Information Theory for Data Science

Information sources

Prof. Giorgio Taricco

Politecnico di Torino – DET

Program of the course (prof. Taricco)

- Time allocation:
  - 30 hours = 20 slots (of 1.5-hours)
- Information sources and source codes (lossless)
  - THEORY (5 slots)
  - ASSIGNMENT (5 slots)
- Lossy source coding
  - THEORY (5 slots)
  - ASSIGNMENT (5 slots)

Tentative schedule, subject to minor modifications

Information sources

- Information sources are statistical devices generating a data stream representing any kind of information
- They emit a sequence of random variables X_n, where $n \in \mathbb{Z}$, the set of integer numbers (n may be a discrete time index)
- The complete description requires all the possible probabilities

$$P(X_{\mathcal{S}} = x_{\mathcal{S}})$$

  - The symbol $\mathcal{S}$ represents any subset of $\mathbb{Z}$
  - The event $X_{\mathcal{S}} = x_{\mathcal{S}}$ corresponds to $X_n = x_n \; \forall \, n \in \mathcal{S}$
  - While the X_n are random symbols generated by the source, the x_n are deterministic values they take on, extracted from a source alphabet

$$\mathcal{X} = \{\xi_1, \dots, \xi_M\}$$

Information sources

- Example 1
  - Binary source, $\mathcal{X} = \{0,1\}$
  - Sample subset: $\mathcal{S} = \{0,2,4,10\}$
  - Sample probability
$$P(X_{\mathcal{S}} = \{0,1,0,1\}) = P(X_0 = 0, X_2 = 1, X_4 = 0, X_{10} = 1)$$

- Example 2
  - Ternary information source, $\mathcal{X} = \{0,1,2\}$
  - Sample subset: $\mathcal{S} = \{-2,0,2\}$
  - Sample probability
$$P(X_{\mathcal{S}} = \{2,1,0\}) = P(X_{-2} = 2, X_0 = 1, X_2 = 0)$$

Stationarity of an information source

- Stationarity
  - For a fixed deterministic vector x of $|\mathcal{S}|$ values,
    $$P(X_{\mathcal{S}} = x) = P(X_{\mathcal{S}+\Delta} = x) \qquad (*)$$
  - $\Delta \in \mathbb{Z}$
  - $\mathcal{S} + \Delta \stackrel{\text{def}}{=} \{n + \Delta, n \in \mathcal{S}\}$
  - <u>Stationarity holds if eq. $(*)$ holds for every $\mathcal{S}$ and every Δ</u>

- Example 2 (cont.)
  - Sample subset: $\mathcal{S} = \{-2,0,2\}$, shift $\Delta = 4$:
  - $P(X_{\mathcal{S}} = \{2,1,0\}) = P(X_{\mathcal{S}+\Delta} = \{2,1,0\})$
  - $P(X_{-2} = 2, X_0 = 1, X_2 = 0) = P(X_2 = 2, X_4 = 1, X_6 = 0)$

Stationarity of an information source

- If $|\mathcal{S}| = 1$, stationarity implies that
$$P(X_n = x) = P(X_0 = x) \stackrel{\text{def}}{=} p_X(x)$$
- So, the probability distribution is independent of n
- The function $p_X(x)$ is the probability distribution function

- If $|\mathcal{S}| = 2$, stationarity implies that
$$P(X_n = x_1, X_{n+\Delta} = x_2) = P(X_0 = x_1, X_\Delta = x_2) \stackrel{\text{def}}{=} p_\Delta(x_1, x_2)$$
- The two-point probability distribution depends only on the time difference Δ

- Stationarity is a strong assumption, sometimes we only have weaker assumptions satisfied

Markov sources

- Markov sources are an important class of information sources *with memory*
- The information source $X_1, X_2, X_3, \ldots$ is called a Markov source with memory L if the following property holds for every $n > L$:

$$P(X_n = x_n | X_{1:n-1} = x_{1:n-1}) = P(X_n = x_n | X_{n-L:n-1} = x_{n-L:n-1})$$

  - The effective part of the conditioning clause is in the last L symbols
  - The previous symbols are irrelevant, so that we say there is memory L
- Among the applications of Markov sources there is modeling written text for data compression

The notation $a{:}\,b$ corresponds to the integer sequence from a to b

Markov sources

- We consider time-invariant* Markov sources such that, for all $n > L$,

$$P(X_n = x_n | X_{n-L:n-1} = x_{n-L:n-1}) = P(X_{L+1} = x_{L+1} | X_{1:L} = x_{1:L})$$

- The symbols $\Sigma_n \stackrel{\text{def}}{=} X_{n-L:n-1}$ represent the state of the source at time n $(> L)$
- If the symbol alphabet is $\mathcal{X}$, the state alphabet is $\mathcal{X}^L$

- We characterize time-invariant Markov sources by the distribution of the initial state $\Sigma_{L+1} = X_{1:L}$ and by the conditional probabilities

$$P(\Sigma_{L+2} = X_{2:L+1} = \sigma_2 \mid \Sigma_{L+1} = X_{1:L} = \sigma_1)$$

*Time-invariance is a restricted form of stationarity

Markov sources

- The conditional probabilities $P(\Sigma_{L+2} = \sigma_2 | \Sigma_{L+1} = \sigma_1)$ form a transition probability matrix $\boldsymbol{P}$ whose entries are defined as

$$(\boldsymbol{P})_{\sigma_1,\sigma_2} = P(\Sigma_{L+2} = \sigma_2 | \Sigma_{L+1} = \sigma_1)$$

- Usually, the states are represented by integer numbers:

$$(\boldsymbol{P})_{i,j}, \quad i,j \in \mathbb{S} \text{ (the \textit{state space})}$$

- Typically, $\mathbb{S} = \{1,2,\dots,|\mathbb{S}|\}$
- The sum of the row elements of $\boldsymbol{P}$ is equal to 1: $\sum_{j\in\mathbb{S}} (\boldsymbol{P})_{i,j} = 1$
- *State probability vectors* $\boldsymbol{p}_n$ are defined by

$$(\boldsymbol{p}_n)_i = P(\Sigma_n = i), \qquad i \in \mathbb{S}$$

Markov chains

- The evolution of the state probability vectors is a *Markov chain*
- The evolution of the state probability distribution is governed by the equations

$$\boldsymbol{p}_{n+1} = \boldsymbol{p}_n \boldsymbol{P}$$

- This is a consequence of the total probability law:

$$(\boldsymbol{p}_{n+1})_j = \boxed{P(\Sigma_{n+1} = j) = \sum_{i \in \mathbb{S}} P(\Sigma_n = i, \Sigma_{n+1} = j)}$$

$$= \sum_{i \in \mathbb{S}} P(\Sigma_n = i\,) P(\Sigma_{n+1} = j | \Sigma_n = i)$$

$$= \sum_{i \in \mathbb{S}} (\boldsymbol{p}_n)_i \, (\boldsymbol{P})_{i,j}$$

Markov chains

- The equation $\boldsymbol{p}_{n+1} = \boldsymbol{p}_n \boldsymbol{P}$ can be applied repeatedly:

$$\boldsymbol{p}_{n+m} = \boldsymbol{p}_{n+m-1}\boldsymbol{P} = \boldsymbol{p}_{n+m-2}\boldsymbol{P}^2 = \cdots = \boldsymbol{p}_n \boldsymbol{P}^m$$

- The matrix $\boldsymbol{P}^m$ corresponds to m consecutive transition steps in the Markov chain:

$$(\boldsymbol{P}^m)_{i,j} = P(\Sigma_{n+m} = j \mid \Sigma_n = i)$$

- In some cases, as $m \to \infty$, $\boldsymbol{P}^m$ converges to a matrix whose rows are all equal
- To characterize Markov chains, we need some additional characterizations

Markov chains – basic definitions

- **Definition** 1: accessibility
  A state $j \in \mathbb{S}$ is accessible from $i \in \mathbb{S}$ if $(\boldsymbol{P}^m)_{i,j} > 0$ for some $m \geq 0$
  - This definition means that "*it is possible to travel from i to j, with positive probability, in a certain number of steps*"
  - A state is always considered accessible from itself since $\boldsymbol{P}^0 = \boldsymbol{I}$ (the identity matrix) even when $(\boldsymbol{P})_{i,i} = 0$
- If *both* "j is accessible from i" and "i is accessible from j," we say that i and j *communicate* and write $i \leftrightarrow j$
- This is an *equivalence relation* since
  - $i \leftrightarrow i$ (reflexivity)
  - $i \leftrightarrow j$ implies $j \leftrightarrow i$ (symmetry)
  - $i \leftrightarrow j$ and $j \leftrightarrow k$ imply $i \leftrightarrow k$ (transitivity)

Markov chains – basic definitions

- The communication equivalence relation $i \longleftrightarrow j$ induces a partition on the space state $\mathbb{S}$ into disjoint sets A_p such that
  - $i \longleftrightarrow j$ for all $i, j \in A_p$
  - $i \nleftrightarrow j$ for all $i \in A_p$ and $j \in A_q$ with $p \neq q$
- The sets are called *communicating classes*

- State properties are called *class properties* if they are shared by all the states in a class

Markov chains – basic definitions

- **Definition** 2: *irreducibility*
  A Markov chain whose state space $\mathbb{S}$ has *a single communicating class* is said *irreducible*, otherwise it is *reducible*
- If a Markov chain is irreducible, then *all its states communicate*

- **Definition** 3a: *first return time*
  For each state $i \in \mathbb{S}$, the first return time is defined by
  $$T_i^r = \inf \{n \geq 1, \Sigma_n = i\}$$
- **Definition** 3b: *mean return time*
  For each pair of states $i, j \in \mathbb{S}$, the mean return time from i to j is
  $$\mu_{i \to j} = E\left[T_j^r \middle| \Sigma_0 = i\right]$$

State classification

- The states of a Markov chain have the following characteristics:
- *Recurrence*: a state $i \in \mathbb{S}$ is *recurrent* if, starting from i, the chain will return to i in a finite time, with probability 1:

$$P(\Sigma_n = i \text{ for some } n \geq 1 \mid \Sigma_0 = i) = 1$$

  - A state is *positive recurrent* if $\mu_{i \to i} < \infty$
  - A state is *null recurrent* if $\mu_{i \to i} \to \infty$
- *Transience*: a state is *transient* when it is *not recurrent*
- Recurrence and transience are *class properties*
- *Absorbency*: a state is *absorbing* if $(\boldsymbol{P})_{i,i} = 1$

State classification

- Consider the state $i \in \mathbb{S}$ and define the sequence

$$S_i = \{m \geq 1: (\boldsymbol{P}^m)_{i,i} > 0\}$$

- The period of the state i is the *greatest common divisor* of the S_i
- If the period is equal to 1, the state is called *aperiodic*
- If *all states* are aperiodic, the Markov chain is called aperiodic
- If $(\boldsymbol{P})_{i,i} > 0$, the state i is aperiodic
- Periodicity is a **class property**

Calculation of $\mu_{i\to j}$

- The calculation can be done recursively:

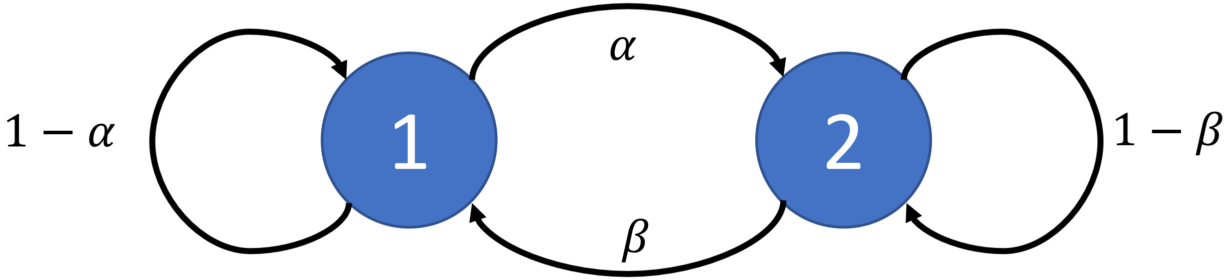
$$\mu_{i\to j} = \underbrace{1\times(\boldsymbol{P})_{i,j}}_{\text{Arrives at } j \text{ in one step}} + \sum_{k\neq j}(\boldsymbol{P})_{i,k}\times\underbrace{\{1+E[T_k^r|\Sigma_0=k]\}}_{\text{Arrives at } j \text{ in one step plus the average number of steps to reach } j \text{ from } k}$$

- Since $\sum_{k\in\mathbb{S}}(\boldsymbol{P})_{i,k}=1$ and $\mu_{i\to j}=E\left[T_j^r\middle|\Sigma_0=i\right]$, the previous equation yields:

$$\mu_{i\to j} = 1+\sum_{k\neq j}(\boldsymbol{P})_{i,k}\,\mu_{k\to j}$$

Calculation of $\mu_{i\to j}$ for the two state Markov chain

- Take, for example, the two state Markov chain:



$$\boldsymbol{P} = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}$$

- In this case,

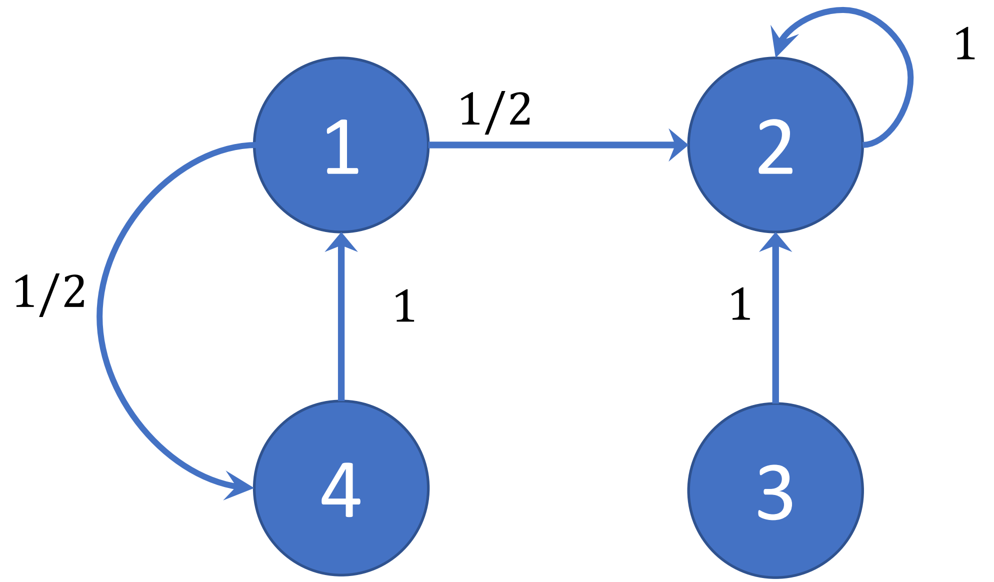
$$\mu_{1\to 1} = 1 + \alpha\mu_{2\to 1}, \qquad \mu_{2\to 1} = 1 + (1-\beta)\mu_{2\to 1}$$
$$\mu_{2\to 2} = 1 + \beta\mu_{1\to 2}, \qquad \mu_{1\to 2} = 1 + (1-\alpha)\mu_{1\to 2}$$

- Solving the linear equations, one gets:

$$\mu_{1\to 1} = 1 + \frac{\alpha}{\beta}, \qquad \mu_{2\to 1} = \frac{1}{\beta}, \qquad \mu_{2\to 2} = 1 + \frac{\beta}{\alpha}, \qquad \mu_{1\to 2} = \frac{1}{\alpha},$$

Reducible aperiodic Markov chain

- If $\boldsymbol{P}$ has one all-zero column, the associated Markov chain is reducible



$$\boldsymbol{P} = \begin{pmatrix} 0 & 1/2 & \boxed{0} & 1/2 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

States 1,4 are recurrent.
State 2 is absorbing since $(\boldsymbol{P})_{2,2} = 1$.
State 3 is transient since $(\boldsymbol{P})_{i,3} = 0$.
The communication classes are:
$$\{1,4\}, \{2\}, \{3\}$$

State evolution

- On the right, we have a sequence of powers of the transition matrix $\boldsymbol{P}$
- Whatever the initial state probability distribution $\boldsymbol{p}_0$, the stationary state distribution is $\boldsymbol{p}_\infty = (0,1,0,0)$

$$\boldsymbol{P} = \begin{matrix} 0 & 0.5000 & 0 & 0.5000 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 1.0000 & 0 & 0 & 0 \end{matrix}$$

$$\boldsymbol{P}^2 = \begin{matrix} 0.5000 & 0.5000 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0.5000 & 0 & 0.5000 \end{matrix}$$

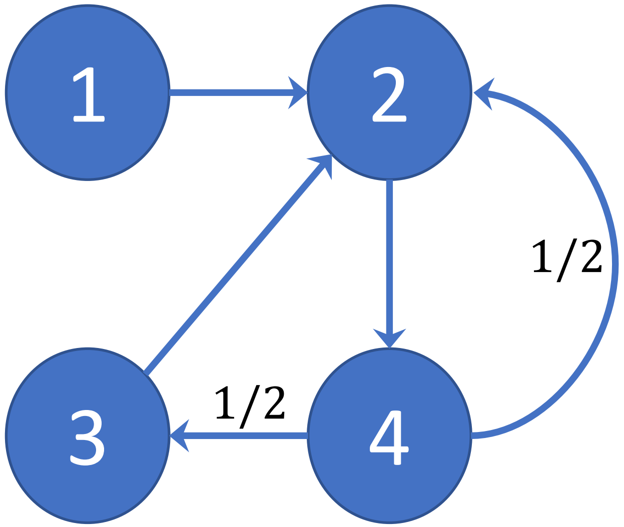
$$\boldsymbol{P}^3 = \begin{matrix} 0 & 0.7500 & 0 & 0.2500 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0.5000 & 0.5000 & 0 & 0 \end{matrix}$$

$$\boldsymbol{P}^5 = \begin{matrix} 0 & 0.8750 & 0 & 0.1250 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0.2500 & 0.7500 & 0 & 0 \end{matrix}$$

$$\boldsymbol{P}^{10} = \begin{matrix} 0.0312 & 0.9688 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0.9688 & 0 & 0.0312 \end{matrix}$$

$$\boldsymbol{P}^\infty = \begin{matrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

Reducible aperiodic Markov chain



State 1 is transient so that the chain is reducible. However, a stationary state distribution exists since the irreducibility condition is not necessary.

$$P = \begin{matrix} 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0.5000 & 0.5000 & 0 \end{matrix}$$

$$P^2 = \begin{matrix} 0 & 0 & 0 & 1.0000 \\ 0 & 0.5000 & 0.5000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 0 & 0.5000 & 0 & 0.5000 \end{matrix}$$

$$P^3 = \begin{matrix} 0 & 0.5000 & 0.5000 & 0 \\ 0 & 0.5000 & 0 & 0.5000 \\ 0 & 0.5000 & 0.5000 & 0 \\ 0 & 0.2500 & 0.2500 & 0.5000 \end{matrix}$$

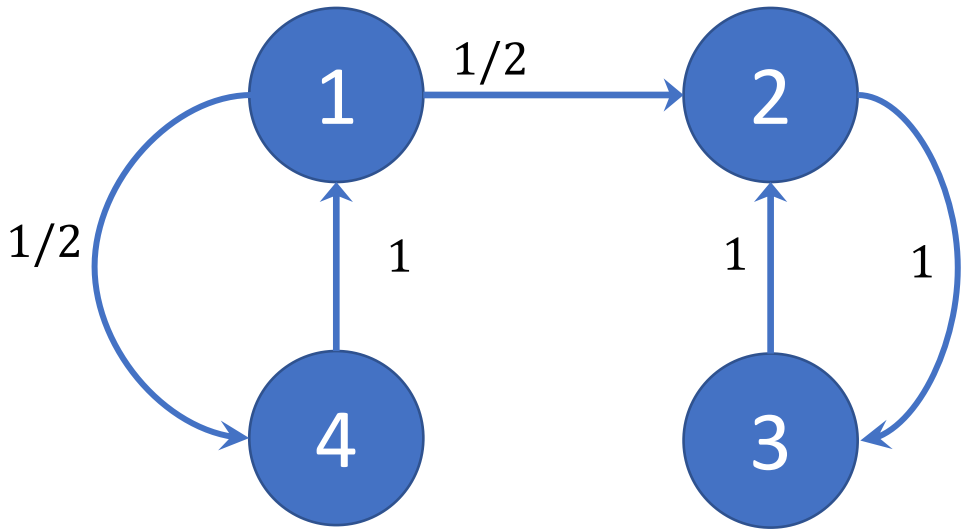
$$P^5 = \begin{matrix} 0 & 0.2500 & 0.2500 & 0.5000 \\ 0 & 0.5000 & 0.2500 & 0.2500 \\ 0 & 0.2500 & 0.2500 & 0.5000 \\ 0 & 0.3750 & 0.1250 & 0.5000 \end{matrix}$$

$$P^{10} = \begin{matrix} 0 & 0.3750 & 0.1875 & 0.4375 \\ 0 & 0.4062 & 0.2188 & 0.3750 \\ 0 & 0.3750 & 0.1875 & 0.4375 \\ 0 & 0.4062 & 0.1875 & 0.4062 \end{matrix}$$

$$P^{\infty} = \begin{matrix} 0 & 0.4000 & 0.2000 & 0.4000 \\ 0 & 0.4000 & 0.2000 & 0.4000 \\ 0 & 0.4000 & 0.2000 & 0.4000 \\ 0 & 0.4000 & 0.2000 & 0.4000 \end{matrix}$$

Reducible periodic Markov chain

- Another example of reducible Markov chain obtained by modifying the previous one:



$$\boldsymbol{P} = \begin{pmatrix} 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The communication classes are:
$\{1,4\}, \{2,3\}$

State evolution

- On the right, we have a sequence of powers of the transition matrix $\boldsymbol{P}$
- The powers of $\boldsymbol{P}$ do not converge
- Therefore, $\boldsymbol{p}_\infty$ does not exist

$$\boldsymbol{P} = \begin{matrix} 0 & 0.5000 & 0 & 0.5000 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 1.0000 & 0 & 0 & 0 \end{matrix}$$

$$\boldsymbol{P}^2 = \begin{matrix} 0.5000 & 0 & 0.5000 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 0.5000 & 0 & 0.5000 \end{matrix}$$

$$\boldsymbol{P}^3 = \begin{matrix} 0 & 0.7500 & 0 & 0.2500 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0.5000 & 0 & 0.5000 & 0 \end{matrix}$$

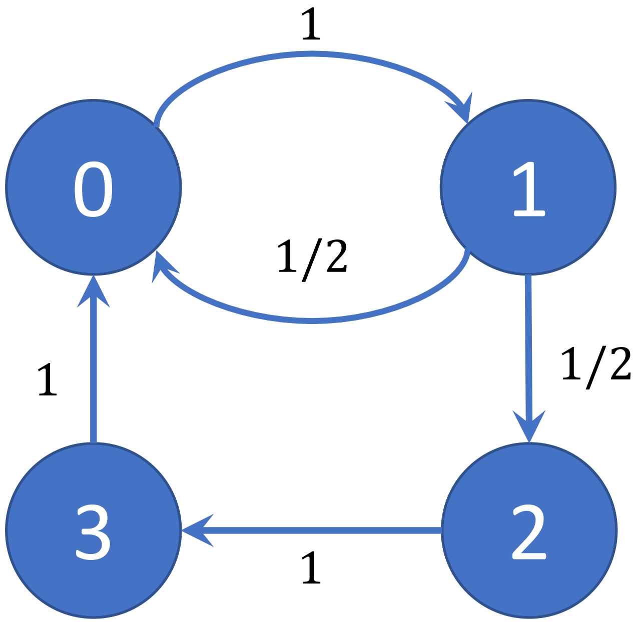
$$\boldsymbol{P}^5 = \begin{matrix} 0 & 0.8750 & 0 & 0.1250 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0.2500 & 0 & 0.7500 & 0 \end{matrix}$$

$$\boldsymbol{P}^{2n} = \begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

$$n \to \infty$$

$$\boldsymbol{P}^{2n+1} = \begin{matrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$$

Irreducible periodic Markov chain



$$\boldsymbol{P} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

- Build the S_i sequences:
- All states have period 2

$$S_0 = \{2,4,6,8,10, \dots\}$$
$$S_1 = \{2,4,6,8,10, \dots\}$$
$$S_2 = \{4,6,8,10,12, \dots\}$$
$$S_3 = \{4,6,8,10,12, \dots\}$$

State evolution

- On the right the sequence of powers of the transition matrix $\boldsymbol{P}$
- We can see that $\boldsymbol{P}^n$ does not converge but oscillates between two different matrices
- For $\boldsymbol{p}_0 = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$, we have $\boldsymbol{p}_\infty = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{6}\right)$
- But for $\boldsymbol{p}_0 = (1,0,0,0)$, $\boldsymbol{p}_\infty$ does not exist

$$\boldsymbol{P} = \begin{matrix} 0 & 1.0000 & 0 & 0 \\ 0.5000 & 0 & 0.5000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 1.0000 & 0 & 0 & 0 \end{matrix}$$

$$\boldsymbol{P}^2 = \begin{matrix} 0.5000 & 0 & 0.5000 & 0 \\ 0 & 0.5000 & 0 & 0.5000 \\ 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \end{matrix}$$

$$\boldsymbol{P}^3 = \begin{matrix} 0 & 0.5000 & 0 & 0.5000 \\ 0.7500 & 0 & 0.2500 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0.5000 & 0 & 0.5000 & 0 \end{matrix}$$

$$\boldsymbol{P}^5 = \begin{matrix} 0 & 0.7500 & 0 & 0.2500 \\ 0.6250 & 0 & 0.3750 & 0 \\ 0 & 0.5000 & 0 & 0.5000 \\ 0.7500 & 0 & 0.2500 & 0 \end{matrix}$$

$$\boldsymbol{P}^{1000} = \begin{matrix} 0.6667 & 0 & 0.3333 & 0 \\ 0 & 0.6667 & 0 & 0.3333 \\ 0.6667 & 0 & 0.3333 & 0 \\ 0 & 0.6667 & 0 & 0.3333 \end{matrix}$$

$$\boldsymbol{P}^{1001} = \begin{matrix} 0 & 0.6667 & 0 & 0.3333 \\ 0.6667 & 0 & 0.3333 & 0 \\ 0 & 0.6667 & 0 & 0.3333 \\ 0.6667 & 0 & 0.3333 & 0 \end{matrix}$$

Markov chain: convergence of $\boldsymbol{p}_n$

- Summarizing the previous cases, we have seen that the state distribution doesn't converge when the Markov chain is periodic
- However, it may converge when the Markov chain is reducible and aperiodic
- The following result provides a sufficient convergence condition:

  If a finite-state Markov chain is irreducible and aperiodic, there exists a unique stationary state probability distribution $\boldsymbol{p}_\infty$ *such that*

$$\boldsymbol{p}_\infty = \lim_{n\to\infty} \boldsymbol{p}_n = \lim_{n\to\infty} \boldsymbol{p}_0 \boldsymbol{P}^n \qquad \forall\, \boldsymbol{p}_0$$

How to calculate $\boldsymbol{p}_\infty$

- The calculation of the successive powers of $\boldsymbol{P}$ converges to a matrix whose rows are $\boldsymbol{p}_\infty$ when convergence is granted
- If a finite-state Markov chain is irreducible and aperiodic, the stationary state probability distribution $\boldsymbol{p}_\infty$ can be obtained by solving the equation
$$\boldsymbol{p}_\infty = \boldsymbol{p}_\infty \cdot \boldsymbol{P} \;\Rightarrow\; \boldsymbol{p}_\infty(\boldsymbol{I} - \boldsymbol{P}) = \boldsymbol{0}$$
- This equation system can be solved since 1 is an eigenvalue of $\boldsymbol{P}$, so that $\det(\boldsymbol{I} - \boldsymbol{P}) = 0$
- In fact, $\boldsymbol{1P} = \boldsymbol{1}$ since the sum of the row elements of $\boldsymbol{P}$ is always 1

all-1 row-vector

How to calculate $\boldsymbol{p}_\infty$

- To solve the homogeneous linear equation system

$$\boldsymbol{p}_\infty(\boldsymbol{I}-\boldsymbol{P})=\boldsymbol{0}$$

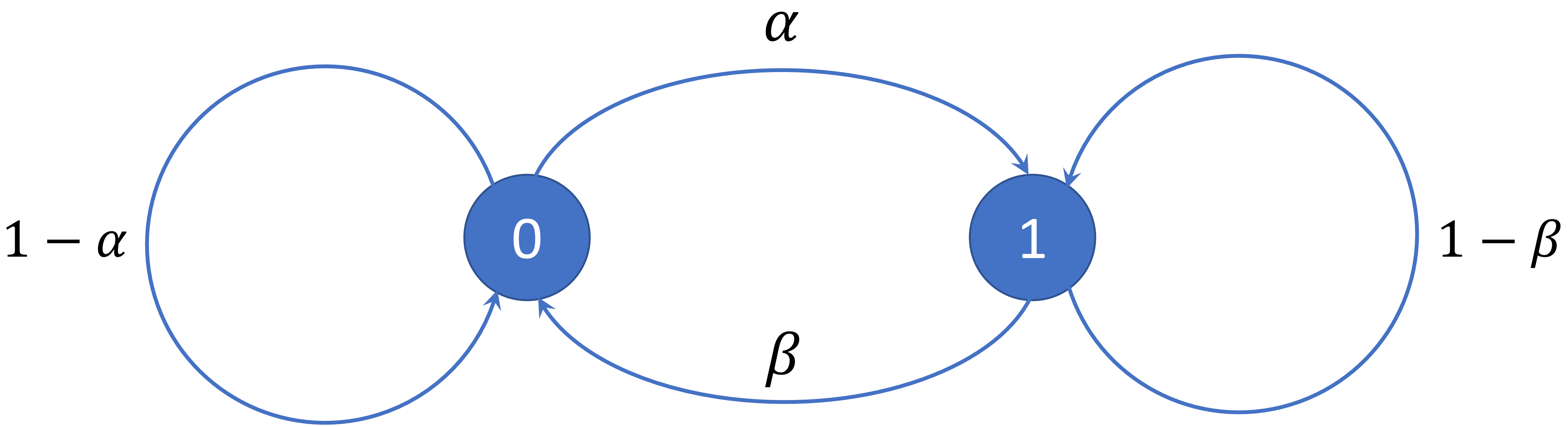
- We proceed as follows:
  - We discard one of the equations because the coefficient matrix $(\boldsymbol{I}-\boldsymbol{P})$ has not full rank
  - We replace this equation by the normalization condition that requires $\boldsymbol{p}_\infty$ to be a probability vector, so that $\boldsymbol{p}_\infty\mathbf{1}^T=1$
  - The vector $\boldsymbol{p}_\infty$ is a left-eigenvector of $\boldsymbol{P}$ corresponding to the eigenvalue 1
  - The vector $\boldsymbol{p}_\infty$ obtained by solving this linear equation system is a probability vector (nonnegative components adding to 1) by the Perron-Frobenius Theorem

Two-state Markov chain

- Probability matrix

$$\boldsymbol{P} = \begin{pmatrix} 1-\alpha & \beta \\ \alpha & 1-\beta \end{pmatrix}, \qquad \alpha, \beta \in [0,1]$$

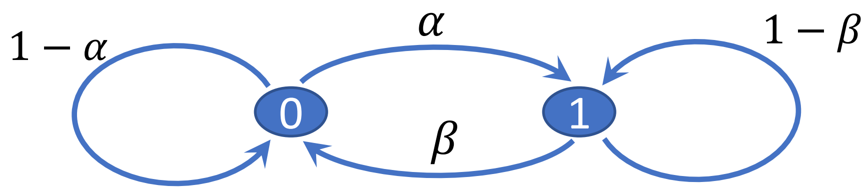
- Graph:



Two-state Markov chain

- The stationary state probability distribution $\boldsymbol{p}_\infty = (p_0, p_1)$ can be obtained by solving the equations:

$$
\begin{aligned}
(1-\alpha)p_0 + \beta p_1 = p_0 &\Rightarrow -\alpha p_0 + \beta p_1 = 0 \\
\alpha\, p_0 + (1-\beta)p_1 = p_1 &\Rightarrow \alpha p_0 - \beta p_1 = 0 \\
p_0 + p_1 = 1 &
\end{aligned}
$$



- The equation can be found by following the paths leading to each state in the graphic representation
- The solution is

$$p_0 = \frac{\beta}{\alpha+\beta}, \qquad p_1 = \frac{\alpha}{\alpha+\beta}$$

Entropy rate of a Markov source

- General definition:

$$\bar{H} \stackrel{\text{def}}{=} \lim_{n\to\infty} \frac{H(X_{1:n})}{n}$$

- Stationary independent source:

$$\bar{H} = H(X)$$

Time index dropped for stationarity

- For a stationary (not independent) source,

$$\bar{H} = \lim_{n\to\infty} H(X_n | X_{1:n-1})$$

- For a Markov source,

$$\bar{H} = H(X|\Sigma) = -\sum_{\sigma\in\mathcal{X}^L} p_\infty(\sigma) \sum_{x\in\mathcal{X}} p(x|\sigma) \log_2 p(x|\sigma)$$

Time index dropped for stationarity

Stationary state distribution

Conditional distribution

Markov source: example 1

- The first example is a two-state Markov source
- In state 0, the source emits symbols with probability distribution vector $\boldsymbol{p}_0$
- In state 1, the source emits symbols with probability distribution vector $\boldsymbol{p}_1$
- The entropy rate is

$$\overline{H} = p_\infty(0)H(\boldsymbol{p}_0) + p_\infty(1)H(\boldsymbol{p}_1)$$

$$= \frac{\beta}{\alpha+\beta}H(\boldsymbol{p}_0) + \frac{\alpha}{\alpha+\beta}H(\boldsymbol{p}_1)$$

Markov source: example 2

- We consider a binary source
  - With memory $L = 2$
  - Defined by the conditional symbol probabilities

$$P(X_n = 0|X_{n-1} + X_{n-2} = 0) = p_0 \qquad \Rightarrow P(X_n = 1|X_{n-1} + X_{n-2} = 0) = 1 - p_0$$
$$P(X_n = 0|X_{n-1} + X_{n-2} = 1) = p_1$$
$$P(X_n = 0|X_{n-1} + X_{n-2} = 2) = p_2$$

- To find the stationary state distribution we must find the matrix $\boldsymbol{P}$
- We define
  - Starting (or departure) state: X_{n-2}, X_{n-1}
  - Ending (or arrival) state: X_{n-1}, X_n

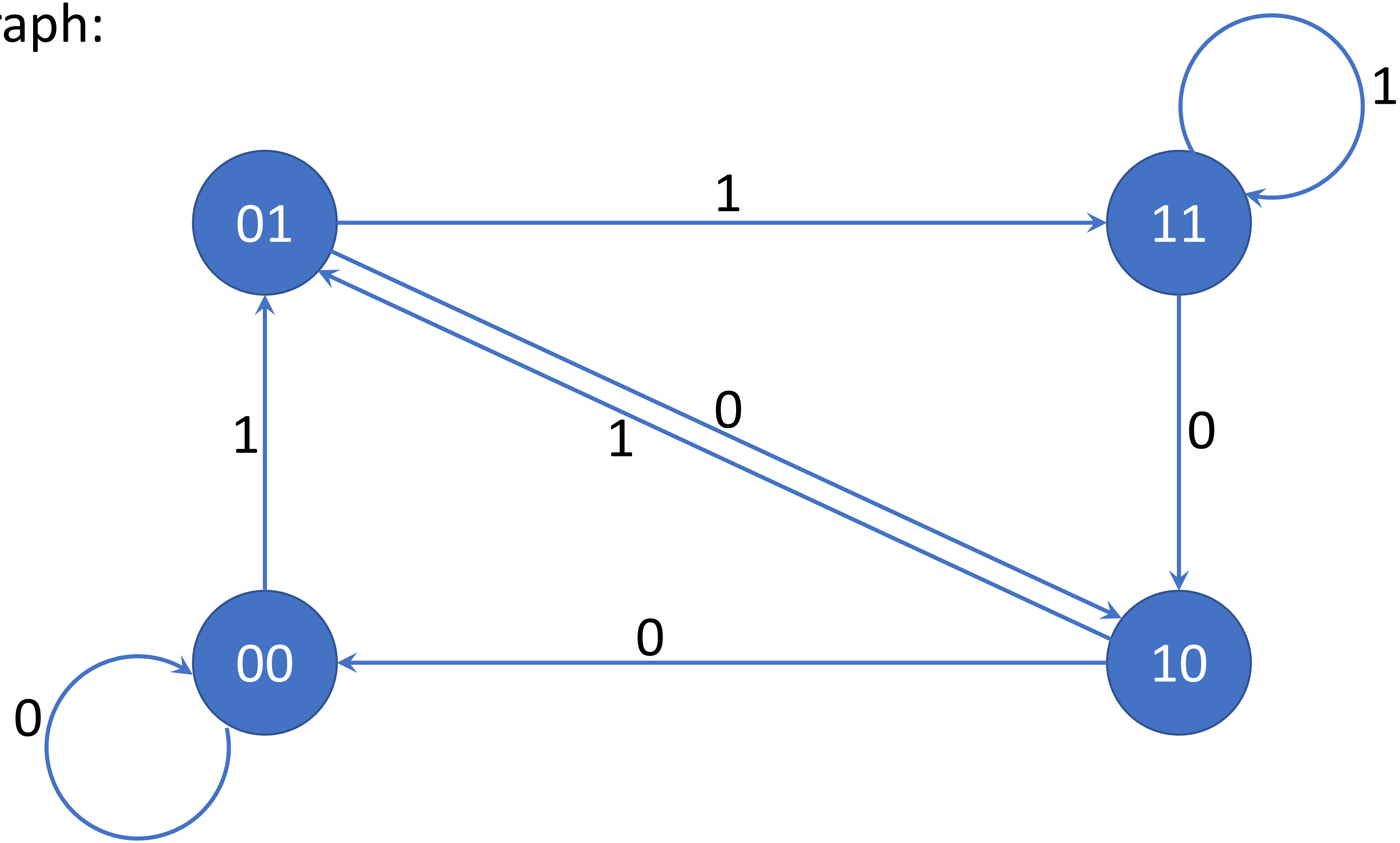
Markov source: example 2

- The state evolution is summarized by the following table:

Starting state	Current symbol	Ending state	Probability
X_{n-2}, X_{n-1}	X_n	X_{n-1}, X_n	
00	0	00	p_0
00	1	01	$1 - p_0$
01	0	10	p_1
01	1	11	$1 - p_1$
10	0	00	p_1
10	1	01	$1 - p_1$
11	0	10	p_2
11	1	11	$1 - p_2$

Markov source: example 2

- Graph:



Markov source: example 2

- Matrix:

$$\boldsymbol{P} = \begin{pmatrix} p_0 & 1-p_0 & 0 & 0 \\ 0 & 0 & p_1 & 1-p_1 \\ p_1 & 1-p_1 & 0 & 0 \\ 0 & 0 & p_2 & 1-p_2 \end{pmatrix}$$

- The states are listed as
  1. 00
  2. 01
  3. 10
  4. 11

Markov source: example 2

- Finally, we solve the stationary state state probability distribution equations:

$$
\begin{aligned}
p_{00} &= p_0 p_{00} + p_1 p_{10} \\
p_{01} &= (1 - p_0) p_{00} + (1 - p_1) p_{10} \\
p_{10} &= p_1 p_{01} + p_2 p_{11} \\
1 &= p_{00} + p_{01} + p_{10} + p_{11}
\end{aligned}
$$

- The solution is

$$
\begin{aligned}
p_{00} &= \alpha p_1 p_2 \\
p_{01} &= p_{10} = \alpha (1 - p_0) p_2 \\
p_{11} &= \alpha (1 - p_0)(1 - p_1) \\
\alpha &= \frac{1}{p_1 p_2 + 2(1 - p_0) p_2 + (1 - p_0)(1 - p_1)}
\end{aligned}
$$

Entropy rate

- Collecting the previous results, we get the entropy rate:

$$\overline{H} = \frac{p_1 p_2 H_b(p_0) + 2(1-p_0)p_2 H_b(p_1) + (1-p_0)(1-p_1)H_b(p_2)}{p_1 p_2 + 2(1-p_0)p_2 + (1-p_0)(1-p_1)}$$

$$H_b(p) \stackrel{\text{def}}{=} -p\log_2 p - (1-p)\log_2(1-p)$$

Information Theory and Applications

Source coding

Prof. Giorgio Taricco
Politecnico di Torino – DET

Source coding

- The goal of source coding is reducing the amount of data necessary to store a large set of symbols on a storage system
- This is sometimes referred to as data compression
- Source coding must be an invertible operation: from encoded data one must be able to retrieve the original data
- Depending on the length of the source and encoded blocks, we have three types of source coding algorithms:
  - Fixed-to-fixed
  - Fixed-to-variable
  - Variable-to-fixed

Fixed-to-fixed source coding

- We consider a block of N source symbols from an alphabet $\mathcal{X}$ with cardinality $M = |\mathcal{X}|$
- There are M^N possible blocks so that a number from 0 to $M^N - 1$ identifies uniquely the source block
- Since n bits are sufficient to represent all integers between 0 and $2^n - 1$, we must have $2^n \geq M^N$ or $n \geq N \log_2 M$
- Since n must be an integer, its minimum value is given by
$$\nu = \lceil N \log_2 M \rceil$$
- The corresponding number of bits per encoded symbol is
$$\bar{n} = \frac{\nu}{N} = \frac{\lceil N \log_2 M \rceil}{\mathrm{N}} \approx \log_2 M$$

Example

- Consider the English alphabet plus some punctuation characters $\mathcal{X} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \dots, \mathrm{Z}, .\,, ',', '\ ', ;\,, :\,, !\}$ with $M = |\mathcal{X}| = 32$ characters
- $\mathcal{X}$ is encoded as follows:

A	00000	B	00001	C	00010	D	00011	E	00100
F	00101	G	00110	H	00111	I	01000	J	01001
K	01010	L	01011	M	01100	N	01101	O	01110
P	01111	Q	10000	R	10001	S	10010	T	10011
U	10100	V	10101	W	10110	X	10111	Y	11000
Z	11001	.	11010	,	11011	‘ ‘	11100	;	11101
:	11110	!	11111						

Example

- Encode the sentence “GOOD NIGHT!”

A	00000	B	00001	C	00010	D	00011	E	00100
F	00101	G	00110	H	00111	I	01000	J	01001
K	01010	L	01011	M	01100	N	01101	O	01110
P	01111	Q	10000	R	10001	S	10010	T	10011
U	10100	V	10101	W	10110	X	10111	Y	11000
Z	11001	.	11010	,	11011	‘ ‘	11100	;	11101
:	11110	!	11111						

Decoding

- Decoding fixed-to-fixed source codes is very simple
- We know how many bits (ν) form one encoded character
- We process the bit stream extracting blocks of ν bits and use a look-up table to obtain the source symbols
- For example
- "OPEN BOOK" $\rightarrow$
  01110 01111 00100 01101 11100 00001 01110 01110 01010

O	P	E	N		B	O	O	K

Fixed-to-variable source coding

- The concept is that a fixed number of symbols is encoded into a variable number of bits
- The approach is advantageous, on average, if the source symbols have different probabilities
  - Most likely source symbols are assigned shorter codewords
  - On the other hand, longer codewords are assigned to less likely source symbols but their impact is limited on the average codeword length

Fixed-to-variable source coding

- Assume a stationary source generating symbols from an alphabet $\mathcal{X}$, encoded individually
- Let the probability distribution be
$$p_i \stackrel{\text{def}}{=} P(X = \xi_i), \qquad i = 1, \dots, (M \stackrel{\text{def}}{=} |\mathcal{X}|)$$
- Let the codeword lengths be ν_i (bit/symbol)
- The average number of bit per symbol for an encoded string is
$$\langle \nu \rangle = \frac{1}{N} \sum_{i=1}^{M} N_i \nu_i$$
  - N: n. of symbols in the string
  - N_i: n. of occurrences of ξ_i

Fixed-to-variable source coding

- For a very long string, the frequencies of the symbols are approximately the probabilities of their occurrences:

$$\frac{N_i}{N} \approx p_i, \qquad i = 1, \ldots, M$$

- Accordingly, the average number of bit per symbol for an encoded string is approximated by

$$\bar{\nu} = \sum_{i=1}^{N} p_i \nu_i$$

- $\bar{\nu}$ is the expected average number of bit per symbol of the source code and represents a quality measure for the source code
- The lower $\bar{\nu}$, the better the source code

Decoding fixed-to-variable source codes

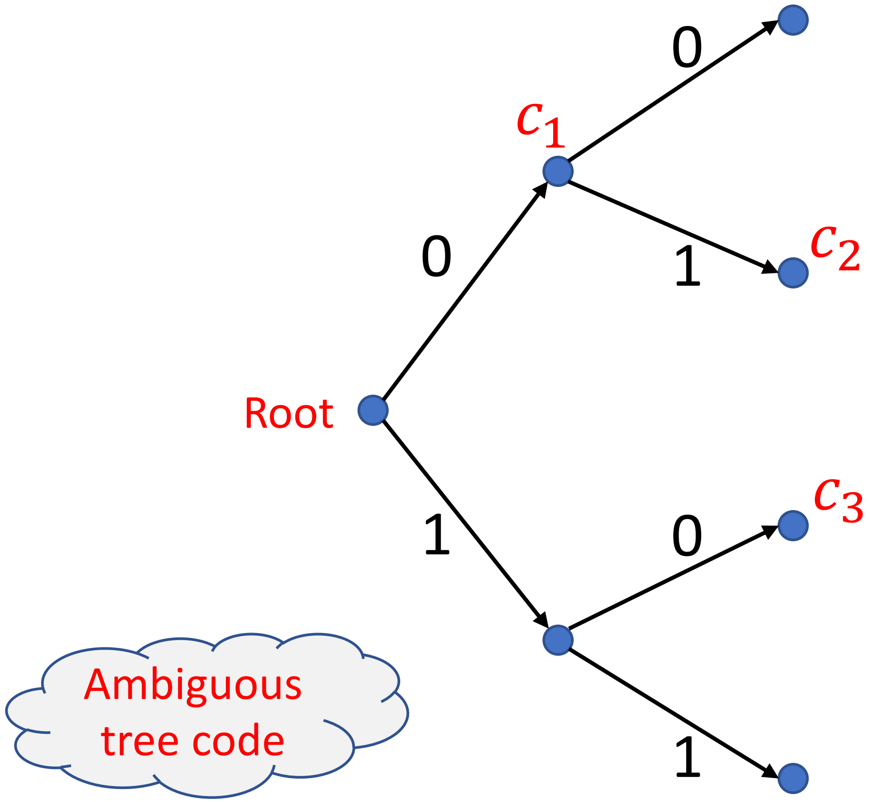
- Decoding is complicated by the fact that one doesn't know where encoded symbols are separated
- This may generate decoding ambiguity
- For example, consider the code

$$c(\xi_1) = 0, \qquad c(\xi_2) = 01, \qquad c(\xi_3) = 10$$

- Apply this code to the sequence symbols
  - $\xi_1\ \xi_3\ \xi_2\ \xi_1$
  - $\xi_2\ \xi_1\ \xi_2\ \xi_1$
- In both cases we obtain 010010

Decoding fixed-to-variable source codes

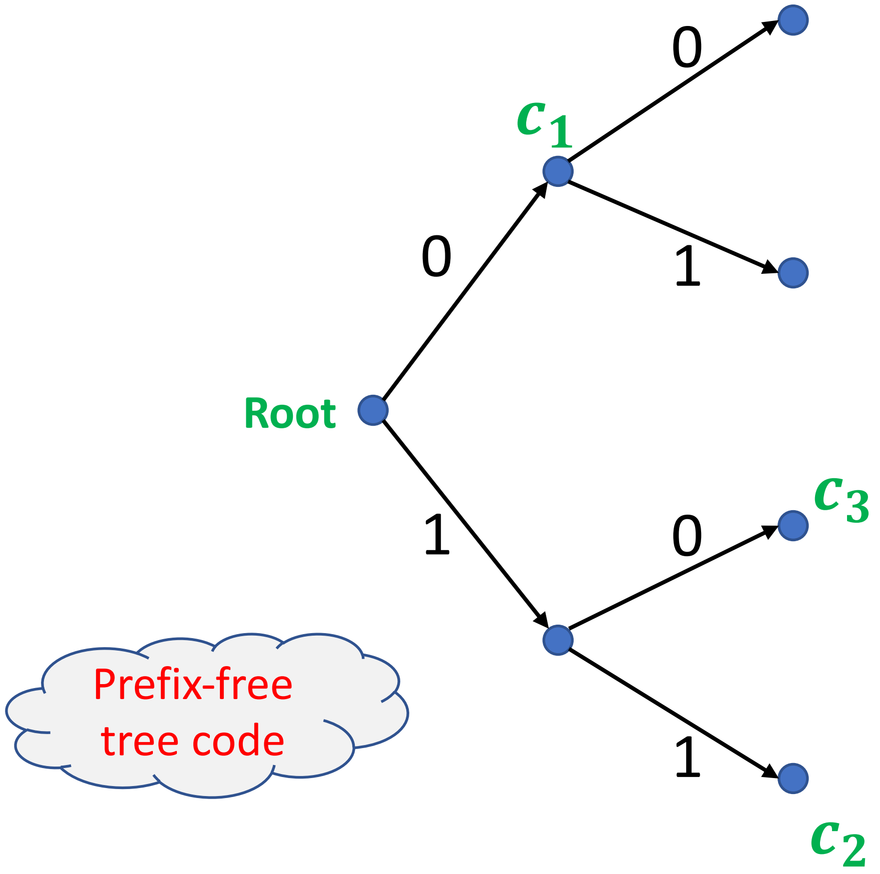
- To resolve decoding ambiguity codes must have a tree structure



- The code words (bit sequences representing each symbol) are located at tree nodes
- The bits of each code word are read off traveling the tree from the root to the node
- For example, c_2 is read passing through a path labelled 0 and another labelled 1
- If one code word is at a node inside the path to another codeword, the former is prefix of the latter
- Fixed-to-variables source codes are uniquely decodable if they satisfy the prefix-free condition

Decoding fixed-to-variable source codes

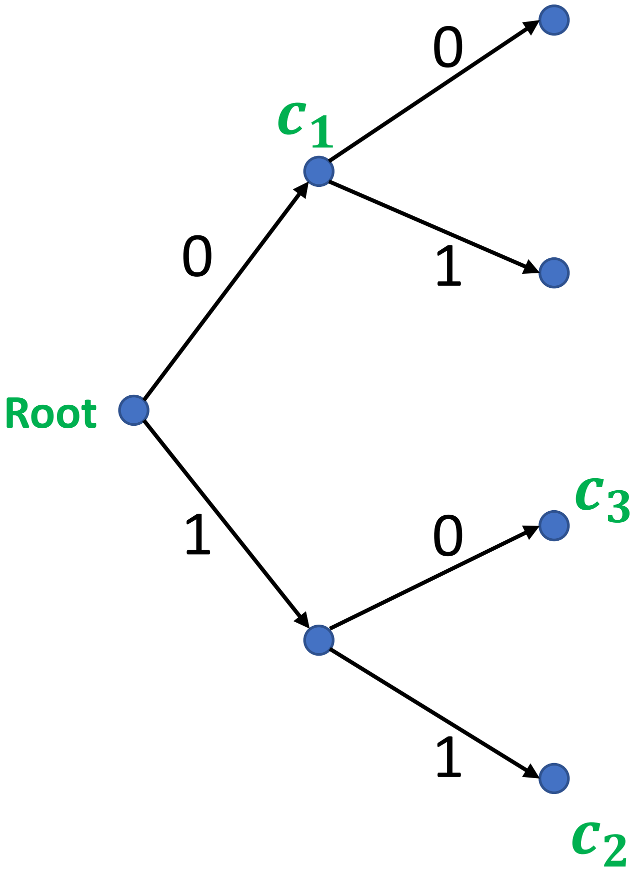
- The following code satisfies the no-prefix condition:



- Apply this code to the sequence symbols
  - $\xi_1\ \xi_3\ \xi_2\ \xi_1$
  - $\xi_2\ \xi_1\ \xi_2\ \xi_1$
- In the first case we get
  - 010110
- In the second case:
  - 110110

Decoding fixed-to-variable source codes

- If a code satisfies the no-prefix condition, decoding is always uniquely possible



- Decode the bit sequence 0111001110

Kraft inequality

- This is a simple inequality characterizing the code word length distribution for a uniquely decodable tree code
- Every code word is located on one node of the tree
- The nodes reachable traveling the tree from this node are said to be controlled
- A node at depth ν_i controls $2^{\nu_{\max}-\nu_i}$ nodes at depth $\nu_{\max}$
- The sub-tree stemming from it up to depth $\nu_{\max}$ does not contain other codewords to satisfy the prefix-free condition
- We have the inequality

$$\sum_{i=1}^{M} 2^{\nu_{\max}-\nu_i} \leq 2^{\nu_{\max}}$$

Kraft inequality

- Inequality

$$\sum_{i=1}^{M} 2^{\nu_{\max}-\nu_i} \leq 2^{\nu_{\max}}$$

- The total number of controlled nodes at depth $\nu_{\max}$ cannot exceed the number of nodes at depth $\nu_{\max}$, i.e., $2^{\nu_{\max}}$
- Dividing both sides, we obtain the Kraft inequality:

$$\sum_{i=1}^{M} 2^{-\nu_i} \leq 1$$

Lower bound to $\bar{\nu}$

- We want to choose the code word lengths ν_i to minimize the average number of bits per symbol

$$\bar{\nu} = \sum_{i=1}^{M} p_i \nu_i$$

- Our constraint is the Kraft inequality:

$$\sum_{i=1}^{M} 2^{-\nu_i} \leq 1$$

- Applying the method of Lagrange multipliers, we build the Lagrangian function for this problem:

$$\mathcal{L}(\nu_1, \dots, \nu_M) = \sum_{i=1}^{M} p_i \nu_i + \lambda \left(\sum_{i=1}^{M} 2^{-\nu_i} - 1 \right)$$

Lower bound to $\bar{\nu}$

- This is a convex optimization problem which can be solved by the following equations:

$$\frac{\partial \mathcal{L}}{\partial \nu_i} = p_i - \lambda \cdot \ln 2 \cdot 2^{-\nu_i} = 0$$

$$\lambda \left(\sum_{i=1}^{M} 2^{-\nu_i} - 1 \right) = 0 \quad (*)$$

- The solution requires $\lambda \cdot \ln 2 = 1$ since $\sum_{i=1}^{M} p_i = 1$ so that $(*)$ is satisfied
- We get $\nu_i = -\log_2 p_i$ and

$$\bar{\nu} = \sum_{i=1}^{M} p_i(-\log_2 p_i) = H(X)$$

Lower bound to $\bar{\nu}$

- The solution $\nu_i = -\log_2 p_i$ is acceptable only if the lengths are all integer numbers
- In that case, $\bar{\nu} = H(X)$
- Otherwise, the source entropy is a strict lower bound:
$$\bar{\nu} > H(X)$$
- Summarizing, Kraft inequality implies that

$$\bar{\nu} \geq H(X)$$

Upper bound to $\bar{\nu}$

- We can replace the lengths $\nu_i = -\log_2 p_i$ by their integer upper bounds:
$$\nu_i = \lceil -\log_2 p_i \rceil$$
- For every real number x, $\lceil x \rceil < x + 1$
- Then,
$$\nu_i = \lceil -\log_2 p_i \rceil < -\log_2 p_i + 1$$
- Thus,
$$\bar{\nu} = \sum_{i=1}^{M} p_i \nu_i < \sum_{i=1}^{M} p_i(-\log_2 p_i + 1) = H(X) + 1$$

Shannon Theorem

- There always exists a uniquely decodable source code such that the average number of bits per symbol $\bar{\nu}$, required to encode a stationary independent source with entropy $H(X)$, satisfies the inequalities

$$H(X) \leq \bar{\nu} < H(X) + 1$$

  - The lower bound is necessary for the existence of the code
  - The upper bound is sufficient

Information Theory and Applications

Huffman codes

Prof. Giorgio Taricco
Politecnico di Torino – DET

Examples

- Example 1: $\boldsymbol{p} = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{2}\right)$ [Check Kraft inequality and build tree code]

- Example 2: $\boldsymbol{p} = \left(\frac{1}{8}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}\right)$ [Check Kraft inequality and build tree code]

- Example 3: $\boldsymbol{p} = \left(\frac{1}{8}, \frac{3}{8}, \frac{1}{2}\right)$ [Check Kraft inequality and build tree code]

Cross entropy

- The precise knowledge of the source symbol distribution is not always possible
- Let p_i be the true probability of generation of the symbol ξ_i and q_i an estimated probability which may be different from p_i
- The optimum source code based on the estimated probabilities q_i has lengths $\nu_i = -\log_2 q_i$
- The average length based on the true probability distribution is the cross entropy:

$$\bar{\nu} = H(\boldsymbol{p}, \boldsymbol{q}) = -\sum_i p_i \log_2 q_i$$

Cross entropy inequalities

- We notice that $H(\boldsymbol{p}) = H(\boldsymbol{p}, \boldsymbol{p})$
- We expect that the mismatch between the true distribution $(\boldsymbol{p})$ and the estimated distribution $(\boldsymbol{q})$ increases $\bar{\nu}$:
$$H(\boldsymbol{p}, \boldsymbol{q}) \geq H(\boldsymbol{p}, \boldsymbol{p})$$
- This can be proved by applying the inequality
$$\ln x \leq x - 1, \qquad (x > 0)$$
- In fact,
$$H(\boldsymbol{p}, \boldsymbol{p}) - H(\boldsymbol{p}, \boldsymbol{q}) = \sum_i p_i \log_2 \frac{q_i}{p_i} = \frac{\sum_i p_i \ln \frac{q_i}{p_i}}{\ln 2} \leq \frac{\sum_i p_i \left(\frac{q_i}{p_i} - 1\right)}{\ln 2} = 0$$

Kullback-Leibler divergence

- The difference between the mismatched and optimum $\bar{\nu}$ is the Kullback-Leibler divergence:

$$D(\boldsymbol{p}||\boldsymbol{q}) \stackrel{\text{def}}{=} H(\boldsymbol{p},\boldsymbol{q}) - H(\boldsymbol{p},\boldsymbol{p}) \geq 0$$

- We also can write

$$D(\boldsymbol{p}||\boldsymbol{q}) \stackrel{\text{def}}{=} \sum_i p_i \log_2 \frac{p_i}{q_i}$$

- There is no ordering, instead, between $H(\boldsymbol{p},\boldsymbol{q})$ and $H(\boldsymbol{p},\boldsymbol{q})$, i.e., the difference $H(\boldsymbol{p},\boldsymbol{q}) - H(\boldsymbol{q},\boldsymbol{q}) = \sum_i (q_i - p_i) \log_2 q_i$ can be negative, zero, or positive

Codes for stationary Markovian sources

- A Markovian source is characterized by a conditional probability distribution $p(x|\sigma)$
  - $x \in \mathcal{X}$ is the source symbol
  - $\sigma \in \mathcal{S}$ is the source state
- Choosing a specific source state σ, we can repeat the previous analysis and select a specific source code depending on σ
  - The optimum lengths are $-\log_2 p(x|\sigma)$
  - The average number of bits per symbol, conditional on σ, satisfies

$$-\sum_{x\in\mathcal{X}} p(x|\sigma)\log_2 p(x|\sigma) \le \bar{\nu}_\sigma$$

Codes for stationary Markovian sources

- There exists a source code such that the average number of bits per symbol, conditional on σ, satisfies

$$\bar{\nu}_\sigma < -\sum_{x\in\mathcal{X}} p(x|\sigma)\log_2 p(x|\sigma) + 1$$

- Averaging with respect to the stationary state distribution $p(\sigma)$ and defining the average number of bits per symbol as ν, we get

$$\underbrace{\left\{\sum_{\sigma\in\mathcal{S}} p(\sigma)\left[-\sum_{x\in\mathcal{X}} p(x|\sigma)\log_2 p(x|\sigma)\right]\right\}}_{\bar{H}=H(X|\Sigma)} \le \sum_{\sigma\in\mathcal{S}} p(\sigma)\bar{\nu}_\sigma = \bar{\nu}$$

- In a similar way:

$$\bar{\nu} < \bar{H} + 1$$

Huffman codes

- Huffman codes are prefix-free tree codes that minimize the average number of bits per symbol
- In this sense they are optimal source codes
- They are derived by constructing the code tree starting from the leaves
- The detailed construction algorithm is described in the following slide for the usual stationary independent source with symbols from
$$\mathcal{X} = \{\xi_1, \dots, \xi_M\}$$
- The symbol probabilities are $p_i = P(X_n = \xi_i)$ for $i = 1, \dots, M$

Huffman code construction algorithm

- The starting point is the set of M leave nodes corresponding to the symbols $\xi_i, i = 1, \ldots, M$
- The nodes are processed sequentially according to the following rules:
  - Select two nodes with minimum probabilities
  - Build a sub-tree stemming from a new node with arcs labelled 0 and 1 reaching the two original nodes
  - Replace the two nodes with the new node and assign it a probability equal to the sum of the probabilities of the two original nodes
  - Terminate the construction when only one node (the tree root) remains

Examples

- Example 1: $\boldsymbol{p} = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{2}\right)$ [Build the Huffman code]
- Example 2: $\boldsymbol{p} = \left(\frac{1}{8}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}\right)$ [Build the Huffman code]
- Example 3: $\boldsymbol{p} = \left(\frac{1}{8}, \frac{3}{8}, \frac{1}{2}\right)$ [Build the Huffman code]

Storage of the compressed data

- The application of a source code to a sequence of source symbols of given length requires two types of data storages:
  - The code tree containing the symbols and their bit encodings
  - The sequence of encoded bits
- Consider for example a ternary source with symbols A,B,C with probabilities $0.25,0.25,0.5$
- The tree (Huffman) code is
  - A $\rightarrow$ 00
  - B $\rightarrow$ 01
  - C $\rightarrow$ 1

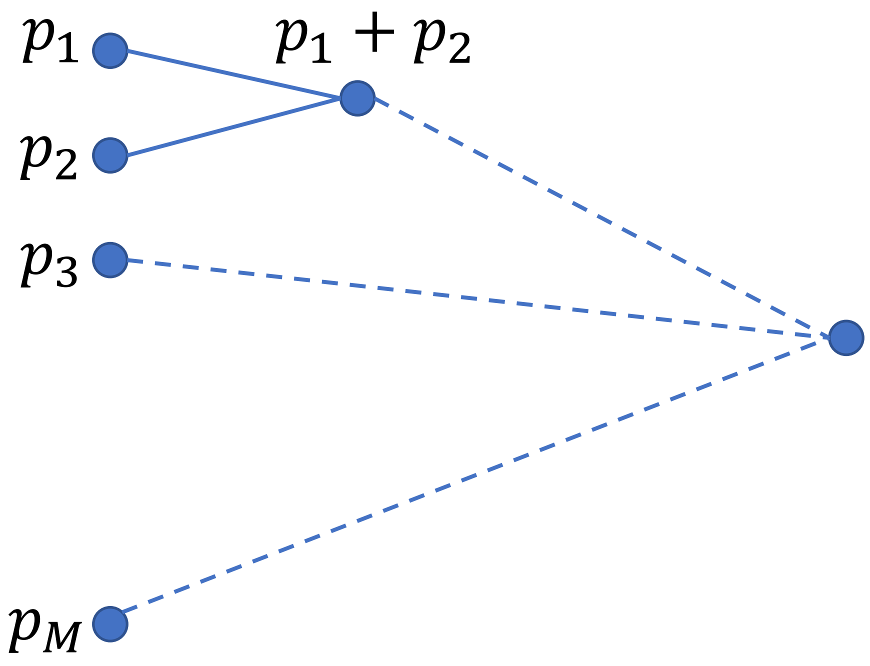
Storage of the compressed data

- Consider the source sequence “ABCCCCBABABACCCC” (16 symbols)
- Its encoding is “000111110100010001001111” (24 bits)
- The stored data are:
  - The tree: 00,A,01,B,1,C
  - The encoded bits: 000111110100010001001111
- Usually, the storage of the tree code is negligible with respect to the storage of the encoded bits
- In this example, the actual number of bits per symbol required is $\bar{\nu} = \frac{24}{16} = 1.5$
- The source entropy is $\bar{H} = H(0.25,0.25,0.5) = 1.5 = \bar{\nu}$

Calculating $\bar{\nu}$ for Huffman codes

- The average number of bits per symbol $\bar{\nu}$ can be calculated recursively
- Clearly, $\bar{\nu}$ is a function of the probability vector $\boldsymbol{p} = (p_1, \dots, p_M)$:
$$\bar{\nu} = \phi(\boldsymbol{p})$$
- This function is invariant to permutations of the probabilities so that we can assume them sorted in increasing (at least nondecreasing) order: $p_1 \leq p_2 \leq \cdots \leq p_M$
- Now we consider one step in the Huffman code construction algorithm

Calculating $\bar{\nu}$ for Huffman codes



- The full code is obtained by adding two nodes to the dashed code
- The contribution to $\bar{\nu}$ in the full code of the two nodes p_1 and p_2 is
$$p_1\nu_1 + p_2\nu_2$$
- If the path to the node $p_1 + p_2$ has length ν_{12}, we have $\nu_1 = \nu_2 = \nu_{12} + 1$
- Then,
$$p_1\nu_1 + p_2\nu_2 = (p_1 + p_2)(\nu_{12} + 1)$$
- The contribution to $\bar{\nu}$ in the dashed code of the node $p_1 + p_2$ is $(p_1 + p_2)\nu_{12}$
- The other nodes p_3 to p_M give the same contribution to $\bar{\nu}$ in the full code and in the dashed code: $p_3\nu_3 + \cdots + p_M\nu_M$
- Then,
$$\phi(p_1, p_2, p_3, \dots, p_M) = p_1 + p_2 + \phi(p_1 + p_2, p_3, \dots, p_M)$$

Examples

- Apply the previous recursive rule to calculate $\bar{\nu}$ with the previous recursive rule and check the inequalities $H(X) \leq \bar{\nu} < H(X) + 1$
  1. $\boldsymbol{p} = (0.1, 0.3, 0.6)$
  2. $\boldsymbol{p} = (0.1, 0.1, 0.2, 0.2, 0.4)$
  3. $\boldsymbol{p} = (0.5, 0.25, 0.125, 0.125)$

Information Theory and Applications

Dictionary methods

Prof. Giorgio Taricco

Politecnico di Torino – DET

Variable-to-fixed encoding

- Complementary approach wrt fixed-to-variable
- Variable length source strings are encoded into fixed length bit vectors
- The set of possible variable length strings is called dictionary so that these are called dictionary methods
- The dictionary changes dynamically, as the source symbols are read off, according to an encoding algorithm
- Dictionary methods were introduced by Jacob Ziv, Abraham Lempel, and Terry Welch

Dictionary methods

- For an extended study of dictionary methods see:
  - W.A. Pearlman and A. Said, *Digital Signal Compression,* Cambridge University Press 2011
  - D. Salomon and G. Motta, *Handbook of Data Compression, 5th ed.,* Springer, 2010
- The fixed-to-variable methods we considered before are statistical methods
- These methods use a statistical model of the data, and the quality of compression they achieve depends on how good that model is
- Dictionary methods are not based on a statistical model, they select strings of symbols and encode each string as a token using a dictionary
- The dictionary holds strings of symbols, and it may be static or dynamic
- Static is permanent, sometimes permitting the addition of strings but no deletions
- Dynamic holds strings previously found in the input stream, allowing for additions and deletions as new input is read

English dictionary method

- The simplest example of a static dictionary is a dictionary of the English language used to compress English text
- Imagine a dictionary with half a million words (without definitions)
- A word is read from the input stream and the dictionary is searched
- If a match is found, an index to the dictionary is written into the output stream
- Otherwise, the uncompressed word is written
- The compressed data contains indexes and raw words
- We distinguish them by an extra bit at the beginning

English dictionary method

- With 19 bits we can select each word from an English dictionary containing $2^{19} = 524{,}288$ words
- If the word to encode is in the dictionary we need 20 bits (including the extra bit)
- If it is not in the dictionary, we may encode it as follows:
  - Extra bit
  - 8 bits denoting the number of characters (1-256)
  - Character encoding according to some alphabet, eg, ASCII
- If the dictionary is comprehensive, most source words require 20 bit

English dictionary method

- A simple improvement can be obtained by using 2 extra bits
  - 00, if the word is in the first 1024 most likely words in the dictionary
  - 01, if it is in the next 16384 most likely words
  - 10, if it is in the remaining words
  - 11, if it is not in the dictionary
- The number of bits required for the encoding is
  - Case 00: $2 + 10 = 12$ bits
  - Case 01: $2 + 14 = 16$ bits
  - Case 10: $2 + 19 = 21$ bits
  - Case 11: variable

Adaptive methods

- The efficiency of a static method depends on the goodness of the statistical model (word probability distribution)
- With text files, it depends on the language, of course
- With other types of files (*eg,* binary files), a statistical model is difficult to predict
- An adaptive method is usually much better
  - It starts with a small default dictionary
  - It adds words as they are found from the source
  - It deletes old unused words to keep the size small and the search time short

String compression

- Source codes based on strings of symbols are more efficient than those based on individual symbols
- Example: stationary source generating binary symbols with probability $P(X_n = 0) = 0.7$
- Apply Huffman coding to strings of m symbols
- The probability distribution of a string of two symbols is:

String	Probability
00	0.49
01	0.21
10	0.21
11	0.09

String compression

- By applying Huffman encoding to progressively longer strings we report the values of the average number of bits per symbol

m	$\frac{\bar{\nu}}{m}$
1	1
2	0.9050
3	0.9087
4	0.8918
5	0.8890
6	0.8882

- The sequence converges very slowly to $H_b(0.7) = 0.8813$

String compression

- The slow convergence is due to the fact that the encoded string length is fixed
- The number of possible strings grows exponentially with their length (it is 2^m)
- Strings to be encoded must be selected more carefully

Lempel-Ziv codes

- Jacob Ziv and Abraham Lempel were the developers of the first dictionary methods for data compression in the late 1970's
- Their first algorithm was LZ77
- The principle of LZ77 is to fill the dictionary with parts of the input stream as it is read off
- The method is based on a sliding window divided in two parts:
    - Search buffer
    - Look-ahead buffer

LZ77

- The search buffer contains already encoded symbols
- The look-ahead buffer contains symbols to be encoded
- In practical implementations, the search buffer is long a few thousands symbols and the look-ahead buffer a few tens
- The encoded output consists of tokens represented by three components:
  - Offset = distance of the first encoded symbol to the end of the search buffer
  - Length = length of the encoded string
  - Next symbol = first symbol after the encoded string
- The two buffers are implemented as sliding windows

LZ77 algorithm

- Initially,
  - the search buffer is empty
  - the symbols to be encoded fill the look-ahead buffer
- Repeat until the look-ahead buffer is empty:
  - Scan the search buffer to find the longest string matching the first symbols in the look-ahead buffer
  - If nothing is found
    - Output token <0,0,first symbol in look-ahead buffer>
  - else
    - Output token <offset, length, next symbol after encoded string in look-ahead buffer>
  - Move the buffer boundary to the right up to the next symbol to encode

LZ77 example: “cat cat catering”

- Assume buffer length is 8
- The $ sign is used to terminate the string to encode

```
Search     Look-ahead  Token
"        " "cat cat " <0,0,"c">
"       c" "at cat c" <0,0,"a">
"      ca" "t cat ca" <0,0,"t">
"     cat" " cat cat" <0,0," ">
"    cat " "cat cate" <4,4,"c">
"at cat c" "atering$" <4,2,"e">
"cat cate" "ring$   " <0,0,"r">
"at cater" "ing$    " <0,0,"i">
"t cateri" "ng$     " <0,0,"n">
" caterin" "g$      " <0,0,"g">
"catering" "$       " <0,0,"$">
```

Another example: “sir sid eastman easily teases sea sick seals”

```
"        " --- "sir sid " --- <0,0,"s">
"       s" --- "ir sid e" --- <0,0,"i">
"      si" --- "r sid ea" --- <0,0,"r">
"     sir" --- " sid eas" --- <0,0," ">
"    sir " --- "sid east" --- <4,2,"d">
" sir sid" --- " eastman" --- <4,1,"e">
"ir sid e" --- "astman e" --- <0,0,"a">
"r sid ea" --- "stman ea" --- <6,1,"t">
"sid east" --- "man easi" --- <0,0,"m">
"id eastm" --- "an easil" --- <4,1,"n">
" eastman" --- " easily " --- <8,4,"i">
"man easi" --- "ly tease" --- <0,0,"l">
"an easil" --- "y teases" --- <0,0,"y">
"n easily" --- " teases " --- <7,1,"t">
"easily t" --- "eases se" --- <8,3,"e">
"ly tease" --- "s sea si" --- <2,1," ">
" teases " --- "sea sick" --- <4,2,"a">
"ases sea" --- " sick se" --- <4,2,"i">
"s sea si" --- "ck seals" --- <0,0,"c">
" sea sic" --- "k seals$" --- <0,0,"k">
"sea sick" --- " seals$ " --- <5,2,"e">
" sick se" --- "als$    " --- <0,0,"a">
"sick sea" --- "ls$     " --- <0,0,"l">
"ick seal" --- "s$      " --- <4,1,"$">
```

ZIP methods

- Several compression methods named *ZIP have been proposed in later years
- The first was PKZIP developed by P.W. Katz in 1987
- The core of PKZIP is the method called DEFLATE
- DEFLATE is a variation of LZ77 combined with Huffman codes

DEFLATE

- DEFLATE does not output tokens as LZ77
- DEFLATE still scans the search buffer for the longest string matching the initial part of the look-ahead buffer but it outputs
  - An unmatched character when the search fails
  - A pair (offset, length) when it was successful
- Unmatched characters and offsets have byte dimension
- Lengths have 15 bits
- DEFLATE output is stored by using a special Huffman code

Book references

- T.M. Cover and J.A. Thomas, Elements of Information Theory. Wiley, 2006
- R.M. Gray, *Entropy and Information Theory*, Springer, 2011
- G. Grimmett and D. Stirzaker, Probability and Random Processes 3rd ed. Oxford, 2001
- G. Grimmett and D. Stirzaker, One Thousand Exercises in Probability. Oxford, 2001
- M. Lefebvre, Basic Probability Theory with Applications. Springer Undergraduate Texts in Mathematics and Technology, 2009
- A. El Gamal and Y.-H. Kim, Network Information Theory. Cambridge University Press, 2011
- M.R.D. Rodriguez and Y. Eldar, Information-Theoretic Methods in Data Science, Cambridge University Press, 2021