Open in app

Follow

598K Followers

Natural Language Processing — Event Extraction

Extracting events from news articles

Rodrigo Nader May 2, 2019 · 6 min read

The amount of text generated every day is mind-blowing. Millions of data feeds are published in the form of news articles, blogs, messages, manuscripts and countless more, and the ability to automatically organize and handle it is becoming indispensable.

With improvements in neural network algorithms, significant computer power increase and easy access to comprehensive frameworks, Natural Language Processing has never been so explored. One of its common applications is called Event Extraction, which is the process of gathering knowledge about periodical incidents found in texts, automatically identifying information about what happened and when it happened.

For example:

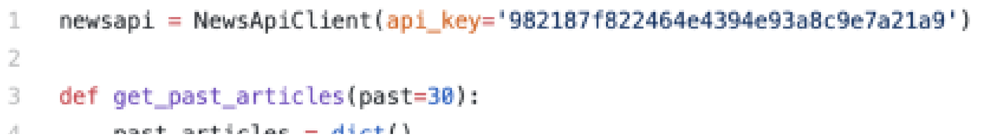
> *2018/10 — President Donald Trump's government banned countries from importing Iranian oil with exemptions to seven countries.*
>
> *2019/04 — US Secretary of State Mike Pompeo announced that his country would open no more exception after the deadline.*
>
> *2019/05 — The United States ended with exemptions that allowed countries to import oil from Iran without suffering from US sanctions.*

This ability to contextualize information allows us to connect time distributed events and assimilate their effects, and how a set of episodes unfolds through time. Those are valuable insights that drive organizations like EventRegistry and Primer.AI, which provide the technology to different market sectors.

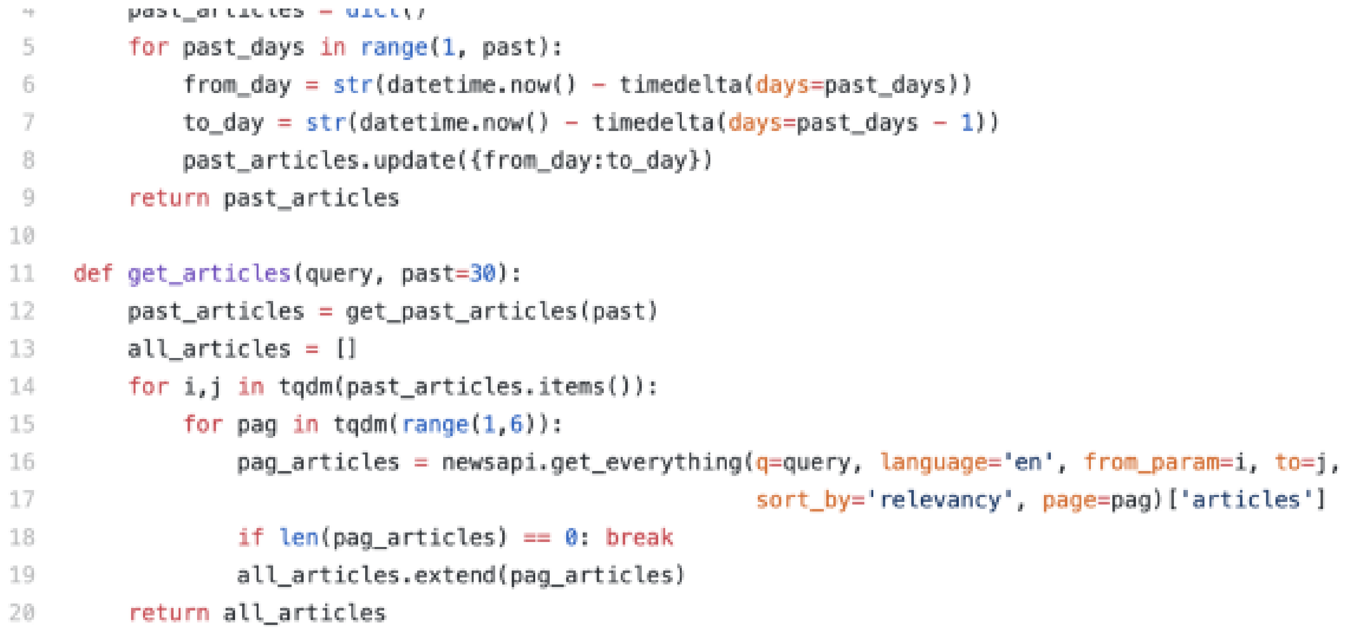
In this article, we're going to build a simple Event Extraction script that takes in news feeds and outputs the events.

Get the data

The first step for this is gathering the data. This could be any type of text as long as it can be represented in a timeline. I chose to use newsapi, since it's an easy-to-use source of news and the developer plan is free up to 500 requests a day. Following are the functions built to handle the requests:

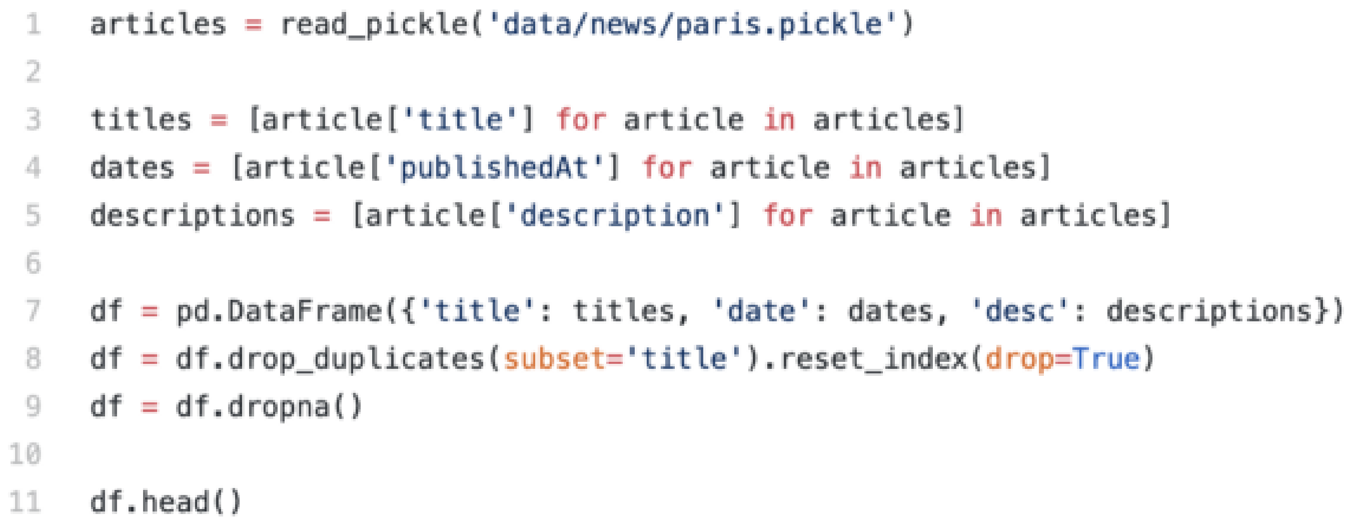


```
newsapi = NewsApiClient(api_key='982187f822464e4394e93a8c9e7a21a9')

def get_past_articles(past=30):
    past_articles = dict()
```



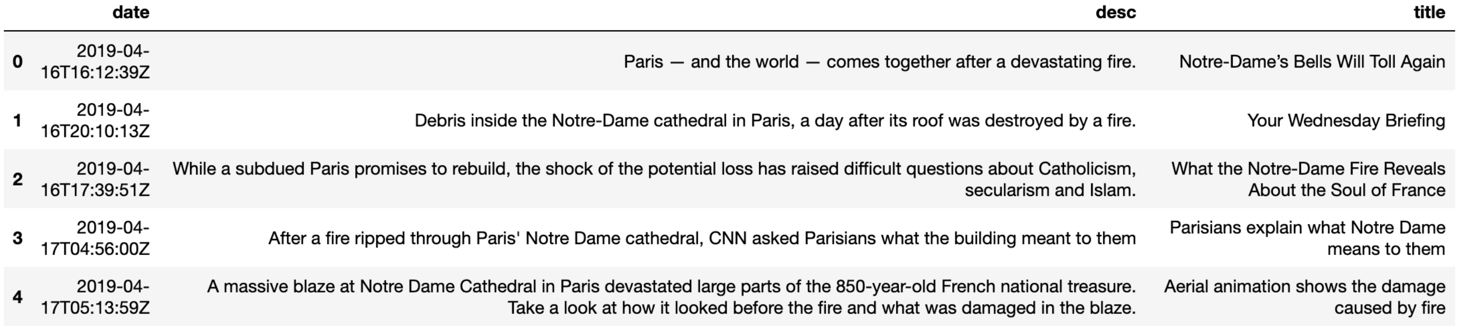
```
    past_articles = dict()
    for past_days in range(1, past):
        from_day = str(datetime.now() - timedelta(days=past_days))
        to_day = str(datetime.now() - timedelta(days=past_days - 1))
        past_articles.update({from_day:to_day})
    return past_articles

def get_articles(query, past=30):
    past_articles = get_past_articles(past)
    all_articles = []
    for i,j in tqdm(past_articles.items()):
        for pag in tqdm(range(1,6)):
            pag_articles = newsapi.get_everything(q=query, language='en', from_param=i, to=j,
                                                  sort_by='relevancy', page=pag)['articles']
            if len(pag_articles) == 0: break
            all_articles.extend(pag_articles)
    return all_articles
```

This last function returns a list of approximately 2.000 articles given a specific query. Our purpose is to extract those articles' events, so in order to simplify the process, I'm keeping only their titles (in theory, titles should already comprise the core message behind the news).



```
articles = read_pickle('data/news/paris.pickle')

titles = [article['title'] for article in articles]
dates = [article['publishedAt'] for article in articles]
descriptions = [article['description'] for article in articles]

df = pd.DataFrame({'title': titles, 'date': dates, 'desc': descriptions})
df = df.drop_duplicates(subset='title').reset_index(drop=True)
df = df.dropna()

df.head()
```

That leaves us with a data frame like the one below, including dates, descriptions, and titles.



	date	desc	title
0	2019-04-16T16:12:39Z	Paris — and the world — comes together after a devastating fire.	Notre-Dame's Bells Will Toll Again
1	2019-04-16T20:10:13Z	Debris inside the Notre-Dame cathedral in Paris, a day after its roof was destroyed by a fire.	Your Wednesday Briefing
2	2019-04-16T17:39:51Z	While a subdued Paris promises to rebuild, the shock of the potential loss has raised difficult questions about Catholicism, secularism and Islam.	What the Notre-Dame Fire Reveals About the Soul of France
3	2019-04-17T04:56:00Z	After a fire ripped through Paris' Notre Dame cathedral, CNN asked Parisians what the building meant to them	Parisians explain what Notre Dame means to them
4	2019-04-17T05:13:59Z	A massive blaze at Notre Dame Cathedral in Paris devastated large parts of the 850-year-old French national treasure. Take a look at how it looked before the fire and what was damaged in the blaze.	Aerial animation shows the damage caused by fire

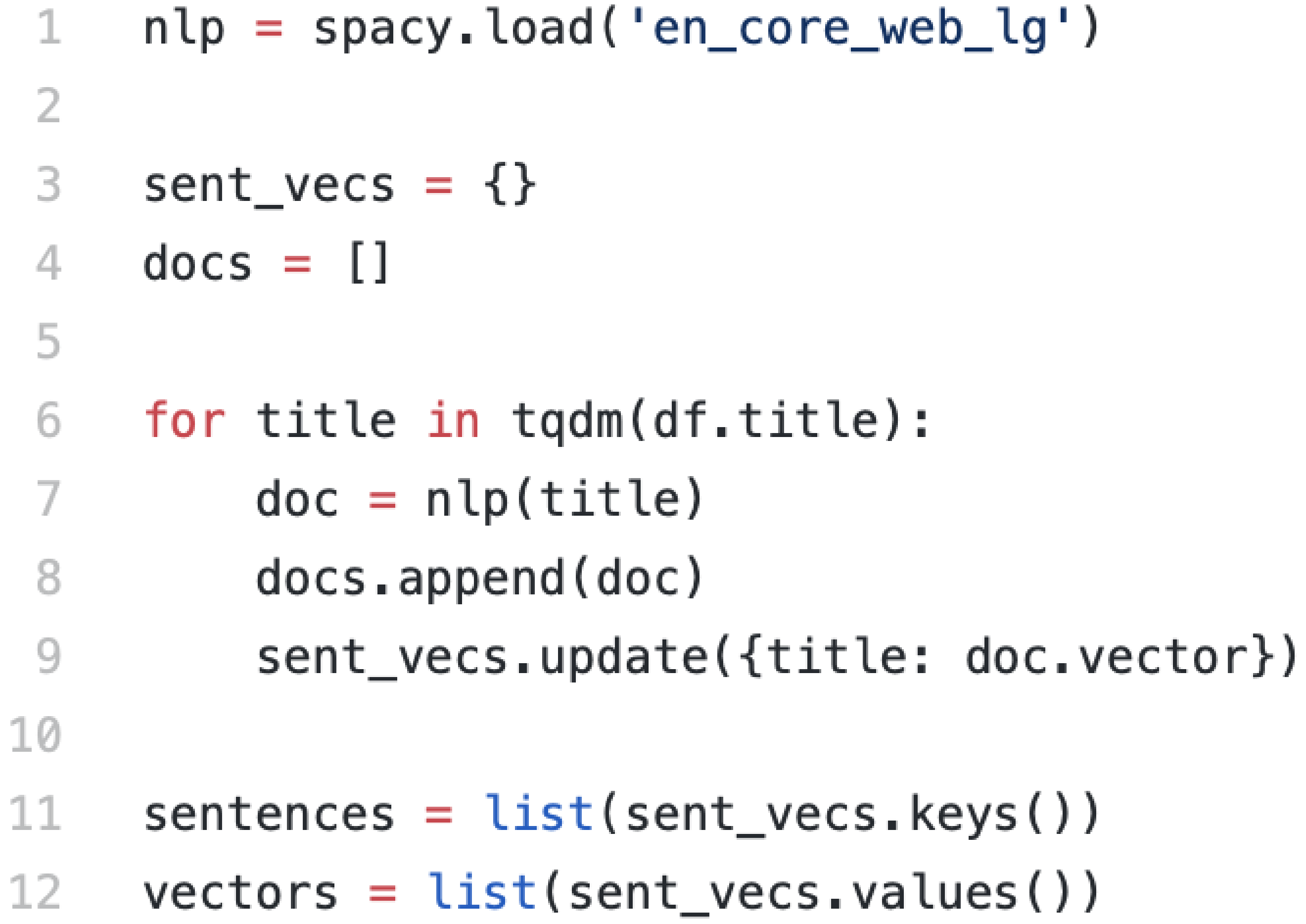
Give meaning to sentences

Now that we have our titles ready, we need to represent them in a way that our algorithms understand. Notice that I'm skipping a whole stage of pre-processing here, simply because that isn't the purpose of this article. But if you are starting with NLP, make sure to include those basic pre-processing steps before applying the models → here is a nice tutorial.

To give meaning to independent words and, consequently, whole sentences, we'll use SpaCy's pre-trained word embeddings models. More specifically, SpaCy's large model (en_core_web_lg), which has pre-trained word vectors for 685k English words. Alternatively, you could be using any pre-trained word representation model (Word2Vec, FastText, GloVe…).

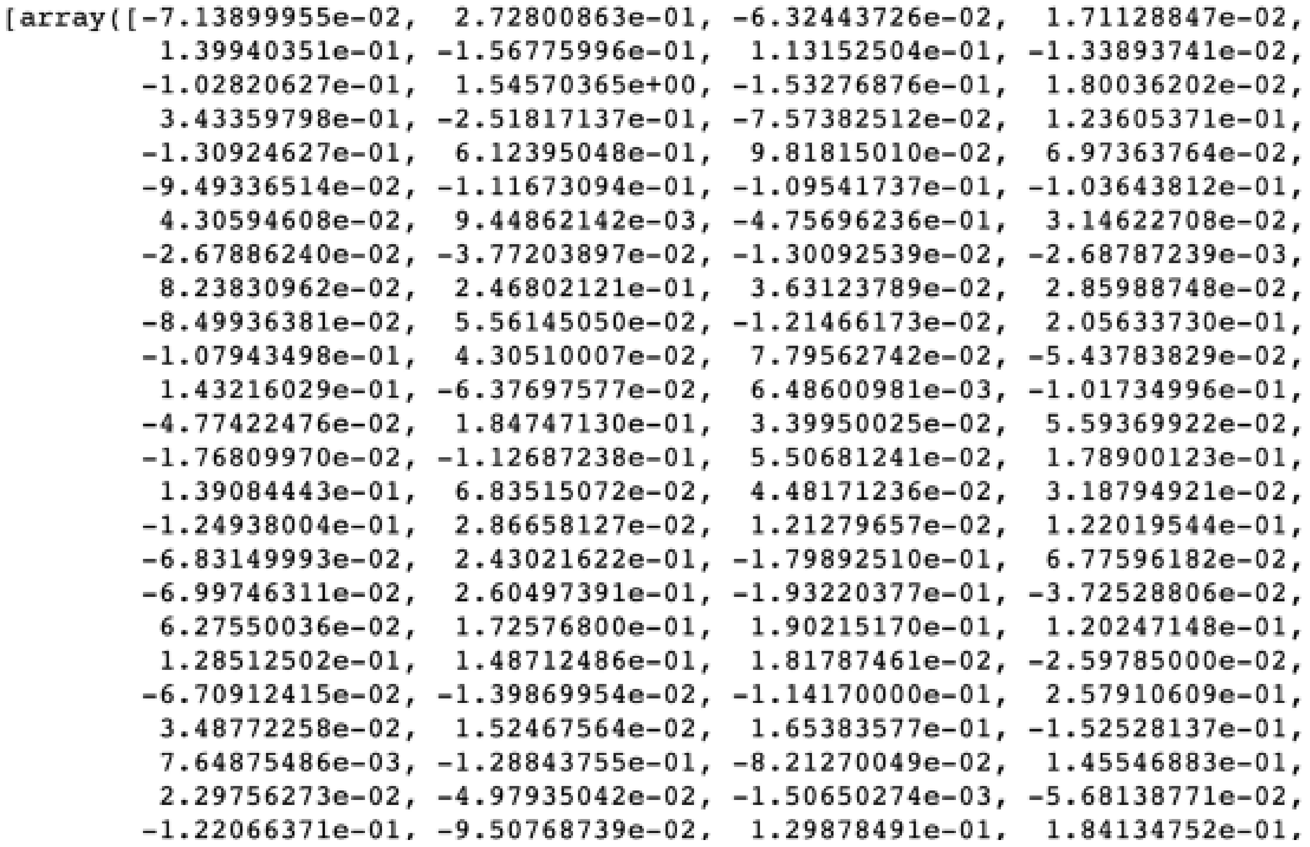
By default, SpaCy considers a sentence's vector as the average between every word's vector. It's a simplistic approach that doesn't take into account the order of words to determine a sentence's vector. For a more sophisticated strategy, we could take a look at models like Sent2Vec and SkipThoughts. This article about unsupervised summarization gives an excellent introduction to SkipThoughts.

For now, let's stick with SpaCy's method:

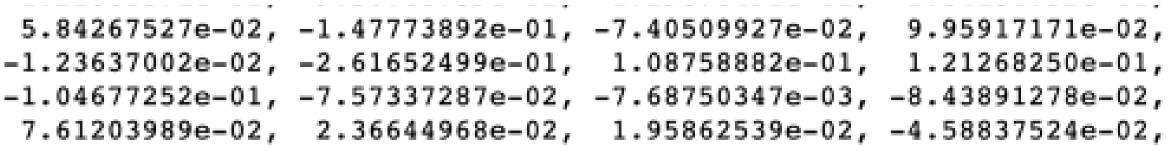


```python
nlp = spacy.load('en_core_web_lg')

sent_vecs = {}
docs = []

for title in tqdm(df.title):
    doc = nlp(title)
    docs.append(doc)
    sent_vecs.update({title: doc.vector})

sentences = list(sent_vecs.keys())
vectors = list(sent_vecs.values())
```

So each title will have a respective 300th-dimensional array, like this:



```
[array([-7.13899955e-02,  2.72800863e-01, -6.32443726e-02,  1.71128847e-02,
         1.39940351e-01, -1.56775996e-01,  1.13152504e-01, -1.33893741e-02,
        -1.02820627e-01,  1.54570365e+00, -1.53276876e-01,  1.80036202e-02,
         3.43359798e-01, -2.51817137e-01, -7.57382512e-02,  1.23605371e-01,
        -1.30924627e-01,  6.12395048e-01,  9.81815010e-02,  6.97363764e-02,
        -9.49336514e-02, -1.11673094e-01, -1.09541737e-01, -1.03643812e-01,
         4.30594608e-02,  9.44862142e-03, -4.75696236e-01,  3.14622708e-02,
        -2.67886240e-02, -3.77203897e-02, -1.30092539e-02, -2.68787239e-03,
         8.23830962e-02,  2.46802121e-01,  3.63123789e-02,  2.85988748e-02,
        -8.49936381e-02,  5.56145050e-02, -1.21466173e-02,  2.05633730e-01,
        -1.07943498e-01,  4.30510007e-02,  7.79562742e-02, -5.43783829e-02,
         1.43216029e-01, -6.37697577e-02,  6.48600981e-03, -1.01734996e-01,
        -4.77422476e-02,  1.84747130e-01,  3.39950025e-02,  5.59369922e-02,
        -1.76809970e-02, -1.12687238e-01,  5.50681241e-02,  1.78900123e-01,
         1.39084443e-01,  6.83515072e-02,  4.48171236e-02,  3.18794921e-02,
        -1.24938004e-01,  2.86658127e-02,  1.21279657e-02,  1.22019544e-01,
        -6.83149993e-02,  2.43021622e-01, -1.79892510e-01,  6.77596182e-02,
        -6.99746311e-02,  2.60497391e-01, -1.93220377e-01, -3.72528806e-02,
         6.27550036e-02,  1.72576800e-01,  1.90215170e-01,  1.20247148e-01,
         1.28512502e-01,  1.48712486e-01,  1.81787461e-02, -2.59785000e-02,
        -6.70912415e-02, -1.39869954e-02, -1.14170000e-01,  2.57910609e-01,
         3.48772258e-02,  1.52467564e-02,  1.65383577e-01, -1.52528137e-01,
         7.64875486e-03, -1.28843755e-01, -8.21270049e-02,  1.45546883e-01,
         2.29756273e-02, -4.97935042e-02, -1.50650274e-03, -5.68138771e-02,
        -1.22066371e-01, -9.50768739e-02,  1.29878491e-01,  1.84134752e-01,
```



```
 5.84267527e-02, -1.47773892e-01, -7.40509927e-02,  9.95917171e-02,
-1.23637002e-02, -2.61652499e-01,  1.08758882e-01,  1.21268250e-01,
-1.04677252e-01, -7.57337287e-02, -7.68750347e-03, -8.43891278e-02,
 7.61203989e-02,  2.36644968e-02,  1.95862539e-02, -4.58837524e-02,
```

Cluster those vectors

Even though we are filtering our articles by a search term, many topics can arise for the same query. For example, searching for “Paris” could result in:

> *Paris comes together after a devastating fire*

Or:

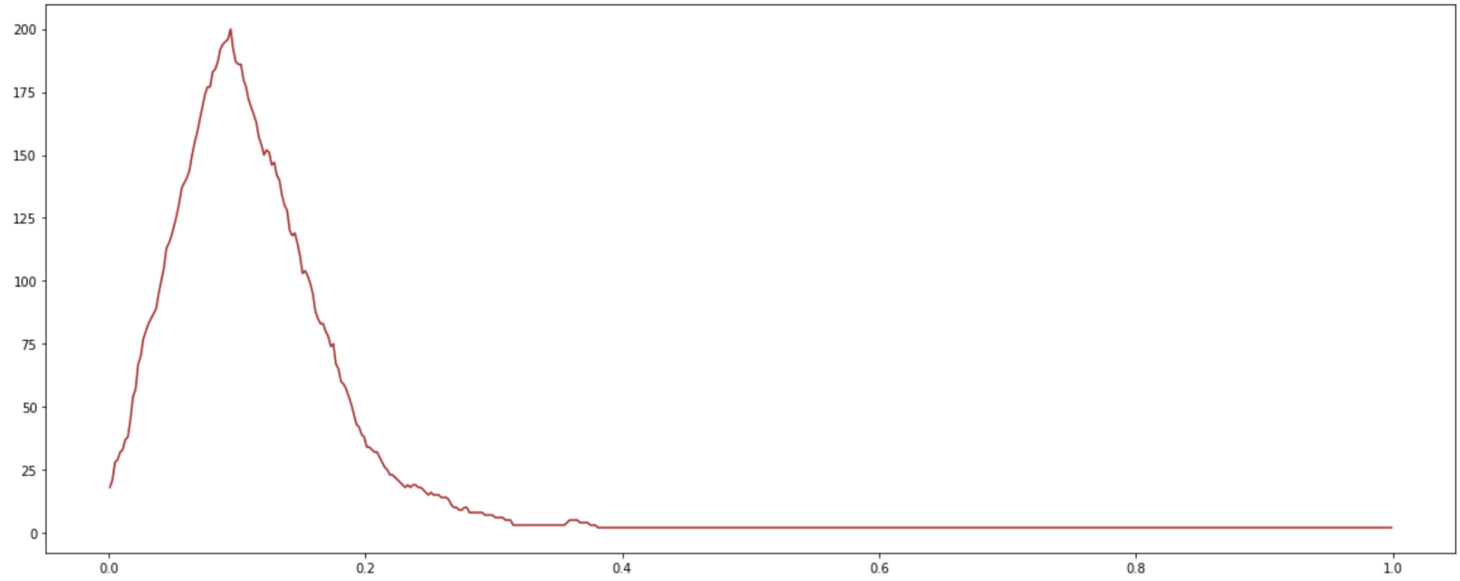
> *Brazil football legend Pele admitted to hospital in Paris*

To group articles from different topics, we’ll use a clustering algorithm.

In this particular case, I wanted to try the DBSCAN algorithm, because it doesn’t require us to previously specify the number of clusters. Instead, it determines by itself how many clusters to create and their sizes.

```
x = np.array(vectors)

n_classes = {}

for i in tqdm(np.arange(0.001, 1, 0.002)):
    dbscan = DBSCAN(eps=i, min_samples=2, metric='cosine').fit(x)
    n_classes.update({i: len(pd.Series(dbscan.labels_).value_counts())})

dbscan = DBSCAN(eps=0.08, min_samples=2, metric='cosine').fit(x)
```

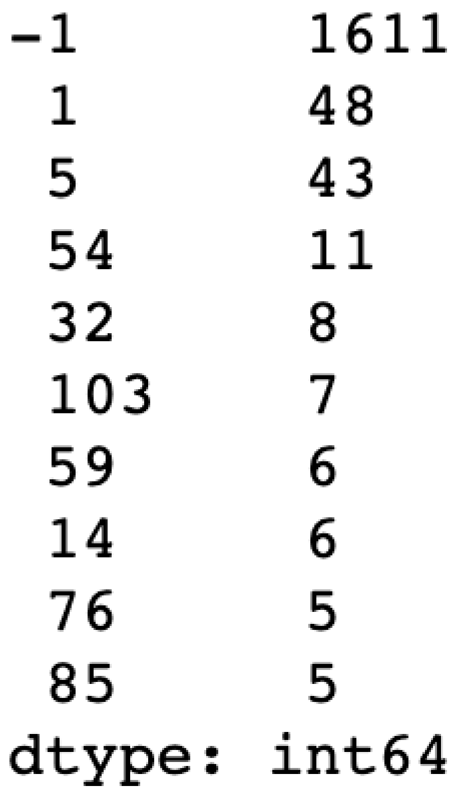
The *epsilon* parameter determines the maximum distance between two samples for them to be considered as in the same neighborhood, meaning that if *eps* is too big, fewer clusters will be formed, but also if it’s too small, most of the points will be classified as not belonging to a cluster (-1), which will result in a few clusters as well. Here is a chart showing the number of clusters by epsilon:



Tunning *eps* value might be one of the most delicate steps because the outcome will vary a lot depending on how much you want to consider sentences as similar. The right value will come up with experimentation, trying to find a value that preserves the similarities between sentences without splitting close sentences into different groups.

In general, since we want to end up with very similar sentences in the same cluster, the target should be a value that returns a higher number of classes. For that reason, I chose a number between 0.08 and 0.12. Check out Scikit Learn documentation to find more about *eps* and other parameters.

Now we can check the size of each cluster:

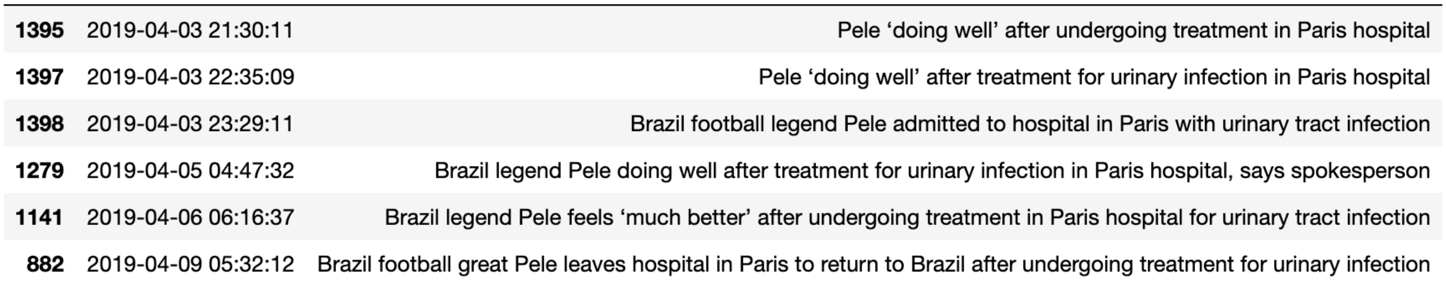


```
-1       1611
 1       48
 5       43
 54      11
 32      8
 103     7
 59      6
 14      6
 76      5
 85      5
dtype: int64
```

The **-1** class stands for sentences with no cluster, while the others are cluster indexes. If we analyze the biggest clusters, we find that those should represent the most important topics (or at least the most commented ones).

Let’s check out one of the clusters:

```
results = pd.DataFrame({'label': dbscan.labels_, 'sent': sentences})
example_result = results[results.label == 59].sent.tolist()
event_df = df[df.title.isin(example_result)][['date', 'title']]
event_df['date'] = pd.to_datetime(event_df.date)
event_df = event_df.sort_values(by='date').dropna()
```



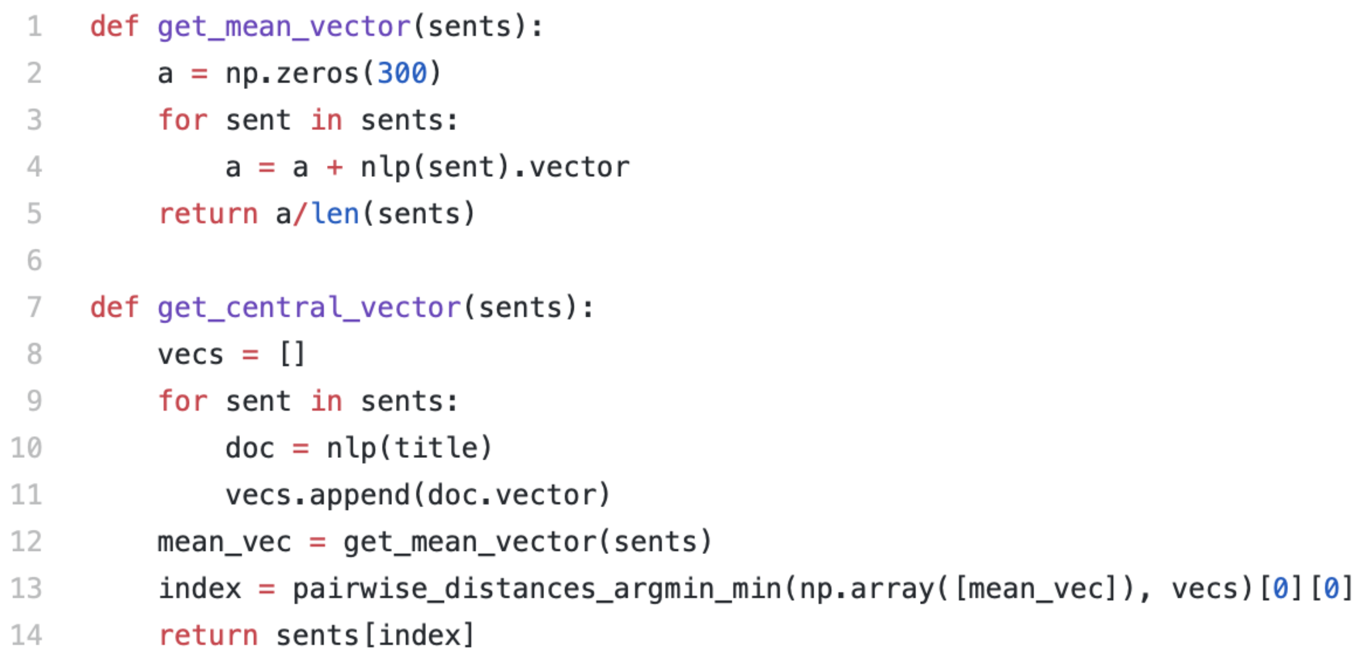
1395	2019-04-03 21:30:11	Pele ‘doing well’ after undergoing treatment in Paris hospital
1397	2019-04-03 22:35:09	Pele ‘doing well’ after treatment for urinary infection in Paris hospital
1398	2019-04-03 23:29:11	Brazil football legend Pele admitted to hospital in Paris with urinary tract infection
1279	2019-04-05 04:47:32	Brazil legend Pele doing well after treatment for urinary infection in Paris hospital, says spokesperson
1141	2019-04-06 06:16:37	Brazil legend Pele feels ‘much better’ after undergoing treatment in Paris hospital for urinary tract infection
882	2019-04-09 05:32:12	Brazil football great Pele leaves hospital in Paris to return to Brazil after undergoing treatment for urinary infection

Transform to Events

We end up with a data frame like the one above for each cluster. Next step is to arrange those sentences in time and to filter them by relevance. I chose to display one article per day so that the timeline is clean and consistent.

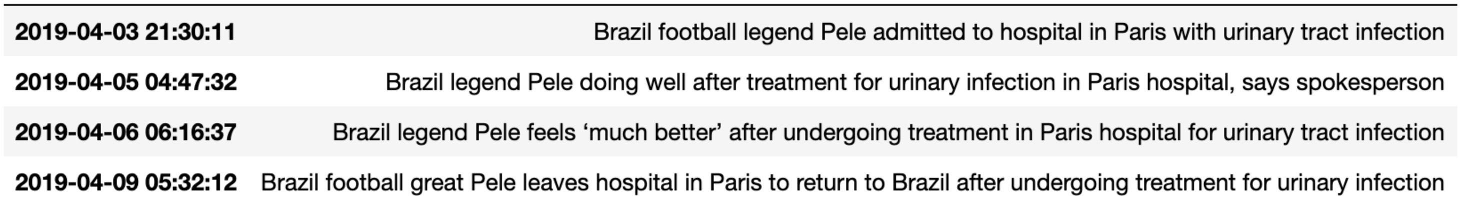
Since there are many titles about the same topic every day, we need a criterium to pick one among them. It should be the sentence that best represents the event, one that comprises the core message which those titles refer to.

In order to achieve that, we can group the daily sentences, and for each group (or cluster), choose the one closest to the cluster center. Here are the functions to find the central vector given a list of sentences:



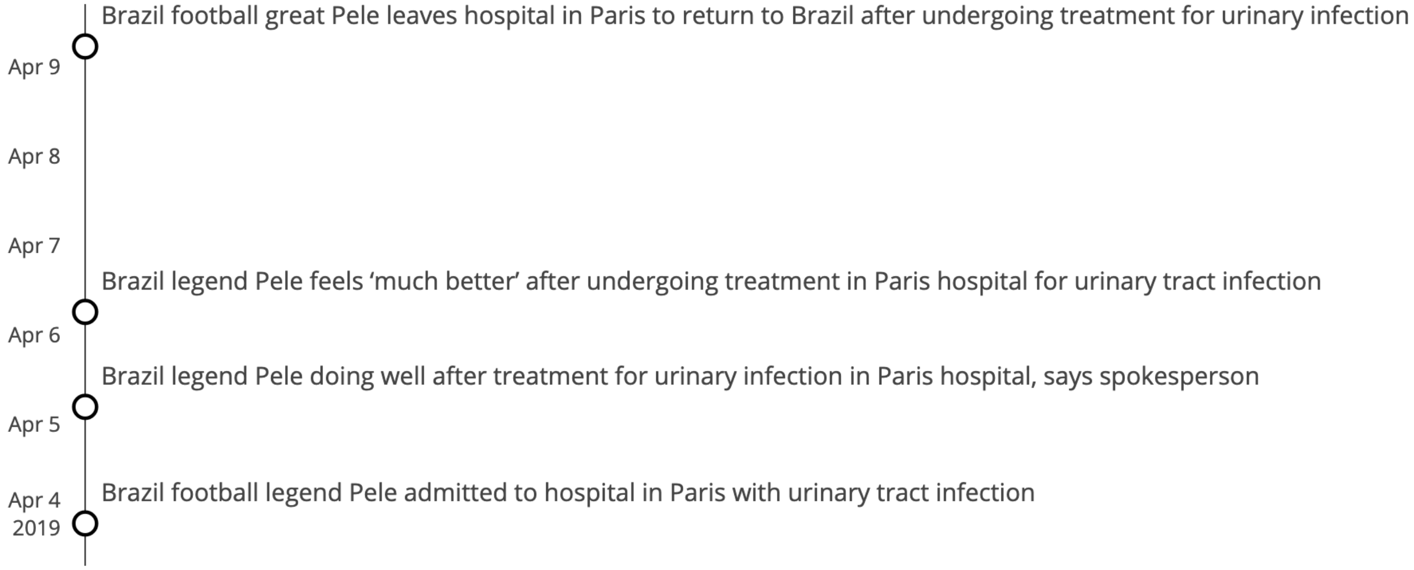
```
def get_mean_vector(sents):
    a = np.zeros(300)
    for sent in sents:
        a = a + nlp(sent).vector
    return a/len(sents)

def get_central_vector(sents):
    vecs = []
    for sent in sents:
        doc = nlp(title)
        vecs.append(doc.vector)
    mean_vec = get_mean_vector(sents)
    index = pairwise_distances_argmin_min(np.array([mean_vec]), vecs)[0][0]
    return sents[index]
```

s



2019-04-03 21:30:11	Brazil football legend Pele admitted to hospital in Paris with urinary tract infection
2019-04-05 04:47:32	Brazil legend Pele doing well after treatment for urinary infection in Paris hospital, says spokesperson
2019-04-06 06:16:37	Brazil legend Pele feels ‘much better’ after undergoing treatment in Paris hospital for urinary tract infection
2019-04-09 05:32:12	Brazil football great Pele leaves hospital in Paris to return to Brazil after undergoing treatment for urinary infection

Neat and tidy. Finally, using Plotly, we can figure out a way to plot a handy timeline chart:



That's it. Using 2.000 articles we made a script to extract and organize events. Now you can imagine how useful it may be to apply this to millions of articles every day. Just take stock markets and the impact of daily news as an example and you'll realize the value of Event Extraction.

Many steps could be included to improve the results, like properly pre-processing the data, including POS tagging and NER, applying better sentence to vector models, and so on. But starting here, a desirable result can be reached very quickly.

Thank you for reading this post. This was an article focused on NLP and Event Extraction. If you want more about Data Science and Machine Learning, make sure to follow my profile and please feel free to leave any ideas, comments or concerns.

Machine Learning NLP Data Science Big Data Data Analysis