204k News Articles and Essays

Overview and the problem:

Firstly the project aims to group together similar topics that are similar but not the same. That would mean a topic about football should be clustered with another topic about basketball. Yes they are both sports.

Secondly, each topic might contain multiple topics. Such as a sporting event for charity or a political debate about human rights and how technology could help.

Thirdly, most News and Articles consist of multiple topics and therefore we will tackle the problem as such using topic-modeling and clustering.

DATA:

The data source is: https://components.one/datasets/all-the-news-articles-dataset The Raw data containts 12 features: id, title, author, date, content, year, month, publication, category, digital, section, url. The features we are using are only the 'title' and 'content'. The data we are not interested in will be dropped/ignored. The 'title' is the headline/name/title of the news/Article/Essay. The 'Content' is the body/content/Essay/Article/News itself.

The data is large enough to not run on our local machine and for that reason we have divided the project into large enough segments. TITLE and CONTENT

Due to the large size of the data the run times we long and the nature of our data did not let us run on clouds. The issue occurred with clouds is that the data was kept being read wrong by the python library provided. We tried multiple options but eventually we gave up due to time constraints as we did not expect to face such an issue. That being said the models are not ran more than 4 times hence not the best we could have done.

EDA:

In the project we have done the same EDA on both routes: TITLE and CONTENT.
So Anything below this has been done twice with different variations to accommodate its data.
Ex: if we say we ran the model twice that means twice for each type(Title and content). Once for title and a second time for content adding up to four times.

1. Dropping unwanted columns
2. Stopword removal
3. Lemmatization.
4. Stemming.
5. Mapping (index, [xx,yy,kk,...])
6. Bag of words for all documents
7. Tokenize
8. Remove extremes
9. Bag of words for each document

10. Vectorize using TF-IDF

Built functions to do:

- Stemming
- Lemmatization

Built-in functions used:

- Stemming
- Lemmatization
- Stopword
- Filter_extreme
- Map

Models:

LDA model was ran twice:

- Tokenized data
- TF-IDF data

Clustering model Kmeans was ran twice:

- Output of the LDA from Tokenized data
- Output of the LDA from TF-IDF data

The output of the models:

LDA

```
Topic: 0
Words: 0.013*"hous" + 0.012*"say" + 0.011*"republican" + 0.010*"breitbart" + 0.010*"white" + 0.010*"obama" + 0.009*"russia" + 0.007*"plan" + 0.007*"hea
lth" + 0.007*"senat"
---

Topic: 1
Words: 0.011*"life" + 0.008*"american" + 0.008*"world" + 0.008*"opinion" + 0.008*"dy" + 0.007*"video" + 0.007*"review" + 0.006*"home" + 0.005*"year" +
0.005*"histori"
---

Topic: 2
Words: 0.029*"opinion" + 0.013*"america" + 0.012*"breitbart" + 0.011*"north" + 0.010*"korea" + 0.010*"presid" + 0.009*"donald" + 0.009*"woman" + 0.008
*"week" + 0.007*"brief"
---

Topic: 3
Words: 0.015*"kill" + 0.015*"attack" + 0.012*"polic" + 0.011*"court" + 0.009*"state" + 0.008*"case" + 0.008*"death" + 0.008*"offic" + 0.008*"shoot" +
0.007*"say"
---

Topic: 4
Words: 0.027*"clinton" + 0.015*"donald" + 0.014*"hillari" + 0.010*"like" + 0.010*"breitbart" + 0.007*"look" + 0.007*"year" + 0.007*"wall" + 0.005*"win"
+ 0.005*"appl"
---
```

LDA + TF-IDF:

```
Topic: 0
Words: 0.004*"home" + 0.004*"opinion" + 0.004*"appl" + 0.004*"wall" + 0.004*"street" + 0.004*"world" + 0.003*"sale" + 0.003*"year" + 0.003*"record" + 0.003*"crash"
---

Topic: 1
Words: 0.007*"north" + 0.007*"opinion" + 0.006*"korea" + 0.006*"court" + 0.005*"say" + 0.005*"state" + 0.005*"attack" + 0.005*"syria" + 0.004*"china" + 0.004*"deal"
---

Topic: 2
Words: 0.006*"dy" + 0.005*"year" + 0.005*"woman" + 0.005*"life" + 0.005*"polic" + 0.004*"review" + 0.004*"shoot" + 0.004*"star" + 0.004*"kill" + 0.004*"death"
---

Topic: 3
Words: 0.012*"clinton" + 0.010*"opinion" + 0.008*"donald" + 0.008*"republican" + 0.007*"hillari" + 0.007*"democrat" + 0.007*"health" + 0.006*"breitbart" + 0.006*"read" + 0.006*"presid"
---

Topic: 4
Words: 0.018*"opinion" + 0.005*"brief" + 0.005*"book" + 0.004*"news" + 0.004*"america" + 0.004*"even" + 0.004*"facebook" + 0.004*"thing" + 0.004*"olymp" + 0.004*"breitbart"
---
```

When tested with: 'Ghostbusters Learned From Force Awakens Mistakes (And It Worked Perfectly)'

The models output is:

LDA

```
Score: 0.5589032769203186
Topic: 0.011*"life" + 0.008*"american" + 0.008*"world" + 0.008*"opinion" + 0.008*"dy" + 0.007*"video" + 0.007*"review" + 0.006*"home" + 0.005*"year" + 0.005*"histori"

Score: 0.24325038492679596
Topic: 0.013*"hous" + 0.012*"say" + 0.011*"republican" + 0.010*"breitbart" + 0.010*"white" + 0.010*"obama" + 0.009*"russia" + 0.007*"plan" + 0.007*"health" + 0.007*"senat"

Score: 0.14723505079746246
Topic: 0.029*"opinion" + 0.013*"america" + 0.012*"breitbart" + 0.011*"north" + 0.010*"korea" + 0.010*"presid" + 0.009*"donald" + 0.009*"woman" + 0.008*"week" + 0.007*"brief"

Score: 0.025336600840091705
Topic: 0.015*"kill" + 0.015*"attack" + 0.012*"polic" + 0.011*"court" + 0.009*"state" + 0.008*"case" + 0.008*"death" + 0.008*"offic" + 0.008*"shoot" + 0.007*"say"

Score: 0.02527468465268612
Topic: 0.027*"clinton" + 0.015*"donald" + 0.014*"hillari" + 0.010*"like" + 0.010*"breitbart" + 0.007*"look" + 0.007*"year" + 0.007*"wall" + 0.005*"win" + 0.005*"appl"
```

LDA + TF-IDF:

```
Score: 0.4485916495323181
Topic: 0.006*"dy" + 0.005*"year" + 0.005*"woman" + 0.005*"life" + 0.005*"polic" + 0.004*"review" + 0.004*"shoot" + 0.004*"star" + 0.004*"kill" + 0.004*"death"

Score: 0.29611530900001526
Topic: 0.012*"clinton" + 0.010*"opinion" + 0.008*"donald" + 0.008*"republican" + 0.007*"hillari" + 0.007*"democrat" + 0.007*"health" + 0.006*"breitbart" + 0.006*"read" + 0.006*"presid"

Score: 0.20367766916751862
Topic: 0.004*"home" + 0.004*"opinion" + 0.004*"appl" + 0.004*"wall" + 0.004*"street" + 0.004*"world" + 0.003*"sale" + 0.003*"year" + 0.003*"record" + 0.003*"crash"

Score: 0.026074189692735672
Topic: 0.007*"north" + 0.007*"opinion" + 0.006*"korea" + 0.006*"court" + 0.005*"say" + 0.005*"state" + 0.005*"attack" + 0.005*"syria" + 0.004*"china" + 0.004*"deal"

Score: 0.025541190057992935
Topic: 0.018*"opinion" + 0.005*"brief" + 0.005*"book" + 0.004*"news" + 0.004*"america" + 0.004*"even" + 0.004*"facebook" + 0.004*"thing" + 0.004*"olymp" + 0.004*"breitbart"
```

Model output meaning:
The LDA model can not define the topics' name but it can tell what the topic contents should be. Example it can't tell football is a sport but it knows it goes with basketball. Its up to us the users to read those outputs and make the connections. In this case we decided to use:

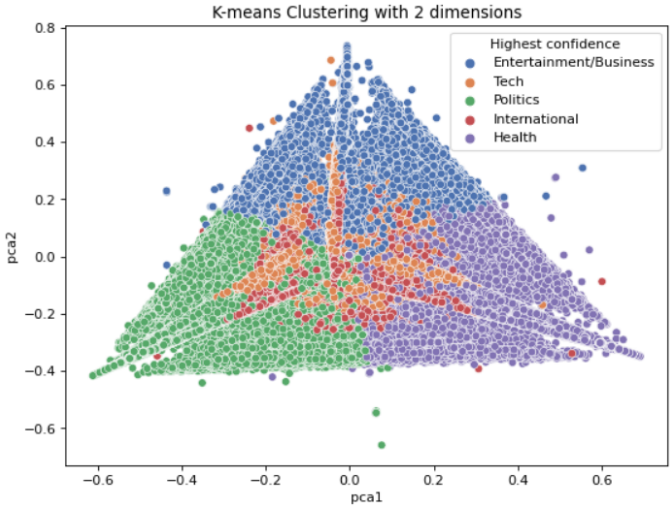
- LDA model
    - Topic 0 = 'Entertainment/Business'
    - Topic 1 = 'Tech'
    - Topic 2 = 'International'
    - Topic 3 = 'Politics'
    - Topic 4 = 'Health'

- LDA + TF-IDF
    - Topic 0 = 'Entertainment/Business'
    - Topic 1 = 'Tech'
    - Topic 2 = 'International'
    - Topic 3 = 'Politics'
    - Topic 4 = 'Health'
    -

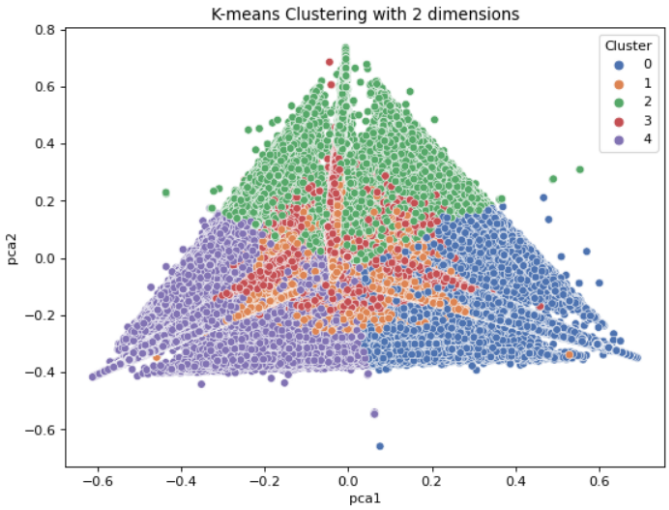
The above are just some of the topics; Please check the code for the rest.

MidPoint:
After we finished training the model. We use the model to build a dataframe consisting of the confidence value for each topic to each title/content.

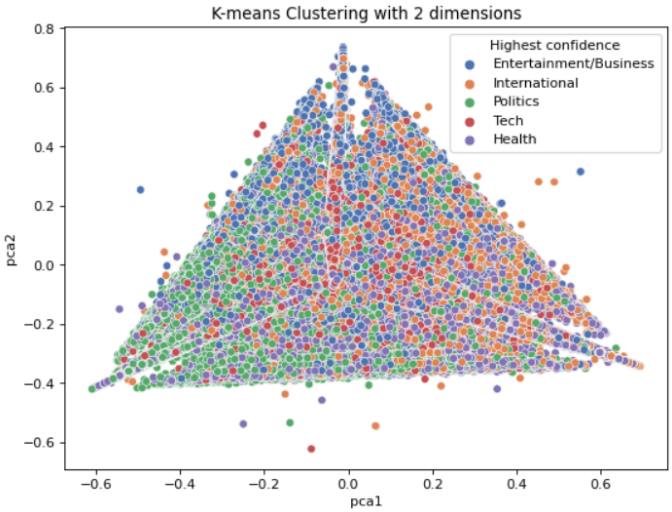
Clustering:
Since we have a dataframe built now with the right data we ran the KMean clustering algorithm. Once that was done we attached the cluster name Y to the data frame. Finally we did a PCA on the same dataframe and plotted a scatterplot giving us these plots.



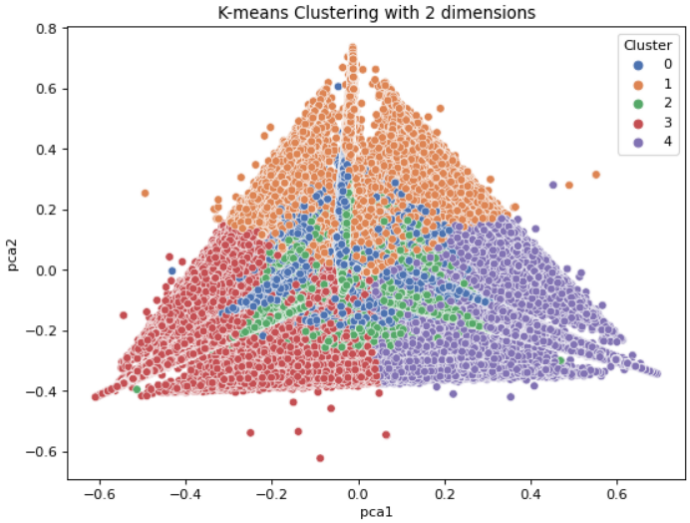
The Above is LDA confidences from tonized data with the target used is LDA output



The Above is LDA confidences from tonized data with the target used is Kmeans output



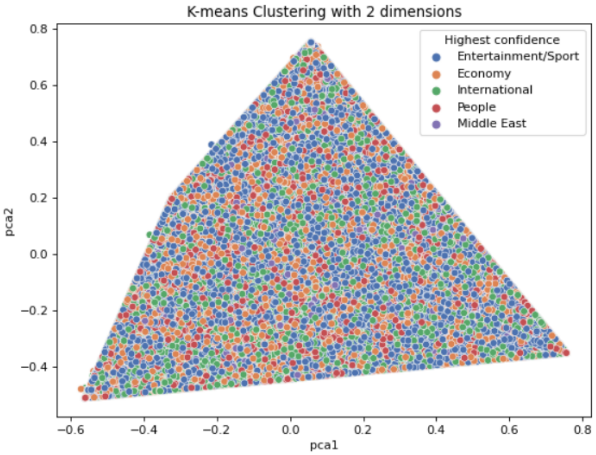
The Above is LDA confidences from TF-IDF data with the target used is LDA output



The Above is LDA confidences from TF-IDF data with the target used is Kmeans output

Note:

We were not sure how good the content would be in clustering so we built the model based on title also. Because of the nature of text and how sensitive Kmeans is. Here is an example of the plot being horrible but the topic modeling is good.



Yet when tested on unseen fresh off the news' site data like below:

"LONDON: Chelsea coach Thomas Tuchel plans to hold talks with Romelo Lukaku after the striker said he was unhappy at the London club and expressed a wish to rejoin Inter Milan. In a pre-recorded interview with Sky Italia released on Thursday, the 28-year-old…"Just a snippet of what is passed to the model. Its too long to put here

.

It placed this at sport/Entertainment with 98% confidence which the correct topic of sports. We believe the model requires significantly longer training time which

we don't have nor have the machine to do as our machines kept crashing. Now We know that our same model worked perfectly on lesser data like title and we believe it should work with the content too. A temporary solution could be that we have trained on ⅓ of the data instead.

Conclusion.

The title and content modeling performed better. But the title model performed better relative to the content. The reason being the content is 93% of the data and was trained less because of the hardware difficulties. Clustering performed way better on the title models. I believe because the topic modeling on the content needs more training.