Agenda: Text → Numerical vectors

- BoW → v. high dim data
- ✓ TF-IDF

Prev: terms, pre-processing

- case: finding similar blogs
- PCA for data-insights

Finding similar Medium articles

`You are working as a Data Scientist at Medium`

- Medium is an online publishing platform which hosts a hybrid collection of blog posts from both amateur and professional people and publications.
- In 2020, about 47,000 articles were published daily on the plaform and it had about 200M visitors every month.

Problem Statement:

- You want to give readers a better reading experience at Medium. To do that, you want to recommend articles to the user on the basis of current article that the user is reading.
- **More concretely, given a Medium article find a set of similar articles.**

How would a human find similar articles in a corpus ?

1. Look at the title - find similar titles.
2. Find articles by the same author.
3. Go through the text, understand it and group the aricles within broader topics.

Solving common preprocessing Problems in NLP.

Expanding Word Contractions

- Contractions are the shortened word of english words or phrases.

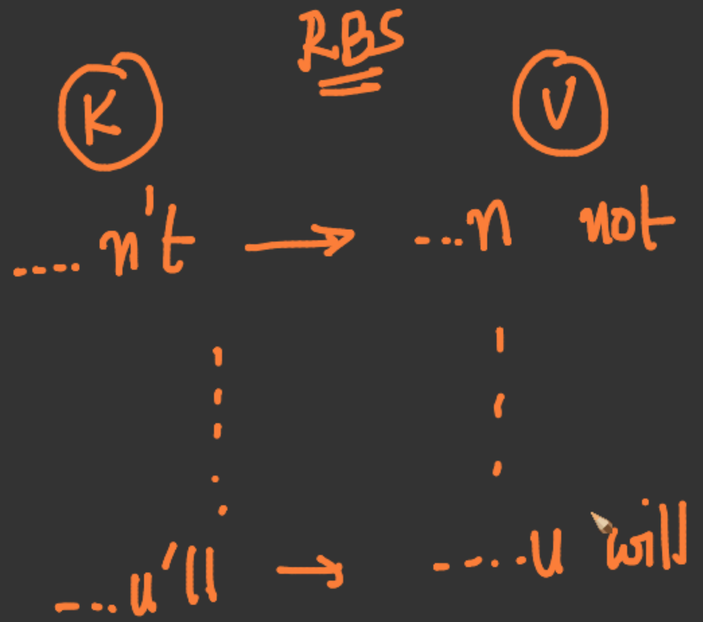
  Example -

  i/p o/p

  $can't \rightarrow can\ not$

  $you're \rightarrow you\ are$

  $you'll've \rightarrow you\ will\ have$



- Why is this a problem?

  Because of the following reasons -

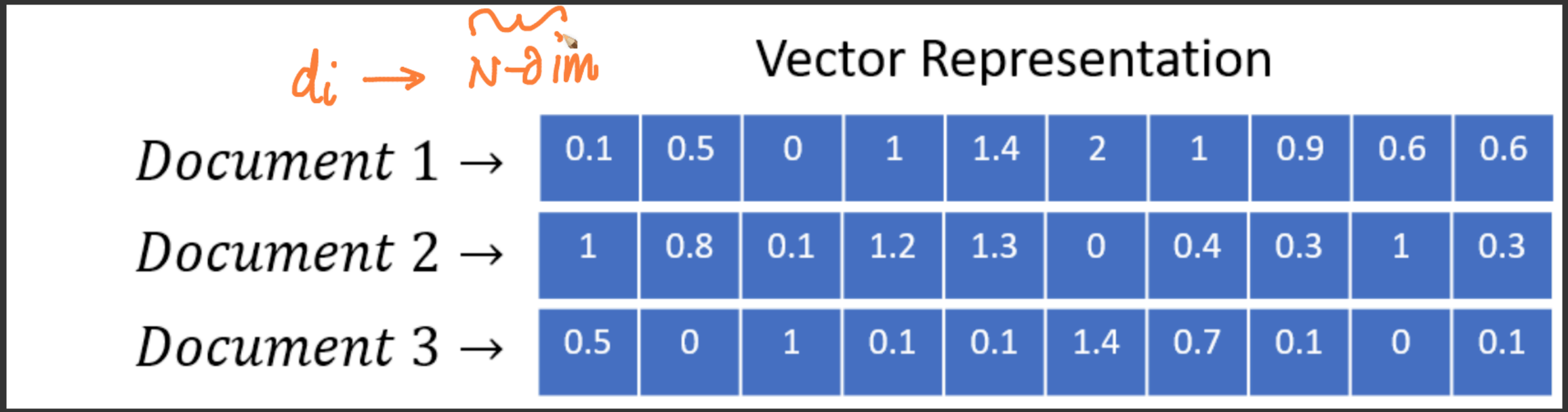
  1. They add in a special apostrophe character (computers cannot understand its meaning) to the text.
  2. They are a combination of two words, hence can cause problems in tokenization.

- How to deal with it ?

  1. Create a custom mapping of expansions.

```
[ ] From: https://drive.google.com/uc?id=1FeEMVCRetrOgwnyIupR8l_MELoOt-U3w
    To: /content/word_emb_ex.png
    100% 7.38k/7.38k [00:00<00:00, 12.9MB/s]
```

```
[ ] Image.open('/content/feature_extraction_dummy.png')
```



- Now once we have vectorized, each document we can use them as features for any machine learning model of our choice.
- There are multiple ways using which we can get a vector representation of text. Let's have a look at a couple of them.

Step 1 - Create a vocabulary from the above corpus

> Vocabulary is simply a set of all unique words in the documents. For the above example, the vocabulary would be
>
> - Lecture
> - on
> - text
> - representation

Step 2 - Each word in the sentence would be represented as below:

OHE

Word	Lecture	on	text	representation
Lecture	1	0	0	0
on	0	1	0	0
text	0	0	1	0
representation	0	0	0	1

```
Lecture → [1 0 0 0], on → [0 1 0 0], text → [0 0 1 0], representation → [0 0 0 1]
```

Step 3 - The entire sentence is then represented as:

Step 2 - Each word in the sentence would be represented as below:

Word	Lecture	on	text	representation
Lecture	1	0	0	0
on	0	1	0	0
text	0	0	1	0
representation	0	0	0	1

OHE

200 docs → 1000 unique words

1000 dim

d_1: 10 words → 10 X 1000 dim

d_2: 50 words → 50 X 1000 dim

d_{120}: 1000 wds

```
Lecture → [1 0 0 0], on → [0 1 0 0], text → [0 0 1 0], representation → [0 0 0 1]
```

Step 3 - The entire sentence is then represented as:

```
sentence = [ [1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1] ]
```

- Intuition behind it is that each bit represents a possible category.
- And if a particular variable cannot fall into multiple categories, then a single bit is enough to represent it.

```
[ ] from sklearn.preprocessing import OneHotEncoder
    import itertools
    # two example documents
```

```
# convert list of token-id lists to one-hot representation
vec = OneHotEncoder(categories="auto")
X = vec.fit_transform(token_ids)
print(X.toarray())
```

```
[[0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1.]]
```

of ensemble

Drawbacks:

1. Cannot measure the importance of a word in a sentence but understand the mere presence/absence of a word in a sentence.
2. High dimensional sparse matrix representation can be memory & computationally expensive.
3. Explosion in feature space if the number of categories is very high.
4. The vector representation of words is orthogonal and cannot determine or measure the relationship between different words.

How can we optimize on One Hot Encoding ?

Can we use the frequency of each word in the document ?

```
# convert list of token-id lists to one-hot representation
vec = OneHotEncoder(categories="auto")
X = vec.fit_transform(token_ids)
print(X.toarray())
```

```
[[0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1.]]
```

Drawbacks:

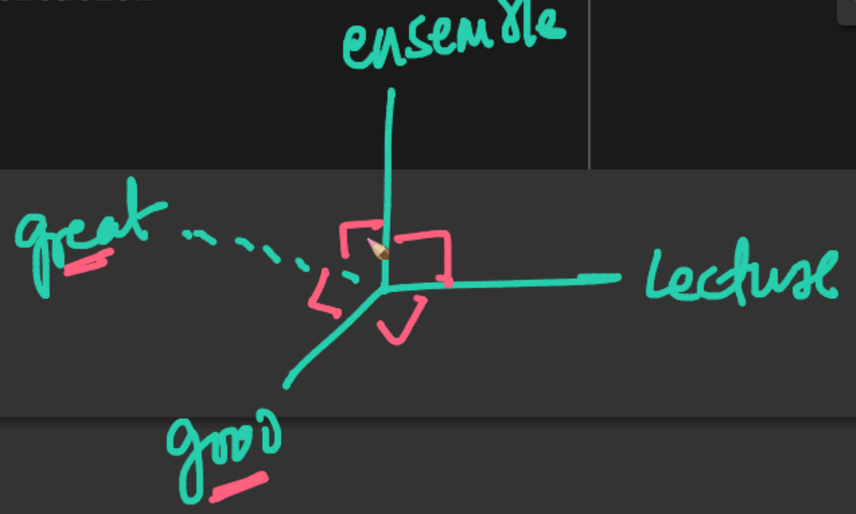
1. Cannot measure the importance of a word in a sentence but understand the mere presence/absence of a word in a sentence.
2. High dimensional sparse matrix representation can be memory & computationally expensive.
3. Explosion in feature space if the number of categories is very high.
4. The vector representation of words is orthogonal and cannot determine or measure the relationship between different words.

How can we optimize on One Hot Encoding ?

Can we use the frequency of each word in the document ?

```
# convert list of token-id lists to one-hot representation
vec = OneHotEncoder(categories="auto")
X = vec.fit_transform(token_ids)
print(X.toarray())
```

```
[[0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1.]]
```



Drawbacks:

1. Cannot measure the importance of a word in a sentence but understand the mere presence/absence of a word in a sentence.
2. High dimensional sparse matrix representation can be memory & computationally expensive.
3. Explosion in feature space if the number of categories is very high.
4. The vector representation of words is orthogonal and cannot determine or measure the relationship between different words.

How can we optimize on One Hot Encoding ?

Can we use the frequency of each word in the document ?

Can we use the frequency of each word in the document ?

- Puts words in a “bag” & computes the frequency of occurrence of each word.

```
Let's start with an example. Let us assume that our corpus contains following 3 texts -
1. It was the best of times
2. It was the worst of times
3. It was the age of wisdom and the age of foolishness
```

Step 1 - Create a vocabulary from the above corpus

Vocabulary is simply a set of all unique words in the documents. For the above example, the vocabulary would be -

- it
- was
- the
- best
- of
- times
- worst
- age

1. Using a binary OR operator between the OHE vectors. The final document vector that we get in this case simply tells the absence or presence of certain words in the document.
2. Using a vector sum opeartor. The final document vector that we get in this case tells the frequency of each word in the document.

Lets apply both the combination techniques in the 3rd sentence -

Word	it	was	the	age	of	wisdom	and	the	age	of	foolishness	Combining using OR	Combining using SUM
it	1	0	0	0	0	0	0	0	0	0	0	1	1
was	0	1	0	0	0	0	0	0	0	0	0	1	1
the	0	0	1	0	0	0	0	1	0	0	0	1	2
best	0	0	0	0	0	0	0	0	0	0	0	0	0
of	0	0	0	0	1	0	0	0	0	1	0	1	2
times	0	0	0	0	0	0	0	0	0	0	0	0	0
worst	0	0	0	0	0	0	0	0	0	0	0	0	0
age	0	0	0	1	0	0	0	0	1	0	0	1	2
wisdom	0	0	0	0	0	1	0	0	0	0	0	1	1
and	0	0	0	0	0	0	1	0	0	0	0	1	1
foolishness	0	0	0	0	0	0	0	0	0	0	1	1	1

Note: The combination using the sum operator is more commonly used, as it conveys more information about the text - like which words are used more frequently (this is obviously done after removing stopwords)

2. Using a vector sum opeartor. The final document vector that we get in this case tells the frequency of each word in the document.

Lets apply both the combination techniques in the 3rd sentence -

OHE Binary BoW Count BoW N

Word	it	was	the	age	of	wisdom	and	the	age	of	foolishness	Combining using OR	Combining using SUM
it	1	0	0	0	0	0	0	0	0	0	0	1	1
was	0	1	0	0	0	0	0	0	0	0	0	1	1
the	0	0	1	0	0	0	0	1	0	0	0	1	2
best	0	0	0	0	0	0	0	0	0	0	0	0	0
of	0	0	0	0	1	0	0	0	0	1	0	1	2
times	0	0	0	0	0	0	0	0	0	0	0	0	0
worst	0	0	0	0	0	0	0	0	0	0	0	0	0
age	0	0	0	1	0	0	0	0	1	0	0	1	2
wisdom	0	0	0	0	0	1	0	0	0	0	0	1	1
and	0	0	0	0	0	0	1	0	0	0	0	1	1
foolishness	0	0	0	0	0	0	0	0	0	0	1	1	1

Note: The combination using the sum operator is more commonly used, as it conveys more information about the text - like which words are used more frequently (this is obviously done after removing stopwords).

The representation of all the 3 texts above are

worst	0	0	0	0	0	0	0	0	0	0	0	**0**	**0**
age	0	0	0	1	0	0	0	0	1	0	0	**1**	**2**
wisdom	0	0	0	0	0	1	0	0	0	0	0	**1**	**1**
and	0	0	0	0	0	0	1	0	0	0	0	**1**	**1**
foolishness	0	0	0	0	0	0	0	0	0	0	1	**1**	**1**

Note: The combination using the sum operator is more commonly used, as it conveys more information about the text - like which words are used more frequently (this is obviously done after removing stopwords).

The representation of all the 3 texts above are -

Sentence	it	was	the	best	of	times	worst	age	wisdom	and	foolishness
It was the best of times	1	1	1	1	1	1	0	0	0	0	0
It was the worst of times	1	1	1	0	1	1	1	0	0	0	0
It was the age of wisdom and the age of foolishness	1	1	2	0	2	0	0	2	1	1	1

d1 d2 d3 $\in \mathbb{R}^{11=N}$

- Here, we see that each of 3 documents are represented as a vector of size 11 (vocabulary size).
- This vectorization technique is easy to implement as well. Plus it also has a implementation in the scikit-learn package.

The above representation is called Bag-of-Words (BOW)

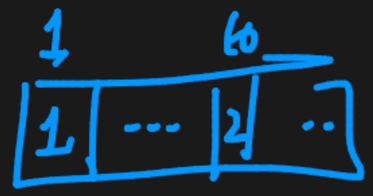
Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:45

Text_Representation.ipynb - C | sklearn.feature_extraction.text

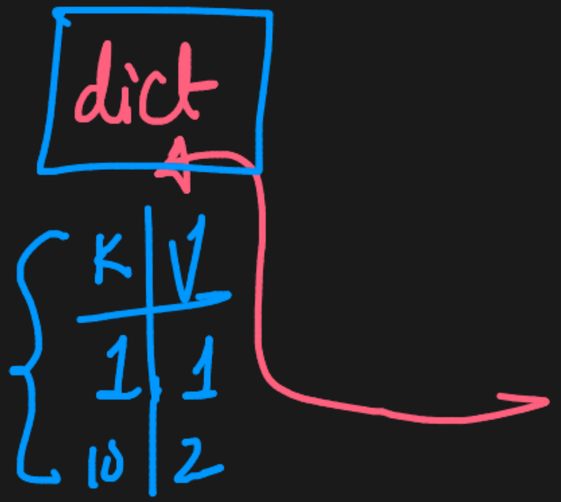
colab.research.google.com/drive/1tNr503prFPk6L-OE54q0RGvLcTnJBTG1#scrollTo=a66ff44f

+ Code + Text Cannot save changes Connect

```python
# sample data
corpus = [
    "it was the best of times",
    "it was the worst of times",
    "it was the age of wisdom and the age of foolishness"
]
# Possible Optimization:
def get_bow_representation(corpus, frequency = True):
    vocabulary = set([x for x in " ".join(corpus).lower().split(" ")])

    bow_rep = []
    for sentence in corpus:
        sentence_rep = dict([(v,0) for v in vocabulary])
        for word in word_tokenize(sentence):
            if frequency:
                sentence_rep[word] += 1
            else:
                sentence_rep[word] = 1
        bow_rep.append(sentence_rep)
    return bow_rep

bow_representation = get_bow_representation(corpus, True)
df = pd.DataFrame(bow_representation)
df.index = corpus
display(df)
```

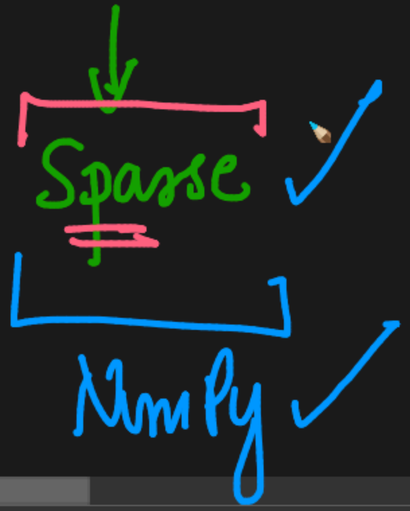


HINT:



Count BoW Vectors

[ase they special]



14 / 14

it was the age of wisdom and the age of foolishness	2	0		1	0	1	0

Problems:

1. The BOW representation considers each word to be equally important (in the final document vector, each word in the text has an equal contribution of 1 at one of the positions).
2. It can not distinguish between the rare important words and the common words.

- **For example:**

> If you're looking at a set of articles about deep learning, then the phrase "neural network" might be present in a lot of articles and hence does not convey a lot of information (considering the corpus).

But the phrase "Reinforcement Learning" might be present in a few, and hence does convey unique information about those specific articles.

▾ Can you think of any drawbacks of BOW representation?

1. This method ignores the location information of the word. It is not possible to grasp the meaning of a word from this representation.
2. The intuition that high-frequency words are more important or give more information about the sentence fails when it comes to stop-words like "is, the, an, I" & when the corpus is context-specific.

2. It can not distinguish between the rare important words and the common words.

- **For example:**

> If you're looking at a set of articles about deep learning, then the phrase "neural network" might be present in a lot of articles and hence does not convey a lot of information (considering the corpus).

But the phrase "Reinforcement Learning" might be present in a few, and hence does convey unique information about those specific articles.

Can you think of any drawbacks of BOW representation?

1. This method ignores the location information of the word. It is not possible to grasp the meaning of a word from this representation.
2. The intuition that high-frequency words are more important or give more information about the sentence fails when it comes to stop-words like “is, the, an, I” & when the corpus is context-specific.
3. For example, in a corpus about covid-19, the word coronavirus may not add a lot of value.

So, is there a way we can create word vectors with rare and important words getting more weightage ?

Advanced BOW

- To suppress the very high-frequency words & ignore the low-frequency words, there is a need to normalize the “weights” of the words accordingly

2. It can not distinguish between the rare important words and the common words.

- **For example:**

> If you're looking at a set of articles about deep learning, then the phrase "neural network" might be present in a lot of articles and hence does not convey a lot of information (considering the corpus).

But the phrase "Reinforcement Learning" might be present in a few, and hence does convey unique information about those specific articles.

Can you think of any drawbacks of BOW representation?

do not like

1. This method ignores the location information of the word. It is not possible to grasp the meaning of a word from this representation.
2. The intuition that high-frequency words are more important or give more information about the sentence fails when it comes to stop-words like “is, the, an, I” & when the corpus is context-specific.
3. For example, in a corpus about covid-19, the word coronavirus may not add a lot of value.

So, is there a way we can create word vectors with rare and important words getting more weightage ?

Advanced BOW

- To suppress the very high-frequency words & ignore the low-frequency words, there is a need to normalize the “weights” of the words accordingly

2. It can not distinguish between the rare important words and the common words.

- **For example:**

> If you're looking at a set of articles about deep learning, then the phrase "neural network" might be present in a lot of articles and hence does not convey a lot of information (considering the corpus).

But the phrase "Reinforcement Learning" might be present in a few, and hence does convey unique information about those specific articles.

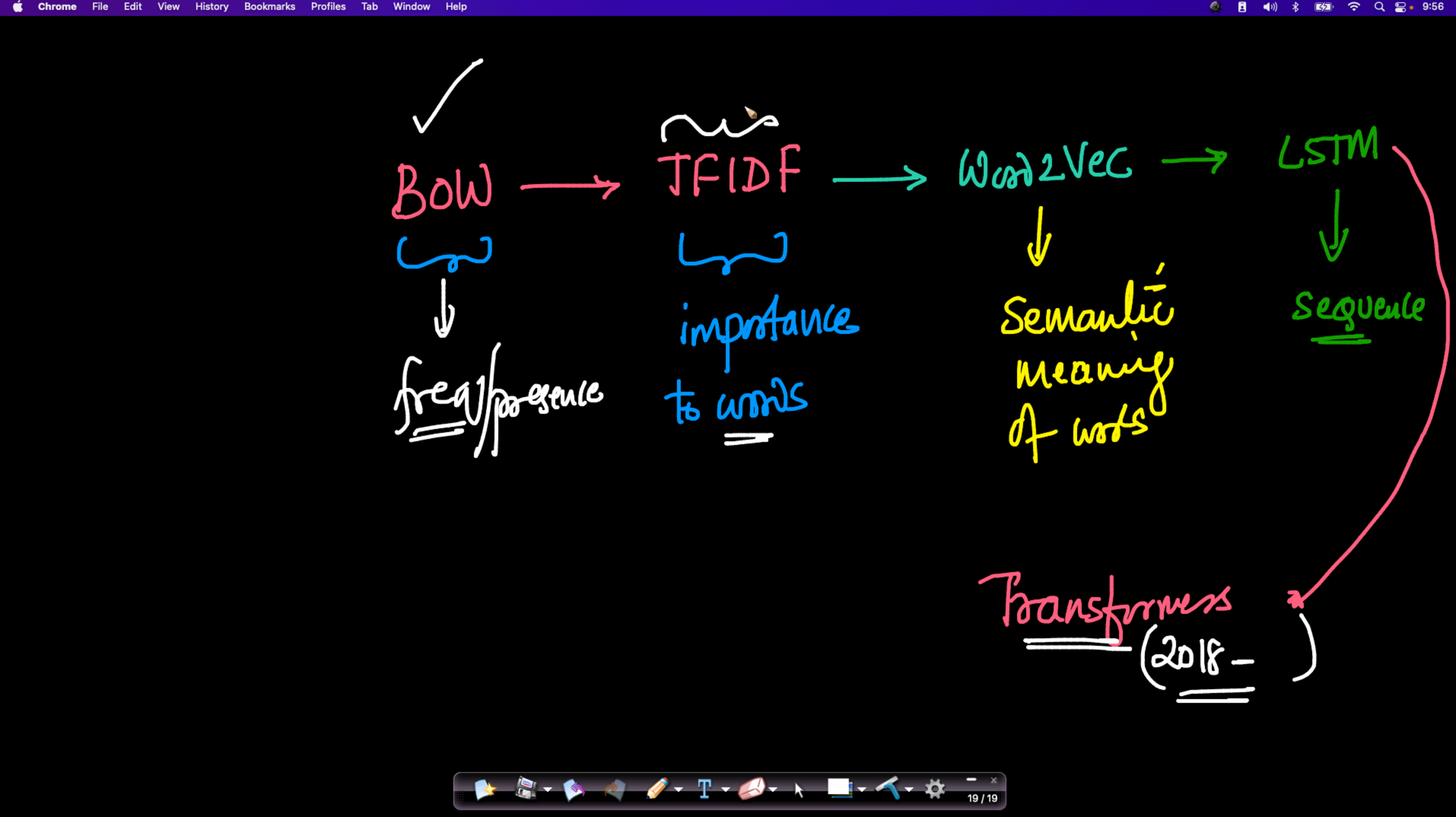
Can you think of any drawbacks of BOW representation?

1. This method ignores the location information of the word. It is not possible to grasp the meaning of a word from this representation.
2. The intuition that high-frequency words are more important or give more information about the sentence fails when it comes to stop-words like “is, the, an, I” & when the corpus is context-specific.
3. For example, in a corpus about covid-19, the word coronavirus may not add a lot of value.

So, is there a way we can create word vectors with rare and important words getting more weightage ?

Advanced BOW

- To suppress the very high-frequency words & ignore the low-frequency words, there is a need to normalize the “weights” of the words accordingly


BoW
TFIDF
Word2Vec
LSTM
freq/presence
importance to words
Semantic meaning of words
Sequence
Transformers (2018-)

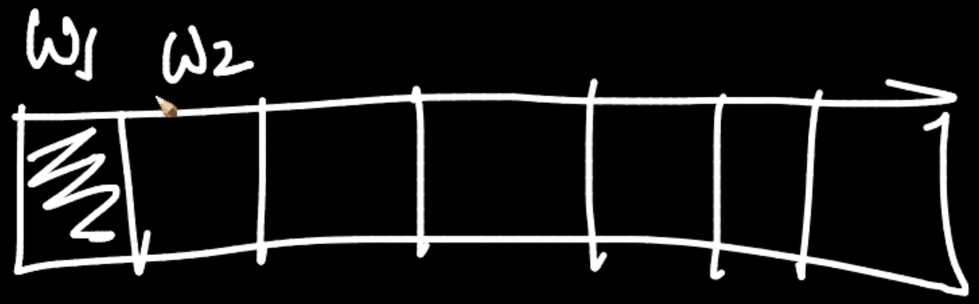
BoW

d_i

BBoW Count BoW

TFIDF

d_i



TF-IDF

↓

term freq

Inv. doc. freq

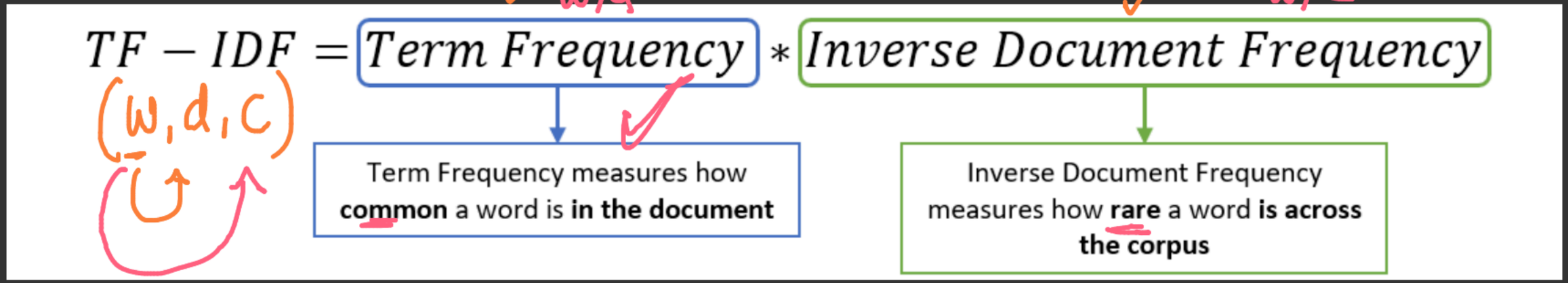
- To suppress the very high-frequency words & ignore the low-frequency words, there is a need to normalize the "weights" of the words accordingly.

TF-IDF

TF-IDF is another very popular technique to get the feature representation for a text in a corpus. Similar to the BOW model, here as well the size of final document vector is equal to the number of tokens considered.

As the name suggests, TF-IDF is mainly composed of two components - TF (Term Frequency) and IDF (Inverse Document Frequency).

```
[ ]  Image.open('/content/tfidf.png')
```



Term Frequency (TF)

[ ]

Term Frequency measures how **common** a word is **in the document**

Inverse Document Frequency measures how **rare** a word **is across the corpus**

Term Frequency (TF)

- Term Frequency is the measure of how common a word (or token) is in the document.
- More common words (or tokens) would have a higher term frequency.This is calculated for every word in a document.
- There are various ways to determine the term frequency. One of the most common formulation of TF is -

$$TF(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

d to t

1000

10

#words in the doc = 1000

Here,

- $TF(t, d)$ is the term Frequency of term t in document d
- $f_{t,d}$ is the frequency of term t in document d

The above formulation is simply the ratio of the frequency of a term in the document to the total number of terms in the document. TF of the term increases as the frequency of the term increases.

Some other formulations for Term Frequency are -

$\sum_{t' \in d} f_{t',d}$

Here,

- $TF(t, d)$ is the term Frequency of term t in document d
- $f_{t,d}$ is the frequency of term t in document d

10/10,000 10/(w)

The above formulation is simply the ratio of the frequency of a term in the document to the total number of terms in the document. TF of the term increases as the frequency of the term increases.

Some other formulations for Term Frequency are -

1. Using the raw count itself, $TF(t, d) = f_{t,d}$
2. Using boolean frequency, $TF(t, d) = 1$ if t occurs in d else 0
3. Logarithmically scaled frquencies, $TF(t, d) = log(1 + f_{t,d})$
4. Augmented frequency to prevent bias towards longer documents,

$$TF(t, d) = 0.5 + 0.5 * \frac{f_{t,d}}{max(f_{t',d} : t' \in d)}$$

Using the same example from before, lets computer the TF for each word in the corpus -

1. It was the best of times
2. It was the worst of times
3. It was the age of wisdom and the age of foolishness

age of foolishness

Inverse Document Frequency (IDF)

- Inverse Document Frequency measures how rare a word (or token) is across corpus.
- A rarer word (or token) would have a larger IDF. There are multiple ways to determine IDF as well.
- One of the most common formulation is -

$$IDF(t, D) = log(\frac{N}{|d \in D : t \in d|})$$

Here,

- $IDF(t, D)$ is the Inverse Document Frequency of term t in corpus D. As we can see from the formulation, IDF is calculated for each word at corpus level and not for individual documents.
- N is the number of documents in the corpus.
- $|d \in D : t \in d|$ is the number of document in corpus D which contains the term t

The above formulation is simply the log scaled inverse fraction of documents which contains the term t in the corpus D.

Log is used as it dampens the effect of huge number of documents in the corpus.

Some other formulations of Inverse Document Frequency are -

age of foolishness

Inverse Document Frequency (IDF)

- Inverse Document Frequency measures how rare a word (or token) is across corpus.
- A rarer word (or token) would have a larger IDF. There are multiple ways to determine IDF as well.
- One of the most common formulation is -

$$IDF(t, D) = log(\frac{N}{|d \in D : t \in d|})$$

Here,

- $IDF(t, D)$ is the Inverse Document Frequency of term t in corpus D. As we can see from the formulation, IDF is calculated for each word at corpus level and not for individual documents.
- N is the number of documents in the corpus.
- $|d \in D : t \in d|$ is the number of document in corpus D which contains the term t

The above formulation is simply the log scaled inverse fraction of documents which contains the term t in the corpus D.

Log is used as it dampens the effect of huge number of documents in the corpus.

Some other formulations of Inverse Document Frequency are -

times	0.166667	0.000000	0.166667	0.166667	0.166667	0.000000	0.166667	0.000000	0.166667	0.000000	0.000000
it was the age of wisdom and the age of foolishness	0.181818	0.090909	0.000000	0.181818	0.000000	0.090909	0.090909	0.090909	0.090909	0.181818	0.000000

Inverse Document Frequency (IDF)

- Inverse Document Frequency measures how rare a word (or token) is across corpus.
- A rarer word (or token) would have a larger IDF. There are multiple ways to determine IDF as well.
- One of the most common formulation is -

$$IDF(t, D) = log(\frac{N}{|d \in D : t \in d|})$$

Here,

- $IDF(t, D)$ is the Inverse Document Frequency of term t in corpus D. As we can see from the formulation, IDF is calculated for each word at corpus level and not for individual documents.
- N is the number of documents in the corpus.
- $|d \in D : t \in d|$ is the number of document in corpus D which contains the term t

The above formulation is simply the log scaled inverse fraction of documents which contains the term t in the corpus D.

it was the age of wisdom and the age of foolishness	0.181818	0.090909	0.000000	0.181818	0.000000	0.090909	0.090909	0.090909	0.090909	0.181818	0.000000

Inverse Document Frequency (IDF)

- Inverse Document Frequency measures how rare a word (or token) is across corpus.
- A rarer word (or token) would have a larger IDF. There are multiple ways to determine IDF as well.
- One of the most common formulation is -

$$IDF(t, D) = log(\frac{N}{|d \in D : t \in d|})$$

Here,

- $IDF(t, D)$ is the Inverse Document Frequency of term t in corpus D. As we can see from the formulation, IDF is calculated for each word at corpus level and not for individual documents.
- N is the number of documents in the corpus.
- $|d \in D : t \in d|$ is the number of document in corpus D which contains the term t

The above formulation is simply the log scaled inverse fraction of documents which contains the term t in the corpus D.

Log is used as it dampens the effect of huge number of documents in the corpus.

Here,

- $IDF(t, D)$ is the Inverse Document Frequency of term t in corpus D. As we can see from the formulation, IDF is calculated for each word at corpus level and not for individual documents.
- N is the number of documents in the corpus.
- $|d \in D : t \in d|$ is the number of document in corpus D which contains the term t

The above formulation is simply the log scaled inverse fraction of documents which contains the term t in the corpus D.

Log is used as it dampens the effect of huge number of documents in the corpus.

Some other formulations of Inverse Document Frequency are -

1. Smoothened IDF,

$$IDF(t, D) = log(\frac{N}{1 + |d \in D : t \in d|}) + 1$$

2. Max IDF,

$$IDF(t, D) = log(\frac{max_{t' \in d}(|d \in D : t' \in d|)}{1 + |d \in D : t \in d|})$$

```
[ ]  def get_inverse_document_frequency(corpus):
         vocabulary = set([x for x in " ".join(corpus).lower().split(" ")])
```

```
            for t, term_freq in tf_dict.items():
                tf_idf_sentence[t] = term_freq * idf[t]
            tf_idf.append(tf_idf_sentence)

        return tf_idf

    tf_idf = get_tf_idf(corpus)
    df = pd.DataFrame(tf_idf)
    df.index = corpus
    display(df)
```

	of	foolishness	times	the	worst	wisdom	it	and	was	age	best
it was the best of times	0.0	0.000000	0.067578	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.183102
it was the worst of times	0.0	0.000000	0.067578	0.0	0.183102	0.000000	0.0	0.000000	0.0	0.000000	0.000000
it was the age of wisdom and the age of foolishness	0.0	0.099874	0.000000	0.0	0.000000	0.099874	0.0	0.099874	0.0	0.199748	0.000000

```
from sklearn.feature_extraction.text import TfidfVectorizer
# using inbuild TfidfVectorizer() function to calculate TF-IDF
tf_idf_vectorizer = TfidfVectorizer()
tf_idf_rep = tf_idf_vectorizer.fit_transform(corpus).todense()
df = pd.DataFrame(tf_idf_rep)
df.columns = tf_idf_vectorizer.get_feature_names()
```

not good

sklearn.feature_extraction.text.TfidfVectorizer

class `sklearn.feature_extraction.text.`**`TfidfVectorizer`**(*, *input='content'*, *encoding='utf-8'*, *decode_error='strict'*, *strip_accents=None*, *lowercase=True*, *preprocessor=None*, *tokenizer=None*, *analyzer='word'*, *stop_words=None*, *token_pattern='(?u)\b\w\w+\b'*, *ngram_range=(1, 1)*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *dtype=<class 'numpy.float64'>*, *norm='l2'*, *use_idf=True*, *smooth_idf=True*, *sublinear_tf=False*) [source]

Unigrams

(1,3)

Convert a collection of raw documents to a matrix of TF-IDF features.

Equivalent to `CountVectorizer` followed by `TfidfTransformer`.

Read more in the User Guide.

$d_1 = \overbrace{w_1 w_3 w_4 w_6 w_1}$

neural networks are very powerful

{ $w_1, w_3, w_4, w_6,$
$w_1w_3, w_3w_4, w_4w_6, w_6w_1,$
$w_1w_3w_4, w_3w_4w_6$ }

Parameters::

input : *{'filename', 'file', 'content'}, default='content'*

- If `'filename'`, the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.
- If `'file'`, the sequence items must have a 'read' method (file-like object) that is called to fetch the bytes in memory.
- If `'content'`, the input is expected to be a sequence of items that can be of type string or byte.

encoding : *str, default='utf-8'*

If bytes or files are given to analyze, this encoding is used to decode.

2. IDF achieves in reducing noise in our matrix.
3. TF-IDF can be used to filter out uncommon & irrelevant words easily helping model train & converge faster.

▸ Drawbacks of TF-IDF representation:

Some of the drawbacks of both BOW and TFIDF method are -

1. For a large corpus, the vocabulary can be huge, hence computation and storage of both BOW and TFIDF can be difficult.
2. Both BOW and TFIDF only considers the presence or absence of words. None of them considers the meaning.

> For example, consider the words 'good', 'nice' and 'bad' in vocabulary. All three of them would be equally different from each other according to the BOW representation, however we know that 'good' and 'nice' are similar in meaning and are both different from the word 'bad'.

▾ Problems with above representations:

- It does not leverage co-occurrence statistics between words. It assumes all words are independent of each other
- This leads to highly sparse vectors with few non zero values
- They do not capture the context or semantics of the word.
- It does not consider spooky & scary as similar but as two independent terms with no commonality between them.

Some of the drawbacks of both BOW and TFIDF method are -

1. For a large corpus, the vocabulary can be huge, hence computation and storage of both BOW and TFIDF can be difficult.
2. Both BOW and TFIDF only considers the presence or absence of words. None of them considers the meaning.

> For example, consider the words 'good', 'nice' and 'bad' in vocabulary. All three of them would be equally different from each other according to the BOW representation, however we know that 'good' and 'nice' are similar in meaning and are both different from the word 'bad'.

Problems with above representations:

- It does not leverage co-occurrence statistics between words. It assumes all words are independent of each other
- This leads to highly sparse vectors with few non zero values
- They do not capture the context or semantics of the word.
- It does not consider spooky & scary as similar but as two independent terms with no commonality between them.
- The representation is directly proportional to vocabulary size. High vocabulary size can lead to memory constraints.

How do we calculate the distance between the words (or how do we measure the difference between the two words) ?

- Each of the word above is representend as a vector of numbers.

Some of the drawbacks of both BOW and TFIDF method are -

1. For a large corpus, the vocabulary can be huge, hence computation and storage of both BOW and TFIDF can be difficult.
2. Both BOW and TFIDF only considers the presence or absence of words. None of them considers the meaning.

> For example, consider the words 'good', 'nice' and 'bad' in vocabulary. All three of them would be equally different from each other according to the BOW representation, however we know that 'good' and 'nice' are similar in meaning and are both different from the word 'bad'.

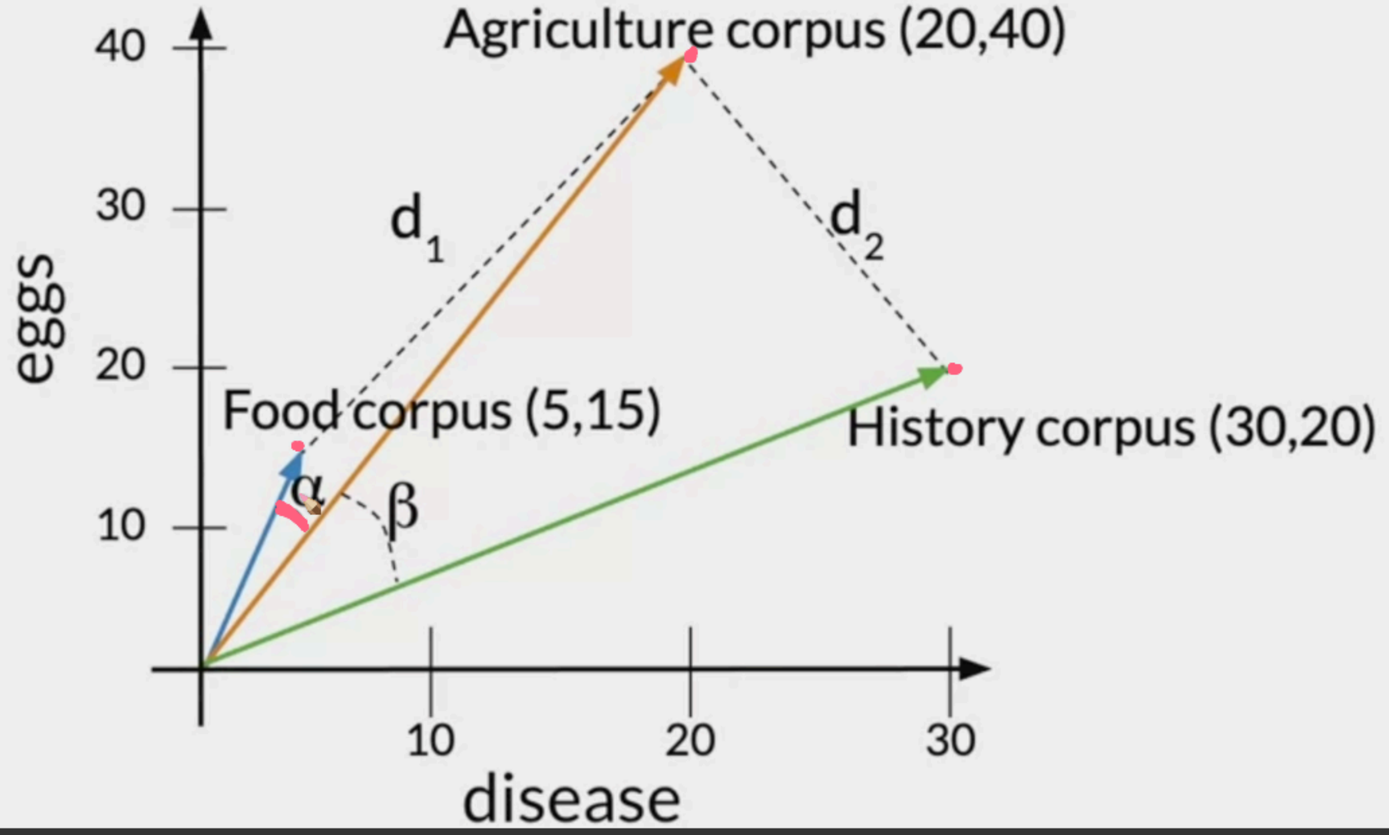
Problems with above representations:

- It does not leverage co-occurrence statistics between words. It assumes all words are independent of each other
- This leads to highly sparse vectors with few non zero values
- They do not capture the context or semantics of the word.
- It does not consider spooky & scary as similar but as two independent terms with no commonality between them.
- The representation is directly proportional to vocabulary size. High vocabulary size can lead to memory constraints.

How do we calculate the distance between the words (or how do we measure the difference between the two words) ?

- Each of the word above is representend as a vector of numbers.

Euclidean distance vs Cosine similarity



Euclidean distance: $d_2 < d_1$

Angles comparison: $\beta > \alpha$

The cosine of the angle between the vectors

Coming back to the original problem -

```
[ ]     return sentence
```

```
from tqdm.notebook import tqdm
# tqdm to see real time progress
tqdm.pandas()

nlp = spacy.load('en_core_web_sm') # English pipeline optimized for CPU

def process_article(article_text, nlp_object):
    processed_article_sentences = []
    # using nltk sentence tokenizer
    for sentence in sent_tokenize(article_text):
        # preprocessing each sentence using our process_sentence function
        processed_article_sentences.append(process_sentence(sentence, nlp_object))
    # joining preprocessed sentence as a complete paragrams of the article
    return " ".join(processed_article_sentences)

articles["processed_text"] = articles["text"].progress_apply(lambda x : process_article(x, nlp))
```

```
100% 208/208 [03:35<00:00, 1.05s/it]
```

At this point we have processed and tokenized the article text. Now finally let us try to find out the similarity between the articles.

If a callable is passed it is used to extract the sequence of features out of the raw, unprocessed input.

Changed in version 0.21.

Since v0.21, if `input` is `filename` or `file`, the data is first read from the file and then passed to the given callable analyzer.

max_df : *float in range [0.0, 1.0] or int, default=1.0*

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

min_df : *float in range [0.0, 1.0] or int, default=1*

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

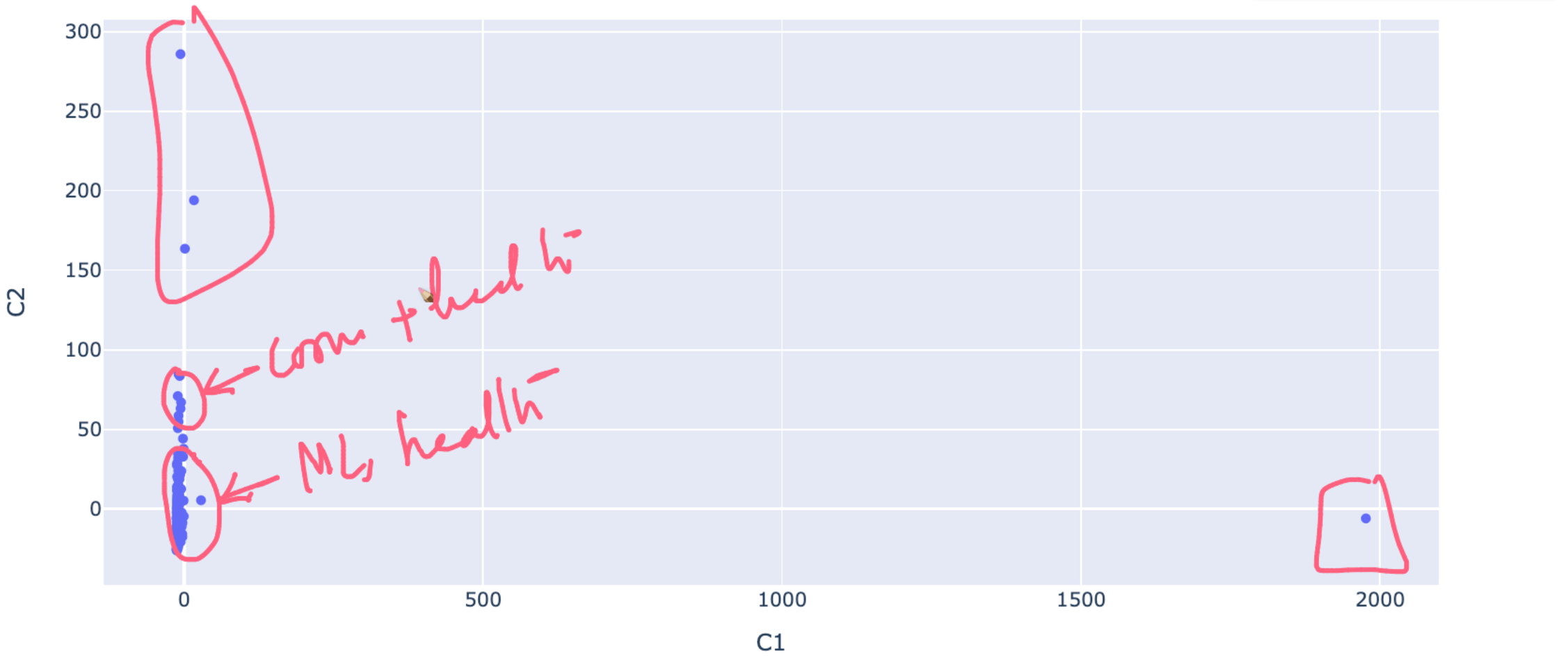
max_features : *int, default=None*

If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.

This parameter is ignored if vocabulary is not None.

vocabulary : *Mapping or iterable, default=None*

Either a Mapping (e.g., a dict) where keys are terms and values are indices in the feature matrix, or an iterable over terms. If not given, a vocabulary is determined from the input documents. Indices in the mapping should not be repeated and should not have any gap between 0 and the largest index.

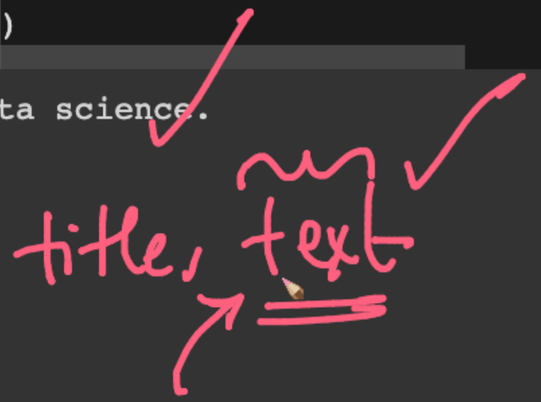


```
    print()

# Let us check top 5 similar articles for some of the articles in our corpus
get_similar_documents(bow_features_df, 90, count_vectorizer.get_feature_names())
get_similar_documents(bow_features_df, 80, count_vectorizer.get_feature_names())
get_similar_documents(bow_features_df, 150, count_vectorizer.get_feature_names())
get_similar_documents(bow_features_df, 205, count_vectorizer.get_feature_names())
```

```
Reference Article : 17 types of similarity and dissimilarity measures used in data science.
**** Similar Articles ****
9 Distance Measures in Data Science
17 Clustering Algorithms Used In Data Science and Mining
Machine Learning Basics with the K-Nearest Neighbors Algorithm
OVER 100 Data Scientist Interview Questions and Answers!
Fundamental Techniques of Feature Engineering for Machine Learning

Reference Article : TensorFlow Tutorial Part 1
**** Similar Articles ****
How to go from a Python newbie to a Google Certified TensorFlow Developer under two months
Enchanted Random Forest
PCA using Python (scikit-learn)
The 7 Best Data Science and Machine Learning Podcasts
Building RNN, LSTM, and GRU for time series using PyTorch

Reference Article : The One Word That Explains Why Economics Professors Are Not Billionaires
**** Similar Articles ****
Why People Still Don't Get Cryptocurrency
```

```
    print()

# Let us check top 5 similar articles for some of the articles in our corpus
get_similar_documents(bow_features_df, 90, count_vectorizer.get_feature_names())
get_similar_documents(bow_features_df, 80, count_vectorizer.get_feature_names())
get_similar_documents(bow_features_df, 150, count_vectorizer.get_feature_names())
get_similar_documents(bow_features_df, 205, count_vectorizer.get_feature_names())
```

```
Reference Article : 17 types of similarity and dissimilarity measures used in data science.
**** Similar Articles ****
9 Distance Measures in Data Science
17 Clustering Algorithms Used In Data Science and Mining
Machine Learning Basics with the K-Nearest Neighbors Algorithm
OVER 100 Data Scientist Interview Questions and Answers!
Fundamental Techniques of Feature Engineering for Machine Learning

Reference Article : TensorFlow Tutorial Part 1
**** Similar Articles ****
How to go from a Python newbie to a Google Certified TensorFlow Developer under two months
Enchanted Random Forest
PCA using Python (scikit-learn)
The 7 Best Data Science and Machine Learning Podcasts
Building RNN, LSTM, and GRU for time series using PyTorch

Reference Article : The One Word That Explains Why Economics Professors Are Not Billionaires
**** Similar Articles ****
Why People Still Don't Get Cryptocurrency
```



+ Code + Text Cannot save changes

Connect

```
You Will Never Be Rich If You Keep Doing These 10 things
How I transformed $6,000 to $3,000,000
The Exact Steps I Followed to Make $1,500+ of Passive Income Every Month

Reference Article : How I lost 10kg in 60 days: My 7-step weight loss plan
**** Similar Articles ****
10 Things You Can Do This Morning To Heal Your Anxiety
How to lose 10+ pounds of fat a month- even if you have a slow metabolism
The Diet & Workout Plan of a Full-Time Traveling Family  HIS
The cure for type 2 diabetes is known, but few are aware
The Easiest Way to Lose 125 Pounds Is to Gain 175 Pounds

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please
```

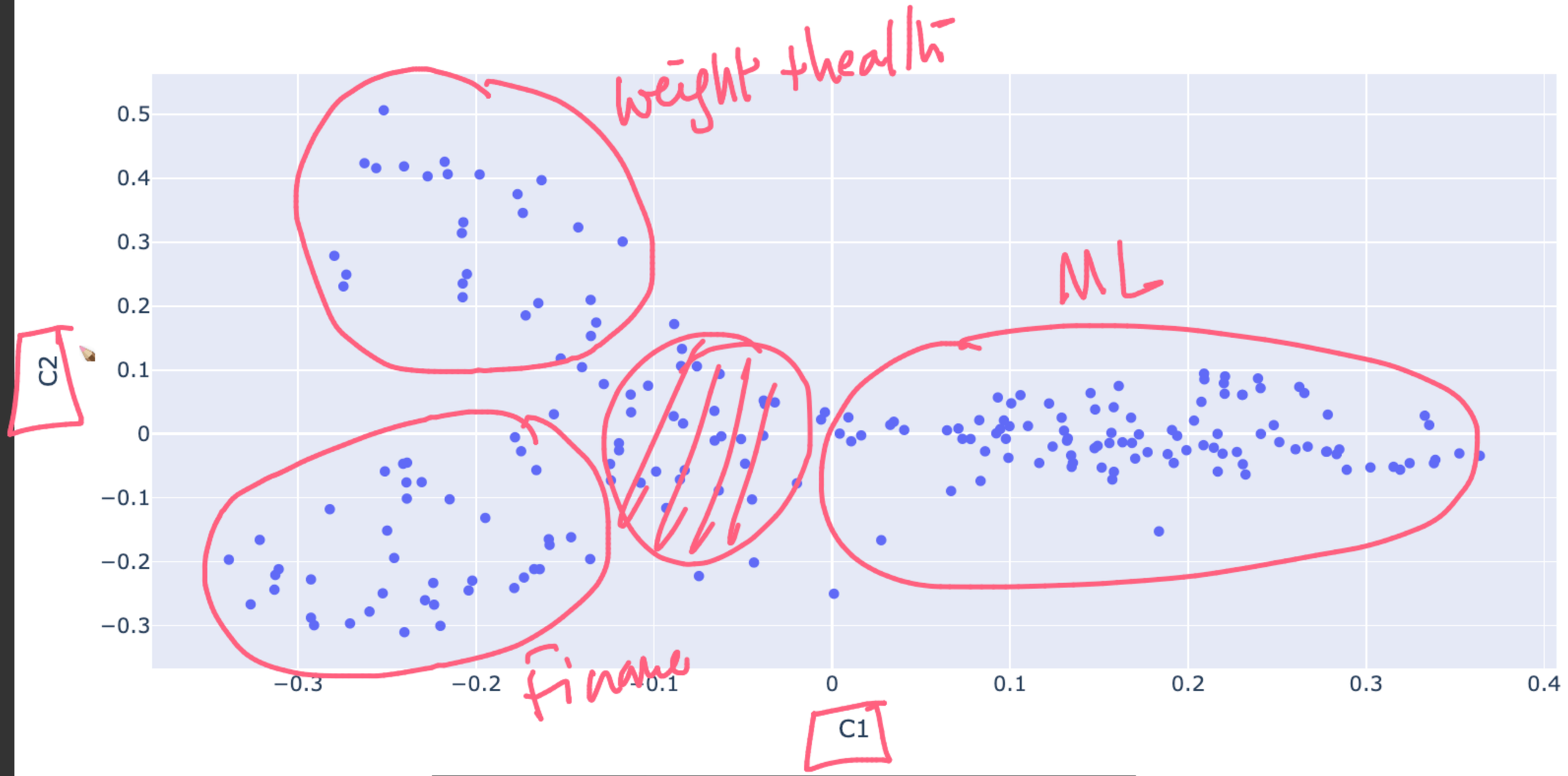
text

42 / 42

```
Building RNN, LSTM, and GRU for time series using PyTorch

Reference Article : The One Word That Explains Why Economics Professors Are Not Billionaires
**** Similar Articles ****
Why People Still Don't Get Cryptocurrency
You May Have A Poor Person's Mindset And Not Know It
You Will Never Be Rich If You Keep Doing These 10 things
How I transformed $6,000 to $3,000,000
The Exact Steps I Followed to Make $1,500+ of Passive Income Every Month

Reference Article : How I lost 10kg in 60 days: My 7-step weight loss plan
**** Similar Articles ****
10 Things You Can Do This Morning To Heal Your Anxiety
How to lose 10+ pounds of fat a month- even if you have a slow metabolism
The Diet & Workout Plan of a Full-Time Traveling Family  HIS
The cure for type 2 diabetes is known, but few are aware
The Easiest Way to Lose 125 Pounds Is to Gain 175 Pounds

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:
```

[ ]

Principal components for TFIDF document representation



Yes you should understand backprop

[ ] PCA using Python (scikit-learn)

Reference Article : The One Word That Explains Why Economics Professors Are Not Billionaires
**** Similar Articles ****
You Will Never Be Rich If You Keep Doing These 10 things
Why People Still Don't Get Cryptocurrency
You May Have A Poor Person's Mindset And Not Know It
How I transformed $6,000 to $3,000,000
4 Reasons Why Economists Make Great Data Scientists (And Why No One Tells Them)

Reference Article : How I lost 10kg in 60 days: My 7-step weight loss plan
**** Similar Articles ****
How to lose 10+ pounds of fat a month- even if you have a slow metabolism
10 Things You Can Do This Morning To Heal Your Anxiety
The Diet & Workout Plan of a Full-Time Traveling Family HIS
The Science Behind Fat Metabolism
The cure for type 2 diabetes is known, but few are aware

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Finding similar Medium articles

BOW, TFIDF, Word2Vec

`You are working as a Data Scientist at Medium`

- Medium is an online publishing platform which hosts a hybrid collection of blog posts from both amateur and professional people and publications.
- In 2020, about 47,000 articles were published daily on the plaform and it had about 200M visitors every month.

LSTM

Problem Statement:

- You want to give readers a better reading experience at Medium. To do that, you want to recommend articles to the user on the basis of current article that the user is reading.
- **More concretely, given a Medium article find a set of similar articles.**

BERT/Transformers

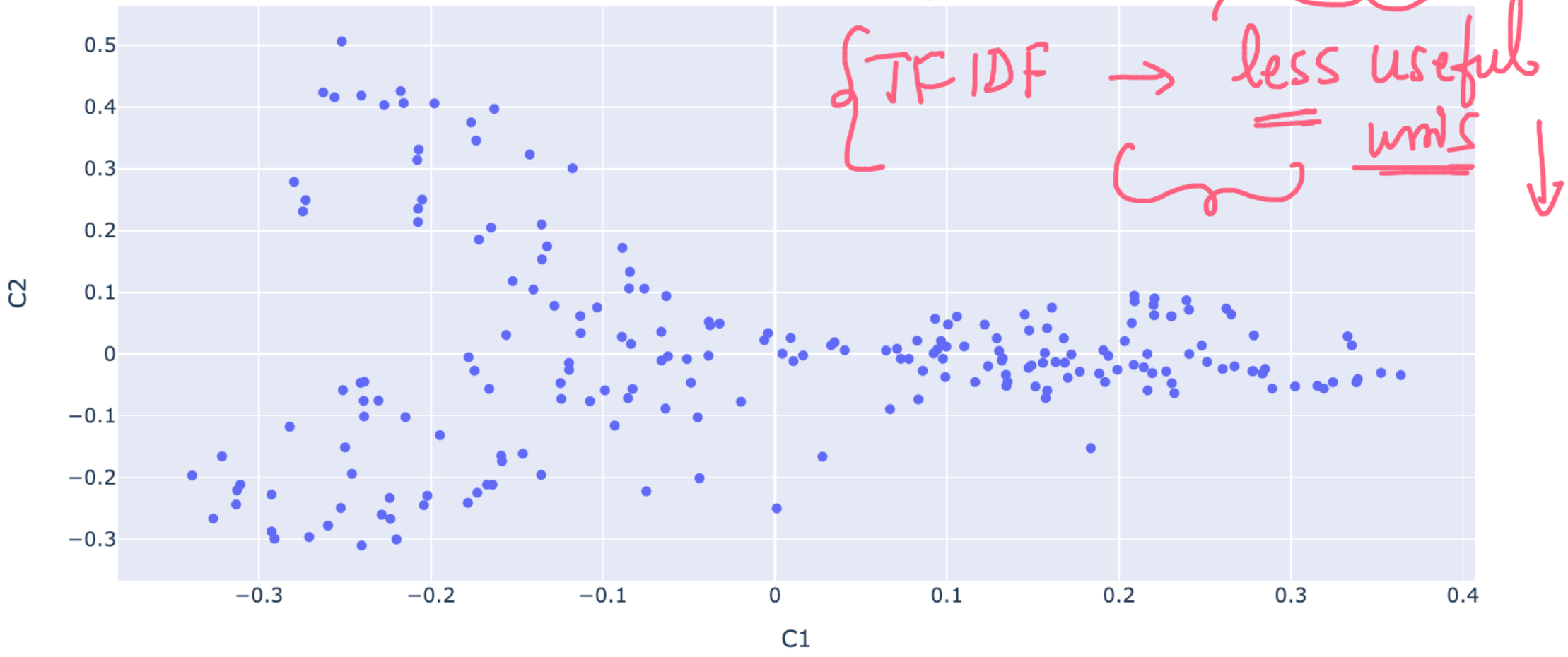
How would a human find similar articles in a corpus ?

1. Look at the title - find similar titles.
2. Find articles by the same author.
3. Go through the text, understand it and group the aricles within broader topics.

Principal components for TFIDF document representation

Count BOW → equal weightage

{TFIDF → less useful words ↓



Finding similar Medium articles

`You are working as a Data Scientist at Medium`

- Medium is an online publishing platform which hosts a hybrid collection of blog posts from both amateur and professional people and publications.
- In 2020, about 47,000 articles were published daily on the plaform and it had about 200M visitors every month.

Problem Statement:

- You want to give readers a better reading experience at Medium. To do that, you want to recommend articles to the user on the basis of current article that the user is reading.
- **More concretely, given a Medium article find a set of similar articles.**

How would a human find similar articles in a corpus ?

1. Look at the title - find similar titles.
2. Find articles by the same author.