Minimum Description Length Principle

With An Application For Credit Risk Models

Paul van Leeuwen (paul.vanleeuwen@devolksbank.nl)

16 May 2023

Introduction

The Speaker

- Paul van Leeuwen, employee of dVB as of September 2022.
  - KRT's RDS 5.0 and Modellen voor Bankbalans.
- Up to 2017 part of the Modelling team at dVB.
- After that as Model Validator at Achmea and Lead Data Scientist at Wageningen University & Research.
- Now self-employed and currently working in the financial sector and providing R workshops.

Why this subject?

- Improve current approaches and modelling techniques.
- Stay in the forefront of statistical innovation.
- Connection with other realms of statistics.

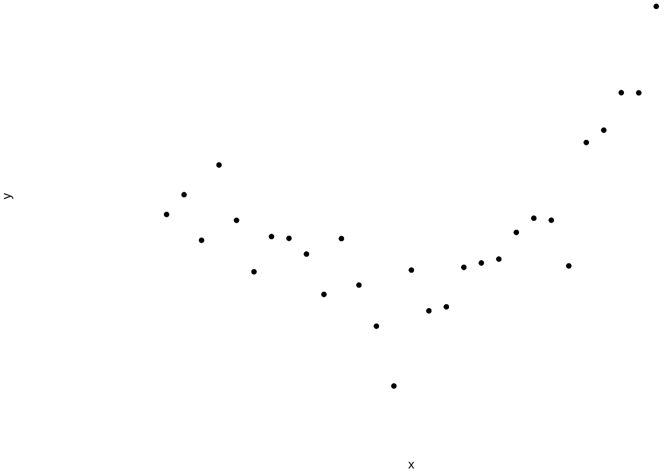
What is MDL?

Introduction

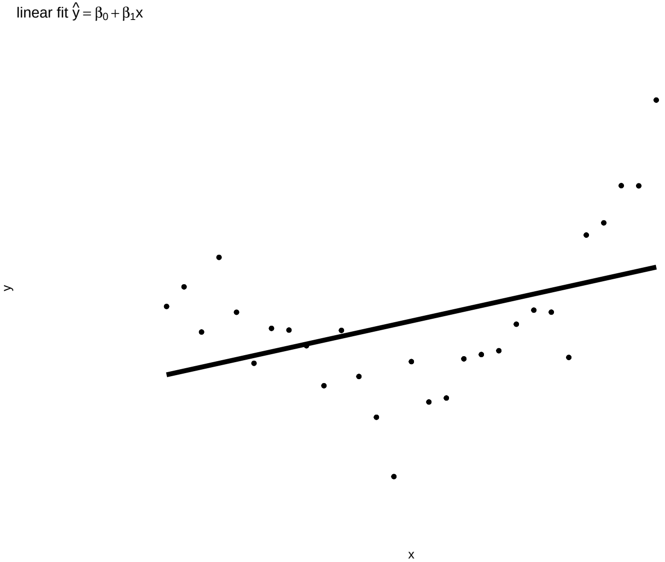
- The Minimum Description Length (MDL) principle aims to describe the data and its description mechanism with the smallest possible information 'length'.
- Applications (among else):
  - model selection (e.g. what order of the Markov model family do we want?);
  - deal with overfitting (e.g. how many explanatory variables to include);
  - exploratory data analysis (what prior knowledge can we confirm?).
- Close ties with frequentist statistics, Bayesian statistics, and machine learning.
- Why is MDL relatively unknown?
  - MDL is the intersection of advanced measure theory, information theory, and statistics.
  - For a decent introduction into MDL, see (Grünwald 2007).

Example of dealing with overfitting

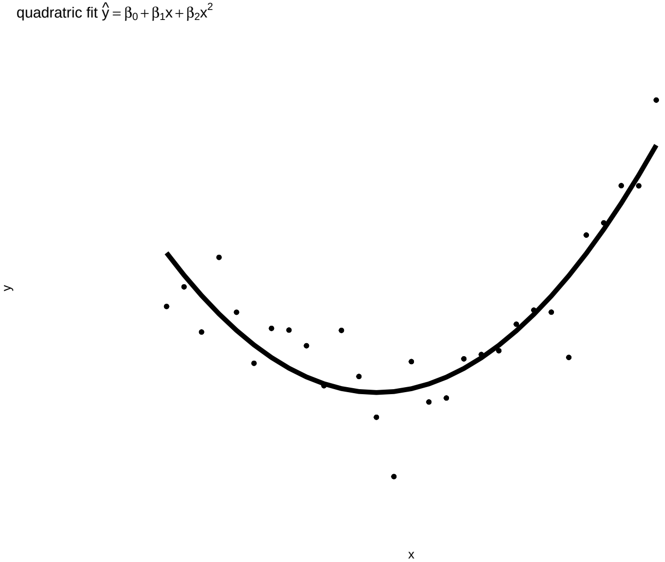
What polynomial generated by this dataset?



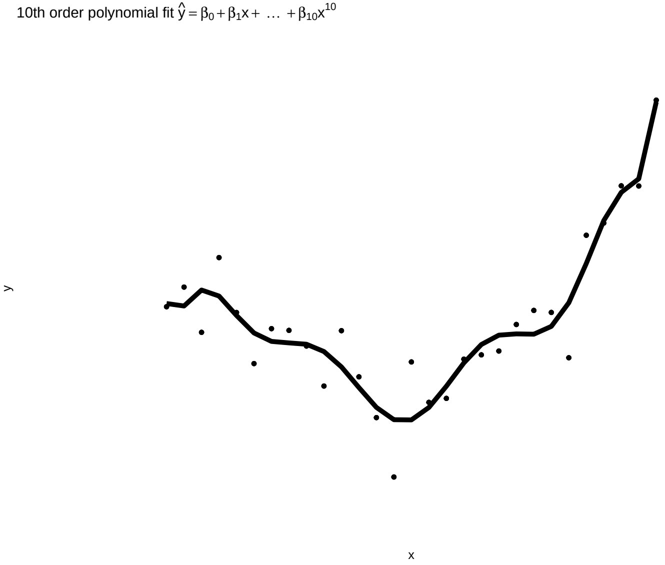
Example of dealing with overfitting



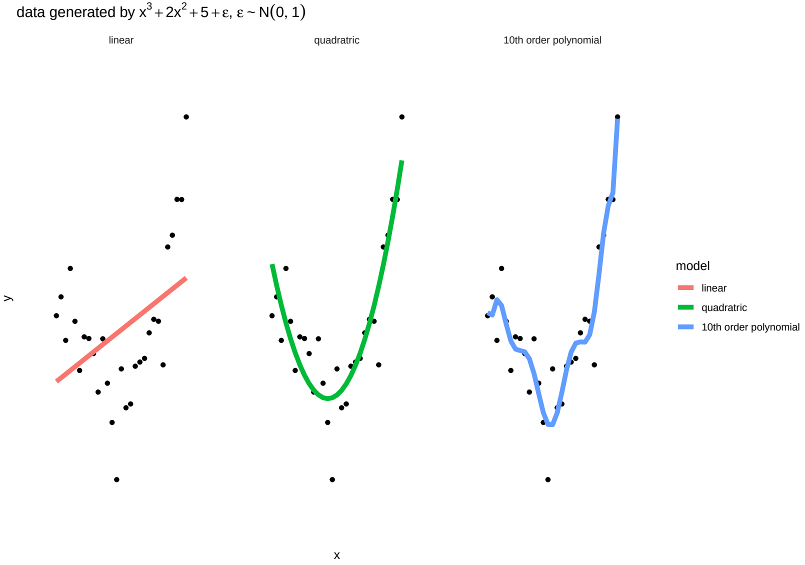
Example of dealing with overfitting



Example of dealing with overfitting



Example of dealing with overfitting



How does MDL work?

- Patterns or regularities in the data can be described with less information 'length' compared to the data alone.
- less information 'length' = compression
- Choose the model that gives the shortest description of the data.
- Note that MDL is an approach, not an algorithm.
    - The modeller has to make choices to implement the MDL principle.

Example of MDL

- Consider three data-generating processes (dgp's) that generate each a binary sequence of length 1000:
  1. 1000100010001000100010001000100010001000100 ... 1000
  2. 1001111110111100011001110001010100110001101 ... 1010
  3. 0100010000000101110010000000001101000110000 ... 0100

Example of MDL

- Consider three data-generating processes (dgp's) that generate each a binary sequence of length 1000:
  1. 1000100010001000100010001000100010001000100 ... 1000
  2. 100111110111100011001110001010100110001101 ... 1010
  3. 0100010000001011100100000000001101000110000 ... 0100
- All sequences written out take 1000 bits to be reproduced.
- However, because of regularities present in the dgp's, we require less bits to reproduce the same sequences.
- Question: how many bits of Matlab code does each sequence take to be *exactly* reproduced?

Example of MDL

- The sequences as before:
  1. 10001000100010001000100010001000100010001000100 . . . 1000:

Example of MDL

- The sequences as before:
  1. 10001000100010001000100010001000100010001000100 ... 1000: repetition of `[1 0 0 0]` 250 times.
     Matlab-code:
     ```
     textDgp_1 = 'repmat([1 0 0 0],1,250)'
     whos_textDgp_1 = whos('textDgp_1')
     whos_textDgp_1.bytes
     ```
     yields 23 bytes = 184 bits, a compression ratio of $1 - \frac{184}{1000} = 81.6\%$!

Example of MDL

- The sequences as before:
  1. 10001000100010001000100010001000100010001000100 ... 1000: repetition of `[1 0 0 0]` 250 times.
     Matlab-code:
     ```
     textDgp_1 = 'repmat([1 0 0 0],1,250)'
     whos_textDgp_1 = whos('textDgp_1')
     whos_textDgp_1.bytes
     ```
     yields 23 bytes = 184 bits, a compression ratio of $1 - \frac{184}{1000} = 81.6\%$!
  2. 10011111011110001100111000101010011000110 1 ... 1010:

Example of MDL

- The sequences as before:
  1. 1000100010001000100010001000100010001000100 . . . 1000: repetition of `[1 0 0 0]` 250 times.
     Matlab-code:
     ```
     textDgp_1 = 'repmat([1 0 0 0],1,250)'
     whos_textDgp_1 = whos('textDgp_1')
     whos_textDgp_1.bytes
     ```
     yields 23 bytes = 184 bits, a compression ratio of $1 - \frac{184}{1000} = 81.6\%!$
  2. 1001111110111100011001110001010100110001101 . . . 1010: a coin toss with heads (0) or tails (1); no compression possible because of the randomness involved.
  3. 0100010000001011100100000000001101000110000 . . . 0100:

Example of MDL

- The sequences as before:
  1. 1000100010001000100010001000100010001000100 ... 1000: repetition of `[1 0 0 0]` 250 times.
     Matlab-code:
     ```
     textDgp_1 = 'repmat([1 0 0 0],1,250)'
     whos_textDgp_1 = whos('textDgp_1')
     whos_textDgp_1.bytes
     ```
     yields 23 bytes = 184 bits, a compression ratio of $1 - \frac{184}{1000} = 81.6\%$!
  2. 1001111101111000110011100010101001100011 01 ... 1010: a coin toss with heads (0) or tails (1); no compression possible because of the randomness involved.
  3. 0100010000001011100100000000001101000110000 ... 0100: a roll with a four-sided die with outcomes 2, 3, 4 assigned to 0 and outcome 1 assigned to 1; no compression possible because of the randomness involved.

The Principle of MDL

- The more randomness involved, the less data compression is possible.
- When the restriction of exact reproduction is alleviated we may obtain some data compression.
- Dgp 2 (the coin toss) implies no data compression.
  - As the data sequence is completely random.
- Dgp 3 (the 4-sided die) implies some possible data compression.
  - Although exact reproduction is not possible the set of this type of data sequences requires around 821 bits.
- Most datasets are almost incompressible.
  - Only a small fraction can be significantly compressed.

Approach

From Data to Model Selection

- From data to code.
- From code to code length.
- From code length to probabilities.
- From probabilities to model selection.

From Data to Code

- Examples:
  - `Hello, world` could map to 0.
  - `aabbaccdaaadd` could map to 110100.
- In general, a description method maps a sequence of symbols to a binary sequence.
  - The coding alphabet $\mathbb{B}$ can be binary ($\mathbb{B} = \{0, 1\}$), the Western alphabet ($\mathbb{B} = \{\mathrm{a}, \mathrm{b}, \ldots, \mathrm{z}\}$), etc.
- Mathematically: a dataset $D = (x_1, \ldots, x_n)$ with $x_i \in \mathbb{B}$ from a sample space $\mathcal{X}^n$ is mapped to $\{0, 1\}^m$ by $C\colon \mathcal{X}^n \mapsto \{0, 1\}^m$.
  - In the first examples above, we could have $\mathcal{X}^n = \{x_1\} = \{'\mathrm{Hello, world}'\}$ and $C(x_1) = 0$.
- We demand the mapping C to be *uniquely* decodable.
  - No multiple interpretations allowed.

From Data to Code

- Suppose we would like to encode a binary data sequence of length 2.
- We are not sure what outcome we observe.
- Let $X_i \in \{0, 1\}$ be the random variable at position $i = 1, 2$.
  - The complete data sequence becomes $X_1X_2 \in \{00, 10, 01, 11\}$.
- Every sequence in $\{00, 10, 01, 11\}$ is assigned a code.
- In general, for the binary alphabet, for n positions there are 2^n possible data sequences.
  - For example, when $n = 3$ we have the data sequences $\{000, 100, 010, \ldots, 111\}$.
  - Without loss of generality, every non-binary alphabet can be mapped to the binary alphabet $\{0, 1\}$.

From Code to Code Length

- Given a data sequence x_i and its corresponding code $C(x_i)$, then we are interested in the code length $L(x_i)$ with $L\colon \mathcal{X}^n \mapsto \mathbb{R}_+$.
- For example, to map an integer from $\{1, \ldots, n\}$ in a uniform way, we need $\lceil \log_2(n) \rceil$ bits.
  - Note that n has to be known in advance.
  - For example, take $n = 64$. Then we have 64 binary data sequences of length $\lceil \log_2(64) \rceil = 6$.
  - Or take $n = 10$. Then we need $\lceil \log_2(10) \rceil = 4$ bits.
    - Note that $2^4 = 16$ data sequences are possible while we only use 10 of them.
- What coding scheme results in the smallest number of *expected* bits?

From Code to Code Length

- Recall the data sequences $\{00, 10, 01, 11\}$.
- Uniform method: the data sequence is the code.
  - $C(00) = 00$, $C(10) = 10$, $C(01) = 01$, and $C(11) = 11$.
  - Expected number of bits: 2.
- In general we can do better!
  - That is, $\mathbb{P}[X_i = 1] \neq \frac{1}{2}$ for at least one $i \in \{1, \ldots, n\}$.

From Code Length to Probabilities

- Suppose $\mathbb{P}[X_i = 1] = \frac{1}{4}$.
- We reserve code 0 for $X_1X_2 = 00$ so $C(00) = 0$, $C(10) = 100$, $C(01) = 110$, and $C(11) = 1110$.
- Then the expected number of bits is

$$1 \cdot \frac{3}{4}^2 + 3 \cdot \frac{3}{4} \cdot \frac{1}{4} + 3 \cdot \frac{1}{4} \cdot \frac{3}{4} + 4 \cdot \frac{1}{4}^2 = 1.9375 < 2$$

- This the Shannon-Fano coding scheme.
  - More general, reserve $\lceil -\log_2(p_i) \rceil$ bits for probability p_i corresponding to data sequence i.
  - The Huffman code is optimal and improves on the Shannon-Fano code.
- Main message:

$$\boxed{\text{higher probability} = \text{smaller code length} = \text{less bits required}}$$

From Probabilities to Model Selection

- To describe any dataset we need $L(D, H)$ bits.
  - Both the description method H and the data D require bits.
- The MDL principle employed for model selection is to minimise the sum of
  - the number of bits to encode the description mechanism $L(H)$ and
  - the number of bits to encode, with the description mechanism H, the data observed $L(D|H)$.
- Information 'length' is this sum $L(H) + L(D|H)$.
  - $L(H)$ is the *model complexity*, $L(D|H)$ is the *fit of the data*.
- The MDL principle is to choose the model as to minimise this sum.
- Main message:

$$\boxed{\text{smaller description length} = \text{better model selection}}$$

MDL and LASSO

- More mathematically, given a set of candidate models $\mathcal{H} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots\}$, select the optimal model $\mathcal{H}^\star \in \mathcal{H}$ as

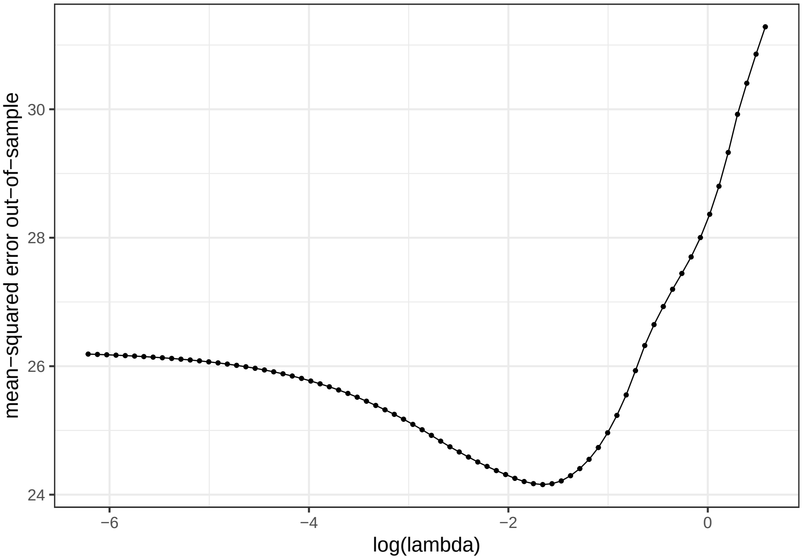
$$\mathcal{H}^\star := \arg\min_{H \in \mathcal{H}} L(D, H) = \arg\min_{H \in \mathcal{H}} \{L(H) + L(D|H)\}$$

- Note the resemblance with penalised model fitting, such as LASSO:
  - with LASSO we apply cross-validation to minimise

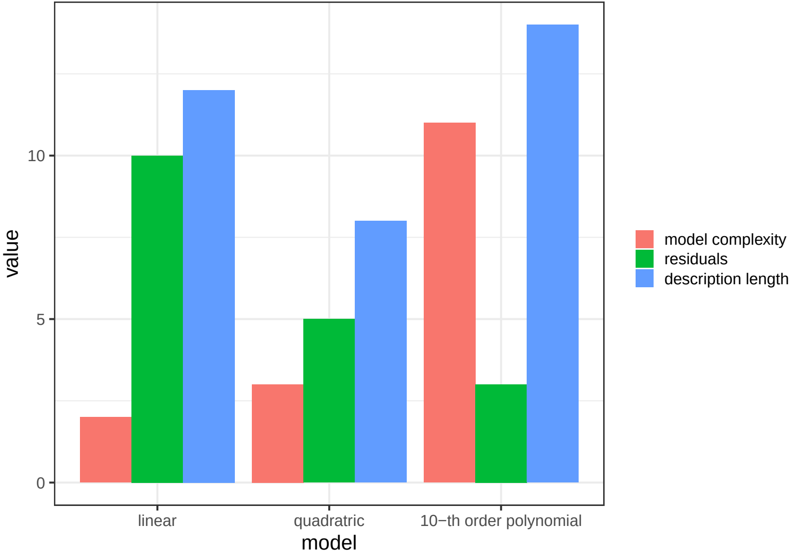
$$\arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{n} \|\boldsymbol{y} - \boldsymbol{X}\beta\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \right\}$$

with the data part $\frac{1}{n}\|\boldsymbol{y} - \boldsymbol{X}\beta\|_2^2$ ($L(D|H)$ in MDL) and $\lambda\|\beta\|_1$ the model complexity part ($L(H)$ in MDL).

Example of LASSO



Example of dealing with overfitting (continued)



Questions to be answered

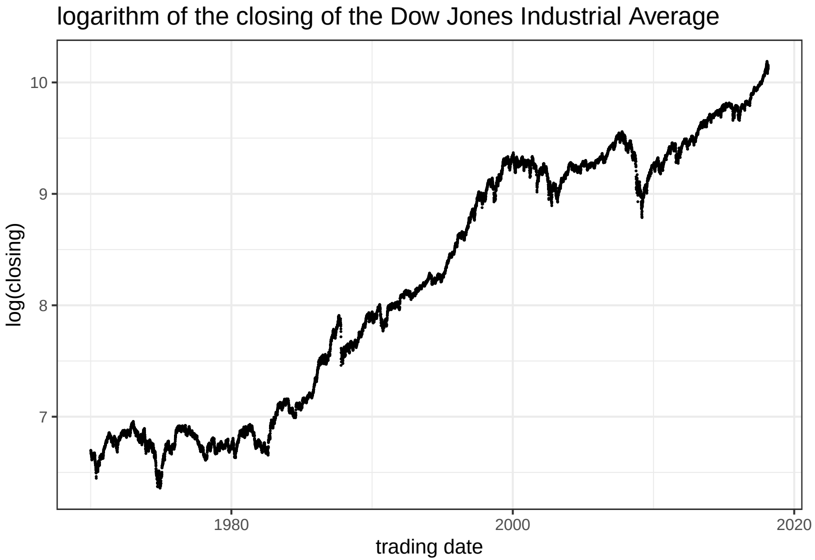
- How does MDL work in practice?
- How do the following approaches compare: frequentist statistics, Bayesian statistics, machine learning, and MDL?
- The coding system to put down the description mechanism should not matter. How can we choose a universal programming language?
  - For example, whether we use `Matlab`, `R` or whatever programming language should not matter for the number of bits at use.
- What are good encoding systems?
- What patterns are generalisable and what not?
- How to incorporate prior knowledge?

MDL in Practice

Dow Jones Industrial Average

- How can we employ MDL to compress a dataset?
  - Example taken from (Hansen and Yu 2001).
- Take the log daily return R_t and the volatility V_t of the Dow Jones Industrial Average (DJIA).
  - t runs from $t_0 =$ July 1962 until $T =$ June 1988, i.e. 6,430 trading days.
  - $R_t = P_t - P_{t-1}$ with P_t the logarithm of the DJIA at day t.
  - $V_t = 0.9V_{t-1} + 0.1R_t^2$ and V_0 the variance of the series P_t.
- R_t has a corresponding indicator: 1 (0) when $P_t > P_{t-1}$ ($P_t < P_{t-1}$).
  - Analogously for V_t.
  - Two binary strings of length 6,430 - 1 = 6,429.
  - R_t has 3,181 (49.49%) ups.
  - V_t has 2,023 (31.47%) ups.

Dow Jones Industrial Average



Dow Jones Industrial Average

- Without data compression we require 6,429 bits per series.
- Using MDL we save 10% on the volatility series V_t and 4% on R_t.
- n is known in advance so costs us $\lceil \log_2 n \rceil$ bits.
- Model the up or down indicator as a Bernoulli probability distribution with probability p on success.
  - Maximum likelihood yields $\hat{p} = k/n$ with k the number of successes.
- Ignoring rounding errors we have $L(H) = \log_2(n)$ and

$$L(D|H) = k\left[-\log_2\left(\frac{k}{n}\right)\right] + (n-k)\left[-\log_2\left(1-\frac{k}{n}\right)\right]$$

Dow Jones Industrial Average

- With $n = 6,429$ and
  - for R_t we have $k = 3181$ and $\hat{p} = \frac{k}{n} = \frac{3181}{6429} = 0.49$ we need

    $$\log_2(6429) + 3181[-\log_2 0.49] + 3248[-\log_2 0.51] = 6442 \text{ bits}$$

  - for V_t we have $k = 2023$ and $\hat{p} = \frac{k}{n} = \frac{2023}{6429} = 0.31$ we need

    $$\log_2(6429) + 2023[-\log_2 0.31] + 4406[-\log_2 0.69] = 5789 \text{ bits}$$

- In conclusion, for R_t we have gained nothing, even lost some bits, but the improvement for V_t is substantial.
  - The savings in storage increases as well as n increases or k/n approaches 0 or 1.
- Main message:

$$\boxed{\text{good data compression} = \text{good model performance}}$$

Comparing Models

- The model that requires the smallest number of bits, is the best.
  - Automatically prevents overfitting while aiming for the best model performance out-of-sample.
- For example, in the DJIA example (Hansen and Yu 2001) achieve a 4% improvement (a savings of 253 bits) on $R_t - R_{t-1}$ by applying a first-order Markov model.
  - This is because $R_t - R_{t-1}$ has a relatively high first-order autocorrelation of -0.42.
- The reduction in $L(D, H)$ justifies a more complex model: for $R_t - R_{t-1}$ a first order Markov-chain is preferred over a simple Bernoulli model.

Approaches compared

How do the approaches compare: Frequentist

- Also known as orthodox or non-Bayesian.
- Main goal: models should be consistent.
  - At least asymptotically: as $n \to \infty$ the estimated model coefficients should converge to the 'true' model parameters.
  - In the example above, one would expect $\hat{\beta} = (5\,0\,2\,1)$ for $n \to \infty$.
- When the model assumptions are fulfilled, this approach yields the best possible predictions.
- Pros: widely known and applicable, easy calculation, models are good with infinite amount of data.
- Cons: assumptions are not realistic (e.g. almost no residual follows a normal probability distribution).

How do the approaches compare: Bayesian

- Main goal: incorporate prior knowledge by assigning a prior probability distribution.
    - Once the prior probability distribution is assigned, the posterior probability distribution can be derived / calculated.
    - Frequentist models consider the data random and the parameters fixed.
    - Bayesian models consider the data and the parameters random.
    - In practice, asymptotically frequentist and Bayesian models converge.
- Pros: mathematically elegant and precise, encompasses the frequentist approach.
- Cons: hard to come up with a useful prior probability distribution.

How do the approaches compare: Machine Learning

- Main goal: find and fit a model that is able to generalise from train to test data.
  - No assumptions about the data generating process.
- Pros: no assumptions required, good model performance.
- Cons: hard to explain predictions, hard to study the data generating process.

How do the approaches compare: MDL

- Main goal: infer useful information from the data and use that to achieve good data compression.
  - Good data compression implies good learning.
  - No assumptions about the data generating process required but are allowed.
- Pros: no assumptions required, has best of all worlds (Bayesian and Machine Learning): employ prior knowledge when viable, good statistical properties, and decent performance.
- Cons: relatively new in literature; intersection of state-of-the-art knowledge of information theory, measure theory and statistics; not standard available in mainstream programming languages, can be computationally challenging.

Discussion

Application to Expected Credit Loss

- Suppose we apply the MDL principle on the calculation of the Expected Credit Loss.
  - No coding scheme discussed here, just the principle.
- Discussion:

  take a moving average of the realised credit losses to calculate the Expected Credit Loss

- Pros:
  - Occam's Razor: simpler is better.
  - Saves us a complete IFRS team to perform tedious calculations.
- Cons:
  - No new developments (e.g. Corona) taken into consideration. Can we with our current models?
  - No intermediate analyses available (e.g. use PD to rank customers in need).

Conclusion

Summary

- MDL is a guidance for model selection, an instrument against overfitting and exploratory data analysis.
- Sound statistics and decent model performance could make MDL a challenger for other statistical approaches.
- Much more to be explored.

What's next?

- A workshop to take a deepdive into the applications of MDL, backed up by statistical theory.
- Take a look at the open questions raised in this presentation.
  - The coding system to put down the description mechanism should not matter. How can we choose a universal programming language?
  - What are good encoding systems?
  - What patterns are generalisable and what not?
  - How to incorporate prior knowledge?
- Dedicated application to IFRS and Regulatory Capital calculations.
- Please let Eelko (eelko.ubels@devolksbank.nl) or me (paul.vanleeuwen@devolksbank.nl) know whether you would like to join!

Questions

?

Appendix

Kolmogorov complexity

- Any application of MDL should not be affected by the programming language of choice.
- Andrey Kolmogorov postulated the invariance theorem:

given any description of an object in a description language A, said description may be used in the optimal description language B with a constant overhead

- The description follows two steps:
  1. describe optimal language B in language A;
  2. describe object in optimal language B via language A.
- Comparable with a user-friendly programming language A (e.g. Matlab, R, Python) and whose code is compiled in an efficient programming language B (e.g. C++, C, FORTRAN).
- Part 1. is overhead independent of the object to be described, hence as the object size grows the overhead becomes small.

Kolmogorov complexity (example)

- Suppose we would like to describe the binary data sequence `1 0 0 0` in C via Matlab. How many bits does it take?
- First translate any code from C to Matlab needed to describe `1 0 0 0`.
  - E.g. C has a certain mapping procedure from the data type bit to its memory.
- The corresponding number of bits of the Matlab-code is $L_{\mathrm{C} \mapsto \mathrm{M}}$, e.g. 10k bits.
- To describe $D =$ `1 0 0 0` in C via Matlab takes $L_{\mathrm{C} \mapsto \mathrm{M}} + L_{\mathrm{M}}(D)$ bits.
- But $L_{\mathrm{C} \mapsto \mathrm{M}}$ is independent of the object to be described.
  - Whether we put in the works of Shakespeare or `Hello, World`, $L_{\mathrm{C} \mapsto \mathrm{M}}$ remains the same.

Bibliography

Grünwald, Peter D. 2007. *The Minimum Description Length Principle*. MIT press.

Hansen, Mark H, and Bin Yu. 2001. "Model Selection and the Principle of Minimum Description Length." *Journal of the American Statistical Association* 96 (454): 746–74.