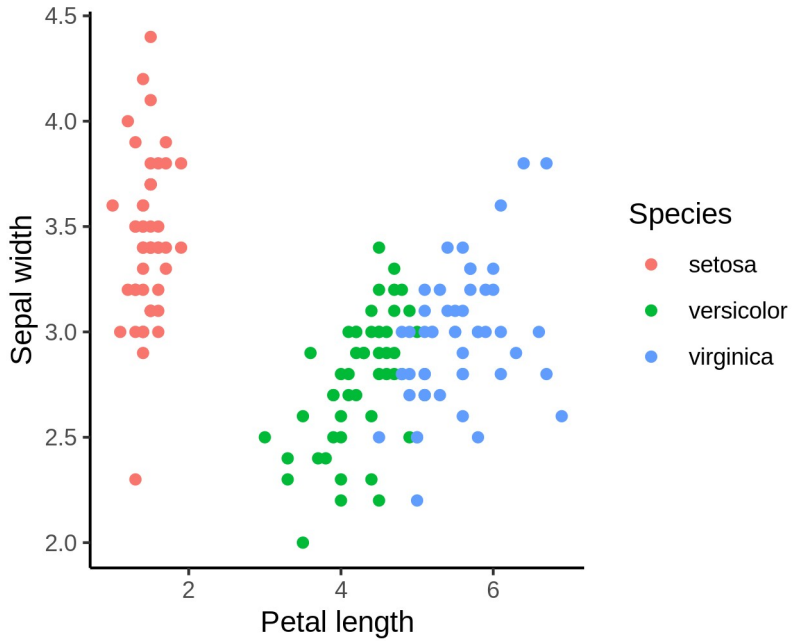
1. Compute entropy for each value **s** of each feature **j**



Fitting trees: Example

- We want to predict flower species from petal and sepal shape

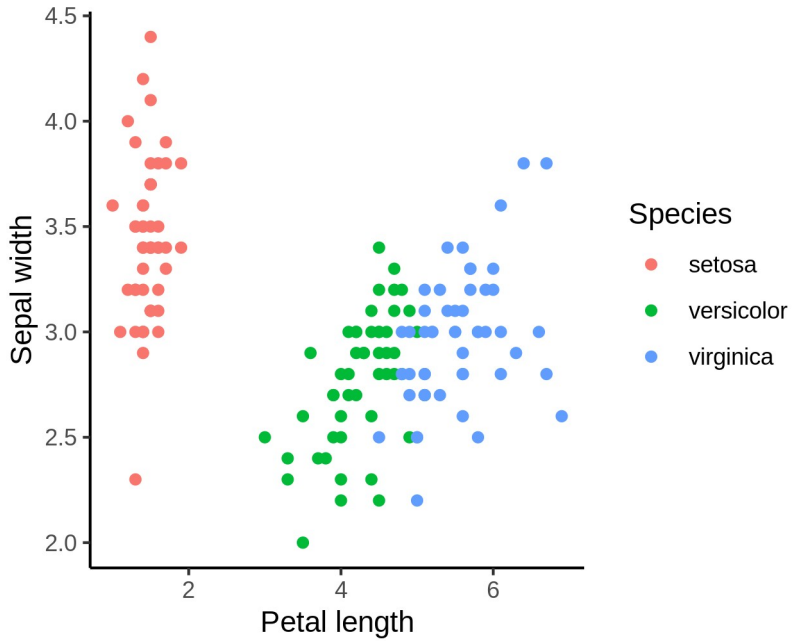
1. Compute entropy for each value **s** of each feature **j**
2. Select pair j, s with maximizing information gain



Fitting trees: Example

- We want to predict flower species from petal and sepal shape

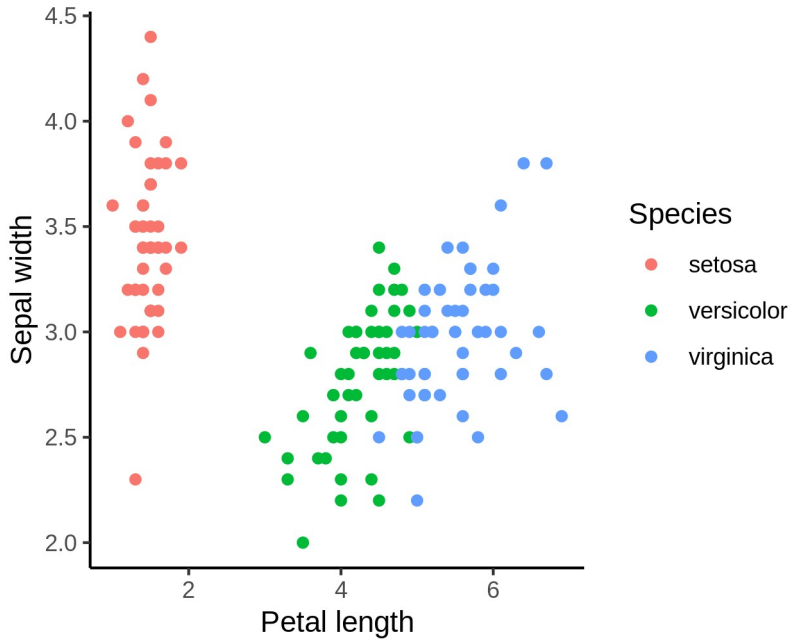
1. Compute entropy for each value **s** of each feature **j**
2. Select pair j, s with maximizing information gain
3. Compute information gain for every value s of j



Fitting trees: Example

- We want to predict flower species from petal and sepal shape

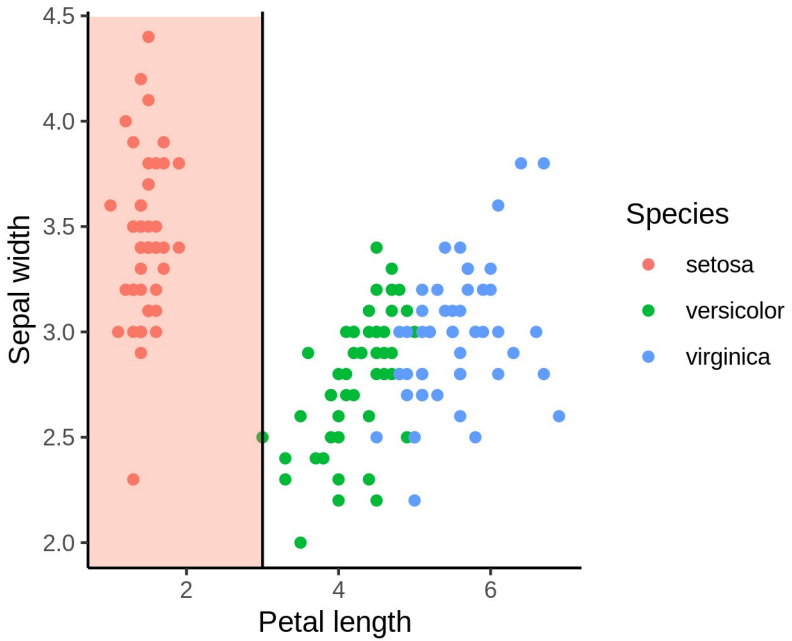
1. Compute entropy for each value **s** of each feature **j**
2. Select pair j, s with maximizing information gain
3. Compute information gain for every value s of j



Fitting trees: Example

- We want to predict flower species from petal and sepal shape

1. Compute entropy for each value **s** of each feature **j**
2. Select pair j, s with maximizing information gain
3. Compute information gain for every value s of j
4. Split dataset X into two regions based on $X_j > s$

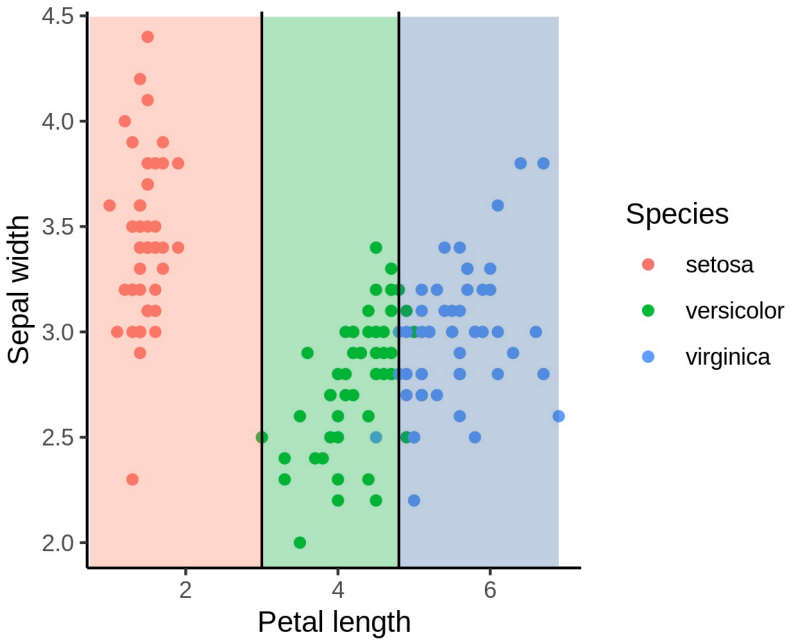


Fitting trees: Example

- We want to predict flower species from petal and sepal shape

1. Compute entropy for each value **s** of each feature **j**
2. Select pair j, s with maximizing information gain
3. Compute information gain for every value s of j
4. Split dataset X into two regions based on $X_j > s$
5. Recurse until reaching base case. E.g:
   - Maximal number of nodes
   - Node smaller than n
   - All samples in node have the same species

   ...



Fitting trees: Example

```
best_s, best_j = 0, None
max_ig = 0
N = len(iris)
for j in [“Petal length”, “Sepal width”]:
     base_entro = H(Species)
     for s in unique(j):
       R1 = Species[j >= s]
       R2 = Species[j < s]
       Ha = len(R1) / N * H(R1) + len(R2) / N * H(R2)
       ig = base_entro - new_entro
       if ig > max_ig:
     max_ig = ig
         best_s, best_j = s, j
       base_entro = new_entro
print(best_s)
```

More resources

- If you want to learn more about the topic:
  - **An introduction to statistical learning, Hastie et al.**: Introduction to ML for students of non-mathematicians. Available for free at: https://faculty.marshall.usc.edu/gareth-james/ISL/
  - **Elements of statistical learning, Hastie et al.**: Explains the maths behind machine learning, starting from scratch. Available for free at: http://www.web.stanford.edu/~hastie/ElemStatLearn/
  - Blog post explaining gradient descent in detail: https://mccormickml.com/2014/03/04/gradient-descent-derivation/
  - Blog post detailing the derivation the normal equation for OLS fitting in linear regression: https://dustinstansbury.github.io/theclevermachine/derivation-normal-equations