Intelligent Algorithms

Comp3032

1 Overview

a) Topics

30% coursework

70% exam

Steve Gunn and Srianda Damaphra

Linear Algebra = important !

Foundations :

- Linear algebra
- recommended to use MatLab

Algorithms

Applications

- Image processing
- Text analysis
- Speech recognision
- bioinformatics
- control

b) Algorithms

Dynamic Programming

Probalistic Models

Linear Programming

Monte Carlo Simulation

Hidden Markov Model

Eigensystem

Multidimensional Optmization

c) Tools

Represent data as vectors

Perform transformations via matrices

d) Probalistic Models and Algorithms

Develop probalistic models of data

Adopt models automatically to data (optmization)

Sample efficently from models (e.g. Monte Carlo simulation)

e) Intelligent Algorithm

Intelligent in computer science

"Having certain data storage and processing capabilities"

Algorithm

"Step-by-step problem solving procedure for solving a problem in a finite number of steps"

2 MATLAB

a) Why MATLAB

Not strictly needed but highly reecommended

Tool for development

GNU Octave opensource version which is largely compliant

b) Variables

V = val

vectors/matrics [v; v1 ; v2] is a 3 element vector

; represents end of the row

V' = V^T or V transposed

z = rand(10, 1) creates a vector of 10 random elements

c) Commands

By default echo their output to terminal

Supress with ; i.e. Z = rand(1000, 1);

lots of built-in commands and functions

lookfor random will search for functions labelled random in DB

whos – lists variables

clear – delete variable

% - comment

3 Linear Algebra

a) Introduction

Resources on course website

Examples in the context of face recognition

Represents objects of the world in a mathematical framework allowing you to perform operations on it

Ideally want to be able to transform from the real world to model and back to a realistic representation of the world

b) Representing data

Data $X=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots\ldots \boldsymbol{x}_n\}, \in R^D$ a set of *D* dimensional vectors

For example an LxL image can be represented a L^2 vector

Goal is to get right of irrelevant and redundant data to make processing faster, look at variantions

c) Reducing data

Want to reduce data in a smaller vector (containing the most significant features) via some function

$W=\{\boldsymbol{w}_{1,}\boldsymbol{w}_{2,}\ldots\ldots, \boldsymbol{w}_n\}, \in R^d \wedge d<D$

The transformation will not be fixed, instead it will adapt to the training data

d) Vectors

A set of vectors are *linearly independent* if none of them can be represented as a combination of the others

{(1, 0, 0), (0, 1, 0), (0, 0, 1)} are linearly independent

{(1, 1 , 1), (1, 0, 0), (0, 1, 1)} are not linearly independent as (1, 0, 0) + (0, 1, 1) = (1, 1, 1)

Two Vectors are orthogonal if their dot product is 0. Diagrammatically they are at 90 degrees to each other

$V_1 \cdot V_2 = 0$

if *n* vectors are mutually orthogonal they form an orthogonal set

$O=\{\boldsymbol{v}_n\}, \forall i, j \in \mathbb{N}\, V_i \cdot V\, j=0$

If all vectors in this set have unit length 1 (i.e. $|V_i|=0$) then the set is an orthonormal set

For a vector *V* and orthonormal set $O=\{v_{1,} v_{2,} \ldots., v_n\}$ want to find numbers $\{a_{1,} \ldots., a_n\}$ such that

$$\boldsymbol{v}=\sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i$$

Example:

{(1, 0, 0), (0, 1, 0), (0, 0, 1)} is orthonormal set, V = (3, 4 , 4)

Then $\alpha_1=3, \alpha_2=4, \alpha_3=4$

Generally

$$\alpha_j=\frac{v \cdot v_j}{|v_j|}$$

If the vectors in the orthonormal set can be used to define a coordinate system then it is *orthonormal basis*

e) Covariance Matrix

Covariance describes how two variables change together

Covariance Matrix, each cell (i, j) shows how dimensions i and j change together

When i==j this is normal variance

f) Eigenvector and Eigenvalue

A is an *n x n* matrix

X is an *n x 1* vector – Eigen vector

lambda is a scalar value – Eigenvalue

Find values such that...

$$\boldsymbol{A}\boldsymbol{x}=\lambda\boldsymbol{x}$$

$(\boldsymbol{A}-\lambda\boldsymbol{I})\boldsymbol{x}=0$ where I is the identiy matrix

For this to be true then

$det(\boldsymbol{A}-\lambda\boldsymbol{I})=0$ which gives an equation in terms of lambda

Example

$$\boldsymbol{A}=\begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}$$

$$\boldsymbol{A}-\lambda\boldsymbol{I}=\begin{pmatrix}2-\lambda & 1\\1 & 2-\lambda\end{pmatrix}$$

$$det(\boldsymbol{A}-\lambda\boldsymbol{I})=(2-\lambda)^2-1=0$$

Solutions are lambda = 1 or 3

These become the *eigen values*

Inserting eigen values into the original equation we get simultaneous equations in terms of a D dimensioned vector for reach eigen value. This comes the eigenvector for this eigen value.

Example

$$\begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}=\lambda\begin{pmatrix}x\\y\end{pmatrix}$$

$$\begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}=3\begin{pmatrix}x\\y\end{pmatrix} \quad \begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}=1\begin{pmatrix}x\\y\end{pmatrix}$$

Solve for x and y (or each dimension in D) to give the eigen vector for each value. See notes on matrix maths at end.

4 PCA

N datapoints already $x_{1,}x_{2,.}..,x_N$ *where* $\forall x_i\in\mathbb{R}^D$

1. Center Data by subtracting the mean

$$y_i=x_i-MEAN(X)$$

$MEAN(X)=\frac{1}{N}\sum_{k=1}^{N} x_k$ so mean is expresses as a vector of the mean values in each dimension

2. Construct the covariance matrix

-- Method for smaller dimensions

$$S=\frac{1}{N-1}\times \boldsymbol{Y}\boldsymbol{Y}^{T}$$

-- Where D >> N i.e. More dimensions than samples use a different method

$$A=\boldsymbol{Y}^{T}\boldsymbol{Y}$$

3. Construct eigen values and eigen vectors in the normal way, eigen vectors can be expressed in a matrix where each column is a vector. Sort in order of eigen values.

4. If D >> N and eigen vectors are held in a matrix V then $\boldsymbol{V}=\boldsymbol{V}\,\boldsymbol{A}$ to get useful vectors. V is now DxN matrix

5. Normalize eigenvectors (change them to their unit vector)

6. Construct projection matrix *w* values $\boldsymbol{P}=\boldsymbol{V}^{T}\boldsymbol{Y}$

P is NxN matrix

7. Reconstruct data

$$New=\boldsymbol{Mean}+\sum_{i=1}^{d}\boldsymbol{V}_{col\,i}\boldsymbol{P}_{row\,i}$$

5 Optmization

a) Idea

Given a function find the mimizer or minimum

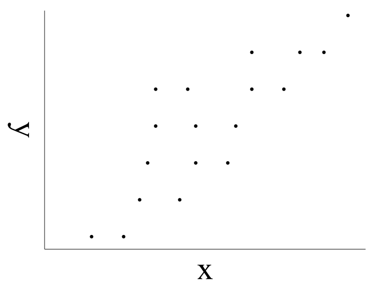
$f:\mathbb{R}^n\rightarrow\mathbb{R}$ $S\subseteq\mathbb{R}^n$ find $x.\forall y\in S f(x)\leq f(y)$

Since the maximum of function F is the minumium of -F it is sufficent only to consider minimization

b) Example Application

Finding the equations of lines around datapoints

Datapoints:



Want to find a function to describe this data

Assume a linear type function in the form $y=w_1 x+w_0$

For a correct function the distance between this function and any existing datapoint in terms of y will be minimized. This can be expresssed in an objective function

$\Phi(\underline{w})=\sum y_i-(w_1 x_i+w_0)^2$ (distance squared)

This minimal of this objective function is gives the desired values for w1 and w0

**note this exact problem can be reduced to a quadratic problem and is fairly simple to solve*

c) Linear/Non linear

General continous problem can be expressed as...

min f(x) subject to g(x) = 0 and h(x) <= 0

$f:\mathbb{R}^n \rightarrow R \quad g:\mathbb{R}^n \rightarrow \mathbb{R}^m \quad h:\mathbb{R}^n \rightarrow \mathbb{R}^k$

if f, g and h are all linear then we have a linear programming problem

otherwise its a non linear problem

d) Global Minimum and Local Minimum

Global minimum point x* if f(x*) <= f(x) for all feasible points x

Local minimum point x* if f(x*) <= f(x) for all feasible points x 'near' x*

Finding the global minimum can be very hard!

Often have to choose lots of different points and find their local minimum

e) Computing

At all minimum points in one dimension the gradient of the line is 0

This can be expressed in multiple dimensions like so:

$x_{min}.\forall x \in S \Rightarrow f(x_{min}) \leq f(x)$

x_{min} *is solution for* $\nabla f(\underline{x})=\underline{0}$ *zero vector*

$\nabla f(\underline{x})$ *gives the gradient as a vector for function* $f:\mathbb{R}^n \rightarrow \mathbb{R}$

$\underline{x}=(x_{1,},\dots,x_n) \quad \nabla f(\underline{x})=(\frac{\delta f}{\delta x_1},\dots,\frac{\delta f}{\delta x_n})$

When $\nabla f(\underline{x})=\underline{0}$ *then* $\underline{x}$ *is a stationary point of* f

To descide if a stationary point is a minimum, maximum or point of inflection look at the second derivative

A Hessian Matrix can be used to express this information

$Hf(x)_{i,j}=\frac{\delta^2 f}{\delta x_i \delta x_j}$ i.e for row i and column j of the matrix the second derivative of f showing the change in the change of f for xi and xj divided by the change in xi multiplied by the change in xj at the point X

6 Clustering

a) Introduction

Given a set of input data, find natural groupings (cluster's) within the data

Data can be represented as Vectors (where nearness is distance) or nodes with weights in a graph

b) k-Means

Given data in the form of vectors

$\{\boldsymbol{x}_{1,} \dots, \boldsymbol{x}_n\} \; \boldsymbol{x}_i \in \mathbb{R}^D$

Assign one of *k* labels to each data point

k is given

Data point d is a member of cluster/label j if distance(prototypej, d) is the minimal for all prototypes

μ_i *prototype i*

x_m *data point m*

$\boldsymbol{Y}_{j,m} \mathit{equals}\; 1\, \mathit{if}\; \forall i \in \{1 \dots k\}\, \mathit{len}(x_m - \mu_j) \leq \mathit{len}(x_m - \mu_i)$ *(membership of cluster)*

Aim is to find prototypes (example vectors/positions) for each label such that the distance to each data point is minimized

$E(\{\mu_j\}) = \sum_{m=1}^{N} \sum_{j=1}^{k} \mathit{len}(x_m - \mu_j) \times \boldsymbol{Y}_{j,m}$ *i.e. for every point minimize the distance to its cluster*

Algorithm:

1. Choose initial prototypes at random
2. Compute ***Y***
3. Set prototypes to be the means of their cluster i.e. for cluster 1 set μ_1 to be..

$$\mu_i = \frac{\sum_{m=1}^{N} \boldsymbol{Y}_{i,m} x_m}{\sum_{m=1}^{N} \boldsymbol{Y}_{i,m}}$$

4. Repeat until convergence

c) k-Means clustering (2)

Concept:

– Iteratively change the cluster a datapoint is in to the nearest mean-value of that cluster

1. Randomly assign each datapoint to a cluster
2. For each cluster calculate a mean point for it by averaging over all the member datapoints
3. Assign each node to the cluster with the nearest (euclidean distance) cluster
4. Repeat 2&3 until convergence

d) Spectural Clustering

Represent data in a graph with Vertex's (nodes) and weighted edges

Idea is to partition graph into sub sets of interconnecting nodes with strongly weighted edges and weak connections between groups

$Graph=(V,E)$ vertex's and edges

Degree $d_i=\sum_j w_{i,j}$ i.e. sum of all the weights to edges a node is connected to

For two subsets A, B the degree to which they are connected can be described as the sum of the weights of all connecting nodes in A and B

$$W(A,B)=\sum_{v_i\in A,v_j\in B} w_{i,j}$$

Metrics for the subset's include the number of nodes in the set, and the sum of the degree's of the set

Weights and degrees can be placed into matrix's

$\boldsymbol{W}_{\boldsymbol{i},\boldsymbol{j}}=w_{i,j}$ $\quad \boldsymbol{D}_{\boldsymbol{i},\boldsymbol{j}}=d_i\, if\, i\, equals\, j\, else\, 0$

Laplacian L computed as $\boldsymbol{L}=\boldsymbol{D}-\boldsymbol{W}$

Take the first k eigenvectors of $\boldsymbol{L}$ and place into a matrix **U**

$U\in\mathbb{R}^{N\times k}$

Take rows from this matrix

$z_i\in\mathbb{R}^{1\text{xk}},i=1\ldots N,$

Cluster them using a vector-based method (e.g. k-Means) into k clusters $C_1\ldots.C_k$

Output clusters $A_i\ldots.A_k$ which are sets of vertex's such that

$A_i=\{v_j\cdot z_j\in C_i\}$

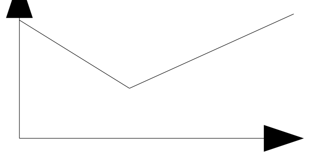
7 Optmisation part 2

a) Unimodality

Property that makes one-dimensional optimisation easier

For an interval of *x* over the function *f*, *f(x*)* is the minimum value and *f* strictly increases for greater values of x and decreases for lower values

Example:



b) Equidistant Sampling

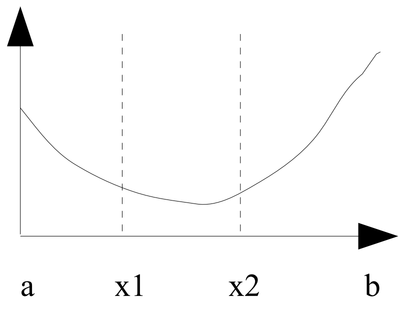
Divide x into equal parts and find *f(x)* for those divisions

c) Golden Section Search

Uses the golden ratio (~0.618) to divide the search space in two places

By comparing the value of *f* at these places it is possible to discard some of the search space

Golden ratio: $T = \frac{(\sqrt{5}-1)}{2}$ x1:x2 is 1-*T* :*T*



On each iteration, make x1 the new a if *f(x1) > f(x2)* else b → x2 and compute one new division

Has a convergence rate of ~0.618 (constant)

d) Convergence

sequence, this concept is of practical importance if we deal with a sequence of successive approximations for an iterative method, as then typically fewer iterations are needed to yield a useful approximation if the rate of convergence is higher. This may even make the difference between needing ten or a million iterations.

The higher the convergence rate, the quicker it is to find a solution

Linear Definition:

- sequence $\{xk\}$ converges to the number L
- Converges linearly if
- $\exists \mu \in (0,1) \cdot \lim_{k \to \infty} \frac{|x_{k+1} - L|}{|x_k - L|} = \mu$

8 Successive parabolic interpolation

a) What is it?

Technique for finding minimums by successively fitting parabola's (polynomials with order 2) between 3 points and replacing the oldest point with the minimum of the current function

Does not require the computation or approximation of function derivatives!

b) Disadvantages

Breaks down when 3 points chosen are co-linear i.e. are on the same line in which case no parabola can be fitted again because the functions has degenerated into a lower order

c) Method

1. Choose 3 random x values in the range of the search space and find their values

   $f(x_1)=f_1, f(x_2)=f_2, f(x_3)=f_3$

2. Can use these values to form a matrix to describe the parabola which can then be solved

   $g(x)=a^2+bx+c$

   $$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix}=\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

3. Only one minima of this function and the x value of this used to update the oldest of the 3 set values and then repeat 2 and 3 until convergence

   $g'(x_{min})=2\mathrm{ax}_{min}+b=0$

   $$x_{min}=\frac{-b}{2\mathrm{a}}$$

d) Convergence

$O(1.324)$ Making it superlinear and the best of the linear convergence methods

However if derivatives can be found then newton's method is far better

9 Newton's Method

a) Origins

Taylor Series expansion for finding another value of a function

i.e. know *f(x)* want to know *f(x+h)*

$$f(x+h)\approx f(x)+f'(x)h+\frac{f''(x)h^2}{2}+.....+\frac{f^{(n)}(x)h^n}{n!}$$

When truncated to n=2, the value of *h* which minimizes this function can be found by:

$$0=h\,f''(x)+f'(x)$$

or

$$h=\frac{-f'(x)}{f''(x)}$$

Although this is only an approximation! So one has to repeat this.

b) For...

Solving non-linear equation *f* for $f'(x)=0$

Iterative method (and approximate) so requires repeating

$$x_{k+1}=x_k-\frac{f'(x_k)}{f''(x_k)}$$

Effectively:

new guess is old guess + approximate value of h to minimize the function

c) Convergence

$O(n^2)$

d) Disadvantages

Requires a first guess to be close to a stationary point or the method could require many iterations

Requires derivatives

Stationary points have to be tested for minima or maxima

10 Algorithms and Probabilities

a) Intro

- Generate pseudo random data which can have output characteristics similar to what has been observed
- Stochastic algorithms use random numbers to simulate noisy systems
- Probabilistic (randomised) algorithms are used for deterministic problems

b) Different probabilities

Discrete Probability:

- Set number of out come each with a unique probability
- i.e. rolling a dice gives a choice of 6 and each has a probability of 1/6 of occuring

Continuous:

- Follows a density function *f(x)*
- $\int_{-\infty}^{\infty} f(x)\,dx = 1$
- $P(a \le x \le b) = \int_a^b f(x)\,dx$ i.e. the probability that x is between a and b

Cumulative

- $f(X) = P(x \le X)$

c) Uniform Distribution

Equal chance for every outcome

Discrete:

- $n = b - a + 1$
- $P(X = x) = 1/n \wedge a \le x \le b$

Continuous:

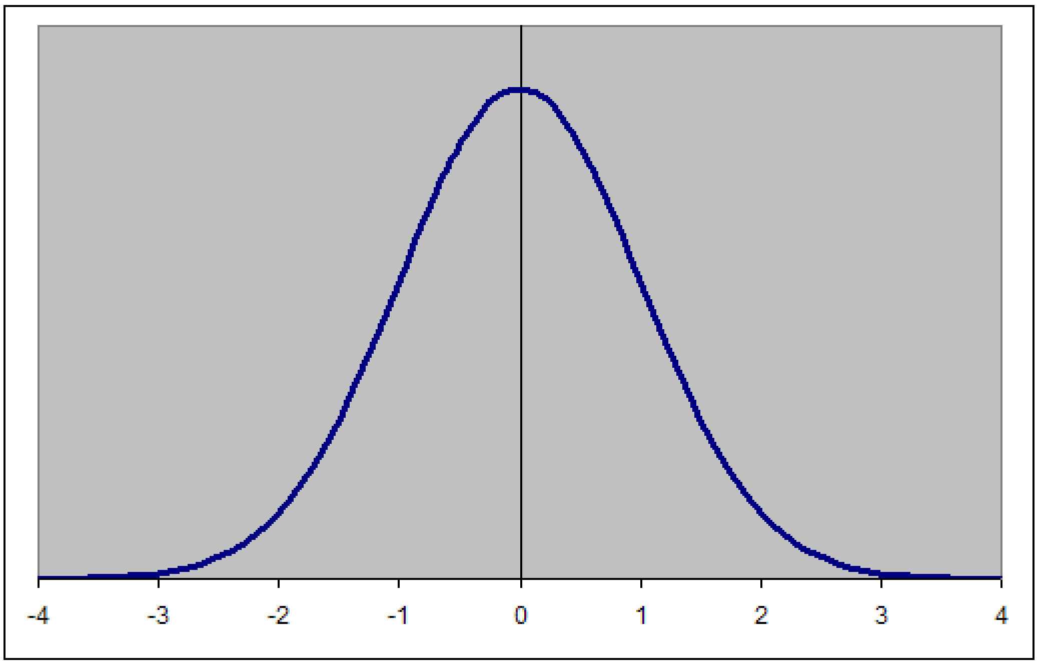
- $x < a \vee x > b \rightarrow P(X = x) = 0$

- $a \leq x \leq b \rightarrow P(X=x) = 1/b-a$

d) Normal Distribution

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

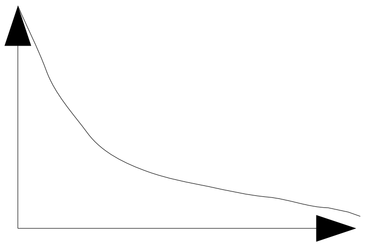
expectation or mean = 1



e) Transformation Method

- Desired density function $f(x)$ and $f^{-1}(x)$ can be found
- Uniform random variable can be generated y
- $f(x) = y$ if x can be found it will be the random variable drawn from the distribution f
- $x = f^{-1}(y)$

f) Expodential Distribution

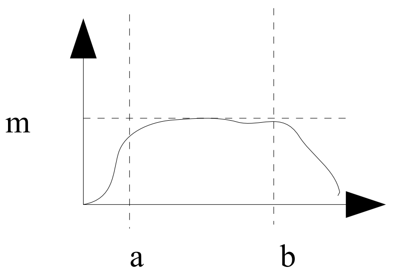


- i.e. the probability of x occurring (P(X = x)) decreases as x increase
- density function: $f(x) = e^{-x}$

- $P(a \leq y \leq b) = \int_a^b e^{-x} d(x)$

g) Rejection Method

- Have known function P(x) which is the desired density function
- and a ranges (a, b) and (0, m) where $m > P(i) \forall i \in (a, b)$
- Uniformly generate random variables u and x
  - if $x \in (a, b) \wedge u < P(x)\, accept\, x$
- A bit like generating random points on a graph and seeing if they are under a line/curve
- Histogram looks like P(x)



- if there is a distribution function *g(x)* for generating random variables then scale it by *c* so that it does not intersect the desired function then generate random *x* values from this function

h) Integrals

- Can use sampling to estimate the integrals of complex functions
- A better solution when in high dimensional space

i) Monte Carlo integration

Problem: calculate $\int_a^b f(x)\, dx$

Algorithm: a large *N* of independent uniform random variables in [a,b] : *x1...xn* with

$x_i \; U(a,b)$

Density of x_i 's is $p(x) = \frac{1}{b-a}$ for $x \in (a,b)$ and 0 everywhere else

$$\int_a^b f(x)\, dx = (b-a) \int_a^b p(x) f(x)\, dx \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

j) Law of large numbers

- Taking lots of samples tends the empricial average to the expected average

- i.e. tossing a coin 10000 times will give more even numbers of heads/tails than tossing a coin 10 times
- EXPECTATION

k) Importance Sampling

- In many cases the contributions to an integral from different regions in X vary considerably
- Want to draw more points from important regions
- Sample points from *p(x)* which resembles *f(x)* closely and write
- $\int_a^b f(x)dx = \int_a^b p(x)\frac{f(x)}{p(x)}dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$
- Where *xi* are drawn randomly from density *p(x)*

l) Markov chains

Consider a sequence of random variables $bold\ X = \{x_{1,}\, x_{2.}..\, x_n\}$

Using conditional probabilities we can write

$$P(x_{1,}\ldots,x_n) = P(x_n|x_{1,}\ldots,x_n-1)\times P(x_{n-1}|x_{1,}\ldots,x_{n-2})\ldots\, P(X_2\vee x_1)\times P(x_1)$$

If the conditional probabilities depend only on the random variable the previous time (not the entire history) for examples $P(x_i|x_{1,}\ldots,x_{i-1}) = P(x_i|x_{i-1})$ then this is *first order markov chain*

m) Transition Matrix

Modules probabilities of changes in state independent of time

$\boldsymbol{T_{k\to l}} = P(x_i+1 = S_l|x_i = S_k)$ i.e each cell contains the probability of transition from state k to state to state l given stare k has occurred previously

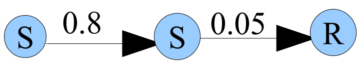
n) Example

$variables\ x_{1,}\, x_{2,}\, x_{3.}\ldots represent\ days$

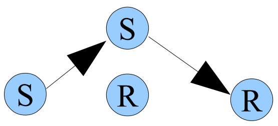
$States = \{SUNNY, RAINY, FOGGY\}$

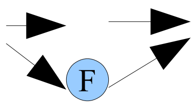
$\boldsymbol{T} = \begin{pmatrix} 0.8 & 0.05 & 0.15 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$ note that rows add up to 1 by convention

$$P(x_3 = RAINY, x_2 = SUNNY | x_1 = SUNNY) = \boldsymbol{T_{1\to1}T_{1\to2}} = 0.8\times0.05 = 0.04$$



$P(x_3 = RAINY | x_1 = SUNNY)$





o) Chaining Example

Using example

(100) i.e. SUNNY on day1

$$\boldsymbol{T}=\begin{pmatrix}0.8 & 0.05 & 0.15\\ 0.2 & 0.6 & 0.2\\ 0.2 & 0.3 & 0.5\end{pmatrix}$$

$(100)\times\boldsymbol{T}$ gives $(\boldsymbol{T}_{1\to1}\boldsymbol{T}_{1\to2}\boldsymbol{T}_{1\to3})$ i.e. probabilities of reaching each state after first day was sunny

$(100)\times\boldsymbol{T}\times\boldsymbol{T}$ gives probabilities of reaching each state on the third day given first day was sunny

p) Chaining Example 2

Given it is sunny on day 1, raining on day 3, what are the probabilities of the different events on day 4

$$\boldsymbol{T}=\begin{pmatrix}0.8 & 0.05 & 0.15\\ 0.2 & 0.6 & 0.2\\ 0.2 & 0.3 & 0.5\end{pmatrix}$$

$$(100)\boldsymbol{T}\begin{pmatrix}T_{1\to2} & 0 & 0\\ 0 & T_{2\to2} & 0\\ 0 & 0 & T_{3\to2}\end{pmatrix}$$

$\begin{pmatrix}T_{1\to2} & 0 & 0\\ 0 & T_{2\to2} & 0\\ 0 & 0 & T_{3\to2}\end{pmatrix}$ signifies that on that day all states must reach state 2

q) Equilibrium

$P(t+1)=P(t)\boldsymbol{T}$

$\lim_{t\to\infty} P\equiv P(t)$ i.e. as the number of steps tends to infinity there becomes less change in the probabilities of any one event occurring in the next step

Achieves *equilibrium* when $P=P\,\boldsymbol{T}$ or $P^T=\boldsymbol{T}^{\boldsymbol{T}}P^T$

P^T is an eigenvector of $\boldsymbol{T}^{\boldsymbol{T}}$ with eigenvalue 1 so..

$(\boldsymbol{T}^{\boldsymbol{T}}-\lambda\boldsymbol{I})P^T=0$ where *p* is a vector containing the equilibrium probabilities of each event happening over an infinite number of steps

r) Example of Equilibrium

$\boldsymbol{T}=\begin{pmatrix}0.4 & 0.6\\ 0.2 & 0.8\end{pmatrix}$ $p=(x,y)^T$

$(\boldsymbol{T}^{\boldsymbol{T}}-\lambda\boldsymbol{I})p=0$

$$\begin{pmatrix} 0.4-1 & 0.2-1 \\ 0.6-1 & 0.8-1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=0$$

Solve simulations equations

$p=(0.25, 0.75)$ i.e. 0.25 chance of reaching state1 after *a long time* and 0.75 chance of reaching state2

s) Maximum Likelihood method

Given a data sequence *D* construct the transition matrix *T* so that the sequence is the most likely to have happened

t) Uses of Markov Chains

1. Document Classification
   - Documents classified into different genre's
   - Each genre has a transition matrix where words are used as states
   - Using each transition matrix a probability is calculated for the likelihood of that document to occur if it was in that genre
   - Genre with the greatest probability is the one the document is most likely a part of

11 Hidden Markov Models

a) Generating the transition matrix

$P(D;\boldsymbol{T})$ Is the probability that a sequence *D* will occur from transition matrix **T**

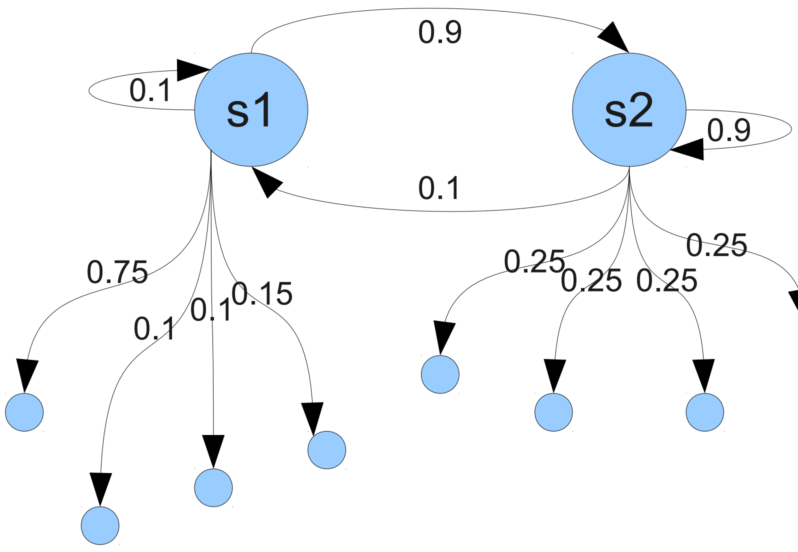
$P(D;\boldsymbol{T})=P(x_1)\prod_{ij}\boldsymbol{T}_{i\to j}^{n_{i\to j}}$ where $n_{i\to j}$ is the number of times that transition took place

Want to maximise $P(D;\boldsymbol{T})$ but normally simplified to minimization with $-\ln P(D;\boldsymbol{T})$

This yields $\boldsymbol{T}_{i\to j}=\frac{n_{i\to j}}{\lambda_{i\to j}}$ where $\lambda_{i\to j}$ is the total number of transitions from state I

There is a bias from the true transition matrix which should be minimised by taking many different data sets, generating their transition matrix and the averaging (or applying similar techniques) to get the true transition matrix

b) Diagram



HMM's have an number of internal states $s_{1,} \ldots, s_n$ which the observer can not see

The observer can only see a series out output symbols $x_{1,} \ldots, x_k$

On each time step, the HMM can output a symbol and change state based on probabilities

Each state has a transition matrix $\boldsymbol{T_n}$ to hold the probabilities of moving to another state

Each state has a probabilities table for emitting a symbol

Hidden because the observer does not know what state the model is in only the emitted symbols

c) Output Sequence

$\boldsymbol{x}=(x_{1,} \ldots., x_T)$ Output sequence

$\boldsymbol{S}=(S_{1,} \ldots, S_T)$ State sequence

Operation (over time) of the HMM can be visualized below

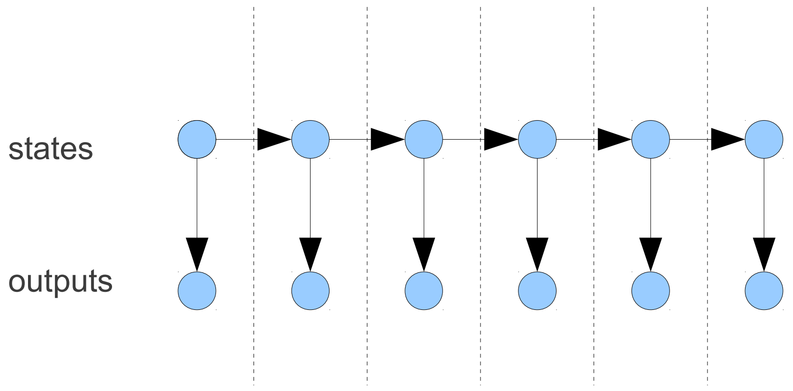


Illustration 1: Visualization of HMM

d) Definitions

Given a sequence **x** and state sequence **S**

$$P(\boldsymbol{x}|\boldsymbol{S})=\prod_{i=0}^{T} P(x_i|S_i)$$

Probability of state sequence **S**

$$P(\boldsymbol{S})=\pi_{S_1} P(S_2|S_1)\ldots P(S_T|S_T-1)$$

$$P(\boldsymbol{x},\boldsymbol{S})=P(\boldsymbol{x}|\boldsymbol{S})P(\boldsymbol{S})$$

$$P(\boldsymbol{x})=\sum_{S} P(\boldsymbol{x}|\boldsymbol{S})P(\boldsymbol{S})$$

e) Forward Algorithm

Calculates $P(\boldsymbol{x})$ i.e. the probability of an output sequence occurring

Forward Algorithm works out this recursively through forward chaining

$\alpha_t(i)=P(S_t=i, x_{1,.}\ldots x_t)$ I..e probability of being in state I and having observed sequence x1 to xt

$\alpha_1(i)=\pi_{S_i} P(x_1|S_i)$ i.e. probability of first symbol from a given state

$\alpha_{t+1}(j)=(\sum_{i\in States} \alpha_t(i)\times a_{i\rightarrow j})\times P(x_{t+1}|j)$ i.e. probability of emitting symbol xt+1 at state j is the probability of reaching that state from any other sequence of states times the probability of moving from these states to j and then times the probability of emitting the symbol from that state

$P(\boldsymbol{x})=\sum_{i} \alpha_T(i)$ i.e. probability is based on taking any path through the states

f) Viterbi Algorithm

Works out the best sequence of states for sequence of outputs (i.e. the one that gives the most probability)

$$P(\boldsymbol{x},\boldsymbol{S})=P(\boldsymbol{x}|\boldsymbol{S})P(\boldsymbol{S})$$

Viterbi finds the **S** which maximizes this value

Application of dynamic programming

$\delta_1(i)=\pi_i \times P(x_1|i)$

$\delta_{t+1}(j)=\max_i[\delta_t(i)\times a_{i\rightarrow j}]\times P(x_{t+1}|j)$

g) Backwards algorithm

$\beta_t(i)=P(x_{t+1},....,x_T|S_t=i)$ ie. Probability of seeing rest of the sequence from a given state

$\beta_T(i)=1$

$\beta_t(i)=\sum_j a_{i\rightarrow j}\beta_{t+1}(j)P(x_{t+1}|j)$

h) Posterior Probability

$$P(S_t=i|\boldsymbol{x})=\frac{P(S_t=i,x_{1,.}...,x_t)P(x_{t+1},...,x_T|S_t=i)}{P(\boldsymbol{x})}$$

$P(S_t=i,x_{1,.}...,x_t)$ can be worked out from forward algorithm

$P(x_{t+1},...,x_T|S_t=i)$ can be worked out from backward algorithm

12 Classification with Perceptrons and Kernels

a) Problem

Classification: Data/Feature → {yes, no}

b) Notation

Training set data:

$D=\{(\boldsymbol{x}_{1,}y_1),....,(\boldsymbol{x}_n,y_n)\}$ where $\boldsymbol{x}_i\in\mathbb{R}^d$ are data vectors and y's are their labels

For classification y comes from a finite set

For regression y is a real number

For unsupervised classification i.e. clustering there is no Y

c) Functions

We will look at binary problems so $y_i\in\{1,-1\}$ and linear classifiers which are Support Vector Machines

$y_i=sign(f(\boldsymbol{x}_i))$ where f is the function we want to learn

Linear function so : $f(x)=w_1x_1.....+w_nx_n+b$ useful trick is to add w_0 and $x_0=1$

$f(x)=w\cdot x$ where w is the weight vector and x is a feature vector

In neural networks or advanced classifiers something more complex than sign() will be used i.e. tanh() to give more of a curve

d) Perceptron Algorithm

```
1. $w = \hat{0}$
2. $err = 0$
3. $for\ i = 1 : N$
        $if\ y_i \times \boldsymbol{w} \cdot x_i \leq 0$   # wrong sign
              $\boldsymbol{w} = \boldsymbol{w} + \eta\, y_i \boldsymbol{x_i}$   #   $\eta$ is learning rate
              $err = err + 1$
4. $if\ err\ != 0\ goto\ 3$
```

Will find **w** if one exists! (proof of convergence)

e) Dual Perceptron Algorithm

W can be seen as a linear combination of the inputs $(\boldsymbol{w} = \boldsymbol{w} + \eta\, y_i x_i)$

$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x_i}$ the weight per example (α) can be found in a similar way to how **w** is found above

```
1. $\alpha = \hat{0}$   # vector for example weightings
2. $err = 0$
3. $for\ i = 1 : N$
        $if\ y_i \times (\sum_{j=1}^{N} \boldsymbol{\alpha_j} y_j \boldsymbol{x_j}) \cdot x_i \leq 0$   # wrong sign
              $\boldsymbol{\alpha_i} = \boldsymbol{\alpha_i} + 1$
              $err = err + 1$
4. $if\ err\ != 0\ goto\ 3$
5. $f(\boldsymbol{x}) = (\sum_{j=1}^{N} \boldsymbol{\alpha_j} y_j \boldsymbol{x_j}) \cdot \boldsymbol{x}$
```

f) Kernel Trick

Sometimes the problem requires a more complex solution than a linear combination

However, inputs can be combined to create a new feature space which can be evaluated linearly

$\Phi : \mathbb{R}^d \rightarrow F$ phi takes a data vector and maps it to a larger-dimensioned feature space vector

I.e $\Phi(\boldsymbol{X}) = (x_1^{2,} x_2^{2,} x_1 x_{2,} x_1 + x_2)$ this could lead to a massive feature space to scan over

However, using the kernel trick this can be reduced...

$k(\boldsymbol{X}, \boldsymbol{Z}) = \Phi(\boldsymbol{X}) \cdot \Phi(\boldsymbol{Z})$ (where X and Z are data vectors) which can sometimes be simplified once expanded out to a simpler form

for comparison function becomes: $f(\boldsymbol{x}) = k((\sum_{j=1}^{N} \boldsymbol{\alpha_j} y_j x_j), \boldsymbol{x})$ or $f(\boldsymbol{x}) = (\sum_{j=1}^{N} \boldsymbol{\alpha_j} y_j \Phi(x_j)) \cdot \Phi(\boldsymbol{x})$

g) Multi-layer Perceptron

Combine different Perceptrons together to get a better result/classifier

Give a weight to each input perceptron and combine

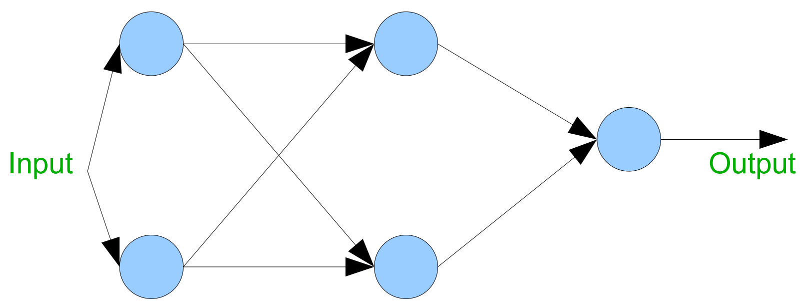


Illustration 2

13 Optimisation cont

a) Safe Guarded methods

Use a hybrid approach between methods such as Golden Section, SPI and Newtons method

This is how must one-dimensional optimisation is done

Switch to which method is most effective at any given point

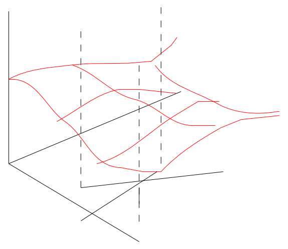
b) Nelder and Mead direct search method

Direct search methods: make no use of function values other than for comparison

work in multidimensional spaces

$f:\mathbb{R}^n \rightarrow \mathbb{R}$ take $n+1$ points forming a *simplex* in $\mathbb{R}^n$

Move worse point so that it is optimised between that point and the centre of the points chosen



Can get very expensive when *n > 3*

c) Steepest Descent Method

$f : \mathbb{R}^n \rightarrow \mathbb{R}$

At point *x* where $-\nabla f(x) < \underline{0}$ then can move downhill to lower values by moving a given amount in the gradient direction

$-\nabla f(x)$ gives local direction of steepest descent

α gives the amount to move by

Start at an initial guess of x_0 and the move

$x_{k+1} = x_k - \alpha \nabla f(x_k)$ then use one-dimensional methods to find α

$\min_{\alpha} f(x_k - \alpha \nabla f(x_k))$ i.e. find value of α which minimizes the function

Tends to zig-zag to the solution

Considered to have a linear convergence close to 1

Can get stuck on local minima

d) Newton's Method in nD

$x_{k+1} = x_k - H_k^{-1}(x_k) \nabla f(x_k)$

Where *H* is the Hessian Matrix of second partial derivatives

Do not explicitly invert the Hessian Matrix but instead solve

$H_f(x_k) s_k = -\nabla f(x_k)$

and take as the next iterate

$x_{k+1} = x_k + s_k$

14 Snakes (Active Contours)

a) Extraction

Deforms a contour to lock onto features of interest

Typically done in images

Typically lines and object boundaries

Transforms feature extraction into a minimization problem

b) Init

Have to start with a parametric function which is an approximation of the solution

i.e. for finding edges of an object you might start with a circle in the middle of the image

Then minimize the points on the contour

c) Objective Function

$v(s) = (x(s), y(s))$

$$E(v)=\int_0^1 E_{\text{int}}(V)+E_{image}(V)ds$$

$$E_{\text{int}}(V)=\alpha\left|\frac{dv}{ds}\right|^2+\beta\left|\frac{d^{2v}}{ds^2}\right|^2$$ (continuity and smoothness)

E_{image} can be made up of many parts which can be given various weighting depending on what is being extracted

$E_{line}=\pm I(v)$ intensity at pixel

$$E_{edgeM}=-|\nabla I(x,y)|$$

$$E_{edge}=-(\nabla I(x,y))^2$$

$$E_{edgeMH}=-(\varrho*\nabla^2 I(x,y))^2$$

15 Matrix Math

a) Solving simulatenous equations

$$\boldsymbol{A}X=\boldsymbol{B}$$

multiply both sides by inverse of A

$$\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{X}=\boldsymbol{A}^{-1}\boldsymbol{B}$$

$$\boldsymbol{A}^{-1}\boldsymbol{A}=\boldsymbol{I}$$

therefore

$$\boldsymbol{X}=\boldsymbol{A}^{-1}\boldsymbol{B}$$

b) Singular Matrix

Determinant == 0

c) Symmetric Matrix

$$\boldsymbol{A}=\boldsymbol{A}^{\boldsymbol{T}}$$

$$\forall i,j\ \boldsymbol{A}_{\boldsymbol{ij}}=\boldsymbol{A}_{\boldsymbol{ji}}$$

“diagonally equal”

$$\begin{pmatrix}1 & 2 & 3\\ 2 & 5 & 6\\ 3 & 6 & 8\end{pmatrix}$$