Adaptive Information Processing

Model complexity and the MDL principle

Tjalling Tjalkens and Bert de Vries

February 22, 2008

Signal Processing Group

Global overview

Part A: The Bayesian Information Criterion

Part B: Descriptive complexity

Part C: Bayesian model estimation and the Context-tree model selection

Part A

The Bayesian Information Criterion

Contents

- ML Estimation
  - Parameter estimation
  - Model estimation
- Examples of ML estimation
  - Linear regression
  - Discrete Markov process
  - Conclusion

Contents (ctd.)

- The BIC
  - Laplace approximation
  - Model comparison and cost factors
- Examples of BIC estimation
  - Linear regression
  - Discrete Markov process
  - Conclusion

Parameter estimation

Define our variables!

Model	$\mathcal{M}_i$	model prior	$p(\mathcal{M}_i)$
Parameters	θ_i	parameter prior	$p(\theta_i \mid \mathcal{M}_i)$
Data	x^n		

A-posteriori parameter distribution

$$p(\theta_i \mid \mathcal{M}_i, x^n) = \frac{p(\theta_i \mid \mathcal{M}_i) p(x^n \mid \mathcal{M}_i, \theta_i)}{p(x^n \mid \mathcal{M}_i)}$$

$$p(x^n \mid \mathcal{M}_i) = \int_{\Theta_i} p(\theta_i \mid \mathcal{M}_i) p(x^n \mid \mathcal{M}_i, \theta_i)\, d\theta_i$$

Parameter estimation

Maximum Likelihood

We want a point estimate for $\boldsymbol{\theta_i}$ (given $\boldsymbol{\mathcal{M}_i}$).

$$\hat{\boldsymbol{\theta}}_i = \arg\max_{\boldsymbol{\theta}_i} \boldsymbol{p}(\boldsymbol{\theta}_i|\boldsymbol{\mathcal{M}}_i, \boldsymbol{x}^n) = \arg\max_{\boldsymbol{\theta}_i} \boldsymbol{p}(\boldsymbol{x}^n|\boldsymbol{\mathcal{M}}_i, \boldsymbol{\theta}_i)$$

Where we assume a uniform prior or want to work without priors.

Model estimation

A-posteriori model distribution

$$p(\mathcal{M}_i|x^n) = \frac{p(\mathcal{M}_i)p(x^n|\mathcal{M}_i)}{p(x^n)}$$

$$p(x^n) = \int_{\mathcal{M}_i} p(\mathcal{M}_i)p(x^n|\mathcal{M}_i)\, d\mathcal{M}_i$$

Model estimation

Maximum Likelihood

We want a point estimate for $\mathcal{M}$.

$$\hat{\mathcal{M}} = \arg\max_{\mathcal{M}_i} p(\mathcal{M}_i|x^n) = \arg\max_{\mathcal{M}_i} p(x^n|\mathcal{M}_i)$$

Where we assume a uniform prior or want to work without priors.

Model estimation

We need to compute

$$p(x^n|\mathcal{M}_i) = \int_{\Theta_i} p(\theta_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i)\,d\theta_i$$

Often $p(\theta_i|\mathcal{M}_i, x^n)$ is sharply peaked and because

$$p(\theta_i|\mathcal{M}_i, x^n) \propto p(\theta_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i),$$

we might be able to approximate the integrand given above.

Attempt 1 (Maximum Likelihood)

We approximate the integrand by its peak (θ_i^{MAP} or θ_i^{ML})

$$p(\theta_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i) \approx$$
$$\delta(\theta_i - \theta_i^{\mathrm{ML}})p(\theta_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i)$$

and find

$$p(x^n|\mathcal{M}_i) = p(\theta_i^{\mathrm{ML}}|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i^{\mathrm{ML}})$$

So we end up with

$$\mathcal{M}^{\mathrm{MAP}} = \arg\max_{\mathcal{M}_i} p(\theta_i^{\mathrm{ML}}|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i^{\mathrm{ML}})$$
$$\mathcal{M}^{\mathrm{ML}} = \arg\max_{\mathcal{M}_i} p(x^n|\mathcal{M}_i,\theta_i^{\mathrm{ML}})$$

Attempt 1: an example

Consider a linear regression model.

$$y_n = \theta^T \underline{x}_n + n_n;$$

$$y_n \in \mathbb{R}; \qquad \theta \in \mathbb{R}^k; \qquad \underline{x}_n \in \mathbb{R}^k; \qquad n_n \sim \mathcal{N}(0, \sigma^2)$$

Observe: $(y_1, \underline{x}_1), (y_2, \underline{x}_2), \ldots, (y_N, \underline{x}_N)$.

ML estimate: $\hat{\theta} = (X^T X)^{-1} X^T \underline{y}$.

Matrix: $X = [\underline{x}_1, \underline{x}_2, \ldots \underline{x}_N]^T$.

Models: $\mathcal{M} \subset \{1, 2, \ldots, k\}$. e.g.

$$\mathcal{M} = \{1, 3\}; \qquad y_n = \theta_1 x_{n1} + \theta_3 x_{n3} + n_n$$

Attempt 1: an example (continued)

$$N = 50; \quad \underline{x} \in [0,1]^3; \quad \theta = (0, 0.6, 0);$$

$$\sigma^2 = 1 \quad \text{actual } \sigma^2 = 0.799$$

$\mathcal{M}$	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$	$\hat{\sigma}^2$	$\ln P_{ML}$ $\sigma^2 = 1$	$\ln P_{ML}$ $\sigma^2 = \hat{\sigma}^2$
$\{\}$	0	0	0	0.949	−69.675	−69.642
$\{1\}$	0.690	0	0	0.804	−66.040	−65.485
$\{2\}$	0	0.604	0	0.799	−65.934	−65.352
$\{3\}$	0	0	0.307	0.912	−68.738	−68.635
$\{12\}$	0.379	0.361	0	0.780	−65.441	−64.728
$\{13\}$	1.171	0	−0.522	0.766	−65.099	−64.286
$\{23\}$	0	0.970	−0.472	0.766	−65.101	−64.287
$\{123\}$	0.908	0.752	−0.940	0.686	−63.097	−61.525

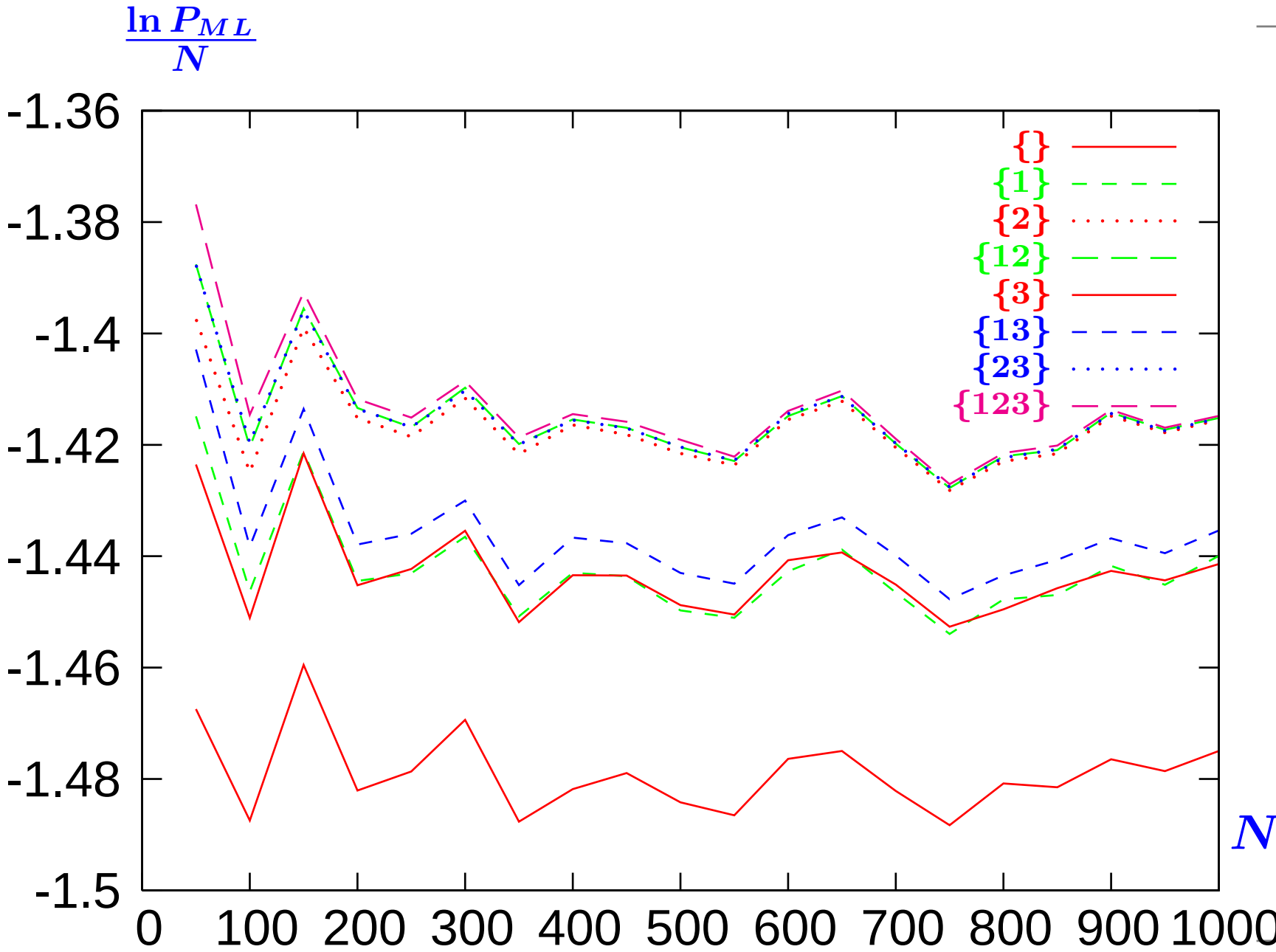
Attempt 1: an example (continued)

$$N = 1000; \quad \underline{x} \in [0,1]^3; \quad \theta = (0, 0.6, 0);$$

$$\sigma^2 = 1 \quad \text{actual } \sigma^2 = 1.015$$

$\mathcal{M}$	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$	$\hat{\sigma}^2$	$\ln P_{ML}$ $\sigma^2 = 1$	$\ln P_{ML}$ $\sigma^2 = \hat{\sigma}^2$
$\{\}$	0	0	0	1.144	−1491	−1486
$\{1\}$	0.435	0	0	1.083	−1460	−1459
$\{2\}$	0	0.619	0	1.015	−1426	−1426
$\{3\}$	0	0	0.507	1.058	−1448	−1447
$\{12\}$	−0.099	0.693	0	1.013	−1425	−1425
$\{13\}$	0.105	0	0.430	1.056	−1447	−1446
$\{23\}$	0	0.549	0.095	1.013	−1426	−1426
$\{123\}$	−0.173	0.622	0.167	1.010	−1424	−1424

Attempt 1: an example (continued)



Attempt 1: another example

A discrete data example.

Consider a binary second order Markov process:
$\mathbf{Pr}\{X_i = 1|x^{i-1}\} = \mathbf{Pr}\{X_i = 1|x_{i-2}x_{i-1}\}$.
So, it is actually a set of four i.i.d. sub-sources.
ML estimate of an i.i.d. binary source:

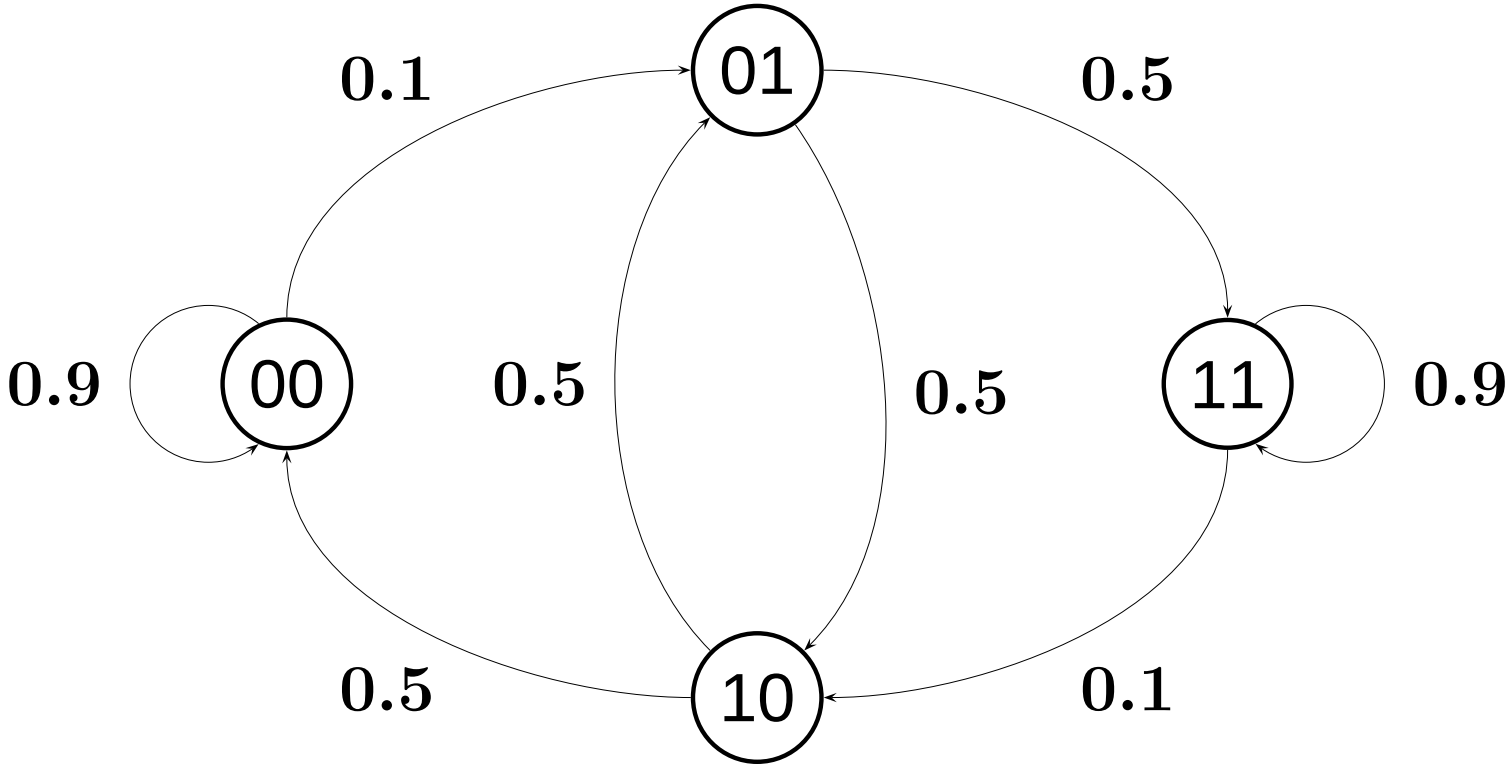
$$N(s|x) = \text{the number of times } s \in \mathcal{X}^* \text{ occurs in } x$$

$$p(x^n|\theta) = (1-\theta)^{N(0|x^n)}\theta^{N(1|x^n)}$$

$$\frac{\partial}{\partial\theta}\ln p(x^n|\theta) = \frac{N(1|x^n) - n\theta}{\theta(1-\theta)} = 0$$

$$\hat{\theta} = \frac{N(1|x^n)}{n}$$

Attempt 1: another example (ctd.)



Attempt 1: another example (ctd.)

Let S be the state variable of an m-th order Markov source, so $S_i = X_{i-m} \dots X_{i-1}$ and $l(S_i) = m$ bits, then

$$\theta_s = \Pr\{X_i = 1 | S_i = s\}$$

The Maximum Likelihood estimator is

$$\hat{\theta}_s = \frac{N(s1|x^n)}{N(s0|x^n) + N(s1|x^n)}$$

With this we find the ML probability for x^n

$$p(x^n|m, \underline{\hat{\theta}}) = p(x_1, \dots x^m) \prod_{s \in \{0,1\}^m} \left\{ \hat{\theta}_s^{N(s1|x^n)} (1 - \hat{\theta}_s)^{N(s0|x^n)} \right\}$$

Attempt 1: another example (ctd.)

```
octave:1> mytest(50,[0.1,0.5,0.5,0.9],4)
ML models sequence logprobs:
Order 0:  logpr = -34.657359
Order 1:  logpr = -14.546445
Order 2:  logpr = -14.883390
Order 3:  logpr = -15.185437
Order 4:  logpr = -15.444986

octave:2> mytest(200,[0.1,0.5,0.5,0.9],4)
ML models sequence logprobs:
Order 0:  logpr = -137.416984
Order 1:  logpr = -102.949521
Order 2:  logpr = -87.992931
Order 3:  logpr = -84.732718
Order 4:  logpr = -80.546002
```

Attempt 1: conclusion

Obviously, this method does not work.

- Any model that includes the actual model assigns essentially the same probability to the data.

Attempt 1: conclusion

Obviously, this method does not work.

- Any model that includes the actual model assigns essentially the same probability to the data.
- We observe that (usually) the higher order models give higher probabilities to the sequence.

Attempt 1: conclusion

Obviously, this method does not work.

- Any model that includes the actual model assigns essentially the same probability to the data.
- We observe that (usually) the higher order models give higher probabilities to the sequence.
- But high order models cannot predict well (too restricted).

Attempt 1: conclusion

Obviously, this method does not work.

- Any model that includes the actual model assigns essentially the same probability to the data.
- We observe that (usually) the higher order models give higher probabilities to the sequence.
- But high order models cannot predict well (too restricted).
- The higher order models are too well tuned.

Attempt 1: conclusion

Obviously, this method does not work.

- Any model that includes the actual model assigns essentially the same probability to the data.
- We observe that (usually) the higher order models give higher probabilities to the sequence.
- But high order models cannot predict well (too restricted).
- The higher order models are too well tuned.

This is undesirable, the estimated model adapts itself to the noise and the resulting model is an over estimation of the actual model.

Attempt 2 (Laplace approximation)

We approximate the integrand by a Gaussian around the peak. The mean and variance of the Gaussian are determined by the integrand.
This approximation turns out to give more interesting results.

Laplace approximation

Suppose we have an arbitrary non-negative real function $f(z)$, where z is a k-dimensional vector. We need an estimate of the normalizing constant Z_f.

$$Z_f = \int f(z)\, dz$$

Let z_0 be a maximum of $f(z)$. Use the Taylor expansion.

$$\ln f(z) \approx \ln f(z_0) - \frac{1}{2}(z - z_0) A (z - z_0)$$

$$A_{ij} = -\frac{\partial^2}{\partial z_i \partial z_j} \ln f(z) \Big|_{z=z_0}$$

Laplace approximation

Approximate $f(z)$ by the unnormalized Gaussian

$$g(z) = f(z_0) \exp\left(-\frac{1}{2}(z - z_0)A(z - z_0)\right)$$

A, not necessarily good, approximation of Z_f is

$$Z_f \approx Z_g = \int g(z)\, dz = f(z_0)\sqrt{\frac{(2\pi)^k}{\det A}}$$

Laplace approximation

Example 1:

$$f(z) = \frac{1}{z^2+1} \quad \text{Has maximum at } z_0 = 0.$$

$$Z_f = \pi$$

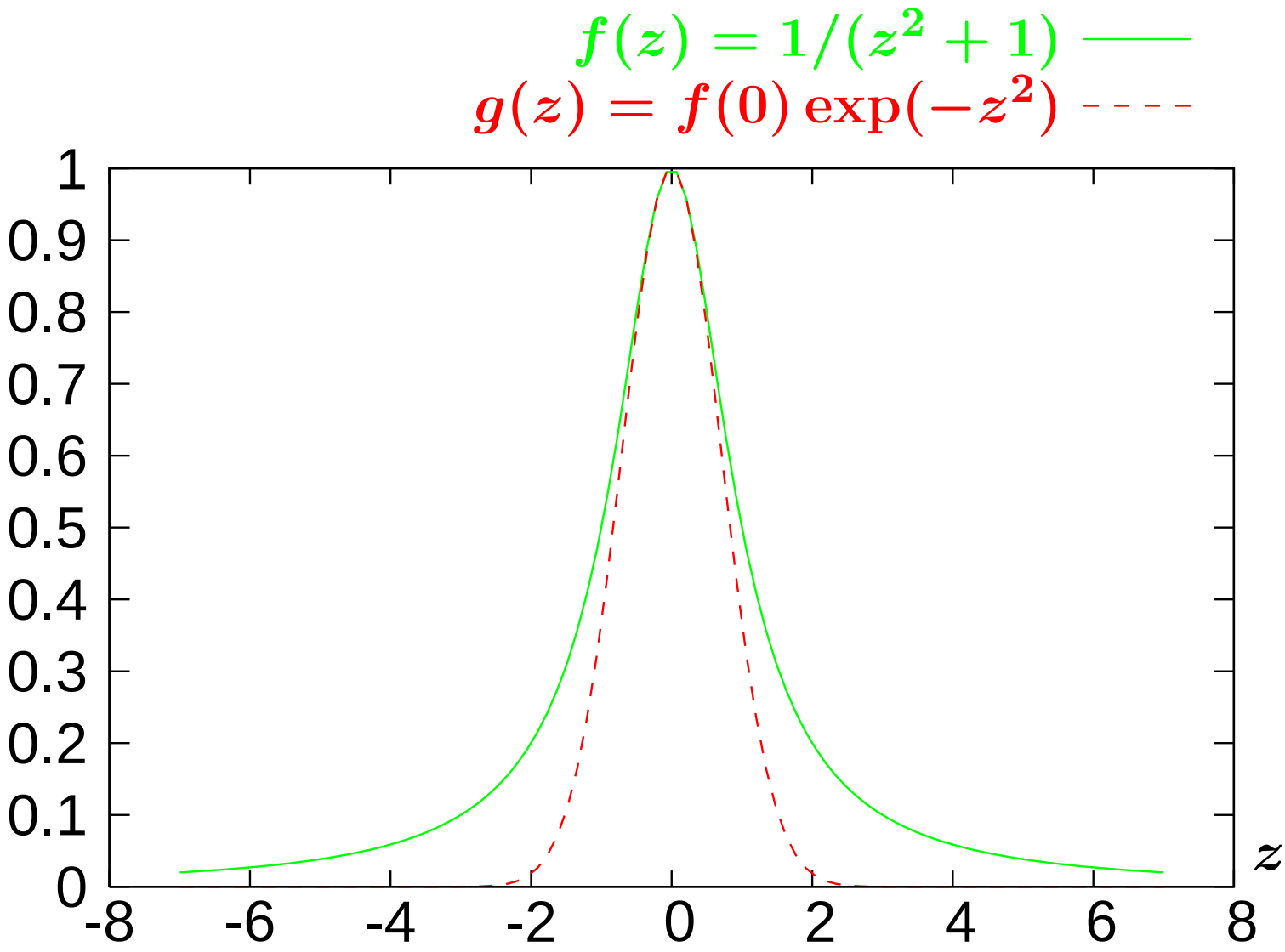
$$A = -\frac{\partial^2}{\partial z^2} \ln f = \frac{f'' f - f'^2}{f^2}$$

$$f(0) = 1; \quad f'(0) = 0; \quad f''(0) = -2; \text{ so } A = 2$$

$$g(z) = f(0) \exp\left(-\frac{1}{2} z A z\right) = e^{-z^2}$$

$$Z_g = \sqrt{\pi}$$

Laplace approximation



Attempt 2 (Laplace approximation)

Consider again $p(x^n|\mathcal{M}_i)$.

$$p(x^n|\mathcal{M}_i) = \int_{\Theta} p(\theta_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i)\,d\theta_i$$

We again use the fact that

$$p(\theta_i|\mathcal{M}_i,x^n) \propto p(\theta_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i)$$

is often sharply peaked, say at $\hat{\theta}_i$. Using the Laplace approximation we may write

$$p(x^n|\mathcal{M}_i) \approx \sqrt{\frac{(2\pi)^k}{\det A}}p(\hat{\theta}_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\hat{\theta}_i)$$

Attempt 2 (Laplace approximation)

Comparing two models give

$$\frac{p(\mathcal{M}_i|x^n)}{p(\mathcal{M}_j|x^n)} \approx \frac{p(\mathcal{M}_i)}{p(\mathcal{M}_j)} \frac{\sqrt{\frac{(2\pi)^{k_i}}{\det A_i}} p(\hat{\theta}_i|\mathcal{M}_i)}{\sqrt{\frac{(2\pi)^{k_j}}{\det A_j}} p(\hat{\theta}_j|\mathcal{M}_j)} \frac{p(x^n|\mathcal{M}_i, \hat{\theta}_i)}{p(x^n|\mathcal{M}_j, \hat{\theta}_j)}$$

Attempt 2 (Laplace approximation)

Comparing two models give

$$\frac{p(\mathcal{M}_i|x^n)}{p(\mathcal{M}_j|x^n)} \approx \frac{p(\mathcal{M}_i)}{p(\mathcal{M}_j)} \frac{\sqrt{\frac{(2\pi)^{k_i}}{\det A_i}} p(\hat{\theta}_i|\mathcal{M}_i)}{\sqrt{\frac{(2\pi)^{k_j}}{\det A_j}} p(\hat{\theta}_j|\mathcal{M}_j)} \frac{p(x^n|\mathcal{M}_i, \hat{\theta}_i)}{p(x^n|\mathcal{M}_j, \hat{\theta}_j)}$$

initial model preference

Attempt 2 (Laplace approximation)

Comparing two models give

$$\frac{p(\mathcal{M}_i|x^n)}{p(\mathcal{M}_j|x^n)} \approx \frac{p(\mathcal{M}_i)}{p(\mathcal{M}_j)} \frac{\sqrt{\frac{(2\pi)^{k_i}}{\det A_i}} p(\hat{\theta}_i|\mathcal{M}_i)}{\sqrt{\frac{(2\pi)^{k_j}}{\det A_j}} p(\hat{\theta}_j|\mathcal{M}_j)} \frac{p(x^n|\mathcal{M}_i, \hat{\theta}_i)}{p(x^n|\mathcal{M}_j, \hat{\theta}_j)}$$

cost of (number of) parameters

Attempt 2 (Laplace approximation)

Comparing two models give

$$\frac{p(\mathcal{M}_i|x^n)}{p(\mathcal{M}_j|x^n)} \approx \frac{p(\mathcal{M}_i)}{p(\mathcal{M}_j)} \frac{\sqrt{\frac{(2\pi)^{k_i}}{\det A_i}} p(\hat{\theta}_i|\mathcal{M}_i)}{\sqrt{\frac{(2\pi)^{k_j}}{\det A_j}} p(\hat{\theta}_j|\mathcal{M}_j)} \frac{p(x^n|\mathcal{M}_i, \hat{\theta}_i)}{p(x^n|\mathcal{M}_j, \hat{\theta}_j)}$$

likelihood ratio

Attempt 2 (Laplace approximation)

Comparing two models give

$$\frac{p(\mathcal{M}_i|x^n)}{p(\mathcal{M}_j|x^n)} \approx \frac{p(\mathcal{M}_i)}{p(\mathcal{M}_j)} \frac{\sqrt{\frac{(2\pi)^{k_i}}{\det A_i}} p(\hat{\theta}_i|\mathcal{M}_i)}{\sqrt{\frac{(2\pi)^{k_j}}{\det A_j}} p(\hat{\theta}_j|\mathcal{M}_j)} \frac{p(x^n|\mathcal{M}_i, \hat{\theta}_i)}{p(x^n|\mathcal{M}_j, \hat{\theta}_j)}$$

Cost factors are: initial model preference, cost of (number of) parameters, likelihood ratio.

This is ML model estimation, it works because we consider the model complexity also!

BIC: Bayesian Information Criterion

A more refined approximation (Schwartz criterion or Bayesian Information Criterion) gives

$$\log \frac{p(\mathcal{M}_i|x^n)}{p(\mathcal{M}_j|x^n)} \approx \log \frac{p(\mathcal{M}_i)}{p(\mathcal{M}_j)} + \log \frac{p(x^n|\mathcal{M}_i, \hat{\theta}_i)}{p(x^n|\mathcal{M}_j, \hat{\theta}_j)} + \frac{1}{2}(k_i - k_j) \log N,$$

where k_i resp. k_j gives the number of free parameters in model $\mathcal{M}_i$ or $\mathcal{M}_j$ respectively.

This BIC is widely applied and turned out to be very usefull.

What happens when we apply the correction term $\frac{k}{2} \log N$? We shall revisit the two examples.

Example 1 with BIC correction

$$N = 50; \quad \underline{x} \in [0,1]^3; \quad \theta = (0, 0.6, 0);$$

$$\sigma^2 = 1 \quad \text{actual } \sigma^2 = 0.852$$

$\mathcal{M}$	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$	$\hat{\sigma}^2$	$\ln P_{BIC}$
$\{\}$	0	0	0	1.068	-72.653
$\{1\}$	0.699	0	0	0.909	-70.632
$\{2\}$	0	0.773	0	0.841	-68.923
$\{3\}$	0	0	0.572	0.944	-71.491
$\{12\}$	0.159	0.662	0	0.837	-70.790
$\{13\}$	0.553	0	0.172	0.905	-72.478
$\{23\}$	0	0.811	-0.050	0.840	-70.869
$\{123\}$	0.240	0.728	-0.159	0.834	-72.670

Example 1 with BIC correction

$$N = 1000; \quad \underline{x} \in [0,1]^3; \quad \theta = (0, 0.6, 0);$$

$$\sigma^2 = 1 \quad \text{actual } \sigma^2 = 0.977$$

$\mathcal{M}$	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$	$\hat{\sigma}^2$	$\ln P_{BIC}$
$\{\}$	0	0	0	1.077	-1457.2
$\{1\}$	0.411	0	0	1.022	-1433.4
$\{2\}$	0	0.551	0	0.976	-1410.3
$\{3\}$	0	0	0.362	1.034	-1439.3
$\{12\}$	-0.017	0.564	0	0.976	-1413.7
$\{13\}$	0.315	0	0.128	1.020	-1435.6
$\{23\}$	0	0.637	-0.117	0.974	-1412.7
$\{123\}$	0.040	0.620	-0.133	0.974	-1416.1

Example 2 with BIC correction

```
octave:1> mytest(50,[0.1,0.5,0.5,0.9],4)
Parameter scaled ML log probabilities:
Order 0:  logpr = -36.613371
Order 1:  logpr = -18.458468
Order 2:  logpr = -22.707436
Order 3:  logpr = -30.833529
Order 4:  logpr = -46.741170

octave:2> mytest(200,[0.1,0.5,0.5,0.9],4)
Parameter scaled ML log probabilities:
Order 0:  logpr = -140.066143
Order 1:  logpr = -108.247838
Order 2:  logpr = -98.589565
Order 3:  logpr = -105.925987
Order 4:  logpr = -122.932541
```

BIC correction

The examples indicate that the correct model (order) is recovered, basically by using an ML selection criterion with an additional penalty term for the model complexity.

However, this BIC is derived as an approximation to the true Bayesian a-posteriori probability.

A better justification for the BIC should exist!

Part B

Descriptive complexity

Contents

- Introduction
- Complexity of a sequence
  - Universal computer
  - Kolmogorov complexity
  - Kolmogorov sufficient statistic
- Probabilistic complexity
  - Shannon complexity
  - Universal data compression
  - Redundancy-capacity theorem

Contents (ctd.)

- Model estimation
  - Meaning of model information
  - Bayesian model estimation
  - Terminology
  - Stochastic complexity
- Coding for Gaussians

Prediction and estimation

Consider an observed sequence v.

$$v = \mathcal{F}(u) + \eta$$

Prediction and estimation

Consider an observed sequence v.

$$v = \mathcal{F}(u) + \eta$$

- $\mathcal{F}(u)$ is the structure part,

Prediction and estimation

Consider an observed sequence v.

$$v = \mathcal{F}(u) + \eta$$

- $\mathcal{F}(u)$ is the structure part,
- η is the independent noise part.

u might or might not be known.

Prediction and estimation

Consider an observed sequence v.

$$v = \mathcal{F}(u) + \eta$$

- $\mathcal{F}(u)$ is the structure part,
- η is the independent noise part.

u might or might not be known.

Assume that $v = v_1 v_2 \ldots v_n$.

prediction Given v, what will be the next symbol v_{n+1}?

Prediction and estimation

Consider an observed sequence v.

$$v = \mathcal{F}(u) + \eta$$

- $\mathcal{F}(u)$ is the structure part,
- η is the independent noise part.

u might or might not be known.

Assume that $v = v_1 v_2 \ldots v_n$.

prediction Given v, what will be the next symbol v_{n+1}?

estimation Given v, what is the original data u?

Prediction and estimation

Consider an observed sequence v.

$$v = \mathcal{F}(u) + \eta$$

- $\mathcal{F}(u)$ is the structure part,
- η is the independent noise part.

u might or might not be known.

Assume that $v = v_1 v_2 \ldots v_n$.

prediction Given v, what will be the next symbol v_{n+1}?

estimation Given v, what is the original data u?

modelling Given v, what is the generating model $\mathcal{F}$?

Prediction and estimation

In all cases we need to estimate $\mathcal{F}$.

We will see that the separation of v into a structure and a noise part has very general power. It can also be seen in the following way. The data sequence contains information about the structure that generated it and the remainder is a random variation of the outcomes that contain no additional information about the structure (i.e. it is “pure” noise).

Closely related to this information is the notion of complexity which we shall discuss first.

The complexity of a sequence

- Descriptive complexity
- The universal computer
- Solomonov-Kolmogorov-Chaitin and Shannon complexity

The complexity of a sequence

- Descriptive complexity
- The universal computer
- Solomonov-Kolmogorov-Chaitin and Shannon complexity

The complexity of a sequence

- Descriptive complexity
- The universal computer
- Solomonov-Kolmogorov-Chaitin and Shannon complexity

The complexity of a sequence

- Descriptive complexity
- The universal computer
- Solomonov-Kolmogorov-Chaitin and Shannon complexity

Descriptive complexity:
How difficult is it to describe this sequence?

Descriptive complexity

0101010101010101010101010101010101010101010101010101010101

Descriptive complexity

0101010101010101010101010101010101010101010101010101

25 repetitions of “01”, simple

1011010100000100111100110011001111111001110111001

Descriptive complexity

0101010101010101010101010101010101010101010101010101

25 repetitions of "01", simple

1011010100000100111001100110011111111001110111001

The first 50 fractional bits of $1/\sqrt{2}$, simple

0011011100100100010011111100001011001000101100110 0

Descriptive complexity

0101010101010101010101010101010101010101010101010101

25 repetitions of "01", simple

1011010100000100111001100110011111111001110111100 1

The first 50 fractional bits of $1/\sqrt{2}$, simple

0011011100100100010011111100001011001000101100110 0

50 coin tosses, complex

Descriptive complexity

0101010101010101010101010101010101010101010101010101

25 repetitions of "01", simple

1011010100000100111001100110011111111001110111100 1

The first 50 fractional bits of $1/\sqrt{2}$, simple

0011011100100100010011111100001011001000101100110 0

50 coin tosses, complex

Simple sequences are "easy" to describe, complex ones must be described symbol by symbol.

The universal computer

- All normally powerful computers can be reduced to Turing Machines and conversely.

The universal computer

- All normally powerful computers can be reduced to Turing Machines and conversely.
- All functions that can be computed are computable by a Turing Machine.

The universal computer

- All normally powerful computers can be reduced to Turing Machines and conversely.
- All functions that can be computed are computable by a Turing Machine.

  Thus we say that a Turing Machine is a “Universal Computer”.

The universal computer

- All normally powerful computers can be reduced to Turing Machines and conversely.
- All functions that can be computed are computable by a Turing Machine.

  Thus we say that a Turing Machine is a “Universal Computer”.

  This implies that any ordinary computer is a Universal Computer.

The universal computer

- All normally powerful computers can be reduced to Turing Machines and conversely.
- All functions that can be computed are computable by a Turing Machine.

  Thus we say that a Turing Machine is a “Universal Computer”.

  This implies that any ordinary computer is a Universal Computer.

  (Assuming sufficient storage capacity and enough patience by the user.)

The universal computer

- If program p, when executed on a (universal) computer $\mathcal{U}$, prints the output sequence x and then halts, we say that x is produced by p on $\mathcal{U}$ and write

$$x = \mathcal{U}(p)$$

The universal computer

- If program p, when executed on a (universal) computer $\mathcal{U}$, prints the output sequence x and then halts, we say that x is produced by p on $\mathcal{U}$ and write

$$x = \mathcal{U}(p)$$

- The length of a program is almost independent of the computer and programming language used.

The universal computer

- If program p, when executed on a (universal) computer $\mathcal{U}$, prints the output sequence x and then halts, we say that x is produced by p on $\mathcal{U}$ and write

$$x = \mathcal{U}(p)$$

- The length of a program is almost independent of the computer and programming language used.
- The complexity of a computation should be related to the length of the program that computes it.

Kolmogorov complexity

Definition: The (Solomonov-)Kolmogorov(-Chaitin) complexity of a sequence x that is computed on a universal machine $\mathcal{U}$ is defined by the length $l(p)$ of the shortest program p that produces x on $\mathcal{U}$, or $\mathcal{U}(p) = x$. It is written as

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} l(p).$$

Kolmogorov complexity

Definition: The (Solomonov-)Kolmogorov(-Chaitin) complexity of a sequence x that is computed on a universal machine $\mathcal{U}$ is defined by the length $l(p)$ of the shortest program p that produces x on $\mathcal{U}$, or $\mathcal{U}(p) = x$. It is written as

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} l(p).$$

The length of a program p is (almost) independent of the computer it runs on, so in the following we shall not mention the computer in the complexities and write $K(x)$ in stead of $K_{\mathcal{U}}(x)$.

Kolmogorov complexity

Example 2: Print the first 1000 digits of the number e.

This program is 43 characters long.

Using 8 bits per character we find that the complexity of this sequence is not more than (approximately) 344 bits.

There are however 10^{1000} sequences of 1000 digits!

Most shortest programs are approximately

$\log_2 10^{1000} \approx 3300$ bits long!!

Kolmogorov complexity

The Kolmogorov complexity is non-computable.
This is related to the halting problem. It has been shown that there is no general procedure to determine for all programs if they will eventually halt.

Kolmogorov complexity

The Kolmogorov complexity is non-computable.
This is related to the halting problem. It has been shown that there is no general procedure to determine for all programs if they will eventually halt.

So when we test all programs, in order of increasing length, to see if they produce x, it is not guaranteed that we can find the answer because unless a program has already stopped we will never know if it will stop eventually.

Kolmogorov complexity

The Kolmogorov complexity is non-computable.
This is related to the halting problem. It has been shown that there is no general procedure to determine for all programs if they will eventually halt.

So when we test all programs, in order of increasing length, to see if they produce x, it is not guaranteed that we can find the answer because unless a program has already stopped we will never know if it will stop eventually.

Definition: The conditional Kolmogorov complexity, knowing the length $l(x)$ is defined as

$$K(x|l(x)) = \min_{p:\mathcal{U}(p,l(x))=x} l(p).$$

Kolmogorov complexity

Theorem 1 *The conditional Kolmogorov complexity is (essentially) less than the length of the sequence.*

$$K(x|l(x)) \leq l(x) + c.$$

Kolmogorov complexity

Theorem 1 *The conditional Kolmogorov complexity is (essentially) less than the length of the sequence.*

$$K(x|l(x)) \leq l(x) + c.$$

Proof: Embed x in the program. Because the length of x is known, the end of the program is well defined. A possible program would be

Print the following l sequence: $x_1 x_2 \ldots x_l$.

Because the length l is known there is no need to describe it in the program and thus the length of this program is $l(x) + c$.

□

Kolmogorov complexity

If the computer does not know the length l of x, then it must be included in the program. A coding technique, due to Elias, will help in the proof of the following theorem.

Kolmogorov complexity

If the computer does not know the length l of x, then it must be included in the program. A coding technique, due to Elias, will help in the proof of the following theorem.

Theorem 2 *[An upper bound to the Kolmogorov complexity.]*

$$K(x) \leq K(x|l(x)) + 2\log_2 l(x) + c.$$

For a description and proof of this theorem and the Elias code we refer to the appendix.

Kolmogorov complexity

As it turns out, there aren't many sequences that have a small complexity because there aren't many short programs.
We list all programs of a length less than k and we get

$$0, 00, 01, 10, 11, 000, \ldots, \underbrace{11\ldots1}_{k-1}$$

Not all programs will halt and any program will produce at most one sequence. So the number of low complexity sequences is less than 2^k, because

$$\sum_{i=1}^{k-1} 2^i = 2^k - 1 < 2^k.$$

Kolmogorov complexity

Example 3: x is a sequence of n zeroes.

A program could be: “Print the given number of zeroes.” This program has a fixed length, independent of n, so we can conclude that

$$K(0^n|n) = c, \text{ for all } n$$

Kolmogorov complexity

Example 4: x consists of the first n bits of π.

A program computes the first n bits of π using a series expansion. This program also has a fixed length, independent of n, so we can conclude that

$$K(\pi_1\pi_2\ldots\pi_n|n) = c, \text{ for all } n$$

Kolmogorov complexity

Example 5: x is an arbitrary sequence of n bits.

x contains k ones and $n-k$ zeroes. Knowing n we first encode k in $\log_2 n + c_1$ bits (c_1 is a constant) and then we must specify which one of the $\binom{n}{k}$ possible sequences with k ones we deal with. This cost another $\log_2 \binom{n}{k} + c_2$ bits. It is well known (and a similar result will be shown soon) that

$$\frac{1}{n+1} 2^{nh(\frac{k}{n})} \leq \binom{n}{k} \leq 2^{nh(\frac{k}{n})},$$

where $h(z) = -z \log_2 z - (1-z)\log_2(1-z)$ is the binary entropy function. Thus we find with $\bar{x} = k/n$

$$K(x^n|n) \leq nh(\bar{x}) + \log_2 n + c, \text{ for all } n$$

The Kolmogorov sufficient statistic

Recall the complexity estimate of a binary string with k ones and $n - k$ zeroes. First we described some general (model) information which limited the set of possible sequences and then we identified the actual sequence in the restricted set. This two stage description idea will be generalized.

The Kolmogorov sufficient statistic

Recall the complexity estimate of a binary string with k ones and $n-k$ zeroes. First we described some general (model) information which limited the set of possible sequences and then we identified the actual sequence in the restricted set. This two stage description idea will be generalized.

1. We look for a simple and small set S that contains our sequence x.

The Kolmogorov sufficient statistic

Recall the complexity estimate of a binary string with k ones and $n - k$ zeroes. First we described some general (model) information which limited the set of possible sequences and then we identified the actual sequence in the restricted set. This two stage description idea will be generalized.

1. We look for a simple and small set $\mathcal{S}$ that contains our sequence x.
2. We identify x in $\mathcal{S}$ using $\log_2 |\mathcal{S}|$ bits, (ignoring rounding up of the log).

The Kolmogorov sufficient statistic

Recall the complexity estimate of a binary string with k ones and $n-k$ zeroes. First we described some general (model) information which limited the set of possible sequences and then we identified the actual sequence in the restricted set. This two stage description idea will be generalized.

1. We look for a simple and small set $\mathcal{S}$ that contains our sequence x.
2. We identify x in $\mathcal{S}$ using $\log_2 |\mathcal{S}|$ bits, (ignoring rounding up of the log).

The first part describes the structure of the sequence. The second part carries no information about the structure.

The Kolmogorov sufficient statistic

To make this more precise we define the smallest set $\mathcal{S}$ that contains x and is itself describable in at most k bits.
For this we define a new program output

$$\mathcal{U}(p, n) = \mathcal{S}.$$

So the output defines a set of sequences, maybe by printing out the indicator function of $\mathcal{S}$.

Definition: The Kolmogorov structure function $K_k(x^n|n)$ is defined as

$$K_k(x^n|n) = \min_{\substack{p: l(p) \leq k \\ \mathcal{U}(p,n)=\mathcal{S} \\ x^n \in \mathcal{S}}} \log_2 |\mathcal{S}|$$

The Kolmogorov sufficient statistic

The total program length is now

$$K_k(x^n|n) + k$$

If $k = K(x^n|n)$ then the structure part describes a set containing x only, so it makes sense to define the Kolmogorov minimal sufficient statistic for a sequence x.

Definition: Let c be a given small constant. k^* is the Kolmogorov minimal sufficient statistic and is defined as the smallest k for which

$$K_k(x^n|n) + k \leq K(x^n|n) + c$$

holds.

The Kolmogorov sufficient statistic

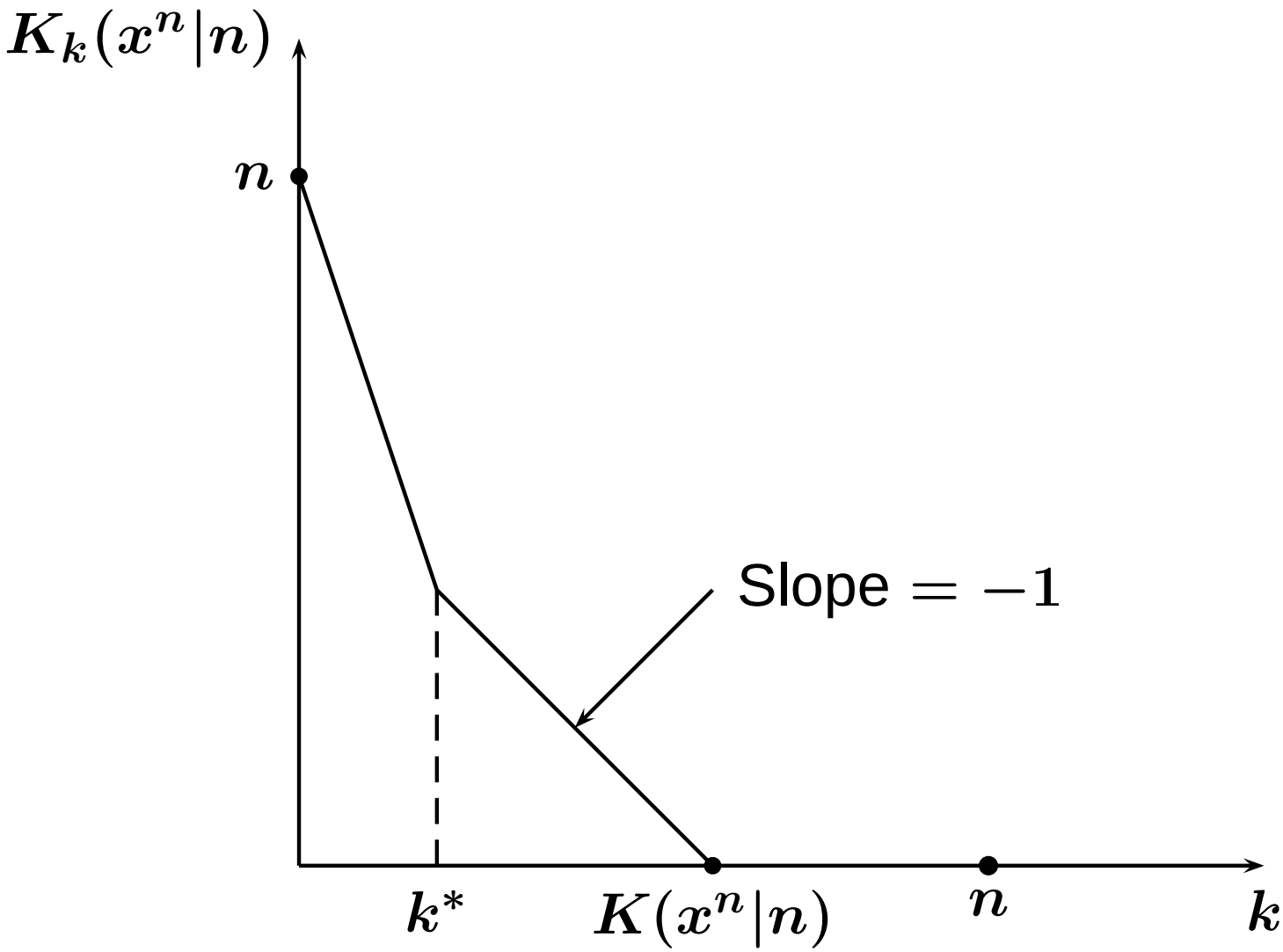
When $k = 0$, no structure can be given, so $\mathcal{S} = \mathcal{X}^n$ and the structure function is maximal, $K_k(x^n|n) = n$, (where we assume $\mathcal{X}$ to be binary).

Then, as k increases, the size of $\mathcal{S}$ drops quickly until $K_k(x^n|n) + k \approx K(x^n|n)$.

After that, each additional bit of k halves the size of the set $\mathcal{S}$, so $K_k(x^n|n)$ drops with slope -1.

When $k \geq K(x^n|n)$ the set $\mathcal{S}$ has become a singleton containing only x, and thus $K_k(x^n|n) = 0$.

The Kolmogorov sufficient statistic



The Kolmogorov sufficient statistic

Example 6: Consider the sequence $x^n = 0^n$.

We know that $K(x^n|n) = c$.

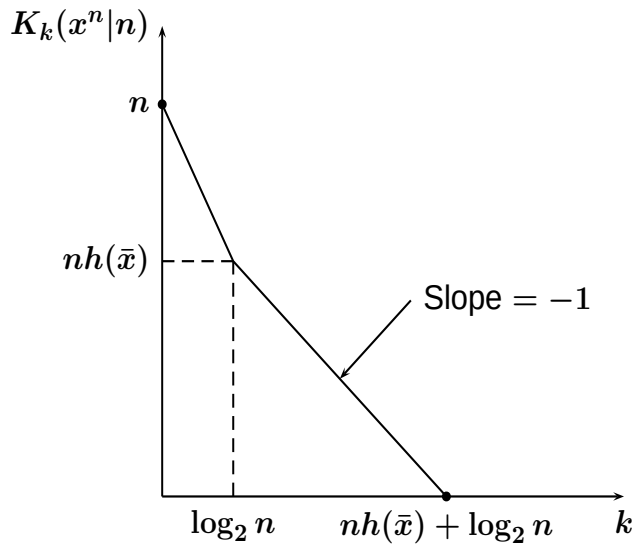
A short and fixed length program describes the printing of zeroes.
So the structure program p is very small and the Kolmogorov minimal sufficient statistic k^* is almost zero.
There is (almost) no structure information!

The set S contains only 0^n so there is also no "noise", i.e. $K_{k^*}(x^n|n) \approx 0$.

The Kolmogorov sufficient statistic

Example 7: Consider a binary sequence x^n drawn from an i.i.d. binary source (with unknown $\Pr\{X = 1\} = \theta$).

The two-part description first describes the parameter θ by giving the number of 1's in x. This usually costs approximately $\log_2 n$ bits. (Except for special values of θ.) The "noise" costs approximately $nh(\bar{x})$ bits.



The Kolmogorov sufficient statistic

Summary

$$K(x^n|n) \approx k^* + K_{k^*}(x^n|n)$$

The Kolmogorov sufficient statistic

Summary

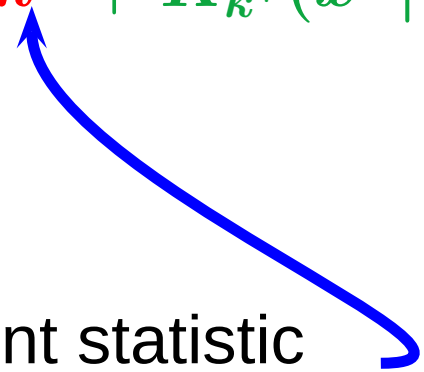
$$K(x^n|n) \approx k^* + K_{k^*}(x^n|n)$$

Kolmogorov complexity

The Kolmogorov sufficient statistic

Summary

$$K(x^n|n) \approx k^* + K_{k^*}(x^n|n)$$



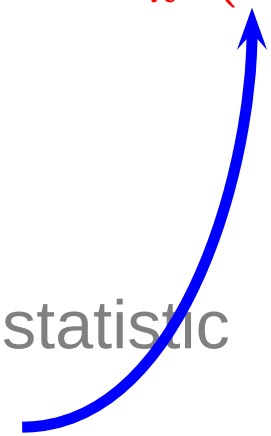
Kolmogorov complexity

Kolmogorov minimal sufficient statistic

The Kolmogorov sufficient statistic

Summary

$$K(x^n|n) \approx k^* + K_{k^*}(x^n|n)$$



Kolmogorov complexity
Kolmogorov minimal sufficient statistic
Kolmogorov structure function

The Kolmogorov sufficient statistic

Summary

$$K(x^n|n) \approx k^* + K_{k^*}(x^n|n)$$

Kolmogorov complexity
Kolmogorov minimal sufficient statistic
Kolmogorov structure function

The minimal sufficient statistic gives the
complexity of the model
that best fits the data.

Shannon complexity

Can the (Shannon) entropy be considered as a measure of complexity?

Yes, but the entropy depends on the probability of a sequence given an underlying source or stochastic data generating process.

Assuming that a source assigns probabilities $\mathbf{Pr}\{X = x\}$ the entropy of the source is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} \mathbf{Pr}\{X = x\} \log_2 \mathbf{Pr}\{X = x\}.$$

This is the expected number of bits needed to represent X.

Shannon complexity

For a sequence x a corresponding notion is the ideal code wordlength given as

$$I(x) = -\log_2 \Pr\{X = x\}.$$

This can be interpreted as the most favorable representation length.

A disadvantage of Shannon's measures seems to be the fact that the complexity of a sequence depends on the probability of the sequence and not on the sequence itself.

Shannon complexity

Example 8: [of the 'unreasonable' interpretation]

Let $\mathcal{X} = \{01101010000010011110, 00110111001001000100\}$ and let the source select between the two sequences with equal probability $(\frac{1}{2}, \frac{1}{2})$.

The entropy of the source is 1 bit per sequence (of 20 symbols)! However, the two strings each appear much more complex than 1 bit!!

The complexity is hidden in the source description, namely in $\mathcal{X}$, which is already known by the receiver. Later, we shall see that universal data compression gives a more fundamental answer to this problem.

Shannon complexity

Suppose x is selected by a memoryless source with alphabet $\mathcal{X}$ and probabilities $p(a)$ for all $a \in \mathcal{X}$.
The following theorem stated that the expected Kolmogorov complexity is very close to the entropy of the memoryless source.

Theorem 3

$$\mathbf{E}\left\{\frac{1}{n}K(X^n|n)\right\} \rightarrow H(X)$$

This implies, together with $K(x|n) \leq n + c$ that for most selected sequences both complexity measures will be essentially equal.

Shannon complexity

Proof: First is is easy to see that the set of shortest programs that halt on a machine must satisfy Kraft's inequality.

$$\sum_{\substack{p:p \text{ halts} \\ p \text{ shortest}}} 2^{-l(p)} \leq 1.$$

Namely if p halts on $\mathcal{U}$ then a program with p as prefix, say $p' = ps$ produces the same result on $\mathcal{U}$ as p.

This directly results in the lower bound because the set of shortest programs for all X^n form a prefix free set and thus have an expected length

$$\sum_{x^n \in \mathcal{X}^n} p^n(x) K(x^n|n) \geq nH(X)$$

Shannon complexity

The proof of the upper bound follows the argument of Example 5 and makes use of Theorem 5 and Theorem 7 (see the Appendix).

First we describe the type of a sequence x in $|\mathcal{X}| \log_2 n + c_1$ bits (Theorem 5) and then index the sequence using $nH(P_x) + c_2$ bits (Theorem 7).

Now we take the expectation.

$$\begin{aligned} \mathbf{E}\{K(x^n|n)\} &\leq \mathbf{E}\{nH(P_x)\} + |\mathcal{X}| \log_2 n + c \\ &\overset{(*1)}{\leq} nH(X) + |\mathcal{X}| \log_2 n + c \end{aligned}$$

Ad (*1): This follows from Jensen's inequality.

Kolmogorov and Shannon complexity

- The Kolmogorov complexity is more general, and deals with individual sequences without an underlying probability assumption, while Shannon's entropy strongly depends on these probabilities.

Kolmogorov and Shannon complexity

- The Kolmogorov complexity is more general, and deals with individual sequences without an underlying probability assumption, while Shannon's entropy strongly depends on these probabilities.
- For (i.i.d.) sources, but also more general sources, both complexities are mostly comparable.

Kolmogorov and Shannon complexity

- The Kolmogorov complexity is more general, and deals with individual sequences without an underlying probability assumption, while Shannon's entropy strongly depends on these probabilities.
- For (i.i.d.) sources, but also more general sources, both complexities are mostly comparable.
- The Kolmogorov complexity is uncomputable and Shannon's entropy can be computed or approximated arbitrarily close.

Universal data compression

Is it also possible to find similar results like the minimal sufficient statistics for Shannon's information measure?

Remember the example of the Kolmogorov complexity of a binary sequence with k ones?

Many Kolmogorov complexities are actually upper bounded by code wordlengths of data compression codes!

But we must consider parametrized classes of sources.

Universal data compression

Example 9:

Parametrized binary source (I.I.D. source class)

Alphabet: $\mathcal{X} = \{0, 1\}$;

Sequence: $x = x_1 \dots x_n$;

(n is the block length)

Probabilities: $\Pr\{X_i = 1\} = 1 - \Pr\{X_i = 0\} = \theta$.

$0 \leq \theta \leq 1$.

Code: $C : \mathcal{X}^n \to \{0, 1\}^*$

Code word: $c(x) = c_1 \dots c_j \in C$

Length: $l_C(x) = l(c_1 \dots c_j) = j$

Universal data compression

Ideal code wordlength

The best possible code wordlengths come from Huffman's algorithm, but these are hard to compute.

The task: minimize over the choice of lengths $l_C(x)$

$$\sum_{x \in \mathcal{X}} p(x) l_C(x)$$

where the lengths must satisfy Kraft's inequality

$$\sum_{x \in \mathcal{X}} 2^{-l_C(x)} \leq 1$$

Universal data compression

Ideal code wordlength

Ignoring the requirement that code wordlengths are integer, we find that the optimal code wordlengths are

$$l_C(x) = -\log_2 p(x)$$

The upward rounded version of these lengths still satisfy Kraft's inequality and the resulting code achieves Shannon's upper bound.

We write $l_C^*(x)$ for these ideal code wordlengths.

$$\begin{aligned} l_C^*(x) &= \lceil -\log_2 p(x) \rceil \\ &< -\log_2 p(x) + 1 \end{aligned}$$

Universal data compression

Remember $N(a|x)$ is the number of times the symbol a occurs in x.

Sequence probability: $p(x) = (1-\theta)^{N(0|x)}\theta^{N(1|x)}$

Expected code word length: $\bar{l_C} = \sum_{x \in \mathcal{X}^n} p(x) l_C(x)$

(Expected) code rate: $R_n = \frac{\bar{l_C}}{n}$

(Expected) code redundancy: $r_n = R_n - h(\theta)$

Universal data compression

First assume that we know that $\theta = \theta_1 = 0.2$ or $\theta = \theta_2 = 0.9$ but we don't know which θ generated x.

Universal data compression

First assume that we know that $\theta = \theta_1 = 0.2$ or $\theta = \theta_2 = 0.9$ but we don't know which θ generated x.

We design a code C_1 assuming that $\theta = \theta_1$.

Universal data compression

First assume that we know that $\theta = \theta_1 = 0.2$ or $\theta = \theta_2 = 0.9$ but we don't know which θ generated x.

We design a code C_1 assuming that $\theta = \theta_1$.

And a code C_2 assuming $\theta = \theta_2$.

Universal data compression

First assume that we know that $\theta = \theta_1 = 0.2$ or $\theta = \theta_2 = 0.9$ but we don't know which θ generated x.

We design a code C_1 assuming that $\theta = \theta_1$.

And a code C_2 assuming $\theta = \theta_2$.

We also create the code C_{12} which uses the smallest code word from C_1 and C_2 with a '0' or '1' prepended to indicate from which code the word comes.

Universal data compression

First assume that we know that $\theta = \theta_1 = 0.2$ or $\theta = \theta_2 = 0.9$ but we don't know which θ generated x.

We design a code C_1 assuming that $\theta = \theta_1$.

And a code C_2 assuming $\theta = \theta_2$.

We also create the code C_{12} which uses the smallest code word from C_1 and C_2 with a '0' or '1' prepended to indicate from which code the word comes.

In all cases the code words are created using the ideal code wordlengths $l_C^*(x)$.

Universal data compression

First assume that we know that $\theta = \theta_1 = 0.2$ or $\theta = \theta_2 = 0.9$ but we don't know which θ generated x.

We design a code C_1 assuming that $\theta = \theta_1$.

And a code C_2 assuming $\theta = \theta_2$.

We also create the code C_{12} which uses the smallest code word from C_1 and C_2 with a '0' or '1' prepended to indicate from which code the word comes.

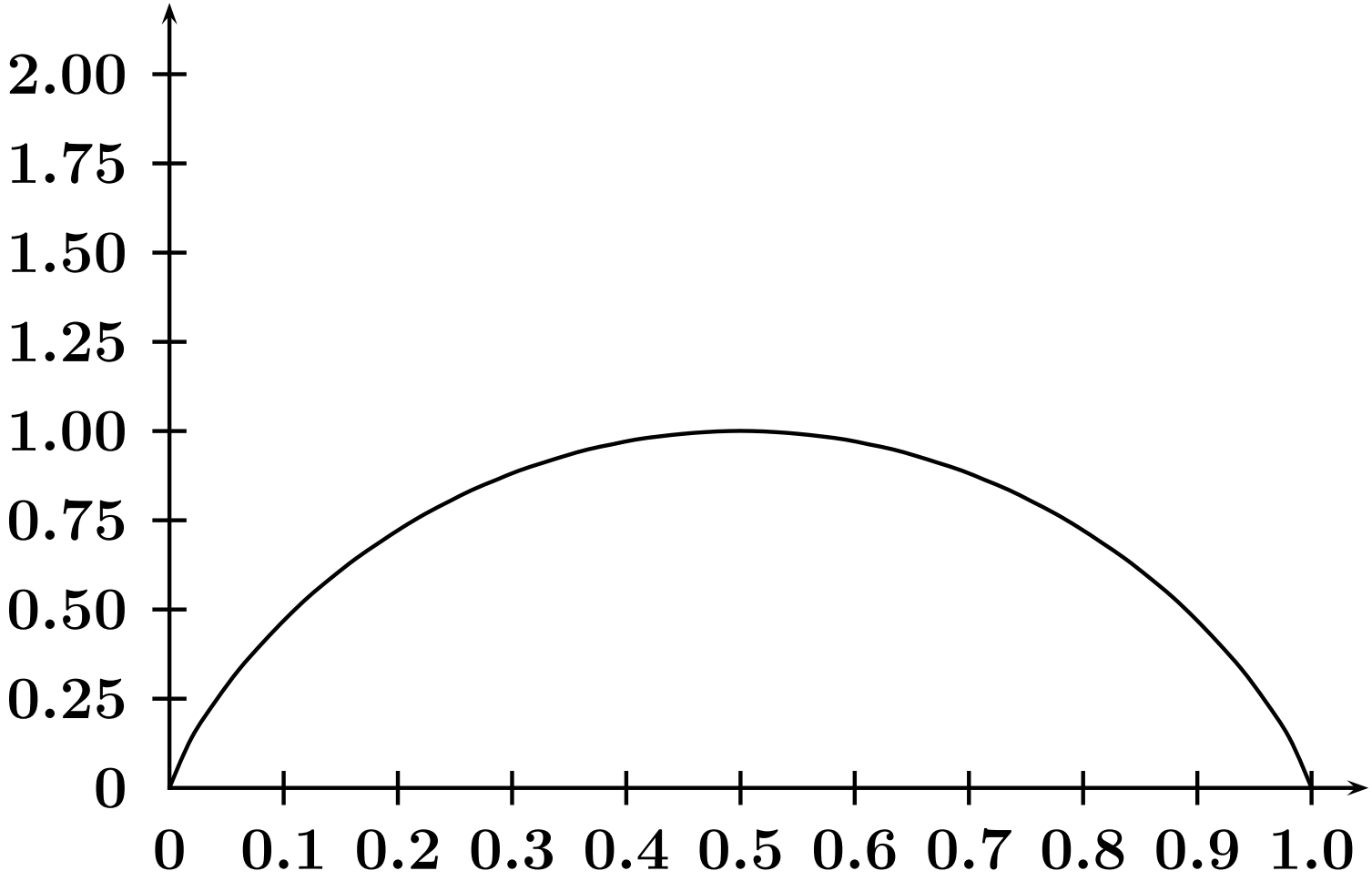
In all cases the code words are created using the ideal code wordlengths $l_C^*(x)$.

The code C_{mix} is make using the mixed (weighted) probabilities

$$p_{\text{mix}}(x) = \frac{p(x|\theta_1) + p(x|\theta_2)}{2}$$

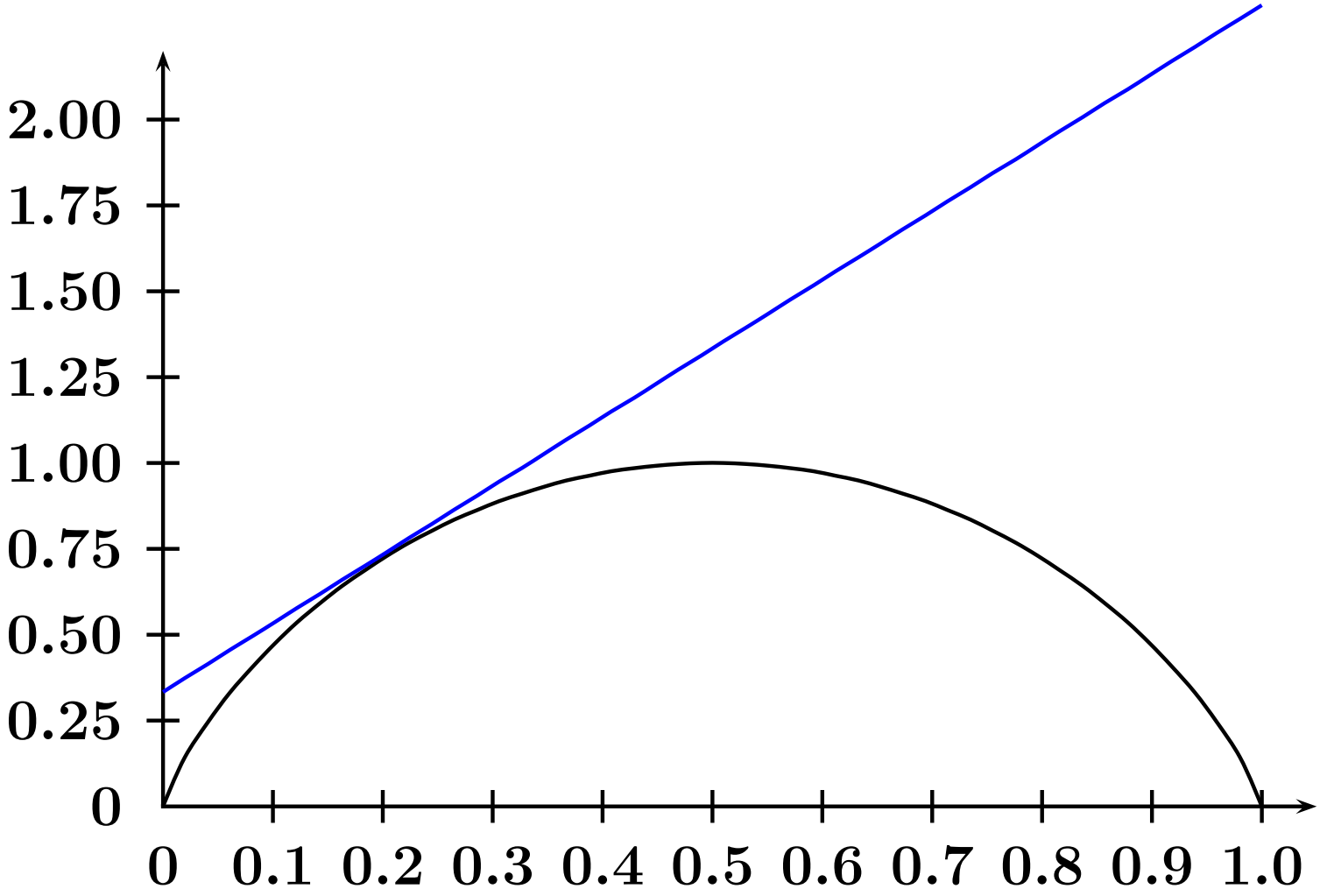
Universal data compression

The results for block length $n = 6$ are shown graphically.



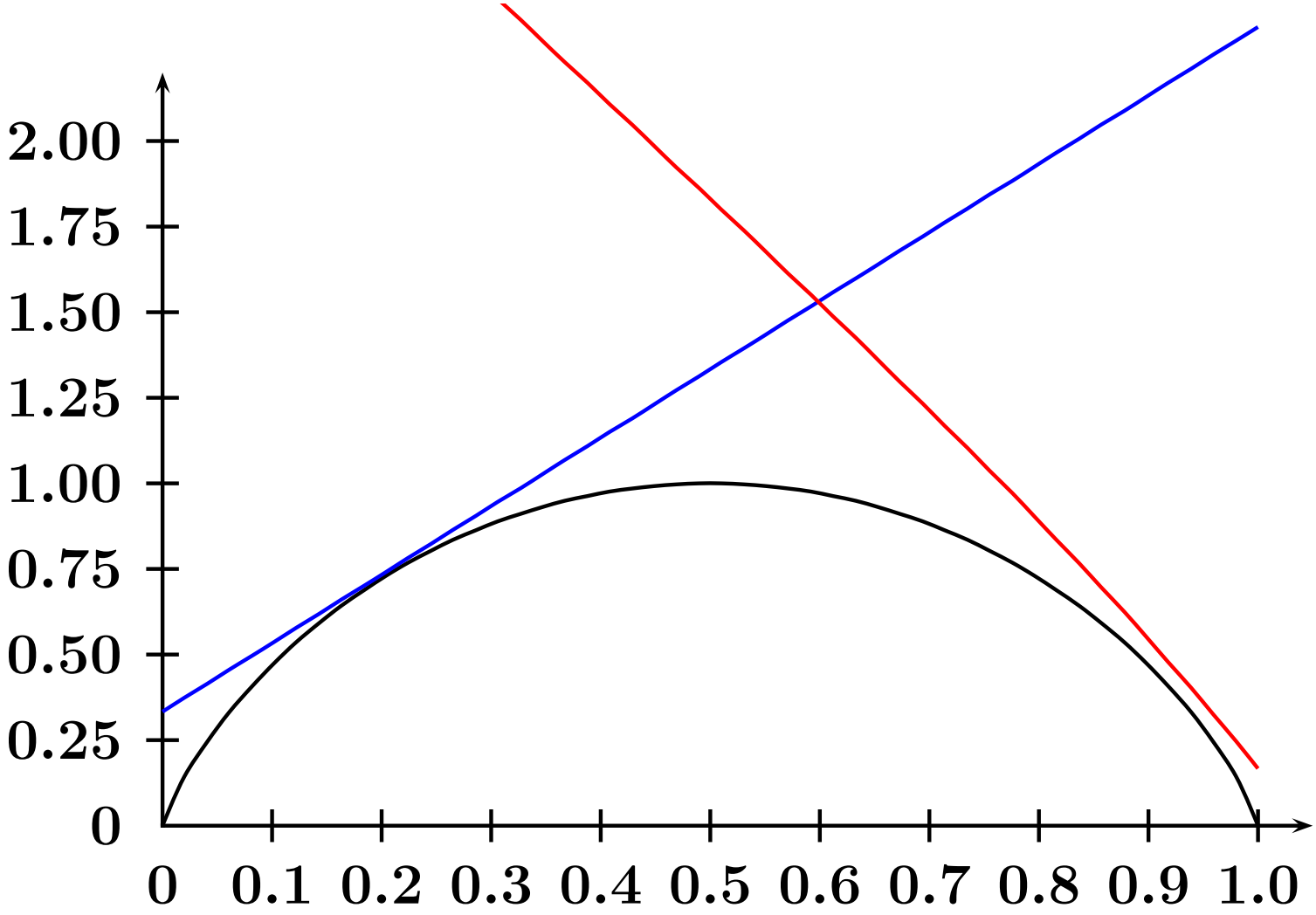
Universal data compression

The results for block length $n = 6$ are shown graphically.



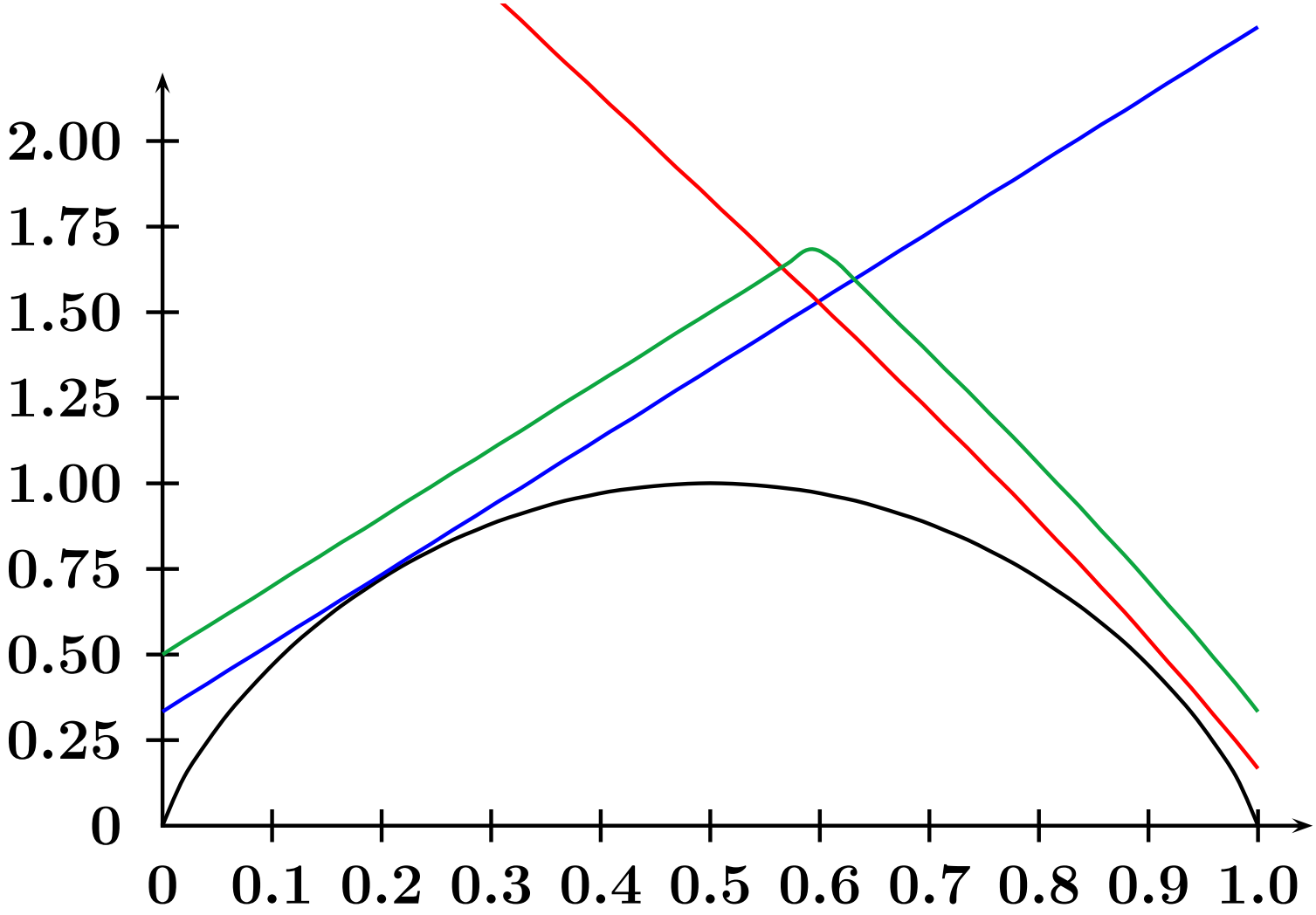
Universal data compression

The results for block length $n = 6$ are shown graphically.



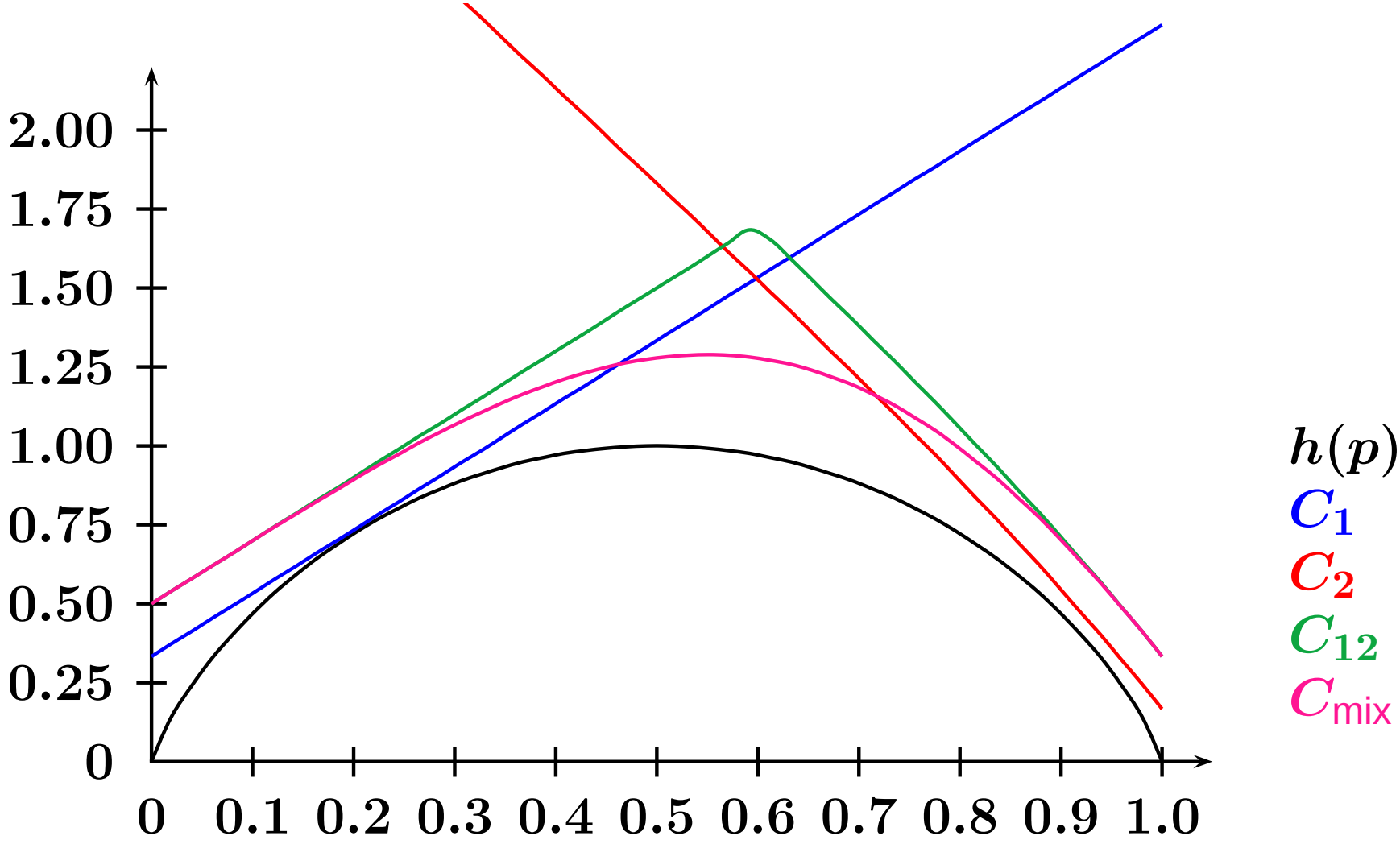
Universal data compression

The results for block length $n = 6$ are shown graphically.



Universal data compression

The results for block length $n = 6$ are shown graphically.



Universal data compression

We conclude that

- Using an ordinary source code only works (well) if we are accurate in predicting the source probabilities.

Universal data compression

We conclude that

- Using an ordinary source code only works (well) if we are accurate in predicting the source probabilities.
- That a two-part code works for more than one source. First part: description of the source (parameters). Second part: the compressed version of the sequence assuming the given source.

Universal data compression

We conclude that

- Using an ordinary source code only works (well) if we are accurate in predicting the source probabilities.
- That a two-part code works for more than one source. First part: description of the source (parameters). Second part: the compressed version of the sequence assuming the given source.
- Mixing (weighting) probabilities works at least as good as the two-part code and can be performed in one run through the data.

Universal data compression

Theorem 4 *[Optimal number of sources] For a sequence x^n generated by an binary i.i.d. source with unknown $\Pr\{X = 1\} = \theta$ the optimal number of alternative sources is of order $\sqrt{n}$ and the achieved redundancy of the resulting code C^*, relative to any i.i.d. source, is bounded as*

$$r_n(C^*) < \frac{\log_2 n}{2n}(1 + \epsilon),$$

and also

$$r_n(C^*) > \frac{\log_2 n}{2n}(1 - \epsilon),$$

for any $\epsilon > 0$ and n sufficiently large.

We shall prove this theorem in the appendix and sketch it here in several steps.

Universal data compression

Select m numbers θ_i such that

$$0 < \theta_1 < \theta_2 < \cdots < \theta_m < 1$$

Every θ_i defines a sequence probability as

$$p(x^n|\theta_i) = (1-\theta_i)^{N(0|x^n)}\theta_i^{N(1|x^n)}$$

The code word for x^n can now be made by

- describing the θ_i used: needs $< \log_2 m + 1$ bits
- producing a word with ideal length: needs $< -\log_2 p(x^n|\theta_i) + 1$ bits

Universal data compression

So,

$$l_C(x^n) < \min_{i=1,\ldots,m} -\log_2 p(x^n|\theta_i) + \log_2 m + 2$$

Note that this is a (sort of) maximum likelihood code!

Assume that the actual source probability is θ. We write for the expected redundancy (unnormalized)

$$\begin{aligned} nr_n &= \mathbf{E}\{l_C(X^n)\} - H_\theta(X^n) \\ &< \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \min_i \log_2 \frac{1}{p(x^n|\theta_i)} - \\ &\quad \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \log_2 \frac{1}{p(x^n|\theta)} + \log_2 m + 2 \end{aligned}$$

Universal data compression

After some calculations we find the bound

$$nr_n < n \min_i d(\theta \| \theta_i) + \log_2 m + 2.$$

Universal data compression

Lemma 1 *[Quadratic divergence bound] For all* p *and* q *with* $0 \leq p, q \leq 1$ *holds*

$$d(p\|q) \leq \log_2 e \frac{(p-q)^2}{q(1-q)}$$

Universal data compression

Lemma 1 *[Quadratic divergence bound] For all p and q with $0 \leq p, q \leq 1$ holds*

$$d(p\|q) \leq \log_2 e \frac{(p-q)^2}{q(1-q)}$$

Proof: Use the log inequality $\log_2 x \leq \log_2 e(x-1)$.

$$\begin{aligned} d(p\|q) &= (1-p)\log_2 \frac{1-p}{1-q} + p \log_2 \frac{p}{q} \\ &\leq \log_2 e \left[(1-p)\left(\frac{1-p}{1-q} - 1\right) + p\left(\frac{p}{q} - 1\right)\right] \\ &= \log_2 e \frac{(p-q)^2}{q(1-q)} \end{aligned}$$

Universal data compression

Lemma 2

$$\min_{i=1,\ldots,m} d(\theta\|\theta_i) \leq \frac{4\log_2 e}{m^2}$$

The (long) proof is given in the appendix. The essential ingredient is a quadratic spacing of the probabilities θ_i.

$$\theta_i = \begin{cases} \frac{2i^2}{m^2} & \text{for } i = 1, 2, \ldots, \frac{m-1}{2} \\ \frac{1}{2} & \text{for } i = \frac{m+1}{2} \\ 1 - \theta_{m+1-i} & \text{for } i = \frac{m+3}{2}, \ldots, m \end{cases}$$

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$

Major terms are: source mismatch cost

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$

Major terms are: model (parameter) description

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$

Major terms are: source mismatch cost and model parameter cost

Universal data compression

The best possible choice for α is: $\alpha = \frac{1}{2}$.

The smaller we select α, the lesser the cost we pay for the model (parameters).

However, if $\alpha < \frac{1}{2}$ then the source mismatch cost decreases essentially slower than the model cost, thus increasing the overall cost!

We finally obtain

$$r_n \leq \frac{\log_2 n}{2n}(1 + \epsilon(n))$$

where $\epsilon(n) \stackrel{n\to\infty}{\longrightarrow} 0$.

This proves the first part of Theorem 4

Universal data compression

Now we must show the second part of Theorem 4.

C is a prefix-free and complete code for binary sequences x^n.
Define the dyadic probabilities as

$$Q_C(x^n) = 2^{-l_C(x^n)}$$

We consider the expected redundancy of C used on sequences from an i.i.d. binary source with parameter θ.

$$\begin{aligned} r_n(C,\theta) &= \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \left(l_C(x^n) + \log_2 p(x^n|\theta)\right) \\ &= \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q_C(x^n)} \end{aligned}$$

Universal data compression

True fact: $\max_x f(x) \geq \mathbf{E}\{f(X)\}$.

Let $w(\theta)$ be an arbitrary distribution over $\theta \in [0, 1]$.

$$\max_{0 \leq \theta \leq 1} r_n(C, \theta) \geq \frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} w(\theta) p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q_C(x^n)} \, d\theta$$

We can still choose the best possible code C. We allow all probabilities for $Q_C(x^n)$ and get another lower bound.

$$\min_C \max_{0 \leq \theta \leq 1} r_n(C, \theta) \geq \min_{Q(x^n)} \max_{0 \leq \theta \leq 1} r_n(Q, \theta) \geq$$

$$\min_{Q(x^n)} \frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} w(\theta) p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q(x^n)} \, d\theta$$

Universal data compression

We can solve this minimization! (some steps omitted)

$$\min_{Q(x^n)} \frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} w(\theta) p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q(x^n)} \, d\theta$$

$$= \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} \bar{p}(x^n) \log_2 \frac{1}{\bar{p}(x^n)} - \int_0^1 w(\theta) h(\theta) \, d\theta$$

Universal data compression

With uniform $w(\theta) = 1$ we find

$$\bar{p}(x^n) = \frac{N(0|x^n)!N(1|x^n)!}{(n+1)!}$$

$$\int_0^1 w(\theta)h(\theta)\,d\theta = \frac{\log_2 e}{2}$$

We know an approximation of the best possible code!

Using this we can now compute a lower bound to the redundancy (complex).

Universal data compression

We start anew:

$$\min_{C} \max_{0 \le \theta \le 1} r_n(C, \theta) \ge$$
$$\frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{\bar{p}(x^n)} \, d\theta$$

Now we make use of Stirling's approximation

$$\sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} < n! < \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$$

Universal data compression

we obtain (after some appropriate approximations)

$$\bar{p}(x^n) < \sqrt{\frac{\pi}{2n}} e^{\frac{1}{3n}} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k}$$

and so

$$\log_2 \frac{p(x^n|\theta)}{\bar{p}(x^n)} > \frac{1}{2}\log_2 n - \frac{1}{2}\log_2 \frac{\pi}{2} - \frac{\log_2 e}{3n} - nd(\frac{k}{n}||\theta)$$

Universal data compression

Using Lemma 1 we find (for the $d(\frac{k}{n}||\theta)$ part)

$$\sum_{k=0}^{n} \binom{n}{k} \theta^k (1-\theta)^{n-k} \frac{(\frac{k}{n}-\theta)^2}{\theta(1-\theta)} = \frac{\log_2 e}{n}$$

Because k has a Bernoulli distribution with mean $\mu_k = n\theta$ and variance $\sigma_k^2 = n\theta(1-\theta)$ we find

$$= \frac{\log_2 e}{n^2\theta(1-\theta)} \sum_{k=0}^{n} \binom{n}{k} \theta^k (1-\theta)^{n-k} (k-n\theta)^2$$

$$= \frac{\sigma_k^2 \log_2 e}{n^2\theta(1-\theta)} = \frac{\log_2 e}{n}$$

Universal data compression

$$\min_{C} \max_{0 \le \theta \le 1} r_n(C, \theta) > \frac{\log_2 n}{2n} - \frac{\log_2 \frac{\pi}{2}}{n} - \frac{\log_2 e}{n^2} - \frac{\log_2 e}{n}$$

$$= \frac{\log_2 n}{2n} (1 - \epsilon(n))$$

where

$$\epsilon(n) = \frac{2 \log_2 \frac{\pi}{2}}{\log_2 n} + \frac{2 \log_2 e}{n \log_2 n} + \frac{2 \log_2 e}{\log_2 n}$$

$$\epsilon(n) \stackrel{n \to \infty}{\longrightarrow} 0.$$

And this, finally, proves the second part of Theorem 4

Universal data compression

Discussion:

For the binary i.i.d. source which is described by one parameter θ, the optimal redundancy is $\frac{\log_2 n}{2n}$.

Universal data compression

Discussion:

For the binary i.i.d. source which is described by one parameter θ, the optimal redundancy is $\frac{\log_2 n}{2n}$.

This apparently is the cost we must pay for not knowing θ.

Universal data compression

Discussion:

For the binary i.i.d. source which is described by one parameter θ, the optimal redundancy is $\frac{\log_2 n}{2n}$.

This apparently is the cost we must pay for not knowing θ.

Or, as in Kolmogorov's minimal sufficient statistic, this is the complexity of the description of the model.

Universal data compression

Discussion:

For the binary i.i.d. source which is described by one parameter θ, the optimal redundancy is $\frac{\log_2 n}{2n}$.

This apparently is the cost we must pay for not knowing θ.

Or, as in Kolmogorov's minimal sufficient statistic, this is the complexity of the description of the model.

It also indicates that the number of discernible sources is roughly $\sqrt{n}$ in this case.

Universal data compression

Discussion:

For the binary i.i.d. source which is described by one parameter θ, the optimal redundancy is $\frac{\log_2 n}{2n}$.

This apparently is the cost we must pay for not knowing θ.

Or, as in Kolmogorov's minimal sufficient statistic, this is the complexity of the description of the model.

It also indicates that the number of discernible sources is roughly $\sqrt{n}$ in this case.

The next result will explain some of these observations.

Redundancy-capacity theorem

In the derivation of the second part of Theorem 4 we can actually find an explanation why we have redundancy when compressing data from parametrized sources.

We again take a Bayesian approach.

Let $\mathcal{Q}_{\mathcal{C}}$ be the set of all dyadic probabilities and $\mathcal{Q}$ be the set of all probabilities.
$\mathcal{S}$ is the set of all sources parametrized by a vector θ that takes values in a parameter space Θ.
We have seen the example of the binary i.i.d. source with a one dimensional parameter $\theta = \mathbf{Pr}\{X = 1\}$ and $\Theta = [0, 1]$.

Redundancy-capacity theorem

If $Q_C \in \mathcal{Q}_\mathcal{C}$ then the redundancy of of the corresponding code C is given by

$$\begin{aligned} r &= \sum_{x \in \mathcal{X}} p(x|\theta) \log_2 \frac{p(x|\theta)}{Q_C(x)} \\ &= D(p(X|\theta) \| Q_C(X)) \end{aligned}$$

Let $w(\theta)$ be a prior distribution over θ. The Bayes redundancy is given by

$$\mathcal{D}(w; Q_C) = \int_\Theta D(p(X|\theta) \| Q_C(X)) w(\theta)\, d\theta$$

Redundancy-capacity theorem

If again we allow all probabilities, not only dyadic ones, we obtain:

$$\mathcal{D}(w;Q) = \int_{\Theta} w(\theta) D(p(X|\theta)\|Q(X))\, d\theta$$

Redundancy-capacity theorem

If again we allow all probabilities, not only dyadic ones, we obtain:

$$\mathcal{D}(w;Q) = \int_{\Theta} w(\theta) D(p(X|\theta)\|Q(X))\, d\theta$$

$$= \int_{\Theta} \sum_{x\in\mathcal{X}} w(\theta) p(x|\theta) \log_2 \frac{p(x|\theta)}{Q(x)}\, d\theta$$

Channel input θ probabilities $w(\theta)$

Redundancy-capacity theorem

If again we allow all probabilities, not only dyadic ones, we obtain:

$$\mathcal{D}(w;Q) = \int_{\Theta} w(\theta) D(p(X|\theta)\|Q(X))\, d\theta$$

$$= \int_{\Theta} \sum_{x\in\mathcal{X}} w(\theta) p(x|\theta) \log_2 \frac{p(x|\theta)}{Q(x)}\, d\theta$$

Channel transition probabilities $p(x|\theta)$

Redundancy-capacity theorem

If again we allow all probabilities, not only dyadic ones, we obtain:

$$\mathcal{D}(w;Q) = \int_{\Theta} w(\theta) D(p(X|\theta)\|Q(X))\, d\theta$$

$$= \int_{\Theta} \sum_{x \in \mathcal{X}} w(\theta) p(x|\theta) \log_2 \frac{p(x|\theta)}{Q(x)}\, d\theta$$

Channel output x probabilities $Q(x)$

Redundancy-capacity theorem

If again we allow all probabilities, not only dyadic ones, we obtain:

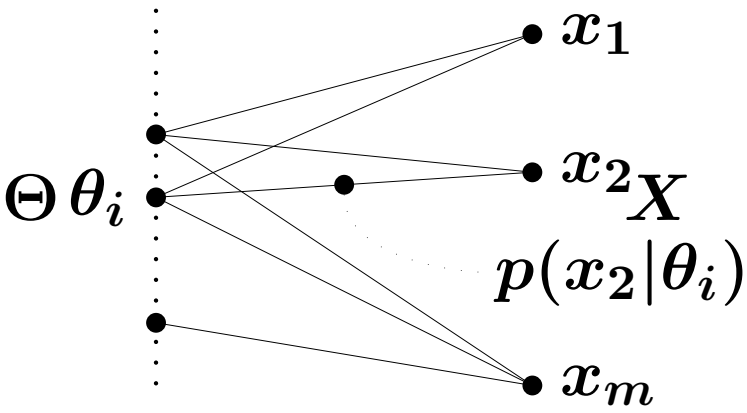
$$\begin{aligned}\mathcal{D}(w;Q) &= \int_{\Theta} w(\theta) D(p(x|\theta)\|Q(X))\, d\theta \\ &= \int_{\Theta} \sum_{x\in\mathcal{X}} w(\theta) p(x|\theta) \log_2 \frac{p(x|\theta)}{Q(x)}\, d\theta \\ &= I(\theta;X)\end{aligned}$$

Redundancy-capacity theorem

We can maximize over all possible priors $w(\theta)$ and find

$$r \geq \max_{w(\theta)} I(\theta; X)$$

So the redundancy is lower bounded by (actually it is equal to) the capacity of the channel from the source parameters to the source output sequence x.



Redundancy: learning source parameters from data

- Source coding: we don't want this.
- Machine learning: this is what's it about.

The meaning of model information

Efficient description of data can be split into two parts:

- Information about the ‘model’

  Kolmogorov minimal sufficient statistic: The complexity of the smallest set describing x.

  Universal compression redundancy: The description of the parameters of the data generating process.

- Selection of one of the ’possible’ sequences.

  Kolmogorov: The structure function. One out of $|\mathcal{X}|$ selected and described with $\log_2 |\mathcal{X}|$ bits.

  Universal compression: Type class (AEP). One out of $|T(P_x)|$ selected and described with $nH(P_x)$ bits.

The meaning of model information

The first part describes what the model ‘can do’.

bits of π: Almost zero complexity. The model is easy to describe and can only generate this sequence. Easy to predict bits.

bits from an i.i.d. source $\theta = \frac{1}{2}$: Highly complex. The model is very simple but the set of possible sequences is large. Hard to predict bits.

The meaning of model information

Occam's razor:

> *One should not increase, beyond what is necessary, the number of entities required to explain anything.*

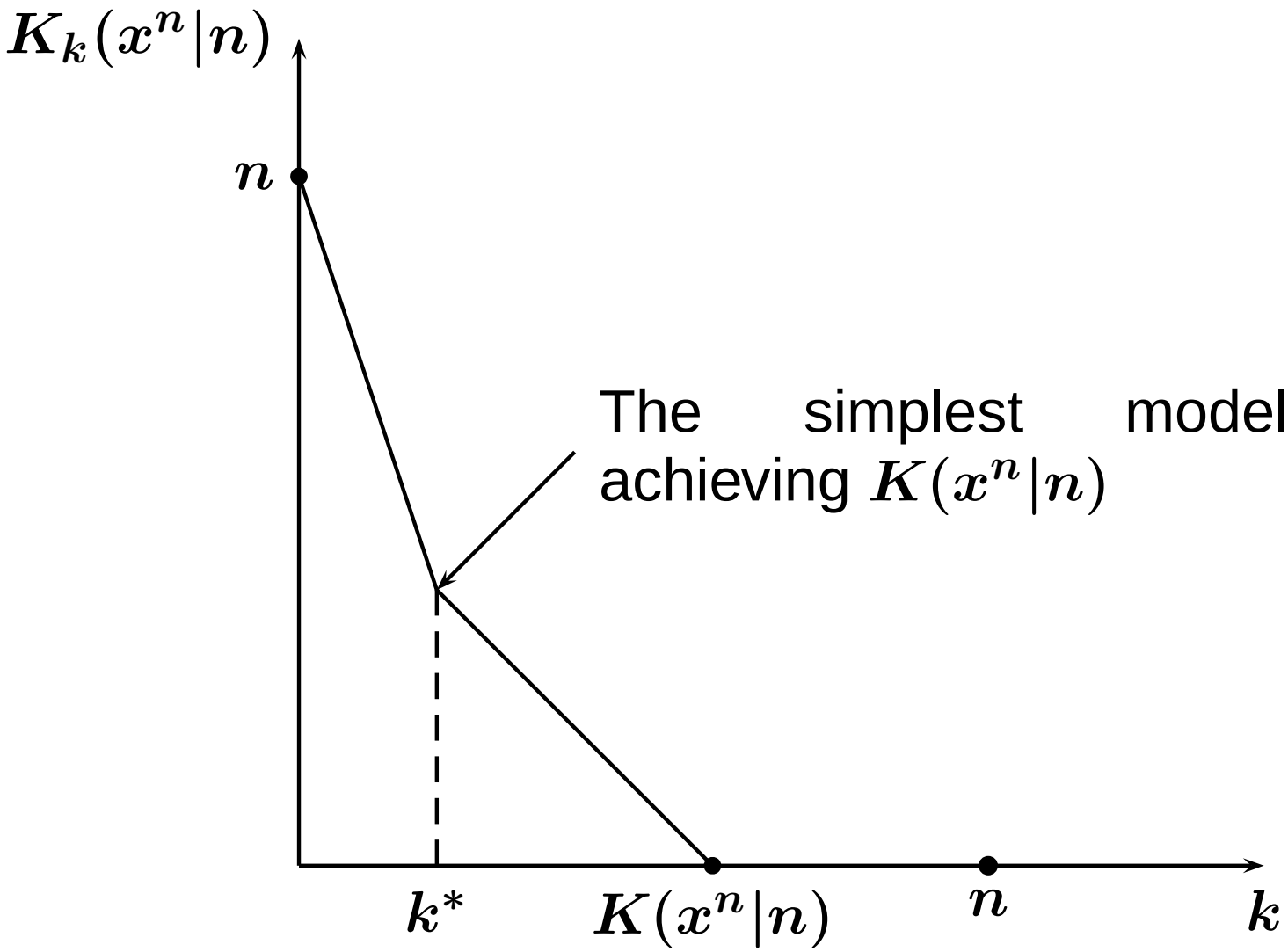
The most useful statement of the principle for scientists is:

> *When you have two competing theories which make exactly the same predictions, the one that is simpler is the better.*

Kolmogorov, Universal source coding:

> *Take the simplest model that describes your data.*

The meaning of model information



Part C

Bayesian model estimation

and the
Context-tree model selection

Contents

- Bayesian model estimation
  - i^{th}-order markov model
  - Likelihood ratio relative to 'actual' model
  - Selection criterion
- Terminology
- Stochastic complexity
- Coding for Gaussians

Contents (ctd.)

- Context trees
- Model posterior for Context trees
- Summary
- Exam requirements

Bayesian model estimation

Example 10: [Revisit first lecture]

Let $\mathcal{M}_i$ be the i-th order binary Markov model (source).

Then $\Theta_i = [0,1]^{2^i}$.

Beta distribution for prior $p(\theta_i|\mathcal{M}_i)$, with $\alpha = \beta = \frac{1}{2}$ (Jeffreys prior).

$$p(\theta_i|\mathcal{M}_i) = \left(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\right)^{2^i} \prod_{s\in\{0,1\}^i} \theta_{i,s}^{\alpha-1}(1-\theta_{i,s})^{\beta-1}$$

$$= \frac{1}{\pi^{2^i}} \prod_{s\in\{0,1\}^i} \theta_{i,s}^{-1/2}(1-\theta_{i,s})^{-1/2}$$

Bayesian model estimation

$$p(x^n|\mathcal{M}_i) = \int_{\Theta_i} p(\theta_i|\mathcal{M}_i)p(x^n|\mathcal{M}_i,\theta_i)\,d\theta_i$$

$$= \frac{1}{\pi^{2^i}} p(x_1,\ldots,x_i)$$

$$\int_{\Theta_i} \prod_{s\in\{0,1\}^i} \theta_{i,s}^{N(s1|x^n)-1/2}(1-\theta_{i,s})^{N(s0|x^n)-1/2}\,d\theta_i$$

$$= \frac{p(x_1,\ldots,x_i)}{\pi^{2^i}}$$

$$\prod_{s\in\{0,1\}^i} \int_{[0,1]} \theta_{i,s}^{N(s1|x^n)-1/2}(1-\theta_{i,s})^{N(s0|x^n)-1/2}\,d\theta_{i,s}$$

$$= p(x^i) \prod_{s\in\{0,1\}^i} \frac{\Gamma(N(s0|x^n)+\frac{1}{2})\Gamma(N(s1|x^n)+\frac{1}{2})}{\pi\Gamma(N(s0|x^n)+N(s1|x^n)+1)}$$

Bayesian model estimation

So we must study the behaviour of

$$P_e(x^n) = P_e(a,b) = \frac{\Gamma(a+\frac{1}{2})\Gamma(b+\frac{1}{2})}{\pi\Gamma(n+1)}$$

$$a = N(0|x^n)$$

$$b = N(1|x^n)$$

It is a memoryless sub-sources of the Markov source. x^n is generated i.i.d. with parameter θ.

The actual probability of x^n under this source is

$$p(x^n|\mathcal{M},\theta) = (1-\theta)^a\theta^b$$

Bayesian model estimation

We can write

$$P_e(a,b) = \frac{\frac{1}{2}\frac{3}{2}\cdots(a-\frac{1}{2})\cdot\frac{1}{2}\frac{3}{2}\cdots(b-\frac{1}{2})}{(a+b)!}$$

Again with the help of Stirling's approximation we can derive, for large a and b the following. (Excercise). Note: $a+b=n$.

$$\log_2 \frac{p(x^n|\mathcal{M},\theta)}{P_e(a,b)} \leq \frac{1}{2}\log_2 n + \frac{1}{2}\log_2 \frac{\pi}{2}$$

Actually, we can prove that for all $a \geq 0$ and $b \geq 0$

$$\log_2 \frac{p(x^n|\mathcal{M},\theta)}{P_e(a,b)} \leq \frac{1}{2}\log_2 n + 1$$

Bayesian model estimation

Back to the i-th order Markov source.

$$p(x^n|\mathcal{M}_i,\theta_i) = p(x^i) \prod_{s\in\{0,1\}^i} \theta_{i,s}^{N(s1|x^n)}(1-\theta_{i,s})^{N(s0|x^n)}$$

$$p(x^n|\mathcal{M}_i) = p(x^i) \prod_{s\in\{0,1\}^i} P_e(N(s1|x^n),N(s0|x^n))$$

Bayesian model estimation

With the previous bound we find

$$\log_2 \frac{p(x^n|\mathcal{M}_i,\theta_i)}{p(x^n|\mathcal{M}_i)} =$$

$$\log_2 \frac{p(x^i)\prod_{s\in\{0,1\}^i}\theta_{i,s}^{N(s1|x^n)}(1-\theta_{i,s})^{N(s0|x^n)}}{p(x^i)\prod_{s\in\{0,1\}^i}P_e(N(s1|x^n),N(s0|x^n))}$$

$$= \sum_{s\in\{0,1\}^i} \log_2 \frac{\theta_{i,s}^{N(s1|x^n)}(1-\theta_{i,s})^{N(s0|x^n)}}{P_e(N(s1|x^n),N(s0|x^n))}$$

$$\leq \sum_{s\in\{0,1\}^i} \frac{1}{2}\log_2 N(s|x^n) + 1 \stackrel{*1}{\leq} \frac{2^i}{2}\log_2\frac{n-i}{2^i} + 2^i$$

(*1): here we use Jensen's inequality.

Bayesian model estimation

So we conclude that for any parameter vector θ_i we have (approximately!)

$$\log_2 p(x^n|\mathcal{M}_i) \approx \log_2 p(x^n|\mathcal{M}_i, \theta_i) - \frac{2^i}{2} \log_2 \frac{n-i}{2^i} - 2^i$$

Maximum Likelihood parameters (and $n \gg \max\{2^i, 2^j\}$)

$$\log_2 \frac{p(\mathcal{M}_i|x^n)}{p(\mathcal{M}_j|x^n)} \approx \log_2 \frac{p(\mathcal{M}_i)}{p(\mathcal{M}_j)} + \log_2 \frac{p(x^n|\mathcal{M}_i, \hat{\theta}_i)}{p(x^n|\mathcal{M}_j, \hat{\theta}_j)} - \frac{2^i - 2^j}{2} \log_2 n$$

So, again we observe the parameter cost!

Terminology

The two-part description separates model information from random selection

Kolmogorov: Minimal sufficient statistic selects the smallest model such that model complexity plus random selection is still optimal.

Universal coding: There is a certain unavoidable cost for parameters in a model. It is the price for learning the parameters.

Distinguishable models (parameter values): For a sequence of length n we can use (selection or weighting) about $\sqrt{n}$ distinct values.

Occam's razor: Take the simplest explanation that explains the observations.

Terminology

This results in the notion of stochastic complexity

$$-\log_2 P(x^n|\mathcal{M})$$

$$P(x^n|\mathcal{M}) = \frac{P(x^n|\mathcal{M}, \hat{\theta}(x^n))}{\sum_{x^n \in \mathcal{X}^n} P(x^n|\mathcal{M}, \hat{\theta}(x^n))}$$

is known as the NML (Normalized Maximum Likelihood). That is must be normalized is reasonable because

$$\sum_{x^n \in \mathcal{X}^n} P(x^n|\mathcal{M}, \hat{\theta}(x^n)) \geq 1$$

And the normalizing constant determines the model cost. Note that we assume here that the model priors $P(\mathcal{M})$ are all equal!

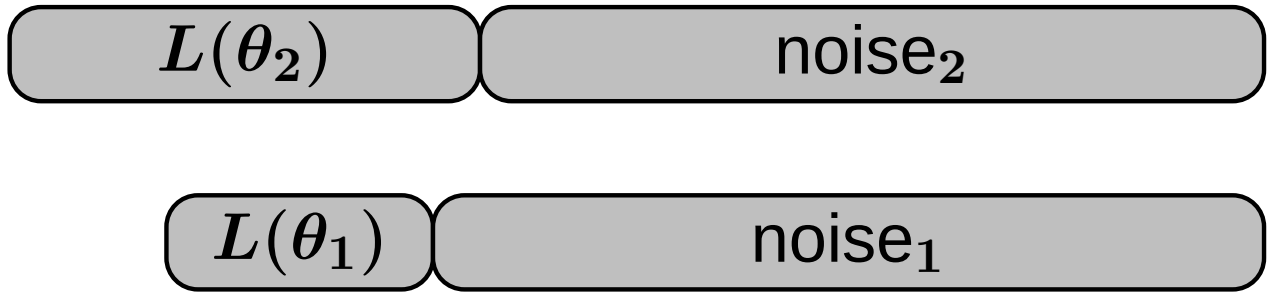
Terminology

Suppose I have two model classes, $\mathcal{M}_1$ and $\mathcal{M}_2$, for my data x^n and the stochastic complexity $-\log_2 P(x^n|\mathcal{M}_1)$ is smaller than $-\log_2 P(x^n|\mathcal{M}_2)$.
Because the model information part is proportional to $\log_2 n$ and the "noise" part is proportional to n, a smaller complexity means "less noise". So $\mathcal{M}_1$ explains more of the data.
This leads to the Minimum Description Length Principle.

The best model for the data is is the model that results in the smallest stochastic complexity.

Terminology

Stochastic complexity $\approx$ ideal codeword length.
Coding interpretation: $L(\theta) = O(\log n)$; $L(\text{noise}) = O(n)$



Terminology

Stochastic complexity $\approx$ ideal codeword length.
Coding interpretation: $L(\theta) = O(\log n)$; $L(\text{noise}) = O(n)$

Stochastic Complexity (MDL)

"Real data model": binary 1^{th} order Markov,

$\boldsymbol{\theta_0 = \Pr\{X_i = 1 | x_{i-1} = 0\} = \frac{1}{4}}$,

$\boldsymbol{\theta_1 = \Pr\{X_i = 1 | x_{i-1} = 1\} = \frac{1}{2}}$

Then: $\boldsymbol{\Pr\{X_i = 1\} = \frac{1}{3}}$.

$\boldsymbol{\mathcal{M}_1}$ is i.i.d. with $\boldsymbol{\hat{\theta}_1 \approx \frac{1}{3}}$.

$\boldsymbol{\mathcal{M}_2}$ is 1^{th} order Markov with $\boldsymbol{\hat{\theta}_2 \approx (\frac{1}{4}, \frac{1}{2})}$.

$\boldsymbol{H(X|\mathcal{M}_1, \hat{\theta}_1) = 0.918}$ bit.

$\boldsymbol{H(X|\mathcal{M}_2, \hat{\theta}_2) = 0.874}$ bit.

Stochastic Complexity (MDL)

Stochastic complexity

$$S.C._1 \sim \frac{\log_2 n}{2} + 0.918n.$$
$$S.C._2 \sim \log_2 n + 0.874n.$$

For $n < 70 : S.C._1 < S.C._2$ and for
$n > 70 : S.C._1 > S.C._2$.

So if there is not enough data the MDL selects a smaller model than the "true" model.

This is good!

There is not enough data to estimate properly a complex model.

Coding for Gaussians

Designing a code for a Gaussian (real value) is not trivial.

Probability distributions in stead of probabilities!

- Entropy: Differential entropy given pdf $p(x)$.

$$\mathcal{H}(X) = -\int p(x) \log_2 p(x)\, dx$$

- Interpretation: Not the length of optimal code!

Coding for Gaussians

- Examples
  - Uniform $p(x) = 1/a$ for $x \in [0, a]$ and zero elsewhere.

$$\mathcal{H}(X) = -\int_0^a \frac{1}{a} \log_2 \frac{1}{a}\, dx = \log_2 a.$$

    Note: $\mathcal{H}(X)$ can be negative ($a < 1$).
  - Gaussian

$$\mathcal{H}(X) = \frac{1}{2} \log_2 2\pi e\sigma^2$$

Coding for Gaussians

- Discrete entropy for real variables.
  - Mean value theorem: There exists a value $x_i \in [i\Delta, (i+1)\Delta]$ such that

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

  - Quantized random variable $X_\Delta = x_i$ if $i\Delta \leq X < (i+1)\Delta$.

Coding for Gaussians

- Discrete entropy for real variables (cont.)
  - Entropy:

$$
\begin{aligned}
H(X_\Delta) &= -\sum_{i=-\infty}^{\infty} p(x_i)\Delta \log_2(p(x_i)\Delta) \\
&= -\sum_{i=-\infty}^{\infty} p(x_i)\Delta \log_2 p(x_i) - \log_2 \Delta \\
&\rightarrow \mathcal{H}(X) - \log_2 \Delta \text{ as } \Delta \rightarrow 0
\end{aligned}
$$

e.g. $\Delta = 2^{-k}$, then $H(X_\Delta)$ is k bits larger than $\mathcal{H}(X)$.

Coding for Gaussians

How to code? (sketch)

1. Take ML estimates for

$$\text{mean } \hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_i \boldsymbol{x}_i$$

$$\text{variance } \hat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_i (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^T$$

2. Quantize with precision $\frac{1}{\sqrt{n}}$. (Works because error in ML estimate is approximately $1/\sqrt{n}$.)
   Then transmit these in $\frac{1}{2}\log_2 n$ bits per parameter.
3. Then encode a quantized version of the data (using $\mathcal{H}(X) - \log_2 \Delta$ bits on the average.

Coding for Gaussians

For model selection and other MDL tasks we use different models but encode with the same precision $\boldsymbol{\Delta}$ so we still compare and select using the parameter cost ($\frac{1}{2}\log_2 n$) and the resulting (differential) entropy.

Warning: This procedure depends completely on the fact that the ML estimates converge to the proper value with an error of $1/\sqrt{n}$.

This holds for parameters such as $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\Sigma}}$, and $\hat{\boldsymbol{p}}$ for the discrete models.

It does not work for e.g. the parameter $\boldsymbol{k}$ of an $\boldsymbol{k}^{\text{th}}$ order Markov process.

Coding for Gaussians

$$\mathcal{M}_0 = \{\mathcal{N}(0,1)\}; \quad \mathcal{M}_1 = \{\mathcal{N}(\theta,1)|\theta\}$$

We look at the (approximated) code wordlength (the Stochastic complexity)

$$S.C. = -\log_2 p(x^n|\hat{\theta}) + \frac{k}{2}\log_2 n$$

$$\mathcal{M}_0 : k = 0; (\theta_0 = 0)$$

$$S.C._0 = \frac{1}{2}\sum_i x_i^2 + \frac{n}{2}\log_2 2\pi$$

$$\mathcal{M}_1 : k = 1; \hat{\theta}_1 = \bar{x}$$

$$S.C._1 = \frac{1}{2}\sum_i (x_i - \bar{x})^2 + \frac{n}{2}\log_2 2\pi + \frac{1}{2}\log_2 n$$

Coding for Gaussians

Comparing these two models complexities we can conclude that we will select $\mathcal{M}_0$ if

$$\bar{x}^2 \leq \frac{\log_2 n}{n}$$

Context trees

Recap: Memoryless binary source: one parameter $\theta = \Pr\{X = 1\}$

Recap: Markov order-k: one parameter per state. There are 2^k states. The k symbols $x_{i-k}, \ldots, x_{i-1}$ form the context of the symbol x_i.

Real world models: Length of context depends on its contents.
e.g. Natural language (English, Dutch, $\cdots$): if context starts with $x_{i-1} =$'q' then no more symbols are needed.

Context trees

A tree source is a nice concept to describe such sources.

A tree source consists of a set $\mathcal{S}$ of suffixes that together form a tree.
To each suffix (leaf) s in the tree there corresponds a parameter θ_s.

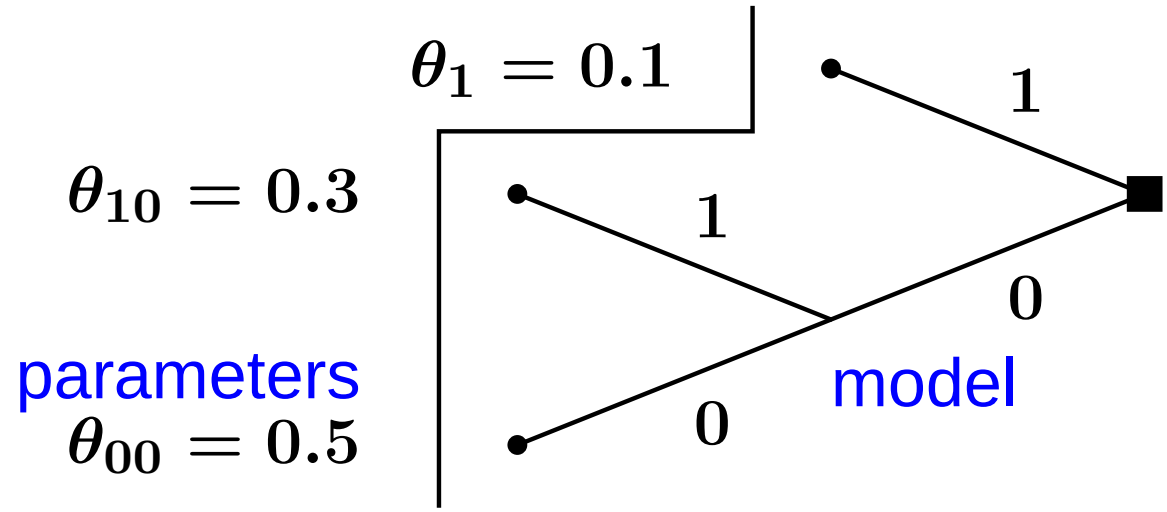
Some more notation: By $x_{|s}^n$ we denote the sub-sequence of symbols from x^n that are preceded by the sequence s.

Example: $x^8 = 01011010$; $s = 01$; then
$x_{|01}^8 = x_3 x_5 x_8 = 010$.

Context trees

Example 11: Let $\mathcal{S} \stackrel{\Delta}{=} \{00, 10, 1\}$ and $\theta_{00} = 0.5, \theta_{10} = 0.3$, and $\theta_1 = 0.1$ then

$$P_a(X_i = 1 | \cdots, x_{i-2} = 0, x_{i-1} = 0) = 0.5,$$
$$P_a(X_i = 1 | \cdots, x_{i-2} = 1, x_{i-1} = 0) = 0.3,$$
$$P_a(X_i = 1 | \cdots, x_{i-1} = 1) = 0.1.$$



Context trees

Just as before ("Bayesian model estimation") we must estimate the sequence probabilities of the memoryless subsources that correspond to the leaves of the tree (states of the source).
Let the full sequence be x^n and the subsequence for state s be written as $x^n_{|s}$. Before we wrote

$$P(x^n_{|s}) = P_e(a,b) = \frac{\Gamma(a+\frac{1}{2})\Gamma(b+\frac{1}{2})}{\pi\Gamma(a+b+1)}$$

$$a = N(1|x^n_{|s})$$

$$b = N(0|x^n_{|s})$$

Context trees

We shall now use the shorthand notation for the estimated probability of the subsequence generated in state s given the full sequence x^i:

$$P_e(a_s, b_s) = \frac{\Gamma(a_s + \frac{1}{2})\Gamma(b_s + \frac{1}{2})}{\pi\Gamma(a_s + b_s + 1)}$$

$$a_s = N(s0|x^n) = N(0|x^n_{|s})$$

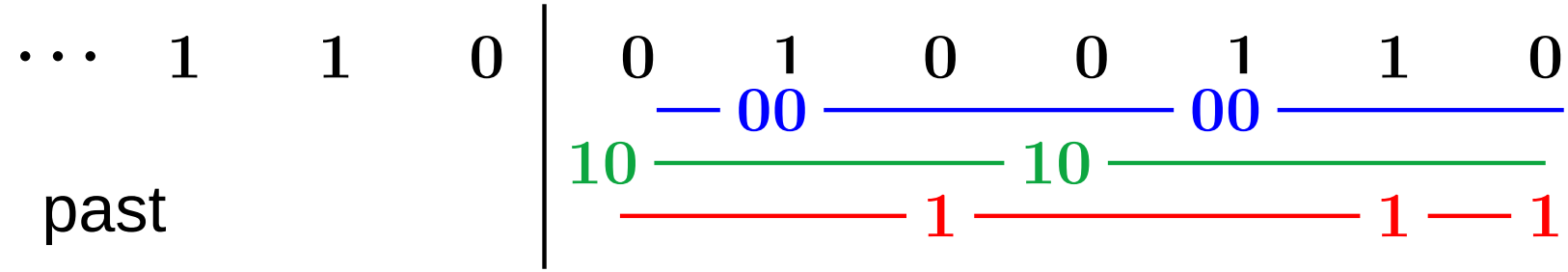
$$b_s = N(s1|x^n) = N(1|x^n_{|s})$$

So we encode using the ideal code wordlength and find

$$l_C(x^n) < -\log_2 \prod_{s \in \mathcal{S}} P_e(a_s, b_s) + 2.$$

Context trees

Example 12: Let $\mathcal{S} = \{00, 10, 1\}$.



$$P(0100110|\cdots 110) = \underbrace{P(00)}_{10}\underbrace{P(11)}_{00}\underbrace{P(010)}_{1}$$

$$= \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{1}{2}\cdot\frac{1}{4}\cdot\frac{3}{6} = \frac{9}{1024}$$

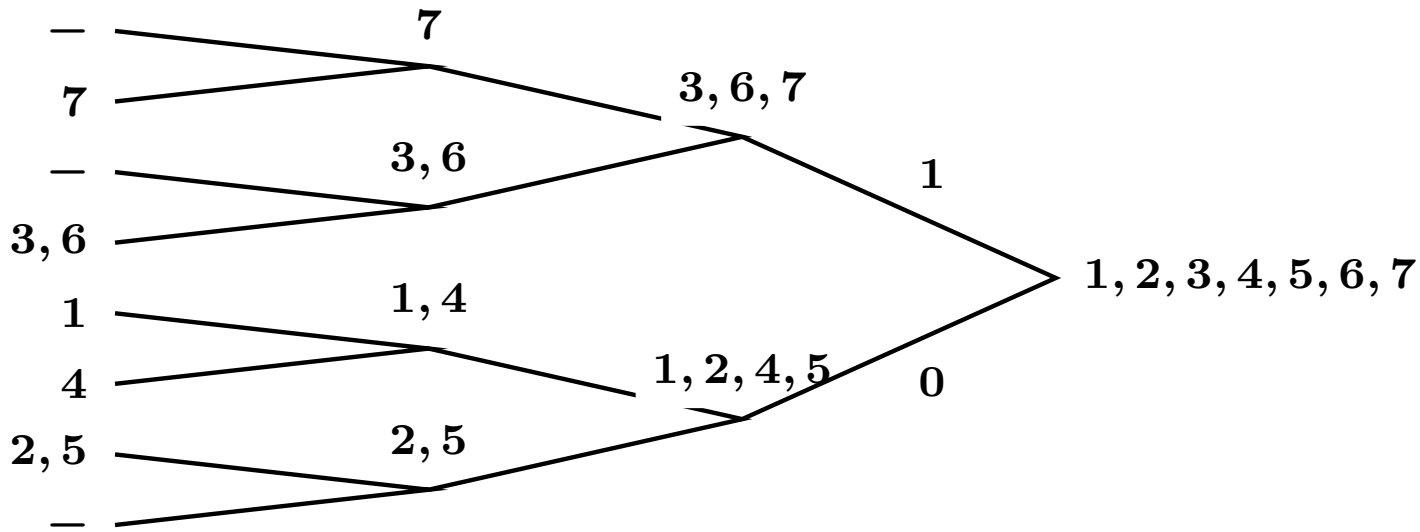
See “Bayesian Estimation”

$$\log_2 \frac{p(x^n|\mathcal{S},\theta)}{\prod_{s\in\mathcal{S}} P_e(a_s,b_s)} \leq \frac{|\mathcal{S}|}{2}\log_2\frac{N}{|\mathcal{S}|} + |\mathcal{S}|$$

Context trees

Problem: We do not know $\mathcal{S}$!

Context tree (of depth D)



$\cdots$	1	1	0	1	0	1	2	3	4	5	6	7
				0	1	0	0	1	1	0
past				source sequence									

In every node t use $a_t = N(t0|x^n)$ and $b_t = N(t1|x^n)$.

Context trees

We shall assign a probability to the subsequence $x^n_{|s}$ for every state s in the context tree.

We shall do this in such a way that in the root of the tree we assign a probability to the whole sequence x^n that is a mixture of all possible tree sources.

We use the following observations to build, recursively, this probability.

The probability we build is written as follows

$$P^s_w = P_w(x^n_{|s}),$$

where P^s_w is the shorthand notation we shall use.

Later we return to this and make the notation more precise.

Context trees

Suppose s is a leaf:
All we know are a_s and b_s so we assign the subsequence probability

$$P_w^s = P_e(a_s, b_s).$$

Now if s is an internal node, we have two options for the subsequence probability.

1: $P_e(a_s, b_s)$.
2: $P_w^{0s} P_w^{1s}$.

We must make a choice or better even, mix these options.
So we set

$$P_w^s = \frac{P_e(a_s, b_s) + P_w^{0s} P_w^{1s}}{2}.$$

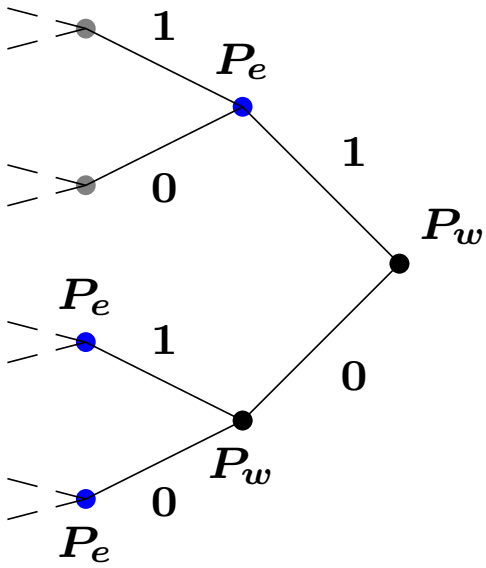
Context trees

Analysis.

Let $\mathcal{S} = \{00, 10, 1\}$ and we use a context tree with depth $D > 2$.

We look at the P_w's for different nodes.

For the nodes $s \in \mathcal{S}$ we consider (in the analysis) only the P_e's.



$$P_w^{00} \geq \frac{1}{2} P_e(a_{00}, b_{00})$$

$$P_w^{10} \geq \frac{1}{2} P_e(a_{10}, b_{10})$$

$$P_w^{1} \geq \frac{1}{2} P_e(a_{1}, b_{1})$$

Context trees

Now we consider nodes nearer to the root and take only the $P_w^{0s} P_w^{1s}$ part.

$$
\begin{aligned}
P_w^0 &\geq \frac{1}{2} P_w^{00} P_w^{10} \\
&\geq \frac{1}{8} P_e(a_{00}, b_{00}) P_e(a_{10}, b_{10}) \\
P_w^\lambda &\geq \frac{1}{2} P_w^0 P_w^1 \\
&\geq \frac{1}{32} P_e(a_{00}, b_{00}) P_e(a_{10}, b_{10}) P_e(a_1, b_1)
\end{aligned}
$$

Here λ denotes the root of the tree.

Context trees

For general trees (or suffix sets) $\mathcal{S}$ we find

$$P_w^\lambda \geq 2^{1-2|\mathcal{S}|} \prod_{s \in \mathcal{S}} P_e(a_s, b_s)$$

This results in the ideal code wordlength
$l_C(x^n) = -\log_2 P_w^\lambda$

$$l_C(x^n) \leq -\log_2 P(x^n|\mathcal{S},\theta) + 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2}\log_2 n + |\mathcal{S}| + 2.$$

Context trees

For general trees (or suffix sets) $\mathcal{S}$ we find

$$P_w^\lambda \geq 2^{1-2|\mathcal{S}|} \prod_{s \in \mathcal{S}} P_e(a_s, b_s)$$

This results in the ideal code wordlength
$l_C(x^n) = -\log_2 P_w^\lambda$

$$l_C(x^n) \leq -\log_2 P(x^n|\mathcal{S},\theta) + 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2}\log_2 n + |\mathcal{S}| + 2.$$

Real sequence probability

Context trees

For general trees (or suffix sets) $\mathcal{S}$ we find

$$P_w^\lambda \geq 2^{1-2|\mathcal{S}|} \prod_{s \in \mathcal{S}} P_e(a_s, b_s)$$

This results in the ideal code wordlength
$l_C(x^n) = -\log_2 P_w^\lambda$

$$l_C(x^n) \leq -\log_2 P(x^n|\mathcal{S},\theta) + 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2}\log_2 n + |\mathcal{S}| + 2.$$

Cost of describing the tree

Context trees

For general trees (or suffix sets) $\mathcal{S}$ we find

$$P_w^\lambda \geq 2^{1-2|\mathcal{S}|} \prod_{s \in \mathcal{S}} P_e(a_s, b_s)$$

This results in the ideal code wordlength
$l_C(x^n) = -\log_2 P_w^\lambda$

$$l_C(x^n) \leq -\log_2 P(x^n|\mathcal{S},\theta) + 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2} \log_2 n + |\mathcal{S}| + 2.$$

Cost of the parameters

Context trees

For general trees (or suffix sets) $\mathcal{S}$ we find

$$P_w^\lambda \geq 2^{1-2|\mathcal{S}|} \prod_{s \in \mathcal{S}} P_e(a_s, b_s)$$

This results in the ideal code wordlength
$l_C(x^n) = -\log_2 P_w^\lambda$

$$l_C(x^n) \leq -\log_2 P(x^n|\mathcal{S},\theta) + 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2}\log_2 n + |\mathcal{S}| + 2.$$

Contributions to the code wordlength are: Real sequence probability; Cost of describing the tree; Cost of the parameters

Context trees

This algorithm achieves the (asymptotically) optimal redundancy bound (not only on the average but also individually if we define individual redundancy as the difference between to code wordlength and the ideal code wordlength.

$$\varrho(x^n) \leq 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2} \log_2 n + |\mathcal{S}| + 2.$$

Another essential property of the “Context-Tree Weighting” (CTW) algorithm is its efficient implementation. The number of trees squares with every increment of D and yet the amount of work is at most linear in $D \cdot n$.

Context trees

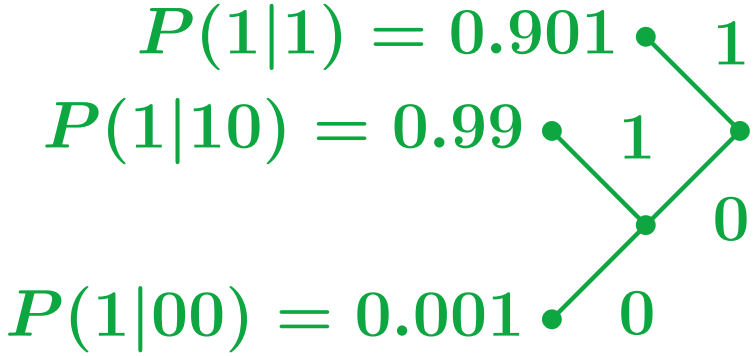
We can even write a stronger result when we realise that the method has no knowledge of a “real model”.
Let $\mathcal{S}_D$ be the set of all tree sources with a maximal depth of atmost D.

$$\varrho(x^n) \leq \min_{\mathcal{S} \in \mathcal{S}_D} \left\{ 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2} \log_2 n + |\mathcal{S}| + 2 \right\}.$$

This algorithm is an instantiation of the MDL principle. It finds (in the class $\mathcal{S}_D$) the model $\mathcal{S}$ that minimizes the code wordlength.

Context trees

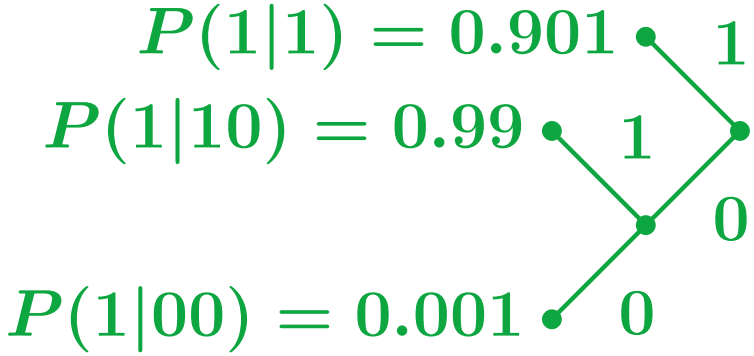
Example: Consider the following actual source.



We shall use the following models.

Context trees

Example: Consider the following actual source.



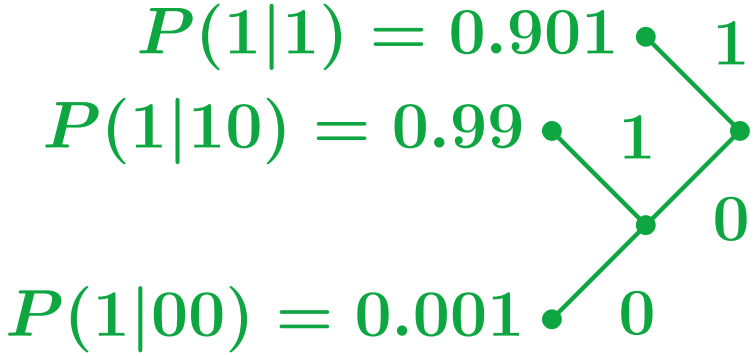
We shall use the following models.

$P(1)$ •

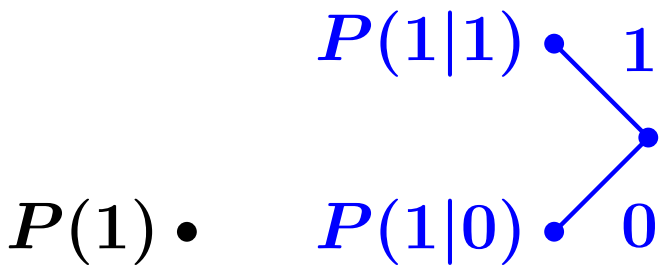
Order-0

Context trees

Example: Consider the following actual source.



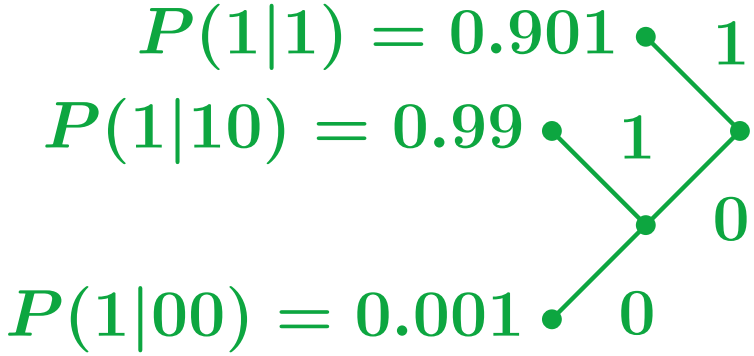
We shall use the following models.



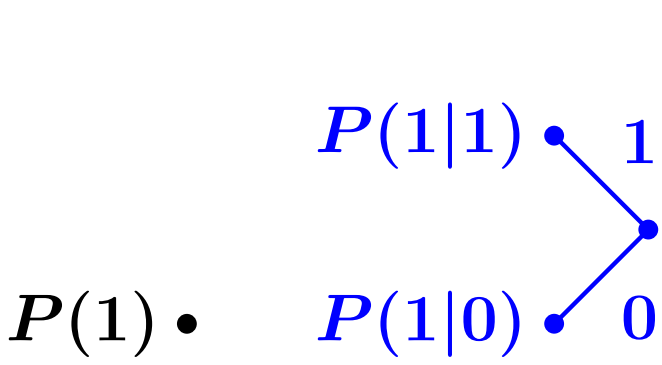
Order-0 Order-1

Context trees

Example: Consider the following actual source.

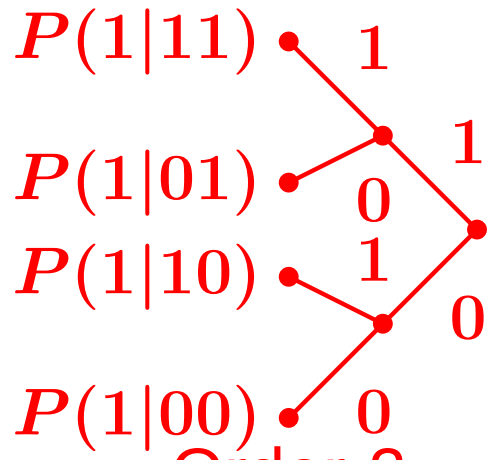


We shall use the following models.



Order-0

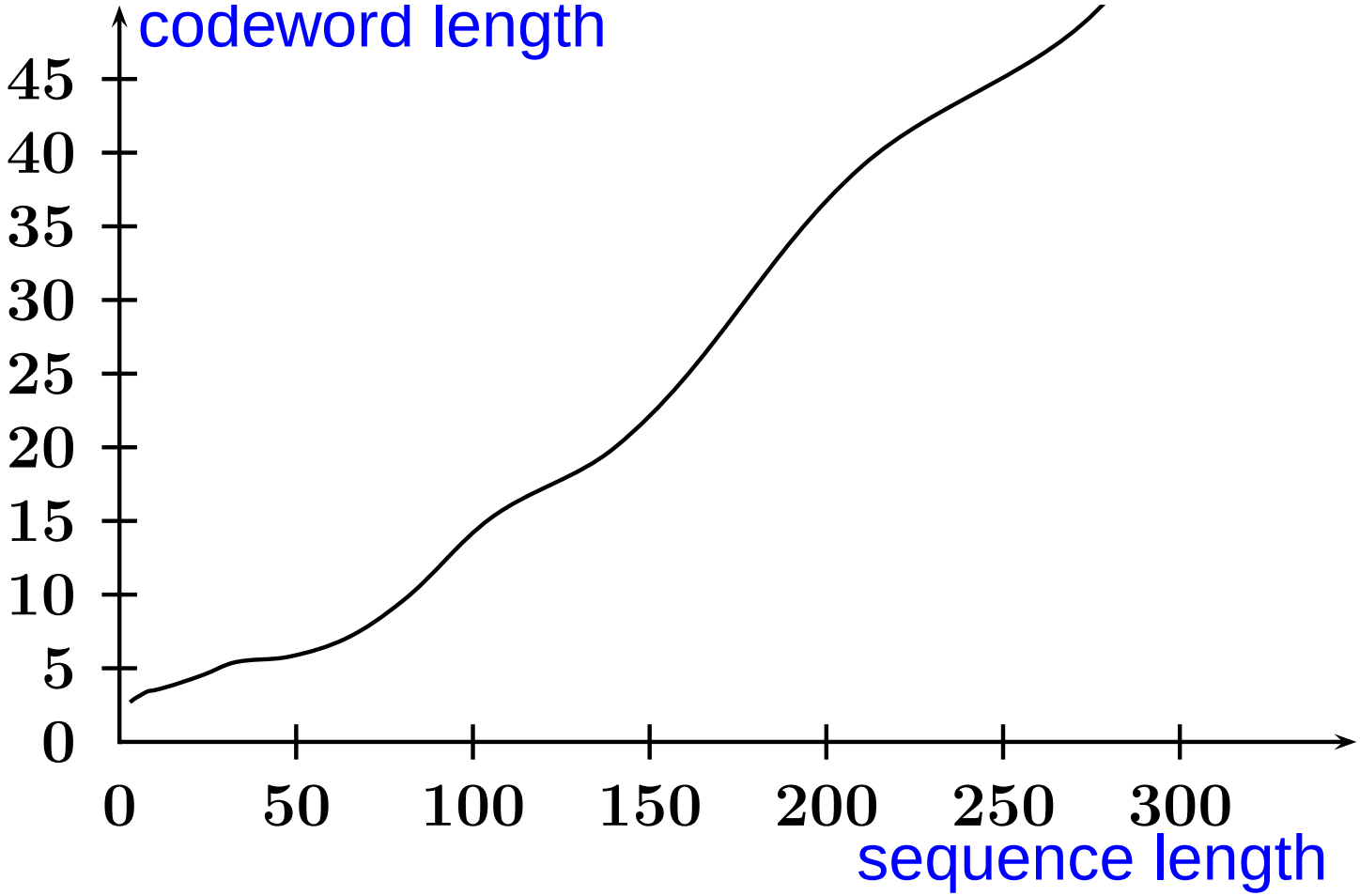
Order-1



Order-2

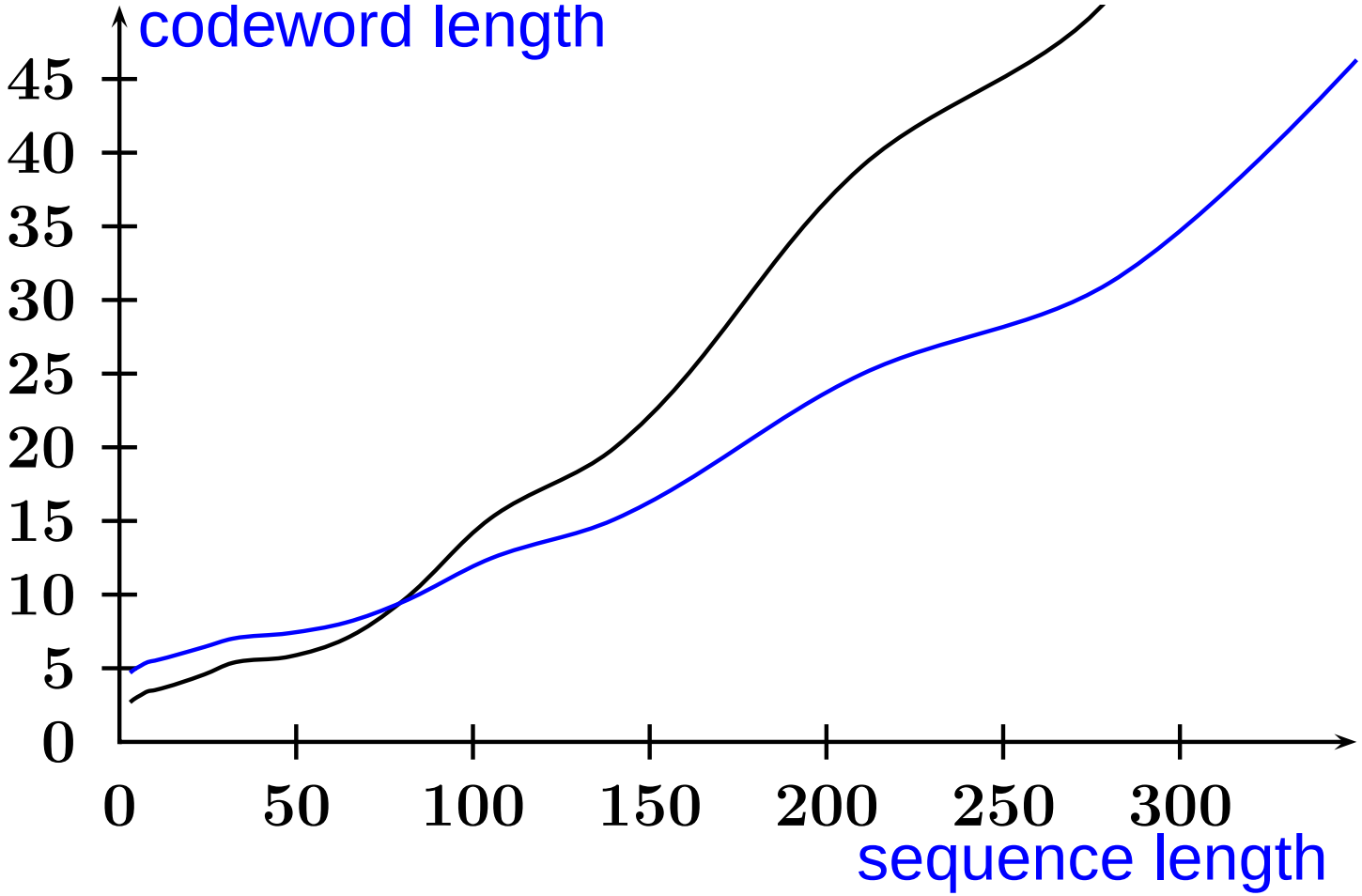
Context trees

The results for sequences of length upto $n = 350$ are shown graphically.



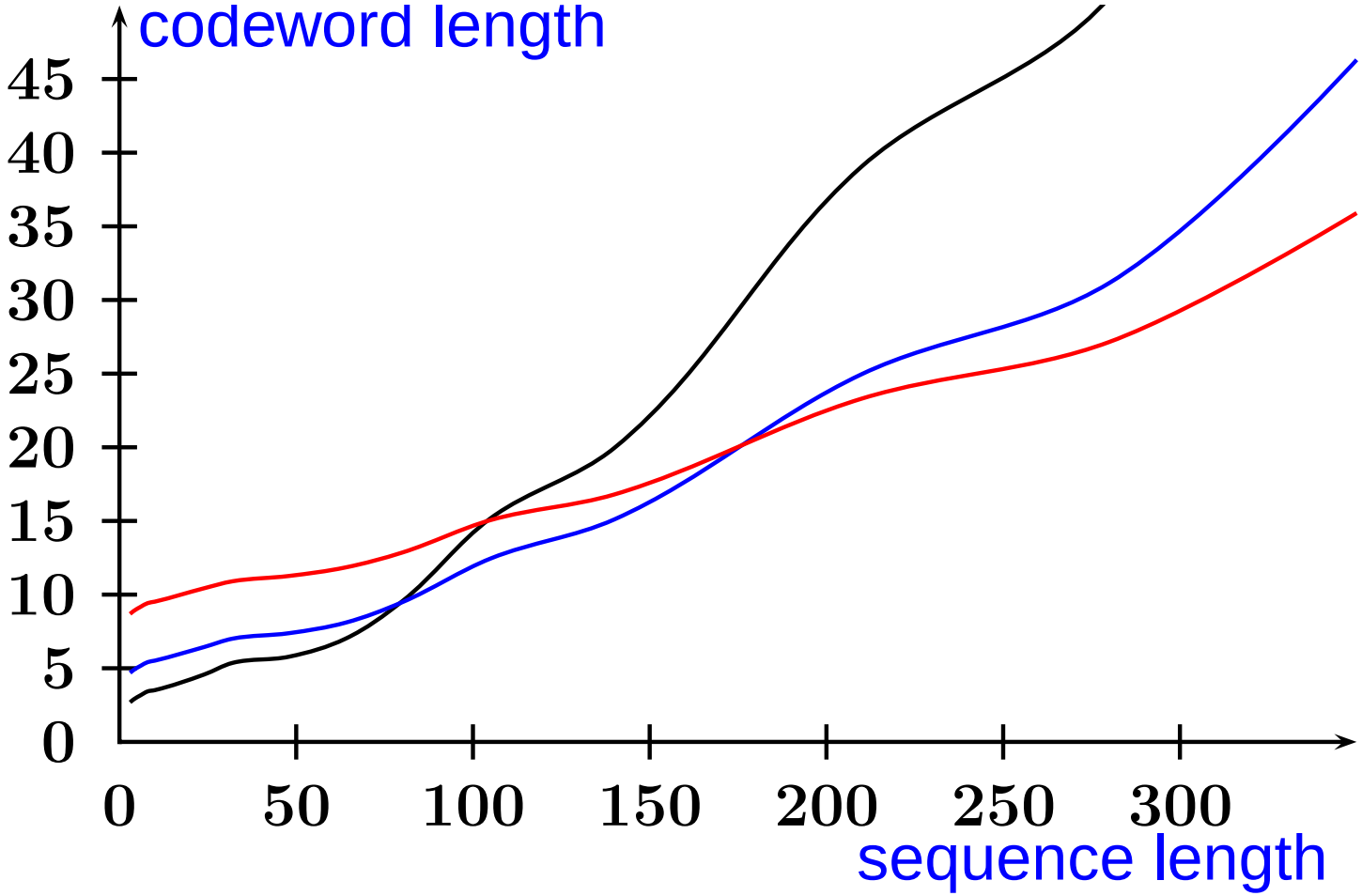
Context trees

The results for sequences of length upto $n = 350$ are shown graphically.



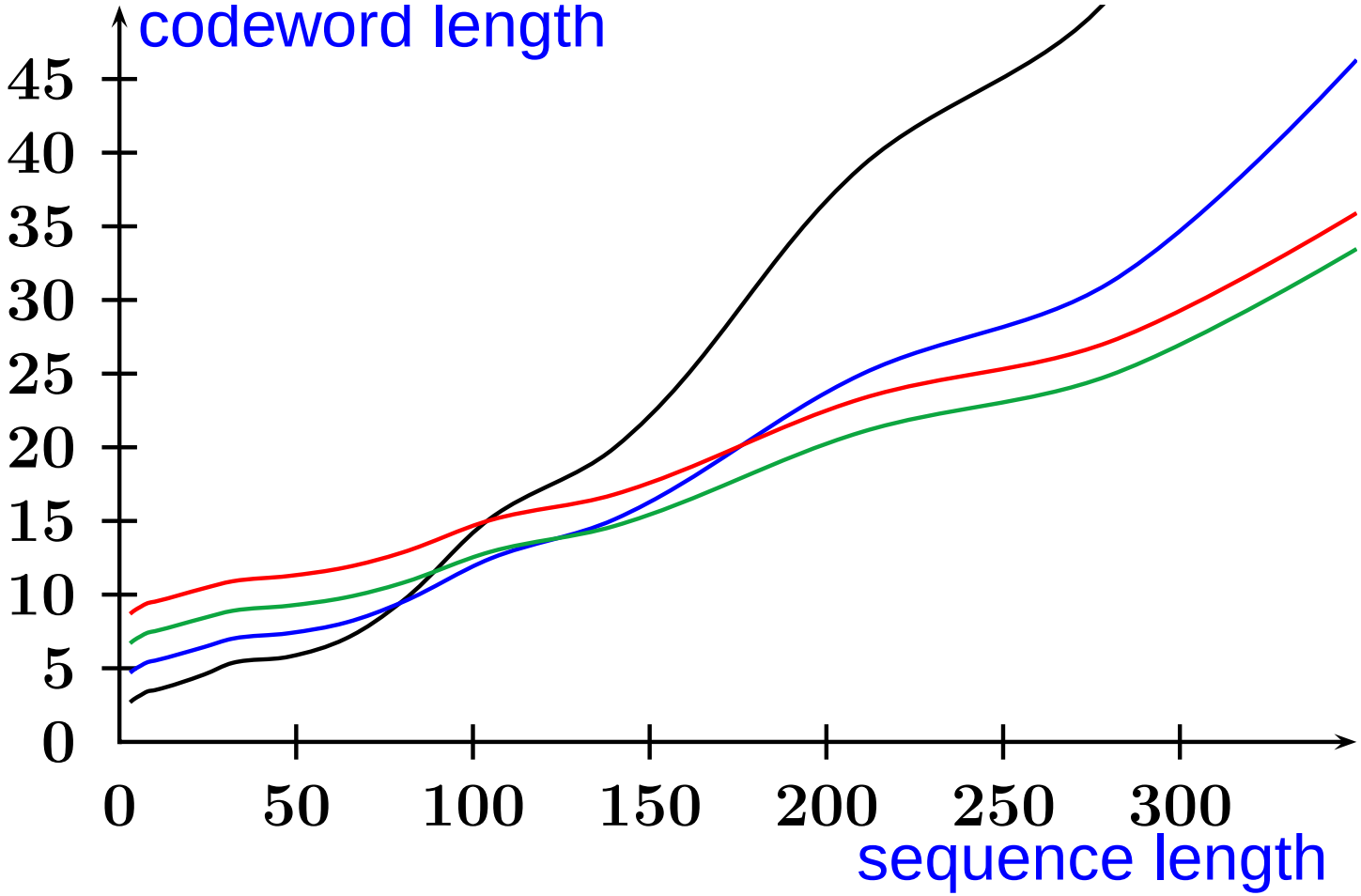
Context trees

The results for sequences of length upto $n = 350$ are shown graphically.



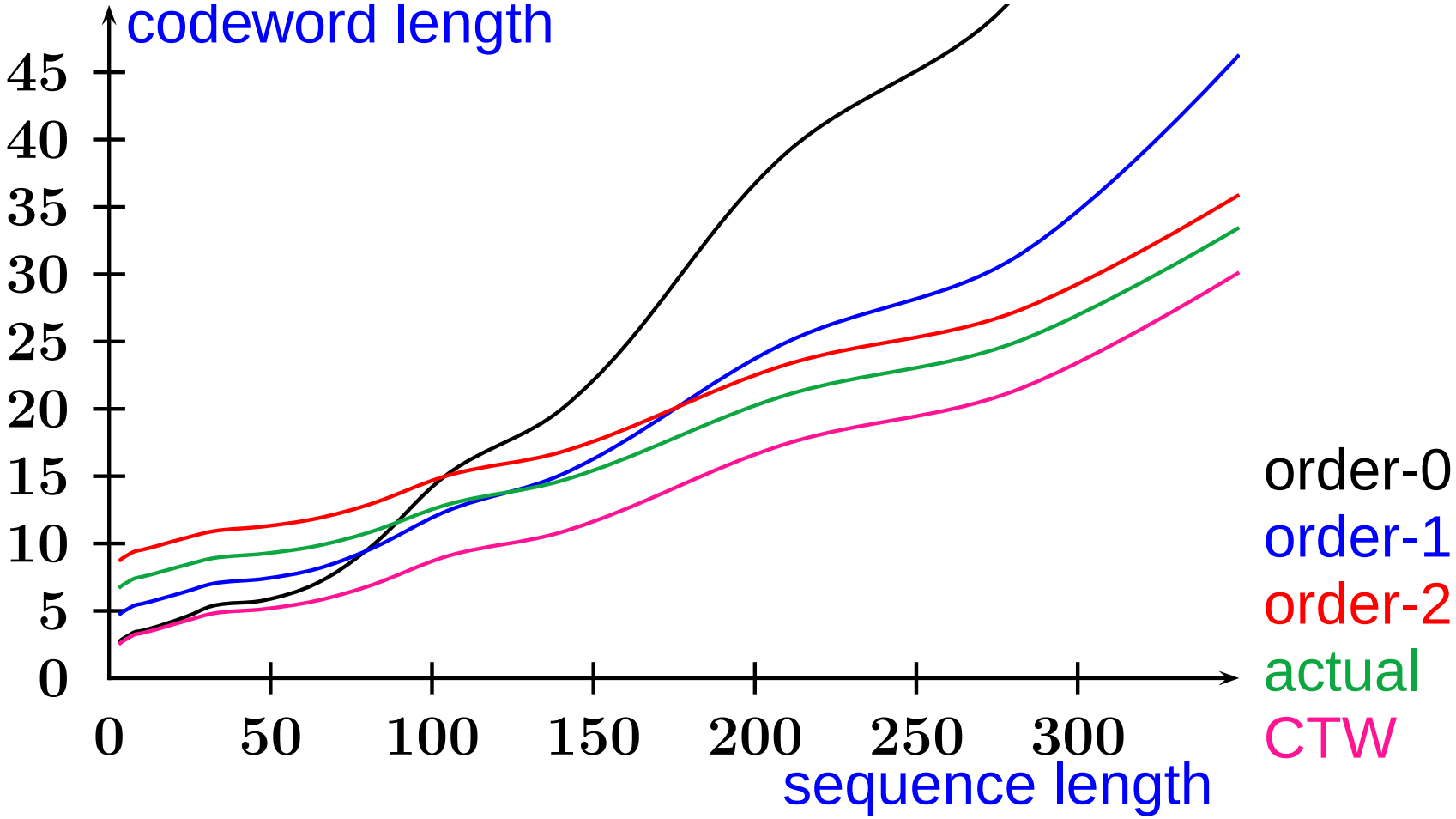
Context trees

The results for sequences of length upto $n = 350$ are shown graphically.



Context trees

The results for sequences of length upto $n = 350$ are shown graphically.



Context trees

We see that initially the memoryless (order-0) model performs even better than the actual model.

After about 80 symbols the order-1 model becomes better than both the order-0 and the actual model.

From 120 symbols on the actual model is better than the simpeler models.

The order-2 model is always worse than the actual model. It describes the same probabilities but has too many parameters.

But the CTW model outperforms all models over the whole range of sequence lengths!

Model posterior for Context trees

We shall now derive an expression, based on the previous method, for the a-posteriori model probability. We consider only binary data but the approach also works for arbitrary alphabets.

First we define our notation.

A model is described by a complete suffix set $\mathcal{S}$.

The suffix set can be seen as the set of leaves of a binary tree. Our model class is the set of all complete binary trees whose depth is not more than D, for a given D. We write $\mathcal{S}_D$ for the model class. So we say that $\mathcal{S} \in \mathcal{S}_D$.
The depth of a tree is the length of the longest path from the root to a leaf.

Model posterior for Context trees

Every model $\mathcal{S}$ has a set of parameters θ_s, one for every state $s \in \mathcal{S}$ of the model. θ_s gives the probability of a 1 given that the previous symbols were s.

$$\theta_s = \mathbf{Pr}\{X_t = 1 | X_{t-\ell}^{t-1} = s\}, \text{ where } \ell = |s|$$

Model posterior for Context trees

The probability of a sequence, given a model $\mathcal{S}$ with parameters θ_s, $s \in \mathcal{S}$ is

$$P(x^n|\mathcal{S},\theta) = \prod_{s\in\mathcal{S}} P(x^n_{|s}|\theta_s)$$

and

$$P(x^n_{|s}|\theta_s) = (1-\theta_s)^{N(0|x^n_{|s})}\theta_s^{N(1|x^n_{|s})}$$

Note (again) that $N(0|x^n_{|s}) = N(s0|x^n)$.

Actually, we silently assume that the first few symbols also have a “context”. So we assume that there are some symbols preceding x^n.

Model posterior for Context trees

We must define some prior distributions. First the prior on the parameters.
We use the beta distribution. (In a non-binary case this generalizes to the Dirichlet distribution.) As done before we select the parameters in the beta distribution to be $\frac{1}{2}$.
So given a model $\mathcal{S}$ then for every $s \in \mathcal{S}$ we take

$$P(\theta_s|\mathcal{S}) = \frac{1}{\pi}\theta_s^{-\frac{1}{2}}(1-\theta_s)^{-\frac{1}{2}}$$

Model posterior for Context trees

This results in the following sequence probability, first assuming one state s only

$$P(x^n_{|s}) = \int_0^1 P(\theta_s|\mathcal{S})\theta_s^{N(s1|x^n)}(1-\theta_s)^{N(s0|x^n)}\,d\theta_s$$
$$= \frac{\Gamma(N(s0|x^n)+\frac{1}{2})\Gamma(N(s1|x^n)+\frac{1}{2})}{\pi\Gamma(N(s|x^n)+1)}$$

Now for any tree model $\mathcal{S}$ we find

$$P(x^n|\mathcal{S}) = \prod_{s\in\mathcal{S}}\int_0^1 P(\theta_s|\mathcal{S})P(x^n_{|s}|\theta_s)\,d\theta_s$$
$$= \prod_{s\in\mathcal{S}}\frac{\Gamma(N(s0|x^n)+\frac{1}{2})\Gamma(N(s1|x^n)+\frac{1}{2})}{\pi\Gamma(N(s|x^n)+1)}$$

Model posterior for Context trees

Next we need a prior on the tree models $\mathcal{S}$ in the set $\mathcal{S}_D$. We wish to use the efficient CTW method of weighting so we choose the corresponding prior.

First define

$$\Delta_D(S) \stackrel{\Delta}{=} 2|\mathcal{S}| - 1 - |\{s \in \mathcal{S} : |s| = D\}|.$$

Then we take the prior

$$P(\mathcal{S}) = 2^{-\Delta_D(\mathcal{S})}$$

We prove that this is a proper prior probability.

Model posterior for Context trees

Obviously, $P(\mathcal{S}) > 0$ for all $\mathcal{S} \in \mathcal{S}_D$. We must show that it sums up to one.

We give a proof by induction.

Step 1: $D = 0$: $\mathcal{S}_0 = \{\lambda\}$, the memoryless source.

$$\Delta_0(\lambda) = 2 \cdot 1 - 1 - 1$$

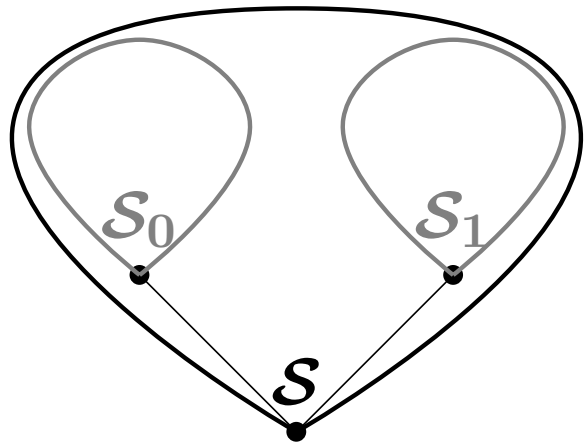
Where the last -1 comes from the fact that the single state of λ is at level $D = 0$ so $P(\lambda) = 1$.

Model posterior for Context trees

Induction: Assume it holds for $D \leq D^*$. Now if $\mathcal{S} \in \mathcal{S}_{D^*+1}$ then

- $\mathcal{S} = \lambda$, i.e. root node only.
- $\mathcal{S}$ contains two trees on level 1, say $\mathcal{S}_0 \in \mathcal{S}_{D^*}$ and $\mathcal{S}_1 \in \mathcal{S}_{D^*}$. We have

$$\Delta_{D^*+1}(\mathcal{S}) = 1 + \Delta_{D^*}(\mathcal{S}_0) + \Delta_{D^*}(\mathcal{S}_1)$$



Model posterior for Context trees

- We repeat: $\mathcal{S}$ contains two trees on level 1, say $\mathcal{S}_0 \in \mathcal{S}_{D^*}$ and $\mathcal{S}_1 \in \mathcal{S}_{D^*}$. We have

$$\Delta_{D^*+1}(\mathcal{S}) = 1 + \Delta_{D^*}(\mathcal{S}_0) + \Delta_{D^*}(\mathcal{S}_1)$$

$$\sum_{\mathcal{S} \in \mathcal{S}_{D^*+1}} 2^{-\Delta_{D^*+1}(\mathcal{S})} = 2^{-1} +$$

$$\sum_{\mathcal{S}_0 \in \mathcal{S}_{D^*}} \sum_{\mathcal{S}_1 \in \mathcal{S}_{D^*}} 2^{-1-\Delta_{D^*}(\mathcal{S}_0)-\Delta_{D^*}(\mathcal{S}_1)}$$

$$= 2^{-1} + 2^{-1} \underbrace{\sum_{\mathcal{S}_0 \in \mathcal{S}_{D^*}} 2^{-\Delta_{D^*}(\mathcal{S}_0)}}_{=1} \underbrace{\sum_{\mathcal{S}_1 \in \mathcal{S}_{D^*}} 2^{-\Delta_{D^*}(\mathcal{S}_1)}}_{=1}$$

$$= 2^{-1} + 2^{-1} = 1$$

Model posterior for Context trees

We now show that the weighted sequence probability

$$P(x^n) = \sum_{\mathcal{S} \in \mathcal{S}_D} P(\mathcal{S}) P(x^n|\mathcal{S}),$$

is produced by the weighting procedure of CTW, so

$$P(x^n) = P_w^{\lambda}.$$

Model posterior for Context trees

We shall prove this using (mathematical) induction.

First assume $D = 0$: $\mathcal{S}_0 = \{\lambda\}$, so the only tree in the set consists of a root only. Therefor $\Delta_0(\lambda) = 0$. So,

$$\begin{aligned} P(x^n) &= P(\lambda)P(x^n|\lambda) \\ &= 2^0 P_e(N(0|x^n), N(1|x^n)) \\ &= P_w^\lambda, \end{aligned}$$

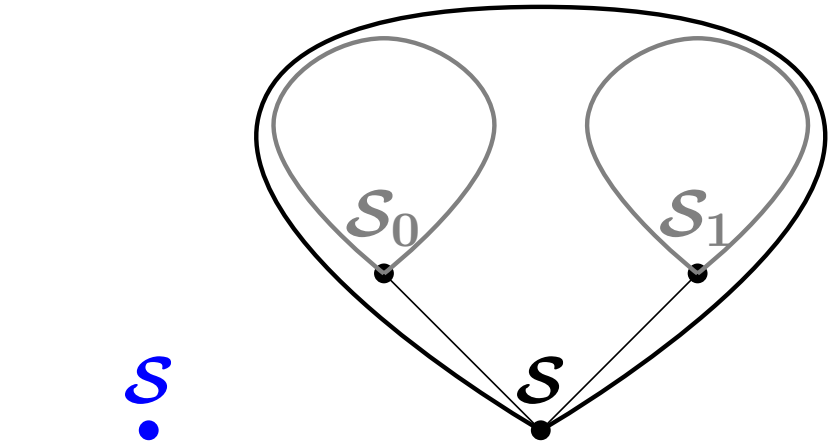
because λ is also a leaf and in a leaf $P_w = P_e$.

Model posterior for Context trees

Now assume that for all $D \leq D^*$

$$\sum_{\mathcal{S} \in \mathcal{S}_D} P(\mathcal{S}) P(x^n | \mathcal{S}) = P_w^{\lambda}$$

The tree $\mathcal{S}$ is either the root only or it consists of a root plus two trees, $\mathcal{S}_0$ and $\mathcal{S}_1$, on level one.



root only split at level 1

Model posterior for Context trees

$$\sum_{\mathcal{S} \in \mathcal{S}_{D^*+1}} P(\mathcal{S}) P(x^n|\mathcal{S}) =$$

$$= 2^{-1} P_e(N(0|x^n), N(1|x^n)) +$$

$$\sum_{\mathcal{S} \in \mathcal{S}_{D^*+1} : \mathcal{S} \neq \lambda} P(\mathcal{S}) P(x^n|\mathcal{S})$$

Model posterior for Context trees

$$\sum_{\mathcal{S}\in\mathcal{S}_{D^*+1}:\mathcal{S}\neq\lambda} P(\mathcal{S})P(x^n|\mathcal{S}) =$$

$$\sum_{\mathcal{S}_0\in\mathcal{S}_{D^*}}\sum_{\mathcal{S}_1\in\mathcal{S}_{D^*}} \frac{1}{2}2^{-\Delta_{D^*}(\mathcal{S}_0)}2^{-\Delta_{D^*}(\mathcal{S}_1)}\times$$

$$P(x^n_{|0}|\mathcal{S}_0)P(x^n_{|1}|\mathcal{S}_1)$$

$$= \frac{1}{2}\sum_{\mathcal{S}_0\in\mathcal{S}_{D^*}} 2^{-\Delta_{D^*}(\mathcal{S}_0)}P(x^n_{|0}|\mathcal{S}_0)\times$$

$$\sum_{\mathcal{S}_1\in\mathcal{S}_{D^*}} 2^{-\Delta_{D^*}(\mathcal{S}_1)}P(x^n_{|1}|\mathcal{S}_1)$$

$$= \frac{1}{2}P_w^0 P_w^1$$

Model posterior for Context trees

And so we find

$$\sum_{\mathcal{S} \in \mathcal{S}_{D^*+1}} P(\mathcal{S}) P(x^n|\mathcal{S}) =$$

$$= \frac{1}{2} P_e(N(0|x^n), N(1|x^n)) + \frac{1}{2} P_w^0 P_w^1$$

$$= P_w^\lambda$$

Model posterior for Context trees

Thus we can compute the a-posteriori model probability.

$$P(\mathcal{S}|x^n) = \frac{P(\mathcal{S})P(x^n|\mathcal{S})}{P(x^n)}$$

Model posterior for Context trees

Thus we can compute the a-posteriori model probability.

$$P(\mathcal{S}|x^n) = \frac{P(\mathcal{S})P(x^n|\mathcal{S})}{P(x^n)}$$

Model posterior for Context trees

Thus we can compute the a-posteriori model probability.

$$P(\mathcal{S}|x^n) = \frac{2^{-\Delta_D(\mathcal{S})} P(x^n|\mathcal{S})}{P(x^n)}$$

Model posterior for Context trees

Thus we can compute the a-posteriori model probability.

$$P(\mathcal{S}|x^n) = \frac{2^{-\Delta_D(\mathcal{S})} \prod_{s \in \mathcal{S}} P_e(N(s0|x^n), N(s1|x^n))}{P(x^n)}$$

Model posterior for Context trees

Thus we can compute the a-posteriori model probability.

$$P(\mathcal{S}|x^n) = \frac{2^{-\Delta_D(\mathcal{S})} \prod_{s \in \mathcal{S}} P_e(N(s0|x^n), N(s1|x^n))}{P_w^{\lambda}}$$

Model posterior for Context trees

Thus we can compute the a-posteriori model probability.

$$P(\mathcal{S}|x^n) = \frac{2^{-\Delta_D(\mathcal{S})} \prod_{s\in\mathcal{S}} P_e(N(s0|x^n), N(s1|x^n))}{P_w^{\lambda}}$$

So, we can use the same computations as in the CTW.

An efficient way to find the Bayesian MAP model exists, but its discussion is not a part of this course.

Summary

We considered methods to select models given the data.

We looked at the BIC for Gaussian and discrete Markov models and found that it allows us to find the “correct” model. At least it avoids the model overestimation problem that the ML approach has.

Summary

We took the Bayesian approach

$$\Pr\{\mathcal{M}|x\} = \frac{\Pr\{\mathcal{M}\}\Pr\{x|\mathcal{M}\}}{\Pr\{x\}}$$

The problem with this approach is that we do not know which model prior we should use.

Using a complexity based approach we found the Kolmogorov minimal sufficient statistic which describes the simplest model that describes our data (Occam's razor).

Summary

This statistic is non-computable so we looked for a computable approximation which we found in Universal Source Coding.

This led to the MDL principle, which is Kolmogorovs minimal sufficient statistic or Occam's razor in disguise.

For some problems we found that the cost of describing a parameter, in terms of increased code wordlength, or exponential decrease of probability was given by

$$\frac{1}{2}\log_2 n$$

Summary

For this result to hold, the parameters under consideration must have ML estimators that converge to the proper value with rate $1/\sqrt{n}$. This is true for the mean and variance of a Gaussian, and also for the parameter of a Bernoulli distribution.

This result was also presented as a parameter learning effect using the redundancy-capacity theorem, where we saw that the redundancy (cost of parameter estimation) was lowerbounded (and often equal) to the capacity of the channel from the parameter space to the output sequences.

Summary

Finally we considered a complex modeling problem, namely the context tree model selection. First we considered the related problem of universal data compression for context models (CTW) and showed that the MDL principle holds.

Then we reformulated the problem in a Bayesian a-posteriori model probability setting and found that, by selecting the appropriate model prior probability, the CTW method can be reformulated to compute this posterior efficiently.

Exam requirements

I expect that you will be able to understand the theory presented here. Every proof makes a point and that can be used as a part of a exam question. However, I do not expect you to reproduce the proofs themselves.
If you know how it works it is not hard to use the ideas.

Working on the exercises should help a lot as the exam questions will be similar.

Related reading I

- Cover, Thomas M., and Thomas, Joy A., Elements Of Information Theory Chapter 7., Wiley, New York, ISBN 0-471-06259-6, 1991.
- Davisson, Lee D., and Leon-Garcia, Alberto, “A Source Matching Approach to Finding Minimax Codes,” IEEE Trans. Inform. Theory, Vol IT-26, No 2, March 1980, pp. 166–174.
- MacKay, David J.C., ,Information Theory, Inference, and Learning Algorithms Cambridge Univ. Press, ISBN 0-521-64298-1, 2003.

Related reading II

- Willems, Frans M.J., Shtarkov, Yuri M., and Tjalkens, Tjalling J., “The Context-Tree Weighting Method: Basic properties,” IEEE Trans.Inform. Theory, IT-41, no 3, pp. 653–664, 1995.
- Grünwald, Peter D., The Minimum Description Length Principle and Reasoning under Uncertainty Ph.D. thesis, Univ. Amsterdam, ILLC publications, ISBN 90-5776-009-6, 1998.
- Hansen, Mark H., Yu, Bin, “Model Selection and the Principle of Minimum Description Length,” J. Am. Stat. Assoc., Vol. 96, No. 454, pp. 746–774, 2001.

Appendices

All appendices are optional reading.

An Elias code and the Kolmogorov bound

This code is used in the proof of Theorem 2 to encode the length of the sequence. The proof of the bound of Theorem 2 follows immediately after.

An Elias code

Let l be an arbitrary positive integer. So the set of possible l values is (countable) infinite.

We wish to assign a prefix-free (decodable) binary codeword to every possible integer. Thus we can append this codeword to a program p so that the computer can know the sequence length $l(x)$.

An Elias code

Let l be an arbitrary positive integer. So the set of possible l values is (countable) infinite.

We wish to assign a prefix-free (decodable) binary codeword to every possible integer. Thus we can append this codeword to a program p so that the computer can know the sequence length $l(x)$.

A convenient choice is to assign to l a codeword of length $\log_2 l$, but this is not decodable.

An Elias code

Let l be an arbitrary positive integer. So the set of possible l values is (countable) infinite.

We wish to assign a prefix-free (decodable) binary codeword to every possible integer. Thus we can append this codeword to a program p so that the computer can know the sequence length $l(x)$.

A convenient choice is to assign to l a codeword of length $\log_2 l$, but this is not decodable.

Consider the binary expansion of l and repeat every bit twice. Then finish the sequence with “01”.

An Elias code

It is clear that this description (codeword) requires

$$l(l) = 2\lfloor \log_2 l + 1 \rfloor + 2 \text{ bits.}$$

An Elias code

It is clear that this description (codeword) requires

$$l(l) = 2\lfloor \log_2 l + 1 \rfloor + 2 \text{ bits.}$$

Note: More efficient Elias codes exist, up to codes with a length $l(l) = \log^* l$.
($\log^* n = \log n + \log\log n + \log\log\log n + \cdots$ where the sum is continued until the last positive term.)

An Elias code

Example 13:

l	codeword	$l(l)$
1	1101	4
2	110001	6
3	111101	6
5	11001101	8
10	1100110001	10
20	110011000001	12
50	11110000110001	14

Kolmogorov complexity

Proof: [of Theorem 2] It is obvious that the simple Elias code of the length of a sequence x can be appended to a program. The length increase is bounded by $2\log_2 l(x) + c$.
This proves the statement of the theorem.

$$K(x) \leq K(x|l(x)) + 2\log_2 l(x) + c.$$

□

The notion of Type

Types

Types are sets of sequences that have the same probability under Bernouilli and multinomial distributions.

The notion of Type

Let $\mathcal{X}$ be a finite alphabet of m symbols. x is a sequence of n letters taken from this alphabet.

$$N(a|x) = \text{the number of times } a \in \mathcal{X} \text{ occurs in } x$$

$$P_x(a) = N(a|x)/n, \text{ the empirical probability}$$

$$T(P_x) = \{y | N(a|y) = N(a|x) \text{ for all } a \in \mathcal{X}\}$$

$$|T(P_x)| = \binom{n}{N(a_1|x), N(a_2|x), \ldots, N(a_m|x)}$$

P_x is called the type of x.

$T(P_x)$ is called a type class.

The notion of Type

Example 14: $\mathcal{X} = \{0, 1, 2\}$; $x = 0102$.

$$P_x(0) = \frac{1}{2}, \qquad P_x(1) = \frac{1}{4}, \qquad P_x(2) = \frac{1}{4}.$$

$$T(P_x) = \left\{ \begin{array}{cccccc} 0012, & 0021, & 0102, & 0120, & 0201, & 0210, \\ 1002, & 1020, & 1200, & 2001, & 2010, & 2100 \end{array} \right\}$$

$$|T(P_x)| = \binom{4}{2,1,1} = \frac{4!}{2!1!1!} = 12$$

The notion of Type

Let $\mathcal{P}_n$ be the set of all possible types generated from sequences with n symbols.

Theorem 5 *The number of types is polynomial in* n*.*

$$|\mathcal{P}_n| \leq (n+1)^{|\mathcal{X}|}.$$

The notion of Type

Let $\mathcal{P}_n$ be the set of all possible types generated from sequences with n symbols.

Theorem 5 *The number of types is polynomial in* n*.*

$$|\mathcal{P}_n| \leq (n+1)^{|\mathcal{X}|}.$$

Proof: A type is specified by a vector with $|\mathcal{X}| = m$ components. Every component can take at most $n+1$ values. Not all combinations are valid so $(n+1)^m$ is an upper bound.

□

The notion of Type

Theorem 6 *For the probability of a sequence x under its own type P_x holds*

$$P^n(x) = 2^{-nH(P_x)}.$$

The notion of Type

Theorem 6 *For the probability of a sequence x under its own type P_x holds*

$$P^n(x) = 2^{-nH(P_x)}.$$

Proof:

$$\begin{aligned} P^n(x) &= \prod_{i=1}^{n} P_x(x_i) = \prod_{a \in \mathcal{X}} P_x(a)^{N(a|x)} \\ &= \prod_{a \in \mathcal{X}} P_x(a)^{nP_x(a)} = \prod_{a \in \mathcal{X}} 2^{nP_x(a) \log_2 P_x(a)} \\ &= 2^{-nH(P_x)} \end{aligned}$$

□

The notion of Type

Theorem 7 *For any type class $T(P)$ we have*

$$\frac{1}{(n+1)^{|\mathcal{X}|}} 2^{nH(P)} \leq |T(P)| \leq 2^{nH(P)}$$

The notion of Type

Theorem 7 *For any type class $T(P)$ we have*

$$\frac{1}{(n+1)^{|\mathcal{X}|}} 2^{nH(P)} \leq |T(P)| \leq 2^{nH(P)}$$

Proof: [of the righthand side]

$$1 \stackrel{(*1)}{\geq} \sum_{x \in T(P)} P^n(x) = \sum_{x \in T(P)} 2^{-nH(p)}$$

$$= |T(P)| 2^{-nH(P)}.$$

Ad (*1): We sum over less than all sequences.

□

The notion of Type

For the second part we need the following simple fact

$$\frac{m!}{n!} \geq n^{m-n}.$$

The notion of Type

For the second part we need the following simple fact

$$\frac{m!}{n!} \geq n^{m-n}.$$

Proof: First assume that $m > n$.

$$\frac{m!}{n!} = m \cdot (m-1) \cdots (n+1) \geq n \cdot n \cdots n = n^{m-n}$$

Now let $m < n$

$$\begin{aligned} \frac{m!}{n!} &= 1/\left(n \cdot (n-1) \cdots (m+1)\right) \\ &\geq 1/\left(n \cdot n \cdots n\right) = 1/n^{n-m} = n^{m-n}. \end{aligned}$$

□

The notion of Type

Proof [of the left-hand side of Theorem 7]
P^n is determined by a fixed type P. $\hat{P}$ is an arbitrary type.

$$\frac{P^n(T(P))}{P^n(T(\hat{P}))} = \frac{\binom{n}{nP(a_1),\ldots,nP(a_m)} \prod_{a\in\mathcal{X}} P(a)^{nP(a)}}{\binom{n}{n\hat{P}(a_1),\ldots,n\hat{P}(a_m)} \prod_{a\in\mathcal{X}} P(a)^{n\hat{P}(a)}}$$

$$= \prod_{a\in\mathcal{X}} \frac{(n\hat{P}(a))!}{(nP(a))!} P(a)^{nP(a)-n\hat{P}(a)}$$

$$\stackrel{(*1)}{\geq} \prod_{a\in\mathcal{X}} (nP(a))^{n\hat{P}(a)-nP(a)} P(a)^{nP(a)-n\hat{P}(a)}$$

$$= \prod_{a\in\mathcal{X}} n^{n(\hat{P}(a)-P(a))} = n^{n\sum_{a\in\mathcal{X}}(\hat{P}(a)-P(a))} = 1.$$

Ad (*1): We use the simple fact of the previous sheet.

The notion of Type

Proof (continued) So $P^n(T(P)) \geq P^n(T(\hat{P}))$. We use this as follows.

$$
\begin{aligned}
1 &= \sum_{Q \in \mathcal{P}_n} P^n(T(Q)) \leq \sum_{Q \in \mathcal{P}_n} \max_{Q'} P^n(T(Q')) \\
&= \sum_{Q \in \mathcal{P}_n} P^n(T(P)) \leq (n+1)^{|\mathcal{X}|} P^n(T(P)) \\
&= (n+1)^{|\mathcal{X}|} \sum_{x \in T(P)} P^n(x) = (n+1)^{|\mathcal{X}|} \sum_{x \in T(P)} 2^{-nH(P)} \\
&= (n+1)^{|\mathcal{X}|} |T(P)| 2^{-nH(P)}.
\end{aligned}
$$

□

Universal data compression

We give the proof of Theorem 4. This theorem proves the existence of codes that perform asymptotically as a Huffman code but now without knowing the source parameter θ. It also shows the *parameter cost* of $\frac{1}{2}\log_2 n$.

Universal data compression

Theorem 4 (Optimal number of sources) *For a sequence x^n generated by an binary i.i.d. source with unknown $\Pr\{X = 1\} = \theta$ the optimal number of alternative sources is of order $\sqrt{n}$ and the achieved redundancy of the resulting code C^*, relative to any i.i.d. source, is bounded as*

$$r_n(C^*) < \frac{\log_2 n}{2n}(1 + \epsilon),$$

and also

$$r_n(C^*) > \frac{\log_2 n}{2n}(1 - \epsilon),$$

for any $\epsilon > 0$ and n sufficiently large.

We shall prove this theorem in several steps.

Universal data compression

Select m numbers θ_i such that

$$0 < \theta_1 < \theta_2 < \cdots < \theta_m < 1$$

Every θ_i defines a sequence probability as

$$p(x^n|\theta_i) = (1-\theta_i)^{N(0|x^n)}\theta_i^{N(1|x^n)}$$

The code word for x^n can now be made by

- describing the θ_i used: needs $< \log_2 m + 1$ bits
- producing a word with ideal length: needs $< -\log_2 p(x^n|\theta_i) + 1$ bits

Universal data compression

So,

$$l_C(x^n) < \min_{i=1,\ldots,m} -\log_2 p(x^n|\theta_i) + \log_2 m + 2$$

Note that this is a (sort of) maximum likelihood code!

Assume that the actual source probability is θ. We write for the expected redundancy (unnormalized)

$$\begin{aligned} nr_n &= \mathbf{E}\{l_C(X^n)\} - H_\theta(X^n) \\ &< \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \min_i \log_2 \frac{1}{p(x^n|\theta_i)} - \\ &\quad \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \log_2 \frac{1}{p(x^n|\theta)} + \log_2 m + 2 \end{aligned}$$

Universal data compression

$$
\begin{aligned}
nr_n &< \sum_{x^n \in \mathcal{X}^n} \min_i p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{p(x^n|\theta_i)} + \log_2 m + 2 \\
&= \sum_{k=0}^{n} \sum_{\substack{x^n \in \mathcal{X}^n \\ N(1|x^n)=k}} \min_i p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{p(x^n|\theta_i)} + \log_2 m + 2 \\
&\leq \min_i \sum_{k=0}^{n} \binom{n}{k} \theta^k (1-\theta)^{n-k} n \left(\frac{k}{n} \log_2 \frac{\theta}{\theta_i} + \right. \\
&\qquad \left. \frac{n-k}{n} \log_2 \frac{1-\theta}{1-\theta_i} \right) + \log_2 m + 2 \\
&= n \min_i \left(\theta \log_2 \frac{\theta}{\theta_i} + (1-\theta) \log_2 \frac{1-\theta}{1-\theta_i} \right) + \log_2 m + 2 \\
&= n \min_i d(\theta \| \theta_i) + \log_2 m + 2
\end{aligned}
$$

Universal data compression

Lemma 1 (Quadratic divergence bound) *For all p and q with $0 \leq p, q \leq 1$ holds*

$$d(p\|q) \leq \log_2 e \frac{(p-q)^2}{q(1-q)}$$

Universal data compression

Lemma 1 (Quadratic divergence bound) *For all p and q with $0 \leq p, q \leq 1$ holds*

$$d(p\|q) \leq \log_2 e \frac{(p-q)^2}{q(1-q)}$$

Proof: Use the log inequality $\log_2 x \leq \log_2 e(x-1)$.

$$\begin{aligned} d(p\|q) &= (1-p)\log_2 \frac{1-p}{1-q} + p \log_2 \frac{p}{q} \\ &\leq \log_2 e \left[(1-p)\left(\frac{1-p}{1-q} - 1\right) + p\left(\frac{p}{q} - 1\right)\right] \\ &= \log_2 e \frac{(p-q)^2}{q(1-q)} \end{aligned}$$

Universal data compression

Lemma 2

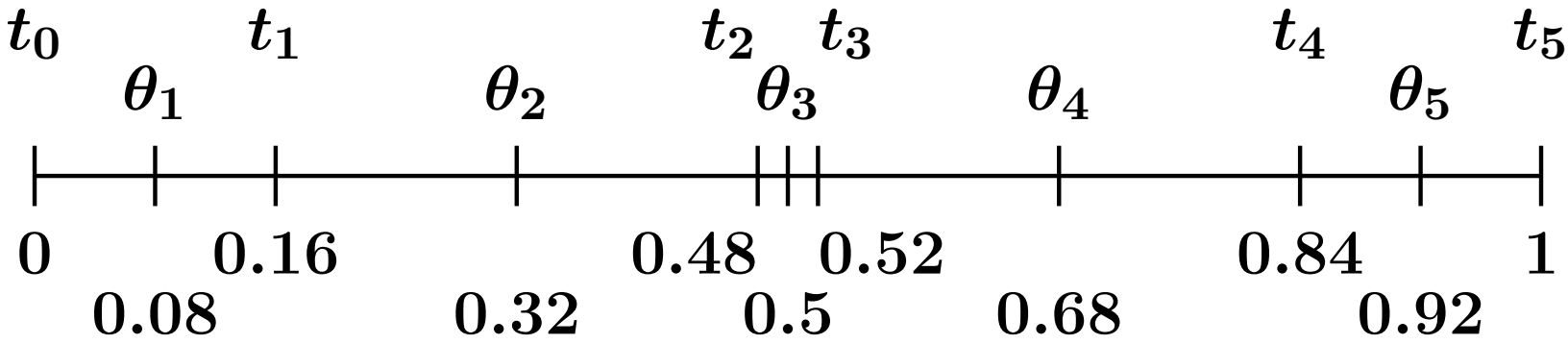
$$\min_{i=1,\ldots,m} d(\theta \| \theta_i) \leq \frac{4 \log_2 e}{m^2}$$

Universal data compression

proof: [only with m odd, m even similar]
Select (resp. define)

$$\theta_i = \begin{cases} \frac{2i^2}{m^2} & \text{for } i = 1, 2, \ldots, \frac{m-1}{2} \\ \frac{1}{2} & \text{for } i = \frac{m+1}{2} \\ 1 - \theta_{m+1-i} & \text{for } i = \frac{m+3}{2}, \ldots, m \end{cases}$$

$$t_j = \begin{cases} \frac{2j(j+1)}{m^2} & \text{for } j = 0, 1, \ldots, \frac{m-1}{2} \\ 1 - t_{m-j} & \text{for } j = \frac{m+1}{2}, \ldots, m \end{cases}$$



Universal data compression

Strategy: If $\theta \in [t_{j-1}, t_j)$ then use θ_j to design the code. We write this j as $j^*(\theta)$.

Obviously

$$\min_{i=1,\ldots,m} d(\theta \| \theta_i) \leq d(\theta \| \theta_{j^*(\theta)})$$

Consider $\theta \in [t_{\frac{m-1}{2}}, t_{\frac{m+1}{2}})$ so we use $\theta_{j^*(\theta)} = \frac{1}{2}$.

Worst case θ is one of the extremal points, so

$$d(\theta \| \frac{1}{2}) \leq d(t_{\frac{m-1}{2}} \| \frac{1}{2})$$

$$\leq \log_2 e \frac{(1/(2m)^2)^2}{(1/2)^2} = \frac{\log_2 e}{m^4}$$

Universal data compression

Now assume that θ is such that $j^*(\theta) \leq \frac{m-1}{2}$, so we use $\theta_{j^*(\theta)} < \frac{1}{2}$. Worst case θ is now $\theta = t_{j^*(\theta)-1}$ or $\theta = t_{j^*(\theta)}$. Using the quadratic bound again we find

$$d(t_j \| \theta_j) \leq \log_2 e \frac{\left(\frac{2j}{m^2}\right)^2}{\frac{2j^2}{m^2}\frac{m^2-2j^2}{m^2}} = \frac{4\log_2 e}{2m^2 - 4j^2}$$

worst case if $j = \frac{m-1}{2}$ so

$$\leq \frac{4\log_2 e}{m^2 + 2m - 1} < \frac{4\log_2 e}{m^2}$$

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$

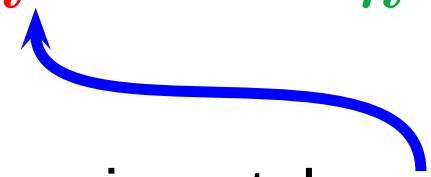
Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$



Major terms are: source mismatch cost

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$

Major terms are: model (parameter) description

Universal data compression

So the overall worst case redundancy is upper bounded by $\frac{4 \log_2 e}{m^2}$.

What m should we select?

Suppose we select m polynomial, i.e. $m = n^\alpha$ for some fixed α. The redundancy is upper bounded by

$$r_n < \frac{4 \log_2 e}{n^{2\alpha}} + \frac{\alpha \log_2 n}{n} + \frac{2}{n}$$

Major terms are: source mismatch cost and model parameter cost

Universal data compression

The best possible choice for α is: $\alpha = \frac{1}{2}$.

The smaller we select α, the lesser the cost we pay for the model (parameters).

However, if $\alpha < \frac{1}{2}$ then the source mismatch cost decreases essentially slower than the model cost, thus increasing the overall cost!

We finally obtain

$$r_n \leq \frac{\log_2 n}{2n}(1 + \epsilon(n))$$

where $\epsilon(n) \stackrel{n\to\infty}{\longrightarrow} 0$.

This finally proves the first part of Theorem 4

Universal data compression

Now we must show the second part of Theorem 4.

C is a prefix-free and complete code for binary sequences x^n.
Define the dyadic probabilities as

$$Q_C(x^n) = 2^{-l_C(x^n)}$$

We consider the expected redundancy of C used on sequences from an i.i.d. binary source with parameter θ.

$$\begin{aligned} r_n(C,\theta) &= \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \left(l_C(x^n) + \log_2 p(x^n|\theta)\right) \\ &= \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q_C(x^n)} \end{aligned}$$

Universal data compression

True fact: $\max_x f(x) \geq \mathbf{E}\{f(X)\}$.

Let $w(\theta)$ be an arbitrary distribution over $\theta \in [0, 1]$.

$$\max_{0 \leq \theta \leq 1} r_n(C, \theta) \geq \frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} w(\theta) p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q_C(x^n)} \, d\theta$$

We can still choose the best possible code C. We allow all probabilities for $Q_C(x^n)$ and get another lower bound.

$$\min_C \max_{0 \leq \theta \leq 1} r_n(C, \theta) \geq \min_{Q(x^n)} \max_{0 \leq \theta \leq 1} r_n(Q, \theta) \geq$$

$$\min_{Q(x^n)} \frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} w(\theta) p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q(x^n)} \, d\theta$$

Universal data compression

We can solve this minimization!

$$\min_{Q(x^n)} \frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} w(\theta) p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{Q(x^n)} \, d\theta$$

$$= \min_{Q(x^n)} \frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} w(\theta) p(x^n|\theta)$$

$$\left(\log_2 \frac{1}{Q(x^n)} + \log_2 p(x^n|\theta) \right) \, d\theta$$

Universal data compression

$$
\begin{aligned}
&= \min_{Q(x^n)} \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} \int_0^1 w(\theta) p(x^n|\theta)\, d\theta \log_2 \frac{1}{Q(x^n)} - \\
&\quad \frac{1}{n} \int_0^1 w(\theta) \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \log_2 \frac{1}{p(x^n|\theta)}\, d\theta \\
&= \min_{Q(x^n)} \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} \bar{p}(x^n) \log_2 \frac{1}{Q(x^n)} - \\
&\quad \frac{1}{n} \int_0^1 w(\theta) n h(\theta)\, d\theta \\
&= \frac{1}{n} \sum_{x^n \in \mathcal{X}^n} \bar{p}(x^n) \log_2 \frac{1}{\bar{p}(x^n)} - \int_0^1 w(\theta) h(\theta)\, d\theta
\end{aligned}
$$

Universal data compression

With uniform $w(\theta) = 1$ we find

$$\bar{p}(x^n) = \frac{N(0|x^n)!N(1|x^n)!}{(n+1)!}$$

$$\int_0^1 w(\theta)h(\theta)\,d\theta = \frac{\log_2 e}{2}$$

We know an approximation of the best possible code!

Using this we can now compute a lower bound to the redundancy (complex).

Universal data compression

We start anew:

$$\min_{C} \max_{0 \le \theta \le 1} r_n(C, \theta) \ge$$
$$\frac{1}{n} \int_0^1 \sum_{x^n \in \mathcal{X}^n} p(x^n|\theta) \log_2 \frac{p(x^n|\theta)}{\bar{p}(x^n)} \, d\theta$$

Now we make use of Stirling's approximation

$$\sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} < n! < \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$$

Universal data compression

we obtain (after some appropriate approximations)

$$\bar{p}(x^n) < \sqrt{\frac{\pi}{2n}} e^{\frac{1}{3n}} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k}$$

and so

$$\log_2 \frac{p(x^n|\theta)}{\bar{p}(x^n)} > \frac{1}{2}\log_2 n - \frac{1}{2}\log_2 \frac{\pi}{2} - \frac{\log_2 e}{3n} - nd(\frac{k}{n}||\theta)$$

Universal data compression

Using Lemma 1 we find (for the $d(\frac{k}{n}||\theta)$ part)

$$\sum_{k=0}^{n} \binom{n}{k} \theta^k (1-\theta)^{n-k} \frac{(\frac{k}{n}-\theta)^2}{\theta(1-\theta)} = \frac{\log_2 e}{n}$$

Because k has a Bernoulli distribution with mean $\mu_k = n\theta$ and variance $\sigma_k^2 = n\theta(1-\theta)$ we find

$$= \frac{\log_2 e}{n^2\theta(1-\theta)} \sum_{k=0}^{n} \binom{n}{k} \theta^k (1-\theta)^{n-k} (k-n\theta)^2$$

$$= \frac{\sigma_k^2 \log_2 e}{n^2\theta(1-\theta)} = \frac{\log_2 e}{n}$$

Universal data compression

$$\min_{C} \max_{0 \le \theta \le 1} r_n(C, \theta) > \frac{\log_2 n}{2n} - \frac{\log_2 \frac{\pi}{2}}{n} - \frac{\log_2 e}{n^2} - \frac{\log_2 e}{n}$$

$$= \frac{\log_2 n}{2n} (1 - \epsilon(n))$$

where

$$\epsilon(n) = \frac{2 \log_2 \frac{\pi}{2}}{\log_2 n} + \frac{2 \log_2 e}{n \log_2 n} + \frac{2 \log_2 e}{\log_2 n}$$

$$\epsilon(n) \stackrel{n \to \infty}{\longrightarrow} 0.$$

And this, finally, proves the second part of Theorem 4