Lossless compression and cumulative log loss

Yoav Freund

January 15, 2025

Outline

Lossless data compression

- The guessing game
- Arithmetic coding
- The performance of arithmetic coding

Outline

Lossless data compression

The guessing game

Arithmetic coding

The performance of arithmetic coding

log loss

Source entropy

Other properties of log loss

Unbiased prediction

Other examples for using log loss

Outline

Lossless data compression
- The guessing game
- Arithmetic coding
- The performance of arithmetic coding

log loss
- Source entropy
- Other properties of log loss
  - Unbiased prediction
  - Other examples for using log loss

universal coding
- Two part codes
- Combining expert advice for cumulative log loss

Outline

- Lossless data compression
  - The guessing game
  - Arithmetic coding
  - The performance of arithmetic coding
- log loss
  - Source entropy
  - Other properties of log loss
    - Unbiased prediction
    - Other examples for using log loss
- universal coding
  - Two part codes
  - Combining expert advice for cumulative log loss
- Combining experts in the log loss framework
  - The online Bayes Algorithm
  - The performance bound

The source compression problem

- **Example:** "There are no people like show people"

The source compression problem

- **Example:** “There are no people like show people”

The source compression problem

- **Example:** "There are no people like show people" $\stackrel{\text{encode}}{\rightarrow}$ $x \in \{0, 1\}^n$

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}} x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}} x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}} x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}}$ $x \in \{0, 1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?
- **Basic idea:** Use short codes for common messages.

The source compression problem

- **Example:** “There are no people like show people”
  $\stackrel{\text{encode}}{\rightarrow} x \in \{0,1\}^n$
  $\stackrel{\text{decode}}{\rightarrow}$ “there are no people like show people”
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?
- **Basic idea:** Use short codes for common messages.
- **Stream compression:**

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}} x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?
- **Basic idea:** Use short codes for common messages.
- **Stream compression:**
  - Message revealed one character at a time.

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}}$ $x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?
- **Basic idea:** Use short codes for common messages.
- **Stream compression:**
  - Message revealed one character at a time.
  - Code generated as message is revealed.

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}}$ $x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?
- **Basic idea:** Use short codes for common messages.
- **Stream compression:**
  - Message revealed one character at a time.
  - Code generated as message is revealed.
  - Decoded message is constructed gradually.

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}} x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?
- **Basic idea:** Use short codes for common messages.
- **Stream compression:**
  - Message revealed one character at a time.
  - Code generated as message is revealed.
  - Decoded message is constructed gradually.
- Easier than block codes when processing long messages.

The source compression problem

- **Example:** "There are no people like show people"
  $\xrightarrow{\text{encode}} x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "there are no people like show people"
- **Lossless:** Message reconstructed perfectly.
- **Goal:** minimize expected length $E(n)$ of coded message.
- Can we do better than $\lceil \log_2(26) \rceil = 5$ bits per character?
- **Basic idea:** Use short codes for common messages.
- **Stream compression:**
  - Message revealed one character at a time.
  - Code generated as message is revealed.
  - Decoded message is constructed gradually.
- Easier than block codes when processing long messages.
- A natural way for describing a distribution.

The Guessing game

- Message reveraled one character at a time

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t
6

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h
6	2

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e
6	2	1

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r
6	2	1	2

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e
6	2	1	2	1

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e	
6	2	1	2	1	1

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a
6	2	1	2	1	1	5

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r
6	2	1	2	1	1	5	2

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e
6	2	1	2	1	1	5	2	1

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e	
6	2	1	2	1	1	5	2	1	1

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e		n
6	2	1	2	1	1	5	2	1	1	4

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e		n	o
6	2	1	2	1	1	5	2	1	1	4	1

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e		n	o	
6	2	1	2	1	1	5	2	1	1	4	1	1

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e		n	o		p
6	2	1	2	1	1	5	2	1	1	4	1	1	5

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e		n	o		p	e
6	2	1	2	1	1	5	2	1	1	4	1	1	5	3

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e		n	o		p	e
6	2	1	2	1	1	5	2	1	1	4	1	1	5	3

- Code = sequence of number of mistakes.

The Guessing game

- Message reveraled one character at a time
- An algorithm predicts the next character from the revealed part of the message.
- If algorithm wrong - ask for next guess.
- **Example**

t	h	e	r	e		a	r	e		n	o		p	e
6	2	1	2	1	1	5	2	1	1	4	1	1	5	3

- Code = sequence of number of mistakes.
- To decode use the same prediction algorithm

Arithmetic Coding (background)

- Refines the guessing game:

Arithmetic Coding (background)

- Refines the guessing game:
  - In guessing game the predictor chooses order over alphabet.

Arithmetic Coding (background)

- Refines the guessing game:
  - In guessing game the predictor chooses order over alphabet.
  - In arithmetic coding the predictor chooses a Distribution over alphabet.

Arithmetic Coding (background)

- Refines the guessing game:
  - In guessing game the predictor chooses order over alphabet.
  - In arithmetic coding the predictor chooses a Distribution over alphabet.
- First discovered by Elias (MIT).

Arithmetic Coding (background)

- Refines the guessing game:
  - In guessing game the predictor chooses order over alphabet.
  - In arithmetic coding the predictor chooses a Distribution over alphabet.
- First discovered by Elias (MIT).
- Invented independently by Rissanen and Pasco in 1976.

Arithmetic Coding (background)

- Refines the guessing game:
  - In guessing game the predictor chooses order over alphabet.
  - In arithmetic coding the predictor chooses a Distribution over alphabet.
- First discovered by Elias (MIT).
- Invented independently by Rissanen and Pasco in 1976.
- Widely used in practice.

Arithmetic Coding (basic idea)

- Easier notation: represent characters by numbers $1 \leq c_t \leq |\Sigma|$. (English: $N \doteq |\Sigma| = 26$)

Arithmetic Coding (basic idea)

- Easier notation: represent characters by numbers $1 \leq c_t \leq |\Sigma|$. (English: $N \doteq |\Sigma| = 26$)
- message-prefix $c_1, c_2, \ldots, c_{t-1}$ represented by line segment $[l_{t-1}, u_{t-1})$

Arithmetic Coding (basic idea)

- Easier notation: represent characters by numbers $1 \leq c_t \leq |\Sigma|$. (English: $N \doteq |\Sigma| = 26$)
- message-prefix $c_1, c_2, \ldots, c_{t-1}$ represented by line segment $[l_{t-1}, u_{t-1})$
- Initial segment $[l_0, u_0) = [0, 1)$

Arithmetic Coding (basic idea)

- Easier notation: represent characters by numbers $1 \leq c_t \leq |\Sigma|$. (English: $N \doteq |\Sigma| = 26$)
- message-prefix $c_1, c_2, \ldots, c_{t-1}$ represented by line segment $[l_{t-1}, u_{t-1})$
- Initial segment $[l_0, u_0) = [0, 1)$
- After observing $c_1, c_2, \ldots, c_{t-1}$, predictor outputs $p(c_t = 1 \,|c_1, c_2, \ldots, c_{t-1}), \ldots, p(c_t = |\Sigma| \; |c_1, c_2, \ldots, c_{t-1})$,

Arithmetic Coding (basic idea)

- Easier notation: represent characters by numbers $1 \leq c_t \leq |\Sigma|$. (English: $N \doteq |\Sigma| = 26$)
- message-prefix $c_1, c_2, \ldots, c_{t-1}$ represented by line segment $[l_{t-1}, u_{t-1})$
- Initial segment $[l_0, u_0) = [0, 1)$
- After observing $c_1, c_2, \ldots, c_{t-1}$, predictor outputs $p(c_t = 1 \,|c_1, c_2, \ldots, c_{t-1}), \ldots, p(c_t = |\Sigma| \; |c_1, c_2, \ldots, c_{t-1})$,
- Distribution is used to partition $[l_{t-1}, u_{t-1})$ into $|\Sigma|$ sub-segments.

Arithmetic Coding (basic idea)

- Easier notation: represent characters by numbers $1 \leq c_t \leq |\Sigma|$. (English: $N \doteq |\Sigma| = 26$)
- message-prefix $c_1, c_2, \ldots, c_{t-1}$ represented by line segment $[l_{t-1}, u_{t-1})$
- Initial segment $[l_0, u_0) = [0, 1)$
- After observing $c_1, c_2, \ldots, c_{t-1}$, predictor outputs $p(c_t = 1 \,|c_1, c_2, \ldots, c_{t-1}), \ldots, p(c_t = |\Sigma| \ |c_1, c_2, \ldots, c_{t-1})$,
- Distribution is used to partition $[l_{t-1}, u_{t-1})$ into $|\Sigma|$ sub-segments.
- next character c_t determines $[l_t, u_t)$

Arithmetic Coding (basic idea)

- Easier notation: represent characters by numbers $1 \leq c_t \leq |\Sigma|$. (English: $N \doteq |\Sigma| = 26$)
- message-prefix $c_1, c_2, \ldots, c_{t-1}$ represented by line segment $[l_{t-1}, u_{t-1})$
- Initial segment $[l_0, u_0) = [0, 1)$
- After observing $c_1, c_2, \ldots, c_{t-1}$, predictor outputs $p(c_t = 1 \,|c_1, c_2, \ldots, c_{t-1}), \ldots, p(c_t = |\Sigma| \,\,|c_1, c_2, \ldots, c_{t-1})$,
- Distribution is used to partition $[l_{t-1}, u_{t-1})$ into $|\Sigma|$ sub-segments.
- next character c_t determines $[l_t, u_t)$
- Code = discriminating binary expansion of a point in $[l_t, u_t)$.

Arithmetic Coding (sequence example)

Arithmetic Coding (sequence example)

Arithmetic Coding (sequence example)

- Simplest case.

Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$

Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$

Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t$,
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111

Arithmetic Coding (sequence example)

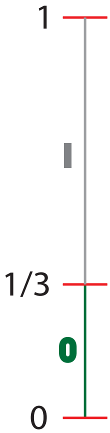
- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111

Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:**
  Assume decoder knows that length of message is 4.

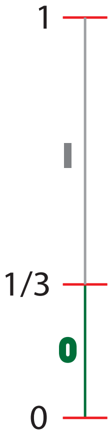
Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:**
  Assume decoder knows that length of message is 4.



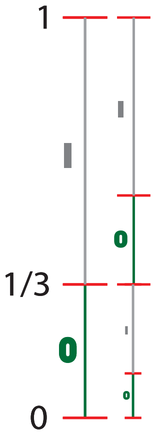
Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:**
  Assume decoder knows that length of message is 4.



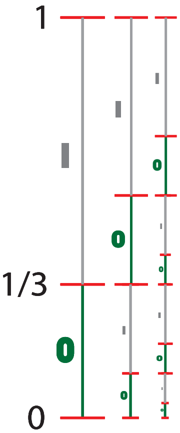
Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t$,
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:** Assume decoder knows that length of message is 4.



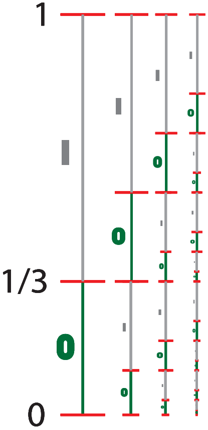
Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:** Assume decoder knows that length of message is 4.



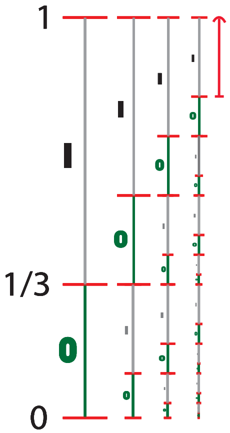
Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t$,
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:** Assume decoder knows that length of message is 4.



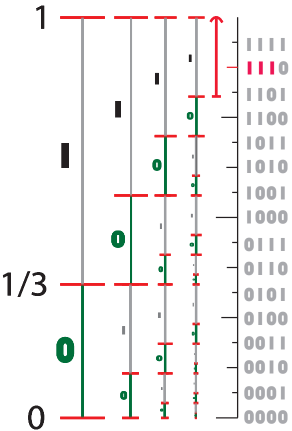
Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:** Assume decoder knows that length of message is 4.

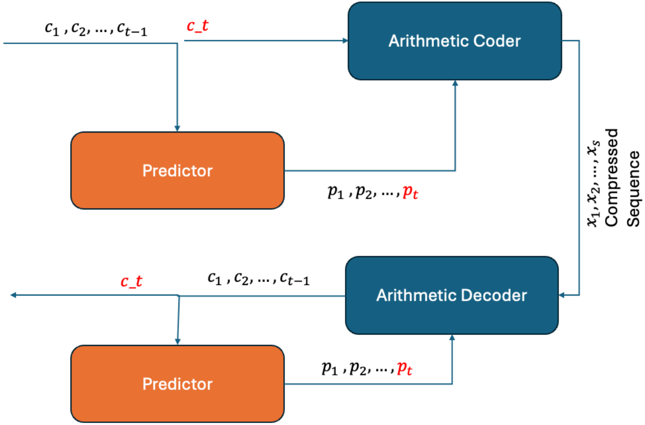


Arithmetic Coding (sequence example)

- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t$,
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111
- **Technical:**
  Assume decoder knows that length of message is 4.



Arithmetic coding (coding/decoding)



The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.

The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.
- Distance between two m bit expansions is 2^{-m}

The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.
- Distance between two m bit expansions is 2^{-m}
- If $l_T - u_T \geq 2^{-m}$ then there must be a point x described by m expansion bits such that $l_T \leq x < u_T$

The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.
- Distance between two m bit expansions is 2^{-m}
- If $l_T - u_T \geq 2^{-m}$ then there must be a point x described by m expansion bits such that $l_T \leq x < u_T$
- Required number of bits is $\lceil -\log_2(u_T - l_T) \rceil$.

The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.
- Distance between two m bit expansions is 2^{-m}
- If $l_T - u_T \geq 2^{-m}$ then there must be a point x described by m expansion bits such that $l_T \leq x < u_T$
- Required number of bits is $\lceil -\log_2(u_T - l_T) \rceil$.
- $u_T - l_T = \prod_{t=1}^{T} p\,(c_t \,|c_1, c_2, \ldots, c_{t-1}\,) \doteq p(c_1, \ldots c_T)$

The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.
- Distance between two m bit expansions is 2^{-m}
- If $l_T - u_T \geq 2^{-m}$ then there must be a point x described by m expansion bits such that $l_T \leq x < u_T$
- Required number of bits is $\lceil -\log_2(u_T - l_T) \rceil$.
- $u_T - l_T = \prod_{t=1}^{T} p(c_t \,|\, c_1, c_2, \ldots, c_{t-1}) \doteq p(c_1, \ldots c_T)$
- Number of bits required to code $c_1, c_2, \ldots, c_T$ is $\lceil -\sum_{t=1}^{T} \log_2 p_t(c_t) \rceil$.

The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.
- Distance between two m bit expansions is 2^{-m}
- If $l_T - u_T \geq 2^{-m}$ then there must be a point x described by m expansion bits such that $l_T \leq x < u_T$
- Required number of bits is $\lceil -\log_2(u_T - l_T) \rceil$.
- $u_T - l_T = \prod_{t=1}^{T} p\left(c_t \,|c_1, c_2, \ldots, c_{t-1}\right) \doteq p(c_1, \ldots c_T)$
- Number of bits required to code $c_1, c_2, \ldots, c_T$ is $\lceil -\sum_{t=1}^{T} \log_2 p_t(c_t) \rceil$.
- We call $-\sum_{t=1}^{T} \log_2 p_t(c_t) = -log_2 p(c_1, \ldots c_T)$ the Cumulative log loss

The code length for arithmetic coding

- Given m bits of binary expansion we assume the rest are all zero.
- Distance between two m bit expansions is 2^{-m}
- If $l_T - u_T \geq 2^{-m}$ then there must be a point x described by m expansion bits such that $l_T \leq x < u_T$
- Required number of bits is $\lceil -\log_2(u_T - l_T) \rceil$.
- $u_T - l_T = \prod_{t=1}^{T} p\left(c_t \,|c_1, c_2, \ldots, c_{t-1}\right) \doteq p(c_1, \ldots c_T)$
- Number of bits required to code $c_1, c_2, \ldots, c_T$ is $\lceil -\sum_{t=1}^{T} \log_2 p_t(c_t) \rceil$.
- We call $-\sum_{t=1}^{T} \log_2 p_t(c_t) = -log_2 p(c_1, \ldots c_T)$ the Cumulative log loss
- Holds for all sequences.

Expected code length

- Fix the messsage length T

Expected code length

- Fix the messsage length T
- Suppose the message is generated at random according to the distribution $p(c_1, \ldots c_T)$

Expected code length

- Fix the messsage length T
- Suppose the message is generated at random according to the distribution $p(c_1, \ldots c_T)$
- Then the expected code length is

$$\sum_{c_1, \ldots c_T} p(c_1, \ldots c_T) \lceil -\log_2 p(c_1, \ldots, c_T) \rceil$$

Expected code length

- Fix the messsage length T
- Suppose the message is generated at random according to the distribution $p(c_1, \ldots c_T)$
- Then the expected code length is

$$\sum_{c_1, \ldots c_T} p(c_1, \ldots c_T)\lceil -\log_2 p(c_1, \ldots, c_T)\rceil$$

Expected code length

- Fix the messsage length T
- Suppose the message is generated at random according to the distribution $p(c_1, \ldots c_T)$
- Then the expected code length is

$$\sum_{c_1,\ldots c_T} p(c_1, \ldots c_T)\lceil -\log_2 p(c_1, \ldots, c_T)\rceil$$

$$\leq \quad 1 - \sum_{c_1,\ldots c_T} p(c_1, \ldots c_T) \log_2 p(c_1, \ldots, c_T)$$

Expected code length

- Fix the messsage length T
- Suppose the message is generated at random according to the distribution $p(c_1, \ldots c_T)$
- Then the expected code length is

$$\sum_{c_1,\ldots c_T} p(c_1, \ldots c_T) \lceil -\log_2 p(c_1, \ldots, c_T) \rceil$$

$$\leq \quad 1 - \sum_{c_1,\ldots c_T} p(c_1, \ldots c_T) \log_2 p(c_1, \ldots, c_T) \doteq 1 + H(p_T)$$

- $H(p_T)$ is the **entropy** of the distribution over sequences of length T:

$$H(p_T) \doteq \sum_{(c_1,\ldots,c_T)} p(c_1, \ldots, c_T) \log \frac{1}{p(c_1, dots, c_T)}$$

Expected code length

- Fix the messsage length T
- Suppose the message is generated at random according to the distribution $p(c_1, \ldots c_T)$
- Then the expected code length is

$$\sum_{c_1,\ldots c_T} p(c_1, \ldots c_T)\lceil -\log_2 p(c_1, \ldots, c_T)\rceil$$

$$\leq \quad 1 - \sum_{c_1,\ldots c_T} p(c_1, \ldots c_T) \log_2 p(c_1, \ldots, c_T) \doteq 1 + H(p_T)$$

- $H(p_T)$ is the **entropy** of the distribution over sequences of length T:

$$H(p_T) \doteq \sum_{(c_1,\ldots,c_T)} p(c_1, \ldots, c_T) \log \frac{1}{p(c_1, dots, c_T)}$$

- Entropy is the expected value of the cumulative log loss

Shannon's lower bound

- Assume p_T is "well behaved". For example, IID.

Shannon's lower bound

- Assume p_T is "well behaved". For example, IID.
- Let $T \to \infty$

Shannon's lower bound

- Assume p_T is "well behaved". For example, IID.
- Let $T \to \infty$
- $H(p) \doteq \lim_{T\to\infty} \frac{H(p_T)}{T}$ exists and is called the per character entropy of the source p

Shannon's lower bound

- Assume p_T is "well behaved". For example, IID.
- Let $T \to \infty$
- $H(p) \doteq \lim_{T\to\infty} \frac{H(p_T)}{T}$ exists and is called the per character entropy of the source p
- The expected code length for any coding scheme is at least
$$(1 - o(1))H(p_T) = (1 - o(1))\ T\ H(p)$$

Shannon's lower bound

- Assume p_T is "well behaved". For example, IID.
- Let $T \to \infty$
- $H(p) \doteq \lim_{T\to\infty} \frac{H(p_T)}{T}$ exists and is called the per character entropy of the source p
- The expected code length for any coding scheme is at least
$$(1 - o(1))H(p_T) = (1 - o(1))\; T\; H(p)$$
- The proof of Shannon's lower bound is not trivial (Can be a student lecture).

log loss encourages unbiased prediction

- Suppose the source is random and the probability of the next outcome is $p\left(c_t \,\middle|\, c_1, c_2, \ldots, c_{t-1}\right)$

log loss encourages unbiased prediction

- Suppose the source is random and the probability of the next outcome is $p(c_t \,|\, c_1, c_2, \ldots, c_{t-1})$
- Then the prediction that minimizes the log loss is $p(c_t \,|\, c_1, c_2, \ldots, c_{t-1})$.

log loss encourages unbiased prediction

- Suppose the source is random and the probability of the next outcome is $p(c_t \mid c_1, c_2, \ldots, c_{t-1})$
- Then the prediction that minimizes the log loss is $p(c_t \mid c_1, c_2, \ldots, c_{t-1})$.
- Note that when minimizing expected number of mistakes, the best prediction in this situation is to put all of the probability on the most likely outcome.

log loss encourages unbiased prediction

- Suppose the source is random and the probability of the next outcome is $p(c_t | c_1, c_2, \ldots, c_{t-1})$
- Then the prediction that minimizes the log loss is $p(c_t | c_1, c_2, \ldots, c_{t-1})$.
- Note that when minimizing expected number of mistakes, the best prediction in this situation is to put all of the probability on the most likely outcome.
- There are other losses with this property, for example, square loss.

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$
- Forecaster assigns probability p_t to rain on day t.

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$
- Forecaster assigns probability p_t to rain on day t.
- If it rains on day t then $b_t = 2b_{t-1}p_t$

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$
- Forecaster assigns probability p_t to rain on day t.
- If it rains on day t then $b_t = 2b_{t-1}p_t$
- If it does not rain on day t then $b_t = 2b_{t-1}(1 - p_t)$

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$
- Forecaster assigns probability p_t to rain on day t.
- If it rains on day t then $b_t = 2b_{t-1}p_t$
- If it does not rain on day t then $b_t = 2b_{t-1}(1 - p_t)$
- At the end of the month, give forecaster b_T

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$
- Forecaster assigns probability p_t to rain on day t.
- If it rains on day t then $b_t = 2b_{t-1}p_t$
- If it does not rain on day t then $b_t = 2b_{t-1}(1 - p_t)$
- At the end of the month, give forecaster b_T
- Risk averse strategy: Setting $p_t = 1/2$ for all days, guarantees $b_T = 1$

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$
- Forecaster assigns probability p_t to rain on day t.
- If it rains on day t then $b_t = 2b_{t-1}p_t$
- If it does not rain on day t then $b_t = 2b_{t-1}(1 - p_t)$
- At the end of the month, give forecaster b_T
- Risk averse strategy: Setting $p_t = 1/2$ for all days, guarantees $b_T = 1$
- High risk prediction: Setting $p_t \in \{0, 1\}$ results in Bonus $b_T = 2^T$ if always correct, zero otherwise.

Monthly bonuses for a weather forecaster

- Before the first of the month assign one dollar to the forecaster's bonus. $b_0 = 1$
- Forecaster assigns probability p_t to rain on day t.
- If it rains on day t then $b_t = 2b_{t-1}p_t$
- If it does not rain on day t then $b_t = 2b_{t-1}(1 - p_t)$
- At the end of the month, give forecaster b_T
- Risk averse strategy: Setting $p_t = 1/2$ for all days, guarantees $b_T = 1$
- High risk prediction: Setting $p_t \in \{0, 1\}$ results in Bonus $b_T = 2^T$ if always correct, zero otherwise.
- If forecaster predicts with the true probabilities then

$$E(\log b_T) = T - H(p_T)$$

and that is the maximal expected value for $E(\log b_T)$

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$
- m horses compete in each race.

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$
- m horses compete in each race.
- Before each race, the odds for each horse are announced: $o_t(1), \ldots o_t(m)$ (arbitrary positive numbers)

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$
- m horses compete in each race.
- Before each race, the odds for each horse are announced: $o_t(1), \ldots o_t(m)$ (arbitrary positive numbers)
- You have to divide *all* your money among the different horses. $\sum_{j=1}^{t} \hat{p}_t(j) = 1$

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$
- m horses compete in each race.
- Before each race, the odds for each horse are announced: $o_t(1), \ldots o_t(m)$ (arbitrary positive numbers)
- You have to divide *all* your money among the different horses. $\sum_{j=1}^{t} \hat{p}_t(j) = 1$
- The horse $1 \leq y_t \leq m$ is winner of the tth race.

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$
- m horses compete in each race.
- Before each race, the odds for each horse are announced: $o_t(1), \ldots o_t(m)$ (arbitrary positive numbers)
- You have to divide *all* your money among the different horses. $\sum_{j=1}^{t} \hat{p}_t(j) = 1$
- The horse $1 \leq y_t \leq m$ is winner of the tth race.
- After iteration t, you have $b_t = b_{t-1}\hat{p}_t(y_t)o_t(y_t)$ dollars

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$
- m horses compete in each race.
- Before each race, the odds for each horse are announced: $o_t(1), \ldots o_t(m)$ (arbitrary positive numbers)
- You have to divide *all* your money among the different horses. $\sum_{j=1}^{t} \hat{p}_t(j) = 1$
- The horse $1 \leq y_t \leq m$ is winner of the tth race.
- After iteration t, you have $b_t = b_{t-1}\hat{p}_t(y_t)o_t(y_t)$ dollars
- After n races, you have $b_n = \prod_{t=1}^{n} \hat{p}_t(y_t)o_t(y_t)$ dollars.

Horse-race betting

- You go to the horse races with one dollar $b_0 = 1$
- m horses compete in each race.
- Before each race, the odds for each horse are announced: $o_t(1), \ldots o_t(m)$ (arbitrary positive numbers)
- You have to divide *all* your money among the different horses. $\sum_{j=1}^{t} \hat{p}_t(j) = 1$
- The horse $1 \leq y_t \leq m$ is winner of the tth race.
- After iteration t, you have $b_t = b_{t-1}\hat{p}_t(y_t)o_t(y_t)$ dollars
- After n races, you have $b_n = \prod_{t=1}^{n} \hat{p}_t(y_t)o_t(y_t)$ dollars.
- Taking logs, we get cumulative log loss.

‘Universal coding

- Suppose there are N alternative predictors / experts.

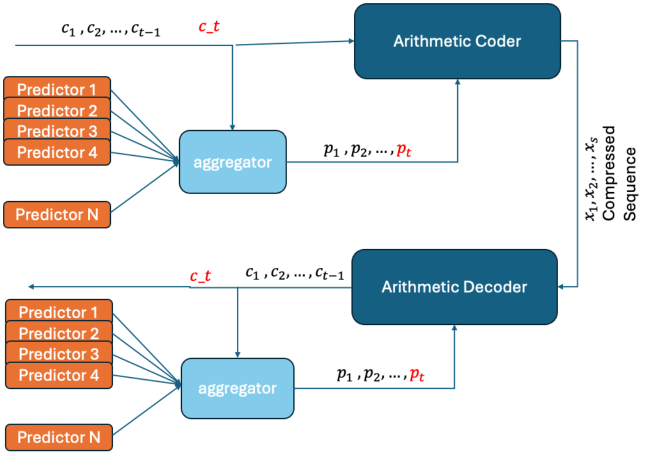
‘Universal coding

- Suppose there are N alternative predictors / experts.
- We would like to code almost as well as the best predictor in hindsight.

‘Universal coding

- Suppose there are N alternative predictors / experts.
- We would like to code almost as well as the best predictor in hindsight.
- In horse race: We would like to make almost as much money as the best expert in hind-site.

Universal arithmetic coding



Two part codes

- Send the index of the coding algorithm before the message.

Two part codes

- Send the index of the coding algorithm before the message.
- Requires $\log_2 N$ additional bits.

Two part codes

- Send the index of the coding algorithm before the message.
- Requires $\log_2 N$ additional bits.
- Requires the encoder to make **two** passes over the data.

Two part codes

- Send the index of the coding algorithm before the message.
- Requires $\log_2 N$ additional bits.
- Requires the encoder to make **two** passes over the data.
- Is the key idea of MDL (Minimal Description Length) modeling.

Two part codes

- Send the index of the coding algorithm before the message.
- Requires $\log_2 N$ additional bits.
- Requires the encoder to make **two** passes over the data.
- Is the key idea of MDL (Minimal Description Length) modeling.
  - Good prediction model = model that minimizes the total code length

Two part codes

- Send the index of the coding algorithm before the message.
- Requires $\log_2 N$ additional bits.
- Requires the encoder to make **two** passes over the data.
- Is the key idea of MDL (Minimal Description Length) modeling.
  - Good prediction model = model that minimizes the total code length
- Often inappropriate because based on lossless coding. **Lossy** coding often more appropriate.

Combining predictors adaptively

- Treat each of the predictors as an “expert”.

Combining predictors adaptively

- Treat each of the predictors as an “expert”.
- Assign a weight to each expert and reduce it if expert performs poorly.

Combining predictors adaptively

- Treat each of the predictors as an “expert”.
- Assign a weight to each expert and reduce it if expert performs poorly.
- Combine expert predictions according to their weights.

Combining predictors adaptively

- Treat each of the predictors as an “expert”.
- Assign a weight to each expert and reduce it if expert performs poorly.
- Combine expert predictions according to their weights.
- Would require only a single pass. Truly online.

Combining predictors adaptively

- Treat each of the predictors as an “expert”.
- Assign a weight to each expert and reduce it if expert performs poorly.
- Combine expert predictions according to their weights.
- Would require only a single pass. Truly online.
- **Goal:** Total loss of algorithm minus loss of best predictor should be at most $\log_2 N$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k) \rangle$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k) \rangle$
- When c^t is revealed, the loss we suffer is $-\log p_A^t(c^t)$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k) \rangle$
- When c^t is revealed, the loss we suffer is $-\log p_A^t(c^t)$
- The cumulative log loss, which we wish to minimize, is $L_A{}^T = -\sum_{t=1}^{T} \log p_A^t(c^t)$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k) \rangle$
- When c^t is revealed, the loss we suffer is $-\log p_A^t(c^t)$
- The cumulative log loss, which we wish to minimize, is $L_A{}^T = -\sum_{t=1}^T \log p_A^t(c^t)$
- $\lceil L_A{}^T \rceil$ is the code length if A is combined with arithmetic coding.

The game

- Prediction algorithm A has access to N experts.

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$
  - Algorithm generates its own prediction $\mathbf{p}_A^t$

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$
  - Algorithm generates its own prediction $\mathbf{p}_A^t$
  - c^t is revealed.

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$
  - Algorithm generates its own prediction $\mathbf{p}_A^t$
  - c^t is revealed.
- **Goal:** minimize regret:

$$-\sum_{t=1}^{T} \log p_A^t(c^t) + \min_{i=1,\ldots,N} \left(-\sum_{t=1}^{T} \log p_i^t(c^t) \right)$$

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

- Weight of expert i

$$w_i^t = w_i^1 e^{-L_i^{t-1}} = w_i^1 \prod_{s=1}^{t-1} p_i^s(c^s)$$

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

- Weight of expert i

$$w_i^t = w_i^1 e^{-L_i^{t-1}} = w_i^1 \prod_{s=1}^{t-1} p_i^s(c^s)$$

- Freedom to choose initial weights.
  $w_t^1 \geq 0, \sum_{i=1}^n w_i^1 = 1$

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

- Weight of expert i

$$w_i^t = w_i^1 e^{-L_i^{t-1}} = w_i^1 \prod_{s=1}^{t-1} p_i^s(c^s)$$

- Freedom to choose initial weights.
  $w_t^1 \geq 0, \sum_{i=1}^n w_i^1 = 1$
- Prediction of algorithm A

$$\mathbf{p}_A^t = \frac{\sum_{i=1}^N w_i^t \mathbf{p}_i^t}{\sum_{i=1}^N w_i^t}$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^{N} w_i^t$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t}$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t}$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^T \log p_A^t(c^t)$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^T \log p_A^t(c^t) = L_A^T$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log W^{T+1} = -\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^{T} \log p_A^t(c^t) = L_A^T$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log W^{T+1} = -\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^{T} \log p_A^t(c^t) = L_A^T$$

EQUALITY not bound!

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$
- Total Weight is at least the weight of the best expert.

$$L_A^T = -\log W^{T+1}$$

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$
- Total Weight is at least the weight of the best expert.

$$L_A^T = -\log W^{T+1}$$

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$
- Total Weight is at least the weight of the best expert.

$$L_A^T = -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1}$$

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$
- Total Weight is at least the weight of the best expert.

$$\begin{aligned} L_A^T &= -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1} \\ &= -\log \sum_{i=1}^{N} \frac{1}{N} e^{-L_i^T} \end{aligned}$$

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$
- Total Weight is at least the weight of the best expert.

$$
\begin{aligned}
L_A^T &= -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1} \\
&= -\log \sum_{i=1}^{N} \frac{1}{N} e^{-L_i^T} = \log N - \log \sum_{i=1}^{N} e^{-L_i^T}
\end{aligned}
$$

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$
- Total Weight is at least the weight of the best expert.

$$
\begin{aligned}
L_A^T &= -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1} \\
&= -\log \sum_{i=1}^{N} \frac{1}{N} e^{-L_i^T} = \log N - \log \sum_{i=1}^{N} e^{-L_i^T} \\
&\leq \log N - \log \max_i e^{-L_i^T}
\end{aligned}
$$

Simple Bound

- Use uniform initial weights $w_i^1 = 1/N$
- Total Weight is at least the weight of the best expert.

$$
\begin{aligned}
L_A^T &= -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1} \\
&= -\log \sum_{i=1}^{N} \frac{1}{N} e^{-L_i^T} = \log N - \log \sum_{i=1}^{N} e^{-L_i^T} \\
&\leq \log N - \log \max_i e^{-L_i^T} = \log N + \min_i L_i^T
\end{aligned}
$$

- Dividing by T we get $\frac{L_A^T}{T} = \min_i \frac{L_i^T}{T} + \frac{\log N}{T}$

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

- We don't pay a penalty for copies.

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

- We don't pay a penalty for copies.
- More generally, the regret is smaller if many of the experts perform well.

How to choose the initial weights?

- When experts are similar - you want to assign each of them less weight.

How to choose the initial weights?

- When experts are similar - you want to assign each of them less weight.
- The min-max prior.

How to choose the initial weights?

- When experts are similar - you want to assign each of them less weight.
- The min-max prior.
- Priors that allow efficient computation.

How to choose the initial weights?

- When experts are similar - you want to assign each of them less weight.
- The min-max prior.
- Priors that allow efficient computation.
- Conjugate priors.