Python per la data science (autore: Vittorio Albertoni)

Premessa

Nelle varie fasi del metodo scientifico ne troviamo due che hanno a che fare con dati: quella delle osservazioni iniziali che supportano la formulazione di una ipotesi e quella dell'analisi dei risultati della sperimentazione dell'ipotesi in modo da decidere se l'ipotesi stessa sia da accettare o da scartare.

Tradizionalmente, sia i dati delle osservazioni iniziali sia i dati della sperimentazione sono disponibili in quantità limitata, in quelle entità che gli statistici chiamano campioni.

E la statistica ci offre strumenti per valutare quanto sia attendibile fondare congetture e derivare conoscenza da quantità limitate di dati e il sotto-modulo `stats` di SciPy, che ho presentato in altra occasione, ci dà modo di fruire di strumenti di questo tipo[1].

Da inizio secolo, da quando il progresso tecnologico e la digitalizzazione hanno reso facile e relativamente poco costoso raccogliere ed archiviare grandi moli di dati, si è consolidata una nuova disciplina, ormai ufficialmente battezzata come *data science*, che non ha tanto l'obiettivo di trarre dall'analisi di dati all'uopo raccolti conferme a ipotesi scientifiche quanto quello di trarre da dati comunque disponibili indicazioni di varia natura su azioni e comportamenti attinenti le materie cui i dati si riferiscono.

La statistica tradizionale opera prevalentemente al servizio di laboratori di ricerca (sociologica, medica, scientifica in genere) a supporto della formulazione di ipotesi scientifiche, la data science opera prevalentemente al servizio di aziende che intendano trarre dall'esame dei dati indicazioni sulla gestione del business.

Gli strumenti di cui si avvalgono le due discipline sono molto spesso gli stessi ma, in aggiunta a tutto ciò che troviamo nella statistica tradizionale, la data science ha bisogno di
. strumenti per la manipolazione e il trattamento di grandi moli di dati,
. strumenti per il trattamento di dati qualitativi oltre che quantitativi,
. strumenti per rendere possibile trarre conoscenza dai dati anche in mancanza di algoritmi espliciti.

Alla prima esigenza Python risponde con il modulo **Pandas**, che offre strutture dati e operazioni per manipolare tabelle numeriche e serie di dati.

A fronte della seconda esigenza abbiamo il modulo **NLTK**, suite di strumenti per l'analisi simbolica e statistica nel campo dell'elaborazione del linguaggio naturale (parole anziché numeri).

A fronte della terza esigenza abbiamo il modulo **Scikit-learn**, libreria di apprendimento automatico.

[1] Si veda, in proposito, l'allegato «python_calcolo_scientifico» al mio articolo «Python e il calcolo scientifico» del settembre sul mio blog *www.vittal.it*.

Indice

1 Pandas

Pandas sta per Panel Data.

All'indirizzo *https://pandas.pydata.org/* troviamo la documentazione ufficiale in lingua inglese e cliccando sul pulsante INSTALL PANDAS NOW ci vengono descritti vari modi per installare Pandas, primo fra tutti quello che utilizza Anaconda[2].

In questa sede ci accontentiamo di farlo con pip.

Quanto alla documentazione, ciò che vedremo qui si riferisce ai primi rudimenti: il file PDF che troviamo sul sito all'indirizzo prima citato si sviluppa su 3509 pagine e questo la dice lunga circa ciò che si possa fare con Pandas.

Per usare Pandas dobbiamo importarlo nello script e lo possiamo fare con

```
import pandas as pd
```

utilizzando la tecnica dell'importazione con alias abbreviativo che ben si adatta a pacchetti molto ricchi di funzioni.

Con Pandas possiamo creare due oggetti che corrispondono a due strutture di dati: la serie e il frame.

1.1 Serie

La serie è un elenco etichettato di dati di qualsiasi tipo. Le etichette ne costituiscono l'indice.

La serie si costruisce con la funzione `Series` di Pandas, con la sintassi

`Series(<dati>, index =` <lista>`)`

dove <dati> può essere una lista, un ndarray, uno scalare o un dizionario e <lista> è una lista.

Se <dati> è un dizionario si deve omettere l'argomento index in quanto le etichette indice vengono assunte dal dizionario.

Se <dati> non è un dizionario e si omette l'argomento index le etichette vengono assegnate come numeri interi partendo da zero.

Esempi:
Avendo importato Pandas con `import pandas as pd`,

```
pd.Series([12, 45, 'a'], index = ['a', 'b', 'c'])
```

crea la serie

```
a     12
b     45
c     a
```

```
pd.Series({'x':45, 'y':15, 'z':22, 'w':12})
```

crea la serie:

```
x     45
y     15
z     22
w     12
```

```
pd.Series([15, 22, 13])
```

crea la serie

```
0     15
1     22
2     13
```

```
pd.Series([15, 22, 13], index = [1, 2, 3])
```

crea la serie

```
1     15
2     22
3     13
```

```
pd.Series(8, index = [10, 20, 30])
```

crea la serie

```
10     8
20     8
30     8
```

[2]Chi voglia approfondire questa strada maestra può utilmente leggere l'allegato in formato PDF «python_anaconda» al mio articolo «Software libero per data scientists» dell'aprile 2019 sul mio blog all'indirizzo *www.vittal.it*.

Con la funzione di Python di base `len` si ottiene il numero di elementi della serie con la sintassi
`len(<serie>)`.

Ad un elemento della serie si accede attraverso l'indice con la sintassi
`<serie>[<indice>]`.

Il valore di un elemento può essere modificato accedendo ad esso ed assegnandogli un altro valore.

Si può aggiungere un elemento a una serie assegnandone il valore ad una nuova posizione di indice.

Esempi:
Costruita la serie s = pd.Series([12,8,22,15], index = [1,2,3,4])

`len(s)` ritorna `4`
`s[3]` ritorna `22`
`s[2] = 16` sostituisce nel secondo elemento il valore `16` al valore `8`
`s[5] = 18` aggiunge un quinto elemento con indice `5` e valore `18`.

Per togliere un elemento da una serie abbiamo la funzione
`drop([<indice_elemento>])`
che toglie l'elemento indicato senza tuttavia modificare la serie originaria.

Data la serie s di prima, comprensiva del quinto elemento aggiunto nell'esempio,
con `s.drop([5])` la leggiamo ignorando il quinto elemento,
con `s = s.drop([5])` la modifichiamo togliendo il quinto elemento.

Per certi versi, la serie di Pandas è simile a un ndarray di Numpy e, data la serie `s` degli esempi, se in una IDLE di Python scriviamo s. apriamo il lungo elenco delle funzioni dell'oggetto Serie di Pandas e vi troviamo le funzioni `min`, `max`, `mean`, `std`, `sum`, `prod`, `cumsum` che sono comuni agli ndarray.

Differente la funzione `sort()` che qui si presenta nella doppia veste di
`sort_index()` per eseguire l'ordinamento in base all'indice e
`sort_values()` per eseguire l'ordinamento in base al valore,
in ogni caso solo in lettura senza modificare la serie originaria.

Per quanto riguarda le operazioni matematiche con gli operatori `+ - * / **`, mentre nel caso degli ndarray di Numpy esse sono possibili solo tra array delle stesse dimensioni, nel caso delle serie di Pandas esse sono sempre possibili, fermo restando che ove non vi sia possibilità di instaurare l'operazione membro a membro a causa della differente dimensione delle serie il risultato sarà NaN (Not a Number).

Ovviamente è possibile moltiplicare una serie Pandas per uno scalare, ottenendo una nuova serie formata dai membri de quella di partenza moltiplicati per lo scalare.

Esempi:
Date le serie
```
a = pd.Series((1,2,4))
b = pd.Series((3,1))
```
`a + b` ritorna
```
0    4.0
1    3.0
2    NaN
```
`a ** b` ritorna
```
0    1.0
1    2.0
2    NaN
```
`a * 5` ritorna
```
0    5
1    10
2    20
```

Dal momento che Pandas nasce per trattare grandi moli di dati (i così detti big data), può accadere che una sua serie contenga centinaia o migliaia di elementi.

Per ispezionarne semplicemente l'inizio o la fine evitando di stampare tutta la serie abbiamo a disposizione le funzioni

`head()` che ci fa vedere i primi cinque elementi,
`tail()` che ci fa vedere gli ultimi cinque elementi.
Come avviene per le altre funzioni, esse si richiamano facendole precedere dal nome della serie e da un punto.

1.2 Frame

Il frame è una tabella con righe e colonne etichettate.

Il frame si costruisce con la funzione `DataFrame` di Pandas, con la sintassi
`DataFrame(<dati>, columns = <lista>, index = <lista>)`
dove
<dati> può essere una lista di tuple, un dizionario o un ndarray bidimensionale.
<lista> è una lista con la quale si danno nomi alle colonne (`columns`) e alle righe (`index`) della tabella.

Esempi:
Avendo importato Pandas con `import pandas as pd`,
possiamo ottenere il seguente frame

	altezza	peso
Vittorio	1,81	92
Luigi	1.78	77

con
`pd.DataFrame([(1.81,92),(1.78,77)], columns=['altezza','peso'], index=['Vittorio','Luigi'])`
oppure con
`pd.DataFrame({'altezza':(1.81,1.78),'peso':(92,77)}, index=['Vittorio','Luigi'])`
oppure, disponendo di un ndarray

$$a = \begin{matrix} 1.81 & 92 \\ 1.78 & 77 \end{matrix}$$

con
`pd.DataFrame(a,columns=['altezza','peso'], index=['Vittorio','Luigi'])`

Se non si indicano gli argomenti `columns` e/o `index`, essi vengono automaticamente inseriti con numeri interi a partire da zero.

Con la sintassi
<frame>[<nome_colonna>]
si accede ad una singola colonna del frame, la quale è, in tutto e per tutto, una serie di Pandas ed è dotata delle relative funzioni.

Esempi:
Posto che il frame degli esempi precedenti sia in una variabile nominata df,
`df['altezza'].mean()` ritorna `1.795`
`df['peso'].max()` ritorna `92`

Per accedere ad un singolo dato del frame abbiamo a disposizione le funzioni
`loc`[<nome_riga>][<nome_colonna>] seleziona per etichetta,
`iloc`[<numero_riga>][<numero_o_nome_colonna>] seleziona per posizione,
sapendo che righe e colonne sono intrinsecamente numerate partendo da zero.

Esempi:
Sempre posto che il frame degli esempi precedenti sia in una variabile nominata df,
`df.loc['Luigi']['peso']` ritorna `77`
`df.iloc[0][0]` ritorna `1.81`
`df.iloc[1]['altezza']` ritorna `1.78`

Con la funzione loc possiamo aggiungere righe al frame con la sintassi
`loc`[<numero_o_nome_riga>] = <lista_valori>
od anche colonne con la sintassi
`loc`[:,<numero_o_nome_colonna>] = <lista_valori>

`df.loc['Giuseppe']=[1.90,88]` aggiunge l'altezza e il peso di Giuseppe al nostro dataframe `df`
`df.loc[:,'massa']=df['peso']/(df['altezza']*df['altezza'])` aggiunge al dataframe df la nuova colonna contenente la massa corporea calcolata secondo la formula peso/altezza^2

Ora il frame df è così

	altezza	peso	massa
Vittorio	1.81	92	28.082171
Luigi	1.78	77	24.302487
Giuseppe	1.90	88	24.376731

Possiamo leggere il frame cancellando righe e colonne con la funzione `drop()` e la seguente sintassi
`drop([`<numero_o_nome_riga>`])` per cancellare una riga,
`drop([`<numero_o_nome_colonna>`], axis = 1)` per cancellare una colonna.
Entrambe le funzioni operano solo in lettura e ciò che ritornano può essere inserito in un'altra variabile o, per modificare il frame di partenza, nella stessa variabile che lo conteneva.

Possiamo anche leggere il frame filtrando i dati con la sintassi
<frame>`[`<condizione>`]`.

Con riferimento al nostro frame df possiamo estrarre i sovrappeso (massa corporea maggiore di 28) con
`df[df['massa'] > 28]`

Possiamo calcolare i valori medi di chi è più alto di 1.80 con
`df[df['altezza'] > 1.8].mean()`

Molto spesso i frame vengono alimentati da tabelle esterne contenute in fogli di calcolo, database, ecc.

Pandas contiene molte funzioni ad hoc, tutte con la radice `read_`, ma, posto che da qualsiasi fonte di dati è possibile generare tabelle in formato `.csv` (comma separated value), da principianti ci possiamo accontentare della funzione
`read_csv()`
che ha come primo argomento obbligato una stringa con il percorso al file `.csv` contenente i dati e, come successivo argomento opzionale, separato da virgola, `delimiter = '`<simbolo_separatore>`'` utile quando il separatore non sia una virgola (,) ma un punto e virgola (;) o una barra (|), ecc.
La prima riga del file viene automaticamente utilizzata per l'intestazione delle colonne.
Se si vuole evitare questo occorre usare l'argomento opzionale `header = None`. In tal caso le colonne vengono numerate con interi a partire da zero e si possono rinominare con <frame>`.columns =` <lista>.
Le righe vengono numerate con interi a partire da zero. Se si vuole utilizzare una colonna per indicizzare le righe lo si può fare con l'argomento opzionale `index_col =` <numero_o_nome_colonna>.

2 NLTK

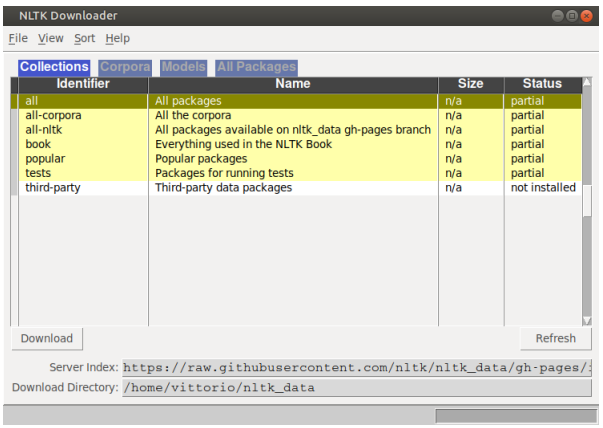
NLTK sta per Natural Language ToolKit ed è una libreria che permette l'analisi di testi.

Troviamo tutto su NLTK all'indirizzo *http://www.nltk.org/*, in particolare troviamo il manuale completo all'indirizzo *https://www.nltk.org/book/*.

Come al solito installiamo NLTK con pip.

Per usarlo lo importiamo con la sintassi
`import nltk as nl`.

Quello che installiamo con pip è il modulo base e può essere arricchito richiamandone la funzione download con il seguente comando nella shell di Python o nell'IDLE
`nl.download()`
che apre la seguente finestra



Si tratta dell'accesso ad un repository che contiene vari tools e testi predisposti per esercitarsi con NLTK.

Per ciò che faremo qui non utilizziamo questo downloader ma, per ciò che dobbiamo caricare, ci basta usare il comando `nl.download()` mettendo tra le parentesi la stringa con il nome di ciò che ci serve.

2.1 Analisi del testo

Per analizzare un testo con NLTK dobbiamo predisporlo con la funzione `Text()`.

Un testo analizzabile si crea passando a questa funzione, come argomento, non già una semplice stringa ma un testo sotto forma di lista di token, cioè di entità linguistiche separate (parole o frasi).

Abbiamo a disposizione due funzioni per ottenere questo
`word_tokenize()` trasforma un testo in una lista di parole,
`sent_tokenize()` trasforma un testo in una lista di frasi.

Prima di usare con profitto queste funzioni dobbiamo però installare un package aggiuntivo al nostro NLTK di base e lo facciamo, previa importazione di NLTK in una shell Python o nella IDLE con
`import nltk as nl`,
proseguendo con
`nl.download('punkt')`.

Da qui in poi il nostro NLTK di base è arricchito con punkt e potremo fare le nostre tokenizzazioni come si deve.

Seguono un paio di esempi.
`nl.word_tokenize('Oggi è una bella giornata.  Ieri era peggio')` ritorna
`['Oggi', 'è', 'una', 'bella', 'giornata', '.', 'Ieri', 'era', 'peggio']`,
`nl.sent_tokenize('Oggi è una bella giornata.  Ieri era peggio')` ritorna
`['Oggi è una bella giornata.', 'Ieri era peggio']`.

Un testo analizzabile con NLTK partendo da una stringa di testo si crea con `word_tokenize`, così:
`testo = nl.Text(nl.word_tokenize(<stringa>)`.

L'oggetto, qui denominato testo, così creato possiede i metodi che ci consentono di analizzarlo e ne troviamo l'elenco scrivendo, nella IDLE, il nome dell'oggetto seguito da un punto.

Cito i principali:

- `concordance(<parola>)` ricerca una parola nel testo e ne mostra il contesto,
- `collocations()` indica le sequenze di parole che nel testo appaiono molto spesso assieme,
- `dispersion_plot(<lista_di_parole>)` mostra graficamente la distribuzione di parole nel testo.

Inoltre possiamo utilizzare i normali metodi del linguaggio Python applicabili alle liste (`len` per ottenere il numero dei token, ricerca di parole utilizzando indici, ecc.).

Infine NLTK ha una funzione che ci fornisce la distribuzione di frequenza delle parole presenti nel testo, con la sintassi

`FreqDist(<nome_testo>)`
potendo indicare con `n` la quantità da mostrare con la sintassi
`FreqDist(<nome_testo>).most_common(n)`.

Esempio:
Ho un file di testo che contiene l'Addio di Lucia ai monti dal Capitolo VIII dei Promessi Sposi e lo voglio esaminare.
Innanzi tutto lo apro con
`f = open('/home/vittorio/Documenti/addio.txt')`
poi dopo `import nltk as nl`, lo rendo adatto all'analisi con NLTK con
`testo = nl.Text(nl.word_tokenize(f.read().lower()))`
(il richiamo della funzione `lower()` serve se si vuole che tutte le parole inizino con la minuscola in quanto alcune funzioni, come la `dispersion_plot()`, distinguono tra maiuscole e minuscole e possono fornire risultati parziali se non se ne tiene conto).
Ora cominciamo l'analisi.

`testo.concordance('addio')`
ritorna la seguente schermata

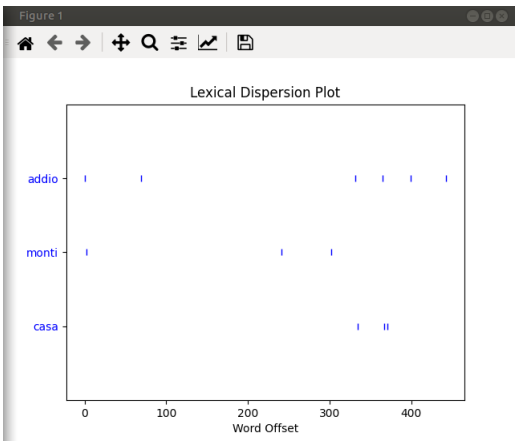
```
Displaying 6 of 6 matches:
 addio , monti sorgenti dall'acque , ed ele
 , come branchi di pecore pascenti ; addio ! quanto è tristo il passo di chi ,
n momento stabilito per il ritorno ! addio , casa natìa , dove , sedendo , con
aspettato con un misterioso timore . addio , casa ancora straniera , casa soggu
rno tranquillo e perpetuo di sposa . addio , chiesa , dove l'animo tornò tante
enir comandato , e chiamarsi santo ; addio ! chi dava a voi tanta giocondità è
```

che mostra le 6 ricorrenze della parola addio nel loro contesto.

`testo.collocations()`
ritorna la seguente schermata

```
cresciuto tra; tante volte; tra voi; più care
```

testo.dispersion_plot(['addio', 'monti', 'casa'])
ritorna questo grafico



con indicate le posizioni delle parole indicate dalla lista nel testo.

`len(testo)`
ritorna 478, che è il numero di token.
Se contiamo le parole del testo sono 406. I token sono di più perché sono token, oltre alle parole, anche i segni di punteggiatura, gli apostrofi, ecc.

`nl.FreqDist(testo)`
ritorna la seguente schermata

```
FreqDist({',': 54, 'e': 17, ';': 14, 'di': 9, 'a': 8, 'non': 8, 'un': 8, 'che':
7, "'": 7, 'il': 7, ...})
```

che elenca le prime dieci maggiori ricorrenze a dizionario.
`nl.FreqDist(testo).most_common(20)`
ritorna la seguente schermata

```
[(',', 54), ('e', 17), (';', 14), ('di', 9), ('a', 8), ('non', 8), ('un', 8), ('
che', 7), ("'", 7), ('il', 7), ('addio', 6), ('chi', 6), ('più', 6), ('!', 5), (
'si', 5), ('.', 5), ('è', 4), ('de', 4), ('se', 4), ('con', 4)]
```

che elenca le prime venti, questa volta come lista di tuple.

2.2 Pulizia del testo

Abbiamo visto nel precedente paragrafo che un testo, se lo suddividiamo nei token che lo compongono, contiene una miriade di simboli (punteggiatura) e ricorrenze (articoli, congiunzioni, ecc.) che nulla aggiungono all'essenza del suo contenuto, servono solo a noi umani per leggerlo meglio e per gustarne le sfumature e, spesso, la musicalità.

Se lo leggiamo con il computer, al fine, per esempio, di stabilire se si tratta di un testo allegro o di un testo triste, come vedremo si possa fare nel prossimo paragrafo, possiamo benissimo lavorare solo sulle parole essenziali ed eliminare tutto ciò che non serve, alleggerendo così il materiale da esaminare.

Per ripulire un testo a questi fini, oltre che attrezzarci per la sua tokenizzazione secondo quanto visto nel paragrafo precedente, dobbiamo procurarci un archivio che contiene i simboli e i termini che possiamo eliminare da un testo senza perderne l'essenza. Nel mondo Python questo archivio si trova in un file denominato `stopwords`.

Lo possiamo installare sul nostro computer in maniera stabile in modo da averlo sempre a disposizione, previa importazione di NLTK in una shell Python o nella IDLE con

`import nltk as nl`,

proseguendo con

`nl.download('stopwords')`.

L'archivio può essere utilizzato per lavorare su testi scritti in una delle oltre venti lingue riconosciute, tanti sono i file delle parole "inutili" che ci vengono proposti.

Bene sapere che il file è un semplice file di testo, con una parola per riga, e lo possiamo modificare con un editor di testo arricchendolo di altre parole o simboli: per esempio questi file non contengono i simboli di punteggiatura (, . : ; ! ? ecc.) e possiamo utilmente aggiungerli indicando ciascuno di essi su altrettante righe.

Il file per la lingua italiana si chiama `italian` e si trova all'indirizzo

`/nltk_data/corpora/stopwords/`,

essendo la directory `nltk_data` nella nostra `home` (se usiamo Linux o Mac) o in `AppData\Roaming` del nostro percorso utente se usiamo Windows.

Il procedimento di ripulitura consiste nello scorrere i token di un file di testo ricopiando in un nuovo file solo i token non appartenenti alla lista delle stopwords. In questo modo avremo un testo derivato dal precedente e ripulito dei termini che non servono per interpretarne l'essenza.

La lista delle stopwords che ci interessa, per esempio quella della lingua italiana, la rendiamo disponibile per il procedimento con i comandi

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('italian'))
```

Esempio:

Il seguente script ripulisce il file di testo contenente l'Addio di Lucia che abbiamo utilizzato nel precedente paragrafo.

```
import nltk as nl
from nltk.corpus import stopwords
stop_words = set(stopwords.words('italian'))
f = open('/home/vittorio/Documenti/addio.txt')
testo = nl.Text(nl.word_tokenize(f.read()))
for parola in testo:
    if not parola in stop_words:
        ff = open('/home/vittorio/Documenti/addio_pulito.txt', 'a')
        ff.write(" " + parola)
        ff.close()
```

In questo modo abbiamo creato il file `addio_pulito.txt` che contiene l'addio di Lucia ridotto a 238 parole, quando il file originario ne contiene 406.

2.3 Sentiment analysis

La sentiment analysis è un'analisi del testo finalizzata a stabilire se il testo stesso esprime un sentimento positivo, negativo o neutrale.

Viene utilizzata con profitto per classificare i giudizi espressi dai consumatori su determinati prodotti senza che per questo sia necessario occupare un operatore alla lettura dei giudizi stessi: questi vengono letti e classificati dal computer.

Bell'esempio di intelligenza artificiale. Il trucco, come per ciò che abbiamo visto nel paragrafo precedente, dove il computer eliminava le parole inutili in quanto noi gli abbiamo detto quali sono, sta nell'istruire il computer su quali parole hanno connotazione positiva, quali hanno connotazione negativa e quali hanno connotazione neutrale.

Per fare questo esiste parecchio materiale già predisposto, sia nel mondo Python, sia in altri contesti.

Qui propongo la versione multi-lingua di VADER (Valence Aware Dictionary and sEntiment Reasoner) che funziona solo con collegamento Internet attivo in quanto si avvale delle Google Translate API attraverso la libreria Python `translatte`.

Installiamo una tantum quanto ci serve con pip
`pip` (o `pip3`) `install vader-multi`.

Per compiere l'analisi dobbiamo importare l'analizzatore con
`from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer`
e poi costruire il nostro analizzatore con
`analizzatore = SentimentIntensityAnalyzer()`.

Sottoponendo al metodo `polarity_scores` dell'analizzatore una stringa di testo otteniamo un dizionario che contiene
. in corrispondenza alla chiave `'neg'` un valore indicante la valenza negativa,
. in corrispondenza alla chiave `'neu'` un valore indicante la valenza neutra,
. in corrispondenza alla chiave `'pos'` un valore indicante la valenza positiva,
. in corrispondenza alla chiave `'compound'` un valore di sintesi complessiva.
I primi tre valori sono compresi tra 0 e 1 ed esprimono il peso delle varie valenze.
Il quarto valore, di sintesi complessiva, varia tra -1 e 1 ed esprime una valenza complessiva tanto più positiva quanto più il valore si avvicina a 1, tanto più negativa quanto più il valore si avvicina a -1 e tendente alla neutralità per valori attorno allo zero.

Esempio di sentiment analysis su frasi:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analizzatore = SentimentIntensityAnalyzer()
analizzatore.polarity_scores('Ho trovato il libro molto noioso')
```
restituisce il dizionario
```
{'neg': 0.341, 'neu': 0.659, 'pos': 0.0, 'compound': -0.3804}
analizzatore.polarity_scores('Oggi è una bellissima giornata')
```
restituisce il dizionario
```
{'neg': 0.0, 'neu': 0.506, 'pos': 0.494, 'compound': 0.5994}
```
Possiamo divertirci anche con altre lingue con lo stesso analizzatore:
```
analizzatore.polarity_scores('VADER is very smart and funny')
```
restituisce il dizionario
```
{'neg': 0.0, 'neu': 0.394, 'pos': 0.606, 'compound': 0.7316}
analizzatore.polarity_scores('Je suis très heureux de vous connaître')
```
restituisce il dizionario
```
{'neg': 0.0, 'neu': 0.6, 'pos': 0.4, 'compound': 0.6115}
```

Il testo da analizzare deve essere in forma di stringa e non necessariamente è necessario passare attraverso la tokenizzazione e la ripulitura di cui abbiamo parlato nei paragrafi precedenti.
Nel seguente esempio vediamo come sia possibile determinare il sentiment dell'addio di Lucia che abbiamo in un file di testo e del relativo derivato ripulito che abbiamo in un altro file di testo:

```
f = open('/home/vittorio/Documenti/addio.txt', 'r')
ff = open('/home/vittorio/Documenti/addio_pulito.txt', 'r')
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analizzatore = SentimentIntensityAnalyzer()
analizzatore.polarity_scores(f.read())
{'neg':  0.068, 'neu':  0.761, 'pos':  0.171, 'compound':  0.9954}
analizzatore.polarity_scores(ff.read())
{'neg':  0.101, 'neu':  0.681, 'pos':  0.218, 'compound':  0.9922}
```

La caratterizzazione altamente positiva del testo è confermata sia dall'analisi del testo originario sia dall'analisi del testo ripulito. Nel secondo caso ha lavorato meno l'analizzatore in quanto le parole da analizzare erano in quantità minore.

Ovviamente, in presenza di grandi moli di frasi da analizzare occorre organizzare un set di dati riconducibile ad uno dei tipi contenitori di Python, ad esempio una lista, ed agire su quella.

Esempio:

Poniamo di avere in una lista Python i seguenti giudizi espressi dai lettori di un libro:

```
giudizi = ["Ho letto il libro e l'ho trovato molto noioso", "L'autore ha confermato la sua maestria
nel presentare personaggi e situazioni", "Ho trovato il libro molto interessante", "Mi verrebbe da
dire che ho letto una schifezza"].
```

Con il seguente script troviamo e stampiamo il risultato dell'analisi:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analizzatore = SentimentIntensityAnalyzer()
positivi = 0; negativi = 0; neutrali = 0
for x in giudizi:
    sentiment = analizzatore.polarity_scores(x)
    if sentiment['compound'] > 0:
        positivi = positivi + 1
    elif sentiment['compound'] < 0:
        negativi = negativi + 1
    else:
        neutrali = neutrali + 1
print("giudizi positivi = ", positivi)
print("giudizi negativi = ", negativi)
print("giudizi neutrali = ", neutrali)
```

Il risultato è:

```
giudizi positivi = 1
giudizi negativi = 2
giudizi neutrali = 1
```

Evidentemente il risultato positivo è attribuito al terzo giudizio, i due risultati negativi sono attribuiti al primo e al quarto giudizio e risulta neutrale il secondo giudizio.

Nulla vieta, infine, che l'analisi del sentiment possa essere applicata in tempo reale a frasi scritte su twitter e c'è chi lo fa anche a nostra insaputa per tastare il polso su certe situazioni.

Addirittura frasi dette al telefono possono essere analizzate.

Lo script seguente mostra come sia possibile organizzare un dialogo con il computer formulando certe risposte in base al sentiment di certe altre.
Per la sintesi e il riconoscimento vocale mi avvalgo di quanto descritto nell'allegato «tts_stt_python» al mio articolo «Python parla e ascolta» pubblicato nell'ottobre 2021 sul mio blog all'indirizzo *www.vittal.it*.

```
import pyttsx3
import speech_recognition as sr
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
recitante = pyttsx3.init()
recitante.setProperty('voice', 'italian')
recitante.say('Come ti chiami?')
recitante.runAndWait()
riconoscitore = sr.Recognizer()
with sr.Microphone() as source:
    audio = riconoscitore.listen(source)
testo = riconoscitore.recognize_google(audio, language = "it")
f = open('/home/vittorio/Documenti/saluto.txt', 'w')
f.write('Ciao' + testo + 'come stai?')
f.close()
```

```
f = open('/home/vittorio/Documenti/saluto.txt', 'r')
recitante.say(f.read())
recitante.runAndWait()
riconoscitore = sr.Recognizer()
with sr.Microphone() as source:
    audio = riconoscitore.listen(source)
testo = riconoscitore.recognize_google(audio, language = "it")
analizzatore = SentimentIntensityAnalyzer()
giudizio = analizzatore.polarity_scores(testo)
if giudizio['compound'] > 0:
    recitante.say("Mi fa piacere")
    recitante.runAndWait()
else:
    recitante.say("Oh.  Mi dispiace!")
    recitante.runAndWait()
```

Nel colloquio prodotto con questo script, se alla domanda «come stai?» rispondiamo con una frase con sentiment positivo, tipo «Molto bene», il computer risponde, a sua volta, con «Mi fa piacere». Se invece rispondiamo con una frase con sentiment negativo, tipo «Male, ho un gran dolore ai denti», il computer risponde, a sua volta, con «Oh. Mi dispiace».

3 Scikit-learn

Scikit-learn è il modulo Python dedicato al machine learning, la nuova frontiera della scienza dei dati; in italiano si dice apprendimento automatico.

Il termine è stato coniato nel lontano 1959 dallo scienziato americano Arthur Lee Samuel, ma la definizione di ciò che si intende oggi per machine learning è più recente e ce l'ha data un altro americano, Tom M. Mitchell, nel 1997: «Si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E.»

Ci si può chiedere come mai un a disciplina della quale si parla da oltre cinquant'anni sia diventata attuale solo da poco più di una decina d'anni.

La risposta è semplice: il machine learning ha senso solo in presenza di una grande quantità di dati e necessita di grande capacità di calcolo, cose che cinquant'anni fa non c'erano.

All'indirizzo *https://scikit-learn.org/stable/* troviamo tutto ciò che c'è da sapere su Scikit-learn e dalla home page apprendiamo che questo modulo Python è basato sui moduli NumPy, SciPy e Matplotlib e ce ne vengono indicati i tre gruppi di applicazioni:
. classificazione,
. regressione,
. clustering.

Il primo ci propone metodi per determinare la categoria di appartenenza di un oggetto (ad esempio per identificare messaggi SPAM nella posta elettronica o per il riconoscimento di immagini).

Il secondo ci propone metodi per prevedere valori associabili ad un oggetto (ad esempio ricerca di legami funzionali tra variabili). Anche in SciPy abbiamo la regressione lineare, che è il principale di questi metodi, ma qui c'è molto altro e la stessa regressione lineare è migliore (per esempio funziona meglio in presenza di variabili non standardizzate).

Il terzo ci propone metodi per il raggruppamento di oggetti simili tra loro (ad esempio per segmentare la clientela). Anche in SciPy abbiamo uno di questi metodi, K-means, ma qui abbiamo molto di più e lo stesso K-means è arricchito dalla possibilità di identificare preventivamente il numero dei gruppi.

Possiamo installare Scikit-learn con pip con il comando
`pip` (o `pip3`) `install scikit-learn`

Il modulo viene importato negli script con il nome abbreviato `sklearn`.

Nella home page di Scikit-learn abbiamo i link alla User Guide e alle API. Basta dare un'occhiata e ci accorgiamo che abbiamo a che fare con qualche cosa che esula dagli interessi e dalle

capacità del principiante cui è dedicato questo manualetto. E già il principiante a cui penso è uno che sa usare un computer ed ha una preparazione di base di matematica generale e statistica.

In questa sede mi limito pertanto ad un paio di esempi, tanto per vedere come funziona.

3.1 Esempio di machine learning supervisionato: regressione multipla

Partiamo da un piccolo dataset, di dimensioni sufficienti per capirci, contenuto in un file .csv, nel quale abbiamo una variabile y accostata al manifestarsi di sei diverse terne di variabili x.

y	x1	x2	x3
34	12	11	21
72	26	20	45
87	44	12	58
28	32	14	37
16	21	11	24
97	76	7	75

Ci proponiamo di verificare quale sia il più probabile valore della variabile y in presenza della terna inedita in cui x1 valga 11, x2 valga 22 e x3 valga 33.

Con questo script

```
import pandas as pd
dati = pd.read_csv('/home/vittorio/Documenti/dati.csv')
X = dati[['x1','x2','x3']]
y = dati['y']
from sklearn.linear_model import LinearRegression
regressore = LinearRegression()
regressore.fit(X, y)
print('intercetta', regressore.intercept_)
coefficienti = pd.DataFrame(regressore.coef_,X.columns,columns=['coefficienti'])
print(coefficienti)
y_calcolati = regressore.predict(X)
from sklearn import metrics
print('R2', metrics.r2_score(y, y_calcolati))
```

otteniamo i seguenti risultati

```
intercetta 2.879968788655681
coefficienti
x1 -2.648351
x2 -2.616547
x3 4.122166
R2 0.972625953228526
```

dove abbiamo l'intercetta (il punto dell'asse ortogonale verticale da cui parte la retta di regressione), i coefficienti delle variabili x e il coefficiente di determinazione, altrimenti chiamato R quadro: se questo indicatore è prossimo all'unità, come accade nel nostro caso, significa che il modello ottenuto interpreta con attendibilità molto elevata le relazioni esistenti tra le variabili.

A questo punto siamo abbastanza sicuri che, in presenza dei valori di x1, x2, x3, rispettivamente 11, 22 e 33, il valore di y sia:

```
2.879968788 - 2.648351 x 11 - 2.616547 x 22 + 4.122166 x 33 = 52.215551788
```

3.2 Esempio di machine learning non supervisionato: clusterting

Stiamo parlando di metodi di analisi applicabili a big data ma ancora esemplifico su dimensioni ridotte, per fare prima e capirci meglio.

Abbiamo fatto un'indagine sui nostri clienti - nel caso sono solo 12 ma potrebbero anche essere 120.000 - e siamo venuti a conoscere ciò che vediamo in questa tabella, memorizzata nel file clienti.csv:

cliente	cambio	prezzo	marca	frequenza
Luigi	5	5	5	3
Maria	7	7	4	2
Vittorio	6	5	5	3
Giovanni	6	6	4	2
Beatrice	5	5	4	3
Giuseppe	9	8	2	4
Elena	8	8	2	5
Antonio	9	7	3	4
Paola	10	7	3	4
Mario	9	8	2	4
Carlo	9	10	2	5
Cecilia	10	8	3	3

Le grandezze numeriche variano da 1 (per niente) a 10 (moltissimo).
La colonna «cambio» indica la propensione del cliente a cambiare negozio.
La colonna «prezzo» indica la sensibilità del cliente al prezzo della merce.
La colonna «marca» indica la preferenza del cliente per articoli di marca.
La colonna «frequenza» indica il numero medio di acquisti in un anno.

Aiutati dalla piccola dimensione del data set e dall'ordine con cui ho esposto i dati notiamo subito a occhio che abbiamo un gruppo di clienti, i primi cinque, tendenzialmente quelli che fanno meno acquisti, che hanno una moderata propensione a cambiare negozio, e sono moderatamente sensibili al prezzo e alla marca; questo gruppo si contrappone al gruppo degli altri sette, con molto più elevata propensione a cambiare negozio, molto più sensibili al prezzo, meno sensibili alla marca e mediamente migliori clienti sul piano della frequenza di acquisto.

Tutte cose utili per le nostre strategie di marketing o semplicemente per qualche intervento tendente a fidelizzare meglio i sette clienti più «volatili».

Ma vediamo come anche la macchina si accorge di ciò che abbiamo adocchiato noi utilizzando il più diffuso algoritmo di clustering, K-means, noto anche per la sua leggerezza a sollievo del lavoro della CPU del computer.

Con questo script

```
import pandas as pd
dati = pd.read_csv('/home/vittorio/Documenti/clienti.csv', usecols = ['cambio','prezzo','marca','frequenza'])
from sklearn.cluster import KMeans
gruppi = KMeans(n_clusters = 2, init = 'k-means++', tol = 0.0001)
gruppi_c = gruppi.fit_predict(dati)
print('raggruppamento clienti:  ', gruppi_c)
print('inerzia:  ', gruppi.inertia_)
```

otteniamo il seguente risultato

```
raggruppamento clienti:  [0 0 0 0 0 1 1 1 1 1 1 1]
inerzia:  21.82857142857143
```

da cui deriviamo che i primi cinque clienti appartengono al cluster 0 e gli altri sette al cluster 1. L'indicatore chiamato «inerzia» è una somma di errori quadratici medi derivanti dal raggruppamento proposto e, per avere una segmentazione accurata deve essere il più basso possibile. Purtroppo il suo valore assoluto che vediamo dice poco e andrebbe confrontato con altri. Di per sé, comunque, avendo la nostra tabella di partenza dati compresi tra 1 e 10, mi pare che un valore di quasi 22 non possa essere considerato basso: ne deriva che la nostra segmentazione non è gran che, almeno dal punto di vista scientifico, anche se sul piano pratico pare funzionare abbastanza bene.

* * *

Con questi due esempi notiamo che anche la scrittura del codice con il modulo Scikit-learn si complica abbastanza.

E' per questo motivo che qualcuno ha pensato di regalare ai principianti, e non solo, quel gioiello del software libero che si chiama Orange, che ho presentato sul mio blog all'indirizzo *www.vittal.it* nel dicembre 2020 attraverso il manualetto "orange", in formato PDF, allegato all'articolo "Orange: data science con Python senza scrivere codice".

I due esempi sono svolti anche in quella sede con l'ausilio di Orange.