Lossless compression and cumulative log loss

Yoav Freund

January 17, 2006

Outline

Outline

Outline

Outline

Outline

Outline

Outline

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" $\xrightarrow{\text{encode}} x \in \{0, 1\}^n$

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"
  $\xrightarrow{\text{encode}} x \in \{0,1\}^n$
  $\xrightarrow{\text{decode}}$ "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:**

The source compression problem

- **Example:** "There are no people like show people"
  $\stackrel{\text{encode}}{\longrightarrow} x \in \{0,1\}^n$
  $\stackrel{\text{decode}}{\longrightarrow}$ "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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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 - as 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: $|\Sigma| = 26$)

Arithmetic Coding (basic idea)

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

Arithmetic Coding (Example)

Arithmetic Coding (Example)

- Simplest case.

Arithmetic Coding (Example)

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

Arithmetic Coding (Example)

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

Arithmetic Coding (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 (Example)

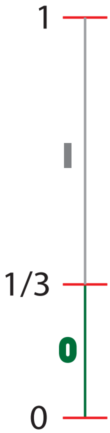
- Simplest case.
- $\Sigma = \{0, 1\}$
- $\forall t,$
  $p(c_t = 0) = 1/3$
  $p_t(c_t = 1) = 2/3$
- Message = 1111
- Code = 111

Arithmetic Coding (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 message
  length)

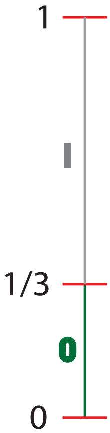
Arithmetic Coding (Example)

- 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 message length)



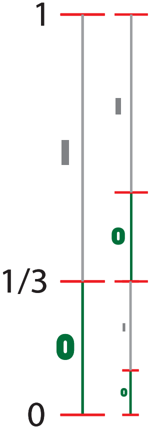
Arithmetic Coding (Example)

- 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 message length)



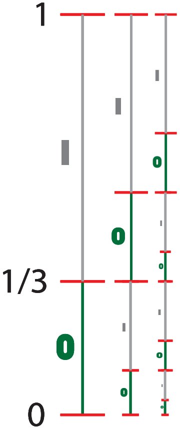
Arithmetic Coding (Example)

- 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 message length)



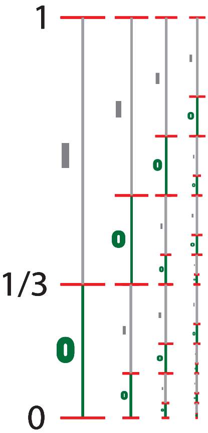
Arithmetic Coding (Example)

- 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 message length)



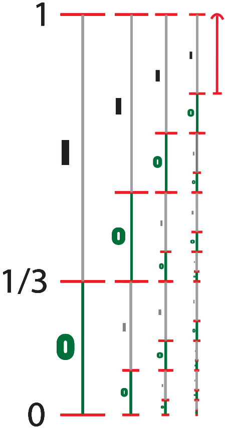
Arithmetic Coding (Example)

- 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 message length)



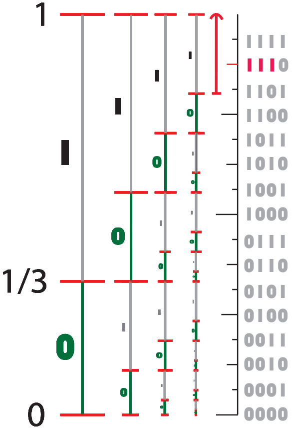
Arithmetic Coding (Example)

- 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 message length)



Arithmetic Coding (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 message length)



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 $\lceil -\log_2(u_T - l_T) \rceil$.
- $u_T - l_T = \prod_{t=1}^{T} p\left(c_t \,|\, c_1, c_2, \ldots, c_{t-1}\right) \doteq p(c_1, \ldots c_T)$

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 $\lceil -\log_2(u_T - l_T) \rceil$.
- $u_T - l_T = \prod_{t=1}^{T} p\left(c_t \,|\, c_1, c_2, \ldots, c_{t-1}\right) \doteq p(c_1, \ldots c_T)$
- Number of bits required to code $c_1, c_2, \ldots, c_T$ is $\lceil -\sum_{t=1}^{T} \log_2 p_t(c_t) \rceil$.

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 $\lceil -\log_2(u_T - l_T) \rceil$.
- $u_T - l_T = \prod_{t=1}^{T} p\left(c_t \middle| c_1, c_2, \ldots, c_{t-1}\right) \doteq p(c_1, \ldots c_T)$
- 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.

Expectation of code length

- Fix the messsage length T

Expectation of code length

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

Expectation of code length

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

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

Expectation of 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

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

Expectation of 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

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

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

Expectation of 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

$$
\begin{aligned}
& \sum_{c_1, \ldots c_T} p(c_1, \ldots c_T) \lceil - \log_2 p(c_1, \ldots c_T) \rceil \\
\leq \quad & 1 + \sum_{c_1, \ldots c_T} p(c_1, \ldots c_T) - \log_2 p(c_1, \ldots c_T) \\
\doteq \quad & 1 + H(p_T)
\end{aligned}
$$

- $H(p)$ is the entropy of the distribution p.

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 $T \rightarrow \infty$
- $H(p) \doteq \lim_{T\rightarrow\infty} \frac{H(p_T)}{T}$ 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 (suggested project for 4 unit students).

log loss encourages unbiased prediction

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

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 $p\left(c_t \,|\, c_1, c_2, \ldots, c_{t-1}\right)$
- Then the prediction that minimizes the log loss is $p\left(c_t \,|\, c_1, c_2, \ldots, c_{t-1}\right)$.
- 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 $E(\log b_T)$

"Universal" coding

- Suppose there are N alternative prediction algorithms.

“Universal” coding

- Suppose there are N alternative prediction algorithms.
- We would like to code almost as well as the best one.

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 $\log_2 N$

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 $\log_2 N$
- Details: next class.