Programmazione Data Security 3

Jacopo Manetti

June 2023

1 Codici di Huffman

1.1 Traccia

Programmare in Python l'algoritmo di Huffman. La procedura accetta una lista di lettere (l'alfabeto), la relativa distribuzione di probabilità e restituisce un codice istantaneo ottimo per la distribuzione data. Programmare anche una procedura di decodifica che, preso come argomento un generico codice prefix-free (non necessariamente Huffman) e una stringa binaria, proceda alla decodifica della stringa.

1.2 Svolgimento

Il codice Python fornito implementa l'algoritmo di Huffman per la codifica di un alfabeto basato su una distribuzione di probabilità data, e include anche una funzione per la decodifica di una stringa binaria data un codice prefix-free. Il codice viene spiegato di seguito:

- Iniziamo con la definizione di una classe **Node**, che rappresenta un nodo nell'albero di Huffman. Ogni nodo contiene un simbolo (che rappresenta una lettera dell'alfabeto), la sua frequenza (calcolata sulla base della distribuzione di probabilità) e riferimenti ai nodi figli sinistro e destro. Il nodo ha anche un attributo *huff* che rappresenta il codice di Huffman assegnato a quel nodo.
- La funzione **calc_freq** prende come argomenti un insieme di lettere e le corrispondenti probabilità, e ritorna un dizionario dove la chiave è la lettera e il valore è la sua probabilità.
- La funzione **huffman_code_node** assegna un codice di Huffman a ogni nodo. Questa funzione viene chiamata ricorsivamente per ciascuno dei figli di un nodo, con il codice del nodo passato come prefisso.
- La funzione **huffman_coding** implementa l'algoritmo di Huffman per creare un codice ottimale basato su una distribuzione di probabilità data. Questa funzione inizia calcolando le frequenze delle lettere, poi crea un

nodo per ogni lettera. Successivamente, costruisce l'albero di Huffman combinando i due nodi con la minor frequenza fino a quando non rimane un solo nodo. Infine, assegna un codice di Huffman a ogni nodo utilizzando la funzione *huffman_code_node*.

- La funzione **decode** prende come argomenti un codice prefix-free e una stringa binaria, e restituisce la stringa di testo decodificata. Questa funzione scansiona la stringa binaria bit per bit, accumulando i bit in *current_code* fino a quando non trova un simbolo nel codice prefix-free che corrisponde a *current_code*. Quando ciò si verifica, aggiunge il simbolo corrispondente al testo decodificato e resetta *current_code*.

Nell'esempio di utilizzo del codice, si forniscono un insieme di lettere e le corrispondenti probabilità. Si chiama quindi la funzione **huffman_coding** per ottenere il codice di Huffman per ciascuna lettera. Infine, si utilizza la funzione **decode** per decodificare una stringa binaria utilizzando il codice di Huffman calcolato, si potrebbe peròutilizzare un qualsiasi altro codice prefix-free.

2 Compressione di digrammi

Si consideri un codice di compressione per i digrammi, cioè le coppie di lettere consecutive (X_1, X_2), nella lingua Inglese. La distribuzione di probabilità dei digrammi, $p_{X_1X_2}$, è per esempio ricavabile dalla tabella in questa pagina, normalizzando opportunamente:

https://en.wikipedia.org/wiki/Bigram#Bigram_frequency_in_the_English_language

(NB: per semplicità la probabilità dei digrammi che non appaiono nella tabella si assume uguale a 0).

Costruire due distinti codici di Huffman per i digrammi: il primo, C_1, basato sulla vera distribuzione $p_{X_1X_2}$, il secondo, C_2, sulla distribuzione prodotto $p_{X_1} \cdot p_{X_2}$ dove si trattano due lettere consecutive come indipendenti (NB: anche nel secondo caso, si considerino solo i digrammi nella tabella di cui sopra, e si normalizzi opportunamente). Confrontare tra loro le lunghezze medie dei codici ottenuti. Verificare in che misura la differenza tra la prima e la seconda, $\Delta := L_p(C_2) - L_p(C_1)$ dove $p = p_{X_1X_2}$, obbedisce alla linking identity. Cioè verificare se

$$\Delta \approx D(p_{X_1X_2} || p_{X_1} \cdot p_{X_2}) = I(X_1; X_2)$$

Verificare empiricamente la differenza comprimendo uno stesso in Inglese testo separatamente tramite i due codici.

2.1 Svolgimento

Il codice implementato esegue una serie di operazioni per risolvere il problema posto dalla traccia, il flusso seguito dal codice è il seguente:

1. Inizialmente, viene riportata la distribuzione dei digrammi presa dalla relativa pagina Wikipedia. Successivamente, vengono calcolate le distribuzioni di probabilità dei digrammi, $p_{X_1X_2}$, e delle lettere costituenti i digrammi, p_{X_1} e p_{X_2}. Da queste distribuzioni, il codice deriva la distribuzione prodotto $p_{X_1} \cdot p_{X_2}$, che assume che le due lettere di ogni digramma siano indipendenti.

2. Dopo aver calcolato queste distribuzioni, il codice crea due codici di Huffman (viene sfruttato il metodo **huffman_coding** importato dal codice usato per l'esercizio precedente): uno basato sulla vera distribuzione dei digrammi $p_{X_1X_2}$ (denotato come C_1) e un altro sulla distribuzione prodotto $p_{X_1} \cdot p_{X_2}$ (denotato come C_2). I codici di Huffman rappresentano un metodo efficace di compressione dei dati che utilizza le probabilità delle entità da comprimere (in questo caso, i digrammi) per minimizzare la lunghezza media del codice.

3. Il codice calcola poi la lunghezza media dei due codici di Huffman, denotata come $L_p(C_1)$ e $L_p(C_2)$, e la differenza tra queste lunghezze, Δ. A seguito di questo, viene calcolata la divergenza di Kullback-Leibler, che rappresenta la distanza tra due distribuzioni di probabilità, e l'informazione mutua, che misura la quantità di informazione che si può ottenere su una variabile casuale osservando un'altra.

4. Infine, il codice legge un file di testo, lo converte in digrammi e lo comprime utilizzando entrambi i codici di Huffman. Viene quindi calcolata e stampata la lunghezza dei testi compressi.

I risultati ottenuti sono i seguenti:

```
Delta: 0.2729586841589988
Divergenza di Kullback-Leibler: 1.1838453528194122
Informazione mutua: 1.1838453528194122
Lunghezza del testo compresso con C1: 8714
Lunghezza del testo compresso con C2: 9222
```

Analizzando i risultati ottenuti, osserviamo che il valore di Δ è positivo, il che indica che, in media, il codice C_2 è più lungo del codice C_1. Questo suggerisce che assumere l'indipendenza tra le due lettere di un digramma (come fatto per il codice C_2) porta a una compressione meno efficiente rispetto a considerare la vera distribuzione dei digrammi (come fatto per il codice C_1).

Notiamo inoltre che la divergenza di Kullback-Leibler e l'informazione mutua hanno lo stesso valore. Questo è in linea con l'identità di collegamento, secondo la quale la differenza tra la lunghezza media di due codici di Huffman dovrebbe essere approssimativamente uguale alla divergenza di Kullback-Leibler tra le due distribuzioni di probabilità, ovvero $I(X_1; X_2) = D(p_{X_1X_2} || p_{X_1} \cdot p_{X_2})$.

Infine, confrontando le lunghezze dei testi compressi, confermiamo l'osservazione precedente che il codice C_2 produce una compressione meno efficiente: il testo

compresso con C_2 è infatti più lungo di quello compresso con C_1. Questo conferma la validità dell'uso della vera distribuzione di probabilità dei digrammi per la compressione dei dati.