POLITECNICO
DI TORINO

Dipartimento
di Scienze Matematiche
G.L. Lagrange

Esercitazioni

3

corso di Matematica per l'Intelligenza Artificiale

Francesco Della Santa

A.A. 2021/2022

Politecnico di Torino

Contenuti i

1. Ripasso Comandi da Terminale

2. Jupyter Lab e Jupyter Notebooks

3. Introduzione a Pandas

4. Gestione di Valori Categorici

Ripasso Comandi da Terminale

Terminali e Muoversi tra cartelle

In questo breve ripasso considereremo tre possibili tipi di shell per terminali:

- **bash** (Mac/Linux);
- **command prompt** (Windows). Abbreviato con **cmd**;
- **PowerShell** (Windows). Abbreviato con **PS**.

Il Comando "cd" (change directory)

Per muoversi tra le cartelle tramite terminale, si può utilizzare il comando cd come nel seguente esempio.

SO della shell	esempio di comando
Mac/Linux	`cd Cartella/Sottocartella/Sottosottocartella`
Windows	`cd Cartella\Sottocartella\Sottosottocartella`

Per "tornare indietro" di una cartella, digitare il comando: `cd ..`

Per indicare la cartella corrente, digitare il comando: `cd .`

Esempio: change directory

Supponiamo di avere il seguente insieme di cartelle e sotto cartelle.

- **Cartella**
  - *CartellaA*
    - SottocartellaA1
  - *CartellaB*
    - SottocartellaB1

Supponiamo di essere in "SottocartellaB1". Per entrare (shell bash) in "SottocartellaA1" possiamo usare il comando:

```
cd ../../CartellaA/SottocartellaA1
```

Contenuto di una Cartella

Per vedere il contenuto di una cartella, usare il seguente comando (da dentro la cartella).

- **bash/PS:** `ls`
- **cmd/PS:** `dir`

Creazione di Ambienti Virtuali Python da Terminale

Per creare un ambiente virtuale basato su Python3.x da terminale, basta lanciare il seguente comando (per più info guardare qui).

SO della shell	comando
Mac/Linux	`python3.x -m venv VEnvFolder_Path/nome_ambiente`
Windows	`PyInstall_Path\python -m venv VEnvFolder_Path\nome_ambiente`

ATTENZIONE

- *VEnvFolder_Path* è il percorso file della cartella che conterrà l'ambiente.
- *PyInstall_Path* è il percorso globale della **cartella di installazione** di Python (p.e.: `C:\Python3x`).
- **N.B.:** prima di "`-m`" si può scrivere anche solo "`python`" (eventualmente specificando versione) dopo opportune modifiche alle variabili di ambiente (default in Mac/Linux).

  Su Win. è possibile scegliere di aggiungere il path tra le variabili di ambiente al momento dell'installazione (attenzione a conflitti tra più versioni!).

(De)Attivazione di un Ambiente Virtuale da Terminale

Per attivare l'ambiente virtuale "`myenv`" all'interno di una cartella indicata dal percorso "`VEnvFolder_Path`", si digiti il seguente comando (per più info guardare questo link).

Shell	comando
bash	`source VEnvFolder_Path/myenv/bin/activate`
cmd	`VEnvFolder_Path\myenv\Scripts\activate.bat`
PS	`VEnvFolder_Path\myenv\Scripts\Activate.ps1`
cmd/PS[1]	`VEnvFolder_Path\myenv\bin\activate`

- ambiente attivato ⇒ a sinistra della riga di comando si vede tra parentesi il nome dell'ambiente (p.e.: "`(myenv)`");
- per disattivare un ambiente, digitare il comando: `deactivate`

[1] per alcune versioni recenti di python/Win.

Installare Pacchettie Moduli da Terminale

Una volta attivato l'ambiente virtuale desiderato, per installare/aggiornare/disinstallare un pacchetto o modulo python presente sul repository ufficiale PyPI, digitare:

- **Installare:** `pip install nome_modulo`
- **Installare versione specifica (p.e.: 0.1.2):** `pip install nome_modulo==0.1.2`
- **Aggiornare:** `pip install --upgrade nome_modulo`
- **Disinstallare:** `pip uninstall nome_modulo`

Per installare un elenco di pacchetti/moduli contenuti in un file *.txt* (p.e.: `requiremments.txt`), digitare:

```
pip install -r requirements.txt
```

N.B.: per maggiori dettagli sull'uso di pip, consultare la guida ufficiale.

Jupyter Lab e Jupyter Notebooks

Jupyter Lab

Jupyter Lab è un'interfaccia utente/editor per lavorare in python (edn altri linguaggi) del progetto jupyter; è basato su browser.

Per lavorare con jupyter lab, si deve installare il modulo python `jupyterlab` e poi lanciare da terminale il comando: `jupyter-lab`

Vantaggi Jupyter Lab

I vantaggi di Jupyter Lab riguardano principalmente l'utilizzo di jupyter notebooks. **In altri casi**, dove l'utilizzo di script è prevalente, si suggerisce l'uso di **altri editor** (p.e. Pycharm).

Un altro vantaggio è la possibilità di utilizzarlo per agevolare l'interfaccia utente per connessione a server remoti tramite browser.

Jupyter Notebooks

I **Jupyter Notebook** sono un particolare tipo di file (estensione `.ipynb`) che consiste in un documento dove testo e codice interattivo si alternano. In particolare, i notebooks integrano la shell interattiva del pacchetto `ipython` con strumenti di editor testuale, quali markdown.

Questi file sono suggeriti per:

- **Report/relazioni** che mostrino esempi di codice e risultati;
- **Visualizzazione**;
- **Didattica**.

Per il **calcolo scientifico**, o più in generale compiti computazionalemente onerosi, è sconsigliato il loro utilizzo a favore di script (file `.py`).

Jupyter Notebooks - Scorciatoie da Tastiera

Nota: dove sono indicati i tasti CTRL o ALT, si sostituiscando con CMD e OPTION per chi utilizza Mac.

- **SHIFT + Enter:** Esegue la cella corrente, seleziona la successiva;
- **CTRL + Enter:** Esegue le celle selzionate;
- **ALT + Enter:** Esegue la cella corrente, inserisce la successiva;
- **Esc:** Usce dalla modalità di inserimento;
- **Enter:** Entra in modalità inserimento;
- **DD:** Elimina cella (fuori modalità inserim.);
- **Z:** Annulla eliminazione cella (fuori modalità inserim.);
- **C:** Copia cella (fuori modalità inserim.);
- **V:** Incolla cella copiata in quella sottostante (fuori modalità inserim.);
- **M:** Cambia tipo cella in markdown (fuori modalità inserim.);
- **Y:** Cambia tipo cella in codice (fuori modalità inserim.);

Introduzione a Pandas

Pandas

Pandas è un modulo python (solitamente importato come **pd**) per la gestione e manipolazione di dati. I suoi oggetti principali sono:

- **Serie (pd.Series):** insieme ordinato mono-"dimensionali" di dati (di qualsiasi tipo) etichettati tramite *indici*.
- **DataFrame (pd.DataFrame):** insieme ordinato N-"dimensionale" di dati etichettati tramite combinazioni di $N-1$ indici ed una colonna (solitamente $N=2$). I dati in una colonna hanno tutti stesso tipo.

Scopo di Pandas

Importare/esportare, pulire, analizzare, trasformare dati conservati in sequenze etichettate (**Serie**) o tabelle (**DataFrame**).

Pandas - Inizializzazione Serie

```
pd.Series(data=None, index=None, dtype=None, name=None,
copy=False)
```

- **data:** oggetto di tipo *array mono-"dimensionale"*, iterabile, *dizionario* o scalare. Questi saranno i dati conservati nella serie. In caso di dizionario, le chiavi saranno usate come indici (ma `index` deve rimanere `None`).
- **index:** sequenza di valori "hashabili" (p.e. come le chiavi di un dizionario) con stessa lunghezza di `data`. Se non specificati, gli indici sono la sequenza di interi da 0 a `len(data)`.
- **dtype:** tipo di dato da assegnare ai dati (maggiori info qui). Se non specificato, il tipo di dato viene inferito dai dati stessi.
- **name:** nome assegnato all'oggetto Serie che si sta creando.
- **copy:** booleano. Crea una copia dei dati in input.

Documentazione ufficiale: qui

Pandas - Inizializzazione DataFrame i

```
pd.DataFrame(data=None, index=None, columns=None,
dtype=None, copy=False)
```

Nota: per semplicità ci limiteremo al caso di tabelle 2-"dimesionali".

- **data:** oggetto di tipo *array 2-"dimensionale"*, iterabile, *dizionario* o un altro DataFrame. Questi saranno i dati conservati nella DataFrame. In caso di dizionario, le chiavi saranno usate come *colonne* (ma `columns` deve rimanere `None`).
- **index:** sequenza di valori "hashabili" con stessa lunghezza di del primo asse di `data`. Se non specificati, gli indici sono la sequenza di interi da 0 a `len(data)`.
- **columns:** sequenza di valori "hashabili" con stessa lunghezza di del secondo asse di `data`. Se non specificati, gli indici sono la sequenza di interi da 0 a `len(data)`.

Pandas - Inizializzazione DataFrame ii

- **dtype:** Un unico tipo di dato da assegnare ai dati nel DataFrame. Se non assegnato, inferisce dai dati stessi (per ogni colonna).
- **copy:** booleano. Crea una copia dei dati in input.

Documentazione ufficiale: qui

Estrarre/Aggiungere Colonna

Sia dato un DataFrame `df`.

- `df[`*nome_colonna*`]` : restituisce una **serie** con dati i valori della colonna, indici gli indici del DataFrame e nome il nome/valore della colonna.
- `df[`*nuova_colonna*`] =` *sequenza_valori* : aggiunge la colonna *nuova_colonna* con i valori indicati.

DataFrame - Attributi i

Di seguito elenchiamo alcuni dei principali attributi dei **DataFrame**. Per una lista completa, seguire il link alla documentazione sopra indicato.

Nota: gli *attributi delle Serie* sono in larga parte gli stessi (o simili) a quelli dei DataFrame, pertanto non verranno introdotti e si rimanda alla documentazione sopra indicata.

- **at**[indice, colonna]**:** accede al singolo valore corrispondente alla coordinata (indice,colonna) indicata.
- **iat**[i, j]**:** accede al singolo valore corrispondente alla coordinata (i-esimo indice, j-esima colonna). **RICORDA:** in python l'indicizzazione di valori parte da 0.
- **index:** oggetto che definisce gli indici.
- **columns:** oggetto che definisce le colonne.
- **axes:** lista contenente l'attributo *index* e l'attributo *columns*.

DataFrame - Attributi ii

- **loc**[lista indici, lista colonne]**:** restituisce un sotto-DataFrame identificato dalle liste di indici e colonne specificate. **Alternativa:** le liste di indici e colonne possono essere indicate anche in termini booleani (p.e. *tutti gli indici maggiori di un certo numero*). **Nota:** Gli operatori booleani per *loc* in pandas sono &, | e $\sim$ e sostituiscono quelli standard di python *and*, *or* e *not* . **Nota:** il simbolo ":" indica l'elenco di tutti gli/le indici/colonne.
- **iloc**[lista indici per ind., lista indici per col.]**:** come *loc* ma gli indici e le colonne sono indicati tramite la loro indicizzazione nel rispettivo elenco di indici/colonne (similmente a *iat*).
- **shape:** tupla contenente la lunghezza degli indici e delle colonne.
- **ndim:** "dimensioni" del DataFrame, cioè la lunghezza della tupla restituita dall'attributo *shape*. In tabelle 2-"dimensionali" è quindi 2.

DataFrame - Attributi iii

- **size:** numero totale di elementi del DataFrame. Uguale quindi al prodotto del valori nella tupla restituita dall'attributo *shape*.
- **values:** rappresentazione del DataFrame come array di numpy.

DataFrame - Metodi i

Di seguito elenchiamo alcuni dei principali metodi dei **DataFrame** (**DF**). Per una lista completa e maggiori informazioni sui metodi elencati, seguire il link alla documentazione precedentemente indicato.

Nota: i *metodi delle Serie* sono in parte gli stessi (o simili) a quelli dei DataFrame, pertanto non verranno introdotti e si rimanda alla documentazione sopra indicata.

Metodi di Esplorazione:

- `head(n=5)`. Stampa a video le prime `n` righe del DF.
- `tail(n=5)`. Stampa a video le ultime `n` righe del DF.
- `info()`. Mostra le informazioni salienti del DF.
- `nunique(axis=0, dropna=True)`. Con i valori unici presenti nel DF rispetto agli/alle indici/colonne (se `axis`=0/1). `dropna` indica se non includere nel conteggio anche i valori mancanti.

DataFrame - Metodi ii

- `isna()/isnull()`. Restituische un DF booleano con `True` per i valori che sono mancanti (NaN).
- `count(axis=0)`. Calcola i valori non-mancanti ($\neq$NaN o simili).
  - **axis:** asse (0 =indici, 1 =colonne) rispetto alla quale svolgere il conteggio.
- `value_counts(subset=None, normalize=False, sort=True, ascending=False)`. Restituisce l'elenco di righe del DF e quante volte si ripetono al suo interno.
  - **subset:** elenco di colonne da considerare (tutte di default).
  - **normalize:** normalizzare il conteggio delle ripetizioni.
  - **sort:** ordinare le righe in base al numero di ripetizioni.
  - **ascending:** ordinamento (se presente) ascendente o no.

Analisi Statistica di Base e Operazioni su Valori:

DataFrame - Metodi iii

- `describe(percentiles=None)`. Restituisce un DataFrame che le statistiche di base dei valori, rispetto le colonne.
  - **percentiles:** lista di valori tra 0 e 1 rappresentanti i percentili. Di default: `[0.25, 0.50, 0.75]`.
- `mean(axis=None, skipna=True)`. Calcola la media dei valori nelle righe/colonne.
  - **axis:** asse (0 =indici, 1 =colonne) rispetto alla quale svolgere i calcoli.
  - **skipna:** booleano. Saltare o meno dal conteggio per la media le celle con valori mancanti (NaN o simili).
- `std, var, sum, min, max, etc.`. Funzionamento equivalente a `mean`. Per maggiri informazioni, controllare la documentazione.
- `corr` e `cov`: calcola la matrice di correlazione/covarianza (esclude i NaN dai calcoli).

DataFrame - Metodi iv

- `sample(n, replace=False, random_state=None, axis=0)`. Campiona uniformemente in maniera casuale righe/colonne di valori dal dataframe
  - **n:** numero di campioni da estrarre.
  - **replace:** booleano. Se `True`, la stessa riga/colonna può essere estratta più volte.
  - **random_state:** seed randomico.
  - **axis:** 0 per campionare righe, 1 per campionare colonne.

Metodi di Trasformazioni/Alterazione:

- `append(other`. Aggiunge "sotto" il DataFrame `other` al DF originale. Se `other` ha altre colonne, aggiunge anche quelle (vedi la documentazione per più info).
- `copy()`. Crea una copia del DF.

DataFrame - Metodi v

- `drop(labels, axis=0, index=None, columns=None, inplace=False)`. Elimina righe/colonne dal DF.
  - **labels:** lista di nomi degli/delle indici/colonne da rimuovere.
  - **axis:** asse rispetto alla quale controllare gli/le indici/colonne da rimuovere (default: righe).
  - **index** e **columns:** si possono dichiarare qui le *labels* senza specificare *axis*.
  - **inplace:** booleano. Se True, il nuovo DF viene sovrascritto all'originale. Altrimenti è un DF separato, restituito in output.
- `drop_duplicates(subset=None, inplace=False)`. Rimuove duplicati di righe.
  - **subset:** come in *value_counts*.
  - **inplace:** come *drop*.
- `dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)`. Elimina righe/colonne dal DF in base alla mancanza di valori (NaN)

DataFrame - Metodi vi

  - **axis**, **subset** e **inplace** come in *drop* e *drop_duplicates*.
  - **how** *any* oppure *all*. Nel primo caso rimuove la riga se presenta almeno un NaN, nel secondo se tutti i valori sono NaN.
  - **thresh:** se assegnato un valore intero, rimuove solo le righe con un numero di NaN maggiore o ugual a quel valore.
- `fillna(value=None, axis=None, inplace=False)`. Riempie i valori NaN del DF.
  - **axis** e **inplace** come in metodi precedenti.
  - **value:** valore con cui riempire i NaN. In alternativa si può passare un dizionario/serie/DataFrame per indicare come sostituire i valori in base a righe/colonne.
- `interpolate(method='linear', axis=0, inplace=False)`. Riempie i valori mancanti tramite interpolazione. Per maggior dettagli, guardare la documentazione ufficiale.

DataFrame - Metodi vii

- `insert(loc, column, value, allow_duplicates=False)`. Inserisce una colonna `column` come nuova (`loc`+1)-esima colonna nel DF, avente come valori la sequenza `value`.
- `rename(index=None, columns=None, inplace=False)`. Cambia i nomi di indici/colonne rispetto ai dizionari passati in input (vecchi nomi come chiavi, nuovi nomi come valori).
- `reset_index(drop=False)`. Resetta gli indici ai valori di default da 0 al numero di righe meno 1. Se `drop` è `False`, i vecchi indici diventano una colonna, altrimenti vengono rimossi.
- `sort_values(by, axis=0, ascending=True, inplace=False)`. Ordina le righe del DF rispetto ai valori di una riga/colonna.
  - **by:** nome (o lista di nomi) di indice/colonna rispetto alla quale ordinare.
  - **axis:** 0/1 a seconda se *by* è il nome di una/un colonna/indice.
  - **ascending:** booleano. Ordinamento ascendente o no.

DataFrame - Metodi viii

  - **inplace:** vedi metodi precedenti.

- `replace(to_replace, value=None, inplace=False)`. Permette di sostituire i valori indicati in `to_replace` con quelli indicati in `value`. l'utilizzo più semplice è tramite singoli valori per i due argomenti o liste della stessa lunghezza per sostituzioni "multiple". In alternativa si può non definire `value` e definire `to_replace` come un dizionario con chiavi i vecchi valori e valori i nuovi valori. Ulteriore alternativa è usare dizionari per `to_replace` e `value` come nel caso di liste ma con le chiavi che specificano le colonne. Per maggiori dettagli, guardare la documentazione ufficiale.

Metodi di Esportazione:

- `to_csv(path=None, sep=',', columns=None, index=True, index_label=None, encoding=None, decimal='.')`.

DataFrame - Metodi ix

- **path:** percorso più nome del file (meglio se anche con estensione). Esempio: *Cartella/tabella.csv*.
- **sep:** carattere di separazione delle colonne.
- **columns:** colonne da esportare (tutte di default).
- **index:** booleano. Esportare o meno una colonna con i valori degli indici. **index_label:** eventuale nome da date alla colonna di indici (se esportati).
- **encoding:** tipo di codifica dei dati (default utf-8). Per esempio, cambiarlo se sono presenti caratteri speciali in stringhe o nomi colonne.
- **decimal:** carattere utilizzato per indicare il passaggio ai decimali.

- `to_excel(excel_writer, sheet_name='Sheet1', columns=None, index=True, index_label=None, startrow=0, startcol=0)`. Per un **corretto funzionamento**, installare il anche il pacchetto `xlrd`.

- **excel_writer:** percorso più nome del file, eventualmente anche già esistente.
- **sheet_name:** nome del foglio nel file excel dove esportare il dataframe.
- **startrow/column:** riga/colonna del foglio dove sarà posizionato l'angolo in alto a sinistra del DataFrame.
- *I restanti argomenti di input sono uguali al caso to_csv*.

- `to_pickle(path)`.
  - **path:** percorso più nome del file (meglio se anche con estensione). Esempio: *Cartella/tabella.pkl*.

Metodi di Visualizzazione:

(**Nota:** richiede l'importazione di *matplotlib*)

DataFrame - Metodi xi

- `plot(*args, **kwargs)`. Permette di eseguire vary tipi di plot specificandolo con l'argomento *kind* (di default, *'line'*). Guaradre la documentazione completa qui.
- `plot(x, y)`. Caso di uso di plot come *line*.
  - **x:** nome colonna per le ascisse.
  - **y:** nome/lista di nomi di colonna/e per le ordinate.
- `plot.scatter(x, y, s=None, c=None)`. Scatterplot fatto rispetto ai valori nelle colonne indicate per gli input `x` e `y`. Gli input `s`,`c` sono rispettivamente per la dimensione dei punti ed il colore. Se si assegna il nome di una colonna a `c`, i colori dipenderanno dai valori nella colonna.
- `plot.bar(x=None, y=None)`. Barplot fatto rispetto ai valori nelle colonne indicate per gli input `x` e `y`. Se non specificato, `x` sono gli indici del DF. Se `y` non è specificato, tutte le colonne numeriche sono considerate.

DataFrame - Metodi xii

- `boxplot(column=None)`. Boxplot per ogni colonna indicata nella lista assegnata all'input `column` (tutte se `None`).
- `plot.density()`. Plot di una stima della funzione di densità di probabilità di ogni colonna nel DF. Se si vuole limitare a singole colonne, eseguire il metodo rispetto ad esse soltanto.

Pandas - Importare DataFrame da File i

Per **importare** un file nell'ambiente di lavoro come pandas DataFrame, si possono usare i seguenti comandi, a seconda del **tipo di file**. Per uleriori tipi di file da importare, consultare la documentazione.

- `pd.read_csv(filepath, sep=',', header=0, index_col=None, usecols=None, thousands=None, decimal='.')`. Carica un file csv come DataFrame.
  - **filepath:** stringa indicante il percorso fino al file da caricare. può essere anche un generico URL.
  - **sep:** carattere da interpretare come separatore delle colonne.
  - **header:** indice della riga da usare per leggere i nomi delle colonne. Se `None`, significa che i nomi delle colonne non sono indicati nel file csv.
  - **index_col:** colonna nel csv da leggere come colonna degli indici del DF.
  - **usecols:** nome delle colonne da importare.

Pandas - Importare DataFrame da File ii

  - **thousands** e **decimals**: caratteri da interpretare come separatori delle migliaia e dei decimali nel csv.

- `pd.read_excel(filepath, sheet_name=0, header=0, index_col=None, usecols=None, thousands=None`. Carica un DataFrame da un foglio di un file excel. Per un **corretto funzionamento**, installare il anche il pacchetto `xlrd`.
  - *Gli input omonimi con quelli di read_csv sono la trasposizione per la lettura di file excel.*
  - **sheet_name:** stringa o intero per indicare il foglio contenente i dati da caricare. Se un numero intero, si riferisce all'indice (0-based) del foglio nel file.
- `pd.read_pickle(filepath)`. Carica un DataFrame precedentemente salvato come pickle.

Operazioni tra DataFrame

Le **classiche operazioni** di somma, sottrazione, ecc. possono essere fatte (componente per componente) con i classici operatori.

Per la **concatenazione** "in orizzontale" di DF, si può usare la funzione

`pd.concat(`*`lista_DF`*`, axis=1)`

con l'opzione *axis* sopra indicata.

In questo modo, i vari DF nella lista di input vengono "affiancati", allineando le righe rispetto a medesimi valori degli indici.

Per maggiori usi della funzione `concat`, consultare la documentazione.

Gestione di Valori Categorici

Valori Categorici

Spesso capita di dover lavorare con feature o target **categorici** e non numerici, p.e.:

Features					Target
Sesso	Nazionalità	Città	Età	Professione	Modello Macchina
M	IT	Torino	50	Medico	Berlina
⋮	⋮	⋮	⋮	⋮	⋮

Conversione in Numeri

In questi casi le varie **categorie** dovranno essere "tradotte" in **valori numerici**.

N.B.: feature e target categorici, possono essere rappresentati anche da valori numerici e non necessariamente da stringhe (p.e. i codici postali o i prefissi telefinici nazionali).

Valori Categorici - Pandas

Per **convertire** il tipo di dati di un DataFrame o di una Serie (e quindi di una colonna di un DF) in pandas, si può utilizzare il metodo `astype`.

`astype(dtype)`.

- **dtype:** tipo di dato (python o numpy.dtype) da assegnare a tutti i valori contenuti nel DF o nella serie. Il tipo di dato può essere espresso anche tramite stringa.

  Nel caso di un DF, `data` può essere un **dizionario** con chiavi i nomi delle colonne a cui cambiare il tipo di dato e con valori i nuovi tipi di dato da assegnare (p.e. {`col_1: 'category', col_5: 'int64'`}).

Dati Categorici: in pandas, per convertire in dati categorici si usi la stringa `'category'`. Il tipo di dato genrico `'object'` viene anche esso spesso interpretato come categorico (ma una conversione è sempre suggerita).

Per **estrarre** l'elenco delle categorie in una colonna categorica di un DF (o una Serie) si usi il comando: `df[`*`colonna`*`].cat.categories`.

Label Encoding

La rappresentazione più banale delle **classi** $c_1, \ldots c_n$ è quella tramite i numeri naturali $0, \ldots, n-1$ secondo una permutazione casuale σ su $\{0, \ldots, n-1\}$ ("label encoding"), cioè: $i \in \{0, \ldots, n-1\}$ rappresenta c_j se $i = \sigma(j)$.

PRO: conversione molto semplice.

CONTRO: implicitamente si creano delle **indesiderate** relazioni d'ordine $(<, \leq, >, \geq)$ tra le classi $c_1, \ldots, c_n$.

N.B.: queste relazioni d'ordine potrebbero **ingannare** l'algoritmo di Machine Learning (ML) nell'apprendimento, facendogli osservare delle cose che in realtà non esistono.

Eccezioni

Ovviamente, esistono anche delle eccezioni. Per esempio, se le categorie rappresentano le fasce d'età "0-25", "26-50", "51-75" e "76+", allora è legittimo valutare se indicarle rispettivamente con numeri interi successivi, p.e. $1, 2, 3, 4$.

Scikit-Learn - LabelEncoder

`LabelEncoder` è una **classe** di `sklearn.preprocessing` per trasformare combinazioni (anche con ripetizione) di n **oggetti** in un'equivalente combinazione dei **numeri** naturali da 0 a $n-1$.

ATTENZIONE: l'assegnazione dei numeri $\{0, \ldots, n-1\}$ NON è casuale rispetto alle classi ma procede in maniera ordinata (creascente o alfabetica) $\Rightarrow$ una banale alternativa può essere l'uso del metodo `replace` se si sta lavorando con DataFrames.

Documentazione ufficiale e completa: qui.

Esempio importazione LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
```

Scikit-Learn - I Metodi del LabelEncoder

I **metodi** di un oggetto della classe `LabelEncoder` sono i seguenti:

- `fit(y)`. Data `y` una **combinazione** contenente tutte le possibili n classi (o anche solamente la lista delle n classi), questo metodo genera la **relazione classe-numero** e la "memorizza" dentro l'oggetto (nello specifico nell'attributo `classes_`, come np.array).
- `transform(y)`. Data `y` una qualsiasi **combinazione** di classi, il metodo restituisce la combinazione corrispondente di numeri naturali, secondo quanto memorizzato con l'esecuzione del metodo `fit`.
- `inverse_transform(z)`. Data `y` una qualsiasi **combinazione** di numeri naturali, il metodo restituisce la combinazione corrispondente di classi, secondo quanto memorizzato con l'esecuzione del metodo `fit`.
- `fit_transform(y)`. Esegue, in una sola operazione, prima il metodo `fit` e poi il metodo `transform` rispetto allo stesso input `y`.

One-hot Encoding

L'approccio maggiormente consigliato per la gestione di valori categorici è il "one-hot encoding". In poche parole, le classi

$$c_1, \ldots, c_n$$

vengono rappresentate rispettivamente come i **vettori della base canonica** di $\mathbb{R}^n$, cioè

$$\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$$

PRO: in questo modo non instauriamo alcuna falsa relazione d'ordine tra le classi.

CONTRO: le dimensioni del problema crescono velocemente.

One-hot Encoding

Riprendendo l'esempio precedente:

Città
Torino
Parigi
⋮
Torino
Roma

$\dashrightarrow$

Torino	Parigi	$\cdots$	Roma
1	0	$\cdots$	0
0	**1**	$\cdots$	0
⋮	⋮	⋮	⋮
1	0	$\cdots$	0
0	0	$\cdots$	**1**

Per eseguire questa operazione in **pandas**, si utilizza la funzione `get_dummies`.

La funzione get_dummies i

La funzione get_dummies restituisce un DF in forma one-hot encoding dai dati forniti in input.

```
pd.get_dummies(data, prefix=None, prefix_sep='_',
dummy_na=False, columns=None, sparse=False,
drop_first=False, dtype=None)
```

- **data:** dati di tipo array, Series o DataFrame da convertire con one-hot encoding.
- **prefix:** stringa, lista di stringhe o dizionario di stringhe. Indica i prefissi delle colonne del DF di output. Se `None` e `data` è un DF, allora il prefisso è il nome della colonna originaria.
- **prefix_sep:** stringa che separa il prefisso dal valore in `data` per il nuovo nome della colonna.

La funzione get_dummies ii

- **dummy_na:** booleano. Se `True`, aggiunge una colonna per i dati mancanti (NaN).
- **columns:** se `data` è un DF, è una lista che indica quali colonne "tradurre" con il one-hot encoding. Se `None`, tutte le colonne con di tipologia `object` o `category` vengono convertite.
- **sparse:** booleano. Indica se i valori del dataframe di output devono essere memorizzati come una matrice sparsa o no.
- **dtype:** dtype del DF di output, default np.uint8.