Recommender Systems with Python — Part I: Content-Based Filtering

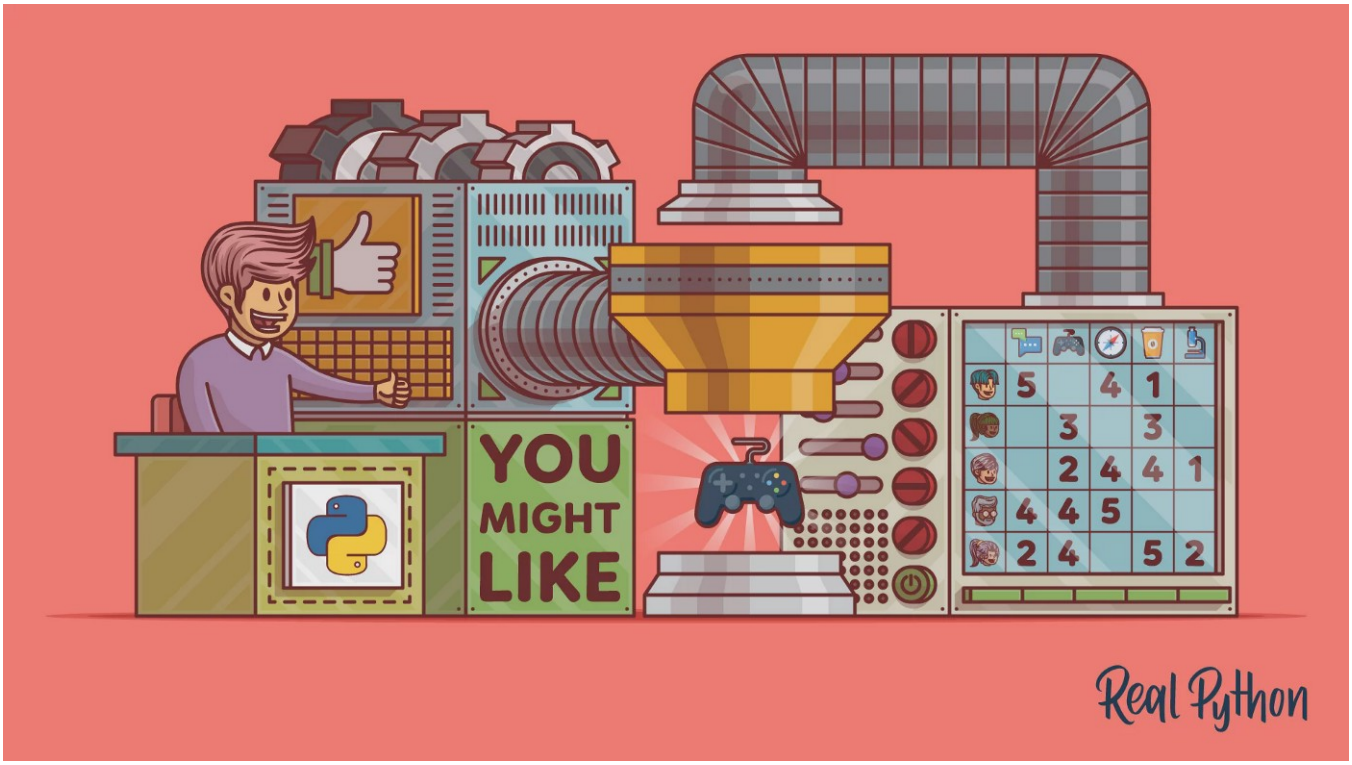
Nikita Sharma
Aug 22, 2019 · 9 min read

Image Source

This post is the first part of a tutorial series on how to build you own recommender systems in Python. To kick things off, we'll learn how to make an e-commerce item recommender system with a technique called **content-based filtering.**

> *What a time to be alive! Artificial intelligence is blooming as we speak, and the feeling of a machine or a system understanding a human, his/her choices, and likes and dislikes is truly amazing. Surprising people by recommending them products, songs, and movies as per their choice is what makes recommender systems so useful and close to our day to-day lives.*

Unfortunately, as of the day of this post's publication, Wikipedia defines recommender systems too narrowly, as "a subclass of information filtering systems that seeks to predict the 'rating' or 'preference' that a user would give to an item". Recommender systems are much more than this definition.



recommender systems with python

Recommendation paradigms

The distinction between approaches is more academic than practical, but it's important to understand their differences.

Broadly speaking, recommender systems are of 4 types:

1. **Collaborative filtering** is perhaps the most well-known approach to recommendation, to the point that it's sometimes seen as synonymous with the field. The main idea is that you're given a matrix of preferences by users for items, and these are used to predict missing preferences and recommend items with high predictions. All you need to get started is user and item IDs and a notion of preference by users for items (ratings, views, etc.). This approach will be discussed in part 2.

2. **Content-based filtering** algorithms are given user preferences for items and recommend similar items based on a domain-specific notion of item content. This approach also extends naturally to cases where item metadata is available (e.g., movie stars, book authors, and music genres).

3. **Social and demographic** recommenders suggest items that are liked by friends, friends of friends, and demographically-similar people. Such recommenders don't need any preferences by the user to whom recommendations are made, making them very powerful.

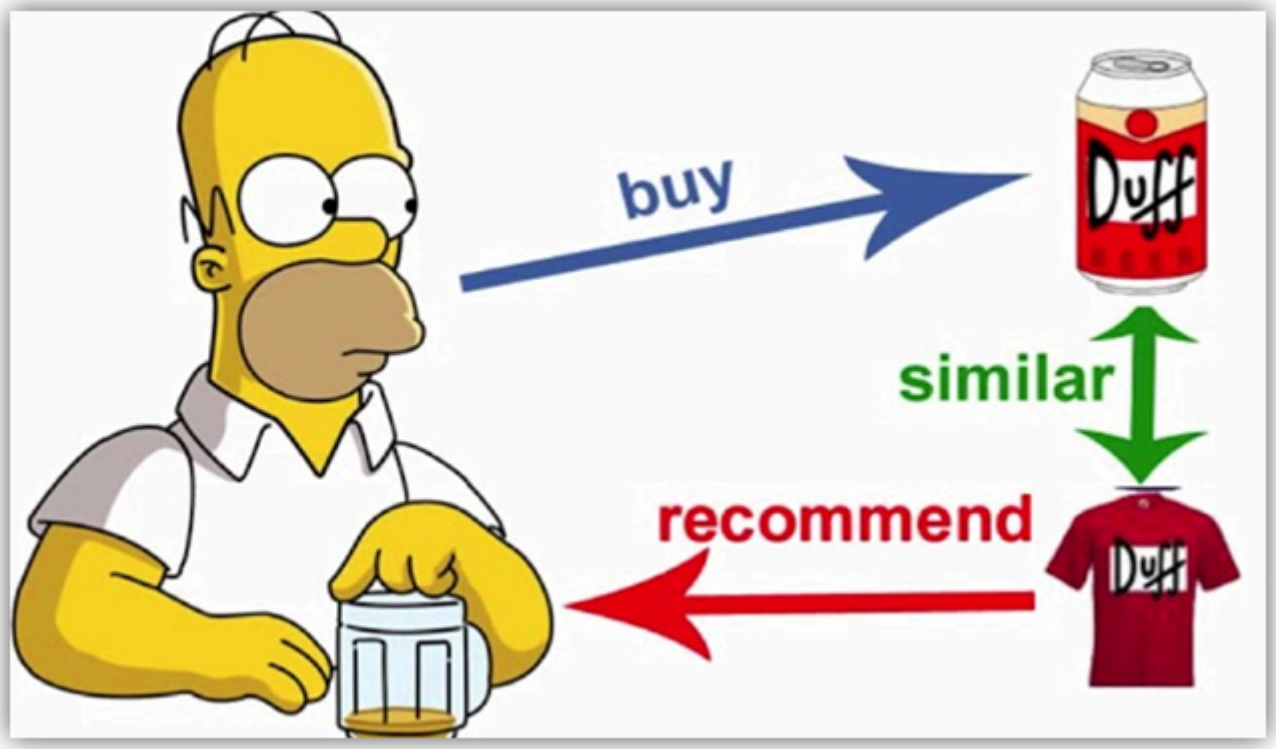
4. **Contextual** recommendation algorithms recommend items that match the user's current context. This allows them to be more flexible and adaptive to current user needs than methods that ignore context (essentially giving the same weight to all of the user's history). Hence, contextual algorithms are more likely to elicit a response than approaches that are based only on historical data.

> *In this post we will learn about content-based filtering with a basic implementation in Python.*

Let's get started!

Content-based recommender systems

Recommender systems are active information filtering systems that **personalize the information** coming to a user based on his interests, relevance of the information, etc. Recommender systems are used widely for recommending movies, articles, restaurants, places to visit, items to buy, and more.



How do content-based recommender systems work?

A content-based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on those recommendations, the engine becomes more and more accurate.

A recommender system has to decide between two methods for information delivery when providing the user with recommendations:

- **Exploitation.** The system chooses documents similar to those for which the user has already expressed a preference.
- **Exploration.** The system chooses documents where the user profile does not provide evidence to predict the user's reaction.

Now that we've taken a broad look at what recommender systems are and the different variations, let's work through an implementation of a content-based filtering system.

Setup Details

1. Jupyter notebook
2. Python==3.5.7
3. scikit-learn

The Dataset

Follow the link below for a dataset of 500 entries of different items like shoes, shirts etc., along with an item-id and a textual description of the item.

nikitaa30/Content-based-Recommender-System

It is a content based recommender system that uses tf-idf and cosine similarity for N Most SImilar Items from a dataset...

github.com

Loading the data

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
ds = pd.read_csv("/home/nikita/Downloads/sample-data.csv")
```

Creating a TF-IDF Vectorizer

Hold on! What is this scary term?

The **TF*IDF algorithm** is used to weigh a keyword in any document and assign the importance to that keyword based on the number of times it appears in the document. Put simply, the higher the TF*IDF score (weight), the rarer and more important the term, and vice versa.

Mathematically [don't worry it's easy :)],

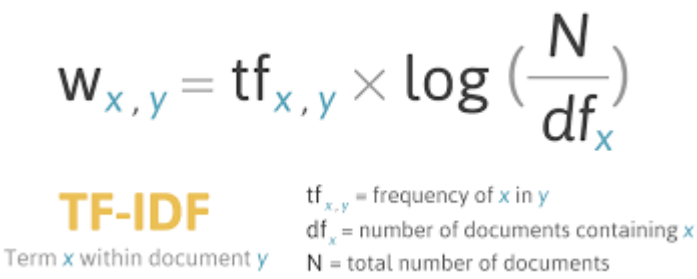
Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TF*IDF weight of that term.

The **TF (term frequency)** of a word is the number of times it appears in a document. When you know it, you're able to see if you're using a term too often or too infrequently.

```
TF(t) = (Number of times term t appears in a document) / (Total
number of terms in the document).
```

The **IDF (inverse document frequency)** of a word is the measure of how significant that term is in the whole corpus.

```
IDF(t) = log_e(Total number of documents / Number of documents with
term t in it).
```



TF-IDF calculation

In Python, scikit-learn provides you a pre-built TF-IDF vectorizer that calculates the TF-IDF score for each document's description, word-by-word.

```
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3), min_df=0,
stop_words='english')
tfidf_matrix = tf.fit_transform(ds['description'])
```

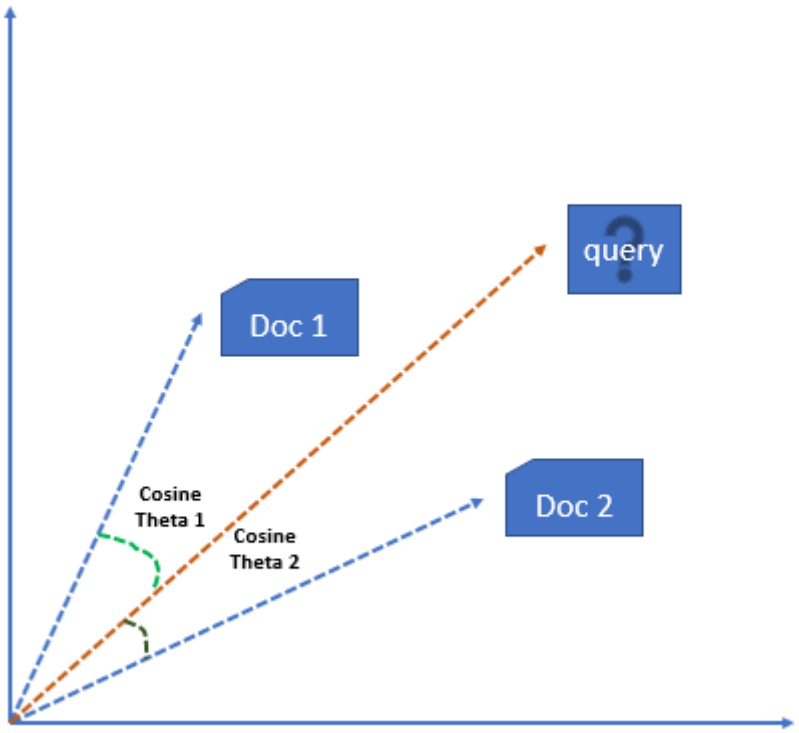
Here, the `tfidf_matrix` is the matrix containing each word and its TF-IDF score with regard to each document, or item in this case. Also, stop words are simply words that add no significant value to our system, like 'an', 'is', 'the', and hence are ignored by the system.

Now, we have a representation of every item in terms of its description. Next, we need to calculate the relevance or similarity of one document to another.

about what's possible with mobile machine learning.

Vector Space Model

In this model, each item is stored as a vector of its attributes (which are also vectors) in an n-dimensional space, and the angles between the vectors are calculated to determine the similarity between the vectors.



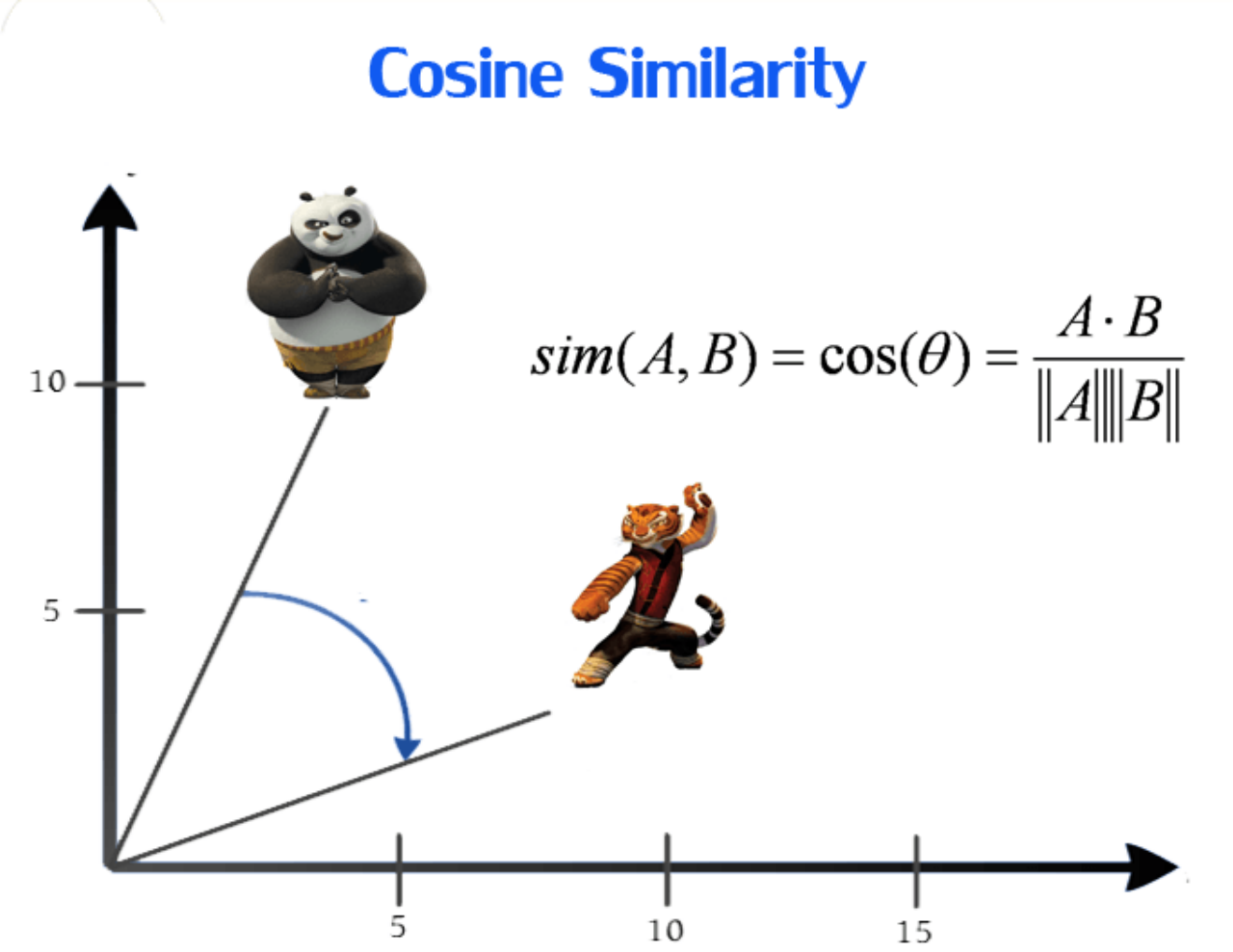
Documents represented as vectors

The method of calculating the user's likes / dislikes / measures is calculated by taking the cosine of the angle between the user profile vector (Ui) and the document vector; or in our case, the angle between two document vectors.

The ultimate reason behind using cosine is that the value of cosine will increase as the angle between vectors with decreases, which signifies more similarity.

The vectors are length-normalized, after which they become vectors of length 1.

Calculating Cosine Similarity



Cosine similarity

```
cosine_similarities = linear_kernel(tfidf_matrix, tfidf_matrix)
results = {}
for idx, row in ds.iterrows():
   similar_indices = cosine_similarities[idx].argsort()[:-100:-1]
   similar_items = [(cosine_similarities[idx][i], ds['id'][i]) for i
in similar_indices]
   results[row['id']] = similar_items[1:]
```

Here we've calculated the cosine similarity of each item with every other item in the dataset, and then arranged them according to their similarity with item `i`, and stored the values in `results`.

Check out this link to learn more~

> Vector Space Model
>
> A representation that is often used for text documents is the vector space model. In the vector space model a document...
>
> recommender-systems.org

Making a recommendation

So here comes the part where we finally get to see our recommender system in action.

```
def item(id):
  return ds.loc[ds['id'] == id]['description'].tolist()[0].split(' -
')[0]

# Just reads the results out of the dictionary.def

recommend(item_id, num):
    print("Recommending " + str(num) + " products similar to " +
item(item_id) + "...")
    print("-------")    recs = results[item_id][:num]
    for rec in recs:
       print("Recommended: " + item(rec[1]) + " (score:" +
str(rec[0]) + ")")
```

Here, we just input an `item_id` and the number of recommendations that we want, and voilà! Our function collects the `results[]` corresponding to that `item_id`, and we get our recommendations on screen.

Results

Here's a glimpse of what happens when you call the above function.

```
recommend(item_id=11, num=5)
```

Recommendations similar to organic cotton jeans-shorts

Recommendations similar to baby sunshade top

Analyzing the Results

Advantages of Content Based Filtering

- **User independence:** Collaborative filtering needs other users' ratings to find similarities between the users and then give suggestions. Instead, the content-based method only has to analyze the items and a single user's profile for the recommendation, which makes the process less cumbersome. Content-based filtering would thus produce more reliable results with fewer users in the system.
- **Transparency:** Collaborative filtering gives recommendations based on other unknown users who have the same taste as a given user, but with content-based filtering items are recommended on a feature-level basis.
- **No cold start:** As opposed to collaborative filtering, new items can be suggested before being rated by a substantial number of users.

Disadvantages of Content Based Filtering

- **Limited content analysis**: If the content doesn't contain enough information to discriminate the items precisely, the recommendation itself risks being imprecise.
- **Over-specialization:** Content-based filtering provides a limited degree of novelty, since it has to match up the features of a user's profile with available items. In the case of item-based filtering, only item profiles are created and users are suggested items similar to what they rate or search for, instead of their past history. A perfect content-based filtering system may suggest nothing unexpected or surprising.

Conclusion

We have learned to make a fully-functional recommender system in Python with content-based filtering. But as we saw above, content-based filtering is not practical, or rather, not very dependable when the number of items increases along with a need for clear and differentiated descriptions.

To overcome all the issues discussed earlier, we can implement collaborative filtering techniques, which have proven to be better and more scalable. We'll work on their implementations in the upcoming parts of the series.

Repository

To get the complete source code, follow the link to my GitHub repo, given below:

nikitaa30/Content-based-Recommender-System

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

Sources

Content-based Filtering

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the...

recommender-systems.org

The TF*IDF Algorithm Explained | Onely

Bartosz Góralewicz takes a look at the TF*IDF algorithm and its

www.onely.com

Please leave your comments and feedback below and help me improve!

Editor's Note: ***Heartbeat*** *is a contributor-driven online publication and community dedicated to exploring the emerging intersection of mobile app development and machine learning. We're committed to supporting and inspiring developers and engineers from all walks of life.*

Editorially independent, Heartbeat is sponsored and published by ***Fritz AI****, the machine learning platform that helps developers teach devices to see, hear, sense, and think. We pay our contributors, and we don't sell ads.*

If you'd like to contribute, head on over to our ***call for contributors****. You can also sign up to receive our weekly newsletters (****Deep Learning Weekly*** *and the* ***Fritz AI Newsletter****), join us on* ***Slack****, and follow Fritz AI on* ***Twitter*** *for all the latest in mobile machine learning.*

Recommender Systems Data Science For MI Heartbeat Python Programming