✨ Member-only story

Segmenting Text Into Paragraphs

A statistical NLP approach based on supervised learning

Arun Jagota · Follow
Published in Towards Data Science
11 min read · Feb 25

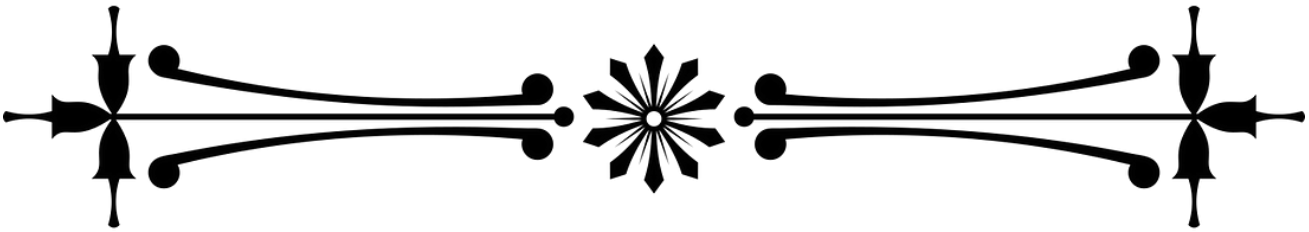


Image by Gordon Johnson from Pixabay

In a previous post on Medium, we discussed segmenting text into sentences [3]. Now we look at a related problem: segmenting text into paragraphs.

At first glance, it may seem that the two problems are essentially the same, only at different levels of chunking. The problem of segmenting text into paragraphs is in fact far more interesting.

For one thing, sentence boundaries have explicit signals such as periods, question marks, or exclamation points. Generally, the issue is this. Which of these occurrences are actual boundaries versus which are embedded within a sentence? That is the issue of false positives.

Segmenting the text into paragraphs is more nuanced. Think this way. Say you have a long sequence of sentences with no paragraph breaks. Where should the paragraph boundaries be? Not an easy problem to solve. Nor does it necessarily have unique solutions. Meaning that there may be more than one good split of a sequence of sentences into paragraphs.

Splitting text into paragraphs may be viewed as a particular case of text segmentation [1]. A text segment is a contiguous segment that preserves some coherency such as being on the same topic. According to this measure of coherency, a segment would transition to another one on a change in topic.

The broader text segmentation problem is more difficult to solve. For several reasons. Including the fact that it is hard to get labeled data. For paragraph segmentation, plenty of labeled data is available. In the form of web pages and Wikipedia articles with paragraph breaks in them. For the broader text segmentation problem this is not so.

In this post, we take the view that an algorithm that can suggest reasonable split boundaries can be helpful to someone writing text. In the same way that Grammarly is helpful. In other words, neither the precision nor the recall needs to be particularly high. The precision needs to be reasonable; the recall could be even less.

The rest of this post should be read keeping this view in mind. We will be satisfied with a solution that has a reasonable precision, possibly even around 50%, and even low recall, possibly around 10%. The point is that even this is useful in a Grammarly-like setting.

Even if paragraph break suggestions are made rarely, so long as they have reasonable precision, they add to the value of a product such as Grammarly.

It goes without saying that if we could get better precision or recall with minimal

effort we would take it.

A Probabilistic Model That Predicts Paragraph Breaks

We'll start with a formal description, explaining its various components in plain English.

Let X1 and X2 denote two adjacent sentences in a document in the training corpus.

We will associate a binary label Y with (X1, X2). Y will be 1 if there is a paragraph break between X1 and X2 and 0 if not.

We will track a third predictor *i*. X1 will be the *i*th sentence in the current paragraph. The predictor "*i*" will imbue our model with the ability to pay attention to paragraph lengths.

Our training set will comprise instances (X1, X2, *i*, Y).

From the training set, we aim to learn a model $P(Y \mid X1, X2, i)$.

$P(Y=1 \mid X1, X2, i)$ will denote the probability that there is a paragraph break between X1 and X2 when X1 is the *i*th sentence in the current paragraph.

$P(Y=0 \mid X1, X2, i)$ will denote the probability that X2 should extend the current paragraph given X1 as the *i*th sentence in the current paragraph.

The model $P(Y \mid X1, X2, i)$ is very complex. This is because X1 and X2 are sentences and can be arbitrarily rare or long. Meaning that there won't be sufficient data to estimate this model even if our training set comprises a few billion labeled instances.

We will need to make certain assumptions.

First off, let's apply the Bayes rule

$P(Y \mid X1, X2, i) = N(X1, X2, i, Y)/Z$

where $N(X1, X2, i, Y)$ equals $P(X1, X2, i \mid Y)\ P(Y)$.

Z is simply $N(X1, X2, i, 0) + N(X1, X2, i, 1)$

Next, we will factor N(X1, X2, *i*, Y) as below.

N(X1, X2, *i*, Y) = *P*(X1 | Y)***P*(X2 | Y)***P*(*i* | Y)***P*(Y)

P(X1 | Y=1) is the distribution of the last sentences in a paragraph. *P*(X1 | Y = 0) is the distribution over the non-last sentences in a paragraph.

P(X2 | Y = 1) is the distribution over the first sentences in a paragraph. *P*(X2 | Y = 0) is the distribution over the non-first sentences in a paragraph.

Now consider *P*(*i* | Y).

Let's remind ourselves that X1 is the *i*th sentence in the current paragraph. So *P*(*i* | Y=1) is effectively the distribution of the length of a paragraph as the paragraph must end right after X1, the ith sentence in the current paragraph.

P(*i* | Y = 1) will tend to be biased towards small *i*. This is because most paragraphs are short.

P(*i* | Y = 0) will tend to be biased towards being even smaller. This is because Y = 0 means that the *i*th sentence X1 in the current paragraph does not end it.

The probability models *P*(X1 | Y) and *P*(X2|Y) are each still too complex. This is because the universe of sentences is unbounded. That is, sentences can be arbitrarily long. And arbitrarily rare.

Can we make further simplifying assumptions? Specifically use the likelihoods not necessarily of the entire sentences but of the first few words in them.

Let's start by looking at real examples.

First, let's see examples of sentences that continue short paragraphs.

Say the next sentence starts with "For example," "Examples of", "More precisely, ", etc. If the current paragraph is sufficiently short, e.g. one or two sentences long, these prefixes in the next sentence predict Y = 0, i.e. extending the paragraph.

To support this hypothesis, we invite the reader to read these pairs of adjacent

sentences from https://en.wikipedia.org/wiki/Deep_learning

> *Deep learning algorithms can be applied to unsupervised learning tasks. This is an important benefit because unlabeled data are more abundant than the labeled data.* ***Examples of*** *deep structures that can be trained in an unsupervised manner are deep belief networks.*

...

> *Deep learning is a class of machine learning algorithms that[8]: 199–200 uses multiple layers to progressively extract higher-level features from the raw input.* ***For example****, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.*

...

> *The word "deep" in "deep learning" refers to the number of layers through which the data is transformed.* ***More precisely****, deep learning systems have a substantial credit assignment path (CAP) depth.*

Would you agree that the bolded word sequences in each predict continuing the paragraph?

These examples suggest that it makes sense to consider simplifying $P(\text{X1} \mid \text{Y})$ to $P(\text{the first few words of X1} \mid \text{Y})$.

This begs the question what is the value of "few" above? We'll tackle this later.

Next, let's see examples of one-sentence paragraphs.

For this, I asked ChatGPT to give me examples of one-sentence paragraphs. Seems like it took this literally. So I rephrased the question to

> *Give me examples of sentences that* ***only*** *form one-sentence paragraphs.*

Now I got good examples.

> *Silence.*
>
> *Stop.*

> *Never again.*
>
> *Why?*
>
> *Yes!*
>
> *I'm sorry.*
>
> *Enough.*
>
> *Remember.*
>
> *Help!*
>
> *Goodbye.*

We would expect each of these sentences to have a high likelihood $P(X1 \mid Y = 1)$. That said, for some or all of them $P(X1 \mid Y = 0)$ may also be somewhat high. This would mean the paragraph does not end right after them.

Nonetheless, we show these examples here, as they do suggest that $P(X1 \mid Y = 1)$ for these sentences is worth modeling.

Next, let's see examples of sentences that begin new paragraphs. We picked up some paragraphs from https://en.wikipedia.org/wiki/Deep_learning and are showing the first few words in their first sentence.

> *Deep learning is part of a broader family …*
>
> *Deep-learning architectures such as …*
>
> *Artificial neural networks (ANNs) were …*
>
> *In deep learning, each level learns to …*
>
> *An ANN is based on a collection of …*
>
> *DNNs can model complex non-linear …*

These examples suggest that $P(X2 \mid Y = 1)$ could be simplified to

P(first few words of X2 | Y = 1)

for a suitable choice of "few".

Let's formalize what we have learned from these examples.

We can simplify P(X1 | Y) and P(X2 | Y) to

P(X1 begins with $w(1)$, $w(2)$, ..., $w(k)$ | Y)

and

P(X2 begins with $w'(1)$, $w'(2)$, ..., $w'(k')$ |Y)

respectively.

Here $w(1)$, $w(2)$, ..., w(k) and $w'(1)$, $w'(2)$, ..., $w'(k')$ are sequences of k and k' words respectively.

The obvious question is what should k and k' be?

One way to address this question is to not fix k and k' in advance, but rather delay the decision until inference time.

Here is how.

First some terminology. We will call a sequence of words that begins a sentence as the sentence's prefix.

Now consider P(X1 | Y=y). We will approximate this as follows.

We will first find the largest prefix of X, call it P, which has sufficient support. We will use P(X1 starts with P | Y) as a proxy for P(X1 | Y).

The support of a prefix P of X1 for the estimation of P(X1 | Y) is defined as the number of instances in the training set in which P is a prefix of X1.

The idea behind this approximation procedure is that we should use the longest prefix of X1 that we can, provided it is seen enough times in the training set (as a prefix of X1).

Similarly, we should estimate P(X2 | Y) as P(Q | Y) where Q is the largest prefix of X2 whose support is sufficiently large.

What does the form our inference has taken mean for a model? We will need to track the probabilities P(X1 starts with P | Y) for all prefixes P of X1. Similarly for P(X2 | Y).

Internally, for modeling P(X1 |Y) and P(X2 | Y), there are lots of word sequences we need to track.

Fortunately, these word sequences can be collected into so-called Trie data structures. These are optimized for compactly representing a huge set of word sequences.

These Tries are built during the training process as follows.

We will use four Tries T10, T11, T20, and T21 respectively. Tiy, i = 1 or 2, will store the prefix sequences and their counts for Y=y for Xi.

Each node in the Trie will store a count.

We will initialize all the tries to start with a single node, the root node, whose count is set to zero.

Now consider an instance (X1, X2, y) in the training set. We have left out i as it will not affect the tries.

Interpreting X1 as a sequence of words, we will look up X1 in T1y, extending the trie with a path comprised of new nodes as needed. Any time a new node is created its count will be initialized to 0.

Now, in the Trie T1y, we will increment the counts of all nodes on the path that represents X1. By one each.

To process X2 we will repeat the same procedure on the Trie T2y.

Numeric Example

Let's now illustrate this process.

The four tries will be initialized to T10, T11, T20, and T21 each being {[]:0}.

Now imagine that we present the first training instance as ([a,b],[A,B,C],1)

> *T11's new state will be {[]:1,[a]:1,[a,b]:1}.*
>
> *T21's new state will be {[]:1,[A]:1, [A,B]:1, [A,B,C]:1}*

Now imagine that we present this training instance: ([a,d],[A,B,E],1)

> *T11's new state will be {[]:2,[a]:2,[a,b]:1,[a,d]:1}*
>
> *T21's new state will be {[]:2,[A]:2, [A,B]:2, [A,B,C]:1,[A,B,E]:1}*

In the above illustration, for visual convenience, we have represented each Trie as a hashmap, i.e. a Dict in Python.

In reality, we can represent the Trie more compactly as a tree by leveraging the structure of the (repeated) prefixes.

The representation of the Trie as a tree is also much more efficient for looking up the counts associated with all the prefixes of a given sequence X in the trie. We simply go down the unique path that the Trie contains for the longest prefix of X. We say "longest prefix" because X may not be fully in the Trie if X was never encountered in the training set in the context in which it would be placed in this Trie. However, there is always a path in the Trie for at least one prefix of X, even if only the empty one.

Inference Using The Tries

Say we are done with training. Now for a given (X1, X2, *i*) we want to compute $P(Y=y|X1,X2,i)$.

The components of this calculation that involve the tries are $P(X1|Y=y)$ and $P(X2|Y=y)$ respectively.

Let's illustrate how to compute one of these, as the other will be similar.

Let's pick $P(X1|Y=y)$.

We run down the tries T10 and T11 to find the longest prefix of X1 that has sufficiently high support. We need to use both tries since the support of a prefix P of X1 is the sum of the counts on the nodes in T10 and T11 at which P ends.

Let's denote the longest prefix of X1 with sufficient support as P(X1).

$P(\text{P(X1)} \mid \text{Y}=y)$ is simply the count at the node where P ends in T1y divided by the count on the root node of the trie T1y. This is simply the number of instances in the training set whose label is y and whose X1 starts with P(X1) divided by the number of instances in the training set whose label is y.

Summary

In this post, we covered the NLP problem of segmenting a text into its paragraphs. We noted that this problem is more challenging than the problem of segmenting text into sentences but less challenging than the problem of segmenting text into coherent units such as by topic.

We framed this problem as one of supervised learning. There is a lot of labeled data readily available. The input is a pair of adjacent sentences. The outcome is whether or not there is a paragraph break between the two. As such this is a supervised learning problem in which the input is a pair of sequences and the outcome is binary.

Next, we applied the Bayes rule under the naive Bayes assumption, one of conditional independence of the predictors given the outcome. We then worked out the likelihood and the prior terms in the resulting formula.

From here we noted that even under the naive assumption the resulting model is too complex. We discussed how to cope with this complexity by modeling each of the two sentences in the input as a collection of prefixes going from the null prefix to the entire sequence. At inference time, we described how to use the "right" prefix for the prediction of the outcome.

We examined several real examples of adjacent sentences in real text to support our case for working off prefixes instead of full sentences.

Finally, we noted that working with all prefixes of sentences rather than the

sentences themselves potentially blows up the model size. For this, we came up with a scheme using Tries. Sequences in the same context are compactly represented in appropriate tries. We discussed in detail how the Tries would be learned during training, and how the Tries would be used during inference.

References

1. Ne[illegible]or[illegible]

Trie Naive Bayes Binary Classification

2. Grammarly
3. https://towardsdatascience.com/segmenting-text-into-sentences-using-nlp-35d8ef55c0fd
4. https://en.wikipedia.org/wiki/Trie

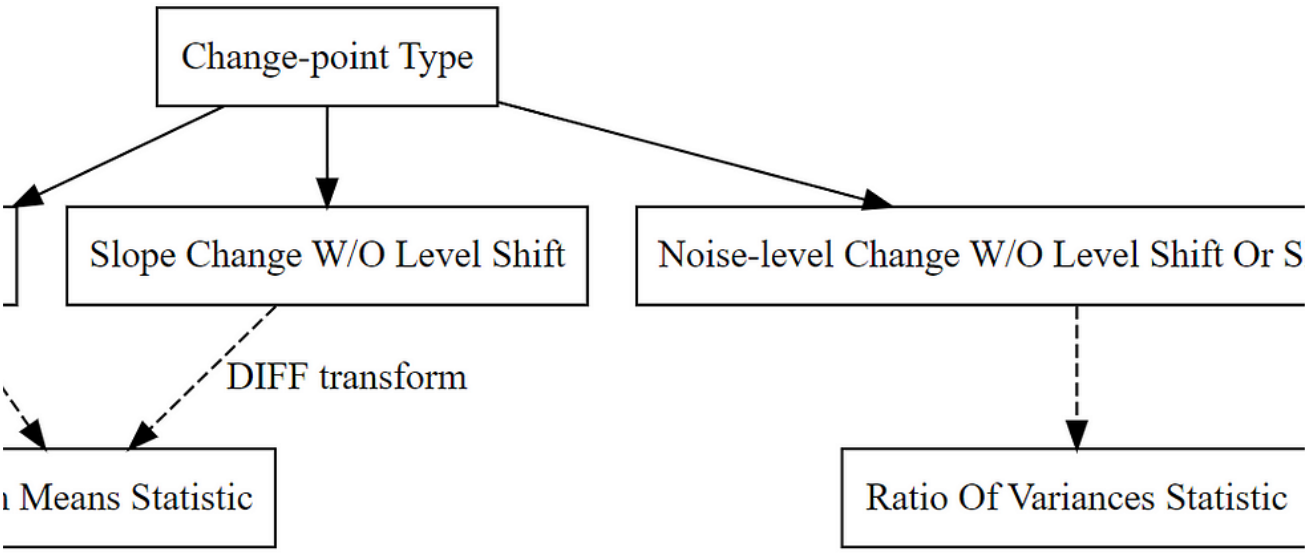
Follow

Written by Arun Jagota

805 Followers · Writer for Towards Data Science

PhD, Computer Science, neural nets. 14+ years in industry: data science algos developer. 24+ patents issued. 50 academic pubs. Blogs on ML/data science topics.

More from Arun Jagota and Towards Data Science



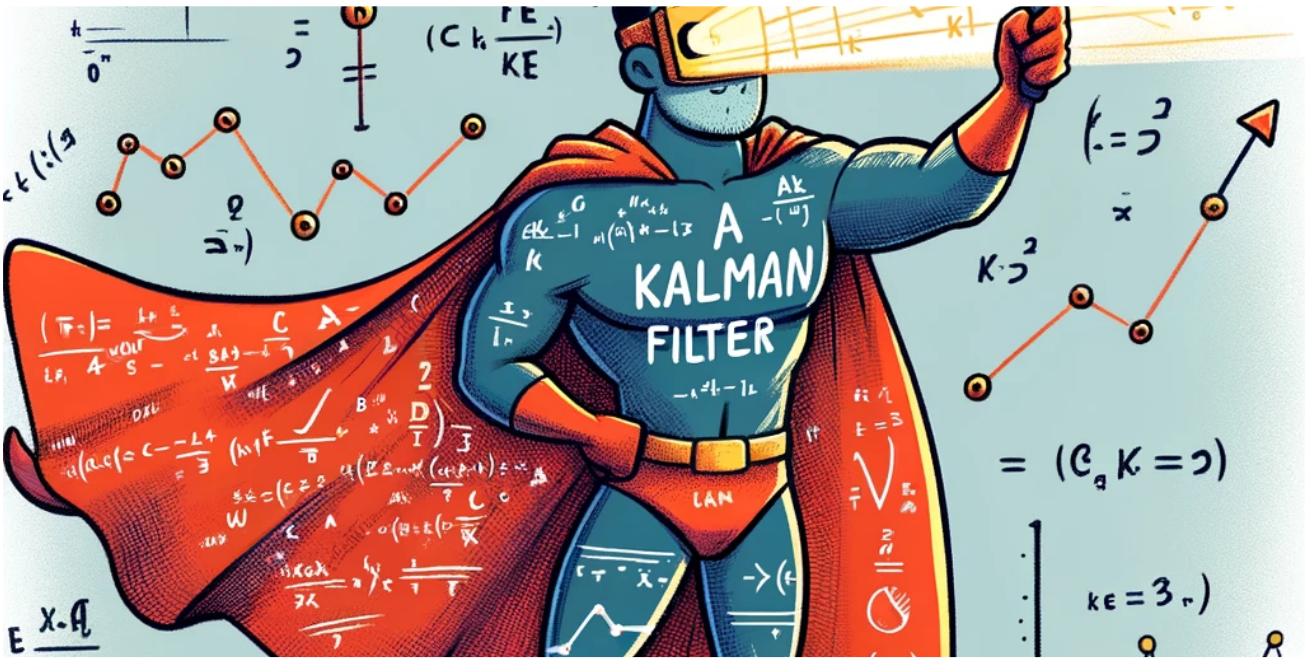
Arun Jagota

Change-point Detection In Time Series

Basic scenarios and methods

· 7 min read · Oct 13

2



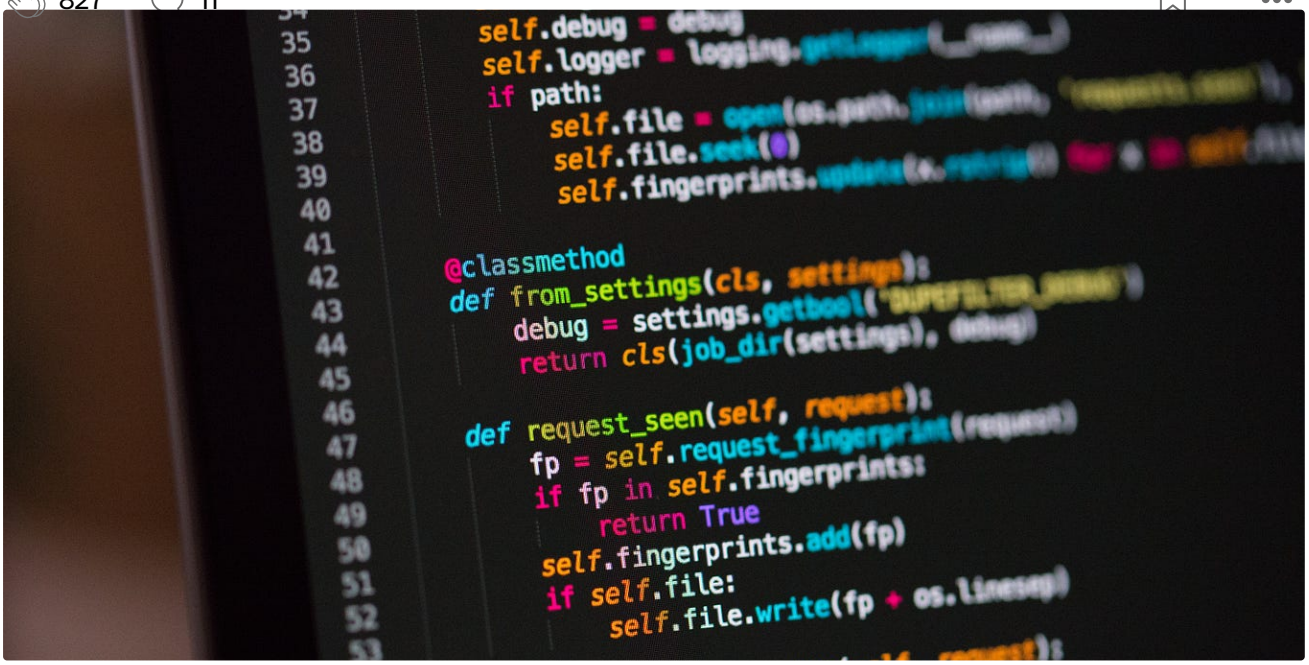
Jimmy Weaver in Towards Data Science

Exposing the Power of the Kalman Filter

As a data scientist we are occasionally faced with situations where we need to model a trend to predict future values. Whilst there is a...

17 min read · Nov 7

827 11



Benjamin Lee in Towards Data Science

The New Best Python Package for Visualising Network Graphs

A guide on who should use it, when to use it, how to use it, and why I was wrong before...

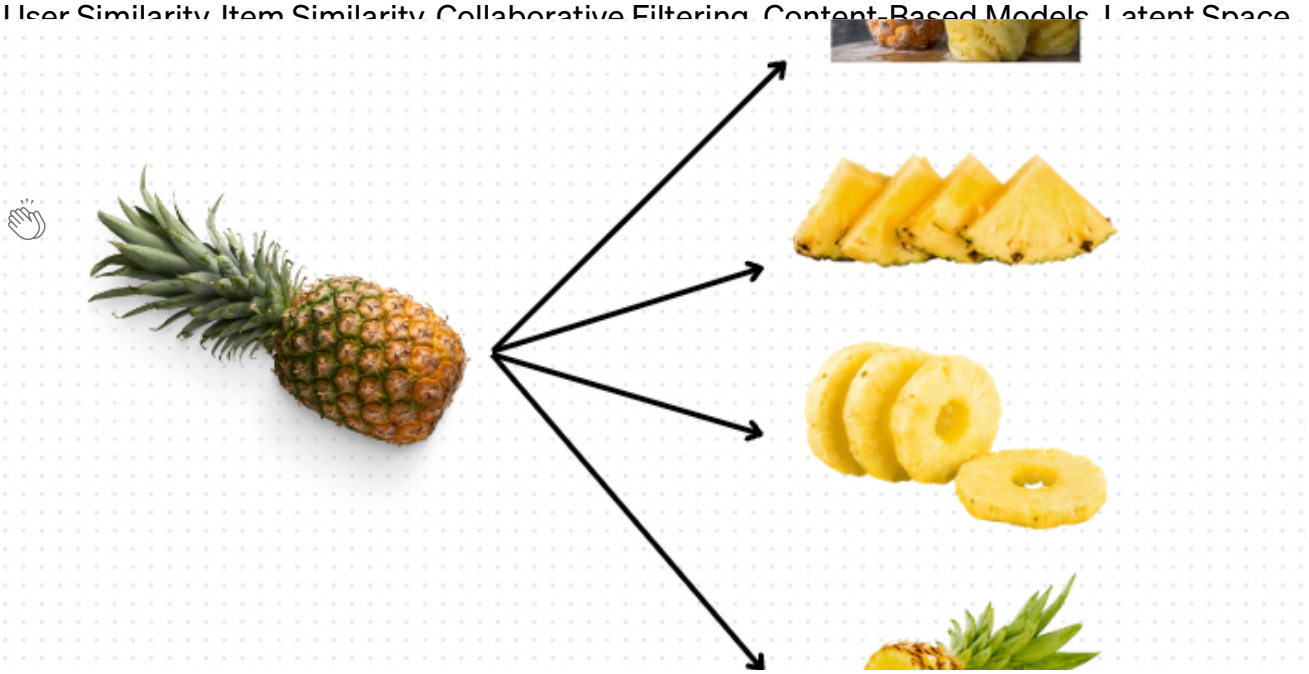
· 10 min read · Nov 23

736 10

Arun Jagota in Towards Data Science

Recommended from Medium
Basics of Recommender Systems

User Similarity, Item Similarity, Collaborative Filtering, Content-Based Models, Latent Space



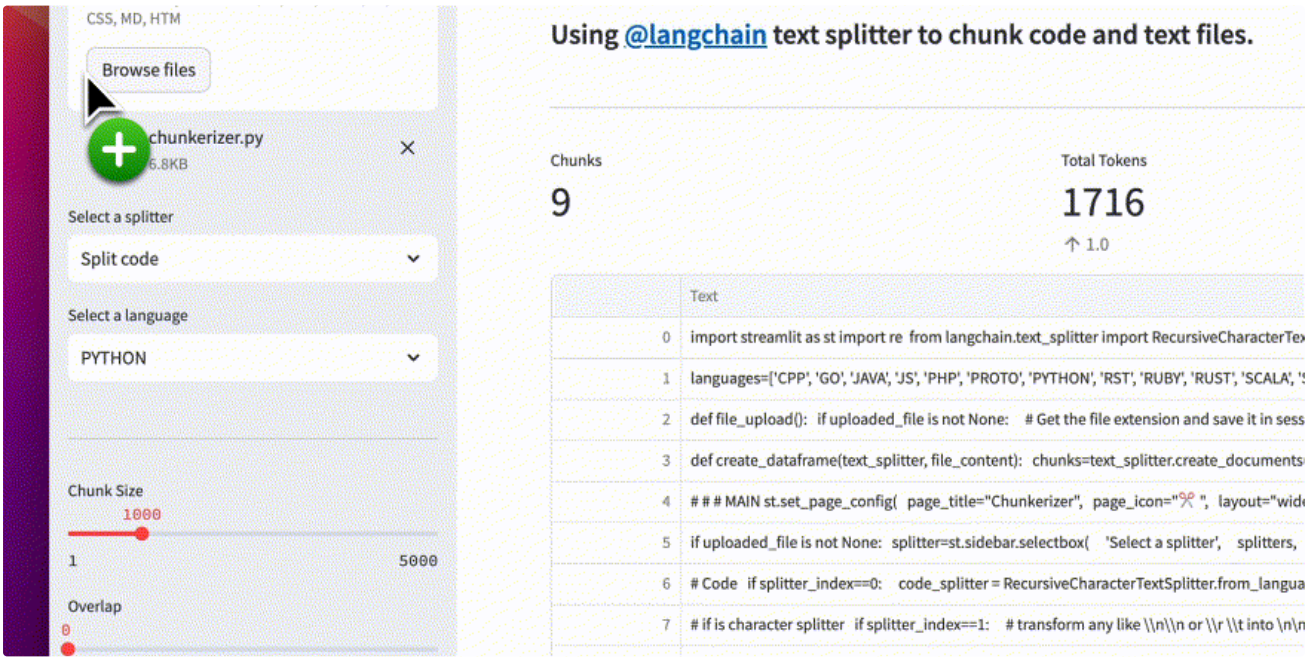
Solano Todeschini in Towards Data Science

How to Chunk Text Data — A Comparative Analysis

Exploring and comparing distinct approaches to text chunking.

17 min read · Jul 20

427 6



Gustavo Espíndola

🦜✂️ Text Splitters: Smart Text Division with Langchain

In the fascinating world of natural language processing, tools for transforming documents and splitting texts have become essential, thanks...

2 min read · Sep 5

6

Lists

Natural Language Processing

917 stories · 432 saves

Gursev Pirge in John Snow Labs

Text Preprocessing: Splitting texts into sentences with Spark NLP

Using Sentence Detection in Spark NLP for Text Preprocessing

8 min read · Jul 21

2

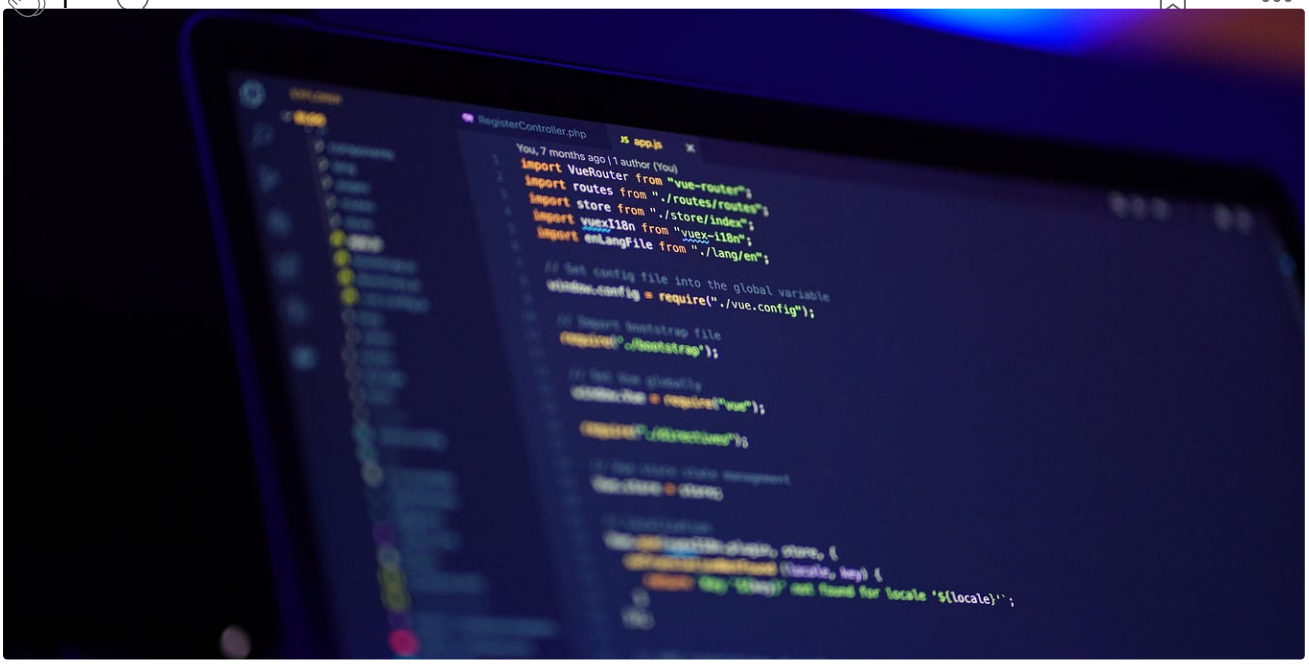
Yuan An, PhD

Named Entity Recognition (NER) Using the Pre-Trained bert-base-NER Model in Hugging Face

This is a series of short tutorials about using Hugging Face. The table of contents is here.

6 min read · Oct 5

1



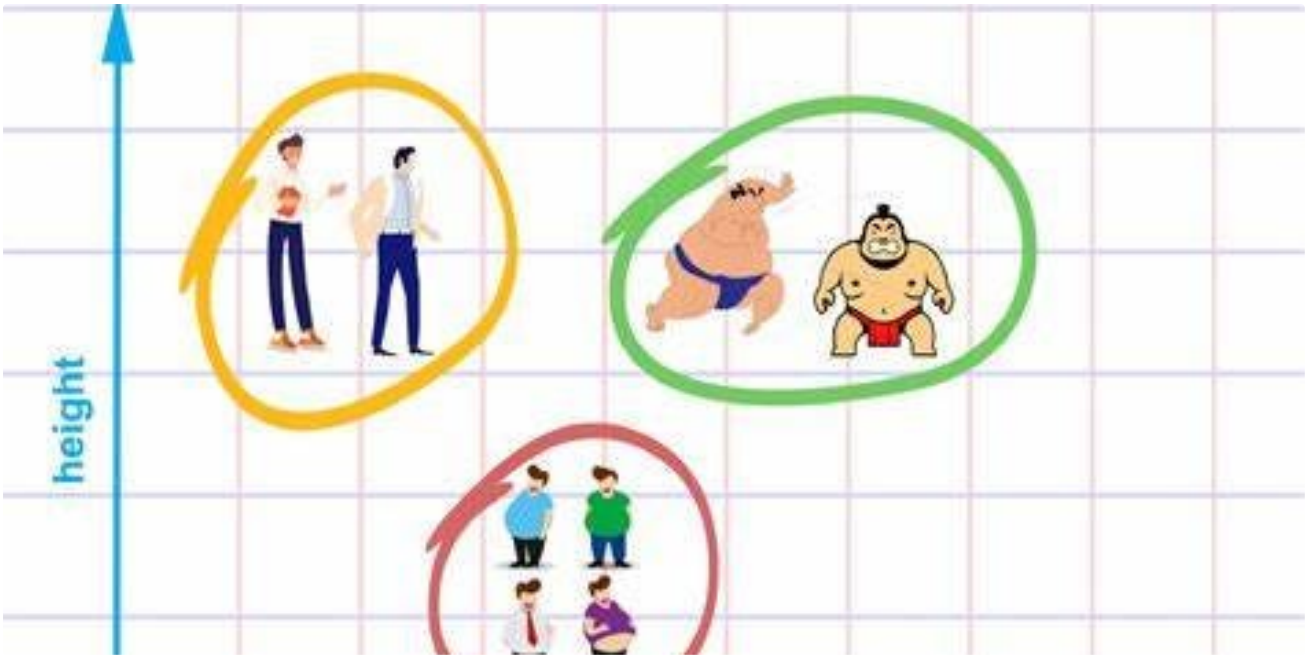
Murage Charles

Named entity recognition (NER) in natural language processing

When it comes to unraveling the intricate details hidden within text, Named Entity Recognition (NER) stands tall as a vital task in the...

10 min read · Jul 6

25



TANIMU ABDULLAHI

Developing a Clustering Model: Utilizing the K-means Algorithm

Introduction

13 min read · Jun 19