Overview

- **Two-Part MDL**
- Two-Part MDL for Grammar Learning
- Two-Part MDL for Probabilistic Hypotheses
- The Big Picture of MDL

Two-Part Code MDL (Rissanen '78)

Given data D, pick the hypothesis $h \in \mathcal{H}$ that minimizes the description length $L(D)$ of the data, which is the sum of:

- the description length $L(h)$ of hypothesis h
- the description length $L(D \mid h)$ of the data D when encoded 'with the help of the hypothesis h'.

$$L(D) \quad = \quad \min_{h \in \mathcal{H}} \quad L(h) \quad + \quad L(D \mid h)$$

Two-Part Code MDL (Rissanen '78)

Given data D, pick the hypothesis $h \in \mathcal{H}$ that minimizes the description length $L(D)$ of the data, which is the sum of:

- the description length $L(h)$ of hypothesis h
- the description length $L(D \mid h)$ of the data D when encoded 'with the help of the hypothesis h'.

$$L(D) = \min_{h \in \mathcal{H}} \underbrace{\boxed{L(h)}}_{\text{complexity}} + \underbrace{\boxed{L(D \mid h)}}_{\text{error}}$$

Two-Part Code MDL (Rissanen ’78)

Given data D, pick the hypothesis $h \in \mathcal{H}$ that minimizes the description length $L(D)$ of the data, which is the sum of:

- the description length $L(h)$ of hypothesis h
- the description length $L(D \mid h)$ of the data D when encoded ‘with the help of the hypothesis h’.

$$L(D) \quad = \quad \min_{h \in \mathcal{H}} \quad \underbrace{\boxed{L(h)}}_{\text{complexity}} \quad + \quad \underbrace{\boxed{L(D \mid h)}}_{\text{error}}$$

- For polynomials, the complexity is related to the degree of the polynomial.
- The error is related to the sum of squared errors / the goodness of fit.

Two-Part Code MDL (Rissanen ’78)

Given data D, pick the hypothesis $h \in \mathcal{H}$ that minimizes the description length $L(D)$ of the data, which is the sum of:

- the description length $L(h)$ of hypothesis h
- the description length $L(D \mid h)$ of the data D when encoded ‘with the help of the hypothesis h’.

$$L(D) = \min_{h\in\mathcal{H}} \underbrace{\boxed{L(h)}}_{\text{complexity}} + \underbrace{\boxed{L(D \mid h)}}_{\text{error}}$$

- For polynomials, the complexity is related to the degree of the polynomial.
- The error is related to the sum of squared errors / the goodness of fit.
- Crucial: Descriptions are based on a **lossless** code.
  (Like (Win)Zip, not like JPG or MP3!)

Two-Part Code MDL (Rissanen ’78)

Given data D, pick the hypothesis $h \in \mathcal{H}$ that minimizes the description length $L(D)$ of the data, which is the sum of:

- the description length $L(h)$ of hypothesis h
- the description length $L(D \mid h)$ of the data D when encoded ‘with the help of the hypothesis h’.

$$L(D) = \min_{h \in \mathcal{H}} \underbrace{\boxed{L(h)}}_{\text{complexity}} + \underbrace{\boxed{L(D \mid h)}}_{\text{error}}$$

Remainder of the lecture: Making $L(h)$ and $L(D \mid h)$ **precise**.

Codes and Codelengths

Code: A code C is a function that maps each object $x \in \mathcal{X}$ to a unique finite binary string $C(x)$.

- For example $C(x) = 010$.
- The 'data alphabet' $\mathcal{X}$: (countable) set of all possible objects that we may wish to encode
- $C(x)$ is called the **codeword** for object x.
- Two different objects cannot have the same codeword. (Otherwise we could not decode the codeword.)

Codelength: The codelength $L_C(x)$ for x is the length (in bits) of the codeword $C(x)$ for object x.

- For example, if $C(x) = 010$, then $L_C(x) = 3$.
- The subscript C emphasizes that this length depends on the code C; It is sometimes omitted.
- In MDL, we always want small codelengths.

Example 1: Uniform Code

Uniform code:

A uniform code assigns codewords of the same length to all objects in $\mathcal{X}$.

Example:

- Let $\mathcal{X} = \{a, b, c, d\}$.
- One possible uniform code for $\mathcal{X}$ is:

$$C(a) = 00, C(b) = 01, C(c) = 10, C(d) = 11$$

Example 1: Uniform Code

Uniform code:

A uniform code assigns codewords of the same length to all objects in $\mathcal{X}$.

Example:

- Let $\mathcal{X} = \{a, b, c, d\}$.
- One possible uniform code for $\mathcal{X}$ is:

$$C(a) = 00, C(b) = 01, C(c) = 10, C(d) = 11$$

- Notice that for all x, $L_C(x) = 2 = \log|\mathcal{X}|$.
- (We always write $\log$ for the logarithm to base 2.
- More generally, we always need $\log n$ bits to encode an element in a set with n elements if we use a uniform code.
- Of course, many other (not necessarily uniform-length) codes are possible as well.

Prefix Codes

Prefix code: A prefix code is a code such that no codeword is a prefix of any other codeword.

Examples:

- Let $\mathcal{X} = \{a, b, c\}$.
- Prefix code: $C(a) = 0, C(b) = 10, C(c) = 11$
- Not a prefix code: $C(a) = 0, C(b) = 01, C(c) = 1$ (because $C(a)$ is a prefix of $C(b)$)

Always use prefix codes:

- Concatenation of two arbitrary codes may not be a code, unless we use comma's to separate codewords:

  For example, 0101 may mean acb, bac, bb, $acac$ in non-prefix code above.
- Concatenation of two prefix codes is again a prefix code.
- If we want to concatenate codes, then we can restrict to prefix codes without loss of generality.
- All description lengths in MDL are based on prefix codes.

Prefix Code for the Integers

Difficulty: The positive integers $1, 2, \ldots$ form an infinite set, so we cannot use a uniform code to encode them. So how to code them?

Inefficient solution:

- $C(x) =$ 'x 1s followed by a 0'
- $L(x) = x + 1$.

Prefix Code for the Integers

Difficulty: The positive integers 1, 2, $\dots$ form an infinite set, so we cannot use a uniform code to encode them. So how to code them?

Inefficient solution:

- $C(x) =$ 'x 1s followed by a 0'
- $L(x) = x + 1$.

Efficient solution:

- $\lceil a \rceil$ denotes rounding up a to the nearest integer.
- First encode $\lceil \log x \rceil$ using the inefficient code.
- This encodes that x is an element of $A = \{2^{\lceil \log x \rceil - 1} + 1, \dots, 2^{\lceil \log(x) \rceil}\}$, which has $2^{\lceil \log x \rceil - 1}$ elements.
- We then use a uniform code for A and get:
- $L(x) = \lceil \log x \rceil + 1 + \log 2^{\lceil \log x \rceil - 1} \approx 2 \log x$.

Overview

- Two-Part MDL
- **Two-Part MDL for Grammar Learning**
- Two-Part MDL for Probabilistic Hypotheses
- The Big Picture of MDL

Making Two-Part MDL Precise

Polynomials:

Making two-part MDL precise for regression with polynomials is quite complicated:

- The parameters of a polynomial are real numbers.
- There are more real numbers than finite binary strings, so we cannot encode them all.
- The solution is to encode the parameters up to a finite precision.
- The precision is chosen to minimize the total description length of the data.

Grammar Learning:

- We will now make two-part MDL precise for grammar learning, for which there are no such complications.

Context-Free Grammars

Idea: A context-free grammar is a set of formal rewriting rules, which naturally captures recursive patterns, like in the grammar of English.

Definition: A context-free grammar (CFG) constists of a tuple $(\mathcal{S}, \mathcal{N}, \mathcal{T}, \mathcal{R})$.

- **Terminals:** $\mathcal{T}$ is a finite set of terminal symbols that stop the recursion. (In our examples these will be English words, like ‘cat’, ‘the’, ‘says’, etc.)
- **Nonterminals:** $\mathcal{N}$ is a finite set of nonterminal symbols, which includes the special starting symbol S. (In our examples these will be parts of English grammar, like ‘N’ (noun), ‘S’ (sentence), etc.)
- **Rules:** $\mathcal{R}$ is a set of rewriting rules of the form $A \longrightarrow B$, where A is a nonterminal and B consists of one or more terminals or nonterminals or nothing (denoted by ϵ). (At least one rule must start with S on the left.)

CFG Example

Abbreviations: The following abbreviations are common: S = sentence, NP = noun phrase, VP = verb phrase, ART = article, N = noun.

A context-free grammar:

- $\mathcal{T} = \{$a, the, man, cat, says, that, bites$\}$
- $\mathcal{N} = \{$S, NP, VP, ART, N$\}$
- Rules:

S $\rightarrow$ NP VP	NP $\rightarrow$ ART N	VP $\rightarrow$ bites NP
ART $\rightarrow$ the	VP $\rightarrow$ bites	VP $\rightarrow$ says that S
ART $\rightarrow$ a	N $\rightarrow$ man	N $\rightarrow$ cat

This grammar can for example generate the sentence “The cat says that a man bites”:

S	(Starting symbol)
S $\rightarrow$ NP VP	(S $\rightarrow$ NP VP)
NP VP $\rightarrow$ ART N VP	(NP $\rightarrow$ ART N)
ART N VP $\rightarrow$ the N VP	(ART $\rightarrow$ the)
the N VP $\rightarrow$ the cat VP	(N $\rightarrow$ cat)
the cat VP $\rightarrow$ the cat says that S	(VP $\rightarrow$ says that S)

CFG Example

Abbreviations: The following abbreviations are common: S = sentence, NP = noun phrase, VP = verb phrase, ART = article, N = noun.

A context-free grammar:

- $\mathcal{T} = \{$a, the, man, cat, says, that, bites$\}$
- $\mathcal{N} = \{$S, NP, VP, ART, N$\}$
- Rules:

S $\rightarrow$ NP VP	NP $\rightarrow$ ART N	VP $\rightarrow$ bites NP
ART $\rightarrow$ the	VP $\rightarrow$ bites	VP $\rightarrow$ says that S
ART $\rightarrow$ a	N $\rightarrow$ man	N $\rightarrow$ cat

This grammar can for example generate the sentence “The cat says that a man bites”:

the cat VP $\rightarrow$ the cat says that S	(VP $\rightarrow$ says that S)
the cat says that S $\rightarrow$ the cat says that NP VP	(S $\rightarrow$ NP VP)
the cat says that NP VP $\rightarrow$ the cat says that ART N VP	(NP $\rightarrow$ ART N)
the cat says that ART N VP $\rightarrow$ the cat says that a N VP	(ART $\rightarrow$ a)
the cat says that a N VP $\rightarrow$ the cat says that a man VP	(N $\rightarrow$ man)
the cat says that a man VP $\rightarrow$ the cat says that a man bites	(VP $\rightarrow$ bites)

Two-Part MDL for Grammar Learning

Problem specification:

- We get a text with n words: $D = t_1, \ldots, t_n$, where each word $t_i \in \mathcal{T}$ is considered a terminal.
- Context-free grammars can be defined not just to generate single sentences, but also to generate entire texts.
- We try to learn the best context-free grammar for this text using the MDL principle.

Applying Two-Part MDL:

- Find the CFG grammar H minimizing $L(H) + L(D \mid H)$.
- To formalise this, we need to design:
  - $L(H)$: a code for encoding CFGs H and
  - $L(D \mid H)$: for each H a code for encoding data 'with the help of H' (making use of the properties of the data that are prescribed by H).

$L(H)$: *Encoding Grammars*

Not optimal, but reasonable:

- Here are some intuitive, reasonable codes that one could use. No claim that these are the 'best', but they are relatively easy to explain.
- Much of modern MDL theory deals with designing 'good' codes.

Encoding $H = (S, \mathcal{N}, \mathcal{T}, \mathcal{R})$:

- Code for $\mathcal{T}$: Will turn out to be irrelevant, so just pick any code $C_{\mathcal{T}}$.
- Codes $C_{\mathcal{N}}$ and $C_{\mathcal{R}}$ for the nonterminals and rules will be specified on the next slide.

$L(H)$: *Encoding Grammars*

Encoding the nonterminals ($C_{\mathcal{N}}$):

- Instead of the standard abbreviations, we use the positive integers $1, \ldots, |\mathcal{N}|$. This does not change which texts can be generated by the grammar. E.g. 1=S, 2=NP, etc.
- To encode the set $\mathcal{N}$ we now only need to encode $|\mathcal{N}|$. Using the efficient code for the integers: $L_{C_{\mathcal{N}}}(\mathcal{N}) = 2\log|\mathcal{N}|$.

Encoding the rules ($C_{\mathcal{R}}$):

- First encode the number of rules: $2\log|\mathcal{R}|$ bits.
- Then encode all nonterminals on the left-hand-side of a rule using the uniform code on $\mathcal{N}$: $|\mathcal{R}| \cdot \log|\mathcal{N}|$ bits.
- Then (non)terminals on right-hand-side (RHS) of rules:

$$\sum_{i=1}^{|\mathcal{R}|} 2\log R_i + R_i \log(|\mathcal{T} \cup \mathcal{N}|) \text{ bits},$$

where R_i is the nr. of elements on the RHS of the ith rule.

$L(D \mid H)$: Encoding Data Given H

H specifies grammatically correct texts:

- To encode data $D = t_1, \ldots, t_n$ literally, we need $n \log |\mathcal{T}| = \log |\mathcal{T}|^n$ bits, since there are $|\mathcal{T}|^n$ possible texts of length n.
- But grammars H impose **constraints** on the set of texts that are allowed. For example, in English, articles cannot be followed by verbs, nouns cannot be followed by articles etc.
- Because of these constraints, the number of grammatically correct texts will be exponentially smaller than $|\mathcal{T}|^n$.

Using H to compress the data:

- First encode n: $2 \log n$ bits.
- Then encode D using a uniform code on all **grammatically correct** texts of length n, where grammatically correct means that the text can be generated by grammar H.
- This takes $\log($nr. of correct texts of length $n)$ bits.

Learning the Best Grammar

We will use MDL to choose between three grammars. Does it find the right one?

- **Promiscuous grammar:**
  This grammar accepts any text of any length: For all $t \in \mathcal{T}$, it contains a rule S $\rightarrow$ t S, and an additional rule S $\rightarrow$ ϵ (the empty string). (Solomonoff, 1964)

Learning the Best Grammar

We will use MDL to choose between three grammars. Does it find the right one?

- **Promiscuous grammar:** $\boxed{\text{Terrible underfitting!}}$
  This grammar accepts any text of any length: For all $t \in \mathcal{T}$, it contains a rule S $\rightarrow$ t S, and an additional rule S $\rightarrow$ ϵ (the empty string). (Solomonoff, 1964)

Learning the Best Grammar

We will use MDL to choose between three grammars. Does it find the right one?

- **Promiscuous grammar:** $\boxed{\text{Terrible underfitting!}}$
  This grammar accepts any text of any length: For all $t \in \mathcal{T}$, it contains a rule S $\rightarrow$ t S, and an additional rule S $\rightarrow$ ϵ (the empty string). (Solomonoff, 1964)
- **Ad hoc grammar:**
  The grammar that accepts only the training text D, and nothing else: Only contains the rule
  S $\rightarrow$ $t_1, \ldots, t_n$.

Learning the Best Grammar

We will use MDL to choose between three grammars. Does it find the right one?

- **Promiscuous grammar:** $\boxed{\text{Terrible underfitting!}}$
  This grammar accepts any text of any length: For all $t \in \mathcal{T}$, it contains a rule S $\rightarrow$ t S, and an additional rule S $\rightarrow$ ϵ (the empty string). (Solomonoff, 1964)
- **Ad hoc grammar:** $\boxed{\text{Terrible overfitting!}}$
  The grammar that accepts only the training text D, and nothing else: Only contains the rule
  S $\rightarrow$ $t_1, \ldots, t_n$.

Learning the Best Grammar

We will use MDL to choose between three grammars. Does it find the right one?

- **Promiscuous grammar:** $\boxed{\text{Terrible underfitting!}}$
  This grammar accepts any text of any length: For all $t \in \mathcal{T}$, it contains a rule S $\rightarrow$ t S, and an additional rule S $\rightarrow$ ϵ (the empty string). (Solomonoff, 1964)
- **Ad hoc grammar:** $\boxed{\text{Terrible overfitting!}}$
  The grammar that accepts only the training text D, and nothing else: Only contains the rule
  S $\rightarrow$ $t_1, \ldots, t_n$.
- **The 'right' grammar:** A good CFG approximation of the real English grammar. (Of course, no perfect CFG for English grammar is possible, but we can get close.) Note that the size of this grammar does not depend on the length n of the text.

What MDL Does

MDL selects the right grammar: Given enough data (large enough n), the total description length $L(H) + L(D \mid H)$ will be much smaller for the 'right' grammar than for either the ad hoc or for the promiscuous grammar.

Explanation:

- **Promiscuous grammar:** Every text is allowed, so $L(D \mid H) \geq n \log |\mathcal{T}|$. Hence $L(H) + L(D \mid H)$ is longer than a literal description of the data. We haven't compressed at all!
- **Ad hoc grammar:** Note that $R_1 = n$, so $L(H) \geq L_{C_\mathcal{R}}(\mathcal{R}) \geq R_1 \log |\mathcal{T} \cup \mathcal{T}| \geq n \log |\mathcal{T}|$. Again we haven't compressed at all!
- **The 'right' grammar:** The size of the right grammar doesn't depend on n, so $L(H)$ is some constant. And $L(D \mid H)$ grows much slower than $n \log |\mathcal{T}|$, because the number of grammatically correct texts is exponentially smaller than the number of possible texts.

Discussion of the Grammar Learning Example

MDL avoided overfitting:

- The promiscuous grammar was rejected because it did not help in compressing the data ($L(D \mid h)$ was too big).
- Even though the ad hoc grammar fit the data very well ($L(D \mid H)$ was very small), it was rejected because the grammar itself was much too complex ($L(H)$ was too big).
- MDL selected the ‘right’ grammar, which struck the right balance between complexity and goodness of fit.

The limits of this example:

- The example does not show what MDL will do if we use it to select a grammar from all possible CFG grammars.
- However, it does show that MDL strongly prefers the right grammar over the silly promiscuous and ad hoc grammars.
- This illustrates how compressing the data protects against overfitting.

Overview

Probabilistic Hypotheses Are Better

Noise causes complications:

- If the data has noise, then the approach for grammars that we just sketched will fail.
- Reason: Noise causes grammatically incorrect texts.
- And grammatically incorrect texts cannot be encoded using the ‘right’ grammar.

Probabilistic hypotheses are better:

- To counter this, it is better to work with probabilistic hypotheses that take the noise into account.
- For example, we could use probabilistic grammars where each rule ‘fires’ with a certain probability.
- (The idea roughly: high probability for grammatically correct rules; low probability for rules that describe noise.)

Codelengths and Probabilities

Few objects can have small codelength:

- If we store our data on a computer, then it is represented internally as a binary sequence. Without loss of generality we can assume that our data is already a binary sequence.
- There are 2^m binary sequences of m bits and $\sum_{i=0}^{a} 2^i = 2^{a+1} - 1$ binary sequences of length at most a.
- By taking $a = m - (k+1)$ we see that the fraction of binary sequences of length m that can be compressed by more than k bits is less than $2^{m-k}/2^m = 1/2^k$, which is very small for large k.

Few objects can have large probability: The probabilities for all objects have to sum to 1.

This suggests an analogy: This analogy can be made precise by **Kraft's inequality**, which relates P to a code C such that $L_C(x) = -\log P(x)$.

Two-Part MDL for Probabilistic Hypotheses

Deterministic hypotheses: For a hypothesis space $\mathcal{H}$ containing deterministic hypotheses, two-part MDL tells us to select the hypothesis that achieves:

$$\min_{H \in \mathcal{H}} L(H) + L(D \mid H)$$

Probabilistic hypotheses:

- Let $\mathcal{M}$ be a model, which contains probabilistic hypotheses.
- Using Kraft's inequality, two-part MDL tells us to select the probabilistic hypothesis achieving:

$$\min_{P \in \mathcal{M}} L(P) - \log P(D)$$

Two-Part MDL for Probabilistic Hypotheses

Deterministic hypotheses: For a hypothesis space $\mathcal{H}$ containing deterministic hypotheses, two-part MDL tells us to select the hypothesis that achieves:

$$\min_{H\in\mathcal{H}} L(H) + L(D \mid H)$$

Probabilistic hypotheses:

- Let $\mathcal{M}$ be a model, which contains probabilistic hypotheses.
- Using Kraft's inequality, two-part MDL tells us to select the probabilistic hypothesis achieving:

$$\min_{P\in\mathcal{M}} L(P) - \log P(D)$$

Penalised maximum likelihood: Minimizing $-\log P(D)$ is equivalent to maximizing $P(D)$, so MDL can be viewed as a form of penalised maximum likelihood. The penalty of each probabilistic hypothesis P depends on its complexity $L(P)$.

Overview

Relation to Bayes

Two-part MDL amounts to a form of Bayesian MAP with a particular choice of prior.[1]

How Bayes avoid overfitting:

- If you use a large model, then almost every probabilistic hypothesis in the model has to get small prior probability.
- Reason: prior probabilities have to sum up to one.

This is similar to:
How MDL avoids overfitting:

- If you use a large model, then almost every (probabilistic) hypothesis H in the model has to get a large codelength $L(H)$.
- Reason: There exist only two codewords of length 1, only four of length 2, etc.

[1]For very large (uncountably infinite) models, there are some technical details about coding the hypotheses only to finite precision.

Modern MDL

MDL is more than two-part codes:

- We have seen two-part codes: Code the data using the hypothesis that minimizes $L(H) + L(D \mid H)$.[2]
- They are (the oldest) special case of **universal codes**.
- More generally, MDL may be based on universal codes.

Universal codes:

- A universal code C for a model $\mathcal{M}$ is a code such that:
  - If there exists a hypothesis $P \in \mathcal{M}$ that can be used (by Kraft's inequality) to compress the data well,
  - Then C also compresses the data (almost as) well.
- Sometimes other universal codes are better than two-part codes.

[2] Mitchell also has a section on two-part code MDL, which you do not have to study.

References

- P. Grünwald, “The Minimum Description Length Principle”, 2007
- T.M. Cover and J.A. Thomas, “Elements of Information Theory,” 1991