Exploring Recommender System using Yelp Dataset

Kanika Sanduja, Muhammad R Muhaimin, Vishal Shukla

Abstract: Recommender system help business to recommend product to their customers based on similarity with other buyer and their previous buying preference. As a part of our Data Mining CMPT 721 project we participated in a kaggle competition where we explored yelp dataset and predicted rating for business by users.

Dataset: The training dataset contains 2,038,130 rows and 5 columns which holds id, business_id of business for which the rating was given, user_id of a user who gave the rating, the actual rating and the date when the rating was given. The figure below shows the first row of our training dataset.

train_id	user_id	business_id	rating	date
1	1	0	4	2015-02-21

We were also given a json dataset where each object contain a business id and a rating comment it receives. Below is a sample object

```
{
     "text": "Beautiful space, great location, staff rock.
     Tiny room, but this was expected. Bathroom amazing.
     Walls, however, paper thin, which is why I can barely
     string a sentence together in this review.",
     "id": 1
}
```

For test data we were given a similar dataset to our training set without the rating column which we predicted as a part of our kaggle competition.

Data Analysis:

User_id and business_id: For the very first step we decided to use the column user_id, business_id and rating to create a matrix to represent user's rating for a business. For our rating row indicates an user and a column indicates a business. For example if having $M_{10} = 4$ that means user with id 1 rated business with id 0 as 4.

Date: Second portion of data we have decided to use is date. Date signifies a lot in our rating system, for example if any business receive a good review in any given day from multiple users the review for that business in the same day would be similar if the user is unbiased. Date also tells us which time of the year we the business is getting review and find any pattern or bias in there. For example when it comes to restaurant, take out lunch places usually have lots of bad review during winter time, because if not packed properly food get cold due to outside temperature and incur bad reviews; when it comes to hotel, in tourist season hotels are really busy and not able to serve all of their customer properly thus incur bad reviews. Also as business goes through lots of management change recent reviews has heavier weight compared to older reviews so taking care of date our algorithm can transform the user, business review matrix into a weighted matrix.

Actual review comment: We can extract the actual review content from JSON file. From the actual review content we can extract two major things, type of business and sentiment in the review. Using the text mining technique from this review we can extract information such as type of business, location of the business, bias associated with the business, sentiment associated with the review etc. However the complexity associated with it might make our solution of the recommender system complicated. For example let's consider the following reviews "*It's always busy here, but to me, it's worth the wait. Every time we are in Vegas we make a stop for White Castle.*" also "*Vegas with my friend..bring some white castle home..I took a bite and was really disappointed... Not worth it. Try it and now I will try to forget about it.*" So the above two reviews are for White Castle restaurant in Las Vegas which shows user biases, first user really liked it despite going in a busy time on the other hand second user doesn't like it at all. So text mining these reviews will also add another complexity for us to decide whether adding these factor can improve our rating prediction or make our model overfit. Also for the first review if we don't have a prior knowledge that White Castle is a chain restaurant we would not be able to predict what kind of business it is. When it comes to sentiment analysis accuracy of sentiment also reflect the accuracy of our prediction. So include text mining for rating prediction will make our problem really complex so we decide not to use text mining for solving this particular problem.

Prediction approach using Collaborative filtering: For predicting the rating firstly we have used collaborative filtering Below is a detail of our approach that we used in collaborative filtering.

Data processing: Before using collaborative filtering algorithm, we were required to create the user, business rating relationship matrix that we discussed in our data analysis section. For data processing approach we tried in two different computer architecture. Single threaded computational architecture using our own personal computer and distributed cluster computation architecture using SFU CSIL cluster. In single threaded processing we used Python Pandas Data Analysis Library and Python Numpy package for reading the datas from disk to memories. We used Python Scikit Surprise package [1] for folding the data which later we used for cross validation. When we work in the cluster we used Apache Spark for reading the data from disk, splitting our available data set into train and validation data, random splitting and preprocessing the data. In terms of preprocessing we need to change our date column's string date value to unix timestamp for which we used python's datetime library. After our data processing we performed the following prediction algorithm to predict ranking for our test set.

Baseline prediction algorithm: For using a baseline prediction algorithm we also decided to use Python Scikit Surprise library. This algorithm is based on the theory discussed in Yehuda Koren's [2]. "*Factor in the neighbors: scalable and accurate collaborative filtering. 2010*" publication which states we have to minimize sum of squared error

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda({b_u}^2 + {b_i}^2)$$

Where R_{train} is our train set r_{ui} is our actual rating. $\mu + b_u + b_i$ is our predicted rating, $\lambda\,(b_u^2 + b_i^2)$ is our regularization parameter, and b_u, b_i are bias associated with users and bias associated with business. With our baseline algorithm we tried both Alternating Least Square Method and Stochastic Gradient Descent. For a better prediction we tuned the following hyper parameters as suggested by Surprise documentation

- ALS: **`'reg_i':`** The regularization parameter associated with business., **'reg_u'**: The regularization parameter associated with users, **`'n_epochs':`** The number of iteration of the Alternating Least Square procedure.
- SGD: **`'reg':`** The regularization parameter of the cost function that is optimized, **`'learning_rate':`** The learning rate of Stochastic Gradient Descent, **`'n_epochs'`**: The number of iteration of the Stochastic Gradient Descent.

Calculating RMSE: For calculating RMSE we used 10 fold cross validation in our training data. Here is the comparison of baseline algorithm using various hyperparameters

ALS				SGD			
n_epochs	16	15	20	n_epochs	50	20	0.001
reg_u	6.123	10	5	reg	0.08	0.1	50
reg_i	2.012	5	3	learning_rate	0.0035	0.005	0.002
RMSE	1.2491	1.2541	1.2503	reg_u	0.07	0.01	0.05
				reg_i	0.2	0.02	0.01
				RMSE	1.2505	1.2576	1.2578

SVD			
lr_all	0.0035	0.005	0.009
reg_all	0.04	0.05	0.07
n_factors	200	200	300
lr_bu	0.01	0.05	0.1
lr_bi	0.01	0.05	0.1
RMSE	1.2585	1.2895	1.3000

Other than surprise we also use Apache Spark's Mllib's library to perform ALS using collaborative filtering [2]. Based on Spark's documentation we were allowed to tune the hyper parameter maxIter the number of iteration of the Alternating Least Square procedure, **`regParam`** regularization parameter , and rank the number of latent factors to be considered. We randomly take 80% of our dataset as training and 20% as our validation set. Using trial and error approach we find optimum value of **`maxIter`** as 10, **`regParam`** as 0.5 and **`rank`** as 40 which gives us RMSE of 1.2.

Matrix factorization based prediction algorithm: For using matrix factorization based prediction algorithm we used Python Surprise Library's implementation of SVD algorithm which is based on Simon Funk's blog where he discussed use of it during Netflix competition. Here each business can be represented by vector q_i and each user can be represented by a vector p_u such that their dot product gives the expected rating. The dot product captures the user's estimated interest in the item.

$$\widehat{r_{ui}} = q_i^T . p_u \qquad ....[4]$$

With considering user and business biases b_u, b_i it will be

$$\widehat{r_{ui}} = \mu + b_u + b_i + q_i^T . p_u$$

Like our last algorithm by minimizing the following regularized squared error we can estimate the unknown

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - r_{ui})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2)$$

And lastly we can perform the minimization using Stochastic Gradient Descent. For a better prediction based on Surprise documentation we tune the hyperparameter lr_bu learning rate to find optimum user bias in when performing SGD, lr_bi learning rate to find optimum business bias in when performing SGD, lr_all learning rate to find optimum value for all other parameters when performing SGD, reg_all regularization value for all parameters, n_factors number of factors.

Prediction approach using Classification Problem: Instead of using collaborative filtering here we also try to do an experiment and see what result we can get if we consider this problem as a classification problem. Where we will have every possible rank (1-5 in our case) as one of the class we are going to predict. As our feature we considered user_id, business_id, day_of_week and month as our feature column. And we use rating as our target column.

Data Processing: For Data processing we use Apache Spark to read the data from disk, using date string given we created two new features day of the week and month of the year. We randomly selected 80% data for training and 20% data for validation.

Classification Algorithm: For classification we ensemble decision tree Random forest. As per Spark's documentation we are allowed to change number of trees and maximum. We converted our test accuracy to RMSE to compare it with other algorithm. For comparing it with RMSE we use the following method

$$\text{RMSE} = \sqrt{\frac{\sum(y - \hat{y})^2}{N}}$$, Where y = actual rating and $\hat{y}$ is predicted rating

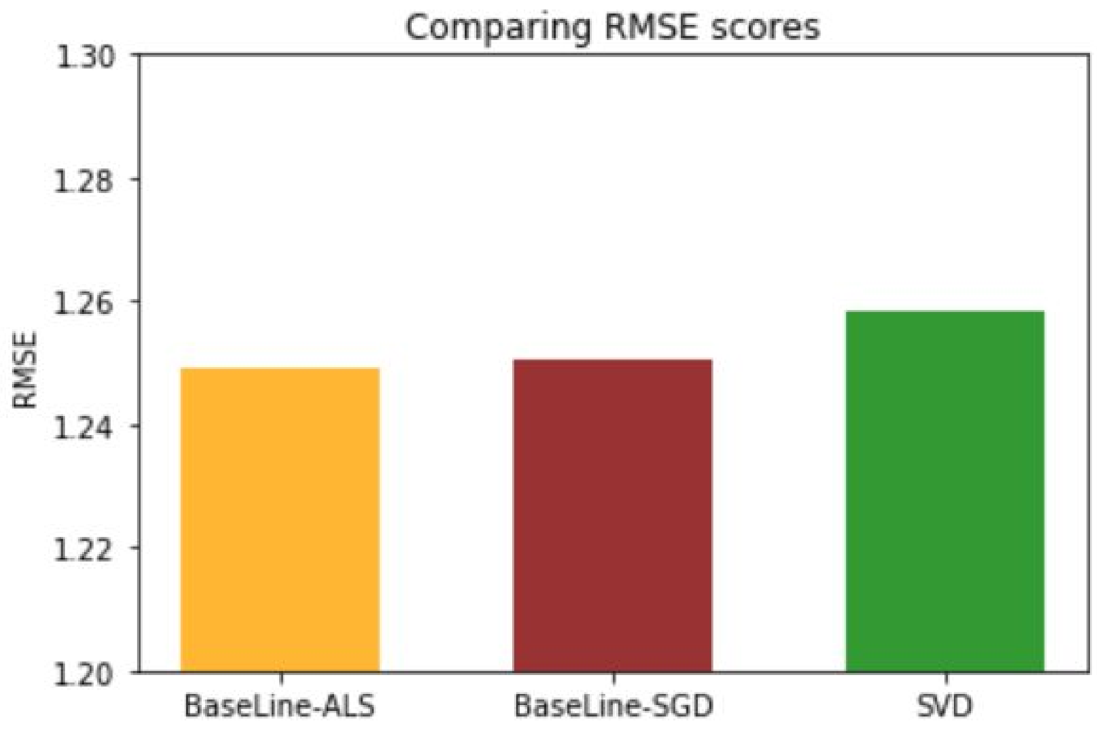
For training our algorithm on 80% data and calculate RMSE on 20% data. Using tree depth of 7 and number of trees as 100 we get RMSE of 1.859

Choices Made: Finally we decided to choose Baseline ALS as it gives us the lowest error.

REFERENCES:

1. http://surprise.readthedocs.io/en/stable/index.html
2. http://www.cs.rochester.edu/twiki/pub/Main/HarpSeminar/Factorization_Meets_the_Neighborhood-_a_Multifaceted_Collaborative_Filtering_Model.pdf
3. https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html
4. http://papers.nips.cc/paper/3208-probabilistic-matrix-factorization.pdf.
5. http://sifter.org/~simon/journal/20061211.html
6. https://medium.com/@m_n_malaeb/singular-value-decomposition-svd-in-recommender-systems-for-non-math-statistics-programming-4a622de653e9

Appendix


Comparing RMSE scores
RMSE
1.30
1.28
1.26
1.24
1.22
1.20
BaseLine-ALS
BaseLine-SGD
SVD

1] Approach with Scikit-Surprise SVD

```
svd.py

from surprise import Dataset, SVD, Reader
import pandas as pd

train_rating_df = pd.read_csv("train_rating.txt", header=0,
index_col=0)
test = pd.read_csv('test_rating.txt', header=0, index_col=0)
test['dummy_rating'] = '-1'
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(train_rating_df[['user_id',
'business_id', 'rating']], reader)
trainset = data.build_full_trainset()
algo = SVD(lr_all=0.0035, reg_all=0.04, n_factors=200, lr_bu=0.01,
lr_bi=0.01)
algo.train(trainset)
testdata = Dataset.load_from_df(test[['user_id', 'business_id',
'dummy_rating']], reader)
predictions =
algo.test(testdata.construct_testset(raw_testset=testdata.raw_rati
ngs))
df = pd.DataFrame(predictions)
newdf = df['est']
newdf.rename('rating', inplace=True)
newdf.to_csv('submission.csv',header='rating',index_label='test_id
')
```

2] Approach using Scikit-Surprise BaseLine ALS

```
from surprise import Dataset, Reader, evaluate, BaselineOnly
import pandas as pd

train_rating_df = pd.read_csv("train_rating.txt", header=0,
index_col=0)
test = pd.read_csv('test_rating.txt', header=0, index_col=0)
test['dummy_rating'] = '-1'
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(train_rating_df[['user_id',
'business_id', 'rating']], reader)
trainset = data.build_full_trainset()
bsl_options = {'method':'als', 'n_epochs':18, 'reg_u':6.10,
'reg_i':1.97}
algo=BaselineOnly(bsl_options=bsl_options)
algo.train(trainset)
testdata = Dataset.load_from_df(test[['user_id', 'business_id',
'dummy_rating']], reader)
predictions =
algo.test(testdata.construct_testset(raw_testset=testdata.raw_rati
ngs))
df = pd.DataFrame(predictions)
newdf = df['est']
newdf.rename('rating', inplace=True)
newdf.to_csv('submission.csv',header='rating',index_label='test_id
')
```

3] Approach with Scikit-Surprise BaseLine SGD method

```
from surprise import Dataset, Reader, BaselineOnly
import pandas as pd

train_rating_df = pd.read_csv("train_rating.txt", header=0,
index_col=0)
test = pd.read_csv('test_rating.txt', header=0, index_col=0)
test['dummy_rating'] = '-1'
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(train_rating_df[['user_id',
'business_id', 'rating']], reader)
trainset = data.build_full_trainset()
bsl_options =
{'method':'sgd','reg':0.08,'n_epochs':50,'learning_rate':0.0035,
'reg_u':0.07, 'reg_i':0.2}
algo=BaselineOnly(bsl_options=bsl_options)
algo.train(trainset)
testdata = Dataset.load_from_df(test[['user_id', 'business_id',
'dummy_rating']], reader)
predictions =
algo.test(testdata.construct_testset(raw_testset=testdata.raw_rati
ngs))
df = pd.DataFrame(predictions)
newdf = df['est']
newdf.rename('rating', inplace=True)
newdf.to_csv('submission.csv',header='rating',index_label='test_id
')
```

4] Approach using Spark MLlib ALS

```
# Based on documentation provided my Spark MLLib official
documentation

from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import SparkSession, functions, types
from datetime import datetime

spark = SparkSession.builder.appName('DM_Coll_Fil').getOrCreate()
sc = spark.sparkContext
sc.setLogLevel("WARN")
def main():
    convert_to_int = functions.udf(lambda value: float(value),
types.FloatType())
    get_timestamp = functions.udf(lambda datestring:
datetime.strptime(datestring, '%Y-%m-%d').timestamp(),
types.LongType())
    fit_in_range_lower =  functions.udf(lambda value: 1.0 if value
< 1 else value, types.FloatType())
    fit_in_range_higher =  functions.udf(lambda value: 5.0 if
value > 5 else value, types.FloatType())

    training_data = spark.read.csv('train_rating.txt',
header=True)
    test_data = spark.read.csv('test_rating.txt',
header=True).drop('date')

    for column in training_data.columns:
        if(column != 'date'):
            training_data = training_data.withColumn(column,
convert_to_int(training_data[column]))
        else:
            training_data = training_data.withColumn(column,
get_timestamp(training_data[column]))

    for column in test_data.columns:
        if (column != 'date'):
            test_data = test_data.withColumn(column,
convert_to_int(test_data[column]))
```

```
        else:
            test_data = test_data.withColumn(column,
get_timestamp(test_data[column]))

    (training, test) = training_data.randomSplit([0.8, 0.2])

    als = ALS(maxIter=10, regParam=0.5, userCol="user_id",
itemCol="business_id", ratingCol="rating",
              coldStartStrategy="drop", alpha=1, rank=40)
    model = als.fit(training)

    # Evaluate the model by computing the RMSE on the test data
    test_predictions = model.transform(test)
    test_predictions = test_predictions.withColumn('prediction',
fit_in_range_lower(test_predictions['prediction']))
    test_predictions = test_predictions.withColumn('prediction',
fit_in_range_higher(test_predictions['prediction']))


    evaluator = RegressionEvaluator(metricName="rmse",
labelCol="rating",
                                    predictionCol="prediction")
    rmse = evaluator.evaluate(test_predictions)
    print("Root-mean-square error = " + str(rmse))
    rmse_string = "Root-mean-square error =
{}\\n".format(str(rmse))
    with open("result_{}.txt".format(datetime.now().isoformat()),
"a") as myfile:
        myfile.write(rmse_string)
    #Uncomment when you are satisfied with training
    # final_pred = model.transform(test_data)
    # final_pred = final_pred.withColumnRenamed('prediction',
'rating')
    #
final_pred.select("test_id","rating").toPandas().to_csv('submissio
n.csv', sep=',', encoding='utf-8',index=False)
    return

if __name__=='__main__':
    # Note: in current version output is only used for debugging
    main()
```

5] Approach as a Ensemble tree classification

```
#
# Original authour Mr. Greg Baker, Senior Lecturer, School of
Computing Science, Simon Fraser University
# Modified by, Muhammad Raihan Muhaimin, mmuhaimi@sfu.ca
#


from pyspark.sql import SparkSession, functions, types
from datetime import datetime
from pyspark.ml.feature import StringIndexer, VectorAssembler,
OneHotEncoder
from datetime import datetime
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
from pyspark.ml.classification import RandomForestClassifier,
MultilayerPerceptronClassifier


spark = SparkSession.builder.appName('Spark_Neural').getOrCreate()
sc = spark.sparkContext



def get_sq_error(actual, prediction):
    return (actual - prediction) ** 2



def main():
    get_pred_error = functions.udf(get_sq_error,
types.FloatType())
    convert_to_int = functions.udf(lambda value: int(value),
types.IntegerType())
    get_day_of_week = functions.udf(lambda timestamp:
datetime.strptime(timestamp, '%Y-%m-%d').weekday(),
types.StringType())
```

```
    get_month = functions.udf(lambda timestamp:
datetime.strptime(timestamp, '%Y-%m-%d').month,
                                    types.StringType())
    training_data = spark.read.csv('train_rating.txt',
header=True)
    test_data = spark.read.csv('test_rating.txt', header=True)

    for column in training_data.columns:
        if(column != 'date'):
            training_data = training_data.withColumn(column,
convert_to_int(training_data[column]))
        else:
            training_data = training_data.withColumn('dow',
get_day_of_week(training_data[column]))
            training_data = training_data.withColumn('month',
get_month(training_data[column]))

    for column in test_data.columns:
        if (column != 'date'):
            test_data = test_data.withColumn(column,
convert_to_int(test_data[column]))
        else:
            test_data = test_data.withColumn('dow',
get_day_of_week(test_data[column]))
            test_data = test_data.withColumn('month',
get_month(test_data[column]))

    training_data = training_data.drop('date')
    test_data = test_data.drop('date')


    discreete_columns = ['dow', 'month']
    string_indexer = [StringIndexer(inputCol='{}'.format(column),
outputCol='{}_ind'.format(column)) for column in
discreete_columns]
    hot_encoders =
[OneHotEncoder(inputCol='{}_ind'.format(column),
outputCol='{}_he'.format(column)) for column in discreete_columns]
    vector_assembler = VectorAssembler(inputCols=["user_id",
"business_id", "dow_he", "month_he"], outputCol="features")

    rf = RandomForestClassifier(numTrees=25, maxDepth=10,
labelCol="rating", seed=42)


    models = [
        ('Rand-forest', Pipeline(stages=string_indexer +
hot_encoders + [vector_assembler, rf]))
    ]

    evaluator =
MulticlassClassificationEvaluator(predictionCol="prediction",
labelCol='rating')

    # split data into training and testing
    train, test = training_data.randomSplit([0.8, 0.2])
    train = train.cache()
    test = test.cache()

    for label, pipeline in models:

        model = pipeline.fit(train)
        predictions = model.transform(test)
        predictions = predictions.withColumn('sq_error',
get_pred_error(predictions['rating'], predictions['prediction']))
        rmse_score =
predictions.groupBy().avg('sq_error').head()[0]
        # calculate a score
        score = evaluator.evaluate(predictions)
        print(label, rmse_score ** 0.5)

    #Uncomment when you are satisfied with training
    # final_pred = model.transform(test_data)
    # final_pred = final_pred.withColumnRenamed('prediction',
'rating')
    #
final_pred.select("test_id","rating").toPandas().to_csv('submissio
n.csv', sep=',', encoding='utf-8',index=False)
    return



if __name__=='__main__':
    main()
```