IAML - Week 4

Antonio León Villares

October 2021

Contents

1 Linear Regression

- Aims to predict a continuous output, given some vectorised input
- Linear comes from the parameters, which allows us to transform the labels non-linearly
- Using the Pseudo-Inverse, we can efficiently compute the model parameters (weights)

1.1 Defining the Linear Regression Problem

- **How do classification and regression differ?**
  - in classification, we have a set of data points, and discrete class labels. Given a new data point, we aim to predict its class label.
  - in regression, we have a set of data points, and continuous class labels (a real number). Given a new data point, we aim to predict an associated real number.
  - for example, predicting IQ score given test scores in Maths, History and English
- **How does training data look like in linear regression?**
  - formally, training is done using a set of n data points of the form:

$$\{\underline{x}_i, y_i\}_{i=1}^{n}$$

    where:
    - $\underline{x}_i \in \mathbb{R}^D$ represent all the features of a given observation as a vector
    - $y_i \in \mathbb{R}$ represent the value associated to the observation
    - for example, if we have a fish, $\underline{x}_i$ can be a set of measurements such as weight, length, and fin length, and y_i could be the age of the fish. For example:

$$\underline{x}_1 = \begin{pmatrix} 200 \\ 1.6 \\ 0.2 \end{pmatrix} \qquad y_1 = 6$$

      corresponds to a single observation of a 6 year old fish, with weight $200kg$, length $1.6m$, and fin length of $0.2m$.
- **Is regression very simple?**
  - it is, if we consider the fact that data is typically not linear

- however, it is more powerful than it seems, and can be applied even for non-linear data
- moreover, it helps teach the principles of more complex regression/classification algorithms

- **In what cases can regression be used?**
  - predicting position of robot arm after a set of forces are applied
  - predicting electricity requirements for the grid 2 days in advanced
  - predict machine failure based on environment and usage

1.2 Linear Models

- **How can we define a linear model for prediction?**
  - a linear model can be thought as a linear combination of all the features in an observation $\underline{x}$:

$$f(\underline{x}, \underline{w}) = w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots + w_D x^{(D)}$$

  - the vector $\underline{w} \in \mathbb{R}^D$ contains the **weights** (or **parameters**) of the linear model:

$$\underline{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$$

  - fitting the model requires us to find $\underline{w}$, such that the predictive power of $f(\underline{x}, \underline{w})$ is the greatest
  - more generally, we can define a vector function $\phi(\underline{x})$:

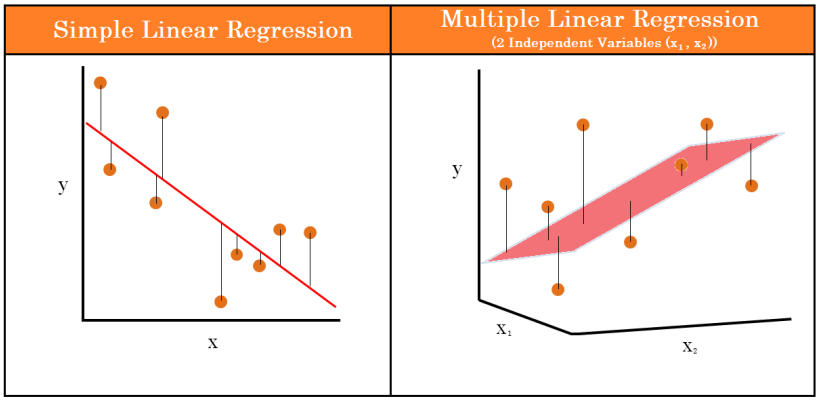
$$\phi(\underline{x}) = \begin{pmatrix} 1 \\ x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \end{pmatrix}$$

    and thus, define the linear model in terms of matrix multiplication:

$$f(\underline{x}, \underline{w}) = \underline{w}^T \phi(\underline{x})$$

- **What is the geometric interpretation of linear model?**
  - if we have D features, $f(\underline{x}, \underline{w})$ will correspond to a hyperplane in $D + 1$ dimensional space (we have D independent variables, and 1 dependent variable y)

- if we have 2 features, this is just a straight line (line of best fit)
- if we have 3 features, this is just a plane



- **What is a design matrix?**
  - a **design matrix $\mathbf{\Phi}$** contains all our training information as a matrix:

$$\mathbf{\Phi} = \begin{pmatrix} \phi(\underline{x}_1)^T \\ \phi(\underline{x}_2)^T \\ \vdots \\ \phi(\underline{x}_n)^T \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(D)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(D)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(D)} \end{pmatrix}$$

  - $\mathbf{\Phi}$ is an $n \times (D+1)$ matrix (n observation, $D+1$ variables)
  - each *row* in the design matrix corresponds to 1 observation of the data
  - each *column* in the design matrix corresponds to the value of a single feature across all the observed data

1.3 Finding Model Parameters: The Pseudo-Inverse Matrix

See this stack exchange for the derivation of the Pseudo-Inverse Matrix

- **How can the regression problem be formalised in terms of the design matrix?**
  - we can define a **target** vector containing all the expected labels:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_D \end{pmatrix}$$

- then, our regression problem becomes finding the weights $\underline{w}$ such that:

$$\underline{\hat{y}} = \mathbf{\Phi}\underline{w}$$

where $\underline{\hat{y}}$ is our **prediction**, and we expect $\underline{\hat{y}}$ to be as close as possible to $\underline{y}$

- expanded:

$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_D \end{pmatrix} = \begin{pmatrix} w_0 + w_1 x_1^{(1)} + w_2 x_1^{(2)} + \ldots + w_D x_1^{(D)} \\ w_0 + w_1 x_2^{(1)} + w_2 x_2^{(2)} + \ldots + w_D x_2^{(D)} \\ \vdots \\ w_0 + w_1 x_n^{(1)} + w_2 x_n^{(2)} + \ldots + w_D x_n^{(D)} \end{pmatrix}$$

- **Why can't we use inverses to compute the weights?**
  - naively, we could think that we just need to invert $\mathbf{\Phi}$, such that:

$$\underline{w} = \mathbf{\Phi}^{-1}\underline{w}$$

  - however, $\mathbf{\Phi}$ need not be square, so its inverse might not exist. This is a consequence of the fact that if we have more equations than variables, the system is **overdetermined**, and it generally has no solutions (unless certain rows are scalar multiples of each other (see this for a visual interpretation)

- **How can we find the model parameters using the Pseudo-Inverse Matrix?**
  - we can't find $\underline{w}$ such that our predictions are perfect
  - thus, we seek $\underline{w}$ such that it *minimises* the error between $\underline{\hat{y}}$ and $\underline{y}$
  - we can use the **sum of squared residuals**:

$$\begin{aligned} O(\underline{w}) &= \sum_{i=1}^{n} (y_i - f(\underline{x}_i, \underline{w}))^2 \\ &= \sum_{i=1}^{n} (y_i - \underline{w}^T \phi(\underline{x}_i))^2 \\ &= (\underline{y} - \mathbf{\Phi}\underline{w})^T (\underline{y} - \mathbf{\Phi}\underline{w}) \end{aligned}$$

  - thus, if we minimise $O(\underline{w})$ with respect to $\underline{w}$, we will obtain $\underline{w}$ such that the residuals are minimised: that is, our prediction approximates the real values in the best way
  - if we do this (partial derivatives with respect to $\underline{w}$, and set the derivatives to 0), we obtain:

$$\underline{w} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \underline{y}$$

- $(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T$ is known as the **pseudo-inverse** of $\mathbf{\Phi}$, since it *acts* as an inverse of the matrix (you can verify that $(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{\Phi} = \mathbf{\Phi}(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T = \mathbb{I}$)

- **What are the weights if a model has no features?**
  - if there are no features, then:
$$\mathbf{\Phi} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$
  - so:
$$\mathbf{\Phi}^T\mathbf{\Phi} = 1 + 1 + \underset{n \text{ times}}{\ldots} + 1 = n$$
  - but then:
$$(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\underline{y} = \frac{y_1 + y_2 + \ldots + y_n}{n}$$
  - in other words, if there are no features, the regression model will always predict the **average** of the target labels

- **What is the general strategy when developing learning algorithms?**
  1. Define a **task** (i.e *regression*)
  2. Decide on **model structure** (i.e *linear regression model)*
  3. Decide on **score function** (i.e *root mean square error)*
  4. Decide on **optimisation** (i.e *analytic solution via partial derivatives)*

1.4 Interpreting Linear Models Probabilistically

- Assume that $y = \mathbf{w}^T\mathbf{x} + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$
- (This is an exact linear relationship plus Gaussian noise.)
- This implies that $y|\mathbf{x}_i \sim N(\mathbf{w}^T\mathbf{x}_i, \sigma^2)$, i.e.

$$-\log p(y_i|\mathbf{x}_i) = \log\sqrt{2\pi} + \log\sigma + \frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}$$

- So minimising $O(\mathbf{w})$ equivalent to maximising likelihood!
- Can view $\mathbf{w}^T\mathbf{x}$ as $E[y|\mathbf{x}]$.
- Squared residuals allow estimation of σ^2

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1.5 Problems With Regression

- **How do outliers affect linear regression models?**
  - outliers can greatly affect the regression parameters:

- **Should we visualise linear regression models?**

- visualising data is always useful:
  - check for outliers
  - check if relationship is linear
  - is there a structure to the residuals
- in general visualising residuals can also indicates good model if there is no correlation between residuals and outputs

1.6 General Regression

- **What is general regression?**
  - we have discussed multiple linear regression as a means to compute a single target value
  - we can generalise this, as to output a **vector**, by computing new models for each vector component
  - that is, we calculate a new set of weights $\underline{w}_i$ for each component
- **How can we fit a general regression model?**
  - to compute each set of weights, we can still use the pesudo-inverse method

1.7 Basis Expansion

The following are what I used to further understand the use of basis functions:

- Medium: Nonlinear Regression Tutorial with Radial Basis Functions
- Princeton: Features and Basis Functions
- Toronto: Modeling Data with Linear Combinations of Basis Functions
- Edinburgh: Linear regression
- Neil Lawrence's Talks: Basis Functions

1.7.1 Basis Functions for Non-Linear Regression

- **Why is linear regression called linear?**
  - contrary to what is expected, linear regression is linear precisely because the set of weights is linear
  - in other words, linear regression can still be performed even if we are considering non-linear features, such as:
    - $x_1 \times x_2$
    - x_1^4
    - e^{x_1}

- **Can we use linear regression even if the underlying data is non-linear?**
  - yes. From the above, it is implied that we can modify each data input, such as to remove non-linearities, and then fit a regresion model on the modified inputs
  - for example, if we have quadratic data, we can preprocess it, square root the values, and then use them for our regression model
- **What are basis functions?**
  - we use **basis functions** to attempt to convert non-linear data to linear data
  - called thus because wea are constructing a new basis for the data, based on these functions
- **How can we generalise design matrices when using basis functions?**
  - if we use basis functions, we can pick an arbitrarily large number of them. This changes the number of columns for our design matrix:

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_1(\underline{x}_1) & \phi_2(\underline{x}_1) & \dots & \phi_m(\underline{x}_1) \\ \phi_1(\underline{x}_2) & \phi_2(\underline{x}_2) & \dots & \phi_m(\underline{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\underline{x}_n) & \phi_2(\underline{x}_n) & \dots & \phi_m(\underline{x}_n) \end{pmatrix}$$

  - each ϕ_i is taken an input vector $\underline{x}_j$ and converting it into a real number, by means of combining the observations in $\underline{x}_j$ in different ways
  - for example, our original design matrix was defined by:

$$\phi_i(\underline{x}_j) = x_j^{(i)}$$

    in other words, just extracting the i component of $\underline{x}_j$:

$$\boldsymbol{\Phi} = \begin{pmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(D)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(D)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(D)} \end{pmatrix}$$

1.7.2 Polynomial Regression

For example, if we want to do a quadratic regression, we can define:

$$\mathbf{\Phi} = \begin{pmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(D)} & \left(x_1^{(1)}\right)^2 & \left(x_1^{(2)}\right)^2 & \dots & \left(x_1^{(D)}\right)^2 \\ 1 & x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(D)} & \left(x_2^{(1)}\right)^2 & \left(x_2^{(2)}\right)^2 & \dots & \left(x_2^{(D)}\right)^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(D)} & \left(x_n^{(1)}\right)^2 & \left(x_n^{(2)}\right)^2 & \dots & \left(x_n^{(D)}\right)^2 \end{pmatrix}$$

or

$$\mathbf{\Phi} = \begin{pmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(D)} & x_1^{(1)}x_1^{(2)} & x_1^{(2)}x_1^{(3)} & \dots & x_1^{(n-1)}x_1^{(n)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(D)} & x_2^{(1)}x_2^{(2)} & x_2^{(2)}x_2^{(3)} & \dots & x_2^{(n-1)}x_2^{(n)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(D)} & x_n^{(1)}x_n^{(2)} & x_n^{(2)}x_n^{(3)} & \dots & x_n^{(n-1)}x_n^{(n)} \end{pmatrix}$$

The way in which we combine the variables is really up to us, as we can technically use as many basis functions as we want to make these combinations.

If we have a single feature, then we can easily express it as:

$$f(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{i=0}^{M} w_i x^i$$

which can be expressed via:

$$\phi(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ \dots \\ x^M \end{pmatrix}$$

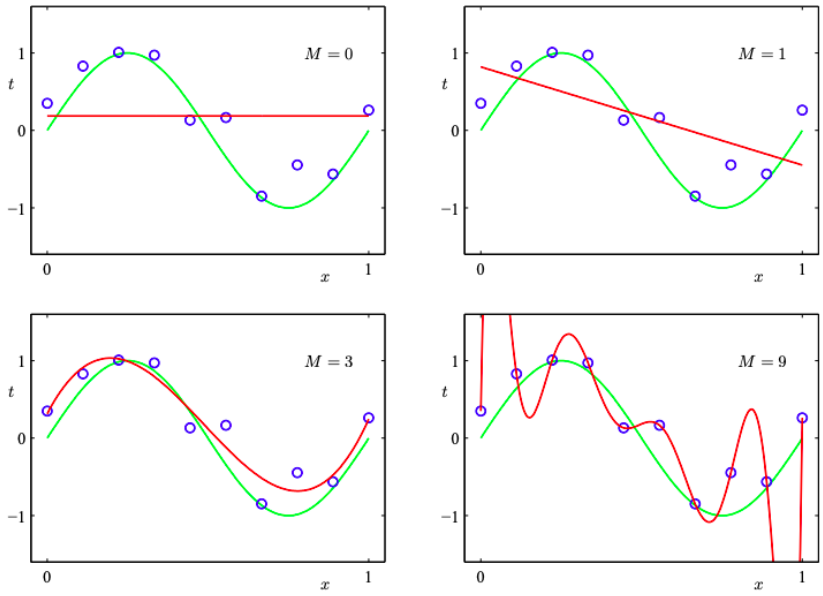


Figure 1: Polynomial Regression using M basis functions

1.7.3 Dealing With Categorical Features

- **What is a bad way of representing categorical features in linear regression?**
  - it would be a bad decision to encode categorical numbers using numbers (ordinally). For example, if one of the features is an operating system:
    - Linux: 1
    - MacOS: 2
    - Windows: 3
  - but then, observations with OS as Windows will lead to a systematically larger regression outputs than the rest
- **What is a good way of representing categorical features in linear regression?**
  - a better strategy is to create new features based on each possible category:
    - x_1: 1 if Linux, 0 otherwise
    - x_2: 1 if MacOS, 0 otherwise
    - x_3: 1 if Linux, 0 otherwise
  - then, by calculating the weights we can adjust the regression output based on which OS is used

1.7.4 Radial Basis Functions

- **What are radial basis functions?**
  - radial basis functions are a particular type of transformation, based on using a Normal Distributions:

$$\phi_i(\underline{x}) = \exp\left(-\frac{|\underline{x} - \underline{c}_i|^2}{2\alpha^2}\right)$$

    where we need to determine $\underline{c}_i$ and α
  - $\underline{c}_i$ are known as **centres**, usually constructed by using subsets of the datapoints
  - a radial basis function is thus larger the more that $\underline{x}$ is close to $\underline{c}_i$
  - we can think of $\underline{c}_i$ as μ in the normal distribution, and of α as σ
  - to find the weights, we can still use the pseudo-inverse method
  - **Example**:

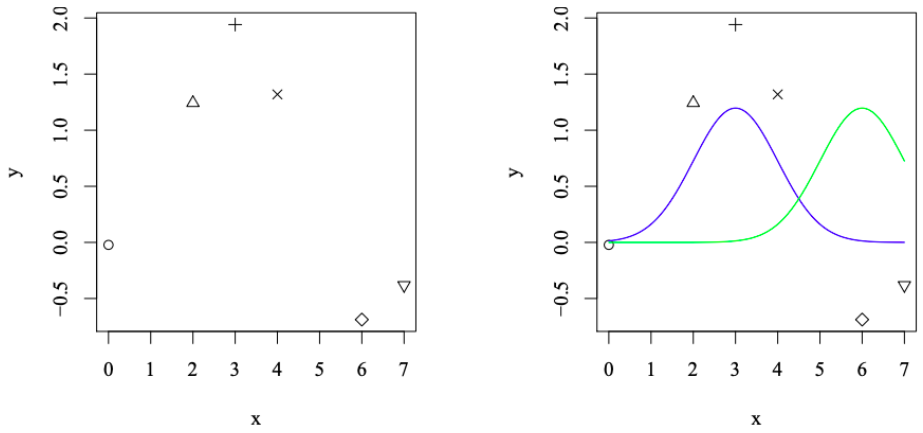


Figure 2: Take this data, and define 2 radial basis functions. ϕ_1 is centered at $c_1 = 3$ (blue curve). ϕ_2 is centered at $c_2 = 6$ (green curve).

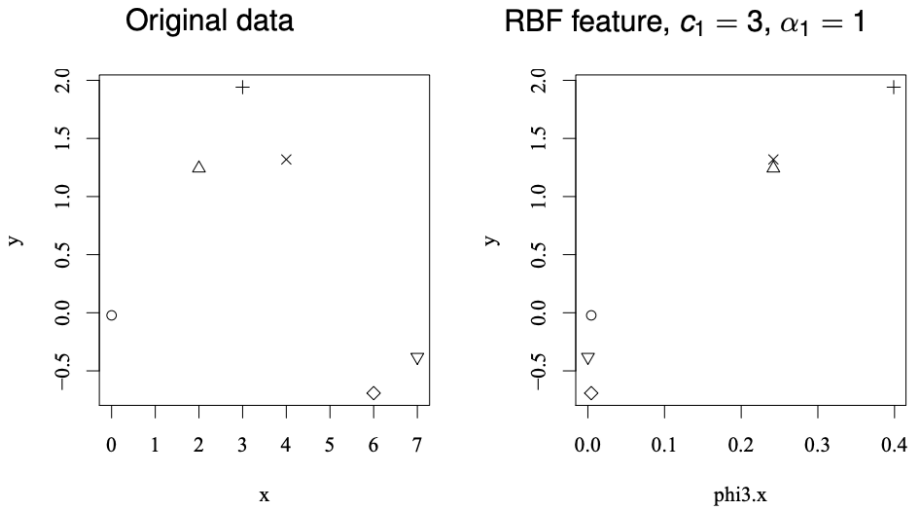


Figure 3: Visualising how ϕ_1 affects the data, we can see that the square cross has the highest value (0.4), since it was closest to $c_1 = 3$

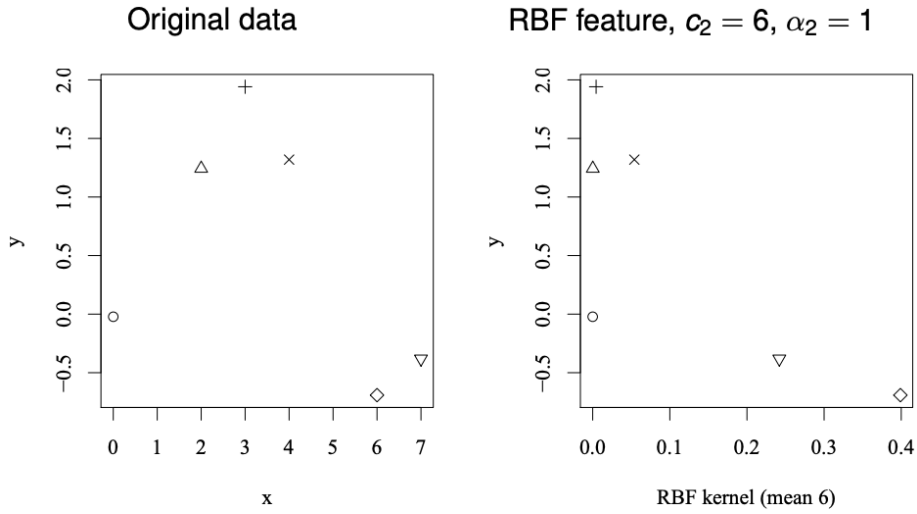


Figure 4: Visualising how ϕ_2 affects the data, we can see that the rhombus has the highest value (0.4), since it was closest to $c_2 = 6$

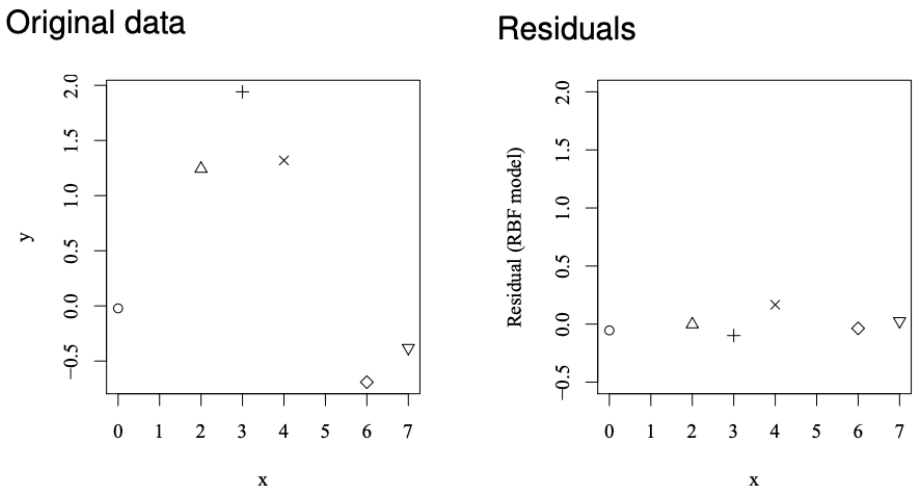


Figure 5: We get a design matrix:

$$\mathbf{\Phi} = \begin{pmatrix} \phi_1(\circ) & \phi_2(\circ) \\ \phi_1(\triangle) & \phi_2(\triangle) \\ \vdots & \vdots \\ \phi_1(\diamond) & \phi_2(\diamond) \end{pmatrix}$$

. If we apply the pseudo-inverse method, we get the residuals on the right, which are very good.

- **What are the issues associated with radial basis functions?**
  - in high dimensions, might need a lot of functions (i.e centre each function at each point of the data), which can lead to overfitting
  - we still don't know how to set $\underline{c}_i$ or α

2 Logistic Regression

- Logistic regression is a discriminative classification model which uses the logistic function to determine probabilities of classifying as a certain class
- Whilst the weights can't be computed analytically, numerical methods can be used
- Linear classifiers produce decision boundaries which are hyperplanes
- Basis functions can be used to solve the linearly separable problem

2.1 Linear Classifiers and Decision Boundaries

For more details on linear classifiers

- **What is a linear classifier?**
  - a classifier in which the outcome is computed by considering a **linear combination** of given **features**
- **What is a decision boundary?**
  - classifiers split the feature space into regions in which one class is selected over the others
  - the boundary between these regions is the **decision boundary**
- **What is the decision boundary of a linear classifier?**
  - linear classifiers produce **linear** decision boundaries (generalising, they produce a hyperplane)
- **How can we represent a binary linear classifier?**
  - a binary, linear classifier will be a line in feature space dividing it into 2 regions, via:
    $$f(\underline{x}, \underline{w}) = \underline{w}^T \underline{x} + b$$
    where:
    - $\underline{w}$ are the parameters of the line (control the "angle"), which we learn from data
    - $\underline{x}$ are the features
    - b is a bias (moves the decision boundary up/down)
  - if we have 2 class labels $y = 0$ or $y = 1$, we typically classify based on a threshold value r:
    $$\hat{y} = \begin{cases} 1, & f(\underline{x}, \underline{w}) \geq r \\ 0, & f(\underline{x}, \underline{w}) < r \end{cases}$$

However, r and b will be constants, so without loss of generality:

$$\hat{y} = \begin{cases} 1, & f(\underline{x}, \underline{w}) \geq 0 \\ 0, & f(\underline{x}, \underline{w}) < 0 \end{cases}$$

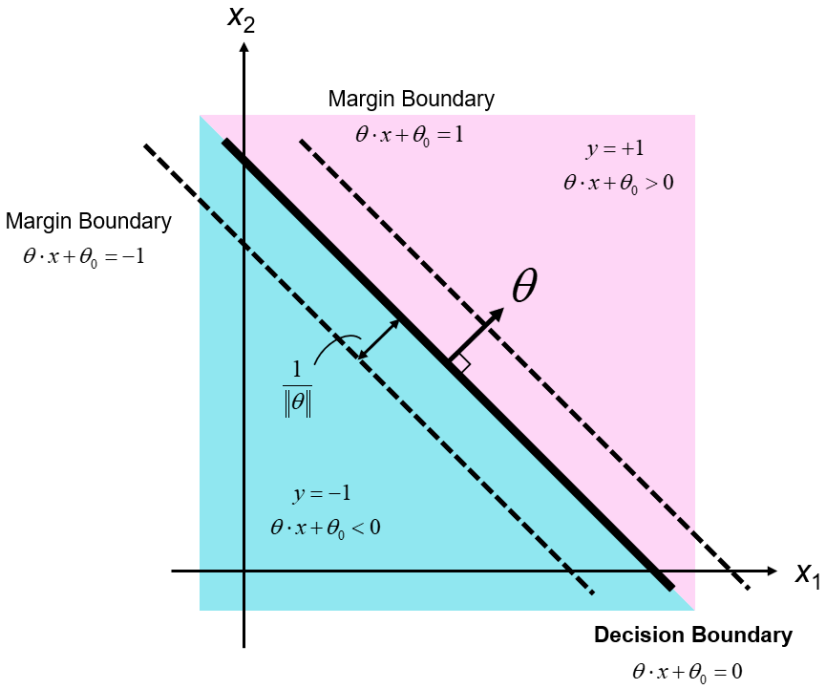


Figure 6: A binary linear classifier. By construction, the parameters $\underline{w}$ will be a vector perpendicular to the line

2.2 The Logistic Function

- **Can we use linear regression for binary classification?**
  - the above looks very similar to linear regression, but the training parameters are completely different (one seeks a line that divides space, one seeks a line that fits the data well)
  - this can lead to the question: can we use linear regression for binary classification?
  - due to the sensitivity of linear regression towards outliers, it is unfeasible to use linear regression as a classifier

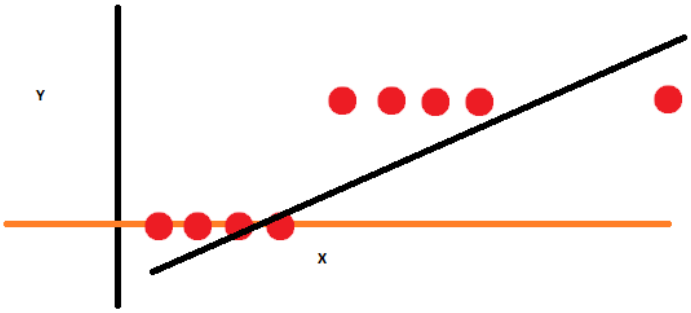


Figure 7: By adding an outlier for a certain class, the linear regression line missclassifies most of the $y = 1$ class instances

- **How can linear, probabilistic models lead to binary classification?**
  - an alternative to interpreting a linear classifier as a hyperplane splitting space (difficult to imagine for high dimensions), is to take input features, and compute the probability of said instances belonging to a class. In other words, calculating:

$$P(y = 1|\underline{x})$$

  - clearly, normal linear models of the form $\underline{w}^T\underline{x}$ are not well suited (their output is not bounded by 0 and 1, as required for probabilities)
  - here we are using $\underline{w} = \langle w_0, w_1, \ldots, w_d \rangle$ and $\langle 1, x_1, \ldots, x_d \rangle$ (more compact to just write $\underline{w}^T\underline{x}$ instead of $\underline{w}^T\underline{x} + b$)
  - instead, we seek a function, such that:

$$P(y = 1|\underline{x}) = f(\underline{w}^T\underline{x})$$

    such that f squishes the real line to values between 0 and 1
  - moreover, it should be such that:

$$P(y = 0|\underline{x}) = 1 - f(\underline{w}^T\underline{x})$$

- **Why do we use the logistic function?**
  - the **logistic function** is one such f:

$$\sigma(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

- to see why, let:

$$p = P(y = 1|\underline{x})$$

- recall the definition of log odds:

$$\ln\left(\frac{p}{1-p}\right)$$

- then, if we consider:

$$\begin{aligned} & \ln\left(\frac{p}{1-p}\right) = \underline{w}^T\underline{x} \\ \implies \quad & \frac{p}{1-p} = e^{\underline{w}^T\underline{x}} \\ \implies \quad & p = e^{\underline{w}^T\underline{x}} - pe^{\underline{w}^T\underline{x}} \\ \implies \quad & p = \frac{e^{\underline{w}^T\underline{x}}}{1 + e^{\underline{w}^T\underline{x}}} \end{aligned}$$

  which is the logistic function
- $\sigma(z)$ goes to 1 as $z \to \infty$, and to 0 as $z \to -\infty$
- moreover $\sigma(0) = 0.5$

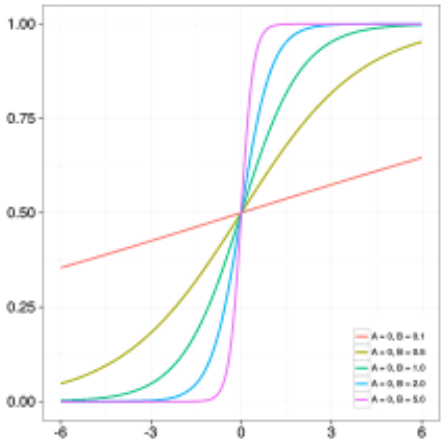


Figure 8: If we consider $\sigma(\alpha x)$, then changing α will alter the shape of the sigmoid. In particular, if α is bigger, the sigmoid will be "steeper" (i.e it converges faster to 0/1)

- **What is logistic regression?**
  - the use of the logistic function to perform classification:

$$P(y = 1|\underline{x}) = \sigma(\underline{w}^T\underline{x})$$

- comes from the fact that we use **linear** weights within a **logistic function**
- the decision boundary occurs when either class is equally likely. In other words, it is the line defined by:

$$\underline{w}^T \underline{x} = 0$$

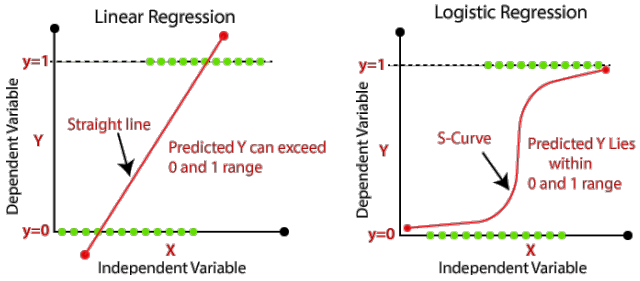


Figure 9: Logistic regression is best suited for classification.

- **How do the weights affect logistic regression classification?**
  - if we ignore the bias (this just shifts up and down):
    - $\underline{w}$ affects the direction of the hyperplane (since they are perpendicular)
    - if $|\underline{w}|$ is small, this is equivalent to $\sigma(\alpha z)$ with small alpha, so the sigmoid will be more "loose", and so, most probabilities close to the decision boundary will be close to 0.5
    - alternatively, larger $|\underline{w}|$ means that for points in the same region, they'll have probabilites closer to 0 or 1
    - in other words, $|\underline{w}|$ quantifies how certain we are about a classification

2.3 Learning Parameters for Logistic Regression

- **How can we learn the parameters of a logistic regression model?**
  - since we are dealing with probabilities, we want to **maximise** the likelihood that our parameters $\underline{w}$ explains the data. That is, we maximise:

$$L(\underline{w}) = P(D_y | D_{\underline{x}}, \underline{w})$$

    where D_y is our data (labels), and $D_{\underline{x}}$ are our data points (features)
  - to simplify the derivation, we can consider the **negative log likelihood**:

$$-\ln(L(\underline{w}))$$

    because:

* computers are better at minimising, so we negate
* logs help because they turn multiplication into addition

– assuming each data point is independent and identically distributed, the likelihood is:

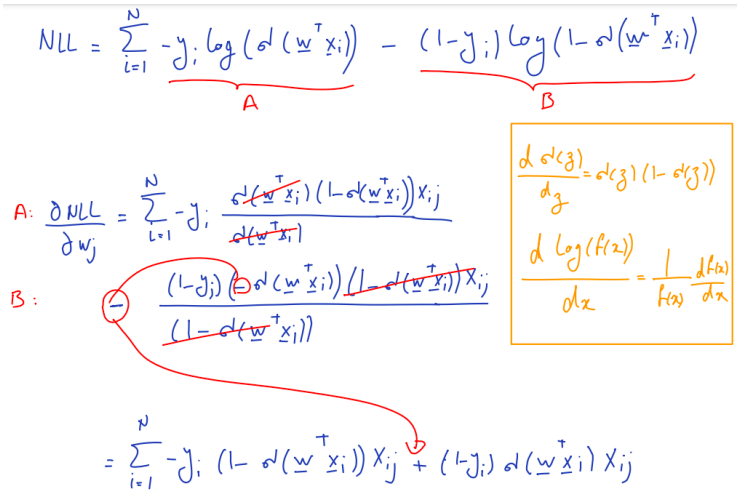
$$\begin{aligned} L(\underline{w}) &= \prod_{i=1}^{n} P(y = y_i | \underline{x}_i, \underline{w}) \\ &= \prod_{i=1}^{n} P(y_i = 1 | \underline{x}_i, \underline{w})^{y_i} \times (1 - P(y_i = 1 | \underline{x}_i, \underline{w}))^{1-y_i} \\ &= \prod_{i=1}^{n} \sigma(\underline{w}^T \underline{x}_i)^{y_i} (1 - \sigma(\underline{w}^T \underline{x}_i))^{1-y_i} \end{aligned}$$

where the second step follows from the fact that if $y_i = 1$, we consider the probability $P(y_i = 1 | \underline{x}_i, \underline{w})$, and if $y_i = 0$, we consider the probability $P(y_i = 0 | \underline{x}_i, \underline{w}) = 1 - P(y_i = 1 | \underline{x}_i, \underline{w})$

– then, the negative log likelihood is:

$$\begin{aligned} NLL(\underline{w}) &= -\ln \left(\prod_{i=1}^{n} \sigma(\underline{w}^T \underline{x}_i)^{y_i} (1 - \sigma(\underline{w}^T \underline{x}_i))^{1-y_i} \right) \\ &= -\sum_{i=1}^{n} y_i \ln(\sigma(\underline{w}^T \underline{x}_i)) + (1 - y_i) \ln(1 - \sigma(\underline{w}^T \underline{x}_i)) \end{aligned}$$

– we can then minimise by taking the partial derivative with respect to $\underline{w}$:



$$= \sum_{i=1}^{N} -y_i x_{ij} + y_i \sigma'(\underline{w}^T \underline{x}_i) x_{ij} + \sigma'(\underline{w}^T \underline{x}_i) x_{ij} - y_i \sigma'(\underline{w}^T \underline{x}_i) x_{ij}$$

$$= \sum_{i=1}^{N} -y_i x_{ij} + \sigma'(\underline{w}^T \underline{x}_i) x_{ij}$$

$$\Rightarrow \frac{\partial NLL}{\partial w_j} = \sum_{i=1}^{N} \left(\sigma'(\underline{w}^T \underline{x}_i) - y_i\right) x_{ij}$$

from which we get:

$$\frac{\partial NLL}{\partial w_j} = \sum_{i=1}^{n} (\sigma(\underline{w}^T \underline{x}_i) - y_i) x_{ij}$$

  - unfortunately, we can't anlytically solve for $\underline{w}$, so we use **numerical optimisation**, such as gradient descent

- **How does logistic regression fit into the general structure of classifiers?**
  - **Task**: discriminative classification
  - **Model Structure**: logistic regression
  - **Score Function**: log likelihood
  - **Optimisation/Search Method**: numerical optimisation (stochastic gradient descent, BFGS)

- **What is a linearly separable problem?**
  - a problem for which there exist weights $\underline{w}$ such that:
    * $y = 1 \implies \underline{w}^T \underline{x} \geq 0$
    * $y = 0 \implies \underline{w}^T \underline{x} < 0$
  - in other words, we can find a linear model which splits the data into 2 perfectly
  - an example of a non-linearly separable problem is that of XOR. This can however be solved by using a non-linear transformation for the input.
  - as with linear regression, we can use basis functions to convert inputs non-linearly

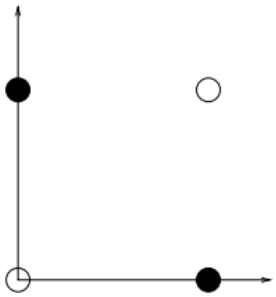


Figure 10: No line can separate the $y = 1$ and $y = 0$ labels

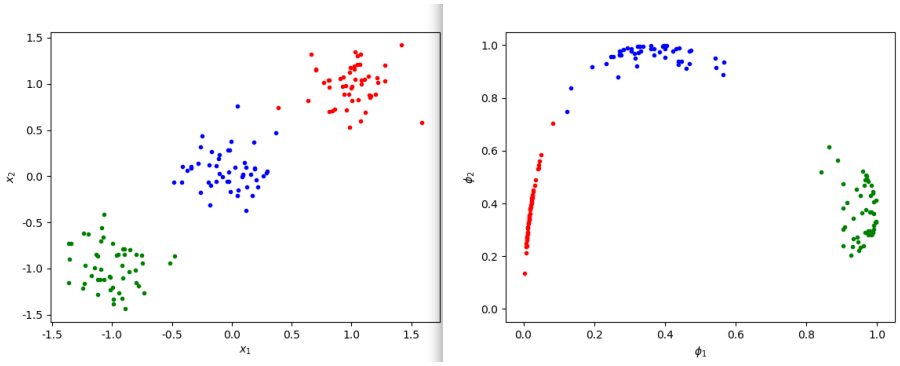


Figure 11: To the left, data which is not linearly separable. By applying 2 Gaussian Basis functions, one centered on the blue cluster (ϕ_2), and one centered in the green cluster (ϕ_1), we are able to separate them.

2.4 Logistic Regression: A Discriminative Classifier

- **Why is logistic regression a discriminative classifier?**
  - Naive Bayes is **generative**, because we modelled how to **generate** the distribution $P(\underline{x}|y)$, and then use this to compute $P(y|\underline{x})$ by using:
    $$P(y|\underline{x}) \propto P(\underline{x}|y)P(\underline{x})$$
  - on the other hand, Logistic Regression is **discriminative**, as it directly learns $P(y|\underline{x})$
  - **discriminative advantage**: what's the point of constructing a model of $P(\underline{x})$ if its given as input?
  - **generative advantage**: good at handling outliers, or missing data. Can also be used to generate new input.

- **Are there linear generative classifiers?**

  Naive Bayes can be a linear classifier in 2 cases

  1. *Gaussian data with equal covariance.* If $p(\mathbf{x}|y=1) \sim N(\boldsymbol{\mu}_1, \Sigma)$ and $p(\mathbf{x}|y=0) \sim N(\boldsymbol{\mu}_2, \Sigma)$ then

     $$p(y=1|\mathbf{x}) = \sigma(\tilde{\mathbf{w}}^T\mathbf{x} + w_0)$$

     for some $(w_0, \tilde{\mathbf{w}})$ that depends on $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, Σ and the class priors
  2. *Binary data.* Let each component x_j be a Bernoulli variable i.e. $x_j \in \{0, 1\}$. Then a Naïve Bayes classifier has the form

     $$p(y=1|\mathbf{x}) = \sigma(\tilde{\mathbf{w}}^T\mathbf{x} + w_0)$$

2.5 Logistic Regression for Multiclass Classification

Logistic regression can also be used for multiclass classification:

1. for each class k, train a logistic regression classifier for a set of weights $\underline{w}_k$. The resulting model determines whether something is k or not.
2. use the **softmax** function to determine the probability of a certain class:

$$P(y=k|\underline{x}) = \frac{\exp\left(\underline{w}_k^T \underline{x}\right)}{\sum_{i=1}^{C} \exp\left(\underline{w}_i^T \underline{x}\right)}$$