CSE6001: BIG DATA FRAMEWORKS

J COMPONENT REPORT

FACULTY: *R. LOKESHKUMAR*

TEAM 11: *HARPAL KAUR DHINDSA (20MCB0008)*

ANKITA SUDHAKAR CHAUDHARI (20MCB0022)

TOPIC: ***BIG DATA ANALYSIS ON YELP REVIEW DATASET***

TABLE OF CONTENTS

ABSTRACT

Recommender systems are evolving day by day. A huge volume of work has been done in this area till date with the introduction of various approaches. These systems are commonly used by various departments like e-tourist, e-government, e-shopping and etc. to predict user's opinion about products. There are different recommendation methods, which focus on similar reason for prescribing things to the clients, these various strategies can be summed up into three types, first being Collaborative filtering, second is the Content-based filtering and third is Hybrid systems. In this report study five specific Collaborative filtering algorithms, namely Baseline, Item-based Collaborative Filtering algorithm, Item-Based CF recommendation system with Jaccard based LSH generated, user-based collaborative filtering and Model- Based CF recommendation system with Spark Mllib (using ALS implementation) which utilizes item similarity. We perform these five algorithms on the Yelp Dataset. Our primary dataset is the Yelp Dataset that contains actual business, user, and users' review data. In this project, Apache Spark is used to exhibit an effective parallel implementation of a collaborative filtering method using ALS.

Various metrics such as execution time, root mean squared error (RMSE) of rating estimation, and rank of the best trained model is used for model evaluation. Two best cases are selected based on the best set of criteria from experimental studies that can lead to a good prediction recommendation system. This report mainly focuses on the comparative studies of collaborative filtering methods to figure out the rating for a particular business as per the rating given by previous users.

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full form
CF	Collaborative Filtering
LSH	Locality Sensitive Hashing
ALS	Alternating Least Square
SVD	Singular Value Decomposition
RMSE	Root Mean Squared Error

LIST OF TABLES

1 INTRODUCTION

1.1 SYSTEM OVERVIEW

The Recommendation system may perhaps be an inordinate phenomenon in the universe of science, but they have been in use in the society over centuries. The first recommender system that was introduced was the Tapestry1 which was established at the Xerox Palo Alto Research Centre.

There are different recommendation methods, which emphases on similar purpose for advising things to the clients, these several strategies can be summarized in three types:

1. Collaborative filtering
2. Content-based filtering
3. Hybrid systems

One substantial personalization focus in the gathering data is content-based personal recommendation systems. These frameworks take in user's explicit profiles from user's input and recommend data custom-made to every individual user's interest without requiring the user's to make an express inquiry. Learning the user's profiles is the center issue for these sorts of Systems.

Content-Based Filtering: This system recommends centered on comparison between the descriptors of the items and a user profile.

Collaborative filtering (CF): Collaborative filtering aggregates the past behavior of all users. It recommends items to a user by linking the items liked by other set of users whose preferences are similar with respect to the user under consideration. This methodology is also called the user-user centered CF. CF based on item-item approach gained popularity later, where to advise an item to a user, the resemblance between items be fond of by the user and other items is calculated. The user-user based CF and item-item based CF can be attained in two different ways, memory-based (neighborhood approach)

and model-based (latent factor model approach).

1. The memory-based approach

Neighborhood methodologies are most operative at detecting most localized relations (neighbors), disregarding other users. But the shortcomings are that, firstly, the data gets sparse which obstructs scalability, and second, their performance is poor when it comes to reducing the RMSE (root-mean-squared-error) paralleled with other compound methods. User-based Filtering and Item-based Filtering are the two options to work with memory-based collaborative filtering.

User-based Filtering: To be able to recommend items to a user u1 by using the user-user based neighborhood method, first a set of users whose preferences align with the user u1 are found using a similarity metric which identifies the intuition that similarity(u1, u2) >similarity(u1, u3) where user u1 and u2 are alike and user u1 and u3 are unlike. Similar users are called as neighbors.

Item-based Filtering: To be able to recommend items to a user u1 by using the item-item based neighborhood method, the resemblance between items the user is fond of and other items is calculated.

2. The model-based approach

Latent factor based collaborative filtering acquires the (latent) user and item summaries (both of K dimension) over and done with matrix factorization by reducing the RMSE (Root Mean Square Error) between the accessible ratings y and their respective predicted values yˆ. Here each item i is linked with a latent vector xi, where each user u is related with a latent vector theta (u), and the rating yˆ (ui).

Latent methods convey prediction correctness superior to other available CF techniques. It also reports the sparsity issue confronted with other neighborhood models in CF. The memory efficiency and simplicity of execution by means of gradient based matrix factorization model (SVD) have made this the mode of select for the Netflix Prize competition. Nevertheless, the latent factor models are only operative at approximating the association between all items all together but is unable to identify strong relationship

among a small set of narrowly related items.

This project is built by making use of collaborative filtering techniques incorporating ALS, LSH, Minhash and using Jaccard similarity measures. The system is using the 70% of the data as the training dataset, 30% of the data as the validation dataset. This recommendation engine is for Yelp Dataset by comparing the four approaches. This project makes use of a machine learning algorithm which is imported from spark. Apache spark is a fast and wide-ranging engine used for the purpose of data processing on large data. Apache Spark is an open source cluster-computing framework. Spark arranges for an interface designing a program for the entire clusters with imbedded data parallelism and fault-tolerance. The system focuses on relating the four algorithms and validate the corresponding output. For our first algorithm we are using the spark.mllib which provisions model-based CF, in which a small set of latent factors are used to describe the users and businesses, that can be used to predict missing entries. To learn these latent factors the spark.mllib makes use of the alternating least squares (ALS) algorithm. Second algorithm is user based CF, third being item based CF and finally the Jaccard based LSH is used for item based collaborative filtering. Jaccard Similarity algorithm is used to get the similarity between two businesses or users.

1.2 OBJECTIVE

The objective is to build a recommendation system, to decreases the effect of data sparsity problem in collaborative filtering systems by using the user review as for additional information for a business.

To construct a project with a solution that allows a personalised star ratings on more features such as the similarity among different users. By using their own interests and those of related users to propose possible alternatives for them,

recommendation systems aim to make this issue simpler for users.

1.3 APPLICATIONS

The project is used to study and determine the selection of parameters that affect the performance of ALS, LSH, Minhash and using Jaccard similarity measures in building a recommender system.

This project uses Spark and because of that high-volume data is processed easily. Some of the areas where it can be applied are:

E-Government Recommender system: - Rapid development in e-government has created an explosion of information, leaving companies and people unable to make effective decisions from the variety of information they are subjected to. Increases in this information overload could obviously impede the efficacy of e-government systems, and challenges in finding the right information for the right users would increasingly have an influence on consumer loyalty. Recommended systems can solve this issue and have been implemented in e-government applications.

E-business recommender system: With the utilization of the web-accessing smartphones, it is now possible to give customised and context-sensitive recommendations to mobile users, and more mobile recommendation services will be required. However, mobile data is typically more complex, and is heterogeneous, chaotic, involves spatial and temporal auto-correlation, and has validation and general issues. Users prefer real-time, location and fine granularity recommendations in e-tourism or e-shopping applications. For example, a customer in a shopping centre needs to get advice in live time and fine granularity from stores and items depending on place and time. Real-time context-based suggestion approaches to handle these criteria.

1.4 LIMITATIONS

These recommender systems are based on large datasets i.e The Yelp dataset, and the huge datasets are the basis of many commercial recommendation programmes. As a consequence, the user-item matrix used for collaborative filtering may be incredibly broad and sparse, which presents challenges to the recommendation's efficiency. The cold start issue is one typical problem caused by the sparsity of results.

2 SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

An existing system, developed for a restaurant recommendation framework based on a novel model in this work that captures correlations between hidden aspects in feedback and numerical ratings. It is motivated by finding the preference of a consumer against an object. Different issues addressed in reviews are affected. In their paper the first approach explores topic analysis to discover hidden components from the text of the review. For users and restaurants, profiles are then generated separately based on Factors found in their reviews. Lastly, to detect the user-restaurant relationship, they have used regression models [1]. Some of the systems are build as forecasting model for customers, they have implemented the concepts and techniques of the recommender system for ratings at the restaurant. In order to classify consumers and information, they have done extraction in collaborative and content-based functionality using Yelp's dataset. In their paper yelp dataset of customers and restaurant profile were consisered. In particular, they implemented singular value decomposition, hybrid cascade of K-nearest neighbour clustering, weighted bi-partite graph projection, and several other learning algorithms. Using Root metrics Mean Squared Error and Mean Absolute Error, they then

evaluated and compare the algorithms' performances [2]. As observed from the survey, the one system is used to predict star ratings of all restaurants for each user using Advanced Collaborative Filtering. In their paper the recommendation system is build using advanced collaborative filtering with the help of 5 algorithm they have done the prediction. Firstly they have used the collaborative filtering (CF) as a baseline, after that the k nearest neighbours , K nearest neighbours with clustering (types of food) and K nearest neighbours with clustering (type of food and style) at last Singular Value Decomposition (SVD) as per their paper result the , k nearest neighbours with restaurant clustering (Type of food and style) performs best out of 5 algorithms [4].

2.2 PROPOSED SYSTEM

The project implements Model-Based, Item-Based, User-Based Collaborative Filtering, and a Matrix-Factorization method comparing the best out of the test dataset RMSE obtained values.

The Item-Based CF using Jaccard, it uses MinHashLSH to find the possible businesses candidates that may be similar to later implement a Pearson correlation to determine the respective weights. The Matrix-Factorization utilizes an Alternated Least Squares (ALS) matrix method (from PySpark Mllib).

Our System is showing the comparative study of five algorithms of collaborative filtering (CF) methods to figure out the rating for a particular business as per the rating given by previous users. This system was made using apache spark. First, we will use basic CF as our baseline. Then using Model-Based CF recommendation system with Spark MLlib (using ALS implementation), User-Based CF recommendation system, Item-Based CF recommendation system and Item-Based CF recommendation system with

Jaccard based LSH then we tried to show how each algorithm performs compared to the others. For the dataset we are using the Yelp Dataset. This project utilizes the json version which has the following files:

● business.json: Contains business data including location data, attributes, and categories.

● review.json: Contains full review text data including the user_id that wrote the review and the business_id the review is written for.

● user.json: User data including the users friend mapping and all the metadata associated with the user.

Each file is composed of a single object type, one JSON-object per-line. Description available at (https://www.yelp.com/dataset/documentation/json) As the focus of this project is on building a recommendation engine, the core files will be used are business.json, review.json, and user.json. Although the majority attention is paid to review.json. And for the evaluation metrics for the recommender system we are using the root mean squared error (RMSE). RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

The five algorithms that the system is working on are:

1) Baseline recommendation System: - As our Baseline recommendation system we predicted the ranking for each analysis in the dataset, and calculated the baseline RMSE for them.

2) Item-based Collaborative Filtering: -The Item-based collaborative filtering technique contrasts a certain range of objects with users and their reviews. The algorithm determines a given users closest neighbours. With an object

similarity measure, and neighbours previously rated items are compared to the item in view. The forecast is estimated by taking the weighted average of each neighbour's ratings of those items when the most comparable items have been determined. In this system we have included the Pearson correlation for the calculation of item similarities can be done.

3) User-based collaborative Filtering: - User-based collaborative recommendation aims to calculate some similarity metric between all pairs of users and then predict a particular user u's rating for an item i by collecting and processing the ratings of u's "neighbourhood" (all the other users with high similarity as compared to u) for i.

4) Item-Based CF recommendation system with Jaccard based LSH generated: - Instead of using items; characteristics to determine their resemblance, we concentrate on the elements. The similarity of the ratings of users for two things. In lieu of the item-profile, that is, we use the vector for an object in the utility matrix, using its column. Next, instead of building a profile vector for users, we represent them via their rows in the Matrix of Utility. Users are identical if, according to others, their vectors are closer measurement of distance such as Jaccard or Cosine Distance. In this we are using the Jaccard similarity measure and it mainly focuses on global ratings. It is the ratio of the proportion of the cardinality of co-rated items to the cardinality of all items rated by both the users.

5) Model- Based CF recommendation system with Spark MLlib (using ALS implementation): - This algorithm using the Spark MLlib. It supports the model- based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. Spark MLlib is implementing using alternating least squares (ALS) algorithm. ALS is

the most imperative techniques which are used for building a movie recommendation engine. The ALS algorithm is one of the models of matrix factorization related CF which is considered as the values in the item list of user matrix.

2.2.1 Benefits of Proposed System

As the system is first implementing the Locality Sensitive Hashing algorithm with Jaccard similarity. In the matter of the yelp data in our system it's beneficial to use LSH because, LSH will help to reduce the dimensionality of high-dimensional data. Also, we are using the Jaccard similarity as it is simple in nature and user-friendly in implementation, is applied in various domain and performed adequately. The Jaccard coefficient measures the similarity between the finite sample sets. This system is using Apache spark so for fast and large-scale data processing it's helpful. This system is giving the comparative study and helping to validate the algorithm which is better.

3 REQUIREMENT SPECIFICATION

3.1 HARDWARE REQUIREMENTS

OS: Windows 10, 64-bit

Processor: Intel® Core™ i7

Memory: 8 GB

3.2 SOFTWARE REQUIREMENTS

Python: 3.9

Spark: 3.0.1

spark.driver.memory: 4g

spark.executor.memory: 4g

4 SYSTEM DESIGN SPECIFICATION

4.1 SYSTEM ARCHITECTURE

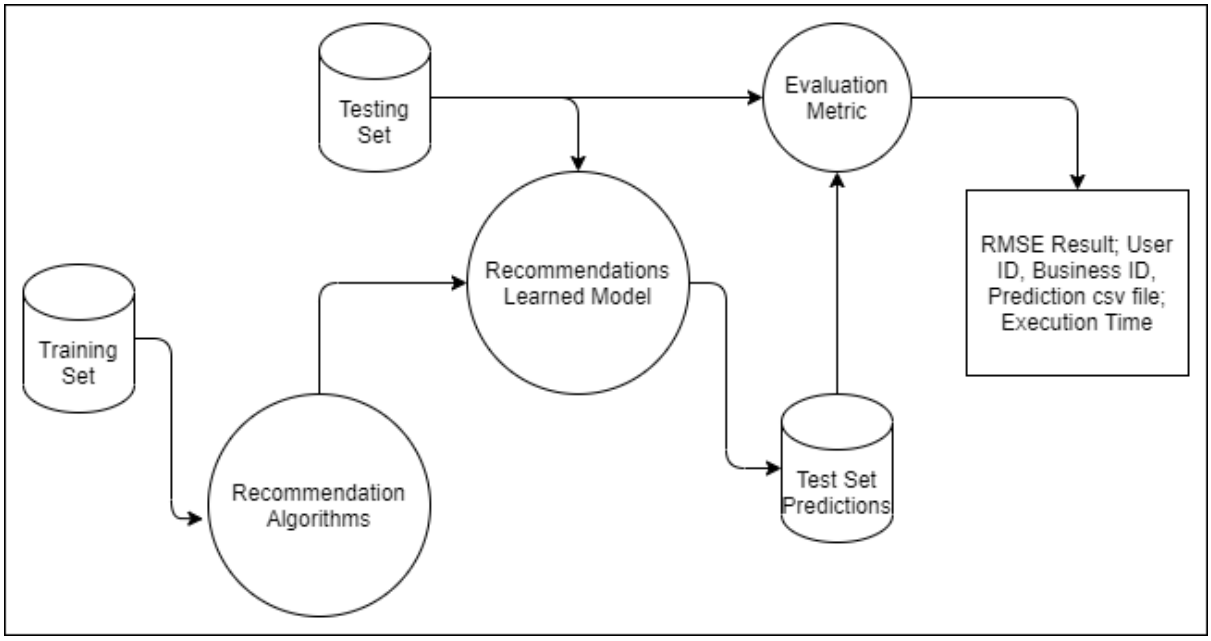

Figure 1. System Architecture

The Yelp_acedemic_dataset_review.json file is used to generate the following two datasets with some filters such as the condition: “state”== “CA”, the training and testing dataset. Generated the following two datasets from the original Yelp review dataset with some filters such as the condition: “state” == “CA”. We randomly took 70% of the data as the training dataset, 30% of the data as the validation dataset.

a. yelp_train.csv: the training data, which only include the columns: user_id, business_id, and stars.

b. yelp_val.csv: the validation data, which are in the same format as training data

4.2 DETAILED DESIGN

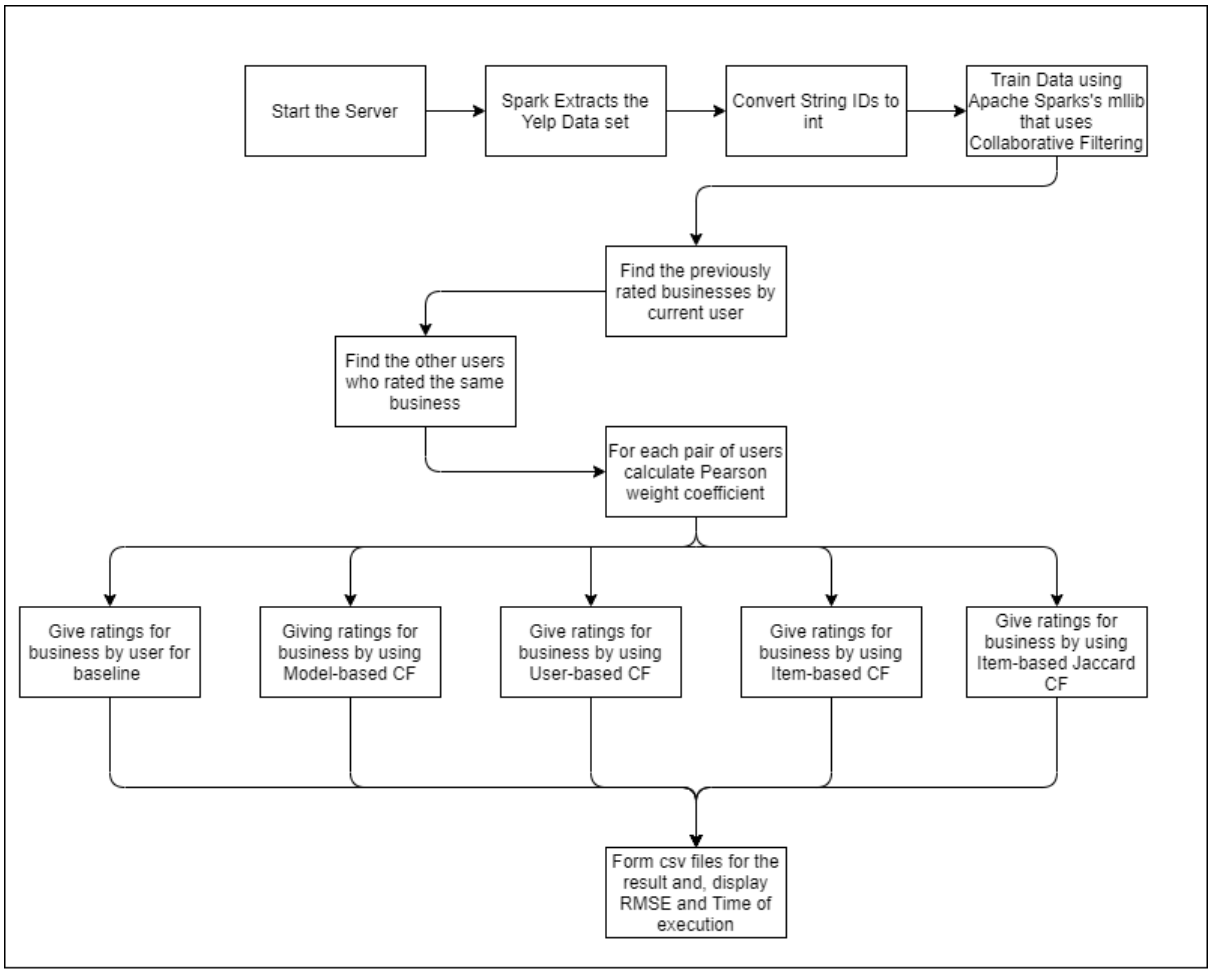

Figure 2. System detailed design

Terminologies:

There are certain terminologies which needs to be understood before moving forward.

Apache Spark: Apache Spark is an open-source distributed general-purpose cluster-computing framework. It can be used with Hadoop too.

Collaborative filtering: Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users. Consider 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.

Alternating least square (ALS) matrix factorization: The idea is basically to

take a large (or potentially huge) matrix and factor it into some smaller representation of the original matrix through alternating least squares. We end up with two or more lower dimensional matrices whose product equals the original one. ALS comes inbuilt in Apache Spark. Recommendation using Alternating Least Squares (ALS), Alternating Least Squares (ALS) matrix factorization attempts to estimate the ratings matrix R as the product of two lower-rank matrices, X and Y, i.e. X * Yt = R. Typically these approximations are called 'factor' matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix. In the below section we will instantiate an ALS model, run hyperparameter tuning, cross validation and fit the model.

LSH: Locality Sensitive Hashing (LSH) is a randomized algorithm for solving Near Neighbor Search problem in high dimensional spaces. LSH has many applications in the areas such as machine learning and information retrieval. In this talk, we will discuss why and how we use LSH at Uber. Then, we will dive deep into the technical details of our LSH implementation. Our LSH library is designed and implemented to optimize the performance on Spark. It supports pluggable distance functions. Out of the box, Jaccard, Cosine, Hamming and Euclidean distance functions are included in the library. It also supports approximate near neighbor searches and self-similarity joins. In the talk, we will also share performance benchmark and our experience of running LSH on Spark in production clusters.

MinHash for Jaccard Distance: MinHash is an LSH family for Jaccard distance where input features are sets of natural numbers. Jaccard distance of two sets is defined by the cardinality of their intersection and union: $d(A,B)=1-|A\cap B||A\cup B|$

MinHash applies a random hash function g to each element in the set and take the minimum of all hashed values: $h(A)=\min_{a\in A}(g(a))$.

The input sets for MinHash are represented as binary vectors, where the vector indices represent the elements themselves and the non-zero values in the vector

represent the presence of that element in the set. While both dense and sparse vectors are supported, typically sparse vectors are recommended for efficiency. For example, Vectors. Sparse (10, Array [(2, 1.0), (3, 1.0), (5, 1.0)]) means there are 10 elements in the space. This set contains elem 2, elem 3 and elem 5. All non-zero values are treated as binary "1" values.

PySpark: PySpark is the collaboration of Apache Spark and Python. PySpark is the Python API for Spark.

4.3 DATABASE DESCRIPTION

Dataset: Yelp provides a portion of its data through Yelp Dataset Challenge event. The dataset includes 42,153 businesses, 252,898 users, and 1,125,458 reviews, which include star rating in the range of 1 to 5 and user's opinions in text. This dataset includes businesses, users and their respective ratings.

5 SYSTEM IMPLEMENTATION

In this project, we use several collaborative filtering (CF) methods to figure out the rating for a particular business as per the rating given by previous users. First, we will use basic CF as our baseline. Then using LSH and ALS, we will show how each algorithm performs compared to the others.

5.1 MODULE DESCRIPTION

Task 1: Implemented the Locality Sensitive Hashing algorithm with Jaccard similarity using yelp_train.csv with focus on the "0 or 1" ratings rather than the actual ratings/stars from the users. Specifically, if a user has rated a business, the user's contribution in the characteristic matrix is "1" If the user hasn't rated the business, the contribution is "0". Identified similar businesses whose similarity >=0.5. The generated results are compared to the ground truth file pure_jaccard_similarity.csv.

Task 2:

This implementation generates recommendation using:

Case 1: Baseline recommendation System

For baseline recommendation we consider the original Yelp review, user and business json files, and calculate the baseline RMSE for them. We convert the business ID and User ID from string to int type. Building the recommendation model using ALS on the training data. Evaluating the data by computing the RMSE on the test data. The prediction is a dataframe consisting of the user ID with the new ratings.

Case 2: Model- Based CF recommendation system with Spark MLlib (using ALS implementation)

Building ALS model using user ratings from train data. Generating predictions from trained ALS. Generating predictions from trained ALS model for ratings using test data. Normalize the predicted ratings for improving RMSE, using min-max normalization strategy.

Case 3: User-Based CF recommendation system

We calculate the weights for each user with other users using Pearson Correlation Coefficient. In case no other user rated the particular item, we take the average rating per user. Next, we find the previously rated businesses by current user. Then we find other users who rated the same business. For each pair of user 1, user 2 calculate Pearson weight Coefficient of co-rated items.

Case 4: Item-Based CF recommendation system

We calculate the weights for each user with other users using Pearson Correlation Coefficient. In case no other user rated the particular item, we take the average rating per business. Next, we find the previously rated businesses by current user. Then we find other businesses which are

rated, from the same business. For each pair of business 1, business 2 calculate Pearson weight Coefficient of co-rated items. Then we get the current rating for current business, and calculate the weighted average. Hence, we find the predicted current rate for the user- business.

Case 5: Item-Based CF recommendation system with Jaccard based LSH generated

Here we have calculated the similarity between the items by using the Jaccard similarity coefficient, minHash estimates coefficient quickly, without explicitly calculating intersection and union. The hash function is used, the locality sensitive hashing, hashes similar inputs into the same "buckets" with high probability. Here the relative distance between the input values is preserved in the relative distance between the output hash values. The Jaccard is used to from similar businesses pairs. Then we calculate the weights for each user with other users using Pearson Correlation Coefficient. In case no other user rated the particular item, we take the average rating per business. Next, we find the previously rated businesses by current user. Then we find other businesses which are rated, from the same business. For each pair of business 1, business 2 calculate Pearson weight Coefficient of co-rated items. Then we get the current rating for current business, and calculate the weighted average. Hence, we find the predicted current rate for the user- business pair.

Methodology	**RMSE**
Baseline	2.200
Model-Based CF	1.1281
User-Based CF	0.9363

Item-Based CF	0.9733
Item-Based CF using Jaccard	0.9733

Table 1. Result Table

6 CONCLUSION AND FUTURE ENHANCEMENTS

There a slight improvement noticed in the accuracy of predictions after incorporating LSH. In case, where there were no previous reviews for the business, it is difficult to find the other businesses similar to the given one as no-user has rated it yet- cold start problem. In this case using the "0/1" Jaccard similarity and LSH to find similar items provides useful information about businesses similar to current one. These set of businesses augmented the set of businesses generated from item-based CF, hence there were more items contributing to the predicted rating. Thus, accuracy improved. Speed of prediction doesn't have a noticeable change.

This project is a very simple implementation of recommendation system. It can be further enhanced with more features like preferences choices, recommendation taking into consideration the user’s location, review analysis for positivity and negativity, analysis on a business’s expansion possibility, and predicting business growth. Recommendation engines provide a great deal of information for business growth and is very profitable solution for many organizations.

7 APPENDICES

7.1 APPENDIX 1 - SAMPLE SOURCE CODE

Baseline.py

```
from pyspark.python.pyspark.shell import spark
from pyspark.sql.functions import col
import json
import math

def dataClean(testDF, usrDF, busDF):
    # load test df
    testDF = spark.read.json(test_path)
    testDF = testDF.select(col('business_id').alias('bus_id'),
                           col('user_id').alias('usr_id'),
                           col('stars').alias('label'))

    # load usr df
    usrDF = spark.read.json(usr_path)
    usrDF = usrDF.select(col('user_id').alias('usr_id'),
                         col('review_count').alias('urew_no'),
                         col('average_stars').alias('uavg_stars'))
    usrDF = usrDF.select('usr_id', 'urew_no',
(col('urew_no')*col('uavg_stars')).alias('usrtemp'))

    # load bus df
    busDF = spark.read.json(bus_path)
    busDF = busDF.select(col('business_id').alias('bus_id'),
                         col('stars').alias('bavg_stars'),
                         col('review_count').alias('brew_no'))
    busDF = busDF.select('bus_id', 'brew_no',
```

```
(col('brew_no')*col('bavg_stars')).alias('bustemp'))

    return testDF, usrDF, busDF



def calculateRMSE(testDF, usrDF, busDF):
    # real calculation
    baseDF = testDF.join(usrDF, testDF.usr_id ==
usrDF.usr_id).drop(usrDF.usr_id) \
            .join(busDF, testDF.bus_id == busDF.bus_id).drop(busDF.bus_id)

    baseDF = baseDF.select('*',

((col('usrtemp')+col('bustemp'))/(col('urew_no')+col('brew_no'))).alias('baserati
ng'))
    rmseDF = baseDF.select('label',
                        'baserating',
                        ((col('label') - col('baserating'))**2).alias('mes'))
    errors = rmseDF.rdd.map(lambda x: x.mes).collect()
    RMSE = math.sqrt(sum(errors)/(len(errors)+0.0001))
    return baseDF,RMSE



if __name__ == '__main__':
    #initial some path
    test_path = '/D:/Documents/harpal/CSE6001-
```

```
Project/yelp_academic_dataset_review.json'
    usr_path = '/D:/Documents/harpal/CSE6001-
Project/yelp_academic_dataset_user.json'
    bus_path = '/D:/Documents/harpal/CSE6001-
Project/yelp_academic_dataset_business.json'
    #json_path = './ratings.json'
    training_fraction = 0.7
    test_fraction = 0.3

    testDF,usrDF,busDF = dataClean('test_path', 'usr_path','bus_path')
    baseDF, RMSE = calculateRMSE(testDF,usrDF,busDF)
    print('Baseline RMSE: %.8f' % RMSE)
```

Recommendation.py

```
import json
from pyspark.context import SparkContext
sc = SparkContext()
from pyspark.python.pyspark.shell import spark
from pyspark.sql.functions import col,rank

import pyspark.sql.functions
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
from pyspark.sql.window import Window
#from pyspark.sql.functions import rank, col
```

```
from pyspark.sql import functions as F


def GetRecomList(predictions, partition_by, order_by, rank_num):

    # predictions is a dataframe which contain id and predict result(score).
    # partition_by is a name of column in predictions which identify the key to partition by
    # order_by is a name of column in predictions which identify the score we need to sort
    # rank_num specify how many records you want to return for each partition

    window = Window \
        .partitionBy(predictions[partition_by])\
        .orderBy(predictions[order_by].desc())

    recomlistdDF = predictions.select(partition_by, order_by, rank().over(window).alias('rank')) \
                        .filter(col('rank') <= rank_num)

    print("Get num of review list: ", recomlistdDF.count())

    return recomlistdDF


def covtDataFormat(ratings, sc):
```

```
# convert the bus_id and usr_id from string type to int type

usr_list = ratings.select('usr_id').distinct()\
                .rdd.map(lambda x: x.usr_id).collect()
bus_list = ratings.select('bus_id').distinct()\
                .rdd.map(lambda x: x.bus_id).collect()

usr = {}
bus = {}
i = 0
for ele in usr_list:
    usr[ele] = i
    i = i + 1

j = 0
for ele in bus_list:
    bus[ele] = j
    j = j + 1

usrnoDF = sc.parallelize([k, v] for k, v in usr.items()) \
            .toDF(['usr_id', 'usr_no'])
usrnoDF.show()

busnoDF = sc.parallelize([k, v] for k, v in bus.items()) \
            .toDF(['bus_id', 'bus_no'])
busnoDF.show()
```

```
    recomDF = ratings.join(usrnoDF, ratings.usr_id ==
usrnoDF.usr_id).drop(usrnoDF.usr_id) \
        .join(busnoDF, ratings.bus_id == busnoDF.bus_id).drop(busnoDF.bus_id)
    recomDF.show()
    recomDF = recomDF.select('usr_no', 'bus_no', 'label')
    return recomDF


def recommsys(training, test, r, it, para):
    # Build the recommendation model using ALS on the training data
    als = ALS(rank=r, maxIter=it, regParam=para, userCol="usr_no",
              itemCol="bus_no", ratingCol="label")
    # data = df.union(df)
    model = als.fit(training)
    predictions = model.transform(test).na.drop()


    predictions = predictions.select('*', F.when(predictions.prediction > 5, 5)
                                     .otherwise(predictions.prediction).alias('temp'))
    predictions = predictions.select('*', F.when(predictions.temp < 1, 1)
                                     .otherwise(predictions.temp).alias('newrating'))

    evaluator = RegressionEvaluator(
        metricName="rmse", labelCol="label", predictionCol="newrating")
    rmse = evaluator.evaluate(predictions)
```

```
    return rmse


if __name__ == '__main__':
    test_path = '/D:/Documents/harpal/CSE6001-
Project/yelp_academic_dataset_review.json'
    testDF = spark.read.json(test_path)
    training_fraction = 0.7
    test_fraction = 0.3

    ratings = testDF.select(col('business_id').alias('bus_id'), col(
        'user_id').alias('usr_id'), col('stars').alias('label'))

    recomDF = covtDataFormat(ratings,sc)
    training, test = recomDF.randomSplit([training_fraction, test_fraction])

    rmse = recommsys(training, test, 10, 5, 0.01)
    print("Root-mean-square error by = " + str(rmse))
```

Task1.py

```
from pyspark import SparkContext, StorageLevel
import sys
import json
import csv
import itertools
from time import time
```

```
import math
import random

if len(sys.argv) != 3:
    print("Usage: ./bin/spark-submit Recommender_task1.py <input_file_path>
<output_file_path>")
    exit(-1)
else:
    input_file_path = sys.argv[1]
    output_file_path = sys.argv[2]

def process(entry):
    revisedEntries= entry[0].split(',')
    return (revisedEntries[0], revisedEntries[1], revisedEntries[2])

def convertValuesToTuple(entrySet):
    newEntrySet = []
    for entry in entrySet:
        newEntrySet += [(entry, 1)]
    return newEntrySet


def generate_minhash_Array(users,num_hashes, num_users):
    users = list(users)
    system_max_value = sys.maxsize
    hashed_users = [system_max_value for i in range(0,num_hashes)]
```

```
    for user in users:
        for i in range(1, num_hashes+1):
            current_hash_code = ((i*user)+ (5*i*13)) % num_users

            if current_hash_code < hashed_users[i-1]:
                hashed_users[i-1] = current_hash_code
    return hashed_users

def applyLSHToSignature(business_id, signatures, n_bands, n_rows):
    signature_tuples = []
    for band in range(0,n_bands):
        band_name = band
        final_signature = signatures[band*n_rows:(band*n_rows)+n_rows]
        final_signature.insert(0, band_name)
        signature_tuple = (tuple(final_signature), business_id)
        signature_tuples.append(signature_tuple)

    return signature_tuples

def generate_similar_businesses(businesses):

    b_length = len(businesses)

    similar_businesses = []
    similar_businesses =
```

```
sorted(list(itertools.combinations(sorted(businesses),2)))

    return similar_businesses


def calculate_jaccard(candidate, business_char_matrix):

    users_c1 = set(business_char_matrix.get(candidate[0]))
    users_c2 = set(business_char_matrix.get(candidate[1]))

    jaccard_intersection = len(users_c1.intersection(users_c2))
    jaccard_union = len(users_c1.union(users_c2))

    jaccard_similarity_value = float(jaccard_intersection)/float(jaccard_union)

    return (candidate, jaccard_similarity_value)



def isPrime(num):
    if num==2 or num==3:
        return True
    if num%2==0 or num<2:
        return False
    for i in range(3, int(num ** 0.5) + 1, 2):
        if num%i==0:
```

```
            return False

    return True


def generateRandomHashCoefficients(num_hashes, num_users):
    random_a =[]
    random_b = []
    random_coeffs = {}

    while num_hashes > 0:
        random_num= random.randint(0,num_users)
        while random_num in random_a:
            random_num = random.randint(0,num_users)
        random_a.append(random_num)

        while random_num in random_a or random_num in random_b:
            random_num = random.randint(0,num_users)
        random_b.append(random_num)

    random_coeffs.update({'a':random_a})
    random_coeffs.update({'b':random_b})

    return random_coeffs

result = []
```

```
SparkContext.setSystemProperty('spark.executor.memory', '4g')
SparkContext.setSystemProperty('spark.driver.memory', '4g')
sc = SparkContext('local[*]', 'task1')

#input_file_path = "/Input/yelp_train.csv"
#output_file_path = "/Input/task1__result.csv"
jaccard_support = 0.5
num_hashes = 80
n_bands = 40
n_rows = 2

start = time()
user_businessRdd = sc.textFile(input_file_path).map(lambda entry:
entry.split('\n')).map(lambda entry: process(entry))
headers = user_businessRdd.take(1)
finalRdd = user_businessRdd.filter(lambda entry: entry[0] !=
headers[0][0]).persist()

users = finalRdd.map(lambda entry: entry[0]).distinct()
num_users = users.count()

user_index_dict = finalRdd.map(lambda entry:
entry[0]).zipWithIndex().collectAsMap()

business_user_map = finalRdd\
    .map(lambda entry: (entry[1], user_index_dict.get(entry[0])))\
```

```
    .groupByKey()\
    .sortByKey()\
    .mapValues(lambda entry: set(entry))\
    .persist()

business_user_map_values = business_user_map.collect()
business_char_matrix = {}

for bu in business_user_map_values:
    business_char_matrix.update({bu[0]:bu[1]})

candidates = business_user_map\
    .mapValues(lambda users: generate_minhash_Array(users, num_hashes,
num_users))\
    .flatMap(lambda entry: applyLSHToSignature(entry[0], list(entry[1]),
n_bands, n_rows))\
    .groupByKey()\
    .filter(lambda entry: len(list(entry[1])) > 1)\
    .flatMap(lambda entry: generate_similar_businesses(sorted(list(entry[1]))))\
    .distinct()\
    .persist()

final_pairs = candidates\
    .map(lambda cd: calculate_jaccard(cd, business_char_matrix))\
    .filter(lambda cd: cd[1] >= jaccard_support)\
    .sortByKey()
```

```
result = final_pairs.collect()

with open(output_file_path, "w+", encoding="utf-8") as fp:
    fp.write("business_id_1,business_id_2,similarity")
    fp.write('\n')
    fp.write('\n'.join('{},{},{}'.format(x[0][0], x[0][1], x[1]) for x in result))

end = time()
print("Count: ", len(result))
print("Duration: " + str(end-start))
```

7.2 APPENDIX 2 - SCREEN SHOTS /OUTPUTs

Task 1 output:

```
business_id_1,business_id_2,similarity
-InU2nAbC9AuS-Um2Cowgw,ZvEJiX8HBIForfbHOfqpmQ,0.5
-OD5KnGJk4Ld8IVa2jAuFA,TlFowSzpX0k-urEBJlT8UA,0.5
0PenhUhRbw2xd_suA68IjA,yqW2lcZ0vb2cqZSPjsFX5w,0.5
0fnIjQK6GfMfVFITNFQDTA,pXaCLoK4sH6V4IM3Pc5ofw,0.5
0l_HQpZ4gsR5T6Ejqcgi2Q,MH_8eiwDpmSAs59FUAB35A,0.5
1K2D2wsxprU54FpkeDZTiQ,p7vxpHPxOQi1Qr5cDoEpQg,0.5
24AA0NJBNS2jMlz6klJ9AA,HodEoYNU4qHzUD2MESDWhw,0.5
2CabGE-ZFsKrw5Ke_mNsaw,lUM741nAWb4WL-pGiqo7Bw,0.5
2Ib4m3BtCEMjghw7D8fFtQ,URRDNLJg4tnQYDvJDVxCVQ,0.5
2IqODXx1v_CwIKT57Xvp6Q,Ma0lzQhJMMQoFJwkJJurWg,1.0
2TFmNewkeZVUXXrHjfHPAQ,XehBm4wtQ6qeeYj_eL-6sw,0.5
2XUJM063C50TLfK-s902cQ,N92Pbr2ygKDLkjmr-4BAPw,0.5
2rrHghWVrJUYrq0I_zLktA,7fa7SPQb9_DbKCtlmRB5GA,0.5
30fH0eHUXorWSR2-C4PuiA,QqVfWRqOO_exljN8IWgeLQ,0.5
3P37jROxsfYz-en2dtCp-w,dVYsMYqnGpPHp0dTGHTUWQ,0.5
3bpMCRbjDe-rNvHAXYvUCw,QnW7oSlVjx-XYrwfO06LBQ,0.5
3vUUOXTq1HaevQiuaWY8jQ,_KyPaRV9LihhWKqylx2Keg,0.5
4TR2P1293f86lqRPV8OF5A,QuytwC8LramTP7QJ-SxFsA,0.5
5_5AT_uw_akxMIu_8LLhew,SH8KerPNCMzYGzEB8fml9A,0.5
5s_f9DAynHW9rSwfOXfUWA,61ZG2kyj_R7iux7yPrJIxg,1.0
5s_f9DAynHW9rSwfOXfUWA,Gw5e_DozTLjO-vz-LXD9og,0.5
61ZG2kyj_R7iux7yPrJIxg,Gw5e_DozTLjO-vz-LXD9og,0.5
6H0kVbkAGlXKDlB4lehgzw,jrGxdAblwBmRG6K-dDpKbQ,0.5
6jsuYyjUM45WAX-6SxOQgQ,7B-suS0RHxqxZBBfVfiPLA,0.5
6yynJxXqG8x9k9wrSvLAEw,ZZ5KRLz9TuNs66Y2KYsxRg,1.0
7siEeRGqAJ730IkwAtcWAQ,lrrUyxzlvqp-niBEj6_6dg,0.5714285714285714
8D2R7acumiJgJXJ5NMF4GA,CAOnmH06liJa74jkK-uZ3A,0.6666666666666666
8TBDAWJCxI9O4S5ik0W3GA,tlBguPOE_NQHoQO11mK5tQ,0.5
8m1HM54_hE6UcQcxWTWMaA,WhYQVBZWT-6bkg7xsoAebg,0.5
944637VhfxWhdJYty63LkA,jei_nG0uhrvTSXEFL2K9bw,0.5
9BOGpsIRhs9cRHuGh9xuJw,l_kfIrvIvF_hcOsVxtmGMA,0.5714285714285714
```

Task 2 outputs:

Case 1:

```
+--------------------+------+
|              usr_id|usr_no|
+--------------------+------+
|KWFiZKiZBANVxuhm4...|     0|
|VmYpF5C3GL-7wFnvO...|     1|
|1Dul59QEe-Q-7OQHT...|     2|
|xS6kmkMXp0PRrFwkS...|     3|
|j56G3m8vYtA_2Io6F...|     4|
|ruHz-qN-j21kg0iyI...|     5|
|M6-A6F0B3kM5i94Kr...|     6|
|z2Gi5vo-8j544qN_g...|     7|
|CzkWUMIYDxUSetfCR...|     8|
|Uf_TVv1Z4s024jdI4...|     9|
|zWWcik1fRPZviBCQL...|    10|
|zEpzcYlc1wQ4YJtFF...|    11|
|4ZfcCa4m5RWvO4EFz...|    12|
|QSWr7Oivp0mClj_PR...|    13|
|FuSn5ZxN2NV_PpK03...|    14|
|T_ReHc3TRnOw9h1qf...|    15|
|7o473jeLWW-zgKN-Q...|    16|
|ucpIv1E0x8IhxeLGD...|    17|
|ouvhv-E3S57SQ4ser...|    18|
|kAxgIZA-kSCKDy6EP...|    19|
+--------------------+------+
only showing top 20 rows
```

```
+--------------------+------+
|              bus_id|bus_no|
+--------------------+------+
|ATeOjZzE3VOI-RW7F...|     0|
|RMjCnixEY5i12Ciqn...|     1|
|vbUABNAQI2iwN7v0x...|     2|
|VHsNB3pdGVcRgs6C3...|     3|
|r-W1HPIZ3V6S2DaXh...|     4|
|cKwg6HFaLYXl7Ar0r...|     5|
|4lCTd6-EzOuKl4YbH...|     6|
|4x8W4quFhhuTZlsoU...|     7|
|V_maCS_uBRMjqa_BC...|     8|
|_Wyo52ijeq3QgcmkI...|     9|
|x1fx7C_tc064cFATj...|    10|
|aiAYA0SKz-wRDj04g...|    11|
|kpbhERZoj1eTDRnMV...|    12|
|ipFreSFhjClfNETuM...|    13|
|juNKWhkMynLaIHK4Y...|    14|
|TdefcbsFAj6WXHwlG...|    15|
|pK83jG-vw6UykrHkw...|    16|
|TIaaQKCaJaWw0Q9m6...|    17|
|YYztMMdcudL4xuSF2...|    18|
|4v6e_afy2uAiQYDq9...|    19|
+--------------------+------+
only showing top 20 rows
```

bus_id	usr_id	label	usr_no	bus_no
--9e1ONYQuAa-CB_R...	0y8ORuC2X1i1UF6SG...	5.0	4812	596
--9e1ONYQuAa-CB_R...	3qz_dfwbFwTQeDRzy...	5.0	917	596
--9e1ONYQuAa-CB_R...	9spixZHaqC1JeN1ld...	2.0	315	596
--9e1ONYQuAa-CB_R...	A4GnBOU7ZCTcoQK4e...	5.0	7867	596
--9e1ONYQuAa-CB_R...	FtUDjNLhVjlIoeFKm...	4.0	483	596
--9e1ONYQuAa-CB_R...	H0tfWQsGjEBuhXD4W...	5.0	151	596
--9e1ONYQuAa-CB_R...	R0KVWeN9xR-F6j4z5...	4.0	155	596
--9e1ONYQuAa-CB_R...	XZaCs-Gs0SXdZgfG3...	4.0	3069	596
--9e1ONYQuAa-CB_R...	dSGINC_8KV6fxNjeQ...	5.0	7920	596
--9e1ONYQuAa-CB_R...	n9DJHwgYflQ_ms8gB...	3.0	796	596
--9e1ONYQuAa-CB_R...	ucXjnxiEKLUOEktHF...	5.0	720	596
--9e1ONYQuAa-CB_R...	xOSVPzpIDSd3-2r6k...	5.0	7872	596
--9e1ONYQuAa-CB_R...	-7hnKm0I8detrDCgs...	5.0	17575	596
--9e1ONYQuAa-CB_R...	1rlB-SWvDU5TnDnym...	5.0	10302	596
--9e1ONYQuAa-CB_R...	KGcyC9KXloxW_6YMG...	4.0	11121	596
--9e1ONYQuAa-CB_R...	MFaazTdvfJ_aa6coa...	5.0	10948	596
--9e1ONYQuAa-CB_R...	jHHRH62tCYNZnh85u...	4.0	10472	596
--9e1ONYQuAa-CB_R...	oqTZC9WriodTCby6j...	3.0	10186	596
--9e1ONYQuAa-CB_R...	5g9q0NUQ-72wzitIz...	2.0	27759	596
--9e1ONYQuAa-CB_R...	9C5QkwIcxyE4SgR1E...	2.0	27576	596

only showing top 20 rows

Root-mean-square error by = 2.2008925906326287

Case 2:

```
RMSE:  1.128164044305417
Duration:  128.76845741271973
```

```
user_id,business_id,prediction
ufc6SMa1EA_VOP3Gmmq3gQ,yVhEA59Q4vcfQ6z7mIqysg,3.1199775988484575
sVn_rNopTpPik0iL_Xm7ag,yVhEA59Q4vcfQ6z7mIqysg,2.291289563245165
s6ArzWxkfV7uRURej7AoQA,yVhEA59Q4vcfQ6z7mIqysg,2.8747945108641417
3sY-Wj7CKDb9YjQm7LG5Ew,yVhEA59Q4vcfQ6z7mIqysg,2.8192919756017076
u8HPA-jU1njsPPAI_anMgA,yVhEA59Q4vcfQ6z7mIqysg,2.7022708403267757
N-9VZPg1XI2vQvL3Fbw-Kg,yVhEA59Q4vcfQ6z7mIqysg,2.736332736032452
cQfcLsUFH-txdue1Z7mxbg,yVhEA59Q4vcfQ6z7mIqysg,2.0039675050763464
1zrizQcPPLu6Fnfy0_ZLdQ,yVhEA59Q4vcfQ6z7mIqysg,2.9886921317501915
nwTRwJZFkqtDEdxunzpZDQ,_Y0SbKocq7yoV8f5_fSyaA,1.6069832537552102
GGTF7hnQi6D5W77_qiKlqg,_Y0SbKocq7yoV8f5_fSyaA,2.901195204777821
lktq5goeWtRJG13Jh1NOLw,_Y0SbKocq7yoV8f5_fSyaA,2.367667903418422
62GNFh5FySkA3MbrQmnqvg,_Y0SbKocq7yoV8f5_fSyaA,2.4442392774041934
6VGY4RRd-najo6dQhL3mdw,G9-OvE0PBQtDZmnGEB3HEQ,3.318919524422384
FjIu5bf6dPW7_V_-0Jg4kQ,G9-OvE0PBQtDZmnGEB3HEQ,3.1797743338772477
Zk7XnX8tVvJncQKa-9298A,G9-OvE0PBQtDZmnGEB3HEQ,3.3055094274002386
6J8koMq6k0GM437QWMHRSw,G9-OvE0PBQtDZmnGEB3HEQ,2.4041959261318944
I-4KVZ9lqHhk8469X9FvhA,G9-OvE0PBQtDZmnGEB3HEQ,4.017927323845484
3hWJxx8-gMutgaKY2T7IKQ,G9-OvE0PBQtDZmnGEB3HEQ,2.9030550897536855
j2b3L64z8OAwn5EqwDRQaw,G9-OvE0PBQtDZmnGEB3HEQ,3.65682253472022
0E2Q9mJsMdQ0hRkKTVWuag,G9-OvE0PBQtDZmnGEB3HEQ,2.703228917316867
5I6-yegWr4p1mtUMwNkVtg,G9-OvE0PBQtDZmnGEB3HEQ,2.706085307298494
q9jCDNZDRUYl2Pwlgxocig,G9-OvE0PBQtDZmnGEB3HEQ,3.3392265522376903
Oz8Bp1NY-Szz4lvSbVsV4Q,G9-OvE0PBQtDZmnGEB3HEQ,3.3992954817285863
CqeLXgQUpMZNbz4GwBz87w,G9-OvE0PBQtDZmnGEB3HEQ,3.179033527309926
GLPlsxMjClSUqikpXVnZ1w,G9-OvE0PBQtDZmnGEB3HEQ,3.652662493515905
q2b4pAo_6tP7jNFKI5Hztw,G9-OvE0PBQtDZmnGEB3HEQ,3.2654210125686554
-HAHTQeOzHX5TCDb7qs9Yw,G9-OvE0PBQtDZmnGEB3HEQ,3.5586824211171972
9ks-80ZxeCZWNObzEbW4-Q,G9-OvE0PBQtDZmnGEB3HEQ,3.132249107057902
1Co-zYolv5AmjKlsISMepQ,G9-OvE0PBQtDZmnGEB3HEQ,2.970544301957432
q5afJ8gTV5TPEOkzyeJ_WQ,G9-OvE0PBQtDZmnGEB3HEQ,3.4644350801104125
hd-S4mtrPY9O6iuBJeqnOO,G9-OvE0PBQtDZmnGEB3HEQ,3.464195870546945
```

Case 3:

```
RMSE:  0.9363732773942004
Duration:  692.1520166397095
```

```
user_id,business_id,prediction
---1lKK3aKOuomHnwAkAow,MQiNywdecInMlkW06WYaCg,3.478304793394594
---1lKK3aKOuomHnwAkAow,RRw9I8pHt5PzgYGT2QeODw,4.08309080409543
---1lKK3aKOuomHnwAkAow,ttH3ZbUcncBRIXqT-YVPCg,4.605284740424546
---1lKK3aKOuomHnwAkAow,GGCVNcBQ9WGviYNiaR8tBw,3.591988749755206
---1lKK3aKOuomHnwAkAow,AxeQEz3-s9_1TyIo-G7UQw,3.9584225998522813
---1lKK3aKOuomHnwAkAow,E3m-twP4h0-qzakUTBYDpw,3.7393979439928096
---1lKK3aKOuomHnwAkAow,CeqWpwHBoaxwRcv5btnv6g,4.438192616079929
---1lKK3aKOuomHnwAkAow,5FPQOwwPkBEiy8df8d0SPQ,4.913607246567635
---1lKK3aKOuomHnwAkAow,TscyJToVcStsohgRG9qM7g,4.430122794158708
---1lKK3aKOuomHnwAkAow,yp2nRId4v-bDtrYl5A3F-g,4.196081000018594
---1lKK3aKOuomHnwAkAow,qtOt27b-3Re6zM5OSBVFNQ,4.760399338657155
---1lKK3aKOuomHnwAkAow,y8d90Pt16Nip-B5UXWBP-w,4.070516829484147
---1lKK3aKOuomHnwAkAow,ZibmYdOPKLlqDM9oR6xzOA,4.065852653899053
---1lKK3aKOuomHnwAkAow,_sh6mIBWZis66mAjkjN8Qg,3.680489273558252
---1lKK3aKOuomHnwAkAow,78TC3sZSYBzBsSJ0z5pyhw,3.3859957600579005
---1lKK3aKOuomHnwAkAow,p5rpYtxS5xPQjt3MXYVEwA,4.16388295510869
---1lKK3aKOuomHnwAkAow,5cbsjFtrntUAeUx51FaFTg,3.403654985104043
---1lKK3aKOuomHnwAkAow,Qy_tDaVTWlS14fEglzo1Tg,3.895297549880592
---1lKK3aKOuomHnwAkAow,GT0K4EdSSxe_LMU6SPr-_A,4.515475941735145
---1lKK3aKOuomHnwAkAow,2VNa2kbbt4o8nQmPKIcHVQ,4.534975909764702
---1lKK3aKOuomHnwAkAow,qmymSqVwHYRqdwfcBatzpQ,4.200270492254465
---1lKK3aKOuomHnwAkAow,nK7JeIqdBli3umEhBIh33g,3.7133217183241767
---1lKK3aKOuomHnwAkAow,CWNMLT-ppaUjLMmrnYDPVg,4.494443748315581
---1lKK3aKOuomHnwAkAow,dM8Yp8StA1NdusK5Ta_j-g,4.326147817063668
---1lKK3aKOuomHnwAkAow,ZgPnRzWjQR5NtiauGBww7g,4.285241000069365
---1lKK3aKOuomHnwAkAow,mz9ltimeAIy2c2qf5ctljw,4.352857721818584
---1lKK3aKOuomHnwAkAow,PIXl4WvAqjl2w9Lj81zXvQ,3.9192529232071687
---1lKK3aKOuomHnwAkAow,jobP3ywRd3QNZ_GCoPG2DQ,3.835369063811667
---1lKK3aKOuomHnwAkAow,g8OnV26ywJlZpezdBnOWUQ,3.798373478260273
---1lKK3aKOuomHnwAkAow,iPM85PQMs7QoAxw-Og9ChQ,3.685826905326026
---1lKK3aKOuomHnwAkAow,qTlDDzDEHvDO6iwiNhpI6A,4.280693202824901
```

Case 4:

```
RMSE:  0.9733090296957071
Duration:  359.62151861190796
```

```
user_id,business_id,original,prediction
-LfTBo0oa_uD454ScEW2XA,aokqWHt8vMf5iwj8KZZ5eg,0.5,1.1481481481481481
-Mm02AeY1PMGg-l-ShMxUg,cKgkSMcPXwWTzPrJRpa2qw,0.5,1.0
0OOmvIS1Tb5ccYOCvq4Yeg,lnU-G15oaYWwqqbUXj9JEA,0.5,1.0
0e2oJ82g4ZfvQzIc8SE9Xw,tZy8IQbEWVB8DW8NdGc9FQ,0.5,1.0
1AyaRLA83q_V9D3bYevRkQ,8m1HM54_hE6UcQcxWTWMaA,0.5,1.0
1FGe55e-ZPkumoBK7cyjnA,EAFuhvRxoAhYARuiXCsUSw,0.5,1.0
2-hzRW7iwQE0gzQIyQnwlA,6h7RuXs79Ok_8RPRbUapJg,0.5,1.0
20vCzyibdvXOxW42h9v_MQ,6QKauV9Qt9VgX38-R7Q--g,0.6666666666666666,1.1481481481481481
2pbOSrbB5AtQIa240AXNvQ,pVLZQRs67CF4gZCEYCe_3Q,0.5,1.0
2qIZlW8YSLL8a7I6KkJdHQ,btoBhgcY7vtvQ2M0itdTBQ,0.5,1.0
3CI95LMvt4u1xqEWq6CbnQ,Tje8uY1C4RMUPXcw4dsScQ,0.5,1.0
3bpMCRbjDe-rNvHAXYvUCw,7hlICFMky5fPFxeYRgg__w,0.5,1.0
3lRFcrfR3IrSAQJtrnn80A,APTiouqg0vmMznY_UoJ6MA,0.5,1.0
4-0ozduHY8Ag6cwXMl8UAQ,g6L-oVN4geIRdABs8we-zw,0.5,1.0
4-0ozduHY8Ag6cwXMl8UAQ,u9zsTjkceESEEbncLBChaQ,0.5,1.0
58TsZIweIkb8B4iOkA4jWQ,AnBIIqzFTO4jZ6z_uI36Kw,0.5,1.0
5J6uI0fpUfIgHe1V_UfcZg,8VLDDOtJmp5pPiYokIlRfQ,0.5,1.0
5UNsySfBLt1zf7iCurXXBA,D8whLHsJ2f_tM3Y2lJgMOg,0.5,1.222222222222222
5_5AT_uw_akxMIu_8LLhew,NUGa_UvqPCmq7G2Qm-AByQ,0.5,1.0
6--bnN_gTtztYNetz3xrrQ,8fR3XpJnMQDGopj0_mlsYA,0.5,1.0
640Up5vPJ8xUE4Nmci6yXw,Y4aFR2VSgSz07idEJEX3yg,0.5,1.0
6QKauV9Qt9VgX38-R7Q--g,8D2R7acumiJgJXJ5NMF4GA,0.5,1.0
6QKauV9Qt9VgX38-R7Q--g,oRmCI0p2mvlrGwilJloFTQ,0.5,1.0
6sCdVRtmhJgNqKh9Vx6JXw,WvkxYyA1fXDPDyqIVn8zBQ,0.5,1.0
7LLU_dpe7iN97nJzYmTvjQ,i95z3PicJ4p_oAYXBioznA,1.0,1.2962962962962963
7jKDOB9b_nijq0-djwyUTw,oYV7mtazpc8LDGMWmbU1Jw,0.5,1.0
7s39xlfRI4AJqOkjexPRMA,aq5t70LUvN3KdUdYzdhd2Q,1.0,1.4444444444444442
Ah4i15g8Ow_zphzcpulTxQ,aANR9KZfeCMVF6kMls58Sw,0.5,1.0
C-JdFdmF130R53GOJRX3Zg,vgVQFufjuqDaHj19ntOiFw,1.0,1.1481481481481481
CSn2-XpArLLYeZ_k2BKzrg,CrIWqmuO2uQWwl3z11K_BA,0.5,1.0
CpqlSOEt7eLl8GVmGMlXLA,rPeHN1B8FvYzI7GzVFam3w,0.5,1.0
```

Case 5:

RMSE: 0.9733090296957071

Duration: 375.3733117580414

```
user_id,business_id,original,prediction
-LfTBo0oa_uD454ScEW2XA,aokqWHt8vMf5iwj8KZZ5eg,0.5,1.1481481481481481
-Mm02AeY1PMGg-l-ShMxUg,cKgkSMcPXwWTzPrJRpa2qw,0.5,1.0
0OOmvIS1Tb5ccYOCvq4Yeg,lnU-G15oaYWwqqbUXj9JEA,0.5,1.0
0e2oJ82g4ZfvQzIc8SE9Xw,tZy8IQbEWVB8DW8NdGc9FQ,0.5,1.0
1AyaRLA83q_V9D3bYevRkQ,8m1HM54_hE6UcQcxWTWMaA,0.5,1.0
1FGe55e-ZPkumoBK7cyjnA,EAFuhvRxoAhYARuiXCsUSw,0.5,1.0
2-hzRW7iwQE0gzQIyQnwlA,6h7RuXs79Ok_8RPRbUapJg,0.5,1.0
20vCzyibdvXOxW42h9v_MQ,6QKauV9Qt9VgX38-R7Q--g,0.6666666666666666,1.1481481481481481
2pbOSrbB5AtQIa240AXNvQ,pVLZQRs67CF4gZCEYCe_3Q,0.5,1.0
2qIZlW8YSLL8a7I6KkJdHQ,btoBhgcY7vtvQ2M0itdTBQ,0.5,1.0
3CI95LMvt4u1xqEWq6CbnQ,Tje8uY1C4RMUPXcw4dsScQ,0.5,1.0
3bpMCRbjDe-rNvHAXYvUCw,7hlICFMky5fPFxeYRgg__w,0.5,1.0
3lRFcrfR3IrSAQJtrnn80A,APTiouqg0vmMznY_UoJ6MA,0.5,1.0
4-0ozduHY8Ag6cwXMl8UAQ,g6L-oVN4geIRdABs8we-zw,0.5,1.0
4-0ozduHY8Ag6cwXMl8UAQ,u9zsTjkceESEEbncLBChaQ,0.5,1.0
58TsZIweIkb8B4iOkA4jWQ,AnBIIqzFTO4jZ6z_uI36Kw,0.5,1.0
5J6uI0fpUfIgHe1V_UfcZg,8VLDDOtJmp5pPiYokIlRfQ,0.5,1.0
5UNsySfBLt1zf7iCurXXBA,D8whLHsJ2f_tM3Y2lJgMOg,0.5,1.222222222222222
5_5AT_uw_akxMIu_8LLhew,NUGa_UvqPCmq7G2Qm-AByQ,0.5,1.0
6--bnN_gTtztYNetz3xrrQ,8fR3XpJnMQDGopj0_mlsYA,0.5,1.0
640Up5vPJ8xUE4Nmci6yXw,Y4aFR2VSgSz07idEJEX3yg,0.5,1.0
6QKauV9Qt9VgX38-R7Q--g,8D2R7acumiJgJXJ5NMF4GA,0.5,1.0
6QKauV9Qt9VgX38-R7Q--g,oRmCI0p2mvlrGwilJloFTQ,0.5,1.0
6sCdVRtmhJgNqKh9Vx6JXw,WvkxYyA1fXDPDyqIVn8zBQ,0.5,1.0
7LLU_dpe7iN97nJzYmTvjQ,i95z3PicJ4p_oAYXBioznA,1.0,1.2962962962962963
7jKDOB9b_nijq0-djwyUTw,oYV7mtazpc8LDGMWmbU1Jw,0.5,1.0
7s39xlfRI4AJqOkjexPRMA,aq5t70LUvN3KdUdYzdhd2Q,1.0,1.4444444444444442
Ah4i15g8Ow_zphzcpulTxQ,aANR9KZfeCMVF6kMls58Sw,0.5,1.0
C-JdFdmF130R53GOJRX3Zg,vgVQFufjuqDaHj19ntOiFw,1.0,1.1481481481481481
CSn2-XpArLLYeZ_k2BKzrg,CrIWqmuO2uQWwl3z11K_BA,0.5,1.0
CpqlSQEt7eLl8GVmGMlXLA,rPeHN1B8FvYzI7GzVFam3w,0.5,1.0
```

8 REFERENCES

8.1 LIST OF JOURNALS

[1] Yifan Gao, Wenzhe Yu, Pingfu Chao, Rong Zhang, Aoying Zhou, Xiaoyan Yang, "A Restaurant Recommendation System by Analyzing Ratings and Aspects in Reviews"

[2] Julian McAuley, Jure Leskovec, "Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text"

[3] Richa Sharma, and Rahul Singh, Indian Journal of Science and Technology, "Evolution of Recommender Systems from Ancient Times to Modern Era: A Survey", Vol 9(20), DOI: 10.17485/ijst/2016/v9i20/88005, May 2016

[4] Dr. Alia Karim Abdul Hassan, Ahmed Bahaa aldeen abdulwahhab, "Reviews Sentiment analysis for collaborative recommender system", http://dx.doi.org/10.24017/science.2017.3.22

[5] Srujana Merugu, "A Scalable Collaborative Filtering Framework based on Co-clustering", Thomas George

[6] Sumedh Sawant, Gina Pai, "Yelp Food Recommendation System"

[7] Chee Hoon Ha, "Yelp Recommendation System Using Advanced Collaborative Filtering"