1 Subjects

- Linear regression
- Perceptron
- Logistic regression

2 Notes

2.1 Linear Regression

https://www.youtube.com/watch?v=rVviNyIR-fI&index=52&list=PLD0F06AA0D2E8FFBA

- Not just about lines and planes
- Linear regression is more than just fitting a line to some data
- You can also fit curves, periodic functions etc
- "It's not just lines"

Setup: Given $D = ((x_i, y_i), \ldots, (x_n, y_n)), x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$.
Goal: Choose $f : \mathbb{R}^d \to R$ to predict the y for a new x.

2.1.1 Basis fns

With a basis function, you can represent things that are non-linear in terms of something that is linear.

$$z \in \mathbb{R}^d,\ y \in \mathbb{R},\ w \in \mathbb{R}^d$$

This is the simplest case of linear regression:

$$f(z) = w^T z = \sum_{i=1}^{d} w_i z_i$$

A more general case is the following:

$$f(z) = w^T \phi(z) = \sum_{i=1}^{m} w_i \phi_i(z)$$
$$\phi : \mathbb{R}^d \to \mathbb{R}^m, w \in \mathbb{R}^m$$
$$\phi(z) = (\phi_1(z), \ldots, \phi_n(z))$$

Note here, ϕ is non-linear but it is linear in w, and that is where linear is significant in linear regression. The ϕ_i's are what we call the basis functions.

Here we can express the simple case of linear regression by just setting $\phi(z) = z$, which is the identity function. Another ϕ's one might use is:
Polynomial: $x \in \mathbb{R}^2, f(z) = w_1 + w_2 z_1 + w_3 z_2 + w_4 z_1^2 + w_5 z_2^2 + w_6 z_1 z_2$
Here, each of these ϕ_i's are: $\phi(z) = (1, z_1, z_2, z_1^2, z_2^2, z_1 z_2)$ which will give us polynomials.

The key point here is, **they are linear in w not necessarily in x.** I.e. linear in the feature-space but not necessarily in the data space.

2.1.2 Learning algorithm

Our error measure for linear regression is:

$$err(h(x), y) = (h(x) - y)^2$$

So our learning algorithm is to find the w that minimizes this:

$$\arg\min_w E_{in}(w) = \frac{1}{n} \sum_{(x,y) \in D} (w^T x - y)^2$$

We can simply find w by solving:

$$w = (X^T X)^{-}1 X^T y$$

In $\mathcal{O}\left(nd^2\right)$ time. However, linear regression is sensitive to data. Which is why the Perceptron algorithm is interesting.

2.2 Perceptron

`https://www.youtube.com/watch?v=5g0TPrxKK6o` Perceptron training, find weights that map inputs to outputs. To primary ways:

- Perceptron rule (thresholded values)
- Gradient descent (unthresholded values)

We have our training set X and label set y, we want to set the weights such that we capture the data-set. We want to do that by modifying weights over time:

$$\begin{aligned} w_i &= w_i + \Delta w_i \\ \Delta w_i &= \eta(y - \hat{y}) x_i \\ \hat{y} &= (\sum_{i=1}^{n} w_i x_i \geq \theta) \end{aligned}$$

If, we add the bias to x as $x_0 = 1$, we can simplify the math a bit, getting:

$$\begin{aligned} w_i &= w_i + \Delta w_i \\ \Delta w_i &= \eta(y - \hat{y}) x_i \\ \hat{y} &= (\sum_{i=0}^{n} w_i x_i \geq 0) \end{aligned}$$

y	$\hat{y}$	$y - \hat{y}$
0	0	0
0	1	-1
1	0	1
1	1	0

Where: x is the input, y is the target, $\hat{y}$ is the output and η is the learning rate. The first line here, states that w_i is w_i plus the amount we change the weight by, which is sort of a tautology. The weight change is defined as follows:
We take the target and compare it with the output $(y - \hat{y})$: So, if we get the same value as the target, don't change otherwise move in either one or the other direction. If the target is 0 and our output is 1, then we want to decrease the weights if vice versa then increase the weights. We then multiply by x_i such that we move it by a distance thaat makes sense in relation to the data-set. Now in order to prevent over-shooting, we multiply it by the learning rate in order to take small steps.

We then run the loop, while there is some error so we stop updating when we don't update the weights anymore.

2.2.1 Linearly separable

If our data is linearly separable, i.e. there is a line that completely separates the two data-sets, then the perceptron algorithm will find it in a finite number of iterations. However, if the data is not linearly separable (which is often the case, especially because of noise) then it will never stop.

2.3 Logistic regression

`https://www.youtube.com/watch?v=-Z2a_mzl9LM&list=PLD0F06AA0D2E8FFBA&index=109`
Example: Suppose you are an actuary, and want to compute the probability of someone dying, we would have the model:

$$\text{model: } P(death|x)$$
$$\text{Vars: } x_1 = \text{age}, x_2 = \text{sex}, x_3 = \text{cholesterol level}$$

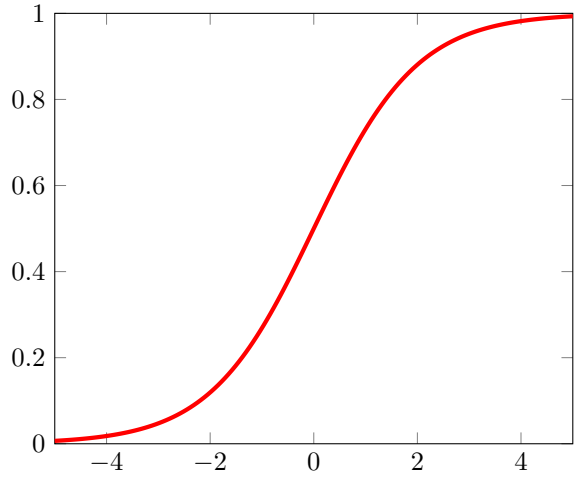
We want to find a simple model, with as few parameters as possible. So we try a linear combination:

$$w_0 + w_1 x_1 + w_2 + x_2 + w_3 x_3$$

Then as age and cholesterol level gets larger, then the value gets larger. The sex, if represented as $m = 0, f = 1$ then this gives us some offset in case being male or female matters. We can express that as:

$$w^T x,\ x = (1, x_1, x_2, x_3)$$

But this isn't a probability, just some plane. We can obtain a probability by putting the value through a sigmoid function, $\sigma(a)$. Sigmoid is a generic term for an S-shaped curve. example:



We can then simple set $P(death|x) = \sigma(w^T x)$. This is the formula for the model for binary classification using logistic regression. The logistic function is defined as:

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

2.3.1 Formally

Given: $D = ((x_1, y_1), \ldots, (x_n, y_n)), x_i \in \mathbb{R}^d, y_i \in \{0, 1\}$
Model:

$$\begin{aligned} y_i \text{ Bernoulli}(\sigma(w^T x)) independt \\ \sigma(a) = \frac{1}{1 + e^{-}a} \text{ “logistic fn”} \\ w = param. \end{aligned}$$

Pros	Cons
Interpretable, the coefficients of the variables (w) is readable, in a sense they mean something. E.g. if w_1 is positive, then a higher age increases the probability of death et vice versa.	*Performance* not necessarily as good as best perf. methods
Small number of parameters, as a result the parameters are easier to estimate. ($d+1$ variables)	
Computationally efficient, to estimate w	
Natural extension to multiclass	
Forms a foundation for more complex methods like neural networks	

2.3.2 Finding the parameters (w)

To find the parameters, which in this case is simply w, we will need to do Maximum Likelihood Estimation. The MLE here is:

$$w_{MLE} \in \arg\max_w p(D|w)$$

We can write $p(D|w)$ as:

$$p(D|w) = \prod_{i=1}^{n} p(y_i|x_i, w)$$

Let's introduce the notation for the sigmoid of x_i as: $\alpha_i = \sigma(w^T x_i)$
Since y_i is a Bernoulli variable, i.e. it has two outcomes one with probability p the other with $1-p$, we can model it as the probability of α_i if y_i is 1: $\alpha_i^{y_i}$ multiplied by the probability that it is 0 if y_i is 0: $(1-\alpha_i)^{1-y_i}$:

$$p(D|w) = \prod_{i=1}^{n} p(y_i|x_i, w) = \prod_{i=1}^{n} \alpha_i^{y_i}(1-\alpha_i)^{1-y_i}$$

We want to maximize this term, it might be tempting try and solve for w by setting it equal to 0, but it turns out that because the logistic function is non-linear, we wont be able to solve analytically for w. Newtons method will allow us to estimate w. So we will take the gradient to the Negative Log Likelihood, as it is a little bit nicer to minimize the NLL than maximizing the MLE.

$$\mathcal{L}(D|w) = -\log p(D|w) = -\sum_{i=1}^{n} y_i \log \alpha_i + (1-y_i)\log(1-\alpha_i)$$

We take the derivative of α_i for some specific feature j:

$$\begin{aligned}
\frac{\partial}{\partial w_{ij}} \log \alpha_i &= \frac{\partial}{\partial w_{ij}} \log \sigma(w^T x_i) \\
&= \frac{\partial}{\partial w_{ij}} \log \frac{1}{1 + e^{-w^T x_i}} \\
&= \frac{\partial}{\partial w_{ij}} - \log(1 + e^{-w^T x_i}) \\
&= \frac{x_{ij} e^{-w^T x_i}}{1 + e^{-w^T x_i}} \\
&= x_{ij}(1 - \alpha_i)
\end{aligned}$$

And then the derivative of $(1 - \alpha_i)$ is:

$$\begin{aligned}
\log(1 - \alpha_i) &= -w^T x_i - \log(1 + e^{-wTx_i}) \\
\frac{\partial}{\partial w_{ij}} \log(1 - \alpha_i) &= -x_{ij} + x_{ij}(1 - \alpha_i) \\
&= -\alpha_i x_{ij}
\end{aligned}$$

Now the derivative of the $\mathcal{L}(D|w)$ can be found as:

$$\begin{aligned}
\frac{\partial}{\partial w_j} \mathcal{L}(D|w) &= -\sum_{i=1}^{n} y_i x_{ij}(1 - \alpha_i) + (1 - y_i) x_{ij} \alpha_i \\
&= -\sum_{i=1}^{n} y_i x_{ij} - y_i x_{ij} \alpha_i - x_{ij} \alpha_i + y_i x_{ij} \alpha_i \\
&= -\sum_{i=1}^{n} (\alpha_i - y_i) x_{ij}
\end{aligned}$$

If we say that the matrix X, is our training data set:

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nd} \end{pmatrix}$$

Then the derivative we found before, just needs to be combined with each column of X:

$$(\alpha - y)^T X$$

Which, transposed, then gives us the gradient:

$$\nabla_w \mathcal{L}(D|w) = X^T(\alpha - y) = -X^T(y - \alpha)$$

We can then do gradient descent by defining our convex function F as:

$$F(w) = \frac{1}{n} \mathcal{L}(D|w) \tag{1}$$

$$w = w - \eta \nabla F(w)$$

over a number of iterations. I.e. move w by the gradient multiplied by the learning rate η.

Doing Stochastic Gradient Descent can speed things up, SGD works similar to gradient descent, but instead of looking at all points, that is the full matrix X, we just look at a specific row x_i and compute the gradient for that row:

$$\nabla F_i(w) = \nabla \mathcal{L}(D|w) = -x_i^T(y_i - \alpha_i)$$