Linear Regression - creating a linear model which is a line of best fit through a data set

https://towardsdatascience.com/linear-regression-derivation-d362ea3884c2?

- *Given* ***D=((x$_1$, y$_1$), ... , (x$_n$, y$_n$)***, ***x$_i$ ∈ R^d, y$_i$∈ R****, we* *choose a function* ***f : R^d -> R*** *to predict* ***y*** *for* a new **x**
- **Linear Model: y = mx + c** (**+ e** *error*)
- We do this by minimising the **sum of squared errors**
  - The distance between the model at x and a data point, squared to remove sign, summed for all points
- **Input:** (x_i, y_i)
- **Model Definition:** $\hat{Y}_i = a + Bx_i$
- **Costs (sum of squared errors):** $$S = \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$
- **Alternative notation of costs:** $$S = \sum_{i=1}^{n}(Y_i - a - Bx_i)^2$$
  - We plug in model definition into costs function

Implementing Linear Regression

- We want to minimise the sum of squared errors
- We do this by finding the derivative of **S (sum of squared errors)** w.r.t **a** and **B**
  - These are **partial derivatives**, which means we are finding ∂**S**/∂**a** and ∂**S**/∂**b**
  - We want these partial derivatives **in terms of x and Y only**, since those are the only values that we know
    - We get the partial derivatives of **a** and **B** using the **chain rule**
    - We then equal the derivatives to 0 and rearrange for **a** and **B**
      - This gives us the formula for **a** and **B** at the place where sum of squared errors is **minimal**
    - When solving for **a**, remember the **sum of E^{n}a = na** $\sum_{i=1}^{n} a = na$
    - We then plug in the formula for **a** into the formula for **B** (or vice versa)
    - When solving for **B**, we isolate it by splitting the sum into two sums, then we can get **B** as a fraction with a sum numerator and a sum denominator
- **Solving for a**
  - $0 = \sum_{i=1}^{n}(Y_i - a - Bx_i)$ | $0 = \sum_{i=1}^{n} Y_i - \sum_{i=1}^{n} a - B\sum_{i=1}^{n} x_i$ | $\sum_{i=1}^{n} a = na$
  - $a = \dfrac{\sum_{i=1}^{n} Y_i - B\sum_{i=1}^{n} x_i}{n}$ | $a = \bar{Y} - B\bar{x}$
- **Solving for B**
  - $0 = \sum_{i=1}^{n}(x_iY_i - ax_i - Bx_i^2)$ | $0 = \sum_{i=1}^{n}(x_iY_i - (\bar{Y} - B\bar{x})x_i - Bx_i^2)$

- $$B = \frac{\sum_{i=1}^{n}(x_i Y_i - \bar{Y}x_i)}{\sum_{i=1}^{n}(x_i^2 - \bar{x}x_i)}$$

Multiple Linear Regression

- Add vectors mx -> w^Tx (c gets absorbed with a $weight_0 = 1$

Online Implementation of Linear Regression
https://www.cs.princeton.edu/courses/archive/spring18/cos511/scribe_notes/0411.pdf

Linear Regression - Nonlinearity via Basis Functions

Linear Basis Function Models

- Linear Regression is **linear in W, not necessarily in X**, that's the key point
- We have inputs **X**, which are vectors consisting of **n inputs ($x_1, x_2, \ldots, x_n$)** to predict a continuous **y**
  - We could use **linear regression**
  - This is good if the relationship between each x_i and **Y** is **linear**
  - If it isn't **linear**, we need to use **linear basis function models**
- **Linear basis function models** **-** Assume that target **y** is a **linear combination** of a **set** of **n** **basis functions Φ**
  - $Y'_i = w_0 + w_1\Phi_1(X) + w_2\Phi_2(X) + \ldots + w_n\Phi_n(X)$
  - This means that we replace each input x_i with a function of the input $\Phi_i(X)$ NOTICE WHOLE X VECTOR
  - When **Φ** is a set of just **identity functions**, we just get **linear regression**
    - **Identity Function -** a function that returns the same value as its argument
  - Choose the functions **Φ** in a way that best models the **non-linearity** of the relationship between **X** and **Y**
    - *Also has to be computationally efficient*

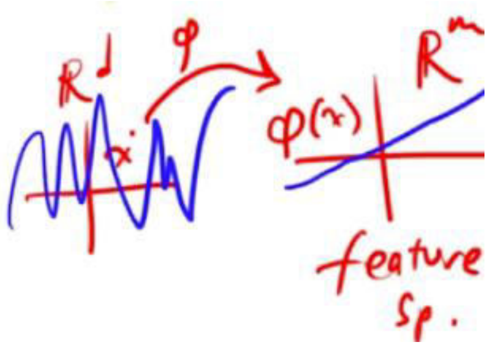
Basis Functions **-** We use basis functions to represent nonlinear problems in terms of something linear (i.e. with LR)

- If we have an input $X \in R^d$, $y \in R$ that we want to model with some function...
  - The **simplest model**,
    - A linear function, $f(X) = W^TX = \Sigma w_i x_i$ (dot product), $w \in R^d$
      - This is the simplest case, which is when we use an **identity function** basis function
  - Alternatively, in the more **general case**, in our linear regression model, we can choose…
    - A linear function, $f(X) = W^T\Phi(X) = \Sigma w_i \varphi_i(X)$, $W \in R^d$, $\Phi : R^d \rightarrow R^m$
      - **Φ - a vector of basis functions** that takes an input **X** of **d** dims and gives vector of **m** dims
      - **Φ** can represent functions that are nonlinear
      - It is still linear in **w**, that is the **linear** part in linear regression
  - There is also the **polynomial case**, which is when our basis functions are polynomial
    - *Often used in **physics models**, although they are mostly used for high-dimensional spaces*
    - A linear function, e.g. $f(X) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_1x_2$
      - Each basis function is a polynomial in the coordinates of **X**
      - This means that $\varphi_1(X) = x_1$, $\varphi_2(X) = x_2$, $\varphi_3(X) = x_1^2$, $\varphi_4(X) = x_1x_2$
        - We write this as $\Phi(X) = (1, x_1, x_2, x_1^2, x_1x_2)$
  - In the **radial case**,
    - Often used in **high-dimensional spaces**
    - Each radial basis function is like a little Gaussian, and taking a combination of them can give you surface which gives you curves at some range of values, ***but flattens out as x goes to inf and -inf***
  - In the **Fourier case**,
    - Often used in **signal processing**, but also useful in physics models
    - Each Fourier basis function is a **sine** or **cosine wave**, and when you take a combination of them, you get **periodic functions**
  - In the **wavelets case**,
    - Often used in **image processing**

        - Kind of like a combination of radial basis functions and Fourier basis functions, they are kind of like a single heartbeat, and they can be <u>linearly</u> combined to create very complicated functions

Definition and Motivation

- **Feature Space -** The feature space is where our **Φ(X)** lives. What this means is that we use **Φ** to map our original data **X** to the feature space of **Φ(X)**, where the **x**-axis becomes the **Φ(x)**-axis, which becomes linear with regards to our weights **W**, even if **Φ(X)** is entirely nonlinear
- We sometimes refer to **z** = **Φ(X)**
  - Therefore, **$f(X) = W^T\Phi(X) = W^TZ$**



Discriminative Models vs Generative Models

- **Discriminative**
  - Simpler
  - Modelling less things, making fewer assumptions
  - Models P(y|X)
    - We only model the conditional probability of ys given the Xs, vs the whole joint distribution

Let's say we choose to make a Discriminative Model

- **Assume** some family/set of conditional distributions parameterized by **θ**, **$P_\theta(y|X)$**, **θ** ∊ **Θ**
  - We estimate **θ** using our data **$D = \{(x_n, y_n)_N\}$**
  - We will assume that there are some parameters **Θ** for which our data comes from, then we figure out what **Θ** it actually came from
- **Steps**
  - Assume some family/set of conditional distributions
    - E.g. assume **Gaussian** distributions (generally pretty good approximation)
      - **$P_\theta(y|X) = N(y|\mu(X), \sigma^2(X))$**, where **μ** and **$\sigma^2$** are functions of **X**
        - In this example, **θ** is a tuple of functions, **$\theta = (\mu, \sigma^2)$**
          - Tuple of functions that are used to get the parameters for **$P_\theta(y|X)$** pdf
        - *Now we need to choose what values μ and σ^2 we use*

Definition: (Gaussian Linear Regression Model)

- GLR is a probabilistic model for our data
- It models our data **D** by assuming the conditional distributions **$P_\theta(y|X) = N(y|W^TX, \sigma^2)$ where $\theta = (W, \sigma^2)$**
  - **μ -** We just take our mean to be the value predicted by our model, so **$\mu = W^TX$**
  - **σ^2 -** Although a function of **X**, we just take **σ^2** to be some constant positive number **$\sigma^2 > 0$**
- We can say the above as **$y = W^TX + \varepsilon$**, where **$\varepsilon \sim N(0, \sigma^2)$**, and **ε** is unknown noise that we choose to model N pdf
  - Doesn't have to be Gaussian, could be whatever distribution as long as it's dependence on **X** is linear to **W**
  -

Learning Basis Function Parameters with Jacobian Matrices

We now return to the problem of fitting the parameters $\boldsymbol{\theta}$ to data via the optimization problem in Eqn 5. This is a direct use of the chain rule above with the Jacobian of $\Phi_{\boldsymbol{\theta}}(\boldsymbol{x})$ with respect to $\boldsymbol{\theta}$:

$$\begin{aligned}
\nabla_{\theta}\left\{\frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n)^2\right\} &= \mathcal{J}_{\theta}\left\{\frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n)^2\right\}^{\mathsf{T}} && (9)\\
&= \frac{1}{2}\sum_{n=1}^{N}\mathcal{J}_{\theta}\left\{(\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n)^2\right\}^{\mathsf{T}} && (10)\\
&= \frac{1}{2}\sum_{n=1}^{N}\left(\mathcal{J}_{z}\{\boldsymbol{w}^{\mathsf{T}}\boldsymbol{z}+b-y_n)^2\}\mathcal{J}_{\theta}\{\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\}\right)^{\mathsf{T}} && (11)\\
&= \sum_{n=1}^{N}\left((\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n)\boldsymbol{w}^{\mathsf{T}}\mathcal{J}_{\theta}\{\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\}\right)^{\mathsf{T}} && (12)\\
&= \sum_{n=1}^{N}\mathcal{J}_{\theta}\{\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\}^{\mathsf{T}}\boldsymbol{w}(\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n)\,. && (13)
\end{aligned}$$

To make it clear that we're looking at a composition with $\Phi_{\boldsymbol{\theta}}(\boldsymbol{x})$, in Eqn 11 above I have used a variable $\boldsymbol{z}$ as the thing we are differentiating with respect to. Just as a sanity check we can see that this has the right dimensional structure. The Jacobian of $\Phi_{\boldsymbol{\theta}}(\boldsymbol{x})$ *with respect to* $\boldsymbol{\theta}$ (not with respect to $\boldsymbol{x}$!) is a $J \times K$ matrix, and $\boldsymbol{w}$ has the dimension of the output of $\Phi_{\boldsymbol{\theta}}(\boldsymbol{x})$, and so is of length J. Therefore the product of the transposed Jacobian and $\boldsymbol{w}$ gives a vector of length K, which is what we want for the gradient of a scalar function with respect to $\boldsymbol{\theta}$.

Having figured out this gradient, we can now write our overall update rules. Here I'm writing them for batch (full-data) gradient descent:

$$\begin{aligned}
\boldsymbol{w}^{(t+1)} &\leftarrow \boldsymbol{w}^{(t)} - \alpha\sum_{n=1}^{N}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)(\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n) && (14)\\
b^{(t+1)} &\leftarrow b^{(t)} - \alpha\sum_{n=1}^{N}(\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n) && (15)\\
\boldsymbol{\theta}^{(t+1)} &\leftarrow \boldsymbol{\theta}^{(t)} - \alpha\sum_{n=1}^{N}\mathcal{J}_{\theta}\{\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\}^{\mathsf{T}}\boldsymbol{w}(\boldsymbol{w}^{\mathsf{T}}\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_n)+b-y_n)\,. && (16)
\end{aligned}$$

In practice one might find that different learning rates work better for the different sets of parameters. Note that unlike everything we have looked at so far, this objective is not likely to be convex. Gradient descent may get stuck in various undesirable locations and not find a global minimum.

Bayes (Linear) Regression

From COM4509 Lecture 6

Once you have two observations, any further observations lead to an **overdetermined system**

- **Overdetermined System -** Always inconsistent (it has no solution)
    - **Solution:** we add noise to our model which represents corruption in our model
    - Noise can be modelled over a distribution

Noise Model - gives us the mismatch between the way we are trying to model in the data and what we are actually seeing in the data

- Gaussian error models can be justified using the **central limit theorem** which states that when independent random variables are added, their normalised sum tends toward a normal distribution

Underdetermined System - There are fewer equations than there are unknown variables

- **Overdetermined System -** There are more equations than there are unknowns
- We can turn an overdetermined system into an underdetermined system by adding extra variables
    - For example, by adding noise, we are adding an extra variable *e* to every instance / measurement
    - We can then get rid of the *e* by modelling it with a Gaussian distribution (or some other distribution)

$$\text{point 1: } x = 1, y = 3$$
$$3 = m + c + \epsilon_1$$
$$\text{point 2: } x = 3, y = 1$$
$$1 = 3m + c + \epsilon_2$$
$$\text{point 3: } x = 2, y = 2.5$$
$$2.5 = 2m + c + \epsilon_3$$

- We can deal with an underdetermined system by solving the unknown variables by taking one of the unknowns and randomly sampling possible values according to some distribution.
    - For example, if we are solving y = mx + c with only one point, we have **m** and **c** as the unknowns
    - We can solve **m** by picking a random value for **c**
    - We could do this by taking random values of **c** according to a distribution, for example, **c ~ *N*(0, 4)**
- Therefore, we can deal with underdetermined systems by making an assumption that the unknown parameters are distributed
    - This is also known as **Bayesian**

Two types of uncertainty

- **Aleatoric Uncertainty (Statistical Uncertainty) -** Represents unknown variables that differ each time we run the same experiment
    - The error is inherently uncertain and cannot be reduced
- **Epistemic Uncertainty (Systematic Uncertainty) -** Represents unknown variables that vary due to things one could in principle know but doesn't in practice
    - We can reduce the uncertainty by adding more data
    - E.g. inaccurate measurements, because the model neglects effects or because some data has been deliberately hidden
        - I.e. We often assume that acceleration due to gravity on earth is 9.8ms^{-2}
        - In an experiment, we could use this value of 9.8ms^{-2} which neglects air resistance and make it more accurate by measuring the air resistance and incorporating it into the reading of the measurements

Bayesian Inference / Update

From COM4509 Lecture 6

- It is the process of **updating** our **prior beliefs** with **data** to form **posterior beliefs**
- Requires a **prior** on the parameters
- **Prior Distribution -** Represents your belief **before** you see the data of the likely value of the parameters
  - This means that we have to generate a distribution before the measurement
  - For linear regression, we can consider a Gaussian prior on **c**, the y-intercept
- **Likelihood Function -** The likelihood function is used so that when we observe some data, we can calculate what was the likelihood of that data being measured given any combinations of model parameters
- **Posterior Distribution -** This is the distribution after accounting for the observed data
  - We can calculate the posterior using the **Bayes' Rule**
  - We have to update the posterior distribution using the new data and the likelihood function.

$$p(c|y) = \frac{p(y|c)p(c)}{p(y)}$$

- You make a guess about something (belief), then you make an observation, then you update the belief
- **Bayes' Filtering (Recursive Bayesian Estimation) -** An approach for estimating an unknown PDF recursively over time using incoming measurements and a mathematical process model