Apartment Machine Learning

User Recommendation Models

Collaborative Filtering Systems

The underlying intuition behind collaborative filtering is that if users A and B have similar taste in a product, then A and B are likely to have similar taste in other products as well. Two types of collaborative filtering:

1. Memory Based approaches
    a. Ratings of user-item combinations are predicted on the basis of their neighborhoods. User based essentially means that like minded users are going to yield strong and similar recommendations.
2. Model based approach
    a. Features associated to the dataset are parameterized as inputs of the model to try to solve an optimization related problem. Model based approaches include using things like decision trees, rule based approaches, latent factor models etc.

Advantages :

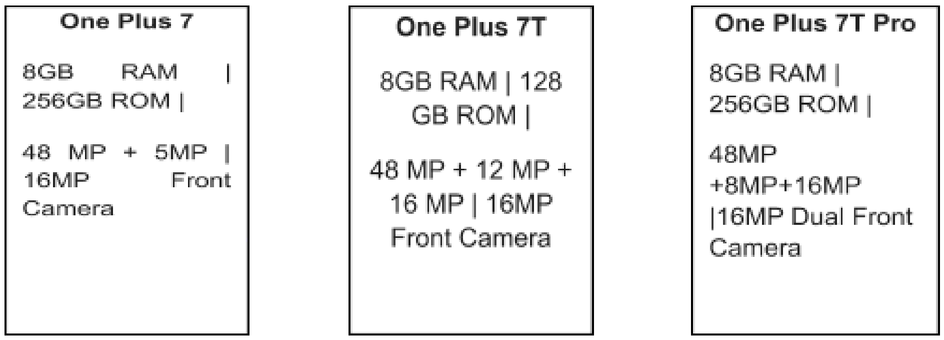
1. its simplicity to implement and the high level coverage they provide. It is also beneficial because it captures subtle characteristics (very true for latent factor models) and does not require understanding of the item content.

Disadvantages:

1. not friendly for recommending new items, this is because there has been no user/item interaction with it. This is referred to as the cold start problem. Memory based algorithms are known to perform poorly on highly sparse datasets.

Content-based Recommendation System

It is another type of recommendation system which works on the principle of similar content. If a user is watching a movie, then the system will check about other movies of similar content or the same genre of the movie the user is watching. There are various fundamentals attributes that are used to compute the similarity while checking about similar content.



One Plus 7

8GB RAM | 256GB ROM |

48 MP + 5MP | 16MP Front Camera

One Plus 7T

8GB RAM | 128 GB ROM |

48 MP + 12 MP + 16 MP | 16MP Front Camera

One Plus 7T Pro

8GB RAM | 256GB ROM |

48MP +8MP+16MP |16MP Dual Front Camera

Figure1: Different models of one plus.

Figure 1 image shows the different models of one plus phone. If a person is looking for one plus 7 mobile then, one plus 7T and one plus 7 Pro is recommended to the user.

To check the similarity between the products or mobile phone in this example, the system computes distances between them. One plus 7 and One plus 7T both have 8Gb ram and 48MP primary camera.

If the similarity is to be checked between both the products, Euclidean distance is calculated. Here, distance is calculated based on ram and camera;

$$\sqrt{(8-8)^2 + (48-48)^2} = 0$$

Euclidean distance (7T,7)

$$\sqrt{(8-12)^2 + (48-48)^2} = 4$$

Euclidean distance (7Pro,7)

Euclidean distance between (7T,7) is 0 whereas Euclidean distance between (7pro,7) is 4 which means one plus 7 and one plus 7T have similarities in them whereas one plus 7Pro and 7 are not similar products.

Content-based recommendation systems require the following sources of data:

1. Item level data source — you need a strong source of data associated to the attributes of the item. For our scenario, we have things like book price, num_pages, published_year, etc. The more information you know regarding the item, the more beneficial it will be for your system.
2. User level data source — you need some sort of user feedback based on the item you're providing recommendations for. This level of feedback can be either implicit or explicit. In our sample data, we're working with user ratings of books they've read. The more user feedback you can track, the more beneficial it will be for your system.

Advantages: Content based models are most advantageous for recommending items when there is an insufficient amount of rating data available. This is because other items with similar attributes might have been rated by the user. Hence, a model should be able to leverage the ratings along with the item attributes to generate recommendations even when there isn't a lot of data.

https://medium.com/@jwu2/content-based-recommender-systems-and-association-rules-599843cb2fd9

https://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/association.html

We find association rules over a set of transactions. We call this set a **transaction history** or *I*.

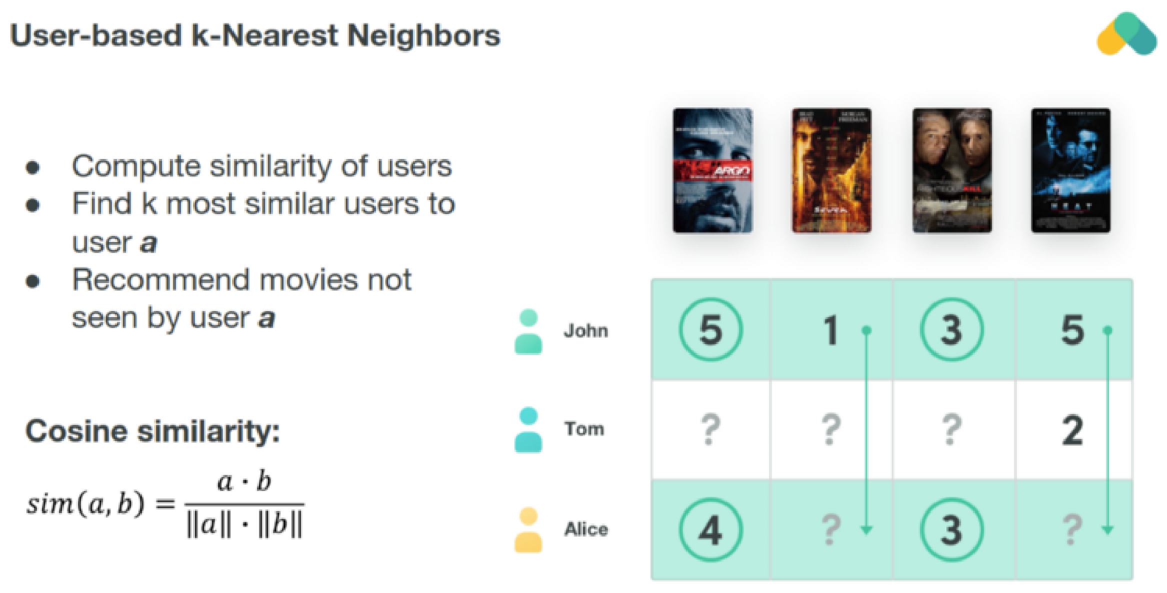
The **support** of item set X in a user's transaction history is the percentile of transactions in which X appears. If you happened to buy ketchup every time you went to a certain grocery, the support of ketchup over all your transactions would be one. The **confidence** of rule X → Y is the strength of the rule — how often the former implies the later. More precisely, it is the conditional probability that Y will be in a transaction if X is within that transaction as well. We can derive this probability by dividing the support of X U Y with the support of x.

If a rule X → Y has high confidence, the rule is strong and a set having X heavily implies that Y will also be in that set. If one knows X → Y, then they can suggest item Y to buyers of X.
A rule X → Y is said to be an association rule at a minimum support of s and minimum confidence of c, if the following two conditions are satisfied:
1. The support of X ∪ Y is at least s.
2. The confidence of X ⇒ Y is at least c.

Benefits of Association Rules

One of the benefits of creating a recommender system with association rules it that its recommendations are highly interpretable. While methods like latent factoring methods are powerful, the trends from which their results are gleaned are cryptic. Association rules are simple: "The books you like frequently have these keywords? We'll recommend you more books like that." While this may seem like a trivial point, remember our design principles! One of our goals was to have users trust that the recommendations we make will be relevant. Giving concrete explanations for each recommended item might give users more of an incentive to believe our recommendations.

Cosine Similarity



User-based k-Nearest Neighbors

- Compute similarity of users
- Find k most similar users to user ***a***
- Recommend movies not seen by user ***a***

Cosine similarity:

$$sim(a,b) = \frac{a \cdot b}{\|a\| \cdot \|b\|}$$

John	5	1	3	5
Tom	?	?	?	2
Alice	4	?	3	?

Cosine Similarity is another form of similarity measure like the Euclidean Distance. “Similarity” in data mining refers to the distance between dimensions in a dataset that indicate the properties of the data object. There will be a high degree of similarity if this distance is small, but a low degree of similarity if the distance is large.

You can use Cosine Similarity to assess how similar data objects are, regardless of their size. Data objects are treated as a vector in a dataset, and to find the similarity between two vectors, you use this formula:

Cos(x, y) = x . y / ||x|| * ||y||

x . y → dot product

||x|| → length of the vector

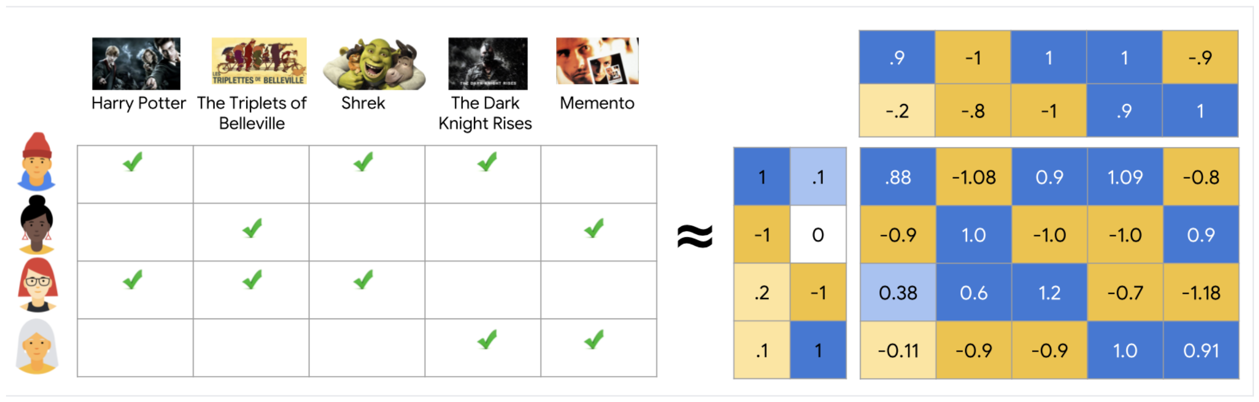
||x|| * ||y|| → cross product

To find the dissimiarlity, all you do is subtract the above result by 1: Dis(x, y) = 1 - Cos(x, y)

Advantages: Cosine similarity can be used to compare documents or even measure how similar are two different sentences. This method ignores differences in the magnitude or scale of the vectors and compares the similarity in the direction or orientation of the vectors. Cosine similarity is also low in complexity, especially for sparse vectors where just the non-zero coordinates need to be taken into account.

Disadvantages: The fact that the magnitude of vectors (term frequency) is not taken into account while calculating cosine similarity is one of its key drawbacks. In effect, this indicates that some value differences are not fully taken into consideration.

Matrix Factorization:



Matrix factorization is a simple embedding model that is used to train machine learning models. In order to represent our data, we would need a matrix of size m by n, where m represents all the users (m is the number of users) we have questioned and n represents the items in question (n is the number of items). The model learns based on the product of the user embedding matrix (with dimensions m x d) and the transpose of the item embedding matrix (with dimensions d x n), which yields an m x n matrix that is represented above.

Through an ordered rectangular array of integers or functions, Matrix Factorization, a mathematical model, enables the system to divide an entity into several smaller entries in order to identify the characteristics or data behind user and item interactions.

This method suggests items based on user preferences. It determines whether the request is met while taking into account the user's prior behaviors, patterns found, or any explicit feedback given by the user, and then makes a recommendation accordingly.

Advantages:

- **No Domain Knowledge Necessary**
- **Serendipity**
- **Great Starting Point**

Disadvantages:

- **Cannot Handle Fresh Items**
- **Hard to Include Side Features for Query / Item**

- https://www.itransition.com/machine-learning/recommendation-systems
- https://developers.google.com/machine-learning/recommendation
- https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed
- https://www.microsoft.com/en-us/research/lab/microsoft-research-asia/articles/personalized-recommendation-systems/
- https://www.upgrad.com/blog/recommendation-system-machine-learning/
- https://towardsdatascience.com/recommendation-systems-explained-a42fc60591ed
- What Are Recommendation Systems in Machine Learning? | Analytics Steps
- Recommendations AI modeling | Google Cloud Blog
- https://medium.com/@jwu2/content-based-recommender-systems-and-association-rules-599843cb2fd9

Price Forecast Models

Autoregressive Integrated Moving Average (ARIMA)

Auto Regressive Integrated Moving Average, ARIMA, models are among the most widely used approaches for time series forecasting. It is actually a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

Parameters:

- p: The number of lag observations included in the model, also called the lag order.
- d: The number of times that the raw observations are differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

Tutorial

http://ucanalytics.com/blogs/step-by-step-graphic-guide-to-forecasting-through-arima-modeling-in-r-manufacturing-case-study-example/

Advantages:

- Only requires the prior data of a time series to generalize the forecast.
- Performs well on short term forecasts.
- Models non-stationary time series.

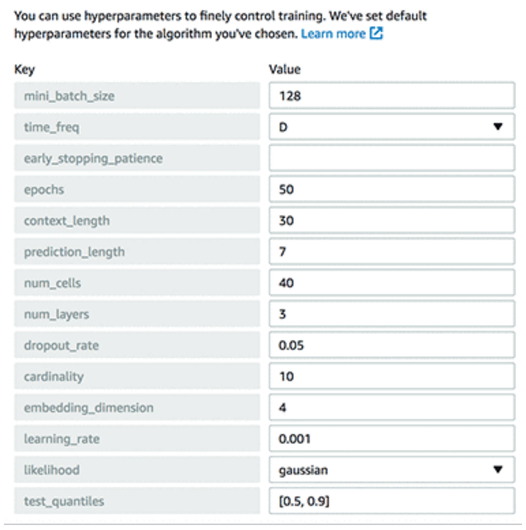
Disadvantages:

- Difficult to predict turning points.
- There is quite a bit of subjectivity involved in determining (p,d,q) order of the model.
- Computationally expensive.
- Poorer performance for long term forecasts.
- Cannot be used for seasonal time series.
- Less explainable than exponential smoothing.

DeepAR

DeepAR developed by Amazon is a probabilistic forecasting model based on autoregressive recurrent neural networks.

Customizable Hyperparameters:



You can use hyperparameters to finely control training. We've set default hyperparameters for the algorithm you've chosen. Learn more

Key	Value
mini_batch_size	128
time_freq	D
early_stopping_patience	
epochs	50
context_length	30
prediction_length	7
num_cells	40
num_layers	3
dropout_rate	0.05
cardinality	10
embedding_dimension	4
learning_rate	0.001
likelihood	gaussian
test_quantiles	[0.5, 0.9]

Tutorial

https://aws.amazon.com/blogs/machine-learning/now-available-in-amazon-sagemaker-deepar-algorithm-for-more-accurate-time-series-forecasting/

Advantages & Disadvantages

- Minimal Feature Engineering: The model requires minimal feature engineering, as it learns seasonal behavior on given covariates across time series.
- Monte Carlo Sampling: It is also possible to compute consistent quantile estimates for the sub-ranges of the function, as DeepAR implements Monte Carlo sampling. This could, for instance, be useful when deciding on safety stock.
- Built-in item supersession: It can predict on items with little history items by learning from similar items
- Variety of likelihood functions: DeepAR does not assume Gaussian noise, and likelihood functions can be adapted to the statistical properties of the data allowing for data flexibility.

Prophet

Prophet, which was released by Facebook's Core Data Science team, is an open-source library developed by Facebook and designed for automatic forecasting of univariate time series data.

Parameters:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

- The Growth Function (and change points):
    - Linear Growth

      Uses a set of piecewise linear equations with differing slopes between change points.

    - Logistic growth

      Useful when your time series has a cap or a floor in which the values you are modeling becomes saturated and can't surpass a maximum or minimum value (think carrying capacity). When logistic growth is used, the growth term will look similar to a typical equation for a logistic curve (see below), except it the carrying capacity (C) will vary as a function of time and the growth rate (k) and the offset(m) are variable and will change value at each change point.

    - Flat

      No growth over time. Fluctuations only from seasonality.

- Seasonality Function
    - A Fourier Series (successive sines and cosines multiplied by some coefficient) to approximate the seasonality (cyclical pattern) in our data.

$$s(t) = \sum_{n=1}^{N} (a_n \, cos(\frac{2\pi nt}{P}) + b_n \, sin(\frac{2\pi nt}{P}))$$

- Holiday Function

Tutorial

https://medium.com/p/f15ecf2c0e3a

Advantages:

- Ability to easily model any number of seasonalities
- Ability to work with missing dates in time-series
- Easily integrates holidays in the model
- Built-in uncertainty modeling with full Bayesian sampling allows for model transparency
- Allowing flexible step-wise linear or logistic trend modeling with user-specified change points

Disadvantages

- Can be volatile with a low number of observations
- Long-horizon forecasting can be volatile with automatic changepoint selection

Temporal Fusion Transformer (Google)

A novel attention-based architecture which combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics.

Parameters

- t, timestep
- w, lookback window
- t max, step ahead window
- x observed past inputs in period [t-k… t]
- x observed future known inputs in period [t+1… t max] which is also the prediction window
- s, static variable if applicable

Advantages

- Rich features

  TFT supports 3 types of features: i) temporal data with known inputs into the future ii) temporal data known only up to the present and iii) exogenous categorical/static variables, also known as time-invariant features.

- Heterogeneous time series

  Supports training on multiple time series, coming from different distributions. To achieve that, the TFT architecture splits processing into 2 parts: local processing which focuses on the characteristics of specific events and global processing which captures the collective characteristics of all time series.

- Multi-horizon forecasting

  Supports multi-step predictions. Apart from the actual prediction, TFT also outputs prediction intervals, by using the quantile loss function.

- Interpretability

  At its core, TFT is a transformer-based architecture. By taking advantage of self-attention, this model presents a novel Muti Head attention mechanism which when analyzed, provides extra insight on feature importances. For example, Multi-Horizon

Quantile Recurrent Forecaster (MQRNN)[3] is another DNN implementation with good performance but does not provide any insight regarding feature interpretability.

- High Performance

  During benchmarks, TFT outperformed traditional statistical models (ARIMA) as well as DNN-based models such as DeepAR, MQRNN and Deep Space-State Models (DSSM)[4].

- Documentation

  Although it is a relatively new model, there are already open source implementations of TFT both in Tensorflow and Python.