Published in Towards Data Science

Elise Landman Follow
Jun 4 · 8 min read ★ · Listen

Session-Based Recommender Systems with Word2Vec

How to use Python to train a basic recommender system with Word2Vec on browser session data.

If you are familiar with the **Word2Vec** algorithm, then it is probably because you have stumbled upon it somewhere in the field of **Natural Language Processing** (NLP). Since its initial publishing in 2013, the paper by T. Mikolov et al. gained increasing popularity and demonstrated state-of-the-art results in various NLP applications. There have been various newer and better-performing models published since then — but still, when working with NLP or in the field of Information Retrieval (IR), there is no way around Word2Vec.

Apart from its classical application in NLP, we can also use the underlying Word2Vec algorithm for other applications, like for certain types of Recommendation System problems.

In this article, we will be looking at how we can make use of **Word2Vec's CBOW algorithm** to build a simple **Recommendation System for session-based data**.

Firstly, we will discuss what could be the motivation behind using the Word2Vec algorithm for such type of problem, and **secondly**, we will be looking at how we can implement this in practice using Python 3.

1 | Why use Word2Vec?

For this article, I assume that you are fairly familiar with how the Word2Vec model and its underlying architectures (CBOW and SkipGram) work. If you are not familiar with those, I can highly recommend checking out this article which explains the concepts very well.

The specific type of Recommendation System we will be focusing on in this article are so-called *session-based*. In these cases we want to focus on giving next-click recommendations to anonymous users, based on their current browser session. **What does that mean?**

Let's consider for example an e-commerce website selling various products from different categories. One day, a user might come to our website and browse through our product catalogue. Our goal obviously would be to have the user adding a product to its cart, and finally purchasing it. With the help of a Recommendation Systems we aim to recommend products that are "relevant" to our user. This enhances the shopping experience for the user, but could also increase the likelihood of a purchase.

Open in app

Get started

Open in app

Get started

Open in app

Get started

Open in app

Get started

Open in app

Get started

Open in app

Get started

them, for example about their previous purchases or about their previous browser sessions. This makes giving "relevant recommendations" increasingly complex.

This is where Word2Vec comes into play.

To summarize: Word2Vec is a model that, when well trained, is capable of capturing the meaning of words, based on their context.

Now considering our above example, we might have the following historical session data of an anonymous user:

```
session_1 = [1, 2, 5]
```

In this current session, a user viewed the products number 1, 2 and 5. If we have enough of this type of session data, we can use these as input vectors in a Word2Vec model, specifically using the CBOW algorithm. With this, our model will be able to give "meaning" to our items, and so determine in which context (i. e. session) they will most likely fit in.

> For our use-case we can translate "words" to "products", and "context" to "session".

You could argue that, instead of using Word2Vec, we could also just recommend products

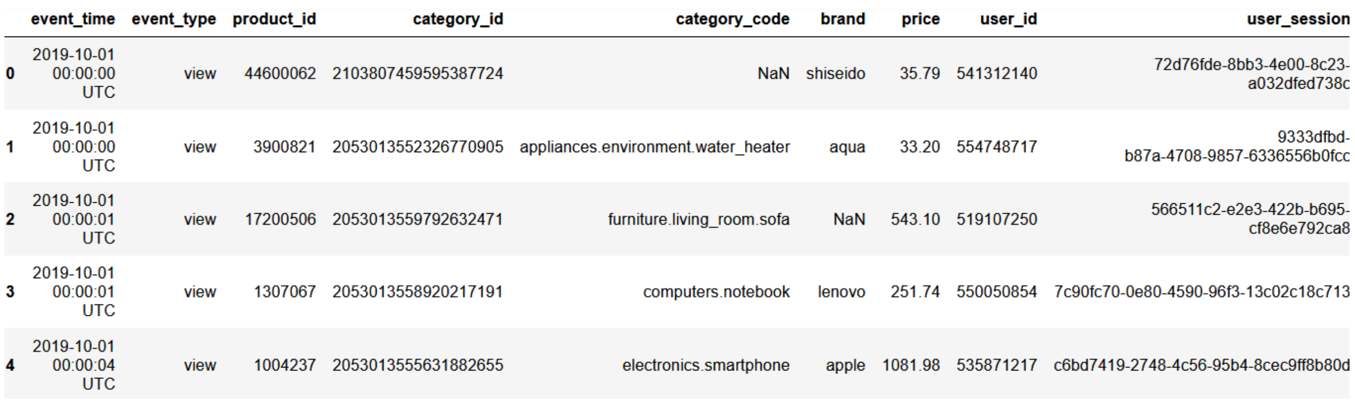
Now that we went through the theory and understand what Word2Vec can do for us in this use-case, let's actually start implementing our idea and see what the results look like.

2 | Implementation in Python

For our use-case we will be using a Kaggle dataset that contains e-commerce data from a multi-category store (source: Kaggle.com and REES46 Marketing Platform). The datasets are fairly big in size, therefore, for the purpose of simplification and demonstration I will be using a subset of the original dataset from October 2019.

Data Exploration

Before we jump into any processing steps, let's first have a look at our data:



	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	2019-10-01 00:00:00 UTC	view	44600062	2103807459595387724	NaN	shiseido	35.79	541312140	72d76fde-8bb3-4e00-8c23-a032dfed738c
1	2019-10-01 00:00:00 UTC	view	3900821	2053013552326770905	appliances.environment.water_heater	aqua	33.20	554748717	9333dfbd-b87a-4708-9857-6336556b0fcc
2	2019-10-01 00:00:01 UTC	view	17200506	2053013559792632471	furniture.living_room.sofa	NaN	543.10	519107250	566511c2-e2e3-422b-b695-cf8e6e792ca8
3	2019-10-01 00:00:01 UTC	view	1307067	2053013558920217191	computers.notebook	lenovo	251.74	550050854	7c90fc70-0e80-4590-96f3-13c02c18c713
4	2019-10-01 00:00:04 UTC	view	1004237	2053013555631882655	electronics.smartphone	apple	1081.98	535871217	c6bd7419-2748-4c56-95b4-8cec9ff8b80d

E-Commerce Data from October 2019 (Dataset Source: Kaggle.com and REES46 Marketing). Image by Author.

We observe various features, whereas only a few will be interesting for our use case: *event_type*, *product_id*, *category_code* and the *user_session*. In this dataset we also have data about our users, since we have the *user_id* values. Nevertheless, we won't be using those since the purpose of this example is to show how we can achieve recommendations solemnly based on **anonymous** user session data.

The *event_type* variable describes the type of event that happened within a session. This can be either "view", "cart" or "purchased". We will be using the vectors of each session containing the *product_id*'s as input into our Word2Vec model. Later on, with the data from our *category_code* variable, we will be able to test the prediction results of our model.

Firstly, our data might contain missing values which we need to remove. Secondly, we need to keep only specific columns from our data, since we do not need all of them for our use case. These two steps are done in the code below:

```python
import pandas as pd
data = pd.read_csv("2019-Oct.csv")

# drop NaN values in specific columns
data = data.dropna(subset=["category_code", "brand", "user_session", "product_id"])

# keep only relevant columns in our dataset
data = data[["event_type", "product_id", "category_code", "brand", "user_session"]]
```

preprocess_data.py hosted with ❤ by **GitHub** **view raw**

As mentioned above, for the purpose of demonstration I will be using only a subset of the original data. As shown in the code below, to create a subset of our data, we will first sort the data by *user_session*. This is to make sure that we keep all data from each session grouped together. Then we select index 2'000'000 as our splitting point.

```python
# select where to split the data
split_at = 2000000

#  make sure the split doesn't cut off session data
while data_clean["user_session"].iloc[split_at-1] == data_clean["user_session"].iloc[sp
    split_at += 1

# perform the split
split_range = list(range(0, split_at))
data_clean_subset = data_clean.iloc[split_range]
```

create_data_subset.py hosted with ❤ by **GitHub** **view raw**

To make sure that our data is not split in the middle of a session, we increment our splitting index *split_at* by 1 as long as the next row in our data is still part of the same previous session. Finally, we perform the actual split ranging from the first row to the split index we

Building the W2V Recommender System

Now that our data is cleaned and processed, we can finally proceed with the building of our actual Word2Vec-based Recommender System. We will be using the **Gensim** library, which has a practical implementation of the Word2Vec algorithms, as published in the original paper.

In order to train our own Word2Vec model with Gensim, we first need to bring our data into the correct shape so that the library can actually train on it. It takes a list of vectors as input, therefore we will transform our data with the following *create_w2v_data* helper function:

```python
# create a dataset that w2v can train on
def create_w2v_data(df):
    sessions, session = [], []
    for index, value in df.iterrows():
        if index != 0:
            if str(value["user_session"]) == str(df.at[index-1, "user_session"]):
                session.append(str(value["product_id"]))
            else:
                if len(session) != 0:
                    sessions.append(session)
                session = [str(value["product_id"])]
        else:
            session.append(str(value["product_id"]))
    return sessions
```

create_w2v_data.py hosted with ❤ by **GitHub** **view raw**

This function takes a Pandas dataframe as input and outputs **a list of each session** in the dataset, which in turn **is a list of all the products that were viewed, put to cart and/or purchased within this session**. It might take a few minutes to run, but then the resulting data looks as following:

```
[['2501061'],
 ['2701673', '2701773'],
```

Since we combine all three event types (view, cart, purchased) in our data, a purchased product will appear three times in one and the same session. This is good: it gives **extra relevancy** to products that were actually purchased, since they will generally appear more often in the dataset.

The average length of our sessions is 4.14 , and the longest session consists of 324 products viewed, put to cart and/or purchased.

Training the Model

Finally we can start the actual training of our model! 🎉

With the help of the Gensim library this can be done in a few lines of code:

```python
from gensim.models import Word2Vec

# training the W2V model
model = Word2Vec(sentences=data_w2v,
                 vector_size=100,
                 window=longest_session,
                 min_count=3,
                 workers=4)
```

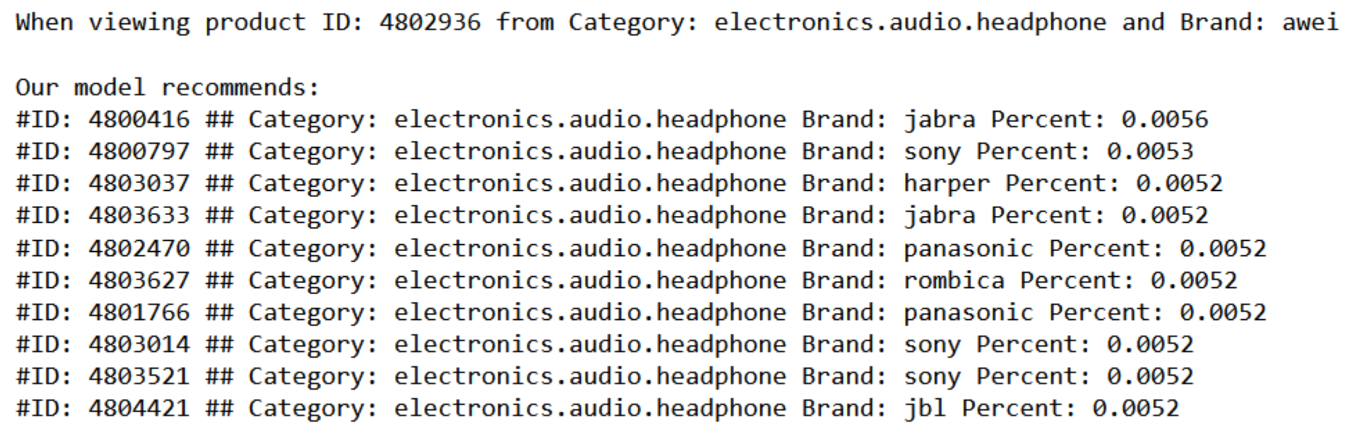
train_w2v.py hosted with ❤️ by **GitHub** **view raw**

We set up a model with our prepared list of sessions as input data, and set the model parameters. We set *min_count* to 3, as this will make sure that our model is only trained on products that appear at least 3 times in the whole dataset — in other words: the product must have been viewed at least 3 times, or viewed twice and put to cart once, or purchased at least once. We also use a *window_size* corresponding to the longest session in our dataset. Why? So that we consider all of the products that were viewed in a session. We leave the rest of the parameters as suggested in the Gensim library.

Hurray, we have trained our model! Next up: let's see what it can do for us.

Testing

Let's look at the resulting recommendations that our model gives us. Let's say a user views a specific type of headphones, saved under *product_id* 4802936. The products our model recommends can be retrieved with the Gensim function *predict_output_word*. The result for the top 10 recommendations is shown below:

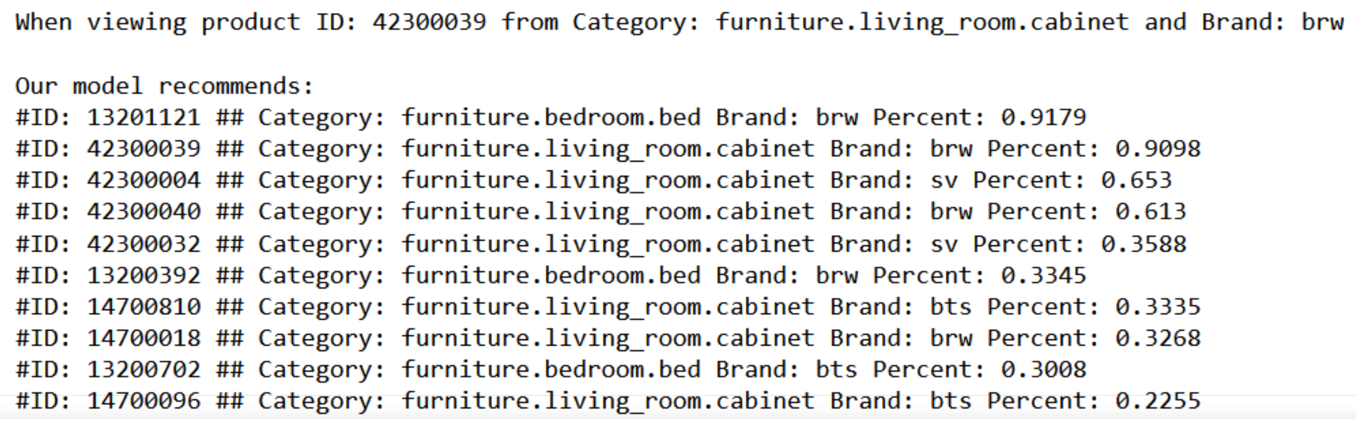

```
When viewing product ID: 4802936 from Category: electronics.audio.headphone and Brand: awei

Our model recommends:
#ID: 4800416 ## Category: electronics.audio.headphone Brand: jabra Percent: 0.0056
#ID: 4800797 ## Category: electronics.audio.headphone Brand: sony Percent: 0.0053
#ID: 4803037 ## Category: electronics.audio.headphone Brand: harper Percent: 0.0052
#ID: 4803633 ## Category: electronics.audio.headphone Brand: jabra Percent: 0.0052
#ID: 4802470 ## Category: electronics.audio.headphone Brand: panasonic Percent: 0.0052
#ID: 4803627 ## Category: electronics.audio.headphone Brand: rombica Percent: 0.0052
#ID: 4801766 ## Category: electronics.audio.headphone Brand: panasonic Percent: 0.0052
#ID: 4803014 ## Category: electronics.audio.headphone Brand: sony Percent: 0.0052
#ID: 4803521 ## Category: electronics.audio.headphone Brand: sony Percent: 0.0052
#ID: 4804421 ## Category: electronics.audio.headphone Brand: jbl Percent: 0.0052
```

Recommendations for Product ID 4802936 (Audio Headphones). Image by Author.

Indeed, we can see that our model managed to correctly understand that these products belong to a similar category, and that there are similarities between these *product_id*'s. Therefore, if a user would view the headphones with id 4802936, the model would recommend other alternative headphones.

We can do the same for product_id 42300039 which is a "cabinet" belonging to the "living room" category:

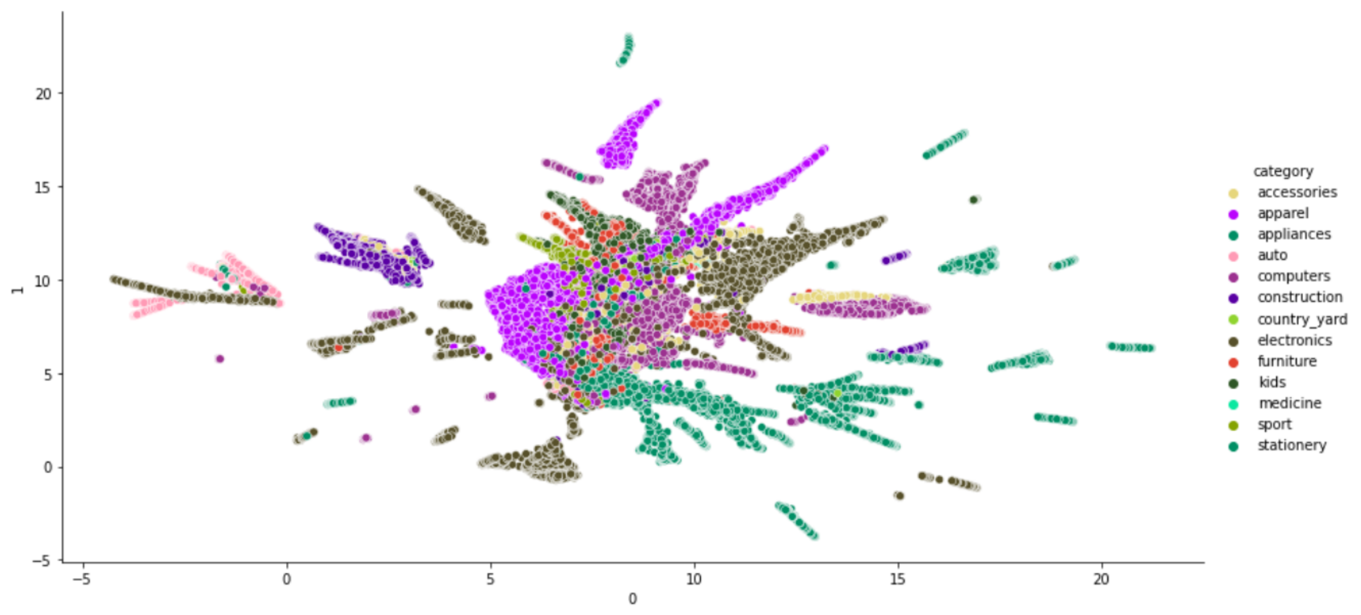

```
When viewing product ID: 42300039 from Category: furniture.living_room.cabinet and Brand: brw

Our model recommends:
#ID: 13201121 ## Category: furniture.bedroom.bed Brand: brw Percent: 0.9179
#ID: 42300039 ## Category: furniture.living_room.cabinet Brand: brw Percent: 0.9098
#ID: 42300004 ## Category: furniture.living_room.cabinet Brand: sv Percent: 0.653
#ID: 42300040 ## Category: furniture.living_room.cabinet Brand: brw Percent: 0.613
#ID: 42300032 ## Category: furniture.living_room.cabinet Brand: sv Percent: 0.3588
#ID: 13200392 ## Category: furniture.bedroom.bed Brand: brw Percent: 0.3345
#ID: 14700810 ## Category: furniture.living_room.cabinet Brand: bts Percent: 0.3335
#ID: 14700018 ## Category: furniture.living_room.cabinet Brand: brw Percent: 0.3268
#ID: 13200702 ## Category: furniture.bedroom.bed Brand: bts Percent: 0.3008
#ID: 14700096 ## Category: furniture.living_room.cabinet Brand: bts Percent: 0.2255
```

Visualizing

We can also have a look at how our data looks by plotting it, as this might give us a better overall understanding of the model. We will reduce the dimensions of our the embedding vectors in our model to 2, which will allow us to plot the data nicely. To do this, we can make use of the *umap* dimension reduction tool from the **umap-learn** library.

The plot will contain the 2-dimensional product embedding vectors which will be colored by their respective categories. For the sake of simplicity, we will only group them by their main category, i. e. the first category that is listed in the category name. The resulting plot is shown below:



Plot of Products by their Categories as captured by the W2C Recommendation System model. Image by Author.

We can see in the above plot very nice results! It shows that indeed, our model is able to group the products fairly well by its categories.

4 | Summary and Conclusion

To summarize, we have demonstrated in this article that it is possible to quickly and easily build a fairly good Recommendation System for anonymous session-based data, with the help of the Word2Vec CBOW algorithm.

recommendations looks like.

I hope this has been informative for you and that you learned something new. Feel free to comment any feedback or questions you might have! 🎉

References:

[1] T. Mikolov et al., Efficient Estimation of Word Representations in Vector Space (2013)

[2] R. Rehurek, Gensim models.word2vec (2022)

[3] T. Mikolov et al., Distributed Representations of Words and Phrases and their Compositionality (2013)