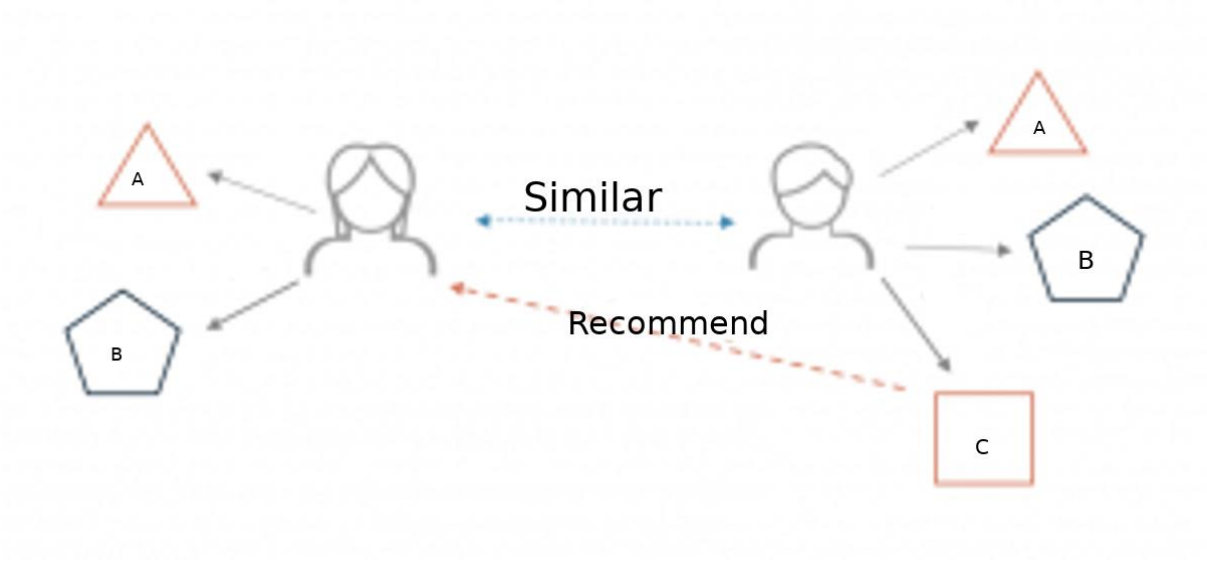
In Depth Analysis

What is a Recommender system?

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you.



Two users buy the same items A and B from an ecommerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

Different types of recommendation engines:

The most common types of recommendation systems are **content based** and **collaborative filtering** recommender systems.

In collaborative filtering the behaviour of a group of users is used to make recommendations to other users. Recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie.

There are two types of collaborative models **Memory-based** methods and **Model-based** methods. The advantage of memory-based techniques is that they are simple to

implement and the resulting recommendations are often easy to explain. They are divided into two:

1. **User-based collaborative filtering**:

In this model products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example if Derrick and Dennis like the same movies and a new movie comes out that Derick likes, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.

2. **Item-based collaborative filtering**:

These systems identify similar items based on users' previous ratings. For example if users A,B and C gave a 5 star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A,B and C.

Model-based methods are based on matrix factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users' rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve the accuracy. Examples of such model-based methods include decision trees, rule-based models, Bayesian methods and latent factor models.

1. SVD Matrix Factorization for Recommendations in Python:

Matrix factorization is the breaking down of one matrix in a product of multiple matrices. It's extremely well studied in mathematics, and it's highly useful. There are many different ways to factor matrices, but singular value decomposition is particularly useful for making recommendations.

SVD in the context of recommendation systems is used as a collaborative filtering (CF) algorithm. For those of you who don't know, collaborative filtering is a method to predict a rating for a user item pair based on the history of ratings given by the user and given to the item. Most CF algorithms are based on user-item rating matrix where each row represents a user, each column an item. The entries of this matrix are ratings given by users to items.

What is singular value decomposition (SVD)?

At a high level, SVD is an algorithm that decomposes a matrix R into the best lower rank (i.e. smaller/simpler) approximation of the original matrix R. Mathematically, it decomposes R into a two unitary matrices and a diagonal matrix:

$$R = U \Sigma V^T$$

Where, R is user's ratings matrix,

U is the user "features" matrix,

Σ is the diagonal matrix of singular values (essentially weights),

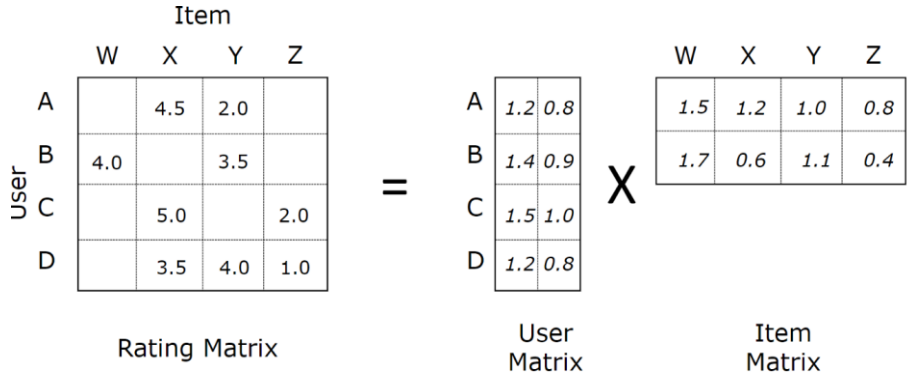
VT is the movie "features" matrix.

U and VT are orthogonal, and represent different things. U represents how much users "like" each feature and V represents how relevant each feature is to each movie.

To get the lower rank approximation, we take these matrices and keep only the top k features, which we think of as the underlying tastes and preferences vectors.

SVD and Matrix factorization:

SVD is a matrix factorization technique that is usually used to reduce the number of features of a data set by reducing space dimensions from N to K where K < N. For the purpose of the recommendation systems however, we are only interested in the matrix factorization part keeping same dimensionality. The matrix factorization is done on the user-item ratings matrix. From a high level, matrix factorization can be thought of as finding 2 matrices whose product is the original matrix.



Each item can be represented by a vector `*qi*`. Similarly each user can be represented by a vector `*pu*` such that the dot product of those 2 vectors is the expected rating

$$expected\ rating = \hat{r}_{ui} = q_i^T p_u$$

It is a form of factorization!!

`*qi*` and `*pu*` can be found in such a way that the square error difference between their dot product and the known rating in the user-item matrix is minimum

$$minimum(p,q) \sum_{(u,i)\epsilon K} \left(r_{ui} - q_i^T . p_u\right)^2$$

Regularization:

For our model to be able to generalize well and not over-fit the training set, we introduce a penalty term to our minimization equation. This is represented by a regularization factor λ multiplied by the square sum of the magnitudes of user and item vectors.

$$minimum(p,q) \sum_{(u,i)\epsilon K} \left(r_{ui} - q_i^T . p_u\right)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

To illustrate the usefulness of this factor imagine we have an extreme case where a low rating given by a user to a movie with no other rating from this user. The algorithm will minimize the error by giving `*qi*`a large value. This will cause all rating from this user to other movies to be very low. This is intuitively wrong. By adding the magnitude of the vectors to the equation, giving vectors large value will minimize the equation and thus such situations will be avoided.

Minimizing with Stochastic Gradient Descent (SGD):

The above equation is minimized using stochastic gradient descent algorithm. From a high level perspective SGD starts by giving the parameters of the equation we are trying to minimize initial values and then iterating to reduce the error between the predicted and the actual value each time correcting the previous value by a small factor. This algorithm uses a factor called learning rate γ which determines the ratio of the old value and the new computed value

after each iteration. Practically, when using high γ one might skip the optimal solution whereas when using low γ values a lot of iterations are needed to reach optimal value.

If I wanted to put this kind of system into production, I'd want to create a training and validation set and optimize the number of latent features (k) by minimizing the Root Mean Square Error. Intuitively, the Root Mean Square Error will decrease on the training set as '**k**' increases.

Making Movie Recommendations:

Finally, with the predictions matrix for every user, I can build a function to recommend movies for any user. All I need to do is return the movies with the highest predicted rating that the specified user hasn't already rated. Though I didn't use actually use any explicit movie content features (such as genre or title), I'll merge in that information to get a more complete picture of the recommendations.

2. Content based recommended system:

It uses metadata such as genre, producer, and actor, musician to recommend items say movies or music. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Similarly you can get music recommendations from certain artists because you liked their music. Content based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

A Content-based recommender system tries to recommend items to users, based on their profile. The user's profile revolves around the user's preferences and tastes, or based on the user ratings.

Encoding your data:

There are a number of popular encoding schemes but the main ones are:

- One-hot encoding
- Term frequency–inverse document frequency (TF-IDF) encoding
- Word embedding

Creating a TF-IDF Vectorizer:

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.

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

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

TF-IDF calculation

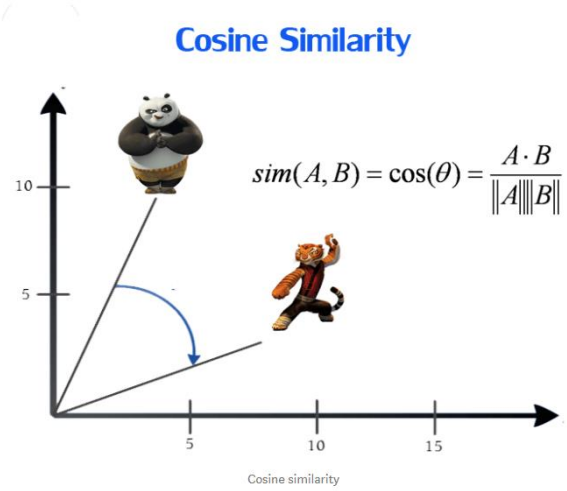
Cosine Similarity:

Well cosine similarity is a measure of similarity between two non-zero vectors. One of the beautiful thing about vector representation is we can now see how closely related two sentence are based on what angles their respective vectors make.

Cosine value ranges from -1 to 1.So if two vectors make an angle 0, then cosine value would be 1, which in turn would mean that the sentences are closely related to each other.

If the two vectors are orthogonal, i.e. cos 90 then it would mean that the sentences are almost unrelated.

Now we start to see how this can be helpful to us: For example, the movies Toy Story and Monsters, Inc. have a *cosine similarity* of 0.74. We would have expected these movies to have a relatively high similarity. In contrast, the cosine similarity between the movies Toy Story and Terminator 2 is 0.28 - as expected much lower.



We can now recommend movies based on the movies that a user has already watched or rated using the cosine similarity. We would recommend movies with the largest similarity to the ones already highly rated by the user.