comp4620/8620: Advanced Topics in AI
Foundations of Artificial Intelligence

Marcus Hutter
Australian National University
Canberra, ACT, 0200, Australia
http://www.hutter1.net/
2 INFORMATION THEORY & KOLMOGOROV COMPLEXITY

- Philosophical Issues
- Definitions & Notation
- Turing Machines
- Kolmogorov Complexity
- Computability Concepts
- Discussion & Exercises
2.1 **Philosophical Issues: Contents**

- Induction/Prediction Examples
- The Need for a Unified Theory
- On the Foundations of AI and ML
- Example 1: Probability of Sunrise Tomorrow
- Example 2: Digits of a Computable Number
- Example 3: Number Sequences
- Occam’s Razor to the Rescue
- Foundations of Induction
- Sequential/Online Prediction – Setup
- Dichotomies in AI and ML
- Induction versus Deduction
Philosophical Issues: Abstract

I start by considering the philosophical problems concerning machine learning in general and induction in particular. I illustrate the problems and their intuitive solution on various (classical) induction examples. The common principle to their solution is Occam’s simplicity principle. Based on Occam’s and Epicurus’ principle, Bayesian probability theory, and Turing’s universal machine, Solomonoff developed a formal theory of induction. I describe the sequential/online setup considered in this lecture series and place it into the wider machine learning context.
Induction/Prediction Examples

Hypothesis testing/identification: Does treatment X cure cancer? Do observations of white swans confirm that all ravens are black?

Model selection: Are planetary orbits circles or ellipses? How many wavelets do I need to describe my picture well? Which genes can predict cancer?

Parameter estimation: Bias of my coin. Eccentricity of earth’s orbit.

Sequence prediction: Predict weather/stock-quote/... tomorrow, based on past sequence. Continue IQ test sequence like 1,4,9,16,?

Classification can be reduced to sequence prediction: Predict whether email is spam.

Question: Is there a general & formal & complete & consistent theory for induction & prediction?

Beyond induction: active/reward learning, fct. optimization, game theory.
The Need for a Unified Theory
Why do we need or should want a unified theory of induction?

- Finding new rules for every particular (new) problem is cumbersome.
- A plurality of theories is prone to disagreement or contradiction.
- Axiomatization boosted mathematics&logic&deduction and so (should) induction.
- Provides a convincing story and conceptual tools for outsiders.
- Automatize induction&science (that’s what machine learning does)
- By relating it to existing narrow/heuristic/practical approaches we deepen our understanding of and can improve them.
- Necessary for resolving philosophical problems.
- Unified/universal theories are often beautiful gems.
- There is no convincing argument that the goal is unattainable.
On the Foundations of Artificial Intelligence

• Example: Algorithm/complexity theory: The goal is to find fast algorithms solving problems and to show lower bounds on their computation time. Everything is rigorously defined: algorithm, Turing machine, problem classes, computation time, ...

• Most disciplines start with an informal way of attacking a subject. With time they get more and more formalized often to a point where they are completely rigorous. Examples: set theory, logical reasoning, proof theory, probability theory, infinitesimal calculus, energy, temperature, quantum field theory, ...

• Artificial Intelligence: Tries to build and understand systems that learn from past data, make good prediction, are able to generalize, act intelligently, ... Many terms are only vaguely defined or there are many alternate definitions.
Example 1: Probability of Sunrise Tomorrow

What is the probability \( p(1|1^d) \) that the sun will rise tomorrow? 
\( (d = \text{past } \# \text{ days sun rose}, 1 = \text{sun rises. } 0 = \text{sun will not rise}) \)

- \( p \) is undefined, because there has never been an experiment that tested the existence of the sun \textit{tomorrow} (ref. class problem).
- The \( p = 1 \), because the sun rose in all past experiments.
- \( p = 1 - \epsilon \), where \( \epsilon \) is the proportion of stars that explode per day.
- \( p = \frac{d+1}{d+2} \), which is Laplace rule derived from Bayes rule.
- Derive \( p \) from the type, age, size and temperature of the sun, even though we never observed another star with those exact properties.

Conclusion: We predict that the sun will rise tomorrow with high probability independent of the justification.
Example 2: Digits of a Computable Number

- **Extend** 14159265358979323846264338327950288419716939937?

- **Looks random?!**

- **Frequency estimate:** $n =$ length of sequence. $k_i =$ number of occurred $i \iff$ Probability of next digit being $i$ is $\frac{k_i}{n}$. Asymptotically $\frac{k_i}{n} \to \frac{1}{10}$ (seems to be) true.

- **But** we have the strong feeling that (i.e. with high probability) the next digit will be 5 because the previous digits were the expansion of $\pi$.

- **Conclusion:** We prefer answer 5, since we see more structure in the sequence than just random digits.
**Example 3: Number Sequences**

**Sequence:** $x_1, x_2, x_3, x_4, x_5, \ldots$

1, 2, 3, 4, ?, ...  

- $x_5 = 5$, since $x_i = i$ for $i = 1..4$.  
- $x_5 = 29$, since $x_i = i^4 - 10i^3 + 35i^2 - 49i + 24$.

**Conclusion:** We prefer 5, since linear relation involves less arbitrary parameters than 4th-order polynomial.

**Sequence:** 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, ?

- 61, since this is the next prime  
- 60, since this is the order of the next simple group

**Conclusion:** We prefer answer 61, since primes are a more familiar concept than simple groups.

**On-Line Encyclopedia of Integer Sequences:**
http://www.research.att.com/~njas/sequences/
Occam’s Razor to the Rescue

• Is there a unique principle which allows us to formally arrive at a prediction which
  - coincides (always?) with our intuitive guess -or- even better,
  - which is (in some sense) most likely the best or correct answer?

• Yes! Occam’s razor: Use the simplest explanation consistent with past data (and use it for prediction).

• Works! For examples presented and for many more.

• Actually Occam’s razor can serve as a foundation of machine learning in general, and is even a fundamental principle (or maybe even the mere definition) of science.

• Problem: Not a formal/mathematical objective principle. What is simple for one may be complicated for another.
# Dichotomies in Artificial Intelligence

<table>
<thead>
<tr>
<th>scope of this course</th>
<th>scope of other lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>(machine) learning / statistical</td>
<td>logic/knowledge-based (GOFAI)</td>
</tr>
<tr>
<td>online learning</td>
<td>offline/batch learning</td>
</tr>
<tr>
<td>passive prediction</td>
<td>(re)active learning</td>
</tr>
<tr>
<td>Bayes</td>
<td>MDL</td>
</tr>
<tr>
<td>uninformed / universal</td>
<td>informed / problem-specific</td>
</tr>
<tr>
<td>conceptual/mathematical issues</td>
<td>computational issues</td>
</tr>
<tr>
<td>exact/principled</td>
<td>heuristic</td>
</tr>
<tr>
<td>supervised learning</td>
<td>unsupervised</td>
</tr>
<tr>
<td>exploitation</td>
<td>exploration</td>
</tr>
<tr>
<td>action</td>
<td>decision</td>
</tr>
</tbody>
</table>
# Induction ⇔ Deduction

Approximate correspondence between the most important concepts in induction and deduction.

<table>
<thead>
<tr>
<th></th>
<th>Induction</th>
<th>⇔</th>
<th>Deduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type of inference:</strong></td>
<td>generalization/prediction</td>
<td>⇔</td>
<td>specialization/derivation</td>
</tr>
<tr>
<td><strong>Framework:</strong></td>
<td>probability axioms</td>
<td>⇐</td>
<td>logical axioms</td>
</tr>
<tr>
<td><strong>Assumptions:</strong></td>
<td>prior</td>
<td>⇐</td>
<td>non-logical axioms</td>
</tr>
<tr>
<td><strong>Inference rule:</strong></td>
<td>Bayes rule</td>
<td>⇐</td>
<td>modus ponens</td>
</tr>
<tr>
<td><strong>Results:</strong></td>
<td>posterior</td>
<td>⇐</td>
<td>theorems</td>
</tr>
<tr>
<td><strong>Universal scheme:</strong></td>
<td>Solomonoff probability</td>
<td>⇐</td>
<td>Zermelo-Fraenkel set theory</td>
</tr>
<tr>
<td><strong>Universal inference:</strong></td>
<td>universal induction</td>
<td>⇐</td>
<td>universal theorem prover</td>
</tr>
<tr>
<td><strong>Limitation:</strong></td>
<td>incomputable</td>
<td>⇐</td>
<td>incomplete (Gödel)</td>
</tr>
<tr>
<td><strong>In practice:</strong></td>
<td>approximations</td>
<td>⇐</td>
<td>semi-formal proofs</td>
</tr>
<tr>
<td><strong>Operation:</strong></td>
<td>computation</td>
<td>⇐</td>
<td>proof</td>
</tr>
</tbody>
</table>

The foundations of induction are as solid as those for deduction.
2.2 Definitions & Notation: Contents

- Strings and Natural Numbers
- Identification of Strings & Natural Numbers
- Prefix Sets & Codes / Kraft Inequality
- Pairing Strings
- Asymptotic Notation
Strings and Natural Numbers

- \( i, k, n, t \in \mathbb{N} = \{1, 2, 3, \ldots\} \) natural numbers,
- \( \mathbb{B} = \{0, 1\} \) binary alphabet,
- \( x, y, z \in \mathbb{B}^* \) finite binary strings,
- \( \omega \in \mathbb{B}^\infty \) infinite binary sequences,
- \( \epsilon \) for the empty string,
- \( 1^n \) the string of \( n \) ones,
- \( \ell(x) \) for the length of string \( x \),
- \( xy = x \circ y \) for the concatenation of string \( x \) with \( y \).
Identification of Strings & Natural Numbers

- Every countable set is $\cong \mathbb{N}$ (by means of a bijection).

- Interpret a string as a binary representation of a natural number.

- Problem: Not unique: $00101 \cong 5 \cong 101$.

- Use some bijection between natural numbers $\mathbb{N}$ and strings $\mathbb{B}^*$.

- Problem: Not unique when concatenated, e.g.
  $5 \circ 2 \cong 10 \circ 1 = 101 = 1 \circ 01 \cong 2 \circ 4$.

- First-order prefix coding $\overline{x} := 1^\ell(x)0x$.

- Second-order prefix coding $x^i := \overline{\ell(x)x}$. [Elias Delta coding]
### Identification of Strings & Natural Numbers

<table>
<thead>
<tr>
<th>$x \in \mathbb{N}_0$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \in \mathbb{B}^*$</td>
<td>$\epsilon$</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td>000</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\ell(x)$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\bar{x} = 1^{\ell(x)}0x$</td>
<td>0</td>
<td>100</td>
<td>101</td>
<td>11000</td>
<td>11001</td>
<td>11010</td>
<td>11011</td>
<td>1110000</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$x' = \overline{\ell(x)}x$</td>
<td>0</td>
<td>100 0</td>
<td>100 1</td>
<td>101 00</td>
<td>101 01</td>
<td>101 10</td>
<td>101 11</td>
<td>11000 000</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

$x'$ is longer than $\bar{x}$ only for $x < 15$, but shorter for all $x > 30$.

With this identification

\[
\log(x+1) - 1 < \ell(x) \leq \log(x+1).
\]

\[
\ell(\bar{x}) = 2\ell(x) + 1 \leq 2\log(x+1) + 1 \sim 2\log x.
\]

\[
\ell(x') \leq \log(x+1) + 2\log(\log(x+1)+1) + 1 \sim \log x + 2\log\log x.
\]

[Higher order code: Recursively define $\epsilon' := 0$ and $x' := 1[^{\ell(x)-1}'x$]
Prefix Sets & Codes

String $x$ is (proper) prefix of $y$ $:\iff \exists z (\neq \epsilon)$ such that $xz = y$.

Set $\mathcal{P}$ is prefix-free or a prefix code $:\iff$ no element is a proper prefix of another.

Example: A self-delimiting code (e.g. $\mathcal{P} = \{0, 10, 11\}$) is prefix-free.

Kraft Inequality

**Theorem 2.1 (Kraft Inequality)**

For a prefix code $\mathcal{P}$ we have $\sum_{x \in \mathcal{P}} 2^{-\ell(x)} \leq 1$.

Conversely, let $\ell_1, \ell_2, \ldots$ be a countable sequence of natural numbers such that Kraft’s inequality $\sum_{k} 2^{-\ell_k} \leq 1$ is satisfied. Then there exists a prefix code $\mathcal{P}$ with these lengths of its binary code.
Proof of the Kraft-Inequality

Proof ⇒: Assign to each \( x \in \mathcal{P} \) the interval \( \Gamma_x := [0.x, 0.x + 2^{-\ell(x)}) \).

Length of interval \( \Gamma_x \) is \( 2^{-\ell(x)} \).

Intervals are disjoint, since \( \mathcal{P} \) is prefix free, hence

\[
\sum_{x \in \mathcal{P}} 2^{-\ell(x)} = \sum_{x \in \mathcal{P}} \text{Length}(\Gamma_x) \leq \text{Length}([0, 1]) = 1
\]

Proof idea ⇐:

Choose \( l_1, l_2, \ldots \) in increasing order.

Successively chop off intervals of lengths \( 2^{-l_1}, 2^{-l_2}, \ldots \) from left to right from \([0, 1)\) and define left interval boundary as code.
Pairing Strings

- $\mathcal{P} = \{\bar{x} : x \in \mathbb{B}^*\}$ is a prefix code with $\ell(\bar{x}) = 2\ell(x) + 1$.

- $\mathcal{P} = \{x' : x \in \mathbb{B}^*\}$ forms an asymptotically shorter prefix code with $\ell(x') = \ell(x) + 2\ell(\ell(x)) + 1$.

- We pair strings $x$ and $y$ (and $z$) by $\langle x, y \rangle := x'y$ (and $\langle x, y, z \rangle := x'y'z$) which are uniquely decodable, since $x'$ and $y'$ are prefix.

- Since $'$ serves as a separator we also write $f(x, y)$ instead of $f(x'y)$ for functions $f$. 
Asymptotic Notation

- \( f(n) \xrightarrow{n \to \infty} g(n) \) means \( \lim_{n \to \infty} [f(n) - g(n)] = 0 \).
  Say: \( f \) converges to \( g \), w/o implying that \( \lim_{n \to \infty} g(n) \) itself exists.

- \( f(n) \sim g(n) \) means \( \exists 0 < c < \infty : \lim_{n \to \infty} f(n)/g(n) = c \).
  Say: \( f \) is asymptotically proportional to \( g \).

- \( a \lesssim b \) means \( a \) is not much larger than \( b \) (precision unspecified).

- \( f(x) = O(g(x)) \) means \( |f(x)| \leq c|g(x)| \) for some \( c \).
  \( f(x) = o(g(x)) \) means \( \lim_{x \to \infty} f(x)/g(x) = 0 \).

- \( f(x) \leq^* g(x) \) means \( f(x) = O(g(x)) \),
  \( f(x) \preceq^* g(x) \) means \( f(x) \leq g(x) + O(1) \),
  \( f(x) \leq^* g(x) \) means \( f(x) \leq g(x) + O(\log g(x)) \).

- \( f \geq^* g :\iff g \leq^* f \), \( f =^* g :\iff f \leq^* g \land f \geq^* g \), \( \ast \in \{+, \times, \log, \ldots\} \)
2.3 Turing Machines: Contents

- Turing Machines & Effective Enumeration
- Church-Turing Theses
- Short Compiler Assumption
- (Universal) Prefix & Monotone Turing Machine
- Halting Problem
**Turing Machines & Effective Enumeration**

- Turing machine (TM) = (mathematical model for an) idealized computer.

- See e.g. textbook [HMU06]

  Show Turing Machine in Action: TuringBeispielAnimated.gif

- Instruction $i$: If symbol on tape under head is 0/1, write 0/1/- and move head left/right/not and goto instruction=state $j$.

- $\{\text{partial recursive functions}\}$
  $\equiv \{\text{functions computable with a TM}\}$.

- A set of objects $S = \{o_1, o_2, o_3, \ldots\}$ can be (effectively) enumerated:
  $\iff \exists$ TM machine mapping $i$ to $\langle o_i \rangle$,
  where $\langle \rangle$ is some (often omitted) default coding of elements in $S$. 
Church-Turing Theses

The importance of partial recursive functions and Turing machines stems from the following theses:

**Thesis 2.2 (Turing)** Everything that can be reasonably said to be computable by a human using a fixed procedure can also be computed by a Turing machine.

**Thesis 2.3 (Church)** The class of algorithmically computable numerical functions (in the intuitive sense) coincides with the class of partial recursive functions.
**Short Compiler Assumption**

**Assumption 2.4 (Short compiler)**

Given two natural Turing-equivalent formal systems $F_1$ and $F_2$, then there always exists a single short program on $F_2$ which is capable of interpreting all $F_1$-programs.

Lisp, Forth, C, Universal TM, ... have mutually short interpreters.

$\Rightarrow$ equivalence is effective

$\Rightarrow$ size of shortest descriptions essentially the same.

Conversion: Interpreter $\sim$ compiler, by attaching the interpreter to the program to be interpreted and by “selling” the result as a compiled version.
Informality of the Theses & Assumption

• The theses are not provable or falsifiable theorems, since human, reasonable, intuitive, and natural have not been defined rigorously.

• One may define intuitively computable as Turing computable and a natural Turing-equivalent system as one which has a small (say $< 10^5$ bits) interpreter/compiler on a once and for all agreed-upon fixed reference universal Turing machine.

• The theses would then be that these definitions are reasonable.
Prefix Turing Machine

For technical reasons we need the following variants of a Turing machine

**Definition 2.5 (Prefix Turing machine $T$ (pTM))**

- one unidirectional read-only input tape,
- one unidirectional write-only output tape,
- some bidirectional work tapes, initially filled with zeros.
- all tapes are binary (no blank symbol!),
- $T$ halts on input $p$ with output $x :\iff T(p) = x$
- $\iff$ exactly $p$ is to the left of the input head
  and $x$ is to the left of the output head after $T$ halts.
- $\{p : \exists x : T(p) = x\}$ forms a prefix code.
- We call such codes $p$ self-delimiting programs.
Monotone Turing Machine

For technical reasons we need the following variants of a Turing machine

**Definition 2.6 (Monotone Turing machine $T$ (mTM))**

- one unidirectional read-only input tape,
- one unidirectional write-only output tape,
- some bidirectional work tapes, initially filled with zeros.
- all tapes are binary (no blank symbol!),
- $T$ outputs/computes a string starting with $x$ (or a sequence $\omega$) on input $p : \iff T(p) = x*$ (or $T(p) = \omega$) :\iff $p$ is to the left of the input head when the last bit of $x$ is output.
- $T$ may continue operation and need not to halt.
- For given $x$, $\{p : T(p) = x*\}$ forms a prefix code.
- We call such codes $p$ **minimal** programs.
Universal Prefix/Monotone Turing Machine

\( \langle T \rangle := \) some canonical binary coding of (table of rules) of TM \( T \)

\[ \Rightarrow \] set of Turing-machines \( \{ T_1, T_2, \ldots \} \) can be effectively enumerated.

\[ \Rightarrow \exists \]

**Theorem 2.7 (Universal prefix/monotone Turing machine \( U \))**

which simulates (any) pTM/mTM \( T_i \) with input \( y^i q \) if fed with input \( y^i i^i q \), i.e.

\[ U(y^i i^i q) = T_i(y^i q) \quad \forall y, i, q \]

For \( p \neq y^i i^i q \), \( U(p) \) outputs nothing. \( y \) is side information.

Proof: See [HMU06] for normal Turing machines.
Illustration

\[ U = \text{some Personal Computer} \]

\[ T_i = \text{Lisp machine,} \]

\[ q = \text{Lisp program.} \]

\[ y = \text{input to Lisp program.} \]

\[ \Rightarrow T_i(y'q) = \text{execution of Lisp program } q \text{ with input } y \text{ on Lisp machine } T_i. \]

\[ \Rightarrow U(y'i'q) = \text{running on Personal computer } U \text{ the Lisp interpreter } i \text{ with program } q \text{ and input } y. \]

Call one particular prefix/monotone \( U \) the reference UTM.
Halting Problem

We have to pay a big price for the existence of universal TM $U$:

Namely the undecidability of the halting problem [Turing 1936]:

Theorem 2.8 (Halting Problem)
There is no TM $T$: $T(i^{i}p) = 1 \iff T_i(p)$ does not halt.

Proof: Diagonal argument:

Assume such a TM $T$ exists

$\Rightarrow \quad R(i) := T(i^{i}i)$ is computable.

$\Rightarrow \quad \exists j : T_j \equiv R$

$\Rightarrow \quad R(j) = T(j^{j}j) = 1 \iff T_j(j) = R(j)$ does not halt. $\dagger$
2.4 Kolmogorov Complexity: Contents

- Formalization of Simplicity & Complexity
- Prefix Kolmogorov Complexity $K$
- Properties of $K$
- General Proof Ideas
- Monotone Kolmogorov Complexity $Km$
Formalization of Simplicity & Complexity

• **Intuition:** A string is simple if it can be described in a few words, like “the string of one million ones”,

• and is complex if there is no such short description, like for a random string whose shortest description is specifying it bit by bit.

• Effective descriptions or codes ⇒ Turing machines as decoders.

• \( p \) is description/code of \( x \) on \( p \text{TMs} \) \( T :\iff T(p) = x \).

• Length of shortest description: \( K_T(x) := \min_p \{ \ell(p) : T(p) = x \} \).

• This complexity measure depends on \( T \).
Universality/Minimality of $K_U$

Is there a TM which leads to shortest codes among all TMs for all $x$?

Remarkably, there exists a Turing machine (the universal one) which “nearly” has this property:

**Theorem 2.9 (Universality/Minimality of $K_U$)**

\[ K_U(x) \leq K_T(x) + c_{TU}, \]

where $c_{TU} < K_U(T) < \infty$ is independent of $x$

Pair of UTMs $U'$ and $U''$: $|K_{U'}(x) - K_{U''}(x)| \leq c_{U'U''}$.

Assumption 2.4 holds $\iff c_{U'U''}$ small for natural UTMs $U'$ and $U''$.

Henceforth we write $O(1)$ for terms like $c_{U'U''}$. 
Proof of Universality of $K_U$

Proof idea: If $p$ is the shortest description of $x$ under $T = T_i$, then $i'p$ is a description of $x$ under $U$.

Formal proof:
Let $p$ be shortest description of $x$ under $T$, i.e. $\ell(p) = K_T(x)$.

$\exists i: T = T_i \Rightarrow U(i'p) = x \Rightarrow K_U(x) \leq \ell(i'p) = \ell(p) + c_{TU}$ with $c_{TU} := \ell(i')$.

Refined proof:

$p := \arg\min_p \{\ell(p): T(p) = x\} =$ shortest description of $x$ under $T$

$r := \arg\min_p \{\ell(p): U(p) = \langle T\rangle\} =$ shortest description of $T$ under $U$

$q := \text{decode } r \text{ and simulate } T \text{ on } p.$

$\Rightarrow U(qrp) = T(p) = x \Rightarrow$

$K_U(x) \leq \ell(qrp) \overset{\dagger}{=} \ell(p) + \ell(r) = K_T(x) + K_U(\langle T\rangle).$
(Conditional) Prefix Kolmogorov Complexity

Definition 2.10 ((conditional) prefix Kolmogorov complexity)

$K(x) := \min_p \{ \ell(p) : U(p) = x \}$,

$K(x|y) := \min_p \{ \ell(p) : U(y'^p) = x \}$

For (non-string) objects: $K(\text{object}) := K(\langle \text{object} \rangle)$,

e.g. $K(x, y) = K(\langle x, y \rangle) = K(x'y)$. 
Upper Bound on $K$

**Theorem 2.11 (Upper Bound on $K$)**

\[ K(x) \leq \ell(x) + 2\log \ell(x), \quad K(n) \leq \log n + 2\log \log n \]

**Proof:**

There exists a TM $T_{i_0}$ with $i_0 = O(1)$ and $T_{i_0}(\epsilon' x') = x$,

then $U(\epsilon' i_0 x') = x$,

hence $K(x) \leq \ell(\epsilon' i_0 x') \leq \ell(x') < \ell(x) + 2\log \ell(x)$. 

\[ \blacksquare \]
Theorem 2.12 (lower bound for most $n$, Kraft inequality)

$$\sum_{x \in \mathbb{B}^*} 2^{-K(x)} \leq 1, \quad K(x) \geq l(x) \quad \text{for 'most' } x$$

$$K(n) \to \infty \quad \text{for } n \to \infty.$$ 

This is just Kraft’s inequality which implies a lower bound on $K$ valid for ‘most’ $n$.

‘most’ means that there are only $o(N)$ exceptions for $n \in \{1, ..., N\}$. 
**Extra Information & Subadditivity**

**Theorem 2.13 (Extra Information)**

\[ K(x|y) \uparrow \leq K(x) \uparrow \leq K(x, y) \]

Providing side information \( y \) can never increase code length,

Requiring extra information \( y \) can never decrease code length.

**Proof:** Similarly to Theorem 2.11

**Theorem 2.14 (Subadditivity)**

\[ K(xy) \uparrow \leq K(x, y) \uparrow \leq K(x) + K(y|x) \leq K(x) + K(y) \]

Coding \( x \) and \( y \) separately never helps.

**Proof:** Similarly to Theorem 2.13
Theorem 2.15 (Symmetry of Information)

\[ K(x|y, K(y)) + K(y) \overset{\pm}{=} K(x, y) \overset{\pm}{=} K(y, x) \overset{\pm}{=} K(y|x, K(x)) + K(x) \]

Is the analogue of the logarithm of the multiplication rule for conditional probabilities (see later).

Proof: \( \geq = \leq \) similarly to Theorem 2.14.

For \( \leq = \geq \), deep result: see [LV08, Th.3.9.1].
Proof Sketch of $K(y|x) + K(x) \leq K(x, y) + O(\log)$

all $+O(\log)$ terms will be suppressed and ignored. Counting argument:

(1) Assume $K(y|x) > K(x, y) - K(x)$.

(2) $(x, y) \in A := \{\langle u, z \rangle : K(u, z) \leq k\}$, $k := K(x, y)$, $K(k) = O(\log)

(3) $y \in A_x := \{z : K(x, z) \leq k\}$

(4) Use index of $y$ in $A_x$ to describe $y$: $K(y|x) \leq \log |A_x|$

(5) $\log |A_x| > K(x, y) - K(x) =: l$ by (1) and (4), $K(l) = O(\log)

(6) $x \in U := \{u : \log |A_u| > l\}$ by (5)

(7) $\{\langle u, z \rangle : u \in U, z \in A_u\} \subseteq A$

(8) $\log |A| \leq k$ by (2), since at most $2^k$ codes of length $\leq k$

(9) $2^l|U| < \min\{|A_u| : u \in U\}|U| \leq |A| \leq 2^k$ by (6),(7),(8), resp.

(10) $K(x) \leq \log |U| < k - l = K(x)$ by (6) and (9). Contradiction! ■
Information Non-Increase

Theorem 2.16 (Information Non-Increase)

\[ K(f(x)) \overset{+}{\leq} K(x) + K(f) \quad \text{for recursive} \quad f : \mathbb{B}^* \rightarrow \mathbb{B}^* \]

Definition: The Kolmogorov complexity \( K(f) \) of a function \( f \) is defined as the length of the shortest self-delimiting program on a prefix TM computing this function.

Interpretation: Transforming \( x \) does not increase its information content.

Hence: Switching from one coding scheme to another by means of a recursive bijection leaves \( K \) unchanged within additive \( O(1) \) terms.

Proof: Similarly to Theorem 2.14
Coding Relative to Probability Distribution, Minimal Description Length (MDL) Bound

Theorem 2.17 (Probability coding / MDL)

\[ K(x) + \log P(x) + K(P) \]

if \( P : \mathbb{B}^* \rightarrow [0, 1] \) is enumerable and \( \sum_{x \in \mathbb{B}^*} P(x) \leq 1 \)

This is at the heart of the MDL principle [Ris89], which approximates \( K(x) \) by \( -\log P(x) + K(P) \).
Proof of MDL Bound

Proof for $\sum_x P(x) = 1$:

[see [LV08, Sec.4.3] for general $P$]

Idea: Use the Shannon-Fano code based on probability distribution $P$.

Let $s_x := \lceil -\log_2 P(x) \rceil \in \mathbb{N}$

$\Rightarrow \sum_x 2^{-s_x} \leq \sum_x P(x) \leq 1.$

$\Rightarrow: \exists$ prefix code $p$ for $x$ with $\ell(p) = s_x$ (by Kraft inequality)

Since the proof of Kraft inequality for known $\sum_x P(x)$ is (can be made) constructive, there exists an effective prefix code in the sense that

$\exists \ p \mathsf{TM} \ T : \forall x \exists p : T(p) = x$ and $\ell(p) = s_x$.

$\Rightarrow K(x) \leq K_T(x) + K(T) \leq s_x + K(T) < -\log P(x) + K(P)$

where we used Theorem 2.9.
General Proof Ideas

- All upper bounds on $K(z)$ are easily proven by devising some (effective) code for $z$ of the length of the right-hand side of the inequality and by noting that $K(z)$ is the length of the shortest code among all possible effective codes.

- Lower bounds are usually proven by counting arguments (Easy for Thm.2.12 by using Thm.2.1 and hard for Thm.2.15)

- The number of short codes is limited. More precisely: The number of prefix codes of length $\leq \ell$ is bounded by $2^\ell$. 
Remarks on Theorems 2.11-2.17

All (in)equalities remain valid if $K$ is (further) conditioned under some $z$, i.e. $K(... \sim K(...|z)$ and $K(...|y) \sim K(...|y,z)$. 

\[ \log(x) + 2\log(\log(x)) \]

\[ K(x) \]

\[ \log(x) \]
Relation to Shannon Entropy

Let $X, Y \in \mathcal{X}$ be discrete random variables with distribution $P(X, Y)$.

**Definition 2.18 (Definition of Shannon entropy)**

\[
\text{Entropy}(X) \equiv H(X) := - \sum_{x \in \mathcal{X}} P(x) \log P(x)
\]

\[
\text{Entropy}(X|Y) \equiv H(X|Y) := - \sum_{y \in \mathcal{Y}} P(y) \sum_{x \in \mathcal{X}} P(x|y) \log P(x|y)
\]

**Theorem 2.19 (Properties of Shannon entropy)**

- **Upper bound:** $H(X) \leq \log |\mathcal{X}| = n$ for $\mathcal{X} = \mathbb{B}^n$
- **Extra information:** $H(X|Y) \leq H(X) \leq H(X, Y)$
- **Subadditivity:** $H(X, Y) \leq H(X) + H(Y)$
- **Symmetry:** $H(X|Y) + H(Y) = H(X, Y) = H(Y, X)$
- **Information non-increase:** $H(f(X)) \leq H(X)$ for any $f$

Relations for $H$ are essentially expected versions of relations for $K$. 
Monotone Kolmogorov Complexity \( Km \)

A variant of \( K \) is the monotone complexity \( Km(x) \) defined as the shortest program on a monotone TM computing a string starting with \( x \):

\[
Km(x) := \min_p \{ \ell(p) : U(p) = x^* \}
\]

has the following properties:

- \( Km(x) \leq \ell(x) \),
- \( Km(xy) \geq Km(x) \in \mathbb{N}_0 \),
- \( Km(x) \leq \h - \log \mu(x) + K(\mu) \) if \( \mu \) comp. measure (defined later).

It is natural to call an infinite sequence \( \omega \) computable if \( Km(\omega) < \infty \).
2.5 Computability Concepts: Contents

- Computability Concepts
- Computability: Discussion
- (Non)Computability of $K$ and $Km$
Computable Functions

Definition 2.21 (Computable functions) We consider functions \( f : \mathbb{N} \to \mathbb{R} \):

\( f \) is **finitely computable** or **recursive** iff there are Turing machines \( T_{1/2} \) with output interpreted as natural numbers and \( f(x) = \frac{T_1(x)}{T_2(x)} \),

\[ \Downarrow \]

\( f \) is **estimable** iff \( \exists \) recursive \( \phi(\cdot, \cdot) \) \( \forall \varepsilon > 0 : |\phi(x, \lfloor \frac{1}{\varepsilon} \rfloor) - f(x)| < \varepsilon \) \( \forall x \).

\[ \Downarrow \]

\( f \) is **lower semicomputable** or **enumerable** iff \( \phi(\cdot, \cdot) \) is recursive and \( \lim_{t \to \infty} \phi(x, t) = f(x) \) and \( \phi(x, t) \leq \phi(x, t + 1) \).

\[ \Downarrow \]

\( f \) is **approximable** iff \( \phi(\cdot, \cdot) \) is recursive and \( \lim_{t \to \infty} \phi(x, t) = f(x) \).
Computability: Discussion

- What we call **estimable** is often just called **computable**.

- If $f$ is estimable we can determine an interval estimate $f(x) \in [\hat{y} - \varepsilon, \hat{y} + \varepsilon]$.

- If $f$ is only approximable or semicomputable we can still come arbitrarily close to $f(x)$ but we cannot devise a terminating algorithm which produces an $\varepsilon$-approximation.

- $f$ is upper semicomputable or co-enumerable $\iff -f$ is lower semicomputable or enumerable.

- In the case of lower/upper semicomputability we can at least finitely compute lower/upper bounds to $f(x)$.

- In case of approximability, the weakest computability form, even this capability is lost.
(Non)Computability of $K$ and $Km$ complexity

Theorem 2.22 ((Non)computability of $K$ and $Km$ Complexity)
The prefix complexity $K : \mathbb{B}^* \to \mathbb{N}$ and the monotone complexity $Km : \mathbb{B}^* \to \mathbb{N}$ are co-enumerable, but not finitely computable.

Proof: Assume $K$ is computable.
⇒ $f(m) := \min\{n : K(n) \geq m\}$ exists by Theorem 2.12 and is computable (and unbounded).

$K(f(m)) \geq m$ by definition of $f$.

$K(f(m)) \leq K(m) + K(f) < 2\log m$ by Theorem 2.16 and 2.11.
⇒ $m \leq 2\log m + c$ for some $c$, but this is false for sufficiently large $m$.

Co-enumerability of $K$ as exercise.
2.6 Discussion: Contents

- Applications of KC/AIT
- Outlook
- Summary
- Exercises
- Literature
KC/AIT is a Useful Tool in/for

- quantifying simplicity/complexity and Ockham’s razor,
- quantification of Gödel’s incompleteness result,
- computational learning theory,
- combinatorics,
- time and space complexity of computations,
- average case analysis of algorithms,
- formal language and automata theory,
- lower bound proof techniques,
- probability theory,
- string matching,
- clustering by compression,
- physics and thermodynamics of computing,
- statistical thermodynamics / Boltzmann entropy / Maxwell daemon
General Applications of AIT/KC

- (Martin-Löf) randomness of individual strings/sequences/object,
- information theory and statistics of individual objects,
- universal probability,
- general inductive reasoning and inference,
- universal sequence prediction,
- the incompressibility proof method,
- Turing machine complexity,
- structural complexity theory,
- oracles,
- logical depth,
- universal optimal search,
- dissipationless reversible computing,
- information distance,
- algorithmic rate-distortion theory.
Industrial Applications of KC/AIT

- language recognition, linguistics,
- picture similarity,
- bioinformatics,
- phylogeny tree reconstruction,
- cognitive psychology,
- optical / handwritten character recognitions.
Outlook

- Many more KC variants beyond $K$, $K_m$, and $KM$.

- Resource (time/space) bounded (computable!) KC.

- See the excellent textbook [LV08].
Summary

• A quantitative theory of information has been developed.
• Occam’s razor serves as the philosophical foundation of induction and scientific reasoning.
• All enumerable objects are coded as strings.
• Codes need to be prefix free, satisfying Kraft’s inequality.
• Augment Church-Turing thesis with the short compiler assumption.
• Kolmogorov complexity quantifies Occam’s razor, and is the complexity measure.
• Major drawback: $K$ is only semicomputable.
Exercises 1–6

1. [C05] Formulate a sequence prediction task as a classification task (Hint: add time tags).

2. [C15] Complete the table identifying natural numbers with (prefix) strings to numbers up to 16. For which \( x \) is \( x' \) longer/shorter than \( \bar{x} \), and how much?

3. [C10] Show that \( \log(x + 1) - 1 < \ell(x) \leq \log(x + 1) \) and \( \ell(x') \lesssim \log x + 2\log\log x \).

4. [C15] Prove \( \Leftarrow \) of Theorem 2.1

5. [C05] Show that for every string \( x \) there exists a universal Turing machine \( U' \) such that \( K_{U'}(x) = 1 \). Argue that \( U' \) is not a natural Turing machine if \( x \) is complex.

6. [C10] Show \( K(0^n) \overset{+}{=} K(1^n) \overset{+}{=} K(n \text{ digits of } \pi) \overset{+}{=} K(n) \leq \log n + O(\log \log n) \).
Exercises 7–12

7. [C15] The halting sequence $h_{1:\infty}$ is defined as $h_i = 1 \iff T_i(\varepsilon)$ halts, otherwise $h_i = 0$. Show $K(h_1...h_n) \leq 2\log n + O(\log \log n)$ and $Km(h_1...h_n) \leq \log n + O(\log \log n)$.

8. [C25] Show that the Kolmogorov complexity $K$, the halting sequence $h$, and the halting probability $\Omega := \sum_{p: U(p) \text{ halts}} 2^{-\ell(p)}$ are Turing-reducible to each other.

9. [C10–40] Complete the proofs of the properties of $K$.

10. [C15] Show that a function is estimable if and only if it is upper and lower semi-computable.

11. [C10] Prove Theorem 2.20 items 1-2.

12. [C15] Prove the implications in Definition 2.21.
Literature


