**PRODUCT RECOMMENDATIONS**

Table of Contents

[PRODUCT RECOMMENDATIONS IMPORTANCE? 2](#_Toc44494243)

[RECOMMENDER PIPELINE 3](#_Toc44494244)

[WHAT ARE THE DIFFERENT TYPES OF RECOMMENDATIONS? 3](#_Toc44494245)

[Collaborative filtering: 4](#_Toc44494246)

[Content-based filtering: 5](#_Toc44494247)

[Hybrid Recommendation systems 6](#_Toc44494248)

[PROCESS FLOW A RECOMMENDATION ENGINE 7](#_Toc44494249)

[Python Libraries 8](#_Toc44494250)

[Choosing components 8](#_Toc44494251)

[App Engine 8](#_Toc44494252)

[Cloud SQL 8](#_Toc44494253)

[Spark 8](#_Toc44494254)

[Collecting the data 9](#_Toc44494255)

[Storing the data 9](#_Toc44494256)

[Analyzing the data 9](#_Toc44494257)

[Filtering the data 10](#_Toc44494258)

[Training the models 11](#_Toc44494259)

[ Rank 11](#_Toc44494260)

[ Lambda 11](#_Toc44494261)

[ Iteration 12](#_Toc44494262)

[Finding the right model 12](#_Toc44494263)

[ Training set 12](#_Toc44494264)

[ Validating set 12](#_Toc44494265)

[ Testing set 12](#_Toc44494266)

[Evaluation metrics for recommendation algorithms 13](#_Toc44494267)

[Big Data: Why and How? 13](#_Toc44494268)

[BENEFITS OF A PRODUCT RECOMMENDATION ENGINE 13](#_Toc44494269)

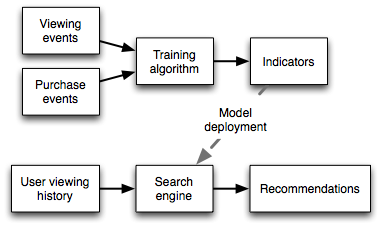
# PRODUCT RECOMMENDATIONS IMPORTANCE?

Shopping based on trends and recommendations have become a reality. Trends are generated from mass recommendations. It has become a major **business process** globally.

Ex. And just so you know, [35% of Amazon.com’s revenue is generated by its recommendation engine](http://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers). So what’s their **strategy**?

Amazon uses recommendations as a targeted marketing tool in both email campaigns and on most of its websites pages. Amazon will recommend many products from different categories based on what you are browsing and pull those products in front of you which you are likely to buy. Like the ‘frequently bought together’ option that comes at the bottom of the product page to lure you into buying the combo. This recommendation has one main goal: increase average order value i.e., to up-sell and cross-sell customers by providing product suggestions based on the items in their shopping cart or below products they’re currently looking at on-site.

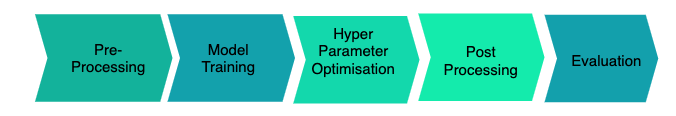
Below Example: Where a demand is being created. **Creating new user demand and keep user engaged** is the business model, based on recommendation.



**Data source**: User browser history.

# RECOMMENDER PIPELINE

A typical recommender system pipeline consists of the following five phases:



# WHAT ARE THE DIFFERENT TYPES OF RECOMMENDATIONS?

Three important types of recommendation engines:

* Collaborative filtering
* Content-Based Filtering
* Hybrid Recommendation Systems



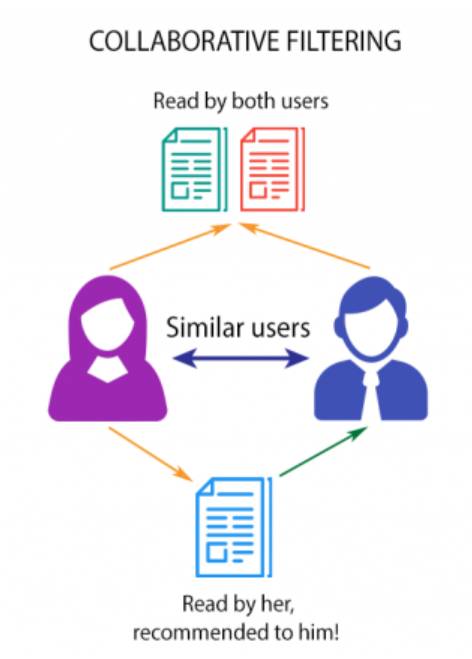
## Collaborative filtering:

Recommendations to user based on similar activity by other users.

**For example**, if a person A likes item 1, 2, 3 and B like 2,3,4 then they have similar interests and A should like item 4 and B should like item 1.

There are several types of algorithms:

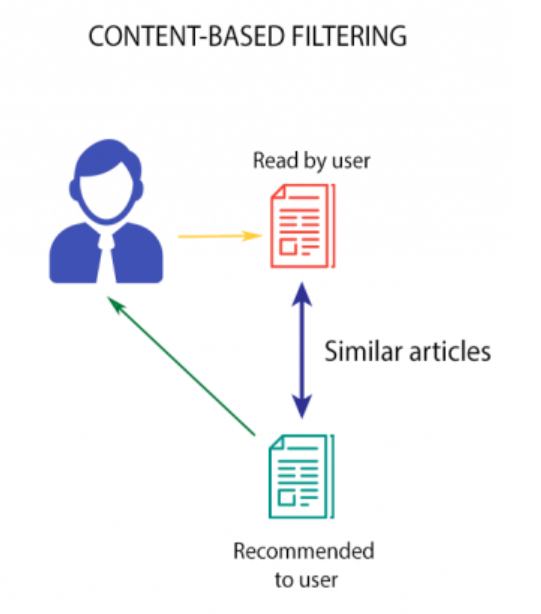
1. User-User Collaborative filtering
2. Item-Item Collaborative filtering



## Content-based filtering:

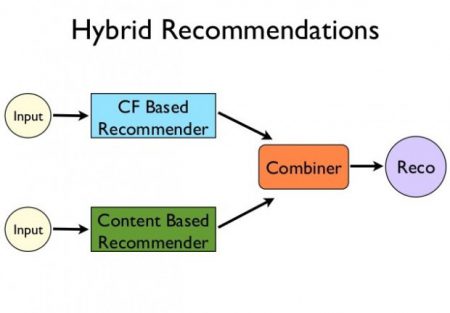
This algorithm based on the **keywords** that user had **liked/disliked** based on his/her **past** search history.

This algorithm can be used across multiple domain platforms like Movie Recommendation, Music recommendations, Product recommendations, News article recommendations, new online courses recommendations etc.



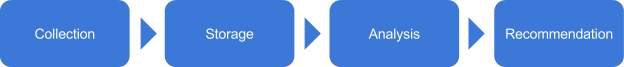
## Hybrid Recommendation systems

This algorithm is a combination of collaborative and content-based recommendation. It can provide more accurate recommendations than standard approaches. Ex. Netflix

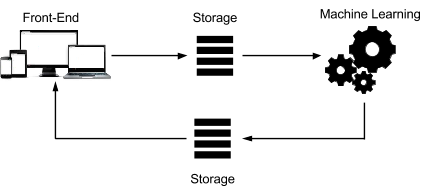


# PROCESS FLOW A RECOMMENDATION ENGINE

A recommendation engine typically processes data through the following four phases:



The architecture of such a system can be represented by the following diagram:



**Each step can be customized to meet the requirements. The system consists of:**

1. A scalable front end that records user interactions to collect data.
2. Permanent storage that can be accessed by a machine learning platform. Loading the data into this storage can include several steps, such as import- export and transformation of the data.
3. A machine learning platform that can analyze the existing content to create relevant recommendations.
4. Storage that can be used by the frontend, in real time or later, based on the timeliness requirements for recommendations.

# PYTHON LIBRARIES

A number of Python libraries are available that are specifically created for recommendation purposes. Here are the most popular ones:

* [**Surprise**](http://surpriselib.com/): A Python [scikit](https://www.scipy.org/scikits.html) building and analyzing recommender systems.
* [**Implicit**](https://implicit.readthedocs.io/en/latest/quickstart.html): Fast Python Collaborative Filtering for Implicit Datasets.
* [**LightFM**](https://lyst.github.io/lightfm/docs/home.html): Python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback.
* [**pyspark.mlib.recommendation**](https://spark.apache.org/docs/2.1.1/api/python/_modules/pyspark/mllib/recommendation.html): Apache Spark’s Machine Learning API.

## Choosing components

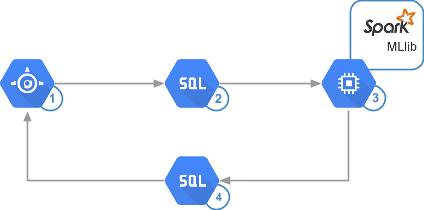
[App Engine](https://cloud.google.com/appengine/docs), [Cloud SQL](https://cloud.google.com/sql/docs), and [Apache Spark](https://spark.apache.org/) running on [App Engine](https://cloud.google.com/compute/docs) using [Dataproc](https://cloud.google.com/dataproc/).

App Engine can handle several tens of thousands of queries per second with little management required. Whether creating the website or saving the data to a backend storage, App Engine lets you write your code and deploy it to production in a matter of seconds.

Cloud SQL also offers simplicity of deployment. Cloud SQL can scale up to 32-core virtual machines with up to 208 GB of RAM, and can grow storage on demand to 10 TB with 30 IOPS per GB and thousands of concurrent connections. These specifications are plenty sufficient for the example in this solution, and for a large number of real-world recommendation engines. Cloud SQL also offers the advantage that it can be directly accessed from Spark.

Spark offers much better performance than a typical Hadoop setup; Spark can be 10 to 100 times faster. With [Spark MLlib](https://spark.apache.org/mllib/), you can analyze several hundreds of millions of ratings in minutes, which increases the agility of the recommendations, enabling the administrator to run the algorithm more often. Spark leverages memory for computing, as much as possible, to reduce round trips to disk. It also tries to minimize I/O. This solution uses Compute Engine to host the analysis infrastructure. Compute Engine helps to keep the price of the analysis as low as possible through its per-second, on-demand pricing.

The following diagram maps to the previous architecture diagram, but shows the technology at use for each step:



## Collecting the data

A recommendation engine can collect data about users based on their implicit *behavior* or their explicit *input*.

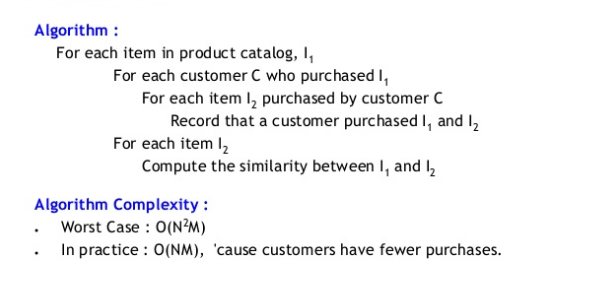
## Storing the data

The more data available to your algorithms, the better the recommendations will be. This means that any recommendations project can quickly turn into a big data project.

## Analyzing the data

Designing the analysis phase requires an understanding of the application's requirements. These requirements include:

* The *timeliness* of a recommendation. How quickly does the application need to present recommendations?
* The *filtering* approach for the data. Will the application base its recommendation on the user's tastes alone, pivot the data based on what other users think, or pivot on which products logically fit together?

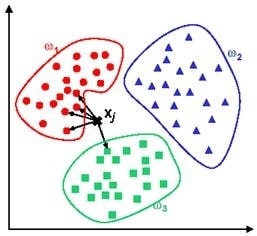


## Filtering the data

A core component of building a recommendation engine is *filtering*. The most common approaches include:

* **Content-based**: A popular, recommended product has similar attributes to what the user views or likes.
* **Cluster**: Recommended products go well together, no matter what other users have done.
* **Collaborative**: Other users, who like the same products the user views or likes, also liked a recommended product.

We use **K-Nearest algorithm, Jaccard’s coefficient, Dijkstra’s algorithm, cosine similarity** to better relate the data sets of people for recommending based on the rating or product.



## Training the models

Spark MLlib implements the [Alternating Least Squares (ALS)](https://dl.acm.org/citation.cfm?id=1608614) algorithm to train the models. You will use various combinations of the following parameters to get the best compromise between variance and bias:

* Rank: The number of unknown factors that led a user to give a rating. These could include factors such as age, gender, or location. The higher the rank, the better the recommendation will be, to some extent. Starting at 5 and increasing by 5 until the recommendation improvement rate slows down, memory and CPU permitting, is a good approach.
* Lambda: A *regularization* parameter to prevent *overfitting*, represented by high *variance*, and low *bias*. Variance represents how much the predictions fluctuate at a given point, over multiple runs, compared to the theoretically correct value for that point. Bias represents how far away the generated predictions are from the true value you're trying to predict. Overfitting happens when the model works well on training data using known noise but doesn't perform well on the actual testing data. The higher the lambda, the lower the overfitting but the greater the bias. Values of 0.01, 1 and 10 are good values to test.

The following diagram shows the relationship between variance and bias. The bullseye represents the value that the algorithm is trying to predict.

| Variance versus Bias (best is on the top left) |
| --- |
| Variance and bias |

* Iteration: The number of times that the training will run. In this example, you will do 5, 10, and 20 iterations for various combinations of rank and lambda.

The following example code shows how to start an ALS model training run in Spark.

from pyspark.mllib.recommendation import ALS  
model = ALS.train(training, rank = 10, iterations = 5, lambda\_=0.01)

## Finding the right model

The collaborative filtering using the ALS algorithm is based on three different sets of data:

* Training set: Contains data with known output. This set is what a perfect result would look like. In this solution, it contains the user ratings.
* Validating set: Contains data that will help tune the training to pick the right combination of parameters and choose the best model.
* Testing set: Contains data that will be used to evaluate the performance of the best trained model. This would be equivalent to running the analysis in a real-world example.

To find the best model, you need to calculate the root-mean-square error (RMSE) based on the model that was calculated, the validation set, and its size. The lower the RMSE, the better the model.

## Evaluation metrics for recommendation algorithms

* 1. [Mean Absolute Error](https://www.sciencedirect.com/topics/computer-science/mean-absolute-error) (MAE)
  2. Root Mean Square Error (RMSE)
  3. ***accuracy metrics:*** that are popularly used are
     + Reversal rate,
     + Weighted errors,
     + Receiver Operating Characteristics (ROC) and
     + Precision Recall Curve (PRC), Precision, Recall and
     + *F*-measure.

# BIG DATA: Why and How?

Since a product recommendation engine mainly runs on data. Your company may not have the storage capacity to store this enormous amount of data from visitors on your site. You can use online frameworks like **Hadoop, Spark** which allows you to store data in multiple devices to reduce dependability on one machine. Hadoop uses **HDFS** to split files into large blocks and distributes them across nodes in a cluster. This allows the dataset to be processed faster and more efficiently.

Finally, we process big data sets using the MapReduce programming model. With this, we can run the algorithm in the distributed file system at the same time and choose the most similar cluster.

# BENEFITS OF A PRODUCT RECOMMENDATION ENGINE

1. Revenue
2. Customer Satisfaction
3. Personalization – Individual users can be taken care of.
4. Accuracy in Reports