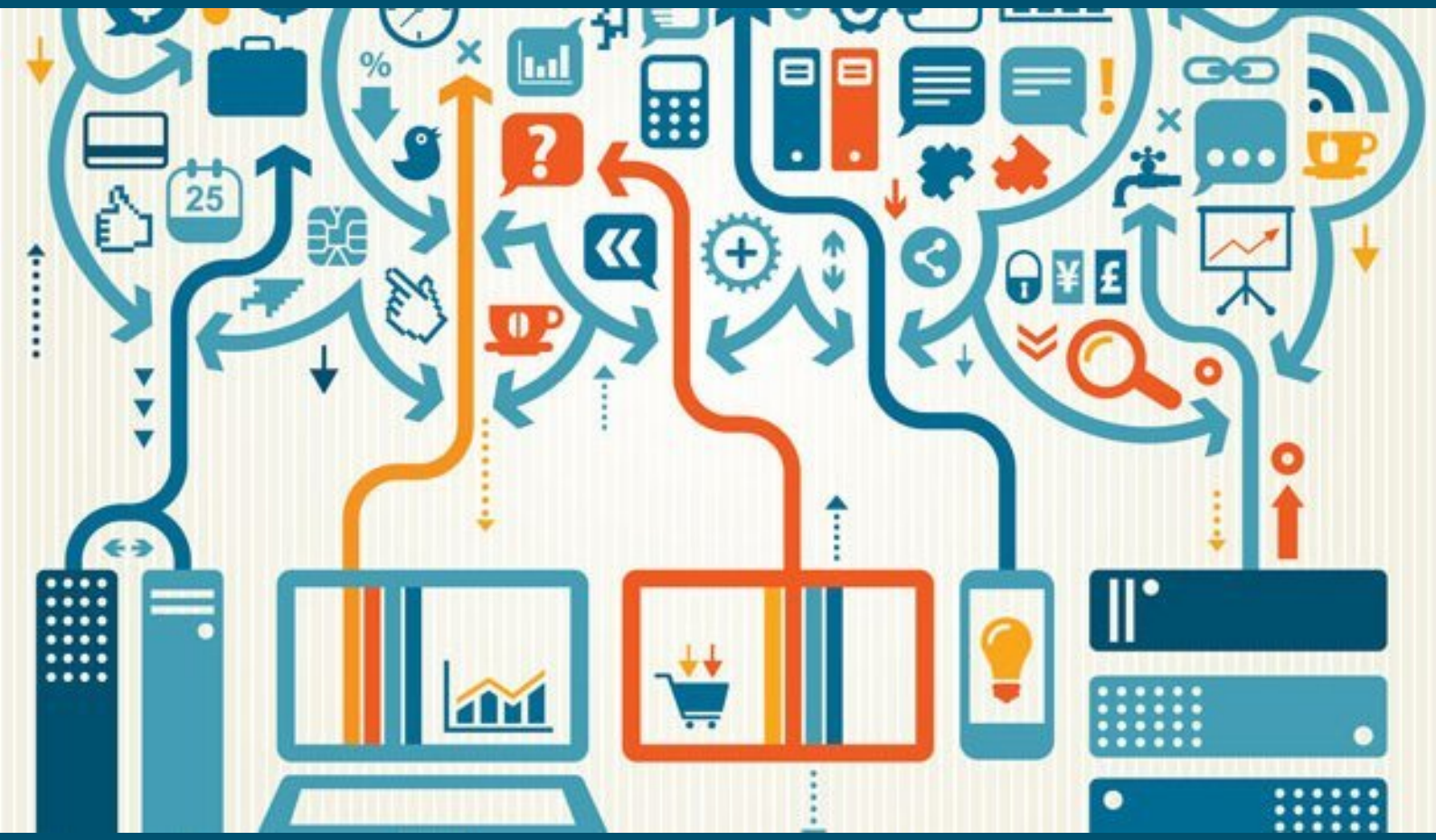
e-commerce

MACHINE LEARNING



PROPOSAL

prepared on 11th April, 2017

by Alexander van Zyl

Content

References

https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python
https://en.wikipedia.org/wiki/Item-item_collaborative_filtering#cite_note-1
http://infolab.stanford.edu/~ullman/mmds/ch9.pdf
https://www.packtpub.com/books/content/content-based-recommendation
https://www.fxpal.com/publications/content-based-recommendation-systems.pdf
https://en.wikipedia.org/wiki/Tf%E2%80%93idf
http://www.recommenderbook.net/media/Recommender_Systems_An_Introduction_Chapter03_Content-based_recommendation.pdf

Domain background

Recommendation Engines have been used to recommend a variety of things from movies, music, articles, products or even a company's next potential employee. For our particular area of interest we will briefly discuss Recommendation Engines related to e-commerce.

When it comes to recommendation engines there are typically two categories of algorithms:

Content Based

Content based algorithms usually work best when recommending similar or related items of the same type or category where the properties and context of each item can easily be determined. The advantage of this approach is no user data is really required to generate recommendations, disadvantage is that it doesn't really help the user discover new products they may be interested in based on their preferences, and/or previous ratings and purchases.

Collaborative Filtering

Collaborative filtering requires some sort of user input and based on a collection of past user behaviour can produce more tailored recommendations for a particular user. Collaborative filtering algorithms can be further broken down into several groups the two main ones are:

- **User-User Collaborative Filtering:** whereas if Person A purchases items 1, 2, 3 and Person B purchase items 2, 3, 4 both Person A and B share similarities (they both purchased items 2 and 3) therefore it is likely the Person A might be interested in item 4 and Person B may be interested in item 1.
- **Item-Item Collaborative Filtering:** instead of matching similar users item-item collaborative filtering matches products that a user has purchased or rated to other similar items. Item-item collaborative filtering was invented and used by Amazon.com in 1998

The disadvantage of Collaborative filtering is they can be relatively heavy on computational power they also require a large amount of user data, an obvious advantage though is recommendations can be more personalised on a per user bases.

Problem statement

An efficient way to automate the selection of related items for a single category specifically Android Tv Boxes.

The current manual approach requires the user to input the details of a product and based on its specifications select up to three products to be listed as a related item. In the case of the Tv boxes this might mean having two products that can both be used for viewing on a 4K resolution Tv screen but one may have 2GB ram and the other 1GB ram amongst other specifications.

The client has stressed that this can become time consuming when 1) the user entering the info. is unfamiliar with the product category and 2) having to manually browse and search products with similar specification.

Key points to keep in mind:

1) We are dealing with only one category of items Tv Boxes with similar specifications.
2) The e-store is relatively new so there is not a lot of user data to work with.
3) Given this is an internet business should the related items be recommended live (online) loading time is an important factor.

Given the above factors to solve this particular issue the obvious approach would be to use a Content Based Filtering algorithm since we are really only concerned with products in a single category where only the specifications/attribute values differ.

Dataset and inputs

Currently the product overview/description and specification are in text format concatenated together. So essentially we will have Model Code - Title - Description + Specification. Should we wish to run this through a TF-IDF (term frequency–inverse document frequency) algorithm we would have to strip english stop words i.e the, and, an etc. luckily this can be accomplished automatically.

Should we also want to experiment with some general clustering algorithms we will also need to parse the data. This will require the extraction of key features of each product that could affect the user's decision process of whether or not to purchase and item. The key features that have been identified are:

- Display Resolution
- OS Version
- Ram
- Internal Memory
- CPU Speed
- Supported video formats
- Price

All data will be parsed accordingly and saved as .csv making easier to work with or we can use the json that is returned directly form directly from the stores private API. Currently there is a total of only 65 products in the chosen category.

Solution statement

Given the initial data we have to work with the first approach of using TF-IDF (term frequency–inverse document frequency) may be the best first option as it will require less parsing of the data set and allow us to immediately test if this model will produce optimal results.

Very briefly TF-IDF work as follows:

Given we have a keyword ***i*** and a document ***j***:
TF (term frequency): Measures, how often a term appears.

- TF(***i, j***) = term frequency of keyword ***i*** in document ***j***

IDF (inverse document frequency): Aims to reduce the weight of terms that appear in all documents.

- IDF(***i***) = $log \frac{N}{n(i)}$
  - ***N*** = number of all recommended documents
  - ***n(i)*** = number of documents from ***N*** in which keyword ***i*** appears

Finally TF-IDF(***i, j***) is calculated as TF(***i, j***) * IDF(***i***)

Where in this case document can be thought of as our product description and specification.

Other alternative methods that may be worth exploring are:
Clustering techniques like K-Means
SVM (Support Vector Machines)

Benchmark model

To create our benchmark model we will manually select a product and 3 products that we think are most relevant to that product. We will then use this model to compare it to our solution model and see if the solution returns either the same 3 relevant products or ideally recommends products that are even more relevant to the selected product than what was previously manually selected.

Evaluation metrics

We will evaluate the performance of each model by measuring the training time in seconds. We will be scoring our similarities between products using cosine similarity function which is readily available from scikit learn.

So for example should Product B, C, and D be related to Product A which has been manually selected, we should see high scores of similarity for those related products versus any other products unless of course the manual selections are less related than what our solution model returns. In this case we will have a model that is better at selecting relevant products than the manual approach.

Project design

As this is related to the web the actual web store will have to interact with an API in which it will pull the related items probably form a Redis Database where the TF-IDF matrices have been saved.

However this is outside the domain of this project where our main focus is to find a model that can provide the most efficient solution. To keep things simple and focused we will be using Python Notebook in conjunction with the scikit learn library and other supporting libraries to test and train our models.

For the first phase will will concentrate on implementing the TF-IDF algorithm to see if it produce satisfactory results. The following action are to take place:

1) From all 65 products we will need to extract the following data from the returned json:

- ID
- Full name and short name
- Overview/Description
- Specification
- Related product ID's
- Retail price
- Meta keywords
- Meta description

2) We will start by training our TF-IDF algorithm just with the specification we will also look at only using the meta keywords as well as a combination of the various data pulled i.e overview with specification, or whether or not introducing any variation of the name produces better or worse results.

3) The TF-IDF will return a matrix which in turn we will use to determine the similarity scores for each item related to each other.
Record the training time

4) We select a master product and compare its current related product id's to the top 3 related product id's predicted by our algorithm. Since we will not be touching any databases we will just save each result to a .json or .csv file.

5) Should any product id's be returned that do not match any of the expected product id's we will inspect the specification of that product against the master product.

Should we not receive the expected results using TD-IDF we can then explore our other options.