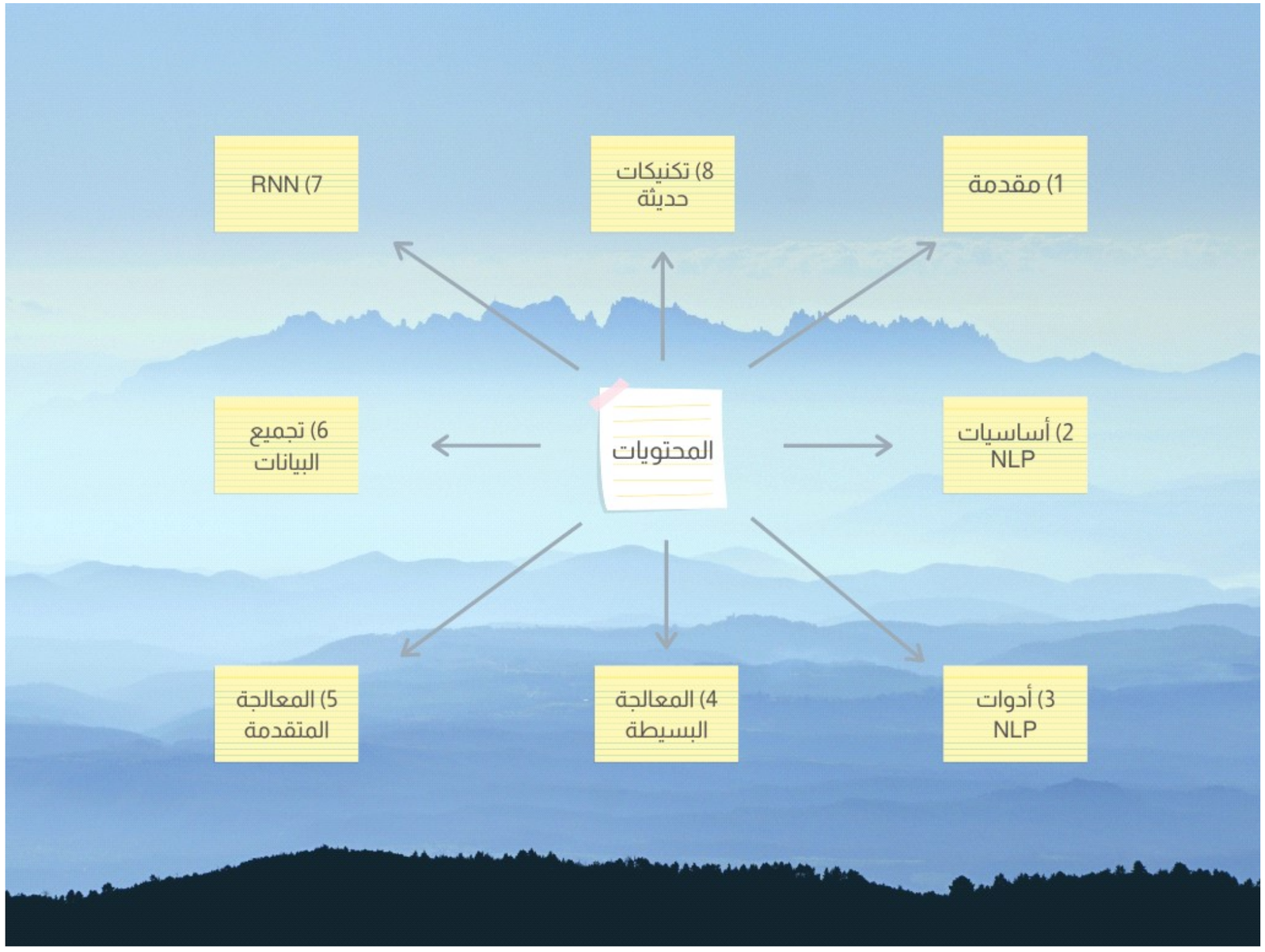
Natural Language Processing المعالجة اللغوية الطبيعية



المحتويات

القسم الخامس : المعالجة المتقدمة للنصوص

الجزء الثالث : Latent Dirichlet Allocation

==========

وهو خوارزم له القدرة علي تصنيف النصوص دون ان يكون لها قيمة label , اي انها تتناول كمية كبيرة من النصوص الـ unsupervised و تقوم بتصنيفها , دون ان نقوم بتدريبه علي ان هذه كلمات ايجابية و سلبية

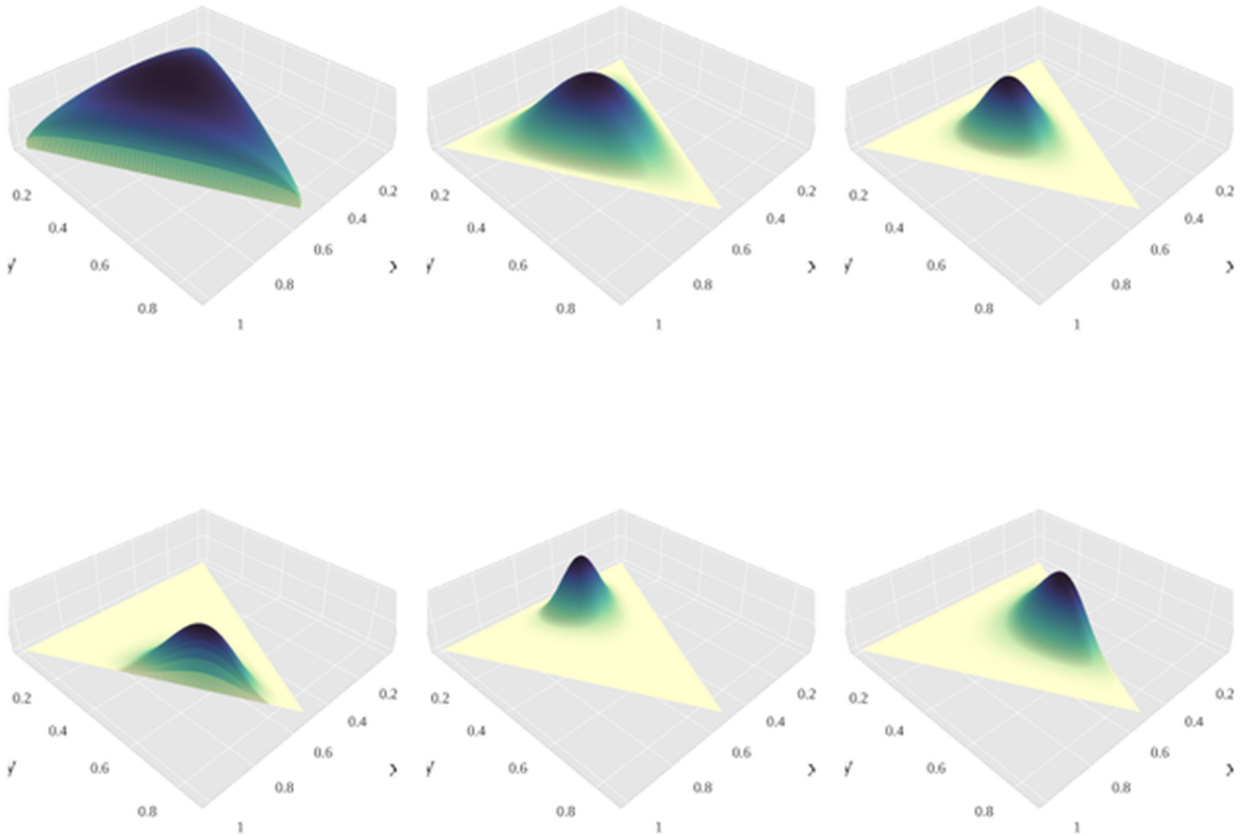
و هذا اصلا هو اغلب التعامل مع النصوص , اذ ان الطبيعي ان النصوص التي ستاتي للخوارزم ستكون غير معنونة ,

و النموذج هو اختصار كلمات Latent Dirichlet Allocation و التي تختصر LDA او احيانا تسمي Topic Modeling اي نمذجة المواضيع

و علينا ان نتجنب الخلط بينها و بين Linear Discriminant Analysis , والتي لا علاقة لها بالـ NLP

-

وطريقة LDA هي مبنية علي فكر الرياضي الالماني جون دريشليت , كما ان هناك توزيع رياضي احصائي باسمه Dirichlet Distribution , وثم تم ابتكارها مؤخرا في 2003 بينما هو كان في القرن التاسع عشر



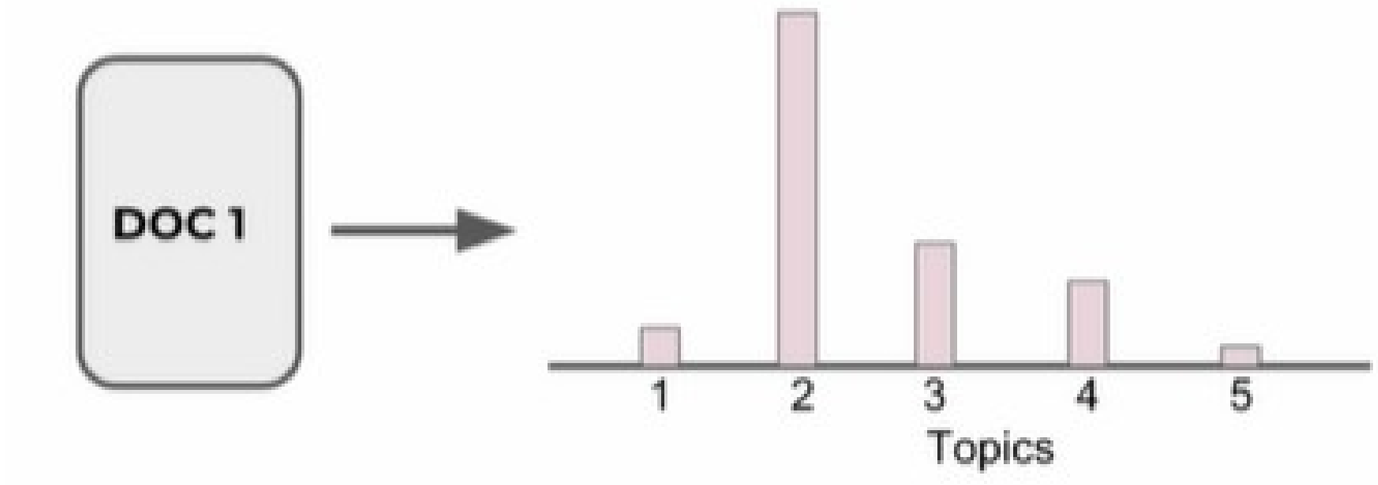
و قبل التعامل مع LDA علينا ان نتعرف علي اساسيات لابد منها

- هناك ما يسمي latent topics اي المواضيع الكامنة , اي المواضيع الموجودة داخل مقال معين , فقد يحتوي مقال معين علي 10 مواضيع كامنة

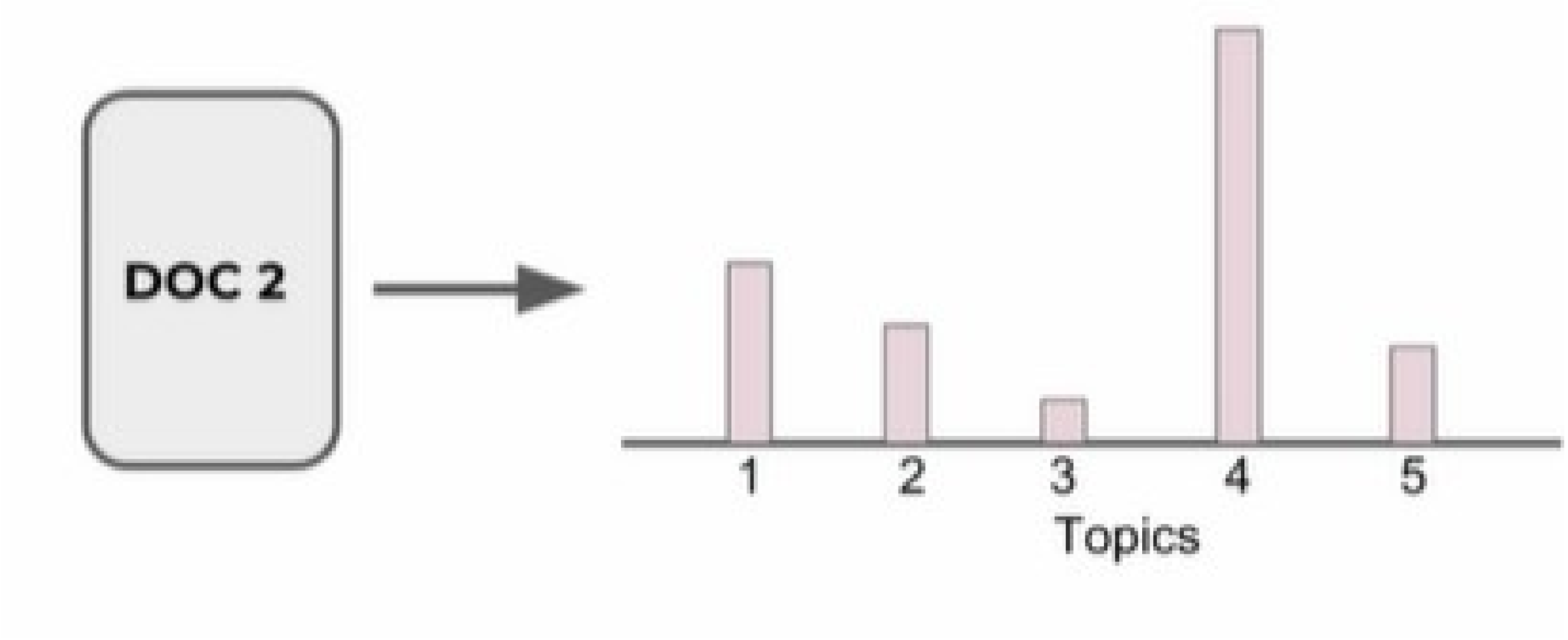
- ان المقالات المتشابهة تستخدم كلمات متشابهة الي حد ما , فاغلب المقالات عن الاقتصاد تستخدم كلمات متشابهة و هكذا

- ان هناك توزيع طبيعي للمواضيع الكامنة , داخل المقال الكامل

فمثلا هنا نري ان نوع الموضوع الكامل رقم 2 له اكبر نسبة

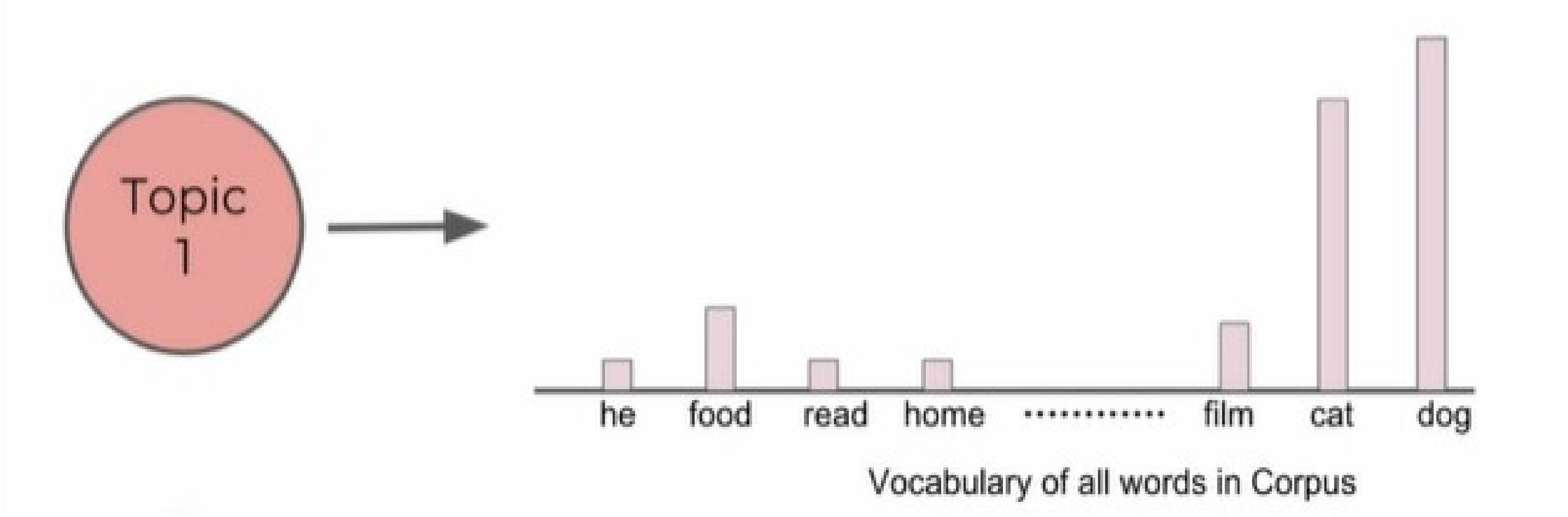


و هنا ان النوع الرابع هو له النسبة الاكبر



و بالتالي عبر استعراض النسب للمواضيع داخل مقال معين , نستطيع ان نستنتج نوع المقال كله

ففي هذا الموضوع , نري ان اغلب الكلمات المستخدمة هي cat , dog فيمكننا ان نستنتج ان هذا المقال عن ال pets و هكذا



_

فتكون الخطوات التي تقوم LDA بعملها كالتالي :

- تتناول LDA المقال و تقوم بفحص المواضيع , ثم الكلمات فيه
- تحديد العدد الكلي للكلمات في المقال N
- معرفة توزيع نسب المواضيع في المقال بالكامل , فممكن ان يكون مثلا : 60% بزنس , 20 % سياسة و هكذا , ويكون هذا لأعلي عدد من المواضيع , مثلا اعلي 5 , وليس للجميع
- ثم علي اساس نسب التوزيع , تقوم LDA بعمل تصنيف clustering باسلوب unsupervised , بحيث تقوم بتقسيم المقالات او الجمل المحددة , لعدد من الاقسام , يتم اعطائها

إذن الـ LDA تقوم مكان طريقة Kmeans لكن في النصوص , حيث تتناول كميات من المواضيع , وتقوم بتصنيفها لـ clusters حسب تواجد الكلمات المتشابهة فيها

_

التطبيق العملي للـ LDA

أولا يوجد هنا رابط لشاشة تفاعلية, يمكن فيها تطبيق خوارزم LDA علي نصوص بسيطة . .

https://lettier.com/projects/lda-topic-modeling/

_

و هنا الكود الخاص به في بايثون . .

يبدأ الأمر بقراءة ملف البيانات

```
import pandas as pd
npr = pd.read_csv('npr.csv')
npr.head()
```

وهو عبارة عن ملف ضخم , فيه حوالي 12 الف مقال , كل مقال يتكلم عن موضوع معين , وبالطبع هناك تشابه في المواضيع , ونريد ان نقوم بتقسيمها لعدد من الاقسام المتجانسة حسب تواجد الكلمات المتشابهة

فنقوم اولا بقراءة المقالات باسلوب countvectorizer للحصول علي مصفوفات الكلمات

مع العلم انه يفضل تحديد قيم مناسبة للبراميتر اثناء عمل الاوبجكت , واهمهم max_df وهي تشير الي maxmum document frequency اي الحد الاقصي من تواجد كلمة معينة , وغالبا ما يتم وضع رقم في حدود 0.9 حتي يتم استبعاد الكلمات المنتشرة جدا مثل then (مالم تكون موجودة في كلمات التوقف)

كذلك min_df حتي يتم استبعاد الكلمات الشاذة التي يندور وجودها .

مع العلم ان القيمتين يمكن اعطاء نسبة او رقم محدد

ثم اخباره ان كلمات التوقف هي تابعة للغة الانجليزية ليستخدمها و يستبعدها

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')
dtm = cv.fit_transform(npr['Article'])
```

و يمكن استعراض الكلمات التي تم اختيارها هنا و التي هي مجموع كل الكلمات في ال12 الف مقالة مع حذف التكرار

```
len(cv.get_feature_names())
```

كما يمكن استعراض عدد من الكلمات بشكل عشوائي

```
import random
for i in range(10):
    random_word_id = random.randint(0,54776)
    print(cv.get_feature_names()[random_word_id])
```

بعد ان تم عمل مصفوفة dtm يتم استخدام LDA , واعطاءه عدد الكلسترز المطلوب و ليكن 7

```
from sklearn.decomposition import LatentDirichletAllocation
LDA = LatentDirichletAllocation(n_components=7,random_state=42)
LDA.fit(dtm)
```

الان تم معرفة ان هناك سبع اقسام , وستكون موجودة في الاتريبيوت components_ هكذا

```
len(LDA.components_)
```

حيث ان components_ هي مصفوفة 7 صفوف , و 54 الف عمود , حيث ان كل صف فيهم هو احد الكلاسترز , وكل عمود هي قيم الكلمات الموجودة فيه ,

```
LDA.compon
```

ويمكن تناول النوع الاول مثلا واعطاء امر argsort وهو الذي يقوم باظهار قيم الاندكس للكلمات الاقل انتشارا ثم الاكثر انتشارا , بالترتيب

```
single_topic = LDA.components_[0]
single_topic.argsort()
```

فهذا معناه ان الكلمة ذات اندكس 2475 هي اقلهم انتشارا , وان الكلمة رقم 42993 هي اكثرها انتشارا

و اذا اردنا عرض اكثر 10 كلمات انتشارا في الكلاستر الاول

```
single_topic.argsort()[-10:]
```

و يمكننا اظهارها عبر استخدام ميثود get_feature-names

```
for index in single_topic.argsort()[-10:]:
    print(cv.get_feature_names()[index])
```

وهو ما يدل علي ان الكلاستر الاول قد يكون سياسي

و يمكننا عرض اهم كلمات في كل كلاستر هكذا :

```
for index,topic in enumerate(LDA.components_):
    print(f'THE TOP 15 WORDS FOR TOPIC #{index}')
    print([cv.get_feature_names()[i] for i in topic.argsort()[-15:]])
    print('\n')
```

الان سنقوم بتطبيق هذا التقسيم علي كل الجمل الـ 12 الف , ويكون عبر امر transform للاوبجكت LDA

```
topic_results = LDA.transform(dtm)
```

لا تنس ان dtm هي ناتج تطبيق countvectorizer علي المقالات ال 12 الف , وهي التي بها 12 الف صف في 54 الف عمود ,باسلوب one-hot-encoder

و هنا تكون نتيجة الترانسفورم , هي مصفوفة 12 الف صف في 7 اعمده , وذلك لانها قامت بعمل سوفتماكسعلي القيم السبعة , فجائت بنسب مئوية لكل قسم فيهم , وهو يعتبر نسبة تواجد هذه المقال في اي قسم من الاقسام السبعة

```
topic_results.shape
```

فمثلا المقال الاول نتيجته هي :

```
topic_results[0].round(2)
```

اي انه باحتمال 68% ان يكون من القسم الثاني و 30 % يكون في القسم الخامس

و باستخدام argmax يمكن تحديد ترتيب القيمة الاكبر, وعمل مصفوفة كاملة

```
topic_results.argmax(axis=1)
```

ثم يمكن عمل عمود جديد فيه رقم الصنف و اضافته للجدول الاساسي

```
npr['Topic'] = topic_results.argmax(axis=1)
npr.head(10)
```

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

كما ان مكتبة genism تقوم بعمل LDA بشكل افضل من sklearn , وهو ما سنراه في القسم الثامن