Three Concepts: Information

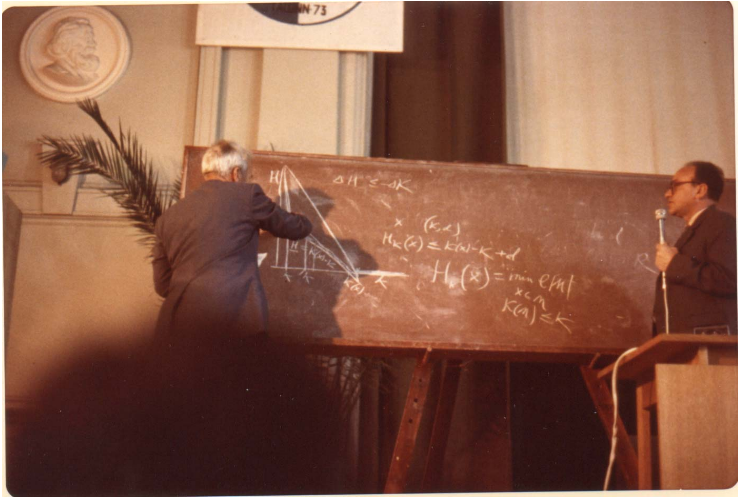
Lecture 6: MDL Principle (contd.)

Teemu Roos

Complex Systems Computation Group
Department of Computer Science, University of Helsinki

Fall 2007

Lecture 6: MDL Principle (contd.)



A.N. Kolmogorov *(left)* introducing the structure function at a Bernoulli Society meeting, Tallinn, 1973

1. Kolmogorov Complexity
   - Definition
   - Basic Properties

1. Kolmogorov Complexity
   - Definition
   - Basic Properties

2. Structure Function
   - Finite Set Models
   - Structure Function
   - Minimal Sufficient Statistic
   - Ideal MDL

1. Kolmogorov Complexity
   - Definition
   - Basic Properties
2. Structure Function
   - Finite Set Models
   - Structure Function
   - Minimal Sufficient Statistic
   - Ideal MDL
3. MDL Principle
   - Definitions
   - Universal Models
   - Prediction & Model Selection

Kolmogorov Complexity

We probably agree that the string

$$1010101010101010101010\ldots10$$

is 'simple'.

Why?

Kolmogorov Complexity

We probably agree that the string

$$1010101010101010101010\ldots10$$

is ‘simple’.

Why?

(One) Solution: The string can be described briefly:

“10 repeated k times”.

Kolmogorov Complexity

We probably agree that the string

$$1010101010101010101010\ldots10$$

is 'simple'.

Why?

(One) Solution: The string can be described briefly:

"10 repeated k times".

Remark: 'Describe' should be understood as meaning "compute by an algorithm" (a formal procedure that halts).

Kolmogorov Complexity

Let $U : \{0,1\}^* \rightarrow \{0,1\}^* \cup \emptyset$ be a computer that given a (binary) program $p \in \{0,1\}^*$ either produces a finite (binary) output $U(p) \in \{0,1\}^*$ or never halts. In the latter case, the output $U(p)$ is said to be undefined ($\emptyset$).

Kolmogorov Complexity

Let $U : \{0,1\}^* \to \{0,1\}^* \cup \emptyset$ be a computer that given a (binary) program $p \in \{0,1\}^*$ either produces a finite (binary) output $U(p) \in \{0,1\}^*$ or never halts. In the latter case, the output $U(p)$ is said to be undefined ($\emptyset$).

Kolmogorov Complexity

For a finite string $x \in \{0,1\}^*$, let $p^*(x)$ be the *shortest* program for which

$$U(p^*(x)) = x \ .$$

The **Kolmogorov complexity** of string x is defined as the length of $p^*(x)$:

$$K_U(x) = \min_{p \,:\, U(p)=x} |p| \ .$$

Kolmogorov Complexity

We assume that the set of programs that halt forms a **prefix-free** set (like symbol codes).

Kolmogorov Complexity

We assume that the set of programs that halt forms a **prefix-free** set (like symbol codes).

The advantage of prefix-free programs is that we can **concatenate** two programs, p and q to form the program pq so that the computer can separate the two programs.

Kolmogorov Complexity

Let U and V be two computers. If computer U is sufficiently 'rich', it can emulate computer V so that it outputs the same output as V for any program p.

Kolmogorov Complexity

Let U and V be two computers. If computer U is sufficiently 'rich', it can emulate computer V so that it outputs the same output as V for any program p.

Universality

A computer U is said to be **universal**, if for *any* other computer V there is a 'translation' program $q \in \{0,1\}^*$ (which depends on V) such that for all programs p we have

$$U(qp) = V(p) \ ,$$

i.e., when given the concatenated program qp, computer U outputs the same string as computer V when given the program p.

Kolmogorov Complexity

For any *universal* computer U, and any other computer V, we have

$$K_U(x) \leq K_V(x) + C \quad ,$$

where C is a constant independent of x.

Kolmogorov Complexity

For any *universal* computer U, and any other computer V, we have

$$K_U(x) \leq K_V(x) + C \ ,$$

where C is a constant independent of x.

Proof: Let q be a the translation program which translates programs of V into programs of U, and let $p^*_V(x)$ be the shortest program for which $V(p^*_V(x)) = x$. Then $U(qp^*_V(x)) = x$ so that

$$K_U(x) \leq |qp^*_V(x)| = |p^*_V(x)| + |q| = K_V(X) + |q| \ . \quad \square$$

Kolmogorov Complexity

For any *universal* computer U, and any other computer V, we have

$$K_U(x) \leq K_V(x) + C \ ,$$

where C is a constant independent of x.

Proof: Let q be a the translation program which translates programs of V into programs of U, and let $p_V^*(x)$ be the shortest program for which $V(p_V^*(x)) = x$. Then $U(qp_V^*(x)) = x$ so that

$$K_U(x) \leq |qp_V^*(x)| = |p_V^*(x)| + |q| = K_V(X) + |q| \ . \quad \square$$

Based on this property, it can be said that Kolmogorov complexity is the length of the **universally** shortest description of x.

Examples

Examples of (virtually) universal 'computers':

Examples

Examples of (virtually) universal 'computers':

1. C (compiler + operating system + computer),

Examples

Examples of (virtually) universal 'computers':

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),

Examples

Examples of (virtually) universal ‘computers’:

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),

Examples

Examples of (virtually) universal 'computers':

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,

Examples

Examples of (virtually) universal ‘computers’:

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,

Examples

Examples of (virtually) universal 'computers':

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,

Examples

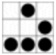
Examples of (virtually) universal 'computers':

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,
7. Arithmetics,

Examples

Examples of (virtually) universal 'computers':

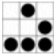
1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,
7. Arithmetics,
8. Game of Life



Examples

Examples of (virtually) universal 'computers':

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,
7. Arithmetics,
8. Game of Life
9. ...

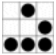


Examples

Examples of (virtually) universal 'computers':

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),

4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,
7. Arithmetics,
8. Game of Life



9. ...

Each of the above can mimic all the others.

Invariance Theorem

From now on we restrict the choice of the computer U in K_U to *universal* computers.

Invariance Theorem

From now on we restrict the choice of the computer U in K_U to *universal* computers.

Invariance Theorem

Kolmogorov complexity is invariant (up to an additive constant) under a change of the universal computer. In other words, for any two universal computers, U and V, there is a constant C such that

$$|K_U(x) - K_V(x)| \leq C \quad \text{for all } x \in \{0,1\}^* \ .$$

Invariance Theorem

From now on we restrict the choice of the computer U in K_U to *universal* computers.

Invariance Theorem

Kolmogorov complexity is invariant (up to an additive constant) under a change of the universal computer. In other words, for any two universal computers, U and V, there is a constant C such that

$$|K_U(x) - K_V(x)| \leq C \quad \text{for all } x \in \{0,1\}^* \ .$$

Proof: Since U is universal, we have $K_U(x) \leq K_V(x) + C_1$. Since V is universal, we have $K_V(x) \leq K_U(x) + C_2$. The theorem follows by setting $C = \max\{C_1, C_2\}$. $\square$

Kolmogorov Complexity

Upper Bound 1

We have the following upper bound on $K_U(x)$:

$$K_U(x) \leq 2|x| + C$$

for some constant C which depends on the computer U but not on the string x.

Kolmogorov Complexity

Upper Bound 1

We have the following upper bound on $K_U(x)$:

$$K_U(x) \leq 2|x| + C$$

for some constant C which depends on the computer U but not on the string x.

Proof: Let q be the program:

`print every even bit that follows`
`until the next odd bit is 0:` $x_1\,1\,x_2\,1\,\ldots\,x_n\,0$.

The length of this program is $2|x| + C$. Prefix-free. □

Kolmogorov Complexity

Upper Bound 2

We have the following upper bound on $K_U(x)$:

$$K_U(x) \leq |x| + 2\log_2 |x| + C$$

for some constant C which depends on the computer U but not on the string x.

Kolmogorov Complexity

Upper Bound 2

We have the following upper bound on $K_U(x)$:

$$K_U(x) \leq |x| + 2\log_2 |x| + C$$

for some constant C which depends on the computer U but not on the string x.

Proof: Let q be the program:

```
read integer n and print the following n bits:
```

$$n_1\, 1 n_2\, 1\, \ldots\, n_{|n|} 0\; x_1\, x_2\, \ldots\, x_n$$

The length of $n = |x|$ is at most $\lceil \log_2 |x| \rceil \leq \log_2 |x| + 1$, so that the length of the program is at most $C' + 2\log_2 |x| + 2 + |x|$. $\square$

Kolmogorov Complexity

Conditional Kolmogorov Complexity

The **conditional Kolmogorov complexity** is defined as the length of the shortest program to print x when y is given:

$$K_U(x \mid y) = \min_{p \,:\, U(\bar{y}\, p) = x} |p| \;,$$

where $\bar{y}$ is a 'self-delimiting' representation of y.

Kolmogorov Complexity

Conditional Kolmogorov Complexity

The **conditional Kolmogorov complexity** is defined as the length of the shortest program to print x when y is given:

$$K_U(x \mid y) = \min_{p \,:\, U(\bar{y}\, p) = x} |p| \ ,$$

where $\bar{y}$ is a 'self-delimiting' representation of y.

Upper Bound 3

We have the following upper bound on $K_U(x \mid |x|)$:

$$K_U(x \mid |x|) \leq |x| + C$$

for some constant C independent x.

Examples

Let $n = |x|$.

Examples

Let $n = |x|$.

1. $K_U(\texttt{0101010101...01} \mid n) = C$.
   Program: `print` $n/2$ `times 01.`

Examples

Let $n = |x|$.

1. $K_U(\texttt{0101010101...01} \mid n) = C$.
   Program: `print` $n/2$ `times 01.`
2. $K_U(\pi_1 \pi_2 \ldots \pi_n \mid n) = C$.
   Program: `print the first` n `bits of` π`.`

Examples

Let $n = |x|$.

1. $K_U(\texttt{0101010101...01} \mid n) = C$.
   Program: `print` $n/2$ `times 01.`
2. $K_U(\pi_1 \pi_2 \ldots \pi_n \mid n) = C$.
   Program: `print the first` n `bits of` π`.`
3. $K_U(\text{English text} \mid n) \approx 1.3 \times n + C$.
   Program: `Huffman code.`
   (Entropy of English is about 1.3 bits per symbol.)

Examples

Let $n = |x|$.

1. $K_U(\texttt{0101010101...01} \mid n) = C$.
   Program: `print` $n/2$ `times 01.`
2. $K_U(\pi_1 \pi_2 \ldots \pi_n \mid n) = C$.
   Program: `print the first` n `bits of` π`.`
3. $K_U(\text{English text} \mid n) \approx 1.3 \times n + C$.
   Program: `Huffman code.`
   (Entropy of English is about 1.3 bits per symbol.)
4. $K_U(\text{fractal}) = C$.
   Program: print # of iterations until $z_{n+1} = z_n^2 + c > T$.

Examples

Martin-Löf Randomness

Examples (contd.):

5. $K_U(x \mid n) \approx n$, for almost all $x \in \{0,1\}^n$.

Martin-Löf Randomness

Examples (contd.):

5. $K_U(x \mid n) \approx n$, for almost all $x \in \{0,1\}^n$.
   Proof: Upper bound $K_U(x \mid n) \leq n + C$. Lower bound by a counting argument: less than 2^{-k} of strings compressible by more than k bits (Lecture 1).

Martin-Löf Randomness

Examples (contd.):

5. $K_U(x \mid n) \approx n$, for almost all $x \in \{0,1\}^n$.
   Proof: Upper bound $K_U(x \mid n) \leq n + C$. Lower bound by a counting argument: less than 2^{-k} of strings compressible by more than k bits (Lecture 1).

Martin-Löf Randomness

String x is said to be **Martin-Löf random** iff $K_u(x \mid n) \geq n$.

Martin-Löf Randomness

Examples (contd.):

5. $K_U(x \mid n) \approx n$, for almost all $x \in \{0,1\}^n$.
   Proof: Upper bound $K_U(x \mid n) \leq n + C$. Lower bound by a counting argument: less than 2^{-k} of strings compressible by more than k bits (Lecture 1).

Martin-Löf Randomness

String x is said to be **Martin-Löf random** iff $K_u(x \mid n) \geq n$.

Consequence of point 5 above: An i.i.d. sequence of unbiased coin flips is with high probability Martin-Löf random.

Universal Prediction

Since the set of valid (halting) programs is required to be **prefix-free** we can consider the probability distribution p_U^n:

$$p_U^n(x) = \frac{2^{-K_U(x|n)}}{C} \quad , \quad \text{where } C = \sum_{x \in \mathcal{X}^n} 2^{-K_U(x|n)} .$$

Universal Prediction

Since the set of valid (halting) programs is required to be **prefix-free** we can consider the probability distribution p_U^n:

$$p_U^n(x) = \frac{2^{-K_U(x|n)}}{C} \ , \quad \text{where } C = \sum_{x \in \mathcal{X}^n} 2^{-K_U(x|n)} .$$

Universal Probability Distribution

The distribution p_U^n is universal in the sense that for any other computable distribution q, there is a constant $C > 0$ such that

$$p_U^n(x) \geq C\, q(x) \quad \text{for all } x \in \mathcal{X}^n .$$

Universal Prediction

Since the set of valid (halting) programs is required to be **prefix-free** we can consider the probability distribution p_U^n:

$$p_U^n(x) = \frac{2^{-K_U(x|n)}}{C} \quad , \quad \text{where } C = \sum_{x \in \mathcal{X}^n} 2^{-K_U(x|n)} .$$

Universal Probability Distribution

The distribution p_U^n is universal in the sense that for any other computable distribution q, there is a constant $C > 0$ such that

$$p_U^n(x) \geq C\, q(x) \quad \text{for all } x \in \mathcal{X}^n .$$

Proof idea: The universal computer U can imitate the Shannon-Fano prefix code with codelengths $\left\lceil \log_2 \frac{1}{q(x)} \right\rceil$.

Universal Prediction

The universal probability distribution p_U^n is a good predictor.

Universal Prediction

The universal probability distribution p_U^n is a good predictor.

This follows from the relationship between codelengths and probabilities (Kraft!):

$K_U(x)$ is small $\Rightarrow$ $p_U^n(x)$ is large

Universal Prediction

The universal probability distribution p_U^n is a good predictor.

This follows from the relationship between codelengths and probabilities (Kraft!):

$$K_U(x) \text{ is small } \Rightarrow p_U^n(x) \text{ is large}$$
$$\Rightarrow \prod_{i=1}^{n} p_U^n(x_i \mid x_1, \ldots, x_{i-1}) \text{ is large}$$

Universal Prediction

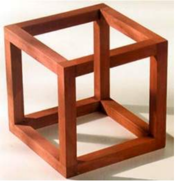
The universal probability distribution p_U^n is a good predictor.

This follows from the relationship between codelengths and probabilities (Kraft!):

$$
\begin{aligned}
K_U(x) \text{ is small } &\Rightarrow\ p_U^n(x) \text{ is large} \\
&\Rightarrow \prod_{i=1}^{n} p_U^n(x_i \mid x_1, \ldots, x_{i-1}) \text{ is large} \\
&\Rightarrow\ p_U^n(x_i \mid x_1, \ldots, x_{i-1}) \text{ is large for most } i \in \{1, \ldots, n\},
\end{aligned}
$$

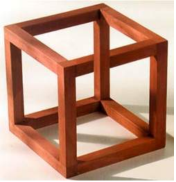
where x_i denotes the ith bit in string x.

Berry Paradox



The smallest integer that cannot be described in ten words?

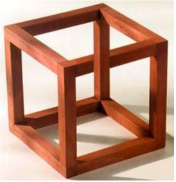
Berry Paradox



The smallest integer that cannot be described in ten words?

Whatever this number is, we have just described (?) it in ten words.

Berry Paradox

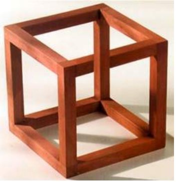


The smallest integer that cannot be described in ten words?

Whatever this number is, we have just described (?) it in ten words.

The smallest uninteresting number?

Berry Paradox



The smallest integer that cannot be described in ten words?

Whatever this number is, we have just described (?) it in ten words.

The smallest uninteresting number?

Whatever this number is, it is quite interesting!

Non-computability

It is impossible to construct a general procedure (algorithm) to compute $K_U(x)$.

Non-Computability

Kolmogorov complexity $K_U : \{0,1\}^* \to \mathbb{N}$ is **non-computable**.

Non-computability

It is impossible to construct a general procedure (algorithm) to compute $K_U(x)$.

Non-Computability

Kolmogorov complexity $K_U : \{0,1\}^* \to \mathbb{N}$ is **non-computable**.

Proof: Assume, by way of contradiction, that it would be possible to compute $K_U(x)$.

Non-computability

It is impossible to construct a general procedure (algorithm) to compute $K_U(x)$.

Non-Computability

Kolmogorov complexity $K_U : \{0,1\}^* \rightarrow \mathbb{N}$ is **non-computable**.

Proof: Assume, by way of contradiction, that it would be possible to compute $K_U(x)$. Then for any $M > 0$, the program

$$\texttt{print a string } x \texttt{ for which } K_U(x) > M\texttt{.}$$

would print a string with $K_U(x) > M$.

Non-computability

It is impossible to construct a general procedure (algorithm) to compute $K_U(x)$.

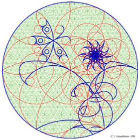
Non-Computability

Kolmogorov complexity $K_U : \{0,1\}^* \to \mathbb{N}$ is **non-computable**.

Proof: Assume, by way of contradiction, that it would be possible to compute $K_U(x)$. Then for any $M > 0$, the program

$$\texttt{print a string } x \texttt{ for which } K_U(x) > M\texttt{.}$$

would print a string with $K_U(x) > M$. A contradiction follows by letting M be larger than the Kolmogorov complexity of this program. Hence, it cannot be possible to compute $K_U(x)$. □



Finite Set Models

Each string $x \in \{0,1\}^n$ can be described in two parts:

1. the regular features of x,

Finite Set Models

Each string $x \in \{0,1\}^n$ can be described in two parts:

1. the regular features of x,
2. the index of x in the set S of strings with those regular features.

Finite Set Models

Each string $x \in \{0,1\}^n$ can be described in two parts:

1. the regular features of x,
2. the index of x in the set S of strings with those regular features.

For set $S \subseteq \{0,1\}^n$, the length of such a two-part description is

$$K_U(S \mid n) + \log_2 |S| + C \geq K_U(x \mid n) \ ,$$

where $K_U(S \mid n)$ is the length of the shortest program to list the members of S (and then halt).

Finite Set Models

Each string $x \in \{0,1\}^n$ can be described in two parts:

1. the regular features of x,
2. the index of x in the set S of strings with those regular features.

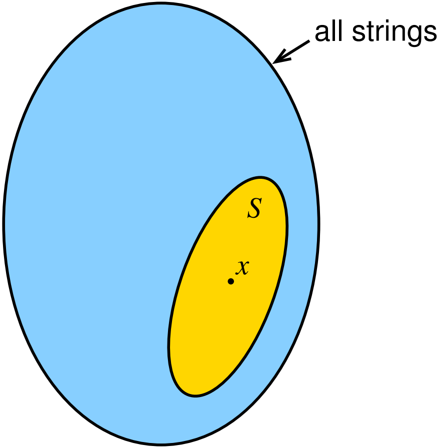
For set $S \subseteq \{0,1\}^n$, the length of such a two-part description is

$$K_U(S \mid n) + \log_2 |S| + C \geq K_U(x \mid n) \ ,$$

where $K_U(S \mid n)$ is the length of the shortest program to list the members of S (and then halt).

We can consider S (regular features) a **model**.

Finite Set Models



Example

For instance, if x is a sequence of biased coin flips, then with high probability the only regular feature is the number of 1s.

Example

For instance, if x is a sequence of biased coin flips, then with high probability the only regular feature is the number of 1s.

Let $S_k^n = \{x \in \{0,1\}^n \ : \ \sum x_i = k\}$. The size of this set is

$$|S_k^n| = \binom{n}{k} = \frac{n!}{k!(n-k)!} \approx 2^{nH(\frac{k}{n})} \ ,$$

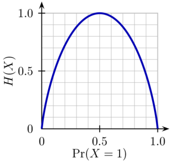
where $H(\frac{k}{n})$ is the binary entropy with parameter $\frac{k}{n}$.

Example

For instance, if x is a sequence of biased coin flips, then with high probability the only regular feature is the number of 1s.

Let $S_k^n = \{x \in \{0,1\}^n \, : \, \sum x_i = k\}$. The size of this set is

$$|S_k^n| = \binom{n}{k} = \frac{n!}{k!(n-k)!} \approx 2^{nH(\frac{k}{n})} \ ,$$



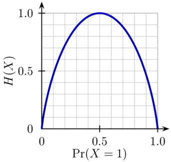
where $H(\frac{k}{n})$ is the binary entropy with parameter $\frac{k}{n}$.

Example

For instance, if x is a sequence of biased coin flips, then with high probability the only regular feature is the number of 1s.

Let $S_k^n = \{x \in \{0,1\}^n \; : \; \sum x_i = k\}$. The size of this set is

$$|S_k^n| = \binom{n}{k} = \frac{n!}{k!(n-k)!} \approx 2^{nH(\frac{k}{n})} \;\; ,$$



where $H(\frac{k}{n})$ is the binary entropy with parameter $\frac{k}{n}$.

Thus, the two-part description has length

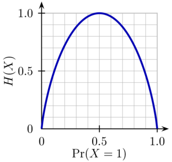
$$K_U(k) + nH(\tfrac{k}{n}) + C \geq K_U(x \mid n) \;\; .$$

Example

For instance, if x is a sequence of biased coin flips, then with high probability the only regular feature is the number of 1s.

Let $S_k^n = \{x \in \{0,1\}^n : \sum x_i = k\}$. The size of this set is

$$|S_k^n| = \binom{n}{k} = \frac{n!}{k!(n-k)!} \approx 2^{nH(\frac{k}{n})} \ ,$$



where $H(\frac{k}{n})$ is the binary entropy with parameter $\frac{k}{n}$.

Thus, the two-part description has length

$$K_U(k) + nH(\tfrac{k}{n}) + C \geq K_U(x \mid n) \ .$$

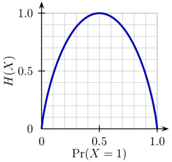
By the **Asymptotic Equipartition Property** (Lecture 3), $nH(\frac{k}{n})$ is with high probability a lower bound ($\approx$) on $K_U(x \mid n)$.

Example

For instance, if x is a sequence of biased coin flips, then with high probability the only regular feature is the number of 1s.

Let $S_k^n = \{x \in \{0,1\}^n \ : \ \sum x_i = k\}$. The size of this set is

$$|S_k^n| = \binom{n}{k} = \frac{n!}{k!(n-k)!} \approx 2^{nH(\frac{k}{n})} \ ,$$



where $H(\frac{k}{n})$ is the binary entropy with parameter $\frac{k}{n}$.

Thus, the two-part description has length

$$K_U(k) + nH(\tfrac{k}{n}) + C \approx K_U(x \mid n) \ .$$

By the **Asymptotic Equipartition Property** (Lecture 3), $nH(\frac{k}{n})$ is with high probability a lower bound ($\approx$) on $K_U(x \mid n)$.

Structure Function

Kolmogorov Structure Function

The **Kolmogorov structure function** is defined as

$$h_x(\alpha) = \min_{\substack{S\,:\,x \in S \\ K_U(S|n) \leq \alpha}} \log_2 |S| \ ,$$

i.e., the log-size of the smallest set containing x that can be described in α bits.

Structure Function

Kolmogorov Structure Function

The **Kolmogorov structure function** is defined as

$$h_x(\alpha) = \min_{\substack{S\,:\,x \in S \\ K_U(S|n) \leq \alpha}} \log_2 |S| \ ,$$

i.e., the log-size of the smallest set containing x that can be described in α bits.

The more bits we can use to describe S, the more regularities we can cover, which makes $|S|$ smaller.

Structure Function

Kolmogorov Structure Function

The **Kolmogorov structure function** is defined as

$$h_x(\alpha) = \min_{\substack{S\,:\,x \in S \\ K_U(S|n) \leq \alpha}} \log_2 |S| \ ,$$

i.e., the log-size of the smallest set containing x that can be described in α bits.

The more bits we can use to describe S, the more regularities we can cover, which makes $|S|$ smaller.

For all $\alpha > 0$, there is a two-part description of length $\alpha + h_x(\alpha)$.

Structure Function

Consider different values of α:

$\alpha \approx 0$:

We can only describe the whole set $\{0,1\}^n$, and not much else, so that $h_x(0) = \log_2 |\{0,1\}^n| = n$.

Structure Function

Consider different values of α:

$\alpha \approx 0$:

We can only describe the whole set $\{0,1\}^n$, and not much else, so that $h_x(0) = \log_2 |\{0,1\}^n| = n$.

$\alpha \approx K_U(x \mid n)$:

We can use the singleton set $S = \{x\}$ since $K_U(\{x\} \mid n) = K_U(x \mid n) + C$. Thus, $h_x(K_U(x)) = \log_2 |\{x\}| = 0$.

Structure Function

Consider different values of α:

$\alpha \approx 0$:
: We can only describe the whole set $\{0,1\}^n$, and not much else, so that $h_x(0) = \log_2 |\{0,1\}^n| = n$.

$\alpha \approx K_U(x \mid n)$:
: We can use the singleton set $S = \{x\}$ since $K_U(\{x\} \mid n) = K_U(x \mid n) + C$. Thus, $h_x(K_U(x)) = \log_2 |\{x\}| = 0$.

$0 < \alpha < K_U(x \mid n)$:
: A two-part description can never be better than optimal:

$$h_x(\alpha) + \alpha \geq K_U(x \mid n) \ .$$

Structure Function

Consider different values of α:

$\alpha \approx 0$:

We can only describe the whole set $\{0,1\}^n$, and not much else, so that $h_x(0) = \log_2 |\{0,1\}^n| = n$.

$\alpha \approx K_U(x \mid n)$:

We can use the singleton set $S = \{x\}$ since $K_U(\{x\} \mid n) = K_U(x \mid n) + C$. Thus, $h_x(K_U(x)) = \log_2 |\{x\}| = 0$.

$0 < \alpha < K_U(x \mid n)$:

A two-part description can never be better than optimal:

$$h_x(\alpha) \geq K_U(x \mid n) - \alpha \ .$$

Structure Function

The **slope** of the structure function $h_x(\alpha)$ is at least as steep as -1 (ignoring constants):

Structure Function

The **slope** of the structure function $h_x(\alpha)$ is at least as steep as -1 (ignoring constants):

For k extra bits in α, we can reduce the set S in a fraction of $\frac{1}{2^k}$ by sorting S alphabetically, dividing in 2^k equal size parts, and encoding the index of the part including x.

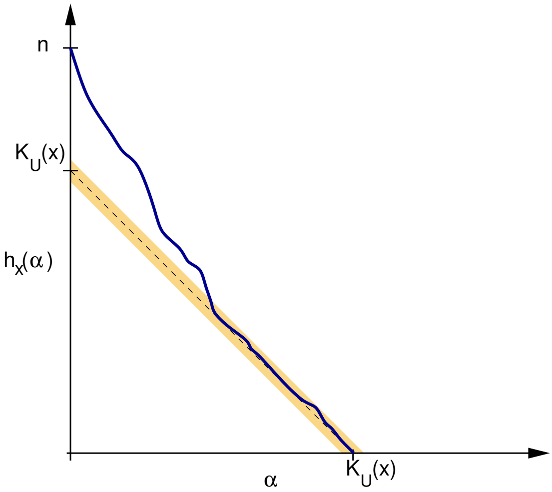
Structure Function

The **slope** of the structure function $h_x(\alpha)$ is at least as steep as -1 (ignoring constants):

For k extra bits in α, we can reduce the set S in a fraction of $\frac{1}{2^k}$ by sorting S alphabetically, dividing in 2^k equal size parts, and encoding the index of the part including x.

The constants related to instructions like "`sort alphabetically and choose second half/third quarter/etc.`" cause **bumps** in the structure function.

Structure Function



[Adapted from (Vereshchagin & Vitányi, 2004)]

Sufficient Statistic

In statistics, a **sufficient statistic** is a function of the data which contains all the information relevant to a parameter. Examples:

Sufficient Statistic

In statistics, a **sufficient statistic** is a function of the data which contains all the information relevant to a parameter. Examples:

1. in coin flipping, the number of 1s is sufficient for the bias parameter.

Sufficient Statistic

In statistics, a **sufficient statistic** is a function of the data which contains all the information relevant to a parameter. Examples:

1. in coin flipping, the number of 1s is sufficient for the bias parameter.
2. in die tossing, the number of times each face is seen is sufficient for the parameters $p_1, \ldots, p_6$.

Sufficient Statistic

In statistics, a **sufficient statistic** is a function of the data which contains all the information relevant to a parameter. Examples:

1. in coin flipping, the number of 1s is sufficient for the bias parameter.
2. in die tossing, the number of times each face is seen is sufficient for the parameters $p_1, \ldots, p_6$.
3. in a Gaussian density, the average $\frac{1}{n} \sum X_i$ is sufficient for the mean.

Sufficient Statistic

In statistics, a **sufficient statistic** is a function of the data which contains all the information relevant to a parameter. Examples:

1. in coin flipping, the number of 1s is sufficient for the bias parameter.
2. in die tossing, the number of times each face is seen is sufficient for the parameters $p_1, \ldots, p_6$.
3. in a Gaussian density, the average $\frac{1}{n} \sum X_i$ is sufficient for the mean.

In all these, the order of the outcomes, for instance, is irrelevant.

Sufficient Statistic

In statistics, a **sufficient statistic** is a function of the data which contains all the information relevant to a parameter. Examples:

1. in coin flipping, the number of 1s is sufficient for the bias parameter.
2. in die tossing, the number of times each face is seen is sufficient for the parameters $p_1, \ldots, p_6$.
3. in a Gaussian density, the average $\frac{1}{n} \sum X_i$ is sufficient for the mean.

In all these, the order of the outcomes, for instance, is irrelevant.

When estimating the parameter, it is *sufficient* to know the sufficient statistic.

Kolmogorov Sufficient Statistic

Kolmogorov Sufficient Statistic

A finite set S is a **Kolmogorov sufficient statistic** iff we have

$$K_U(S \mid n) + \log_2 |S| = K_U(x \mid n) + C \ ,$$

i.e., the two-part description using S is optimal.

Kolmogorov Sufficient Statistic

Kolmogorov Sufficient Statistic

A finite set S is a **Kolmogorov sufficient statistic** iff we have

$$K_U(S \mid n) + \log_2 |S| = K_U(x \mid n) + C \ ,$$

i.e., the two-part description using S is optimal.

A Kolmogorov sufficient statistic tells everything about the *structure* of the data x.

Kolmogorov Sufficient Statistic

Kolmogorov Sufficient Statistic

A finite set S is a **Kolmogorov sufficient statistic** iff we have

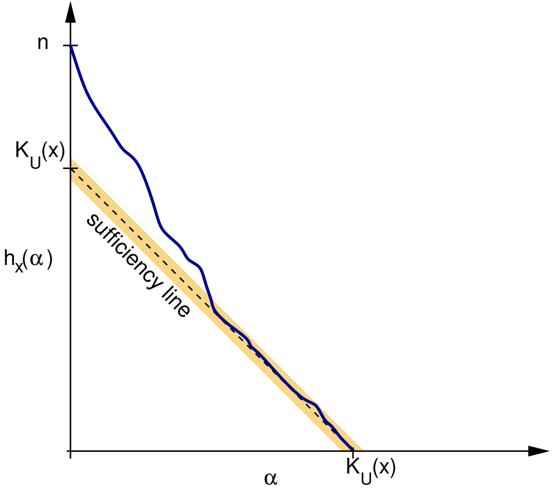
$$K_U(S \mid n) + \log_2 |S| = K_U(x \mid n) + C \quad ,$$

i.e., the two-part description using S is optimal.

A Kolmogorov sufficient statistic tells everything about the *structure* of the data x.

In coin flipping, the number of 1s is with high probability (also) a Kolmogorov sufficient statistic.

Structure Function



[Adapted from (Vereshchagin & Vitányi, 2004)]

Minimal Sufficient Statistic

If S is a Kolmogorov sufficient statistic, i.e., we have

$$K_U(S \mid n) + \log_2 |S| = K_U(x \mid n) \ ,$$

then for all α within the range $K_U(S \mid n) < \alpha \leq K_U(x \mid n)$, there is another sufficient statistic with complexity α:

$$\alpha = K_U(S \mid n) + k \ : \quad K_U(S \mid n) + k + \log_2 \frac{|S|}{2^k} = K_U(x \mid n) \ .$$

Minimal Sufficient Statistic

If S is a Kolmogorov sufficient statistic, i.e., we have

$$K_U(S \mid n) + \log_2 |S| = K_U(x \mid n) \ ,$$

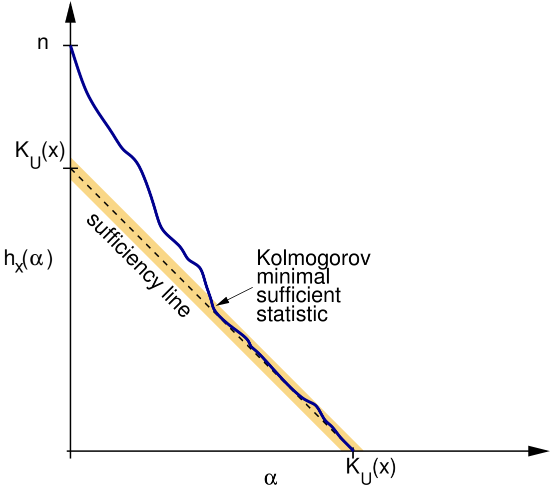
then for all α within the range $K_U(S \mid n) < \alpha \leq K_U(x \mid n)$, there is another sufficient statistic with complexity α:

$$\alpha = K_U(S \mid n) + k \ : \quad K_U(S \mid n) + k + \log_2 \frac{|S|}{2^k} = K_U(x \mid n) \ .$$

Kolmogorov Minimal Sufficient Statistic

The least complex Kolmogorov sufficient statistic is called the **Kolmogorov minimal sufficient statistic**. It contains all the information about the *structure* of x but nothing more.

Structure Function



[Adapted from (Vereshchagin & Vitányi, 2004)]

Example: Mona Lisa

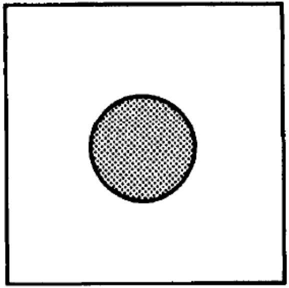
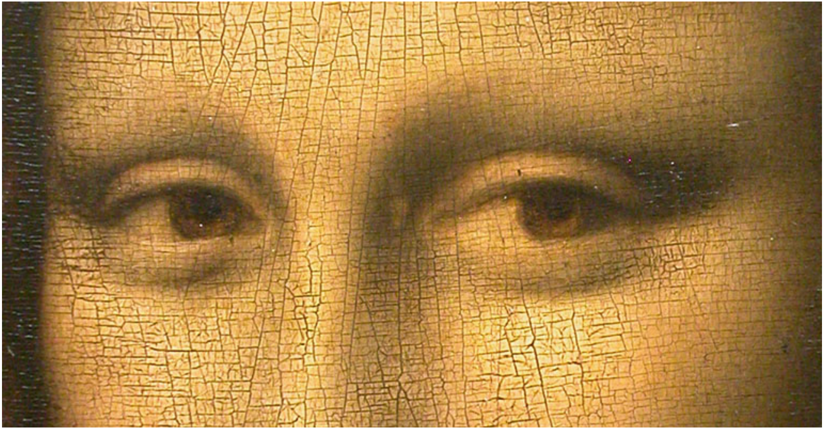


Figure 7.6. Mona Lisa.

Source: Cover & Thomas, 1991

Example: Mona Lisa



Ideal MDL

Ideal MDL

Given data x, choosing the Kolmogorov minimal sufficient statistic as the preferred model is called **"ideal MDL"**.

Ideal MDL

Ideal MDL

Given data x, choosing the Kolmogorov minimal sufficient statistic as the preferred model is called **"ideal MDL"**.

Extracts all regularity from data, and leaves out noise.

Ideal MDL

Ideal MDL

Given data x, choosing the Kolmogorov minimal sufficient statistic as the preferred model is called **"ideal MDL"**.

Extracts all regularity from data, and leaves out noise.

$$\begin{array}{ccccc} \textit{Complexity} & = & \textit{Information} & + & \textit{Noise} \\ & = & \textit{Regularity} & + & \textit{Randomness} \\ & = & \textit{Algorithm} & + & \textit{Compressed file} \end{array}$$

Ideal MDL

The finite set models can be replaced by (computable) probability distributions — distribution p is a *sufficient statistic* iff

$$K_U(p \mid n) + \log_2 \frac{1}{p(x)} = K_U(x \mid n) \ ,$$

i.e., two-part code optimal.

Ideal MDL

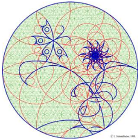
The finite set models can be replaced by (computable) probability distributions — distribution p is a *sufficient statistic* iff

$$K_U(p \mid n) + \log_2 \frac{1}{p(x)} = K_U(x \mid n) \ ,$$

i.e., two-part code optimal.

All things essentially unchanged:

- Finite set S can be replaced by a uniform distribution over S,
- Distribution p can be replaced by the typical set under p.



Ideal vs. Practical MDL

There are two problematic issues with ideal MDL:

Ideal vs. Practical MDL

There are two problematic issues with ideal MDL:

1. Uncomputability of Kolmogorov complexity.

Ideal vs. Practical MDL

There are two problematic issues with ideal MDL:

1. Uncomputability of Kolmogorov complexity.
2. Hidden constants in the definitions and theorems.
   $\Rightarrow$ Says nothing about individual strings.

Ideal vs. Practical MDL

There are two problematic issues with ideal MDL:

1. Uncomputability of Kolmogorov complexity.
2. Hidden constants in the definitions and theorems.
   $\Rightarrow$ Says nothing about individual strings.

Practical MDL aims to solve these issues by:

Ideal vs. Practical MDL

There are two problematic issues with ideal MDL:

1. Uncomputability of Kolmogorov complexity.
2. Hidden constants in the definitions and theorems.
   $\Rightarrow$ Says nothing about individual strings.

Practical MDL aims to solve these issues by:

1. Replace computer programs by probabilistic models.
   $\Rightarrow$ Computable.

Ideal vs. Practical MDL

There are two problematic issues with ideal MDL:

1. Uncomputability of Kolmogorov complexity.
2. Hidden constants in the definitions and theorems.
   $\Rightarrow$ Says nothing about individual strings.

Practical MDL aims to solve these issues by:

1. Replace computer programs by probabilistic models.
   $\Rightarrow$ Computable.
2. Replace universal computer U by a **universal model**.
   $\Rightarrow$ No hidden constants.

Definitions

We call a probability distribution $p : \mathcal{D} \to [0, 1]$ a **model**.

A **model class** $\mathcal{M} = \{p_\theta : \theta \in \Theta\}$ is a set of probability distributions (models).

Definitions

We call a probability distribution $p : \mathcal{D} \rightarrow [0, 1]$ a **model**.

A **model class** $\mathcal{M} = \{p_\theta : \theta \in \Theta\}$ is a set of probability distributions (models).

The model within $\mathcal{M}$ that achieves the shortest codelength for data x is the **maximum likelihood (ML) model**:

$$\min_{\theta \in \Theta} \log_2 \frac{1}{p_\theta(D)} = \log_2 \frac{1}{p_{\hat{\theta}}(D)} \quad .$$

Definitions

We call a probability distribution $p : \mathcal{D} \rightarrow [0, 1]$ a **model**.

A **model class** $\mathcal{M} = \{p_\theta : \theta \in \Theta\}$ is a set of probability distributions (models).

The model within $\mathcal{M}$ that achieves the shortest codelength for data x is the **maximum likelihood (ML) model**:

$$\min_{\theta \in \Theta} \log_2 \frac{1}{p_\theta(D)} = \log_2 \frac{1}{p_{\hat{\theta}}(D)} \ .$$

$\boxed{\text{Depends on } D!}$

Definitions

We call a probability distribution $p : \mathcal{D} \to [0,1]$ a **model**.

A **model class** $\mathcal{M} = \{p_\theta : \theta \in \Theta\}$ is a set of probability distributions (models).

The model within $\mathcal{M}$ that achieves the shortest codelength for data x is the **maximum likelihood (ML) model**:

$$\min_{\theta \in \Theta} \log_2 \frac{1}{p_\theta(D)} = \log_2 \frac{1}{p_{\hat{\theta}}(D)} \quad .$$

$\boxed{\text{Depends on } D!}$

For model q, the excess codelength or "**regret**" over the ML model in $\mathcal{M}$ is given by

$$\log_2 \frac{1}{q(D)} - \log_2 \frac{1}{p_{\hat{\theta}}(D)} \quad .$$

Stochastic Complexity

A model (code) for which the regret grows slower than n is said to be a **universal model** (code) relative to model class $\mathcal{M}$:

$$\lim_{n \to \infty} \frac{1}{n} \left[\log_2 \frac{1}{q(D)} - \log_2 \frac{1}{p_{\hat{\theta}}(D)} \right] = 0 \ .$$

Stochastic Complexity

A model (code) for which the regret grows slower than n is said to be a **universal model** (code) relative to model class $\mathcal{M}$:

$$\lim_{n\to\infty} \frac{1}{n}\left[\log_2 \frac{1}{q(D)} - \log_2 \frac{1}{p_{\hat{\theta}}(D)}\right] = 0 \ .$$

The universal computer U is universal relative to all computers, while a universal model is universal relative to a model class $\mathcal{M}$.

Stochastic Complexity

A model (code) for which the regret grows slower than n is said to be a **universal model** (code) relative to model class $\mathcal{M}$:

$$\lim_{n\to\infty} \frac{1}{n}\left[\log_2 \frac{1}{q(D)} - \log_2 \frac{1}{p_{\hat{\theta}}(D)}\right] = 0 \ .$$

The universal computer U is universal relative to all computers, while a universal model is universal relative to a model class $\mathcal{M}$.

Stochastic Complexity

The **stochastic complexity** of data D relative to model class $\mathcal{M}$ is the codelength achieved by a universal model.

Two-Part Universal Model

The **two-part code** (Lecture 5) consists of

1. optimally quantized parameter values θ^q, and
2. encoding of the data under model p_{θ^q}.

Two-Part Universal Model

The **two-part code** (Lecture 5) consists of

1. optimally quantized parameter values θ^q, and
2. encoding of the data under model p_{θ^q}.

The total codelength is

$$\log_2 \frac{1}{p_{\theta^q}(D)} + \ell(\theta^q) \quad .$$

Two-Part Universal Model

The **two-part code** (Lecture 5) consists of

1. optimally quantized parameter values θ^q, and
2. encoding of the data under model p_{θ^q}.

The total codelength is

$$\log_2 \frac{1}{p_{\theta^q}(D)} + \ell(\theta^q) \ .$$

For 'smooth' parametric models, and optimal quantization, the codelength becomes ($\approx$)

$$\log_2 \frac{1}{p_{\hat{\theta}}(D)} + \frac{k}{2} \log_2 n \ ,$$

so that the regret is $\dfrac{k}{2} \log_2 n$. Since $\log_2 n$ grows slower than n, the **two-part code is universal**.

Mixture Universal Model

There are universal codes that are strictly better than the two-part code.

Mixture Universal Model

There are universal codes that are strictly better than the two-part code.

For instance, given a code for the parameters, let w be a distribution over the parameter space Θ (quantized if necessary) defined as

$$w(\theta) = \frac{2^{-\ell(\theta)}}{c} \ , \quad \text{where } c = \sum_{\theta \in \Theta} 2^{-\ell(\theta)} .$$

Mixture Universal Model

There are universal codes that are strictly better than the two-part code.

For instance, given a code for the parameters, let w be a distribution over the parameter space Θ (quantized if necessary) defined as

$$w(\theta) = \frac{2^{-\ell(\theta)}}{c} \;, \quad \text{where } c = \sum_{\theta \in \Theta} 2^{-\ell(\theta)} .$$

Let p^w be a **mixture distribution** over the data-sets $D \in \mathcal{D}$, defined as

$$p^w(D) = \sum_{\theta \in \Theta} p_\theta(D)\, w(\theta) \;,$$

i.e., an "average" distribution, where each p is weighted by w.

Mixture Universal Model

The codelength of the **mixture model** p^w is given by

$$\log_2 \frac{1}{\sum_{\theta \in \Theta} p(D \mid \theta)\, w(\theta)} \le \log_2 \frac{1}{\max_{\theta \in \Theta} p(D \mid \theta)\, w(\theta)}$$
$$= \log_2 \frac{1}{\max_{\theta \in \Theta} p(D \mid \theta)} + \log_2 \frac{c}{2^{-\ell(\theta)}} \quad .$$

Mixture Universal Model

The codelength of the **mixture model** p^w is given by

$$\log_2 \frac{1}{\sum_{\theta \in \Theta} p(D \mid \theta)\, w(\theta)} \leq \log_2 \frac{1}{\max_{\theta \in \Theta} p(D \mid \theta)\, w(\theta)}$$
$$= \log_2 \frac{1}{\max_{\theta \in \Theta} p(D \mid \theta)} + \log_2 \frac{c}{2^{-\ell(\theta)}} \quad .$$

The right-hand side is equal to

$$\underbrace{\log_2 \frac{1}{p_{\hat{\theta}}(D)} + \ell(\theta)}_{\text{two-part code}} \underbrace{- \log_2 \frac{1}{c}}_{\leq 0} \quad ,$$

Mixture Universal Model

The codelength of the **mixture model** p^w is given by

$$\log_2 \frac{1}{\sum_{\theta\in\Theta} p(D \mid \theta)\, w(\theta)} \le \log_2 \frac{1}{\max_{\theta\in\Theta} p(D \mid \theta)\, w(\theta)}$$
$$= \log_2 \frac{1}{\max_{\theta\in\Theta} p(D \mid \theta)} + \log_2 \frac{c}{2^{-\ell(\theta)}} \quad .$$

The right-hand side is equal to

$$\underbrace{\log_2 \frac{1}{p_{\hat{\theta}}(D)} + \ell(\theta)}_{\text{two-part code}} \underbrace{- \log_2 \frac{1}{c}}_{\le 0} \quad ,$$

The mixture code is always at least as good as the two-part code.

Normalized Maximum Likelihood

Consider the maximum likelihood model

$$p_{\hat{\theta}}(D) = \max_{\theta \in \Theta} p_{\theta}(D) \quad .$$

It is the best probability assignment achievable under model $\mathcal{M}$.

Normalized Maximum Likelihood

Consider the maximum likelihood model

$$p_{\hat{\theta}}(D) = \max_{\theta \in \Theta} p_\theta(D) \ .$$

It is the best probability assignment achievable under model $\mathcal{M}$.

Unfortunately, it is not possible to use the ML model for coding because is not a probability distribution, i.e.,

$$C = \sum_{D \in \mathcal{D}} p_{\hat{\theta}}(D) > 1 \ ,$$

unless $\hat{\theta}$ is constant wrt. D.

Normalized Maximum Likelihood

Normalized Maximum Likelihood

The **normalized maximum likelihood (NML) model** is obtained by normalizing the ML model:

$$p_{\mathrm{nml}}(D) = \frac{p_{\hat{\theta}}(D)}{C} \ , \quad \text{where } C = \sum_{D \in \mathcal{D}} p_{\hat{\theta}}(D) \ .$$

Normalized Maximum Likelihood

Normalized Maximum Likelihood

The **normalized maximum likelihood (NML) model** is obtained by normalizing the ML model:

$$p_{\mathrm{nml}}(D) = \frac{p_{\hat{\theta}}(D)}{C} \ , \quad \text{where } C = \sum_{D \in \mathcal{D}} p_{\hat{\theta}}(D) \ .$$

The regret of NML is given by

$$\log_2 \frac{1}{p_{\mathrm{nml}}(D)} - \log_2 \frac{1}{p_{\hat{\theta}}(D)} = \log_2 \frac{C}{p_{\hat{\theta}}(D)} - \log_2 \frac{1}{p_{\hat{\theta}}(D)} = \log_2 C \ ,$$

which is constant wrt. D.

Normalized Maximum Likelihood

Let q be any distribution other than p_{nml}. Then

- there must a data-set $D' \in \mathcal{D}$ for which we have

$$q(D') < p_{\mathrm{nml}}(D')$$

Normalized Maximum Likelihood

Let q be any distribution other than p_{nml}. Then

- there must a data-set $D' \in \mathcal{D}$ for which we have

$$
\begin{aligned}
& q(D') < p_{\mathrm{nml}}(D') \\
\Leftrightarrow\ & \underbrace{\log_2 \frac{1}{q(D')} - \log_2 \frac{1}{p_{\hat{\theta}}(D')}}_{\text{regret of } q} > \underbrace{\log_2 \frac{1}{p_{\mathrm{nml}}(D')} - \log_2 \frac{1}{p_{\hat{\theta}}(D')}}_{\text{regret of } p_{\mathrm{nml}}} ,
\end{aligned}
$$

Normalized Maximum Likelihood

Let q be any distribution other than p_{nml}. Then

- there must a data-set $D' \in \mathcal{D}$ for which we have

$$q(D') < p_{\mathrm{nml}}(D')$$
$$\Leftrightarrow \underbrace{\log_2 \frac{1}{q(D')} - \log_2 \frac{1}{p_{\hat{\theta}}(D')}}_{\text{regret of } q} > \underbrace{\log_2 \frac{1}{p_{\mathrm{nml}}(D')} - \log_2 \frac{1}{p_{\hat{\theta}}(D')}}_{\text{regret of } p_{\mathrm{nml}}} ,$$

For D', the regret of q is greater than $\log_2 C$, the regret of p_{nml}.

Normalized Maximum Likelihood

Let q be any distribution other than p_{nml}. Then

- there must a data-set $D' \in \mathcal{D}$ for which we have

$$
\begin{aligned}
& q(D') < p_{\mathrm{nml}}(D') \\
\Leftrightarrow\ & \underbrace{\log_2 \frac{1}{q(D')} - \log_2 \frac{1}{p_{\hat{\theta}}(D')}}_{\text{regret of } q} > \underbrace{\log_2 \frac{1}{p_{\mathrm{nml}}(D')} - \log_2 \frac{1}{p_{\hat{\theta}}(D')}}_{\text{regret of } p_{\mathrm{nml}}} ,
\end{aligned}
$$

For D', the regret of q is greater than $\log_2 C$, the regret of p_{nml}.

Thus, the worst-case regret of q is greater than the (worst-case) regret of NML. $\Rightarrow$ NML has the least possible **worst-case regret**.

Universal Models

For 'smooth' parametric models, the regret of NML, $\log_2 C$, grows slower than n, so **NML is also a universal model**.

Universal Models

For 'smooth' parametric models, the regret of NML, $\log_2 C$, grows slower than n, so **NML is also a universal model**.

We have seen three kinds of universal models:

1. two-part,
2. mixture,
3. NML.

Universal Models

For 'smooth' parametric models, the regret of NML, $\log_2 C$, grows slower than n, so **NML is also a universal model**.

We have seen three kinds of universal models:

1. two-part,
2. mixture,
3. NML.

So what do we do with them?

Universal Models

For ‘smooth’ parametric models, the regret of NML, $\log_2 C$, grows slower than n, so **NML is also a universal model**.

We have seen three kinds of universal models:

1. two-part,
2. mixture,
3. NML.

So what do we do with them?

We can use them for (at least) three purposes:

1. compression,

Universal Models

For 'smooth' parametric models, the regret of NML, $\log_2 C$, grows slower than n, so **NML is also a universal model**.

We have seen three kinds of universal models:

1. two-part,
2. mixture,
3. NML.

So what do we do with them?

We can use them for (at least) three purposes:

1. compression,
2. prediction,

Universal Models

For 'smooth' parametric models, the regret of NML, $\log_2 C$, grows slower than n, so **NML is also a universal model**.

We have seen three kinds of universal models:

1. two-part,
2. mixture,
3. NML.

So what do we do with them?

We can use them for (at least) three purposes:

1. compression,
2. prediction,
3. model selection.

MDL Prediction

Prediction is done like in Kolmogorov complexity: universal probability distribution/universal model achieves

- **good compression:** $\ell(D)$ is small,
- **good predictions:** $p(D_i \mid D_1, \ldots, D_{i-1})$ is large for most $i \in \{1, \ldots, n\}$.

MDL Prediction

Prediction is done like in Kolmogorov complexity: universal probability distribution/universal model achieves

- **good compression:** $\ell(D)$ is small,
- **good predictions:** $p(D_i \mid D_1, \ldots, D_{i-1})$ is large for most $i \in \{1, \ldots, n\}$.

For instance, the mixture code gives a natural predictor which is equivalent to **Bayesian prediction**.

MDL Prediction

Prediction is done like in Kolmogorov complexity: universal probability distribution/universal model achieves

- **good compression:** $\ell(D)$ is small,
- **good predictions:** $p(D_i \mid D_1, \ldots, D_{i-1})$ is large for most $i \in \{1, \ldots, n\}$.

For instance, the mixture code gives a natural predictor which is equivalent to **Bayesian prediction**.

The NML model gives predictions that are good relative to the best model in the model class, **no matter what happens**.

MDL Model Selection

Recall (from Lecture 5) the multi-part codes used when multiple model classes, $\mathcal{M}_1, \mathcal{M}_2, \ldots$ are available:

MDL Model Selection

Recall (from Lecture 5) the multi-part codes used when multiple model classes, $\mathcal{M}_1, \mathcal{M}_2, \ldots$ are available:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.

MDL Model Selection

Recall (from Lecture 5) the multi-part codes used when multiple model classes, $\mathcal{M}_1, \mathcal{M}_2, \ldots$ are available:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.
2. Encoding of the parameter (vector): $C_i(\theta)$, $\theta \in \Theta_i$.

MDL Model Selection

Recall (from Lecture 5) the multi-part codes used when multiple model classes, $\mathcal{M}_1, \mathcal{M}_2, \ldots$ are available:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.
2. Encoding of the parameter (vector): $C_i(\theta)$, $\theta \in \Theta_i$.
3. Encoding of the data: $C_\theta(D)$, $D \in \mathcal{D}$.

MDL Model Selection

Recall (from Lecture 5) the multi-part codes used when multiple model classes, $\mathcal{M}_1, \mathcal{M}_2, \ldots$ are available:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.
2. Encoding of the parameter (vector): $C_i(\theta)$, $\theta \in \Theta_i$.
3. Encoding of the data: $C_\theta(D)$, $D \in \mathcal{D}$.

If we are interested in choosing a model class (and not the parameters), we can improve parts 2 & 3 by combining them into a better universal code than two-part:

MDL Model Selection

Recall (from Lecture 5) the multi-part codes used when multiple model classes, $\mathcal{M}_1, \mathcal{M}_2, \ldots$ are available:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.
2. Encoding of the parameter (vector): $C_i(\theta)$, $\theta \in \Theta_i$.
3. Encoding of the data: $C_\theta(D)$, $D \in \mathcal{D}$.

If we are interested in choosing a model class (and not the parameters), we can improve parts 2 & 3 by combining them into a better universal code than two-part:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.

MDL Model Selection

Recall (from Lecture 5) the multi-part codes used when multiple model classes, $\mathcal{M}_1, \mathcal{M}_2, \ldots$ are available:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.
2. Encoding of the parameter (vector): $C_i(\theta)$, $\theta \in \Theta_i$.
3. Encoding of the data: $C_\theta(D)$, $D \in \mathcal{D}$.

If we are interested in choosing a model class (and not the parameters), we can improve parts 2 & 3 by combining them into a better universal code than two-part:

1. Encoding of the model class index: $C_0(i)$, $i \in \mathbb{N}$.
2. Encoding of the data: $C_{\mathcal{M}_i}(D)$, $D \in \mathcal{D}$, where $C_{\mathcal{M}_i}$ is a universal code (e.g., mixture, NML) based on model class $\mathcal{M}_i$.

MDL Model Selection

The idea is the same as in the Kolmogorov minimal sufficient statistic (ideal MDL): **Extract all the structure from the data.**

MDL Model Selection

The idea is the same as in the Kolmogorov minimal sufficient statistic (ideal MDL): **Extract all the structure from the data.**

MDL Explanation of MDL

The success in extracting the structure from data can be measured by the codelength.

MDL Model Selection

The idea is the same as in the Kolmogorov minimal sufficient statistic (ideal MDL): **Extract all the structure from the data.**

MDL Explanation of MDL

The success in extracting the structure from data can be measured by the codelength.

In practical MDL, we only find the structure that is ‘visible’ to the used model class(es). For instance, the Bernoulli (coin flipping) model only sees the number of 1s.

MDL & Bayes

The MDL model selection criterion

$$\text{minimize } \ell(\theta) + \ell_\theta(D)$$

can be interpreted (via $p = 2^{-\ell}$) as

$$\text{maximize } p(\theta) \times p_\theta(D) \quad .$$

MDL & Bayes

The MDL model selection criterion

$$\text{minimize } \ell(\theta) + \ell_\theta(D)$$

can be interpreted (via $p = 2^{-\ell}$) as

$$\text{maximize } p(\theta) \times p_\theta(D) \quad .$$

In Bayesian probability, this is equivalent to **maximization of posterior probability**:

$$p(\theta \mid D) = \frac{p(\theta)\, p(D \mid \theta)}{p(D)} \quad ,$$

where the term $p(D)$ (the marginal probability of D) is constant wrt. θ and doesn't affect model selection.

MDL & Bayes

The MDL model selection criterion

$$\text{minimize } \ell(\theta) + \ell_\theta(D)$$

can be interpreted (via $p = 2^{-\ell}$) as

$$\text{maximize } p(\theta) \times p_\theta(D) \ .$$

In Bayesian probability, this is equivalent to **maximization of posterior probability**:

$$p(\theta \mid D) = \frac{p(\theta)\, p(D \mid \theta)}{p(D)} \ ,$$

where the term $p(D)$ (the marginal probability of D) is constant wrt. θ and doesn't affect model selection.

$\Rightarrow$ **Three Conceps: Probability**

Example: Denoising

$$\begin{aligned} \textit{Complexity} &= \textit{Information} + \textit{Noise} \\ &= \textit{Regularity} + \textit{Randomness} \\ &= \textit{Algorithm} + \textit{Compressed file} \end{aligned}$$

Example: Denoising

$$\begin{aligned} \textit{Complexity} &= \textit{Information} + \textit{Noise} \\ &= \textit{Regularity} + \textit{Randomness} \\ &= \textit{Algorithm} + \textit{Compressed file} \end{aligned}$$

Denoising means the process of removing noise from a signal.

Example: Denoising

$$\begin{aligned} \textit{Complexity} &= \textit{Information} + \textit{Noise} \\ &= \textit{Regularity} + \textit{Randomness} \\ &= \textit{Algorithm} + \textit{Compressed file} \end{aligned}$$

Denoising means the process of removing noise from a signal.

The MDL principle gives a natural method for denoising since the very idea of MDL is to separate the total complexity of a signal into information and noise.

Example: Denoising

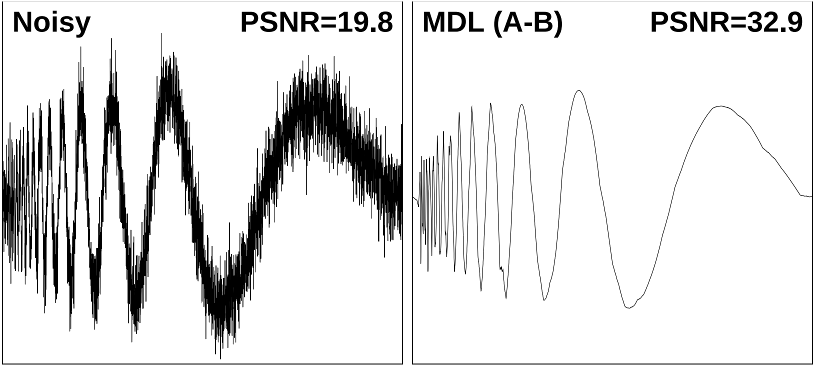
Complexity	=	*Information*	+	*Noise*
	=	*Regularity*	+	*Randomness*
	=	*Algorithm*	+	*Compressed file*

Denoising means the process of removing noise from a signal.

The MDL principle gives a natural method for denoising since the very idea of MDL is to separate the total complexity of a signal into information and noise.

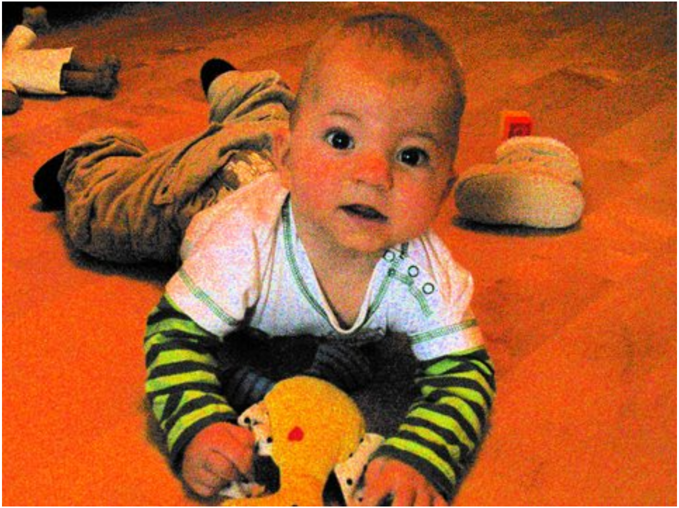
First encode a smooth signal (information), and then the difference to the observed signal (noise).

Example: Denoising

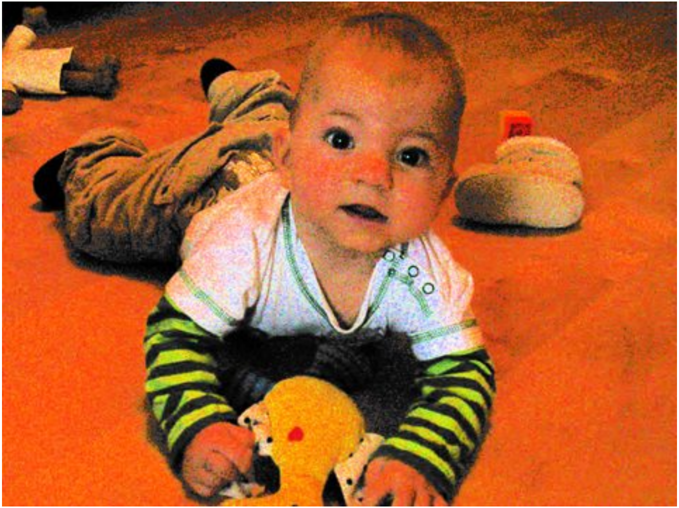


Example: Denoising

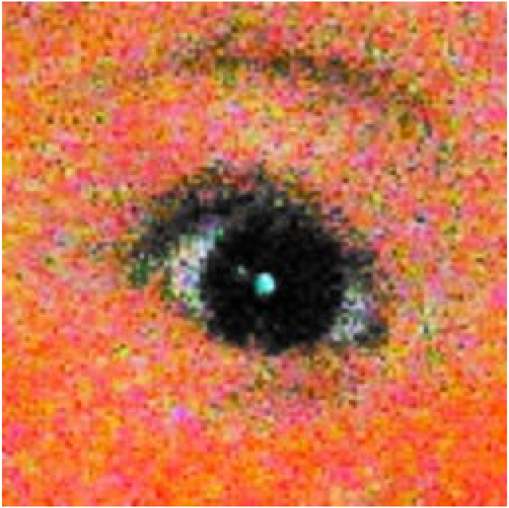
Example: Denoising



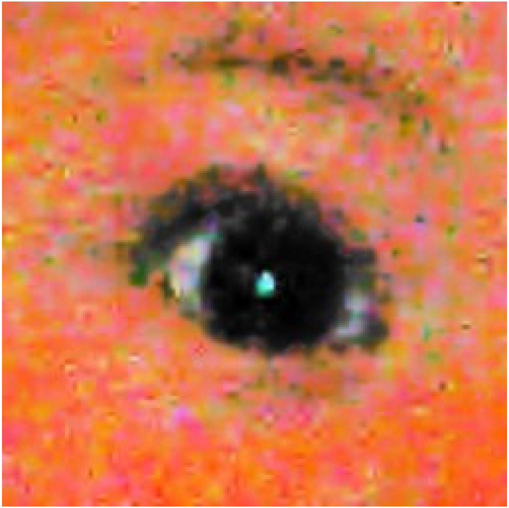
Example: Denoising



Example: Denoising



Example: Denoising



Last Slide

The End.