DATA SCIENCE / PYTHON NLP SNIPPETS

From DataFrame to N-Grams

A quick-start guide to creating and visualizing n-gram ranking using nltk for natural language processing.

Ednalyn C. De Dios · Follow
Published in Towards Data Science
5 min read · May 22, 2020

Photo by Tanner Mardis on Unsplash

When I was first starting to learn NLP, I remember getting frustrated or intimidated by information overload so I've decided to write a post that covers the bare minimum. You know what they say, "Walk before you run!"

This is a very gentle introduction so we won't be using any fancy code here.

In a nutshell, natural language processing or NLP simply refers to the process of reading and understanding written or spoken language using a computer. At its simplest use case, we can use a computer to read a book, for example, and count how many times each word was used instead of us manually doing it.

NLP is a big topic and there's already been a ton of articles written on the subject so we won't be covering that here. Instead, we'll focus on how to quickly do one of the simplest but useful techniques in NLP: N-gram ranking.

N-Gram Ranking

Simply put, an n-gram is a sequence of *n* words where *n* is a discrete number that can range from 1 to infinity! For example, the word "cheese" is a 1-gram (unigram). The combination of the words "cheese flavored" is a 2-gram (bigram). Similarly, "cheese flavored snack" is a 3-gram (trigram). And "ultimate cheese flavored snack" is a 4-gram (qualgram). So on and so forth.

In n-gram ranking, we simply rank the n-grams according to how many times they appear in a body of text — be it a book, a collection of tweets, or reviews left by customers of your company.

Let's get started!

Getting the Data

First, let's get our data and load it into a dataframe. You can download the sample dataset here or create your own from the Trump Twitter Archive.

```
import pandas as pd
df = pd.read_csv('tweets.csv')
```

Using `df.head()` we can quickly get acquainted with the dataset.

```
df.head()
```

	source	text	created_at	retweet_count	favorite_count	is_retweet	id_str
0	Twitter for iPhone	LOSER! https://t.co/p5imhMJqS1	05-18-2020 14:55:14	32295	135445	False	1262396333064892416
1	Twitter for iPhone	Most of the money raised by the RINO losers of...	05-05-2020 18:18:26	19706	82425	False	1257736426206031874
2	Twitter for iPhone	....because they don't know how to win and the...	05-05-2020 04:46:34	12665	56868	False	1257532112233803782
3	Twitter for iPhone	....lost for Evan "McMuffin" McMullin (to me)....	05-05-2020 04:46:34	13855	62268	False	1257532114666508291
4	Twitter for iPhone	....get even for all of their many failures. Y...	05-05-2020 04:46:33	8122	33261	False	1257532110971318274

A sample of President Trump's tweets.

Importing Packages

Next, we'll import packages so we can properly set up our Jupyter notebook:

```
# natural language processing: n-gram ranking
import re
import unicodedata
import nltk
from nltk.corpus import stopwords

# add appropriate words that will be ignored in the analysis
ADDITIONAL_STOPWORDS = ['covfefe']


import matplotlib.pyplot as plt
```

In the code block above, we imported pandas so that we can shape and manipulate our data in all sorts of different and wonderful ways! Next, we imported `re` for regex, `unicodedata` for Unicode data, and `nltk` to help with parsing the text and cleaning them up a bit. And then, we specified additional stop words that we want to ignore. This is helpful in trimming down the noise. Lastly, we imported `matplotlib` matplotlib so we can visualize the result of our n-gram ranking later.

Next, let's create a function that will perform basic cleaning of the data.

Basic Cleaning

```
def basic_clean(text):
  """
  A simple function to clean up the data. All the words that
  are not designated as a stop word is then lemmatized after
  encoding and basic regex parsing are performed.
  """
  wnl = nltk.stem.WordNetLemmatizer()
  stopwords = nltk.corpus.stopwords.words('english') + ADDITIONAL_STOPWORDS
  text = (unicodedata.normalize('NFKD', text)
    .encode('ascii', 'ignore')
    .decode('utf-8', 'ignore')
    .lower())
  words = re.sub(r'[^\w\s]', '', text).split()
  return [wnl.lemmatize(word) for word in words if word not in stopwords]
```

The function above takes in a list of words or text as input and returns a cleaner set of words. The function does normalization, encoding/decoding, lower casing, and lemmatization.

Let's use it!

```
words = basic_clean(''.join(str(df['text'].tolist())))
```

Above, we're simply calling the function `basic_lean()` to process the `'text'` column of our dataframe `df` and making it a simple list with `tolist()`. We then assign the results to `words`.

```
words[:20]
```

```
['loser',
 'httpstcop5imhmjqs1',
 'money',
 'raised',
 'rino',
 'loser',
 'socalled',
 'lincoln',
 'project',
 'go',
 'pocket',
 'ive',
 'done',
 'judge',
 'tax',
 'regulation',
 'healthcare',
 'military',
 'vet',
 'choice']
```

A list of already cleaned, normalized, and lemmatized words.

N-grams

Here comes the fun part! In one line of code, we can find out which bigrams occur the most in this particular sample of tweets.

```
(pd.Series(nltk.ngrams(words, 2)).value_counts())[:10]
```

```
(pd.Series(nltk.ngrams(words, 2)).value_counts())[:10]
```

```
(hater, loser)        42
(total, loser)        32
(loser, hater)        14
(amp, loser)          13
(donald, trump)       11
(hater, amp)          11
(separate, winner)     8
(winner, loser)        8
(loser, like)          6
(wear, wig)            6
dtype: int64
```

We can easily replace the number 2 with 3 so we can get the top 10 trigrams instead.

```
(pd.Series(nltk.ngrams(words, 3)).value_counts())[:10]
```

```
(pd.Series(nltk.ngrams(words, 3)).value_counts())[:10]
```

```
(hater, amp, loser)              10
(separate, winner, loser)         8
(loser, person, reacts)           6
(winner, loser, person)           6
(person, reacts, new)             6
(reacts, new, twist)              6
(new, twist, fate)                6
(hater, loser, happy)             6
(everyone, including, hater)      5
(ego, ill, show)                  5
dtype: int64
```


Voilà! We got ourselves a great start. But why stop now? Let's try it and make a little eye candy.

Bonus Round: Visualization

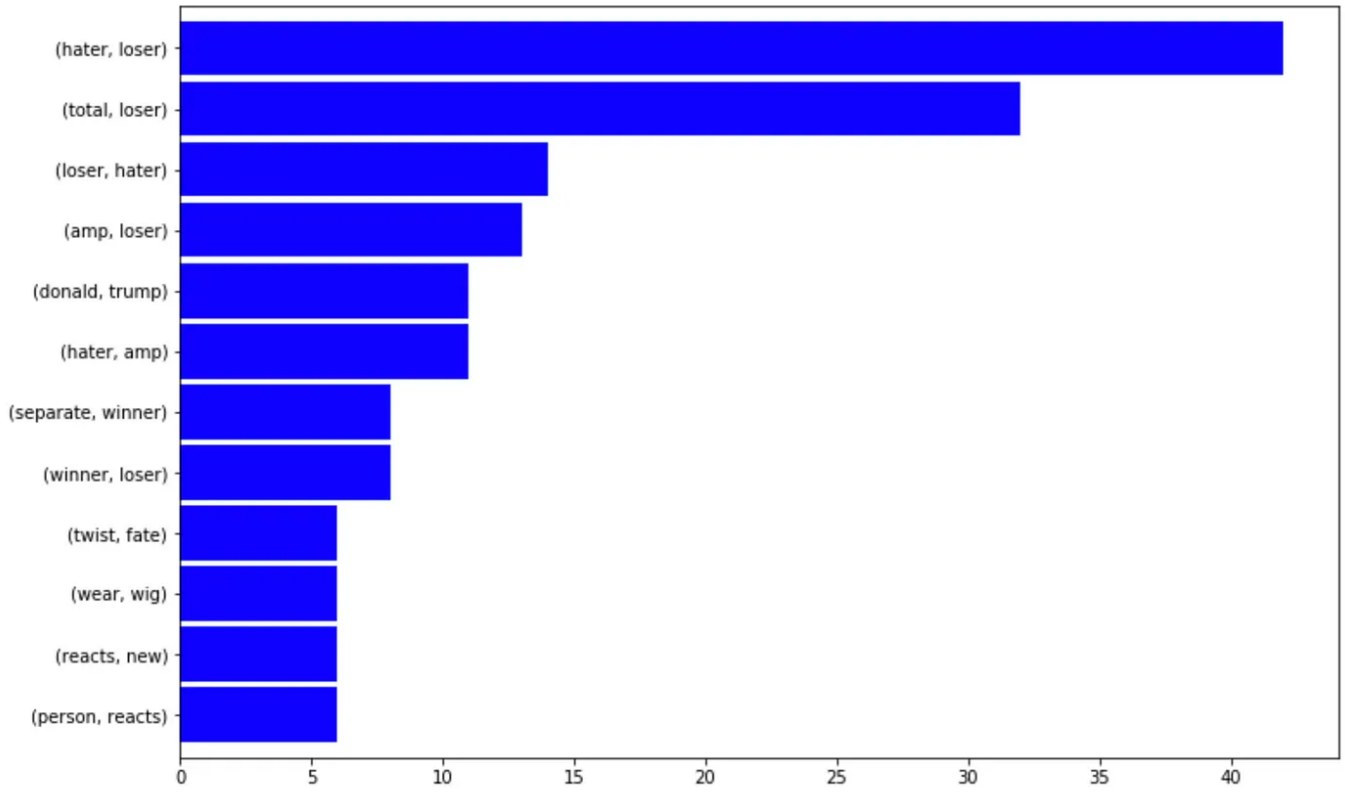
To make things a little easier for ourselves, let's assign the result of n-grams to variables with meaningful names:

```
bigrams_series = (pd.Series(nltk.ngrams(words, 2)).value_counts())[:12]
trigrams_series = (pd.Series(nltk.ngrams(words, 3)).value_counts())[:12]
```

I've replaced `[:10]` with `[:12]` because I wanted more n-grams in the results. This is an arbitrary value so you can choose whatever makes the most sense to you according to your situation.

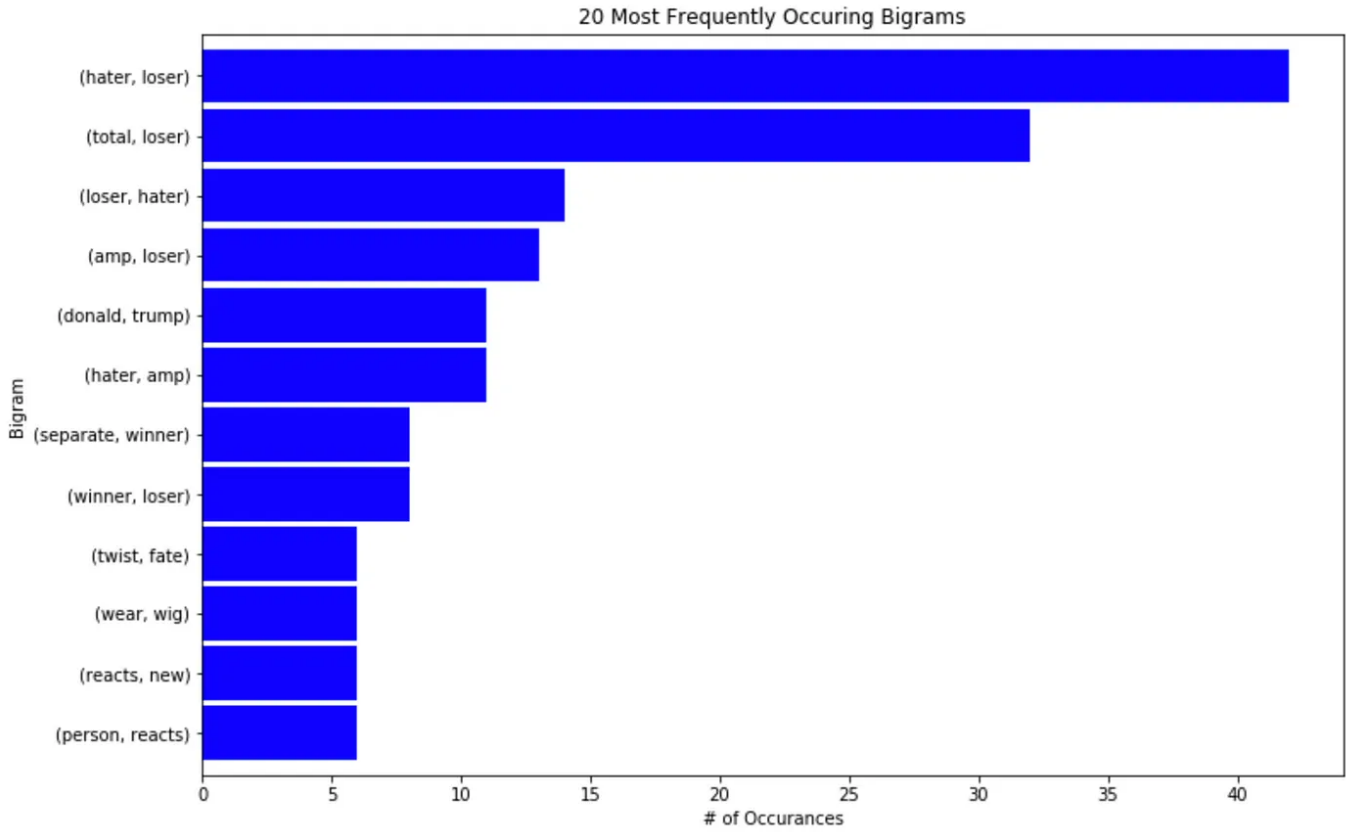
Let's create a horizontal bar graph:

```
bigrams_series.sort_values().plot.barh(color='blue', width=.9, figsize=(12, 8))
```



And let's spiffy it up a bit by adding titles and axis labels:

```
bigrams_series.sort_values().plot.barh(color='blue', width=.9, figsize=(12, 8))
plt.title('20 Most Frequently Occuring Bigrams')
plt.ylabel('Bigram')
plt.xlabel('# of Occurances')
```



And that's it! With a few simple lines of code, we quickly made a ranking of n-grams from a Pandas dataframe and even made a horizontal bar graph out of it.

I hope you enjoyed this one. Natural Language Processing is a big topic but I hope that this gentle introduction will encourage you to explore more and expand your repertoire.

In the next article, we'll visualize an n-gram ranking in Power BI with a few simple clicks of the mouse and a dash of Python!

Stay tuned!

You can reach me on Twitter or LinkedIn.

Data Science NLP Nltk Python Pandas

Follow

Written by Ednalyn C. De Dios

970 Followers · Writer for Towards Data Science

Machine Learning Engineer / Data Scientist / Python Developer / Food Lover https://medium.com/@ednalyn.dedios/membership

More from Ednalyn C. De Dios and Towards Data Science

Ednalyn C. De Dios in Data Science Nerd

Bring ChatGPT into your web app using Semantic Kernel, Python, and Flask

A hands-on introduction to Microsoft's Semantic Kernel

✨ · 6 min read · Jan 14

350 2

Sheila Teo in Towards Data Science

How I Won Singapore's GPT-4 Prompt Engineering Competition

A deep dive into the strategies I learned for harnessing the power of Large Language Models (LLMs)

· 24 min read · Dec 28, 2023

9.93K 118

Thu Vu in Towards Data Science

How to Learn AI on Your Own (a self-study guide)

If your hands touch a keyboard for work, Artificial Intelligence is going to change your job in the next few years.

· 12 min read · Jan 5

2.2K 24

Ednalyn C. De Dios in Towards Data Science

Read a Multi-Column PDF Using PyMuPDF in Python

A step-by-step introduction into the wonderful world of OCR. With pictures!

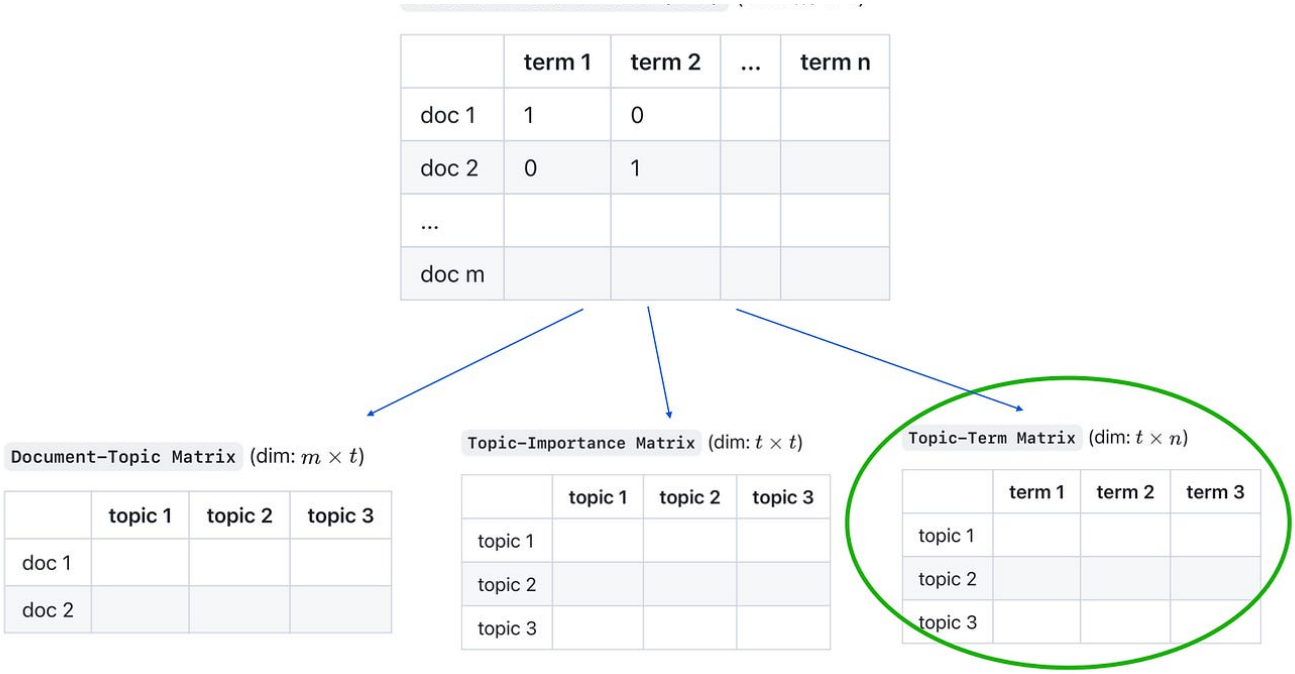
· 5 min read · Feb 22, 2022

192 1

See all from Ednalyn C. De Dios

See all from Towards Data Science

Recommended from Medium



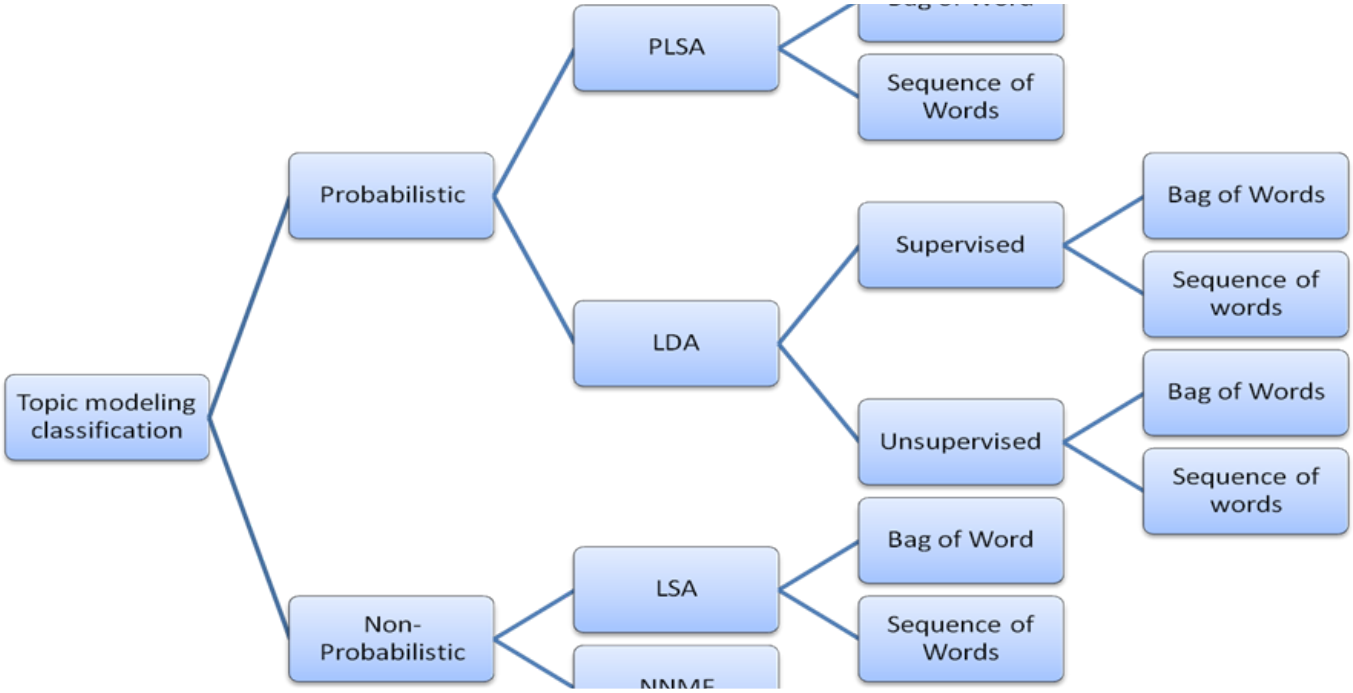
Madhurima Nath, PhD

Topic modeling algorithms

Learn about the mathematical concepts behind LDA, NMF, BERTopic models

8 min read · Aug 21, 2023

5



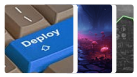
Dr. Pooja Kherwa

Topic Modeling

Topic modelling is the new revolution in text mining. It is a probabilistic statistical technique for revealing the underlying semantic…

7 min read · Jan 1

Lists

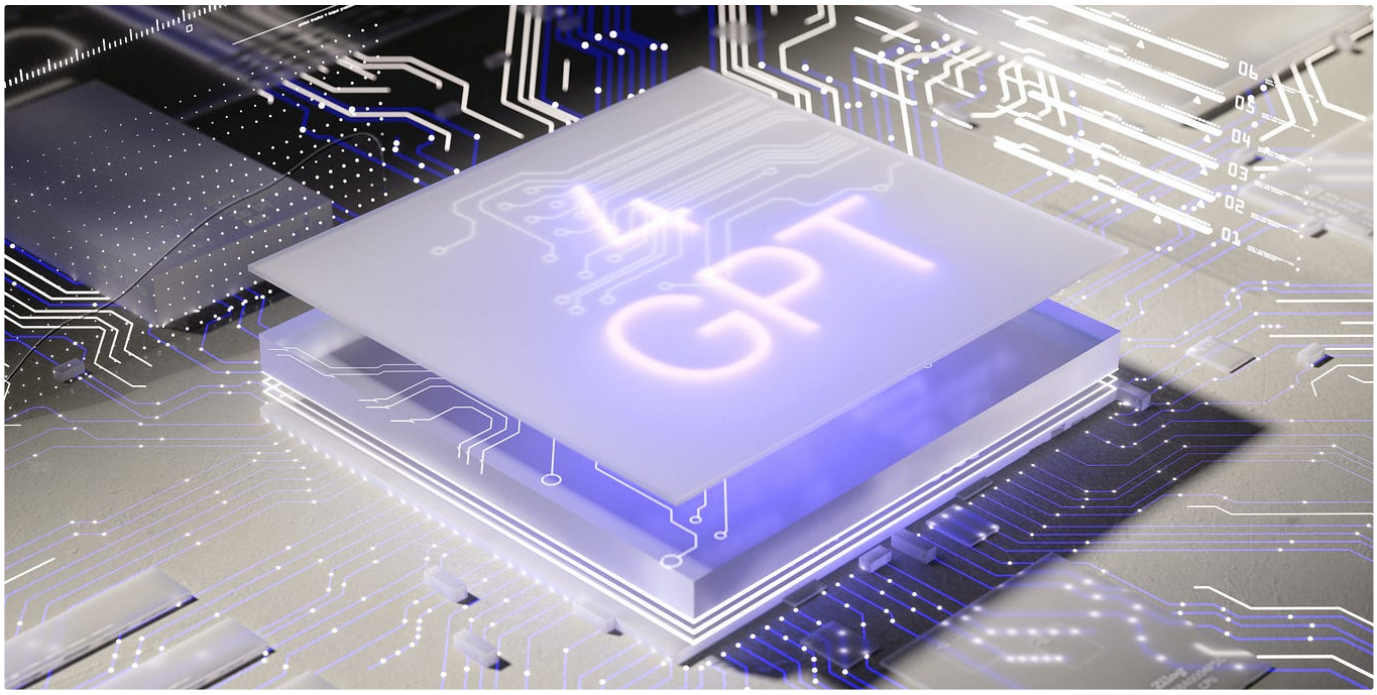


Predictive Modeling w/ Python
20 stories · 826 saves

Coding & Development
11 stories · 398 saves

Practical Guides to Machine Learning
10 stories · 953 saves

Natural Language Processing
1118 stories · 589 saves



Namita

Implementing LDA on AI generated data for topic modelling

Generative AI is out there, available for everyone but only a few know how to leverage it to the fullest.

11 min read · Sep 8, 2023

51

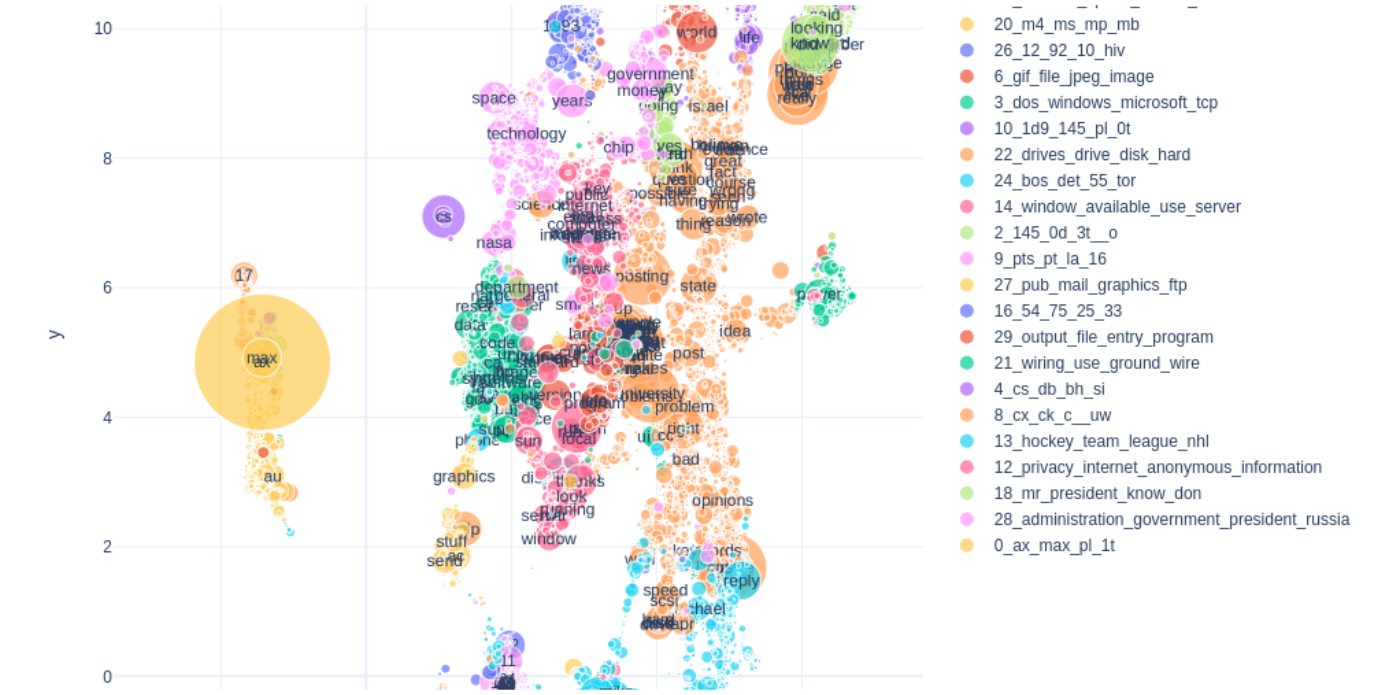
Everton Gomede, PhD

Exploring N-gram Models in Natural Language Processing

Abstract

5 min read · Aug 18, 2023

4



Márton Kardos

Unsupervised Text Classification with Topic Models and Good Old Human Reasoning

Use your brain and your data interpretation skills, and create production-ready pipelines without labeled data

9 min read · Aug 3, 2023

63 1

KH Huang

Gensim — Topic Modelling in Python

Gensim is a popular open-source library in Python for natural language processing and machine learning on textual data. One of its primary...

2 min read · Aug 18, 2023