Simple Supervised Learning Algorithms

Martha Lewis

Department of Engineering Mathematics
University of Bristol
martha.lewis@bristol.ac.uk

Outline

- k-Nearest Neighbour - classification and regression
- Linear Regression
- Logistic Regression
- Naive Bayes Classifier

k-Nearest Neighbour

- Classification problem: we want to learn a function $f : \Omega \to \mathcal{O}$ where $\mathcal{O} = \{c_1, ...c_n\}$
- We are given a dataset consisting of pairs of input data $\{\vec{x}_1, ..., \vec{x}_m\}$ and their classes $\{y_1, ..., y_m\}$, where $y_i \in \{c_1, ...c_n\}$.

kNN Classification Algorithm

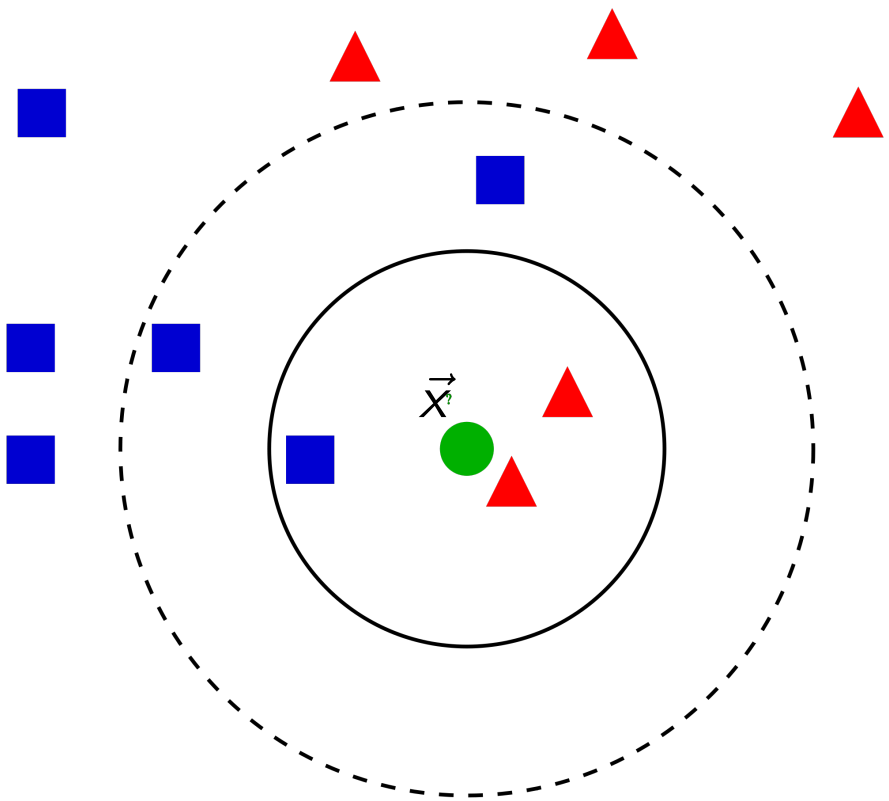
Training: just store the dataset

Prediction: Given a new example $\vec{x}$:

Find the k data points $(\vec{x}_i, y_i)$ where $||\vec{x} - \vec{x}_i||$ is smallest
Return the majority class from the y_i

k-Nearest Neighbour

- Given an unclassified pattern $\vec{x}$ find the closest k labelled patterns in the training data.
- Then classify as the majority class.
- You can also weight by distance.



k-NN Regression

- In a regression problem we are trying to find a function $f : \Omega \to \mathcal{O}$ where $\mathcal{O} = \mathbb{R}$
- Our dataset is a set of pairs of vectors and the value they are mapped to: $(\vec{x}_i, y_i)$

kNN Regression Algorithm

Training: just store the dataset

Prediction: Given a new example $\vec{x}$:

Find the k data points $(\vec{x}_i, y_i)$ where $||\vec{x} - \vec{x}_i||$ is smallest
Return a weighted average of the y_i

k-NN Regression

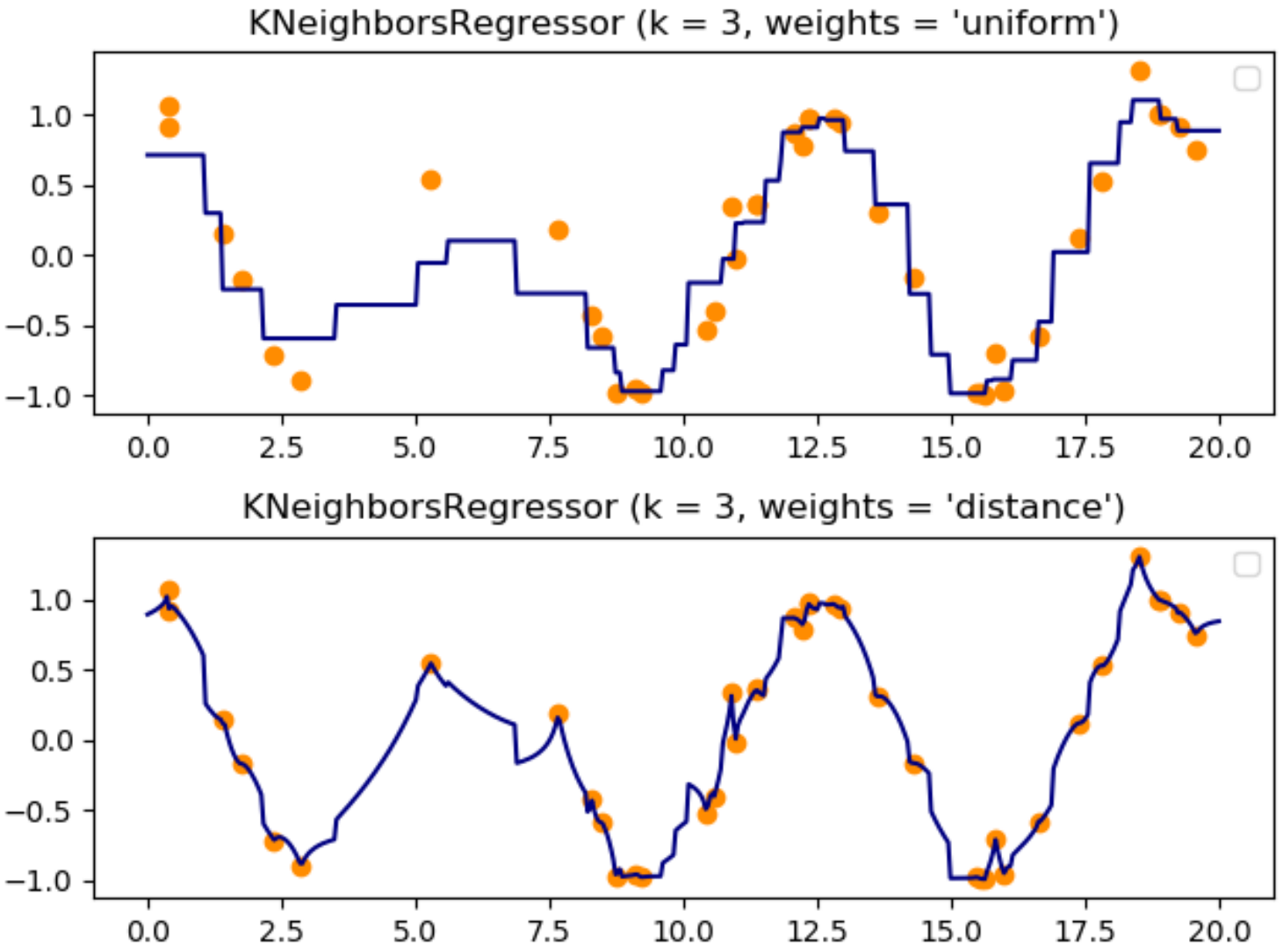
- For $\vec{x}$ identify the closest k input vectors in the training set.
- Suppose these are $\vec{x_1}, \ldots, \vec{x_k}$ with associated output values $y_1, \ldots, y_k$.
- Uniform weights:

$$\hat{f}(\vec{x}) = \frac{\sum_{i=1}^{k} y_i}{k}$$

- Distance weights:

$$\hat{f}(\vec{x}) = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i} \text{ where } w_i = \frac{1}{||\vec{x} - \vec{x_i}||}$$

k-NN Regression



Summary: k-Nearest Neighours

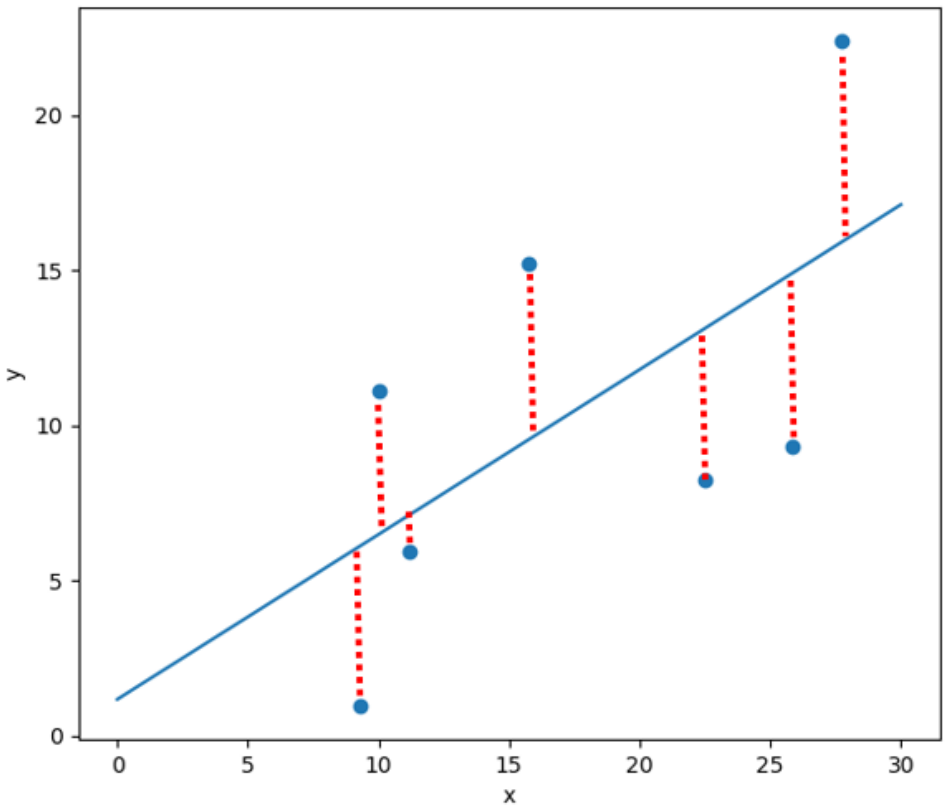
- Simple algorithm
- Looks at the nearest neighbours in the feature space
- Can be used for classification or regression

Worksheet

- Use kNN to classify data in a binary dataset
- Write your own version of kNN and compare with implementation in scikit-learn
- Use cross-validation on the training set to select the best value of k

Linear Regression

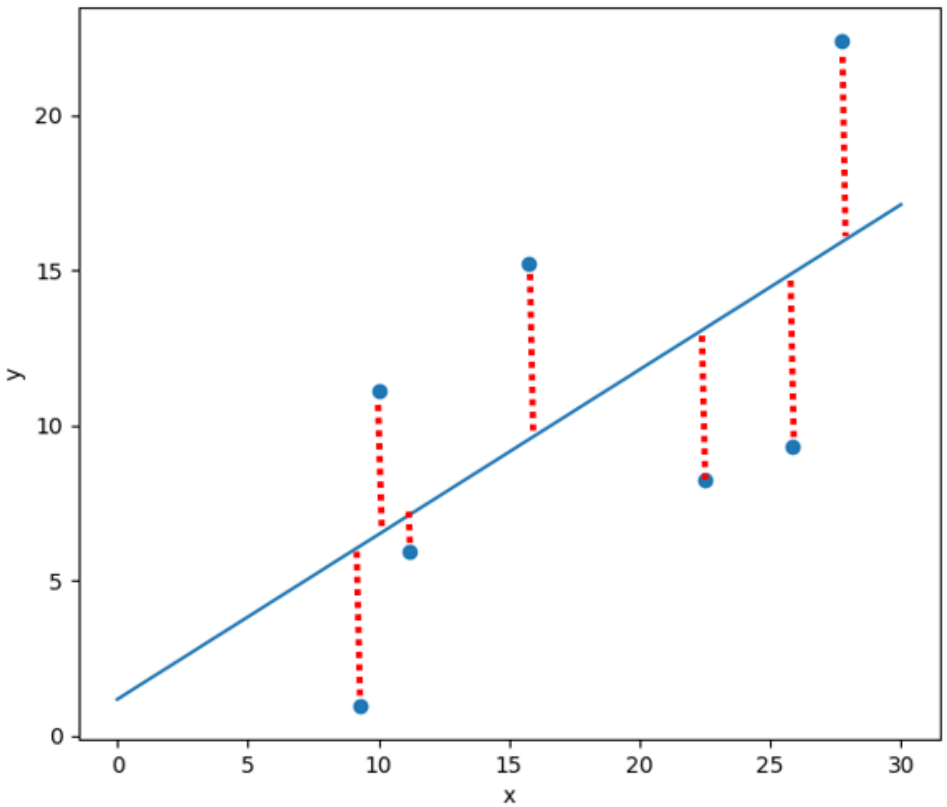
- Regression problem: we want to find a function $f : \Omega \to \mathbb{R}$, and our dataset is of the form $\{(\vec{x}_1, y_1), ...(\vec{x}_k, y_k)\}$
- We assume that f has a particular form: $f(\vec{x}) = \sum_{i=1}^{n} a_i x_i + b$



Linear Regression

- Assume that $f(\vec{x}) = \sum_{i=1}^{n} a_i x_i + b$
- Select $a_1, \ldots, a_n, b$ to minimize the mean squared error over the training set; for N training patterns,

$$E = \frac{\sum_{(\vec{x},y)} (y - f(\vec{x}))^2}{N}$$



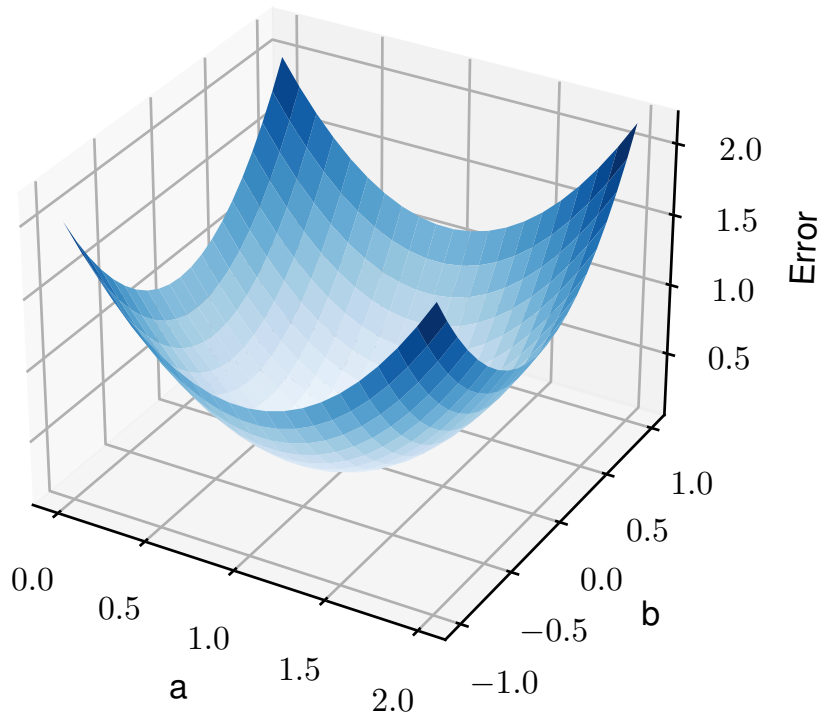
Simple Linear Regression

Let's think about the case where $\vec{x}$ is 1-dimensional, i.e. $x \in \mathbb{R}$
Then $f(x) = ax + b$ and $E = \frac{1}{N} \sum_{(x,y)} (y - ax - b)^2$.
Let $\overline{x} = \frac{1}{N} \sum_{(x,y)} x$, and similarly for other expressions such as y, x^2, ...
Expanding the expression for E, we get the following:

$$\begin{aligned} E &= \frac{1}{N} \sum_{(x,y)} (y - ax - b)^2 \\ &= \frac{1}{N} \sum_{(x,y)} y^2 - 2xya - 2yb + 2xab + x^2a^2 + b^2 \\ &= \overline{y^2} - 2\overline{xy}a - 2\overline{y}b + 2\overline{x}ab + \overline{x^2}a^2 + b^2 \end{aligned}$$

Simple Linear Regression

$$E = \overline{y^2} - 2\overline{xy}a - 2\overline{y}b + 2\overline{x}ab + \overline{x^2}a^2 + b^2$$



To find optimal values of a and b we must solve $\frac{\partial E}{\partial a} = 0$ and $\frac{\partial E}{\partial b} = 0$.

Simple Linear Regression

$$E = \overline{y^2} - 2\overline{xy}a - 2\overline{y}b + 2\overline{x}ab + \overline{x^2}a^2 + b^2$$

$$\frac{\partial E}{\partial b} = 0 \implies -2\overline{y} + 2\overline{x}a + 2b = 0 \implies b = \overline{y} - \overline{x}a$$

$$\frac{\partial E}{\partial a} = 0 \implies -2\overline{xy} + 2\overline{x}b + 2\overline{x^2}a = 0 \implies a = \frac{\overline{xy} - \overline{x}b}{\overline{x^2}}$$

Substituting $b = \overline{y} - \overline{x}a$ we obtain

$$a = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}$$

Linear Regression

- Suppose we have the training set: $(x_{i,1}, \ldots, x_{i,n}, y_i)$ for $i = 1, \ldots, N$.
- Let $\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,n} & 1 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ x_{N,1} & x_{N,2} & \ldots & x_{N,n} & 1 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$
- Let $\mathbf{\Theta} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \\ b \end{pmatrix}$ be the model parameters.
- The sum of squared error is given by;

$$E = \sum_{i=1}^{N} (y_i - \sum_{j=1}^{n} a_j x_{i,j} - b)^2 = ||\mathbf{y} - \mathbf{X\Theta}||^2$$
$$= (\mathbf{y} - \mathbf{X\Theta})^T (\mathbf{y} - \mathbf{X\Theta})$$

Parameter Optimisation

$$
\begin{aligned}
E &= (\mathbf{y} - \mathbf{X\Theta})^T(\mathbf{y} - \mathbf{X\Theta}) \\
&= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X\Theta} - (\mathbf{X\Theta})^T y + (\mathbf{X\Theta})^T\mathbf{X\Theta} \\
&= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X\Theta} - \mathbf{\Theta}^T\mathbf{X}^T\mathbf{y} + \mathbf{\Theta}^T\mathbf{X}^T\mathbf{X\Theta}
\end{aligned}
$$

We again wish to minimise E with respect to $\mathbf{\Theta}$, so we compute $\frac{\partial E}{\partial \mathbf{\Theta}}$. Relevant matrix differentiation identities are:

$$
\frac{\partial \mathbf{A\Theta}}{\partial \mathbf{\Theta}} = \mathbf{A}^T, \quad \frac{\partial \mathbf{\Theta}^T\mathbf{A}^T}{\partial \mathbf{\Theta}} = \mathbf{A}, \quad \frac{\partial \mathbf{\Theta^T A\Theta}}{\partial \mathbf{\Theta}} = 2\mathbf{A}^T\mathbf{\Theta}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial \mathbf{\Theta}} = 0 &= -\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X\Theta} \\
&= 2\mathbf{X}^T\mathbf{X\Theta} - 2\mathbf{X}^T\mathbf{y} \\
\implies \hat{\mathbf{\Theta}} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}
\end{aligned}
$$

Parameter Optimisation

- The quantity $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is know as the pseudoinverse of $\mathbf{X}$, and there is a function to compute this in NumPy.
- Problem: if you have a large number of features, $(\mathbf{X}^T\mathbf{X})^{-1}$ can become difficult to compute.
- Solution: We can use gradient descent to compute the parameters (we will cover this in a few weeks)

Linear Regression: Example

House Prices

Suppose we have the training set: $(x_{i,1}, \ldots, x_{i,n}, y_i)$ for $i = 1, \ldots, N$

Crime	Industry	Rooms	Tax	Value
0.063	2.31	6.6	296	24.0
0.027	7.07	6.4	242	21.6
0.027	7.07	7.2	242	34.7
0.032	2.18	7.0	222	33.4
0.023	2.18	6.4	222	?

$$\mathbf{X} = \begin{pmatrix} 0.063 & 2.31 & 6.6 & 296 & 1 \\ 0.027 & 7.07 & 6.4 & 242 & 1 \\ 0.027 & 7.07 & 7.2 & 242 & 1 \\ 0.032 & 2.18 & 7.0 & 222 & 1 \end{pmatrix} \quad \mathbf{\Theta} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ b \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} 24.0 \\ 21.6 \\ 34.7 \\ 33.4 \end{pmatrix}$$

$$\mathbf{X\Theta} = \mathbf{y}$$

Linear Regression: Example

The value of $\boldsymbol{\Theta}$ that minimizes prediction error is given by:

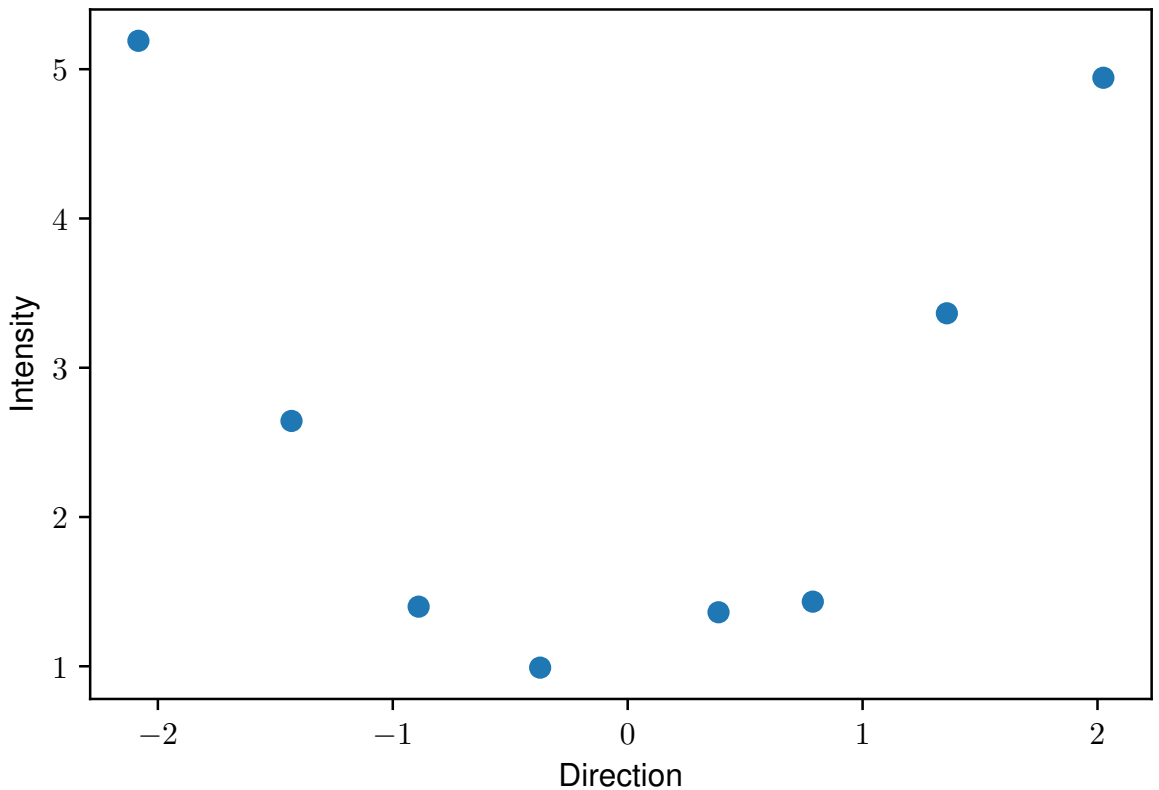
$$\hat{\boldsymbol{\Theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Using NumPy, we can compute this to be

$$\hat{\boldsymbol{\Theta}} = \begin{pmatrix} 4.037 \\ -0.237 \\ 16.375 \\ -0.039 \\ -0.720 \end{pmatrix}$$

Polynomial Regression

The term 'linear' in linear regression refers only to the coefficients. We can fit polynomial curves using exactly the same techniques, by adding extra features to the dataset.



Polynomial Regression

Suppose we have the training set: (x_i, y_i) for $i = 1, \ldots, N$.
We can incorporate higher-order terms by setting

$$\mathbf{X} = \begin{pmatrix} x_1^k & x_1^{k-1} & \ldots & x_1 & 1 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ x_N^k & x_N^{k-1} & \ldots & x_N & 1 \end{pmatrix}, \boldsymbol{\Theta} = \begin{pmatrix} a_k \\ \vdots \\ a_1 \\ b \end{pmatrix}, \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

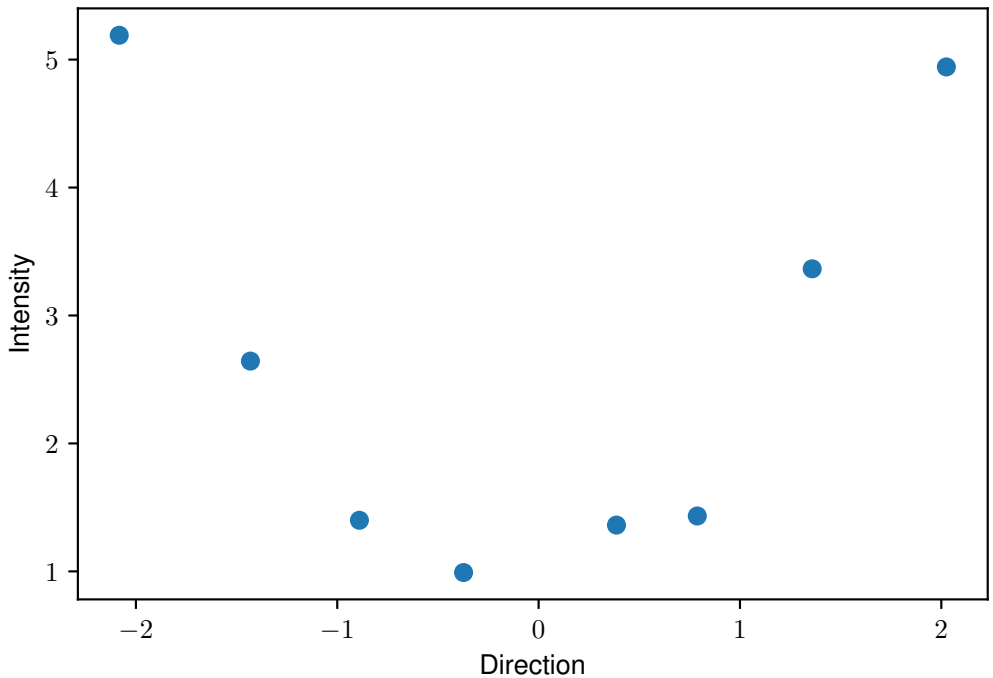
The model parameters $\boldsymbol{\Theta}$ are estimated in exactly the same way:

$$\hat{\boldsymbol{\Theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Polynomial Regression: Example

Suppose we have data as follows:

Direction	Intensity
-2.08	5.19
-1.43	2.64
-0.89	1.40
-0.37	0.99
0.39	1.36
0.79	1.43
1.35	3.36
2.03	4.94



Polynomial Regression: Example

We can try a quadratic fit to the data. We expand our feature set with a quadratic feature.

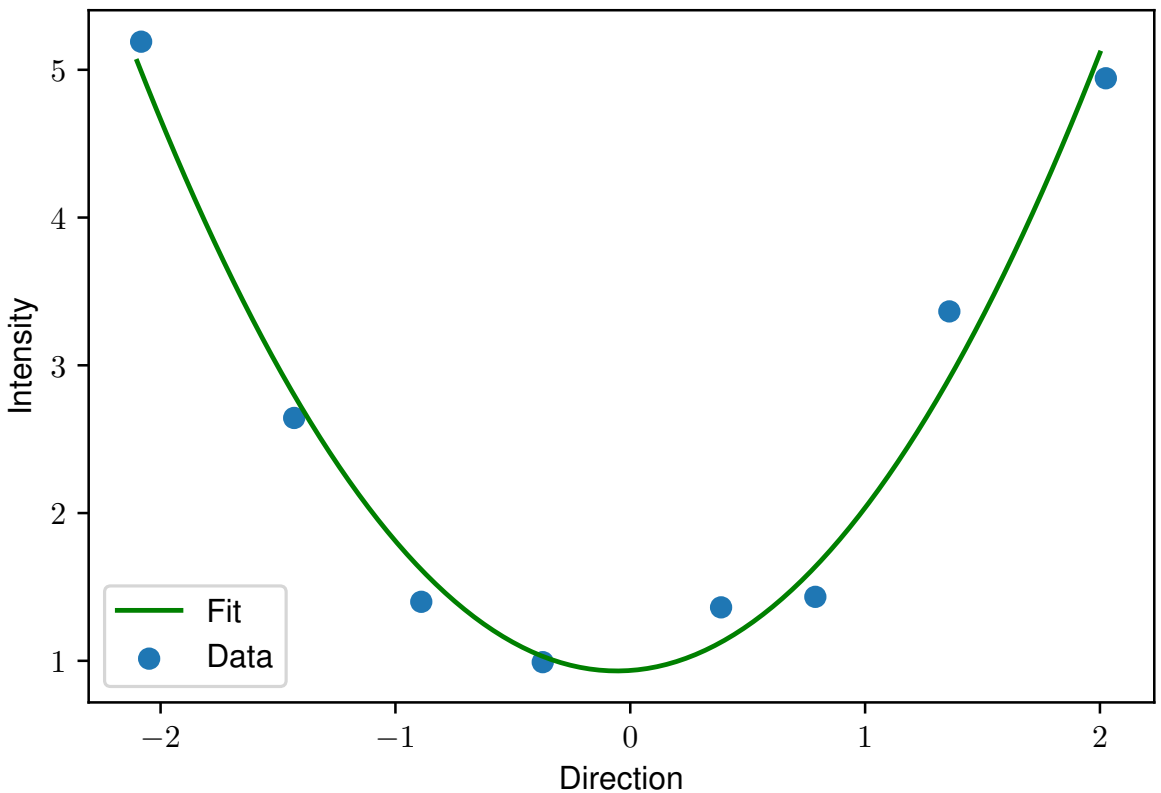
$$\mathbf{X} = \begin{pmatrix} 4.33 & -2.08 & 1 \\ 2.04 & -1.43 & 1 \\ 0.79 & -0.89 & 1 \\ 0.13 & -0.37 & 1 \\ 0.14 & 0.39 & 1 \\ 0.62 & 0.79 & 1 \\ 1.84 & 1.35 & 1 \\ 4.10 & 2.03 & 1 \end{pmatrix}, \quad \mathbf{\Theta} = \begin{pmatrix} a_2 \\ a_1 \\ b \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 5.19 \\ 2.64 \\ 1.40 \\ 0.99 \\ 1.36 \\ 1.43 \\ 3.36 \\ 4.94 \end{pmatrix}$$

$$\hat{\mathbf{\Theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Polynomial Regression: Example

Using NumPy we obtain $\hat{\boldsymbol{\Theta}} = \begin{pmatrix} 0.989 \\ 0.113 \\ 0.935 \end{pmatrix}$ giving us the line

$y = 0.989x^2 + 0.113x + 0.935$



Summary: Linear regression

- Linear regression assumes that our data is of a particular form: $f(\vec{x}) = \sum_{i=1}^{n} a_i x_i + b$
- We can find an analytic solution to minimize the error and thereby give the best fitting parameters:

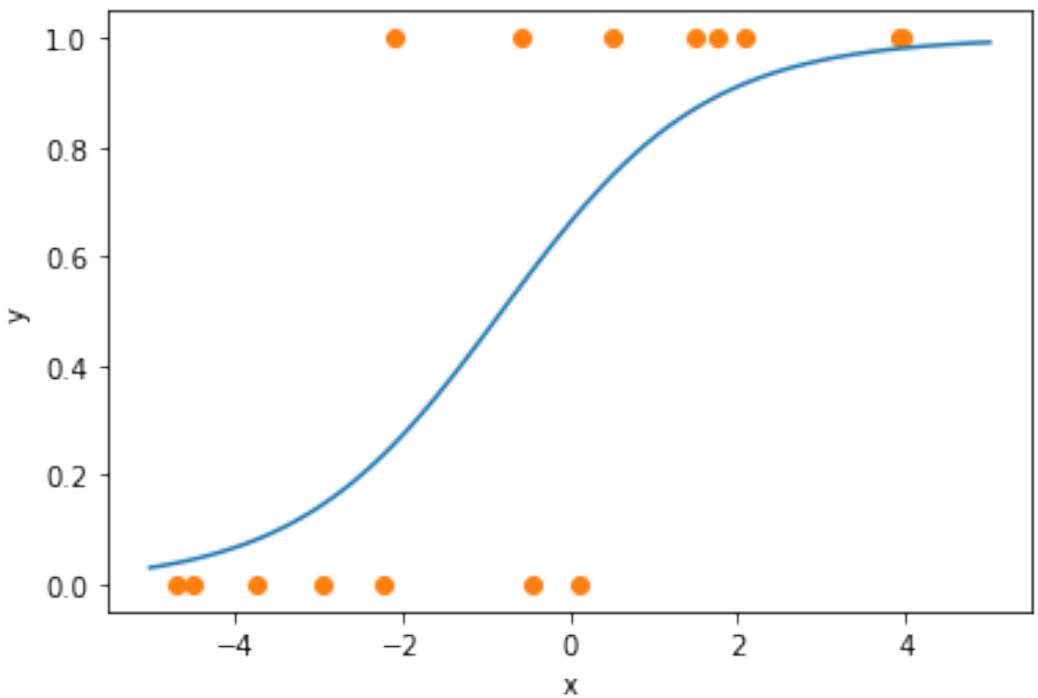
$$\hat{\mathbf{\Theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- We can also fit polynomial curves using the same methods
- Worksheet: implement linear regression using this analytic solution

Logistic Regression

- Logistic regression is a **classification** algorithm! Suppose we want to find a function $f : \Omega \to \{0, 1\}$, and our dataset is of the form $\{(\vec{x}_1, y_1), ...(\vec{x}_k, y_k)\}$.
- We assume that f has a particular form:

$$f(\vec{x}) = \sigma\left(\sum_{i=1}^{n} a_i x_i + b\right) = \frac{1}{1 + e^{(-\sum_{i=1}^{n} a_i x_i + b)}}$$



Logistic Regression

- As in the linear case, suppose we add an extra 1 to our $\vec{x}$ and suppose we express our parameters a_i and b as a vector Θ, then we have

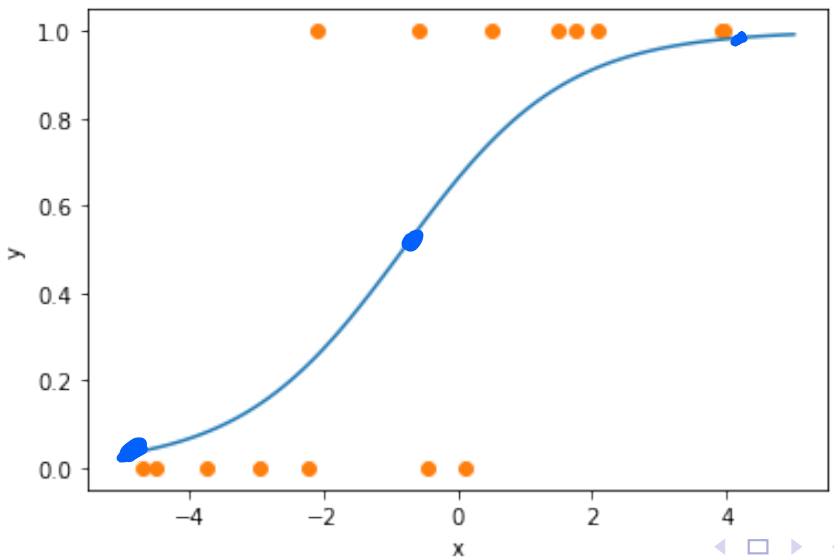
$$f_\Theta(\vec{x}) = \sigma(\Theta^T \vec{x}) = \frac{1}{1 + e^{-\Theta^T \vec{x}}}$$

$(a_1 \dots a_n b)$ $\begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix}$

- We have:

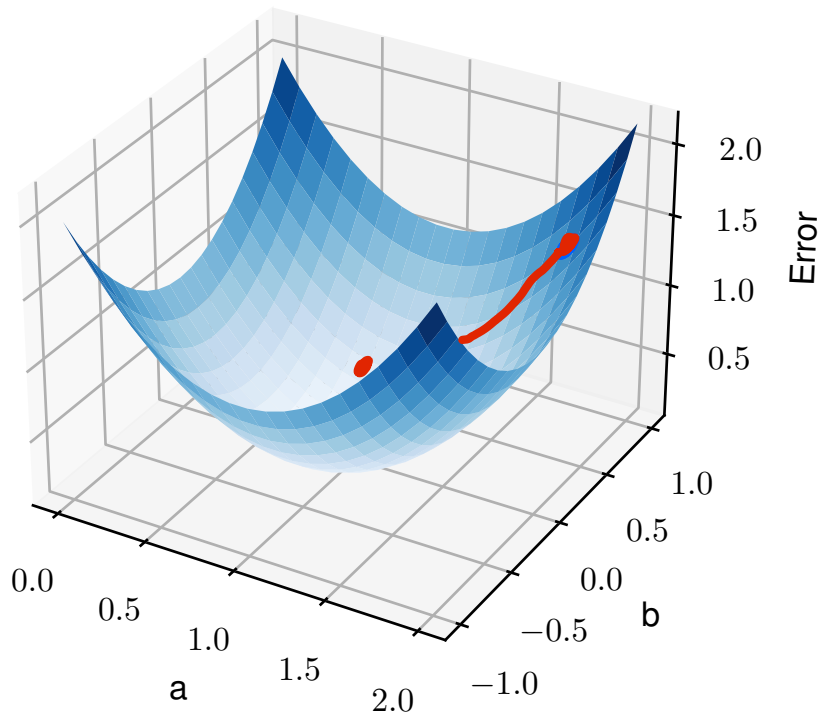
$$P(y^{(i)} = 1) = f_\Theta(\vec{x}^{(i)}) = \sigma(\Theta^T \vec{x}^{(i)})$$

$$P(y^{(i)} = 0) = 1 - P(y^{(i)} = 1) = 1 - \sigma(\Theta^T \vec{x}^{(i)})$$



How do we find Θ?

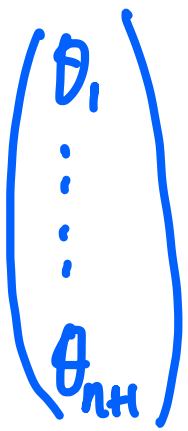
We will find an expression that we want to minimise and then use **gradient descent**.



Gradient descent

cost function

Given a function $J(\Theta)$ that we want to minimise, we start with some random values of Θ and update them by setting



$$\Theta_j \mapsto \Theta_j - \alpha \frac{\partial J(\Theta)}{\partial \Theta_j}$$

$$\Theta \mapsto \Theta - \alpha \nabla_\Theta J(\Theta)$$

What is $J(\Theta)$?

Remember we want to classify our data points as 0 or 1, i.e. dogs or cats, safe or dangerous, etc.

$$\begin{cases} P(y^{(i)} = 1) = f_\Theta(\vec{x}^{(i)}) = \sigma(\Theta^T \vec{x}^{(i)}) \\ P(y^{(i)} = 0) = 1 - P(y^{(i)} = 1) = 1 - \sigma(\Theta^T \vec{x}^{(i)}) \end{cases}$$

Note that we can rewrite this as:

$$P(y|\vec{x}, \Theta) = f_\Theta(\vec{x})^y (1 - f_\Theta(\vec{x}))^{(1-y)}$$

$y, \vec{x}$

1

What is $J(\Theta)$?

We want to maximise the probability of the conjunction of all the $y^{(i)}$, giving us

$$L(\Theta) = \prod_{i=1}^{n} P(y^{(i)}|\vec{x}^{(i)}, \Theta)$$

$$\prod_{i=1}^{n} f_\Theta(\vec{x}^{(i)})^{y^{(i)}}(1 - f_\Theta(\vec{x}^{(i)}))^{(1-y^{(i)})}$$

That's a lot of things multiplied together! And we need to find the derivative, so that we can do gradient descent. What can we do? We take the logarithm. Note we have also added a negative sign so that we can minimise.

$$J(\Theta) = -\sum_{i=1}^{n} y^{(i)} \log(f_\Theta(\vec{x}^{(i)})) + (1 - y^{(i)}) \log((1 - f_\Theta(\vec{x}^{(i)})))$$

Finding the gradient

We find the gradient of this expression, which turns out to be

$$\frac{\partial J(\Theta)}{\partial \Theta_j} = \sum_i x_j^{(i)}(f_\Theta(x^{(i)}) - y^{(i)})$$

(we can go through this derivation in the live lecture if you'd like)
Then the vector of derivatives is given by

$$\nabla_\Theta J(\Theta) = \sum_i x^{(i)}(f_\Theta(x^{(i)}) - y^{(i)})$$

Putting it together

Given a set of data and targets $\{(\vec{x}^{(i)}, y^{(i)})\}$, where $y^{(i)} \in \{0, 1\}$, we can learn the parameters Θ of a function

$$f_\Theta(\vec{x}) = \sigma(\Theta^T \vec{x}) = \frac{1}{1 + e^{-\Theta^T \vec{x}}}$$

by updating the parameters Θ via the equation

$$\Theta \mapsto \Theta - \alpha \nabla_\Theta J(\Theta)$$

$$\Theta \mapsto \Theta - \alpha \sum_i \vec{x}^{(i)} (f_\Theta(x^{(i)}) - y^{(i)})$$

We will revisit the concept of gradient descent in later weeks.

Revisiting the linear case

We can also use this strategy in the case of linear regression, for example if we have many features. In the case of linear regression our function f_Θ is

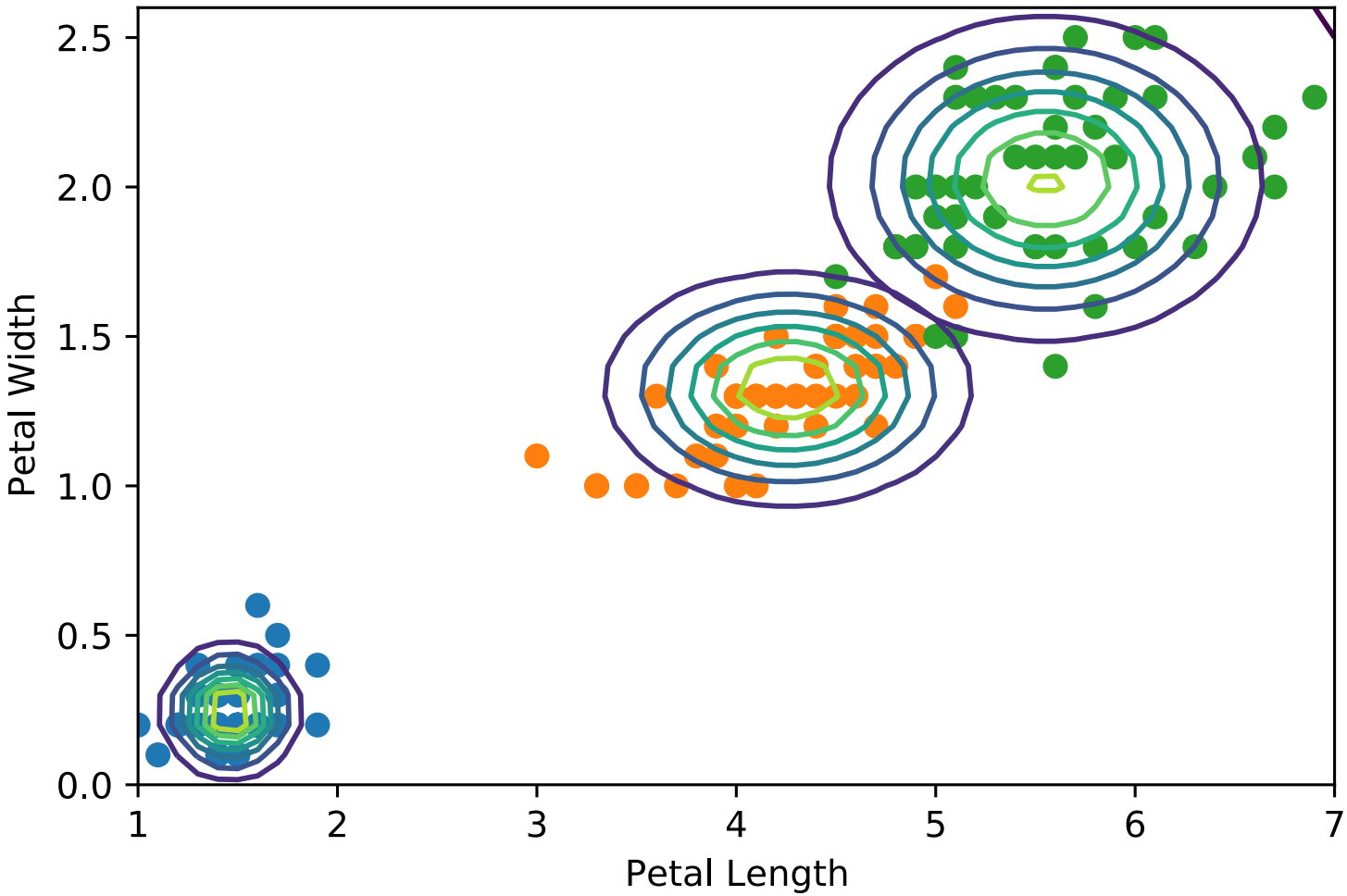
$$f_\Theta(\vec{x}) = \Theta^T \vec{x}$$

And the function we want to minimise is

$$J(\Theta) = \frac{1}{2} \sum_{i=1}^{n} (f_\Theta(\vec{x}^{(i)} - y^{(i)})^2$$

What is ∇_Θ? How does it compare to the logistic regression case?

Naive Bayes



Naive Bayes Classifier

- Consider a classification problem where $f : \Omega \rightarrow \{c_1, \ldots, c_k\}$ where Ω is an n-dimensional feature space and c_i are classes.
- A Bayesian classifier applies Bayes theorem so that for input pattern $(x_1, \ldots, x_n)$;

$$P(c_i|x_1, \ldots, x_n) = \frac{P(x_1, \ldots, x_n|c_i)P(c_i)}{\sum_{j=1}^{k} P(x_1, \ldots, x_n|c_j)P(c_j)}$$

- The classifier returns the prediction

$$\hat{c} = argmax_i P(c_i|x_1, \ldots, x_n) = \frac{P(x_1, \ldots, x_n|c_i)P(c_i)}{\sum_{j=1}^{k} P(x_1, \ldots, x_n|c_j)P(c_j)}$$

That is, it returns the class that has the highest probability conditioned on the datapoint $(x_1, ...x_n)$

Naive Bayes Classifier

- A Bayesian classifier applies Bayes theorem so that for input pattern $(x_1, \ldots, x_n)$;

$$P(c_i|x_1, \ldots, x_n) = \frac{P(x_1, \ldots, x_n|c_i)P(c_i)}{\sum_{j=1}^{k} P(x_1, \ldots, x_n|c_j)P(c_j)}$$

- The quantity $\sum_{j=1}^{k} P(x_1, \ldots, x_n|c_j)P(c_j)$ is equal to $P(x_1, \ldots, x_n)$. This means it doesn't depend on the class c_j.
- Therefore we have

$$P(c_i|x_1, \ldots, x_n) \propto P(x_1, \ldots, x_n|c_i)P(c_i)$$

And

$$\hat{c} = argmax_i P(x_1, \ldots, x_n|c_i)P(c_i)$$

Estimating Likelihoods

$$\hat{c} = argmax_i P(x_1, \ldots, x_n | c_i) P(c_i)$$

- The quantity $P(c_k)P(x_1, x_2, ..., x_n|c_k)$ is equal to $P(c_k, x_1, x_2, ..., x_n)$. However, we don't know this distribution. We factor out as follows:

$$\begin{aligned} P(c_k, x_1, x_2, ..., x_n) &= P(x_1, x_2, ..., x_n, c_k) \\ &= P(x_1|x_2, ..., x_n, c_k)P(x_2|x_3, ..., x_n, c_k)...P(x_n|c_k)P(c_k) \\ &= P(c_k)\prod_i P(x_i|x_{i+1}, ..., x_n, c_k) \end{aligned}$$

- We can estimate $P(c_j)$ from the training data by

$$P(c_j) = \frac{N_j}{N}$$

where N is the total number of training patterns and N_j is the number of training patterns labelled as class c_j.

The Curse of Dimensionality

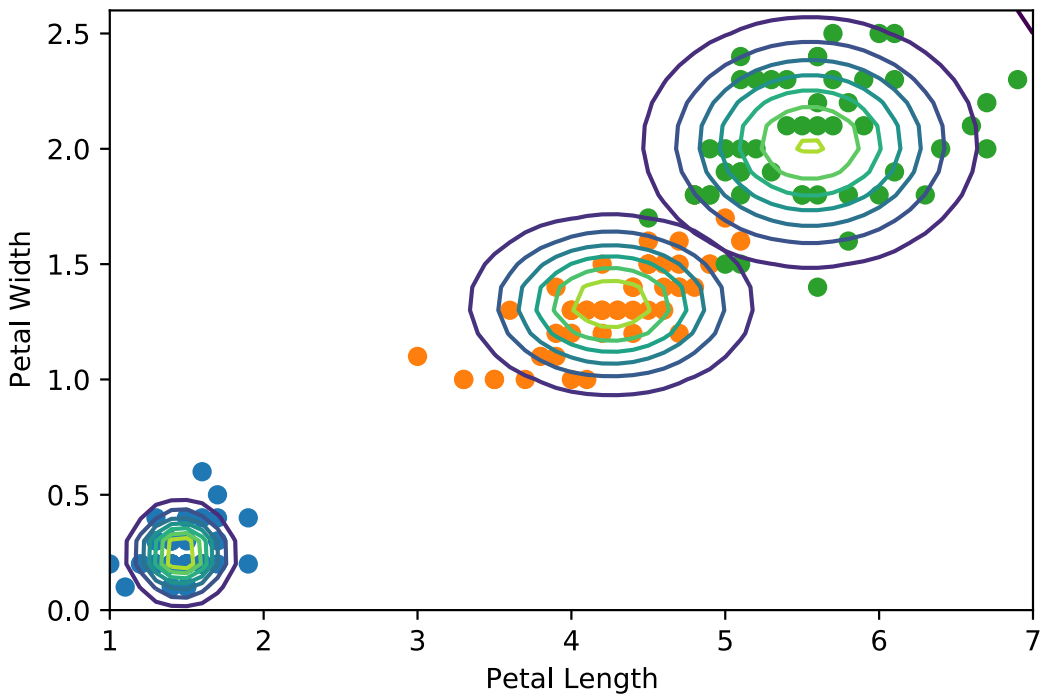
- Estimating $P(x_1, \ldots, x_n | c_j)$ is difficult due to the *curse of dimensionality*.
- The amount of data required to accurately estimate probabilities grows exponentially with dimension.
- If each x_i for $i = 1, \ldots, n$ are binary variables, specifying $P(x_1, \ldots, x_n | c_j)$ requires $2^n - 1$ values.
- If we make the assumption that the variables $x_1, \ldots, x_n$ are independent of each other given c_j then the problem reduces to one of estimating n 1-dimensional distributions.

$$P(x_1, \ldots, x_n | c_j) = \prod_{i=1}^{n} P(x_i | x_{i+1}, ..., x_n, c_k) = \prod_{i=1}^{n} P(x_i | c_j)$$

Continuous Variables

- In the case that x_i is a continuous variable we can use density estimation to determine the required conditional distributions.
- For example, a simple parameterised approach is to fit a Gaussian distribution.

$$\hat{c} = argmax_i P(x_1, \ldots, x_n | c_i) P(c_i) = P(c_i) \prod_{j=1}^{n} P(x_j | c_i)$$



Naive Bayes: Summary

- Given a datapoint, compute probability of each class and return the class that has the highest probability
- Makes the naive assumption that features are not correlated
- Use statistics of the dataset to build distributions for a datapoint given the class

Worksheet:

- We use a naive Bayes classifier where the probability distributions of each feature, given the class, are modelled by normal distributions
- You will code your own naive Bayes classifier and compare with the the implementation in sklearn
- You will also compare the performance of the naive Bayes classifier with the k-nearest neighbours classifier

More information:
Manning, Raghavan and Schütze, Introduction to Information Retrieval, 2008, Chapter 13 sec. 4 `https://nlp.stanford.edu/IR-book/pdf/13bayes.pdf`
Jurafsky and Martin, Statistical Language Processing Chapter 4
`https://web.stanford.edu/~jurafsky/slp3/4.pdf`

Summary and next week

- We have gone through 3 key simple supervised learning algorithms
- We will visit other supervised learning algorithms in more detail in future weeks
- Next week: unsupervised learning algorithms