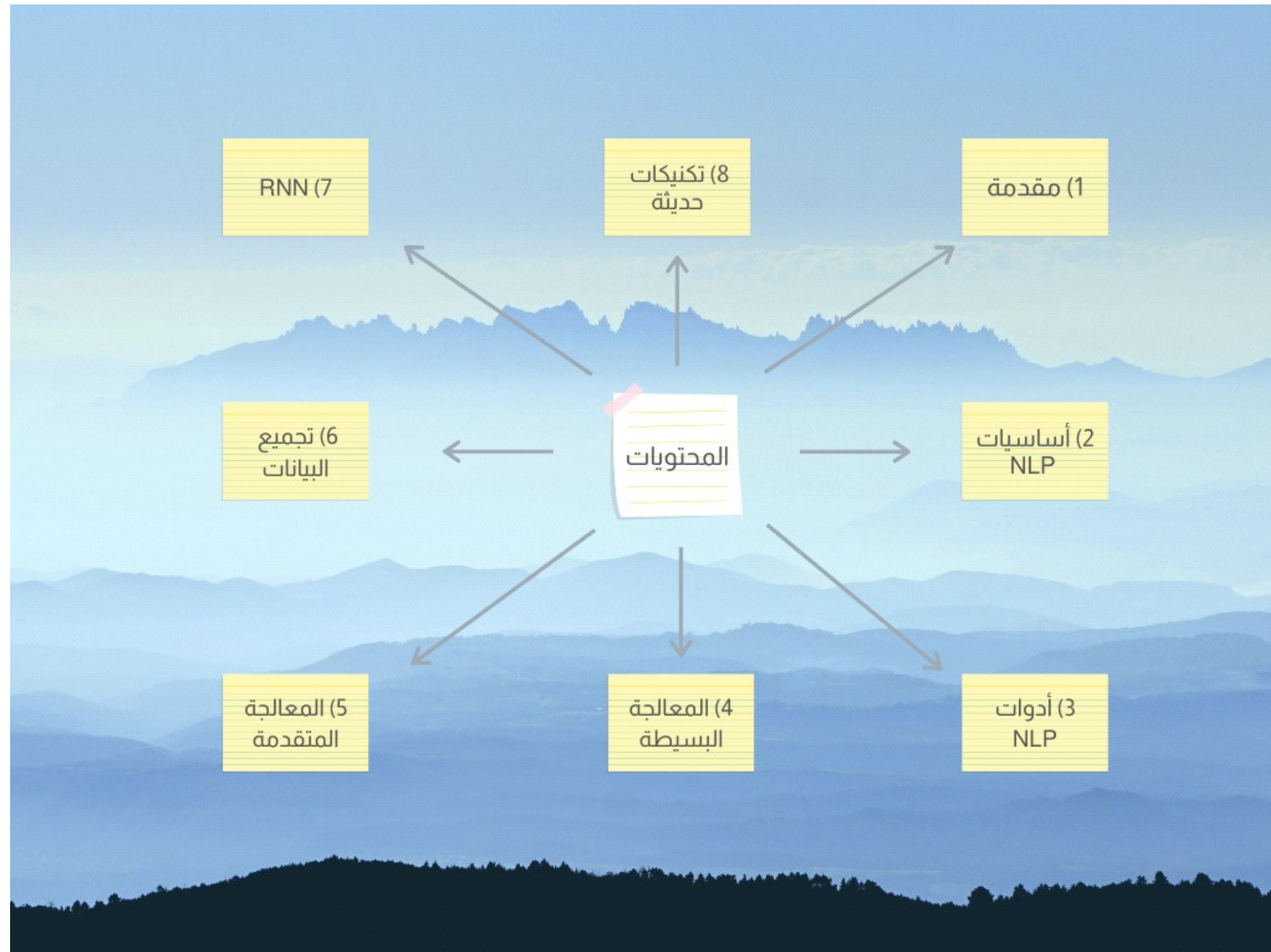


NATURAL LANGUAGE PROCESSING

المعالجة اللغوية الطبيعية



المحتويات

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

القسم الرابع : المعالجة البسيطة للنصوص

الجزء السادس : TF-IDF

=====

و هي طريقة بالغة الأهمية في NLP , وتستخدم بكثير في text classification و Sentimental Anslysis

وهي تعد تطوير شامل لفكرة BOW حيث تعتمد علي مدي تواجد كلمات معينة في النصوص , ولكن بطريقة اكثر احترافية ,
ولتجنب العيوب الموجودة في BOW

و يمكن أن نتناول تطور الفكرة منذ البداية . .

* * * * *

فلو كان لدينا عدد من روايات شكسبير , وتم ذكر عدد من أبطالها , فيمكن عمل BOW بينها بهذه الطريقة

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

و هي معتمدة علي الفكرة الخاصة بسؤال : هل تتواجد كلمة معينة او لا تتواجد في المستند

و لكن هل يكفي ان نعلم اذا كانت الكلمة موجودة ام لا ؟ , يمكن ان نقوم بعمل تطوير للفكرة , بحيث تحتوي علي عدد مرات تواجد الكلمة هكذا :

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

و لكن من مشاكل فكرة حساب عدد مرات التكرار انها لا تهتم بالموضع , فمثلا جملة : (فرنسا دولة غنية بينما ألمانيا يتراجع اقتصادها) , تتساوي بالضبط مع جملة (ألمانيا دولة غنية بينما فرنسا يتراجع اقتصادها) , وهذا غير منطقي , لذا فالحل في فكرة الترقيم الموضوعي التي ستحل هذه المشكلة

و قانونها هو :

$$(1 + \log \text{tf}_{t,d})$$

حيث TF هي : حاصل قسمة عدد مرات تكرار الكلمة المطلوبة , علي العدد الكلي للكلمات في الملف

$$\text{TF (term)} = \frac{\text{Number of times term appears in a document}}{\text{Total number of terms in the document}}$$

و كلمة TF-IDF , هي اختصار : Term Frequency-Inverse Document Frequency , هي تعتمد علي قيمتين هامتين , أولا , Term Frequency, اي معدل تكرار الكلمة المطلوبة , و ثانيا Inverse Document Frequency و الذي يشير الي مدي ندرة او شيوع هذه الكلمة , ذكرنا الآن القيمة الأولى TF , ماذا عن الثانية ؟ ؟

القيمة الثانية تدل علي مدي شيوع الكلمة وهي تتناسب عكسيا مع قوة الكلمة , فلاحظ ان الكلمة نفسها اذا كانت منتشرة و شائعة بين النصوص كلها , وزاد استخدامها في جميع المستندات الأخرى فهذا يجعلها ذات تأثير ضعيف في نفسها

أي أن أي كلمة تنتشر في العديد من النصوص و المستندات ذات المعاني المختلفة , فهذه الكلمة غالبا تكون اقل قيمة و تأثير في المعني , فكلمة (جدا , نعم , لا , يزيد , ينقص , . .) هي كلمات شائعة ذات معني عام و لن يكون لها تأثير في تحديد نوع النص

بينما الكلمات المحددة التي يندر وجودها الا في معاني معينة مثل (استثمار , فوائد , بكتيريا , برمجة , شحم) , فهذه الكلمات لا تتواجد الا في عدد محدود من المستندات , وبالتالي تكون بمعني اقوي

فالكلمات ذات التكرار الاكثر في الملفات , قد يكون لها تأثير اكبر بالفعل , ولكن في حالة كانت هذه الكلمة هي في الاساس نادرة الاستخدام

فكلمة مثل : الذهب , هي كلمة غير منتشرة في عموم الكلام , و بالتالي اذا استخدمت عدد من المرات في ملف ما , فهذا يعني شئ ايجابي

بينما كلمة مثل : بالفعل , هي كلمة منتشرة في الكثير و الكثير من الملفات , و بالتالي اذا تم استخدامها كثيرا في ملف ما , فهذا لا يعني شئ معين

و بالتالي نريد ان نحسب مدي انتشار و استخدام الكلمات علي نطاق واسع , فالكلمات ذائعة الشهرة , يكون لها وزن اقل , والكلمات قليلة الاستخدام يكون لها وزن أعلى

* * * * *

و كيف يمكن حسابها ؟ ؟

$$\text{idf}_t = \log_{10} (N/\text{df}_t)$$

و هذا يتم عبر حساب رقمين

أولا N وهو العدد الكلي للملفات التي لدينا , وثانيا رقم df وهو عدد الملفات التي ذكرت فيها الكلمة المطلوبة

و النموذج يسمى العدد العكسي للملفات idf لأنه يعتمد عكسيا علي مدي انتشار الكلمة في الملفات , لذا فإننا نقوم بقسمة N علي df , و نقوم بعمل لوغاريتم لها

و اذا كانت الكلمة نادرة الاستخدام , وظهرت في عدد قليل من المستندات , فسيكون اللوغاريتم لرقم كبير (مليون مثلا) و لها قيمة كبيرة 6 , اما اذا ظهرت في عدد كبير من المستندات و ليكن نصفها , فسيكون لوغاريتم 2 وهو رقم قليل , و اذا ظهرت في كل الملفات , فسيكون لوغاريتم 1 اي 0

term	df _t
calpurnia	1
animal	100
sunday	1,000
fly	10,000
under	100,000
the	1,000,000

و من استخدامات هذا الامر , هو تحديد اهمية كلمات البحث , فلو كان لدينا جملة مثل try to buy an insurance

فيجب قبل ان يبدأ جوجل في البحث , ان يحدد اي الكلمات ذات وزن اكبر , فلو ظهرت كلا الكلمتين في ملفات معينة 10 الاف مرة , لكن كلمة insurance كان لها الظهور في 4 الاف ملف , وكلمة try في 8 الاف , فهذا يعني ان insurance اقل انتشار , فهي اهم و اقوي في الوزن

لذا فمعادلة حساب قوة الكلمة تعتمد علي زيادة تواجدها , ولكن أيضا علي قلة عدد المستندات التي تتواجد فيها , لذا فاسمها term frequency اي تكرار الكلمة , و لكن inverse document frequency اي معكوس عدد المستندات , او بالقسمة عليها , و بالتالي معادلة IDF هي :

$$\text{DF (term)} = \frac{\text{d(Number of documents containing a given term)}}{\text{D(the size of the collection of documents)}}$$

$$\text{IDF (term)} = \log \left[\frac{\text{Total number of documents}}{\text{Number of documents with a given term in it}} \right]$$

و أحيانا يتم إضافة رقم 1 إلى الكسر , لتجنب الحصول علي قيمة $\log 1$ و التي تساوي 0
فإذا كانت الكلمة منتشرة في جميع المستندات , يكون الكسر يساوي 1 , وتكون قيمة IDF هي $\log 2$ اي 0.3
اما لو كانت موجودة في نصف عدد المستندات , تكون القيمة 0.47 , واذا كانت موجودة في 1% من المستندات تكون القيمة
2 , وهكذا , و في النهاية يتم ضرب قيمة TF في IDF

* * * * *

الآن إلى المعادلة النهائية , للجمع بين TF و IDF

فالصيغة النهائية للجمع بين الميزتين كلاهما هي هكذا

$$W_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

و هي تعتمد علي أن الكلمة يكون لها وزن أكبر حينما :

• يزداد تواجدها في الملف

• يقل انتشارها في باقي الملفات

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

و يتم تقييم العلاقة بين ال query و ال document بمجموع جميع الكلمات المتقاطعة بينهما بهذا الأمر

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

مثال :
لدينا عدد من الجمل هي :

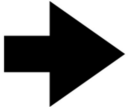
- . *The sky is blue.*
- . *The sun is bright today.*
- . *The sun in the sky is bright.*
- . *We can see the shining sun, the bright sun.*

نقوم أولا بحساب قيمة TF هكذا :

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t'} f_{t',d}}$$

$f_{t,d}$

	blue	bright	can	see	shining	sky	sun	today
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	1
3	0	1	0	0	0	1	1	0
4	0	1	1	1	1	0	2	0



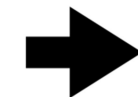
	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

ثم نقوم بحساب قيمة IDF هكذا :

$$\text{idf}(t, D) = \log_{10} \frac{N}{n_t}$$

$$f_{t,d}$$

	blue	bright	can	see	shining	sky	sun	today
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	1
3	0	1	0	0	0	1	1	0
4	0	1	1	1	1	0	2	0
n _t	1	3	1	1	1	2	3	1



$N = 4$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

$\log_{10} \frac{4}{1} = 0.602$
 $\log_{10} \frac{4}{3} = 0.125$

أخيرا يتم ضرب القيمتين معا , لتحديد الكلمات ذات الأهمية الأعلى , وقيمة كل كلمة فيها

$\text{tf}(t, d)$

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

x

$\text{idf}(t, D)$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$



	blue	bright	can	see	shining	sky	sun	today
1	0.301	0	0	0	0	0.151	0	0
2	0	0.0417	0	0	0	0	0.0417	0.201
3	0	0.0417	0	0	0	0.100	0.0417	0
4	0	0.0209	0.100	0.100	0.100	0	0.0417	0

- TF-IDF: Multiply TF and IDF scores, use to rank importance of words within documents
- Most important word for each document is highlighted

كما أن أداة TF-IDF تستخدم لتهيئة البيانات النصية قبل عمل خوارزم توقع او تصنيف , عبر الخطوات التالية :

- سرد جميع الكلمات في الملف
- تحديد مدي تواجد كل كلمة في الجملة المختارة
- حساب قيمة tf-idf ووضعها في مكانها المحدد مع الجملة

	userId	4	5	10	14	15	18	19	26	31	34	...	283199	283204	283206	283208	283210	283215	283219	283222	283224	283228
movieId																						
1	4.0	0.0	5.0	4.5	4.0	0.0	0.0	0.0	0.0	5.0	0.0	...	5.0	0.0	0.0	4.5	0.0	4.0	4.0	0.0	0.0	4.5
2	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	4.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0

و هنا تكون هذه القيم هي features جاهزة لاي موديل

وهنا الكود الخاص بتطبيق TF-IDF

نقوم اولاً بتحميل بانداس

```
import pandas as pd
```

مثال جملتين

```
documentA = 'the man went out for a walk'  
documentB = 'the children sat around the fire'
```

فصل الكلمات

```
bagOfWordsA = documentA.split(' ')  
bagOfWordsB = documentB.split(' ')
```

عمل BOW

```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))  
uniqueWords
```

إضافة الكلمات

```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
```

```
for word in bagOfWordsA:  
    numOfWordsA[word] += 1  
numOfWordsA
```

```
numOfWordsB = dict.fromkeys(uniqueWords, 0)
```

```
for word in bagOfWordsB:  
    numOfWordsB[word] += 1
```

```
numOfWordsB
```

حساب TF

```
def computeTF(wordDict, bagOfWords):  
    tfDict = {}  
    bagOfWordsCount = len(bagOfWords)  
    for word, count in wordDict.items():  
        tfDict[word] = count / float(bagOfWordsCount)  
    return tfDict
```



```
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfA
```

```
tfB = computeTF(numOfWordsB, bagOfWordsB)
tfB
```

حساب IDF

```
def computeIDF(documents):
    import math
    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
```

```
return idfDict
```

```
idfs = computeIDF([numOfWordsA, numOfWordsB])  
idfs
```

```
def computeTFIDF(tfBagOfWords, idfs):  
    tfidf = {}  
    for word, val in tfBagOfWords.items():  
        tfidf[word] = val * idfs[word]  
    return tfidf
```

```
tfidfA = computeTFIDF(tfA, idfs)  
tfidfA
```

```
tfidfB = computeTFIDF(tfB, idfs)  
tfidfB
```

حساب TF-IDF

و يمكن استخدام sklearn لتقوم بنفس المهمة

```
df = pd.DataFrame([tfidfA, tfidfB])
df.head()
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform([documentA, documentB])
print(vectors)
```

```
feature_names = vectorizer.get_feature_names()
feature_names
```

```
dense = vectors.todense()  
dense
```

```
denselist = dense.tolist()
df = pd.DataFrame(denselist, columns=feature_names)
df.head()
```

* * * * *

كما يمكن استخدام اداة tf-idf لعمل sparse matrix لتهيئة البيانات لخوارزم ML

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
data = pd.read_csv('Questions.csv', sep=',')
data.head()
```

```
X = data['Title']
y = data['Score']
```

```
VecModel = TfidfVectorizer()
X_Vec = VecModel.fit_transform(X)
X_Vec = pd.DataFrame.sparse.from_spmatrix(X_Vec)
```

```
print(f'The new shape for X is {X_Vec.shape}')
X_Vec.head()
```

```
for i in range(5):
    print(sorted(list(X_Vec.iloc[i,:]),reverse = True)[:20])
    print('-----')
```

```
X_train, X_test, y_train, y_test = train_test_split(X_Vec, y, test_size=0.25, random_state=42)
```

* * * * *

و تدعم أداة tf-idf اللغة العربية بكفاءة كاملة