Information-Theoretic Modeling

Lecture 11: Further Topics

Teemu Roos

Department of Computer Science, University of Helsinki

Fall 2009

Lecture 11: Further Topics

(Peter Falk as *Columbo*, NBC)

1. Kolmogorov Complexity
   - Definition
   - Basic Properties

1. Kolmogorov Complexity
   - Definition
   - Basic Properties

2. Gambling
   - Gambler's Ruin
   - Kelly Criterion

1. Kolmogorov Complexity
   - Definition
   - Basic Properties
2. Gambling
   - Gambler's Ruin
   - Kelly Criterion
3. Lossy Compression
   - Rate–Distortion
   - Image Compression
   - Video Compression

Kolmogorov Complexity

We probably agree that the string

$$1010101010101010101 0\ldots 10$$

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

$$1010101010101010101010\ldots 10$$

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 $U : \{0,1\}^* \rightarrow \{0,1\}^* \cup \emptyset$ 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

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

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

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

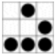
Examples of (virtually) universal 'computers':

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':

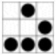
1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,
7. Arithmetics,
8. Game of Life



Examples

Examples of (virtually) universal ‘computers’:

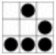
1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,
7. Arithmetics,
8. Game of Life
9. ...



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 $C = \max\{C_1, C_2\}$. □

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 $C' + 2\log_2|x| + 2 + |x|$. □

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. $K_U(0101010101\ldots01 \mid n) = C$.
   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. $K_U(0101010101\ldots01 \mid n) = C$.
   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. $K_U(0101010101\ldots01 \mid n) = C$.
   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:

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

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!):

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

Universal Prediction

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

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

$$K_U(x) \text{ is small} \;\Rightarrow\; p_U^n(x) \text{ is large}$$

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

Universal Prediction

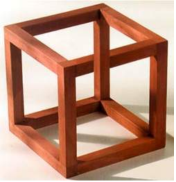
The universal probability distribution p_U^n is a good predictor.

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

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

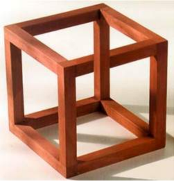
where x_i denotes the ith bit in string x.

Berry Paradox



The smallest integer that cannot be described in ten words?

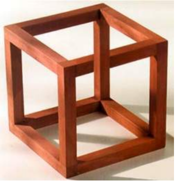
Berry Paradox



The smallest integer that cannot be described in ten words?

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

Berry Paradox

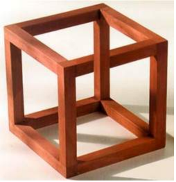


The smallest integer that cannot be described in ten words?

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 $K_U : \{0,1\}^* \to \mathbb{N}$ 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 $K_U(x) > M$.

Non-computability

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

Non-Computability

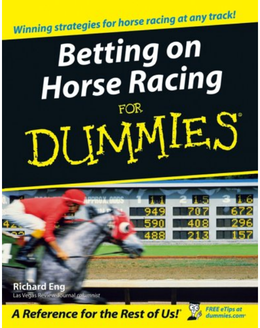
Kolmogorov complexity $K_U : \{0,1\}^* \to \mathbb{N}$ is **non-computable**.

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

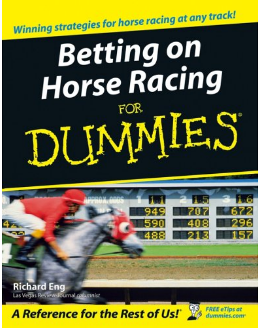
$$\texttt{print a string } x \texttt{ for which } K_U(x) > M\texttt{.}$$

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 $K_U(x)$. □

Gambling

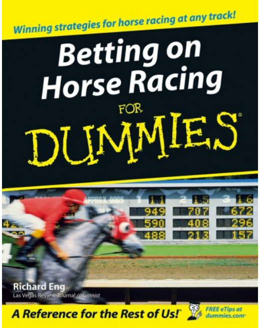

Winning strategies for horse racing at any track!
Betting on Horse Racing
FOR
DUMMIES
Richard Eng
A Reference for the Rest of Us!

Gambling



Bet money b_x on horse x. Get money $\alpha_x b_x$ if x wins (odds).

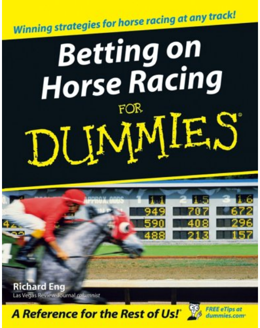
Gambling



Bet money b_x on horse x. Get money $\alpha_x b_x$ if x wins (odds).

Expected win $E[b_x \alpha_x] = \sum p_x \alpha_x b_x$.

Gambling



Bet money b_x on horse x. Get money $\alpha_x b_x$ if x wins (odds).

Expected win $E[b_x \alpha_x] = \sum p_x \alpha_x b_x$.

Maximized by betting everything on $\arg\max p_x \alpha_x$.

Gambling

If odds are "fair", then $\alpha_x = \frac{1}{p_x}$, and hence $p_x \alpha_x b_x = b_x$ for all i.

Gambling

If odds are "fair", then $\alpha_x = \dfrac{1}{p_x}$, and hence $p_x \alpha_x b_x = b_x$ for all i.

Assume now that we have **inside information** about the winning horse.

Gambling

If odds are "fair", then $\alpha_x = \frac{1}{p_x}$, and hence $p_x \alpha_x b_x = b_x$ for all i.

Assume now that we have **inside information** about the winning horse.

In the extreme case, $\hat{X} = X$, we know the outcome:

$$V_n = \alpha_{x_i} \alpha_{x_2} \cdots \alpha_{x_n} V_0$$

where V_t is the capital on tth step

Gambling

If odds are "fair", then $\alpha_x = \frac{1}{p_x}$, and hence $p_x \alpha_x b_x = b_x$ for all i.

Assume now that we have **inside information** about the winning horse.

In the extreme case, $\hat{X} = X$, we know the outcome:

$$V_n = \alpha_{x_i} \alpha_{x_2} \cdots \alpha_{x_n} V_0 = \left(2^G\right)^n V_0$$

where V_t is the capital on tth step, and $G = \frac{\log \sum \alpha_{x_i}}{n}$.

Gambling

If odds are "fair", then $\alpha_x = \frac{1}{p_x}$, and hence $p_x \alpha_x b_x = b_x$ for all i.

Assume now that we have **inside information** about the winning horse.

In the extreme case, $\hat{X} = X$, we know the outcome:

$$V_n = \alpha_{x_i} \alpha_{x_2} \cdots \alpha_{x_n} V_0 = \left(2^G\right)^n V_0$$

exponential rate of growth, G

where V_t is the capital on tth step, and $G = \frac{\log \sum \alpha_{x_i}}{n}$.

Gambling

If the channel is noisy, so that $q_{x_i} = p(x_i \mid \hat{x}_i) < 1$, then our final capital is

$$V_n = \alpha_{x_1}\beta_{x_1|\hat{x}_1}\,\alpha_{x_2}\beta_{x_2|\hat{x}_2}\cdots\alpha_{x_n}\beta_{x_n|\hat{x}_n}V_0,$$

where $\beta_{x_i|\hat{x}_i} = \dfrac{b_{x_i}}{V_{i-1}}$ is the proportion of capital on x_i given $\hat{x}_i$.

Gambling

If the channel is noisy, so that $q_{x_i} = p(x_i \mid \hat{x}_i) < 1$, then our final capital is

$$V_n = \alpha_{x_1}\beta_{x_1|\hat{x}_1}\,\alpha_{x_2}\beta_{x_2|\hat{x}_2}\cdots\alpha_{x_n}\beta_{x_n|\hat{x}_n}V_0,$$

where $\beta_{x_i|\hat{x}_i} = \dfrac{b_{x_i}}{V_{i-1}}$ is the proportion of capital on x_i given $\hat{x}_i$.

Again, expected wealth maximized by betting everything on $\arg\max q_{x_i}\alpha_{x_i}$.

Gambling

If the channel is noisy, so that $q_{x_i} = p(x_i \mid \hat{x}_i) < 1$, then our final capital is

$$V_n = \alpha_{x_1}\beta_{x_1|\hat{x}_1}\,\alpha_{x_2}\beta_{x_2|\hat{x}_2}\cdots\alpha_{x_n}\beta_{x_n|\hat{x}_n}V_0,$$

where $\beta_{x_i|\hat{x}_i} = \dfrac{b_{x_i}}{V_{i-1}}$ is the proportion of capital on x_i given $\hat{x}_i$.

Again, expected wealth maximized by betting everything on $\arg\max q_{x_i}\alpha_{x_i}$.

Gambler's Ruin

This strategy is guaranteed to lead to bankruptcy sooner or later!

Gambling

If the channel is noisy, so that $q_{x_i} = p(x_i \mid \hat{x}_i) < 1$, then our final capital is

$$V_n = \alpha_{x_1}\beta_{x_1|\hat{x}_1}\,\alpha_{x_2}\beta_{x_2|\hat{x}_2}\cdots\alpha_{x_n}\beta_{x_n|\hat{x}_n}V_0,$$

where $\beta_{x_i|\hat{x}_i} = \dfrac{b_{x_i}}{V_{i-1}}$ is the proportion of capital on x_i given $\hat{x}_i$.

Again, expected wealth maximized by betting everything on $\arg\max q_{x_i}\alpha_{x_i}$.

Gambler's Ruin

This strategy is guaranteed to lead to bankruptcy sooner or later!

Conclusion: Maximum expected wealth is not the thing to consider.

Maximum Growth Rate

What if we maximize the average growth rate of capital instead?

$$G = \frac{1}{n} \log \frac{V_n}{V_0} = \frac{1}{n} \log \prod_{i=1}^{n} \alpha_{x_i} \beta_{x_i | \hat{x}_i} .$$

Maximum Growth Rate

What if we maximize the average growth rate of capital instead?

$$G = \frac{1}{n} \log \frac{V_n}{V_0} = \frac{1}{n} \sum_{i=1}^{n} \log \alpha_{x_i} \beta_{x_i | \hat{x}_i} .$$

Maximum Growth Rate

What if we maximize the average growth rate of capital instead?

$$G = \frac{1}{n} \log \frac{V_n}{V_0} = \frac{1}{n} \sum_{i=1}^{n} \log \alpha_{x_i} \beta_{x_i | \hat{x}_i} .$$

With fair odds $\alpha_{x_i} = \frac{1}{p_{x_i}}$, this becomes

$$G = \frac{1}{n} \sum_{i=1}^{n} \log \frac{\beta_{x_i | \hat{x}_i}}{p_{x_i}}$$

Maximum Growth Rate

What if we maximize the average growth rate of capital instead?

$$G = \frac{1}{n} \log \frac{V_n}{V_0} = \frac{1}{n} \sum_{i=1}^{n} \log \alpha_{x_i} \beta_{x_i | \hat{x}_i} .$$

With fair odds $\alpha_{x_i} = \frac{1}{p_{x_i}}$, this becomes

$$E[G] = \frac{1}{n} \sum_{i=1}^{n} E\left[\log \frac{\beta_{x_i | \hat{x}_i}}{p_{x_i}} \right]$$

Maximum Growth Rate

What if we maximize the average growth rate of capital instead?

$$G = \frac{1}{n} \log \frac{V_n}{V_0} = \frac{1}{n} \sum_{i=1}^{n} \log \alpha_{x_i} \beta_{x_i \mid \hat{x}_i} .$$

With fair odds $\alpha_{x_i} = \frac{1}{p_{x_i}}$, this becomes

$$E[G] = \frac{1}{n} \sum_{i=1}^{n} E \left[\log \frac{\beta_{x_i \mid \hat{x}_i}}{p_{x_i}} \right] = \qquad \sum_{x, \hat{x} \in \mathcal{X}} p_{x,\hat{x}} \log \frac{\beta_{x_i \mid \hat{x}_i}}{p_x}$$

Maximum Growth Rate

What if we maximize the average growth rate of capital instead?

$$G = \frac{1}{n} \log \frac{V_n}{V_0} = \frac{1}{n} \sum_{i=1}^{n} \log \alpha_{x_i} \beta_{x_i|\hat{x}_i} .$$

With fair odds $\alpha_{x_i} = \frac{1}{p_{x_i}}$, this becomes

$$E[G] = \frac{1}{n} \sum_{i=1}^{n} E\left[\log \frac{\beta_{x_i|\hat{x}_i}}{p_{x_i}}\right] = \underbrace{\sum_{x,\hat{x} \in \mathcal{X}} p_{x,\hat{x}} \log \frac{\beta_{x_i}}{p_x}}_{\sum_{\hat{x} \in \mathcal{X}} p_{\hat{x}} \sum_{x \in \mathcal{X}} p_{x|\hat{x}} \log \beta_{x_i|\hat{x}_i} + H_p(X)}$$

Maximum Growth Rate

What if we maximize the average growth rate of capital instead?

$$G = \frac{1}{n} \log \frac{V_n}{V_0} = \frac{1}{n} \sum_{i=1}^{n} \log \alpha_{x_i} \beta_{x_i | \hat{x}_i} .$$

With fair odds $\alpha_{x_i} = \frac{1}{p_{x_i}}$, this becomes

$$E[G] = \frac{1}{n} \sum_{i=1}^{n} E\left[\log \frac{\beta_{x_i|\hat{x}_i}}{p_{x_i}}\right] = \underbrace{\sum_{x,\hat{x} \in \mathcal{X}} p_{x,\hat{x}} \log \frac{\beta_{x_i}}{p_x}}_{\sum_{\hat{x} \in \mathcal{X}} p_{\hat{x}} \sum_{x \in \mathcal{X}} p_{x|\hat{x}} \log \beta_{x_i|\hat{x}_i} + H_p(X)}$$

Gibbs' inequality: Maximized by $\beta_{x_i|\hat{x}_i} = q_{x_i} = p_{x_i|\hat{x}_i}$.

Kelly Criterion

Theorem (Kelly, 1956)

Assuming fair odds, $\alpha_x = \dfrac{1}{p_x}$,

1. the growth rate G is maximized by betting proportion $q_x = p(x \mid \hat{x})$ of the capital on $x \in \mathcal{X}$,

Kelly Criterion

Theorem (Kelly, 1956)

Assuming fair odds, $\alpha_x = \dfrac{1}{p_x}$,

1. the growth rate G is maximized by betting proportion $q_x = p(x \mid \hat{x})$ of the capital on $x \in \mathcal{X}$,
2. then the growth rate is given by

$$G = H(X) - H(X \mid \hat{X}),$$

i.e., the channel capacity,

Kelly Criterion

Theorem (Kelly, 1956)

Assuming fair odds, $\alpha_x = \frac{1}{p_x}$,

1. the growth rate G is maximized by betting proportion $q_x = p(x \mid \hat{x})$ of the capital on $x \in \mathcal{X}$,
2. then the growth rate is given by

$$G = H(X) - H(X \mid \hat{X}),$$

   i.e., the channel capacity,
3. gambling using any other strategy will eventually yield less profit.

Kelly Criterion

The same strategy is optimal even if the odds are not fair in the sense $\alpha_x = \frac{1}{p_x}$, as long as there is no "track take", i.e.,

$$\sum_{x \in \mathcal{X}} \frac{1}{\alpha_x} = 1.$$

Kelly Criterion

The same strategy is optimal even if the odds are not fair in the sense $\alpha_x = \frac{1}{p_x}$, as long as there is no "track take", i.e.,

$$\sum_{x \in \mathcal{X}} \frac{1}{\alpha_x} = 1.$$

Note that this implies that you should ignore the odds when betting!

Kelly Criterion

The same strategy is optimal even if the odds are not fair in the sense $\alpha_x = \frac{1}{p_x}$, as long as there is no "track take", i.e.,

$$\sum_{x \in \mathcal{X}} \frac{1}{\alpha_x} = 1.$$

Note that this implies that you should ignore the odds when betting!

The analysis can be extended to the case where there is a "track take", but the results are not quite as neat.

1. Kolmogorov Complexity
   - Definition
   - Basic Properties

2. Gambling
   - Gambler's Ruin
   - Kelly Criterion

3. Lossy Compression
   - Rate–Distortion
   - Image Compression
   - Video Compression

Rate–Distortion

Relax the requirement that the decoder must be able to recover the source string *exactly*.

Rate–Distortion

Relax the requirement that the decoder must be able to recover the source string *exactly*.

What level of **distortion** is tolerated?

Rate–Distortion

Relax the requirement that the decoder must be able to recover the source string *exactly*.

What level of **distortion** is tolerated?

Define a *distortion function* $d : (\mathcal{X}, \mathcal{X}) \to \mathbb{R}^+$, that measures the difference, $d(x, y)$, between a source signal x and the decoded signal y.

Rate–Distortion

Relax the requirement that the decoder must be able to recover the source string *exactly*.

What level of **distortion** is tolerated?

Define a *distortion function* $d : (\mathcal{X}, \mathcal{X}) \rightarrow \mathbb{R}^+$, that measures the difference, $d(x, y)$, between a source signal x and the decoded signal y.

The **rate–distortion function** gives the minimum rate of coding (compression) such that

$$D(X, Y) = E[d(X, Y)] < D^*.$$

Rate–Distortion

Shannon Lower Bound

Continuous case: For squared distortion $d(x, y) = (x - y)^2$, the minimum coding rate is bounded by

$$R(D) \geq h(X) - h(D),$$

where $h(D)$ is the differential entropy of $\mathcal{N}(0, D)$.

Rate–Distortion

Shannon Lower Bound

Continuous case: For squared distortion $d(x,y) = (x-y)^2$, the minimum coding rate is bounded by

$$R(D) \geq h(X) - h(D),$$

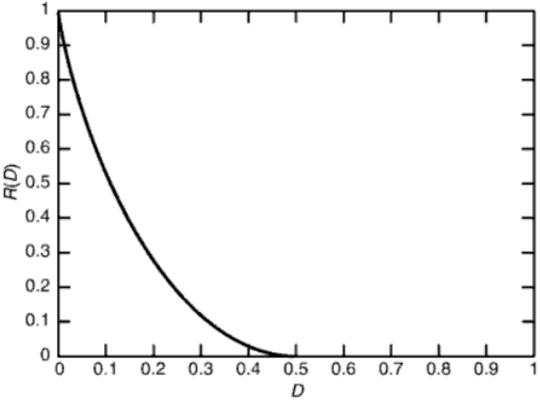
where $h(D)$ is the differential entropy of $\mathcal{N}(0, D)$.

Binary case: For Hamming distortion $d(x,y) = |x-y|$, the minimum coding rate is bounded by

$$R(D) = H(X) - H(D),$$

where $H(\cdot)$ is the binary entropy function.

Rate–Distortion



Rate–distortion function for Bernoulli $\left(\frac{1}{2}\right)$. *Source:* Cover & Thomas.

Image Compression

The key in both noiseless and noisy compression is to find a good model for the source.

Image Compression

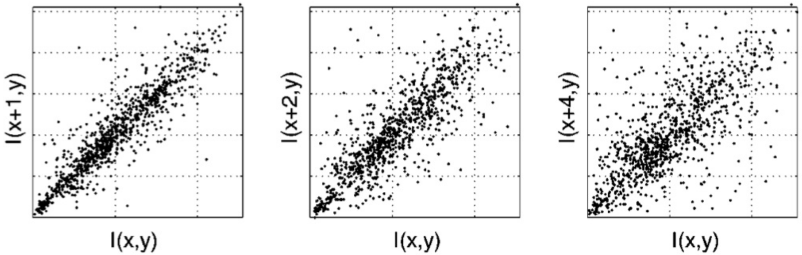
The key in both noiseless and noisy compression is to find a good model for the source.

For images, the correlation of neighboring pixels is one property to exploit.

Image Compression

The key in both noiseless and noisy compression is to find a good model for the source.

For images, the correlation of neighboring pixels is one property to exploit.



Source: Simoncelli & Olshausen, "Natural Image Statistics and Neural Representation", 2001

JPEG Artifacts

JPEG Artifacts

JPEG Artifacts

JPEG Artifacts

JPEG Artifacts

Wavelet Compression

Approximations with Daubechies (N=4) wavelets

1	2	…	164	327	819	1638	3277
(0.01%)	(0.01%)		(1.0%)	(2.0%)	(5.0%)	(10.0%)	(20.0%)

Video Compression

[redacted]

[redacted]

[redacted]

Video compression usually involves:

Video Compression

[redacted]

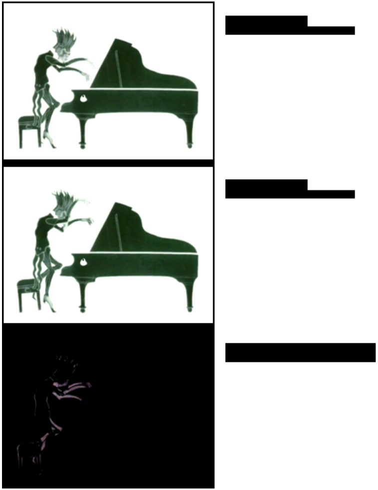
[redacted]

[redacted]

Video compression usually involves:

1. encoding still images using image compression techniques,

Video Compression



Video compression usually involves:

1. encoding still images using image compression techniques,
2. encoding update ("delta") frames to describe what has changed.

Video Compression

Source: `dvd-hq.info`

Last Slide

The End.