DATA SCIENCE AND MACHINE LEARNING

Mauricio Labadie, PhD
Quantitative Trader
Credit Suisse
London, UK

Lectures in Quantitative Finance

Facultad de Ciencias - UNAM

Mexico City, 27 February – 1 March 2018

Disclaimer

Everything I say during these lectures, in written and/or orally:

- ✓ Is my own and personal opinion
- ✓ Does not represent my employer's point of view
- ✓ Does not commit me or my employer to anything
- ✓ It is meant to be solely for educational purposes
- ✓ Does not constitute any kind of investment advice

www.svsamiti.org

A bit about myself

UPMC
SORBONNE UNIVERSITÉS

DAUPHINE
UNIVERSITÉ PARIS

Table of Contents

1. Introduction
   - What is Machine Learning?
   - Training, Cross-validation and Test sets

2. Linear regression
   - Definition of the cost function
   - Closed-form solution
   - Overfitting, bias, variance, regularisation

3. Logistic regression
   - Example Classification of trading flow based on "good" or "bad" returns
   - Single-class vs Multi-class logistic regression
   - Gradient Descent

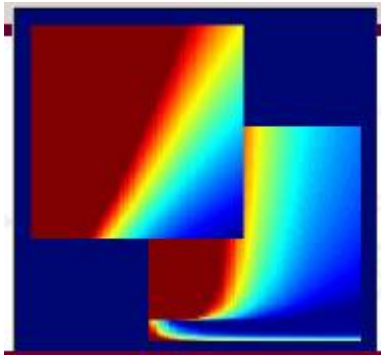
4. Neural Networks
   - Forward and backward propagation
   - Learning algorithm and caveats
   - Research example: prediction of prices with high frequency data on limit order books

1. Introduction

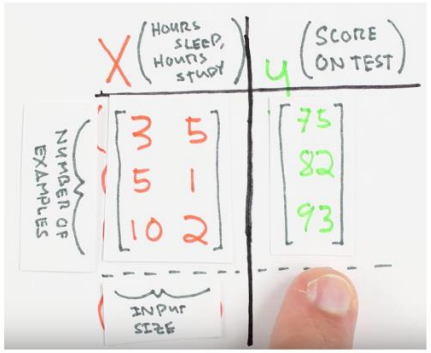
References on Machine Learning

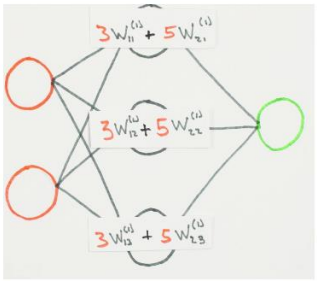
- Stanford
  - Coursera MOOC
  - Machine Learning
  - Andrew Ng

blog.netapp.com



- CalTech
  - Online lectures
  - Machine Learning
  - Yaser Abu-Mostafa
  - Book: *"Learning from Data"*

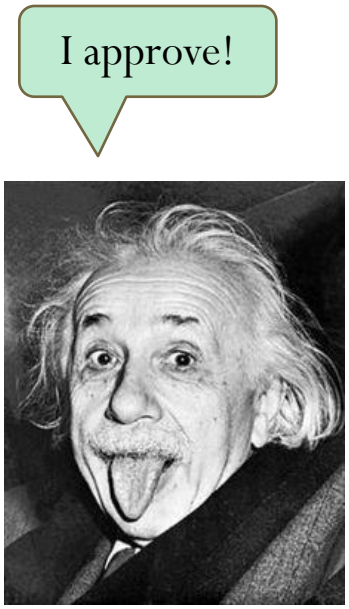




- YouTube videos
  - Neural Networks Demystified
  - Welch Labs

Am I qualified to give these lectures?

- Of course
- Because I have this!



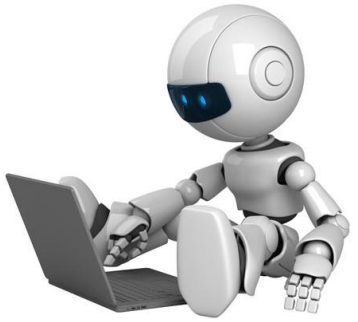
wikipedia.org

#einsteinapproves

- OK Seriously, this MOOC is very good and I highly recommend you to take it
  - It is free… unless you want to have this diploma in LinkedIn :P
  - I have no commission if you sign up :P

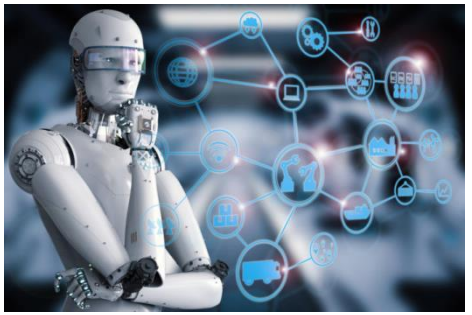
Definitions of Machine Learning

- The term "Machine Learning" (ML) was coined in 1959 by Arthur Samuel
  - He defined ML as a **field of study**
  - that gives computers the ability to learn
  - without being explicitly programmed

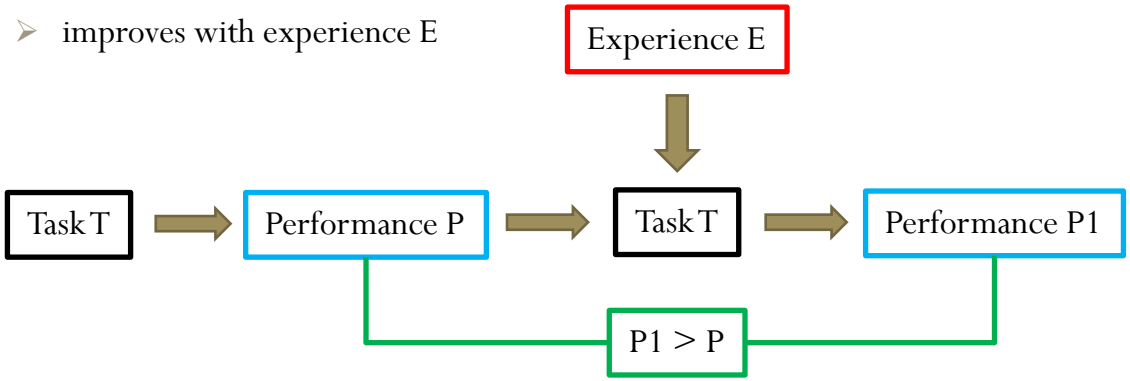


cdn-images-1.medium.com

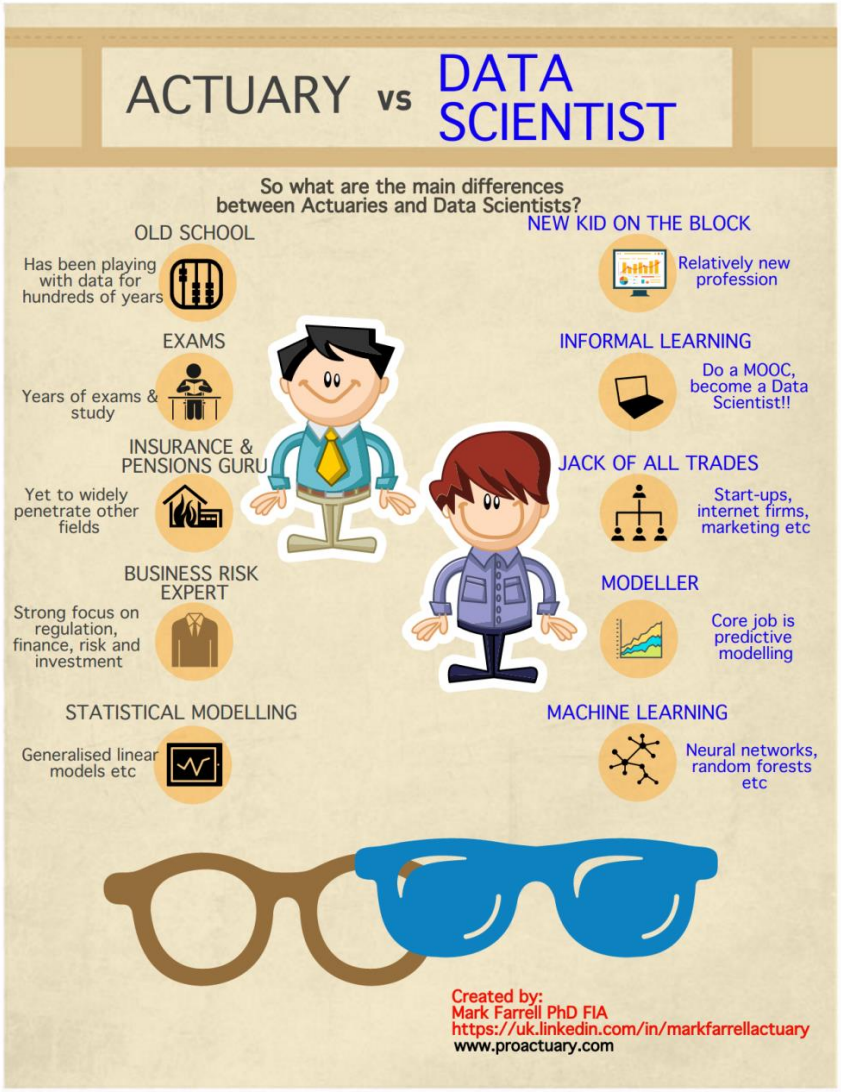
- There is another ML definition from Tom Mitchell in 1998
  - He defined ML as a **well-posed learning problem**:
  - A computer program is said to learn from experience E
  - with respect to some task T
  - and some performance measure P
  - if its performance on T
  - as measured by P
  - improves with experience E



images.idgesg.net



Data Scientist = do a MOOC?



- New kid on the block?
  - Mathematical tools from at least 1700s (Newton, Lagrange, Legendre, Cauchy, Bayes)
- Informal Learning?
  - Requires at least undergraduate level in Probability, Statistics, Linear Algebra, Multivariate Calculus, Numerical Methods and Programming
- Jack of all trades and modeller - OK
  - Mathematics has always been the language of Science and Economics
  - The new "universal language" is now data analysis: crunching numbers and separating data signals from noise
- Machine Learning?
  - ML is useful, but it is NOT the only way we deal with data
  - **Actuaries do Machine Learning too!**
    - ✓ **Linear regression is Machine Learning!**

Professionals vs Amateurs

"Everybody can fit, but the ability to deal with overfitting is what separates professionals from amateurs in Machine Learning."

Yaser Abu-Mostafa, CalTech

www.tucsondjservice.com

- The biggest minds in Machine Learning and AI are all PhDs:
  - Andrew Ng – PhD @ Berkeley
  - Yaser Abu-Mostafa – PhD @ CalTech
  - Yann LeCun – PhD @ UPMC

- The biggest companies in AI are led by PhDs:
  - Deep Blue (beat Chess champion Kasparov)
    - Murray Campbell – PHD @ Carnegie Mellon
  - Deep Mind (beat Go champion)
    - Demis Hassabis – PhD @ University College London
  - Open AI (beat the DOTA 2 champion 1v1)
    - Ilya Sutskever – PhD @ University of Toronto

pbs.twimg.com

Data Science and the Scientific Method

- Data gathering
  - Before Big Data this was very complicated
  - Now Big Data makes this much easier (but not trivial)
- Data cleaning and labelling
  - This requires deep insight and knowledge of the problem
  - For example, removing outliers and re-scaling variables
- Choice of the model
  - Inputs: parameters, variables, etc
  - Outputs: performance/error measure
- Separating the data in subsamples
  - Training set
  - Cross Validation set (optional)
  - Test set

images.carsondellosa.com

Training, Cross-validation and Test

- Training set
  - In-sample data
  - We can do whatever we want in this set to improve the model
  - This is where we calibrate the parameters
  - In other words, where the "machine learns"

- Cross-Validation set
  - "In-sample bis" or "pre-out-of-sample"
  - It is used to compare different models
  - E.g. choosing the degree of a polynomial linear regression
  - We treat this set as a pre-test set

- Test set
  - Out-of-Sample
  - This is where we evaluate the chosen model
  - It is generally used to compute the accuracy of the model

- **Rule of thumb for a data set**
  - 70% training, 30% test
  - 60% training, 20% cross-validation, 20% test

Google images

Overfitting: definition

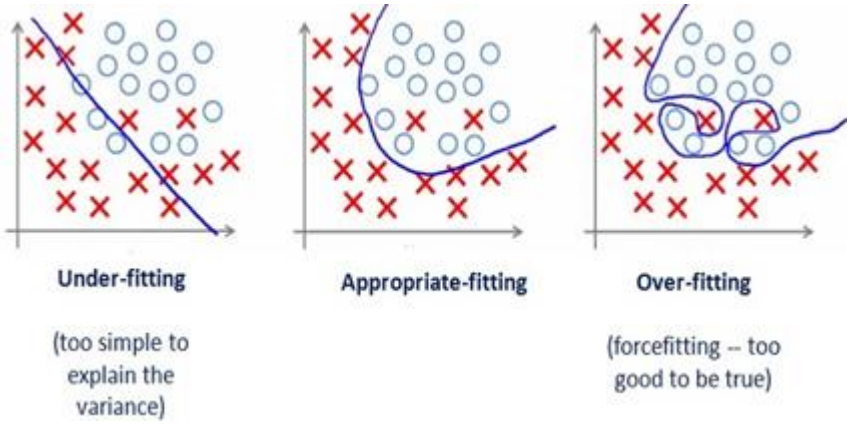
- The goal is to approximate the "true" objective function $f(x)$
  - We assume there is a pattern, otherwise there is nothing to do
- We start with a family of possible functions
  - For example, linear functions (Linear Regression)
- Let us denote the family as $h(x;\theta)$
  - x are the features
  - θ are the parameters
- We need to decide what kind of approximation we want
  - Do we want to be very good in the Training set?
  - Or to have comparable errors in Training and Validation sets?
  - We cannot have both in general, unless we compromise
- Very good in-sample is not always a blessing
  - By forcing a very high accuracy in the Training set we could be sacrificing predictive power on the Test set
  - In other words, we could be fitting the noise, not the signal
  - This is known as **overfitting**

i.sportstalkflorida.com

i.pinimg.com

Overfitting: bias and variance

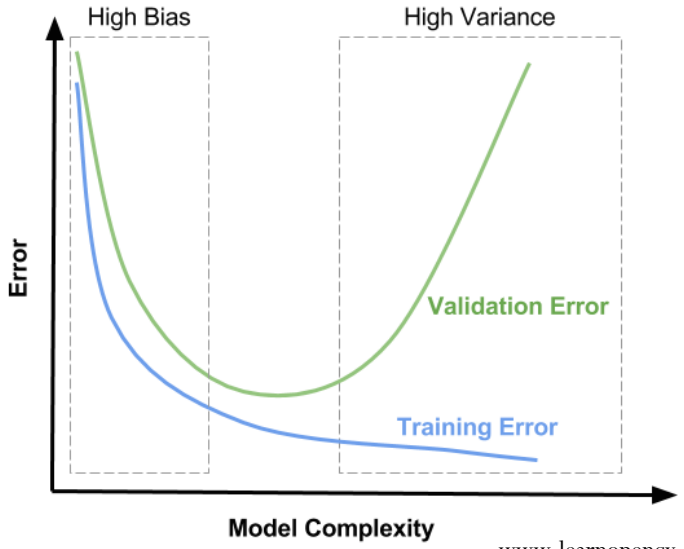
- We need to choose the right model
  - Too simple → underfitting
  - Too complex → overfitting



vitalflux.com

- High bias → underfitting
  - The errors in Training and Validation are comparable
    - ✓ The model performs well
  - But the errors are too high in the Training and Validation sets
    - ✓ The model cannot do better with the current complexity

- High variance → overfitting
  - The error in the Training set is small
    - ✓ But the error in the Validation set is big
  - We need to give up some accuracy on the Training set
    - ✓ So we can reduce error in the Validation set

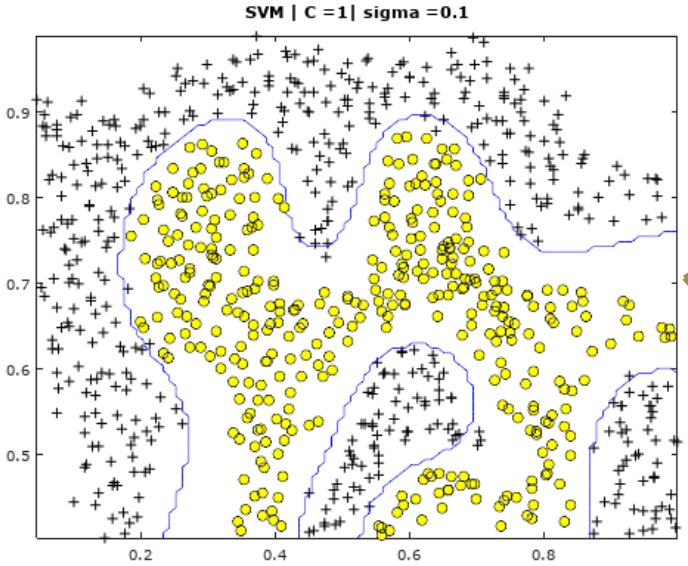


www.learnopencv.com

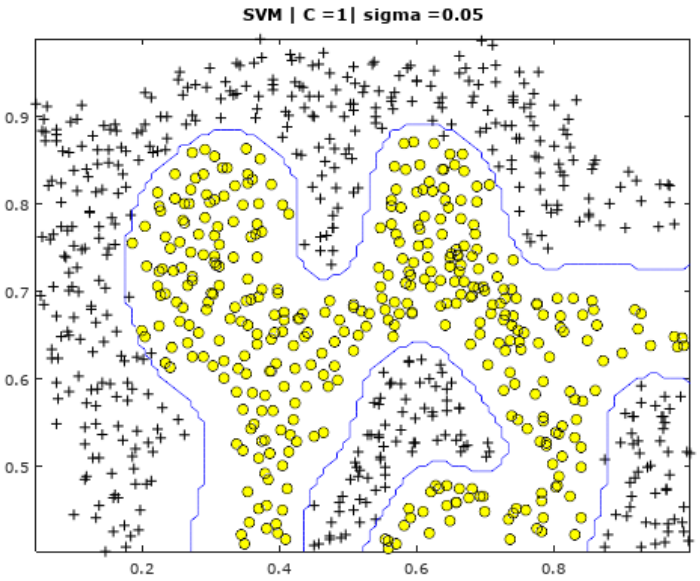
- **Rule of thumb for "sweet spot" of model complexity:**
  - Number of features ≤ number of samples / 10

Example of Overfitting

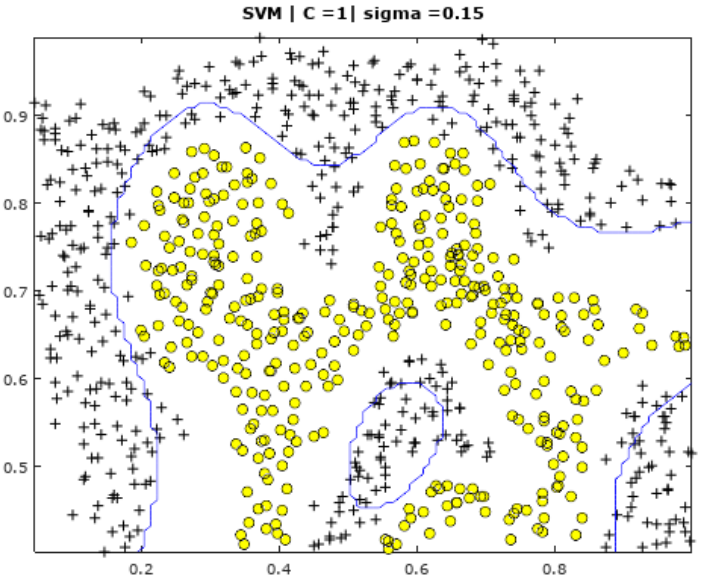
- Support Vector Machines
- Gaussian Kernel
  - $C = 1, \sigma = 0.1$



decrease σ
"Overfit"



"increase σ
Underfit"

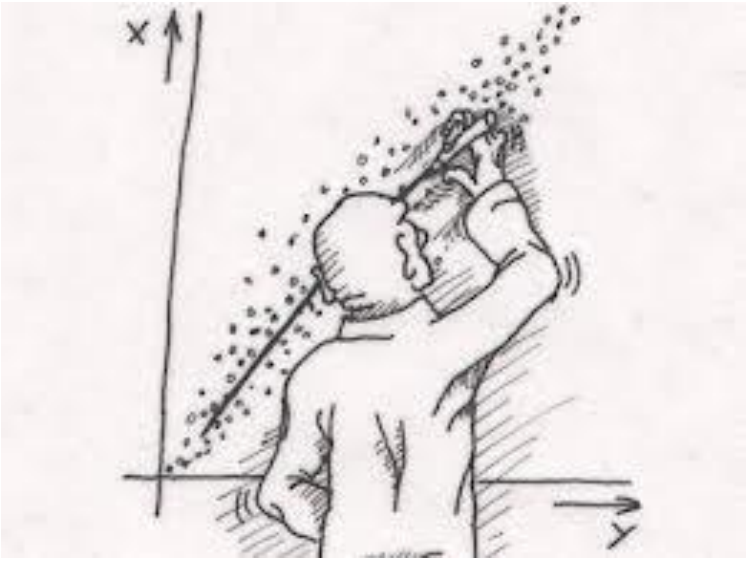


- You will do this in the ML MOOC from Stanford - Coursera

2. Linear Regression

Linear regression explained

- We start with a **real-valued objective**
  - $y \in R$
- That can be described with N features
  - $x = (1, x_1, \dots, x_N)$
- Assume that a linear combination of the features
  - $\theta_0 + \theta_1 x_1 + \cdots + \theta_N x_N = x\theta$
- Can **approximate the objective**
  - $x\theta \approx y$
- Given a set of M samples and their respective objectives
  - $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \dots, \left(x^{(M)}, y^{(M)}\right)$
- We want to **find the parameters** that better approximate the objective
  - $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_N \end{bmatrix} \rightarrow x^{(m)}\theta \approx y^{(m)}$ for all $m = 1, \dots, M$



algo-trades.com

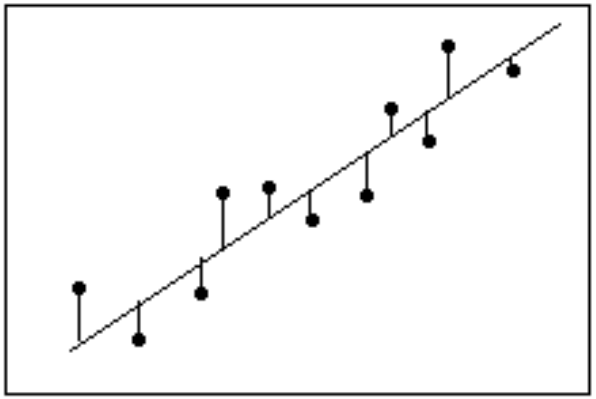
Defining the cost function

- Define the Cost function as the error between the objective and our approximation

$$Cost(\theta) = \frac{1}{2M}\sum_{m=1}^{M}\left(h\left(x^{(m)};\theta\right) - y^{(m)}\right)^2$$

- Recall that $h\left(x^{(m)};\theta\right) = x^{(m)}\theta$, hence

$$Cost(\theta) = \frac{1}{2M}\sum_{m=1}^{M}\left(x^{(m)}\theta - y^{(m)}\right)^2$$



cdn2.content.compendiumblog.com

- Let us write the equation in vectorial form. If we define

$$X = \begin{bmatrix} 1 & {x_1}^{(1)} & \cdots & {x_N}^{(1)} \\ 1 & {x_1}^{(2)} & \cdots & {x_N}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & {x_1}^{(M)} & \cdots & {x_N}^{(M)} \end{bmatrix}, \quad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(M)} \end{bmatrix}$$

- Then

$$Cost(\theta) = \frac{1}{2M}(X\theta - Y)^T(X\theta - Y)$$

Minimising the cost function

- We just wrote the cost function as

$$Cost(\theta) = \frac{1}{2M}(X\theta - Y)^T(X\theta - Y)$$

- If we want to minimise the Cost we need to compute its gradient

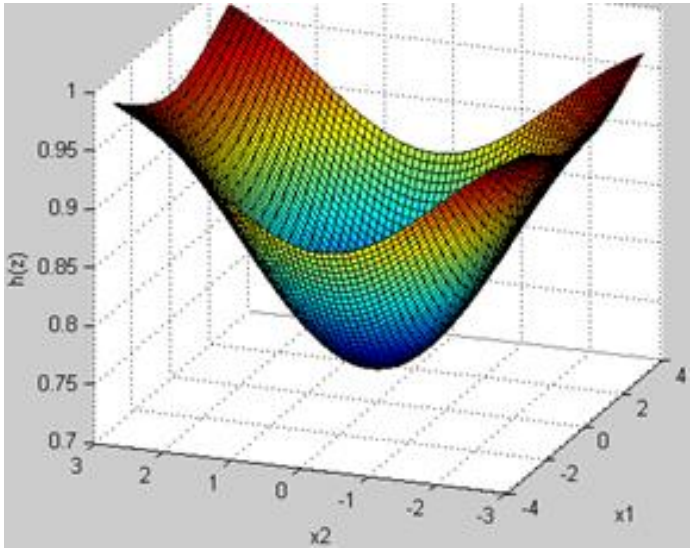
$$\nabla_\theta Cost(\theta) = \frac{1}{M}X^T(X\theta - Y)$$

- Making $\nabla_\theta Cost(\theta) = 0$ we found the optimal parameters

$$\theta = (X^T X)^{-1}X^T Y$$

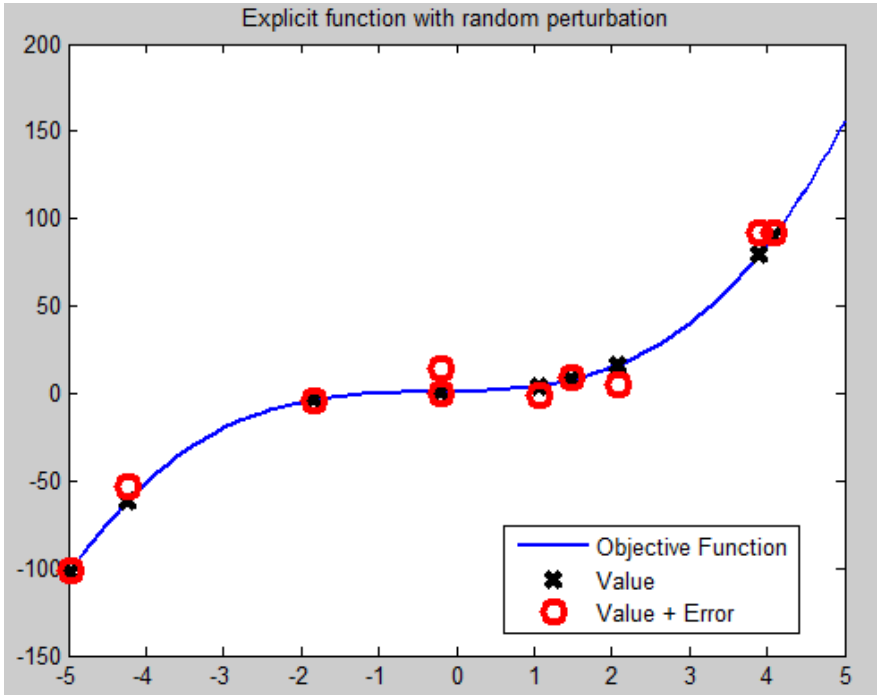
- Let us see that θ we just found is a global minimum:
  - The Hessian (matrix of second derivatives) is $H = \frac{1}{M}X^T X$
  - It is positive-definite because for any vector $w \neq 0$ we have

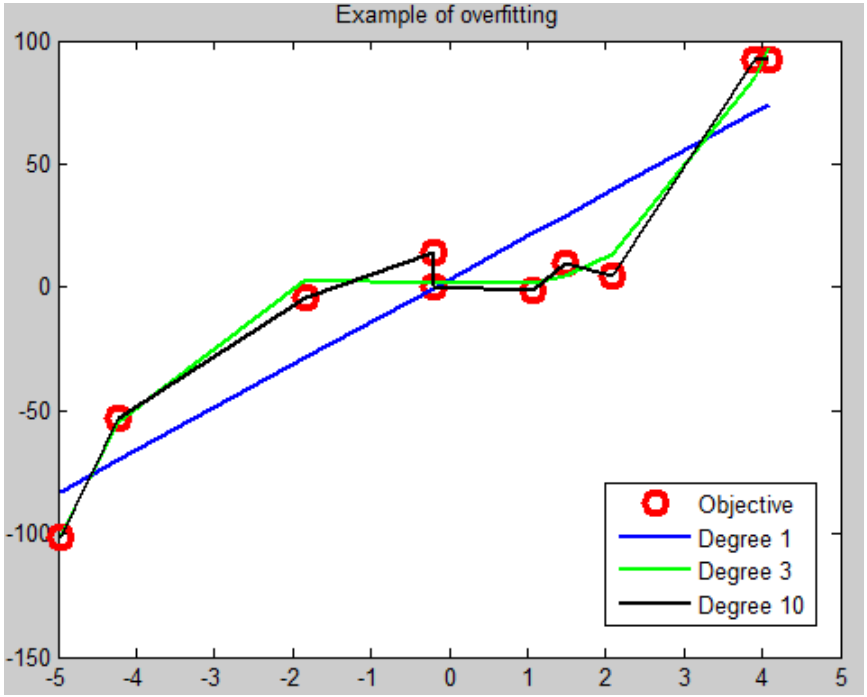
$$w^T Hw = \frac{1}{M}(Xw)^T(Xw) > 0$$



Overfitting a linear regression

$$F(x) = 1 + x + x^2 + x^3$$





- We have 10 samples and we use a 10-degree polynomial to fit
- We are not following our rule of thumb:
  - Number of features ≤ number of samples / 10

Fixing overfitting: regularisation

- We add a **penalty** to the cost function to **"force θ to be small"**

$$Cost(\theta) = \frac{1}{2M}\sum_{m=1}^{M}\left(h\left(x^{(m)};\theta\right) - y^{(m)}\right)^2 + \frac{\lambda}{2M}\sum_{n=1}^{N}{\theta_n}^2$$

exclude θ_0

- Define J as the identity matrix but with a zero in its first entry:

$$J = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

- Then

$$Cost(\theta) = \frac{1}{2M}(X\theta - Y)^T(X\theta - Y) + \frac{\lambda}{2M}(J\theta)^T(J\theta)$$

- The gradient is

$$\nabla_\theta Cost(\theta) = \frac{1}{M}X^T(X\theta - Y) + \frac{\lambda}{M}J\theta$$

- Therefore

$$\theta = (X^TX + \lambda J)^{-1}X^TY$$

- The regularisation will make it harder for $h(x;\theta)$ to take very convoluted shapes
  - Smoother shape → Reduced variance
  - "Less is more"

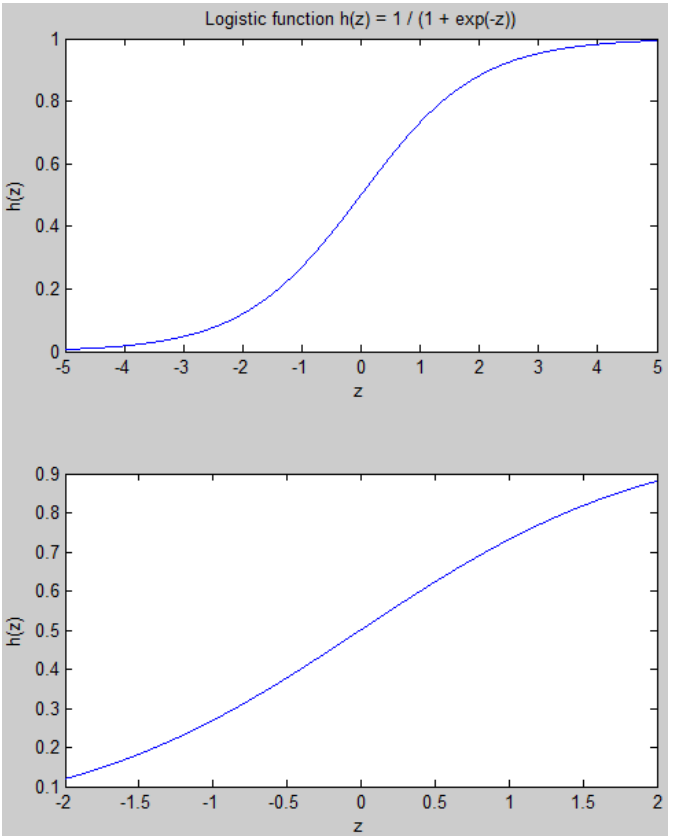
spiritclothing.ie

cdn.shopify.com

3. Logistic Regression

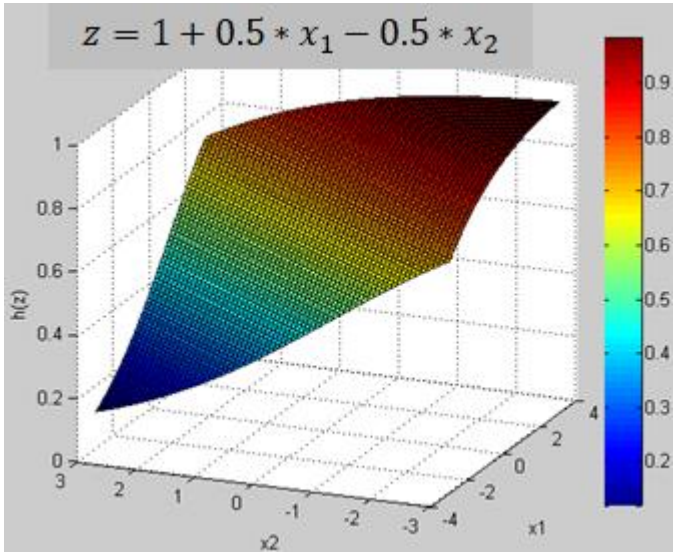
Logistic regression explained

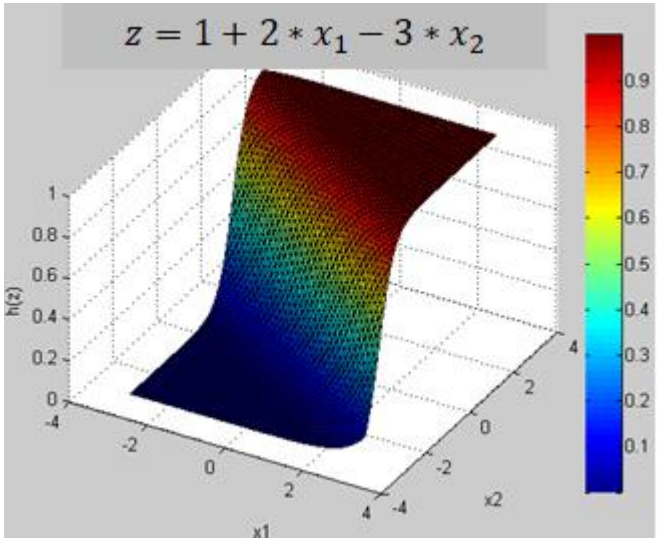
- We start with a **binary objective**
  - $y \in \{0,1\}$
- That can be described with a number of features
  - $x = (1, x_1, \dots, x_N)$
- Assume that a linear combination of the features
  - $\theta_0 + \theta_1 x_1 + \cdots + \theta_N x_N = x\theta$
- Together with a **logistic function**
  - $z \to h(z) \in (0,1)$
- Can **approximate the probability of the objective**
  - $h(x\theta) \approx P[y = 1]$

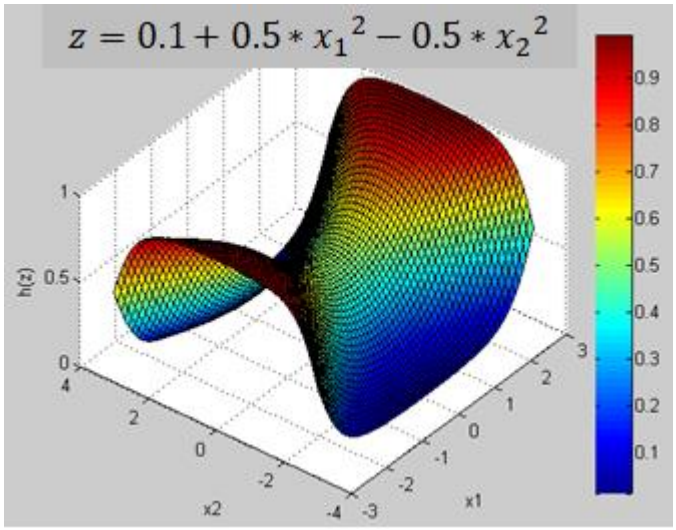


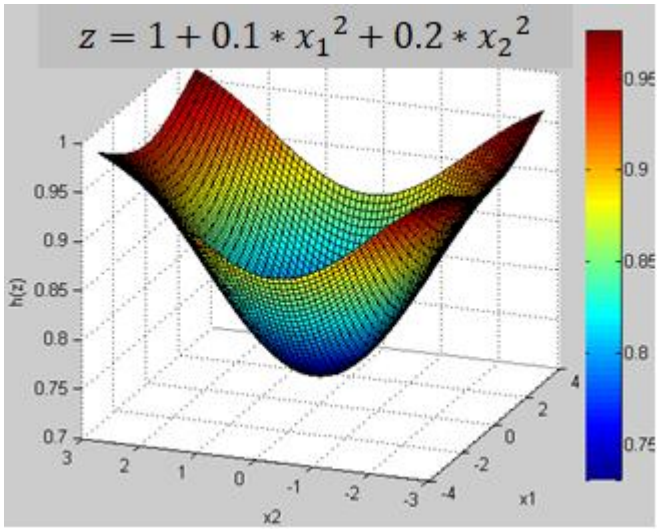
- We want to **find the parameters** that better approximate the probability of the objective
  - $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_N \end{bmatrix} \to h(x^{(m)}\theta) \approx P[y^{(m)} = 1]$ for all $m = 1, \dots, M$

Classification boundaries

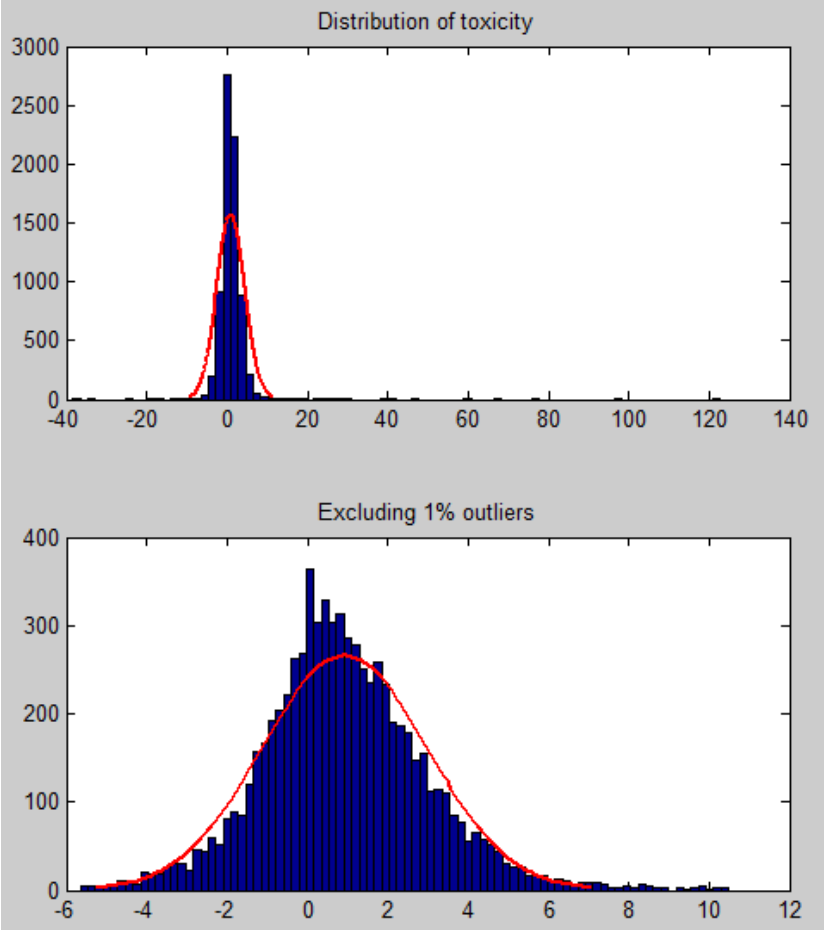








Example: "toxicity" of trades



- We label trades based on execution costs:
  - "**Good**" if the **cost is lower** than expected
  - "**Toxic**" if the **cost is higher** than expected
- To determine the label, for each trade we have defined a metric called "**toxicity**"
  - Good if Toxicity < 1
  - Toxic if Toxicity > 1
- Why this is important?
  - "Good" trades can be executed with simple algorithms (e.g. VWAP)
  - "Toxic" trades require sophisticated executions (and probably human intervention)
- **Goal**:
  - Build a **classification model** that, given a trade, will label it as Good or Toxic

Preparing the data

12281x6 table

	1 VolatilityPct	2 SpreadBps	3 AdvPct	4 Factor1	5 Factor2	6 Toxicity
1	16.1659	8.4360	0.3774	0.9118	1.0275	1.5519
2	17.3522	6.2469	1.2605	2.0551	1.0275	-1.0618
3	15.6961	6.9970	3.2743	0.7892	1.0275	0.9216
4	22.8170	6.5632	3.1146	0.7892	1.0275	-1.3854
5	15.2506	6.9724	2.1265	0.8743	1.0275	-1.6234
6	17.6174	6.6135	3.2121	0.8743	1.0275	2.9291
7	22.3882	6.4957	4.0615	0.8743	1.0275	0.3520
8	20.0119	6.6200	0.3403	0.8743	1.0275	1.9252
9	15.8866	6.0107	4.5123	0.8743	1.0275	-0.0434
10	15.7958	5.8956	0.5968	0.8743	1.0275	2.1466
11	20.2320	6.0577	1.7850	0.8743	1.0275	-1.2878
12	17.8535	6.1534	3.5647	0.8743	1.0275	-1.3685
13	14.1365	6.2211	3.0945	0.8743	1.0275	-0.3788
14	15.5020	6.2357	3.5233	0.8743	1.0275	0.7334
15	19.5245	10.0757	3.2914	0.8743	1.0275	-0.2846
16	18.8058	10.0923	4.9634	0.8743	1.0275	0.2295
17	18.8058	10.0923	5.2615	0.8743	1.0275	0.6499
18	16.0946	6.2299	2.9995	-0.2582	1.0275	3.5644
19	15.3196	9.8494	8.9541	-0.2582	1.0275	-0.2809
20	16.6556	7.9850	1.3000	1.0135	1.0275	1.0134

- Five features to explain "Toxicity" of a trade
- Train set
  - 60% i.e. 7,369 samples
  - Here we calibrate the ML model
  - Optimal parameters for the logistic regression
- Cross-validation set
  - 20% i.e. 2,456 samples
  - Here we pick the optimal threshold for the logistic regression
- Test set
  - 20% i.e. 2,456 samples
  - Here we only check the accuracy of the model

Training the ML model

- Preparing the data
  - Transform Toxicity into booleans
  - Normalise the variables
    - ✓ Substracting the mean
    - ✓ Dividing by range, either max-min or std dev

```
y_good = vecToxicity < 0;
y_neutral = vecToxicity >= 0 & vecToxicity < 1.5;
y_toxic = vecToxicity >= 1.5;
```

- Training the ML model
  - It is just one line of code!

```
[theta, dev, stats] = glmfit(mtxData,y_train,'binomial','link','probit');
```

- Predicting outcomes based on the ML model
  - Compute the logistic with the calibrated parameters

```
x = [ones(size(mtxData,1),1),mtxData] * theta;
h = 1 ./ (1 + exp(-x));
```

- Compute the accuracy of the model in the training set

```
threshold = 0.5;
vecPred = h > threshold;
accuracyTrain = mean(vecPred == y_train)
```

Training for Good

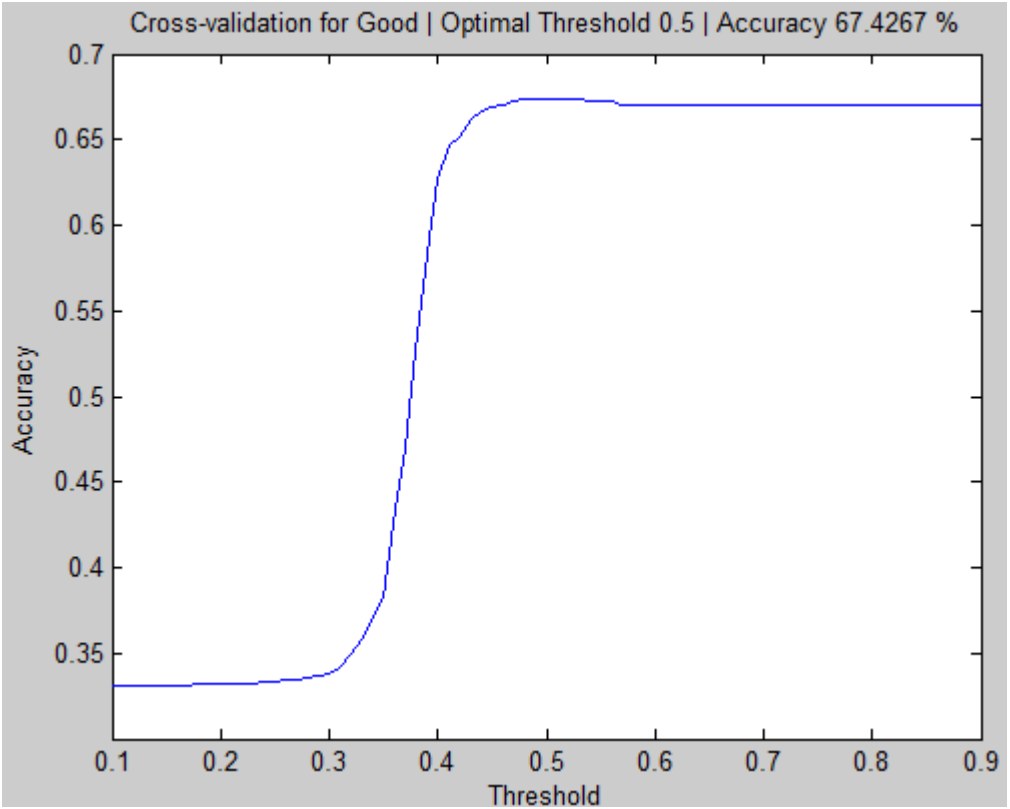
6x3 table

	1 feature	2 theta	3 pvalue
1	'None'	-0.4953	4.3030e-228
2	'VolatilityPct'	-0.0170	0.3586
3	'SpreadBps'	0.0430	0.0186
4	'AdvPct'	0.0134	0.4147
5	'Factor1'	-0.1683	7.0431e-23
6	'Factor2'	-0.0247	0.1060

- Features that matter (p-value < 0.05):
  - None
  - SpreadBps
  - Factor1

- Features that we can ignore:
  - VolatilityPct
  - AdvPct
  - Factor2

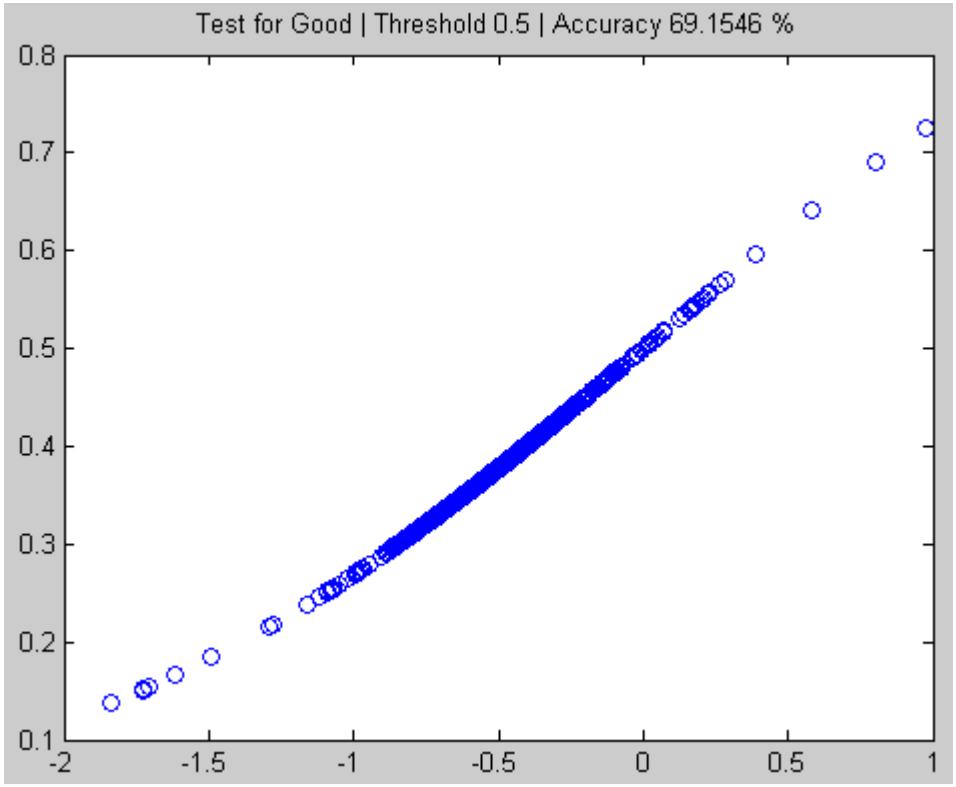
Cross-validation for Good



- Pick optimal threshold that maximises accuracy:
  - 0.49 and 0.5
- I love symmetry
  - Hence I choose **0.5**

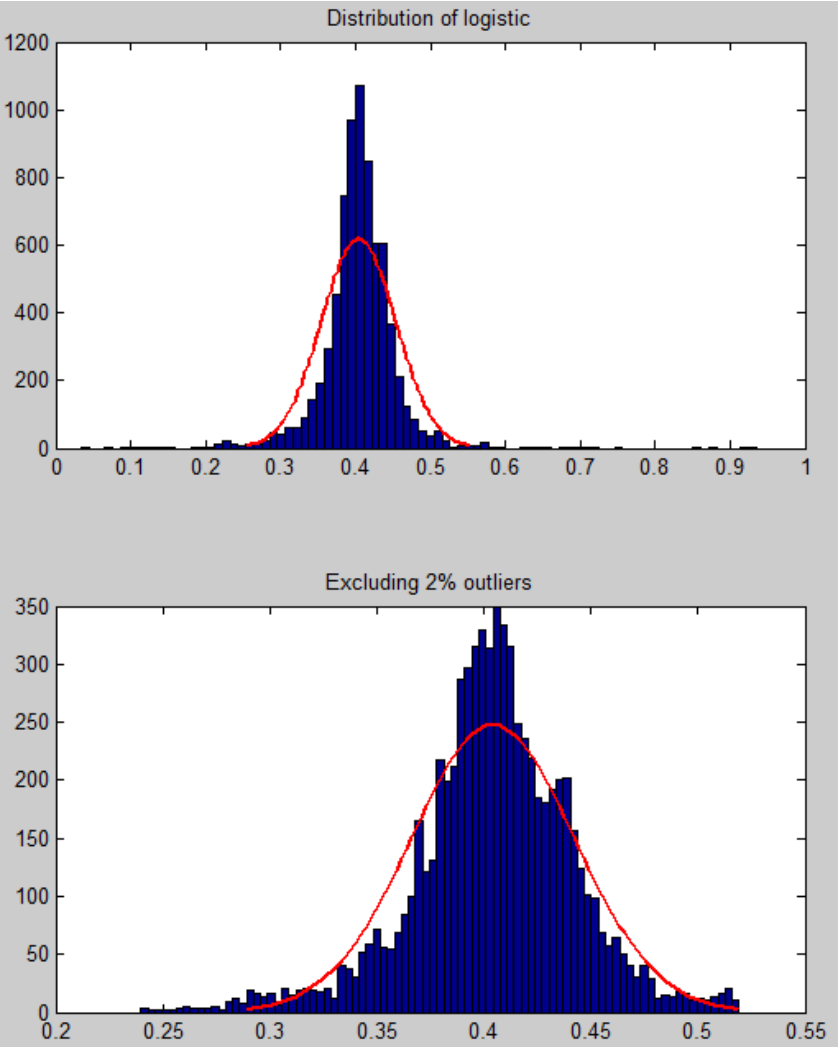
static7.depositphotos.com

Test for Good: we are done, right?



- This result seems OK
  - 69% accuracy
  - Better than random coin toss i.e. 50%
- Better than 2/3 of chance
  - In line with the accuracy in the training set i.e. 69%

Not really…



- Only 2% of samples have $h > 0.5$
- We know that around 33% of samples are good
  - Around 67% accuracy by predicting "not good"
- Using $h \equiv 1$
  - We predict all samples are not good
- So a constant prediction is as good as our sophisticated ML model?

  <sarcasm>
  - Thank you ML, you are so useful!

  </sarcasm>
- We need to change our approach
  - ML is a continuous iterative process

Choose the threshold differently

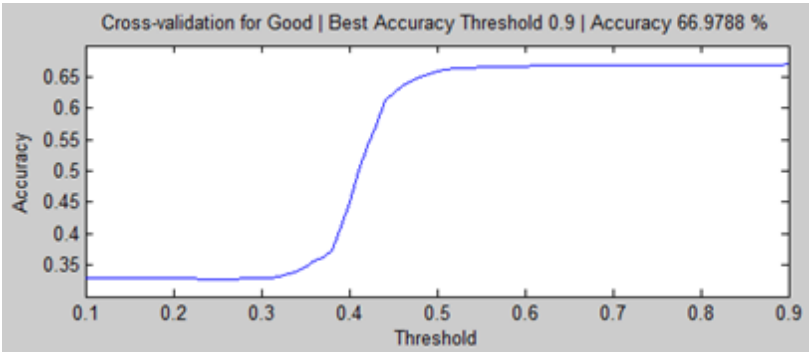
- Let us compare our predictions vs the actual values:

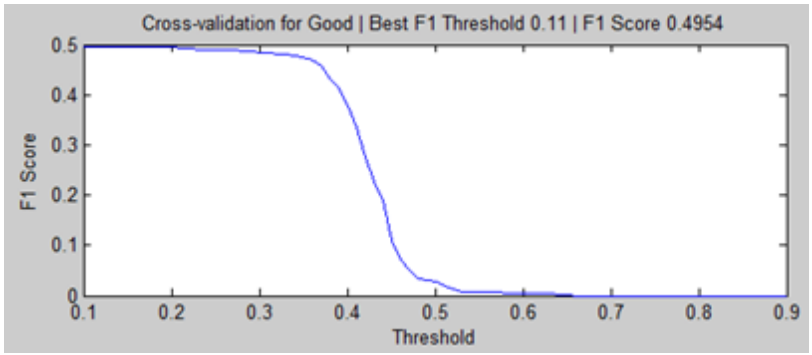
		Actual	
		1	0
Predicted	1	True Positive	False Positive
	0	False Negative	True Negative

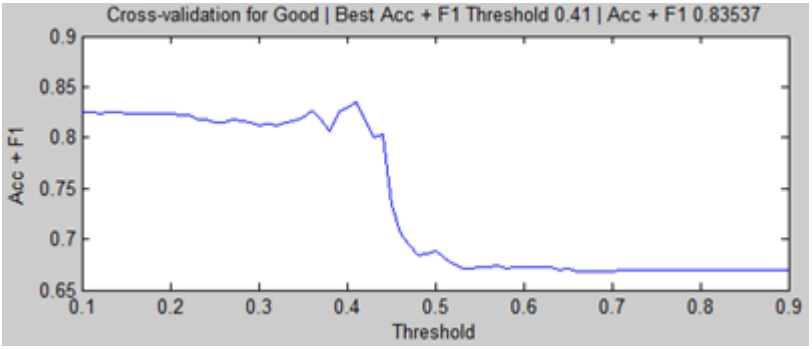
www.ancientpages.com

- We will use the F1-Score
  - Precision $P = \frac{True\ Positives}{Predicted\ Positives}$
  - Recall $R = \frac{True\ Positives}{Actual\ Positives}$
  - $F = 2\frac{PR}{P+R}$
- Choose the threshold in cross-validation that maximises F

Comparing Accuracy and F1 Score

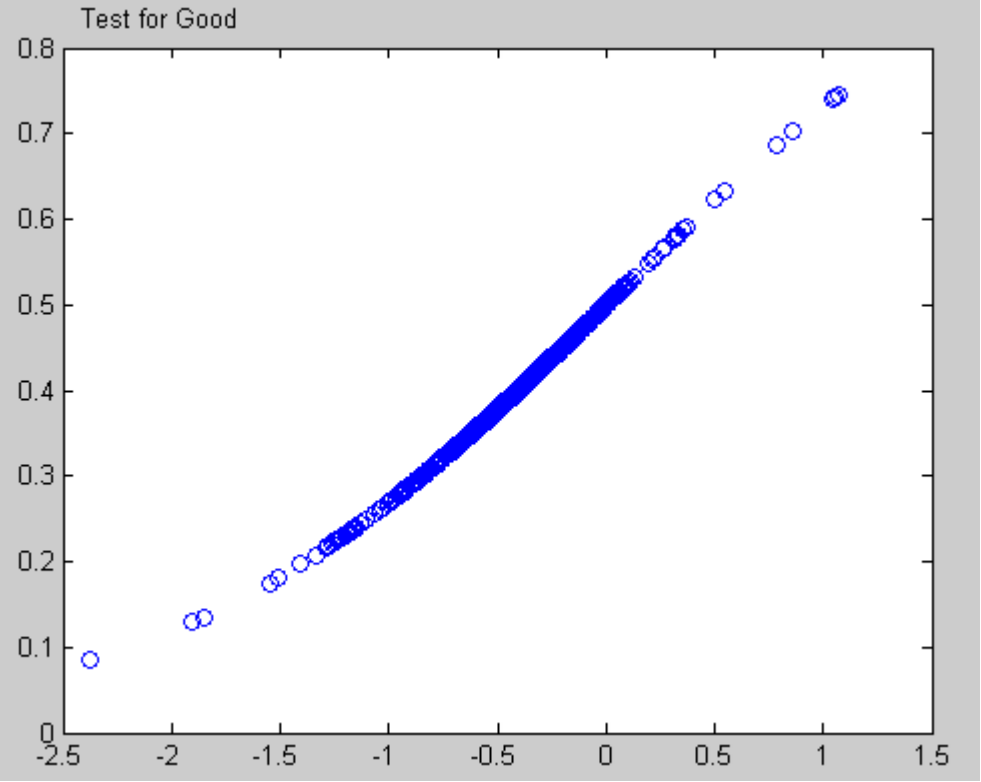






- Best accuracy is for threshold = 0.9
  - But we are almost predicting all samples are not Good, i.e. $h \equiv 1$
- Best F1 score is for threshold = 0.11:
  - But we are almost predicting all samples are Good, i.e. $h \equiv 0$
- What if we add Accuracy and F1 score?
  - We are trying to predict well without being either $h \equiv 0$ or $h \equiv 1$
  - Best threshold is 0.41
    - ✓ Accuracy is 49%
    - ✓ F1 Score is 0.33
    - ✓ It is around the median for h

Test for Good 2: nailed it already?



- This result seems better than the previous one:
  - Lower accuracy: 49%
  - Random choice of Good, Neutral and Toxic is around 33%
  - 95% confidence assuming Bernoulli is
    - ✓ $1.96 * \sqrt{\frac{0.33*0.67}{2456}} \sim 2\%$
- In general terms, the in-sample accuracy should be similar to out-of-sample accuracy

Logistics for multi-class problems

- We know how to deal with binary classifications
  - Good
  - Neutral
  - Toxic

- But what if we have multiple classes?
  - Toxicity: Good, Neutral, Toxic
  - Number recognition: 0,1,2,3,4,5,6,7,8,9
  - Chess: pick which piece to move
  - MMORPGs: pick what spell/skill to use
    - Maximise DPS, aggro, defence, heals, etc

- The philosophy of the One vs All approach
  - We already have the single-class logistics
  - We can compare them all and pick the best

http://assets1.ignimgs.com

One vs All: from single logistics …

- We choose Good, Neutral and Toxic based on 33% quantiles of toxicity

```
y_good = vecToxicity < 0;
y_neutral = vecToxicity >= 0 & vecToxicity < 1.5;
y_toxic = vecToxicity >= 1.5;
```

- We run 3 logistic regressions, one for each class type

```
% calibration
[theta_good, dev_good, stats_good] = glmfit(mtxData,y_good,'binomial','link','probit');
[theta_neutral, dev_neutral, stats_neutral] = glmfit(mtxData,y_neutral,'binomial','link','probit');
[theta_toxic, dev_toxic, stats_toxic] = glmfit(mtxData,y_toxic,'binomial','link','probit');
```

- This gives us 3 sets of optimal parameters for the 3 logistics

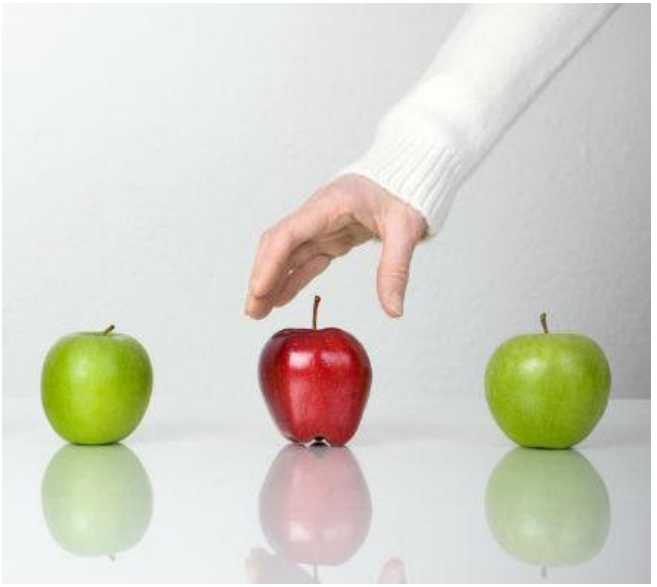
1 feature	2 theta_good	3 theta_neutral	4 theta_toxic
'None'	-0.0074	-6.9571e-04	0.0074
'VolatilityPct'	-0.0134	0.0278	0.0134
'SpreadBps'	0.1102	-0.0083	-0.1102
'AdvPct'	0.0118	0.4495	-0.0118
'Factor1'	-0.1854	-0.0187	0.1854
'Factor2'	-0.0273	-0.0022	0.0273

… to the best outcome

- We have computed 3 logistic regressions
  - Good
  - Neutral
  - Toxic

- We now apply the One vs All method:
  - Label each sample based on the highest logistic value

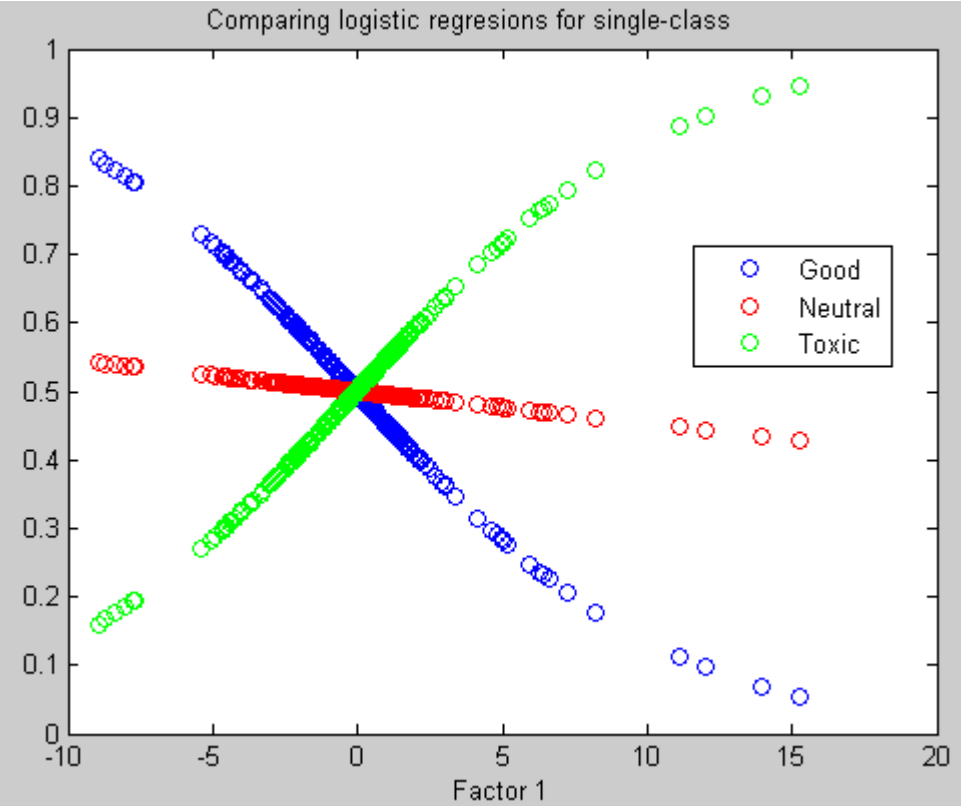
```
% prediction
theta = [theta_good,theta_neutral,theta_toxic];
x = [ones(size(mtxData,1),1),mtxData] * theta;
x = sort(x);
h = 1 ./ (1 + exp(-x));
% 1 vs all
[~,class_pred] = max(h,[],2);
```



www.tes.com

- Here we do not need to calibrate the threshold for classification
  - Only need Train set and Test set, no Cross-validation set

One vs All: comparing logistics



- Let us focus on only one feature
  - For example, Factor1
  - Make zero all other factors
- If Factor1 increases then:
  - Less likely to be a good sample
  - More likely to be a toxic sample
  - Neutral is more or less invariant
- We could try again without the Neutral class
  - **Homework!**

knowyourmeme.com

One vs All: Results

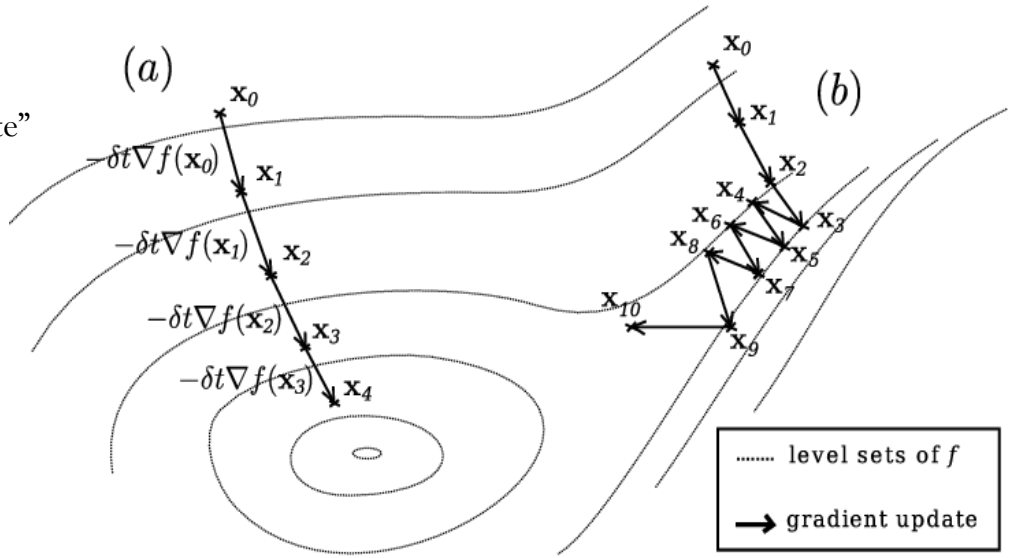
	Accuracy	Confidence Interval 95%			Significant
		Error	Lower Bound	Upper Bound	
Train Set	37.88%	1.14%	36.73%	39.02%	Yes
Test Set	37.66%	1.40%	36.26%	39.06%	Yes

- Did we succeed in the ML model for toxicity?
- Statistically speaking, yes
  - Predictive power (in-sample and out-of-sample) above 33% with 95% confidence
- But in practice, is this model good enough to trade real money?
  - I think it is a very good start
  - But we could do much better if we work the model a bit more
- What can be done?
  - Use regularisation to compute the optimal parameters
  - Add more features
- If you do not know what features to add, use **Neural Networks (tomorrow)**

Gradient Descent explained

- What happens if we do not have a closed-form solution of the parameters?
  - We need to find the minimum numerically
- Gradient Descent for a function $C(\theta)$
  - Move in the direction where $C(\theta)$ has fastest decrease
  - That is $-\nabla C(\theta)$
- We will move in the parameter space at a given **learning rate** α
  - $\theta(k+1) = \theta(k) - \alpha \nabla C(\theta(k))$
- Sometimes it is written as a "programming update"
  - $\theta := \theta - \alpha \nabla C(\theta)$

ludovicarnold.altervista.org



Gradient Descent and convergence

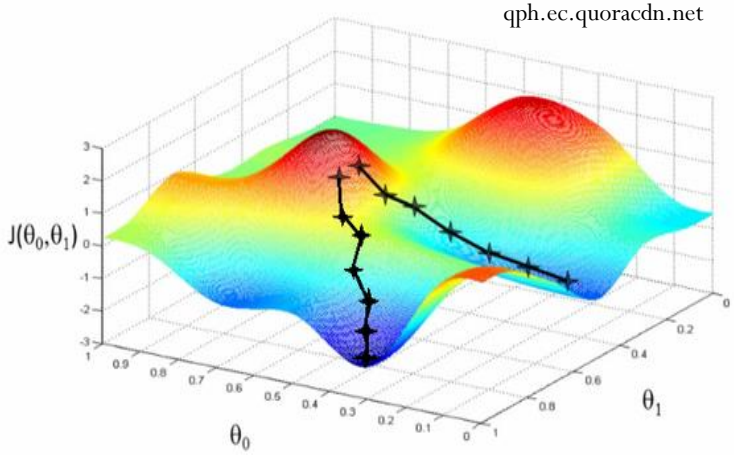
- The learning algorithm for gradient descent, we just found, is
  - $\theta := \theta - \alpha \nabla C(\theta)$

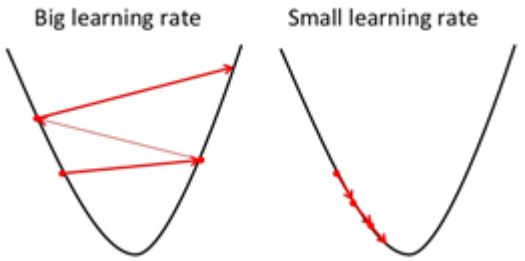
- **Initial condition** is important
  - Esp. when the cost function is not convex
  - We could converge towards a local minimum instead
  - This is one of the biggest headaches in Neural Networks

- **Learning rate** is important
  - Small learning rate could take ages to converge
  - Big learning rate could "overshoot" the target and diverge

- **Rule of thumb**:
  - Plot $C(\theta(k))$ vs iteration k to check if it is decreasing

qph.ec.quoracdn.net





media.licdn.com

Gradient Descent for logistic regression

- For Logistic Regression, the cost function we will use is

$$Cost(\theta) = -\frac{1}{M}\sum_{m=1}^{M}\left[y^m \log h\left(x^{(m)}\theta\right) - (1-y^m)\log(1-h\left(x^{(m)}\theta\right))\right]$$

- This cost function is convex in $h\left(x^{(m)}\theta\right)$ and has a unique global minimum.

- If we compute the partial derivatives, using $h' = h(1-h)$ we obtain

$$\frac{\partial Cost}{\partial \theta_n} = \frac{1}{M}\sum_{m=1}^{M}\left(h\left(x^{(m)}\theta\right) - y^m\right) {x_n}^{(m)}$$

- Coincidentally, this is the same derivative form for the Linear Regression, but different $h\left(x^{(m)}\theta\right)$.

- In vectorial form

$$\nabla_\theta Cost = \frac{1}{M}X^T(h(X\theta) - Y)$$

- In consequence, the learning algorithm is

$$\theta := \theta - \alpha\frac{1}{M}X^T(h(X\theta) - Y)$$

si.com

Gradient Descent with regularisation

- We just add the regularisation term, as before

$$Cost(\theta) = -\frac{1}{M}\sum_{m=1}^{M}\left[y^m \log h\left(x^{(m)}\theta\right) - (1-y^m)\log\left(1-h\left(x^{(m)}\theta\right)\right)\right] + \frac{\lambda}{2M}\sum_{n=1}^{N}{\theta_n}^2$$

- The gradient in vectorial form is easy to compute:

$$\nabla_\theta Cost = \frac{1}{M}(h(X\theta) - Y)^T X + \frac{\lambda}{M} J\theta$$

- In consequence, the learning algorithm is

$$\theta := \theta - \alpha\left[\frac{1}{M}(h(X\theta) - Y)^T X + \frac{\lambda}{M} J\theta\right]$$

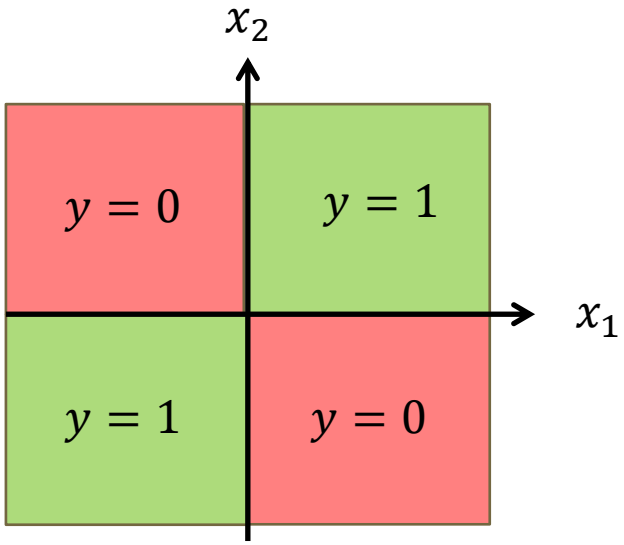
i.pinimg.com

cdn.shopify.com

4. Neural Networks

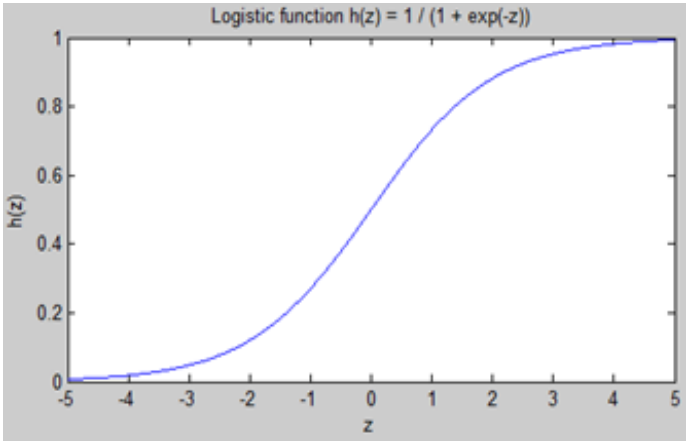
Nonlinear classification: motivation

- How do we deal with a nonlinear classification problem?
  - $y = 1$ in the first and third quadrant
  - $y = 0$ in the second and fourth quadrant



- Let us simplify the problem
  - $x_1, x_2 \in \{-1,1\}$
  - $y = 1$ for $(x_1, x_2) = (1,1)$ or $(-1,-1)$
  - $y = 0$ for $(x_1, x_2) = (1,-1)$ or $(-1,1)$

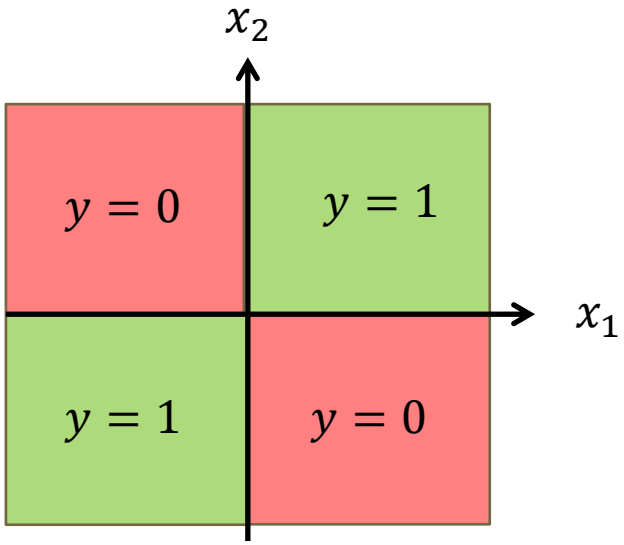
- If we want to use the logistic regression, we can:
  - Add features that are nonlinear in the original features (x_1, x_2)
  - Use the fact that
    - $h(z) \approx 0$ for $z \leq -5$
    - $h(z) \approx 1$ for $z \geq +5$



- Of course, it will only work if we know (or at least suspect) what nonlinear features to add
  - What features could be good candidates?

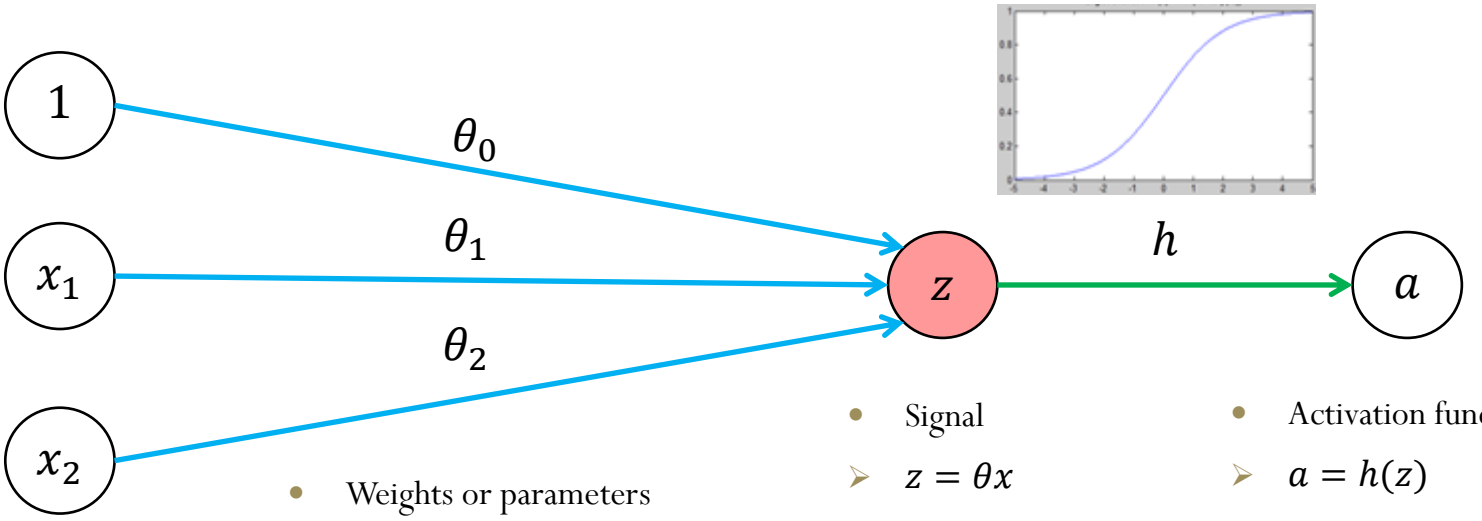
Solution via nonlinear logistic

- Let us try quadratic terms
  - $x = (1, x_1, x_2, x_3, x_4, x_5) = (1, x_1, x_2, {x_1}^2, {x_2}^2, x_1 x_2)$
- We need $\theta^T = (\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5)$ such that
  - $h(x\theta) \approx 1$ for $(x_1, x_2) = (1,1)$ or $(-1,-1)$
  - $h(x\theta) \approx 0$ for $(x_1, x_2) = (1,-1)$ or $(-1,1)$
- But we know that this can be translated into
  - $x\theta \geq 5$ for $(x_1, x_2) = (1,1)$ or $(-1,-1)$
  - $x\theta \leq -5$ for $(x_1, x_2) = (1,-1)$ or $(-1,1)$
- There are several solutions, but a very simple one is
  - $\theta^T = (0,0,0,0,0,5)$
- But what if we do not have the intuition for quadratic terms?
- Or if the quadratic solution did not work?
  - We need a systematic way to add extra nonlinear features



x_1	x_2	$5x_1x_2$	$h(5x_1x_2)$
1	1	5	1
-1	1	-5	0
-1	-1	5	1
1	-1	-5	0

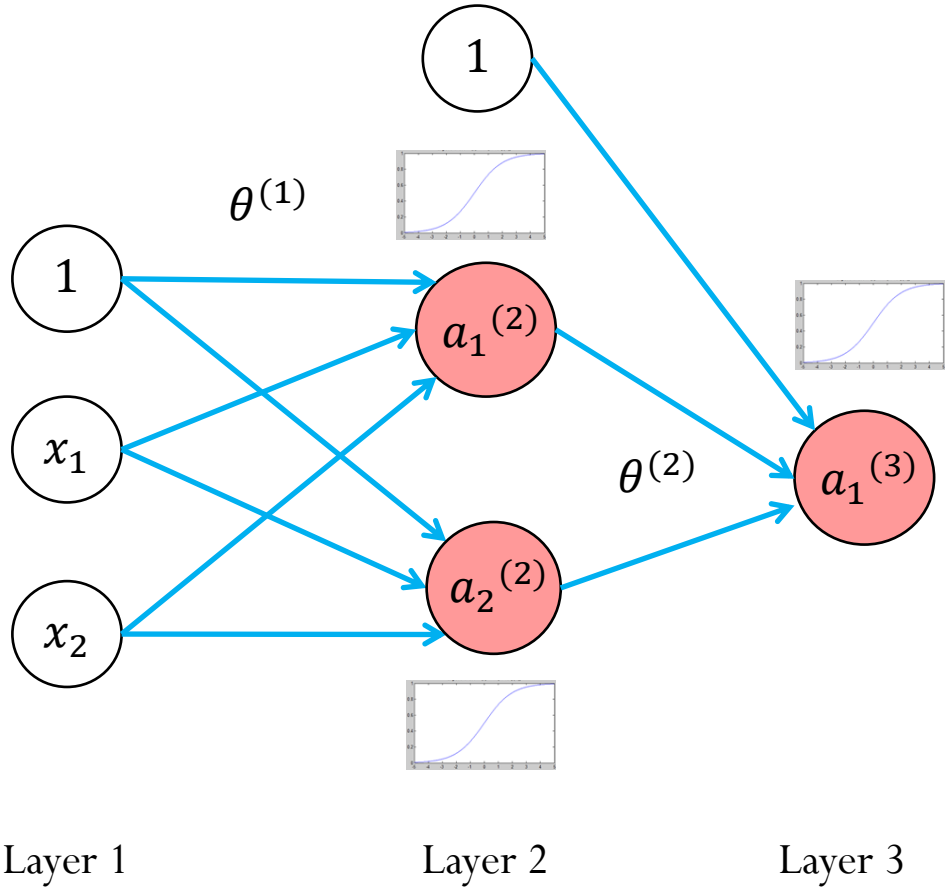
What is a logistic unit?



- Input
  - $x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$
- Weights or parameters
  - $\theta = [\theta_0, \theta_1, \theta_2]$
  - **Blue arrow** → product
- Signal
  - $z = \theta x$
  - **Red ball** → sum
- Activation function
  - $a = h(z)$
  - **Green arrow** → apply logistic function

- For simplicity, we will merge the signal z and the activation a in the **red ball**
- In Neural Networks jargon, a logistic unit is called a "neuron"

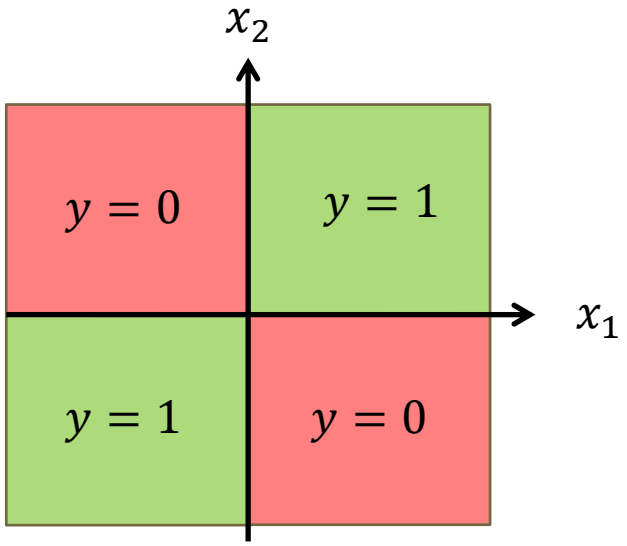
Combining logistic units



- **Blue arrow** → product
- **Red ball** → sum and activation
- $a^{(1)} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$
- $\theta^{(1)} = \begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} \\ \theta_{20}^{(1)} & \theta_{20}^{(1)} & \theta_{20}^{(1)} \end{bmatrix}$
- $z^{(2)} = \theta^{(1)} a^{(1)}$
- $a^{(2)} = \begin{bmatrix} 1 \\ h(z^{(2)}) \end{bmatrix}$
- $\theta^{(2)} = \begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} \end{bmatrix}$
- $z^{(3)} = \theta^{(2)} a^{(2)}$
- $a^{(3)} = \begin{bmatrix} 1 \\ h(z^{(3)}) \end{bmatrix}$

Nonlinear classification example

- Let us go back to our example
  - $x_1, x_2 \in \{-1,1\}$
  - $y = 1$ for $(x_1, x_2) = (1,1)$ or $(-1,-1)$
  - $y = 0$ for $(x_1, x_2) = (1,-1)$ or $(-1,1)$

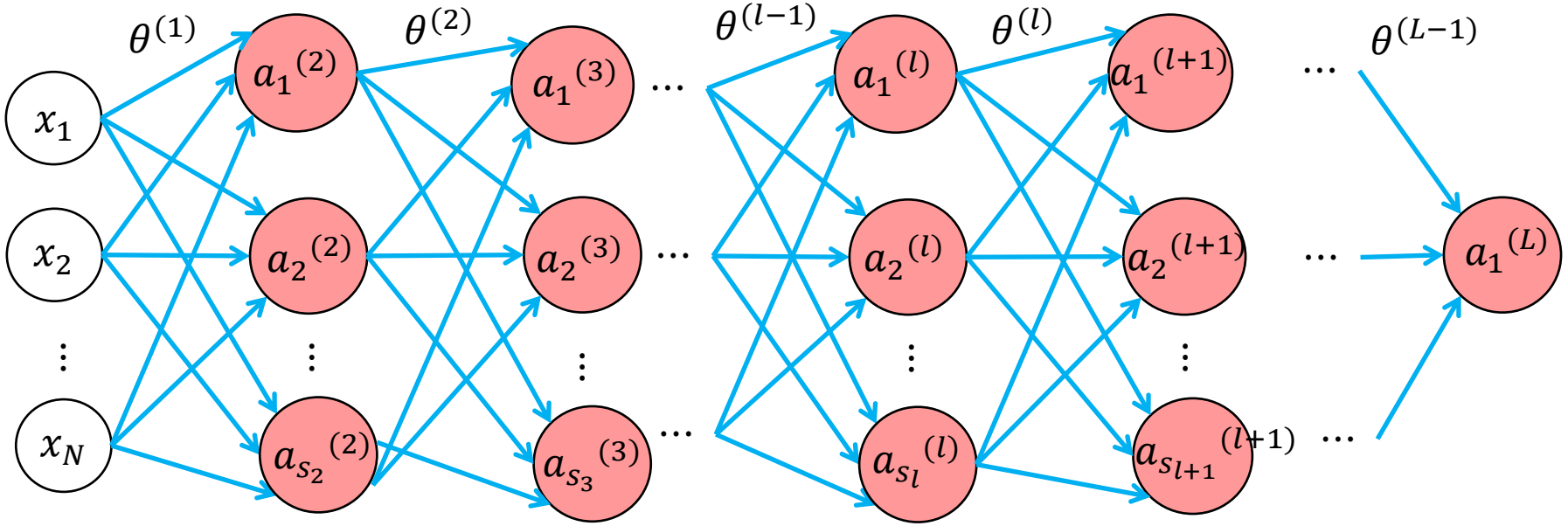


- Let us divide the problem in 3 logistic units:
  1. First quadrant: "AND" unit
     - ✓ ${a_1}^{(2)} = 1$ for $({a_1}^{(1)}, {a_2}^{(1)}) = (1,1)$ and zero elsewhere
  2. Third quadrant: "AND" unit
     - ✓ ${a_2}^{(2)} = 1$ for $({a_1}^{(1)}, {a_2}^{(1)}) = (-1,-1)$ and zero elsewhere
  3. "OR" unit
     - ✓ ${a_1}^{(3)} = 0$ for $({a_1}^{(2)}, {a_2}^{(2)}) = (0,0)$ and 1 elsewhere

x_1	x_2	${a_1}^{(2)}$	${a_2}^{(2)}$	${a_1}^{(3)}$
1	1	1	0	1
1	-1	0	0	0
-1	1	0	0	0
-1	-1	0	1	1

- This can be achieved by choosing:
  - $\theta^{(1)} = \begin{bmatrix} -15 & 10 & 10 \\ -15 & -10 & -10 \end{bmatrix}$
  - $\theta^{(2)} = [-5 \quad 10 \quad 10]$

General shape of a Neural Network



- We have L layers
  - 1 input layer $l = 1$, 1 output layer $l = L$ and $L - 2$ hidden layers $1 < l < L$
  - The network is called **deep learning** if there are **multiple hidden layers** (3 or more hidden layers)
- In Layer l we have s_l units or neurons (not counting the bias unit)
- $a_i^{(l)}$ is the activation of the i-th unit in Layer l
  - $a^{(l)} = h(z^{(l)})$
- $\theta^{(l)}$ is the matrix of weights for the transition from Layer l to Layer $l + 1$
  - $z^{(l+1)} = \theta^{(l)} a^{(l)}$
  - It has size $(S_{l+1}) \times (S_l + 1)$ (counting the bias unit)

Cost Function

- The cost function to use is the same as in logistic regression

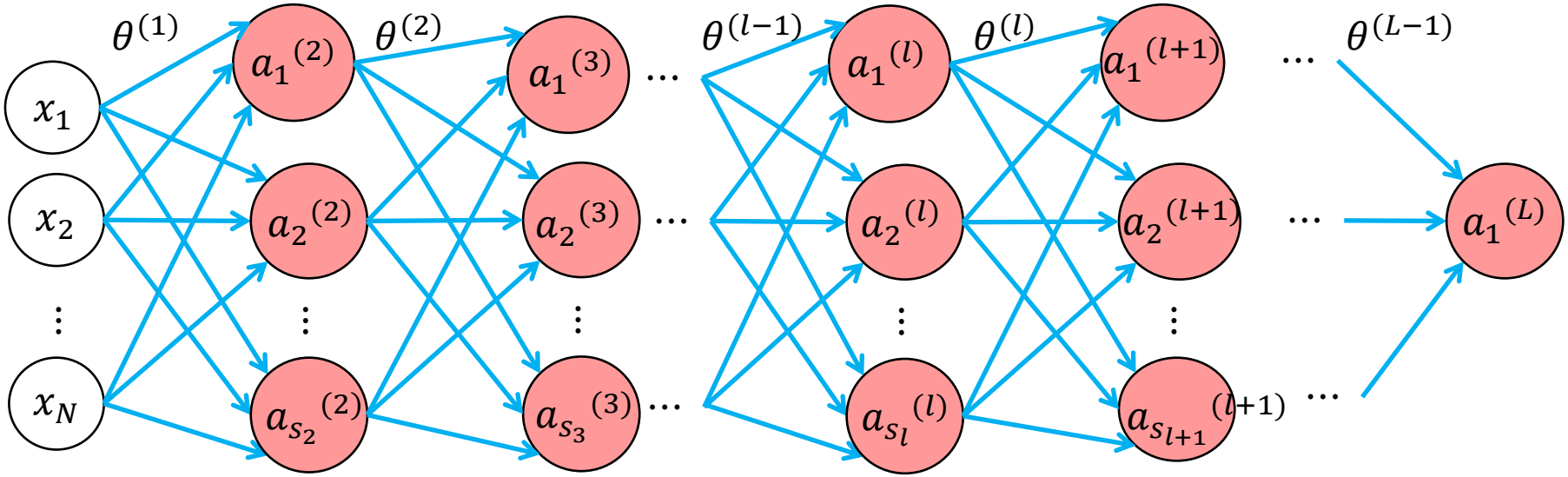
$$Cost(\theta) = -\frac{1}{M}\sum_{m=1}^{M}\left[y^m \log h\left(x^{(m)};\theta\right) - (1-y^m)\log(1-h\left(x^{(m)};\theta\right))\right]$$

- However, here the function $h\left(x^{(m)};\theta\right)$ is not a simple logistic but the result of compounding several layers of logistic units:
  - θ represents the vector of all weights
  - $\theta = \begin{bmatrix} \theta^{(1)}(:) \\ \theta^{(2)}(:) \\ \vdots \\ \theta^{(L-1)}(:) \end{bmatrix}$ where $\theta^{(l)}(:)$ is the "unrolled" vector version of the matrix $\theta^{(l)}$ e.g. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$
- In the case of regularisation, the cost function becomes

$$Cost(\theta) = -\frac{1}{M}\sum_{m=1}^{M}\left[y^m \log h\left(x^{(m)};\theta\right) - (1-y^m)\log(1-h\left(x^{(m)};\theta\right))\right] + \frac{\lambda}{2M}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}[\theta_{ij}{}^{(l)}]^2$$

- The very nonlinear nature of $h\left(x^{(m)};\theta\right)$ complicates the task of calibrating the parameters θ:
  - There can be thousands of partial derivatives (esp. true for deep learning)
  - There can be multiple local minima, making the numerical solution dependent on initial conditions

Forward propagation



- We start with a given set of weights $\theta = \begin{bmatrix} \theta^{(1)}(:) \\ \theta^{(2)}(:) \\ \vdots \\ \theta^{(L-1)}(:) \end{bmatrix}$ and an input $a^{(1)} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$
- We then propagate the values all along the network:

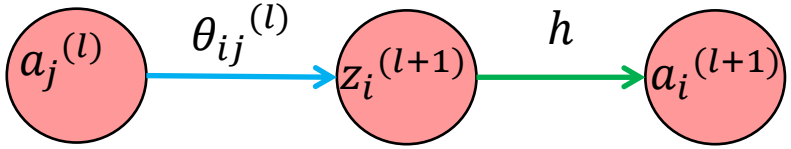
$$\boldsymbol{z^{(l+1)} = \theta^{(l)} a^{(l)}} \quad \ldots (1)$$

$$\boldsymbol{a^{(l+1)} = h\left(z^{(l+1)}\right)} \quad \ldots (2)$$

- We end with the output $\hat{y}$ of the network, which we can compare to the real output y:

$$\boldsymbol{\hat{y} = a^{(L)}} \quad \ldots (3)$$

Backward propagation



- From forward propagation we have:

$$z^{(l+1)} = \theta^{(l)} a^{(l)}$$

$$z_i^{(l+1)} = \sum_{j=1}^{s_l} \theta_{ij}^{(l)} a_j^{(l)}$$

$$a^{(l)} = h(z^{(l)})$$

$$a_i^{(l)} = h(z_i^{(l)})$$

- By the chain rule

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} Cost(\theta) = \frac{\partial}{\partial z_i^{(l+1)}} Cost(\theta) \frac{\partial z_i^{(l+1)}}{\partial \theta_{ij}^{(l)}}$$

- Define

$$\boldsymbol{\delta_i^{(l+1)} := \frac{\partial}{\partial z_i^{(l+1)}} Cost(\theta) \dots (4)}$$

- Therefore

$$\boldsymbol{\frac{\partial}{\partial \theta_{ij}^{(l)}} Cost(\theta) = \delta_i^{(l+1)} a_j^{(l)} \dots (5)}$$

- We will compute the deltas backwards, starting from $l = L$.

Delta for final layer L

- In the case of a **single example** $(\boldsymbol{x}, \boldsymbol{y})$ we have

$$Cost(\theta) = -\left[y \log h(\theta^{(L-1)} a^{(L-1)}) - (1-y) \log(1 - h(\theta^{(L-1)} a^{(L-1)}))\right]$$

- Its partial derivative is

$$\frac{\partial}{\partial {\theta_{1j}}^{(L-1)}} Cost(\theta) = -\left[y \frac{\partial}{\partial {\theta_{1j}}^{(L-1)}} \log h(\theta^{(L-1)} a^{(L-1)}) - (1-y) \frac{\partial}{\partial {\theta_{1j}}^{(L-1)}} \log(1 - h(\theta^{(L-1)} a^{(L-1)}))\right]$$

- Let us compute:

$$\frac{\partial}{\partial {\theta_{1j}}^{(L-1)}} \log h(\theta^{(L-1)} a^{(L-1)}) = \frac{h'}{h} {a_j}^{(L-1)}$$

$$\frac{\partial}{\partial {\theta_{1j}}^{(L-1)}} \log(1 - h(\theta^{(L-1)} a^{(L-1)}) = = \frac{h'}{1-h} {a_j}^{(L-1)}$$

- Using $h' = h(1-h)$ and simplifying we obtain

$$\frac{\partial}{\partial {\theta_{1j}}^{(L-1)}} Cost(\theta) = \left[h(\theta^{(L-1)} a^{(L-1)}) - y\right] {a_j}^{(L-1)}$$

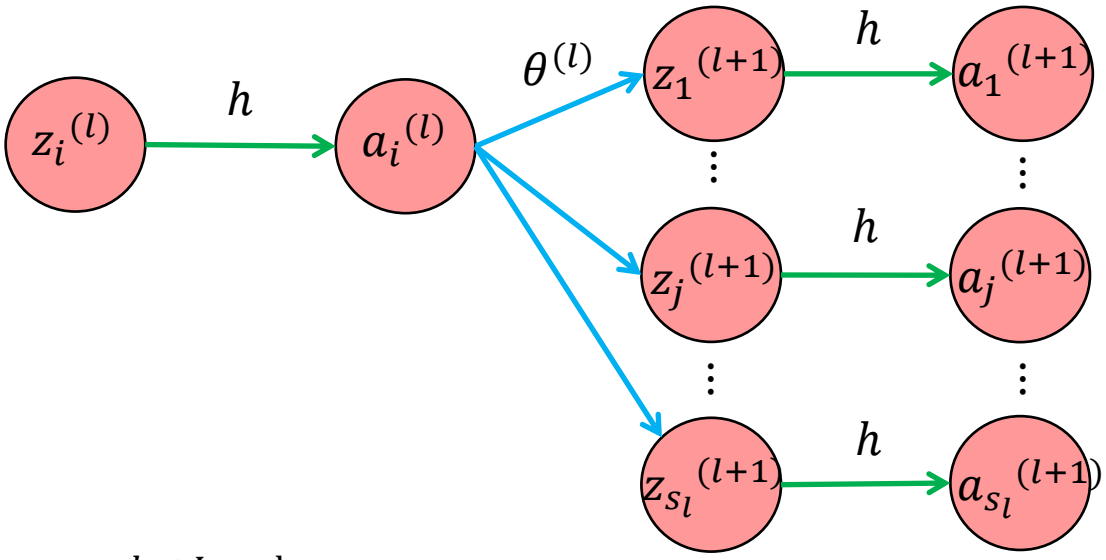
- Define

$$\boldsymbol{{\delta_1}^{(L)} = {a_1}^{(L)} - y} \ \ldots (6)$$

- Using $a^{(L)} = h(\theta^{(L-1)} a^{(L-1)})$ we obtain

$$\boldsymbol{\frac{\partial}{\partial {\theta_{1j}}^{(L-1)}} Cost(\theta) = {\delta_1}^{(L)} {a_j}^{(L-1)}} \ \ldots (7)$$

Deltas for general layers



- From forward propagation we have:

$$z_j^{(l+1)} = \sum_{i=1}^{s_l} \theta_{ji}^{(l)} a_i^{(l)}$$

$$a_i^{(l)} = h(z_i^{(l)})$$

- For $l < L$ we have

$$\frac{\partial}{\partial z_i^{(l)}} Cost(\theta) = \sum_{j=1}^{s_l} \frac{\partial Cost(\theta)}{\partial z_j^{(l+1)}} \times \frac{\partial z_j^{(l+1)}}{\partial a_i^{(l)}} \times \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} = \sum_{j=1}^{s_l} \delta_j^{(l+1)} \times \theta_{ji}^{(l)} \times h'(z_i^{(l)})$$

- Therefore

$$\boldsymbol{\delta_i^{(l)} = h'(z_i^{(l)}) \sum_{j=1}^{s_l} \delta_j^{(l+1)} \theta_{ji}^{(l)} \dots (8)}$$

- In vectorial form, using the Matlab operator ".* " for element-wise multiplication, (8) becomes

$$\boldsymbol{\delta^{(l)} = h'(z^{(l)}) .* (\theta^{(l)})^T \delta^{(l+1)} \dots (9)}$$

Learning algorithm

- Start with a training set $(x^1, y^1), \dots, (x^m, y^m), \dots (x^M, y^M)$
- Randomly **initialise** weights θ
  - Initialise gradients ${\Delta_{ij}}^{(l)} = 0$ for all layers $l = 1, \dots, L$
  - **Loop**: for every training example (x^m, y^m)
    - ✓ Forward propagation to compute $a^{(l)}$ for all layers $l = 1, \dots, L$
    - ✓ Backward propagation to compute $\delta^{(l)}$ for all layers $l = 2, \dots, L$
    - ✓ Update gradients: ${\Delta_{ij}}^{(l)} = {\Delta_{ij}}^{(l)} + \frac{1}{M} {\delta_i}^{(l+1)} {a_j}^{(l)}$
  - Add regularisation (optional but recommended)
    - ✓ ${\Delta_{ij}}^{(l)} = {\Delta_{ij}}^{(l)} + \lambda {\theta_{ij}}^{(l)}$ if $j \neq 0$
  - **Update** θ via gradient descent
    - ✓ ${\theta_{ij}}^{(l)} = {\theta_{ij}}^{(l)} - \alpha {\Delta_{ij}}^{(l)}$
  - Repeat until convergence
- Repeat for more random initialisations
  - Keep the value of θ such that the cost function is the smallest amongst all random initialisations

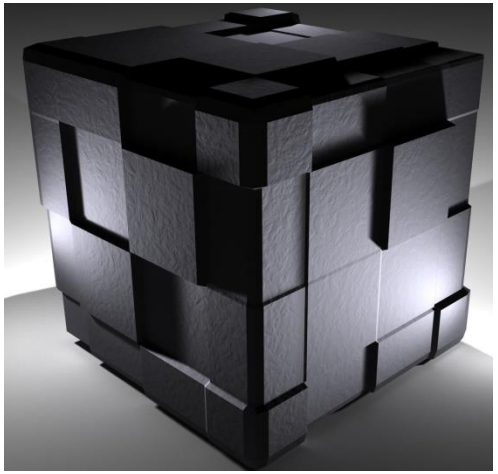
www.gettyimages.com

5. Neural Networks
Example of HFT price prediction

Neural Networks are black boxes!

- More hidden layers
  - More complicated nonlinear transformations
  - More difficult to interpret the effect of each parameter in the outcome
- You do not fully understand what is going on
  - You cannot explain the "nonlinear factors" the model is picking
  - Like Forrest Gump's boxes of chocolates:
    - "You never know what you are going to get"
  - It is difficult to audit a neural network model
- You normally hit a local minimum, not the global minimum
  - The local minimum depends on the initial conditions
  - Several random initialisations are needed to get a better local minima
- But the black box approach works!
  - Neural networks have shown their worth in multiple applications
  - Deep learning is impressive
    - Even if its Black Box is actually "darker than black"

wallpaperlayer.com

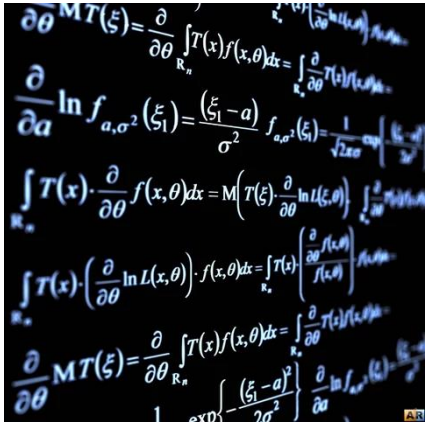


ih1.redbubble.net

Neural Networks and HFT

- Justin Sirignano, "Deep Learning for Limit Order Books", preprint ArXiv 2016

- In a **Model-driven approach** we decide the equations and relations governing the dynamics of the system
  - Market orders follow a Poisson process (no-memory) or a Hawkes processes (memory)
  - The Limit Order Book (LOB) replenishes itself at a certain rate
  - The mid-price follows a stochastic process e.g. Ito process, Levy process



powerlisting.wikia.com

- In a **Data-driven approach** there are no assumptions on the dynamics of the system
  - Conditional probability of future prices given the current state of the LOB
    - ✓ Given the volumes and prices of several levels of the LOB at time t
    - ✓ Predict the future best bid and ask prices at time $t + h$

swisscognitive.ch

Discrete LOB model

- The LOB model
  - The paper builds a joint distribution of best bid and ask prices at time $t + h$
  - Conditioned on the state of the LOB at time t
    - ✓ 10 bid and ask levels of the LOB, including best bid and ask prices

www.paneuropeannetworks.com

- Adding spacial distribution to neural networks
  - Standard neural networks have outputs on a finite set
  - The paper extends this to an infinite, discrete output set e.g. Z

- Adding "closeness" in the LOB
  - Normally, all samples in a neural network are independent
  - But in reality, if two samples are "close" then they should behave "similarly"
  - The spacial neural networks allows to define a probability notion of "closeness"

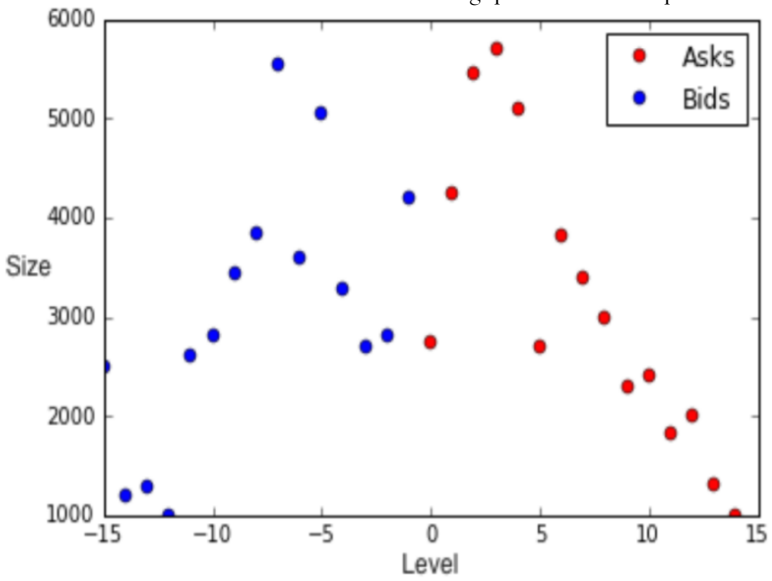
www.pinterest.es

- The bid-ask spread is not constant i.e. the bid and ask prices do not move together "in lockstep"
  - For the majority of the NASDAQ stocks, more than 50% time there is no lockstep move
  - For half of the NASDAQ stocks, they move in locksteps only of the17% of the time

Hypotheses

- Infinite price levels for the LOB
  - Price levels in Z (zero is best ask price)
  - For classical neural networks it has to be capped e.g. $[-50,50]$

arxiv.org/pdf/1601.01987.pdf

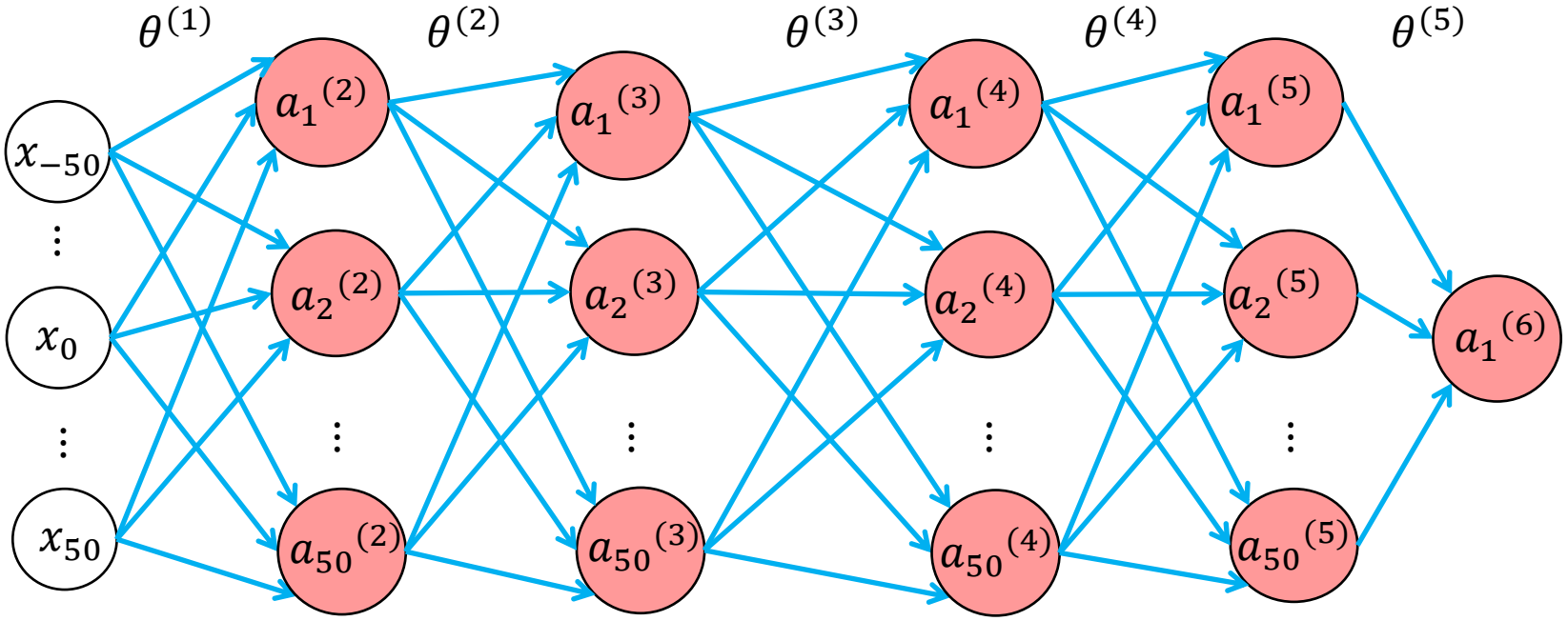


- Bid and ask price distributions
  - Some research consider them separately and independent
  - But then they need to add a variable for the spread
    - ✓ Constant or mean-reverting
  - This paper models bid and ask prices together

- Fixed time horizon of $h = 1$ second
  - NASDAQ is open from 9:30 to 16:00
  - 6.5 hours → 23,400 seconds
  - This is enough to calibrate the model on a daily basis
  - In the paper the training sample is 500 NASDAQ stocks over 20 months in 2014-2015

- A 1-second forecast of price moves qualifies as HFT model

Results



- Neural networks outperform nonlinear logistic regressions
  - Neural networks have lower out-of-sample error
- The spacial neural network outperforms standard neural networks
  - And there are less parameters due to its local spacial structure (170,000 vs 20,000)

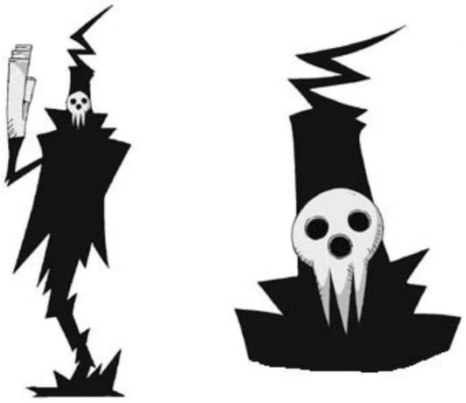
5. Conclusions

We do not fear ML jargon anymore

- Data Science
- Machine Learning
- Cost function
- Linear regression
- Training a model
- Cross-validation
- Test set
- Overfitting
- Bias
- Variance
- Regularisation

- Logistic regression
- Classification boundary
- Accuracy
- Confidence Interval
- Precision
- Recall
- F1 score
- Single class
- Multi class
- One vs All
- Gradient descent
- Learning rate

- Logistic unit
- Nonlinear features
- Neural Networks
- Weights
- Hidden layer
- Deep Learning
- Forward propagation
- Backward propagation
- Non-convex optimisation
- Random initialisation
- Data-driven approach



souleater.wikia.com

References on ML (updated)

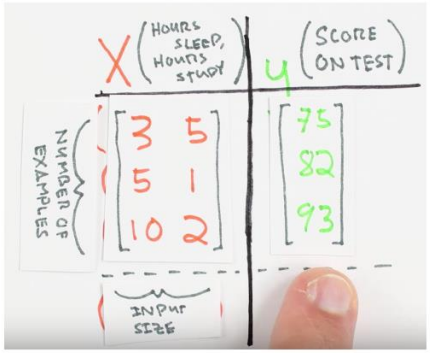
Machine Learning

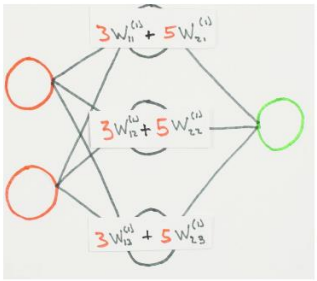
- Stanford
  - Coursera MOOC
  - Machine Learning
  - Andrew Ng

Books on **Statistical Learning**

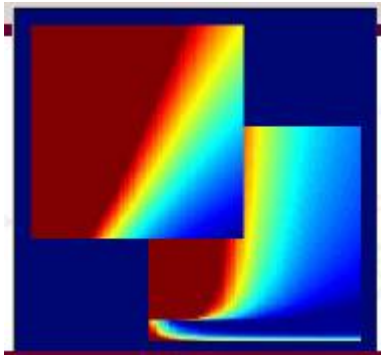
- Hastie, Tibshirani *et al* "Elements of Statistical Learning" (Data mining, Inference, Prediction)
- Hastie, Tibshirani *et al* "An introduction to Statistical Learning" (with applications to R)

There is a MOOC associated to them as well





- YouTube videos
  - Neural Networks Demystified
  - Welch Labs



- CalTech
  - Online lectures
  - Machine Learning
  - Yaser Abu-Mostafa
  - Book: *"Learning from Data"*

Thank you for your attention

KEEP
CALM
PRESENTATION IS OVER
ANY
QUESTIONS?