



Search Medium

Write



★ Member-only story

How to Train a Word2Vec Model from Scratch with Gensim

In this article we will explore Gensim, a very popular Python library for training text-based machine learning models, to train a Word2Vec model from scratch



Andrea D'Agostino · Follow

Published in Towards Data Science · 9 min read · Feb 7



100



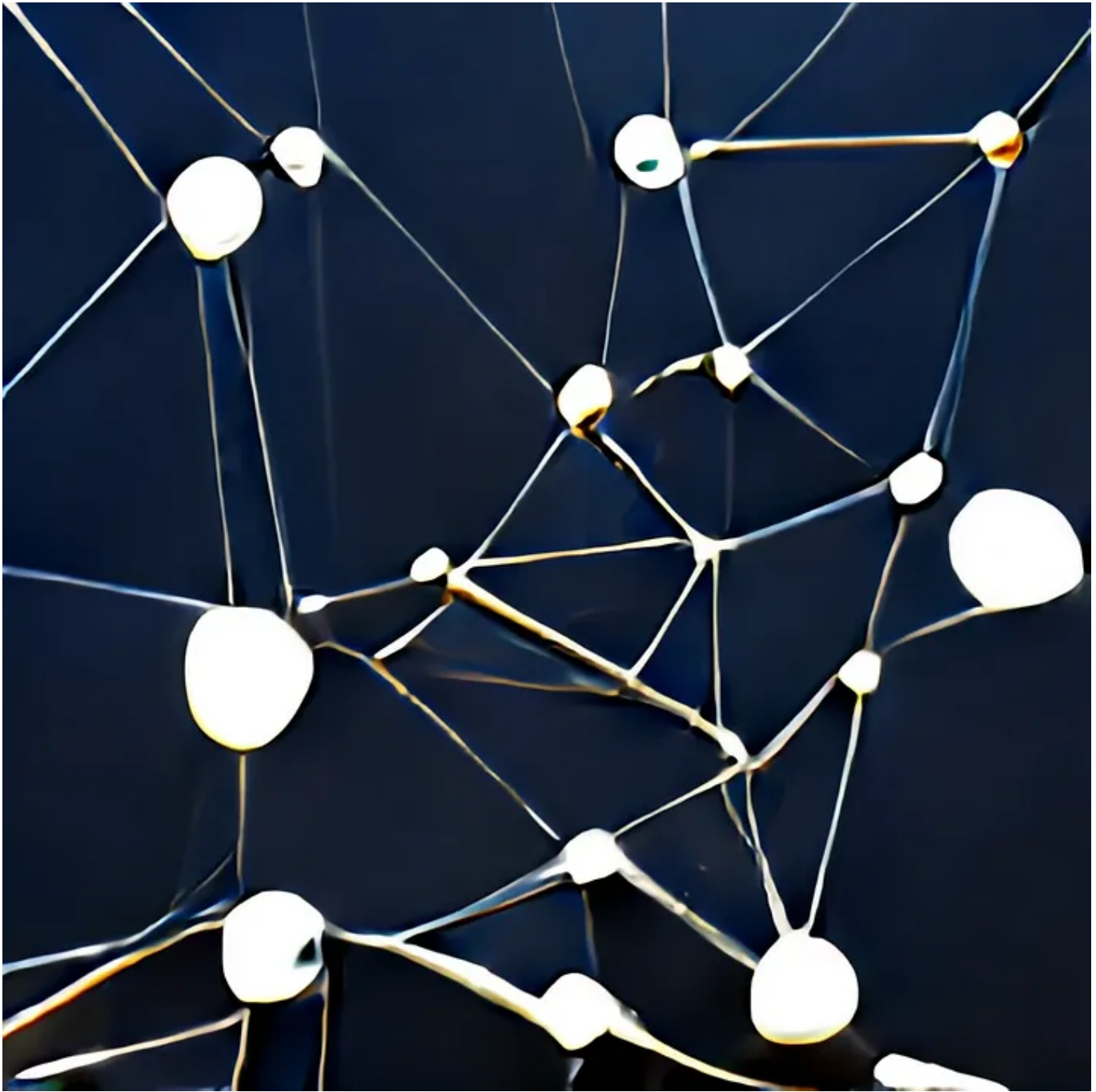


Image by author.

Word2Vec is a machine learning algorithm that allows you to create vector representations of words.

These representations, called **embeddings**, are used in many natural language processing tasks, such as word clustering, classification, and text generation.

The Word2Vec algorithm marked the beginning of an era in the NLP world when it was first introduced by Google in 2013.

It is based on word representations created by a neural network trained on very large data corpuses.

The output of Word2Vec are vectors, one for each word in the training dictionary, that effectively capture relationships between words.

Vectors that are close together in vector space have similar meanings based on context, and vectors that are far apart have different meanings. For example, the words “strong” and “mighty” would be close together while “strong” and “Paris” would be relatively far away within the vector space.

This is a significant improvement over the performance of the bag-of-words model, which is based on simply counting the tokens present in a textual data corpus.

In this article we will explore Gensim, a popular Python library for training text-based machine learning models, **to train a Word2Vec model from scratch.**

I will use the articles from my from my personal blog in Italian to act as a textual corpus for this project. Feel free to use whatever corpus you wish —

the pipeline is extendable.

This approach is adaptable to any textual dataset. You'll be able to create the embeddings yourself and visualize them.

Let's begin!

Project requirements

Let's draw up a list of actions to do that serve as foundations of the project.

1. We'll create a new virtual environment
(read here to understand how: [How to Set Up a Development Environment for Machine Learning](#))
2. Install the dependencies, among which Gensim
3. Prepare our corpus to deliver to Word2Vec
4. Train the model and save it
5. Use TSNE and Plotly to visualize embeddings to visually understand the vector space generated by Word2Vec
6. BONUS: Use the Datapane library to create an interactive HTML report to share with whoever we want

By the end of the article we will have in our hands an excellent basis for developing more complex reasoning, such as clustering of embeddings and more.

I'll assume you've already configured your environment correctly, so I won't explain how to do it in this article. Let's start right away with downloading the blog data.

Dependencies

Before we begin let's make sure to install the following project level dependencies by running `pip install XXXXX` in the terminal.

- `trafilatura`
- `pandas`
- `gensim`
- `nltk`
- `tqdm`
- `scikit-learn`
- `plotly`
- `datapane`

We will also initialize a `logger` object to receive Gensim messages in the terminal.

Retrieve the corpus data

As mentioned we will use the articles of my personal blog in Italian (diariodiunanalista.it) for our corpus data.

Here is how it appears in Deepnote.

● Ready

▶ Run

URLs: 100% ██████████ 54/54 [00:32<00:00, 1.68it/s]

Websites: 100% ██████████ 1/1 [00:32<00:00, 32.11s/it]

df



url object		article object	
https://www.diariodiunanalista.it	1.9%	Scopri come creare una classe chiamata Be...	9.3%
https://www.diariodiunanalista.it/abo...	1.9%	Ciao! Sono Andrea, analista nel settore del ...	1.9%
52 others	96.3%	48 others	88.9%
0	https://www.diariodiunanalista.it	Scopri come creare una classe chiamata Benchmark in Python per confrontare e valutare le prestazioni dei model...	
1	https://www.diariodiunanalista.it/about/	Ciao! Sono Andrea, analista nel settore del marketing e business intelligence con oltre 6 anni di esperienza a...	
2	https://www.diariodiunanalista.it/author/andrea-dagostino/	Scopri come creare una classe chiamata Benchmark in Python per confrontare e valutare le prestazioni dei model...	
3	https://www.diariodiunanalista.it/author/kasun/	Scopri come creare una classe chiamata Benchmark in Python per confrontare e valutare le prestazioni dei model...	
4	https://www.diariodiunanalista.it/contact/	Contatti NOME INDIRIZZO EMAIL MESSAGGIO INVIA Il tuo messaggio è stato correttamente inviato. Oops! Qualcosa ...	
5	https://www.diariodiunanalista.it/glossario/	Le informazioni in questa pagina sono in continuo aggiornamento. Accuratezza L'accuratezza è una metrica ...	
6	https://www.diariodiunanalista.it/posts/6-cose-da-fare-prima-di-addestrare-il-tuo-modello/	In questo articolo scriverò di una metodica utile per prepararsi alla fase di addestramento di un modello. Si...	
7	https://www.diariodiunanalista.it/posts/analisi-esplorativa-dei-dati-con-python-e-pandas/	L'analisi esplorativa del dato (exploratory data analysis, EDA) è di fondamentale importanza perché permette...	
8	https://www.diariodiunanalista.it/posts/bag-of-words-cosa-e-e-come-funziona/	Il modello più comune per rappresentare numericamente del testo è il modello bag of words. L'idea è molto semplic...	
9	https://www.diariodiunanalista.it/posts/classificazio-ne-binaria-di-immagini-con-tensorflow/	In questo post vedremo come costruire un modello di classificazione binaria con Tensorflow per differenziare tr...	

The data we collected in pandas dataframe format. Image by author.

The textual data that we are going to use is under the *article* column. Let's see what a random text looks like

```
from pprint import pprint
pprint(df.iloc[6].article)
```

```
('In questo articolo scriverò di una metodica utile per prepararsi alla fase '
'di addestramento di un modello. Si tratta di un processo formato da 6 step, '
'ognuno utile per assicurare che un corretto flusso di dati sia immesso nel '
'nostro modello. Seguire questi passaggi mi ha reso più efficiente nel lavoro '
'e più sicuro di me quando mostravo i miei risultati.\n'
'Voglio ringraziare Andriy Burkov e il suo lavoro con il libro Machine '
'Learning Engineering. Ne ho tratto ispirazione per scrivere questo articolo '
'ed è nel complesso un'ottima lettura: leggetelo se non l'avete già fatto.\n"
'Cominciamo.\n'
'1. Il file di schema\n'
'Il file schema viene utilizzato per tenere traccia di quali sono le tue '
'feature, di come si comportano e delle loro proprietà generali. È utile '
'perché garantisce che tutto il team sia aggiornato su ciò che viene fornito '
'al modello, in tutte le fasi dello sviluppo. Fornisce inoltre al team una '
'linea guida formale da seguire durante il debug del modello.\n'
'Crea un file che soddisfi almeno questi criteri:\n'
'- contenga il nome delle feature\n'
'- il loro tipo (categoriale, numerico, ...)\n'
'- i valori minimi e massimi consentiti\n'
'- media e deviazione standard\n'
'- se consente la presenza di zeri\n'
'- se consente la presenza di valori non definiti (NaN, None, ...)\n'
'Sentitevi liberi di creare qualsiasi tipo di file per contenere queste '
'informazioni. Ecco un esempio di un file schema.json che contiene queste '
'informazioni per un set di dati sintetico.\n'
'2. Il file README.md\n'
'Una delle mie attività preferite è scrivere i miei pensieri prima e durante '
'il mio progetto di machine learning. Sfrutto il file README.md, che è il '
```

Regardless of language, this should be processed before being delivered to the Word2Vec model. We have to go and remove the Italian stopwords, clean up punctuation, numbers and other symbols. This will be the next step.

Preparation of the data corpus

The first thing to do is to import some fundamental dependencies for preprocessing.

```
# Text manipulation libraries
import re
import string
import nltk
from nltk.corpus import stopwords
# nltk.download('stopwords') <-- we run this command to download the stopwords i
# nltk.download('punkt') <-- essential for tokenization

stopwords.words("italian")[:10]
>>> ['ad', 'al', 'allo', 'ai', 'agli', 'all', 'agl', 'alla', 'alle', 'con']
```

Now let's create a `preprocess_text` function that takes some text as input and returns a clean version of it.

```
def preprocess_text(text: str, remove_stopwords: bool) -> str:
    """Function that cleans the input text by going to:
    - remove links
    - remove special characters
    - remove numbers
    - remove stopwords
    - convert to lowercase
    - remove excessive white spaces
    Arguments:
        text (str): text to clean
        remove_stopwords (bool): whether to remove stopwords
    Returns:
        str: cleaned text
    """
    # remove links
    text = re.sub(r"http\S+", "", text)
    # remove numbers and special characters
    text = re.sub("[^A-Za-z]+", " ", text)
    # remove stopwords
    if remove_stopwords:
        # 1. create tokens
        tokens = nltk.word_tokenize(text)
        # 2. check if it's a stopwords
        tokens = [w.lower().strip() for w in tokens if not w.lower() in stopwords]
```



```
# return a list of cleaned tokens  
return tokens
```

Let's apply this function to the Pandas dataframe by using a lambda function with `.apply`.

```
df["cleaned"] = df.article.apply(  
    lambda x: preprocess_text(x, remove_stopwords=True)  
)
```

We get a clean series.

```
df["cleaned"] = df.article.apply(lambda x: preprocess_text(x, remove_stopwords=True))
```

```
0    [scopri, creare, classe, chiamata, benchmark, ...  
1    [ciao, andrea, analista, settore, marketing, b...  
2    [scopri, creare, classe, chiamata, benchmark, ...  
3    [scopri, creare, classe, chiamata, benchmark, ...  
4    [contatti, nome, indirizzo, email, messaggio, ...  
5    [informazioni, pagina, continuo, aggiornamento...  
6    [articolo, scrire, metodica, utile, preparars...  
7    [analisi, esplorativa, dato, exploratory, data...  
8    [modello, pi, comune, rappresentare, numericam...  
9    [post, vedremo, costruire, modello, classifica...  
10   [analista, indipendentemente, settore, lavorat...  
11   [calcolare, similarit, due, testi, attivit, mo...
```

Each article has been cleaned and tokenized. Image by author.

Let's examine a text to see the effect of our preprocessing.

```
pprint(df["cleaned"].iloc[6])
```



```
['articolo',  
 'scriver',  
 'metodica',  
 'utile',  
 'prepararsi',  
 'fase',  
 'addestramento',  
 'modello',  
 'tratta',  
 'processo',
```

How a single cleaned text appears. Image by author.

The text now appears to be ready to be processed by Gensim. Let's carry on.

Word2Vec training

The first thing to do is create a variable `texts` that will contain our texts.

```
texts = df.cleaned.tolist()
```

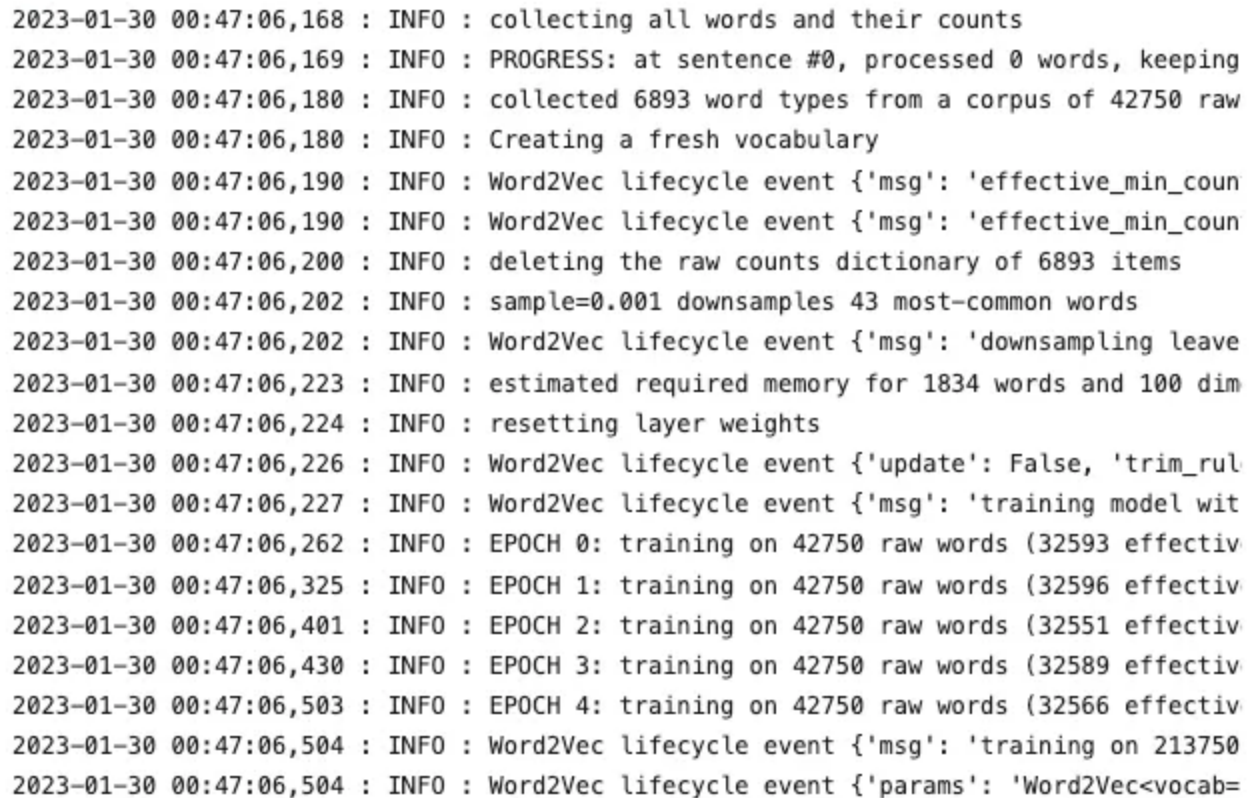
We are now ready to train the model. Word2Vec can accept many parameters, but let's not worry about that for now. Training the model is straightforward, and requires one line of code.

```
from gensim.models import Word2Vec  
  
model = Word2Vec(sentences=texts)
```

```
from gensim.models import Word2Vec
```

[55]

```
model = Word2Vec(sentences=texts)
```

A terminal window showing the output of a Word2Vec training process. The output consists of multiple lines of log messages, each starting with a timestamp (e.g., 2023-01-30 00:47:06,168) and an INFO level message. The messages describe the progress of the training, including collecting words, creating a vocabulary, downsampling, and training over several epochs. The last line is partially cut off.

```
2023-01-30 00:47:06,168 : INFO : collecting all words and their counts
2023-01-30 00:47:06,169 : INFO : PROGRESS: at sentence #0, processed 0 words, keeping
2023-01-30 00:47:06,180 : INFO : collected 6893 word types from a corpus of 42750 raw
2023-01-30 00:47:06,180 : INFO : Creating a fresh vocabulary
2023-01-30 00:47:06,190 : INFO : Word2Vec lifecycle event {'msg': 'effective_min_coun
2023-01-30 00:47:06,190 : INFO : Word2Vec lifecycle event {'msg': 'effective_min_coun
2023-01-30 00:47:06,200 : INFO : deleting the raw counts dictionary of 6893 items
2023-01-30 00:47:06,202 : INFO : sample=0.001 downsamples 43 most-common words
2023-01-30 00:47:06,202 : INFO : Word2Vec lifecycle event {'msg': 'downsampling leave
2023-01-30 00:47:06,223 : INFO : estimated required memory for 1834 words and 100 dim
2023-01-30 00:47:06,224 : INFO : resetting layer weights
2023-01-30 00:47:06,226 : INFO : Word2Vec lifecycle event {'update': False, 'trim_rul
2023-01-30 00:47:06,227 : INFO : Word2Vec lifecycle event {'msg': 'training model wit
2023-01-30 00:47:06,262 : INFO : EPOCH 0: training on 42750 raw words (32593 effectiv
2023-01-30 00:47:06,325 : INFO : EPOCH 1: training on 42750 raw words (32596 effectiv
2023-01-30 00:47:06,401 : INFO : EPOCH 2: training on 42750 raw words (32551 effectiv
2023-01-30 00:47:06,430 : INFO : EPOCH 3: training on 42750 raw words (32589 effectiv
2023-01-30 00:47:06,503 : INFO : EPOCH 4: training on 42750 raw words (32566 effectiv
2023-01-30 00:47:06,504 : INFO : Word2Vec lifecycle event {'msg': 'training on 213750
2023-01-30 00:47:06,504 : INFO : Word2Vec lifecycle event {'params': 'Word2Vec<vocab=
```

Word2Vec training process. Image by author.

Our model is ready and the embeddings have been created. To test this, let's try to find the vector for the word *overfitting*.

```
model.wv["overfitting"]
```

```
array([-0.22198907,  0.28976303,  0.11608665, -0.00550745,  0.06410174,
        -0.48978075,  0.07284468,  0.47414976, -0.29580674, -0.10198367,
        -0.07399186, -0.43323174, -0.07892533,  0.04981417,  0.02673437,
        -0.1427754 ,  0.02709657, -0.3608234 ,  0.0594544 , -0.310252 ,
         0.10276494,  0.2477989 ,  0.21537398, -0.04340618, -0.1311449 ,
         0.09594796, -0.13757147, -0.20405295, -0.22720689, -0.12387417,
         0.3743956 , -0.04883793, -0.07318234, -0.08002541, -0.15508465,
         0.40453225,  0.02826039, -0.16891725, -0.09861258, -0.35998333,
         0.00665827, -0.25738716, -0.10918178,  0.09559031,  0.17996764,
        -0.03532981, -0.20710756,  0.01731221,  0.14729872,  0.2630963 ,
         0.1575338 , -0.24533197, -0.02607119,  0.04679034, -0.20006879,
         0.16065556,  0.28003168, -0.01486896, -0.23266923, -0.02492916,
         0.04792764,  0.17320415, -0.21811853, -0.17410101, -0.1026457 ,
         0.15739748,  0.10373436,  0.15233436, -0.17445575,  0.2989354 ,
        -0.13804555, -0.01456112,  0.2776745 , -0.08357789,  0.16453102,
         0.225563 , -0.19896458, -0.04129031, -0.24628311,  0.06993658,
        -0.10684197, -0.06381005, -0.21999122,  0.25840524, -0.05988453,
         0.12119435, -0.08025485,  0.19577762,  0.3067177 ,  0.08113635,
         0.10115074,  0.16811635,  0.02040015,  0.00261409,  0.41897896,
         0.2655418 ,  0.13347937, -0.30184868, -0.06821583,  0.00569795],
      dtype=float32)
```

Word embeddings for the word “overfitting”. Image by author.

By default, Word2Vec creates 100-dimensional vectors. This parameter can be changed, along with many others, when we instantiate the class. In any case, the more dimensions associated with a word, **the more information the neural network will have about the word itself and its relationship to the others.**

Obviously this has a **higher computational and memory cost.**

Please note: one of the most important limitations of Word2Vec is **the inability to generate vectors for words not present in the vocabulary** (called OOV — out of vocabulary words).

```
model.wv["serendipity"]
```

KeyError: "Key 'serendipity' not present"

[Show error details](#)[Search on Stack Overflow](#)

A major limitation of W2V is the inability to map embeddings for out of vocabulary words. Image by author.

To handle new words, therefore, we'll have to either train a new model or add vectors manually.

Calculate the similarity between two words

With the cosine similarity we can calculate how far apart the vectors are in space.

With the command below we instruct Gensim to find the first 3 words most similar to *overfitting*

```
model.wv.most_similar(positive=['overfitting'], topn=3)
```

```
model.wv.most_similar(positive=['overfitting'], topn=3)
```

```
[('apprendimento', 0.9901367425918579),  
 ('modelli', 0.9874876737594604),  
 ('quando', 0.9824953079223633)]
```

The most similar words to "overfitting". Image by author.

Let's see how the word "when" (*quando* in Italian) is present in this result. It will be appropriate to include similar adverbs in the stop words to clean up

the results.

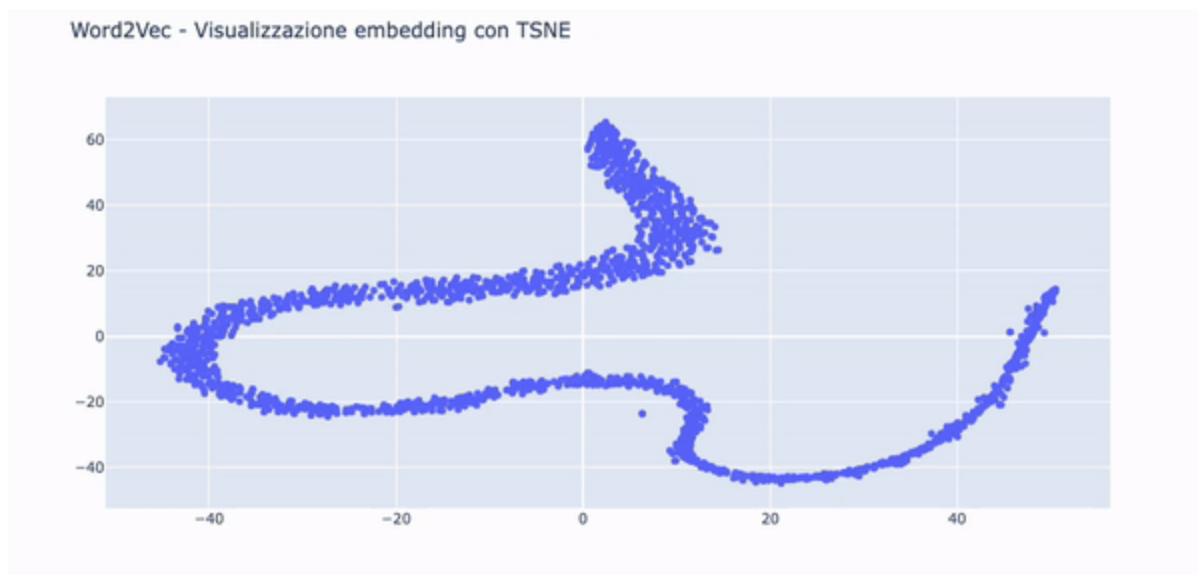
To save the model, just do `model.save("./path/to/model")`.

Visualize embeddings with TSNE and Plotly

Our vectors are 100-dimensional. It's a problem to visualize them unless we do something to **reduce their dimensionality**.

We will use the TSNE, a technique to reduce the dimensionality of the vectors and create two components, one for the X axis and one for the Y axis on a scatterplot.

In the .gif below you can see the words embedded in the space thanks to the Plotly features.



How embeddings of my Italian blog appear in TSNE projection. Image by author.

Here is the code to generate this image.

```
def reduce_dimensions(model):  
    num_components = 2 # number of dimensions to keep after compression  
  
    # extract vocabulary from model and vectors in order to associate them in the  
    vectors = np.asarray(model.wv.vectors)  
    labels = np.asarray(model.wv.index_to_key)  
  
    # apply TSNE  
    tsne = TSNE(n_components=num_components, random_state=0)  
    vectors = tsne.fit_transform(vectors)  
  
    x_vals = [v[0] for v in vectors]  
    y_vals = [v[1] for v in vectors]  
    return x_vals, y_vals, labels  
  
def plot_embeddings(x_vals, y_vals, labels):  
    import plotly.graph_objs as go  
    fig = go.Figure()  
    trace = go.Scatter(x=x_vals, y=y_vals, mode='markers', text=labels)  
    fig.add_trace(trace)  
    fig.update_layout(title="Word2Vec - Visualizzazione embedding con TSNE")  
    fig.show()  
    return fig  
  
x_vals, y_vals, labels = reduce_dimensions(model)  
  
plot = plot_embeddings(x_vals, y_vals, labels)
```

This visualization can be useful for noticing semantic and syntactic tendencies in your data.

For example, it's very useful for pointing out anomalies, such as groups of words that tend to clump together for some reason.

Parameters of Word2Vec

By checking on the Gensim website we see that there are many parameters that Word2Vec accepts. The most important ones are `vectors_size`, `min_count`, `window` and `sg`.

- **vectors_size** : defines the dimensions of our vector space.
- **min_count**: Words below the `min_count` frequency are removed from the vocabulary before training.
- **window**: maximum distance between the current and the expected word within a sentence.
- **sg**: defines the training algorithm. 0 = CBOW (continuous bag of words), 1 = Skip-Gram.

We won't go into detail on each of these. I suggest the interested reader to take a look at the [Gensim documentation](#).

Let's try to retrain our model with the following parameters

```
VECTOR_SIZE = 100
MIN_COUNT = 5
WINDOW = 3
SG = 1

new_model = Word2Vec(
    sentences=texts,
    vector_size=VECTOR_SIZE,
    min_count=MIN_COUNT,
    sg=SG
)

x_vals, y_vals, labels = reduce_dimensions(new_model)

plot = plot_embeddings(x_vals, y_vals, labels)
```




A new projection based on the new parameters for Word2Vec. Image by author.

The representation changes a lot. The number of vectors is the same as before (Word2Vec defaults to 100), while `min_count`, `window` and `sg` have been changed from their defaults.

I suggest to the reader to change these parameters in order to understand which representation is more suitable for his own case.

BONUS: Create an interactive report with Datapane

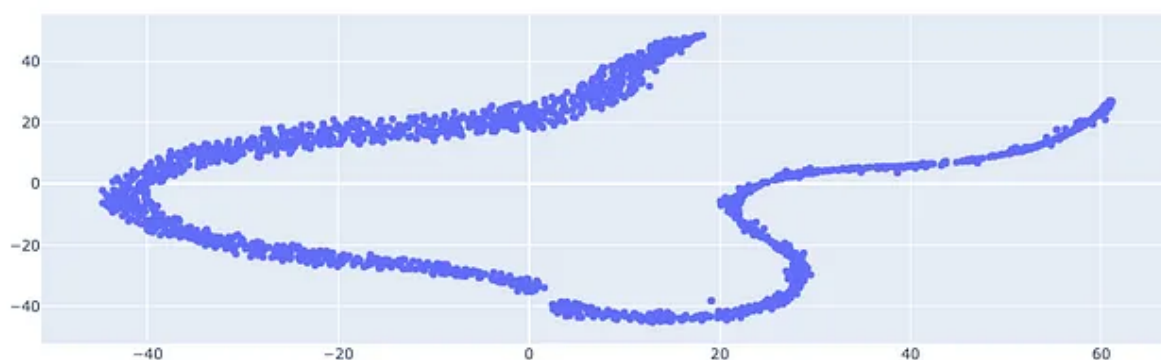
We have reached the end of the article. We conclude the project by creating an **interactive report in HTML with Datapane**, which will allow the user to **view the graph previously created with Plotly directly in the browser**.



Word2Vec - Visualizing embeddings with TSNE

Scatterplot

Word2Vec - Visualizing embeddings with TSNE



test created on 3/2/2023, 01:13:01



Creation of an interactive report with Datapane. Image by author.

This is the Python code

```
import datapane as dp

app = dp.App(
    dp.Text(text='# Visualizzazione degli embedding creati con Word2Vec'),
    dp.Divider(),
    dp.Text(text='## Grafico a dispersione'),
    dp.Group(
        dp.Plot(plot),
```

```
        columns=1,  
    ),  
)  
app.save(path="test.html")
```

Datapane is highly customizable. I advise the reader to study the documentation to integrate aesthetics and other features.

Conclusion

We have seen how to build embeddings from scratch using Gensim and Word2Vec. This is very simple to do if you have a structured dataset and if you know the Gensim API.

With embeddings we can really do many things, for example

- do **document clustering**, displaying these clusters in vector space
- **research similarities** between words
- use embeddings as **features in a machine learning model**
- lay the foundations for **machine translation**

and so on. If you are interested in a topic that extends the one covered here, leave a comment and let me know 👍

With this project you can enrich your portfolio of NLP templates and communicate to a stakeholder expertise in dealing with textual documents in the context of machine learning.

To the next article 🙌

If you want to support my content creation activity, feel free to follow my referral link below and join Medium's membership program. I will receive a portion of your investment and you'll be able to access Medium's plethora of articles on data science and more in a seamless way.

Join Medium with my referral link - Andrea D'Agostino

Read every story from Andrea D'Agostino (and thousands of other writers on Medium). Your membership fee directly...

medium.com

Word2vec

Gensim

Python

Machine Learning

NLP

More from the list: "NLP"

Curated by Himanshu Birla



Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

★ · 11 min read · Sep 4, 2021



Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

★ · 6 min read · Sep 3, 2021



Jon Gi... in

The Word2vec

★ · 15 min read

[View list](#)

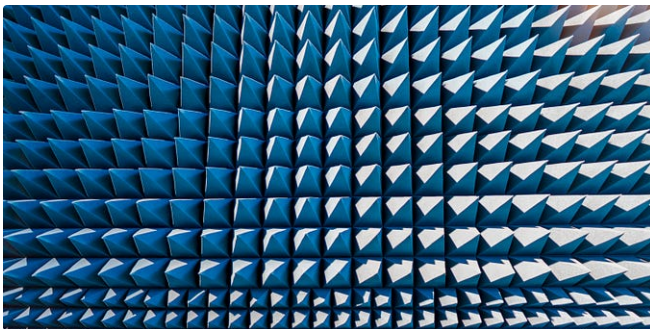
Written by Andrea D'Agostino

1K Followers · Writer for Towards Data Science

[Follow](#)

Data scientist. I write about data science, machine learning and analytics. I also write about career and productivity tips to help you thrive in the field.

More from Andrea D'Agostino and Towards Data Science



Andrea D'Agostino in Towards Data Science

Introduction to PCA in Python with Sklearn, Pandas, and Matplotlib



Antonis Makropoulos in Towards Data Science

How to Build a Multi-GPU System for Deep Learning in 2023

Learn the intuition behind PCA in Python and Sklearn by transforming a multidimensional...

★ · 13 min read · Sep 7



92



This story provides a guide on how to build a multi-GPU system for deep learning and...

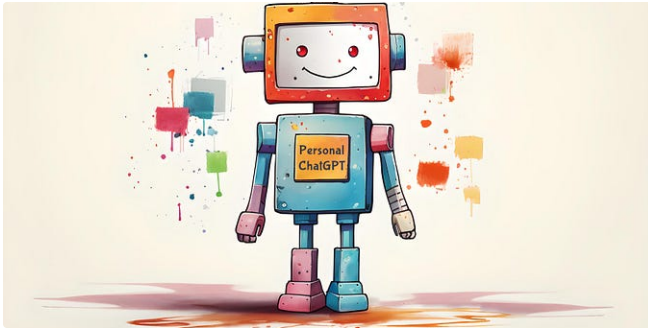
10 min read · Sep 17



549



11



Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

★ · 15 min read · Sep 8



595



7



Andrea D'Agostino in Towards Data Science

Feature Selection with Boruta in Python

Learn how the Boruta algorithm works for feature selection. Explanation + template

6 min read · Nov 30, 2021



100




See all from Andrea D'Agostino

See all from Towards Data Science

Recommended from Medium



 Rayan Imran

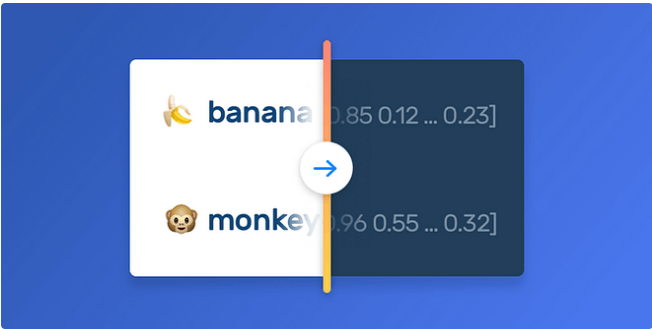
Comparing Text Preprocessing Techniques: One-Hot Encoding,...

Introduction

6 min read · Jun 12

 2 



 mayank khulbe in The Good Food Economy

Skipgram implementation from scratch—Pytorch

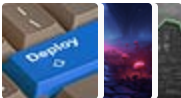
In recent times, there has been an exponential increase in the use-cases pertaining to...

12 min read · May 23

 24 

Lists



Predictive Modeling w/ Python

20 stories · 452 saves



Natural Language Processing

669 stories · 283 saves



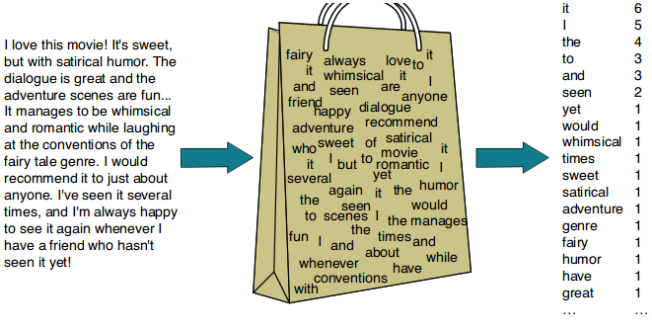
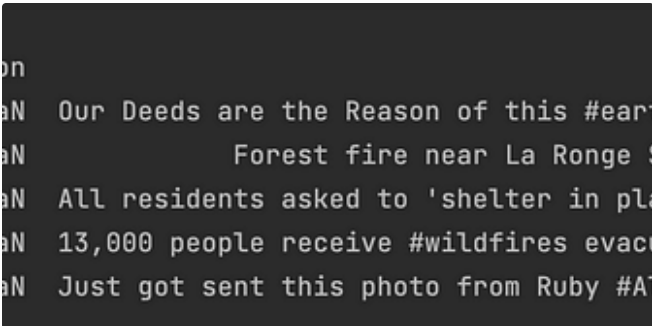
Practical Guides to Machine Learning

10 stories · 519 saves



Coding & Development

11 stories · 200 saves





Hatice Şeyma Koç

Fasttext & Doc2Vec for Text Classification

Hello everyone,

5 min read · Jul 7



11



...



Vamshi Prakash

An Introduction to Bag of Words (BoW)

Topic :-An Introduction to Bag of Words (BoW)

10 min read · Jun 27



2



...

	gender	wealth	power	weight	speak
King	1.0	1.0	1.0	0.8	1.0
Queen	0.0	1.0	0.7	0.4	1.0
Man	1.0	0.3	0.2	0.6	1.0
Woman	0.0	0.3	0.2	0.5	1.0
Monkey	1.0	0.0	0.0	0.3	0.0



Md. Ishtiuk Ahammed

Word2Vec

Word2Vec text vectorization technique explanation.

2 min read · May 20



26



...



Susovan Dey

From Traditional to Modern: A Comprehensive Guide to Text...

Natural Language Processing (NLP) is a rapidly growing field that focuses on enablin...

6 min read · Apr 27



3



...

See more recommendations