The cosine similarity and its use in recommendation systems

Naomy Duarte Gomes · Follow

5 min read · Jan 31, 2023

Listen Share More

Cosine similarity is a metric based on the cosine distance between two objects and can be used in recommendation systems such as movie and book recommenders. In this article, we will learn what it is and how it can be used to make recommendations by identifying similar items.

Introduction

Recommendation systems are part of our everyday life. Just to cite some common examples, we get recommendations when we shop online on Amazon and when we open Youtube or Netflix. The purpose of these systems is to recommend what you may want to buy based on what you have in your bag or what video or movie to watch next based on the ones you already watched.

Fig. 1 — Similarities. Image from Shutterstock.

A recommendation system can be as complicated as we want, such as the ones that use deep learning [1,2], but they can also be simple and based on the similarity between items. And one way to calculate the similarity between items is with the use of **cosine similarity**.

Frequently bought together
The StatQuest Illustrated Guide To Machine Learning
TRIPLE BAM!!!
+
ACE THE DATA SCIENCE INTERVIEW
+
Designing Machine Learning Systems

Fig. 2— The “frequently bought together” recommendations at Amazon.

The cosine between two vectors

To understand this metric of similarity, we first need to understand some concepts. Suppose we have a table of books 1 and 2 containing their genre, according to Fig. 3. To each word in the genre table, we create another column in a second table, where if the word is in the genre, we give it 1, if it's not, then is 0. Since we have the genres Science Fiction and Fiction, we create another table with these two words. If we draw a graph where the x-axis is the Science axis and the Y-axis is the Fiction axis, we can associate a point to each book. For example, book 1 will be the blue point with a Sience-axis of 1 and a Fiction-axis of 1 (Science Fiction). Book 2 will be the yellow point with a Sience-axis of 0 and a Fiction-axis of 1 (Fiction). We draw a vector from the origin to the points, which we call the book-vector.

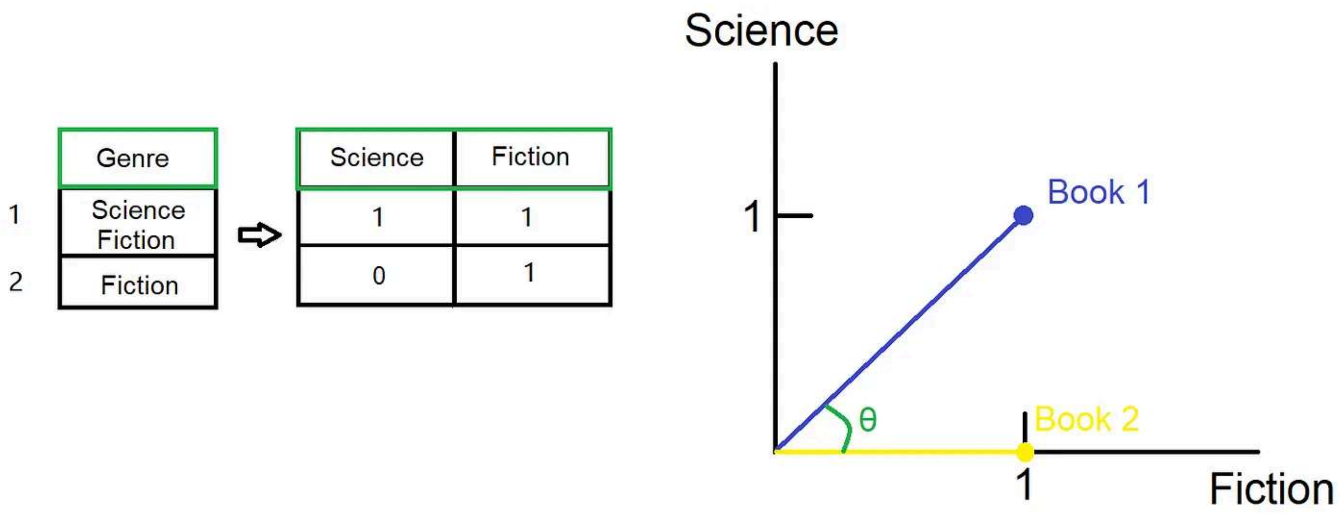


Fig. 3— Book table containing genres and a graph showing the book-vectors.

Now, we can see that the book-vectors form an angle θ with each other. The cosine of this angle is our measure of similarity, and it is given by:

$$\cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

where **A** and **B** are the vectors we are considering, ||**A**|| and ||**B**|| are their norm (length). The **Ai** and **Bi** in the formula are the components of each vector. Book-vector 1 is (1,1), and book-vector 2 is (0,1). Let's calculate the cosine similarity:

$$\cos\theta = \frac{(1\times0)+(1\times1)}{\sqrt{1^2+1^2}\sqrt{0^2+1^2}} = \frac{1}{\sqrt{2}\sqrt{1}} = 0.71,$$

which says two things: first, these vectors have some similarity, and second, θ is 45^o, something that we already expected and could calculate by using the Pythagorean theorem and calculating the cosine using the sides of the triangle.

If both books were Science Fiction, we would have the same book-vectors (1,1) and the cosine would be 1, meaning they are the same. But if book 1 were Science Fiction (1,1) and book 2 Terror (0,0), in this case, they would have nothing in common, and the cosine would be 0. Therefore, *high similarity means a cosine close to 1, and low similarity, a cosine close to 0*.

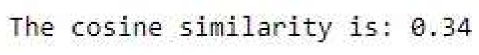
Calculating using Python

We can cite at least two ways to calculate the cosine similarity between two given vectors. One is using numpy:

```
import numpy as np
from numpy.linalg import norm

A = np.array([1,8])
B = np.array([9,2])

cos_sim = np.dot(A,B)/(norm(A)*norm(B))
print(f"The cosine similarity is: {round(cos_sim,2)}")
```



```
The cosine similarity is: 0.34
```

Another way is using sklearn:

```
from sklearn.metrics.pairwise import cosine_similarity

cos_sim = cosine_similarity(A.reshape(1, -1),B.reshape(1, -1))
print(f"The cosine similarity is: {cos_sim}")
```

Either way, calculating the cosine similarity is straightforward and, if done by hand, only requires a simple mathematical formula.

Application as a book recommender

In this repository, I utilize the cosine similarity to build a book recommender.

GitHub - naomyduarteg/LIA-FastAPI-SQL: LIA is a Library In API created with Python, FastAPI and…

LIA is a Library In API created with Python, FastAPI and SQLite to save information about the books you read and get…

github.com

For this project, the main idea is that a user will enter the books she/he read and classify them from 1 to 5, with 5 being the highest grade, thus creating a table that contains the columns: book title, author, description, genre, pages, and classification. Once the table is created, we convert it to pandas dataframe, then normalize using the function *normalize(data)* and one-hot encode the categorical columns using the function *ohe(df, enc_col)* (which is basically what we did when we create the other table in Fig. 2). It is important to normalize numerical columns since big numbers can bias the results.

```
#importing all the libraries needed
import pandas as pd
import numpy as np
from numpy import dot
from numpy.linalg import norm

#building the main function to recommend books based on a choosen book_id and owner_id
def f_recommend(book_id, owner_id):
    '''
    Gets the csv file, converts to pandas dataframe, normalizes and one hot encodes columns,
    adjusts dataframe to recommend books using cosine similarity as metric, then returns dict o
    '''
    df = pd.read_csv(r'd:\naomy\LIA-FastAPI-MySQL\data\data.csv') #reading the csv file contain
    df_copy = df.copy()
    df.rename(columns={'id':'book_id'},inplace=True)
    #normalizing pages and book rating columns
    df['pages_norm'] = normalize(df['pages'].values)
    df['book_rating_norm'] = normalize(df['classification'].values)
    #one hot encodes the categorial columns
    df = ohe(df = df, enc_col = 'genre')
    df = ohe(df = df, enc_col = 'author')
    cols = ['pages', 'genre', 'description', 'title', 'author','classification']
    df.drop(columns = cols, inplace = True)
    df.set_index('book_id', inplace = True)

    df_recommendations = recommend(df.index[book_id], owner_id, df)
    df_seila= df_copy.loc[df_copy['id'].isin(df_recommendations.index.values), ['title']]#retur
    df_seila.drop_duplicates(inplace=True)
    list_books = df_seila.to_dict(orient='list')
    return list_books

def normalize(data):
    '''
    Gets dataframe and normalizes input data to be between 0 and 1
    '''
    min_val = min(data)
    if min_val < 0:
        data = [x + abs(min_val) for x in data]
    max_val = max(data)
    return [x/max_val for x in data]

def ohe(df, enc_col):
    '''
    One hot encodes specified columns and adds them back
    onto the input dataframe
    '''

    ohe_df = pd.get_dummies(df[enc_col])
    ohe_df = pd.get_dummies(df[ohe_col])
    ohe_df.reset_index(drop = True, inplace = True)
    return pd.concat([df, ohe_df], axis = 1)

def cosine_sim(v1,v2):
    '''
    Calculates the cosine similarity between two vectors
    '''
    return dot(v1,v2)/(norm(v1)*norm(v2))

def recommend(book_id, owner_id, df):
    """
    Content based recommendations.
    Calls the cosine similarity function to calculate similarities and returns the
    most similar books, excluding the ones listed by the user
    """
    book_id = book_id - 1 #because it starts from zero in the csv, while the ids in the databas

    # calculate similarity of input book_id vector and all other vectors in the table
    inputVec = df.loc[book_id].values #gets values of the book_id inputed
    df['sim']= df.apply(lambda x: cosine_sim(inputVec,x.values), axis=1)  #goes through all the
                                                         #and creates a column containing the v

    df_rec = df.nlargest(10, columns='sim')#gets only the 10 most similars (10 bigger values of
    df_final = df_rec.loc[df['owner_id'] != owner_id]#excludes books already listed by the user
    # returns up to 10 similar books that the user didnt list already
    return df_final
```

The function *recommend(book_id, owner_id, df)* will calculate the cosine similarity between the book inputted by the user and all the other books in the table, creating a column with these values and then selecting the 10 most similar books to recommend to the user. The cosine similarity calculation itself is pretty straightforward and is defined by the function *cosine_sim(v1, v2), s*imilar to what we did with numpy, the only difference here is that the vectors have more than 2 dimensions.

You can run the API and test the recommendations.

Conclusions

The cosine similarity gives a useful measure of how similar two objects are. It is a rather simple mathematical concept and easy to implement computationally. It can be used for many purposes: in machine learning as a distance metric, with textual data to compare two documents, and in recommendation systems.

References

[1] Da'u, A., Salim, N. Recommendation system based on deep learning methods: a systematic review and new directions. *Artif Intell Rev* **53**, 2709–2748 (2020). https://doi.org/10.1007/s10462-019-09744-1

[2] Schedl, M. (2019). Deep Learning in Music Recommendation Systems. *Frontiers in Applied Mathematics and Statistics*, *5*. https://doi.org/10.3389/fams.2019.00044

Cosine Similarity Machine Learning Recommendation System Python Mathematics

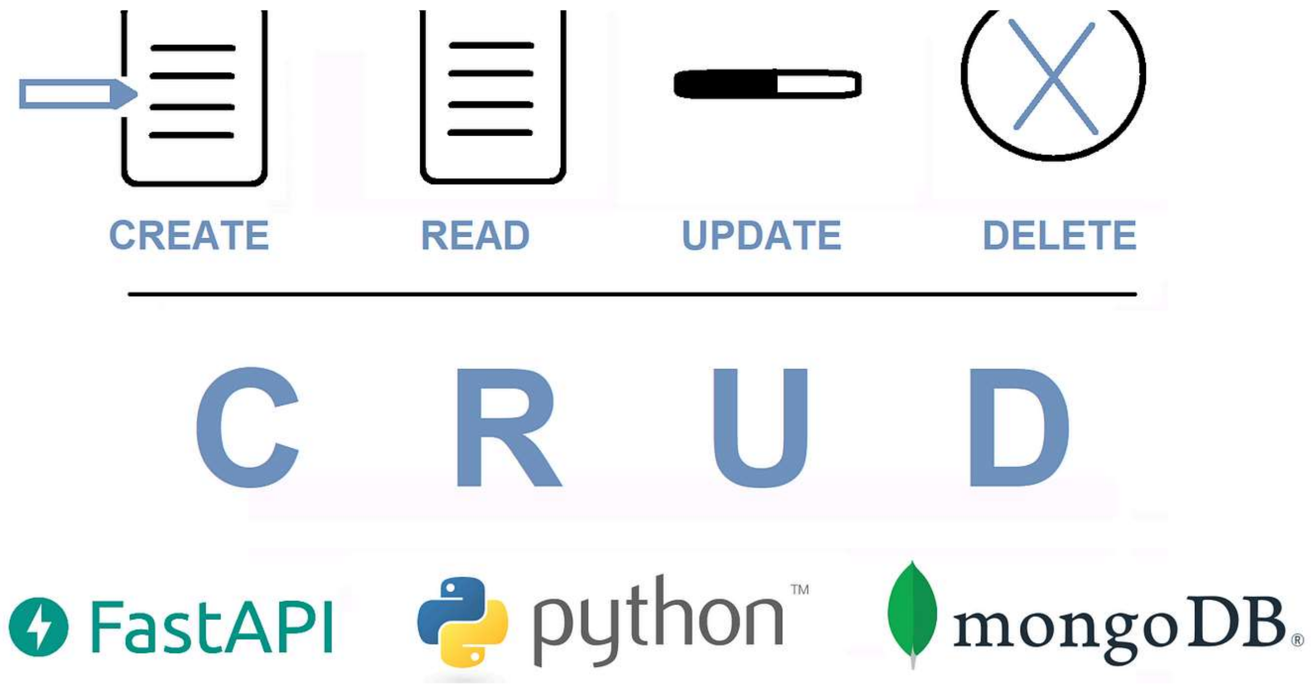
Follow

Written by Naomy Duarte Gomes

67 Followers

PhD in Physics, MBA in Data Science and Data Scientist writing about data.

More from Naomy Duarte Gomes



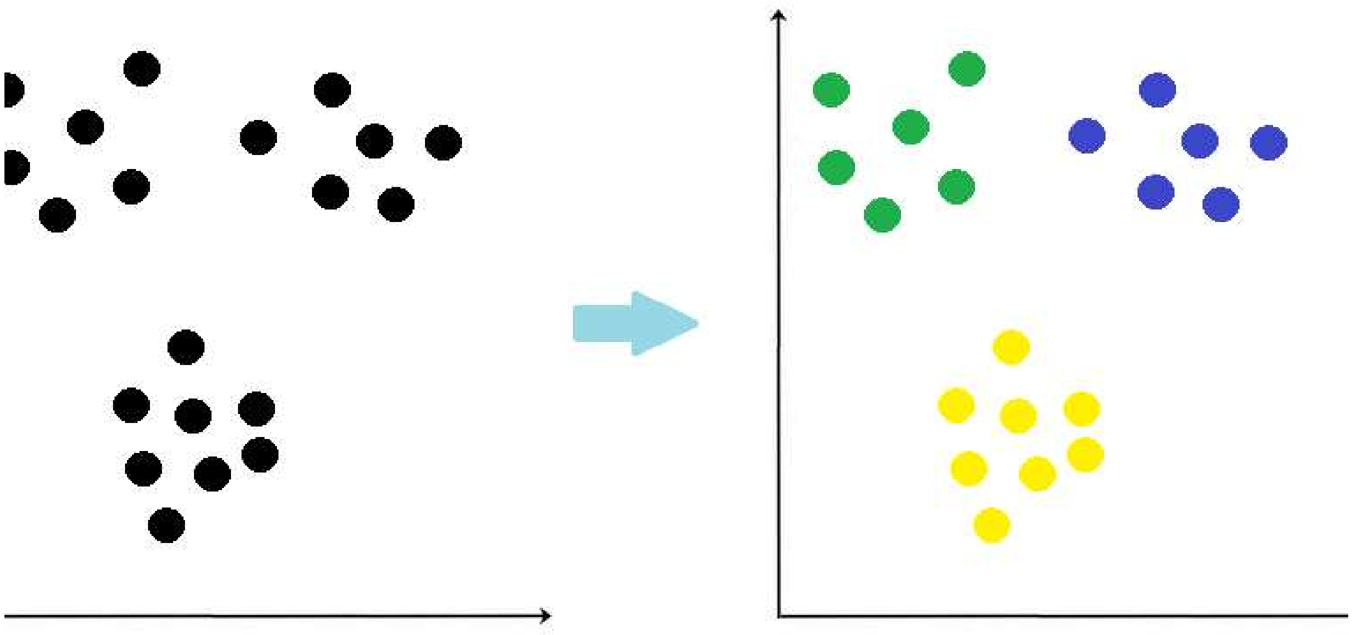
Naomy Duarte Gomes

CRUD with Python, FastAPI, and MongoDB

CRUD (Create, Read, Update, Delete) operations are the basic operations a software application must perform. In this tutorial, we will...

9 min read · Oct 28, 2022

62 2



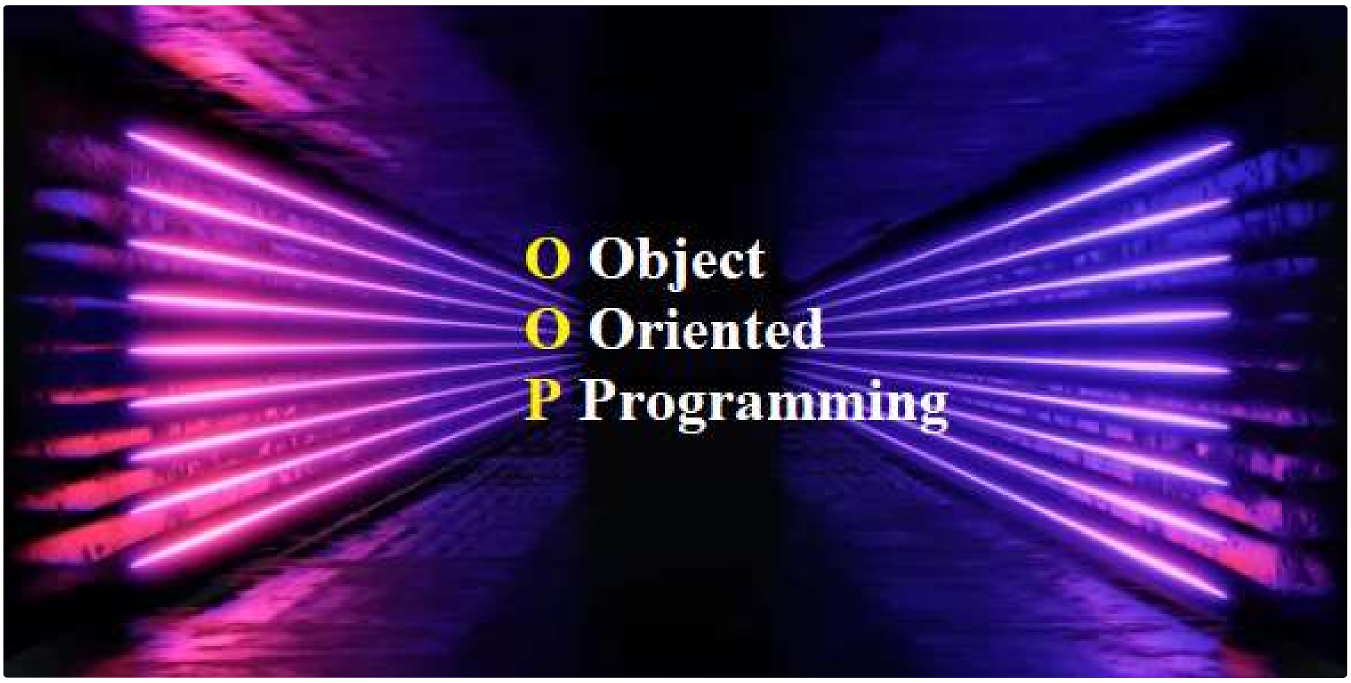
Naomy Duarte Gomes

K-means clustering explained with Python

K-means clustering is an unsupervised machine learning algorithm to analyze the formation of clusters in the data, which means finding...

7 min read · Jun 10, 2022

111



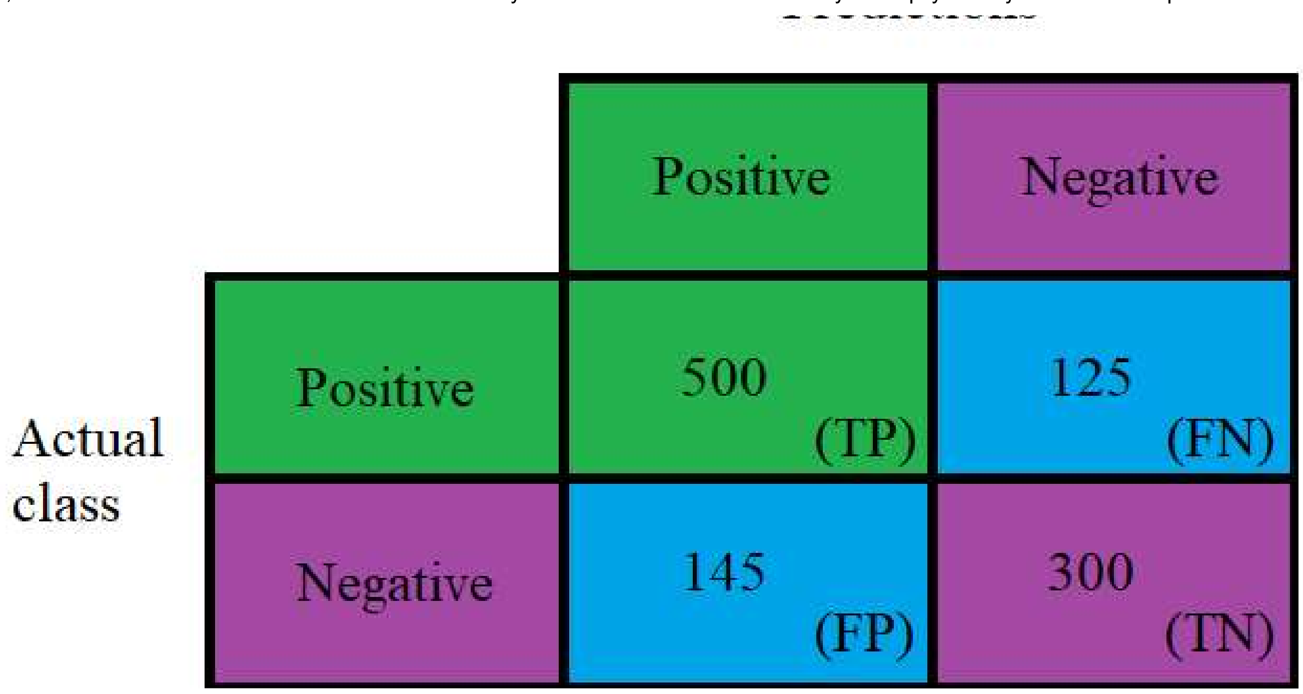
Naomy Duarte Gomes

Object-oriented programming with Python: core principles and examples

Object-oriented programming (OOP) is a very efficient approach to writing software. Its main difference concerning procedural programming...

5 min read · Sep 11, 2022

239



		Positive	Negative
Actual class	Positive	500 (TP)	125 (FN)
	Negative	145 (FP)	300 (TN)

Naomy Duarte Gomes

Metrics to evaluate classification models

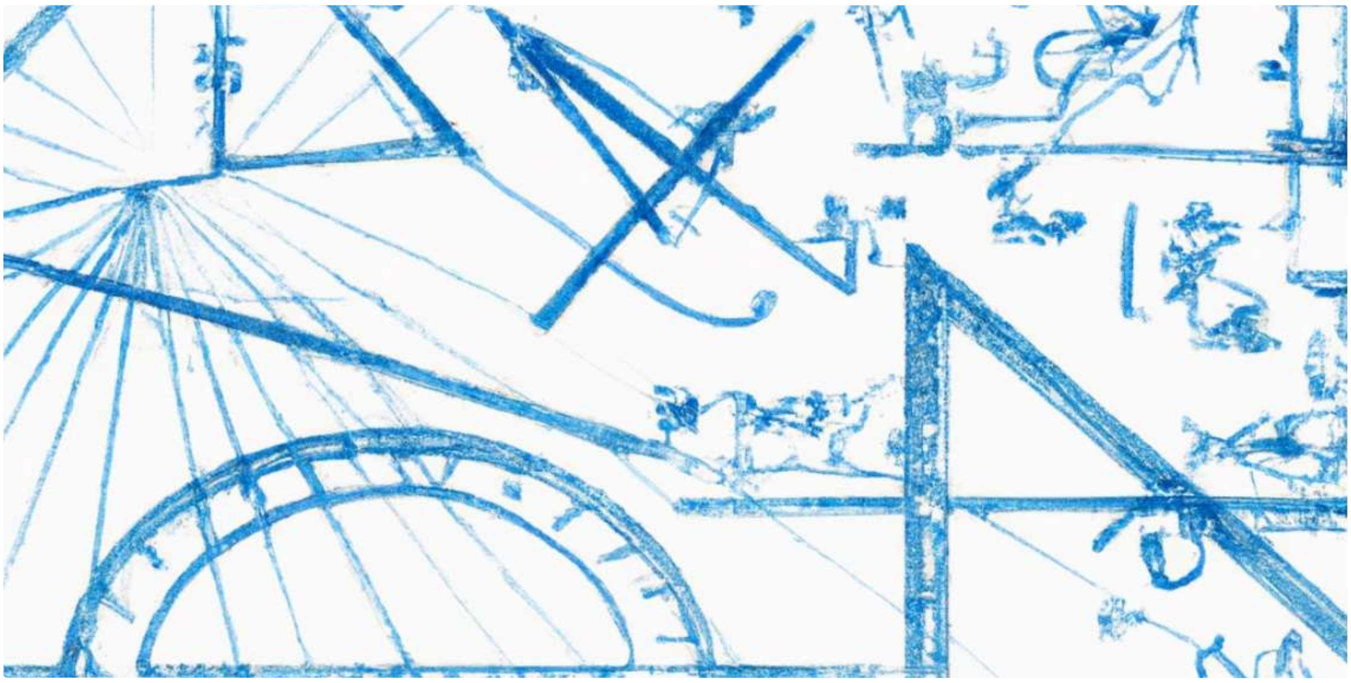
Machine learning classification models assign each point in a dataset to a class. Once this categorization is performed, how can we...

7 min read · Jul 1, 2022

132 1

See all from Naomy Duarte Gomes

Recommended from Medium



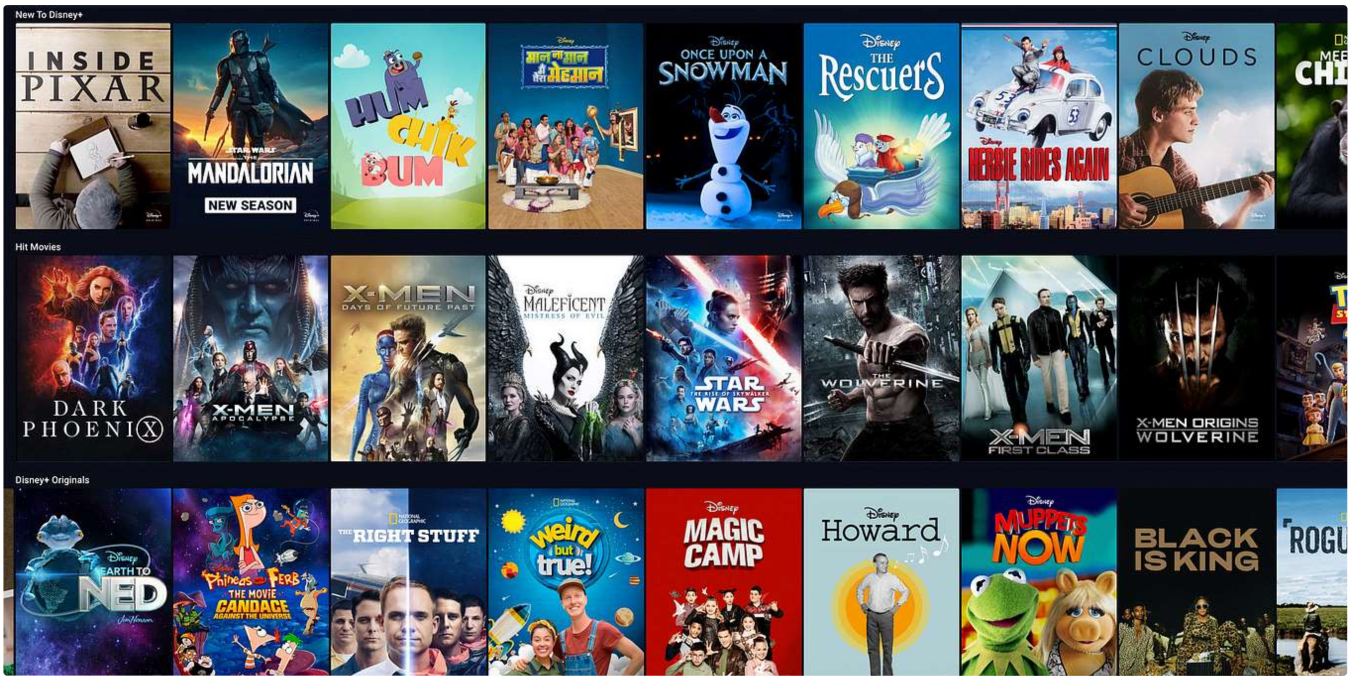
DataStax

How to Implement Cosine Similarity in Python

By Phil Miesle

4 min read · Nov 30, 2023

20



Jishnu mohan

Movie Recommendation System using Cosine Similarity

Imagine having a movie buddy who knows your taste in films better than you do—always ready with the perfect suggestion for your movie...

5 min read · Jan 30, 2024

50

Lists

Predictive Modeling w/ Python

20 stories · 1181 saves

Practical Guides to Machine Learning

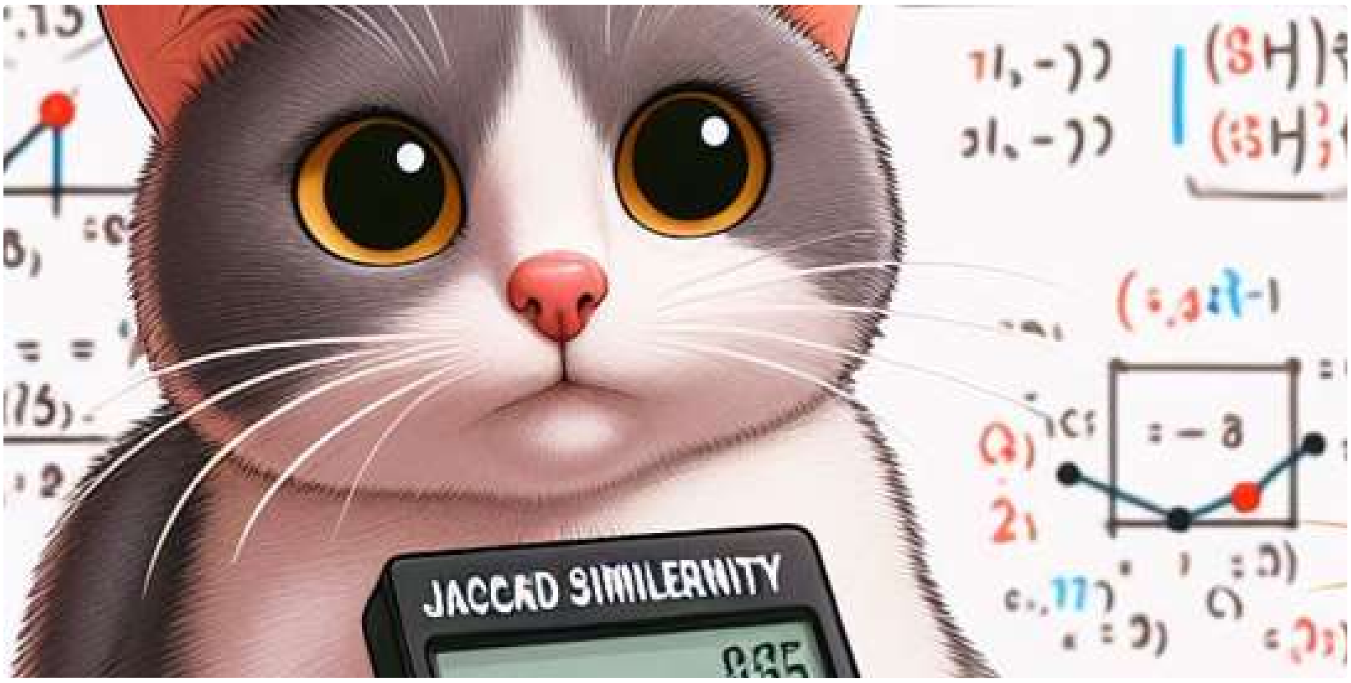
10 stories · 1426 saves

Coding & Development

11 stories · 606 saves

Natural Language Processing

1444 stories · 946 saves



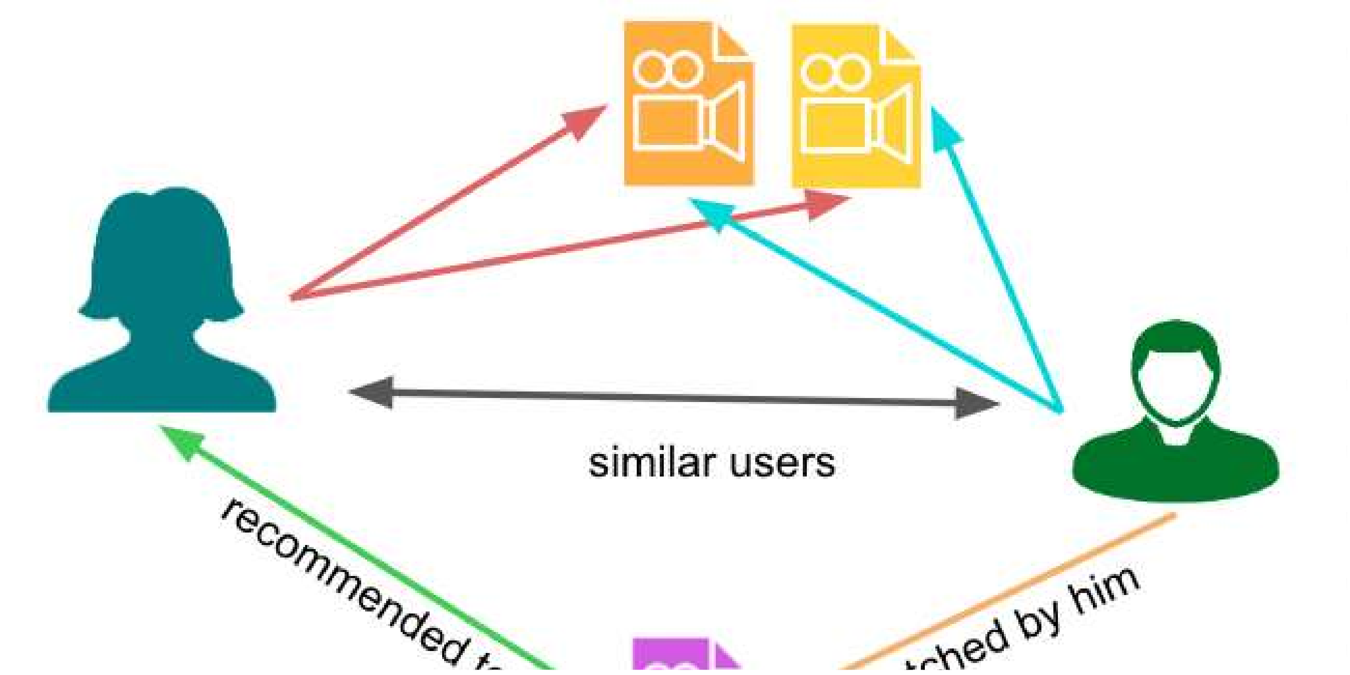
Alexander Shmyga

Vectors similarity. Jaccard Similarity.

In one of my previous posts, I described the essence of vector similarity search and there I explained the two most popular vector...

4 min read · Dec 19, 2023

6



Ashmi Banerjee

Pros and Cons of Collaborative Filtering

In this blog, we will learn the advantages and disadvantages of using collaborative filtering.

4 min read · Mar 19, 2024

Python Programming

Recommender Systems with Python Code Examples

A Comprehensive Guide to Recommender Systems

· 6 min read · Dec 16, 2023

53

Atef Ataya in AI Advances

Recommendation System using OpenAI, SciPy, and Atlas

· 12 min read · May 1, 2024

154 1

See more recommendations