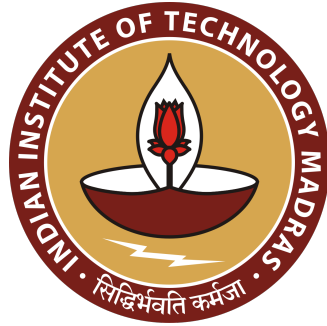


Indian Institute of Technology
Madras

Spark

Pruthvi Raj R G



15-4-2020

Contents

1	Overview	1
1.1	ALS	1
1.2	FP Growth	2
2	Problem 1 : ALS	3
2.1	Steps followed to implement the ALS Algorithm	3
2.2	Results for different train test splits	4
3	Problem 2 - FP Growth	5
3.1	FP Growth - Part 1	5
3.2	Result	6
3.3	FP Growth - Part 2	6
3.4	Result	6

1. Overview

In this assignment we will be implementing two algorithms, namely FP growth and ALS using the Spark Machine Learning library. We will be using google colaboratory to implement the algorithms. First let us understand what these algorithms do and how they are implemented in Spark.

1.1 ALS

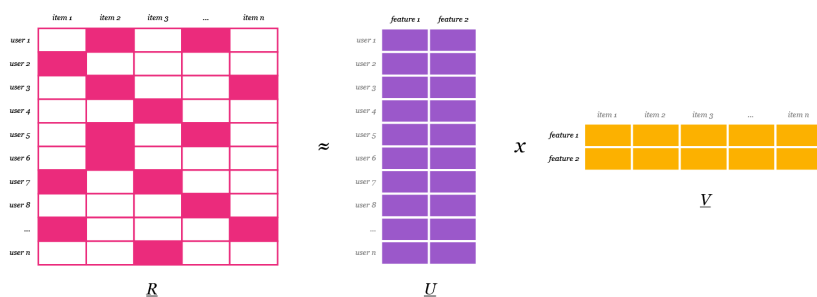
Alternating Least Squares (ALS) is a the model we'll use to fit our data and find similarities. But before we dive into how it works we should look at some of the basics of matrix factorization which is what we aim to use ALS to accomplish.

The idea is basically to take a large (or potentially huge) matrix and factor it into some smaller representation of the original matrix. You can think of it in the same way as we would take a large number and factor it into two much smaller primes. We end up with two or more lower dimensional matrices whose product equals the original one.

When we talk about collaborative filtering for recommender systems we want to solve the problem of our original matrix having millions of different dimensions, but our “tastes” not being nearly as complex. Even if i’ve viewed hundreds of items they might just express a couple of different tastes. Here we can actually use matrix factorization to mathematically reduce the dimensionality of our original “all users by all items” matrix into something much smaller that represents “all items by some taste dimensions” and “all users by some taste dimensions”. These dimensions are called latent or hidden features and we learn them from our data.

Doing this reduction and working with fewer dimensions makes it both much more computationally efficient and but also gives us better results since we can reason about items in this more compact “taste space”.

If we can express each user as a vector of their taste values, and at the same time express each item as a vector of what tastes they represent. You can see we can quite easily make a recommendation. This also gives us the ability to find connections between users who have no specific items in common but share common tastes.



1.2 FP Growth

Frequent patterns are collections of items which appear in a data set at an important frequency (usually greater than a predefined threshold) and can thus reveal association rules and relations between variables. Frequent pattern mining is a research area in data science applied to many domains such as **recommender systems** (what are the set of items usually ordered together), bioinformatics (what are the genes co-expressed in a given condition), decision making, clustering, website navigation.

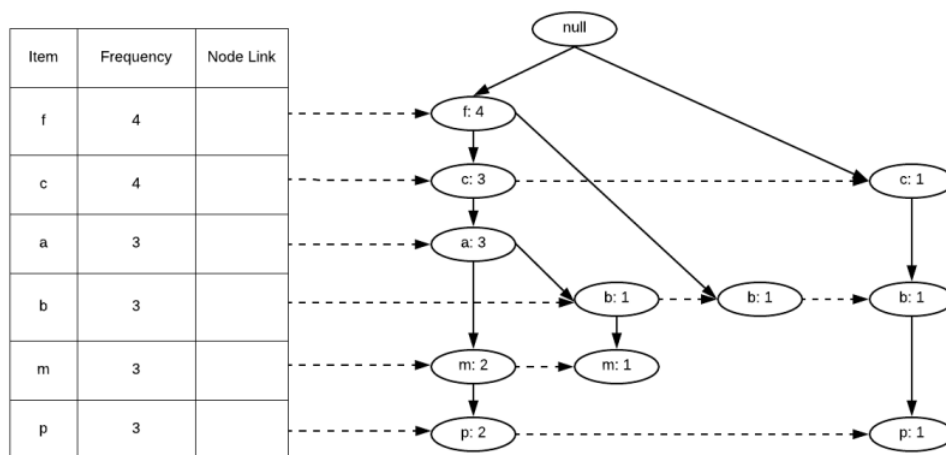
FP tree algorithm uses data organized by horizontal layout. It is the most computationally efficient algorithm. It only performs 2 database scans and keeps the data in an easily exploitable tree structure.

A frequent pattern represents a set of items co-occurring in comparatively more transactions, for instance in the horizontal layout example item1 and item2 appear frequently together. This frequency is quantified using the support metric. Itemset support is the number of transactions where the itemset elements appear together divided by the total number of transactions.

Minimum support is a threshold used by the following algorithms in order to discard sets of items from the analysis which don't appear frequently enough. The **strength** of the association rule between 2 items, (for instance item1 and item2) or the association confidence represents the number of transactions containing item1 and item 2 divided by the number of transactions containing item1. The **confidence** metric estimates the likelihood that a transaction containing item1 will include also item2

The first database scan sorts all items in the order of their global occurrence in the database (the equivalent of applying a counter to the unraveled data of all transactions). The second pass iterates line by line through the list of transactions and for each transaction it sorts the elements by the global order (previous step corresponding to the first database pass) and introduces them as nodes of a tree grown in depth. These nodes are introduced with a count value of 1. Continuing the iterations, for each line new nodes are being added to the tree at the point where the ordered items differ from the existing tree. If the same pattern already exists, all common nodes will increase their count value by one.

The FP tree can be pruned by removing all nodes having a count value inferior to a minimum threshold occurrence. The remaining tree can be traversed and for instance all paths from the root node to leaves correspond to clusters of frequently occurring items.

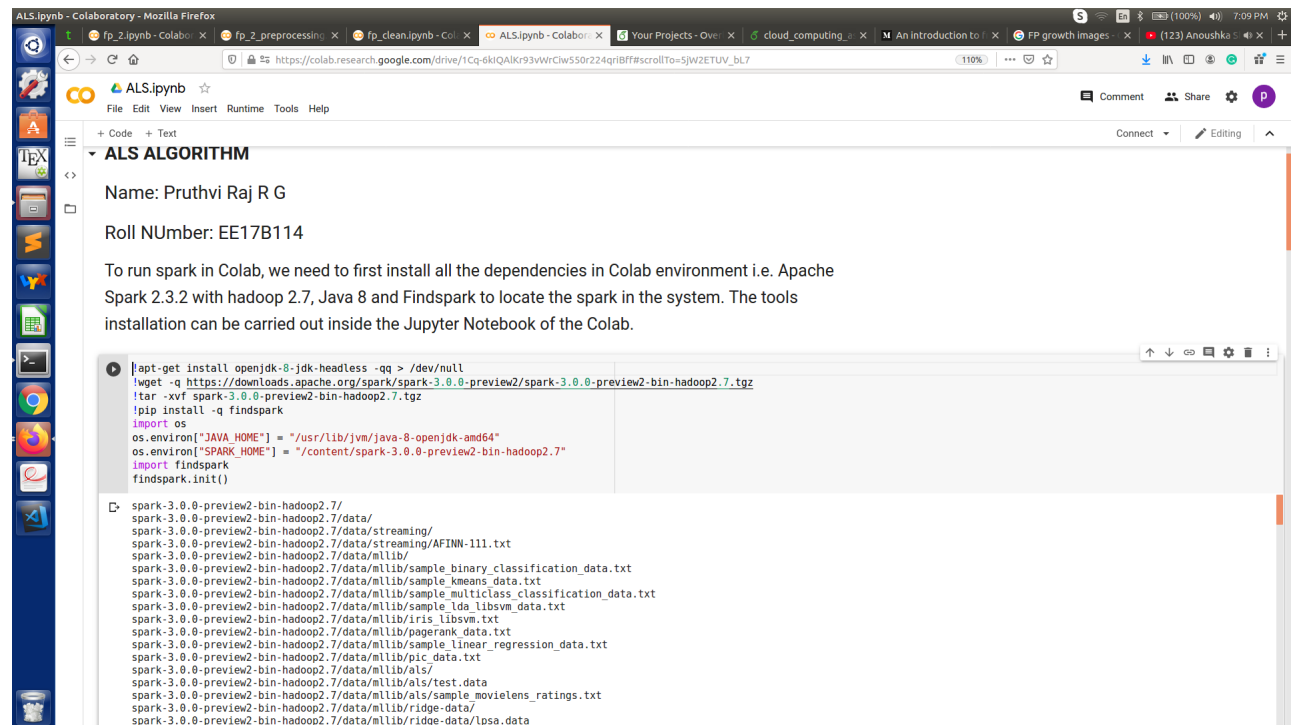


2. Problem 1 : ALS

We are given a movie review data set with the following details: userID, MovieID, Rating seperated by double colons(::)

2.1 Steps followed to implement the ALS Algorithm

1. Install spark.



The screenshot shows a Google Colab notebook interface. The notebook title is 'ALS.ipynb'. The code cell is titled 'ALS ALGORITHM'. The code content is as follows:

```
Name: Pruthvi Raj R G
Roll Number: EE17B114

To run spark in Colab, we need to first install all the dependencies in Colab environment i.e. Apache
Spark 2.3.2 with hadoop 2.7, Java 8 and Findspark to locate the spark in the system. The tools
installation can be carried out inside the Jupyter Notebook of the Colab.

!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.0.0-preview2/spark-3.0.0-preview2-bin-hadoop2.7.tgz
!tar -xvf spark-3.0.0-preview2-bin-hadoop2.7.tgz
!pip install -q findspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-preview2-bin-hadoop2.7"
import findspark
findspark.init()

spark-3.0.0-preview2-bin-hadoop2.7/
spark-3.0.0-preview2-bin-hadoop2.7/data/
spark-3.0.0-preview2-bin-hadoop2.7/data/streaming/
spark-3.0.0-preview2-bin-hadoop2.7/data/streaming/AFINN-111.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/sample_binary_classification_data.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/sample_kmeans_data.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/sample_multiclass_classification_data.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/sample_lda_libsvm_data.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/iris_libsvm.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/pagerank_data.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/sample_linear_regression_data.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/pic_data.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/als/
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/als/test_data
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/als/sample_movielens_ratings.txt
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/ridge-data/
spark-3.0.0-preview2-bin-hadoop2.7/data/mllib/ridge-data/lpsa.data
```

2. Read the data as an RDD.
3. Build a DataFrame from the spark RDD.
4. Split it into train and test sets.
5. Use ALS package available in pyspark.ml.recommendation to fit/train the model.
6. Evaluate the performance of the model using RegressionEvaluator to find its RMSE(root mean square) score.

```

[5] from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

"ALS.txt" contains userID, MovieID, Rating seperated by double colons(::)

[7] lines = spark.read.text("./ALS.txt").rdd
parts = lines.map(lambda row: row.value.split("::"))
ratingsRDD = parts.map(lambda p: Row(userID=int(p[0]), movieID=int(p[1]), rating=float(p[2])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])

"Build the recommendation model using ALS on the training data"
Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics

[8] als = ALS(maxIter=5, regParam=0.01, userCol="userID", itemCol="movieID", ratingCol="rating", coldStartStrategy="drop")
model = als.fit(training)

"Evaluate the model by computing the RMSE on the test data"

[9] predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

Root-mean-square error = 1.6186433339755346

RMS error of the fitted model : 1.6186433339755346

```

2.2 Results for different train test splits

Train/test split	RMSE
90/10	1.2815
80/20	1.6186
70/30	1.9616
60/40	2.0380

We clearly observe that the **RMSE is increasing as we increase the size of the test set** by reducing the size of the training set. This increase in error can be attributed to decrease in available information for the model to train on.

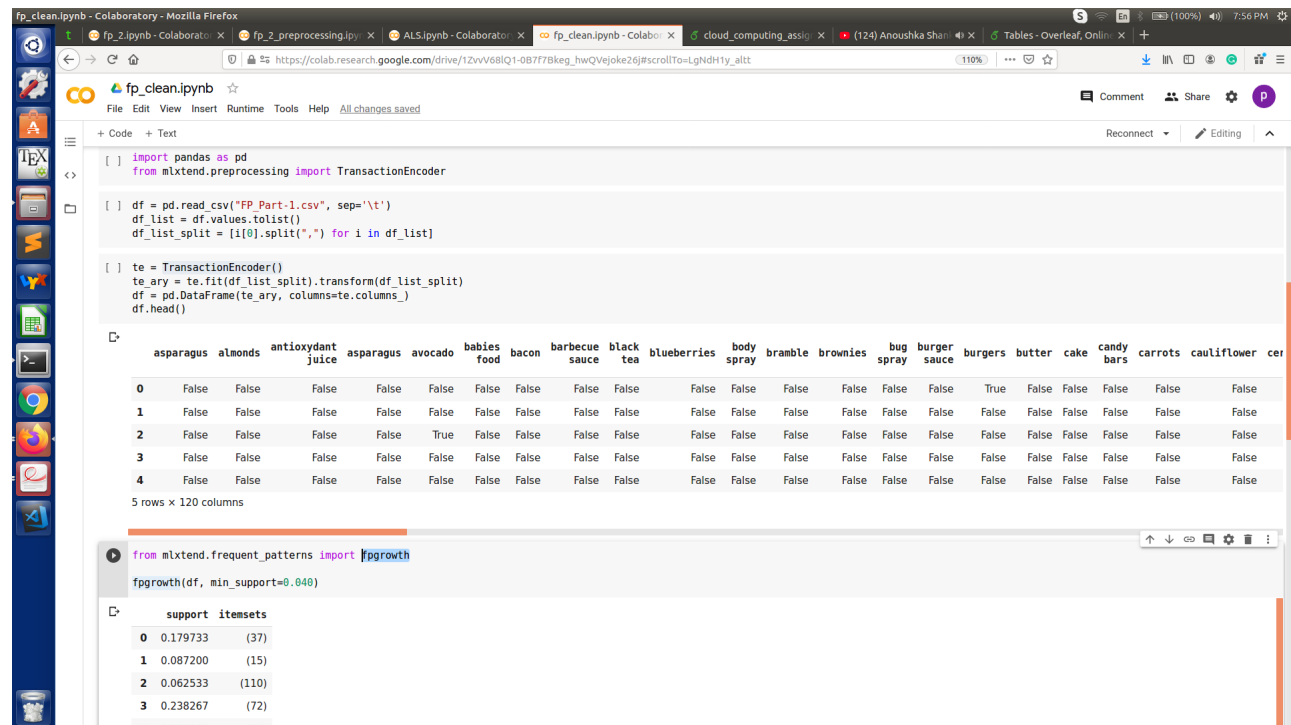
3. Problem 2 - FP Growth

3.1 FP Growth - Part 1

In this problem we are given a dataset containing the shopping information about a group of users. We are expected to mine the association rules and report the top five most frequent pairs of items.

"FP-Part-1.csv" contains the dataset of market basket. Each row in this dataset contains the list of products purchased together.

1. Install Spark.
2. Intall a newer version of mlxtend.
3. Read the dataset as a spark DataFrame.
4. Generate a binary dataframe consisting of True or False associated with the original dataset using TransactionEncoder().
5. Fit the fp-growth model using the fpgrowth package.
6. Tune the confidence and support parameter to extract the most frequent pair.



```
[ ] import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

[ ] df = pd.read_csv("FP-Part-1.csv", sep='\t')
df_list = df.values.tolist()
df_list_split = [i[0].split(",") for i in df_list]

[ ] te = TransactionEncoder()
te_ary = te.fit(df_list_split).transform(df_list_split)
df = pd.DataFrame(te_ary, columns=te.columns_)
df.head()
```

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	body spray	bramble	brownies	bug spray	burger sauce	burgers	butter	cake	candy bars	carrots	cauliflower	cer
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

5 rows x 120 columns

```
from mlxtend.frequent_patterns import fpgrowth
fpgrowth(df, min_support=0.040)
```

	support	itemsets
0	0.179733	(37)
1	0.087200	(15)
2	0.062533	(110)
3	0.238267	(72)
4	0.137000	(64)

3.2 Result

The most occurring pairs found were as follows:

1. 'eggs', 'mineral water' - Support factor: 0.050933
2. 'milk', 'mineral water' - Support factor: 0.048000
3. 'spaghetti', 'mineral water' - Support factor: 0.059733
4. 'chocolate', 'mineral water' - Support factor: 0.052666
5. 'ground beef', 'mineral water' - Support factor: 0.040933

The complete output can be found in the file output-fp-1.csv

3.3 FP Growth - Part 2

The given dataset contains the data in the following format: InvoiceNo, StockCode, Product Description, quantity, InvoiceDate, UnitPrice, CustomerID and country details for each purchased product.

This dataset requires some preprocessing as the format is not standard. The details of preprocessing can be found in the file fp2-2-preprocessing.ipynb. We have saved the processed file as formatted.csv

Now we load this dataset as dicussed before and use fpgrowth package to build the model.

3.4 Result

The most occuring pairs found by our FP growth algorithm are listed below with thier support parameter.

1. 'JUMBO BAG RED RETROSPOT', 'JUMBO BAG PINK POLKADOT' Support Factor - 0.034075
2. 'JUMBO STORAGE BAG SUKI', 'JUMBO BAG RED RETROSPOT' Support Factor - 0.029984
3. 'ROSES REGENCY TEACUP AND SAUCER ', 'GREEN REGENCY TEACUP AND SAUCER' Support Factor - 0.032070
4. 'ALARM CLOCK BAKELIKE GREEN', 'ALARM CLOCK BAKELIKE RED 'Support Factor - 0.026425
5. 'JUMBO SHOPPER VINTAGE RED PAISLEY', 'JUMBO BAG RED RETROSPOT' Support Factor - 0.027939