Probabilistic Turing machines and algorithms, and encryption

Adam Boult (www.bou.lt)

November 2, 2023

Contents

Preface

This is a live document, and is full of gaps, mistakes, typos etc.

Part I

Probabilistic Turing machines and algorithms

Part II

Stochastic methods

Chapter 1

Creating pseudo-random numbers

1.1 Pseudo random numbers

1.1.1 Seeds

1.1.2 Period

Chapter 2

Stochastic methods for integration

2.1 Introduction

Chapter 3

Stochastic optimisation

3.1 Random search

3.1.1 Random search

We start with a random set of parameters, x.

We then loop through the following:

- We define a search space local to our current selection.
- We randomly select a point from this space.
- We compare the new point to our current point. If the new point is better we move to that.

3.1.2 Random optimisation

This is similar to random search, however we use a multivariate Gaussian distribution around our current point rather than a hypersphere.

3.1.3 Simulated annealing

Introduction

We can use a version of Metropolis-Hastings to find the global maximum of a function $f(x)$.

We start with an arbitrary point x_0.

We move randomly from this to identiy a candidate point x_c.

We accept this with probability depending on the the relationship between x_0 and x_c.

This process will converge on the global maximum.

Hyperparameter

There is a hyperparameter for selection. At the extreme this becomes a greedy function.

3.2 Bayesian optimisation

3.2.1 Bayesian optimisation

Introduction

If we have sampled from the hyperparameter space we know something about the shape.

Can we use this to inform where we should next look?

The shape of the function is $y = f(\mathbf{x})$

We have observations $\mathbf{X}$ and $\mathbf{y}$.

So what's our posterior, $P(y|\mathbf{X}, \mathbf{y})$?

Exploration and exploitation

The can be a tradeoff between:

- Exploring - which gives us a better shape for $y = f(x)$; and
- Exploiting - which gives us a better estimate for the global optimum.

The surrogate function

We do not know $y = f(x)$, but we model it as:

$z(x) = y(x) + \epsilon$

We can then maximise z

Proposing new candidates

We want an algorithm which maps from our history of observations to a new candidate.

There are different approaches:

- Probability of improvement - Choosing one with the highest chance of a more optimal value
- Expected improvement - Choosing one with the biggest expected increase in the optimal value

- Entropy search - choosing one which reduces uncertainty about the global maximum.

3.3 Evolutionary algorithms

3.3.1 Evolutionary algorithms

Initialisation

We generate a set of candidate parameter values, x.

Evaluate using the fitness function

We evaluate each of these against a fitness function (the function we are optimising).

We assign fitness values to each individual.

Crossover and mutation

We generate a second generation. We select "parents" randomly using the fitness values as weightings.

The values of the new individual are a function of the values of the parents, and noise (mutation).

We do this for each member in the next generation.

We iterate this process across successive generations.

3.4 Differential evolution

3.4.1 Differential evolution

3.5 Particle swarms

3.5.1 Particle swarms

Chapter 4

Calculus of stochastic processes

4.1 Introduction

4.1.1 Ito integrals

4.1.2 Stochastic differential equations

Chapter 5

Advanced lossless compression

5.1 Huffman encoding

Chapter 6

Lossy compression

6.1 Lossy compression

Chapter 7

Non-cryptographic hashes

7.1 Data integrity checks

7.1.1 Hash functions

Hash functions (take input and return fixed length output) (h=hash(m))

Data integrity checks

Needs to be very different for small changes. so typo has different hash for example. corrput data needs to be noticed.

Checksums

if two files are the same then hashes the same

Introduction

Want following properties for a hash function

Deterministic, so the same hash is always created.

Quick to compute hash

Cannot generate input from hash, except for brute forcing inputs

Small changes to document should cause large charges to hash, such that the two hashes appear uncorrelated

Can't find multiple documents with the same hash, practically.

Can be used to verify files, check passwords.

So possible vulernabilities are:

Given hash, find message (Pre-image resistence)

Given input, find another input with the same hash (second pre-image resistance)

Collission resistance (find two inputs with same hash)

We want to prevent accidental changes to file, and deliberate changes to file. Vulerabilites are more importnat for latter.

7.2 Example of non-cryptogrphic hash functions

7.2.1 Introduction

Part III

Sampling

Chapter 8

Rejection sampling

8.1 Direct sampling

8.1.1 Density estimation through direct sampling

I THINK THE STUFF HERE IS LIMITATIONS TO REJECTION SAMPLING??

DIRECT SAMPLING IS DOING PHYSICAL SAMPLES, MANUALLY PICKING BALLS FROM URL ETC?

There is distribution $P(x)$ which we want to know more about.

If the function was closed, we could estimate it by using values of x.

8.1.2 Limitations of direct sampling

However if the function does not have such a form, we cannot do that.

We can't plug in values, because the function is complex.

Sometimes we may know a function of the form:

$f(x) = cP(x)$

That is, a multiple of the function.

This can happen from Bayes' theorem:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

We may be able to estimate $P(x|y)$ and $P(y)$, but not $P(x)$

This means be have

$P(y|x) = cP(x|y)P(y)$

8.2 Acceptance-rejection sampling

8.2.1 Introduction

Used to sample from propability distribution function.

Useful when can't use direct sampling, because no closed form.

MORE GENERALLY FRAME THESE FIRST AS SAMPLING FROM PROBABILITY FUNCTION.

Generate pairs of (x, y). If $y < P(x)$ then keep x.

Metropolis-Hastings and Gibb's sampling are extensions of this.

Part IV

Communication

Chapter 9

Cryptographic hashes

9.1 Adversaries

9.1.1 Brute force attacks

9.1.2 Pre-image attacks

Given hash value h, can we find message m?

9.1.3 Defence from pre-image attacks

9.1.4 Second pre-image attacks

Given m_1, can we find m_2 with same hash?

Defence from second pre-image attacks

9.1.5 Hash collision

Can i find any two matching messages?

Hash collision attacks

I can get someone to vouch for one of the messages, and then claim they vouched the other.

Hash collision defence

9.2 Passwords

9.2.1 Plaintext databases

9.2.2 Hashed passwords

9.2.3 Rainbow tables

9.2.4 Dictionary attacks

9.2.5 Salting

It is possible to brute force hashes, especially for smaller inputs such as short passwords.

If password hashes for a hashing algorithm were brute forced, then passwords could easily be recovered from another hash table.

To prevent this a salt can be added to the document.

If a password is "apple", then instead the salt "xyz" could be added to create "applexyz". This prevents the previous cracking of "apple" to be used.

The salt would then be stored in plaintext alongside the password hash.

9.3 Examples of cryptographic hash functions

9.3.1 SHA

Chapter 10

Classical encryption

10.1 Introduction

10.1.1 Plaintext and ciphertext

10.1.2 ROT13

Rotate 13. It is its own inverse.

10.1.3 Atbash

Revese the alphabet. It is its own inverse.

10.2 Verifying decryptions

10.2.1 Corpus

verifying solutions when spaces are omitted. can rate fitness using corpus information on popularity

10.3 Caesar

10.3.1 Caesar ciphers

Shift along in alphabet by c.

10.3.2 Affine cipher

page on affine cipher too. like caesar but rather than +c, mx+c

10.3.3 Breaking

For Caesar, only 26 possible keys, can just brute force.

For Affine, can also brute force.

10.4 Monoalphabetic substitution

10.4.1 Monoalphabetic substitution ciphers and keys

(key plus algorithm encrpyts and decrypts)

10.4.2 Breaking monoalphabetic substitution ciphers with frequency analysis

(need to identify algorithm and needs to identify key)

finding substitution cyphers

Search space is larger, $26! = 4 * 10^26$. need alternative to brute force.

Letter popularity. Compare against popularity for corpus. Monogram (ie letters); ngrams(ie n letter in a row frequency); common words.

Single letter words are I or A. More generally. corpus smaller for fewer letters

Can test substitution cypher by matching each word against a corpus

10.5 Polyalphabetic substitution

10.5.1 Polyalphabetic ciphers

Multiple substitution

Vigenere

Rotor machines

The Enigma machine

10.5.2 Breaking polyalphabetic ciphers with the Kasiski examination

10.6 Other

10.6.1 Codebooks

((eg sdrgdr is code for "meet at x on y")

10.6.2 Transposition ciphers

10.6.3 Book cipher

Eg use Bible.

10.6.4 One-time pads

Chapter 11

Modern symmetric encryption

11.1 Methods

11.1.1 Block ciphers

11.1.2 Stream ciphers

11.1.3 Motivation

Increased computer power. How to be secure?

kerckhoff's principle. choose cipher such that secure even if everything but key is known

11.2 Symmetric encryption

11.2.1 Symmetric

We have a document we want to be able to transfer on an insecure medium.

We use a key to encrypt the file, and a key to decrypt the file.

With symmetric encryption these are the same key.

11.3 Options for algorithms

11.3.1 Integer factorisation

Option for algorithm.

11.3.2 Elliptical-curve cryptography

Chapter 12

Modern asymmetric encryption

12.1 Asymmetric encryption

12.1.1 Public keys

12.1.2 RSA

12.1.3 Message signing

12.1.4 Pretty Good Privacy (PGP)

12.1.5 Using public keys to facilitate symmetric encryption

12.1.6 Elliptical-curve cryptography

12.1.7 Asymmetric encryption

Here we use different keys to encrypt and decrypt the file.

Consider two users who wish to send a message securely.

One option would be to use symmetric encrpytion. They would have to meet and share this key securely, however, as transferring it over an insecure network would mean it could be copied.

With public key encryption each user has a public and a private key. The private key is kept secure locally, while the public key can be broadcasted.

In order to encrypt the file, the recipient's public key is used, while both the private and public key are needed to decrypt the file.

As a result anyone can encrypt a file to send to the user, but only the user can read what is sent.

Public-key encryption can be used to facilitate symmetric encryption. If only one party has a public key then the other user can send a symmetric key securely using the public key.

Using this, asymmetric encrpyiton is only used at the start.

This is how HTTPS operates, where the website has a public key, but the client does not.

Each user still needs to trust that the public key is accurate. This could be done by hosting the public key on a secure location.

RSA is an algorithm used for public-key encrpytion, including for HTTPS handshakes and PGP.

12.1.8 Sort

Pages for:

+ Public keys + RSA + Message signing + PGP + Public keys to facilitate symmetric encryption

12.2 Exchanging keys

12.2.1 Diffie-Hellman key exchange