

COMP5611M: Machine Learning

Introduction to the Module

Nishant Ravikumar
Marc de Kamps

School of Computing

January 24, 2022

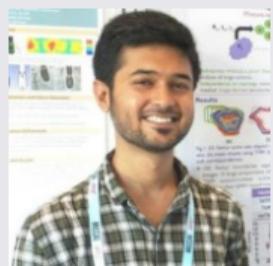


UNIVERSITY OF LEEDS

Introduction to the Module

Who are we?

Nishant Ravikumar



Lecturer in Computer Science.

Research background: Medical Image Computing.

Marc de Kamps



Lecturer in Computer Science.

Research background: Computational Neuroscience; Cognitive Modelling.

Lecture Objectives

Objectives for this lecture:

- Introduction to the module staff
- Setting out the module prerequisites
- Description of module format
- Broad description of what is and isn't in the module

Lecture Outcomes

At the end of this lecture, you should

- Know where to find the teaching materials
- Know who can be contacted for help
- Have a broad idea of what we consider to be machine learning

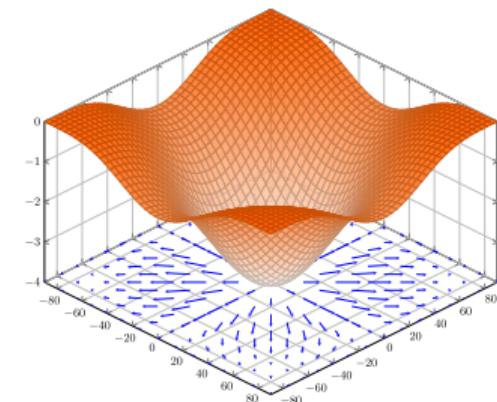
What are the Prerequisites?

Prerequisites

The Module is designed for Computer Science students with little previous exposure to machine learning. We expect a general computer science background.

In particular you should:

- be familiar with matrices, vectors and their symbolic manipulations
- be able to use *numpy* to implement matrix/vector calculations in Python at the level of **Programming for Data Science**
- be able to differentiate and use the product and chain rules confidently
- differentiate multivariate functions



Source: Wikipedia;

Creative Commons CC0 1.0 Universal Public

Domain Dedication

I don't have that!

Am I doomed?

What can I do to remedy deficiencies?

- It is very hard to do a machine learning course without at least a basic understanding of matrices and vectors.
- If you are not very confident in differentiation, this is less of a problem. There are plenty of practice sites for differentiation.
- If you are good at differentiation, but less experienced in multivariate differentiation, you're probably in decent shape. Operationally, they are very similar and you mostly need the geometrical interpretation
- We will (quickly) cover basic statistics

Delivery

Module consists of

- Twenty lectures, delivered synchronously online
- Some optional background
- Synchronously, so opportunity for asking questions!
- Labwork exercises
- Only labwork, no 'tutorials'

Assessment

- One coursework, summative 60 % of total mark, set week 22 (21-25 March), due date week 24 (2-6 May)
- One exam, online during the examination period (see time table)

Delivery

Material suitable for online study

- However, we welcome you to visit the labs in person
- Nishant and Marc will aim to come into the labs
- We will always have someone up to speed with the modules during the lab sessions
- We will also run a virtual lab session; aiming for the same times as physical labs, we are aware that some people are abroad, vulnerable, or studying remotely
- If someone's really struggling, we can organise a limited number of one-on-ones with TAs

What is Machine Learning Anyway?

There is considerable conflation of statistics, machine learning, data analytics artificial intelligence ...

Machine Learning

Machine Learning is the study of computer algorithms that improve automatically through experience. (Mitchell, 1997)

- Not statistics, but related to it and heavily dependent on it
- Not data analytics, but it makes no sense to develop algorithms without an end user in mind.

What is module about?

Machine Learning ...

What topics are covered?

- Unit 1: Review of probability
- Unit 2: Linear Regression, Bayesian and frequentists versions
- Unit 3: Logistic Regression and Introduction to Neural Networks
- Unit 4: Decision Trees, Random Forests, Boosting
- Unit 5: Mixture Models, E/M Algorithm, Variational Methods

When time permits

Topics Not Covered

The following topics are not covered:

- Deep Learning (separate module)
- Reinforcement Learning (separate module in development)
- Genetic Algorithms (but see COMP5400M)

We considered the following topics, but considering the module load we omit

- Bayesian Networks and Sampling Methods
- Support Vector Machines and Kernel Based Methods

On the last two topics material is available for self-study

Why These Topics?

Generative Models

MNIST, dataset handwritten numerals:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

Idea

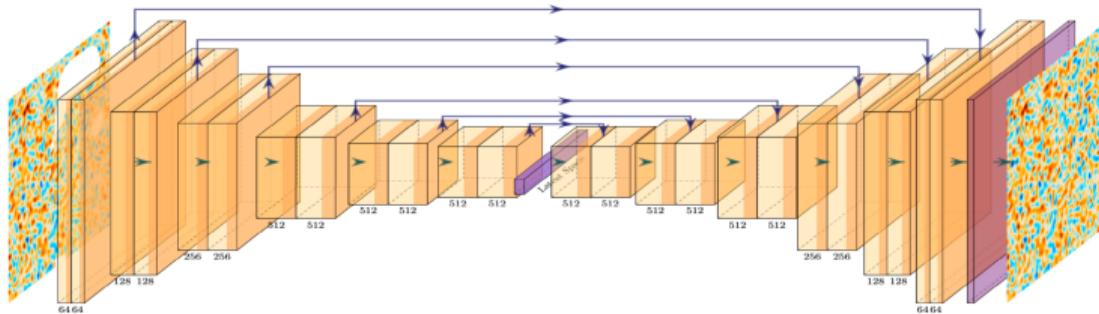
Can we learn to reduce images to their 'essentials' ?

Test:

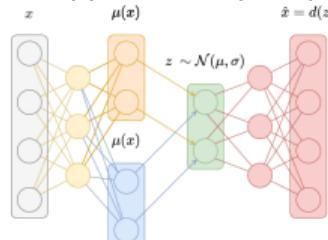
- Synthetically recreate images from 'essentials',
- Look similar? We may have captured elements of the data generation process

Generative Models

Example: Variational Autoencoder (VAE)

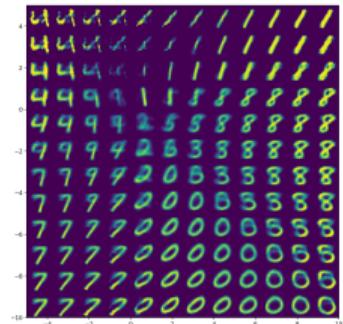


Yi et al, <https://arxiv.org/abs/2001.11651>

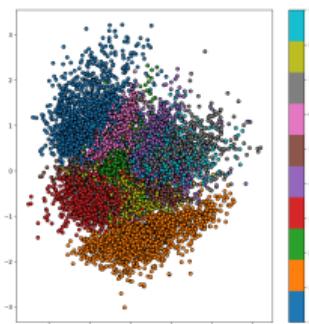
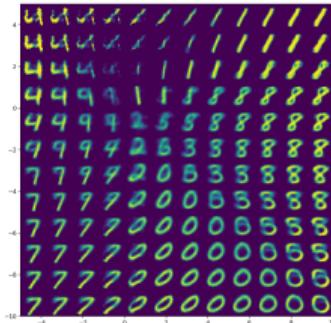


<https://github.com/avandekleut/avandekleut.github.io>

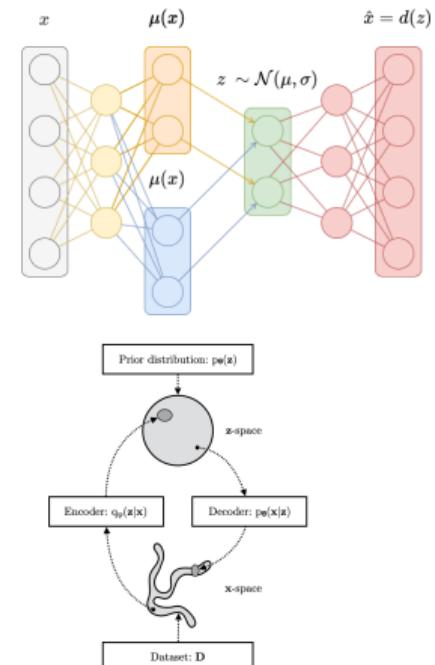
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9



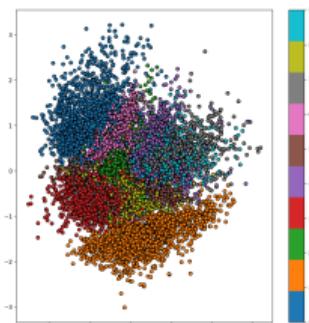
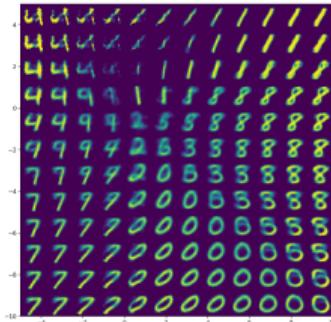
Some Really Bold Ideas Here!



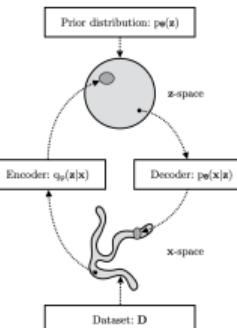
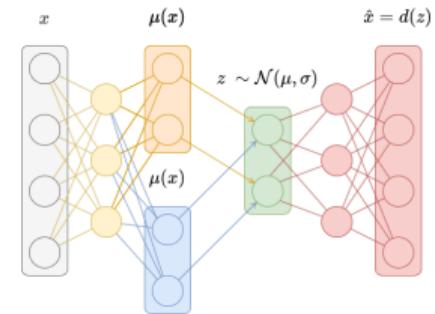
- \mathcal{I} : space of all possible images. **Massive!**
- Similar images (cows, cars, etc.) form a surface in this massive space
- Submanifold: parameterise by a small number of parameters
- Parameterise submanifold: small number of parameters!
- More conventionally known as **latent space**



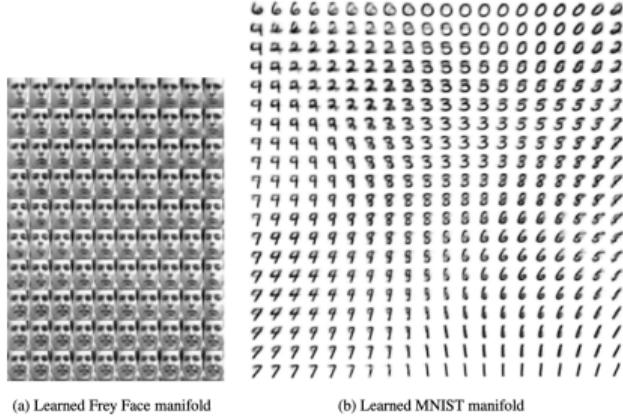
Some Really Bold Ideas Here!



- Represent the latent space as a mixture of Gaussian distributions
- **Encoder** learns a mapping
- Mapping from data space x to the parameters of Gaussian
- **Decoder** learns mapping from latent space to original image



Why?



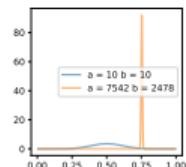
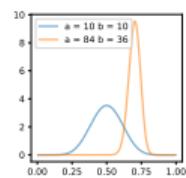
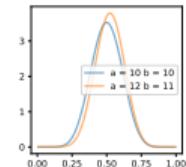
Kingma & Welling, <https://arxiv.org/pdf/1312.6114.pdf>

- Compression
- Does the latent space preserve structure of data?
- Looks like it!
- Sample latent space and use the decoder to generate new images
- Investigate structure of latent space

Interesting but it went over my head!

No worries! In Unit 1 we will review elementary concepts of probability:

- Probabilities; prob distributions; state space
- Ubiquitous distributions: Bernoulli, Binomial, Normal
- Maximum Likelihood Estimates
- Multivariate distributions; joint; marginal; conditional
- Bayes law (maybe less familiar)
- Some Information Theory: *cross-entropy*

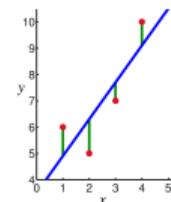


We estimate whether a coin is fair or not. We start with the assumption the coin is fair $p = 0.5$, but gathering evidence, we find we're playing a cheater! At all times our estimate of the coin value remains a distribution: model uncertainty.

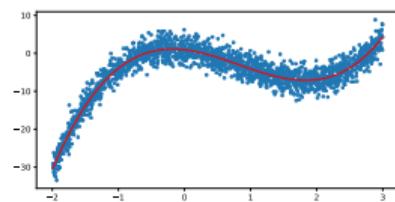
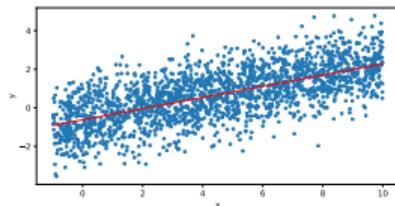
Unit 2: Linear Regression

Maximum Likelihood and Bayesian

- May be familiar ..
- Ram a line through a cloud of data points
- Well known solution: Ordinary Least Squares
- 'Line' can be the graph of non-linear function
- But the fit expression is linear in the weights (parameters)



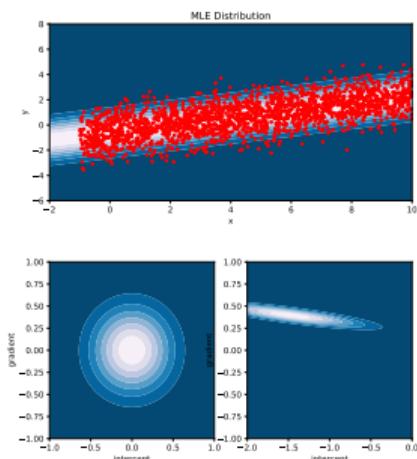
Source: Wikimedia, CC BY



Unit 2: Linear Regression

Maximum Likelihood and Bayesian

- Often a model can be understood as $y = f(x) + \epsilon$
- $f(x)$ is a function modelling a *deterministic* part of some process
- ϵ indicates *additive Gaussian* noise
- Maximum likelihood estimation is equivalent to ordinary least squares here
- Since we have a likelihood, with appropriate prior we can find posterior distribution
- Model parameters not a single point (MLE) but a distribution! *Model Uncertainty*
- No overfitting; no cross validation; regularisation

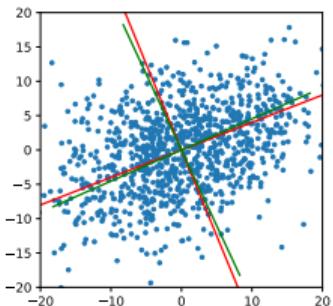
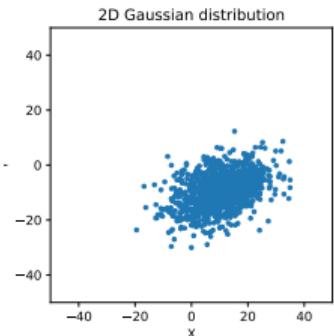


Unit 2: Linear Regression

Multivariate Gaussian

To do this we need:

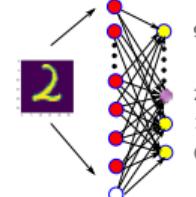
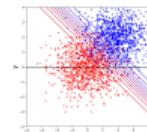
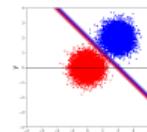
- Good understanding of multivariate Gaussian distribution is essential!
- μ : centre of cloud
- Eigenvectors of Σ : main 'directions' of the data
- Estimation of these parameters
- Creating synthetic datasets



Unit 3: Logistic Regression and Neural Networks

To do this we need:

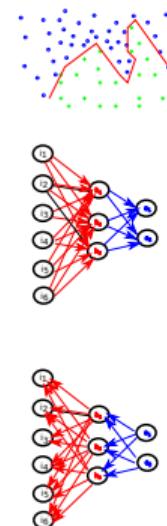
- Ideas from biological neural networks: perceptron
- Graded perceptron: *logistic regression*. Simplest neural network!
- Generative model for classification: also logistic regression!
- Neural networks are also regressors but not polynomial. They learn their own basis functions!
- Deep neural networks, e.g. multi-layer perceptron mitigate against the *curse of dimensionality*



Unit 3: Logistic Regression and Neural Networks

To do this we need:

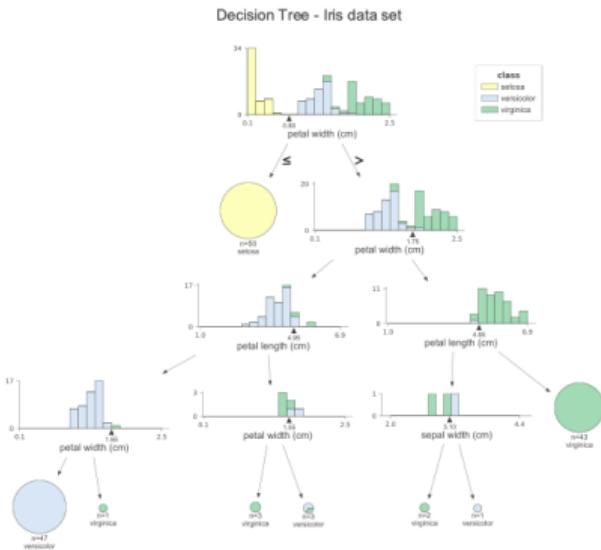
- Deep learning depends on a very large number of parameters
- Learning them could be very difficult ...
- ... but in layered network the *backpropagation* algorithm keeps the complexity under control
- Backpropagation is a form of *steepest gradient descent*, a numerical optimisation for minimising *loss functions*
- We will learn to calculate gradients by hand, but will see that PyTorch's *autograd* facility makes gradient calculations painless
- **Bayesian logistic regression is quite hard!** : options: Laplace approximation; variational inference; Monte Carlo sampling (not this module)



Unit 4: Decision Trees and Random Forests

To do this we need:

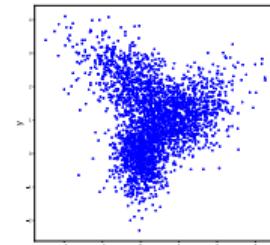
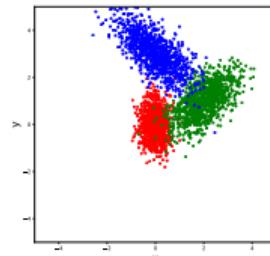
- Decision Trees are slightly off course in this module, but too important to ignore
- Capable of non-linear regression and classification
- Easy to interpret, but very prone to overfitting
- Bootstrapping: random forests and
- Boosting make for highly competitive algorithms:
XGBoost comes out on top in several competitions



Unit 5: Mixture Models and E/M Algorithms

The Origins of Variational Inference

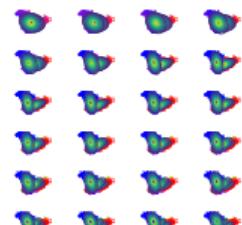
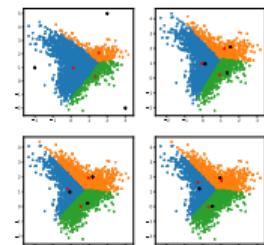
- Gaussian mixtures are often used to model multi-modal distributions
- You have to solve two problems simultaneously
 - Find means and covariances of the clusters
 - Infer to which cluster a data point belongs
- When the cluster labels are given, this is straightforward (Unit 2)
- When not the algorithm itself must identify clusters
- This is an example of unsupervised learning: the unlabelled data does not give any hint how to do this. The algorithm must do this by itself
- This is unlike most forms of classification and regression, where *training sets* are required: data points *and* desired output are fed into the algorithm



Unit 5: Mixture Models and E/M Algorithms

The Origins of Variational Inference

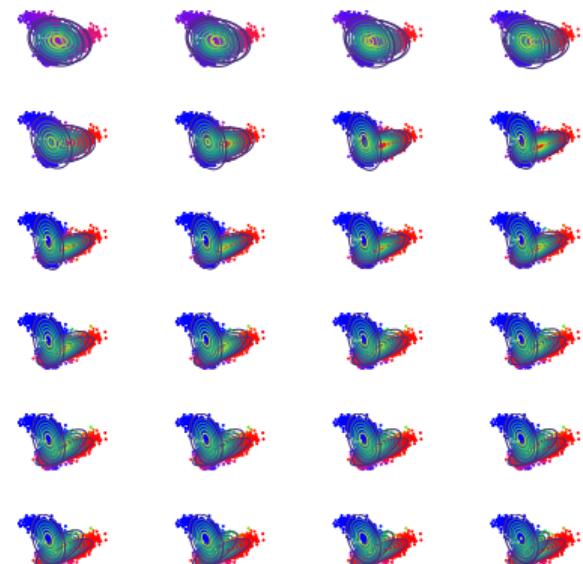
- Assigning a data point to a cluster does not work well
- A probabilistic assignment works better
- This requires a *latent variable*, a stochastic variable that controls cluster assignment
- This variable is *unobserved*, but must be inferred through indirect methods
- Application of Bayes' law
- E/M Algorithm: two stages
 - Estimate the means and covariances based on the latent variables
 - Update the latent variables
- This algorithm requires optimising the Evidence Lower BOund (ELBO)
- These ideas lead to VAEs in a way that we will try to explain in Unit 5



Unit 5: Mixture Models and E/M Algorithms

The Origins of Variational Inference

- The Gaussian mixture can achieve enormous data reduction
- Regularisation of components: Bayes!
- We really would like to have a prior distribution on the mixing components
- Aim: Infer posterior of the latent distribution
- Analytic treatment: too hard; solution *variational inference*: assume independence of latent space variables



Practical Work

- Practical work is organised per unit
- Combination of notebooks and worksheets
- You will develop a full machine learning pipeline using *scikit-learn*
- Most algorithms will be exemplified in notebooks in Python
- Crash course notebooks in *numpy* and *pandas* are available
- Please make use of TA assistance!

Recap

- Today: Roadmap for the module, selection of topics
- Organisation of the practicalities
- Next lecture: review of probability with emphasis on Bayes' law

COMP5611M: Machine Learning

Introduction to the Module

Nishant Ravikumar
Marc de Kamps

School of Computing

January 24, 2022

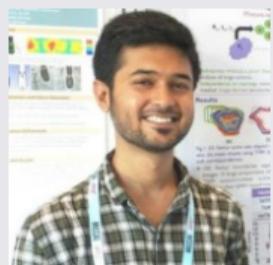


UNIVERSITY OF LEEDS

Introduction to the Module

Who are we?

Nishant Ravikumar



Lecturer in Computer Science.

Research background: Medical Image Computing.

Marc de Kamps



Lecturer in Computer Science.

Research background: Computational Neuroscience; Cognitive Modelling.

Lecture Objectives

Objectives for this lecture:

- Introduction to the module staff
- Setting out the module prerequisites
- Description of module format
- Broad description of what is and isn't in the module

Lecture Outcomes

At the end of this lecture, you should

- Know where to find the teaching materials
- Know who can be contacted for help
- Have a broad idea of what we consider to be machine learning

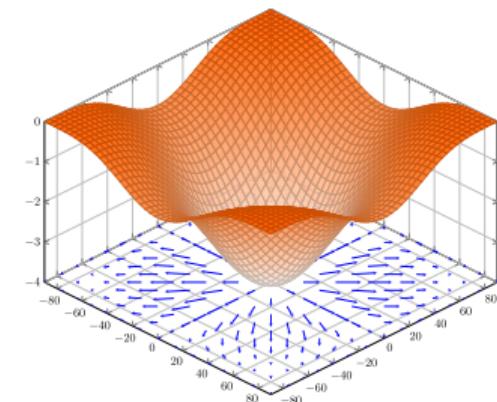
What are the Prerequisites?

Prerequisites

The Module is designed for Computer Science students with little previous exposure to machine learning. We expect a general computer science background.

In particular you should:

- be familiar with matrices, vectors and their symbolic manipulations
- be able to use *numpy* to implement matrix/vector calculations in Python at the level of **Programming for Data Science**
- be able to differentiate and use the product and chain rules confidently
- differentiate multivariate functions



Source: Wikipedia;
Creative Commons CC0 1.0 Universal Public
Domain Dedication

I don't have that!

Am I doomed?

What can I do to remedy deficiencies?

- It is very hard to do a machine learning course without at least a basic understanding of matrices and vectors.
- If you are not very confident in differentiation, this is less of a problem. There are plenty of practice sites for differentiation.
- If you are good at differentiation, but less experienced in multivariate differentiation, you're probably in decent shape. Operationally, they are very similar and you mostly need the geometrical interpretation
- We will (quickly) cover basic statistics

Delivery

Module consists of

- Twenty lectures, delivered synchronously online
- Some optional background
- Synchronously, so opportunity for asking questions!
- Labwork exercises
- Only labwork, no 'tutorials'

Assessment

- One coursework, summative 60 % of total mark, set week 22 (21-25 March), due date week 24 (2-6 May)
- One exam, online during the examination period (see time table)

Delivery

Material suitable for online study

- However, we welcome you to visit the labs in person
- Nishant and Marc will aim to come into the labs
- We will always have someone up to speed with the modules during the lab sessions
- We will also run a virtual lab session; aiming for the same times as physical labs, we are aware that some people are abroad, vulnerable, or studying remotely
- If someone's really struggling, we can organise a limited number of one-on-ones with TAs

What is Machine Learning Anyway?

There is considerable conflation of statistics, machine learning, data analytics artificial intelligence ...

Machine Learning

Machine Learning is the study of computer algorithms that improve automatically through experience. (Mitchell, 1997)

- Not statistics, but related to it and heavily dependent on it
- Not data analytics, but it makes no sense to develop algorithms without an end user in mind.

What is module about?

Machine Learning ...

What topics are covered?

- Unit 1: Review of probability
- Unit 2: Linear Regression, Bayesian and frequentists versions
- Unit 3: Logistic Regression and Introduction to Neural Networks
- Unit 4: Decision Trees, Random Forests, Boosting
- Unit 5: Mixture Models, E/M Algorithm, Variational Methods

When time permits

Topics Not Covered

The following topics are not covered:

- Deep Learning (separate module)
- Reinforcement Learning (separate module in development)
- Genetic Algorithms (but see COMP5400M)

We considered the following topics, but considering the module load we omit

- Bayesian Networks and Sampling Methods
- Support Vector Machines and Kernel Based Methods

On the last two topics material is available for self-study

Why These Topics?

Generative Models

MNIST, dataset handwritten numerals:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

Idea

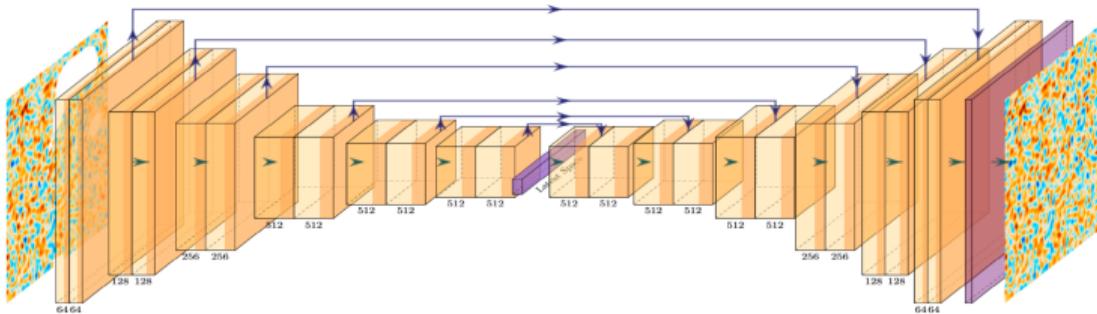
Can we learn to reduce images to their 'essentials' ?

Test:

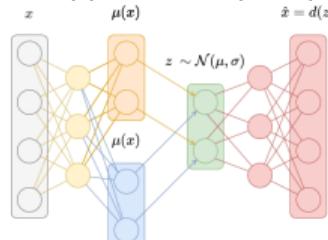
- Synthetically recreate images from 'essentials',
- Look similar? We may have captured elements of the data generation process

Generative Models

Example: Variational Autoencoder (VAE)

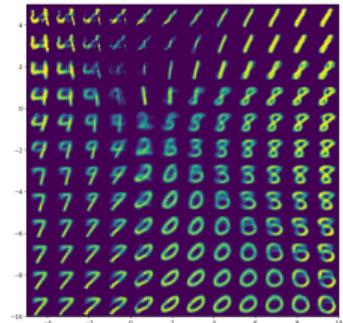


Yi et al, <https://arxiv.org/abs/2001.11651>

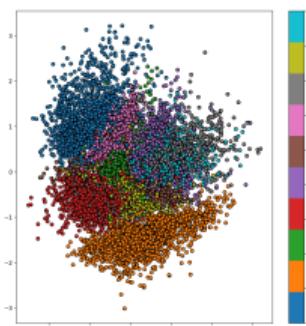
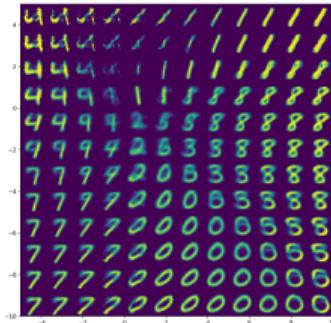


<https://github.com/avandekleut/avandekleut.github.io>

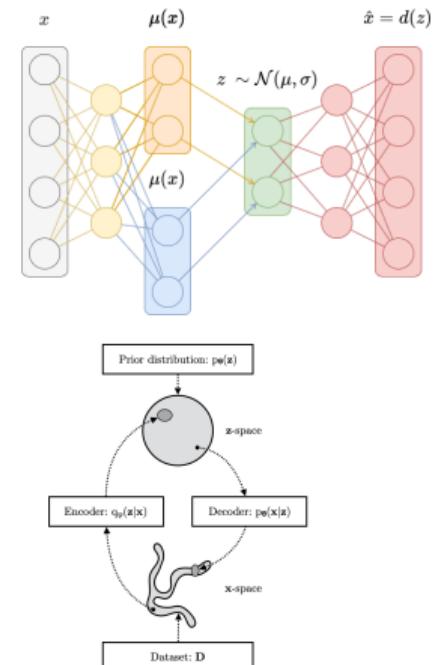
0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9



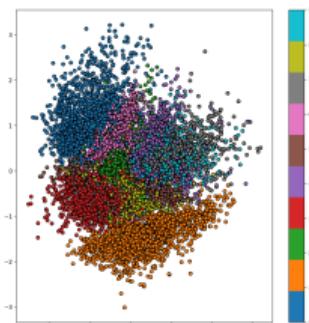
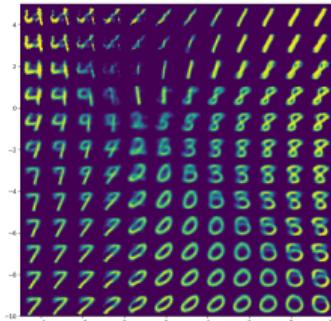
Some Really Bold Ideas Here!



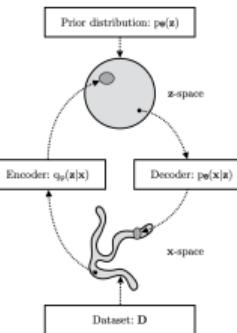
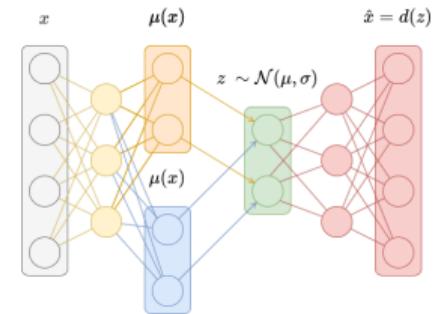
- \mathcal{I} : space of all possible images. **Massive!**
- Similar images (cows, cars, etc.) form a surface in this massive space
- Submanifold: parameterise by a small number of parameters
- Parameterise submanifold: small number of parameters!
- More conventionally known as **latent space**



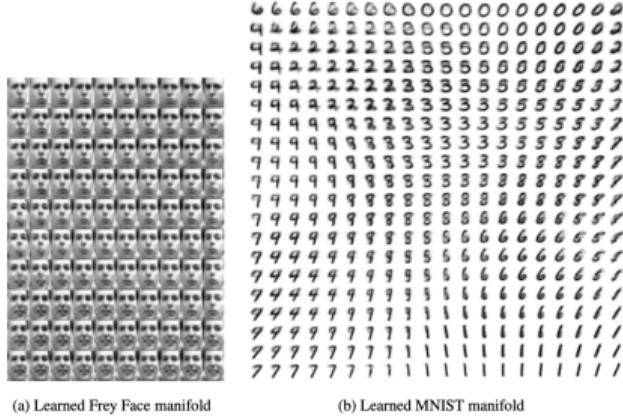
Some Really Bold Ideas Here!



- Represent the latent space as a mixture of Gaussian distributions
- **Encoder** learns a mapping
- Mapping from data space x to the parameters of Gaussian
- **Decoder** learns mapping from latent space to original image



Why?



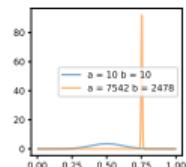
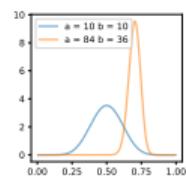
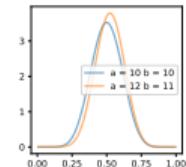
Kingma & Welling, <https://arxiv.org/pdf/1312.6114.pdf>

- Compression
- Does the latent space preserve structure of data?
- Looks like it!
- Sample latent space and use the decoder to generate new images
- Investigate structure of latent space

Interesting but it went over my head!

No worries! In Unit 1 we will review elementary concepts of probability:

- Probabilities; prob distributions; state space
- Ubiquitous distributions: Bernoulli, Binomial, Normal
- Maximum Likelihood Estimates
- Multivariate distributions; joint; marginal; conditional
- Bayes law (maybe less familiar)
- Some Information Theory: *cross-entropy*

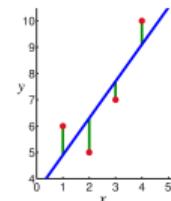


We estimate whether a coin is fair or not. We start with the assumption the coin is fair $p = 0.5$, but gathering evidence, we find we're playing a cheater! At all times our estimate of the coin value remains a distribution: model uncertainty.

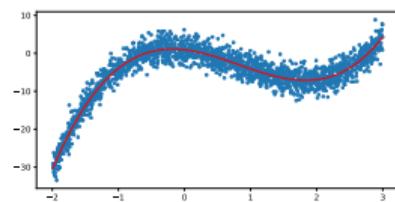
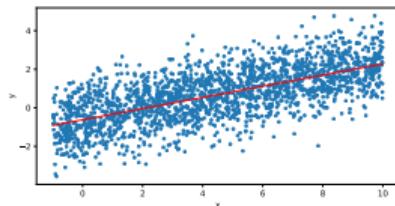
Unit 2: Linear Regression

Maximum Likelihood and Bayesian

- May be familiar ..
- Ram a line through a cloud of data points
- Well known solution: Ordinary Least Squares
- 'Line' can be the graph of non-linear function
- But the fit expression is linear in the weights (parameters)



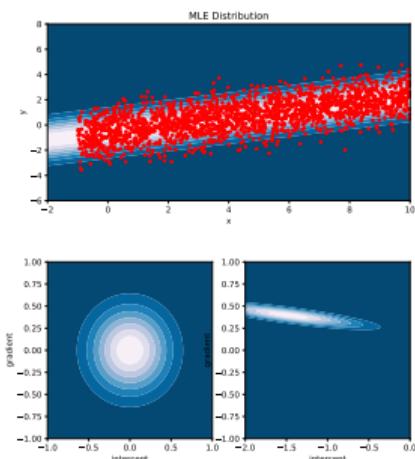
Source: Wikimedia, CC BY



Unit 2: Linear Regression

Maximum Likelihood and Bayesian

- Often a model can be understood as $y = f(x) + \epsilon$
- $f(x)$ is a function modelling a *deterministic* part of some process
- ϵ indicates *additive Gaussian* noise
- Maximum likelihood estimation is equivalent to ordinary least squares here
- Since we have a likelihood, with appropriate prior we can find posterior distribution
- Model parameters not a single point (MLE) but a distribution! *Model Uncertainty*
- No overfitting; no cross validation; regularisation

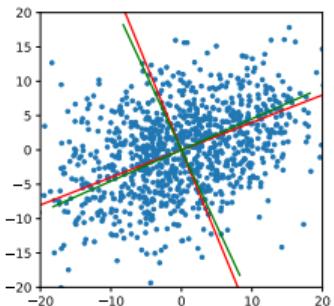
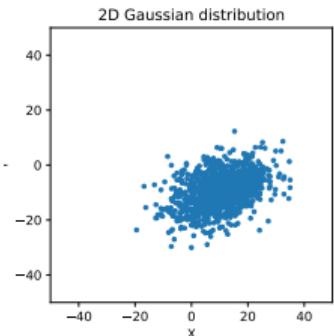


Unit 2: Linear Regression

Multivariate Gaussian

To do this we need:

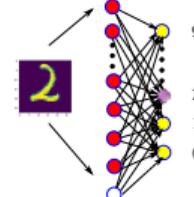
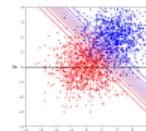
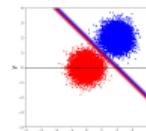
- Good understanding of multivariate Gaussian distribution is essential!
- μ : centre of cloud
- Eigenvectors of Σ : main 'directions' of the data
- Estimation of these parameters
- Creating synthetic datasets



Unit 3: Logistic Regression and Neural Networks

To do this we need:

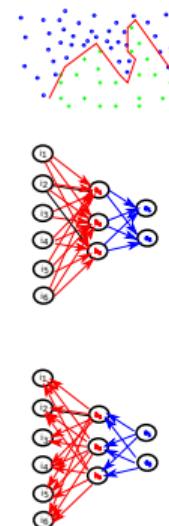
- Ideas from biological neural networks: perceptron
- Graded perceptron: *logistic regression*. Simplest neural network!
- Generative model for classification: also logistic regression!
- Neural networks are also regressors but not polynomial. They learn their own basis functions!
- Deep neural networks, e.g. multi-layer perceptron mitigate against the *curse of dimensionality*



Unit 3: Logistic Regression and Neural Networks

To do this we need:

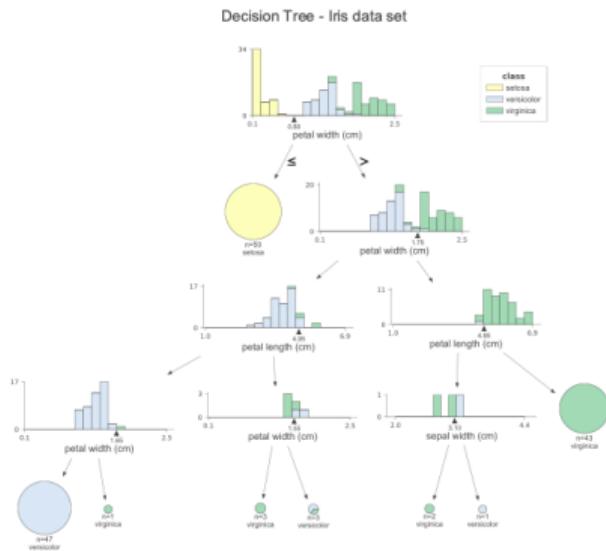
- Deep learning depends on a very large number of parameters
- Learning them could be very difficult ...
- ... but in layered network the *backpropagation* algorithm keeps the complexity under control
- Backpropagation is a form of *steepest gradient descent*, a numerical optimisation for minimising *loss functions*
- We will learn to calculate gradients by hand, but will see that PyTorch's *autograd* facility makes gradient calculations painless
- **Bayesian logistic regression is quite hard!** : options: Laplace approximation; variational inference; Monte Carlo sampling (not this module)



Unit 4: Decision Trees and Random Forests

To do this we need:

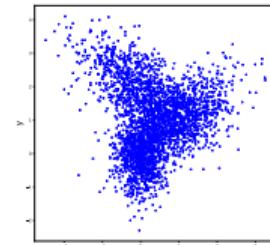
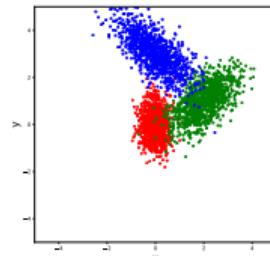
- Decision Trees are slightly off course in this module but too important to ignore
 - Capable of non-linear regression and classification
 - Easy to interpret, but very prone to overfitting
 - Bootstrapping: random forests and
 - Boosting make for highly competitive algorithms:
XGBoost comes out on top in several competitions



Unit 5: Mixture Models and E/M Algorithms

The Origins of Variational Inference

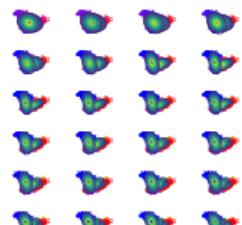
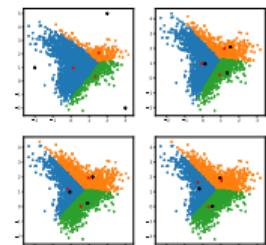
- Gaussian mixtures are often used to model multi-modal distributions
- You have to solve two problems simultaneously
 - Find means and covariances of the clusters
 - Infer to which cluster a data point belongs
- When the cluster labels are given, this is straightforward (Unit 2)
- When not the algorithm itself must identify clusters
- This is an example of unsupervised learning: the unlabelled data does not give any hint how to do this. The algorithm must do this by itself
- This is unlike most forms of classification and regression, where *training sets* are required: data points *and* desired output are fed into the algorithm



Unit 5: Mixture Models and E/M Algorithms

The Origins of Variational Inference

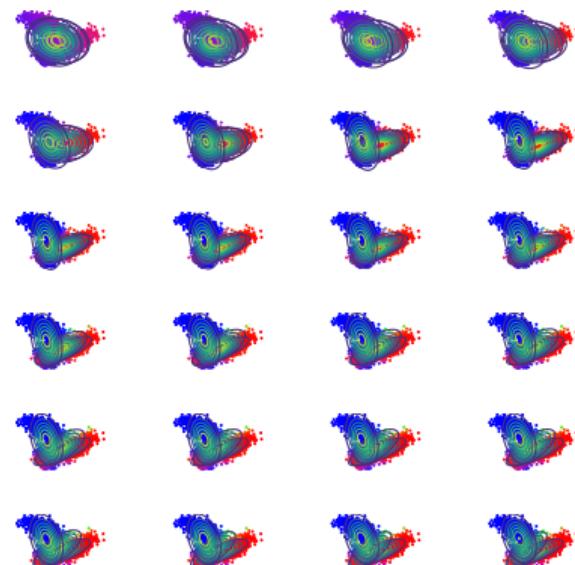
- Assigning a data point to a cluster does not work well
- A probabilistic assignment works better
- This requires a *latent variable*, a stochastic variable that controls cluster assignment
- This variable is *unobserved*, but must be inferred through indirect methods
- Application of Bayes' law
- E/M Algorithm: two stages
 - Estimate the means and covariances based on the latent variables
 - Update the latent variables
- This algorithm requires optimising the Evidence Lower BOund (ELBO)
- These ideas lead to VAEs in a way that we will try to explain in Unit 5



Unit 5: Mixture Models and E/M Algorithms

The Origins of Variational Inference

- The Gaussian mixture can achieve enormous data reduction
- Regularisation of components: Bayes!
- We really would like to have a prior distribution on the mixing components
- Aim: Infer posterior of the latent distribution
- Analytic treatment: too hard; solution *variational inference*: assume independence of latent space variables



Practical Work

- Practical work is organised per unit
- Combination of notebooks and worksheets
- You will develop a full machine learning pipeline using *scikit-learn*
- Most algorithms will be exemplified in notebooks in Python
- Crash course notebooks in *numpy* and *pandas* are available
- Please make use of TA assistance!

Recap

- Today: Roadmap for the module, selection of topics
- Organisation of the practicalities
- Next lecture: review of probability with emphasis on Bayes' law

COMP5611M: Machine Learning

Bayes' Law

Nishant Ravikumar
Marc de Kamps

School of Computing

January 31, 2022

Learning Objectives

- Quick and dirty recap of statistics:
- Multivariate distributions; joint, conditional and marginal probability distributions
- Bayes' law
- Bayesian inference of a coin discrete; continuous
- Comparison of MLE and Bayesian inference

Learning Outcomes

At the end of this lecture you should be able to:

- Derive Bayes' law from the definitions of conditional and joint probabilities
- Apply Bayes' law in simple settings

Discrete Distributions

Bernoulli

- Data point $x_i, i \dots, N$
- $x_i \in \{0, 1\}$

$$\text{Ber}(x | \mu) = \mu^x(1 - \mu)^{1-x}, 0 \leq \mu \leq 1 \quad (1)$$

How does this work? There are only two possibilities:

1. $x = 0 : \text{Ber}(x | \mu) = (1 - \mu)$
2. $x = 1 : \text{Ber}(x | \mu) = \mu$

This is exactly what we expect from a **biased** coin. Talking about expectations:

$$\begin{aligned} \mathbb{E}[x] &= \mu \\ \text{var}[x] &= \mu(1 - \mu) \end{aligned} \quad (2)$$



Likelihood vs Probability

Example: Bernoulli Process

- Look at a set of data points from a Bernoulli process
 $x_1 = 'H'$, $x_2 = 'T'$, $x_3 = 'T'$
- If we consider μ as given, the probability to realise these particular data points is

$$(1 - \mu)\mu^2 = \text{Ber}(x = 0 | \mu)\text{Ber}(x = 1 | \mu)\text{Ber}(x = 1 | \mu)$$

- If we **don't** know μ and consider this quantity a function of μ , we call it a *likelihood*



Inference

Maximum Likelihood Estimation (MLE)

- A natural problem is that of **inference**. Given a sequence of observations, e.g. 'HTTTHHHTTTTHHHT', can we say something about μ
- Strategy: write down the likelihood of the observation and **maximise** the likelihood wrt μ
- This is called *maximum likelihood estimation*
- Observe that events in Bernoulli are independently identically distributed: **idd.**
- This means you can rearrange the observations in any order without affecting the likelihood



Inference

Maximum Likelihood Estimation (MLE)

$$\ln P(\mathcal{D} \mid \mu) = N_1 \ln \mu + (N - N_1) \ln(1 - \mu)$$

We find the value of μ that maximises the likelihood, μ_{ML} by:

$$\frac{d \ln P(\mathcal{D} \mid \mu)}{d\mu} \Big|_{\mu=\mu_{ML}} = 0$$

this works out as:

$$\frac{N_1}{\mu_{ML}} - \frac{N - N_1}{1 - \mu_{ML}} = 0,$$

solving for μ :

$$\mu_{ML} = \frac{N_1}{N}$$



We will later see that the MLE is prone to overfitting

Joint Probabilities

Two Dice

Dice 1 Dice 2	'1'	'2'	'3'	'4'	'5'	'6'
'1'	0.0282	0.0282	0.0282	0.0286	0.0286	0.0282
'2'	0.0299	0.0299	0.0299	0.0302	0.0302	0.0299
'3'	0.0249	0.0249	0.0249	0.0252	0.0252	0.0249
'4'	0.0232	0.0232	0.0232	0.0235	0.0235	0.0232
'5'	0.0266	0.0266	0.0266	0.0269	0.0269	0.0266
'6'	0.0332	0.0332	0.0332	0.0336	0.0336	0.0332



Independent Variables

- Table: outer (or tensor) product of two vectors

$$p_1 = (0.17, 0.18, 0.15, 0.14, 0.16, 0.2)^T$$

$$p_2 = (0.166, 0.166, 0.166, 0.168, 0.168, 0.166)^T,$$

- Individual state spaces: 6 elements
- Combined state space 6^2 , **curse of dimensionality**
- Table is asymmetric:
 $P(X_1 = '1')P(X_2 = '2') \neq P(X_1 = '2')P(X_2 = '1')$
- But displays independence

$$P(X_1 = 'i', X_2 = 'j') = P(X_1 = 'i')P(X_2 = 'j')$$



'Outer Product'

- Given \vec{a}, \vec{b} , possibly different dimensions (say N,M)
- Order the quantities $a_i b_j$ in a matrix \mathcal{M} with:

$$\mathcal{M}_{ij} = a_i b_j$$

- \mathcal{M}_{ij} the components of \mathcal{M}

Then \mathcal{M} is called the tensor product of vectors \vec{a}, \vec{b} .

Tensor products can be convenient to create joint probability tables of independent variables

Height \ Nationality	Zobany	Grabandan	Σ
$L < 150$	0.0068	0.0573	0.0641
$150 \leq L < 160$	0.0158	0.2074	0.2231
$160 \leq L < 170$	0.0300	0.3285	0.3585
$170 \leq L < 180$	0.0371	0.2074	0.2445
$180 \leq L < 190$	0.0300	0.0520	0.0820
$190 \leq L < 200$	0.0158	0.0051	0.0209
$L > 200$	0.0068	0.0002	0.0070
Σ	0.1422	0.8578	1.0000

Table: The joint probability for encountering an individual of certain nationality and height.

What do you think? Is this table created from independent variables ('is it an outer product')?

Marginal Probability

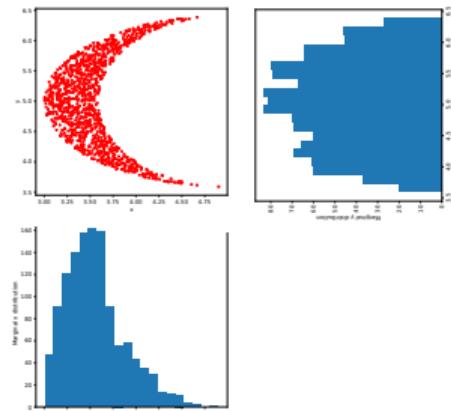
$$P(X) = \sum_Y P(X, Y)$$

Sum is over all elements of Y . Shorthand for:

$$P(X = a) = \sum_{b \in S_y} P(X = a, Y = b),$$

where S_y is the sample space of stochastic variable Y .

1. Not possible to reconstruct joint probability distribution from the marginals. Marginalisation generally constitutes a significant loss of information.
2. Exception: independence
3. Marginalisation is hard! (How so? It's just summing columns or rows ... Well, the curse of dimensionality)



Conditional Probability

$$P(Y = y_i | X =' Z'),$$

- Probability that the person is in height category y_i , given that they are a Zobany national.
- How can this even be a probability given that all probabilities in the Zobany capital add up to 0.85?

$$\sum_i P(X =' Z', Y = y_i) = P(X =' Z')$$

- If we add all possibilities for someone to be of a certain height, whilst knowing that someone is Zobanian, we must have the probability that someone is Zobanian.
- A Zobanian must fall in *some* height category. So

$$\sum_i \frac{P(X =' Z', Y = y_i)}{P(X =' Z')} = 1$$

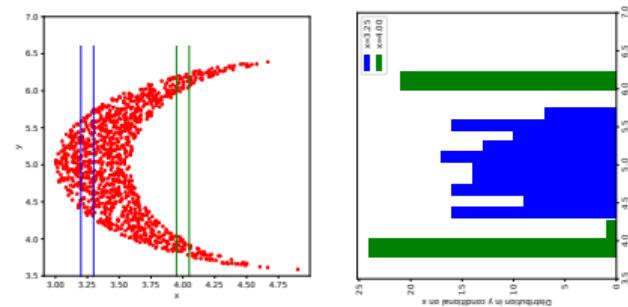
Conditional Probability

It is clear that the numbers:

$$P(Y = y_i \mid X =' Z') \equiv \frac{P(X =' Z', Y = y_i)}{P(X =' Z')} \quad (3)$$

are properly normalised probabilities.

Conditional probabilities



Sum and Product Rule

The *sum rule*

$$P(X) = \sum_Y p(X, Y) \quad (4)$$

The *product rule*

$$p(X, Y) = p(Y | X)P(X) \quad (5)$$

Bayes' Rule

or Law or Theorem or ...

Consider the product rule and observe that it may be written in two ways:

$$P(X, Y) = P(Y | X)P(X), \quad (6)$$

or:

$$P(X, Y) = P(X | Y)P(Y) \quad (7)$$

These decompositions are equally valid.

$$P(X | Y) = \frac{P(Y | X)}{P(Y)}P(X) \quad (8)$$

or using the sum rule:

$$P(X | Y) = \frac{P(Y | X)}{\sum_x P(Y | X)P(X)}P(X) \quad (9)$$

Bayes

A Real World Example

- Joe is randomly chosen from a population of which 3 % is a heroin user
- Joe is tested: a test with a *sensitivity* of 95 %
 - *sensitivity* is the fraction of heroine users that are correctly identified by the test
- The test has *specificity* of 90 %
 - *specificity* is the fraction of non-heroin users that are correctly identified by the test

Joe tests positive! Is he a heroine user? What do you think?

This example comes from:

<https://plato.stanford.edu/entries/bayes-theorem/supplement.html>. You are strongly encouraged to visit this page

Bayes

Translating the Problem

In solving this problem you need to recognise that some conditional probabilities have been provided:

- H : Joe is a heroine user ($\neg H$, he is not); '+': positive test, '-': negative test
- $P(H) = 0.03$. The prior probability that Joe uses heroine
- $P('+' | H) = 0.95, P('-' | \neg H) = 0.90$

We're interested in $P(H | '+'')$, not $P('+' | H)$. The dependency is the wrong way.

Bayes!

Formulating Bayes' Law

Bayes' law reads:

$$P(H | '+'') = \frac{P('+' | H)P(H)}{P('+' | H)P(H) + P('+' | \neg H)P(\neg H)}$$

Now observe:

- $P(\neg H) = 1 - P(H) = 0.97$
- $P('+' | \neg H) = 1 - P('-' | \neg H) = 0.1$

so

$$P(H | '+'') = \frac{0.95 \cdot 0.03}{0.95 \cdot 0.03 + 0.1 \cdot 0.97} = 0.227$$

If you've not seen examples like this before, this is probably a surprise!

Bayesian Inference

MLE vs Bayes

Assume Bernoulli process. Observe 'HHH' (code 'T' = 0, 'H' = 1).

- What is μ ?
- MLE $\mu = N_1/N = 1$

Why do we measure μ ? Prediction!

- What is our best estimate for a future series of events?
- 'HHHHHHHHHHHHHHHH'

This doesn't seem right ... What to do?

Bayesian Inference

Discrete Example

For simplicity assume that μ can take five discrete values:

$$\mu \in [0.025, 0.5, 0.75, 1.0]$$

How can we employ Bayes?

- Remember that we have an expression for the likelihood

$$P(x_1 =' H', x_2 =' H', x_3 =' H' | \mu) = \mu^3$$

- We can write this as $P(\mathcal{D} | \mu)$. Read this as 'the probability of the observed data given μ '.
- Note that if we know μ , this is the joint probability for observing the data
- If we don't know μ , we consider this the likelihood of the data given an unknown μ

If we perform inference, we are interested in $p(\mu | \mathcal{D})$, what we have is $p(\mathcal{D} | \mu)$.

Do you spot the pattern?

Bayesian Inference

- A choice for μ is a hypothesis about the data generation mechanism
- A likelihood $p(\mathcal{D} | \mu)$ is the 'probability of data, given the hypothesis'
- We want a 'probability for the hypothesis, given the data'
- Bayes law allows us to convert one into the other

$$P(H | \mathcal{D}) = \frac{p(\mathcal{D} | H)p(H)}{p(\mathcal{D})}$$

Neat but a little abstract ...

Bayesian Inference

Specifying the Prior

What does $P(H)$ mean?

- Remember that H stands for a hypothesis on data generation
- In this case μ determines what kind of data we can generate
- μ is a discrete stochastic variable, sample space
 $\mu \in [0.00, 0.25, 0.50, 0.75, 1.00]$
- Picking $P(H)$ amounts to assigning a probability distribution to values of μ

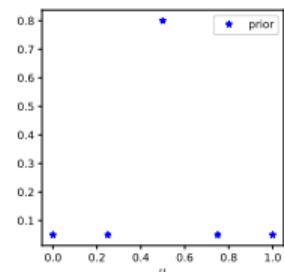
$$P_{prior}(\mu = 0.00) = 0.05$$

$$P_{prior}(\mu = 0.25) = 0.05$$

$$P_{prior}(\mu = 0.50) = 0.8$$

$$P_{prior}(\mu = 0.75) = 0.05$$

$$P_{prior}(\mu = 1.00) = 0.05$$



So $p(H)$ is something we pick. We must make a reasoned choice, to the best of our ability.

Our choice here reflects we strongly believe the coin is unbiased.

Bayesian Inference

Specifying the Likelihood

What does $P(\mathcal{D} | H)$ mean?

- The likelihood of the data!
- This is both the simple and the complicated part!

For Bernoulli:

$$p(\mathcal{D} | H) = p(x_1 = 'H', x_2 = 'H', x_3 = H | \mu) = \mu^3$$

In general, for N observations, of which N_1 turn out to be 'H':

$$p(\mathcal{D} | H) = \mu^{N_1} (1 - \mu)^{N - N_1}$$

- Hard because the likelihood is your physical model of data generation
- Easy because once you have that, writing down $p(\mathcal{D} | H)$ is routine.

Bayesian Inference

What is $p(\mathcal{D})$?

Bayes' law now states:

$$p(H | \mathcal{D}) = \frac{p(\mathcal{D} | H)p(H)}{p(\mathcal{D})}$$

Ignore $p(\mathcal{D})$ for a second and concentrate on:

$$p(\mathcal{D} | H)p(H)$$

What does this mean? It means that for every possible value of μ we need to calculate this quantity, e.g. for $\mu = 0.5$

$$P(\mathcal{D} | \mu = 0.5)P_{prior}(\mu = 0.5)$$

But we know these numbers: for our observations we have

$P(\mathcal{D} | \mu) = \mu^3$ and we chose $P_{prior}(\mu = 0.5)$ ourselves!

$$P_{prior}(\mu = 0.00) = 0.05$$

$$P_{prior}(\mu = 0.25) = 0.05$$

$$P_{prior}(\mu = 0.50) = 0.8$$

$$P_{prior}(\mu = 0.75) = 0.05$$

$$P_{prior}(\mu = 1.00) = 0.05$$

Bayesian Inference

What is $p(\mathcal{D})$?

We can make a table for all values of μ of $P(\mathcal{D} | H)P(H)$

$\mu = 0$	$\mu^3 P_{prior}(\mu = 0)$	0
$\mu = 0.25$	$\mu^3 P_{prior}(\mu = 0.25)$	0.00078125
$\mu = 0.5$	$\mu^3 P_{prior}(\mu = 0.5)$	0.1
$\mu = 0.75$	$\mu^3 P_{prior}(\mu = 0.75)$	0.02109375
$\mu = 1.0$	$\mu^3 P_{prior}(\mu = 1.0)$	0.05
		$P(\mathcal{D}) = 0.171875$

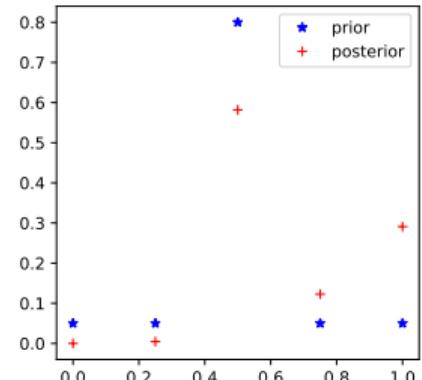
- $P(\mathcal{D})$ is the sum over the last entries

$$P(\mathcal{D}) = \sum_i P(\mathcal{D} | \mu_i)P_{prior}(\mu = \mu_i)$$

- The last column are the posterior probabilities up to normalisation!

Posterior

- To find the posterior values, we just need to normalise, using $p(\mathcal{D})$
- Still a broad distribution but more mass at $\mu = 1$
- Prediction: sample from posterior
- Plenty of uncertainty: no overfitting
- Do not always need normalisation:
 - If only interested in the maximum: MAP
 - If only two outcomes: odds ratio



Recap

It is hard to overestimate the impact of Bayes' law.

- Lab: real-world example
- Demonstrates our intuition can be led astray badly if we don't use it
- Next lecture: Bayesian Inference (on a coin at first; is it fair)?
- Examples of discrete and continuous versions of Bayesian inference

COMP5611M: Machine Learning

Information Theory: Entropy and Cross Entropy

Nishant Ravikumar
Marc de Kamps

School of Computing

February 6, 2022

Learning Objectives

- Formulation of Entropy and Cross Entropy
- Jensen's Inequality
- Kullback-Leibler Divergence
- We need these concepts to understand the
 - the loss function for *logistic regression*
 - the E/M Algorithm for inferring Gaussian Mixture Models
- We explain it here because we have just been talking about probability distributions.
- In the process we will see some examples of expectation values.

Designing a Code

An Optimal Code

Suppose we have a remote observatory somewhere. It needs to observe whether events occur and transmit the occurrence of the event. Our bandwidth is severely limited. Can we design an optimal code that limits the overall amount of transformation that we need to transmit?

- Assume for now a discrete probability distribution
- A binary code
- Key idea: give likely events a small number of bits
- Use a larger number of bits for unlikely events
- Hope that you reduce the overall information transfer
- **Can we say something about the amount of information that we need to transfer?**



Source: NASA

Bits

State space: 8 possible outcomes?

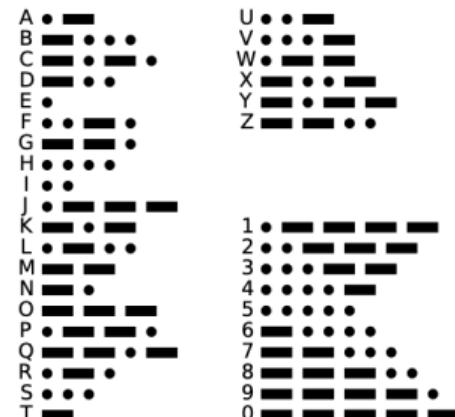
- We need three bits to represent 8 possible outcomes:

000, 001, 010, 011, 100, 101, 110, 111

- In general, if we have (at most) 2^N outcomes
- We need N bits
- At least if we assign the same number of bits to each possible outcome

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



Information Capacity

State space: 8 possible outcomes?

- 8 events $\{a, b, c, d, e, f, g, h\}$,
- Probabilities: $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$,
- If we just go by the number of outcomes, we need 3 bits/event

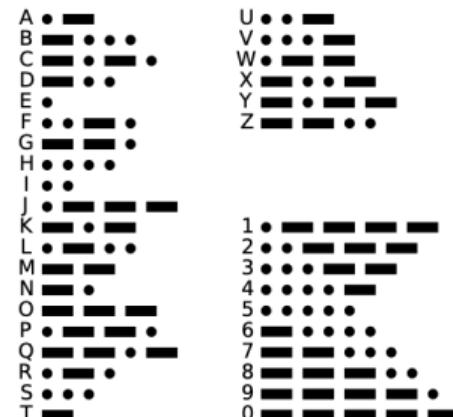
It is now possible to develop a code that transmits less than 3 bits/event on average?

A typical sequence might look like:

caaabgaababaabch...

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



Efficient Coding

- 8 events $\{a, b, c, d, e, f, g, h\}$,
- Probabilities: $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$

Now consider the following coding:

0, 10, 110, 1110, 111100, 111101, 111110, 111111

- This code assigns less bits to more frequent items
- Still makes it possible to resolve tokens from a long sequence

We can calculate the expected number of bits/event:

$$\frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{16} \times 4 + 4 \times \frac{1}{64} \times 6 = 2,$$

This is less than 3 bits/event.

Entropy

When we say we have 8 outcomes with equal probability we observe:

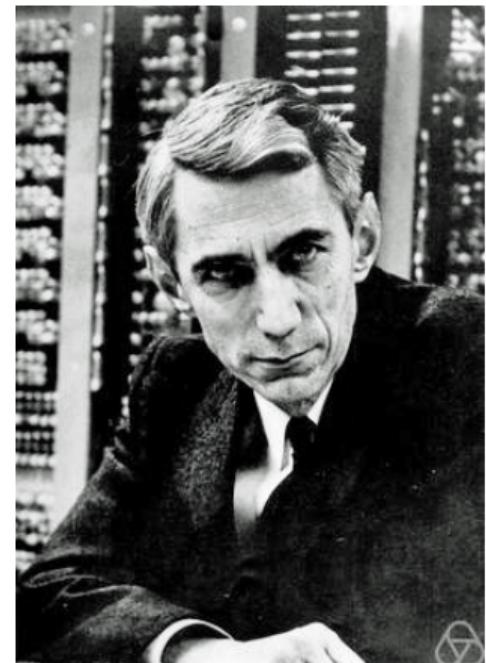
$$p_i = \frac{1}{8} = \frac{1}{2^3} \quad i = 1, \dots, 8$$

We define the information content of a particular event j as:

$$I = -2 \log p_j$$

(This is logarithm with base 2)

- For $p_i = \frac{1}{8}$, $I = 3$
- This corresponds with our intuition that we need 3 bits to code the event
- The lower the probability, the more informative the event!



Claude Shannon, Wikipedia

Entropy

We define entropy as the expectation value of I :

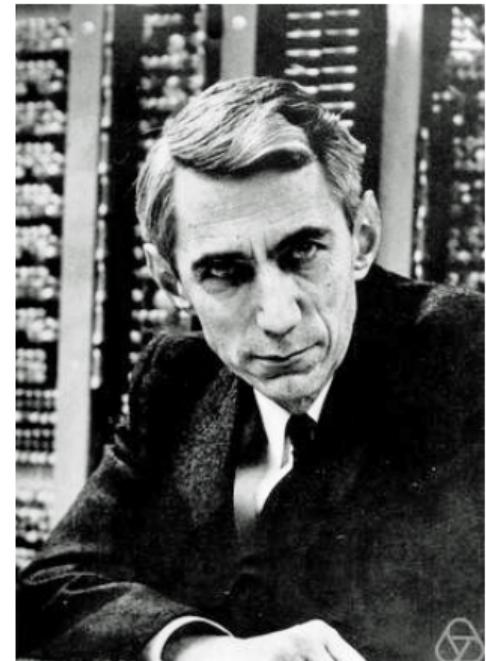
$$H = \mathbb{E}[I] = \mathbb{E}_{p_i}[-^2 \log p_i] = - \sum_i p_i \cdot ^2 \log p_i$$

- Known in statistical physics for a long time
- Introduced by Claude Shannon in the context of Information Theory

Now apply it to 8 equiprobable outcomes, i.e. $p_i = \frac{1}{8}$:

$$H = -8 \cdot \frac{1}{8} \cdot ^2 \log \frac{1}{8} = 3$$

This suggests that on average 3 bits are required to code for an event



Claude Shannon, Wikipedia

Entropy

$$H = \mathbb{E}[I] = \mathbb{E}_{p_i}[-^2 \log p_i] = - \sum_i p_i \cdot ^2 \log p_i$$

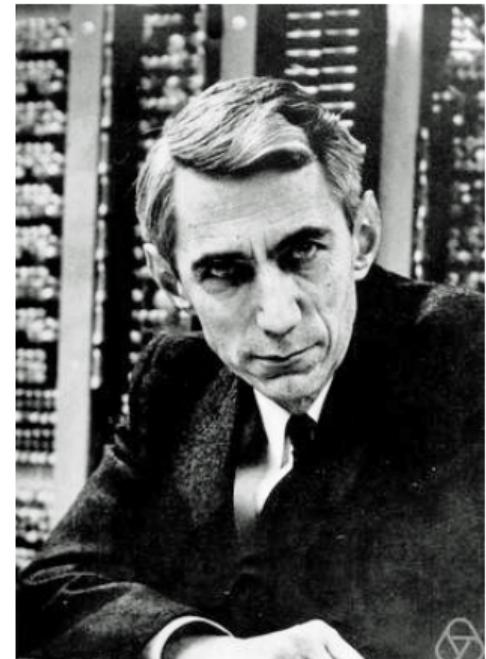
Now let us revisit the case where unequal probabilities led to a more efficient coding scheme:

- 8 events $\{a, b, c, d, e, f, g, h\}$,
- Probabilities: $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$

$$H = \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + 4 \cdot \frac{1}{16} + 6 \cdot \frac{1}{64} = 2$$

Take a moment to verify that this calculation is the same!

- The outcome suggests the existence of a more efficient coding scheme
- The calculation is just in terms of the probability of outcomes; not predicated on coding!



Claude Shannon, Wikipedia

Noiseless Coding Theorem

- You can always calculate the entropy over a probability distribution
- But you cannot always find a code that achieves the transmission rate predicted by the energy

Why not?

Noiseless Coding Theorem (Shannon, 1948)

Entropy is a lower bound on the number of bits needed to transmit the state of a random variable.

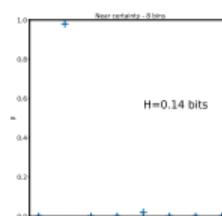
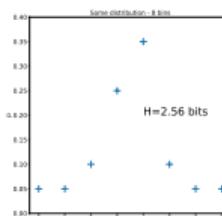
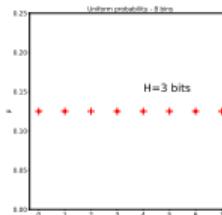


- Thinking about coding length is a scaffold.
- Ultimately our results don't depend on coding

Some Intuition Building

The Smallest/Largest Entropy?

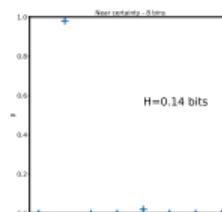
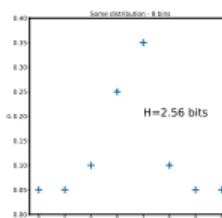
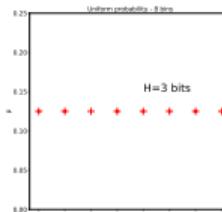
- Let's calculate some entropies
- Do this yourself. Play!
- Consider the three distributions for 8 bins
- Can you explain the results?
- How do you deal with the formula $H = -\sum_i p_i \log p_i$ when some p_i are 0?
- What happens if I change the base of the logarithm?



Some Intuition Building

The Smallest/Largest Entropy?

- Let's calculate some entropies
- Do this yourself. Play!
- Consider the three distributions for 8 bins
- Can you explain the results?
- How do you deal with the formula $H = -\sum_i p_i \log p_i$ when some p_i are 0?
- What happens if I change the base of the logarithm?
- I change the units but nothing else: $2^I \log_2$: bits, \ln : nats



Some Intuition Building

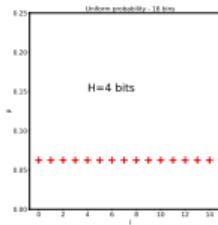
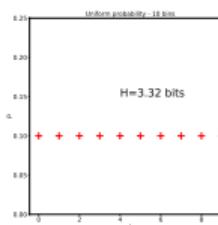
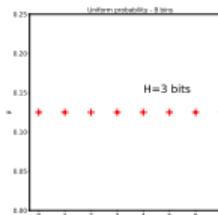
The Smallest/Largest Entropy?

- Largest entropy given number outcomes: **uniform distribution**
- Maximise $H = -\sum_i p_i \ln p_i$ subject to $\sum_i p_i = 1$
- Maximise (Lagrangian multiplier):
$$-\sum_i p_i \ln p_i + \lambda(\sum_i p_i - 1)$$

so

$$p_i = \frac{1}{N}, H = -\ln N$$

- Explain this in terms of coding length
- We are in trouble with continuous distributions!



Continuous Distributions

Differential Entropy

- $p(x)$ continuous, there exists an x_i such that

$$\int_{i\Delta}^{(i+1)\Delta} p(x)dx = p(x_i)\Delta$$

- Discretisation of the entropy:

$$\mathbb{H}_\Delta = - \sum_i p(x_i)\Delta \ln(p(x_i)\Delta) = \sum_i p(x_i)\Delta \ln p(x_i) - \ln \Delta$$

- Omit the second term
- Consider the limit $\Delta \rightarrow 0$.

$$\lim_{\Delta \rightarrow 0} \sum_i p(x_i)\Delta \ln p(x_i) = - \int p(x) \ln p(x) dx$$

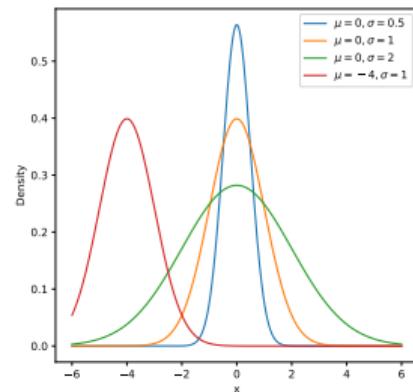
- The integral is called *differential entropy*.

Continuous Distributions

Maximum Differential Entropy

- Somewhat surprising:
- Given first and second moment (mean and variance)
- Continuous distribution with maximum differential entropy is Gaussian (see Bishop (2006))

$$H[x] = \frac{1}{2} \{1 + \ln(2\pi\sigma^2)\}$$



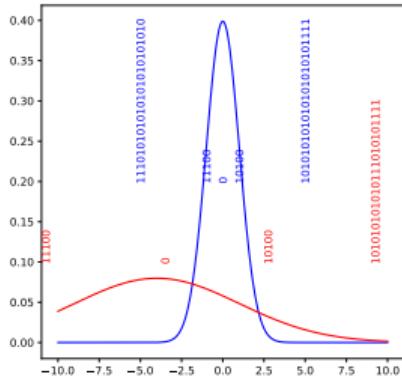
The Wrong Codebook!

Kullback-Leibler Divergence

- We can look at entropy as the average information transmitted by events
- Cost for transmitting event: $-\ln p_i$
- Assumed distribution $q(x)$
- True distribution $p(x)$
- On average we need to transmit more information than planned

$$KL(p \parallel q) = - \int p(x) \ln q(x) dx - (- \int p(x) \ln p(x) dx)$$

- Kullback-Leibler divergence is the amount of extra information
- We can detect this because our disks fill faster than expected!

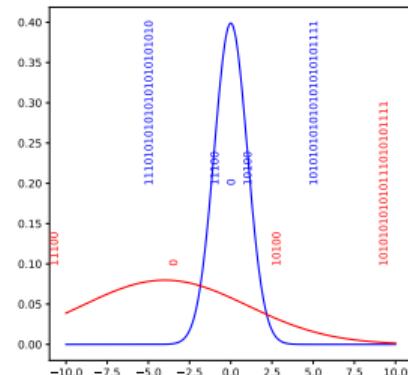


Properties of KL-divergence

Jensen's Inequality

We actually don't need the coding metaphor to prove:

- $KL(p \parallel q) \geq 0$
- Equality only if $q(x) = p(x)$
- Bit like a distance but not symmetric in $p(x), q(x)$
- Events generated by the blue distribution are moderately expensive under blue code book
- Many events generated by the red distributions are **extremely expensive under red code book**
- The proof rests on a fairly trivial observation on convex functions



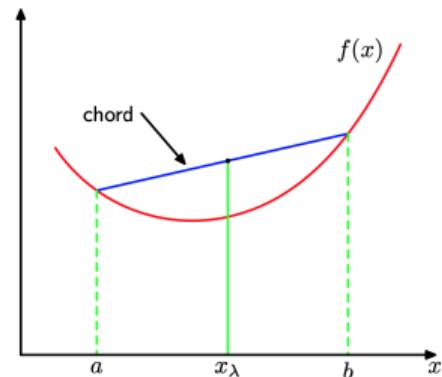
Convex Function

- Convex (strict) function $f''(x) > 0$
- Consider $f(a), f(b), a < b$
- The linear cord between a, b is always above the function

$$\lambda f(a) + (1 - \lambda)f(b) > f(\lambda a + (1 - \lambda)b)$$

- By induction for $\lambda_i > 0, \sum_i \lambda_i = 1$

$$f\left(\sum_{i=1}^N \lambda_i x_i\right) \leq \sum_{i=1}^N \lambda_i f(x_i)$$



Convex Function

- For $\lambda_i > 0$, $\sum_i \lambda_i = 1$
- The λ_i can be interpreted as probabilities

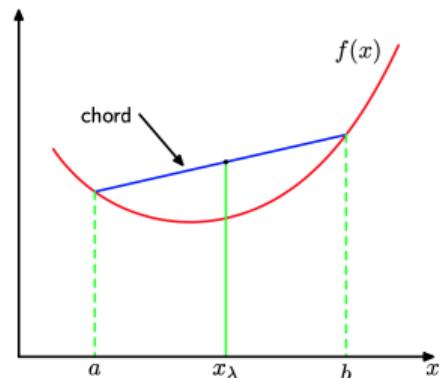
$$f\left(\sum_{i=1}^N \lambda_i x_i\right) \leq \sum_{i=1}^N \lambda_i f(x_i)$$

- This can be expressed in terms of expectation values

$$f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$$

- The continuous version reads:

$$f\left(\int p(x)dx\right) \leq \int f(x)p(x)dx$$



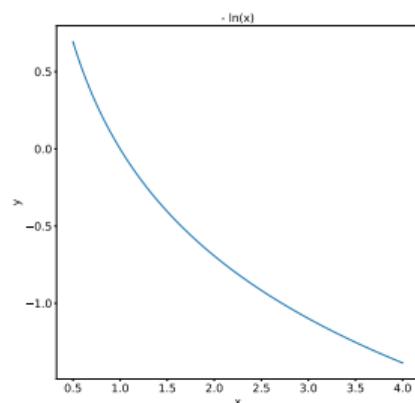
Convex Function

- $-\ln(x)$ is convex function

$$KL(p \parallel q) = - \int p(x) \ln \frac{q(x)}{p(x)} dx \geq -\ln \left\{ \int q(x) dx \right\} = 0$$

- The first term is the *cross entropy*
- Close inspection of Jensen's inequality shows

$$KL(p \parallel q) = 0 \iff p(x) = q(x)$$



- How do we use this?

Using the KL-divergence

- We often have an approximation $q(x | \theta)$
- True probability $p(x)$
- We want to minimise $KL(p || q)$
- Problem: we don't know $p(x)$
- We may have a data set $x_i, i = 1, \dots, N$
- Estimate:

$$KL(p || q) \approx \sum_{i=1}^N \{-\ln q(x_i | \theta) + \ln p(x_i)\}$$

- Only $q(x | \theta)$ is dependent on θ ; the p -term is constant
- Maximising the log likelihood is minimising the KL-divergence is minimising the cross entropy
- Why not just keep a tally of events and compare p and q directly?

COMP5611M: Machine Learning

Multivariate Gaussian Distributions

Nishant Ravikumar

Marc de Kamps

School of Computing

February 10, 2022



UNIVERSITY OF LEEDS

Lecture Objectives

Objectives for this lecture:

- Introduction to Multivariate Gaussian distributions & MLE
- Spectral decomposition of the covariance matrix
- Look at some real world applications

Learning Outcomes

At the end of this lecture, you should be able to

- Explain what a multivariate Gaussian distribution is, and what its covariance matrix describes
- Explain what spectral decomposition of the covariance matrix achieves and its practical relevance in ML
- Understand MLE for parameters of multivariate Gaussians
- Describe some real world examples where these concepts are useful

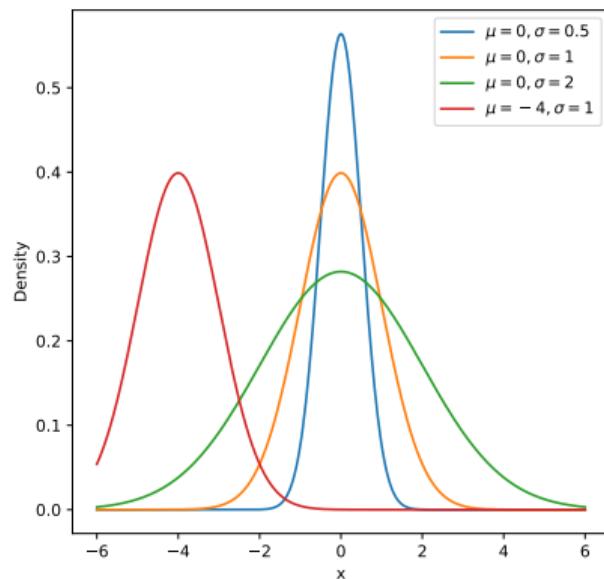
Recap: Univariate Gaussian distribution

- Occurs often in Nature: e.g. height, IQ
- Fixed length sums of variables of any distribution follow approximate Gaussian: **Central Limit Theorem**
- μ : position of centre; σ : width of distribution

$$\mathcal{N}(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \quad (1)$$

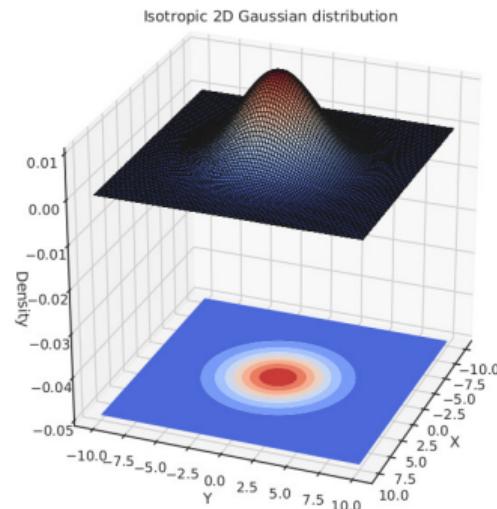
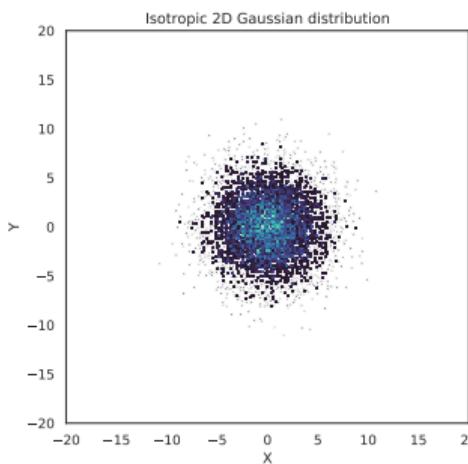
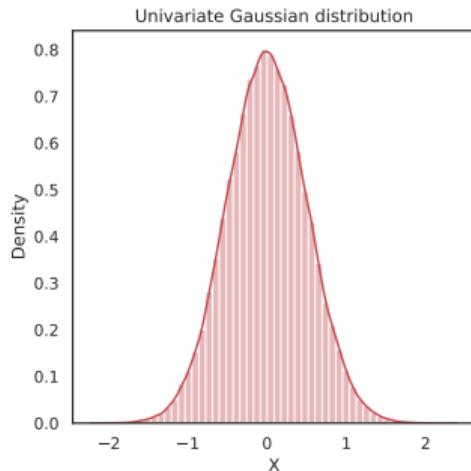
$$\mathbb{E}[\mathcal{N}(x | \mu, \sigma^2)] = \mu \quad (2)$$

$$\text{var}[\mathcal{N}(x | \mu, \sigma^2)] = \sigma^2$$



Multivariate Gaussian Distribution

- Ubiquity of the Gaussian distribution can be explained by the **Central Limit Theorem**
- This holds for Univariate and Multivariate Gaussian distributions
- What is a **multivariate** distribution?



(a) Univariate Gaussian distribution

(b) Samples from bivariate Gaussian

(c) Density of bivariate Gaussian

Multivariate Gaussian Distribution

- For an N -dim vector \mathbf{x} the multivariate Gaussian distribution is given by:

$$\mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{N/2}} \frac{1}{|\det(\boldsymbol{\Sigma})|^{\frac{1}{2}}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\} \quad (3)$$

- Here, $\boldsymbol{\mu}$ is an N -dim mean vector and $\boldsymbol{\Sigma}$ is an $N \times N$ covariance matrix
- For example for a bivariate (2D) Gaussian: $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy}^2 \end{bmatrix}$
- The functional dependence of \mathcal{N} on \mathbf{x} is quadratic:

$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \quad (4)$$

where, Δ is known as the **Mahalanobis distance**. This becomes the Euclidean distance when $\boldsymbol{\Sigma} = \mathbf{I}$.

Multivariate Gaussian Distribution

- The covariance matrix Σ can take on different forms:
 - If $\Sigma = \sigma^2 \mathbf{I}$, where \mathbf{I} is an N -dim identity matrix then → said to be **isotropic**
$$\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$$
 - If Σ can be expressed as $\text{diag}(\sigma_{i=1\dots N}^2)$ → its called a **diagonal** covariance matrix
$$\Sigma = \begin{bmatrix} \sigma_{xx}^2 & 0 \\ 0 & \sigma_{yy}^2 \end{bmatrix}$$

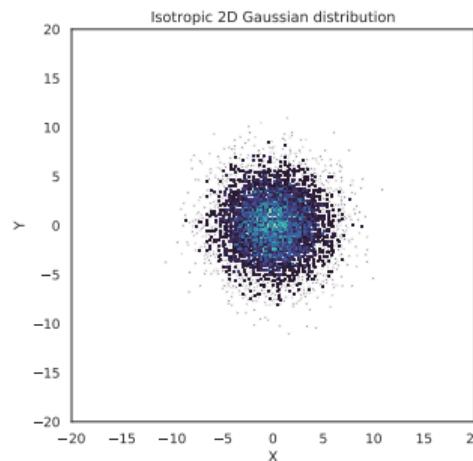
- And otherwise, its called a **full/general** covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy}^2 \end{bmatrix}$$

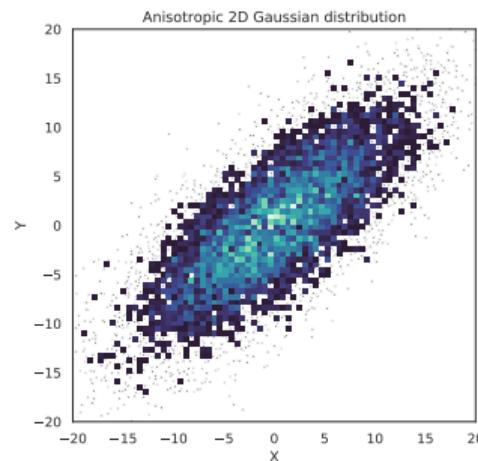
- What is the significance of using different types of covariance when modelling your data?

Multivariate Gaussian Distribution

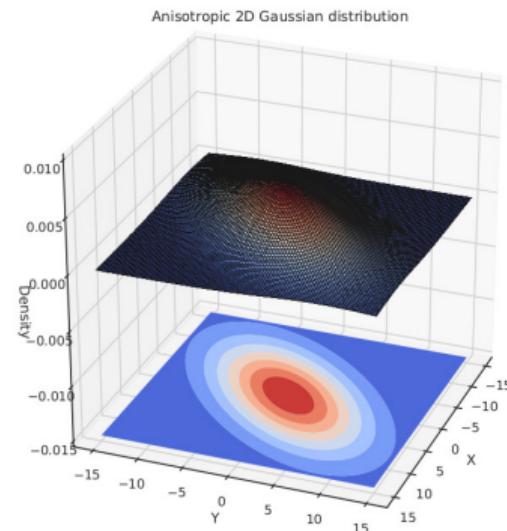
- What if Σ is not isotropic? I.e. it is *anisotropic*?
- Σ can no longer be expressed using a scalar σ^2
- The shape of the distribution changes - example of a *full/general* covariance matrix



(a) Samples of isotropic bivariate-Gaussian



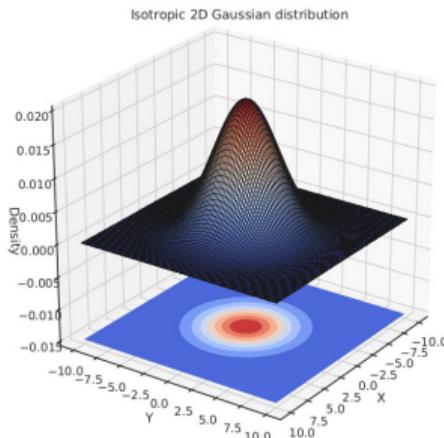
(b) Samples of anisotropic bivariate-Gaussian



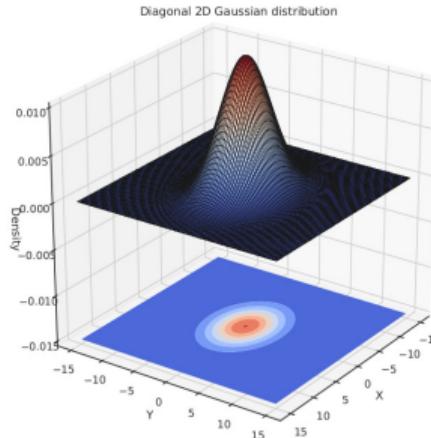
(c) Density of anisotropic bivariate-Gaussian

What information does the covariance matrix contain?

- If we consider the univariate case, σ^2 describes the spread of the distribution along one dimension
- Can we explain the spread of a bivariate (2D) Gaussian distribution using variances ($\sigma_{i=1\dots 2}^2$) alone?
- Yes, if the covariance is isotropic or diagonal!, i.e. $\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$ or $\begin{bmatrix} \sigma_{xx}^2 & 0 \\ 0 & \sigma_{yy}^2 \end{bmatrix}$



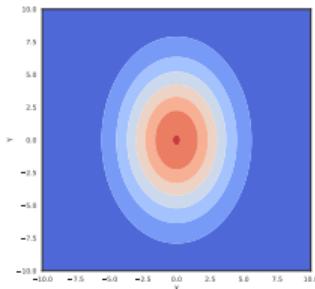
(a) $\Sigma = \begin{bmatrix} 8.0 & 0.0 \\ 0.0 & 8.0 \end{bmatrix}$



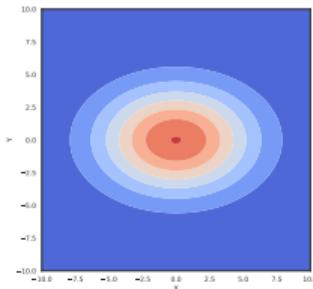
(b) $\Sigma = \begin{bmatrix} 16.0 & 0.0 \\ 0.0 & 8.0 \end{bmatrix}$

What information does the covariance matrix contain?

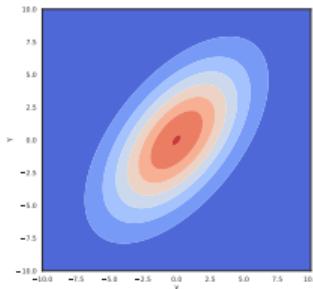
Lets look at a few more examples of 2D Gaussian distributions and their covariances: $\Sigma = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy}^2 \end{bmatrix}$



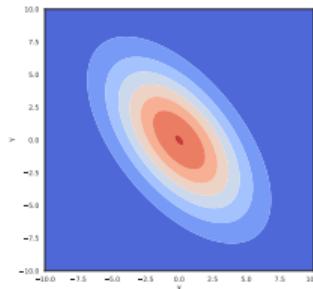
(a) $\Sigma = ?$



(b) $\Sigma = ?$



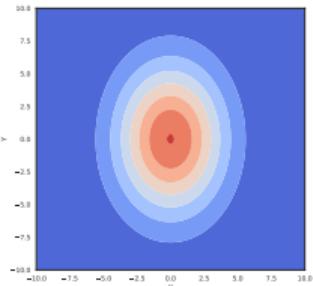
(c) $\Sigma = ?$



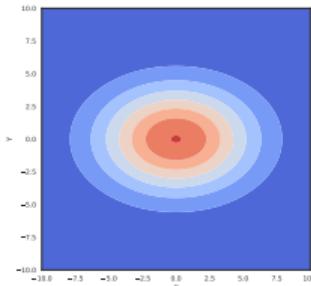
(d) $\Sigma = ?$

What information does the covariance matrix contain?

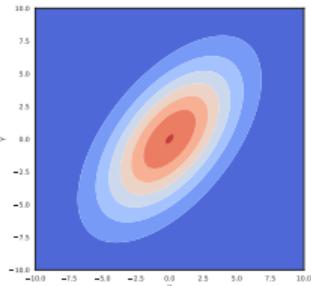
Lets look at a few more examples of 2D Gaussian distributions and their covariances: $\Sigma = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy}^2 \end{bmatrix}$



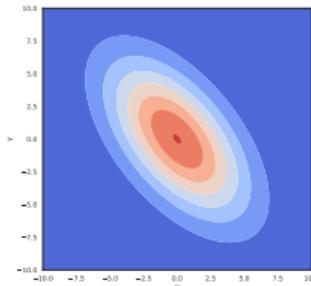
$$(a) \Sigma = \begin{bmatrix} 8.0 & 0.0 \\ 0.0 & 16.0 \end{bmatrix}$$



$$(b) \Sigma = \begin{bmatrix} 16.0 & 0.0 \\ 0.0 & 8.0 \end{bmatrix}$$



$$(c) \Sigma = \begin{bmatrix} 12.0 & 8.0 \\ 8.0 & 16.0 \end{bmatrix}$$



$$(d) \Sigma = \begin{bmatrix} 12.0 & -8.0 \\ -8.0 & 16.0 \end{bmatrix}$$

- In (a) and (b) σ_{xx}^2 & σ_{yy}^2 determine the axes along which the distribution is **elongated**
- In (c) σ_{xy} & σ_{yx} are non-zero and **positive**, i.e. x and y are positively correlated
- In (d) σ_{xy} & σ_{yx} are non-zero and **negative**, i.e. x and y are negatively correlated

What information does the covariance matrix contain?

- The covariance matrix Σ of a multivariate Gaussian distribution is a symmetric matrix
- What is a symmetric matrix? Formally a matrix Σ where, $\Sigma = \Sigma^T$
- Symmetric matrices can be diagonalised through a suitable coordinate transformation
- I.e. from the original coordinate system to a new one spanned by an **orthonormal basis of eigenvectors** of Σ
- Factorising a real, symmetric matrix is known as **spectral decomposition** (special case of **eigendecomposition**)

Eigenvectors and Eigenvalues of the covariance matrix

- The eigenvector equation for the covariance matrix is:

$$\Sigma \mathbf{u}_i = \lambda_i \mathbf{u}_i, \quad (5)$$

where $i = 1, \dots, N$.

- Here, \mathbf{u}_i and λ_i represent the i^{th} eigenvector and eigenvalue, respectively
- Any N -dim real, symmetric matrix such as Σ has N real eigenvalues $\lambda_{i=1\dots N}$
- And the eigenvectors corresponding to these eigenvalues can be chosen to be orthonormal, i.e.

$$\mathbf{u}_i^T \mathbf{u}_j = I_{ij}, \quad (6)$$

where I_{ij} is the (i,j) element of \mathbb{I} , an N -dimensional identity matrix and $I_{ij} = 1$ if $i = j$.

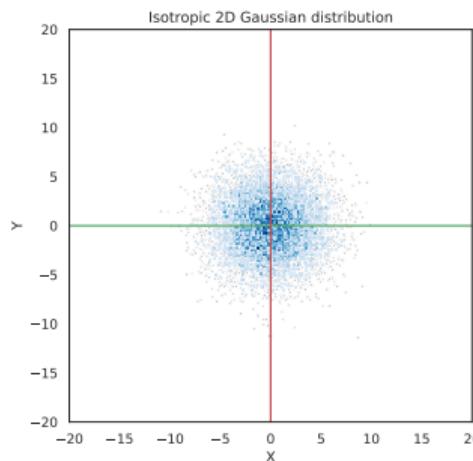
Eigenvectors and Eigenvalues of the covariance matrix

- Geometric representation of the eigenvectors \mathbf{u}_i and eigenvalues λ_i of Σ
- As before lets consider 2D Gaussian distributions with the following covariances:

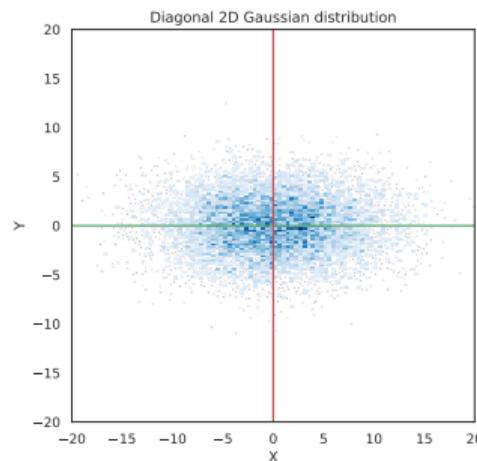
$$(a) \Sigma_1 = \begin{bmatrix} 8.0 & 0.0 \\ 0.0 & 8.0 \end{bmatrix},$$

$$(b) \Sigma_2 = \begin{bmatrix} 32.0 & 0.0 \\ 0.0 & 8.0 \end{bmatrix},$$

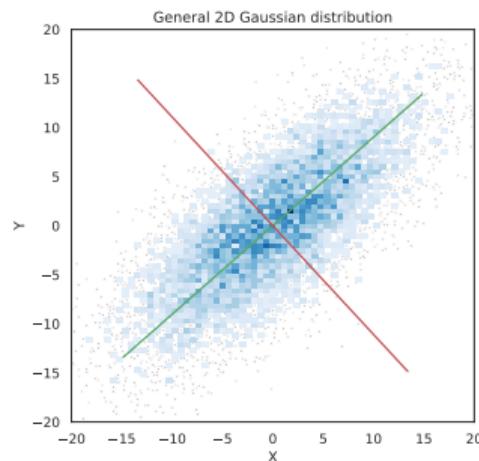
$$(c) \Sigma_3 = \begin{bmatrix} 51.0 & 35.0 \\ 35.0 & 43.0 \end{bmatrix}$$



(a) Eigvecs of isotropic Gaussian distribution



(b) Eigvecs of diagonal Gaussian distribution



(c) Eigvecs of full Gaussian distribution

Eigenvectors and Eigenvalues of the covariance matrix

- Now, let's introduce a new set of coordinates for the data in the new coordinate system spanned by $\mathbf{u}_{i=1\dots N}$;

$$y_i = \mathbf{u}_i^T (\mathbf{x} - \boldsymbol{\mu}) \quad (7)$$

- y_i are **new** coordinates which are translated and rotated with respect to the **old** \mathbf{x}
- The new coordinate system here is defined by the **orthonormal vectors** \mathbf{u}_i
- In matrix form, for $\mathbf{y} = (y_1, \dots, y_N)$ this can be expressed as:

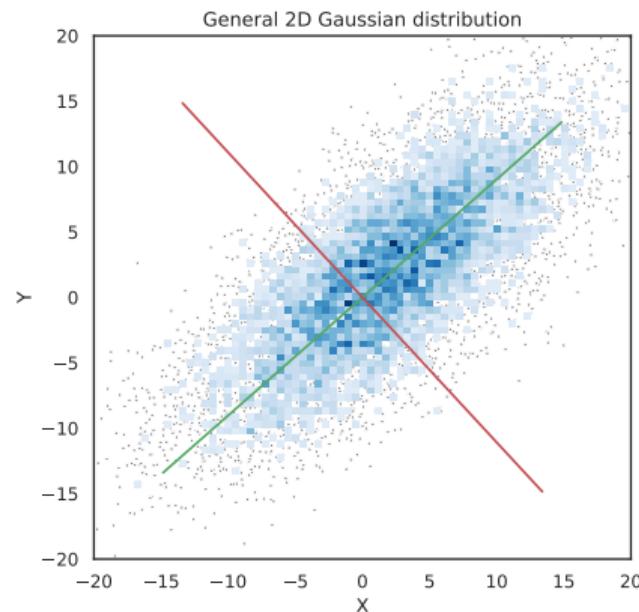
$$\mathbf{y} = \mathbf{U}(\mathbf{x} - \boldsymbol{\mu}) \quad (8)$$

where, rows of **orthogonal** matrix \mathbf{U} are given by \mathbf{u}_i^T and $\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbb{I}$

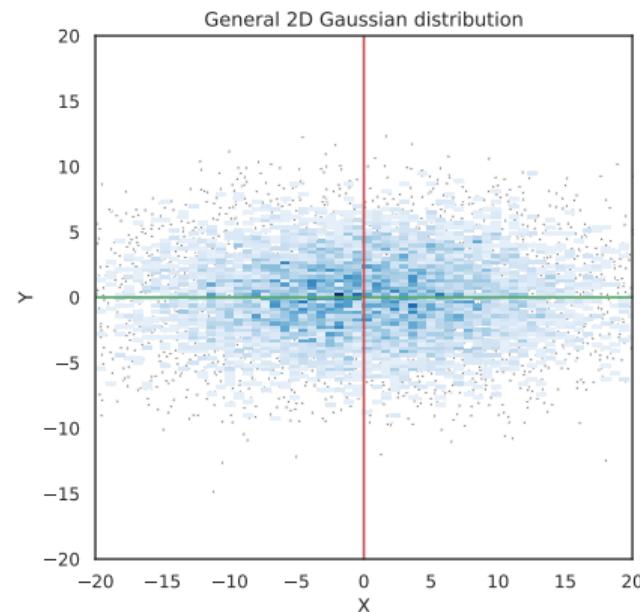
- Can you describe what is happening in equation 8?

Eigenvectors and Eigenvalues of the covariance matrix

Lets look at an example of coordinate transformation through a change in basis vectors:



(a) Original Gaussian distribution



(b) After co-ordinate transformation with \mathbf{U}

Spectral decomposition of the covariance matrix

- Distribution is centered on the origin and the rotation aligns the coordinate axes with the eigenvectors
- This is possible because the eigenvectors are orthonormal
- We can express the covariance matrix Σ as an expansion in terms of its eigenvectors
- Matrix form of eigenvector equation is:

$$\Sigma \mathbf{U} = \mathbf{U} \Lambda \quad (9)$$

$$\Sigma = \sum_{i=1}^N \lambda_i \mathbf{u}_i \mathbf{u}_i^T, \quad \Sigma^{-1} = \sum_{i=1}^N \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T \quad (10)$$

- Or in matrix form as:

$$\Sigma = \mathbf{U} \Lambda \mathbf{U}^T, \quad \Sigma^{-1} = \mathbf{U} \Lambda^{-1} \mathbf{U}^T \quad (11)$$

- This is known as the spectral decomposition theorem
- So why is this important/relevant?

Spectral decomposition of the covariance matrix

- Remember the functional dependence of \mathcal{N} on \mathbf{x} :

$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \quad (12)$$

- Substituting the eigenvector expansion of $\boldsymbol{\Sigma}^{-1}$ in equation 12:

$$\Delta^2 = \sum_{i=1}^N \frac{y_i^2}{\lambda_i} \quad (13)$$

where, we have previously defined $y_i = \mathbf{u}_i^T (\mathbf{x} - \boldsymbol{\mu})$

- We see now that $\boldsymbol{\Sigma}$ has been *diagonalised* as: $\text{diag}(\lambda_1, \dots, \lambda_N)$
- And for a diagonal covariance matrix, the distribution **factorises**... meaning?

Spectral decomposition of the covariance matrix

- The multivariate Gaussian distribution can be expressed as the product of N independent univariate Gaussian distributions in the new coordinate system:

$$p(\mathbf{y}) = \prod_{i=1}^N \frac{1}{(2\pi\lambda_i)^{1/2}} \exp\left\{-\frac{y_i^2}{2\lambda_i}\right\} \quad (14)$$

- Spectral decomposition of Σ has several useful applications in pattern recognition and machine learning
- Common techniques that utilise it include - Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), regression models, synthetic data generation algorithms, unsupervised clustering, etc.

Maximum Likelihood Estimation (MLE) for Multivariate Gaussian Distribution

- Parameters to estimate when fitting a multivariate Gaussian to data are: μ, Σ
- Lets look at deriving MLE for μ, Σ note: we will not assess you on the derivation itself!
- Given a dataset $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)^T$ of N i.i.d data points each forming a row of the matrix,
- Using the product rule we define the likelihood function as:

$$p(\mathbf{X}|\mu, \Sigma) = \prod_{i=1}^N p(\mathbf{x}_i|\mu, \Sigma) \quad (15)$$

- And we know that for a single data point \mathbf{x}_i we have:

$$p(\mathbf{x}_i|\mu, \Sigma) = \mathcal{N}(\mathbf{x}_i | \mu, \Sigma) = \frac{1}{(2\pi)^{N/2}} \frac{1}{|\det(\Sigma)|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) \right\} \quad (16)$$

Maximum Likelihood Estimation (MLE) for Multivariate Gaussian Distribution

- To derive MLE for μ & Σ we need to maximise the likelihood function w.r.t each parameter
- I.e. we need to solve:

$$\frac{\partial}{\partial \mu} p(\mathbf{X} | \mu, \Sigma) |_{\mu=\mu_{ML}} = 0 \quad (17a)$$

$$\frac{\partial}{\partial \Sigma} p(\mathbf{X} | \mu, \Sigma) |_{\Sigma=\Sigma_{ML}} = 0 \quad (17b)$$

- Easier to work with the log-likelihood function (\mathcal{L}):

$$\mathcal{L} = \ln p(\mathbf{X} | \mu, \Sigma) = -\frac{ND}{2} \ln(2\pi) - \frac{N}{2} \ln \det(\Sigma) - \frac{1}{2} \sum_{i=1}^N (\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) \quad (18)$$

Maximum Likelihood Estimation (MLE) for Multivariate Gaussian Distribution

- Considering just the terms in (\mathcal{L}) dependent on μ & Σ :

$$\mathcal{L}(\mu) = \sum_{i=1}^N (\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) \quad (19a)$$

$$\mathcal{L}(\Sigma) = -\frac{N}{2} \ln \det(\Sigma) - \frac{1}{2} \sum_{i=1}^N (\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) \quad (19b)$$

- Solving $\frac{\partial \mathcal{L}(\mu)}{\partial \mu} = 0$ & $\frac{\partial \mathcal{L}(\Sigma)}{\partial \Sigma} = 0$ gives:

$$\mu_{ML} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \quad (20a)$$

$$\Sigma_{ML} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T \quad (20b)$$

Practical applications

Spectral decomposition:

- PCA: Form of unsupervised learning for dimensionality reduction - under **Gaussian assumption**
- Core idea:
 - Identify ‘principal components’ or ‘eigenvectors’ of your data covariance matrix
 - Find **low-dimensional** representation of data, i.e. a **new** co-ordinate system based on a basis transformation
- **Sound familiar?**
- PCA used for data compression, feature extraction/selection, data analysis & visualisation
- Ex: Lets take brief look at PCA being used for analysing shape and its variation across a population: <http://www.cistib.org/~nravikumar/>

Summary

- Multivariate Gaussian distributions can have **isotropic, diagonal or full covariance matrices**
- Covariance matrix tells you whether variables are **uncorrelated or correlated**
- Spectral decomposition allows the covariance matrix to be **diagonalised**
- **Eigenvectors and eigenvalues** of covariance matrix define the axes and scaling of elliptical contours of constant density for Gaussians
- **Diagonalising** the covariance matrix allows multivariate Gaussian distributions to be expressed as a **product of independent univariate Gaussian distributions**

COMP5611M: Machine Learning

Linear Regression

Nishant Ravikumar
Marc de Kamps

School of Computing

February 12, 2022



Lecture Objectives

Objectives for this lecture:

- Introduction to linear regression and OLS
- Modelling non-linear relationships via linear regression
- Evaluating regression models, over-/underfitting
- Gaussian interpretation of linear regression

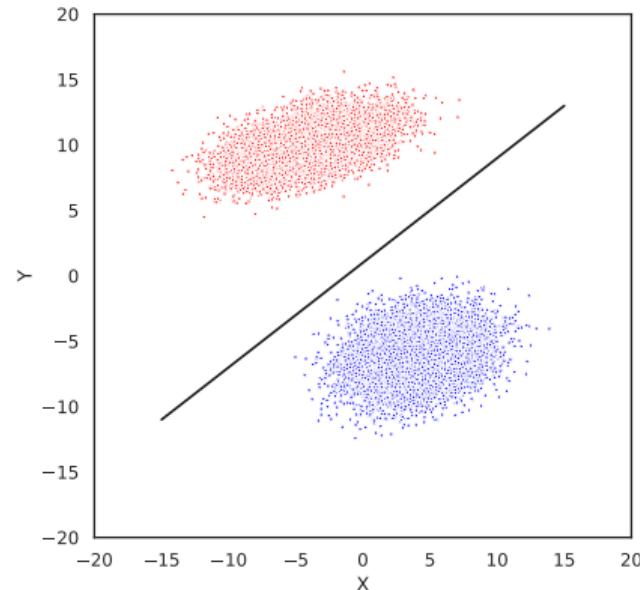
Learning Outcomes

At the end of this lecture, you should be able to

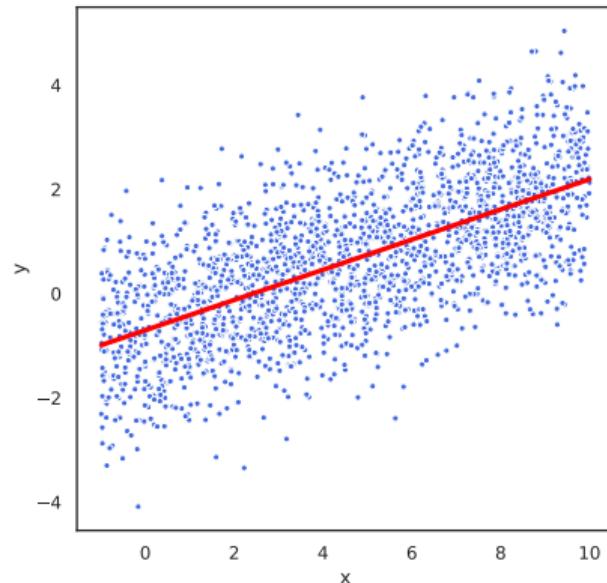
- Explain what linear regression is and the OLS method
- Express design matrix for any combination of basis functions
- Understand concept of over-/underfitting and be able to evaluate regression models
- Explain the relationship between MLE and least-squares solution to linear regression

Classification vs Regression

What are they?



(a) Classification



(b) Regression

Linear Regression

- Assumes a linear functional relationship between the predictor and target variables:
 $y = f(x, \omega) + \epsilon$
- A learned regression model can be used to *predict y* for new values of *x*
- Linear and non-linear *basis functions* can be used to construct a linear regression function
- So why is it called linear regression?

Ex: $y = \omega_0 + \omega_1 x + \omega_2 x^2 + \omega_3 x^3 + \epsilon$

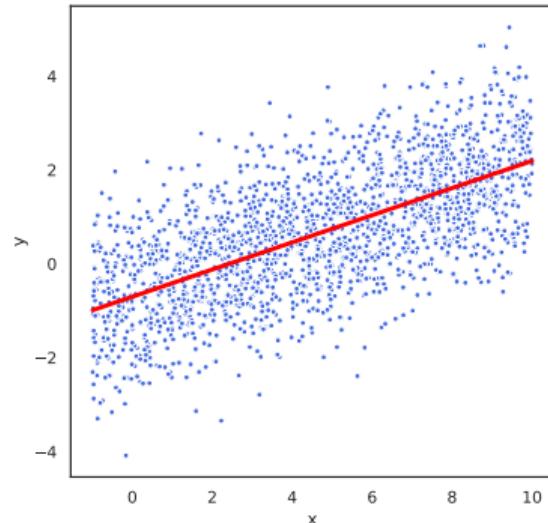


Figure: Linear regression: polynomial of degree 1

Linear Regression

- Let's say we have training data comprising $\{x_i, y_i\}_{i=1..N}$ pairs
- x_i : BMI & y_i : cardiac outputs (ejection fraction)
- We want to fit a polynomial of **degree 1** to the data such that

$$\hat{y}_i = \omega_0 + \omega_1 x_i \quad (1)$$

- Where, \hat{y}_i is the *predicted* value for y_i
- ω_0 : **intercept** and ω_1 : **slope**
- So how do we estimate ω_0, ω_1 ?

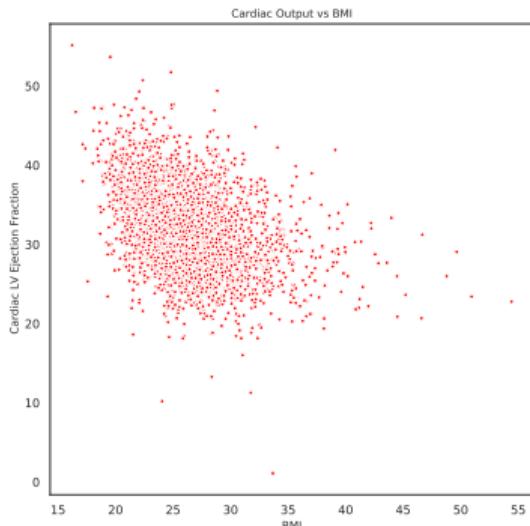


Figure: Cardiac output vs BMI

Ordinary Least Squares (OLS)

- First define the residual for a single data point i as:

$$r_i = (y_i - (\omega_0 + \omega_1 x_i)) = (y_i - \hat{y}_i) \quad (2)$$

- Key idea in OLS - find ω_0, ω_1 that minimises the sum of squared residuals:

$$R(\omega) = \sum_{i=1}^N r_i^2 = \sum_{i=1}^N (y_i - \hat{y}_i(x_i, \omega))^2 \quad (3)$$

- So how do we minimise $R(\omega)$?

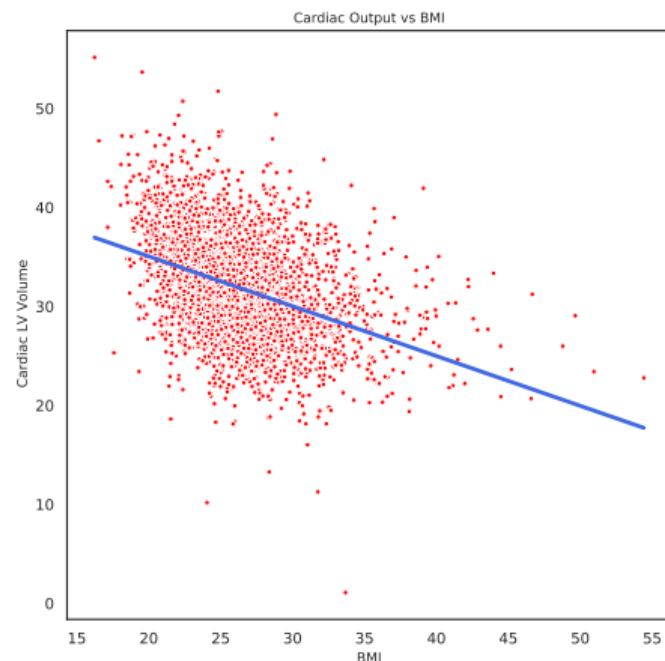


Figure: Cardiac output vs BMI

Ordinary Least Squares (OLS)

- Setting derivates of R w.r.t ω to 0:

$$\frac{\partial R(\omega_0, \omega_1)}{\partial \omega_0} = 0 \quad (4a)$$

$$\frac{\partial R(\omega_0, \omega_1)}{\partial \omega_1} = 0 \quad (4b)$$

- This results in a linear system of two equations with two unknowns (ω_0, ω_1) :

$$\sum_{i=1}^N x_i y_i = \omega_0 \sum_{i=1}^N x_i + \omega_1 \sum_{i=1}^N x_i^2 \quad (5a)$$

$$\sum_{i=1}^N y_i = \omega_0 N + \omega_1 \sum_{i=1}^N x_i \quad (5b)$$

Ordinary Least Squares (OLS)

- Expressing the system of equations in matrix form:

$$\begin{pmatrix} \sum_i^N x_i y_i \\ \sum_i^N y_i \end{pmatrix} = \begin{pmatrix} \sum_i^N x_i & \sum_i^N x_i^2 \\ N & \sum_i^N x_i \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \end{pmatrix} \quad (6a)$$

$$\begin{pmatrix} \omega_0 \\ \omega_1 \end{pmatrix} = \begin{pmatrix} \sum_i^N x_i & \sum_i^N x_i^2 \\ N & \sum_i^N x_i \end{pmatrix}^{-1} \begin{pmatrix} \sum_i^N x_i y_i \\ \sum_i^N y_i \end{pmatrix} \quad (6b)$$

- Note: order of summation in regression function affects row and column ordering in matrix-vector formulation
- Equation 6b is the OLS solution for ω_0, ω_1 for a degree= 1 polynomial regression function

OLS: Design Matrix

- So how can we generalise this to arbitrary degree polynomials?
- Lets redefine our regression function as:

$$\hat{y}_i(x_i, \omega) = \sum_{j=0}^M \omega_j \phi_j(x_i) = \omega^T \phi(x_i) \quad (7)$$

- For a degree=2 polynomial we have: $\hat{y}_i(x_i, \omega) = \omega_0 + \omega_1 x_i + \omega_2 x_i^2$

$$\omega = \begin{pmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{pmatrix}, \quad \phi(x_i) = \begin{pmatrix} \phi_0(x_i) \\ \phi_1(x_i) \\ \phi_2(x_i) \end{pmatrix} = \begin{pmatrix} 1 \\ x_i \\ x_i^2 \end{pmatrix} \quad (8)$$

- Where we see that the basis functions in $\phi(x_i)$ are given by monomials

OLS: Design Matrix

- Similarly for polynomials of arbitrary degree we have:

$$\hat{y}_i(x_i, \omega) = \sum_{j=0}^M \omega_j \phi_j(x_i) = \omega^T \phi(x_i) \quad (9a)$$

$$\omega = \begin{pmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_M \end{pmatrix}, \quad \phi = \begin{pmatrix} \phi_0(x_i) \\ \phi_1(x_i) \\ \vdots \\ \phi_M(x_i) \end{pmatrix} \quad (9b)$$

- As before ω is the weights vector; ϕ is the *basis functions* vector
- Each $\phi_j(x_i)$ is a basis function ; $\phi_0(x_i) = 1$ is a dummy basis function for ω_0 (intercept)
- This form is generalisable to other types of basis functions as well

OLS: Design Matrix

- Fitting an M -degree polynomial requires estimation of $M + 1$ parameters in ω
- Using ω, ϕ we can define $R(\omega)$ as: $R(\omega) = \sum_{i=1}^N (y_i - \omega^T \phi(x_i))^2$
- As before we can estimate ω_{OLS} by minimising $R(\omega)$ w.r.t ω ; $\frac{\partial R(\omega)}{\partial \omega} = 0$

$$\frac{\partial R(\omega)}{\partial \omega} = \sum_{i=1}^N (y_i - \omega^T \phi(x_i)) \phi(x_i)^T = 0, \quad (10a)$$

$$\sum_{i=1}^N y_i \phi^T(x_i) = \omega^T \left(\sum_{i=1}^N \phi(x_i) \phi^T(x_i) \right) \quad (10b)$$

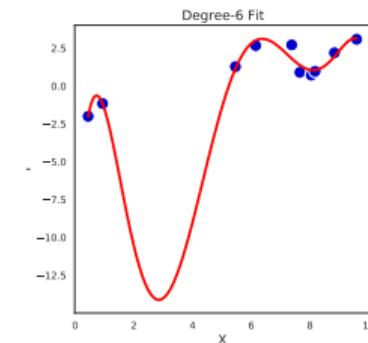
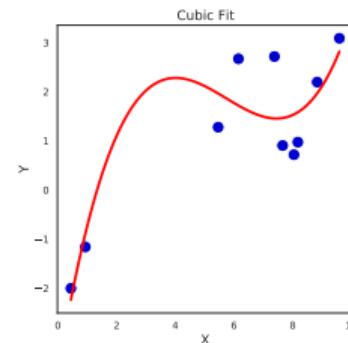
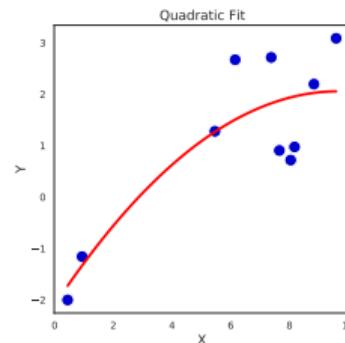
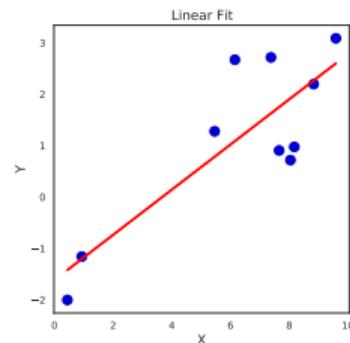
- Equation 10b can be expressed in matrix-vector form by defining the *design matrix* Φ
- Φ is an $N \times (M + 1)$ matrix where each x_i has an associated basis function vector $\phi(x_i)$

OLS: Estimating regression weights

- By defining design matrix $\Phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_M(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_M(x_2) \\ \vdots & \vdots & & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_M(x_N) \end{pmatrix}$
 - ω_{OLS} is given by solving: $\Phi^T \mathbf{y} = \Phi^T \Phi \omega$
- $$\omega_{OLS} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} \quad (11)$$
- $(\Phi^T \Phi)^{-1} \Phi^T \equiv \Phi^\dagger$ is also known as the *Moore-Penrose pseudoinverse* of Φ
 - Directly computing these matrix operations can lead to numerical instability - solved by using SVD to compute Φ^\dagger

Regression Examples

- Given some training data $\{x_i, y_i\}_{i=1..N}$ with limited number of samples ($N = 10$):



(a) Polynomial degree=1

(b) Polynomial degree=2

(c) Polynomial degree=3

(d) Polynomial degree=6

- So how do we decide which polynomial provides the **best** fit?

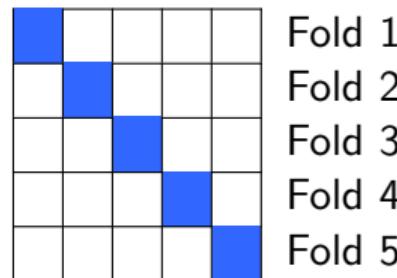
Evaluating Regression Models

- Common **metrics** for evaluating regression models:
 - Coefficient of determination or $R^2 = 1 - \frac{\sum_i(y_i - \hat{y}_i)^2}{\sum_i(y_i - \bar{y})^2}$; \bar{y} is the mean of the observed targets
 - Mean absolute error (MAE) = $\frac{1}{N} \sum_i^N |y_i - \hat{y}_i|$
 - Mean squared error (MSE) = $\frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$
 - Root mean squared error (RMSE) = $\sqrt{\frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2}$
 - ... and **several others!**
- Are **summary** evaluation metrics such as these enough?
- Ok, we have some evaluation metrics.. but how do we compare regression models?

Evaluating Regression Models

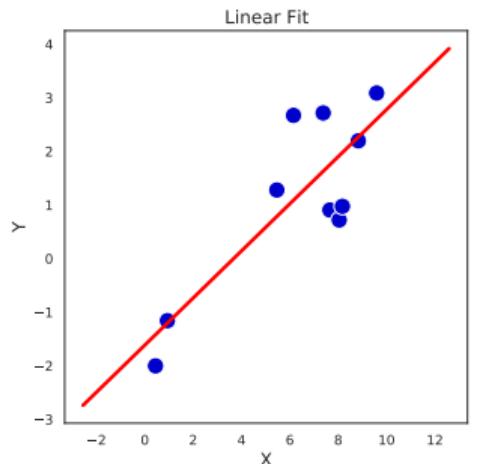
- First - we need an independent *validation* set
- Evaluate trained regression models on the *validation* set and compare fits
- But is that enough? What if we switched around the train-val samples?
- For small data sets *cross-validation* is necessary for thorough evaluation
- Conducting k -fold cross-validation experiments is pretty standard practice in ML:

Table: Example of train-val split for 5-fold cross-validation

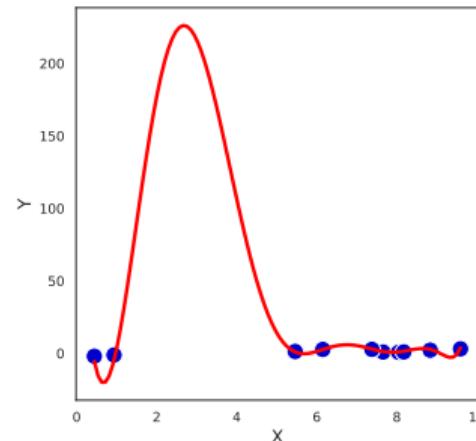


Evaluating Regression Models: Over- & Underfitting

- Over- & underfitting are common issues encountered in ML
- Lets take a look at these phenomena in the context of regression



(a) Polynomial degree=1

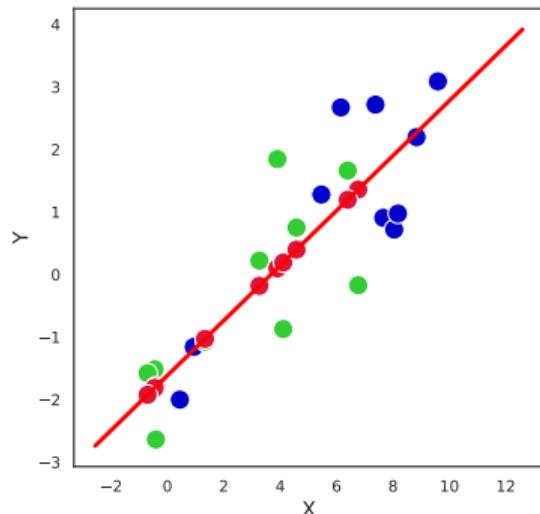


(b) Some higher degree polynomial

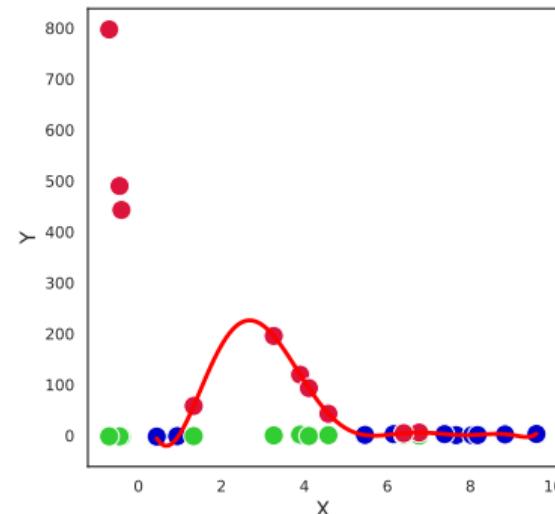
- Function in (b) fits all points exactly.. but common sense tells us something is off

Evaluating Regression Models: Over- & Underfitting

- As discussed, cannot evaluate predictive performance based on training data
- What happens if we evaluate these regression models on new data? **Overfitting!**



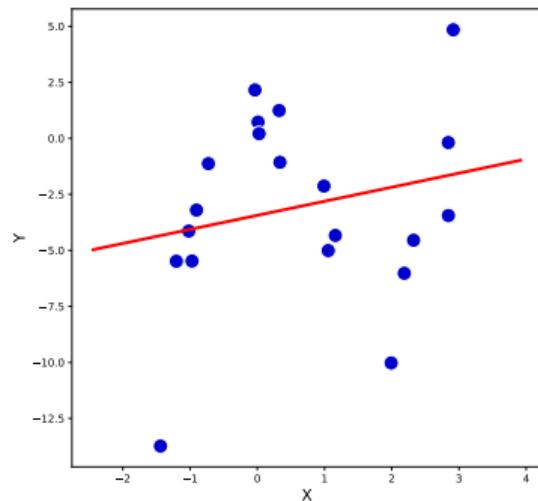
(a) Degree=1 with new points (green) and predictions (red)



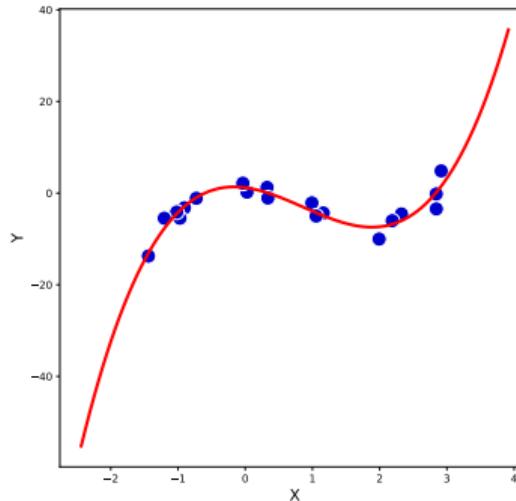
(b) Some higher degree polynomial with new points (green) and predictions (red)

Evaluating Regression Models: Over- & Underfitting

- What about underfitting? What might this look like?



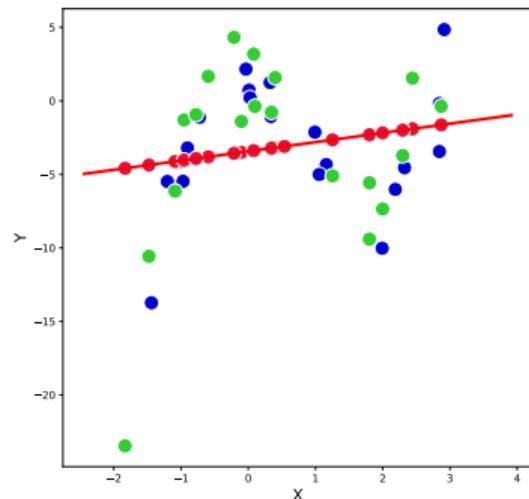
(a) Polynomial degree=1



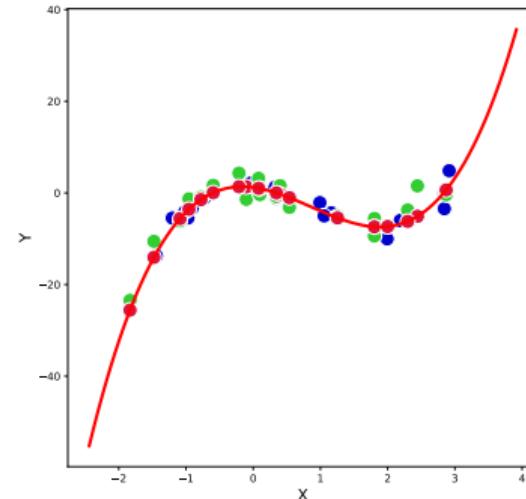
(b) Some higher degree polynomial

Evaluating Regression Models: Over- & Underfitting

- Effect of underfitting apparent when regression models are applied to new data



(a) Degree=1 with new points (green) and predictions (red)



(b) Some higher degree polynomial with new points (green) and predictions (red)

Evaluating Regression Models

- So we've seen the effects of **over-/underfitting**
- And looked at evaluation metrics and the need for **cross-validation** to mitigate these
- But are these the best/only options? What **drawbacks** can you see to this approach?



Figure: Source: Wikipedia, CC BY 2.0

- Need a better way to balance model complexity and predictive power..

Linear Regression: Probabilistic View

- Cross-validation can increase number of experiments drastically for large ML models
- Need a better approach where **model complexity** is not chosen by cross-validation
- This brings us to a Bayesian treatment of linear regression
- But before we get there, lets look at OLS again:

$$R(\omega) = \sum_{i=1}^N (y_i - \hat{y}_i(x_i, \omega))^2 \quad (12a)$$

$$\arg \min_{\omega} R(\omega) \quad (12b)$$

- How did we end up with this cost function $R(\omega)$?

Linear Regression: Probabilistic View

- Consider again the linear relationship between target y & predictor x :

$$\mathbf{y} = \hat{y}(\mathbf{x}, \boldsymbol{\omega}) + \epsilon \quad (13)$$

where, ϵ is some *unexplained* noise; $\hat{y}(\cdot)$ is a linear combination of basis functions

- Lets assume ϵ is a univariate Gaussian variable with zero mean and precision $\beta = \frac{1}{\sigma^2}$
 - β used for convenience of notation
- Then the probability distribution of a target y *conditioned* on $\mathbf{x}, \boldsymbol{\omega}$ is given by:

$$p(y | \mathbf{x}, \boldsymbol{\omega}, \beta) = \mathcal{N}(y | \hat{y}(\mathbf{x}, \boldsymbol{\omega}), \beta^{-1}) \quad (14)$$

Linear Regression: MLE & OLS

- Given a data set $\mathbf{X} = \{\mathbf{x}_i\}_{i=1\dots N}$ & targets $\mathbf{y} = \{y_i\}_{i=1\dots N}$, assuming data are *i.i.d.*,
- We can formulate the likelihood function as a function of ω, β :

$$p(\mathbf{y} | \mathbf{X}, \omega, \beta) = \prod_{i=1}^N \mathcal{N}(y_i | \omega^T \phi(x_i), \beta^{-1}) \quad (15)$$

where we have used the previous result: $\hat{y}(x_i, \omega) = \omega^T \phi(x_i)$

- But we have seen this form before for **MLE of multivariate Gaussians...**
- From Equation 15 we can express the log-likelihood \mathcal{L} as:

$$\mathcal{L} = \ln p(\mathbf{y} | \mathbf{X}, \omega, \beta) = -\frac{ND}{2} \ln(2\pi) - \frac{N}{2} \ln \beta - \frac{1}{2} \sum_{i=1}^N \beta(y_i - \omega^T \phi(x_i))^2 \quad (16)$$

Linear Regression: MLE & OLS

- Looking at the quadratic term in \mathcal{L} :

$$\mathcal{L} = \ln p(\mathbf{y} | \mathbf{X}, \boldsymbol{\omega}, \beta) = -\frac{ND}{2} \ln(2\pi) - \frac{N}{2} \ln \beta - \frac{\beta}{2} R(\boldsymbol{\omega}) \quad (17a)$$

$$R(\boldsymbol{\omega}) = \sum_{i=1}^N (y_i - \boldsymbol{\omega}^T \phi(x_i))^2 = \sum_{i=1}^N (y_i - \hat{y}_i(x_i, \boldsymbol{\omega}))^2 \quad (17b)$$

- But this is the **same** as the **sum of squared residuals** in OLS!
- $\boldsymbol{\omega}$ which **maximises** \mathcal{L} also **minimises** R : $\arg \max_{\boldsymbol{\omega}} \mathcal{L} \equiv \arg \min_{\boldsymbol{\omega}} (-\mathcal{L})$

$$\arg \min_{\boldsymbol{\omega}} (-\mathcal{L}(\boldsymbol{\omega})) = \arg \min_{\boldsymbol{\omega}} \sum_{i=1}^N (y_i - \hat{y}_i(x_i, \boldsymbol{\omega}))^2 = \arg \min_{\boldsymbol{\omega}} R(\boldsymbol{\omega}) \quad (18)$$

- So $\boldsymbol{\omega}_{MLE} = (\Phi^T \Phi)^{-1} \Phi^T$ under **additive Gaussian noise assumption**

Summary

- Linear and non-linear basis functions may be used to formulate a *linear regression function*
- OLS used to estimate linear regression weights by **minimising sum of squared residuals**
- OLS solution boils down to computing **pseudoinverse of the Design Matrix**
- OLS by itself does not consider any noise model, MLE interpretation assumes Gaussian noise

COMP5611M: Machine Learning

Bayesian Linear Regression

Nishant Ravikumar
Marc de Kamps

School of Computing

February 15, 2022



Lecture Objectives

Objectives for this lecture:

- Recap linear regression and relationship between MLE and OLS
- Bayesian inference for univariate Gaussian distributions
- Introduction to Bayesian inference for linear regression

Learning Outcomes

At the end of this lecture, you should be able to

- Explain difference between Bayesian inference and MLE
- Describe ‘completing the square’ for Bayesian inference with Gaussians

Recap: Linear Regression

- Assumed linear relationship between target y & predictor x :

$$y = \hat{y}(x, \omega) + \epsilon \quad (1)$$

where, ϵ is some *unexplained* noise;

- Assuming ϵ is a univariate Gaussian variable with zero mean and precision β
- Conditional distribution of y given x, ω is:

$$p(y | x, \omega, \beta) = \mathcal{N}(y | \omega^T \phi(x_i), \beta^{-1}) \quad (2)$$

where as previously: $\hat{y}(x_i, \omega) = \omega^T \phi(x_i)$

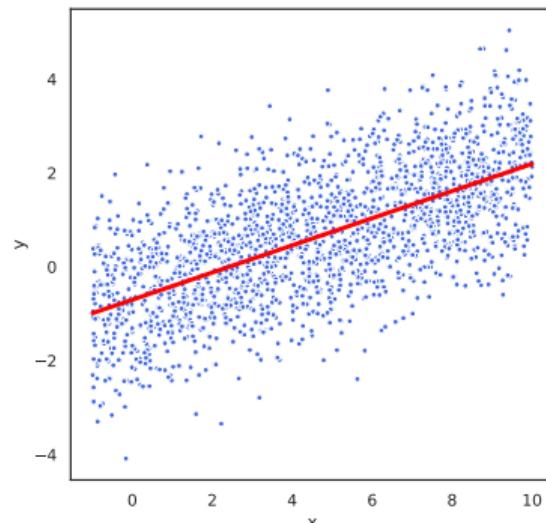


Figure: Linear regression: polynomial of degree 1

Linear Regression: MLE & OLS

- Given a data set $\mathbf{X} = \{\mathbf{x}_i\}_{i=1\dots N}$ & targets $\mathbf{y} = \{y_i\}_{i=1\dots N}$, assuming data are *i.i.d.*,
- We can formulate the likelihood function as a function of ω, β :

$$p(\mathbf{y} | \mathbf{X}, \omega, \beta) = \prod_{i=1}^N \mathcal{N}(y_i | \omega^T \phi(x_i), \beta^{-1}) \quad (3)$$

- But we have seen this form before for **MLE of multivariate Gaussians...**
- From Equation 3 we can express the log-likelihood \mathcal{L} as:

$$\mathcal{L} = \ln p(\mathbf{y} | \mathbf{X}, \omega, \beta) = -\frac{ND}{2} \ln(2\pi) - \frac{N}{2} \ln \beta - \frac{1}{2} \sum_{i=1}^N \beta(y_i - \omega^T \phi(x_i))^2 \quad (4)$$

Linear Regression: MLE & OLS

- Looking at the quadratic term in \mathcal{L} :

$$\mathcal{L} = \ln p(\mathbf{y} | \mathbf{X}, \boldsymbol{\omega}, \beta) = -\frac{ND}{2} \ln(2\pi) - \frac{N}{2} \ln \beta - \frac{\beta}{2} R(\boldsymbol{\omega}) \quad (5a)$$

$$R(\boldsymbol{\omega}) = \sum_{i=1}^N (y_i - \boldsymbol{\omega}^T \phi(x_i))^2 = \sum_{i=1}^N (y_i - \hat{y}_i(x_i, \boldsymbol{\omega}))^2 \quad (5b)$$

- But this is the **same** as the **sum of squared residuals** in OLS!
- $\boldsymbol{\omega}$ which **maximises** \mathcal{L} also **minimises** R : $\arg \max_{\boldsymbol{\omega}} \mathcal{L} \equiv \arg \min_{\boldsymbol{\omega}} (-\mathcal{L})$

$$\arg \min_{\boldsymbol{\omega}} (-\mathcal{L}(\boldsymbol{\omega})) = \arg \min_{\boldsymbol{\omega}} \sum_{i=1}^N (y_i - \hat{y}_i(x_i, \boldsymbol{\omega}))^2 = \arg \min_{\boldsymbol{\omega}} R(\boldsymbol{\omega}) \quad (6)$$

- So $\boldsymbol{\omega}_{MLE} = (\Phi^T \Phi)^{-1} \Phi^T$ under **additive Gaussian noise assumption**

Limitations of MLE

- From the unfair coin toss example we have seen that **MLE is prone to overfitting**
- In linear regression we have seen:
 - There exists an MLE interpretation of OLS
 - Choosing model complexity for a given regression task can be computationally expensive
 - Prefer to automatically control model complexity and reduce **over-/underfitting**
- MLE for Gaussians are also prone to overfitting as with Bernoulli
- So what can we do?

Recap: Bayes' Law

- Using the sum and product rules of probability for discrete values of θ (or events):

$$P(\theta | X) = \frac{P(X | \theta)P(\theta)}{P(X)} \quad (7)$$

where, the marginal probability $P(X) = \sum_{\theta} P(X | \theta)P(\theta)$

- For continuous variables, i.e. all possible values of θ we can do the same:

$$p(\theta | X) = \frac{p(X | \theta)p(\theta)}{p(X)} \quad (8)$$

except now, $p(\theta)$ is a **continuous prior distribution** rather than **discrete** probabilities

- Now the marginal probability density function $p(X) = \int p(X|\theta)p(\theta)d\theta$
- So we can say: *posterior \propto likelihood \times prior*

Bayesian vs Frequentist

- Key difference in Frequentist vs Bayesian paradigms:
 - Frequentist: model parameters θ are **fixed**; computed using some *estimator* such as **MLE**
 - Confidence in estimates for θ evaluated through multiple experiments (**cross-validation**) for different data sets
 - Bayesian: model parameters θ are **random variables**
 - Here, there is only **one** data set (actually observed) and **uncertainty in θ** is expressed as a probability distribution over θ

Bayesian inference for univariate Gaussian

- Given a dataset $\mathbf{X} = \{x_i\}_{i=1..N}$ of N independent Gaussian random variables
- Assume σ^2 is known and μ is unknown
- We know the likelihood function is:

$$p(\mathbf{X} | \mu) = \prod_{i=1}^N p(x_i | \mu) \quad (9a)$$

$$p(x_i | \mu) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \left\{ -\frac{1}{2\sigma^2} (x_i - \mu)^2 \right\} \quad (9b)$$

$$\prod_{i=1}^N p(x_i | \mu) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^N (x_i - \mu)^2 \right\} \quad (9c)$$

Bayesian inference for univariate Gaussian

- Remember the likelihood $p(\mathbf{X} | \mu)$ is *not* a probability distribution and is not normalised
- Likelihood function is an exponential of a quadratic form of μ
- So what if we choose our **prior** $p(\mu)$ to also be a Gaussian?
- $p(\mu)$ is then a **conjugate** to the likelihood function
- As **posterior** \propto **likelihood** \times **prior**, posterior distribution will also be Gaussian
- But how can this be? We have not computed the marginal distribution and **normalised** the posterior yet.. **Any ideas?**

Bayesian inference: Completing the Square

- Given just a single data point such that $p(x_1 | \mu) \sim \mathcal{N}(\mu, \sigma^2)$, and prior $p(\mu) \sim \mathcal{N}(\mu_0, \sigma_0^2)$
- Remember σ^2 is known and μ is unknown

$$p(\mu | x_1) \propto p(x_1 | \mu)p(\mu) \propto \exp \left\{ -\frac{(x_1 - \mu)^2}{2\sigma^2} - \frac{(\mu - \mu_0)^2}{2\sigma_0^2} \right\} \quad (10a)$$

$$p(\mu | x_1) \propto \exp \left\{ \frac{-1}{2} \left\{ \mu^2 \left[\frac{1}{\sigma^2} + \frac{1}{\sigma_0^2} \right] - 2\mu \left[\frac{x_1}{\sigma^2} + \frac{\mu_0}{\sigma_0^2} \right] + const \right\} \right\} \quad (10b)$$

- By completing the square we can express this as:

$$p(\mu | x_1) \propto \exp \left\{ \frac{-1}{2\sigma_{post}^2} (\mu - \mu_{post})^2 \right\}, \mu_{post} = \sigma_{post}^2 \left\{ \frac{x_1}{\sigma^2} + \frac{\mu_0}{\sigma_0^2} \right\}, \sigma_{post}^2 = \frac{\sigma_0^2 \sigma^2}{\sigma_0^2 + \sigma^2} \quad (11)$$

- So we can say $p(\mu | x_1) \sim \mathcal{N}(\mu_{post}, \sigma_{post}^2)$

Bayesian inference for univariate Gaussian

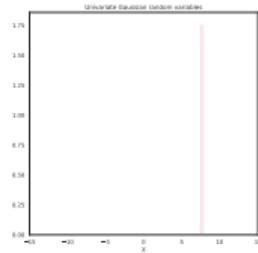
- Similarly, we can do this for N observations resulting in:

$$\frac{1}{\sigma_{post}^2} = \left\{ \frac{1}{\sigma_0^2} + \frac{N}{\sigma^2} \right\} = \frac{\sigma_0^2 \sigma^2}{N\sigma_0^2 + \sigma^2} \quad (12a)$$

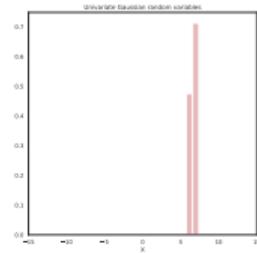
$$\mu_{post} = \left\{ \frac{N\sigma_0^2}{N\sigma_0^2 + \sigma^2} \mu_{ML} + \frac{\sigma^2}{N\sigma_0^2 + \sigma^2} \mu_0 \right\} \quad (12b)$$

- So we see that the μ_{post} is a weighted average of μ_0 and μ_{ML} given N observations
- In general, we can do the same for scenarios of unknown variance and known mean; unknown mean & unknown variance

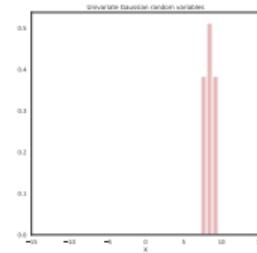
Bayesian inference for univariate Gaussian: Examples



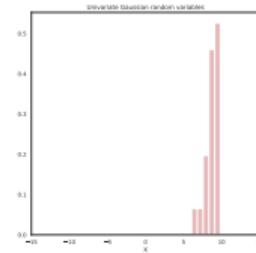
(a)



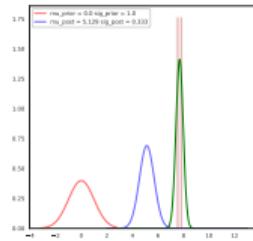
(b)



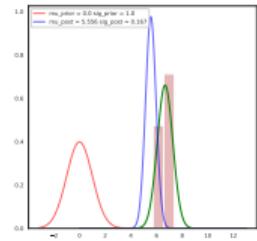
(c)



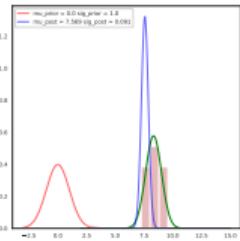
(d)



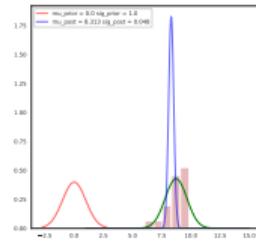
(e)



(f)



(g)



(h)

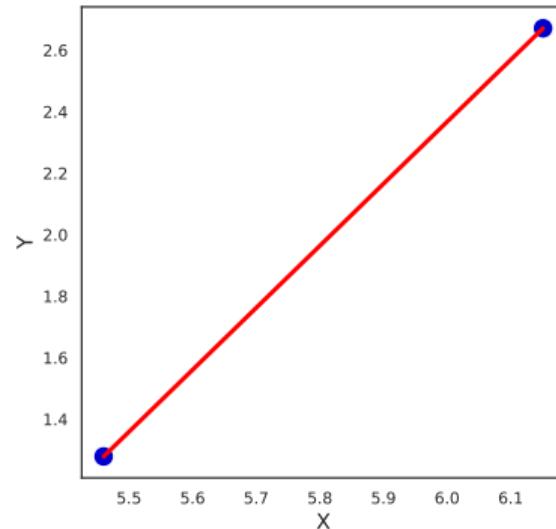
Figure: Data sampled from $\mathcal{N}(8.0, 1.0)$ - (a): $N = 2$, (b): $N = 5$, (c): $N = 10$, (d): $N = 20$; (e-h) posterior distributions for μ (blue); $\mu_{prior} \sim \mathcal{N}(0.0, 1.0)$ (red); likelihood function (green)

Linear Regression revisited: Bayesian perspective

- Consider linear regression with a polynomial of degree = 1; $\hat{y}_i = \omega_0 + \omega_1 x_i$
- If we have just two predictors and their targets: $\{x_i, y_i\}_{i=1,2}$, we can fit a line exactly

- Here we have two unknowns and two observations
- Can determine the slope (ω_1) and intercept (ω_0) just using the observations
- We can solve for ω_1 as follows and then estimate ω_0 :

$$\omega_1 = \frac{y_2 - y_1}{x_2 - x_1} \quad (13)$$



Linear Regression revisited: Bayesian perspective

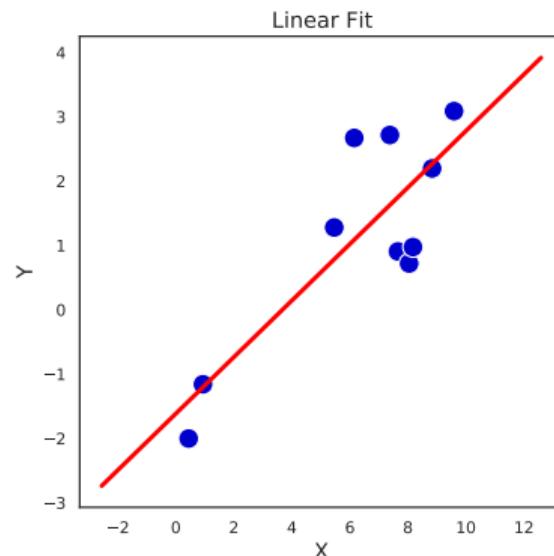
- But what if we had more than two points?

- Then we have an *overdetermined* system
- We can use OLS or assume a noise model
- Then system of linear equations are:

$$y_i = \omega_0 + \omega_1 x_i + \epsilon_i \quad (14)$$

where, we can assume $\epsilon \sim \mathcal{N}(0, \sigma^2)$

- And this leads to $\omega_{MLE} \equiv \omega_{OLS}$



Linear Regression revisited: Bayesian perspective

- What if we have an *underdetermined* system? E.g. just one observation
- We have two unknowns and one observation: $y_1 = \omega_0 + \omega_1 x_1 + \epsilon_1$
- What can we do if we want to fit a line to this?

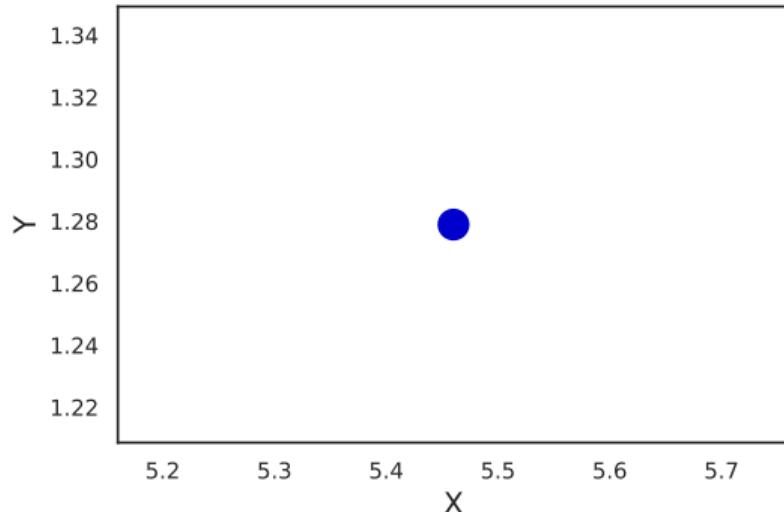


Figure: Just one observations (x_1, y_1)

Linear Regression revisited: Bayesian perspective

- What if we have an *underdetermined* system? E.g. just one observation
- We have two unknowns and one observation: $y_1 = \omega_0 + \omega_1 x_1 + \epsilon_1$
- We have an infinite number of possible solutions..
- How can we represent a **family** of ω_0 ?

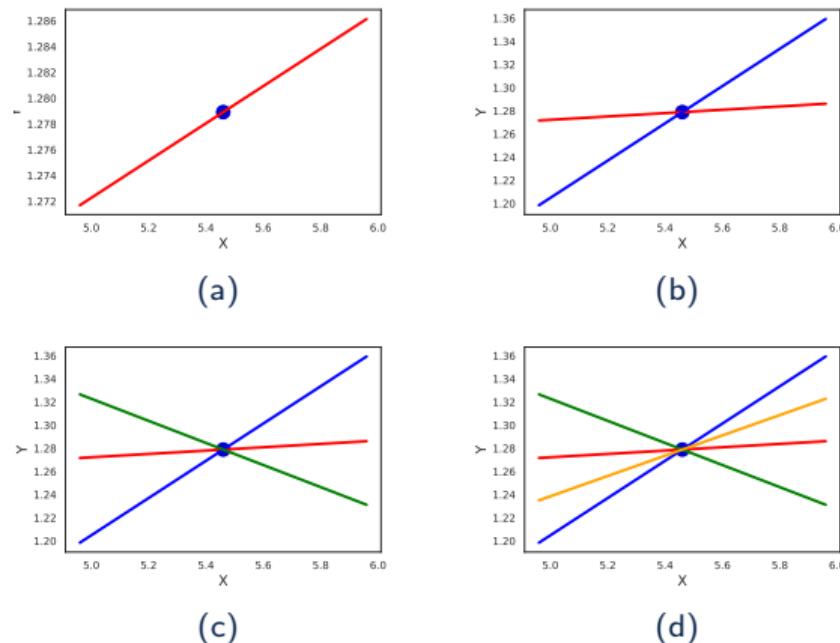


Figure: (a): $\omega_0 = 1.2$, (b): $\omega_0 = 0.4$, (c): $\omega_0 = 1.8$,
(d): $\omega_0 = 0.8$

Linear Regression revisited: Bayesian perspective

- We can make an **assumption for the distribution** of the unknown parameter ω_0 !
- And that is **Bayesian Inference**!
- Given a data set $\mathbf{X} = \{\mathbf{x}_i\}_{i=1\dots N}$ & targets $\mathbf{y} = \{y_i\}_{i=1\dots N}$, assuming data are *i.i.d.*,
- As previously the likelihood function is given by:

$$p(\mathbf{y} | \mathbf{X}, \omega, \beta) = \prod_{i=1}^N \mathcal{N}(y_i | \omega^T \phi(x_i), \beta^{-1}) \quad (15)$$

- We tried to **maximise the likelihood** with respect to ω
- Bayesian perspective: based on some **a priori belief** about ω , integrate across ω
- I.e. instead of fixed ω we look at expectation of likelihood for a range of plausible ω

Linear Regression revisited: Bayesian perspective

- For Bayesian inference we first define a **prior distribution** on the unknowns/parameters (ω)
- Prior represents our belief about ω **before we see the data**
- Linear regression: let's consider a Gaussian prior on the intercept (ω_0) (ignore ω_1 for now):

$$p(\omega_0) \sim \mathcal{N}(0, \alpha) \quad (16)$$

- Given N i.i.d observations $\{x_i, y_i\}_{i=1\dots N}$ and $p(\omega_0)$ the posterior distribution of ω_0 is:

$$p(\omega_0 | \mathbf{y}, \mathbf{x}, \omega_1, \sigma^2) \propto \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - \omega_1 x_i - \omega_0)^2 - \frac{\omega_0^2}{2\alpha} \right\} \quad (17a)$$

$$\ln p(\omega_0 | \mathbf{y}, \mathbf{x}, \omega_1, \sigma^2) = \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - \omega_1 x_i - \omega_0)^2 - \frac{\omega_0^2}{2\alpha} \right\} + const \quad (17b)$$

Linear Regression revisited: Bayesian perspective

- Once again by **completing the square** we find:

$$\ln p(\omega_0 \mid \mathbf{y}, \mathbf{x}, \omega_1, \sigma^2) = -\frac{1}{\sigma_{post}^2}(\omega_0 - \mu_{post})^2 + const \quad (18a)$$

$$p(\omega_0 \mid \mathbf{y}, \mathbf{x}, \omega_1, \sigma^2) \sim \mathcal{N}(\mu_{post}, \sigma_{post}^2) \quad (18b)$$

$$\mu_{post} = \frac{N\alpha}{N\alpha + \sigma^2}\mu_{ML}, \quad \sigma_{post}^2 = \frac{\sigma^2\alpha}{N\alpha + \sigma^2} \quad (18c)$$

- Where μ_{ML} is the **MLE for the intercept ω_0** (refer to C. Bishop's book pg. 142)
- For MLE we estimated all regression weights jointly
- We can infer the **posterior distribution over all weights** by assuming a **prior $p(\omega)$** over all - next lecture!

COMP5611M: Machine Learning

Bayesian Linear Regression (Part 2)

Nishant Ravikumar
Marc de Kamps

School of Computing

February 18, 2022



Lecture Objectives

Objectives for this lecture:

- Bayesian linear regression in detail with examples
- Regularisation in linear regression and its relationship to Bayesian inference

Learning Outcomes

At the end of this lecture, you should be able to

- Explain Bayesian inference procedure for linear regression weights
- Describe regularisation for linear regression models & relationship to Bayesian approach

Recap: Bayesian inference for univariate Gaussian

Given data $\mathbf{X} = \{x_i\}_{i=1\dots N}$ that are i.i.d Gaussian random variables

- For unknown mean (μ) and known variance σ^2 we can choose $p(\mu)$ to be a conjugate prior

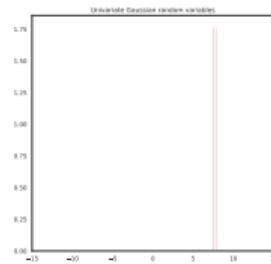
$$\text{i.e. } p(\mu) \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

- As $\text{posterior} \propto \text{likelihood} \times \text{prior}$, posterior distribution will also be Gaussian
- No need for integration - just need to complete the square
- And you end up with -

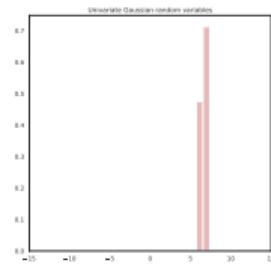
$$\frac{1}{\sigma_{\text{post}}^2} = \left\{ \frac{1}{\sigma_0^2} + \frac{N}{\sigma^2} \right\} = \frac{\sigma_0^2 \sigma^2}{N \sigma_0^2 + \sigma^2} \quad (1a)$$

$$\mu_{\text{post}} = \left\{ \frac{N \sigma_0^2}{N \sigma_0^2 + \sigma^2} \mu_{ML} + \frac{\sigma^2}{N \sigma_0^2 + \sigma^2} \mu_0 \right\} \quad (1b)$$

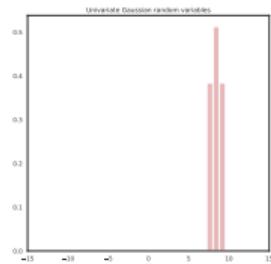
Recap: Bayesian inference for univariate Gaussian



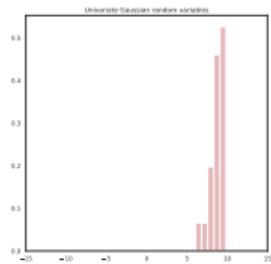
(a)



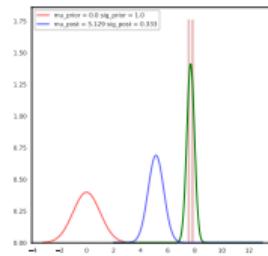
(b)



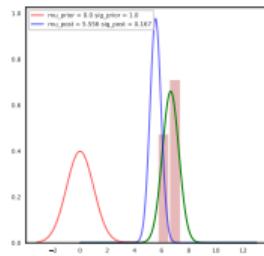
(c)



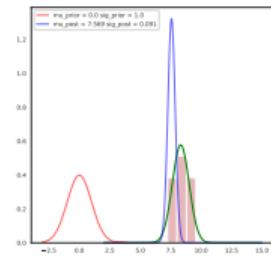
(d)



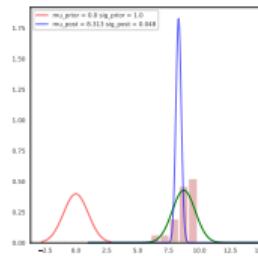
(e)



(f)



(g)



(h)

Figure: Data sampled from $\mathcal{N}(8.0, 1.0)$ - (a): $N = 2$, (b): $N = 5$, (c): $N = 10$, (d): $N = 20$; (e-h) posterior distributions for μ (blue); $\mu_{prior} \sim \mathcal{N}(0.0, 1.0)$ (red); likelihood function (green)

Bayesian Inference for Linear Regression

- Previously, we assumed a prior over the intercept for a linear regression function
- What about a **prior over all** regression weights?
- Given a data set N i.i.d observations: $\mathbf{y} = \hat{y}(\mathbf{x}, \boldsymbol{\omega}) + \boldsymbol{\epsilon}$

$$p(\mathbf{y} | \mathbf{x}, \boldsymbol{\omega}, \beta) = \prod_{i=1}^N \mathcal{N}(y_i | \boldsymbol{\omega}^T \phi(x_i), \beta^{-1}) \quad (2a)$$

$$p(\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega} | \mathbf{0}, \alpha^{-1} \mathbb{I}) \quad (2b)$$

- Then posterior distribution over weights is given by:

$$p(\boldsymbol{\omega} | \mathbf{y}, \mathbf{x}, \beta) \propto p(\mathbf{y} | \mathbf{x}, \boldsymbol{\omega}, \beta) p(\boldsymbol{\omega}) \quad (3)$$

- We will treat the precision β as a **known constant** - but this may be inferred as well

Bayesian Inference for Linear Regression

- We know if $p(\omega)$ is a **conjugate prior** then $p(\omega | \mathbf{y}, \mathbf{x})$ is a Gaussian distribution
- As seen previously, given $p(\omega) = \mathcal{N}(\omega | \mathbf{0}, \alpha^{-1}\mathbb{I})$:

$$\ln p(\mathbf{w} | \mathbf{y}, \mathbf{x}) = -\frac{\beta}{2} \sum_{i=1}^N \left\{ y_i - \mathbf{w}^T \phi(x_i) \right\}^2 - \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + \text{const} \quad (4)$$

- Once again by **completing the square**:

$$p(\omega | \mathbf{y}, \mathbf{x}) = \mathcal{N}(\omega | \mathbf{m}_{post}, \mathbf{S}_{post}) \quad (5a)$$

$$\mathbf{m}_{post} = \beta \mathbf{S}_{post} \Phi^T \mathbf{y}, \quad \mathbf{S}_{post}^{-1} = \alpha \mathbb{I} + \beta \Phi^T \Phi \quad (5b)$$

where, Φ is the design matrix, \mathbf{m}_{post} , \mathbf{S}_{post} are the mean and covariance of the posterior distribution for ω

Bayesian Inference for Linear Regression

- MLE for linear regression: $\omega_{MLE} = \Phi^\dagger$
- MLE gives point estimates for ω , but a predictive distribution in *data space*
- Remember:
 $p(y_i | x_i, \omega, \beta) = \mathcal{N}(y_i | \omega^T \phi(x_i), \beta^{-1})$
- Ex: Fitting degree=1 polynomial; $\beta = 1.0$
- Predictive distribution is:
 $\mathcal{N}(y | (\omega_0 + \omega_1 x), 1.0)$
- What do you notice about the MLE distribution?

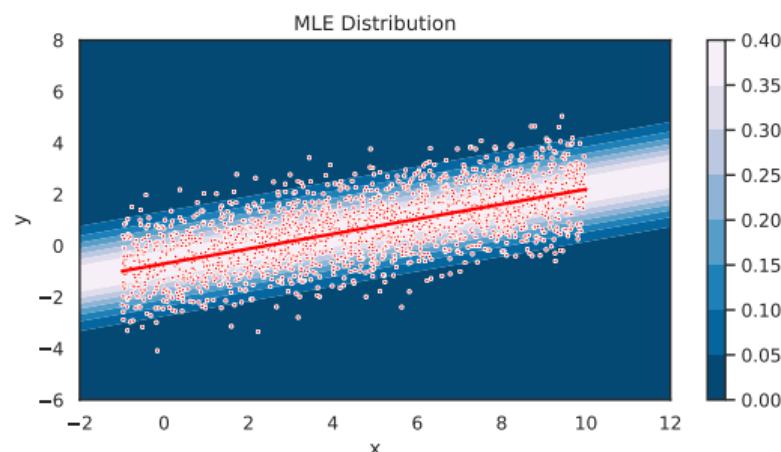


Figure: Polynomial degree=1 fit to data; contour plot depicts predictive distribution

Bayesian Inference for Linear Regression

- Bayesian inference for linear regression: posterior distribution of weights $p(\omega | \mathbf{y}, \mathbf{x})$
- I.e. we have a predictive distribution for each **possible ω** from $p(\omega | \mathbf{y}, \mathbf{x})$
- Consider $N = 10$ i.i.d observations, $\beta = 1.0$, prior $p(\omega) \sim \mathcal{N}(\mathbf{0}, \alpha^{-1} \mathbb{I})$
- We infer $p(\omega | \mathbf{y}, \mathbf{x}) = \mathcal{N}(\omega | \mathbf{m}_{post}, \mathbf{S}_{post})$: What do you **notice**?

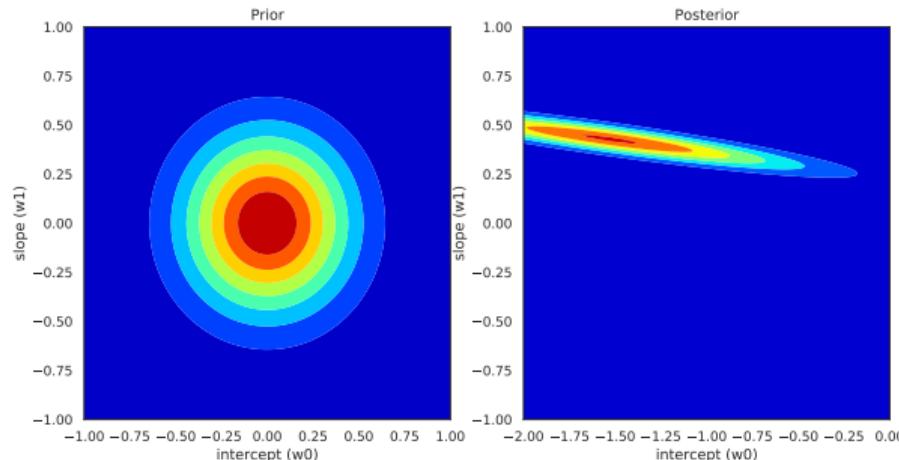


Figure: Left: Prior distribution, $\alpha = 10.0$; Right: Posterior distribution after $N = 10$ observations

Bayesian Inference for Linear Regression: Examples

- Let's see what the impact of varying the number of observations is on the posterior
- What do you **notice?**
- x-axis: intercept (ω_0);
y-axis: slope (ω_1)

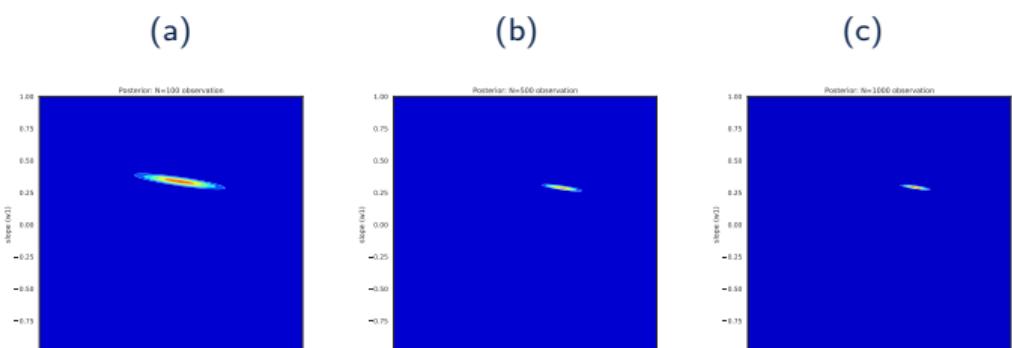
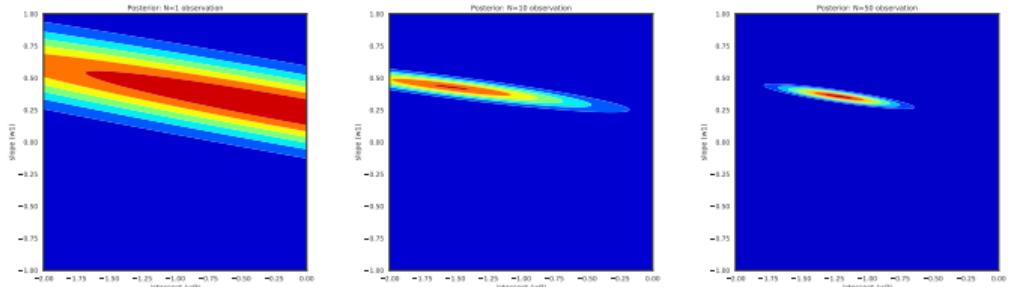


Figure: (a) $N = 1$; (b) $N = 10$; (c) $N = 50$; (d) $N = 100$; (e) $N = 500$; (f) $N = 1000$

Bayesian Inference for Linear Regression: Examples

- *Model uncertainty reduces with increase in observations*
- What about the predictive distribution?
- Let's sample 10 instances of ω from $\mathcal{N}(\mathbf{m}_{post}, \mathbf{S}_{post})$
- What do you notice?

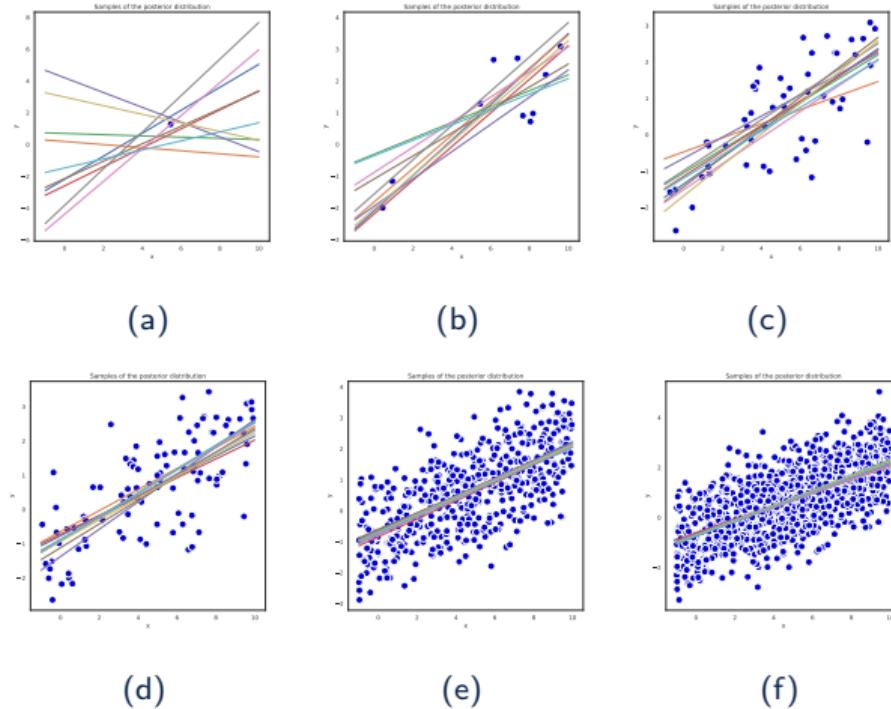


Figure: (a) $N = 1$; (b) $N = 10$; (c) $N = 50$; (d) $N = 100$; (e) $N = 500$; (f) $N = 1000$

Bayesian Inference for Linear Regression: Predictions

- To use the posterior distribution of ω to predict for new x we need to evaluate:

$$p(y_{new} | \mathbf{y}, \alpha, \beta) = \int p(y_{new} | \omega, \mathbf{x}_{new}, \beta) p(\omega | \mathbf{y}, \alpha, \beta) d\omega \quad (6)$$

- I.e. the *predictive distribution* - weighted expectation (or average) over posterior
- Product of two Gaussians (in this case), hence Gaussian distribution itself:

$$p(y_{new} | \mathbf{y}, \alpha, \beta) = \mathcal{N}(y_{new} | \mathbf{m}_{post}^T \phi(\mathbf{x}_{new}), \sigma_{new}^2) \quad (7a)$$

$$\sigma_{new}^2 = \frac{1}{\beta} + \phi(\mathbf{x}_{new})^T \mathbf{S}_{post} \phi(\mathbf{x}_{new}) \quad (7b)$$

- σ_{new}^2 expresses influence of **noise on the data (β)** and **uncertainty in the parameters**

Bayesian Inference for Linear Regression: Predictions

- Simple estimate - sample from the posterior and average the resulting predictions
- For many problems (especially of higher dimensions) this is **overkill**
- Consider what is important for your regression problem
- For approximate predictions on new data: MAP is a good compromise
 - For this Gaussian case mode of the posterior is the mean and hence MAP is given by \mathbf{m}_{post}
- With limited data, high penalty for incorrect prediction - **predictive distribution** is valuable

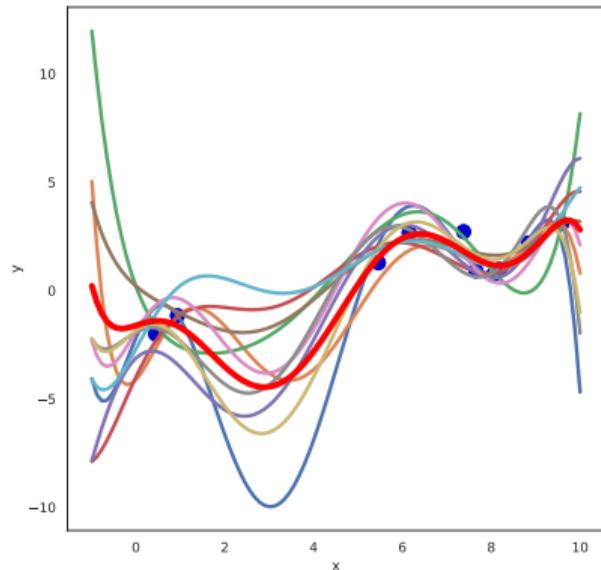


Figure: Uncertainty

Regularisation for Linear Regression

- Lets recap: given N observations and prior $p(\omega) = \mathcal{N}(\omega | \mathbf{0}, \alpha^{-1}\mathbb{I})$:

$$\ln p(\mathbf{w} | \mathbf{y}, \mathbf{x}) = -\frac{\beta}{2} \sum_{i=1}^N \left\{ y_i - \mathbf{w}^T \phi(x_i) \right\}^2 - \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + \text{const} \quad (8)$$

- Notice the **quadratic term** of just the weights vector $\frac{\alpha}{2} \mathbf{w}^T \mathbf{w}$
- Lets say we try to **maximise this posterior distribution** with respect to ω
 - Notice - in the Gaussian case this is MAP
 - What **effect** do you think the second term has?

Regularisation for Linear Regression

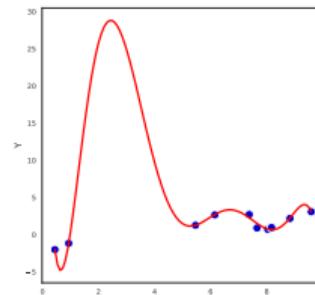
- Maximising the log likelihood is equivalent to minimising the negative log likelihood:

$$L = \frac{1}{2} \sum_{i=1}^N \left\{ y_i - \mathbf{w}^T \phi(x_i) \right\}^2 + \frac{\alpha}{2\beta} \mathbf{w}^T \mathbf{w} \quad (9)$$

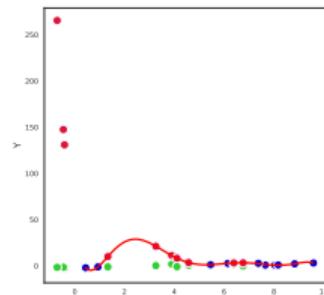
- Minimisation of this cost function in regression is also known as *ridge regression*
- Second term is called the *regularisation term* and $\frac{\alpha}{2\beta}$ is the *regularisation coefficient*
- Quadratic regulariser - **shrinks the weights** - reducing overfitting
- Other types of regularisers exist each with its own properties for constraining the weights
- Procedure for fitting regression models by minimising: $R(\omega) + \lambda R_g(\omega)$ called regularised least squares

Regularisation for Linear Regression: Examples

- Consider $N = 10$ observations: let's compare MLE (top row) vs MAP (bottom row) for polynomial degree=8

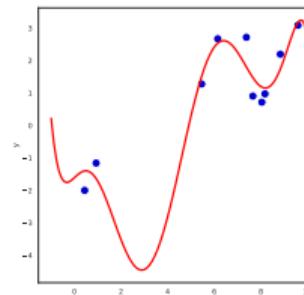


(a)

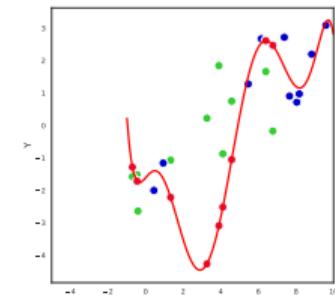


(b)

- Training data, test, predictions
- What do you notice?
- And why does this happen?



(c)



(d)

Bayesian Linear Regression: Some additional points

- We have looked at the special case of known variance for noise in the data & $p(\mu) \sim \mathcal{N}(\mathbf{0}, \alpha \mathbb{I})$
- Both these assumptions need not be true/applicable
- Reader and labs cover estimate for $\mathbf{m}_{post}, \mathbf{S}_{post}$ for $p(\mu) \sim \mathcal{N}(\mathbf{m}_0, \mathbf{S}_0)$
- We looked at linear regression using *batch learning* - i.e. use all of the data to construct Φ
- *Sequential learning* also possible - experiment in the lab

Summary of Unit 2

- Linear regression models can be fit to data using **OLS**
- Under **additive Gaussian noise** assumption MLE is equivalent to minimising sum of squared residuals
- MLE methods are **prone to overfitting** - need cross-validation experiments to choose model
- **Bayesian inference** naturally reduces overfitting
- **MAP estimate** for Bayesian linear regression results in quadratic regulariser of weights (Gaussian case)
- **Quadratic regulariser (ridge regression)** - shrinks weights, reduces overfitting (form of regularised least squares)

COMP5611M: Machine Learning

Linear Regression Summary

Nishant Ravikumar
Marc de Kamps

School of Computing

February 21, 2022



Lecture Objectives

Objectives for this lecture:

- Recap MAP estimation in Bayesian linear regression
- Regularisation in linear regression
- Summary of Unit 2

Learning Outcomes

At the end of this lecture, you should be able to

- Describe common types of regularisation in linear regression
- Describe pros and cons of different approaches to fitting linear regression models

Bayesian Linear Regression: MAP

- Given N observations $\{x_i, y_i\}_{i=1..N}$ and assuming:
 - Linear functional relationship: $\mathbf{y} = \hat{\mathbf{y}}(\mathbf{x}, \boldsymbol{\omega}) + \boldsymbol{\epsilon}$
 - Additive Gaussian noise with known precision (β)

$$p(\mathbf{y} | \mathbf{x}, \boldsymbol{\omega}, \beta) = \prod_{i=1}^N \mathcal{N}(y_i | \boldsymbol{\omega}^T \phi(x_i), \beta^{-1}) \quad (1)$$

- Prior $p(\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega} | \mathbf{0}, \alpha^{-1} \mathbb{I})$
- posterior \propto likelihood \times prior*
- For linear regression posterior over weights given by:

$$p(\boldsymbol{\omega} | \mathbf{y}, \mathbf{x}) \propto \exp \left\{ \frac{-\beta}{2} \sum_{i=1}^N \{y_i - \boldsymbol{\omega}^T \phi(x_i)\}^2 - \frac{\alpha}{2} \boldsymbol{\omega}^T \boldsymbol{\omega} \right\} \quad (2)$$

Bayesian Linear Regression: MAP

$$\ln p(\omega | \mathbf{y}, \mathbf{x}) = -\frac{\beta}{2} \sum_{i=1}^N \left\{ y_i - \omega^T \phi(x_i) \right\}^2 - \frac{\alpha}{2} \omega^T \omega + \text{const} \quad (3)$$

- Lets say we want to **maximise this posterior distribution** with respect to ω
- For the **Gaussian case** ω_{MAP} is given by the mean of the posterior distribution $p(\omega | \mathbf{y}, \mathbf{x})$
- Notice the **quadratic term** of just the weights vector $\frac{\alpha}{2} \omega^T \omega$
- What **effect** do you think the second term has? - Acts as a **regulariser!**
- Taking $-\ln(p(\omega, \mathbf{y}, \mathbf{x}))$ results in an error function **similar to $R(\omega)$** we used in OLS

Regularisation for Linear Regression

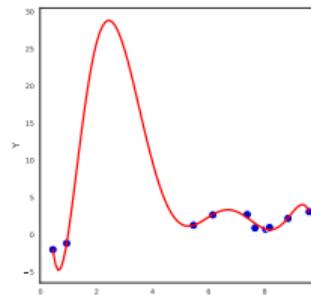
- Instead of inferring $p(\omega | \mathbf{y}, \mathbf{x})$ to estimate ω_{MAP} , we can also directly minimise L w.r.t ω :

$$L(\omega) = \frac{1}{2} \sum_{i=1}^N \left\{ y_i - \omega^T \phi(x_i) \right\}^2 + \frac{\alpha}{2\beta} \omega^T \omega \quad (4)$$

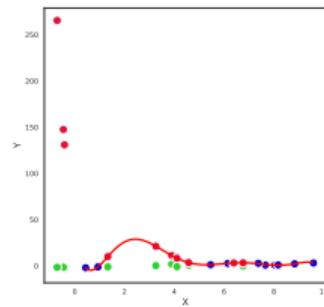
- Minimisation of this cost function in regression is also known as *ridge regression*
- Second term is called the *regularisation term* and $\frac{\alpha}{\beta}$ is the *regularisation coefficient*
- Quadratic regulariser - **shrinks the weights** - reducing overfitting
- Other types of regularisers exist each with its own properties for constraining the weights
- Procedure for fitting regression models by minimising: $R(\omega) + \lambda R_g(\omega)$ called regularised least squares

Regularisation for Linear Regression: Examples

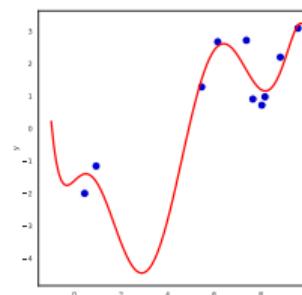
- Consider $N = 10$ observations: let's compare MLE (top row) vs MAP (bottom row) for polynomial degree=8
- $\beta = 1.0, \alpha = 10.0$
- Training data, test, predictions
- What do you notice?
- And why does this happen?



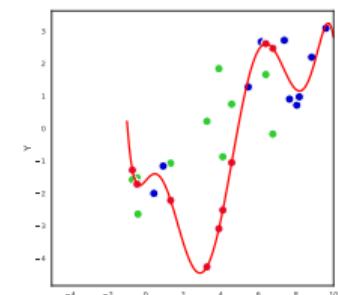
(a)



(b)



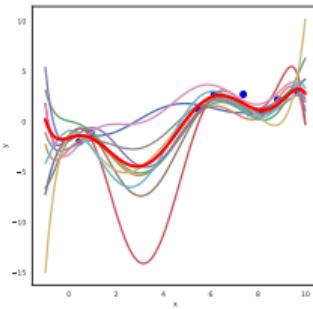
(c)



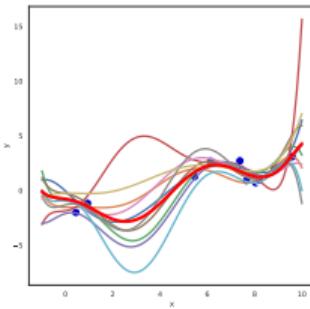
(d)

Regularisation for Linear Regression: Examples

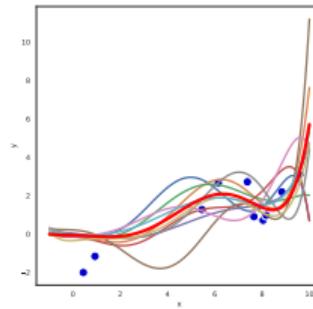
- For the Bayesian Gaussian case, let's see the effect of varying α
- Remember $\lambda = \frac{\alpha}{\beta}$
- Can you guess how α is changing from left to right?



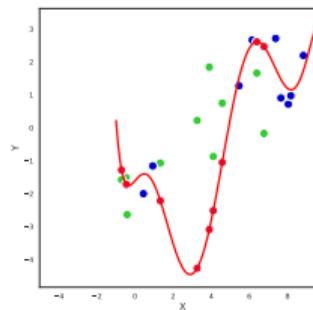
(a)



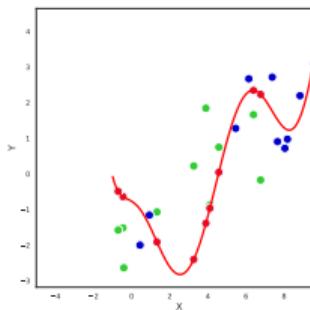
(b)



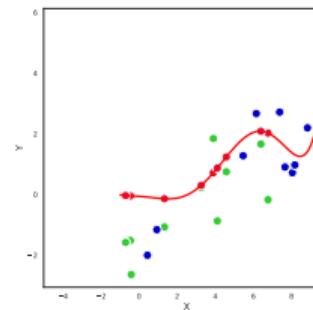
(c)



(d)



(e)



(f)

Regularisation for Linear Regression: Regularised least squares

- A general form for the regularised cost function is expressed as:

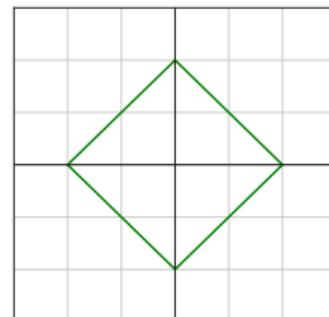
$$L(\omega) = R(\omega) + \lambda R(\omega) \quad (5a)$$

$$L(\omega) = \frac{1}{2} \sum_{i=1}^N \left\{ y_i - \omega^T \phi(x_i) \right\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |\omega_j|^q \quad (5b)$$

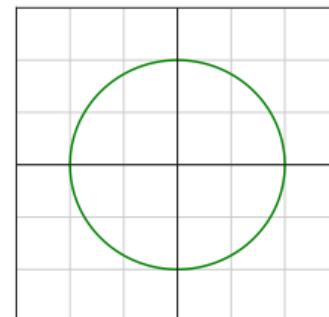
- Here $q = 2$ corresponds to the quadratic regulariser - *ridge regression*
- Generalised prior over ω that is no longer conjugate to the likelihood leads to Eq. 5b
- Estimating ω_{MAP} corresponds to **minimising the regularised cost/error function**
- λ represents the *trade-off* between the how well you want to **fit to your data** and how much you want to **constrain your model**

Regularisation for Linear Regression: Types of regularisers

- Different values for q correspond to different norms over ω - i.e. different types of constraints
- Contours shown illustrate these regularisation terms/constraints for $q = 1$ (left) & $q = 2$ (right)
- We know what $q = 2$ corresponds **ridge regression**
- What about $q = 1$? What effect does this have?



(a)



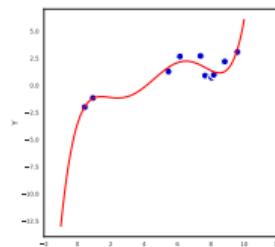
(b)

Regularisation for Linear Regression: Lasso vs Ridge

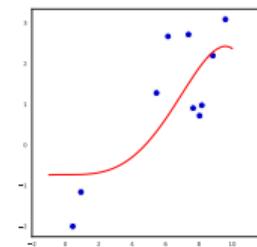
- Using $q = 1$ results in an L1-norm over ω and is known as *lasso regression*
- Key differences between *lasso* and *ridge* regression are:
 - Lasso: L1-norm, Ridge: L2-norm
 - Lasso: encourages *sparse* solutions, Ridge: *shrinks* weights towards zero
 - Lasso: *No closed-form* solution for ω , Ridge: *admits closed-form* solution for ω
- Estimating weights for lasso regression requires *numerical techniques* - e.g quadratic programming or convex optimisation

Regularisation for Linear Regression: Lasso vs Ridge

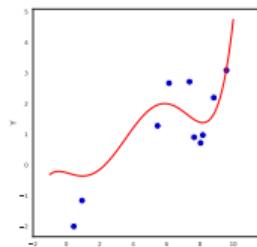
- Lets try fitting fitting degree=5 polynomial to some toy data
- Reference unregularised solution visible in (a) - **OLS**
- Examples of **lasso (b)** & **ridge (c)** regression with $\lambda = 10.0$



(a)



(b)

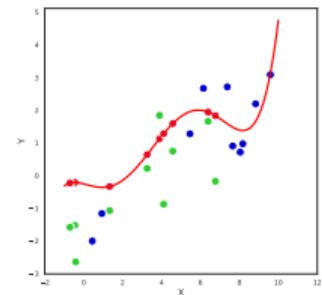
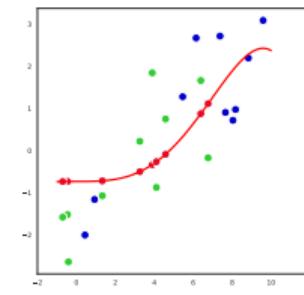
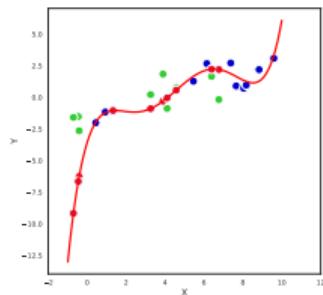
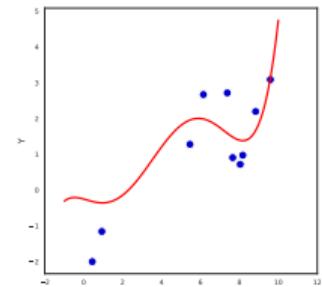
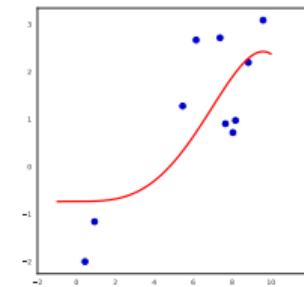
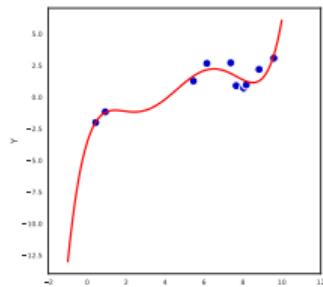


(c)

- Interesting thing here are the resulting weights for each case:
- OLS: $[4.979, -3.44, 1.01, -0.12, 0.005]$, intercept: -3.5775224
- Lasso: $[0.00, 0.00, 5.22e^{-03}, 7.80e^{-04}, -9.91e^{-05}]$, intercept: -0.73
- Ridge: $[-0.15, -0.06, 0.12, -0.02, 0.001]$, intercept: -0.25
- What do you **notice**?

Regularisation for Linear Regression: Lasso vs Ridge

- Now lets predict on some new data using the trained regression models (d-f)
- What do you notice?



(d)

(e)

(f)

Regularisation for Linear Regression: Bias-Variance

- Choosing a regulariser results in a type of *bias* - i.e. determines how weights should be constrained in the absence of data
- Not to be confused with data-related bias - this is specific to the model
- Regularisation allows *control over model complexity* - preventing severe overfitting
- Determining *optimal model complexity* changes from determining *number of basis functions (OLS)* to *suitable value for λ*
- λ controls the *bias-variance* trade-off of regularised linear regression models
- Higher the value for λ - *higher the bias and lower the variance*, and vice-versa
- Refer to pg. 147 in C. Bishop's book (Pattern Recognition and Machine Learning) for more details

Summary of Unit 2

- Linear regression models can be fit to data using **OLS**
- Under **additive Gaussian noise** assumption MLE is equivalent to minimising sum of squared residuals
- MLE methods are **prone to overfitting** - need cross-validation experiments to choose model
- **Bayesian inference** naturally reduces overfitting
- **MAP estimate** for Bayesian linear regression results in quadratic regulariser of weights (Gaussian case)
- **Quadratic regulariser (ridge regression)** - shrinks weights, reduces overfitting (form of regularised least squares)

Summary of Unit 2

- Different types of regularisers can be used to constrain the estimation of regression weights (e.g. lasso vs ridge)
- Regularisation helps control model complexity and reduce overfitting for complex models with limited data
- Regularisation coefficient controls bias-variance trade-off: i.e. the rigidity vs flexibility of a model

COMP5611M: Machine Learning

Introduction to the Neural Networks

Nishant Ravikumar
Marc de Kamps

School of Computing

February 28, 2022

Learning Objectives

- A lightning review of real neurons
- Motivation for artificial neurons
- Perceptron: a linear discriminant
- Linear separability, decision boundary
- The perceptron algorithm

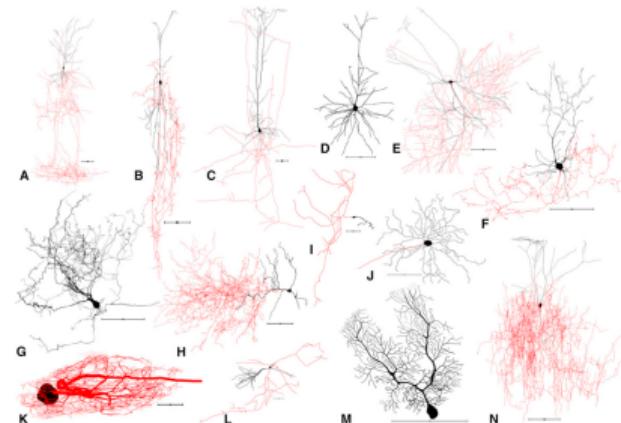
Learning Outcomes

At the end of this lecture you should be able to:

- Formulate the concept of an artificial neuron
- Graphically determine the decision boundary in a 2D classification problem
- Implement and apply the perceptron algorithm to spaces of arbitrary dimension

What is a Neural Network?

- It is good to distinguish between neural networks and **Artificial Neural Networks**
- There are some common ideas:
 - Integration of input
 - Non linear processing of integration
- There are substantial differences, biology important:
 - Neural **morphology** plays a role in computation
 - Subtle, non-hierarchical architecture
 - Dynamics, e.g. important for winner-take-all
 - Chaos
 - Extremely low energy consumption



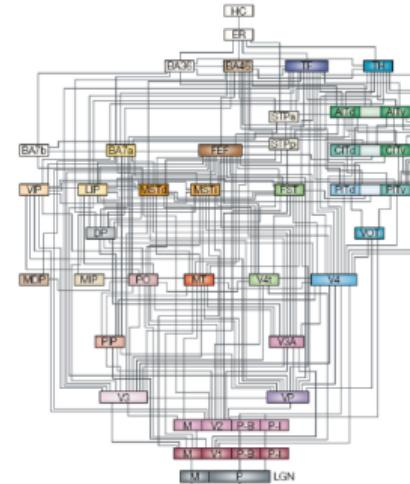
Parekh and Ascoli, *Neuron*, 2013

What is a Neural Network?

- It is good to distinguish between neural networks and **Artificial Neural Networks**
- There are some common ideas:
 - Integration of input
 - Non linear processing of integration
- There are substantial differences, biology important:
 - Neural **morphology** plays a role in computation
 - Subtle, non-hierarchical architecture
 - Dynamics, e.g. important for winner-take-all
 - Chaos
 - Extremely low energy consumption



Felleman and van Essen, 1991, Cerebral Cortex

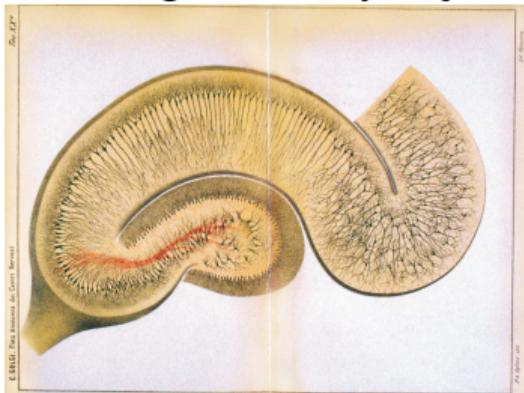


Neurons are Hard to See!

- The nature of the brain was under debate until late 19-th century
- The universality of the **cell hypothesis** was accepted by then
- But the nature of the brain was still a mystery: **it is transparent under the microscope**
- Golgi invented staining technique (bath of silver nitrate) that made neurons visible
- Ironically, Golgi staining was used to great effect by Ramon-y-Cajal who formulated the *neuron doctrine* ...
- ... to which Golgi was opposed



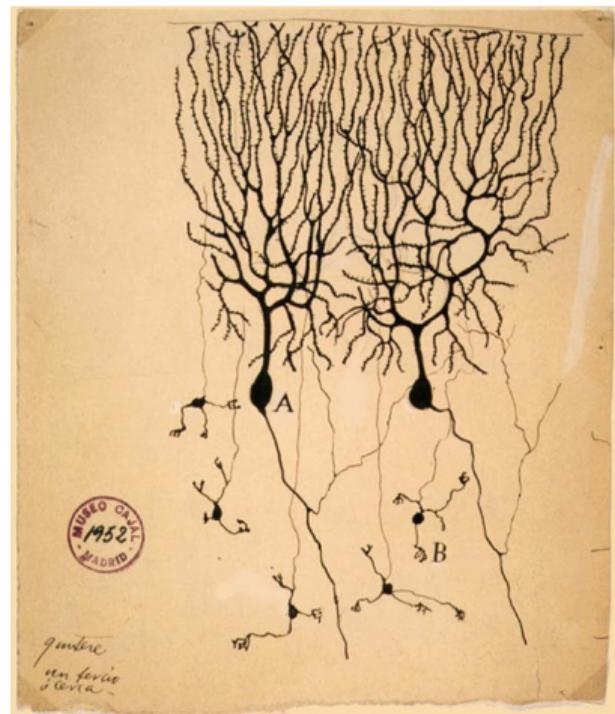
Santiago Ramon-y-Cajal



The Neuron Doctrine

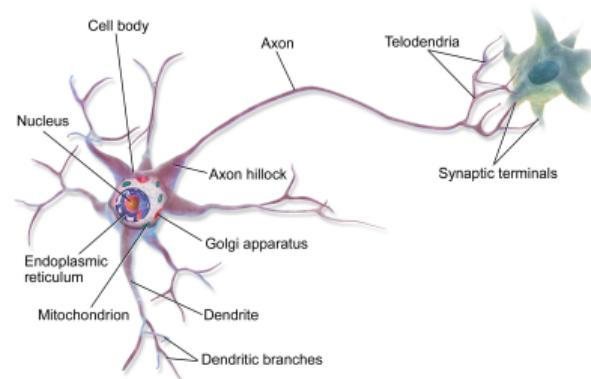
Cajal's and others' observations led to the *neural doctrine*, which among others states that https://en.wikipedia.org/wiki/Neuron_doctrine:

- The brain is made up of individual units that contain specialized features such as dendrites, a cell body, and an axon.
- These individual units are cells as understood from other tissues in the body.
- Although the axon can conduct in both directions, in tissue there is a preferred direction for transmission from cell to cell.



A Simplified Neural Model

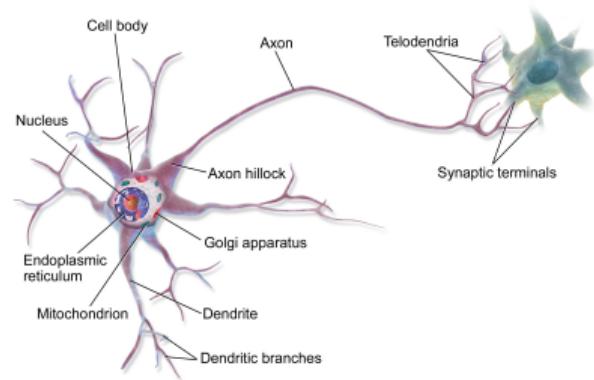
- Neurons are normal cells
- But maintain a potential difference (-70 mV) between inside and outside
- Neurotransmitters induce electrochemical activity at synapses
- Mainly dendritic tree, leads to local potential differences
- Often leak away, sometimes not
- If the membrane is *depolarised* sufficiently, Na^+ ions flow in, depolarising the membrane even further



Source: Wikimedia

A Simplified Neural Model

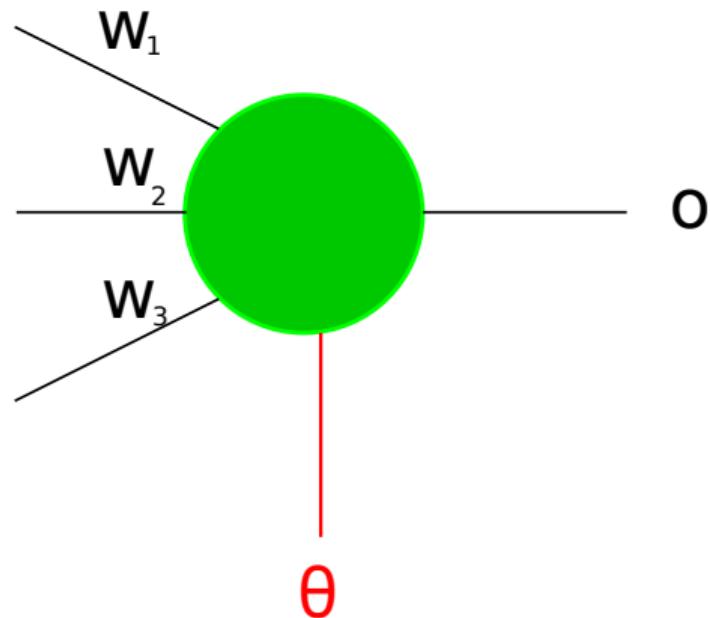
- This phenomenon affects cell patches in the neighbourhood
- The effect is like a *fuse*, rather than a cable signal
- The axon is specialised to mediate this *spike over considerable distances*
- The strong depolarisation opens other ion channels restoring the membrane potential, closing the Na^+ channels in the process
- Refractory period (the neuron is less responsive)



Source: Wikimedia

The Perceptron

- The perceptron is a highly stylised version of a real neuron
- It has a number of inputs (N), which can be set a numerical value $x_i, i = 1, \dots, N$
- Each input has a weight, also a numerical value w_i
- It has threshold θ



The Perceptron

$$h = w_1x_1 + w_2x_2 + w_3x_3 - \theta$$

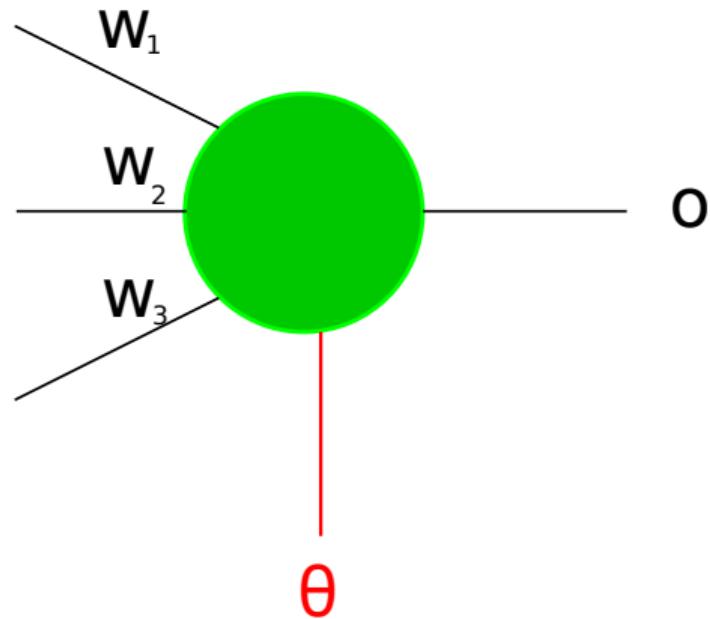
h is called the *local field* x_i are the numerical values of the input, w_i weights and θ is called threshold (or bias)

$$o = \mathcal{H}(h),$$

where:

$$\mathcal{H}(x) = \begin{cases} x < 0 : & 0 \\ x \geq 0 : & 1 \end{cases}$$

Also called the *step function*

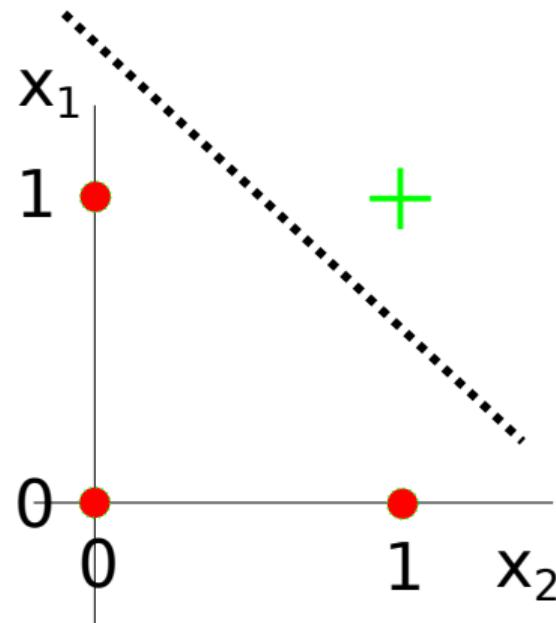


A 2D Example

- **AND** gate is a simplified example
- Demonstrates the original thinking behind neural networks (McCulloch & Pitts, 1943)

x_1	x_2	x_3	o
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

Can you solve this classification problem with a perceptron?



The Need for a Training Algorithm

- The example is so simple that you may wonder why you need an algorithm at all!
- But consider having 10 variables
- This entails finding a hyperplane in a 10 dimensional space!
- We then need 10 weights and a threshold!
- Multiplying weights (and threshold) by a positive factor does not change the decisions
- Multiplying by a negative factor *reverses* the decision. *Why?*

Intuition

If classification is wrong and desired output is one, my weights may be too large. If classification is wrong and desired output is 0, my weights may be too small. Why not add (subtract) the input pattern to the weights?

Absorbing the Threshold

Simple Observation

You can consider the threshold as an extra input which is always clamped to one. If the weight of this input is minus the threshold, you will have a perceptron taking the same decision.

This follows from:

$$w_1x_1 + w_2x_2 + w_3x_3 = w_1x_1 + w_2x_2 + w_3 = w_1x_1 + w_2x_2 - \theta$$

In practice I need to add an extra column of ones to my dataset:

x_1	x_2	x_3	o
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

I can present the *perceptron algorithm* without treating the threshold separately. Which is good as the threshold needs to adapted just as the weights. (Why?)

The Perceptron Algorithm

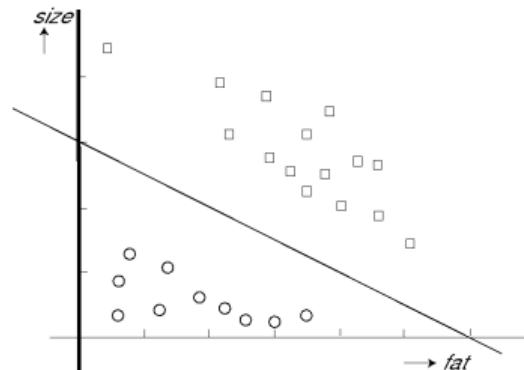
The algorithm is as follows. Start with a perceptron with $N + 1$ inputs, one of which is clamped to value $+1$, and $N + 1$ weights so that a threshold is unnecessary.

1. **Start:** Choose a random set of weights: \mathbf{w}_0 . For later reference, denote the current set of weights by \mathbf{w}_i , so at the start $\mathbf{w}_i = \mathbf{w}_0$.
2. **Continue:** Pick a random data point $(\mathbf{x}_j, d_j), j = 1, \dots, D$, where \mathbf{x}_j is a point for which classification $d_j \in \{0, 1\}$ is given.
3. Evaluate $\mathbf{w}_i^T \mathbf{x}_j$. If
$$\begin{cases} \mathbf{w}^T \mathbf{x}_j < 0 & \text{AND } d_j = 0 : \text{goto Continue} \\ \mathbf{w}^T \mathbf{x}_j < 0 & \text{AND } d_j = 1 : \text{goto Add} \\ \mathbf{w}^T \mathbf{x}_j \geq 0 & \text{AND } d_j = 1 : \text{goto Continue} \\ \mathbf{w}^T \mathbf{x}_j \geq 0 & \text{AND } d_j = 0 : \text{goto Subtract} \end{cases}$$
4. **Add:** $\mathbf{w}_{i+1} = \mathbf{w}_i + \mathbf{x}$; Goto **Continue**
5. **Subtract:** $\mathbf{w}_{i+1} = \mathbf{w}_i - \mathbf{x}$; Goto **Continue**

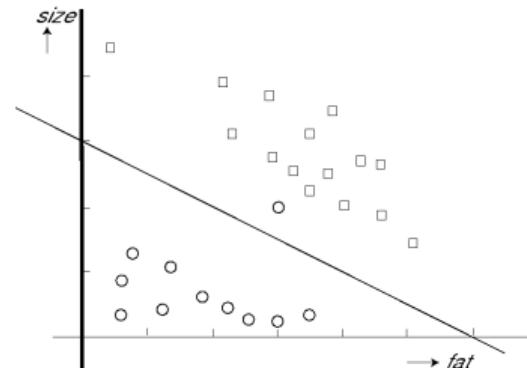
An Example

Crisps: portion size and fat content

- Box: Should be classified as a health hazard
- Circle: OK for consumption



This perceptron is fine. Give weight and threshold values.



This perceptron misclassifies the point (3,3).
Apply the algorithm

The Perceptron Theorem

- In plain English: if data is linearly separable, the perceptron algorithm will find a set of weights that separates the two classes
- Mathematically, it is formulated like this:
- Assume a single class of points exists that is on one side of a plane through the origin
- For every $\vec{x} \in \mathcal{C}$, assume there is a \vec{w}^* such that $\vec{w}^* \cdot \vec{x} \geq \delta > 0$, for all patterns \vec{x} .
- Wait! we had two classes?

Let \mathcal{C} be a set of patterns.

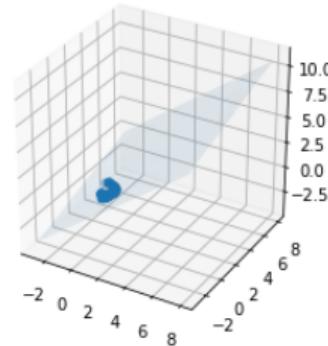
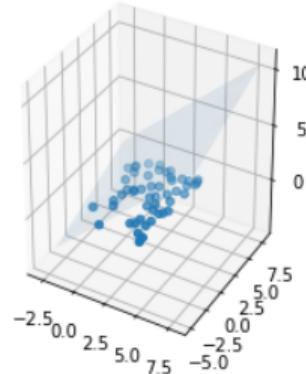
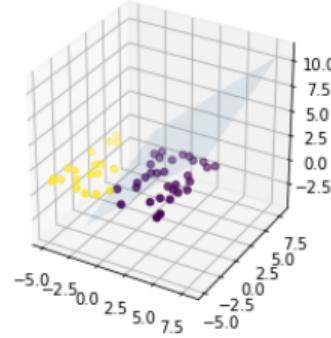
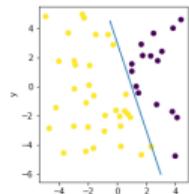
Perceptron Algorithm

If for every $\vec{x} \in \mathcal{C}$, there is a \vec{w}^* such that $\vec{w}^* \cdot \vec{x} \geq \delta > 0$, for all patterns \vec{x} , then the perceptron algorithm will go through a finite number of steps **Add** and **Subtract**

- Implication: algorithm has settled on a set of weights that classify correctly
- Not necessarily \vec{w}^*

Geometrical Ideas of Proof

- Absorb threshold
- Assume there are two classes of patterns, one: \mathcal{C}^+ that must be classified as 1, one \mathcal{C}^- that needs to be classified as 0
- If $\vec{x} \in \mathcal{C}^-$, replace by $-\vec{x} \in \mathcal{C}^+$
- For good measure: normalise $\vec{x} \rightarrow \frac{\vec{x}}{|\vec{x}|}$
- Now the conditions for the Perceptron Theorem satisfied



Proof of the Perceptron Theorem

Follow the following quantity as we step through the algorithm:

$$\frac{\vec{w}^* \cdot \vec{w}_n}{|\vec{w}^*| |\vec{w}_n|} = \cos \theta$$

- Can we do this? We don't even know \vec{w}^*
- It turns out we can!
- Observe that this quantity is a **cosine**
- So must be smaller than one.
- If every update of the algorithm increases $\cos \theta$ by a finite amount it must stop
- Even if we can't calculate $\cos \theta$ itself

Proof of the Perceptron Theorem

What Happens to the Numerator?

- Imagine we already have been through n steps of the algorithm
- The algorithm goes through **Add**, so the weights change

$$\vec{w}^* \cdot \vec{w}_{n+1} = \vec{w}^* \cdot (\vec{w}_n + \vec{x}) < \vec{w}^* \cdot \vec{x} + \delta$$

- so the numerator increases by at least δ

Proof of the Perceptron Theorem

What Happened to the Denominator?

- Remember we normalised all input patterns $|\vec{x}| = 1$
- Again, we have already stepped n times through **Add**

$$|\vec{w}^*| \parallel \vec{w}_{n+1} | = |\vec{w}^*| \parallel \vec{w}_n + \vec{x} |$$

- \vec{w}^* never changes, so we only have to consider:

$$(\vec{w}_n + \vec{x})^2 = (\vec{w}_n^2 + 2\vec{w}_n \cdot \vec{x} + 1)^2 < \vec{w}_n^2 + 1$$

- The latter step: $\vec{w}_n \cdot \vec{x} < 0$, otherwise the algorithm would not have done **Add**

The Perceptron Algorithm

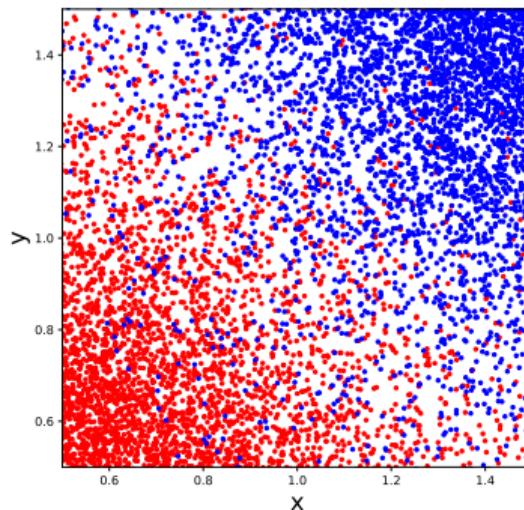
The proof of the algorithm relies on the fact that each **Add** step increases the cosine of angle between \vec{w}^* and \vec{w}_n by at least δ ; a *finite* amount. This means the number of steps must remain finite.

- The perceptron algorithm is an iterative procedure for determining the weights (parameters) of a neural network
- We say we *train* the neural network

Limitations of the Perceptron

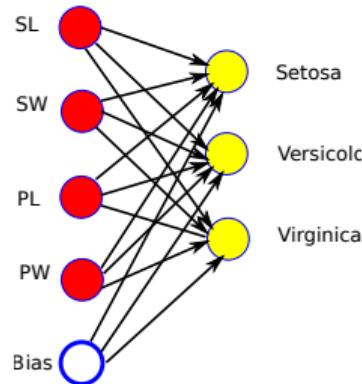
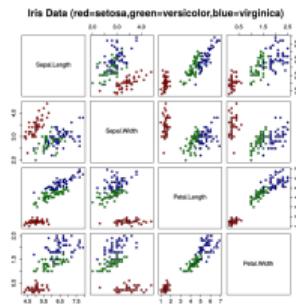
There are substantial problems with the perceptron algorithm

- There is no real stopping criterion
- One can be improvised
- E.g. what to do with an almost linearly separable dataset?
- It seems a linear decision boundary would still be reasonable



Limitations of the Perceptron

There are substantial problems with the perceptron algorithm



- More importantly: some datasets are not linearly separable
- The perceptron theorem in this form can't handle such data

- Only two nodes in this network can be successfully trained
- Why?

Recap

- Historically, the perceptron algorithm has had a large influence on the study of neural networks
- It is not the only algorithm to train so-called linear discriminants: Fisher's discriminant (Bishop, 2006) is probably more widely used
- The perceptron naturally leads to *logistic regression* (next lecture).
- The quantity δ plays an important role in *support vector machines*
- We will not cover them, but you may run across them later in your career.

COMP5611M: Machine Learning

Introduction to Logistic Regression

Nishant Ravikumar
Marc de Kamps

School of Computing

March 4, 2022

Learning Objectives

- Logistic Regression as a 'graded perceptron'
- Loss functions for logistic regression
- Steepest gradient descent and beyond
- A generative model for logistic regression
- The need for multi-layered perceptrons

Learning Outcomes

At the end of this lecture you should be able to:

- Derive learning rules for logistic regression
- Explain the relationship between logistic regression and the perceptron
- Explain the difference between generative and discriminative models in the context of logistic regression
- Implement logistic regression in the *Pytorch* frame work (associated lab)

Motivation

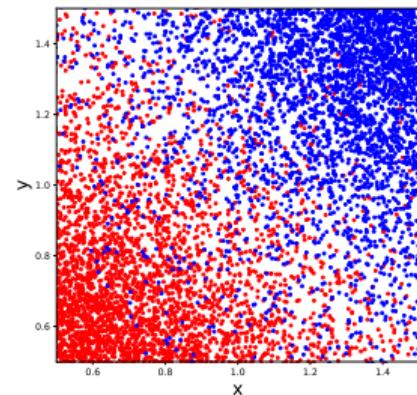
- The Perceptron Algorithm fails on this dataset
- We would prefer a loss function
- There is one, but it is hard to use:

$$E = \frac{1}{2}(o_i - d_i)^2,$$

where

$$o_i = \mathcal{H}\left(\sum_k w_k x_{ik}\right)$$

and i labels the points in the dataset



Essentially counts the number of misclassifications.

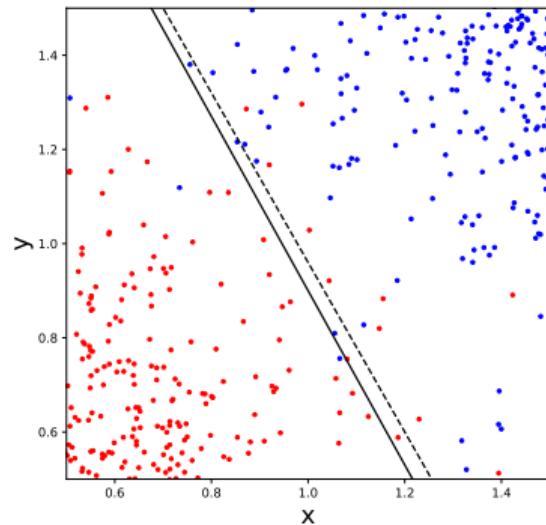
A Graded Perceptron

$$E = \frac{1}{2}(o_i - d_i)^2,$$

where

$$o_i = \mathcal{H}\left(\sum_k w_k x_{ik}\right)$$

- As an optimisation problem this is highly non-linear!
- Visualise slowly moving a decision line around the plane
- Often nothing happens but if the line crosses a point E jumps!
- Ordinarily: differentiate with respect to the parameters and set the derivative to 0
- There's nothing to differentiate here! Explain why in two ways



The Perceptron Algorithm optimised this loss function only indirectly!

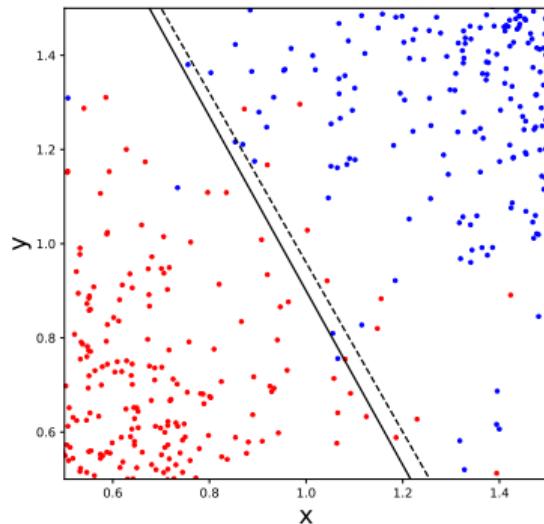
A Minimum for the Loss Function

$$E = \frac{1}{2}(o_i - d_i)^2,$$

where

$$o_i = \mathcal{H}\left(\sum_k w_k x_{ik}\right)$$

- This is quite frustrating!
- Especially since for this dataset the error function has a minimum?
- Why?
- Is this a global minimum?



A Graded Perceptron

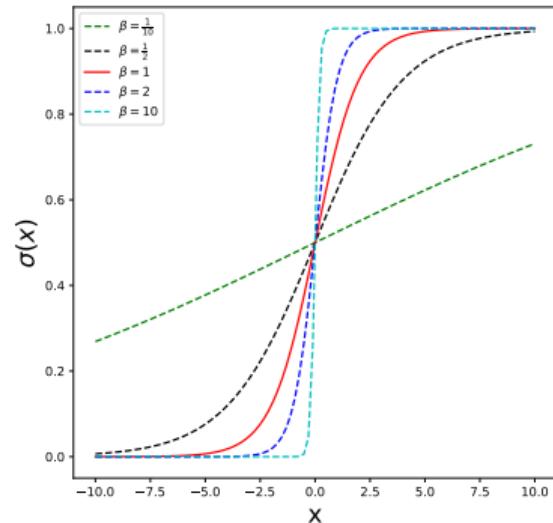
- Imagine that we adapt the perceptron formula a little bit:

$$o = \sigma\left(\sum_i w_i x_i\right),$$

with

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

- We usually fix $\beta = 1$
- But can pick larger values to resemble step function
- Now the loss function is differentiable w.r.t. weights



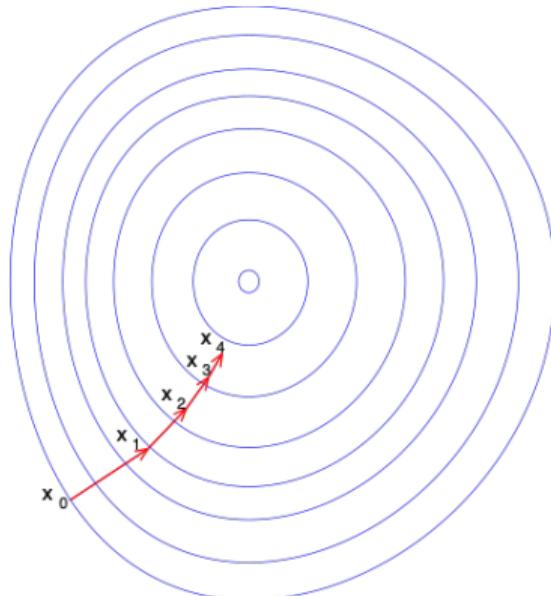
Minimising the Loss Function

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

- $\sigma(x)$ is called the *logistic function*
- There is indeed a relationship with *logistic regression*
- The condition for the minimum gives a non linear equation:

$$\sum(o_i - d_i)\sigma(\sum w_i j x_j)w_i = 0$$

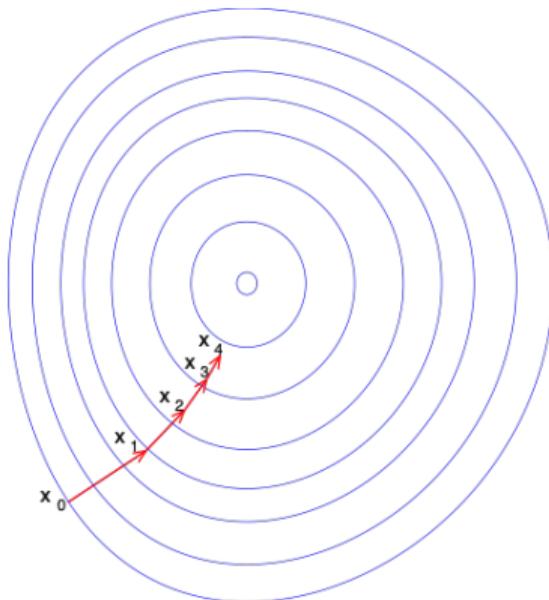
We need an iterative solution scheme



Steepest Gradient Descent

- Steepest Gradient Descent can be used to minimise complex non linear functions
- It is always possible to compute the gradient with respect to minimisation parameters
- Start with a 'random' choice, then step a little bit against the direction of the gradient at that point
- Remember self-assessment exercises?

$$\mathbf{w} \rightarrow \mathbf{w} - \lambda \frac{\partial \mathcal{E}}{\partial \mathbf{w}}$$



Steepest Gradient Descent

We need to apply the chain rule:

$$\frac{\partial \mathcal{E}}{\partial \mathbf{w}} = \sum_i (o_i - d_i) \frac{\partial}{\partial \mathbf{w}} o_i.$$

Since

$$o_i = \sigma(\mathbf{w}^T \mathbf{x}_i),$$

so that

$$\frac{\partial}{\partial \mathbf{w}} o_i = \sigma'(\mathbf{w}^T \mathbf{x}_i) \frac{\partial \mathbf{w}^T \mathbf{x}_i}{\partial \mathbf{w}}$$

Steepest Gradient Descent

When we write out $\mathbf{w}^T \mathbf{x} = \sum_j w_j x_j$, it is easy to see that

$$\frac{\partial}{\partial w^i} \sum_j w^j x_j = x_i,$$

or in vector notation:

$$\frac{\partial}{\partial \mathbf{w}} \mathbf{w}^T \mathbf{x} = \mathbf{x}$$

Ultimately:

$$\frac{\partial}{\partial \mathbf{w}} o_i = \sigma'(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i$$

so that

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_i (o_i - d_i) \sigma'(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i$$

This formula holds generally, not just for the logistic function

Steepest Gradient Descent

Mean Squared Error Loss Function

- For the logistic function:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- This leads to the so-called *delta rule*

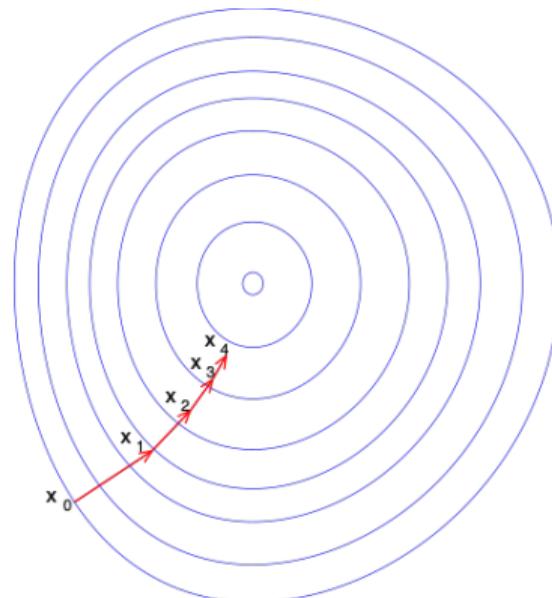
$$\mathbf{w} \rightarrow \mathbf{w} - \lambda \sum_i o_i(1 - o_i)(d_i - o_i)\mathbf{x}_i$$

- Or

$$\mathbf{w} \rightarrow \mathbf{w} - \lambda \sum_i \Delta_i \mathbf{x}_i,$$

with

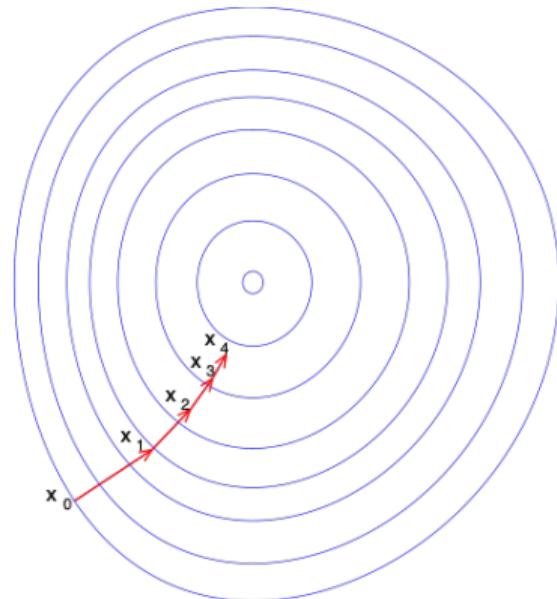
$$\Delta_i \equiv o_i(1 - o_i)(d_i - o_i).$$



MSE Loss

$$\mathbf{w} \rightarrow \mathbf{w} - \lambda \sum_i o_i(1 - o_i)(d_i - o_i)\mathbf{x}_i$$

- Observe that this is the perceptron learning rule in disguise
- The rule is inefficient **Why?**
- In fact for classification MSE loss is not great
- We will redo this for *cross entropy* (which is one of the terms in the KL-divergence)

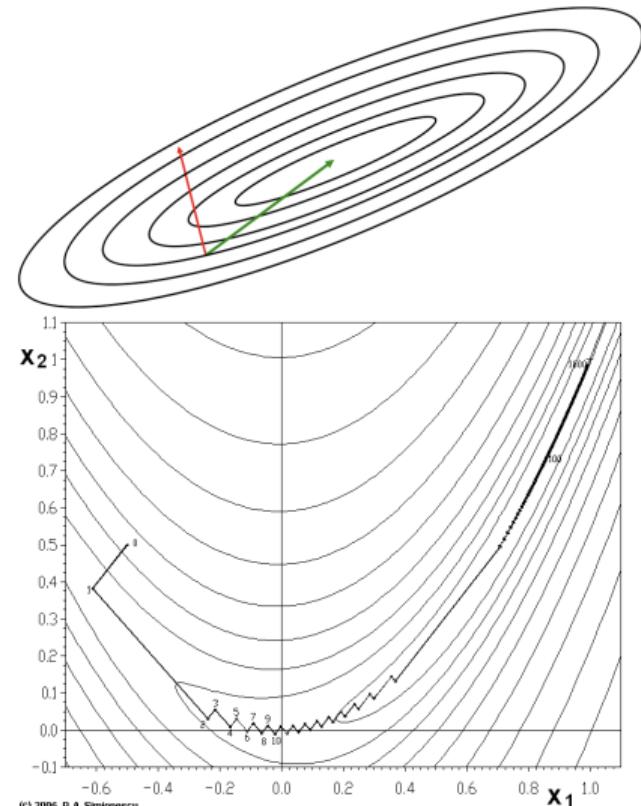


Numerical Optimisation

- Steepest gradient descent often needs to be modified
- Technically the gradient is calculated over all training patterns: *batch* learning
- It works to better to select a small sample at random *minibatch*
- Often momentum is a good idea:

$$\Delta \mathbf{w}_t = -\lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} + \alpha \Delta \mathbf{w}_{t-1}$$

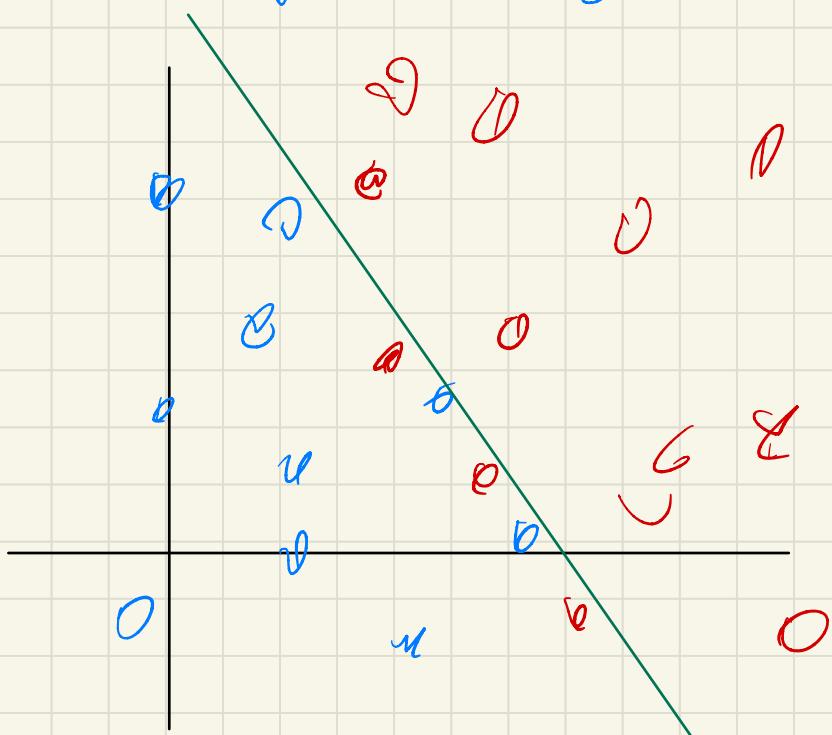
- There is a raft of numerical approaches that *Pytorch* facilitates



Outlook

- Pytorch dramatically simplifies the process of find a gradient
- Autograd transparently calculates the gradient
- Autograd is backpropagation
- All you need to do is specify the loss function
- The choice of the right loss function is paramount!
- In logistic regression second order methods bring enormous speed up compared to steepest gradient descent
- In neural networks in general stochastic gradient descent is the method of choice
- Bayesian logistic regression is hard: Laplace approximation
- Notebooks contain practical examples

Logistic Regression



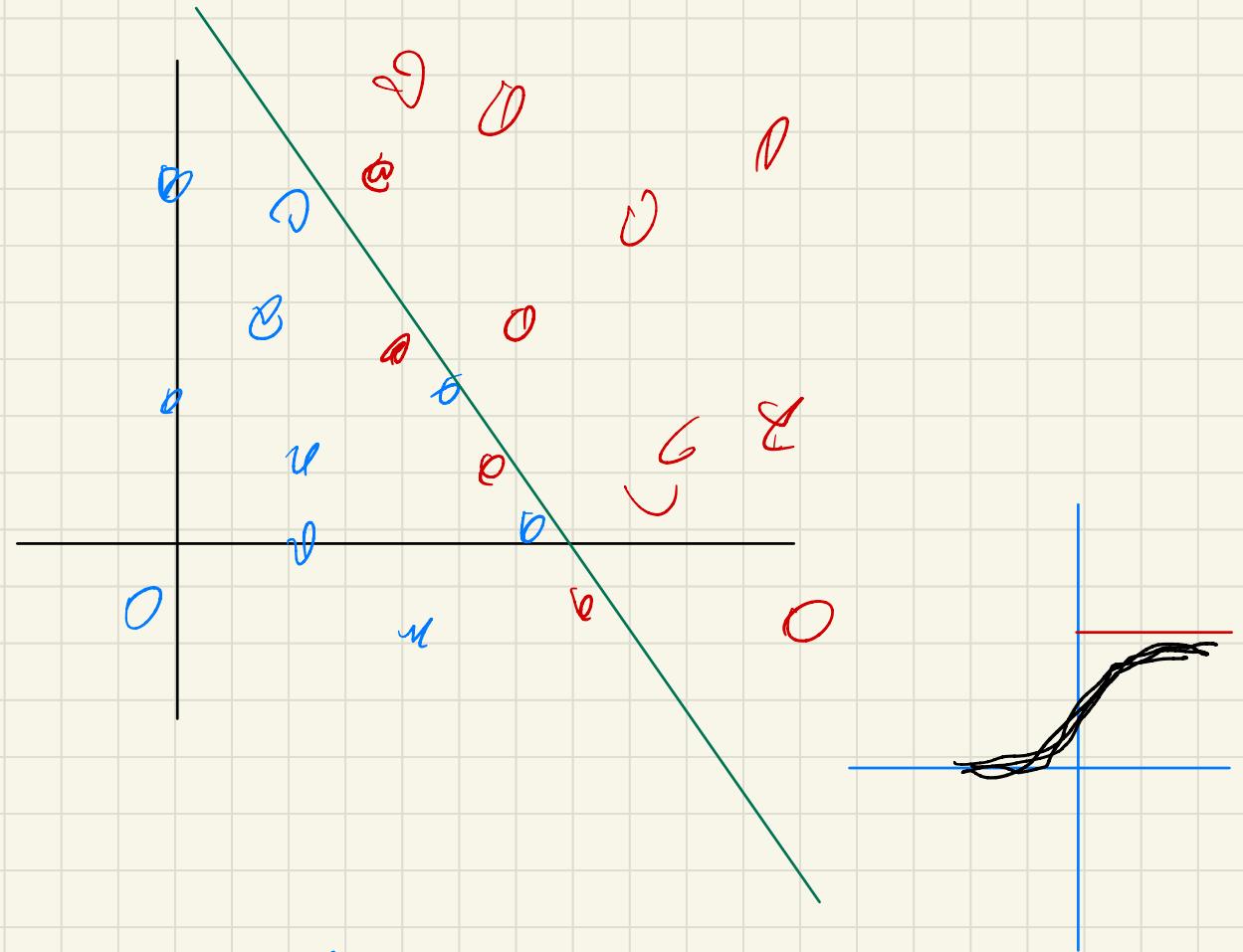
$$E = \frac{1}{2} \sum_i (o_i - c_i)^2$$

desired

Dataset
 (x_1, d_1)
 (x_2, d_2)
 (x_3, d_3)

$$d_i \in \{0, 1\}$$

$$\left. \begin{array}{l} \\ \\ \end{array} \right\} o_i = f(\sum_j w_j x_j)$$



$$o_i = f\left(\sum_j w_j x_j^{(i)}\right)$$

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

$$E = \frac{1}{2} \left(f(w_j x_j^{(i)}) - d_i \right)^2$$

$$\frac{\partial E}{\partial w_j} = 0$$

$$E = \frac{1}{2} \sum_i (f(w_j x_j^{(i)}) - d_i)^2$$

\int

$$\frac{\partial E}{\partial w_p}$$

$$E = \frac{1}{2} \left(f(\sum_j w_j x_j) - d \right)^2$$

$$= (f(\sum_j w_j x_j) - d) f'(\sum_j w_j x_j) \frac{\partial \sum_j w_j x_j}{\partial w_p}$$

$$= (o - d) f'(\sum_j w_j x_j) x_p$$

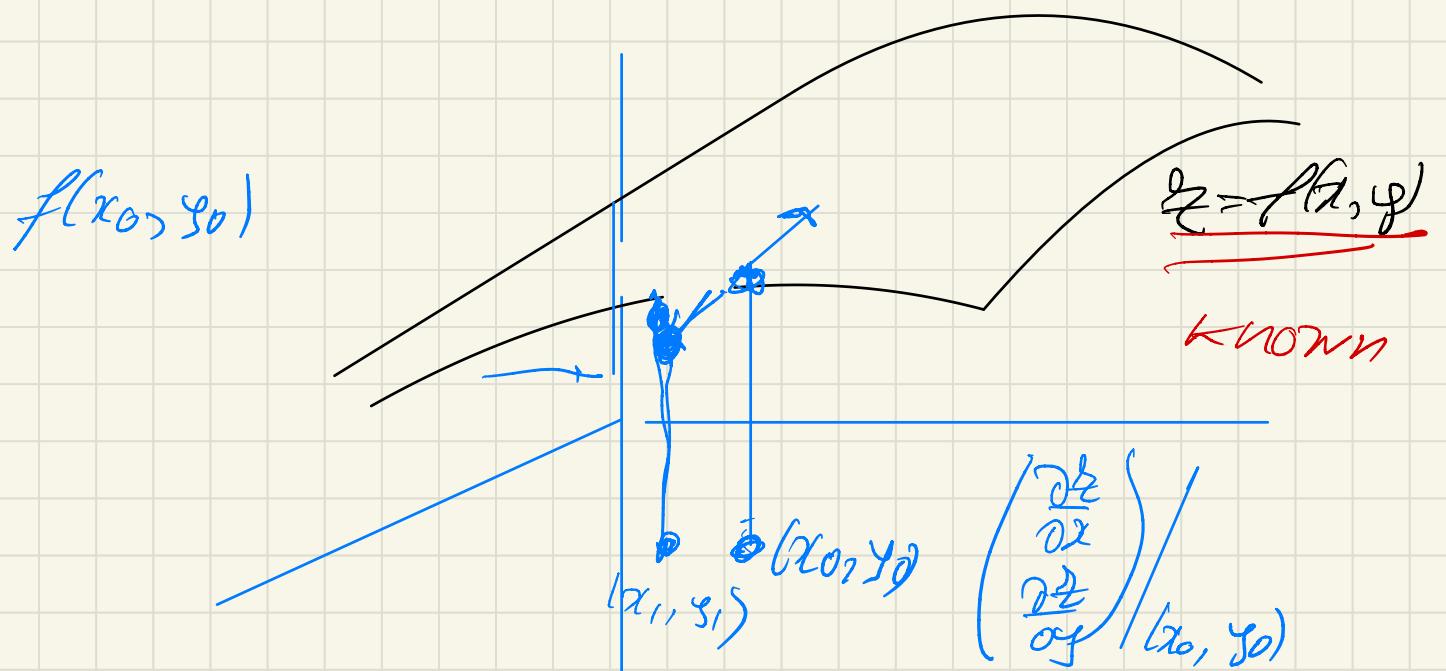
$$(o - d) f'(\sum_j w_j x_j) x_p$$

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

$$o = f(\sum_j w_j x_j)$$

$$f'(x) = f(1-f)$$

$$\frac{\partial E}{\partial w_p} = (o - d) o(1-o) x_p$$



Gradient: in the direction of fastest increase

$$(x_0, y_0)$$

$$\nabla z = \nabla f(x_0, y_0)$$

$$\begin{pmatrix} \frac{\partial z}{\partial x} \\ \frac{\partial z}{\partial y} \end{pmatrix} \Big|_{(x_0, y_0)}$$

$$\frac{\partial z}{\partial x}$$

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} - \varepsilon \begin{pmatrix} \frac{\partial z}{\partial x} \\ \frac{\partial z}{\partial y} \end{pmatrix} \Big|_{(x_0, y_0)} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\frac{\partial E}{\partial w_p} = (o - d) o(1-o) x_p$$

\vec{w}_0 is random

- we calculate error function
- we calculate the gradient Example

$$\frac{\partial E}{\partial w_0} \vdash \left. \frac{\partial E}{\partial \vec{w}} \right|_{\vec{w}_0}$$

$$o = .8$$

$$d = 0$$

$$(o - d) > \text{positive}$$

$$\vec{w}_1 = \vec{w}_0 - \alpha \left. \frac{\partial E}{\partial \vec{w}} \right|_{\vec{w}_0}$$

Learning rate ("small")

→ Only step $d - o \neq 0$

→ Direction x_p "SUBTRACT"
"ADD"

$$\begin{aligned} \text{ADD} : \quad \vec{w} &\rightarrow \vec{w} + \vec{x}_p \\ \text{SUB} : \quad \vec{w} &\rightarrow \vec{w} - \vec{x}_p \end{aligned}$$

COMP5611M: Machine Learning

Introduction to Logistic Regression

Nishant Ravikumar
Marc de Kamps

School of Computing

March 11, 2022

Learning Objectives

- Logistic Regression as a generative process
- More than one loss function to get a result ...
- ... and usually one is better than the other
- Obtaining the gradient is similar though
- Some datasets are unsuitable for logistic regression
- Multi class Logistic Regression

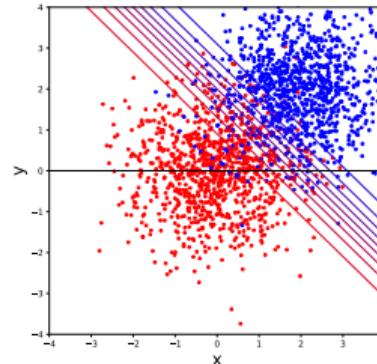
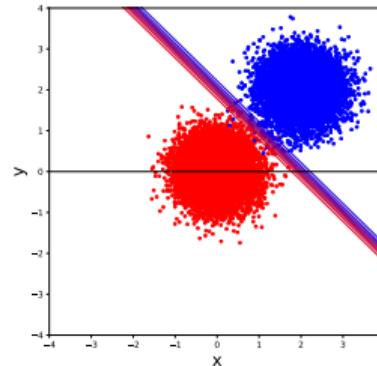
Learning Outcomes

At the end of this lecture you should be able to:

- Derive gradient rule for *cross entropy* loss
- Compare to the 'graded perceptron'
- Multi class logistic regression
- Explain why logistic regression is sometimes inappropriate
- Give rationales for multilayered networks
- Explain why it is called backpropagation

Logistic Regression as Generative Process

- So far we presented logistic regression as a graded perceptron
- With a perceptron-like gradient descent rule
- Logistic Regression also emerges from simple statistical considerations
- Imagine a process where a data point can be created by two different Gaussian processes
- Data point: We don't see the label, just two clusters
- For a new data point, can we establish which cluster it is likely to belong?



Logistic Regression as Generative Process

$$p(\vec{x} \mid \mathcal{C}_1)$$

- Assume two Gaussians:

$$\vec{\mu}_1 = (0, 0)^T \quad (1)$$

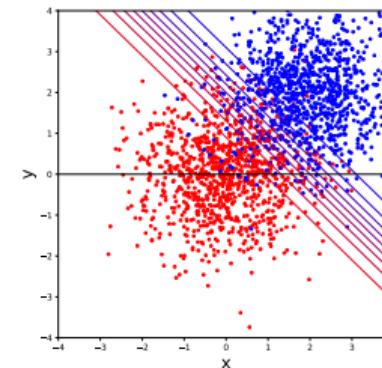
$$\vec{\mu}_2 = (2, 2)^T \quad (2)$$

- For both distributions same Σ :

$$\Sigma = \begin{pmatrix} k & 0 \\ 0 & k \end{pmatrix}$$

Probability that a red point ends up a given distance from (0,0):

$$p(x \mid \mathcal{C}_1) = \frac{1}{(2\pi)^{D/2}} \frac{1}{\Sigma^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_1)^T \Sigma^{-1} (x - \mu_1) \right\}$$



Logistic Regression as Generative Process

$$p(\mathcal{C}_1 \mid \vec{x})$$

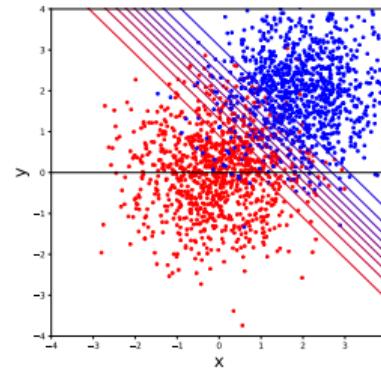
- We're not interested in $p(\vec{x} \mid \mathcal{C}_k)$
- We want $p(\mathcal{C}_1 \mid \vec{x})!$
- Bayes!

$$p(\mathcal{C}_1 \mid x) = \frac{p(x \mid \mathcal{C}_1)p(\mathcal{C}_1)}{p(x \mid \mathcal{C}_1)p(\mathcal{C}_1) + p(x \mid \mathcal{C}_2)p(\mathcal{C}_2)}$$

Rearrange:

$$p(\mathcal{C}_1 \mid x) = \frac{1}{1 + \exp(-a)},$$

$$a = \ln \frac{p(x \mid \mathcal{C}_1)p(\mathcal{C}_1)}{p(x \mid \mathcal{C}_2)p(\mathcal{C}_2)}$$



Logistic Regression as Generative Process

$$p(\mathcal{C}_1 \mid \vec{x})$$

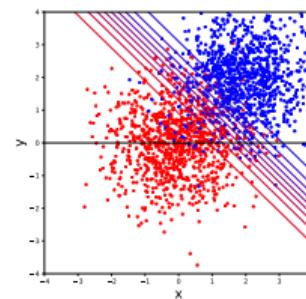
- Less abstract than it looks!
- Equal sized clusters: $p(\mathcal{C}_1) = p(\mathcal{C}_2) = 0.5$
- In the ratio, the normalisation of the Gaussian drops out

$$p(\mathcal{C}_1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0)$$

- where

$$\mathbf{w} = \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

$$w_0 = -\frac{1}{2} \boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_2^T \Sigma^{-1} \boldsymbol{\mu}_2 + \ln \frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}$$



Two quadratic terms in \vec{x} cancel due to the Σ being identical for both distributions

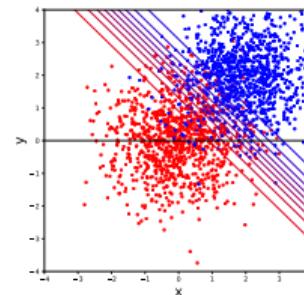
Logistic Regression as Generative Process

The Result

- Verify that for our choice of numbers

$$p(\mathcal{C}_1 \mid \mathbf{x}) = \sigma\left(-\frac{2}{k}(x_1 + x_2 - 2)\right) = \frac{1}{1 + e^{-\frac{2}{k}(x_1 + x_2 - 2)}}$$

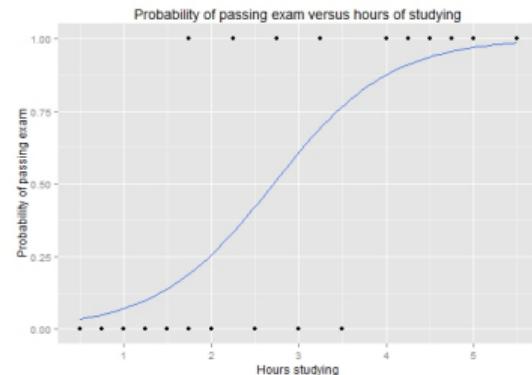
- For distributions with different covariances, isoprobability lines are quadratic
- Argument same
- Generative model: determine mean and covariances, then infer pdf
- A lot more work, but underpins VAEs



Logistic Regression as Generative Process

1D version

- You may already be familiar with the 1D example of logistic regression
- Same idea though
- Our example shows how linear or quadratic weighting of features can emerge
- In practice fit to find w, w_0 .
- Modified perceptron rule not great
- Image shows that the outcome is probability, what about likelihood?



Source: Wikipedia (CCBY-4.0)

Discriminative modelling

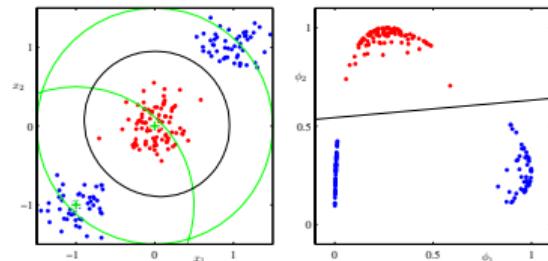
Cross Entropy

Example of problem that becomes simpler using the right basis functions

- Gaussian processes lead to sigmoid:

$$p(\mathcal{C}_1 | \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + w_0)}}$$

- Can I estimate \mathbf{w}, w_0 without finding μ_1, μ_2, Σ ?
- Basis functions again, like linear regression:
- Same reasons, the argument of the sigmoid is still linear in the weights
- Also, different covariances: quadratic isoprobabilities



Discriminative Modelling

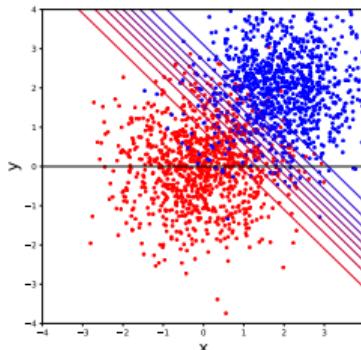
Likelihood

$$p(\mathcal{C}_1 \mid \phi) = \sigma(\mathbf{w}^T \phi),$$

- with dataset: $\{\phi_i, t_i\}$
- with $\phi_i = \phi(\mathbf{x}_i)$
- $t_i \in \{0, 1\}$. \mathbf{t} for (t_1, \dots, t_N)
- Treat points as independent and write down likelihood:

$$p(\mathbf{t} \mid \mathbf{w}) = \prod_{i=1}^N y_i^{t_i} (1 - y_i)^{1-t_i},$$

where $y_i = p(\mathcal{C}_1 \mid \phi_i) = \sigma(\mathbf{w}^T \phi(\mathbf{x})_i)$.



Discriminative Modelling

Log likelihood and Gradient

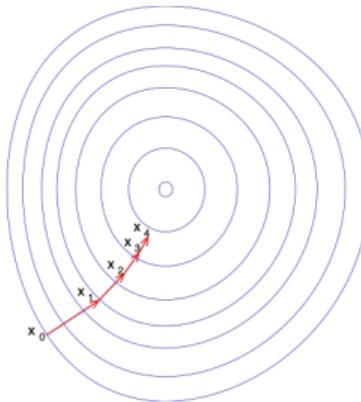
$$p(\mathbf{t} \mid \mathbf{w}) = \prod_{i=1}^N y_i^{t_i} (1 - y_i)^{1-t_i},$$

- The negative log likelihood is a lot simpler:

$$\mathcal{E}(\mathbf{w}) = -\ln p(\mathbf{t} \mid \mathbf{w}) = \sum_{i=1}^N t_i \ln y_i + (1-t_i) \ln(1-y_i)$$

- The gradient is: (verify!):

$$\frac{\partial \mathcal{E}}{\partial \mathbf{w}} = \sum_{i=1}^N (y_i - t_i) \phi_i$$



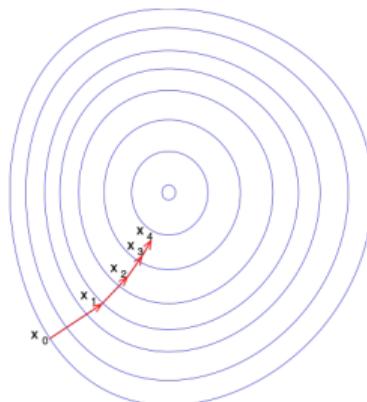
Discriminative Modelling

Log likelihood and Gradient

$$\frac{\partial \mathcal{E}}{\partial \mathbf{w}} = \sum_{i=1}^N (y_i - t_i) \phi_i$$

- Note the difference between discriminative and generative modelling:
- All we need are data points and labels
- But: generative modelling can simulate the dataset
- Compare to the 'perceptron' gradient
- Different *loss functions*: MSE loss ('perceptron') vs cross entropy

$$\frac{\partial \mathcal{E}}{\partial \mathbf{w}} = \sum_{i=1}^N y_i(1 - y_i) \phi_i$$



Loss Functions

For the graded perceptron, we minimise:

$$\mathcal{E} = \frac{1}{2} \sum_i (y_i - t_i)^2$$

where we adapted the notation to the one we used for logistic regression. For logistic regression, we minimise:

$$\mathcal{E}(\mathbf{w}) = \sum_{i=1}^N t_i \ln y_i + (1 - t_i) \ln(1 - y_i)$$

Now remember the lecture on information theory: Kullback-Leibler divergence:

$$\text{KL}(p \parallel q) = - \int p(\mathbf{x}) \ln q(\mathbf{x}) d\mathbf{x} - (- \int p(\mathbf{x}) \ln p(\mathbf{x}) d\mathbf{x})$$

For a discrete probability distribution this works out as:

$$\sum_{i \in C_j} p_i \ln q_i + \sum_{i \in C_j} p_i \ln p_i$$

Loss Functions

Cross Entropy vs MSE

But for two classes

$$\sum_{i \in C_1} p_i \ln q_i + \sum_{i \in C_2} p_i \ln p_i$$

works out as:

$$p_1 \ln q_1 + (1 - p_1) \ln(1 - q_1)$$

- We modify the first term of the KL-divergence to minimise. **Cross entropy**
- MSE loss minimises the numerical difference between p_i (true distribution) and q_i (our estimate of true distribution).
- Nothing directly refers to probabilities in MSE loss, purely a numerical comparison
- KL-divergence is a metric that compares probability distributions directly
- It severely punishes outliers with respect to true distribution
- **KL-divergence should be used for classification**

Multi class Logistic Regression

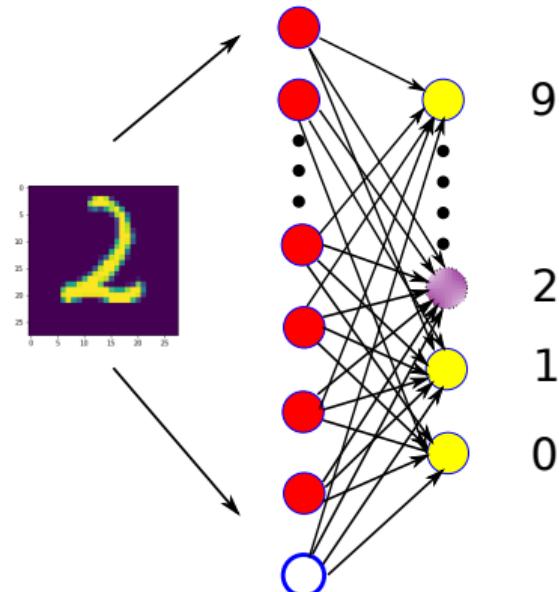
- Use softmax

$$p(\mathcal{C}_k \mid \phi) = y_k(\phi) = \frac{\exp a_k}{\sum_j \exp a_j},$$

- with

$$a_k = \mathbf{w}_k^T \phi.$$

- There is no minus sign in softmax
- Numerical evaluation can lead to large values
- The notebook on logistic regression describes a safe implementation
- Example: MNIST

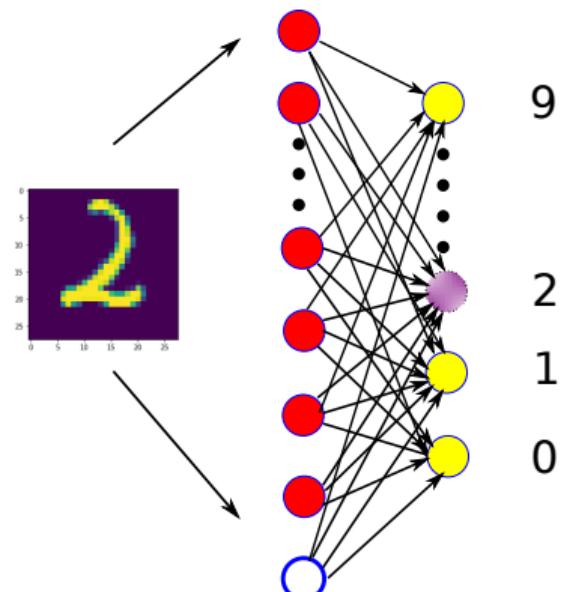


Multi class Logistic Regression

- Use softmax

$$p(\mathcal{C}_k \mid \phi) = y_k(\phi) = \frac{\exp a_k}{\sum_j \exp a_j},$$

- Target pattern is a one-hot-encoding: all outputs 0 except one.
- E.g. MNIST a pattern correspond to one, and only one numeral
- Output of softmax:
 $0 \leq a_k \leq 1, \sum_k a_k = 1$. Interpretable as probabilities!



Multi class Logistic Regression

Likelihood (K classes, N data points):

$$p(\mathbf{T} \mid \mathbf{w}_1, \dots, \mathbf{w}_k) = \prod_{k=1}^K \prod_{i=1}^N p(C_k \mid \phi_i^{t_{ik}}) = \prod_{k=1}^K \prod_{i=1}^N y_{ik}^{t_{ik}}.$$

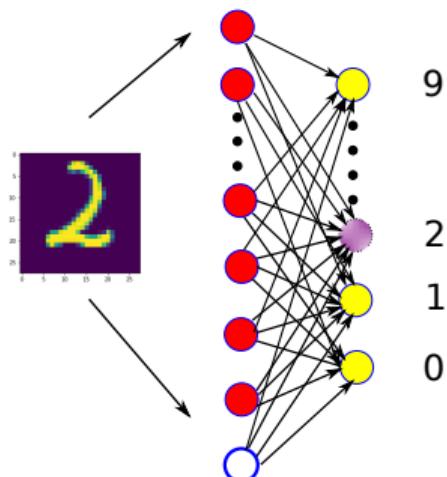
Here $y_{ik} = y_k(\phi_i)$ and \mathbf{T} is an $N \times K$ matrix of target variables with components t_{ik} .

Gradient (similar calculation for two classes but more work):

$$\frac{\partial E}{\partial \mathbf{w}_j} = \sum_{i=1}^N (y_{ij} - t_{ij}) \phi_i.$$

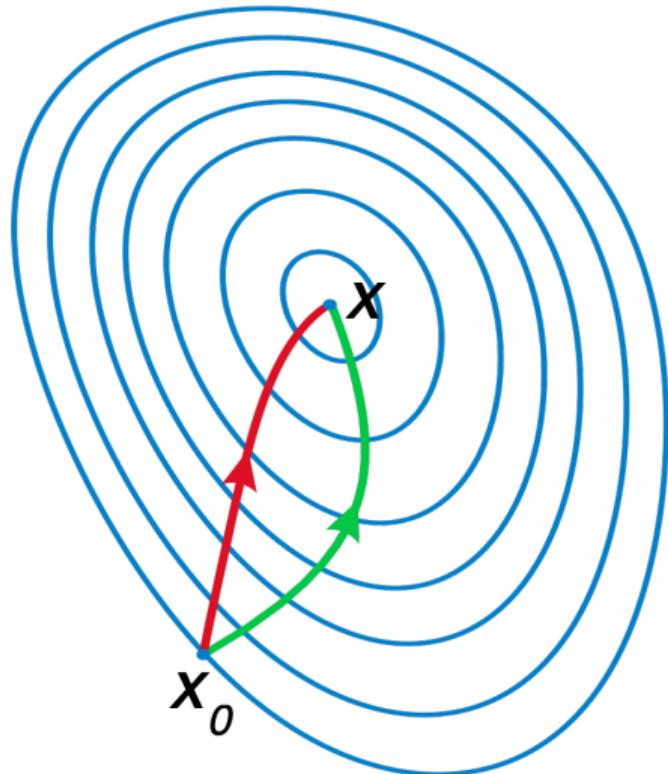
We will provide a notebook with this implementation.

About 85 % correct on test set (no cross validation yet)



Second Order Methods

- Logistic Regression has global maximum
- No saddle points
- This makes second order methods available: Newton-Raphson
- This does not work for neural networks: saddle points
- but is efficient for Logistic Regression
- See Reader for details



Outlook

- We already have seen that some datasets are not linearly separable
- Logistic regression will not do well on this
- Logistic regression can be seen as two-layer networks (see MNIST)
- We can add more layers: increased computational capability
- More sophisticated training method required **Backpropagation**

COMP5611M: Machine Learning

Decision Trees

Nishant Ravikumar
Marc de Kamps

School of Computing

March 21, 2022



UNIVERSITY OF LEEDS

Lecture Objectives

Objectives for this lecture:

- Recap of Classification concepts
- Introduction to Decision Trees (DTs) for classification tasks
- Learning algorithms for training DT classifiers: ID3

Learning Outcomes

At the end of this lecture, you should be able to

- Describe how DTs for classification work
- Define and compute entropy as impurity measure for training DTs
- Understand and explain the ID3 learning algorithm

What is Classification?

- Classification is a form of supervised learning
- Goal: assign each sample in a dataset to one or more pre-defined categories or *classes*



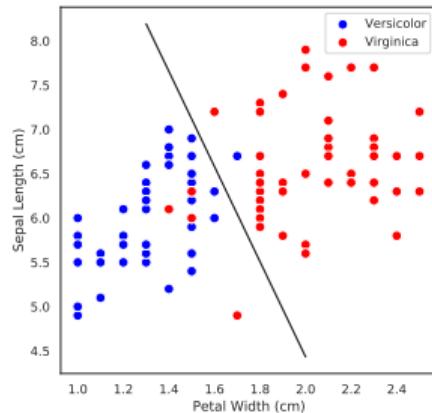
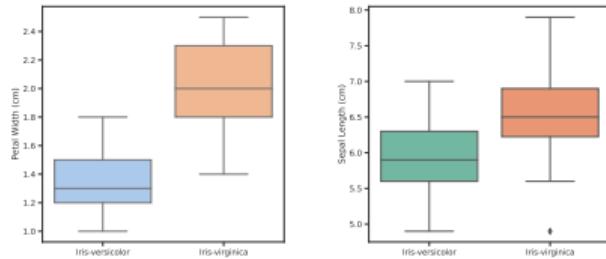
Figure: Classify samples into two distinct species of Iris flower

Source: Wikipedia; Creative Commons License CC BY-SA 3.0

- Typical pipeline: collect data, preprocessing, extract features, **train classifiers** and **predict using trained classifiers (inference)**

What is Classification?

- Another way of looking at it - approximate a decision boundary given some features
- Extract best (most discriminative) features from each flower image
 - Reduce data and dimensionality
 - Ex: Petal length, petal width, sepal length, sepal width, etc.
- Each sample can be represented by a set of features x with associated class label (species) y



What is Classification?

- More formally classification can be defined as:
- Given a dataset \mathbb{X} of N samples where each sample is represented by a feature set $\mathbf{x}_{i=1\dots N} \in \mathbb{X}$; and its corresponding discrete class label y_i denoting its membership to a specific class $\mathcal{C}_{k=1\dots K}$, we want to:

Learn a model that maps each feature set \mathbf{x}_i to its class label y_i by approximating the decision boundary(ies) that best separates samples belonging to each $\mathcal{C}_{k=1\dots K}$

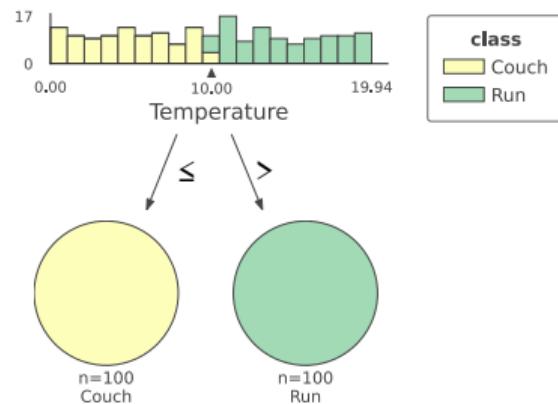
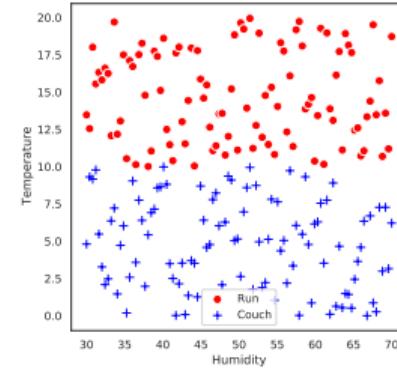
Classification Task	Feature Set	Class Labels
Sorting music into genres	Features derived from audio signals	Rock, Jazz, Blues
Categorising emails	Features derived from text data in emails	Spam, non-spam
Triaging COVID-19 patients according to severity of infection	Features derived from chest X-rays or CT images	High, moderate, low

Classification Techniques

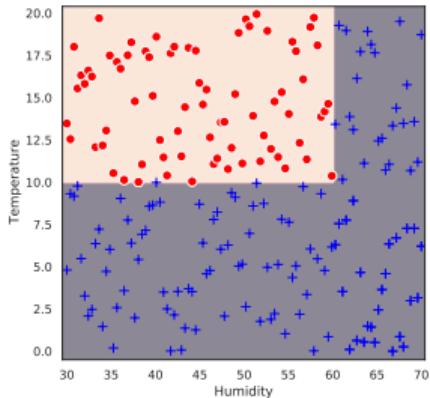
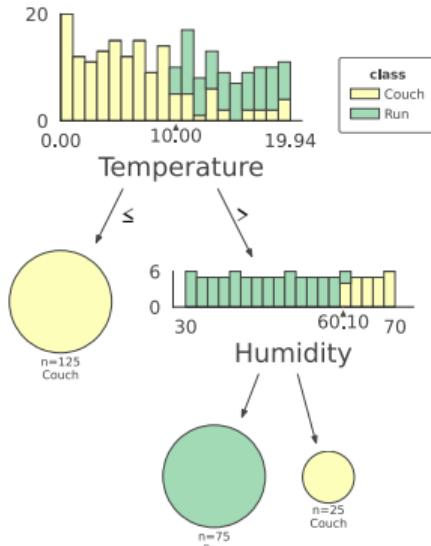
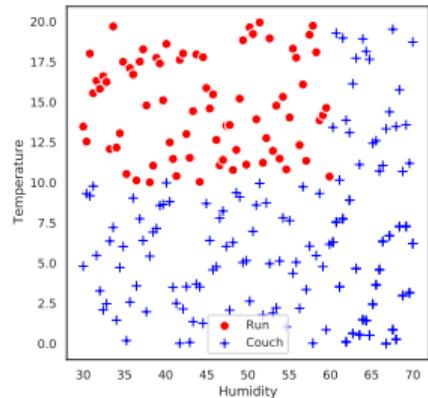
- Base classifiers
 - Decision Trees
 - Nearest-neighbour
 - Perceptron
 - Logistic regression & Multi-class logistic regression
 - Multi-layer perceptron (Neural Networks)
 - Support vector machines
 - Naive Bayes *
 - Bayesian belief networks
 -
- Ensemble classifiers
 - Bagging (applied to trees - Random Forests)
 - Boosting (applied to trees - gradient tree boosting, XGBoost)
 - Mixture of Experts
 -

Decision Trees

- DTs are a **non-parametric** supervised learning technique for classification and regression
- Learn simple decision rules to create axis-aligned partitions in feature space
- Special type of graph where each node can only have one parent (**except root node**)
 - DT components: **Root node**, branches/edges, internal nodes (parent/child), **leaves**
- Ex: DT to determine whether to go for a run or not?

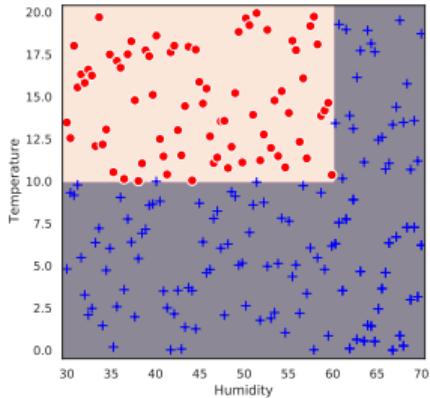
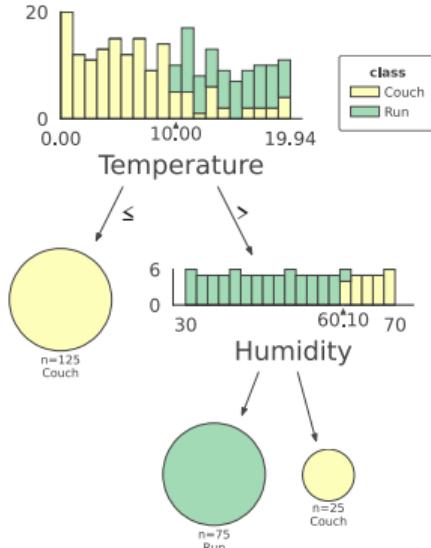
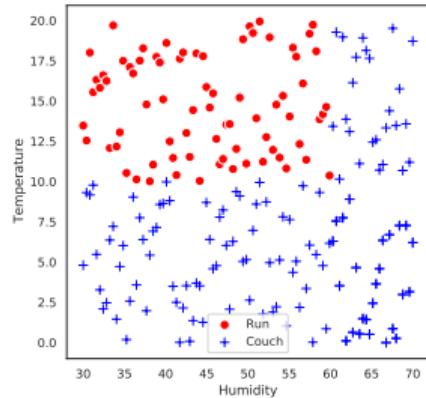


Decision Trees



- Now, let's say if $\text{humidity} > 60$ - 'sit on the couch', resulting in a **non-linear decision boundary**
- DTs can approximate ***non-linear decision boudaries*** easily
- What do you notice about **each split?** And about **each path** from the **root node** to **leaves?**

Decision Trees



- What do you notice about each split? And about each path from the root node to leaves?
- DT defines axis-aligned partitions in feature space, each path from root-to-leaf is a partitioned region

Decision Trees: Pros & Cons

- DTs are simple to implement and easy to interpret
- Can handle mixed feature types (continuous and/or categorical) and missing features
- Can be used for binary and multi-class classification, and regression
- DTs have high variance, i.e. sensitive to small changes in the data
- Not very good at handling continuous valued data (example in later lectures)
- Can easily overfit to your data - in practice need small but informative trees (i.e. regularisation)
- Vanilla DTs are however **not** competitive with other supervised learning approaches
- But, DTs when *ensembled* **are** competitive - *random forests* and *gradient boosting*
- Ensembling methods grow and combine multiple trees to provide a consensus prediction (i.e. reduce variance and bias) - covered in subsequent lectures
- Improvement in prediction accuracy with ensembling techniques is often at the price of interpretability

Decision Trees: Inference and Interpretability

- Another DT example: continuous valued features for tumour classification
- Class labels: Benign or Malignant
- Continuous features used: geometric properties - mean radius and mean smoothness
- Total number of samples = 569
- Train-test split for DT training and inference on 'unseen' data
- 90% for training and 10% for test

Decision Trees

- DT training on continuous valued features for tumour classification

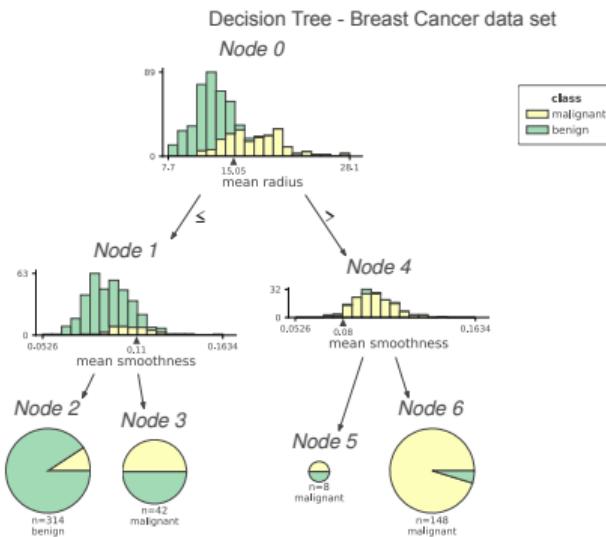


Figure: DT training to classify tumours as benign or malignant based on two geometric features. Using 90% for training.

Decision Trees

- DT inference on 'unseen' data
- Visualise prediction path for a random test sample - enables interpretation

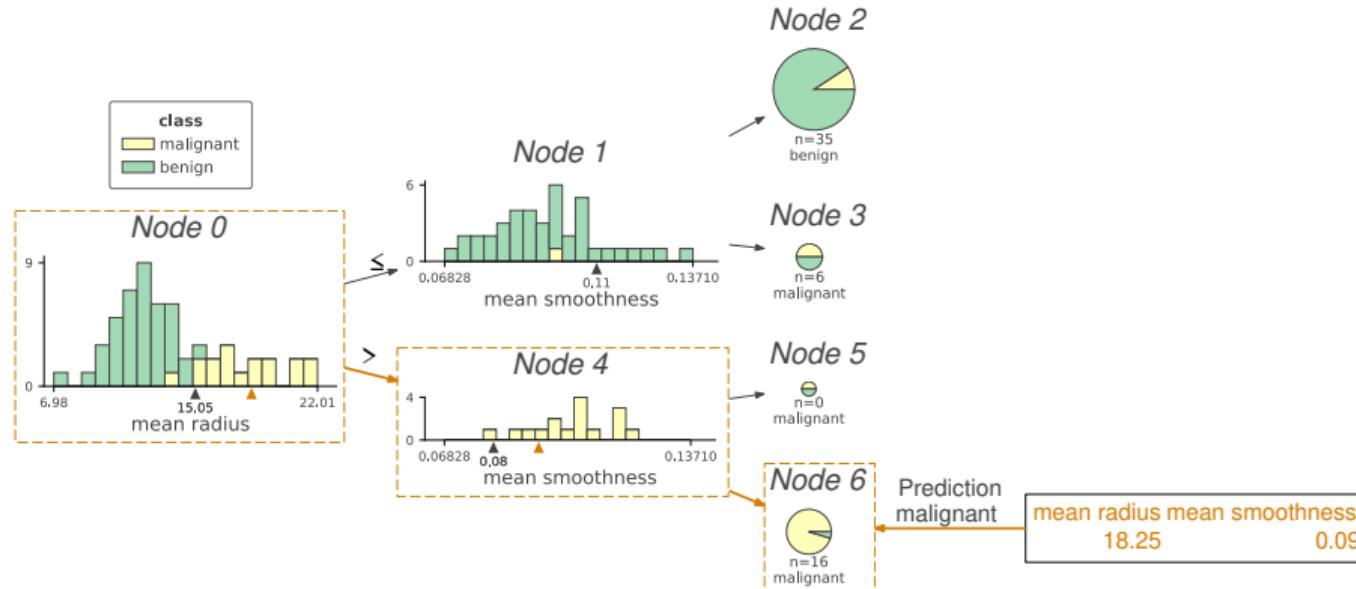


Figure: DT inference to classify tumours as benign or malignant based on two geometric features. Using 10% for test. Prediction path for random test sample in orange. Overall test accuracy = 0.88.

So how are DTs built?

- Start at root node by picking the *best* feature and split based on some criterion (in other words do a test)
- Grow the tree by splitting on next best feature sequentially - i.e. conditioned on the previous choice pick another attribute and do a test
- Assign class to leaf nodes by majority voting & repeat the procedure for other branches
- DTs are built by greedy, top-down, recursive partitioning of the feature space
- I.e. decide order and criteria for splitting features and partition feature space

So how are DTs built?

More formally:

- Given a dataset \mathbb{X} of N samples where each sample is represented by a feature set $\mathbf{x}_{i=1\dots N} \in \mathbb{X}$; and its corresponding discrete class label y_i denoting its membership to a specific class $\mathcal{C}_{k=1\dots K}$;
- We are looking for a splitting function s_p that partitions the feature space into regions R_p :

$$s_p(i, t) = (\{\mathbf{X} | \mathbf{x}_i \leq t, \mathbf{X} \in R_1\}, \{\mathbf{X} | \mathbf{x}_i > t, \mathbf{X} \in R_2\})$$

- Splits defined by feature ‘ i ’ and threshold ‘ t ’
- But how do we choose these splits? And when do we decide to stop splitting?
- We use information theory to determine these parameters

So how are DTs built?

- Let's recap the definitions of **information** and **entropy**
- You can think of information as a measure of **uncertainty** in an event/outcome
 - **High uncertainty** for improbable outcomes
 - **No uncertainty** if its deterministic or little uncertainty if very probable
- For a discrete random variable X , we can say the information content associated with an outcome $X = x$ is,

$$I(X = x) = \log_2 \frac{1}{P(X = x)} = -\log_2 P(X = x) \quad (1)$$

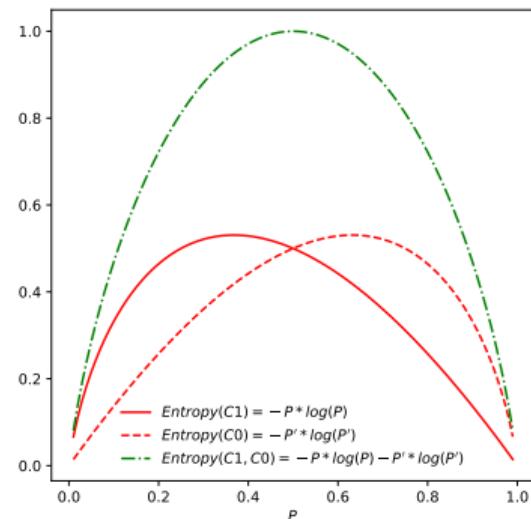
- We know from IT that entropy is the **expected** value of the information associated with possible outcomes:

$$H(X) = - \sum_{x \in X} P(X = x) \log_2 P(X = x) \quad (2)$$

- How can we use this to choose the order of splitting attributes?

So how are DTs built?

- For the case of two possible outcomes (or classes):
- Given probability $P(C_0) = P$ and $P(C_1) = 1 - P = P'$:
- $H(C_0) = -P * \log_2(P)$ & $H(C_1) = -P' * \log_2(P')$
- $H(C_0, C_1) = -P * \log_2(P) - P' * \log_2(P')$
- Entropy is maximum when both outcomes/classes are equally likely



- To train DTs - split feature space to maximise information gain
- And reduce entropy/uncertainty in classes with each split in resulting subsets/partitions
- *Information Gain = Entropy of Parent Node - weighted Entropy of Children Nodes*

Learning algorithms for building DTs

Many algorithms exist based on different node *impurity* measures:

- Hunt's algorithm (one of the earliest)
- ID3 (Iterative Dichotomiser 3): Uses *entropy* for defining splits, only works with categorical attributes and classification tasks
- CART (Classification and Regression Trees): Works with categorical and continuous attributes, uses *Gini Impurity/Index* for defining splits
- C4.5, C5.0 - continuous attribute extensions of ID3, suitable for classification
- SLIQ, SPRINT

Decision Trees: ID3

- Consider the following dataset with categorical features/attributes: Att1 - COVID symptoms, Att2 - Close contact with COVID+

Patient Name	COVID symptoms	Close contact with COVID+	Class Label
Hannah	No	No	<i>Don't Isolate</i>
Joe	No	Yes	<i>Don't Isolate</i>
Mark	Yes	Yes	<i>Isolate</i>
Cynthia	Yes	No	<i>Isolate</i>
Tom	No	No	<i>Don't Isolate</i>
Lisa	No	Yes	<i>Don't Isolate</i>

Decision Trees: ID3

- DT trained on categorical attributes of toy COVID data:
 - Here categorical attributes converted to numeric: Yes: 1; No: 0

Decision Tree - Toy example

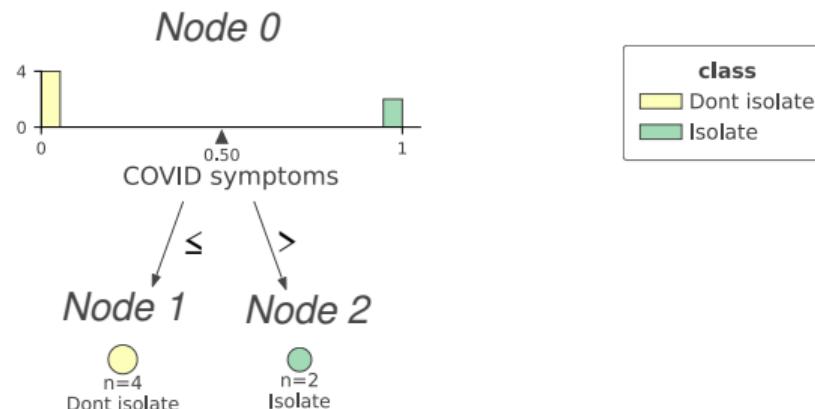
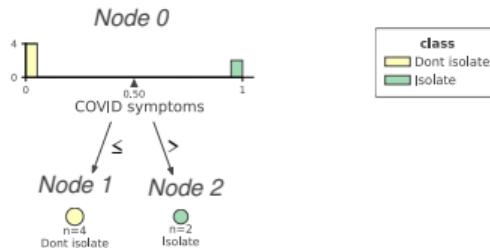


Figure: DT to classify tumours as benign or malignant based on two geometric features

- But why was COVID symptoms picked as the **first attribute?**

Decision Trees: ID3

Decision Tree - Toy example



Name	Att1	Att2	Class Label
Hannah	No	No	Don't Isolate
Joe	No	Yes	Don't Isolate
Mark	Yes	Yes	Isolate
Cynthia	Yes	No	Isolate
Tom	No	No	Don't Isolate
Lisa	No	Yes	Don't Isolate

- Entropy of the set of outcomes is: $H(X) = -\sum_{x \in X} P(X = x) \log_2 P(X = x)$
- Here we have $N = 6$, Isolate:2, Don't Isolate:4

$$P(X = \text{isolate}) = \frac{2}{6}; \quad P(X = \text{Don't isolate}) = \frac{4}{6} \quad (3)$$

- Entropy of the set is: $H(X) = -\frac{2}{6} \log_2 \frac{2}{6} - \frac{4}{6} \log_2 \frac{4}{6} = 0.9183$
- First lets calculate $H(X | Att1)$ and $H(X | Att2)$ and compare

Decision Trees: ID3

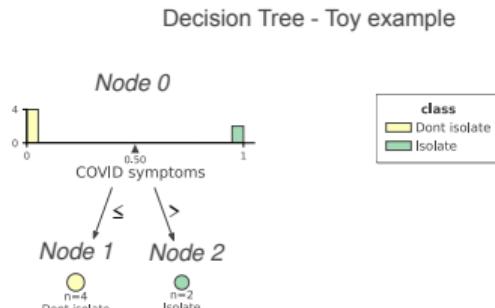


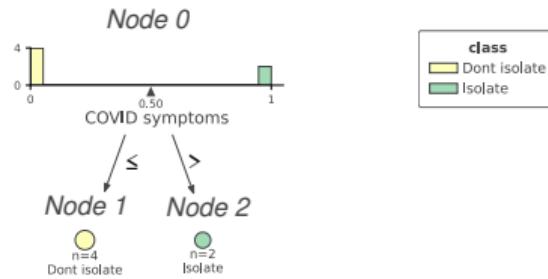
Figure: Remember 1: Yes, 0: No

Name	Att1	Att2	Class Label
Hannah	No	No	Don't Isolate
Joe	No	Yes	Don't Isolate
Mark	Yes	Yes	Isolate
Cynthia	Yes	No	Isolate
Tom	No	No	Don't Isolate
Lisa	No	Yes	Don't Isolate

- Entropy of the set is: $H(X) = -\frac{2}{6} \log_2 \frac{2}{6} - \frac{4}{6} \log_2 \frac{4}{6} = 0.9183$
- $H(X | Att1) = \frac{4}{6} * (-\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4}) + \frac{2}{6} * (-\frac{0}{2} \log_2 \frac{0}{2} - \frac{2}{2} \log_2 \frac{2}{2}) = 0.67 * (0) + 0.33 * (0) = 0$
- What does zero entropy indicate?
- $H(X | Att2) = \frac{3}{6} * (-\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3}) + \frac{3}{6} * (-\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3}) = 0.5 * (0.9183) + 0.5 * (0.9183) = 0.9183$

Decision Trees: ID3

Decision Tree - Toy example



Decision Tree - Toy example

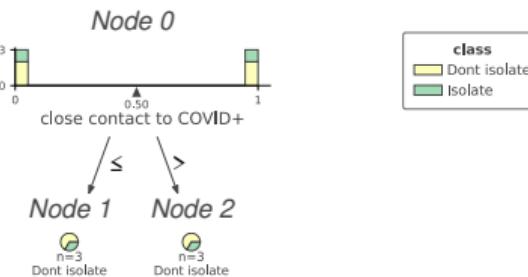


Figure: Remember 1: Yes, 0: No

- So, we have $H(X) = 0.9183$, $H(X | Att1) = 0$, & $H(X | Att2) = 0.9183$
- $IG(X, Att1) = H(X) - H(X | Att1) = 0.9183 - 0 = 0.9183$
- $IG(X, Att2) = H(X) - H(X | Att2) = 0.9183 - 0.9183 = 0$
- IG is higher by splitting on Attribute 1, i.e. COVID symptoms
- Splitting on Att1 results in zero entropy, i.e. **no uncertainty** in classes within these leaves/partitions (can stop splitting and growing the tree)

COMP5611M: Machine Learning

Decision Trees & Classification Metrics

Nishant Ravikumar

Marc de Kamps

School of Computing

March 21, 2022



UNIVERSITY OF LEEDS

Lecture Objectives

Objectives for this lecture:

- Decision Trees for Classification
- CART: Gini impurity and Gini Gain
- Classification metrics: Binary & Multi-class

Learning Outcomes

At the end of this lecture, you should be able to

- Describe how DTs for classification work (ID3 and CART)
- Define and compute entropy/Gini index as impurity measure for training DTs
- Understand and explain the ID3/CART learning algorithms
- Choose classification metrics to evaluate & compare performance of classification models

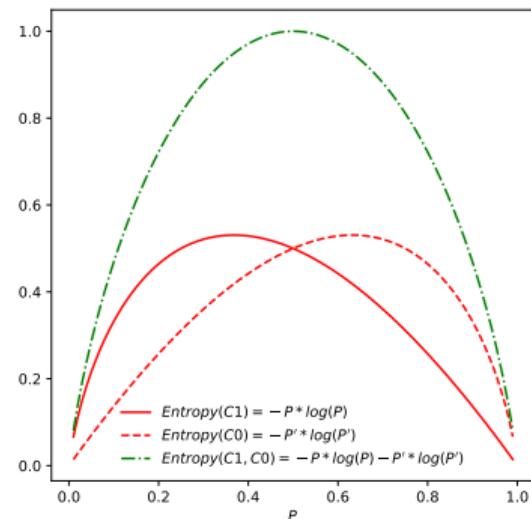
Recap: ID3 for Classification with DTs

- DTs are built by greedy, top-down, recursive partitioning of the feature space
- Decide order of discrete/categorical features for partitioning feature space based on **entropy**
- Entropy is the ***expected*** value of the information associated with possible outcomes:

$$H(X) = - \sum_{x \in X} P(X = x) \log_2 P(X = x) \quad (1)$$

Recap: ID3 for Classification with DTs

- For the case of two possible outcomes (or classes):
- Given probability $P(C_0) = P$ and $P(C_1) = 1 - P = P'$:
- $H(C_0) = -P * \log_2(P)$ & $H(C_1) = -P' * \log_2(P')$
- $H(C_0, C_1) = -P * \log_2(P) - P' * \log_2(P')$
- Entropy is maximum when both outcomes/classes are equally likely



- To train DTs - split feature space to maximise information gain
- And reduce entropy/uncertainty in classes with each split in resulting subsets/partitions
- *Information Gain = Entropy of Parent Node - weighted Entropy of Children Nodes*

Decision Trees: ID3

- Consider the following dataset with categorical features/attributes: Att1 - COVID symptoms, Att2 - Close contact with COVID+

Patient Name	COVID symptoms	Close contact with COVID+	Class Label
Hannah	No	No	<i>Don't Isolate</i>
Joe	No	Yes	<i>Don't Isolate</i>
Mark	Yes	Yes	<i>Isolate</i>
Cynthia	Yes	No	<i>Isolate</i>
Tom	No	No	<i>Don't Isolate</i>
Lisa	No	Yes	<i>Don't Isolate</i>

Decision Trees: ID3

- DT trained on categorical attributes of toy COVID data:
 - Here categorical attributes converted to numeric: Yes: 1; No: 0

Decision Tree - Toy example

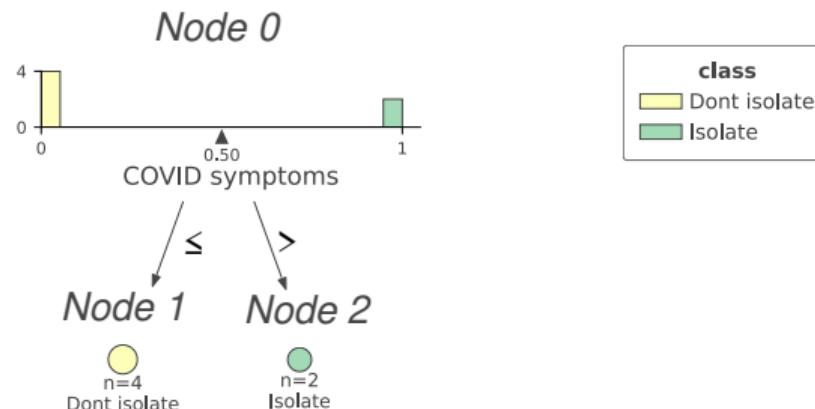
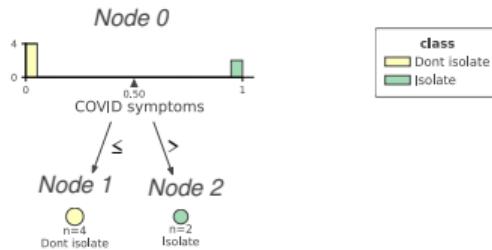


Figure: DT to classify tumours as benign or malignant based on two geometric features

- But why was COVID symptoms picked as the **first attribute?**

Decision Trees: ID3

Decision Tree - Toy example



Name	Att1	Att2	Class Label
Hannah	No	No	Don't Isolate
Joe	No	Yes	Don't Isolate
Mark	Yes	Yes	Isolate
Cynthia	Yes	No	Isolate
Tom	No	No	Don't Isolate
Lisa	No	Yes	Don't Isolate

- Entropy of the set of outcomes is: $H(X) = -\sum_{x \in X} P(X = x) \log_2 P(X = x)$
- Here we have $N = 6$, Isolate:2, Don't Isolate:4

$$P(X = \text{isolate}) = \frac{2}{6}; \quad P(X = \text{Don't isolate}) = \frac{4}{6} \quad (2)$$

- Entropy of the set is: $H(X) = -\frac{2}{6} \log_2 \frac{2}{6} - \frac{4}{6} \log_2 \frac{4}{6} = 0.9183$
- First lets calculate $H(X | Att1)$ and $H(X | Att2)$ and compare

Decision Trees: ID3

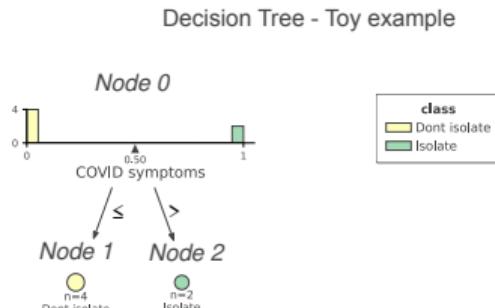


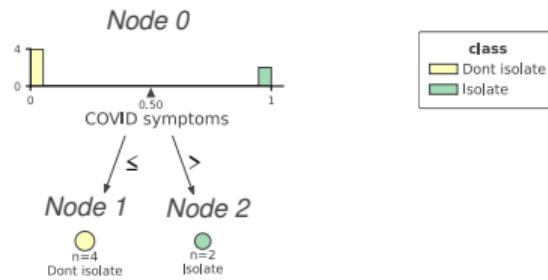
Figure: Remember 1: Yes, 0: No

Name	Att1	Att2	Class Label
Hannah	No	No	Don't Isolate
Joe	No	Yes	Don't Isolate
Mark	Yes	Yes	Isolate
Cynthia	Yes	No	Isolate
Tom	No	No	Don't Isolate
Lisa	No	Yes	Don't Isolate

- Entropy of the set is: $H(X) = -\frac{2}{6} \log_2 \frac{2}{6} - \frac{4}{6} \log_2 \frac{4}{6} = 0.9183$
- $H(X | Att1) = \frac{4}{6} * (-\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4}) + \frac{2}{6} * (-\frac{0}{2} \log_2 \frac{0}{2} - \frac{2}{2} \log_2 \frac{2}{2}) = 0.67 * (0) + 0.33 * (0) = 0$
- What does zero entropy indicate?
- $H(X | Att2) = \frac{3}{6} * (-\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3}) + \frac{3}{6} * (-\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3}) = 0.5 * (0.9183) + 0.5 * (0.9183) = 0.9183$

Decision Trees: ID3

Decision Tree - Toy example



Decision Tree - Toy example

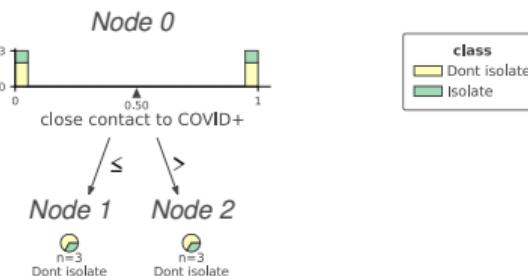
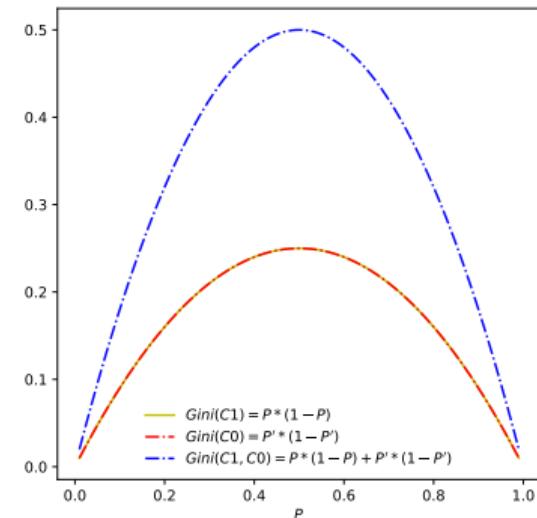


Figure: Remember 1: Yes, 0: No

- So, we have $H(X) = 0.9183$, $H(X | Att1) = 0$, & $H(X | Att2) = 0.9183$
- $IG(X, Att1) = H(X) - H(X | Att1) = 0.9183 - 0 = 0.9183$
- $IG(X, Att2) = H(X) - H(X | Att2) = 0.9183 - 0.9183 = 0$
- IG is higher by splitting on Attribute 1, i.e. COVID symptoms
- Splitting on Att1 results in zero entropy, i.e. **no uncertainty** in classes within these leaves/partitions (can stop splitting and growing the tree)

Decision Trees: CART

- Alternative impurity measure for training DTs for classification: **Gini index/impurity**
- Used by CART algorithm for deciding order of features for splitting feature space
- Advantage over ID3 is **CART** can be used for **continuous and categorical** features
- Gini index for two classes given by:
$$G(C0, C1) = \sum_{i=1}^2 P_i(1 - P_i)$$
- Gini index is maximum when both classes are equally likely
- DT growing/learning process exactly the same as ID3, based on **maximising Gain** but using Gini index



Decision Trees: CART

Decision Tree - Toy example

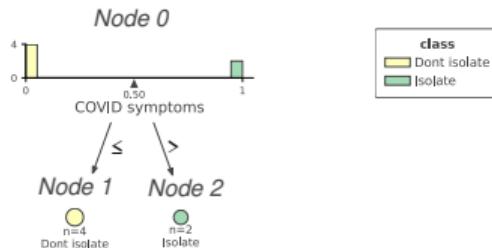


Figure: Remember 1: Yes, 0: No

Name	Att1	Att2	Class Label
Hannah	No	No	Don't Isolate
Joe	No	Yes	Don't Isolate
Mark	Yes	Yes	Isolate
Cynthia	Yes	No	Isolate
Tom	No	No	Don't Isolate
Lisa	No	Yes	Don't Isolate

- Gini of the set is: $G(X) = \frac{2}{6} * (1 - \frac{2}{6}) + \frac{4}{6} * (1 - \frac{4}{6}) = 0.44$
- $G(X | Att1) = \frac{4}{6} * (\frac{4}{4} * (1 - \frac{4}{4}) + \frac{0}{4} * (1 - \frac{0}{4})) + \frac{2}{6} * (\frac{0}{2} * (1 - \frac{0}{2}) + \frac{2}{2} * (1 - \frac{2}{2})) = 0$
- What does zero Gini Index indicate?
- $G(X | Att2) = \frac{3}{6} * (\frac{2}{3} * (1 - \frac{2}{3}) + \frac{1}{3} * (1 - \frac{1}{3})) + \frac{3}{6} * (\frac{2}{3} * (1 - \frac{2}{3}) + \frac{1}{3} * (1 - \frac{1}{3})) = 0.5 * (0.44) + 0.5 * (0.44) = 0.44$
- Gini Gain: $GG(X | Att1) = G(X) - G(X | Att1) = 0.44, GG(X | Att2) = 0$

CART: Classification with Continuous Features

- DTs are grown by choosing the order of categorical features to split, which maximises Information/Gini Gain
- With CART and C4.5 can do the same with **continuous features**
- This is done by creating discrete intervals for each feature and testing different splits:
 $s_p(i, t) = (\{\mathbf{X} | \mathbf{x}_i \leq t, \mathbf{X} \in R_1\}, \{\mathbf{X} | \mathbf{x}_i > t, \mathbf{X} \in R_2\})$
- As before, the threshold (t) defining the intervals and the order in which features are chosen, is determined by **maximising the Gini Gain**
- I.e. iteratively consider all possible partitions ($\mathbf{x}_i \leq t$) & ($\mathbf{x}_i > t$) and select (t) which maximises $GG(\mathbf{X} | \mathbf{x}_i) = G(\mathbf{X}) - \frac{N_1}{N}G(R_1) - \frac{N_2}{N}G(R_2)$
- Maximising $GG(\mathbf{X} | \mathbf{x}_i)$ favours splits of \mathbf{X} into partitions/regions R_1 & R_2 that are as '**pure**' or **homogeneous** in classes as possible

CART: Classification with Continuous Features

Let's look at an example for tumour classification:

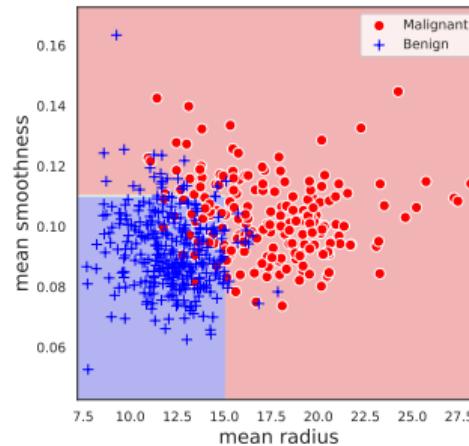
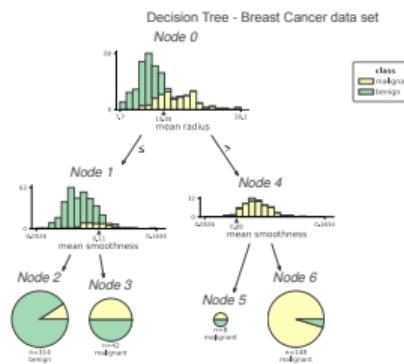
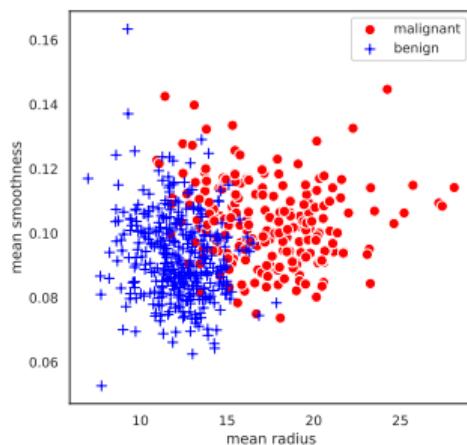
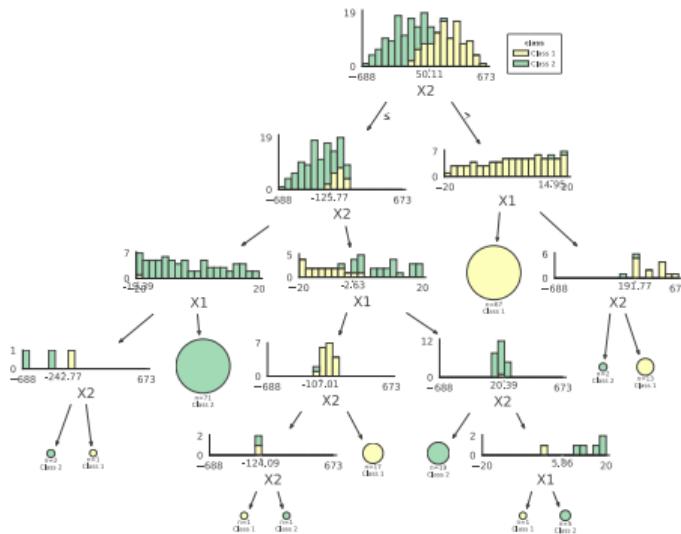
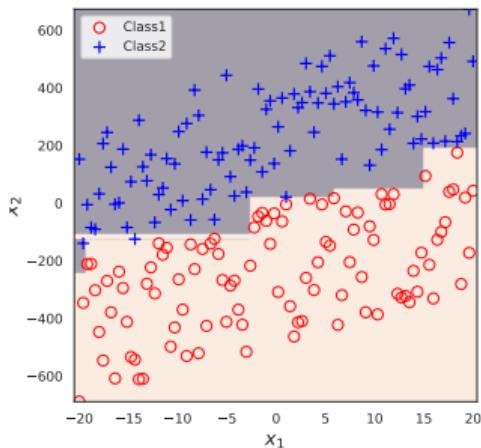


Figure: Accuracy on training set = 0.88

- **Multiple intervals** tested for features: mean radius, mean smoothness, splits defined based on GG
- Resulting tree does reasonably well with a termination criterion of maximum depth of tree = 2
- Another way to think of DTs: defines a **splitting function** that is **constant** in each R_p

CART: Classification with Continuous Features

Drawbacks of DTs for continuous features:



- Diagonal decision boundaries cause issues - several splits required
- Can combine multiple features to address this - but scales badly when number of features and classes grow

Evaluation of Classification Models

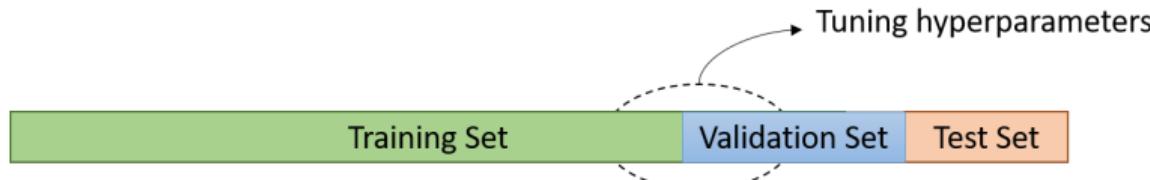
- What are **classification metrics** & why do we need them?

Common Classification Metrics
Accuracy
Precision
Recall/Sensitivity
Specificity
F1-score
ROC/AUC
Precision-Recall Curve
...

- Need to **quantitatively evaluate** performance of classifiers to choose the best option

Evaluation of Classification Models

- As with regression, good practice to partition your data into **Training**, **Validation** and **Test** sets
- If you have a very large dataset of labelled samples - randomise and split



- Why do you need separate **Validation** and **Test** sets? Any ideas?
- What if you have a small dataset? - Need **cross-validation!**
- As discussed previously - **Never mix** samples between these distinct sets!
 - Mixing samples - no idea of true performance of models
 - What we want to know - do our models **generalise** well to unseen/test data?

Binary Classification Metrics

- Separation of inputs into one of two classes: Positive (1) and Negative (0)
- Common metric: Accuracy

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}} = \frac{TP + TN}{N}$$

- TP: True Positives, TN: True Negatives, FP: False Positives, FN: False Negatives
- But is Accuracy enough? Can we say our model is good based on just Accuracy?
- So, what else can we do?

Confusion Matrix

- Consider a binary classification problem: Positive (Heart Attack), Negative (Healthy)
- Dataset available has **class imbalance** - i.e. one class under-represented relative to the other
- $N = 100$ samples in the dataset, 90 from **Negative Class**, 10 from **Positive Class**
- Train a classifier and obtain *Accuracy* = 0.90. Can we say our classifier is good?
- No we can't! We need to understand the **types of errors** incurred - i.e. **FP** and **FN**
- Best way to get an overall sense of performance - **Confusion Matrix**

	Predicted Negative (0)	Predicted Positive (1)
Negative Class (0)	TN	FP
Positive Class (1)	FN	TP

- Confusion Matrix - summary of **Predicted** and **Ground Truth** Classes
- Tells us about the types of errors, helps understand other metrics

Problems with Accuracy

- Let's consider the same classification problem again - we train two more classifiers
- All three models have an *Accuracy* = 0.90 - are they equal?
- Let's take a look at their confusion matrices:

Table: Model 1

	Pr. (0)	Pr. (1)
GT (0)	90	0
GT (1)	10	0

Table: Model 2

	Pr. (0)	Pr. (1)
GT (0)	85	5
GT (1)	5	5

Table: Model 3

	Pr. (0)	Pr. (1)
GT (0)	80	10
GT (1)	0	10

- So which is the **best** model? Any ideas?
- Also notice that: $Accuracy = \frac{TN+TP}{TN+TP+FN+FP}$

Classification Metrics

- So it's clear we need additional metric to thoroughly evaluate performance
- Using the **confusion matrix** we can **derive** other metrics:
- $Precision = \frac{TP}{TP+FP}$ → when you want to minimise FP
- $Sensitivity/Recall = \frac{TP}{TP+FN}$ → when you want to minimise FN
- $Specificity = \frac{TN}{TN+FP}$ → when care about Negative class
- $F_1 - score = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$ → summarises confusion matrix
- F_1 score does not consider TN → **sensitive** to which category is considered **Positive** class
- Alternative is **Matthew's Correlation Coefficient** which is **invariant** to this definition:

$$MCC = \frac{TP*TN - FP*FN}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$$

Receiver Operating Characteristic (ROC) Curve

- We've looked at metrics based on discrete classes predicted by classifiers
- Some classifiers output probabilities (e.g. Logistic regression)
- Can be seen as degree of confidence of the classifier in predicted classes
- **ROC curve** → evaluate performance using predicted class membership probabilities
- Summarises trade-off between TPR (Recall/Sensitivity) and FPR (1 - Specificity)
- How is ROC evaluated? Why isn't it a single point?

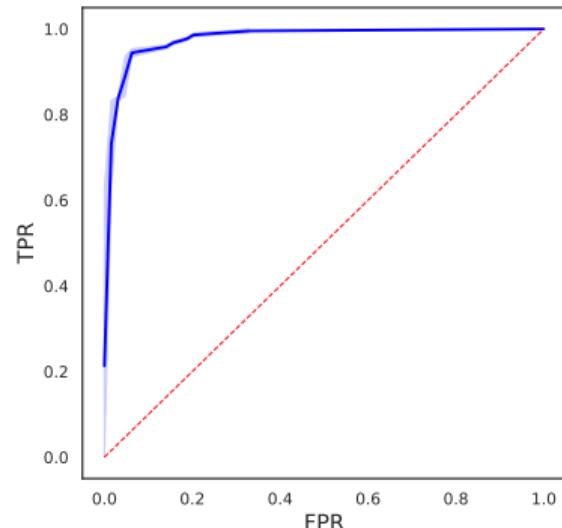


Figure: ROC curve

ROC-AUC

- ROC curve evaluated by **changing decision threshold** (default: 0.5)
- Changing decision threshold changes the number of FP and FN
- ROC evaluated using **different thresholds** to estimate $TPR = \frac{TP}{TP+FN}$, $FPR = \frac{FP}{FP+TN}$
- But what if we want to **automatically** choose the best model?
- Can estimate **Area Under the Curve (AUC)** as a measure of model performance
- Can also compute Precision-Recall curves and compute AUC

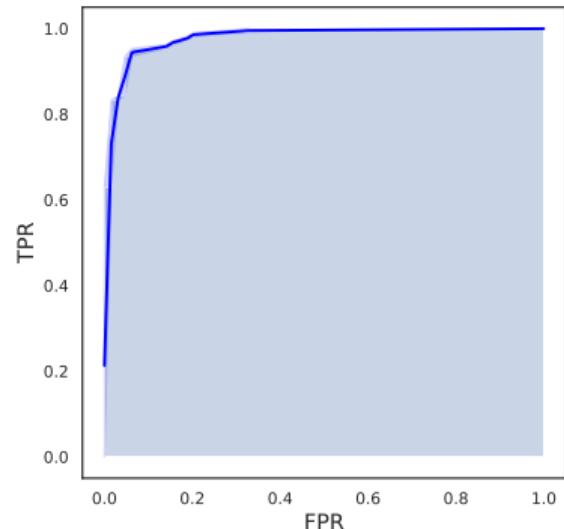


Figure: $AUC = 0.98$

Multi-Class Classification Metrics

- We've discussed metrics for binary classification, what about multi-class?
- Can re-use the same metrics as for binary classification
- For example let's start with the confusion matrix
- But how do we compute Precision, Recall, etc? Any ideas?

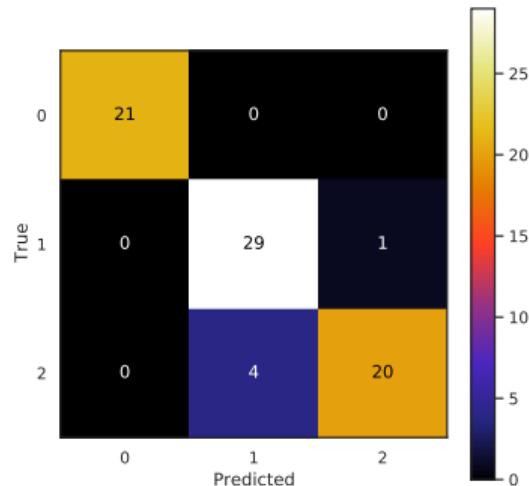


Figure: Three-class confusion matrix

Multi-Class Classification Metrics

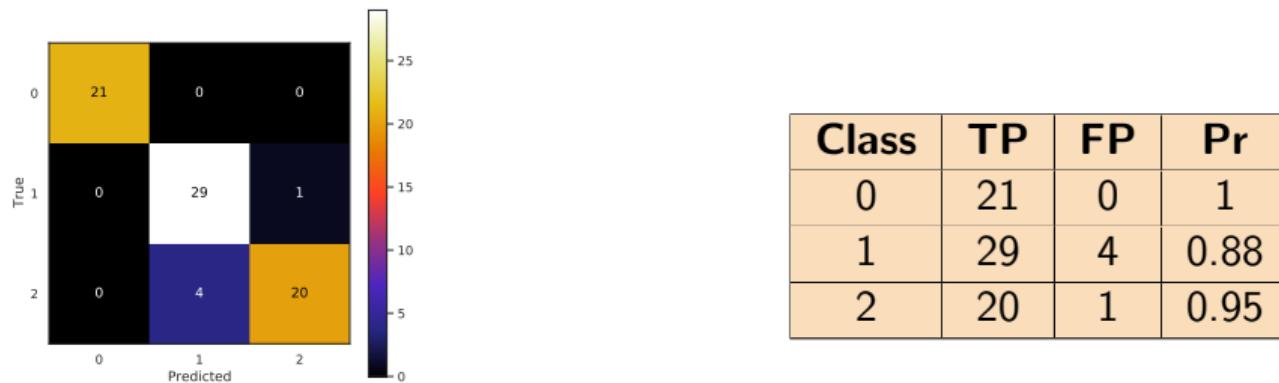


Figure: Three-class confusion matrix

- We can calculate all metrics per class by treating it as **one vs the rest**
- This allows us to calculate TP, TN, FP, FN for each class
- Can then compute different types of averages of each metric: **Micro-avg** - each sample contributes equally, **Macro-avg** - each class contributes equally, **Weighted-avg** - each classes contribution is weighted by its size

Pitfalls, Context & Model Fairness

- No 'one fits all' evaluation metric - application/context dependent
- Bias : Is your model 'fair'?
 - Is a subset of your data under-represented based on an attribute (not class!)
 - Or is there an inherent bias in your model's predictions for a specific group?

Table: Overall Performance

	Pr. (0)	Pr. (1)
GT (0)	70	5
GT (1)	5	20

Table: Male Subset

	Pr. (0)	Pr. (1)
GT (0)	40	0
GT (1)	0	10

Table: Female Subset

	Pr. (0)	Pr. (1)
GT (0)	30	5
GT (1)	5	10

- Overall: Precision = 0.80, Recall = 0.80 - looks promising
- Male subset: Precision = 1.0, Recall = 1.0 - amazing!
- Female subset: Precision: 0.66, Recall = 0.66 - not good!

COMP5611M: Machine Learning

Regression Trees & Ensemble Methods

Nishant Ravikumar

Marc de Kamps

School of Computing

March 25, 2022



Lecture Objectives

Objectives for this lecture:

- Decision trees for regression (regression trees)
- Introduction to ensemble methods
- Bagging for decision trees (Random Forests)

Learning Outcomes

At the end of this lecture, you should be able to

- Describe how regression trees are trained
- Describe why ensembling improves performance of DTs
- Describe what Random Forests are and how they are trained

Regression Trees

- **CART** is the most common algorithm used for regression with DTs
- Similar to Classification but targets y_i are now continuous, i.e. greedy, top-down, recursive splits
- We are looking for $\mathbf{y} = f(\mathbf{x}) + \epsilon$, except now $f(\cdot)$ is estimated **non-parametrically**
- We are still look for a splitting function that partitions are feature space:
 $s_p(i, t) = (\{\mathbf{X}|\mathbf{x}_i \leq t, \mathbf{X} \in R_1\}, \{\mathbf{X}|\mathbf{x}_i > t, \mathbf{X} \in R_2\})$
- But what **node impurity** do we use to define splits?
- And how do we **estimate targets** following splitting of feature space? Any ideas?

Regression Trees

- We use **MSE as the node impurity measure** to decide on the threshold and order of splits
- Lets say we have partitioned into M regions: R_1, R_2, \dots, R_m and estimated a constant value in each c_m

$$f(\mathbf{x}) = \sum_{i=1}^M c_m S_p(\mathbf{x} \in R_m)$$

- And if criterion for each split is minimisation of: $\sum(y_i - f(x_i))^2$, what is each c_m ?
- Its just the average of y_i in each R_m i.e.

$$c_m = \text{average}(y_i \mid x_i \in R_m)$$

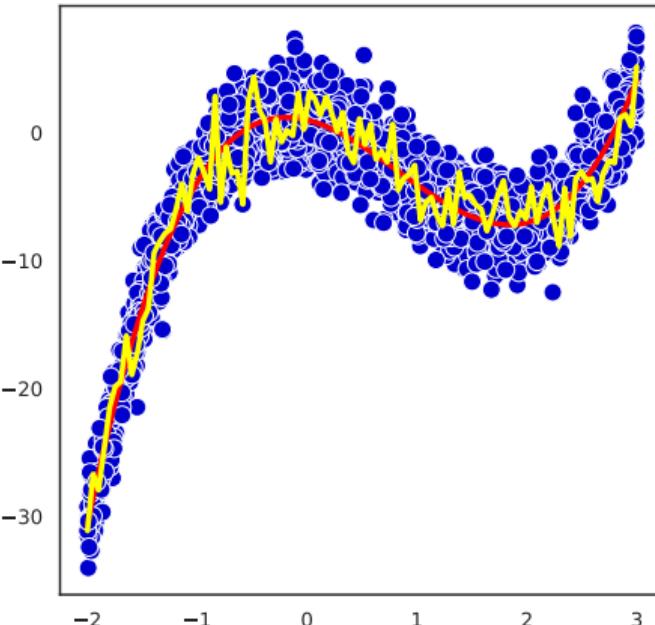
- So in regression trees - order of features and threshold to split on is given by:

$$\arg \min_{i,t} [MSE(y_i \mid x_i \leq t)_{x_i \in R_1(i,t)} + MSE(y_i \mid x_i > t)_{x_i \in R_2(i,t)}]$$

- Classification - class assigned to leaves by majority vote
- Same for regression, except **constant value** assigned as average of each partitioned region

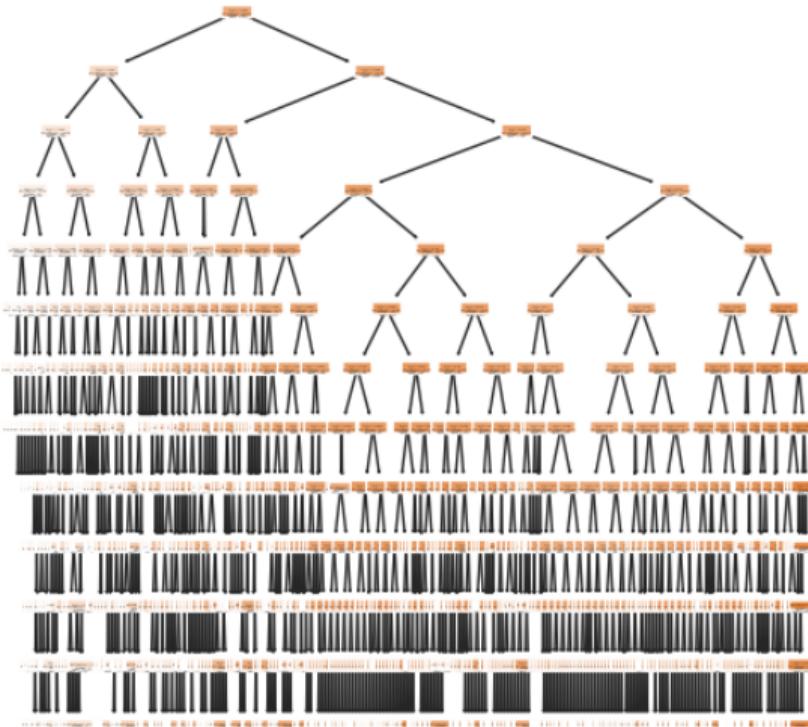
Regression Trees

- Lets look at an example - given some features x and targets y
- First fit a polynomial linear regression model to the data (red)
- And a regression tree (yellow), what do you notice?



Regression Trees

- Previous example seems to be overfitting!
Let's look at the resulting tree
- As with classification, we need
regularisation to prevent overfitting

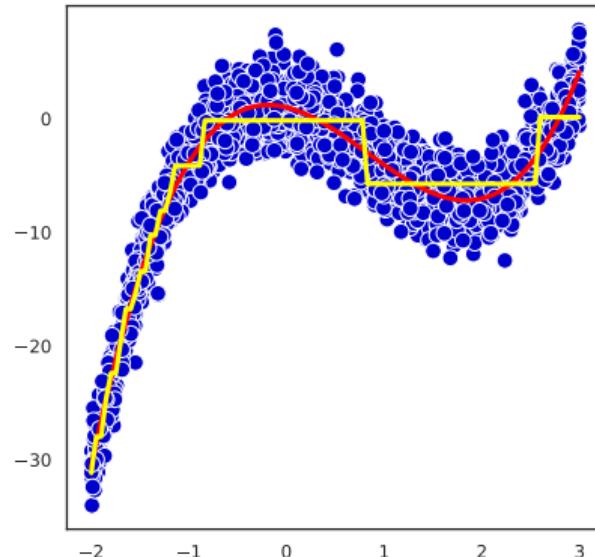
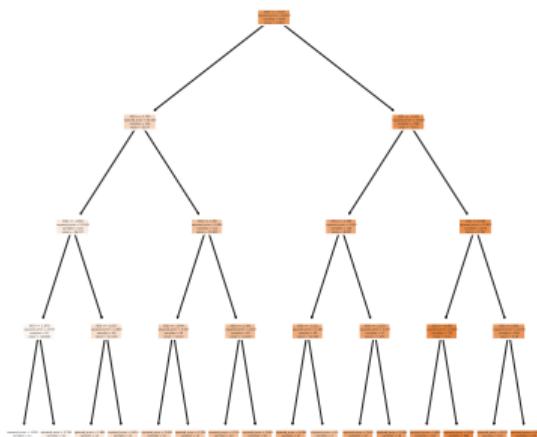


Regularisation for Decision Trees

- Regularisation in DTs comprises tuning hyperparameters that terminate growing the tree
- Common ones include: Maximum depth/number of splits, minimum number of observations in leaves
- Tuning hyperparameters requires K-Fold cross-validation. Any ideas why?
- Another strategy is to grow a large tree, stopping the splitting based on some criterion and then the *prune* the tree
- Pruning involves collapsing internal nodes to find a sub-tree - done by minimising cost-complexity criterion

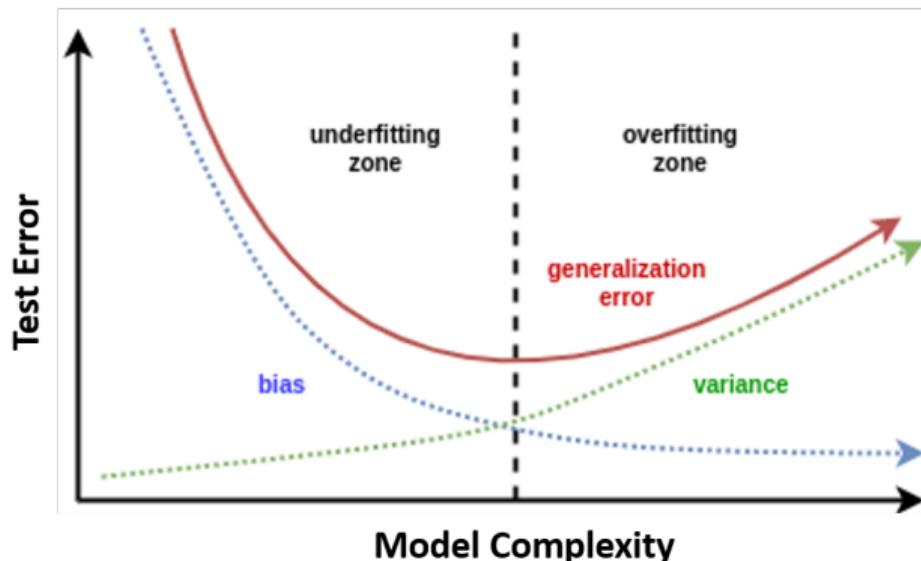
Regularisation for Decision Trees

- Looking at the previous example again, by setting maximum tree depth = 4
- Approximation of regression function looks more reasonable - remember hyperparameter must be chosen through cross-validation



Ensemble Methods in Machine Learning

- Ensemble methods are general to ML, not specific to DTs
- Combination of multiple models to build a *better* model than any individual one
- Two main approaches: Boosting and Bagging
- Why do we need them? Remember bias-variance trade-off:



Ensemble Methods for Decision Trees

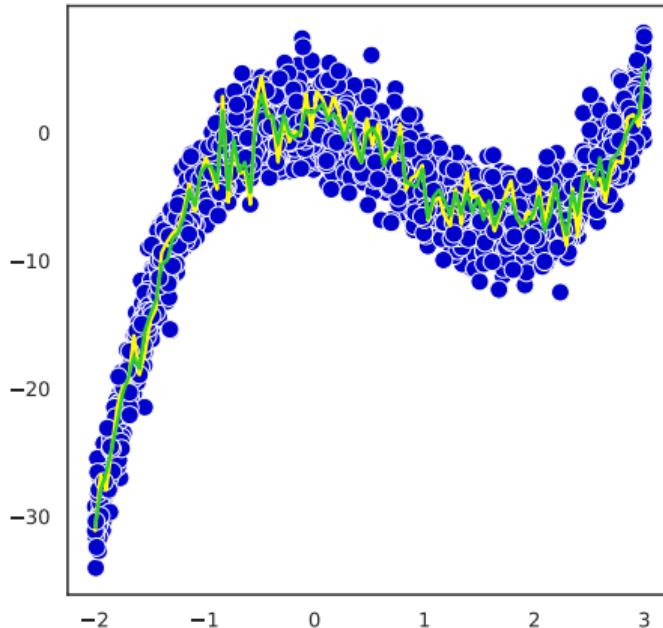
- Boosting: Start with a *weak learner* and iteratively create *new weak models* by repeatedly training it on the training set, but *updating sample weights* each time. Final model given by linear combination of weak models (discussed in more detail in next lecture)
- Boosting tries to *reduce bias*
- Bagging: Stands for 'bootstrap aggregation'
 - Simpler than boosting, fancy way model averaging $D(x) = \frac{1}{N} \sum_i^N D_i(x)$
 - Build a bunch of models and average them to *reduce variance*
 - Models are not built sequentially as with boosting, each D_i is built on a bootstrapped dataset of size $n < N$ in parallel
 - Bootstrapping: sampling randomly with repetitions, to select n samples from N
 - Bagging applied to models with *low bias* and *high variance*, e.g. full grown DTs - referred to as Random Forests

Random Forests

- RFs improve performance relative any individual fully grown DT, as averaging reduces variance, while bias is low for fully grown DTs
- RFs not exactly the same as Bagging, introduce additional randomness to decrease dependence between bootstrapped DTs
- This is done by sampling random subsets of features for defining splits in each DT
- This ensures additional difference between each DT as each would learn to partition distinct subsets of feature space
- RFs designed based on heuristic choices to average models and reduce variance, but turns out it works pretty well!

Random Forests

- RFs used to be very competitive classification models - still in many areas considered best off-the-shelf classifiers
- RFs require very little tuning of hyperparameters (e.g. size of DT, but usually fully grown, subset of features for DT splits)
- Lets look at the previous example again -



COMP5611M: Machine Learning

Ensemble Methods: Random Forests & Boosting

Nishant Ravikumar
Marc de Kamps

School of Computing

April 25, 2022



Lecture Objectives

Objectives for this lecture:

- Recap of ensemble methods
- Bagging for decision trees (Random Forests)
- Boosting for decision trees (AdaBoost, Gradient boosting, XGBoost)

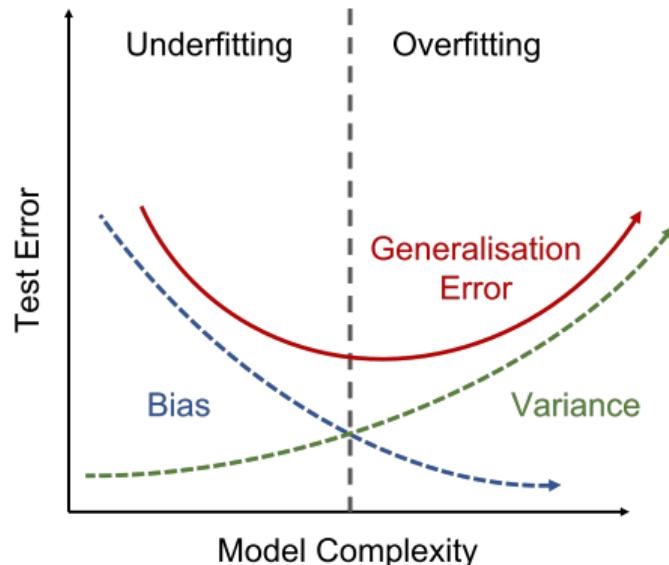
Learning Outcomes

At the end of this lecture, you should be able to

- Describe why ensembling improves performance of DTs
- Describe what Random Forests are and how they are trained
- Describe boosting for DTs, how they are trained and difference to RFs

Ensemble Methods in Machine Learning

- Ensemble methods are general to ML, not specific to DTs
- Combination of multiple models to build a *better* model than any individual one
- Two main approaches: Boosting and Bagging
- Why do we need them? Remember bias-variance trade-off:



Ensemble Methods for Decision Trees

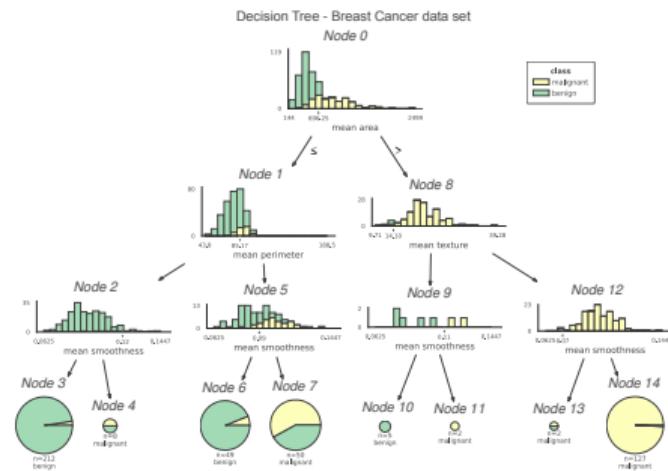
- Boosting: Iterative, sequential training of **weak learners** where each **new weak model** is conditioned on the errors incurred by the preceding weak model
 - Boosting applied to models with **high bias** and **low variance**
 - Final model given by weighted linear combination of weak models
 - Example algorithms of boosting in DTs: AdaBoost, Gradient boosted Trees, XGBoost
 - Boosting tries to **reduce bias**
- Bagging: Stands for 'bootstrap aggregation'
 - Simpler than boosting, uses model averaging $D(x) = \frac{1}{N} \sum_i^N D_i(x)$ to **reduce variance**
 - Classification: Majority vote, Regression: Averaging predictions
 - Models are not built sequentially as with boosting, each D_i is built on a bootstrapped dataset of size $n < N$ in parallel
 - Bootstrapping: sampling randomly with repetitions, to select n unique samples from N
 - Bagging applied to models with **low bias** and **high variance**, e.g. full grown DTs - referred to as Random Forests

Random Forests

- RFs improve performance relative to individual fully grown DT, as averaging reduces variance, while variance is high for fully grown DTs
- RFs not exactly the same as Bagging, introduce additional randomness to decrease dependence between bootstrapped DTs
- This is done by sampling random subsets of features for defining splits in each DT
- This ensures additional difference between each DT as each would learn to partition distinct subsets of feature space
- RFs require tuning of few hyperparameters (e.g. number of DTs, number of features in subsets)
- RFs designed based on heuristic choices to average models and reduce variance, but turns out it works pretty well!

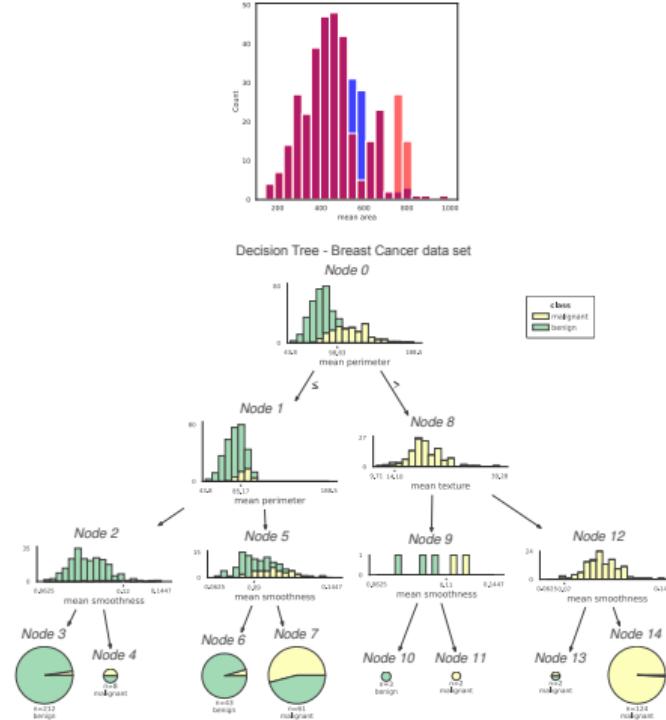
Random Forests

- Lets look at an example where RFs address limitations of DTs
- Consider the breast cancer classification problem:
Benign vs Malignant
- Tumour features: mean radius, mean texture, mean perimeter, mean area, mean smoothness
- DT trained with max depth=3
- We know DTs have *high* variance - i.e. sensitive to small changes in data
- What happens if we **perturb** subset of the data?



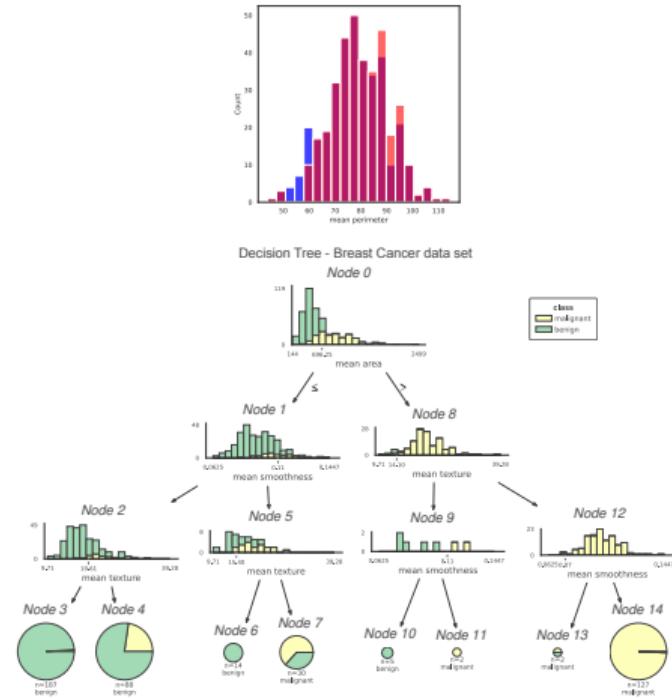
Random Forests

- 455 training & 114 validation samples
- Mean area of 37 samples is perturbed (see histograms, blue: original, red: perturbed)
- DT is fit to perturbed data - shown in Fig.
- What do you notice?
- Partitioning of feature space changes



Random Forests

- Similarly, perturbing mean perimeter of just 21 samples changes feature-partitions again
- Indicates high variance of DTs & tendency to overfit to training data
- So how do RFs address this limitation?
 - By first '*Bootstrapping*' data into subsets
 - Randomly sampling features in bootstrapped subsets to train DTs
 - '*Aggregating*' predictions of all DTs via majority voting (classification)/averaging (regression)
 - In other words through **Bootstrap Aggregation** or **Bagging**



Random Forests

- RF and DT results on val-set on original data & perturbed data:
 - Original data:

Method	F1-score	Precision	Recall
DT	0.86	0.96	0.77
RF	0.92	0.93	0.92

- Mean area changed:

Method	F1-score	Precision	Recall
DT	0.83	0.95	0.74
RF	0.92	0.93	0.92

- Mean Perimeter changed:

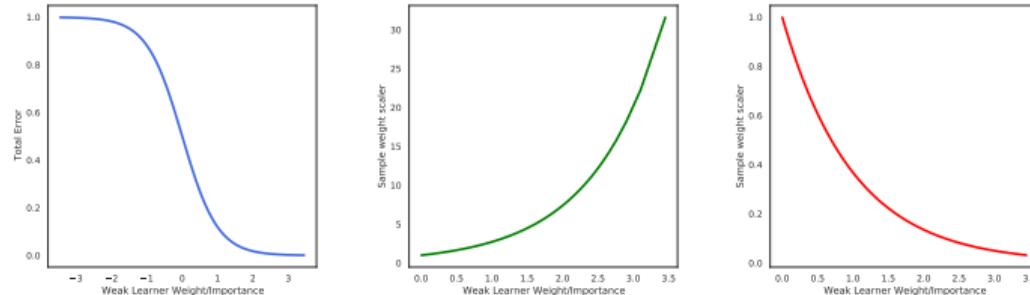
Method	F1-score	Precision	Recall
DT	0.87	0.94	0.81
RF	0.93	0.94	0.92

- RF results remain the **same or show minor changes** with perturbations to features (**low variance**)
- DT results **vary significantly** with feature perturbations (**high variance**)

Boosting for DTs: AdaBoost, Gradient Boost & XGBoost

- Boosting tries to **reduce bias** through sequential training and weighted combination of **weak learners** to form a **strong learner**
- Difference between boosted DTs and RFs:
 - Boosted DTs: Simple/shallow trees, sequential learning, adaptive sample weights, weighted averaging
 - RFs: Full-grown trees, parallel learning, equal sample weights, simple averaging
- AdaBoost: early boosting algorithm for DTs based on adaptively resampling data
 - Trained with 'stumps' - DTs with single node split into leaves
 - Given training data with N samples first assume equal sample weights $\{\nu\}_{n=1\dots N} = \frac{1}{N}$
 - First stump S_1 is trained on the data incurring some total error E_1
 - E_1 represents sum of sample weights of misclassified samples
 - We want to use E_1 to:
 - (a) Determine importance of S_1 (or weight) for final prediction &
 - (b) Update sample weights for training next weak learner S_2

AdaBoost



- Importance of S_1 based on E_1 given by: $\omega_1 = 0.5 * \log(\frac{1-E_1}{E_1})$
- Next re-weight samples using ω_1 & normalise between $[0, 1]$
- Increase the weight of misclassified samples: $\nu_{new} = \nu_{old} \times \exp(\omega_1)$
- Decrease the weight of correctly classified samples: $\nu_{new} = \nu_{old} \times \exp(-\omega_1)$
- Main idea: If S_1 performed well, large increase in weight for misclassified samples and small decrease in weight for correctly classified samples and vice versa

AdaBoost

- AdaBoost tries to continually weight 'difficult' samples higher to train new weak learners $S_{i=1\dots M}$, M denotes maximum number of stumps
- Revised sample weights estimated by each S_i used to create a new dataset of the same size with repetitions
- Sample weights ν_{new}^n determine frequency of sample repetitions (i.e. by under- or oversampling) in modified dataset used to train S_{i+1}
- Process repeated iteratively, number of 'stumps' is a hyperparameter to be tuned
- Classification of a new observation (\mathbf{x}) given by 'importance weighted' vote:

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^M \omega_i S_i(\mathbf{x})\right)$$

this formulation considers labels $\mathbf{y} \in \{-1, 1\}$

- Alternatively, class membership of each sample assigned according to the largest sum of importance (ω_i) based on weak learners' votes for each class

Gradient Boosted Trees

- GBTs are similar to AdaBoost for DTs with the following differences:
- GBTs use **gradient descent optimisation** and a **differentiable loss function** to create new weak learners unlike AdaBoost
- AdaBoost starts with training a single stump on equally weighted samples. GBTs start with a leaf, i.e. initial guess of classes/targets for all samples
- GBTs also train weak learners **sequentially** based on previous model's errors but not restricted to stumps, use **small fixed-size trees**
- So how are GBTs trained? In a nutshell, given observations $\{x_n, y_n\}_{n=1..N}$ try to minimise:

$$\arg \min_{\omega} \mathcal{L}(\mathbf{y}, T(\mathbf{x}, \omega)), \quad T(\mathbf{x}, \omega) = \sum_{m=0}^M \omega_m t_m(\mathbf{x})$$

where, $T(\cdot)$ is a sum of M weak learners, $\mathcal{L}(\cdot)$ is a differentiable loss function (e.g. cross-entropy for classification, MSE for regression)

Gradient Boosted Trees

1. Start with initial prediction $T(\mathbf{x}) = t_0(\mathbf{x})$
2. Compute the pseudo-residuals for each observation:

$$r_{nm} = -\left[\frac{\partial \mathcal{L}(y_n, T(x_n, \omega))}{\partial T(x_n, \omega)} \right]$$

where, r_{nm} is the ‘pseudo-residual’ of the n^{th} sample estimated using the m^{th} model (i.e. the most recent model, initially $m=0$)

3. Fit a new weak learner to the data with the **pseudo-residuals** (r_{nm}) as the new targets: $t_{m+1}(\mathbf{x})$
4. Update the current model (t_m) by weighted addition of the new model (t_{m+1}). Weight given by line search to minimise: (note: λ is the learning rate and can also be fixed to a constant value)

$$\omega_{m+1} = \arg \min_{\lambda} \sum_{n=1}^N \mathcal{L}(y_n, t_m(x_n) + \lambda t_{m+1}(x_n))$$

5. This gives us the new model: $T(\mathbf{x}, \omega) = t_m(\mathbf{x}) + \omega_{m+1}(t_{m+1}(\mathbf{x}))$, which is **nothing but gradient-descent!**
6. Steps 2 to 6 are repeated until convergence

XGBoost

- XGBoost stands for **extreme gradient boosting** - big ML model with lots of parts - routinely wins Kaggle competitions
- Optimised implementation of GBT for large data sets with **regularisation (prevents overfitting)** - i.e. balances complexity of trees created with accuracy of the model
- Tree complexity is typically measured as a combination of number of leaves and magnitude of leaf scores
- **Sparsity-aware** algorithm - i.e. handles missing values and lots of zero-valued entries
- Uses **weighted approximate quantile sketch** - i.e. calculate quantiles using only data seen allowing XGBoost to calculate splits without looking at all of the data available
- Uses block compression of data and parallelisation to enable efficient training/inference on large data sets
- Several other customisable benefits that makes it a flexible, multi-purpose ML algorithm

Summary

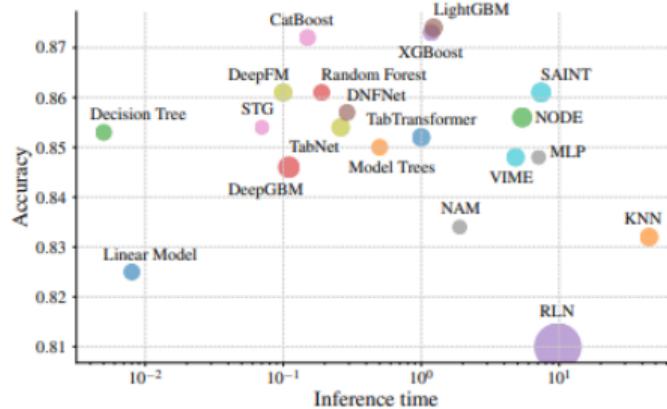


Figure: Benchmarking study of ML/DL for tabular data: Borisov, Vadim, et al. "Deep neural networks and tabular data: A survey." arXiv preprint arXiv:2110.01889 (2021).

- While deep neural nets have replaced traditional ML approaches in computer vision problems, tree-based ensemble approaches remain competitive for tabular data
- New research streams attempting to combine ideas/advantages from DL and tree-based ensembling are emerging - e.g. deep adaptive neural trees
- Hence, knowledge of tree-based ensembling methods remain relevant and critical

COMP5611M: Machine Learning

Mixture Models and the E/M Algorithm

Nishant Ravikumar
Marc de Kamps

School of Computing

May 3, 2022

Learning Objectives

- Clustering
- K-means algorithm and its deficiencies
- Structure of Gaussian Mixture Models (GMMs)
- Maximum Likelihood Estimates of GMMs

Learning Outcomes

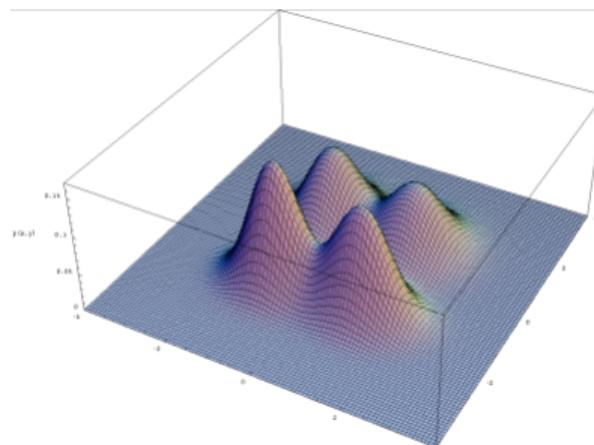
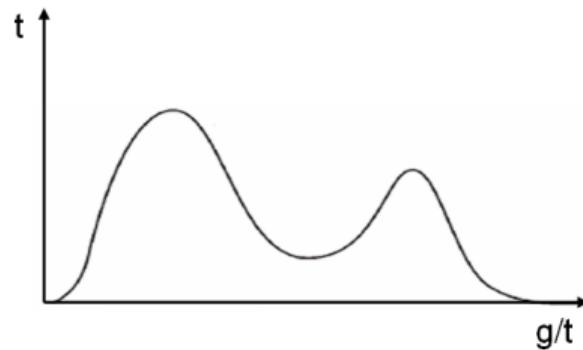
At the end of this lecture you should be able to:

- Implement GMMs
- Explain the difficulties in obtaining MLEs for GMMs

Multimodal Distributions

Many distributions are **multimodal**

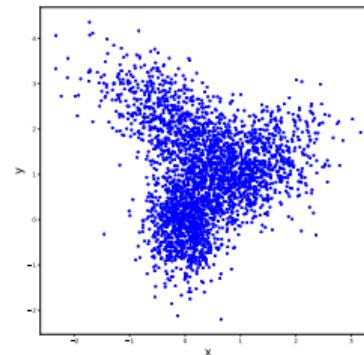
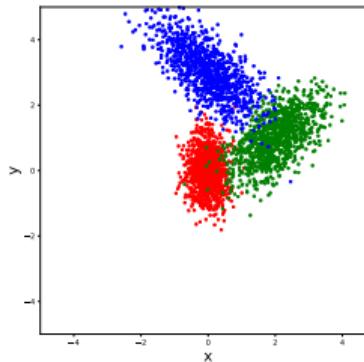
- Can e.g. happen if more than one process contributes to the dataset
- Gaussians are single modal
- Can we model multimodal distributions as being composed of several Gaussians
- **Mixture models!**
- Not necessarily Gaussians, mixtures of Bernoulli etc.



Inference of Mixture Models

- Inference of mixture models is much harder!
- Consider colour labelled points:
- We can infer the three clusters separately
- This is not harder than earlier stuff
- When the colour labels are not available, we are in trouble immediately!
 - How many clusters are there?
 - Which point belongs to what cluster?
 - What are the properties of the individual clusters?

We introduced a new concept: clustering!

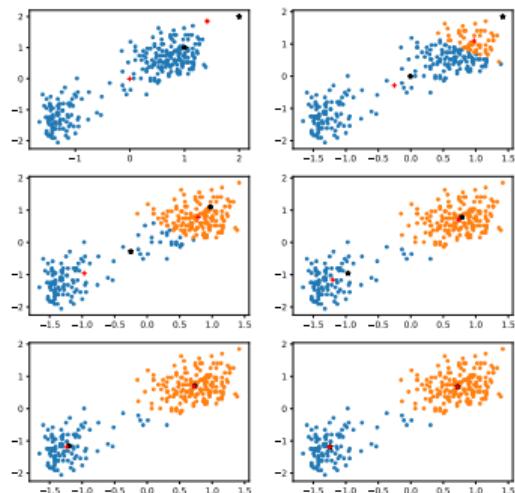


Clustering

Clustering

The process of clustering serves to assign a data point to some partitioning of the dataset.

- It often involves the notion of a **metric**: points that are close in some sense are clustered
- Sometimes this is too naive: K-means vs mixture models
- Clustering closely related to classification
- Famous clustering algorithms: **K-means**, **K-nearest neighbour**



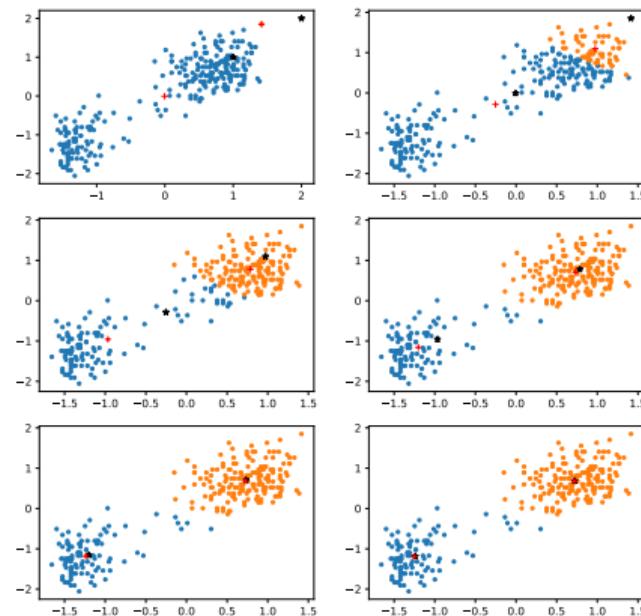
Example of K-means clustering progression

K-means clustering

- K-means is easy to understand
- Later we will recognise it as a special case of Expectation/Maximisation (E/M) algorithms
- N data points, K clusters
- r_{ik} an $N \times K$ matrix.
- Each row i is a one-over- K encoding for data point i , indicating which cluster k point i belongs to

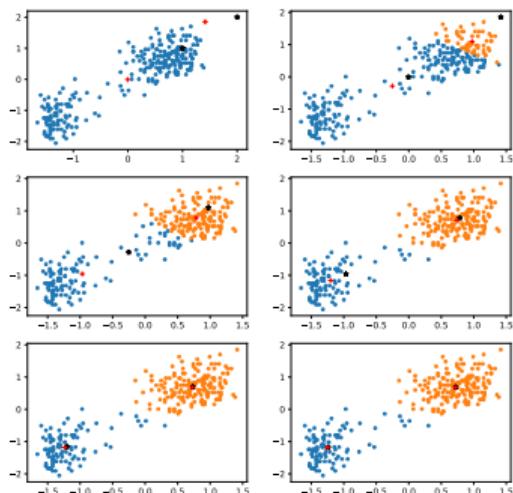
The K-means algorithm aims to minimise the *distortion measure*:

$$J = \sum_{i=1}^N \sum_{k=1}^K r_{ik} \| \mathbf{x}_i - \boldsymbol{\mu}_k \|^2$$



K-means clustering

- We need to optimise both the cluster centres $\mu_k \dots$
- ... and decide which point each cluster belongs to :
 r_{ik}
- We can't do that simultaneously
- Common sense:
 1. Pick random μ_k
 2. Assign each data point to the closest cluster
 3. Calculate centre-of-gravity of the resulting clusters to estimate new μ_k
- Hence *K*-means
- Easy to program (**Try!**), easy to use (**scikit-learn**)



- Black marker start estimate of cluster centre
- Red marker end estimate of cluster

K-means

- Simple as K-means is, you can use it for **vector quantisation**
- Create clusters in RGB space
- Instead of labelling pixels by colour, label them by cluster index
- Notice that we perform two steps:
 1. Associate points with clusters (**E-step**)
 2. Update cluster parameters (**M-step**)
- We will later show that K-means is an example of **E/M optimisation**
- In the process, we will prove convergence

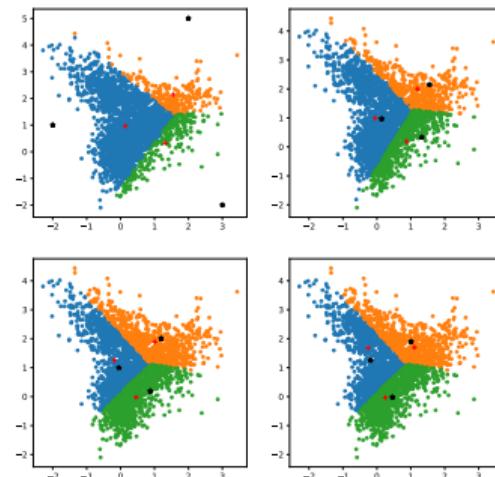


Figure 9.3 Two examples of the application of the K -means clustering algorithm to image segmentation showing the initial images together with their K -means segmentations obtained using various values of K . This also illustrates the use of vector quantization for data compression, in which smaller values of K give higher compression at the expense of poorer image quality.

K-means

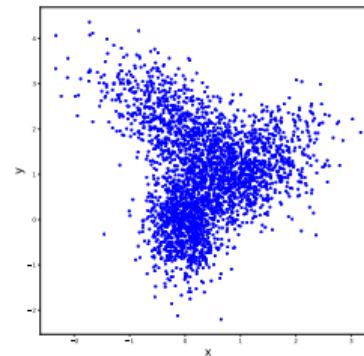
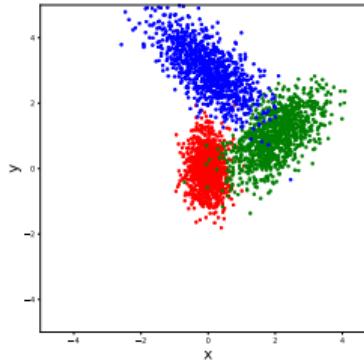
Drawbacks

- K-means works well on isolated clusters
- On overlapping clusters, the reliance on distance makes it inflexible
- The necessity to use distance as a criterion makes overlapping clusters impossible
- In a generative view of data generation this is not acceptable
- Data points generated by different processes **can** share the same data space



Mixtures of Gaussians

- Mixtures of Gaussians can share the same data space
- Used to model realistic data generation processes
 - There really may be different physical/causal processes involved in generating a dataset
- Used as a model for complex multimodal distributions

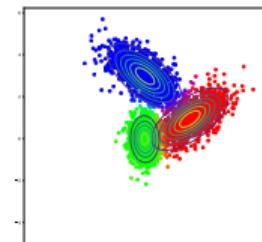
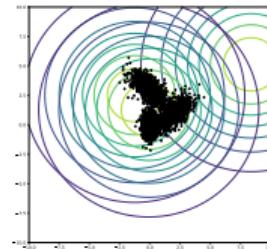


Mixtures of Gaussians

Definition of mixture of Gaussians (Gaussian Mixture Models; GMMs)

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- Easiest understood as two-step process
 1. Randomly pick one of the Gaussians
 2. Generate a data point using this Gaussian distribution
 3. Repeat
- In the K-means algorithm, we just needed the cluster centres: represented by a single point
- In GMMs, we represent the Gaussians by elliptic contours, visualising both centres and covariances



Mixtures of Gaussians

Step 1: Pick a Gaussian

What is π_k ?

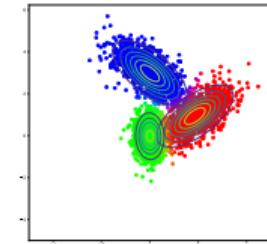
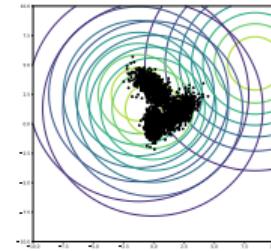
$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

We model the selection of a cluster by generating a one-over- K representation, e.g.

$$\mathbf{z} = (0, 1, 0)^T$$

represents the selection of the second Gaussian, i.e. for the actual generation of the data point we use

$$\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$$



Mixtures of Gaussians

Step 1: Pick a Gaussian

What is π_k ? In general there are three possible vectors \vec{z} :

$$\mathbf{z}_1 = (1, 0, 0)^T$$

$$\mathbf{z}_2 = (0, 1, 0)^T$$

$$\mathbf{z}_3 = (0, 0, 1)^T$$

The three probabilities π_1, π_2, π_3 are simply the probabilities for picking $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3$, so

$$\pi_k = p(z_k = 1)$$

Mixtures of Gaussians

Step 1: Pick a Gaussian

What is π_k ?

The vectors z_k are called **latent variables**, as they contribute to the data generation process, but are never observed directly. We write:

$$p(z) = \prod_{k=1}^K \pi_k^{z_k}$$

E.g.:

$$p((0, 1, 0)^T) = \pi_1^0 \pi_2^1 \pi_3^0 = \pi_2$$

Conditional probabilities for a *given* vector z_k are Gaussian:

$$p(x | z_k = 1) = \mathcal{N}(x | \mu_k, \Sigma_k)$$

or

$$p(x | z) = \prod_{k=1}^K \mathcal{N}(x | \mu_k, \Sigma_k)^{z_k}$$

Mixtures of Gaussians

Joint and Marginal Distribution

The joint probability distribution function can then be written as:

$$P(\mathbf{x}, \mathbf{z}) = \prod_{k=1}^K \pi_k^{z_k} \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

you should validate that the marginal distribution over \mathbf{x} is:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x} | \mathbf{z}) = \prod_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

It may seem odd now to define a joint probability over variable you can't observe, but you need to get familiar with this notation as the latent variables deserve full consideration as stochastic variables!

Mixtures of Gaussians

Inference

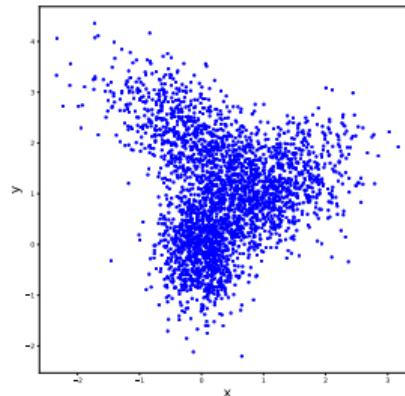
- How to perform inference?
- Given the data, we can write down the likelihood for the joint distribution

$$p(X, Z | \pi, \mu, \sigma)$$

where

$$X, Z = \{x_i, z_i\}_{i=1}^N$$

- ... but we don't know z_i
- so we can't do maximum likelihood estimation!



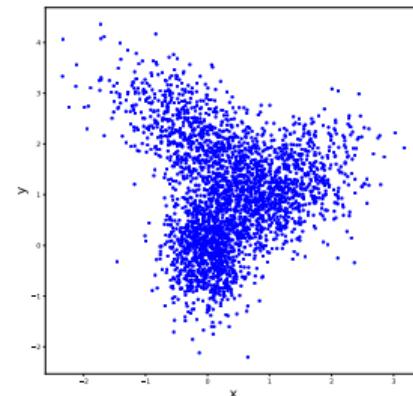
Mixtures of Gaussians

Inference

- Well, why not use the marginal log likelihood?
- As this only involves observable variables:

$$\ln p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{i=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

- This again, like for K-means, requires a two-step process
- Since we don't know the mixing coefficients
- In one step we assume the $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ known (E-step) and estimate π_k
- In the next step (M-step), we assume the π_k known and estimate the $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$



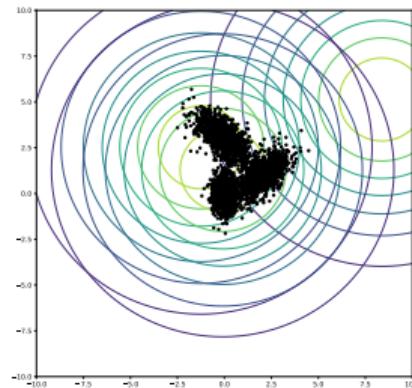
Inference

E-step

- We start by making guesses for all parameters
- Usually starting with isotropic Gaussians
- We then estimate the posterior values of the latent variables:

$$\gamma(z_k) \equiv p(z_k = 1 | \mathbf{x})$$

- These values are called **responsibilities**
- High values indicate high probability of data point belonging to a certain cluster
- E.g. $\gamma(z_2) = 0.9$ means a 90 % probability that the data point belongs to cluster 2
- With only a 10 % probability for other clusters



Inference

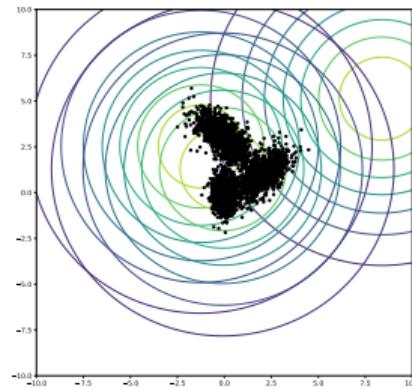
E-step

- How can we estimate responsibilities when we can't observe z_k directly?
- Remember that we wrote down the probability for a data point conditional on the latent variable?

$$p(\mathbf{x} \mid z_k = 1) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- We can use Bayes' law:

$$\begin{aligned}\gamma(z_k) = p(z_k = 1 \mid \mathbf{x}) &= \frac{p(z_k = 1)p(\mathbf{x} \mid z_k = 1)}{\sum_{j=1}^K p(z_j = 1)p(\mathbf{x} \mid z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}\end{aligned}$$



Inference

M-step

- The calculation of the responsibilities is the first step in the algorithm (**E-step**)
- The second step is the maximum likelihood estimation of the marginal log likelihood

$$\ln p(\mathbf{X} \mid \boldsymbol{\mu}, \Sigma, \boldsymbol{\pi}) = \sum_{i=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \Sigma_k) \right\}$$

- Differentiation with respect to $\boldsymbol{\mu}_k$ and setting to 0 gives:

$$-\sum_{i=1}^N \frac{\pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \Sigma_k)}{\pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \Sigma_j)} \Sigma_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) = 0$$

Note that the first factor is given by the responsibilities. It turns out that the estimates for $\boldsymbol{\mu}_k$, Σ_k and π_k can all be neatly expressed in terms of the responsibilities.

1. Determine K , the number of clusters.
2. Initialise the means μ_k and covariances Σ_k and mixing coefficients π_k .
3. **E-step.** Evaluate the responsibilities:

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

4. **M-step.** Re-estimate the parameters using the responsibilities:

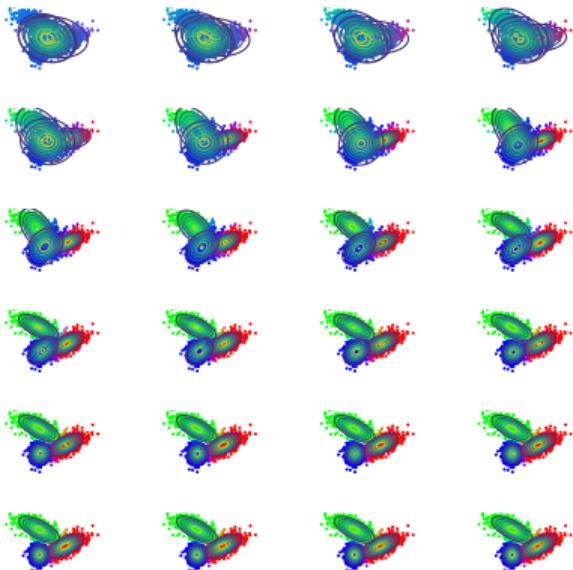
$$\begin{aligned}\boldsymbol{\mu}_k^{new} &= \frac{1}{N_k} \sum_{i=1}^N \gamma(z_{ik}) \mathbf{x}_i \\ \boldsymbol{\Sigma}_k^{new} &= \frac{1}{N_k} \sum_{i=1}^N \gamma(z_{ik}) (\mathbf{x}_i - \boldsymbol{\mu}_k^{new})^T (\mathbf{x}_i - \boldsymbol{\mu}_k^{new}) \\ \pi_k &= \frac{N_k}{N}, N_k = \sum_{i=1}^N \gamma(z_{ik})\end{aligned}$$

Solving GMMs with E/M Algorithm

- Initialise π, μ, Σ with random values or start a simpler clustering algorithm
- Cycle through consecutive E/M steps
- Evaluate the log likelihood at every step:

$$\ln p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{i=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

- Stop when converged



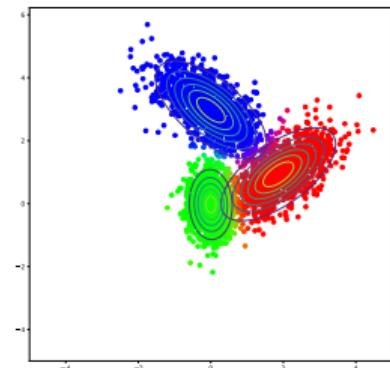
Observation

- Estimators are simple modifications of the standard Gaussians ones:

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{i=1}^N \gamma(z_{ik}) \mathbf{x}_i$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{i=1}^N \gamma(z_{ik}) (\mathbf{x}_i - \boldsymbol{\mu}_k^{new})^T (\mathbf{x}_i - \boldsymbol{\mu}_k^{new})$$

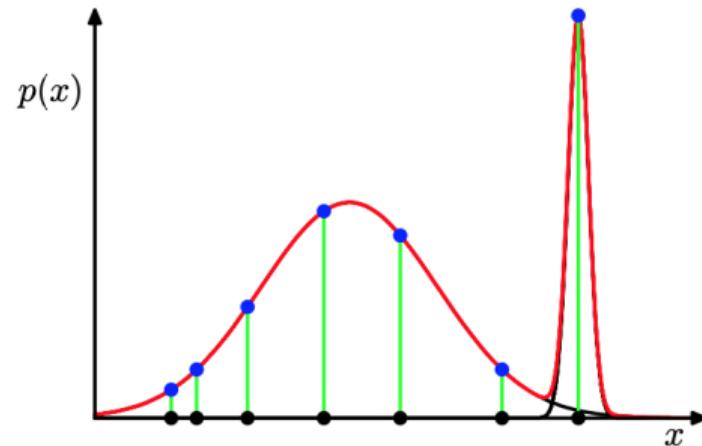
$$\pi_k = \frac{N_k}{N}, N_k = \sum_{i=1}^N \gamma(z_{ik})$$



- N_k effective number of points in each cluster
- Point can have significant responsibilities for more than one cluster

Stability

- If one of the data points coincides with μ_k the algorithm is unstable
- It can increase the log likelihood by reducing the Σ_k
- This is not possible in 1D (why not?)
- Reinitialise when this happens



- The derivation of the algorithm is unpleasant (see reader for details)
- Reason:

$$\ln \mathcal{L} = \ln(\pi_1 e^{\cdots} + \pi_2 e^{\cdots} + \cdots)$$

- In mixture models the log no longer eats the exponent
- In this form it also is not clear that it converges
- After all, cannot undo the E-step the progress of the previous M-step?
- Answer is no, but that is not clear from the current derivation
- Elephant in the room: How do we pick K ?

Next Lecture

- We will look at the same algorithm ...
- ... from a different perspective
- We will look at $\mathbb{E}_{P(Z)}[P(X, Z)]$
- Does this notation makes sense to you?
- Introduction of the Evidence Lower Bound (ELBO)
- A window on modern variational inference techniques ...
- Briefly comment on Bayesian version