Applied Machine Learning (AML)

Linear Regression

Oisin Mac Aodha • Siddharth N.

Regression

The Regression Problem

- For **classification** problems the target is discrete, i.e. $y \in \{1, ..., C\}$
- For **regression** problems the target is continuous, i.e. $y \in \mathbb{R}$
- For **linear regression** the relationship between the features x and the target y is linear
- Although this is simple and may appear limited, it is
  - More powerful than you would expect
  - The basis for more complex nonlinear methods

Example Regression Problems

- Robot inverse dynamics: predicting what torques are needed to drive a robot arm along a given trajectory
- Electricity load forecasting: generating hourly forecasts days in advance
- Predicting staffing requirements at help desks based on historical data and product and sales information
- Predicting the time to failure of equipment based on utilization and environmental conditions
- Predicting the depth of objects in an image

The Linear Model

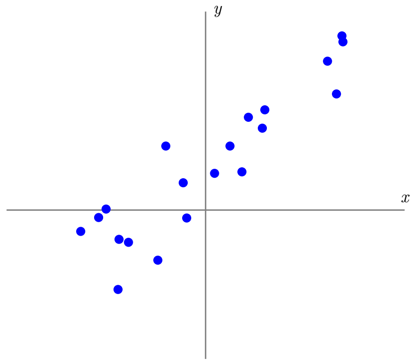
- In **simple** linear regression we have a scalar input x and a scalar output

$$\begin{aligned} f(x;\boldsymbol{w}) &= w_o + w_1 x \\ &= \boldsymbol{w}^\intercal \phi(\boldsymbol{x}) \qquad \text{i.e. the dot product} \end{aligned}$$

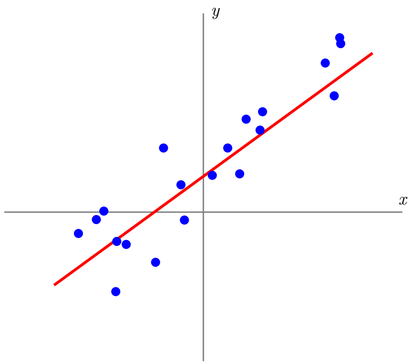
where $\boldsymbol{w} = [w_0, w_1]^\intercal$ and $\phi(x) = [1, x]^\intercal$

- We use the notation $\phi(\boldsymbol{x})$ to make generalisation easy later

Simple Linear Regression Example



Simple Linear Regression Example



The **red** line depicts our linear fit to the data with two weights/parameters, **intercept** w_0 and **slope** w_1.

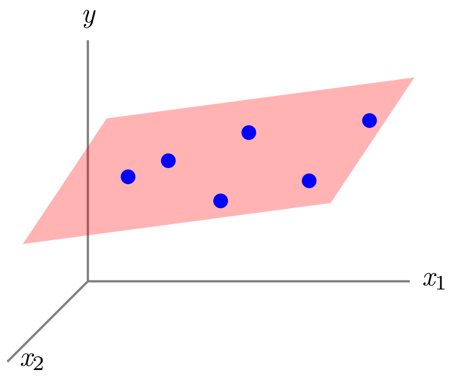
Multiple Linear Regression

- In **multiple** linear regression we have a vector $\boldsymbol{x}$ of inputs and a scalar output

$$\begin{aligned} f(\boldsymbol{x};\boldsymbol{w}) &= w_o + w_1 x_1 + \ldots + w_D x_D \\ &= w_o + \sum_{d=1}^{D} w_d x_d \\ &= \boldsymbol{w}^\intercal \phi(\boldsymbol{x}) \end{aligned}$$

where $\boldsymbol{w} = [w_0, w_1, \ldots, w_D]^\intercal$ and $\phi(\boldsymbol{x}) = [1, x_1, \ldots, x_D]^\intercal$

Multiple Linear Regression



In 2D, instead of a line, we have a **plane**.
In higher dimensions, this would be a **hyperplane**.

Multiple Linear Regression - Example

- Given information about a local habitat, the task is to predict how tall a tree will be ten years after being planted

Multiple Linear Regression - Example

- Given information about a local habitat, the task is to predict how tall a tree will be ten years after being planted

- y_i the height of a tree at location i
- $\boldsymbol{x}_i$ are features describing that habitat at location i
  - x_1 is the average rainfall
  - x_2 is the average temperature
  - x_3 is the percentage of a particular nutrient in the soil
- We will assume there is a linear relationship between these features and the target

$$\hat{y} = w_o + w_1 x_1 + w_3 x_3 + w_3 x_3$$

Interpreting the Model Weights

- Can we interpret the model weights?

$$\hat{y} = w_o + w_1 x_1 + w_3 x_3 + w_3 x_3$$

- The solved weights tell us the contribution of each feature to the final prediction, e.g.
  - A weight that is close to 0 indicates that that feature does not influence the output
  - A large **positive** value, indicates that there is a strong positive relationship
  - A large **negative** value, indicates that there is a strong negative relationship
- However, need to ensure that the data is *standardised* so that the scale of each feature is similar

Standardising the Data

- The input features may have very different scales, i.e. small versus large numbers
- To ensure that we can interpret the relative model weights across the different dimensions it is advisable to **standardise** the data

- This simply involves computing the mean and standard deviation for *each* feature dimension from the data the *training* set
- We then subtract this mean and divide by this standard deviation for the data in both the *training* and *test* sets

Fitting the Model to Data

Fitting the Linear Regression Model to Data

- Assume we are given a training set of N pairs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$
- We can write these out using matrix notation:

$$\Phi = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

- This **design matrix** Φ is of size $N \times (D+1)$ and an entry x_{ij} is the j'th component of the training input $\boldsymbol{x}_i$
- Thus, $\hat{\boldsymbol{y}} = \Phi \boldsymbol{w}$ is the model's predicted outputs for the training inputs

Solving for Model Weights

- This looks like something we have seen in linear algebra:

$$\boldsymbol{y} = \Phi \boldsymbol{w}$$

- We know $\boldsymbol{y}$ for our training data and the entries of Φ, but we do *not* know $\boldsymbol{w}$

- So why not take $\boldsymbol{w} = \Phi^{-1}\boldsymbol{y}$?
- Three reasons:
  - Φ is not square. Its size is $N \times (D+1)$
  - The system is over constrained - (N equations for $D+1$ weights)
  - The data has noise

Measuring ‘Goodness of Fit’

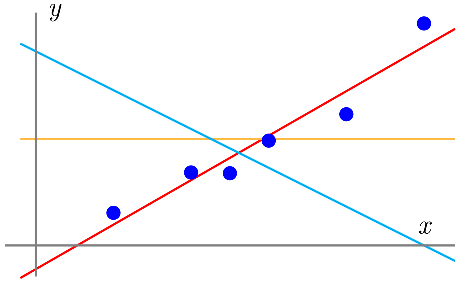
- We want a loss function that can tell us how good our fit is.
- One intuitive option is the **Sum of Squared Errors** (SSE):

$$L_{SSE}(\boldsymbol{w}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$
$$= \sum_{i=1}^{N} (y_i - \boldsymbol{w}^\intercal \phi(\boldsymbol{x}_i))^2$$

- This penalises large mistakes $y - \hat{y}$ more than small ones

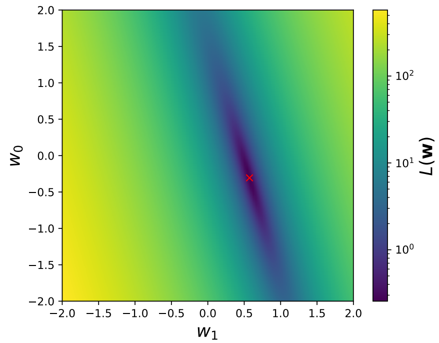
Measuring ‘Goodness of Fit’

- Different models (i.e. choices of weights $\boldsymbol{w}$) will result in different loss values
- For example:
  - For $y = -0.31 + 0.57x$, the SSE $= 0.25$
  - For $y = 1.37 + 0.00x$, the SSE $= 3.61$
  - For $y = 2.50 - 0.50x$, the SSE $= 12.63$



Error Surface

- For 1D data we can visualise the error for each set of possible weights
- Here, the minimum of this convex error surface is indicated by the values at x



Fitting the Linear Model to Data

- We can write out our loss for the training data as:

$$L_{SSE}(\boldsymbol{w}) = \sum_{i=1}^{N} (y_i - \boldsymbol{w}^\intercal \phi(\boldsymbol{x}_i))^2$$
$$= ||\boldsymbol{y} - \Phi \boldsymbol{w}||^2$$
$$= (\boldsymbol{y} - \Phi \boldsymbol{w})^\intercal (\boldsymbol{y} - \Phi \boldsymbol{w})$$

- To solve for $\boldsymbol{w}$, we take the partial derivative of $L_{SSE}(\boldsymbol{w})$ wrt $\boldsymbol{w}$ and set it to 0, i.e. $\frac{\partial L_{SSE}(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$

Deriving the Least Squares Solution - 1

- We begin by rewriting the terms of the SSE loss:

$$
\begin{aligned}
L_{SSE}(\boldsymbol{w}) &= (\boldsymbol{y} - \Phi\boldsymbol{w})^\top(\boldsymbol{y} - \Phi\boldsymbol{w}) \\
&= (\boldsymbol{y}^\top - \boldsymbol{w}^\top\Phi^\top)(\boldsymbol{y} - \Phi\boldsymbol{w}) \\
&= \boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{y}^\top\Phi\boldsymbol{w} - \boldsymbol{w}^\top\Phi^\top\boldsymbol{y} + \boldsymbol{w}^\top\Phi^\top\Phi\boldsymbol{w} \\
&= \boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{w}^\top\Phi^\top\boldsymbol{y} + \boldsymbol{w}^\top\Phi^\top\Phi\boldsymbol{w}
\end{aligned}
$$

Deriving the Least Squares Solution - 2

- Next we take the partial derivative:

$$
\frac{\partial L_{SSE}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}}\left[\boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{w}^\top\Phi^\top\boldsymbol{y} + \boldsymbol{w}^\top\Phi^\top\Phi\boldsymbol{w}\right]
$$

- We can do this one part at a time:

$$
\begin{aligned}
\frac{\partial\,(\boldsymbol{y}^\top\boldsymbol{y})}{\partial \boldsymbol{w}} &= 0 \\
\frac{\partial\,(-2\boldsymbol{w}^\top\Phi^\top\boldsymbol{y})}{\partial \boldsymbol{w}} &= -2\Phi^\top\boldsymbol{y} \\
\frac{\partial\,(\boldsymbol{w}^\top\Phi^\top\Phi\boldsymbol{w})}{\partial \boldsymbol{w}} &= 2\Phi^\top\Phi\boldsymbol{w}
\end{aligned}
$$

Deriving the Least Squares Solution - 3

- From the previous slide we obtained:

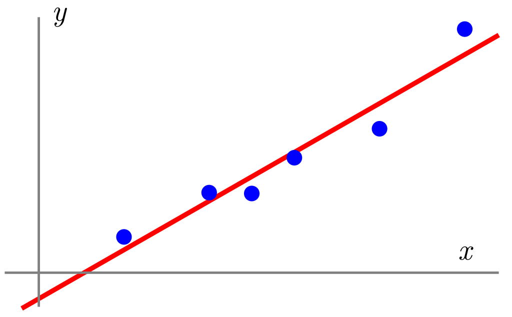
$$
\frac{\partial L_{SSE}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -2\Phi^\top\boldsymbol{y} + 2\Phi^\top\Phi\boldsymbol{w}
$$

- We set this to 0 to find the closed-form solution, i.e. $\frac{\partial L_{SSE}(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$:

$$
\begin{aligned}
0 &= -2\Phi^\top\boldsymbol{y} + 2\Phi^\top\Phi\boldsymbol{w} \\
2\Phi^\top\Phi\boldsymbol{w} &= 2\Phi^\top\boldsymbol{y} \\
\Phi^\top\Phi\boldsymbol{w} &= \Phi^\top\boldsymbol{y} \\
\boldsymbol{w} &= (\Phi^\top\Phi)^{-1}\Phi^\top\boldsymbol{y}
\end{aligned}
$$

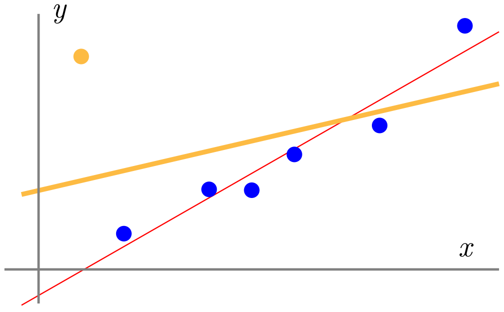
Sensitivity to Outliers

- Linear regression is sensitive to *outliers*
- Suppose $y = 0.5x + \epsilon$, where ϵ is some noise



Sensitivity to Outliers

- What happens if we add an ‘outlier‘ at x=0.5 and y=2.5?
- Here, we are simply adding one new training example

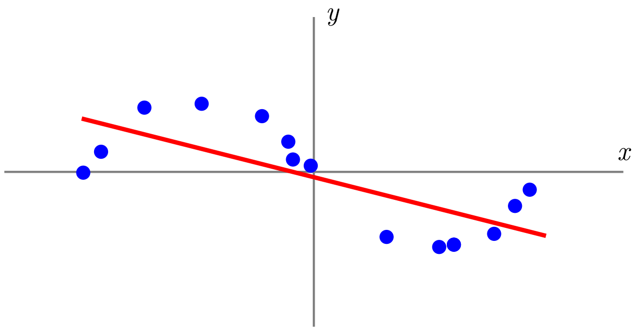


Diagnositics

- Graphical diagnostics can be useful for checking:
  - Is the relationship obviously nonlinear? Look for structure in errors?
  - Are there obvious outliers?
- The goal is not to find all problems - this is difficult. The goal is to find obvious ones

Nonlinear Regression

Nonlinear Regression



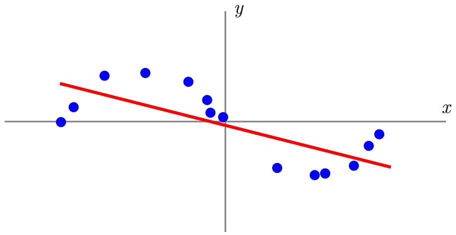
What if there is a nonlinear relationship between your features and the target you wish to predict?

Nonlinear Regression - Transforming Inputs

- Up until now we have set $\phi(x) = [1, x]^\intercal$
- However, we can transform our inputs in different ways

- One example is **polynomial regression**, $\phi(x) = [1, x, x^2, ..., x^M]^\intercal$
- Here, the dimensionality of our weights $\boldsymbol{w}$ will be the same as $\phi(x)$

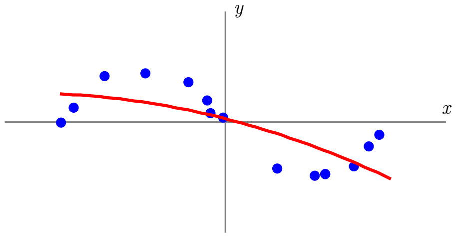
Polynomial Regression



$M = 1$

$\phi(x) = [1, x]^\intercal$

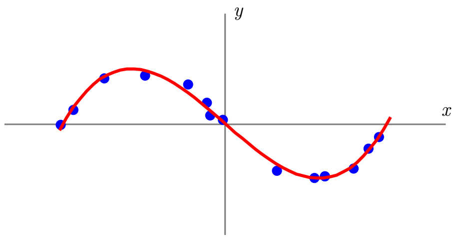
Equivalent to simple linear regression.



$M = 2$

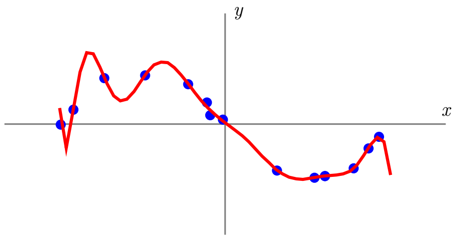
$\phi(x) = [1, x, x^2]^\intercal$

Polynomial Regression



$M = 3$

$\phi(x) = [1, x, x^2, x^3]^\intercal$



$M = 12$

$\phi(x) = [1, x, x^2, ..., x^{12}]^\intercal$

We have **overfit** the training data.

Basis Expansion

- We can easily transform the original features $\boldsymbol{x}$ **non-linearly** into $\phi(\boldsymbol{x})$ and perform linear regression on the transformed features
- For example, we can use a set of M basis functions $\phi(\boldsymbol{x}) = [1, \psi_1(\boldsymbol{x}), \psi_2(\boldsymbol{x}), ..., \psi_M(\boldsymbol{x})]^\intercal$
- Each of these basis functions takes a vector as input and outputs a scalar value

Basis Expansion

- Now our **design matrix** is of size $N \times (M+1)$, where we have M basis functions

$$\Phi = \begin{bmatrix} 1 & \psi_1(\boldsymbol{x}_1) & \psi_2(\boldsymbol{x}_1) & \dots & \psi_M(\boldsymbol{x}_1) \\ 1 & \psi_1(\boldsymbol{x}_2) & \psi_2(\boldsymbol{x}_2) & \dots & \psi_M(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \psi_1(\boldsymbol{x}_N) & \psi_2(\boldsymbol{x}_N) & \dots & \psi_M(\boldsymbol{x}_N) \end{bmatrix}$$

- Again, we let $\boldsymbol{y} = [y_1, ..., y_N]^\intercal$
- We can then minimise $L_{SSE}(\boldsymbol{w}) = ||\boldsymbol{y} - \Phi\boldsymbol{w}||^2$ using the same analytical solution as before

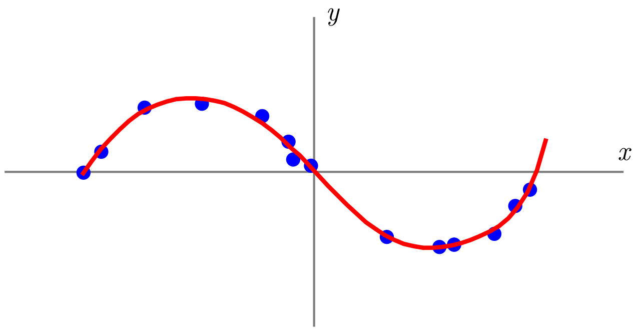
Radial Basis Functions

- One popular choice of basis functions are **Radial Basis Functions** (RBFs)
- Each RBF $\psi()_m$ has two parameters: a centre $\boldsymbol{c}_m$ and a width σ_m^2, and outputs a single scalar

$$\psi_m(\boldsymbol{x}) = \exp\left(-0.5\frac{||\boldsymbol{x} - \boldsymbol{c}_m||^2}{\sigma_m^2}\right)$$

- One needs to position each basis function at a specified centre location with a given width
- There are many ways to do this but choosing a subset of the datapoints as centres is one approach that is quite effective

Radial Basis Functions - Example



- In this example, we have a RBF centred on each training point and we use the same value of σ^2 for each
- The quality of the fit can strongly depend on the choice of RBF parameters

Dealing with Multiple Outputs

- Suppose there are K different targets for each input $\boldsymbol{x}$, i.e. $\boldsymbol{y} \in \mathbb{R}^K$
- We introduce a different $\boldsymbol{w}_k$ for each target dimension, and do regression separately for each one

Summary

- Linear regression is often useful out of the box
- It is more useful than it would be seem because linear means linear *in the weights*.
  You can do a nonlinear transform of the data first, e.g., polynomial, RBF, etc.
- The solution for the model weights is computationally efficient to obtain (pseudo-inverse)
- Danger of overfitting, especially with many features or basis functions