**KNN Cosine Similarity –**Cosine similarity is used mainly to calculate similarity between two vectors. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in the same direction.

**Memory-based algorithms**

Memory-based algorithms approach the collaborative filtering problem by using the entire database. It takes ratings given by users and uses their preferences to predict movies similar to a movie a user likes.

**Similarity Measurements**

In order to measure similarity, we want to find the correlation between two users. This gives us a value from -1 to 1 which determines how alike two users are. A value of 1 means that they both rate in the exactly the same manner, whereas a value of -1 means that they rate things exactly opposite (i.e. one high, the other low or vice versa).

Pearson correlation coefficient is a basic correlation algorithm for samples adapted for rating information. It tries to measure how much two users vary together from their normal votes - that is, the direction/magnitude of each's vote in comparison to their voting average. If they vary in the same way on the items they have rated in common, they will get a positive correlation; otherwise, they will get a negative correlation.

**Advantages**

* The quality of predictions are rather good.
* This is a relatively simple algorithm to implement for any situation.
* It is very easy to update the database, since it uses the entire database every time it makes a prediction.

**Disadvantages**

* It uses the entire database every time it makes a prediction, so it needs to be in memory it is very, very slow.
* Even when in memory, it uses the entire database every time it makes a prediction, so it is very slow.
* It can sometimes not make a prediction for certain active users/items. This can occur if the active user has no items in common with all people who have rated the target item.
* Overfits the data. It takes all random variability in people's ratings as causation, which can be a real problem. In other words, memory-based algorithms do not generalize the data at all.

CSE3120 MOVIELENS1

USING MOVIELENS 20M DATASET, LOAD DATA AS SPARK DATAFRAMES

The correlation function in the ml subpackage pyspark.ml.stat requires us to provide a column of type Vector. So we convert columns into a vector column first using the VectorAssembler and then apply the correlation. Then we get the result as a numpy array. But return result as DataFrame

EXTRACTING YEAR FROM title COLUMN OF movie\_df, ATTACHING THE EXTRACTED YEARS AS A NEW COLUMN

createorReplaceTempView is used when you want to store the table for a particular spark session

NO. OF MOVIES GIVEN EACH RATING IN DESCENDING ORDER

BUILD RECOMMENDATION MODEL USING ALS ON TRAINING DATA

MAKE PREDICTIONS FOR TEST DATA THEN EVALUATE USING RMSE

ALS is used to predict ratings

Alternating Least Squares (ALS) is a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for a larges-scale collaborative filtering problems. Spark MLlib library for Machine Learning provides a Collaborative Filtering implementation by using Alternating Least Squares. The implementation in MLlib has these parameters:•maxIter specfies maximum iterations defaults to 10•regParam specifies the regularization parameter in ALS (defaults to 1.0).

A NaN result is due to SPARK-14489 and because the model can't predict values for users for which there's no data. A temporary workaround is to exclude rows with predicted NaN values or to replace them with a constant, for instance, the general mean rating.

MOVIES RATED BY USER12, RATING PREDICTION FOR USER12

MOVIELENS LATEST SMALL DATASET

REMOVING NOISE:FINAL DATASET CONSISTS OF MOVIES VOTES FOR BY ATLEAST 10 USERS,

FINAL DATASET RETAINS USERS WHO’VE VOTED FOR ATLEAST 50 MOVIES

Our final\_dataset has dimensions of 2121 \* 378 where most of the values are sparse. To reduce the sparsity we use the csr\_matrix function from the scipy library.

To find similar movies and sort them based on their similarity distance and output only the top 10 movies with their distances from the input movie KNN cosine similarity is used.

RECOMMENDATIONS FOR IRON MAN AND MEMENTO

To read in a CSV file, we access the DataFrameReader class through read and then call the csv() method on it. We also specify option("header", "true") so that the first row of the file is used for the column headers. Each row of the ratings DataFrame represents one rating for one movie (movieId) by one user (userId). The ratings use a 5-star scale with half-star increments from 0.5 stars up to 5.0 stars. We can print the DataFrame's column names and types using the printSchema() method. Each row of the movies DataFrame represents one movie and its title and genre(s), indexed by the key movieId. We will use this DataFrame to get the movie titles out so we know which movie the ratings in the ratings DataFrame are actually referring to.

LOADING movies AND ratings DATASETS

To get the most popular movies, we are looking for the movies with the highest number of ratings. To do this, we will perform the following transformations on the ratings DataFrame:group by movieId, count the number of users (userId) associated with each movie, rename this column to num\_ratings, sort by num\_ratings in descending order

In the next cell, we perform these transformations in pySpark and store the DataFrame as most\_popular. The DataFrame methods we have used here are:groupBy - groups the DataFrame by the given column, agg - allows us to perform an aggregate calculation on grouped data (this can be a built-in aggregation function such as count or a user defined function), withColumnRenamed - renames an existing column with a new column name, sort - sorts by the specified column(s)

Because transformations are lazy in Spark, the transformations above aren't performed until we call an action, such as show(), take(), or collect().

MOVIES AND most\_popular ARE JOINED

This DataFrame contains only the movieId and num\_ratings. The actual title of the movie is stored in the movies DataFrame. To get the movie titles, we can join our most\_popular DataFrame with the movies DataFrame on movieId. By default, join performs an inner join which is what we want in this case.We now have a list of the most popular (or most rated) movies on the MovieLens website. As expected, the titles listed here are indeed all well-known movies.

Top rated movies

We've got the top 10 most popular movies, but now we want to see which movies are perceived to be the best. To get the top rated movies, we are looking for the movies with the highest average rating. To do this, we will use the ratings DataFrame and:group by movieId, calculate the average rating for each movie, rename this column to avg\_rating, sort by avg\_rating in descending order

We will again join this DataFrame with the movies DataFrame so we know which movie each movieId is referring to.

The movies listed here appear to be quite niche. We want to focus on top rated movies that also have a decent number of ratings, so want to take into account both the average rating and the number of ratings. We can easily create a DataFrame which has both of these columns by specifying multiple expressions within one agg() call.

We see that all of the movies with an average rating of exactly 5.0 have 2 or less ratings. We would like to only consider movies that have achieved some minimum number of ratings. To determine an appropriate threshold, we should investigate the distribution of num\_ratings. We can do this by calculating some summary statistics within Spark. To calculate quantiles we use the approxQuantile method. This method can calculate the quantiles of the specified column approximately or exactly, depending on the value of the relative error parameter. If the relative error parameter is set to 0 then the quantiles are calculated exactly, however this can be expensive.

The mean is much greater than the median value, suggesting that this distribution is skewed to the right. We will choose a minimum threshold of 500 ratings, however there is no right or wrong answer here and the reader is encouraged to experiment with different values for this threshold.

We've now gotten a list of the top rated movies on MovieLens, which includes the usual movies considered to be all time greats such as The Shawshank Redemption and Casablanca. Interestingly, nearly all of these movies appear in the top 100 of the IMDb top rated movies list as well, with the exception of the The Third Man (listed as #135) and Band of Brothers which is technically a TV series rather than a movie.

What's also interesting is that this list of movies is not the same as the list of the most popular movies. The Shawshank Redemption, Schindler's List, and The Usual Suspects were all popular movies which also appear in this list. However, other movies such as Pulp Fiction, Forrest Gump, and The Silence of the Lambs made the top 10 most popular but not the top 10 (or even top 20) most rated. This suggests that some movies actually divide opinion.

Most polarising movies (Marmite movies) are the movies that divide opinion, with people tending to rate them either really highly or really poorly. Again, we only want to consider movies that achieve some minimum number of ratings - we will stick with our previous threshold of 500 ratings.

To approach this, we will look for the movies with the highest standard deviation in rating. This is a measure of how much the data varies from the mean, so in this case, how much a movie's ratings vary around its mean rating. A high standard deviation would suggest that the movie's ratings are highly variable. There are other approaches to this as well, for instance, what proportion of the ratings are very positive or very negative.

LOADING users.dat, ratings.dat and movies.dat FROM MOVIELENS 1M DATASET

Ages and occupation are encoded as integers.Movie genre is a pipe-separated string, so analysis of genre require some transformation to bring this column in a usable form. We want to analyse the mean ratings for a specific movie by age and gender. Next we need to merge all data into one dataframe to make handling easier. Merging strategy is to first merge ratings with users and followed by merging movies. Pandas infers which columns to use for the merge by finding overlapping names.The function pivot table reshapes the data frame depending on index and columns from input parameters. We create a new data frame with mean ratings for each move title and split gender as columns (F and M).Filter movies with low rating (less than 250).•group data by title and get size()•extract list of titles with ratings by title >= 250•use list to select mean ratings

TOP MOVIES LIST : Finding movies with highest disagreement of viewers independent of gender:•calculate variance and standard deviation of the ratings•filter titles with ratings equal or higher 250 (list active\_titles from earlier)•sort movies in descending order

CSE3120 MOVIELENS2:PARSE AND LOAD ratings.dat IN RDD

INVERTED INDEX: Typical operations on RDDs require grouping on a specific part of each record and then calculating specific counts given the groups. While this operation can be achieved with the groupBy family of functions, it is often useful to create a structure called an inverted index. An inverted index creates an 1..n mapping from the record part to all occurencies of the record in the dataset. Inverted indexes enable us to quickly access precalculated partitions of the dataset. Compute an inverted index on the rating field of `ratings-student.dat. Measure the time (in seconds) it takes to make this computation. Create a data frame from the ratings RDD and count the number of lines in it. Also register the data frame as an SQL table. Provide the statistical summary of the column containing ratings (use Spark function that returns a table with count, mean, stddev, min, max)

ITEM-BASED COLLABORATIVE AFTER MERGING movie and rating.csv from 20M dataset

ITEM-BASED MOVIE SUGGESTION FOR FINDING NEMO

Using the user-movie matrix, we can calculate the correlations. In user\_movie\_df the columns were the movie name, then if we fetch this column the user id-movie scores will come. This will be assigned to a variable named movie name.

ITEM-BASED COLLABORATIVE FILTERING by merging u.data and u.item from 100K dataset

Since we want the item-based collaborative filtering we will transpose the rating\_crosstab matrix.

SVD, Using Correlation Pearson find similar movies

CSE3120RATINGS\_MOVIES\_DAT Load 1M users.dat, rating.dat, movie.dat

The genre column has data with pipe separators which cannot be processed for recommendations as such. Hence, we need to generate columns for every genre type such that if the movie belongs to that genre its value will be 1 otherwise 0 (Sort of one hot encoding). Also, we need to split the release of year out of the movie\_title column and generate a new column for it. Drop genre

Similarity Scores

In this implementation the similarity between the two users will be calculated on the basis of the distance between the two users (i.e. Euclidean distances) and by calculating Pearson Correlation between the two users.

Calculating similarity scores based on the distances have an inherent problem. We do not have a threshold to decide how much distance between two users is to be considered for calculating whether the users are close enough or far enough. On the other side, this problem is resolved by pearson correlation method as it always returns a value between -1 & 1 which clearly provides us with the boundaries for closeness as we prefer.

Most Similar Users

The objective is to find out Most Similar Users to the targeted user. Here we have two metrics to find the score i.e. distance and correlation.

The output is list of tuples indicating the similarity scores of the top 5 similar number of the users asked for with user id against the targeted user. The metric used here is Pearson Correlation.

Getting Movie Recommendations for Targeted User

First, we iterate over only those movies not watched(or rated) by the targeted user and the sub-setting items based on the users highly correlated with targeted user. Here, we have used a weighted similarity approach where we have taken product of rating and score into account to make sure that the highly similar users affect the recommendations more than those less similar. Then, we have sorted the list on the basis of score along with movie ids and returned the movie titles against those movie ids.