

MODULI FORMATIVI ESTATE 2023

INTELLIGENZA ARTIFICIALE: PROFESSIONE FUTURO

# Natural Language Processing

LEZIONE II - 29/08/2023



ARTIFICIAL INTELLIGENCE  
& DATA ANALYTICS

[ai.units.it](http://ai.units.it)

# Natural Language Processing

- **Natural Language** (linguaggio naturale)
  - Come le persone comunicano tra di loro
- **Processing**
  - Come un calcolatore elabora le informazioni

## Natural Language Processing

Come un calcolatore gestisce testi in linguaggio naturale

# Cosa è NLP?

Tecniche per il trattamento dell'informazione espressa in linguaggio naturale:

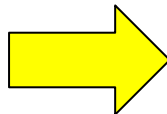
- **Tecniche grammaticali**: basate sulla struttura grammaticale o morfologica delle espressioni linguistiche
- **Tecniche sintattiche**: principalmente basate sul lessico e sulle distribuzioni statistiche dei termini
- **Tecniche semantiche**: basate sui significati dei termini e delle espressioni

# Applicazioni di NLP?

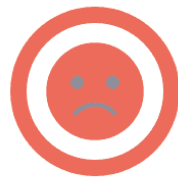
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut eget efficitur lorem, et maximus massa. Fusce faucibus quis nunc sollicitudin commodo. Nam condimentum justo at purus malesuada tempor. Curabitur vitae lacus finibus, aliquet risus nec, aliquet felis. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum ex felis, eleifend eu magna id, bibendum fermentum ante. Mauris sapien eros, interdum scelerisque egestas iaculis, eleifend sed erat. Donec eleifend lorem dictum eros imperdiet, interdum ultrices lorem tempus. Aenean id lacus eu ex eleifend mollis. Donec leo ex, tempus et eleifend id, sollicitudin sit amet leo. Maecenas in sem posuere nisl varius interdum feugiat cursus mauris. Nam ut bibendum elit. Pellentesque laoreet, orci tristique fringilla auctor, augue ipsum cursus turpis, in viverra odio ex eu diam. Nullam finibus arcu id mauris commodo faucibus. Aenean lobortis purus at sem scelerisque, vitae pellentesque arcu consectetur.

Cras imperdiet efficitur leo, sed dapibus sem porttitor ut. Vestibulum at neque turpis. Praesent tincidunt pretium urna, nec bibendum velit maximus ac. Quisque non mauris ultrices turpis malesuada faucibus. Suspendisse potenti. Mauris pretium tempor finibus. Suspendisse sit amet consequat dui. Maecenas luctus nunc libero, in viverra odio tristique ut. Sed sed tellus laoreet, rhoncus magna at, sagittis lectus. Sed quis elementum neque, quis pharetra dui.

Mauris non tincidunt quam. Aenean urna ligula, porttitor a purus a, convallis efficitur turpis. Duis tempus arcu est, ut imperdiet dolor tincidunt sit amet. Aliquam risus magna, dignissim vel consectetur in, porta eget ligula. Vivamus ullamcorper quis sem sit amet aliquet. Donec nec dolor rutrum, dictum libero sed, bibendum quam. In hac habitasse platea dictumst. Ut quis nunc sit amet quam dapibus finibus quis nec tellus. Sed porttitor vestibulum lacus nec semper. Vivamus odio velit, consequat sit amet auctor ut, pharetra eget mi. Suspendisse nec porttitor elit, in placerat justo. Nulla ac porttitor elit, vel sollicitudin odio.



Positive



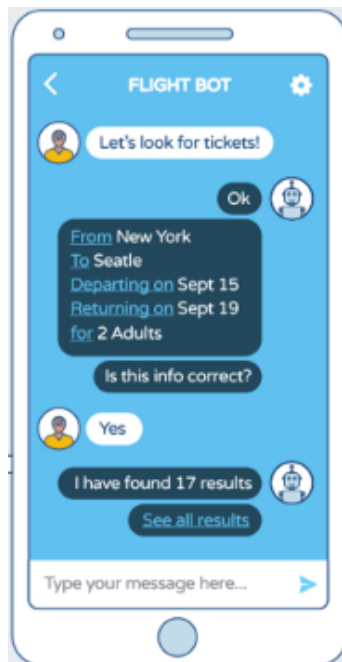
Negative



Neutral

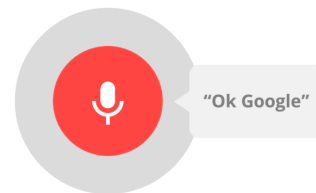
## Sentiment analysis

# Applicazioni di NLP?



ChatBot

# Applicazioni di NLP?

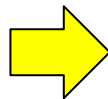
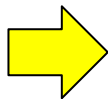


Hey Siri

Speech recognition

# Applicazioni di NLP?

Μῆνιν ἄειδε θεὰ Πηληϊόδεω Ἀχιλῆος·  
οὐλομένην, ἣ μυρί' Ἀχαιοῖς ἄλγε' ἔθηκε,  
πολλὰ δ' ἴφθιμους ψυχὰς Ἀϊδὶ προΐψεν  
ἠρώων, αὐτοῦ δ' ἑλδ' ἔλδ' ὀριᾶτο  
ἑὺχέ' κύνεσσιν οἰώνοισί τε παῖσι,  
Διὸς δ' ἐτελεύετο βουλή,  
ἔξ οὔ δὴ τὰ πρότα' δι' ἁστῆτην ἔρσαντε



Cantami, o Diva, del Pelide Achille  
L'ira funesta che infiniti addusse  
Lutti agli Achei, molte anzi tempo all'Orco  
Generose travolse alme d'eroi,  
di cani e d'augelli orrido pasto  
Lor salme abbandonò (così di Giove

L'alto consiglio s'adempia), da quando  
Primamente disgiunse aspra contesa  
Il re de' prodi Atride e il divo Achille.

Machine translation

# Applicazioni di NLP?



Controllo ortografico



Information extraction



Ricerca parole chiave



Advertising



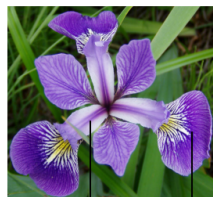
# Processare informazioni

iris setosa



petal sepal

iris versicolor



petal sepal

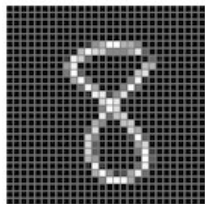
iris virginica



petal sepal

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

# Computer vision

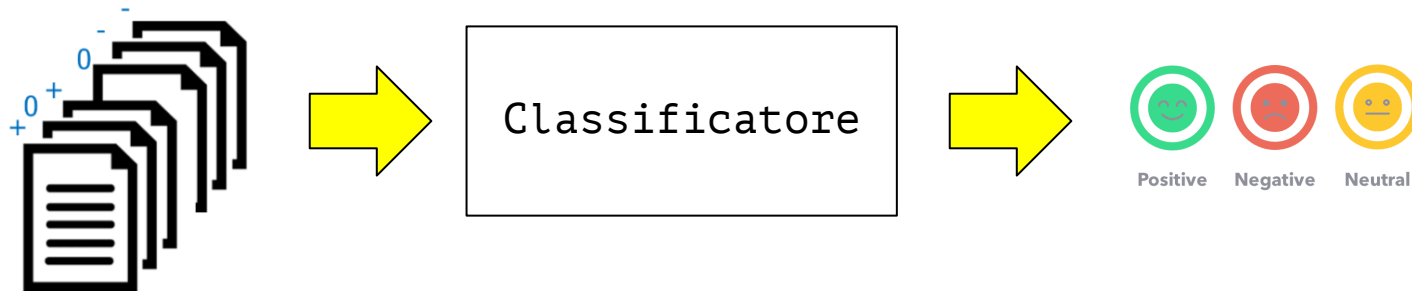


28 x 28  
784 pixels

[illegible]

# Sentiment analysis

- Input: testo
- Output: classificazione **polarità** documento
  - Positiva
  - Negativa
  - Neutra



# Che **cosa** andiamo a “**processare**”?

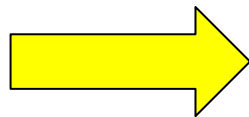
- Come elaboriamo il testo? Come lo rendiamo “digeribile” per un calcolatore?
- Ci serve una rappresentazione, diverse possibilità:
  - Sequenza di **lettere**?
  - Sequenza di **parole**?
  - Sequenza di **numeri**?

# Tokenization

Riduzione di un testo in **unità** atomiche (indivisibili) di nostra **scelta**

**Esempio 1:** parole

The quick brown fox  
jumps over the lazy  
dog



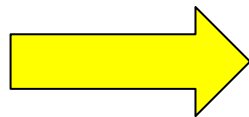
The  
quick  
brown  
fox  
jumps  
over  
the  
lazy  
dog

# Tokenization

Riduzione di un testo in **unità** atomiche (indivisibili) di nostra **scelta**

**Esempio 2:** caratteri

The quick brown fox  
jumps over the lazy  
dog

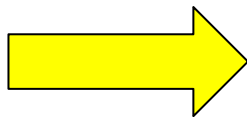


T h e  
q u i c k  
b r o w n  
f o x  
j u m p s  
o v e r  
t h e  
l a z y  
d o g

# Bag of words

**Esempio 3:** usare un vettore per contare quante volte una unità (token, parola) appare in un documento

The quick brown fox  
jumps over the lazy  
dog



The	2
quick	1
brown	1
fox	1
jumps	1
over	1
lazy	1
dog	1

# Bag of words per analizzare parole

Proviamo a cercare “**affinità**” tra le parole *gatto*, *matto* e *felino* **contando** quante parole hanno in comune su Wikipedia

Gatto (Felis silvestris catus)		Matto (Il Matto)		Felino (Felidae)	
di	279	il	54	-	46
e	200	di	46	gatto	31
il	194	e	41	di	28
la	166	è	37	e	22
È	120	un	32	genere	18



# Parole funzionali

Alcune parole sono **molto frequenti**, non hanno un significato proprio ma hanno una funzione grammaticale importante.

## Stop words:

- Non hanno un significato
- Di solito sono brevi (pochi caratteri)
- Modificano il significato di altre parole

Posso prendere ***una*** macchina

Posso prendere ***la*** macchina

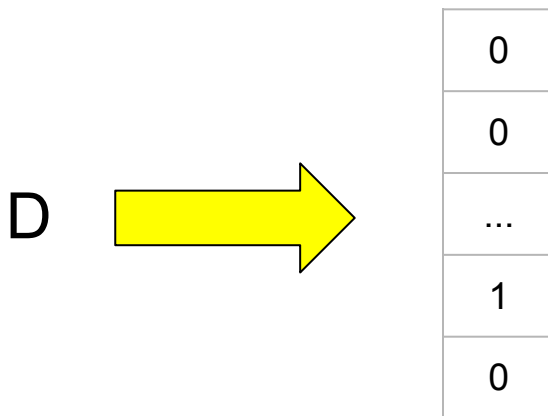
# Bag of words per analizzare parole

Proviamo a cercare “affinità” tra le parole *gatto*, *matto* e *felino* **contando** quante parole hanno in comune su wikipedia, **eliminando** le stop words

Gatto (Felis silvestris catus)		Matto (Il Matto)		Felino (Felidae)	
gatto	118	==	20	-	46
gatti	62	matto	16	gatto	31
===	34	the	12	genere	18
==	32	può	12	leopardus	15
può	31	altri	9	felidi	14

# Bag of words: interpretazione

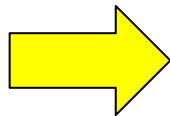
Abbiamo un documento D che contiene una **sola parola**



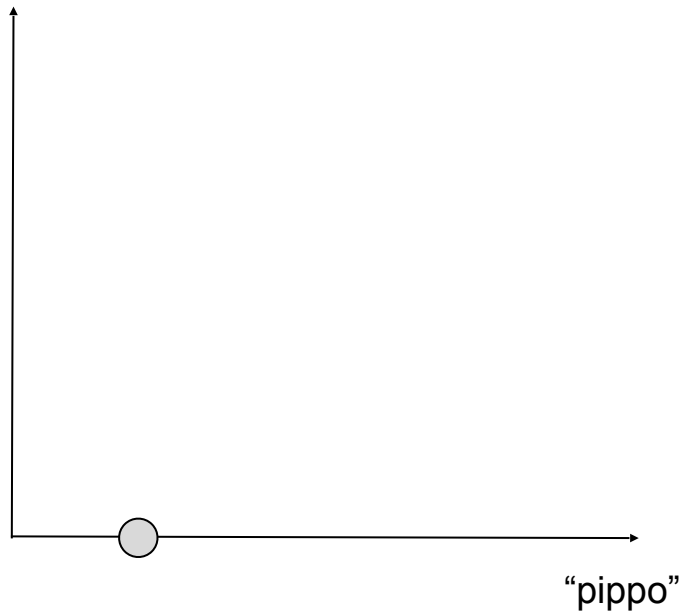
**Interpretazione:** “una parola è un punto di **coordinate** 1 sul proprio asse”

# Bag of words: interpretazione

Pippo

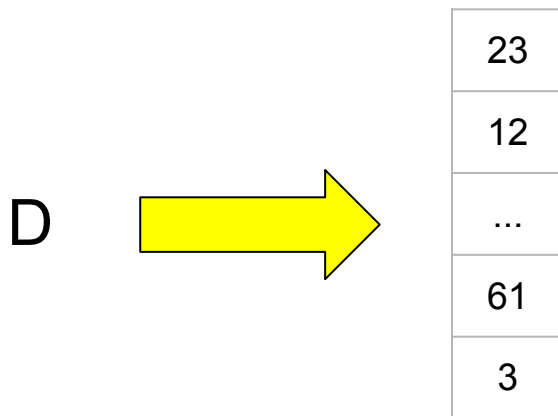


	0
	0
	0
	0
pippo	1
	0
	0
	0



# Bag of words: interpretazione

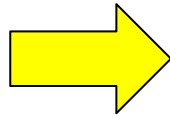
Abbiamo un documento D che contiene **molte parole**



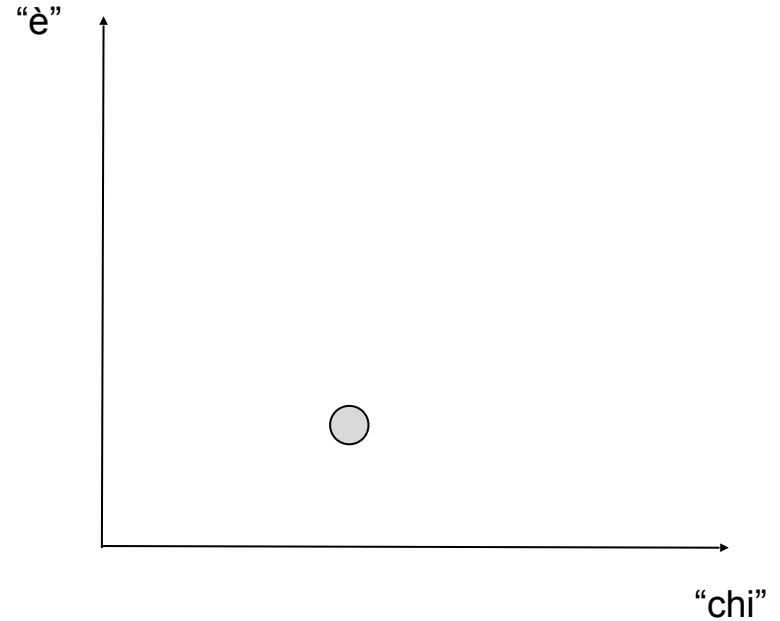
**Interpretazione:** “un documento è il punto **risultante** dalla somma delle parole che contiene”

## Bag of words: interpretazione

Chi è chi?



	0
chi	2
	0
	0
	0
	0
	0
è	1



# Può funzionare?

- Una bag of words permette di risolvere qualche **semplice** problema...
- ...e permette di valutare la **distanza euclidea** tra le bag of words...
- ...ma per approcci più generali dovremmo usare vettori molto lunghi
- Ogni elemento del vettore fa riferimento ad una parola del dizionario
- Occupa decisamente **troppo spazio!**
- Inoltre, parole che non sono nel dizionario, non esistono!

# Distanza tra parole

Abbiamo visto come misurare una distanza tra documenti che hanno molte parole.

Possiamo misurare distanza tra parole **singole**?

- Un vettore bag of words con una sola parola prende il nome di **one-hot**
- Problema di fondo: tutte le parole hanno la **stessa** distanza tra di loro!
- **Nessuna** informazione di carattere **semantico**
- Le parole singole *gatto*, *matto* e *felino* hanno la stessa distanza tra di loro



# Perdita di informazioni

- In una rappresentazione bag of words tutte le parole hanno la **stessa importanza**
- Per fortuna esistono altre rappresentazioni, che **pesano** ogni singola parola (per esempio la rappresentazione TF-IDF)

# Idea

Usando one-hot abbiamo uno spazio con...

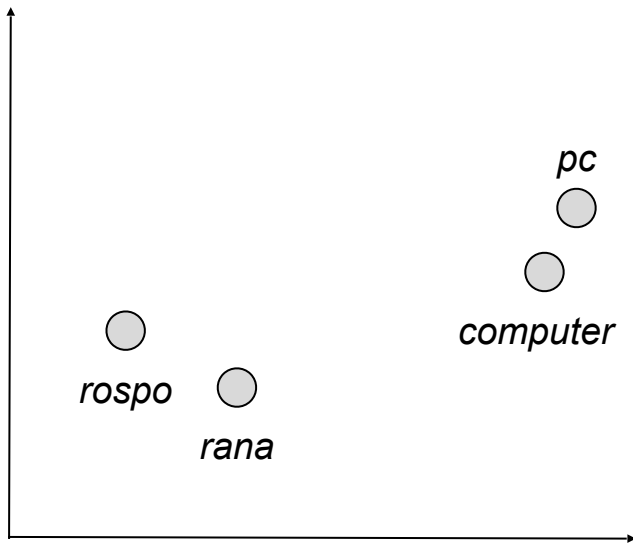
- Tante **dimensioni** (assi) quante le parole
- Distanza in questo spazio **non** hanno valore **semantico**

...ma vorremmo:

- Un numero **minore** di dimensioni
- Delle distanze con un valore **semantico**

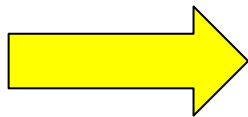
# Nuova rappresentazione

Vogliamo costruire uno spazio di rappresentazione dove parole con significato **simile** sono **vicine** tra di loro

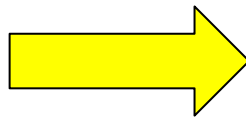


# Nuova rappresentazione

*rana*



0
0
0
...
0
1
0



0.16
0.63
...
0.01
0.57

# TF-IDF

Sta per: **term frequency–inverse document frequency**.

Per ogni parola  $w$  in un documento  $D$ :

- Quantifico la “**specificità**” di  $w$  in  $D$ 
  - Se  $w$  è presente **spesso** in  $D$  la sua specificità è bassa
  - Se  $w$  è presente **raramente** in  $D$  la sua specificità è alta
- **Peso**  $w$  con la sua specificità in  $D$

# TF-IDF

Sta per: **term frequency–inverse document frequency**.

Per ogni parola  $w$  in un documento  $D$ :

- Quantifico la “**specificità**” di  $w$  in  $D$ 
  - Se  $w$  è presente **spesso** in  $D$  la sua specificità è bassa
  - Se  $w$  è presente **raramente** in  $D$  la sua specificità è alta
- **Peso**  $w$  con la sua specificità in  $D$

Il vettore  $D$  in questo modo si allunga negli assi più specifici e si accorcia in quelli meno specifici: **cambia** la **direzione** complessiva

# Specificità

Come si **calcola** la “specificità” (ci sono tanti modi, questo è uno):

- $\text{conta}(w, D)$  = numero di volte che la parola  $w$  compare in  $D$
- $\text{specificità}(w, D) = \log(|D|/\text{conta}(w, D))$

In questo modo:

- Se una una parola è **comune**, allora  $\text{conta}(w, D)$  è grande e la specificità tende a 0
- Se una parola è **rara**, allora  $\text{conta}(w, D)$  è piccola e la specificità tende a crescere

# TF-IDF

Sta per: **term frequency–inverse document frequency**:

- **Term frequency:**

$\text{conta}(w, D)$  = numero di volte che la parola  $w$  compare in  $D$

- **Inverse document frequency:**

$\text{specificità}(w, D) = \log(|D|/\text{conta}(w, D))$

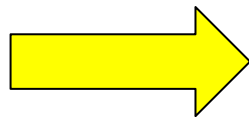


# Tokenization

Riduzione di un testo in **unità** atomiche (indivisibili) di nostra **scelta**

**Esempio 4:** digrammi

The quick brown fox  
jumps over the lazy  
dog



T he e  
q qu ui ic ck k  
b br ro ow wn n  
f fo ox x  
j ju um mp ps s  
o ov ve er r  
t th he e  
l la az zy y  
d do og g

# n-grammi

A volte può essere più interessante spezzare un testo in **unità** diverse dalle “parole” o dalle “lettere”

- **n-gramma**: sotto-sequenza di lunghezza  $n$  di un testo
- Più flessibile rispetto all'usare parole intere (occupa meno spazio)
- Offre delle informazioni in più rispetto alle singole lettere
- Diversi usi, anche al di fuori di NLP (per esempio: sequenziamento del DNA)

# Individuare la lingua

**Problema:** dato un documento di testo vogliamo capire quale è la lingua in cui è scritto il documento

- Abbiamo a disposizione **tanti** esempi di testi in varie lingue
- Arriva un documento di cui **non** conosciamo la lingua
- Vogliamo automaticamente capire che lingua sia

# Individuare la **lingua**

Costruiamo un profilo per **ogni** lingua

Algoritmo per costruire un profilo:

- Spezziamo il testo in token **rimuovendo** punteggiatura
- Per ogni token costruiamo gli **n-grammi** (n da 1 a 5)
- **Misuriamo** quante volte compaiono i singoli n-grammi
- **Teniamo** i 300 n-grammi più frequenti

# Profilo linguistico

Intuitivamente:

- Gli n-grammi più frequenti saranno gli **1-grammi**
- Gli 1-grammi rispecchieranno le frequenze delle **lettere** in una lingua
- Subito dopo ci saranno gli n-grammi che rappresentano le **stop-words** di una specifica lingua
- Oltre la 300° posizione avremmo n-grammi relativi al **contenuto**

# Individuare la **lingua**

Algoritmo per **assegnare** la lingua ad un documento D:

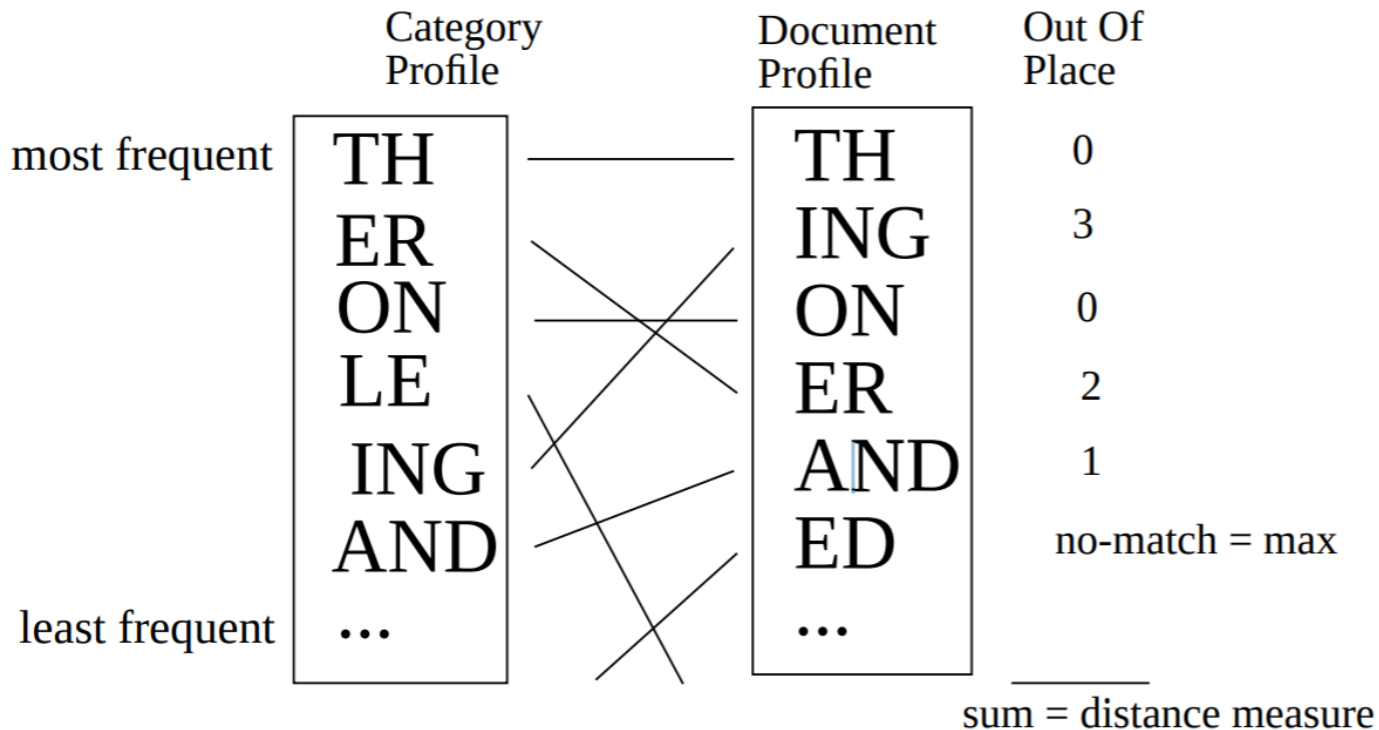
- Spezziamo il testo in token **rimuovendo** punteggiatura
- Per ogni token costruiamo gli **n-grammi** (n da 1 a 5)
- **Misuriamo** quante volte compaiono i singoli n-grammi
- **Teniamo** i 300 n-grammi più frequenti
- Misuriamo la **distanza** tra il profilo di D e quello di ogni lingua
- Il profilo a distanza **minore** sarà quello della lingua di D
- **come** misuriamo la distanza?

# Ranking distance

Prendiamo due **profili**, quello del documento D e quello di una lingua:

- **Scorriamo** gli n-grammi di un profilo dal primo all'ultimo
- Per ogni n-gramma, guardiamo se è nella **stessa** posizione nel secondo profilo
- Se è fuori posizione, contiamo di quante posizioni è “**sfasato**”
- **Sommiamo** tutte le differenze di posizione
- La somma finale sarà la distanza tra i profili!

# Misurare la distanza





# Word embedding

Esistono delle **rappresentazioni** già pronte:

- *glove*
- *Word2vec*
- ...

Si tratta di trasformazioni che **proiettano** le nostre parole in nuovo spazio

# Proprietà interessanti

Le parole più **vicine** a *San Francisco*:

- *Los Angeles*
- *Golden Gate*
- *Oakland*
- *California*
- *San Diego*
- *Pasadena*
- *Seattle*

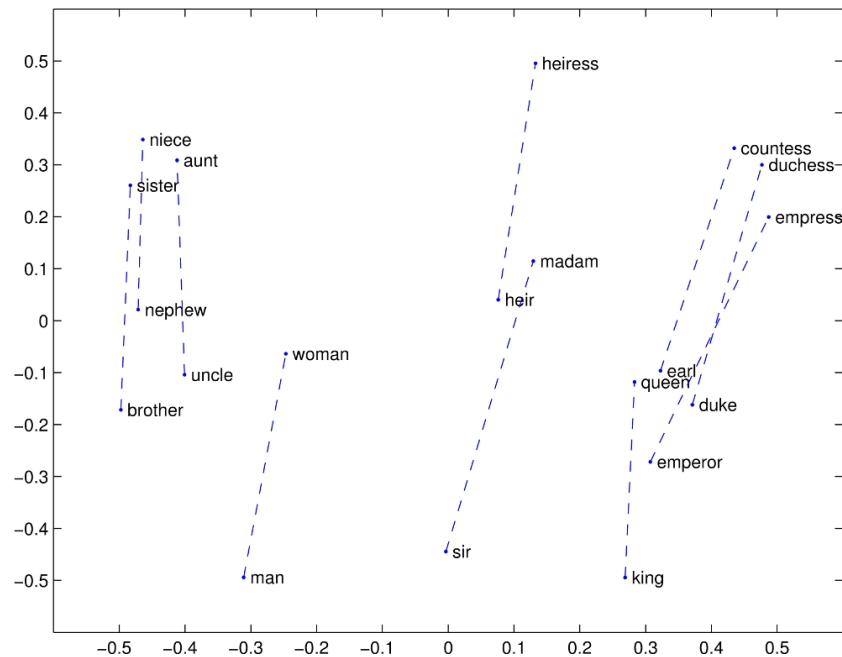
# Proprietà interessanti

Le parole più **vicine** a *France*:

- *Spain*
- *Belgium*
- *Netherlands*
- *Italy*
- *Switzerland*
- *Portugal*
- *Russia*

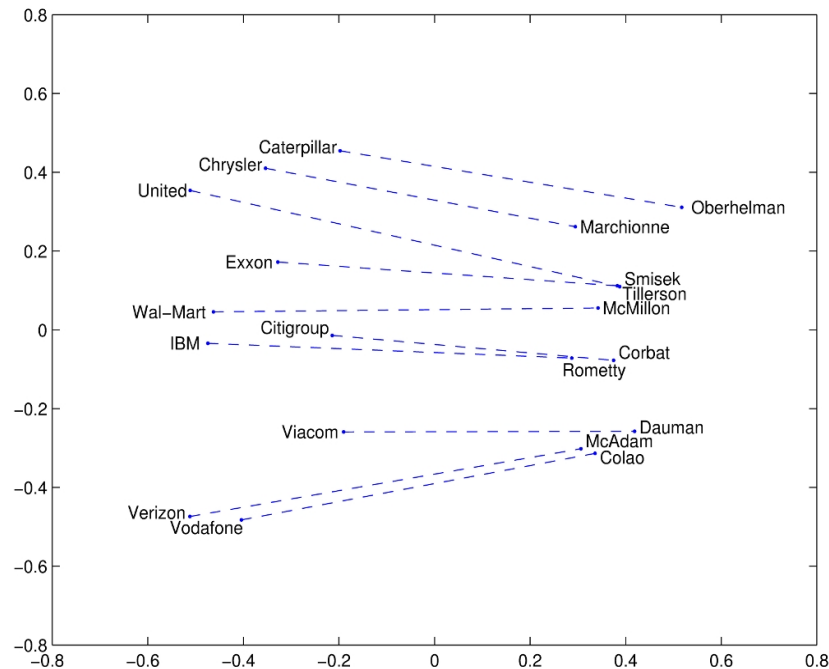
# Maschio-femmina

Le distanze tra coppie maschile-femminile sono molto **simili**



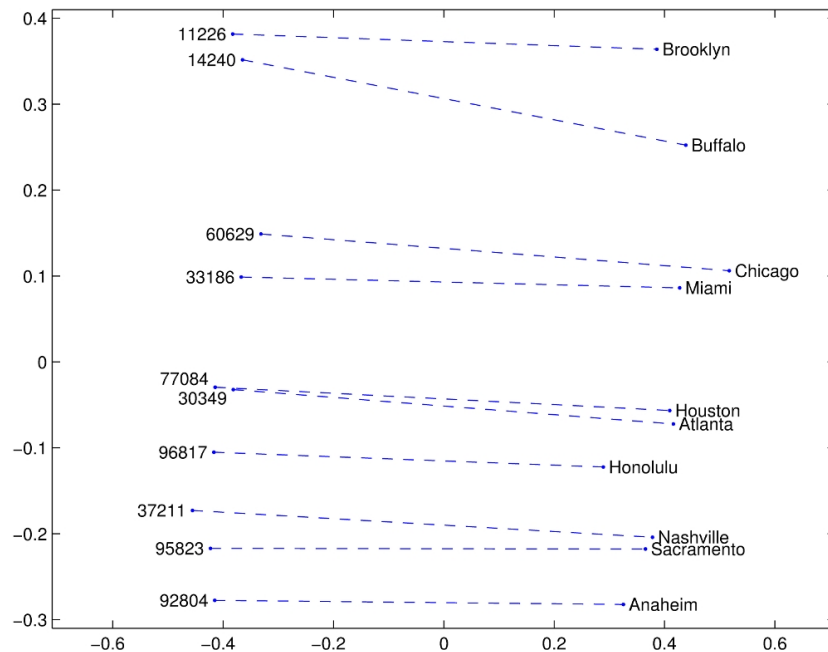
# Azienda-CEO

Anche le distanze tra coppie azienda-CEO sono molto **simili**



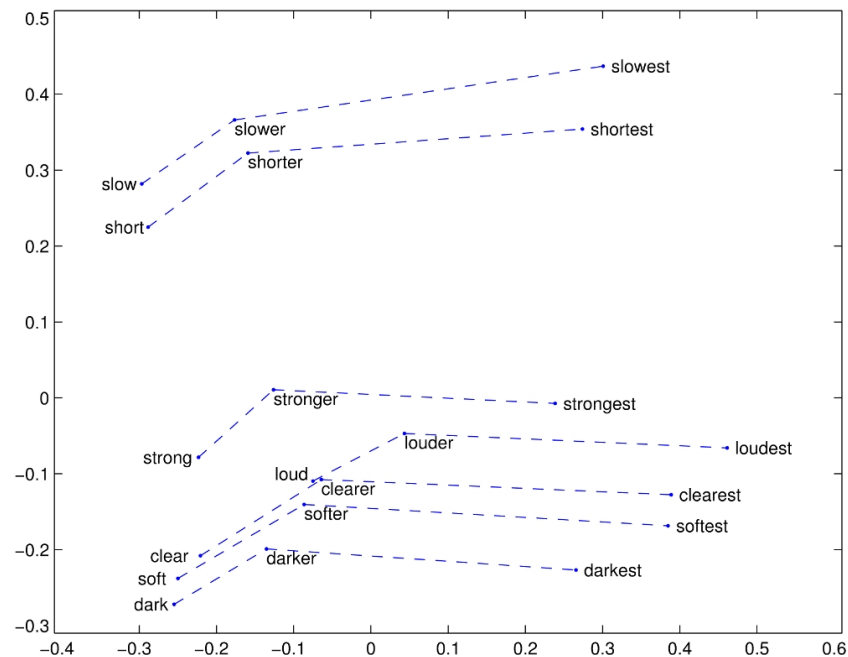
# Città-codice postale

Anche le distanze tra coppie città-codice postale sono molto **simili**



# Comparativo-superlativo

Anche le distanze tra coppie comparativo-superlativo (e base) sono molto **simili**



# Proprietà interessanti

Cosa succede se “**sommo**” due parole? Si tratta di **vettori**, posso farlo

*chinese + river*

Otengo un nuovo vettore, se cerco le parole più vicine trovo:

- *Yangtze river*
- *Yangtze*
- *Qiantang river*



# Risultato notevole

Operando sulle parole come **vettori**:

$$\textit{king} - \textit{man} + \textit{woman} = \textit{queen}$$

# Word embedding

Cenni dell'idea alla base:

- "Words that occur in the same **contexts** tend to have **similar** meanings" (Harris, 1954)
- Dobbiamo identificare il significato delle parole in base a quello che le **circonda**

*The **kid** said he would grow up to be Superman*

*The **child** said he would grow up to be Superman*

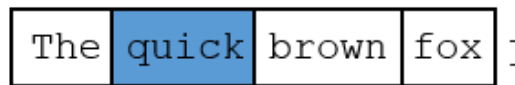
# Skip-gram

- Ho una **grande** quantità di testo
- Costruisco un vocabolario
- Addestro una **rete neurale**:
  - Input: 1 parola
  - Output: le parole **più probabili** come parole vicine

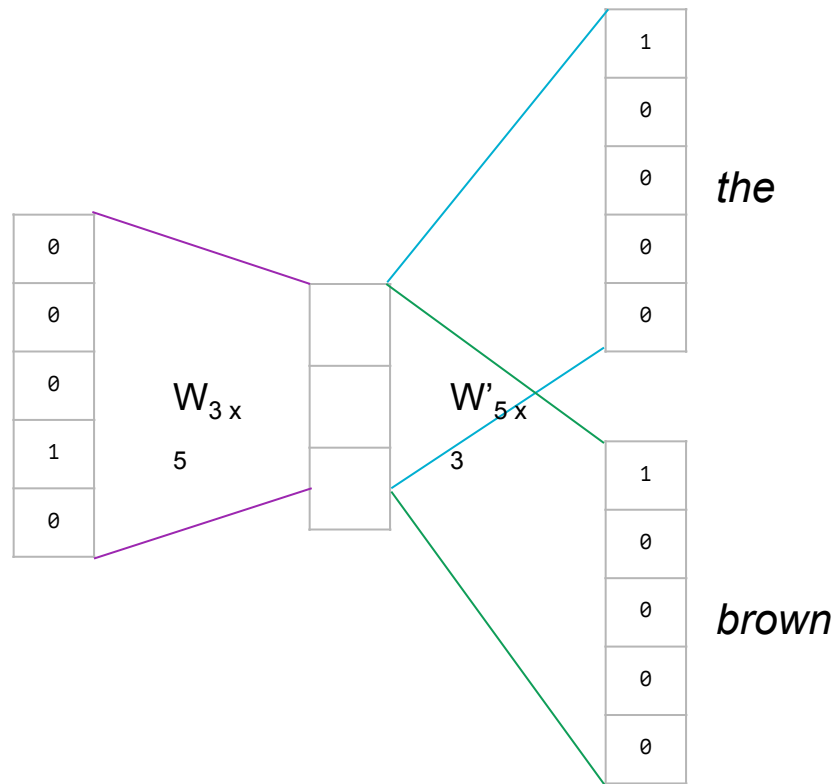
# Esempio di skip-gram

Source Text	Training Samples						
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. ➡	The	quick	brown	(the, quick) (the, brown)			
The	quick	brown					
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. ➡	The	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)		
The	quick	brown	fox				
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. ➡	The	quick	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)	
The	quick	brown	fox	jumps			
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. ➡	The	quick	brown	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
The	quick	brown	fox	jumps	over		

# Esempio di skip-gram



*quick*



# Continuous bag of words (CBOW)

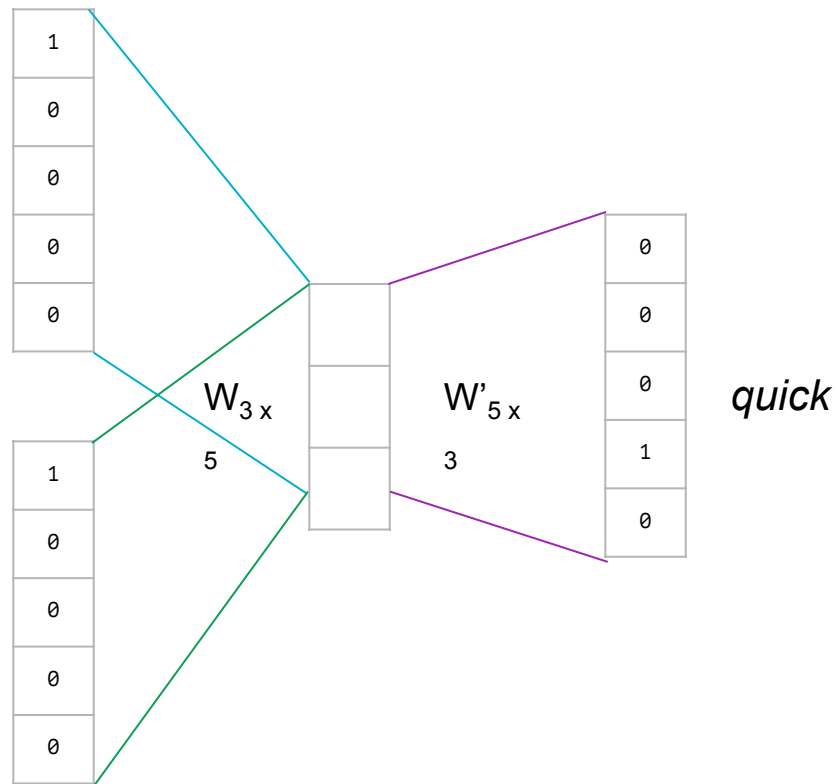
- Ho una **grande** quantità di testo
- Costruisco un vocabolario
- Addestro una **rete neurale**:
  - Input: più di 1 parola (**contesto**)
  - Output: la parola **più probabile** per quel contesto

# Esempio di CBOW

The	quick	brown	fox
-----	-------	-------	-----

*the*

*brown*



# Natural language generation

Cerco di sfruttare la possibilità di trovare il **contesto** di una parola:

- Prendo una parola a **caso** (o una sequenza di parole)
- Sfrutto una **rete neurale** per trovare la parola che segue **più probabile**
- Aggiungo la nuova parola alla mia sequenza e ricomincio

Per fare questa operazione ho bisogno di una rete che abbia **memoria**



# Esempio di NLG

*Ciao, mi chiamo*

*Ciao, mi chiamo Andrea*

*Ciao, mi chiamo Andrea e*

*Ciao, mi chiamo Andrea e sono*

*Ciao, mi chiamo Andrea e sono un*

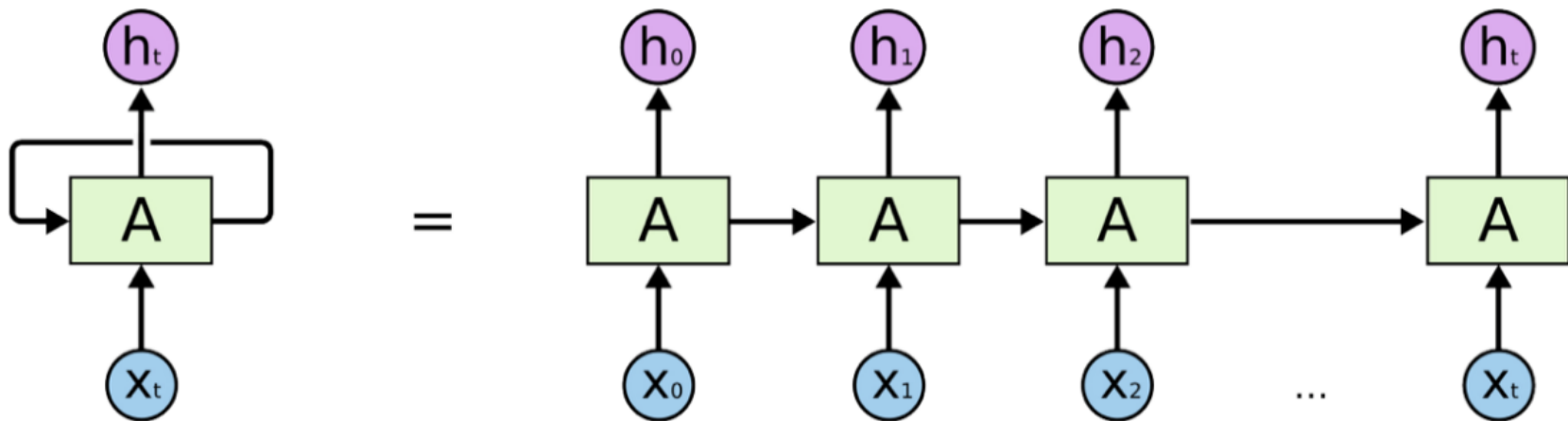
*Ciao, mi chiamo Andrea e sono un ricercatore*

# Reti neurali ricorrenti

Per NLG abbiamo bisogno di una rete neurale che abbia **memoria**:

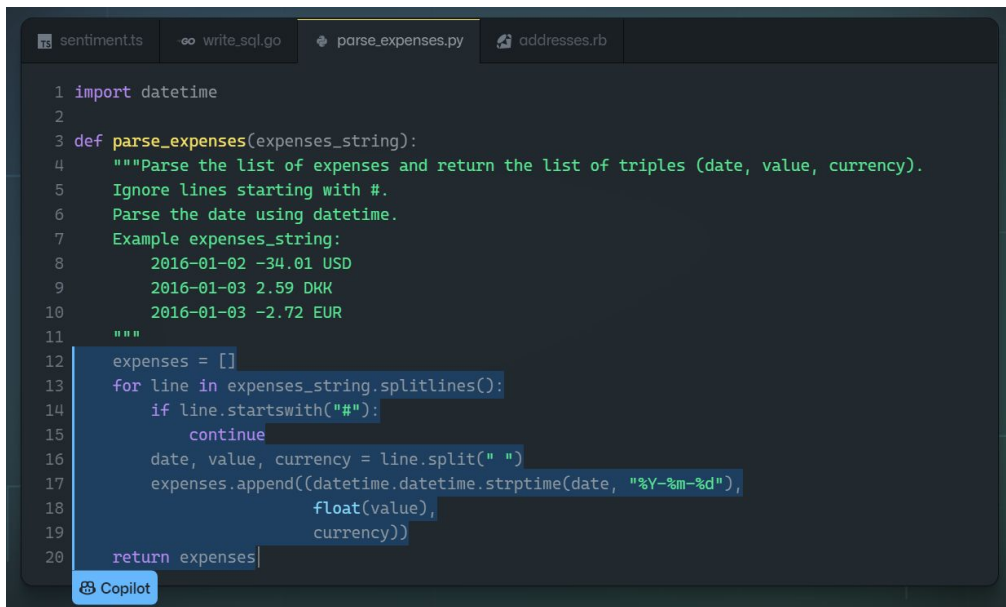
- Abbiamo visto reti che prendono come input **una** parola e danno un vettore
- Per generare un testo siamo interessati all'**ordine** e al **contesto**
- Abbiamo bisogno di ricordare l'ordine con cui sono apparse le parole

# Reti neurali ricorrenti



# Esempio di NLG: generazione di codice

<https://copilot.github.com/>



```
1 import datetime
2
3 def parse_expenses(expenses_string):
4     """Parse the list of expenses and return the list of triples (date, value, currency).
5     Ignore lines starting with #.
6     Parse the date using datetime.
7     Example expenses_string:
8         2016-01-02 -34.01 USD
9         2016-01-03 2.59 DKK
10        2016-01-03 -2.72 EUR
11    """
12    expenses = []
13    for line in expenses_string.splitlines():
14        if line.startswith("#"):
15            continue
16        date, value, currency = line.split(" ")
17        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
18                        float(value),
19                        currency))
20    return expenses
```

Copilot