HOW POLITICIANS CHANGE THEIR MIND

Long Version

image by MidJourney AI

Matteo Biglioli, Roberto Staino

- Text Mining and Sentiment Analysis
- Knowledge Extraction and Information Retrieval

938199 bigliolimatteo@gmail.com bigliolimatteo

967485 roberto.staino97@gmail.com RobertoStaino

INTRODUCTION

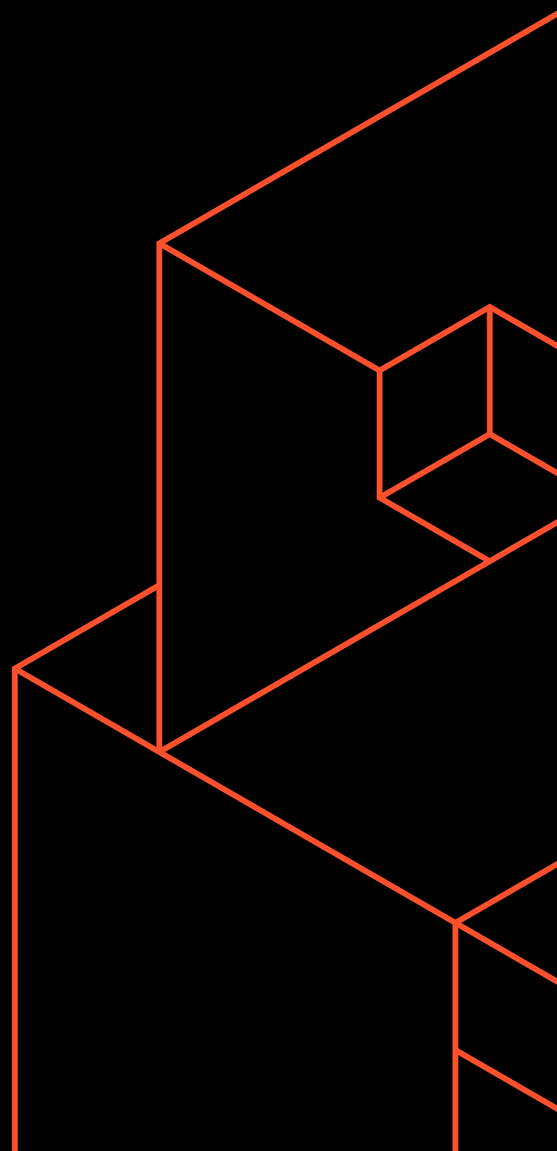
In these past years we experienced a boost in the use of social media platforms by politicians, which moved from not even being present to posting multiple times a day following accurate marketing strategies.

In this project we will analyse tweets from Italian politicians and cluster them into topics using different techniques.

The anticipation would be to find cluster of tweets addressing immigration, justice, taxes, retirements benefits, etc...

We will discuss our findings by comparing different politicians based on the topics addressed in their social communication strategy.

We will also compare the cluster distribution obtained from the different embeddings.



DISCLAIMER

Due to limitations of the Twitter API we were able to download just 5k tweets per user.

For this reason the timeframe of our analysis span from the 22nd of July to the 25th of September (2022).

It should also be noted that we did not take into account retweets, in order to reduce the noise of the dataset.

POLITICIANS - RIGHT WING

Giorgia Meloni
@GiorgiaMeloni
Joined April 2010

Matteo Salvini
@matteosalvinimi
Joined March 2011

Silvio Berlusconi
@berlusconi
Joined October 2017

Ignazio La Russa
@Ignazio_LaRussa
Joined December 2011

Antonio Tajani
@Antonio_Tajani
Joined March 2012

POLITICIANS - LEFT WING

Enrico Letta
@EnricoLetta
Joined November 2011

Emma Bonino
@emmabonino
Joined January 2010

Luigi Di Maio
@luigidimaio
Joined June 2009

Pier Luigi Bersani
@pbersani
Joined June 2009

BenedettoDellaVedova
@bendellavedova
Joined February 2011

Nicola Fratoianni
@NFratoianni
Joined December 2011

Dario Franceschini
@dariofrance
Joined July 2009

Riccardo Magi
@riccardomagi
Joined April 2011

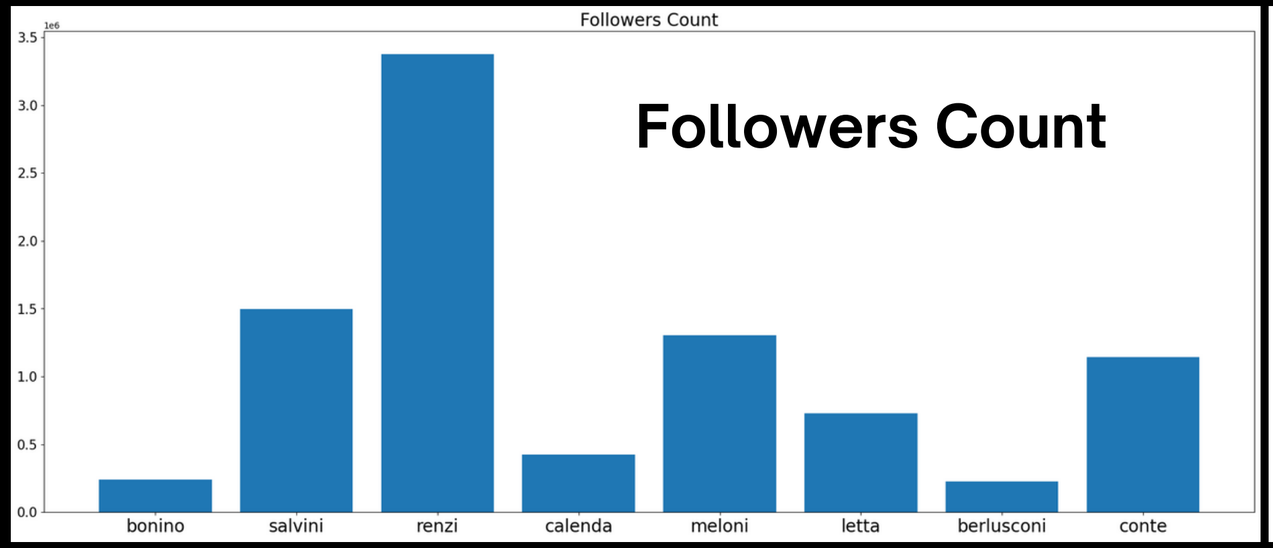
POLITICIANS - OTHERS

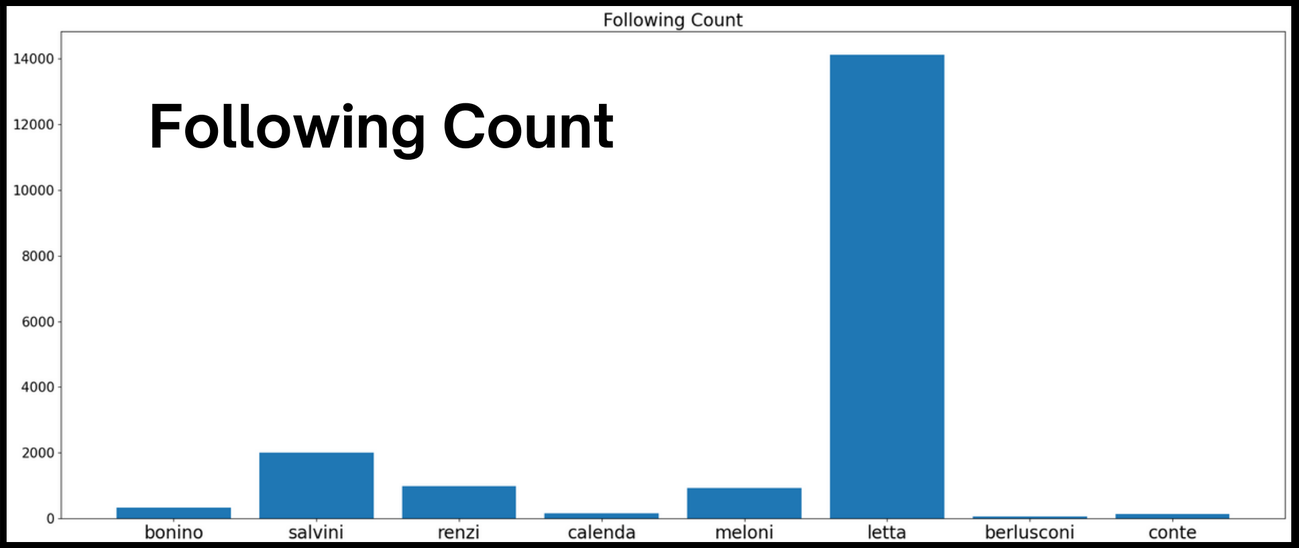
Carlo Calenda
@CarloCalenda
Joined March 2014

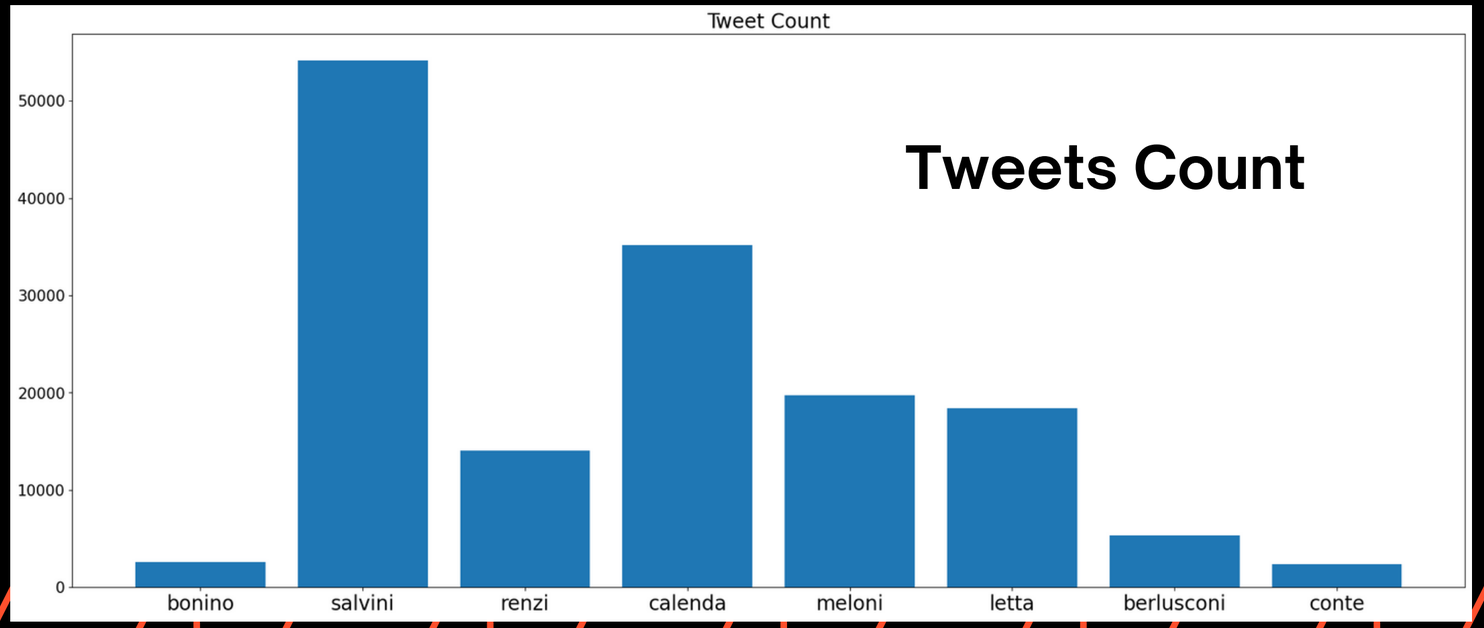
Giuseppe Conte
@GiuseppeConteIT
Joined May 2018

Matteo Renzi
@matteorenzi
Joined January 2009

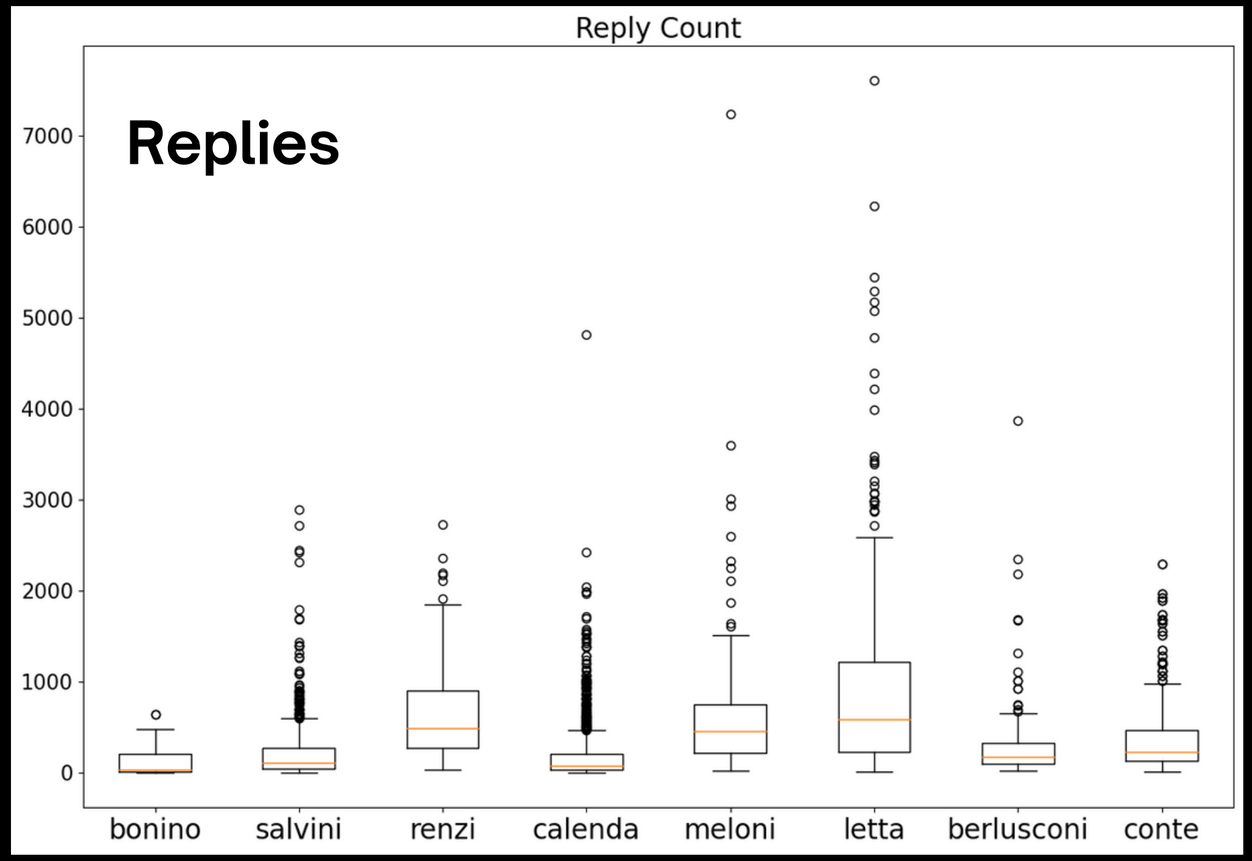
POLITICIANS - PUBLIC METRICS

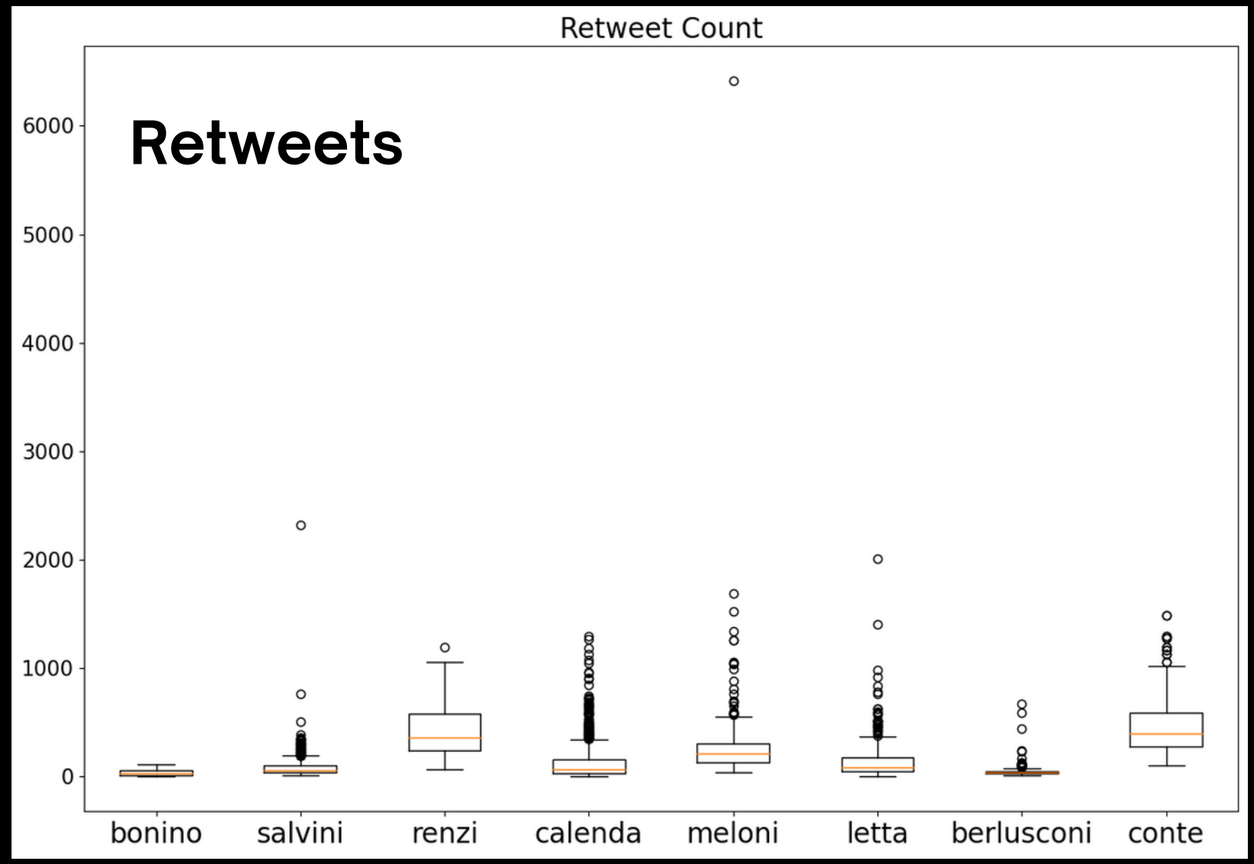




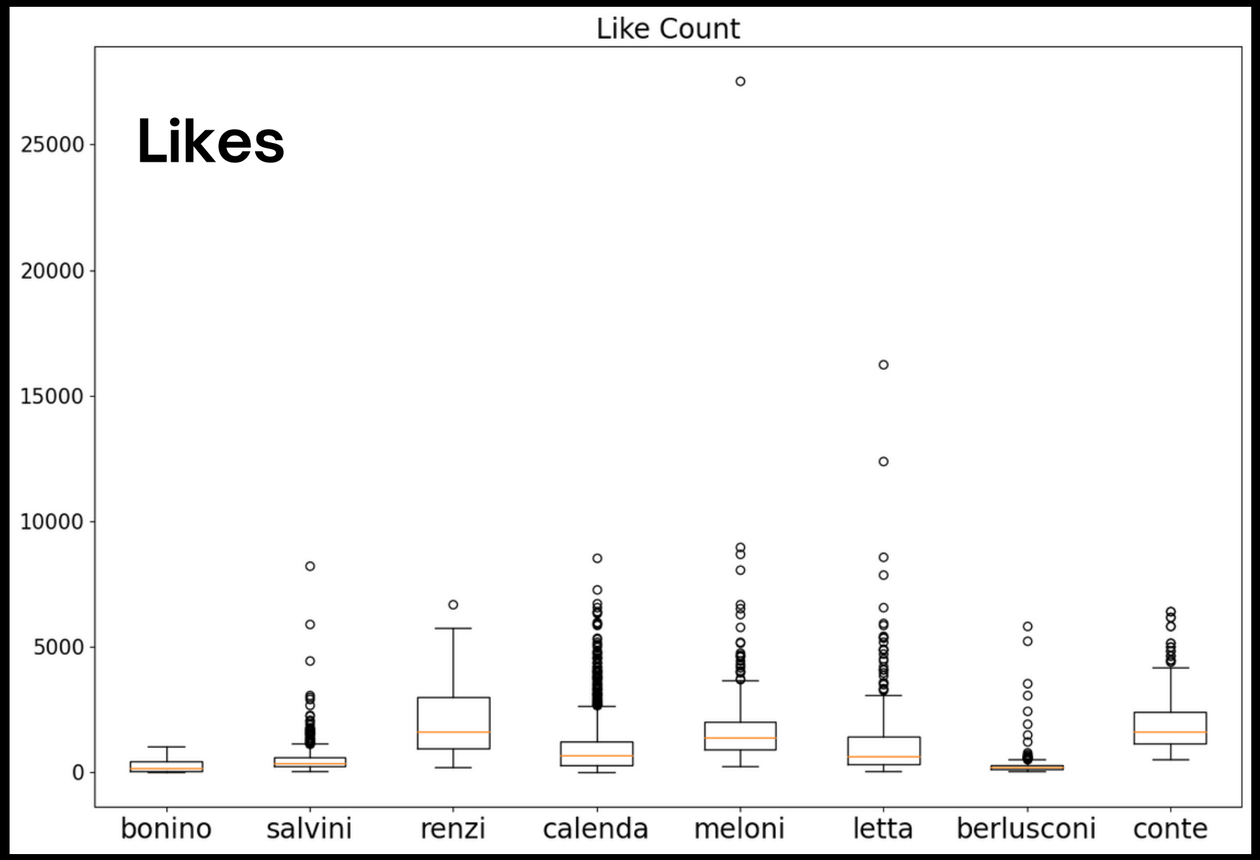


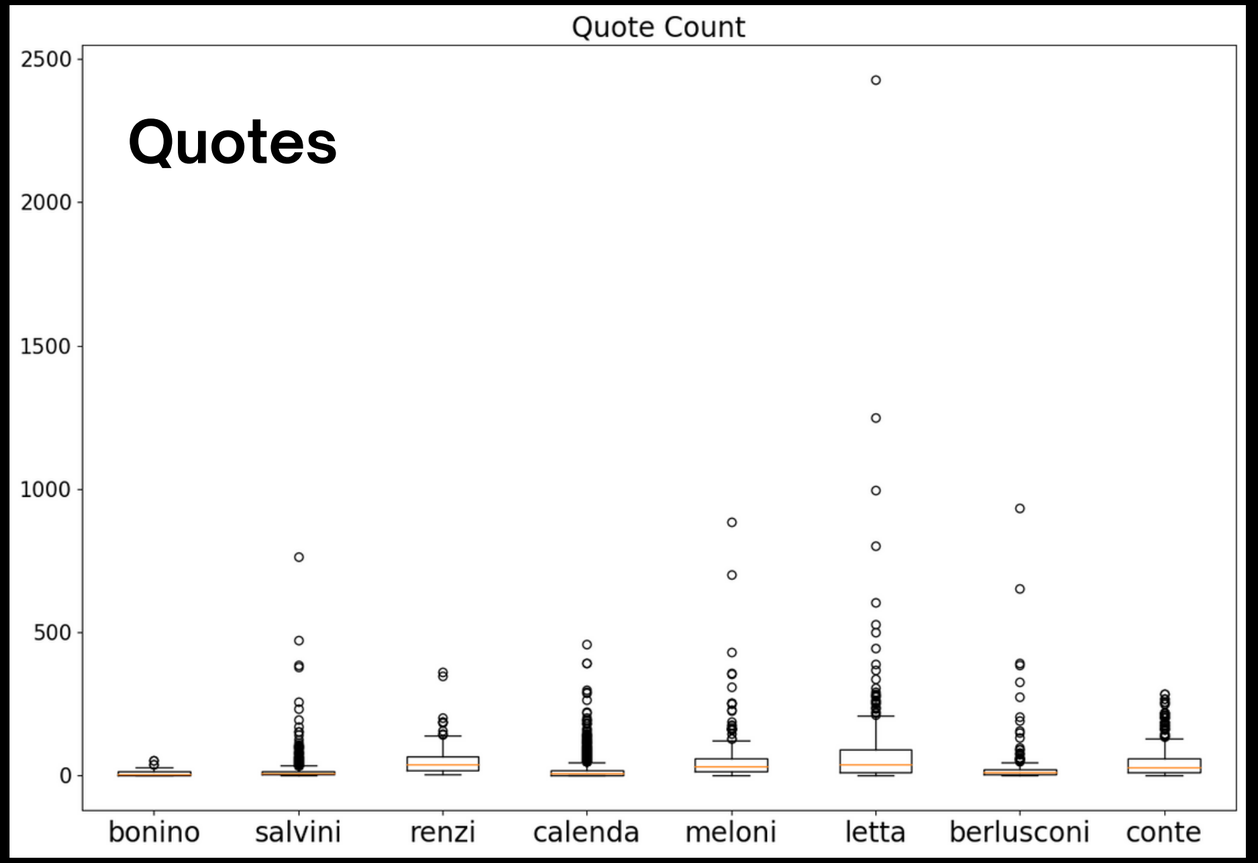
TWEETS - PUBLIC METRICS



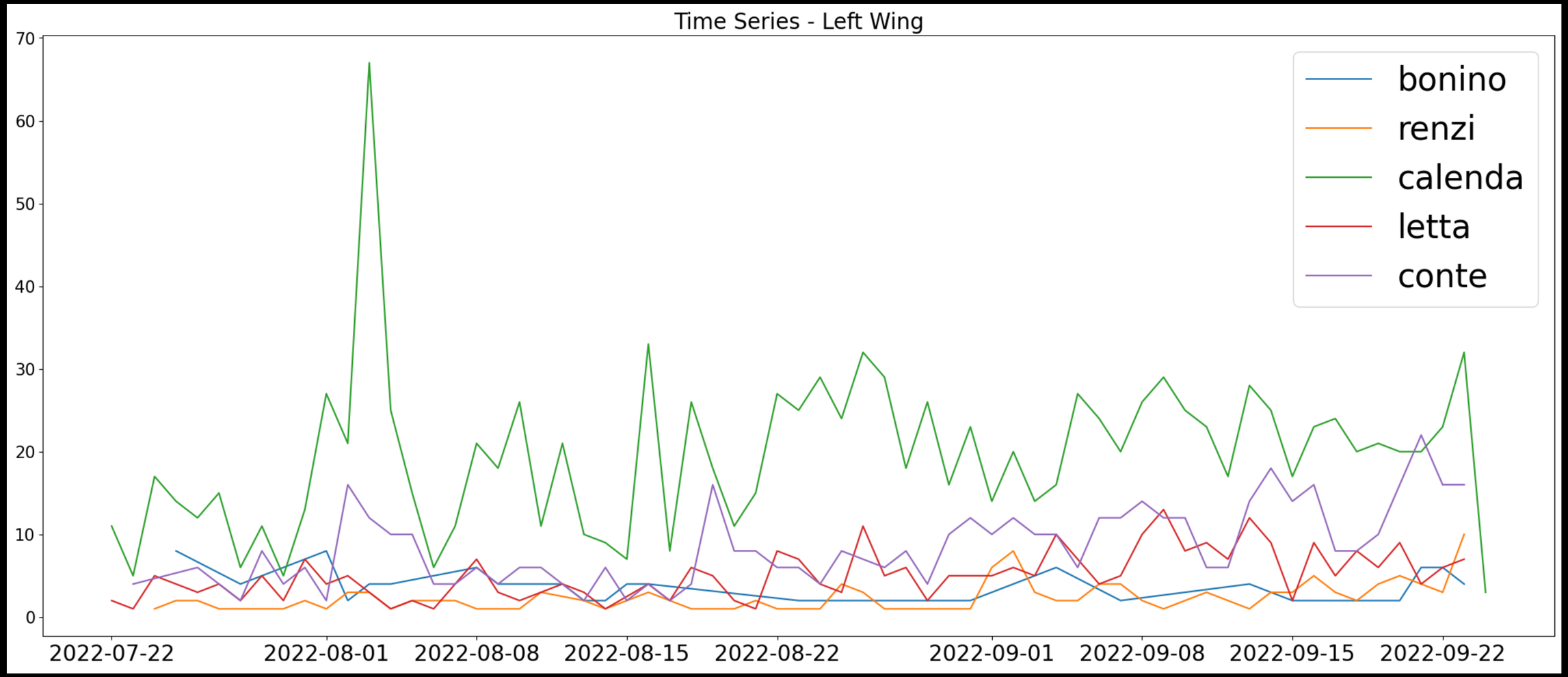


TWEETS - PUBLIC METRICS

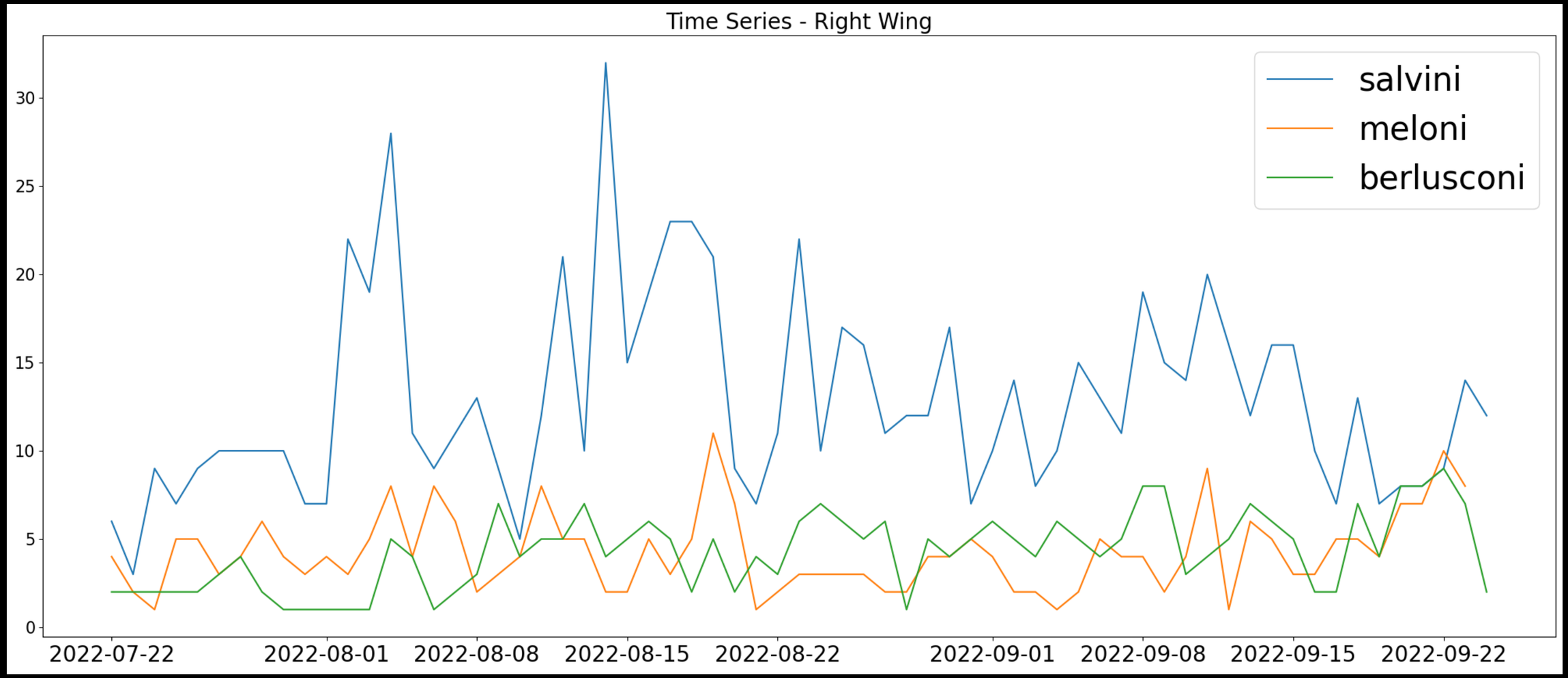




TWEETS - TIME SERIES



TWEETS - TIME SERIES



DATA CLEANING

Before applying any text mining techniques we had to clean the data

Removing tweets out of our interval of interest.

Because the official API has a limit on the number of tweets which can be downloaded for each account, we had to find a common date-range for all the subjects).

```
# Remove tweets out of date bound
data["created_at"] = pd.to_datetime(data["created_at"], format="%Y-%m-%dT%H:%M:%S.%fZ")
date_filtered_data = data.loc[(data['created_at'] >= start_date) & (data['created_at'] < end_date)]
```

```
# Remove duplicates
unique_date_filtered_data = date_filtered_data.drop_duplicates("text")
```

Removing duplicated tweets

which we judged non-significant due to the fact that the majority of this kind of content is just noise (e.g. #Pontida2022)

DATA CLEANING

Before applying any text mining techniques we had to clean the data

Removing retweets

We took this choice because we realized that keeping all the retweets would have unnecessarily increased the complexity of our job by adding confusion to the actual topics addressed by politicians.

```python
# Remove retweets
non_retweet_date_filtered_data = unique_date_filtered_data[unique_date_filtered_data\
    .apply(lambda row: not row['text'].startswith("RT @"), axis=1)]
```

```python
def join_threads(data):
    # The only problem here could be cases in which a politician replies to himself without the goal of creating a thread and
    # inserts two tokens like r'[0-9]/[0-9]' in all the tweets of the same conversation ID
    tweets_thread_condition = data["text"].str.contains(r'[0-9]+/[0-9]+')

    cleaned_data_no_threads = data.drop(data[tweets_thread_condition].index)
    cleaned_data_just_threads = data[tweets_thread_condition]

    aggregation_dict = {"id": "first",
                "public_metrics.retweet_count" : "sum",
                "public_metrics.reply_count" : "sum",
                "public_metrics.like_count" : "sum",
                "public_metrics.quote_count" : "sum",
                'created_at':'first',
                "referenced_tweets": "first",
                'text': lambda x: ','.join(x)}

    joined_threads = cleaned_data_just_threads.groupby(['politician', "conversation_id"]).agg(aggregation_dict).reset_index()

    return pd.concat([cleaned_data_no_threads, joined_threads])
```

Joining threads

It is common, because of the 150 word limit of Twitter, to see multiple tweets which are meant to be coupled in a single document; we managed to find these posts and join them.

DATA PREPROCESSING

Data must be preprocessed, so that the models can use it properly for the analysis.
This step reduces the noise and it's crucial for the final results.

```python
def tokenize_tweet(self, tweet: str):
    return [word for word in self.tokenizer.tokenize(tweet) \
        if not self.is_punctuation(word) and not self.is_stopword(word)]
```

Tokenization

We tokenized the tweets by applying the TweetTokenizer, we removed stopwords and puntuation and we stemmed the remaining tokens.

Stemming

We intentionally leveraged a Stemmer - SnowballStemmer - instead of a Lemmatizer due to the fact that we found few lemmatizers working with Italian data, and all of them show issues in properly identifying most of the tokens in our corpus.

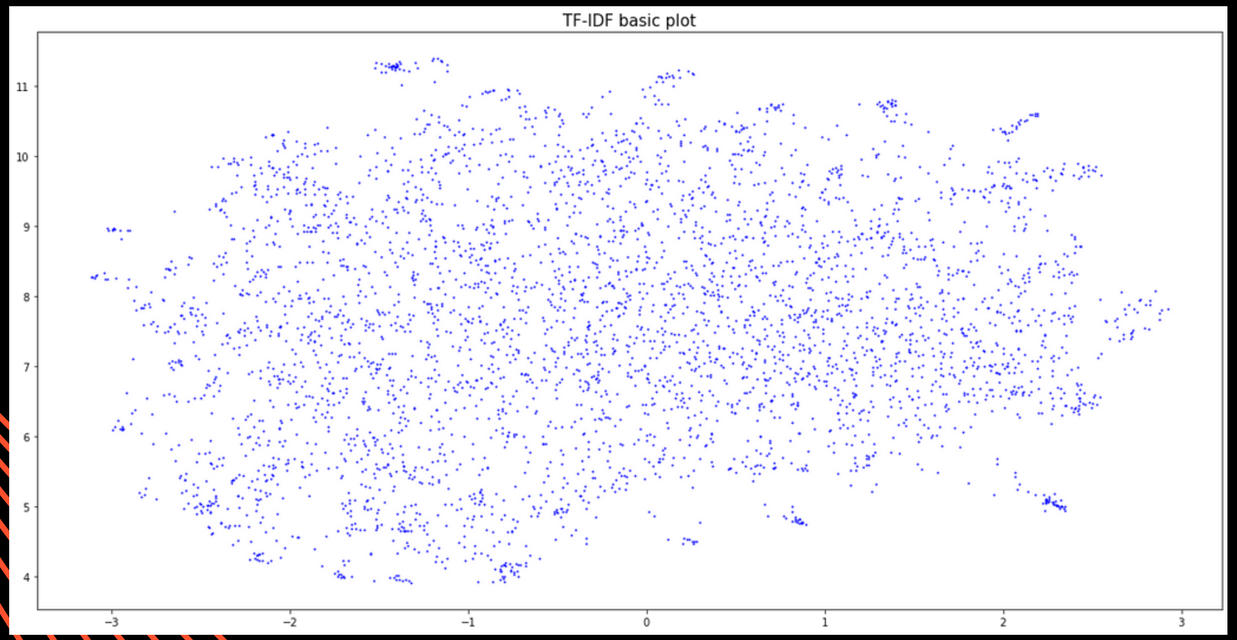
```python
def stem_tweet(self, tweet: list):
    return [self.stemmer.stem(word) for word in tweet]

def stem_data(self, data):
    stemmed_data = data.copy()
    stemmed_data["text"] = data["text"].map(lambda text: self.stem_tweet(text))
    # Drop empty tweets
    return stemmed_data[stemmed_data["text"].astype(str) != "[]"]
```

We slightly modified the Stemmer code in order to preserve some custom tokens which we did not want to be stemmed, like Meloni or Draghi.

```python
self.stemmer = OurStemmer(open("custom_tokens.txt", "r").read().split('\n'))
```

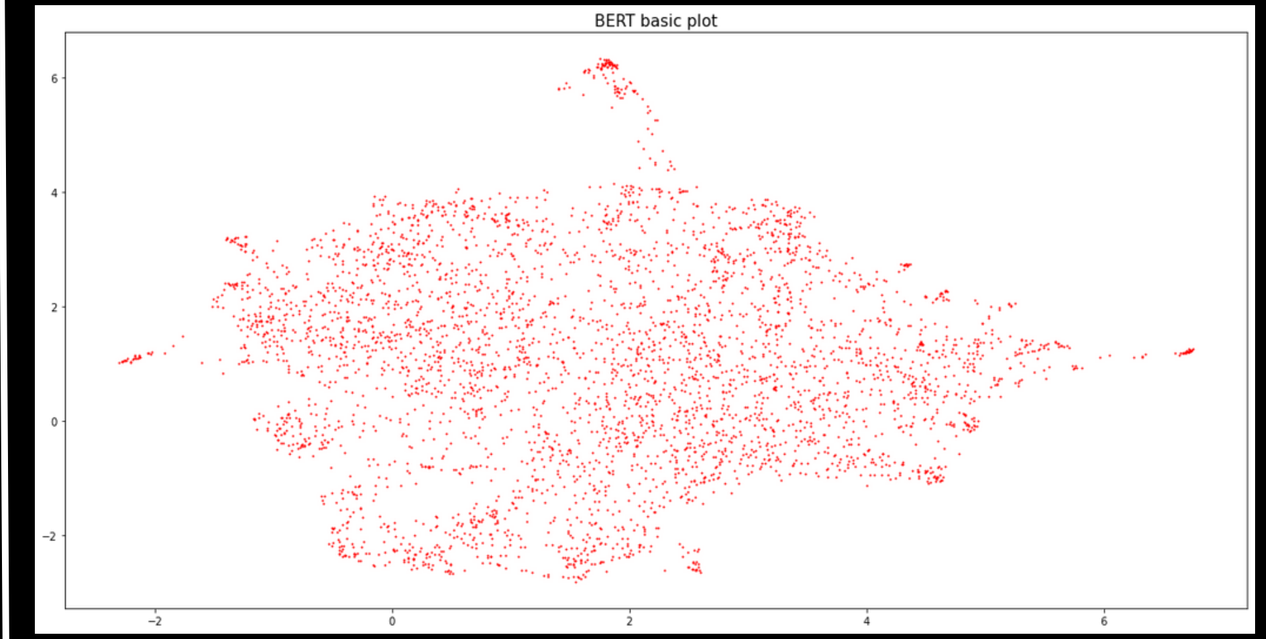
EMBEDDINGS

In this project we experimented with two different embedding techniques, namely TF-IDF and BERT. We employed UMAP to perform dimension reduction.



TF-IDF

BERT



CLUSTERS - NAÏVE

We leverage HDBSCAN to perform the clustering of our data.
We started by naively applying the algorithm with the default hyperparameters.

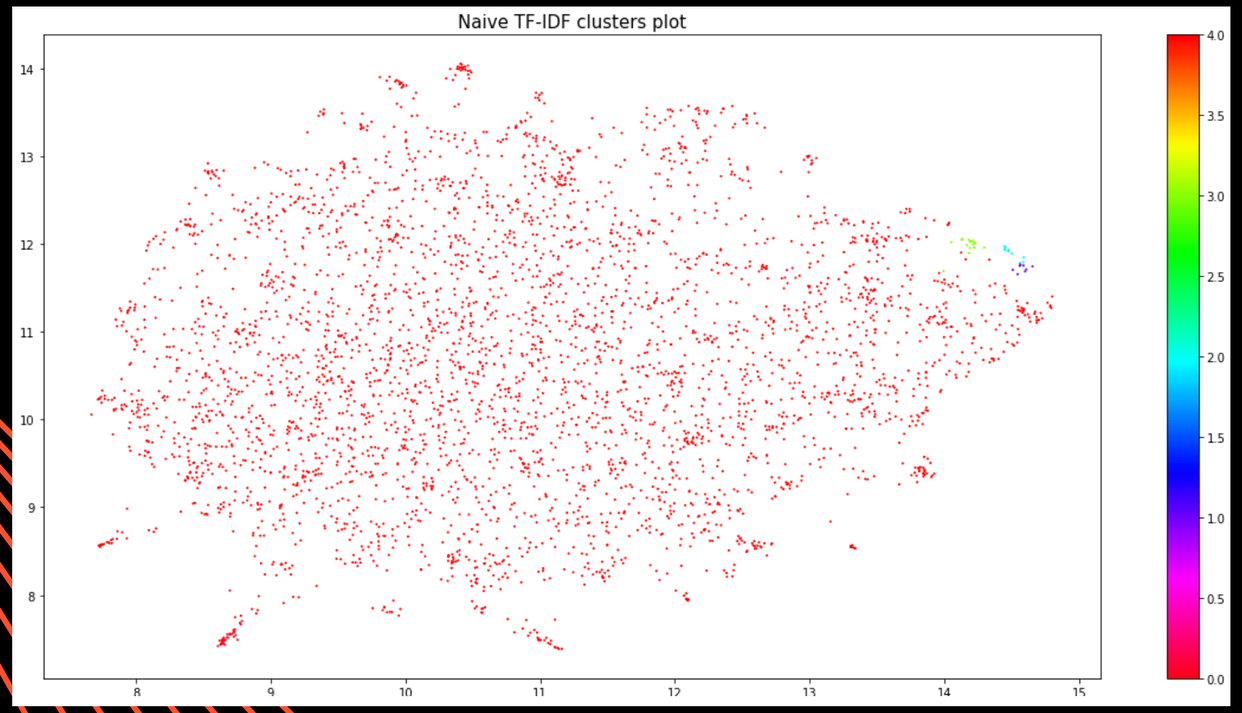
```
[16] # Dimension Reduction
     naive_tfidf_reduced = umap.UMAP(random_state=42).fit_transform(X_tfidf)
     naive_bert_reduced = umap.UMAP(random_state=42).fit_transform(embeddings)
```

```
[17] # Cluster algorithm
     naive_tfidf_cluster = hdbscan.HDBSCAN().fit(naive_tfidf_reduced)
     naive_bert_cluster = hdbscan.HDBSCAN().fit(naive_bert_reduced)
```

```
[18] # Cluster visualization (projected w/ UMAP)
     plot_vectors(X_tfidf, naive_tfidf_cluster.labels_, title="Naive TF-IDF clusters plot")
     plot_vectors(embeddings, naive_bert_cluster.labels_, title="Naive BERT clusters plot")
```

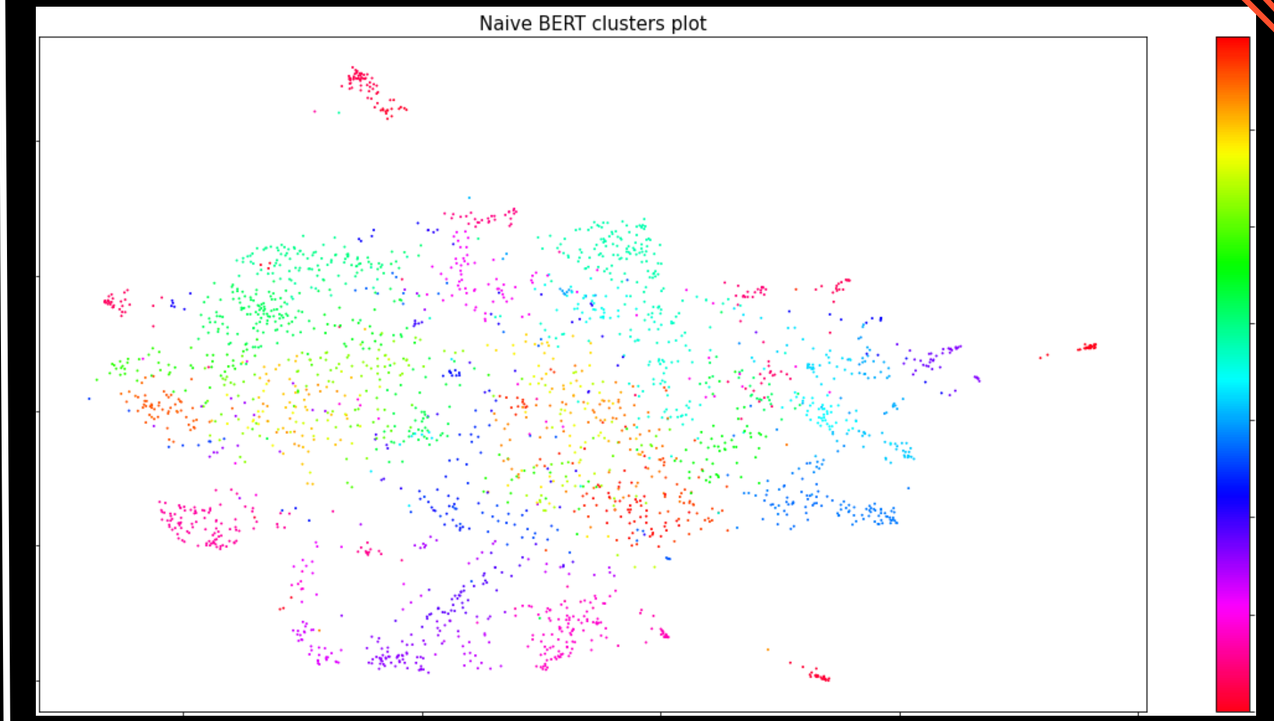
CLUSTERS - NAÏVE

The output of the default clustering approach is hardly acceptable,
as it computes respectively too many and too little clusters.



TF-IDF

BERT



HYPERPARAMETERS TUNING

UMAP

- *n_neighbors*: specify the local neighborhood UMAP will look at when attempting to learn the manifold structure of the data.
- *n_components*: dimension of the reduced space.

Documentation

https://umap-learn.readthedocs.io/en/latest/parameters.html

https://hdbscan.readthedocs.io/en/latest/index.html

HDBSCAN

- *min_cluster_size*: minimum number of elements in a cluster.
- *min_samples*: measure of how conservative you want you clustering to be. Larger values imply more noise but better defined clusters.
- *cluster_selection_method*: determine how it selects flat clusters from the cluster tree hierarchy. 'eom' generally has better results than 'leaf'.
- *metric*: select the measure of distance preferred.

CODE SNIPPETS

```
for i in trange(num_evals):
    n_neighbors = random.choice(space['n_neighbors'])
    n_components = random.choice(space['n_components'])
    min_cluster_size = random.choice(space['min_cluster_size'])
    min_samples = random.choice(space['min_samples'])
    metric = random.choice(space['metric'])
    cluster_selection_method = random.choice(space['cluster_selection_method'])

    clusters = generate_clusters(embeddings,
                                 n_neighbors = n_neighbors,
                                 n_components = n_components,
                                 min_cluster_size = min_cluster_size,
                                 min_samples = min_samples,
                                 metric = metric,
                                 cluster_selection_method = cluster_selection_method,
                                 random_state = 42)

    label_count, cost = score_clusters(clusters, prob_threshold = 0.05)
```

Specify the parameters of the grid search

```
def score_clusters(clusters, prob_threshold = 0.05):

    cluster_labels = clusters.labels_
    label_count = len(np.unique(cluster_labels))
    total_num = len(clusters.labels_)
    cost = (np.count_nonzero(clusters.probabilities_ < prob_threshold)/total_num)

    return label_count, cost
```

GRID SEARCH

```
param_dist = {'n_neighbors':[10,15,20],
              'n_components': [5,8,12],
              'min_samples': [1,3,5],
              'min_cluster_size':[5,8,10,15],
              'cluster_selection_method' : ['eom','leaf'],
              'metric' : ['sokalsneath','rogerstanimoto','manhattan','euclidean']
             }
```

RESULTS

TF-IDF

	Run_id	N_neighbors	N_componen	Min_cluster_	Min_samples	Metric	Cluster_selec	Label_count	Cost
799	799	15	8	8	1	euclidean	eom	163	0.29
1316	1316	15	8	8	1	euclidean	eom	163	0.29
1794	1794	15	8	8	1	euclidean	eom	163	0.29
195	195	15	12	8	1	manhattan	eom	139	0.29
1439	1439	15	12	8	1	manhattan	eom	139	0.29
1985	1985	15	12	8	1	manhattan	eom	139	0.29
988	988	15	12	8	1	manhattan	eom	139	0.29
530	530	15	12	8	1	manhattan	eom	139	0.29
240	240	10	8	8	1	euclidean	leaf	177	0.29
1349	1349	10	5	15	1	manhattan	eom	79	0.29
55	55	10	5	15	1	manhattan	eom	79	0.29
229	229	10	5	15	1	manhattan	eom	79	0.29
622	622	10	8	8	1	manhattan	leaf	173	0.3
1029	1029	10	8	8	1	manhattan	leaf	173	0.3
1190	1190	10	8	8	1	manhattan	leaf	173	0.3
1608	1608	10	8	10	1	manhattan	eom	128	0.3
1447	1447	10	8	10	1	manhattan	eom	128	0.3
1484	1484	10	8	10	1	manhattan	eom	128	0.3
450	450	10	8	10	1	manhattan	eom	128	0.3
455	455	20	8	5	1	manhattan	eom	251	0.3
1126	1126	20	8	5	1	manhattan	eom	251	0.3
940	940	20	8	5	1	manhattan	eom	251	0.3
1543	1543	20	8	5	1	manhattan	eom	251	0.3
1117	1117	10	5	10	1	manhattan	leaf	135	0.3

BERT

	Run_id	N_neighbors	N_componen	Min_cluster_	Min_samples	Metric	Cluster_selec	Label_count	Cost
1084	1084	15	8	8	1	manhattan	eom	113	0.25
883	883	15	8	8	1	manhattan	eom	113	0.25
294	294	15	8	8	1	manhattan	eom	113	0.25
134	134	15	8	8	1	manhattan	eom	113	0.25
651	651	10	5	15	1	manhattan	eom	59	0.27
435	435	10	5	15	1	manhattan	eom	59	0.27
282	282	15	8	15	1	manhattan	eom	46	0.28
682	682	15	8	15	1	manhattan	eom	46	0.28
604	604	15	8	15	1	manhattan	eom	46	0.28
1416	1416	15	8	10	1	manhattan	eom	80	0.28
552	552	15	8	10	1	manhattan	eom	80	0.28
1414	1414	15	8	10	1	manhattan	eom	80	0.28
1139	1139	15	5	5	1	euclidean	eom	246	0.29
386	386	15	5	5	1	euclidean	eom	246	0.29
404	404	15	5	5	1	euclidean	eom	246	0.29
632	632	15	5	5	1	euclidean	eom	246	0.29
1143	1143	10	5	5	1	manhattan	eom	285	0.29
836	836	10	5	5	1	manhattan	eom	285	0.29
867	867	10	5	5	1	manhattan	eom	285	0.29
198	198	10	12	8	1	euclidean	eom	142	0.29
964	964	10	12	8	1	euclidean	eom	142	0.29
611	611	10	12	8	1	euclidean	eom	142	0.29
131	131	15	12	5	1	euclidean	eom	227	0.29
626	626	15	12	5	1	euclidean	eom	227	0.29

CLUSTERS - ADVANCED

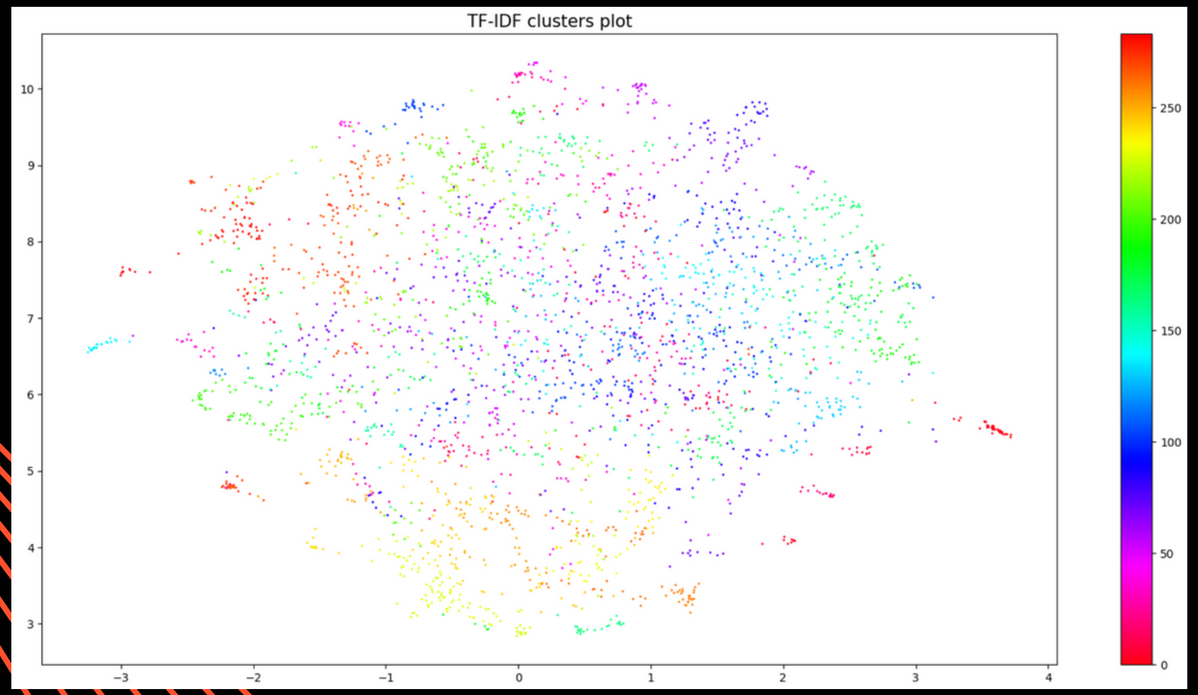
We can then apply the two algorithm with the selected hyperparameters.

```
[ ]  # Dimension Reduction
     tfidf_reduced = umap.UMAP(n_neighbors=10, n_components=15,
                               metric="cosine", random_state=42).fit_transform(X_tfidf)

     bert_reduced = umap.UMAP(n_neighbors=15, n_components=8,
                              metric="cosine", random_state=42).fit_transform(embeddings)
```

```
[ ]  # Cluster algorithm
     tfidf_cluster = hdbscan.HDBSCAN(min_cluster_size=5, min_samples=1,
                                     metric='manhattan', cluster_selection_method='eom').fit(tfidf_reduced)

     bert_cluster = hdbscan.HDBSCAN(min_cluster_size=15, min_samples=1,
                                    metric='manhattan', cluster_selection_method='eom').fit(bert_reduced)
```

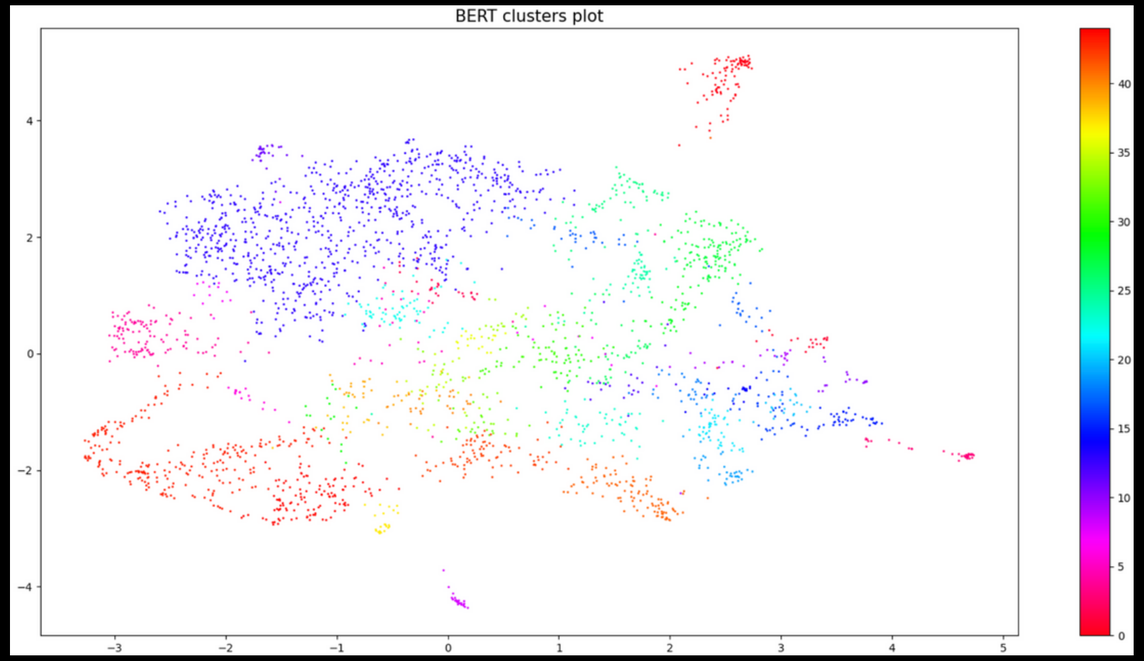
CLUSTERS - ADVANCED

The results definitely improved, as it is obvious that the number of clusters in both approaches is now manageable



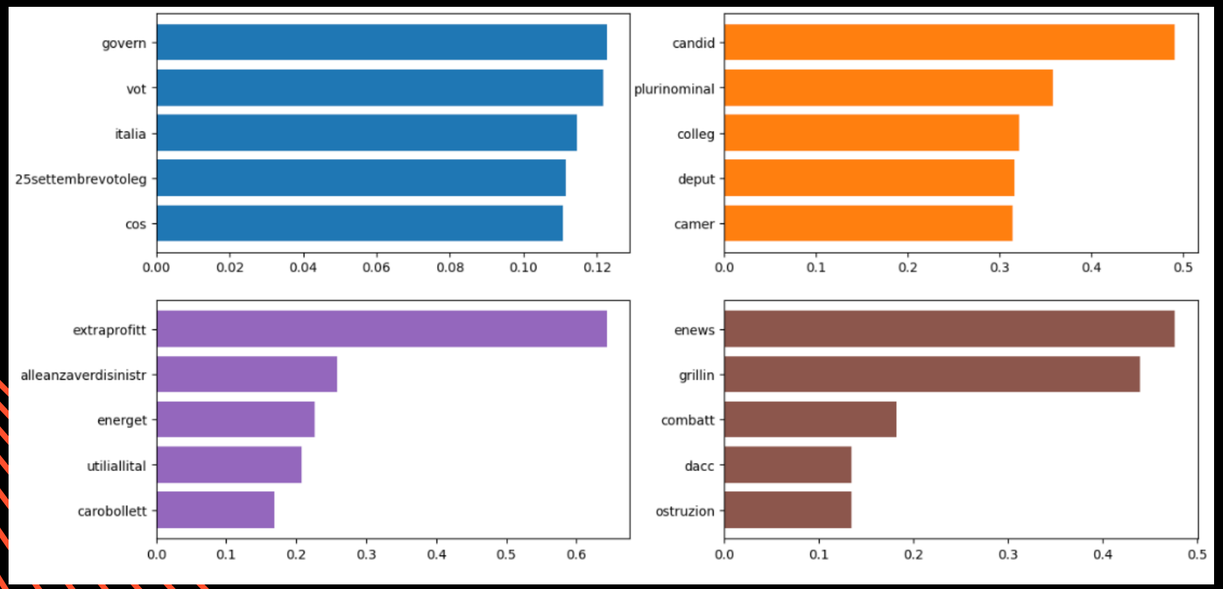
TF-IDF

BERT



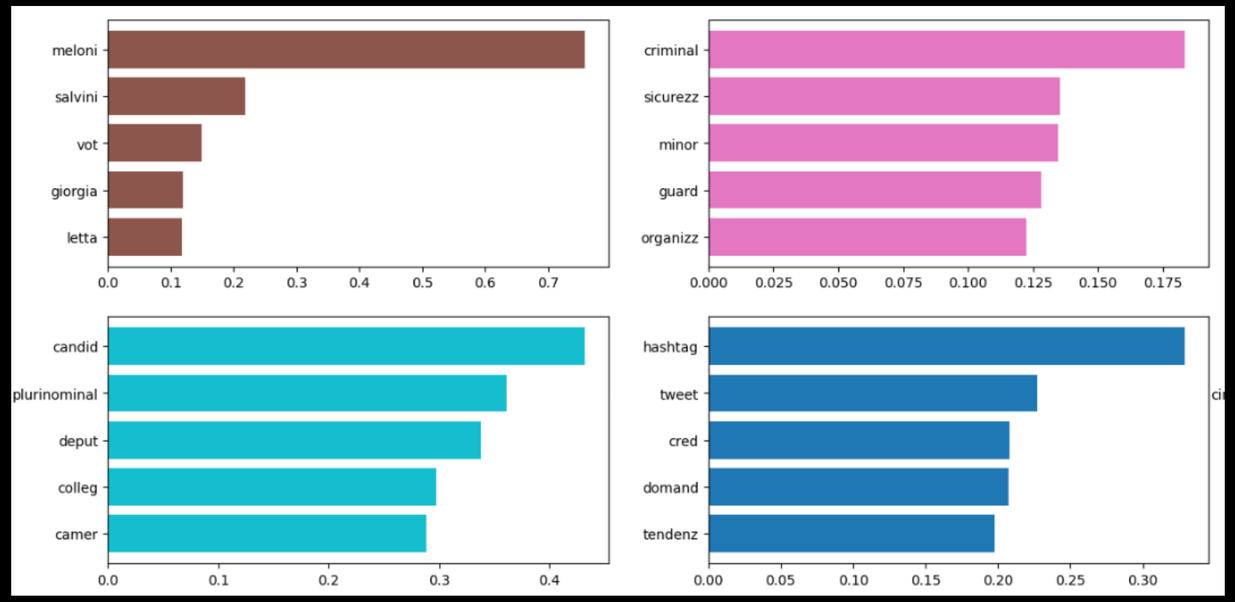
CLUSTERS - EXAMPLES

We can now leverage TF-IDF in order to highlight the most representative words for different clusters in order to give them a clear meaning



TF-IDF

BERT



BERTOPIC

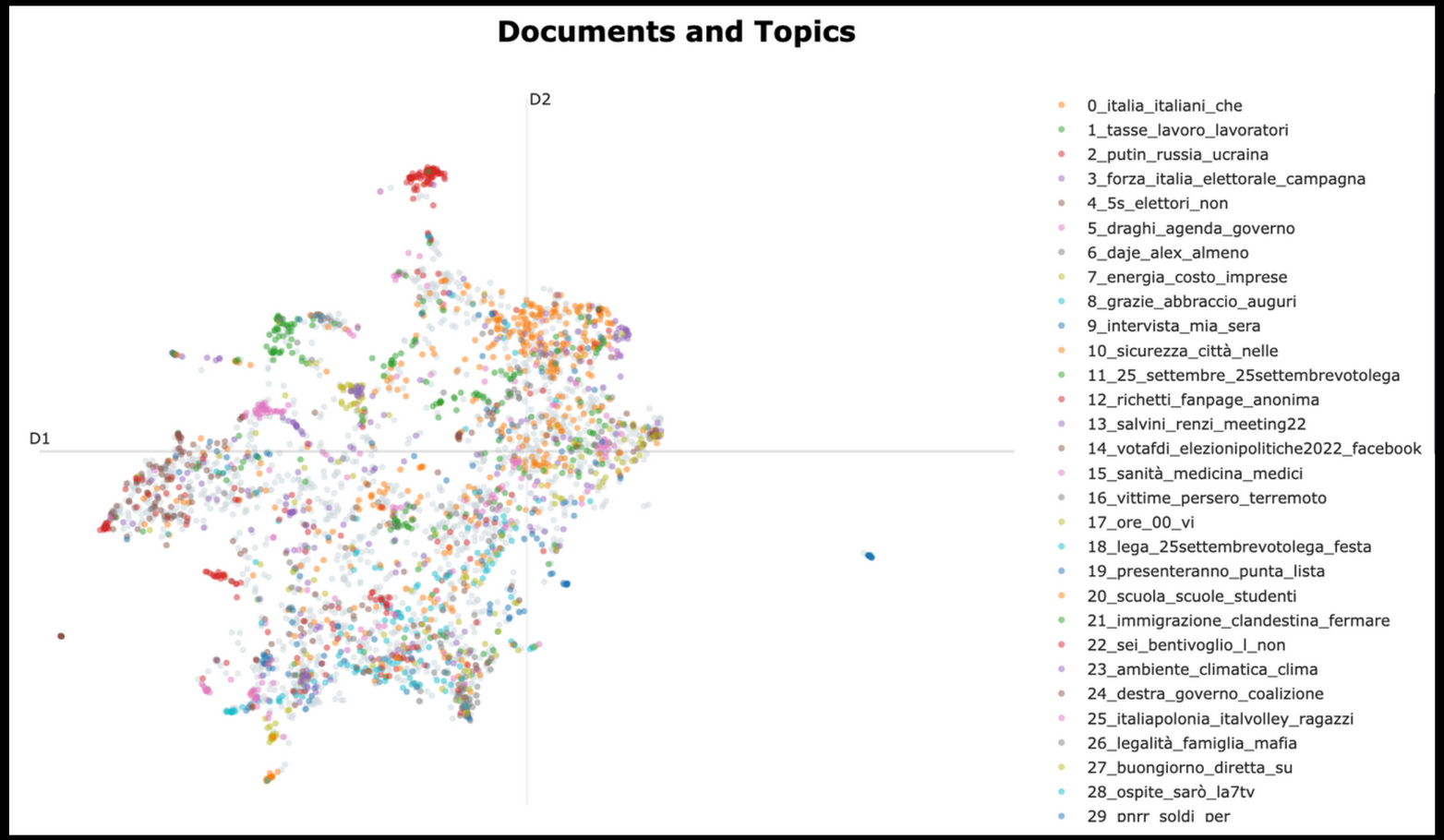
It should be noted that there exists libraries which already perform analysis like the one before with little to no effort.

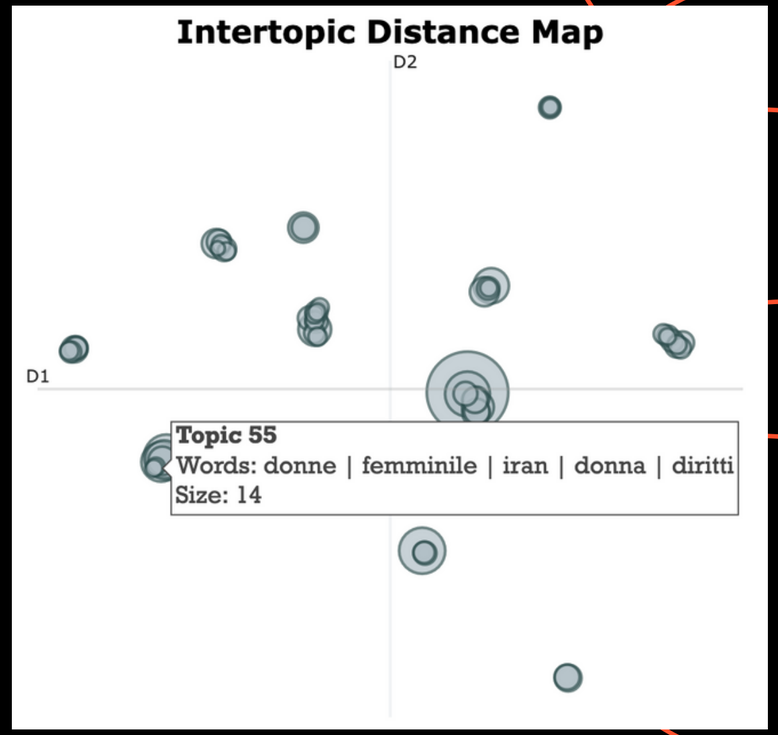
BERTopic is a topic modeling technique that leverages transformers and c-TF-IDF to create dense clusters allowing for easily interpretable topics whilst keeping important words in the topic descriptions.

```
[36] from bertopic import BERTopic

     bertopic_model = BERTopic(language="multilingual",
                               calculate_probabilities=True, verbose=False)
     topics, probs = bertopic_model.fit_transform(all_tweets_original_text)
```

BERTOPIC

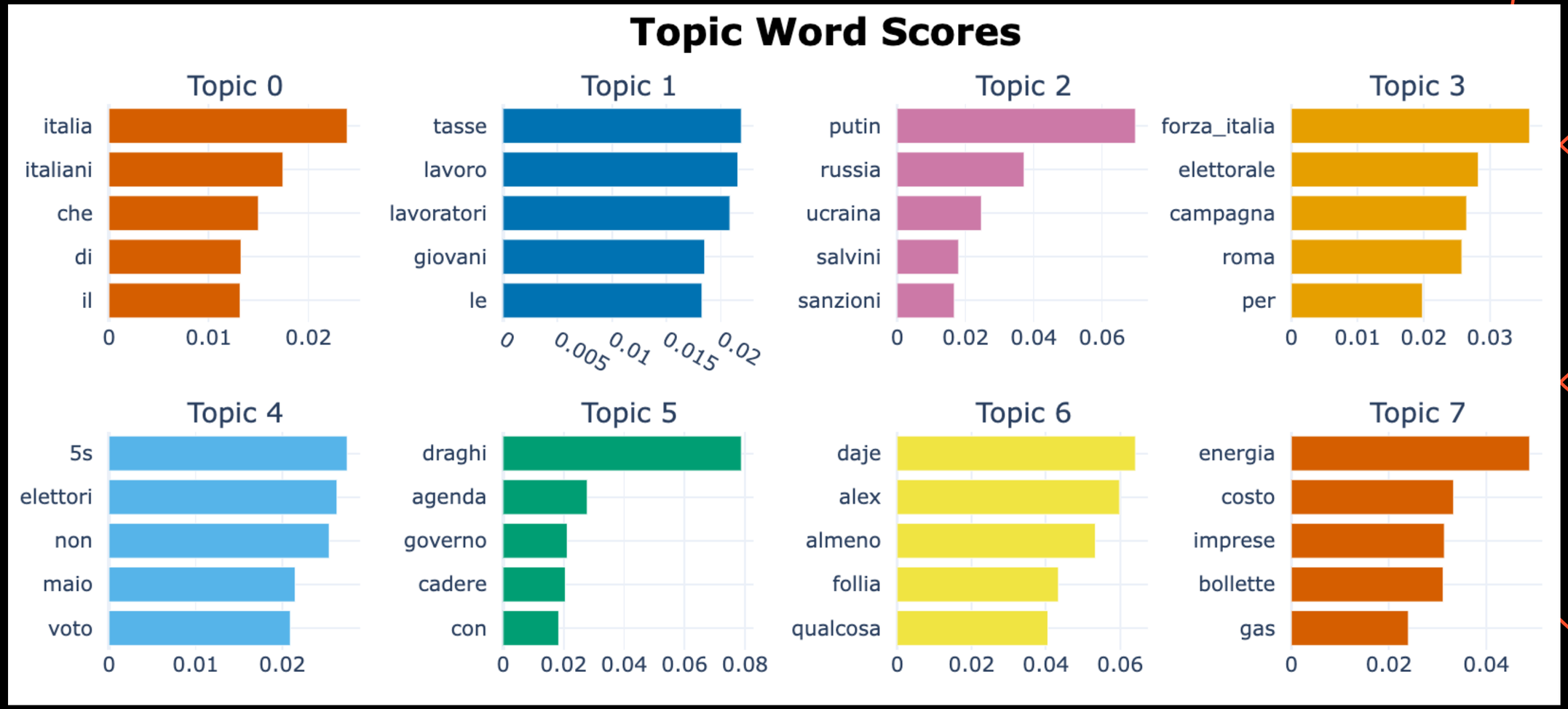
Because this library is developed for text-focused analysis, it provides excellent graphs and plots to visualize the data.





BERTOPIC

We can even easily extract the meaning of the different clusters.



LATENT DIRICHLET ALLOCATION

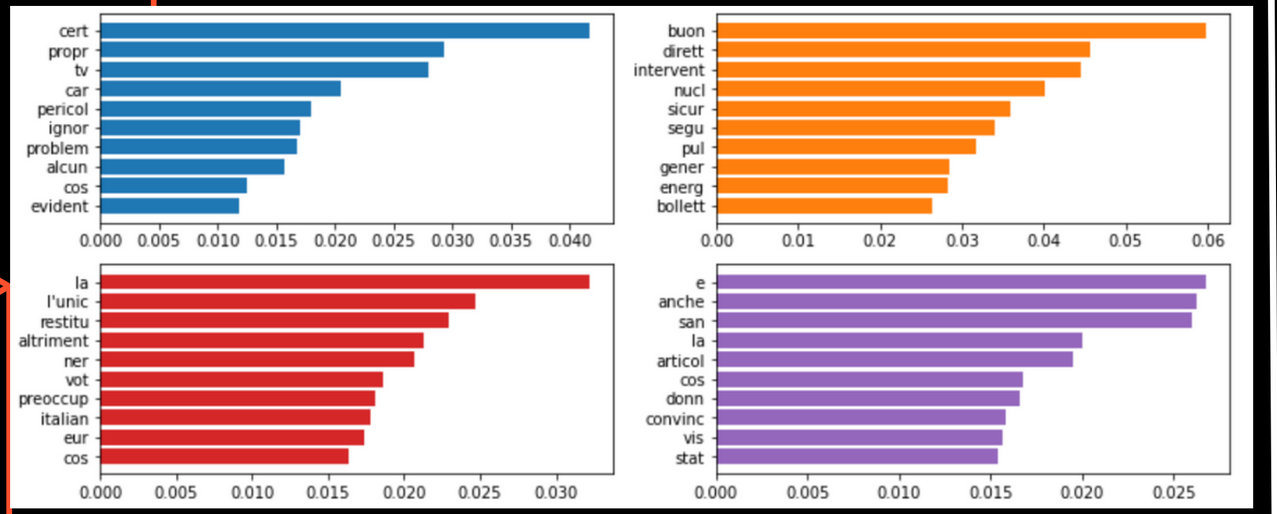
We also experimented with LDA in order to extend our analysis.
We did so using on two different approaches, Bag of Words and TF-IDF.

```
[8]  from gensim import corpora, models

     # Extract only the needed data
     data_words = list(data.text.values)

     # Create Dictionary
     dictionary = corpora.Dictionary(data_words)

     # Filter out tokens that appear in
     #   less than 10 tweets (absolute number)
     #   more than 70% of tweets
     dictionary.filter_extremes(no_below=10, no_above=0.7)

     # Compute Bag of Words and TF-IDF embedding
     corpus_bow = [dictionary.doc2bow(text) for text in data_words]
     corpus_tfidf = models.TfidfModel(corpus_bow)[corpus_bow]
```

```
[13] # Build LDA models
     lda_model_bow = models.LdaMulticore(corpus=corpus_bow, id2word=dictionary,
                                         random_state=42, passes=10)

     lda_model_tfidf = models.LdaMulticore(corpus=corpus_tfidf, id2word=dictionary,
                                           random_state=42, passes=10)
```

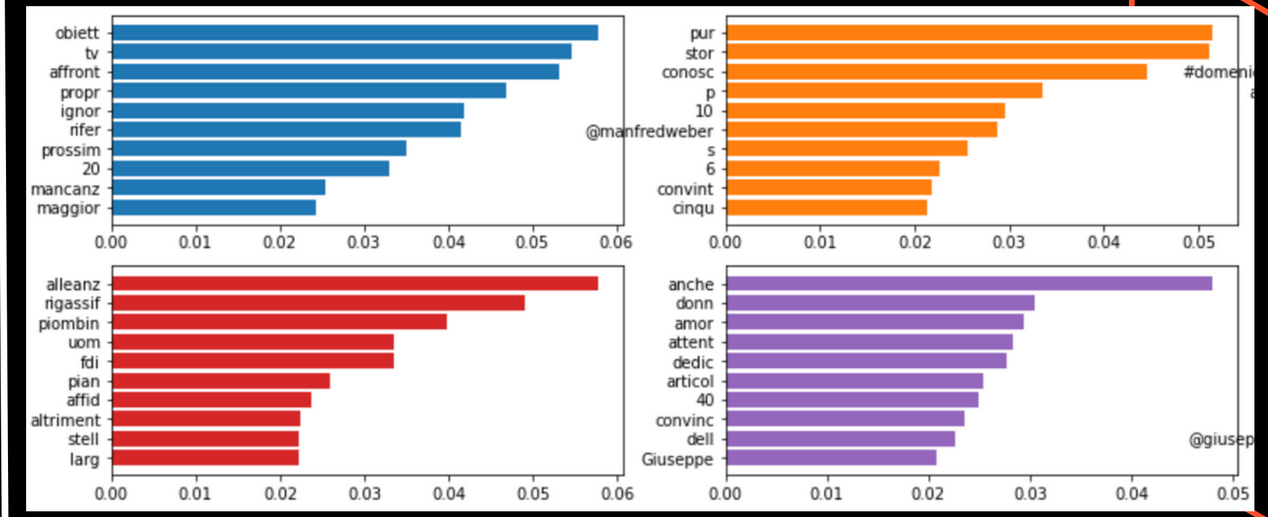
LATENT DIRICHLET ALLOCATION

We can leverage the same approach used before in order to understand the meaning of the cluster generated by this technique.



BOW

TF-IDF

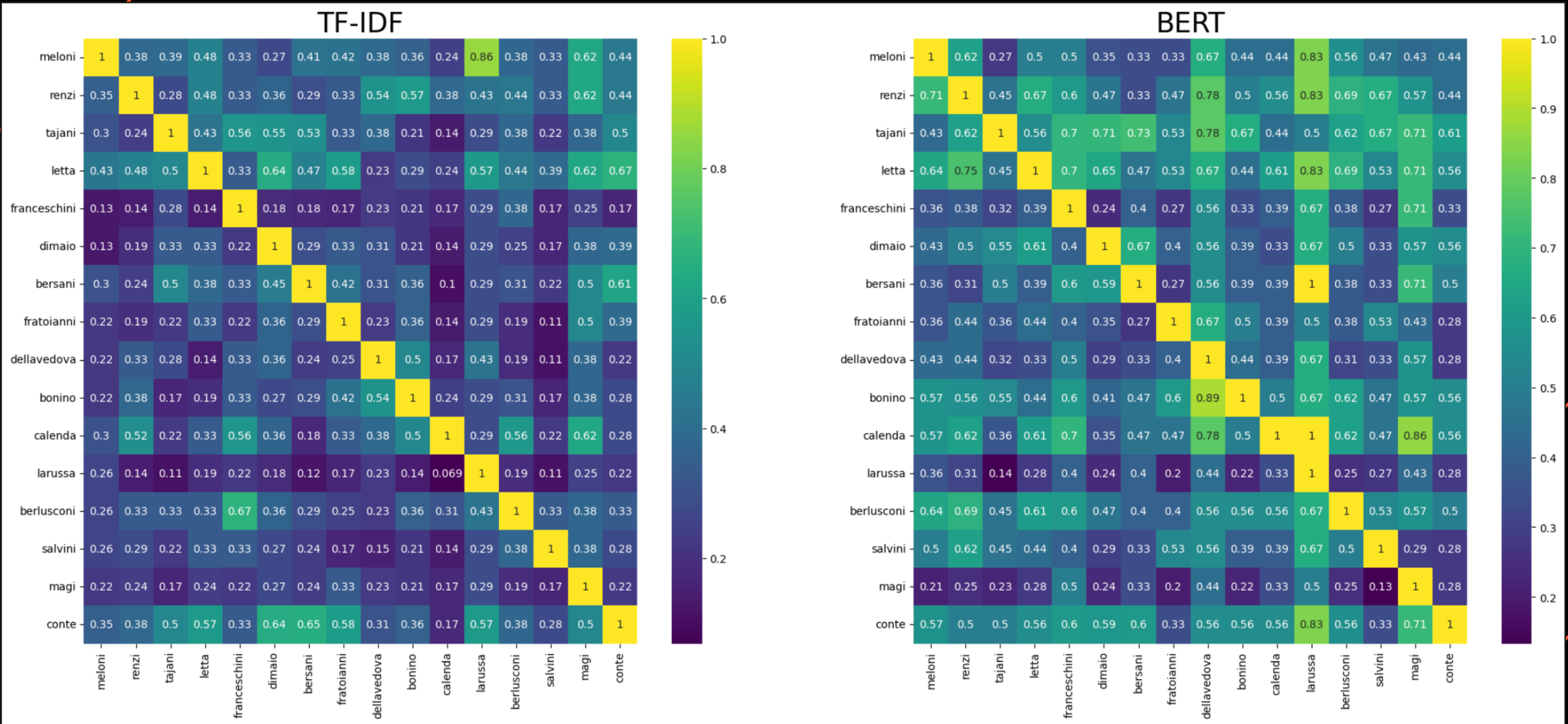


CLUSTER ANALYSIS - CORRELATION MATRIX

We started the analysis by computing a correlation matrix in order to analyse the similarity across different politicians based on the topics they speak about.
We expect to identify some similarity among politicians belonging to the same coalition, and a very low correlation across different coalitions.

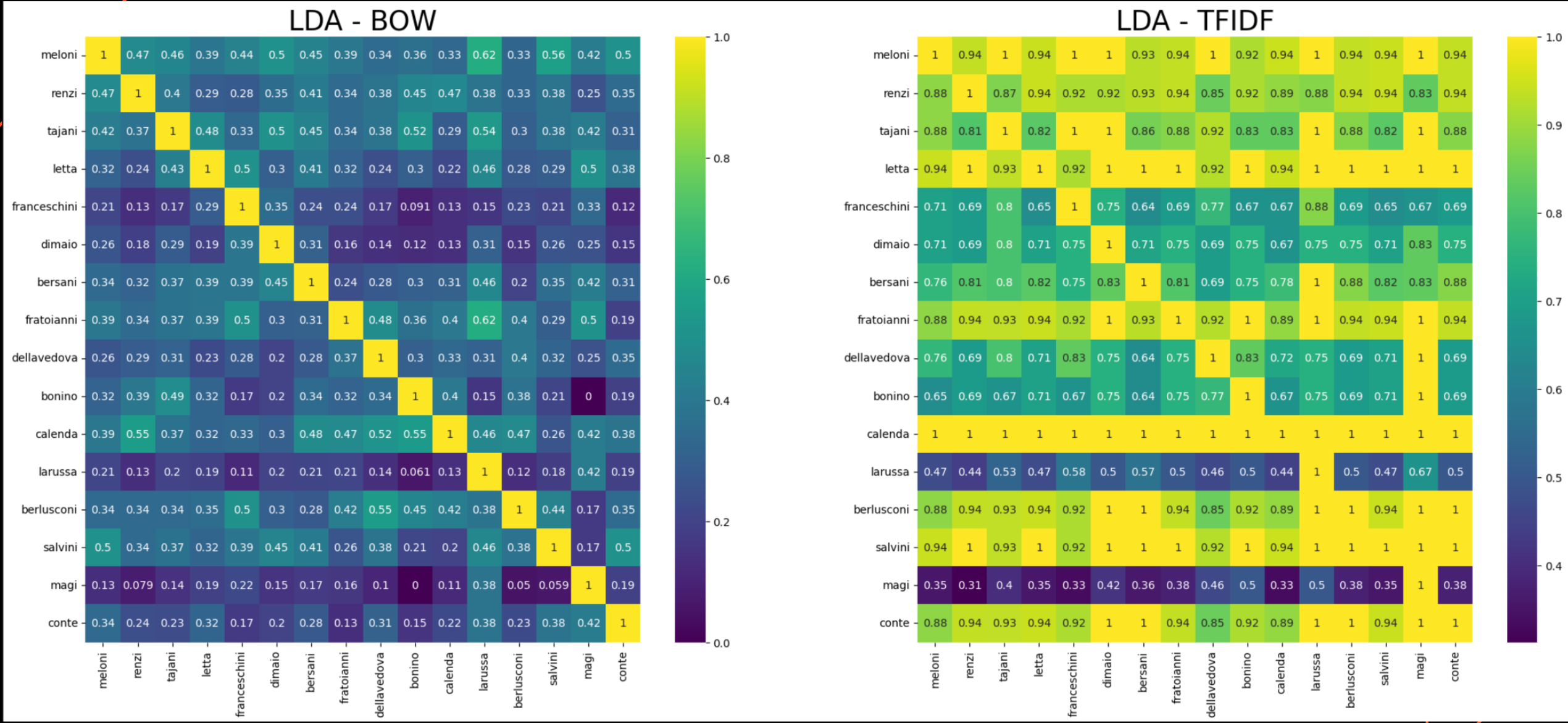
```
def prepare_correlation_values(topic_politician_tweets_df_count):
  corr = pd.DataFrame(index=politicians)

  for politician in politicians:
    politician_topics = \
      topic_politician_tweets_df_count[topic_politician_tweets_df_count["politician"] == politician].topic_id.values
    shared_topics = list()
    for other_politician in politicians:
      other_politician_topics = \
        topic_politician_tweets_df_count[topic_politician_tweets_df_count["politician"] == other_politician].topic_id.values
      shared_topics.append(len(set(politician_topics).intersection(other_politician_topics)))
    corr[politician] = np.array(shared_topics)/len(set(politician_topics))

  return corr
```

CLUSTER ANALYSIS - CORRELATION MATRIX



*NOTE that this matrix is meant to be read by row

CLUSTER ANALYSIS - CORRELATION MATRIX



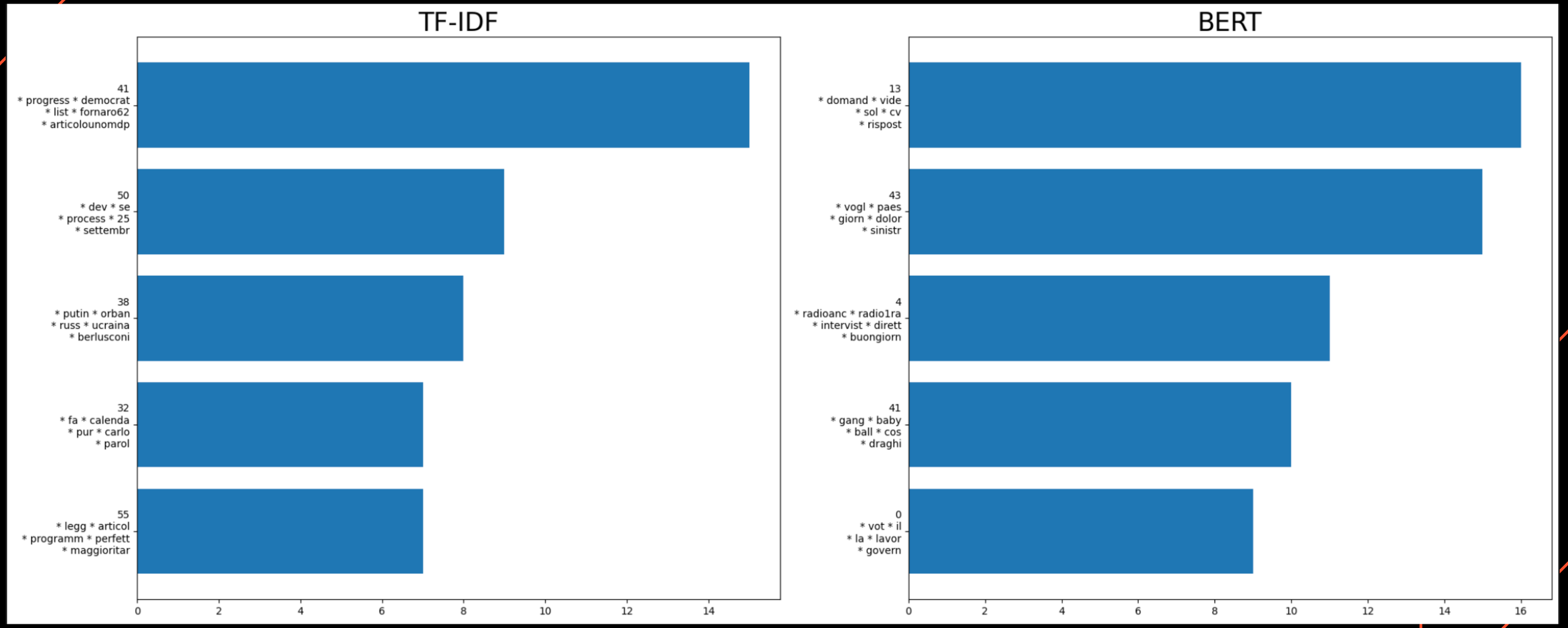
*NOTE that this matrix is meant to be read by row

CLUSTER ANALYSIS - MOST SHARED TOPIC

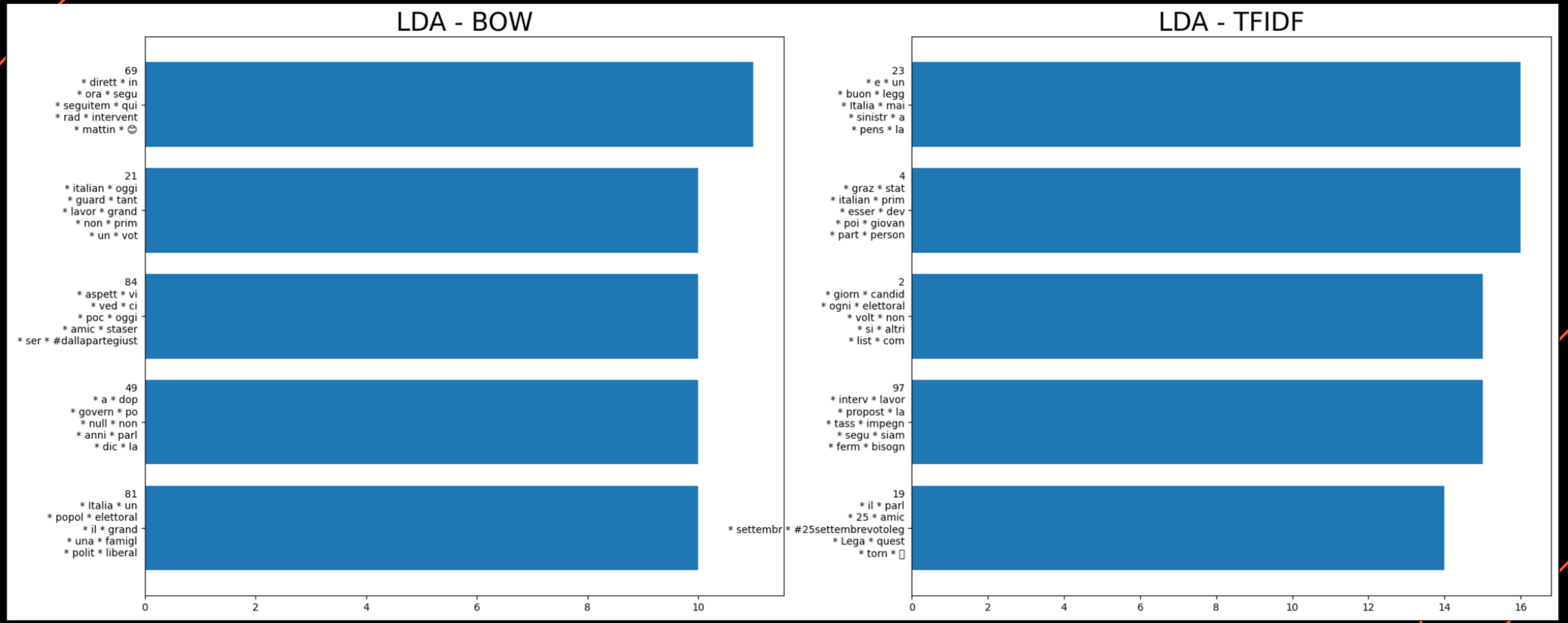
We also addressed the issue of finding the most shared topic across all the subjects

```
def prepare_shared_topic_df(topic_politician_tweets_df_count, topic_definition):
  return topic_politician_tweets_df_count\
                    .drop('tweet_count', axis=1)\
                    .groupby(['topic_id'], as_index = False)\
                    .count()\
                    .rename(columns={'politician': 'politician_count'})\
                    .merge(topic_definition, on='topic_id')\
                    .sort_values(by=["politician_count"], ascending=False)
```

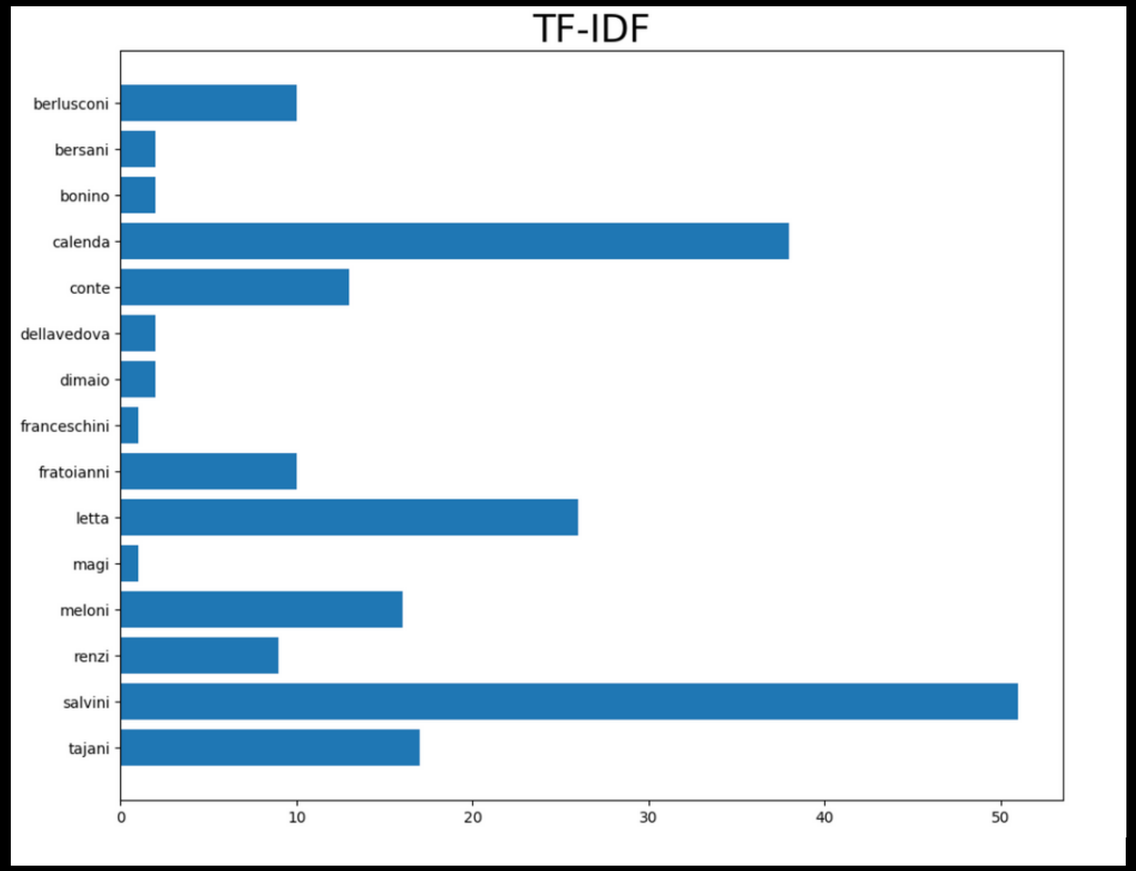
CLUSTER ANALYSIS - MOST SHARED TOPIC



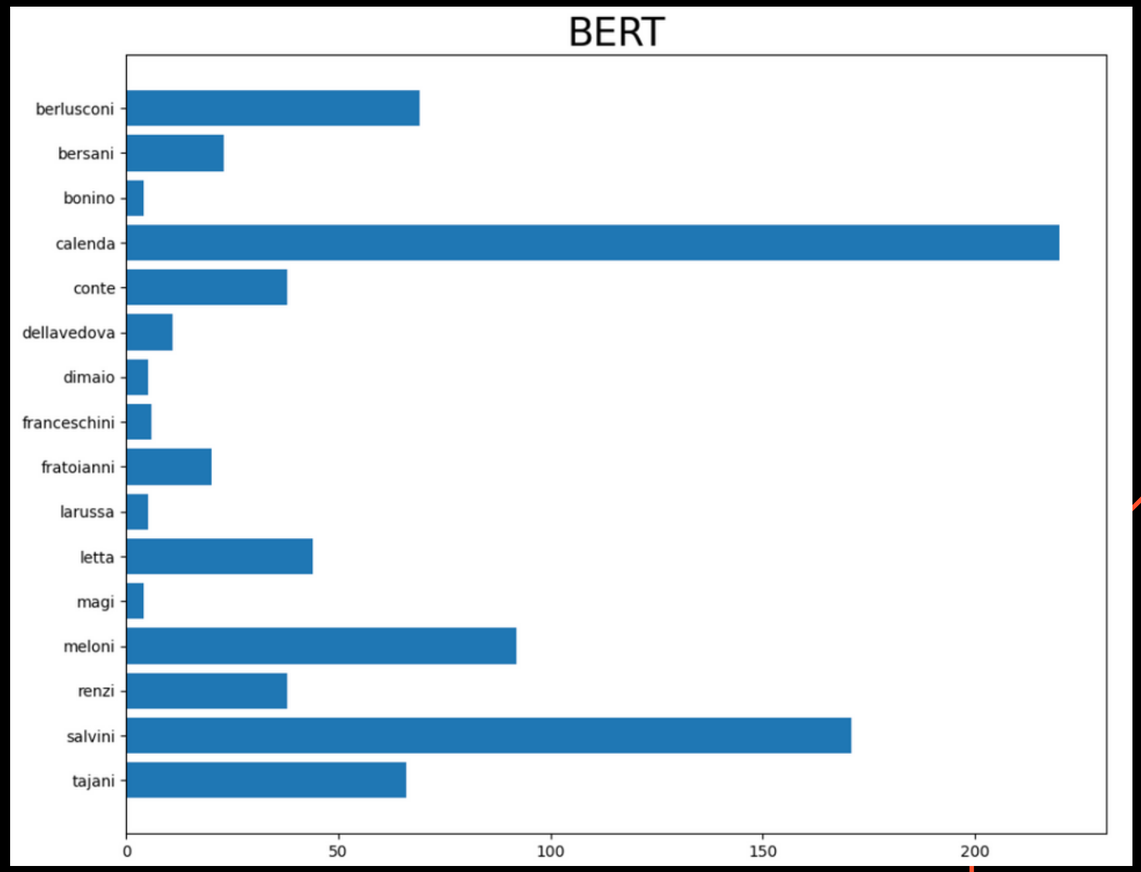
CLUSTER ANALYSIS - MOST SHARED TOPIC



CLUSTER ANALYSIS - MOST SHARED TOPIC

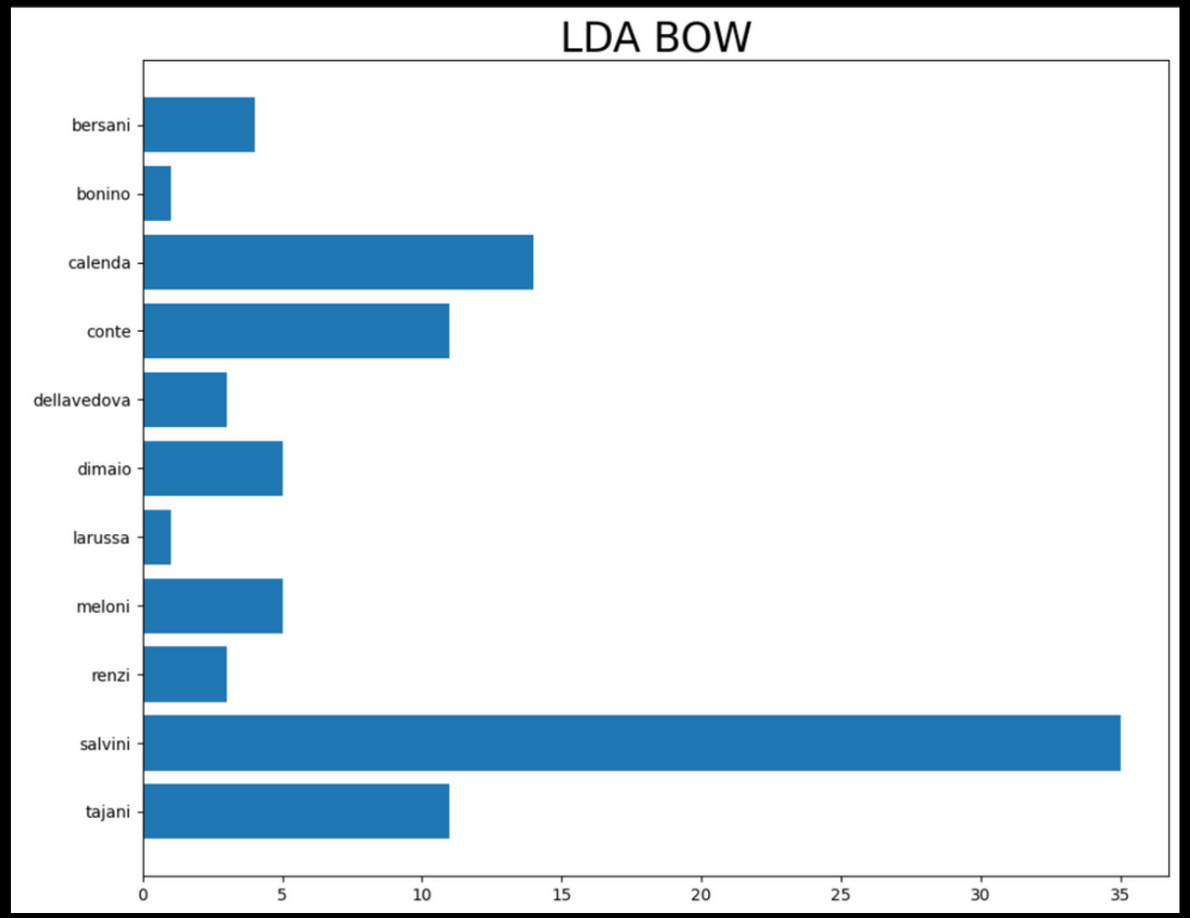


CLUSTER ID = 41

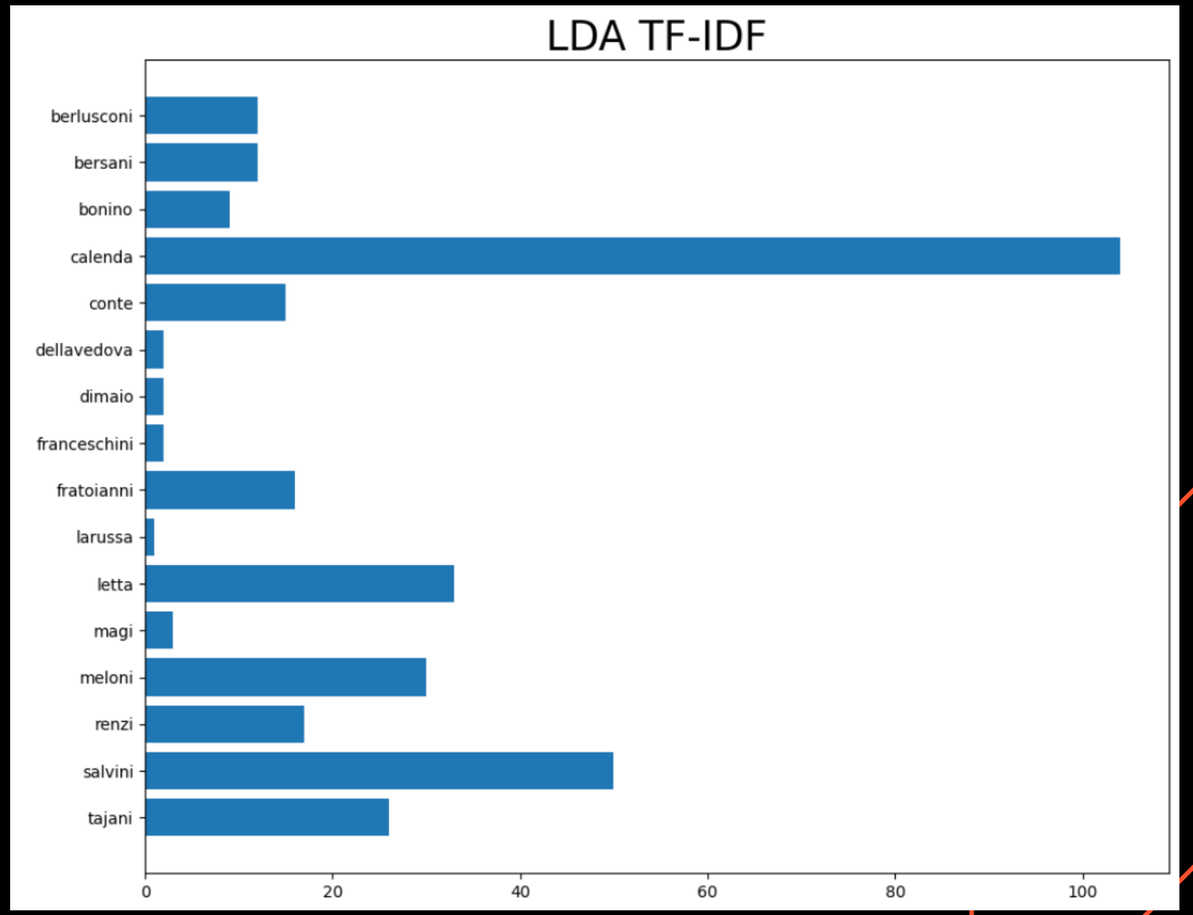


CLUSTER ID = 13

CLUSTER ANALYSIS - MOST SHARED TOPIC



CLUSTER ID = 69



CLUSTER ID = 23

CLUSTER ANALYSIS - MOST REPRESENTATIVE TOPIC

Finally we computed the most representative topic for each politician.
A representative topic for a politician would be one that he often tweeted about and others did not.

```
def compute_topic_uniqueness_by_politician(topic_politician_tweets_df_count, topic_definition):
  dfs = list()

  for politician in politicians:
    politician_tweets = topic_politician_tweets_df_count[topic_politician_tweets_df_count["politician"] == politician]\
                          .rename(columns={'tweet_count': 'politician_tweet_count'})

    other_politicians_tweets = topic_politician_tweets_df_count[topic_politician_tweets_df_count["politician"] != politician]\
                               .groupby(['topic_id'], as_index = False)\
                               .sum().rename(columns={'tweet_count': 'other_politicians_tweet_count'})

    merged_df = politician_tweets.merge(other_politicians_tweets, on='topic_id')
    merged_df["representation_score"] = merged_df.politician_tweet_count/merged_df.other_politicians_tweet_count
    merged_df["politician"] = politician
    dfs.append(merged_df)

  return pd.concat(dfs, axis=0)\
                     .merge(topic_definition, on='topic_id')\
                     .sort_values(by=["politician", "representation_score"], ascending=False)
```

CLUSTER ANALYSIS - MOST REPRESENTATIVE TOPIC

TF-IDF

politician	topic_id	politician_tweet_count	other_politicians_tweet_count	representation_score	definition
berlusconi	52	49	26	1.884615	52 - we european peopl family energy
bersani	40	14	5	2.800000	40 - scuol ragazz scolast lup bocc
bonino	62	7	13	0.538462	62 - poc tra interv dirett rtl1025
calenda	9	20	1	20.000000	9 - sud pont pnrr strett mezzogiorn
conte	64	23	2	11.500000	64 - vot terz pol tesser pd
dellavedova	37	19	12	1.583333	37 - tour gandolf velletr 25settembrevotoforza...
dimaio	26	19	19	1.000000	26 - sanzion luc gas bollett impres
franceschini	8	1	10	0.100000	8 - pan morningnews latt azzer past
fratoianni	58	15	5	3.000000	58 - ser appunt interv portaaport tg2post
larussa	55	4	17	0.235294	55 - legg articol programm perfett maggioritar
letta	18	29	10	2.900000	18 - ester moral italian gen fdi
magi	5	1	6	0.166667	5 - sì infrastruttur ravenna brescia tempa
meloni	42	14	2	7.000000	42 - vittim grand dolor famigl un
renzi	56	3	6	0.500000	56 - interv tg1 la corr tir
salvini	44	50	4	12.500000	44 - 00 18 piazz gianicol campagn
tajani	45	15	1	15.000000	45 - dirett in seguitem domenicavotoleg rai

CLUSTER ANALYSIS - MOST REPRESENTATIVE TOPIC

BERT

politician	topic_id	politician_tweet_count	other_politicians_tweet_count	representation_score	definition
berlusconi	52	49	26	1.884615	52 - we european peopl family energy
bersani	40	14	5	2.800000	40 - scuol ragazz scolast lup bocc
bonino	62	7	13	0.538462	62 - poc tra interv dirett rtl1025
calenda	9	20	1	20.000000	9 - sud pont pnrr strett mezzogiorn
conte	64	23	2	11.500000	64 - vot terz pol tesser pd
dellavedova	37	19	12	1.583333	37 - tour gandolf velletr 25settembrevotoforza...
dimaio	26	19	19	1.000000	26 - sanzion luc gas bollett impres
franceschini	8	1	10	0.100000	8 - pan morningnews latt azzer past
fratoianni	58	15	5	3.000000	58 - ser appunt interv portaaport tg2post
larussa	55	4	17	0.235294	55 - legg articol programm perfett maggioritar
letta	18	29	10	2.900000	18 - ester moral italian gen fdi
magi	5	1	6	0.166667	5 - sì infrastruttur ravenna brescia tempa
meloni	42	14	2	7.000000	42 - vittim grand dolor famigl un
renzi	56	3	6	0.500000	56 - interv tg1 la corr tir
salvini	44	50	4	12.500000	44 - 00 18 piazz gianicol campagn
tajani	45	15	1	15.000000	45 - dirett in seguitem domenicavotoleg rai

CLUSTER ANALYSIS - MOST REPRESENTATIVE TOPIC

LDA-BOW

politician	topic_id	politician_tweet_count	other_politicians_tweet_count	representation_score	definition
berlusconi	85	7	4	1.750000	85 - fatt già Renzi fin 👉 Italia prim ma cos ...
bersani	89	5	6	0.833333	89 - mai progress democrat Italia propost il a...
bonino	13	4	6	0.666667	13 - legg molt l'interv comment venerd il vot ...
calenda	55	43	1	43.000000	55 - Fratoianni di Maio Bonelli elettor cos pd...
conte	35	6	3	2.000000	35 - programm altro stat stip impres cittadin ...
dellavedova	0	3	3	1.000000	0 - @rtl1025 Letta interv programm vot garant ...
dimaio	93	2	6	0.333333	93 - stess con strad vot 💪 coalizion candid l...
franceschini	79	3	20	0.150000	79 - grand Italia paes volontar chiud tutt ins...
fratoianni	63	41	29	1.413793	63 - ospit #alleanzaverdisinistr ore #elezioni...
larussa	34	1	15	0.066667	34 - min non poch ragazz miglior prim anni per...
letta	80	11	3	3.666667	80 - #scegl ore lavor priorit forz liber le fa...
magi	76	1	4	0.250000	76 - ret la 4 camb dop te cos #zonabianc aver ...
meloni	23	24	9	2.666667	23 - #votafd #melon dirett #elezionipolitiche2...
renzi	13	4	6	0.666667	13 - legg molt l'interv comment venerd il vot ...
salvini	91	33	2	16.500000	91 - #25settembrevotoleg 25 scegl settembr Leg...
tajani	40	17	5	3.400000	40 - buon amic Draghi buongiorn dirett giorn @...

CLUSTER ANALYSIS - MOST REPRESENTATIVE TOPIC

LDA - TFIDF

politician	topic_id	politician_tweet_count	other_politicians_tweet_count	representation_score	definition
berlusconi	52	49	26	1.884615	52 - we european peopl family energy
bersani	40	14	5	2.800000	40 - scuol ragazz scolast lup bocc
bonino	62	7	13	0.538462	62 - poc tra interv dirett rtl1025
calenda	9	20	1	20.000000	9 - sud pont pnrr strett mezzogiorn
conte	64	23	2	11.500000	64 - vot terz pol tesser pd
dellavedova	37	19	12	1.583333	37 - tour gandolf velletr 25settembrevotoforza...
dimaio	26	19	19	1.000000	26 - sanzion luc gas bollett impres
franceschini	8	1	10	0.100000	8 - pan morningnews latt azzer past
fratoianni	58	15	5	3.000000	58 - ser appunt interv portaaport tg2post
larussa	55	4	17	0.235294	55 - legg articol programm perfett maggioritar
letta	18	29	10	2.900000	18 - ester moral italian gen fdi
magi	5	1	6	0.166667	5 - sì infrastruttur ravenna brescia tempa
meloni	42	14	2	7.000000	42 - vittim grand dolor famigl un
renzi	56	3	6	0.500000	56 - interv tg1 la corr tir
salvini	44	50	4	12.500000	44 - 00 18 piazz gianicol campagn
tajani	45	15	1	15.000000	45 - dirett in seguitem domenicavotoleg rai

CLUSTER DISTRIBUTION COMPARISON

In order to compare the cluster distributions we merge the stemmed text of every tweet for each cluster, obtaining a single document representing each cluster

	id	text	stemmed_text	tfidf_labels	bert_labels	lda_bow_labels	lda_tfidf_labels
0	1573424323548831746	Io ce l'ho messa tutta, ma adesso tocca a voi....	io ce mess tutt adess tocc domen #25settembr g...	-1	13	65	22
1	1573417445309792267	Un grazie e un abbraccio forte a tutte le volo...	un graz abbracc fort tutt volontar volontar og...	-1	13	16	76
2	1573416354933518336	Ce l'abbiamo messa tutta in queste settimane d...	ce mess tutt settiman campagn #elezionipolitic...	-1	-1	65	89
3	1573411398562070537	Le nostre idee per cambiare il Paese.\n#Allean...	le ide camb paes #alleanzaverdisinistr #elezio...	58	-1	11	86
4	1573358784487067648	Sono le ultime ore di campagna elettorale, dom...	son ultim ore campagn elettoral domen #25sette...	-1	13	19	2
...	...	...	...	...	...	...	...

Example for some tweets

CLUSTER DISTRIBUTION COMPARISON

Using the cosine similarity across the TFIDF vectors for each cluster, it's possible to identify the most similar cluster. Let's take into consideration the max value to identify the related cluster in the comparing model.

tfidf_labels	-1	0	1	2	3	4	5	6	7	8	...
-1	0.895238	0.442953	0.065731	0.444150	0.141216	0.533446	0.379232	0.231033	0.251200	0.128458	...
0	0.255502	0.121733	0.019768	0.093864	0.008516	0.123153	0.097987	0.057677	0.064447	0.046425	...
1	0.140838	0.051655	0.007472	0.035202	0.015956	0.098245	0.055595	0.012026	0.066516	0.972456	...
2	0.219212	0.089219	0.027876	0.072592	0.022925	0.086519	0.083710	0.046710	0.057996	0.021983	...
3	0.254578	0.107822	0.032713	0.091393	0.059991	0.121857	0.115937	0.048137	0.141290	0.016251	...
...	...	...	...	...	...	...	...	...	...	...	...

```python
def compare_docs(m_from, m_to):
    vect = TfidfVectorizer()
    df = pd.DataFrame(columns=list(m_to.iloc[:,0]))
    for i in m_from.iloc[:,0]:
        string_list = m_to.iloc[:,1]
        test_string = m_from[m_from.iloc[:,0] == i].iloc[:,1]
        all_strings = list(string_list) + list(test_string)
        tfidf_matrix = vect.fit_transform(all_strings)
        similarity_scores = cosine_similarity(tfidf_matrix[-1], tfidf_matrix)
        df.loc[len(df)] = similarity_scores[0][:-1]
    df = df.set_index(m_from.iloc[:,0])
    return df
```

CLUSTER DISTRIBUTION COMPARISON

We then defined a similarity metric, that we called "Injective Index" , which aims to quantify the similarity of two different cluster's distribution, and it does so by computing the number of exact matches among two different clustering.

	tfidf_la...	stemme...	bert_la...	lda_bo...	lda_tfid...
5	4	sì fat calm ...	13	93	2
6	5	mai lavor p...	7	75	21
7	6	in period c...	13	68	21
8	7	azzer lva b...	6	77	61
9	8	attent met...	13	17	18

	bert_la...	stemme...	tfidf_la...	lda_bo...	lda_tfid...
6	5	poi Pd 5st...	-1	16	18
7	6	ha ragion ...	7	98	61
8	7	#bellacia c...	5	19	89
9	8	due perso...	1	22	84
10	9	per scriv b...	38	34	89

```python
def count_injectives(model_A, model_B):
    model_A_name = model_A.columns[0]
    model_B_name = model_B.columns[0]
    n = 0
    for i in model_A.iloc[:,0]:
        return_i = int(model_A[model_A.iloc[:,0] == i][model_B_name])
        if return_i != -1 and int(model_A[model_A.iloc[:,0] == i].iloc[:,0]) == int(model_B[model_B.iloc[:,0] == return_i][model_A_name]):
            #print('model A index:', i, ' - ', 'model B index:', return_i, ' - ', 'check:', int(model_B[model_B.iloc[:,0] == return_i].iloc[:,0]))
            n += 1
    return n
```

```python
def injective_index(model_A, model_B):
    r = 2 * count_injectives(model_A, model_B) / (len(model_A) + len(model_B) - 2)
    return r
```

CLUSTER DISTRIBUTION COMPARISON

Here's the final result.

model	tfidf_labels	bert_labels	lda_bow_labels	lda_tfidf_labels
tfidf_labels	1.000000	0.325203	0.314607	0.107527
bert_labels	0.325203	1.000000	0.193103	0.066667
lda_bow_labels	0.325843	0.193103	1.000000	0.139130
lda_tfidf_labels	0.107527	0.066667	0.156522	1.000000

NEXT STEPS

Increase the dataset size

It is clear that the time dimension is extremely relevant in a setting like the one analysed, due to the fact that social media strategies leverage trends and news to gain traction among followers.
This analysis could be performed on different timespans in order to highlight how behavioural differences change in time.

Add time into the analysis

As stated at the beginning, we only worked with - at most- 5k tweets per politicians due to limitations of the Twitter API, this limit could be removed in order to enhance the analysis

New ways to compare different cluster distributions

In this project did not analyse all the possible ways in which we could compare "cluster distributions" generated by different models. A next step could be to think of different ways to perform this analysis and compare the results with the ones obtained here