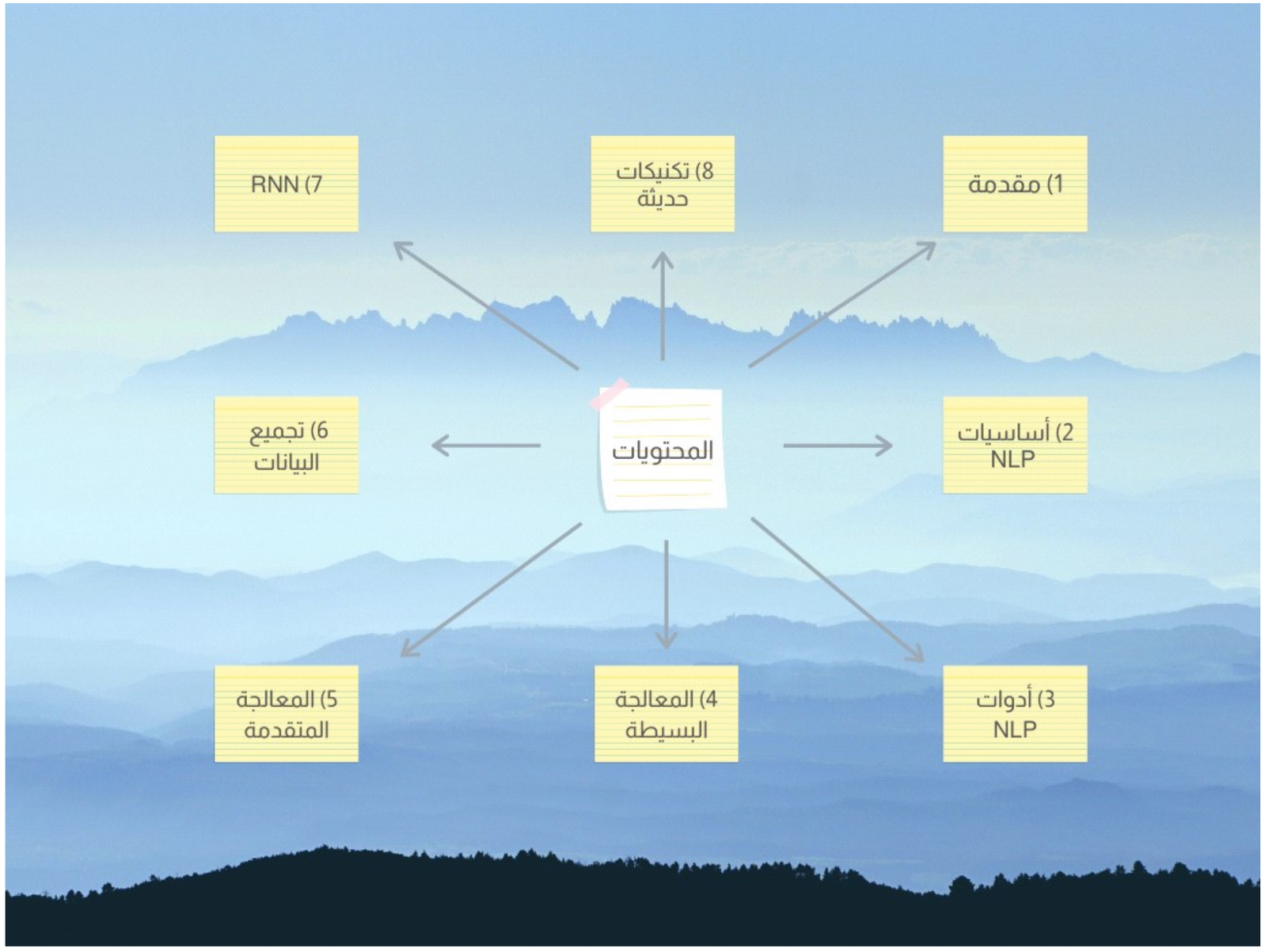
Natural Language Processing المعالجة اللغوية الطبيعية



المحتويات

1) مقدمة	المحتويات	ما هو NLP	تاريخ NLP	العقبات و التحديات	التطبيقات				
2) أساسيات NLP	المكتبات	الملفات النصية	ملفات pdf	البحث في النصوص					
3) أدوات NLP	Tokenization	Sent. Segm.	POS	Stem & Lemm	NER	Stopwords	Matchers	Syntactic Struc.	T.Visualization
4)المعالجة البسيطة	Word Meaning	Word embed	T. Vectors	Word2Vec	BOW	TF-IDF	Text Similarity	Dist. Similarity	
5)المعاجلة المتقدمة	T. Classification	T. Clustering	LDA	NMF	L. Modeling	GloVe	Lexicons	NGrams	T. Generation
	Sent. Analysis	Naïve Bayes	Vader	Auto Correct	Ans. Questions		Summarization & Snippets		
6)تجميع البيانات	Tweet Collecting	Data Scraping	Information Extraction		Information Retrieval		Relative Extraction		Search Engine
7) RNN	Seq to Seq	LSTM	GRU	Rec NN\TNN					
8)تكنيكات حديثة	Word Cloud	CNN	T. Forcing	Attention Model	Transformer	Bert	FastText	Gensim	Chat Bot

القسم الرابع : المعالجة البسيطة للنصوص

الجزء الخامس : Bag Of Words

==========

جعبة الكلمات Bag of Words

نتناول الآن هذه الأداة الهامة , التي كانت تستخدم في كثيرا من تطبيقات الـ NLP , لكنها الان اقل استخداما بسبب عيوبها

يقصد بها أن يتم تحديد مدي تواجد كل كلمة في النص , و مدي تكرارها , والذي قد يشير الي معني محدد لهذا النص

و هناك خطوتين أساسيتين لعمل BOW هي :

- سرد جميع الكلمات المستخدمة في النص (و يمكن عمل معالجة لها مثل عمل small او ازالة stopwrds)
- قياس مدي تواجد كل كلمة فيهم , في الجملة المختارة

-

فلو كان لديها عبارات من رواية "قصة مدينتين " لتشارلز ديكنز :

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness
it was the epoch of belief,
it was the epoch of incredulity,
it was the season of Light,
it was the season of Darkness,
it was the spring of hope,
it was the winter of despair,

فأولا يتم تحديد جميع الكلمات في النص , مع حذف التكرار و التي ستكون :

“it” “was” “the” “best” “of” “times” “worst” “age” “wisdom” “foolishness”

ثم نقوم بعمل مصفوفة لكل جملة علي حدة , لمعرفة مدي تواجد كل كلمة فيها , هكذا :

	“it”	“was”	“the”	“best”	“of”	“times”	“worst”	“age”	“wisdom”	“foolishness”
It was the best of times,	1	1	1	1	1	1	0	0	0	0
it was the worst of times,	1	1	1	0	1	1	1	0	0	0
it was the age of wisdom,	1	1	1	0	1	0	0	1	1	0
it was the age of foolishness,	1	1	1	0	1	0	0	1	0	1

و من هنا يمكن استخدام هذه المصفوفة لمعرفة المعني التقريبي للجملة , او لعمل تصنيف لها و تحديد هل هي مناسبة ام لا

و يكون هذا هو متجه كل جملة فيهم :

"it was the best of times" = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
"it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
"it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
"it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]

كما يمكن عمل NGram , اي ان يكون البحث و التقييم علي كلمتين معا , و ليس كلمة واحدة , مثل : look for , have to , والذي سيكون لها معني اقوي , ولكن قد تكون مضللة

كما يمكن عمل جميع الكلمات small اولا , و مسح كلمات التوقف , و حذف ال punctuation اولا للتسهيل

و بعد هذا يتم استخدام هذه الارقام لعمل خوارزم معين لتوقع نوع الجملة , او لعمل SA

علي ان من عيوبها عدم التركيز علي معاني الكلمات , والبطيء الكبير في العملية

لا تنس أن تكنيك CBOW هو امتداد للـ BOW لكن لا تعتمد علي كلمة واحدة , ولكن كلمات متعددة كما سنري

* *

و منها نتناول ما يسمي **vector space ,** او متجهات الكلمات

و هي الاداة التي تجعل الموديل يتمكن من فهم الكلمات و علاقاتها ببعضها البعض , خاصة في ظل سياقها و الجملة المحيطة , وهي مرتبطة بفكرة BOW بشكل ما

فلو كان لدينا جملتين :

Where are you from ?
Where are you going ?

فعلي الرغم من التشابه التام بين الجملتين باستثناء اخر كلمة , الا ان المعني مختلف تماما

و ايضا جملتي :

How old are you ?
What is your age ?

هنا يوجد اختلاف تام بين الجملتين , لكن المعني متطابق

كذلك جملة :

You need to buy these things before you sell products

فأداة VS ستتمكن من الربط بين كلمتي buy , sell لانهما معتمدتان علي بعضهما

_

ثم نأتي لمفهومين هامين وهو **word by word** و **word by doc**

والمفهوم الاول خاص بعلاقة الكلمات مع بعضها البعض و مدي تواجدها بالقرب من بعضها البعض , بينما المفهوم الثاني خاص بعلاقة الكلمة بالملف او المستند الذي يحتوي عليها

و مفهوم **word by word** يعتمد علي عدد مرات تواجد كلمة معينة , جوار كلمة معينة اخري , كي نقيس مدي تواجدهما معا . .

فمثلا كلمة "الحرب" , تتواجد غالبا بالقرب من كلمة "دبلوماسية" , او "السلاح" , او "ضحايا" ..

بينما كلمة "تضخم" , تتواجد بالقرب من كلمات مثل "العملة" , او "البنك" , او "الأزمة" . .

فلو كانت قيمة تواجد كلمة جديد منتشرة مثل "سيلفي" عالية مع كلمات مثل "جميل" , "مكياج" , "رائع", و قليلة مع كلمات مثل "علماء" , "فيزياء" , "علم" , فيمكن الفهم التقريبي لمعني هذه الكلمة , ويقاس نفس الأمر مع كلمة قد لا تكون مفهومة مثل "الأناركية" . .

لكن لاحظ ان هناك كلمات يمكن ان تسبب misleading مثل "الجامعة" , او "قمر"

و خطوات حسابها عبر تحديد رقم k و هو مدي اقتراب الكلمة المطلوبة من باقي الكلمات , وليكن 2 , ثم تحديد الكلمة المطلوبة و مثلا data , فيتم قياس الكلمات القريبة منها , وحساب عددها هكذا

Number of times they *occur together within a certain distance k*

I like simple data

I prefer simple raw data

k=2

	simple	raw	like	I
data	2	1	1	0

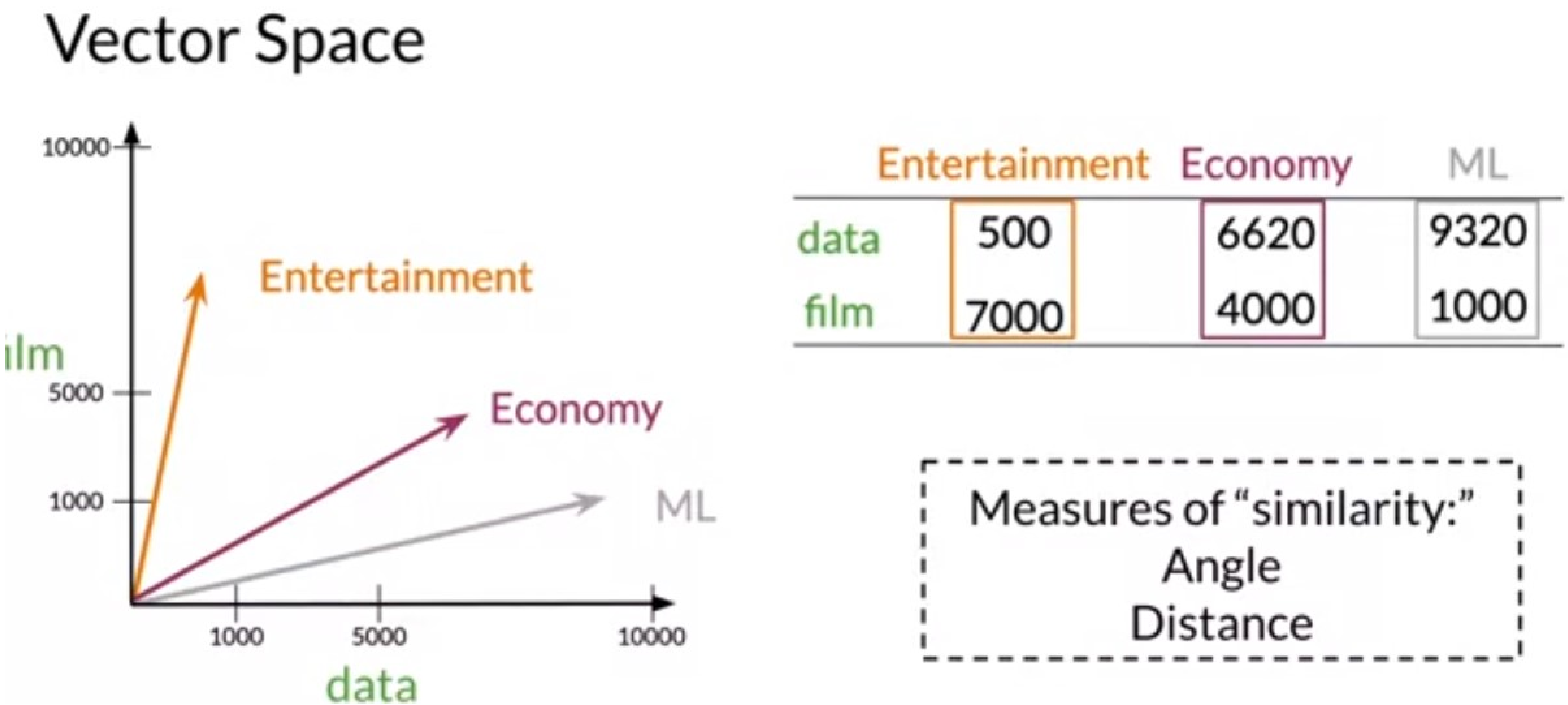
وبالتالي نستطيع بسهولة تحديد مدي علاقة كلمة ما بباقي الكلمات المحيطة بها , ويستخدم في الخوارزم المطلوب

_

المفهوم الثاني وهو word by doc و هو خاص بتواجد كلمة معينة في مستندات من موضوع معين , فمثلا يتم البحث عن كلمتي data , film في مستندات من انواع مختلفة , والبحث عن مدي تكرار كلا منهما هكذا :

	Entertainment	Economy	Machine Learning
data	500	6620	9320
film	7000	4000	1000

و بالتالي يتم رسم بياني يربط الكلمتين بعضهما البعض , بحيث يتم رسم خط من ال origin الي مدي تلاقي الرقمين معا , وهذا الامر يتمكن من رسم العلاقات بين المواضيع بعضها البعض , فموضوع economy & ML اقرب لبعضهما من ال entertainment و قد يتم ايضا حساب الزاوية بينهما , كما قمنا في محاضرة Text Vectors



و يمكن بالطبع توسيع الدائرة لتشمل عشرات المواضيع

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*

و يمكن تطبيقها بهذا الكود

```
import nltk
import re
import numpy
from nltk.corpus import stopwords
```

نقوم أولا بتنظيف الكلمات و حذف كلمات التوقف و عملها small

```
def word_extraction(sentence):
    words = re.sub("[^\w]", " ",  sentence).split()
    cleaned_text = [w.lower() for w in words if w not in stopwords.words('english')]
    return cleaned_text
```

ثم رد جميع التوكينز فيها بدون تكرار , وعمل ترتيب لها

```
def tokenize(sentences):
    words = []
    for sentence in sentences:
        w = word_extraction(sentence)
        words.extend(w)
    words = sorted(list(set(words)))
    return words
```

نقوم بالتجريب هنا

```
text = "Mary and Samantha arrived at the bus station early but waited \
until noon for the bus"

word_extraction(text)

tokenize(word_extraction(text))
```

و هنا الدالة التي تقوم بتناول الكلمات , و اعطاء رقم لكل كلمة فيها , ثم عمل bow للجملة

```
def generate_bow(allsentences):
    vocab = tokenize(allsentences)
    print("Word List for Document \n{0} \n".format(vocab))
    for sentence in allsentences:
        words = word_extraction(sentence)
        bag_vector = numpy.zeros(len(vocab))
        for w in words:
            for i,word in enumerate(vocab):
                if word == w:
```

```
                bag_vector[i] += 1
        print("{0}\n{1}\n".format(sentence,numpy.array(bag_vector)))
        print('-----------------------------------------------')
```

تجربة علي كلمات

```
generate_bow(word_extraction(text))
```

تجربة علي جمل

```
allsentences = ["Joe waited for the train",
               "The train was late",
               "Mary and Samantha took the bus",
              "I looked for Mary and Samantha at the bus station",
               "Mary and Samantha arrived at the bus station early but waited until noon for the bus"]

generate_bow(allsentences)
```

مع العلم ان هذا الكود يمكن ان يتم استبداله باداة vectorizer في sklearn

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(allsentences)

for i in range(len(allsentences)) :
    print(allsentences[i])
    print(list(X.toarray()[i]))
    print('-----------------------------------------------')
```

_

و يتم هذا الأمر في اللغة العربية بنفس الكفاءة دون تغيير

```
text = "ذهب محمد و شريف الي المدرسة هذا اليوم"

word_extraction(text)

tokenize(word_extraction(text))
```

```
generate_bow(word_extraction(text))

allsentences = ["ذهب محمد الي الجامعة",
                "نجحت مني في الاختبار",
                "تمكن محمد من السفر هذا اليوم",
                "اعتقد ان جورج سيتصل بنا اليوم",
                "لا اظن ان الاختبار سيكون سهلا"]

generate_bow(allsentences)
```

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(allsentences)

for i in range(len(allsentences)) :
    print(allsentences[i])
    print(list(X.toarray()[i]))
    print('----------------------------------------------')
```