MSBA 6330 SEC 001

BESTBUY PRODUCTS RECOMMENDATION

How to recommend the item a customer is most interested in, in the least amount of time

TEAM 1
ARPIT SIDANA
ROHIT NADGIRI
SANDEEP CHITTA
WILLIAM EERDMANS
XINYAO QU

Agenda

Introduction

With the development of mobile devices, people spend increasing amounts of time on mobile. A recent study from comScore presented mobile is now the top growth driver for technology (Sterling, 2016). People already spend 65 percent of digital media time on mobile. In the meantime, computers have become the second choice when people search digitally (Sterling, 2016). Therefore, learning customer behavior from mobile web visits is a novel and efficient way to understand the customer and recommend services or products that they might enjoy.

Seeking to utilize these customer-collected insights, our goal for this project was to predict which Best Buy product a mobile web visitor will be most interested in, from their search history and behavior in the past two years and to also discover the most efficient recommendation system to do this process.

For our project, we used Mahout, Spark and GraphLab to build parallel recommendation systems separately, and then compared these three models based on speed in evaluation stage. With increase usage of recommendations across many organizations, the appetite for the cloud platforms where building the machine learning models swiftly is increasing rapidly across organizations. Azure is one such service from Microsoft which is pioneering in online machine learning computations. We would give a brief illustration of how to build recommendation model in Azure.

Background

Currently, Best Buy is the biggest consumer electronics corporation. Best Buy has provided a large dataset to aid in their efforts to target customers with their top recommended item. Best Buy can then use this prediction to advertise certain products to customers who are most likely to buy them, in order to promote sales and profits. Our project uses a suite of big data technologies to predict which Best Buy product a mobile web visitor will be most interested in from their search history and behavior in past two years.

We obtained the data set from Kaggle previous competition. For our dataset, we have around 67 million clicks, 27 million searches, 8 million users, and 1 million products. We have three attributes in our dataset as follows.

- user: A user ID
- sku: The stock-keeping unit (item) that the user clicked on
- rating: A score the user give to the product, with a scale of 1-5.

Firstly, we used Hive and Pig to do initial data processing. Secondly, we built item-based, user-based, and Alternating Least Squares (ALS) recommendation systems using Mahout, Spark and GraphLab separately. Finally, we compared these three platforms and their run times on item-based and ALS recommendation systems to find a more efficient system based on speed.

Recommender Systems Methodology

Recommender systems can help to match users with items. In other words, by obtaining a likelihood score or top-K recommendation items, recommendation systems can help customers find the items they are willing to buy inferred through their on-line shopping behaviors. User-based and Items-based are two kinds of collaborative filtering approaches. Collaborative filtering is the most popular technique. In our project, we utilized these two approaches both in Spark and Mahout and compare them.

Item-based collaborative filtering is a model-based recommendation algorithm. This approach provides k most similar items and their similarities by digging into items that user has rated, and computes the similarity between those rated items and the target item (Sarwar, 2001). User-based collaborative filtering predicts whether the target user would be interested in the target item by comparing rating histories of users with similar profiles (Wang 2006). On the other hand, item-based recommendation looks at the similarity between items to make a recommendation to a final user. The key difference between user-based approach and item-based approach is that they take different bases to calculate similarity.

We also used Alternating Least Squares (ALS) matrix factorization. This technique takes the user-item pairs and user-ratings pairs and combines them into a unified matrix. This matrix is then decomposed so that items a user might not have rated could be recommended. ALS and other matrix factorization methods have steadily become the preferred method among professionals due to its performance, ability to handle sparse datasets, and how it can include many explicit or implicit attributes.

Returning to the business problem, we decided that in order to provide the quickest and most efficient recommendation to a user required exploring each different recommendation technique. Along with looking at different techniques, we also decided to explore the different "Big Data" tools utilizing recommendation systems.

We rated these tools on speed in order to provide a quick recommendation to the user. Two ways of looking at this were tallying total time to create a recommendation given incremental changes in the total dataset (10% sample, 20% sample, etc.) and also the reaction of the recommender system to increasing increments in iterations.

Big Data Tools Used

In an attempt to use the recommender system, we tried various big data tools that have helped us understand several associated nuances. The list of tools tried out are as below:

- Hive
- Mahout
- Spark
- GraphLab
- Azure
- H_2O
- Vowpal Wabbit

Process Flow across Big Data platforms:

Mahout methodology

Mahout is a platform created to make quickly scalable machine learning processes (Apache, 2016). The three techniques for recommendation systems used in Mahout were item-based collaborative filtering, user-based collaborative filtering, and ALS. We began our analysis by loading in the ratings data by dragging it into the virtual machine training folder. Below is a step-by-step walkthrough of how to get Mahout up and running:

Input Data

1. Create directory with, mkdir Project, within Local
2. Double click on "training's Home" icon on the left hand side
3. You should be able to see folders such as demos, Desktop, lib, etc.
4. Double click into Project
5. Now outside the terminal, have your data.csv ready with users, items, and ratings
6. Easily drag this file from your computer into the project folder

Load data into Hive

1. Enter hive shell
2. set hive.cli.print.header=true;   #This has columns show up
3. Create Table ratings before putting data in
4. Load the data into the ratings table

Time comparison based on number of records:

In order to accommodate the sequential samples of 10% of the dataset, the CREATE TABLE statement was used (See Appendix for code). However, a CREATE TABLE SELECT query was used to populate table given. For instance the HiveQL statement to insert a 10% sample into the table ten_perc can be seen in the Appendix, too. This was repeated until there were tables with 10%, 20%, 30%, and up to 100% of the data for analysis.

After the tables were created, the item-based recommender could finally be created. The process involves creating a temporary directory and an output directory. These were deleted each round for tallying the runtime. The below examples in the Appendix use the ten_perc table.

For the Mahout item-based recommender, each table and recommendation was ran for the samples of the dataset and the time were recorded. These times can be found in the findings section. Also, utilizing the ALS method in Mahout is easily implemented. Also, for future comparison with Pyspark and Graphlab, this method allows for changing the number of iterations each time the recommender is ran.

Time comparison based on number of iterations:

The same as before, this process was run 10 times with the different samples of data. However, the iterations were also changed. Since we had difficulties going over 30 iterations in Pyspark, we all standardized our tests on iterations 3 to 30, iterating by 3. Thus we had another 10 timings to compare the efficiency of the different platforms.

Lastly, Mahout does not have a streamlined command to do user-based recommendations. Thus, another tool, Eclipse, was used to create the user-based recommender in Java. The process was lengthy and a tutorial and video were attached to help describe the process (Apache 2014, Cook 2014). However, once the data was ingested into the Eclipse workbench and Maven was also downloaded, we began creating the recommender.

An interesting note is that many of the processes required to be imported and the code can be found in the Appendix. After running this system, we decided to narrow our scope with item-based and ALS recommendation systems. The user-based system in Eclipse took 16 hours to run on the full dataset, whereas we saw that Pyspark and Graphlab ran much quicker.

Spark Mllib Methodology:

"One of the most commonly used recommender systems is Collaborative filtering. This method basically aims at populating the missing spaces in user-item association matrix.

Spark.mllib supports model-based collaborative filtering, in which users and products are described by a set of latent factors that can be used to predict missing entries. spark.mllib uses the alternating least squares (ALS) algorithm to learn these latent factors."[1]

Keeping the external settings similar to that in Mahout, we evaluated the performance of Spark MLlib.

Time comparison based on number of records:

1. We import the necessary libraries and dependencies (ALS, MatrixFactorizationModel and Rating).
2. We load in our "ratings_final.csv" into an RDD.
3. We carry out several pre-processing activities like filtering, mapping etc to drill down to the final RDD to be evaluated
4. We create a list "fraction" that stores the percentages of data that would be sampled from our RDD to build recommendations against iteratively.
5. We initialize rank and number of iterations to 10 and 20 respectively. We train our data using ALS. train and evaluate performance on the test data set.
6. We also incorporate a time function in the loop to record time taken to train the recommender for each iteration.

Time comparison based on number of iterations:

1. We import the necessary libraries and dependencies (ALS, MatrixFactorizationModel and Rating).
2. We load in our "ratings_final.csv" into an RDD.

1 http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html#collaborative-filtering

3. We carry out several pre-processing activities like filtering, mapping etc to drill down to the final RDD to be evaluated
4. We initialize rank to 10 and increase the number of iterations in steps of 3 from 3 to 30. We train our data using ALS.train and evaluate performance on the test data set.
5. We also incorporate a time function in the loop to record time taken to train the recommender for each iteration.

After Mahout, we observed that Spark performed far better owing to use of in-memory process. The observed reduction in time was substantial and the same has been included as part of the findings of this report.

GraphLab Methodology:

GraphLab, an open source project, helps manage Big Data better through storage on S-Frames along with Big Data scalability functionality with Hadoop 2 (YARN), Amazon EC2 and even on the local. It enables to work with terabytes of data since it offers disk based as against memory based storage. S-Frames are largely operationally similar to Data Frames in Python and R.

The complete architecture of GraphLab is shown below:

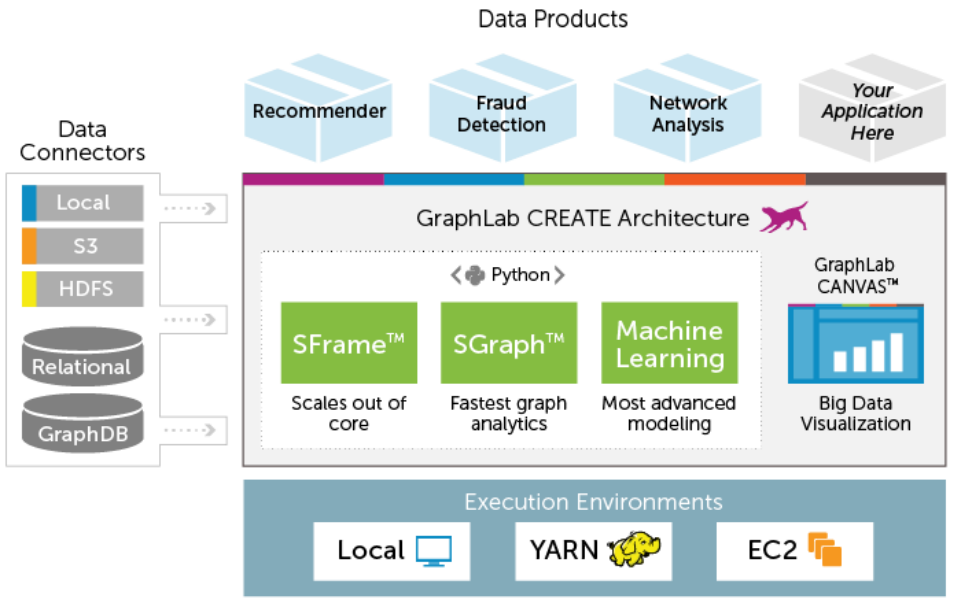


fig-1

Benefits of GraphLab

- Handles Large Data:
    - SFrame: It is a disk-based tabular data structure that avoids data size limitation by facilitating disk based storage as against memory (RAM) based storage. Every column can handle different data types and all columns have to be of the same length. Essentially, each column is an SArray i.e. a series of elements. Moreover, each column is mutable.(Z-nation, 2016).
    - GraphLab can be easily integrated with various data sources like S3, ODBC, JSON, CSV, HDFS and many more.
- Feature Engineering: GraphLab has an inbuilt feature engineering options like binning, imputation, TF-IDF, etc. to enhance model performance.
- Modeling: GraphLab also allows us to perform various modeling options like clustering or regression in a very succinct manner and also provides scope for specialized ML algorithms such as anomaly detection and churn prediction.

Process Flow

We used the academic license registered with Turi GraphLab Create for our project.

While we acknowledge that GraphLab has advanced functionality for recommender systems such as popularity based recommender and content based recommender, we focussed only on item based recommender and matrix factorization (ALS) in order to make relevant performance comparisons against Spark Mllib and Mahout.

Post installation of GraphLab Create, we import GraphLab with our academic license in a Ipython environment on the local client.

Time comparison based on number of records:

1. We import the necessary libraries and dependencies (graphlab, random, time and os) and set the working directory.

2. We load in our “ratings_final.csv” into an SFrame.
3. We create a list “fraction” that stores the percentages of data that would be sampled from our SFrame to build recommendations against iteratively.
4. We write a for loop to randomly sample the specified percentage of data in each step (as defined by values in list “fraction”) and train graphlab.recommender.factorization_recommender.create along with user, item and rating variables specified from our defined SFrame.
5. We also incorporate a time function in the loop to record time taken to train the recommender for each iteration.

Time comparison based on number of iterations:

1. We import the necessary libraries and dependencies (graphlab, random, time and os) and set the working directory.
2. We load in our “ratings_final.csv” into an SFrame.
3. We create a list “numIters” that stores the number of iterations we carried out on the data in our SFrame to build recommendations against iteratively.
4. We write a for loop to train graphlab.recommender.factorization_recommender.create in each step for a new value of number of iterations (as defined by values in list “numIters”) along with user, item and rating variables specified from our defined SFrame.
5. We also incorporate a time function in the loop to record time taken to train the recommender for each iteration. We note that default number of max iterations in GraphLab is 50.

Thus we can see that GraphLab is faster than Mahout by a factor of 100 and faster than Spark Mllib by a factor of 10 on time comparison (number of records in dataset).

Performance Comparison: Spark vs Mahout vs GraphLab

In order to evaluate performance, we tried two different options (percent of dataset) and number of iterations. By changing these parameters in an incremental fashion, we clearly observed that Graphlab was faster than Spark, where both were faster than Mahout. Below is a graph of the various times taken for the code to run.

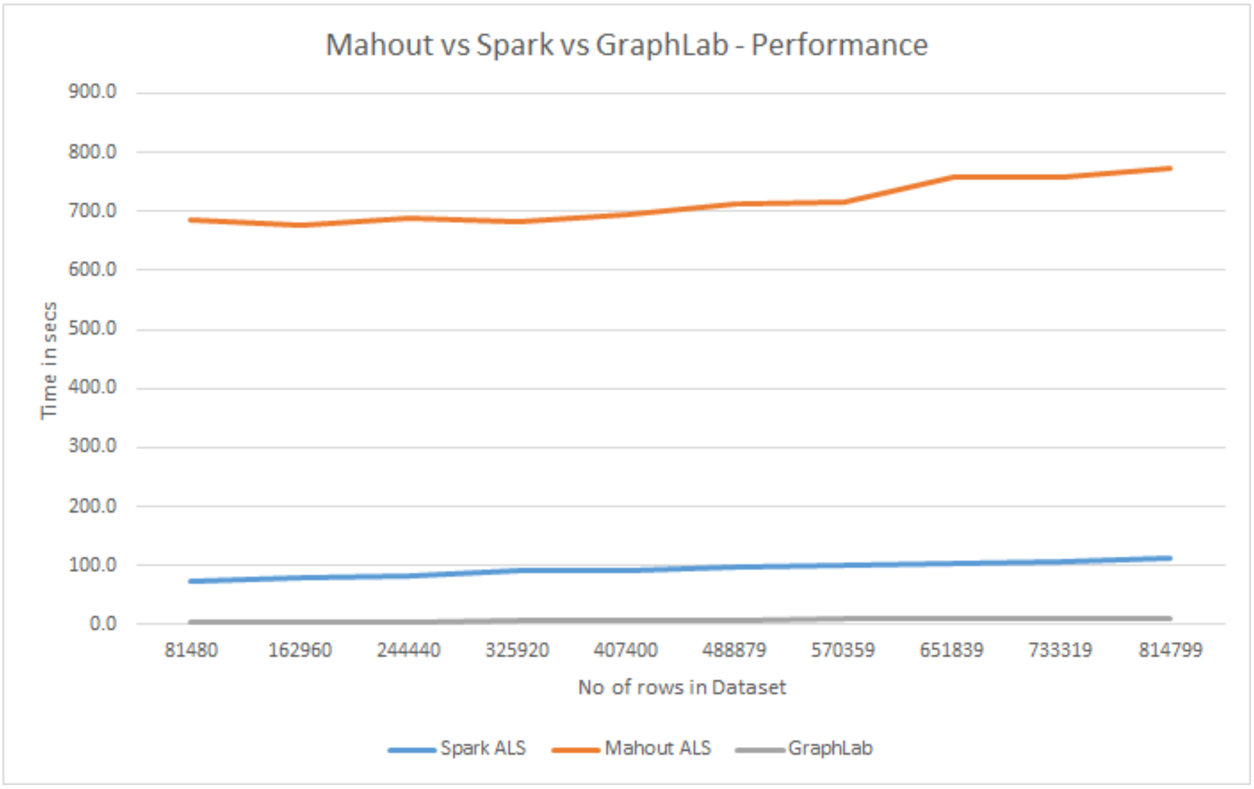

Mahout vs Spark vs GraphLab - Performance
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
Time in secs
81480
162960
244440
325920
407400
488879
570359
651839
733319
814799
No of rows in Dataset
Spark ALS
Mahout ALS
GraphLab

fig-2

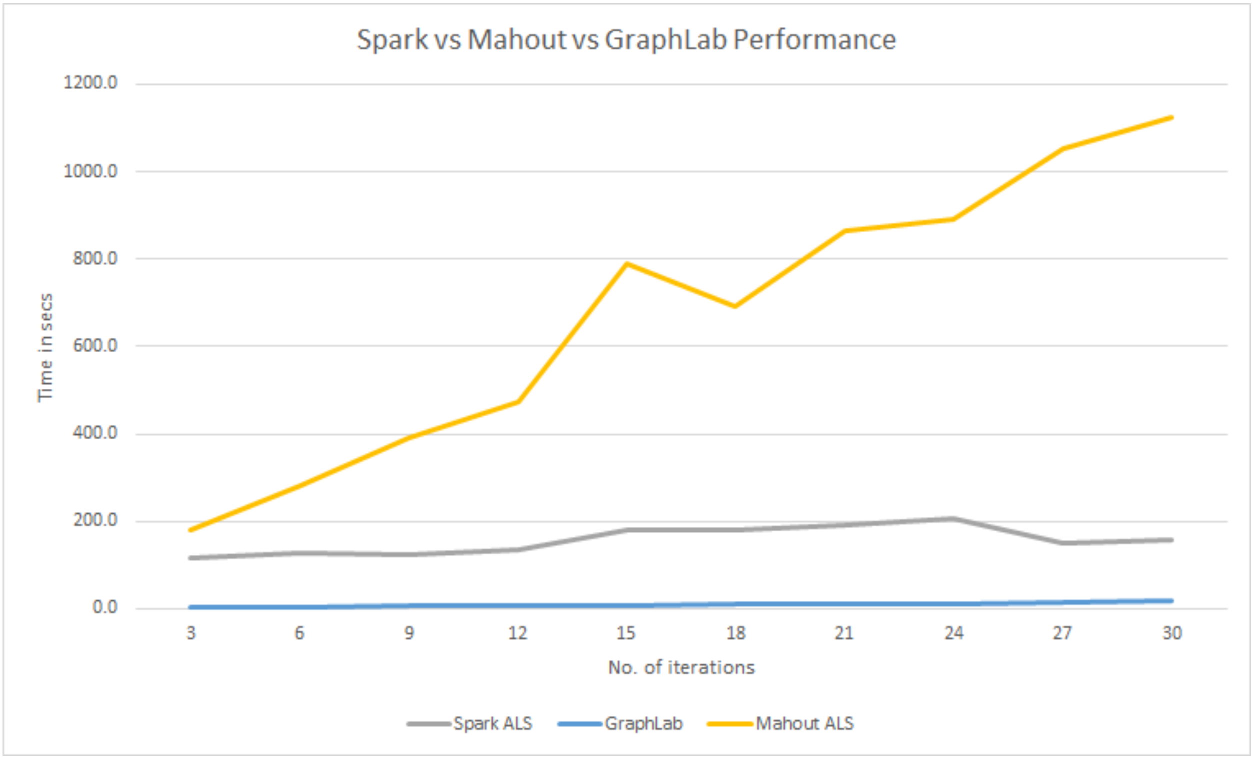
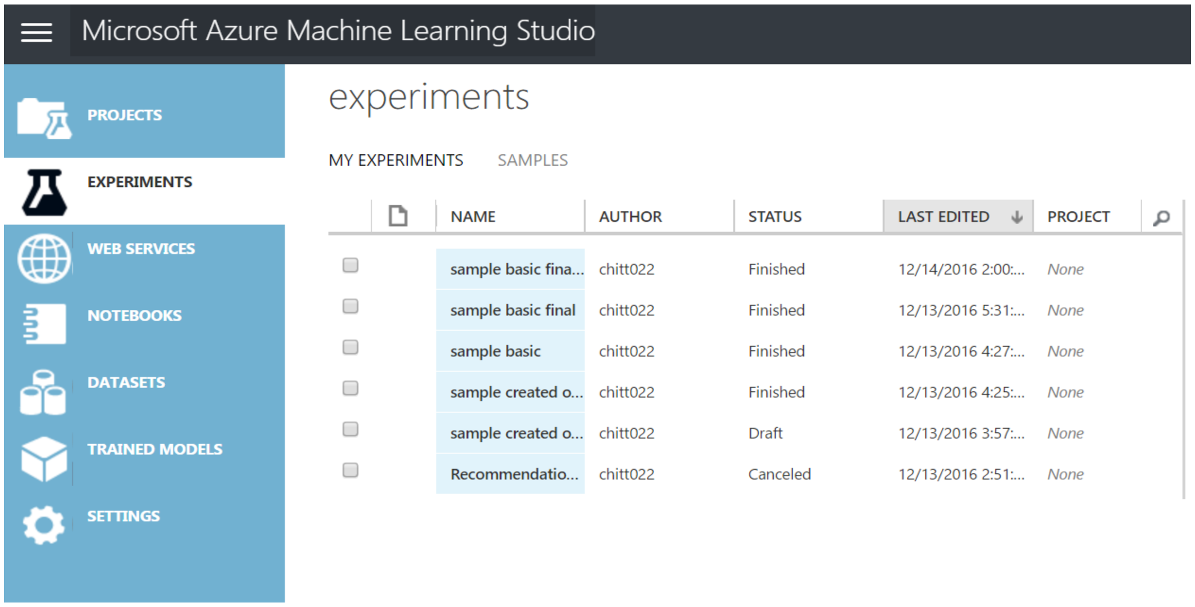


fig-3

From the above two graphs we observe that Spark MLLib outruns Mahout and the the prime reason for this would be the method of functionality. Mahout is disk based and operates with read-write functionality over and and over again which makes it time consuming as compared to Spark MLlib that is memory based and uses read once feature. Over and above this, we believe GraphLab outperforms Spark MLib owing to the use of columnar format despite being disk based.

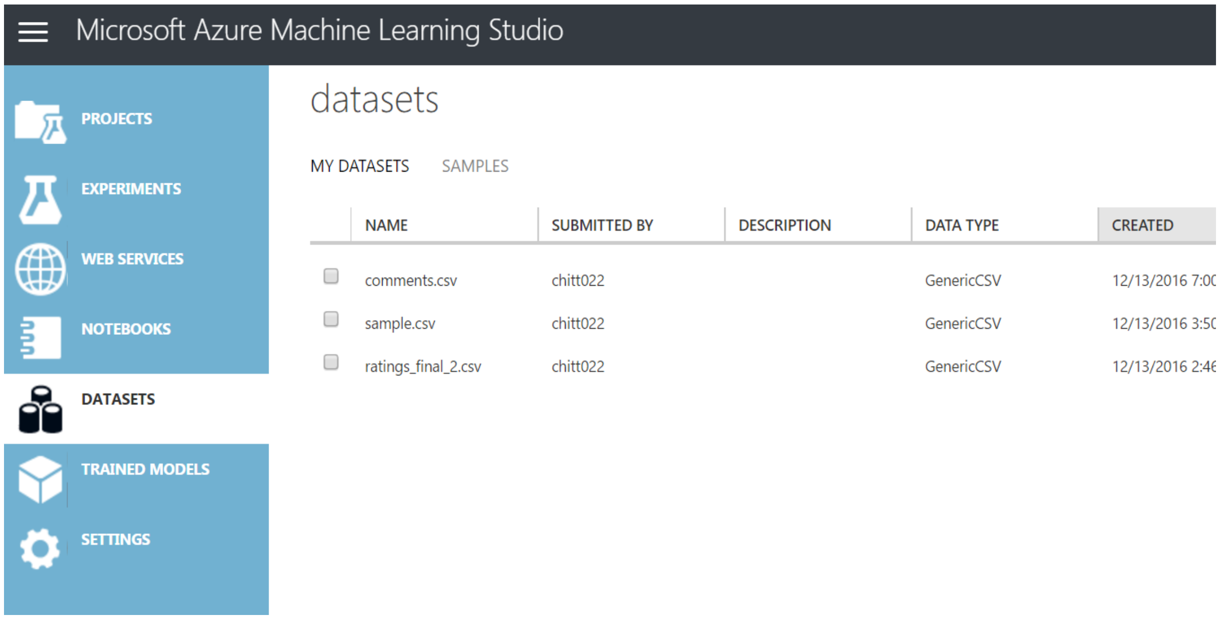
Microsoft Azure

Another way to deploy our models for Best Buy is by using a mature machine learning product made called Microsoft Azure. Azure like any other cloud computational platform, has tools that can store the data, has plethora of machine learning algorithms that support cloud computing and ample options to choose different clusters. In *fig-4*, we see the homepage of microsoft azure machine learning studio. On the left of the page, we see options like Experiments, Datasets(*fig-5*), Trained models(*fig-6*) & Web Services. Experiments is where we build a model as like a stand alone, in this phase we would not create a cluster. Datasets is where we store the data required for modeling more like S3-bucket in AWS. Trained models is where we store our built models so that we can readily use those models to predict swiftly rather than built the model again, which is time consuming task & finally Web Services is where we deploy the model built in Experiment into clusters of hadoop or spark.



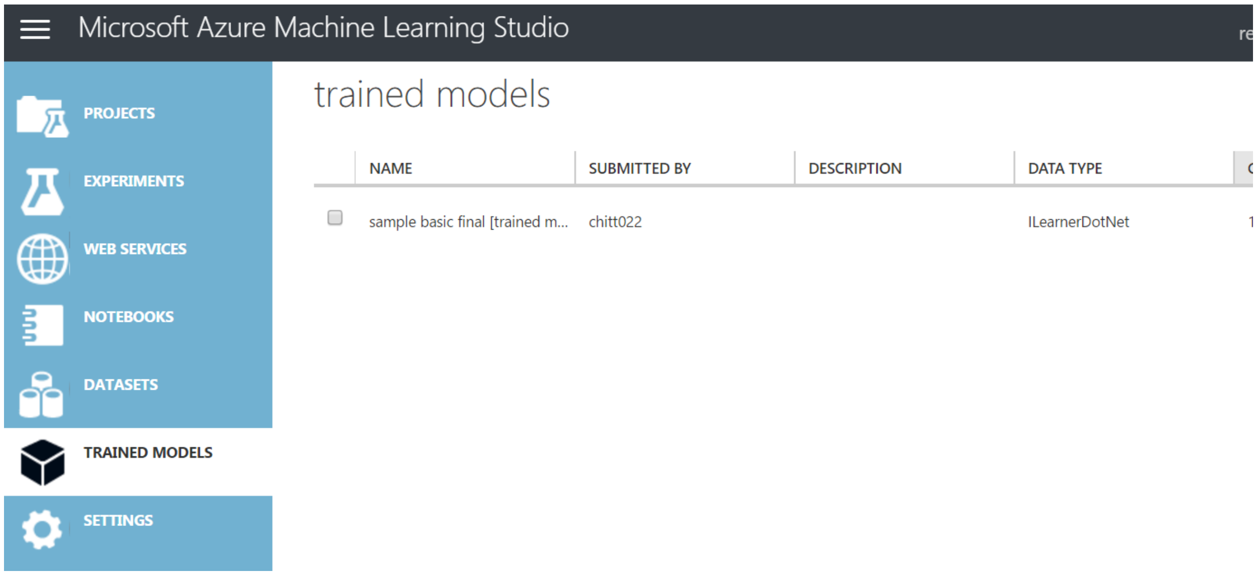
NAME	AUTHOR	STATUS	LAST EDITED	PROJECT
sample basic fina...	chitt022	Finished	12/14/2016 2:00:...	None
sample basic final	chitt022	Finished	12/13/2016 5:31:...	None
sample basic	chitt022	Finished	12/13/2016 4:27:...	None
sample created o...	chitt022	Finished	12/13/2016 4:25:...	None
sample created o...	chitt022	Draft	12/13/2016 3:57:...	None
Recommendatio...	chitt022	Canceled	12/13/2016 2:51:...	None

fig-4



NAME	SUBMITTED BY	DESCRIPTION	DATA TYPE	CREATED
comments.csv	chitt022		GenericCSV	12/13/2016 7:0
sample.csv	chitt022		GenericCSV	12/13/2016 3:5
ratings_final_2.csv	chitt022		GenericCSV	12/13/2016 2:4

fig-5

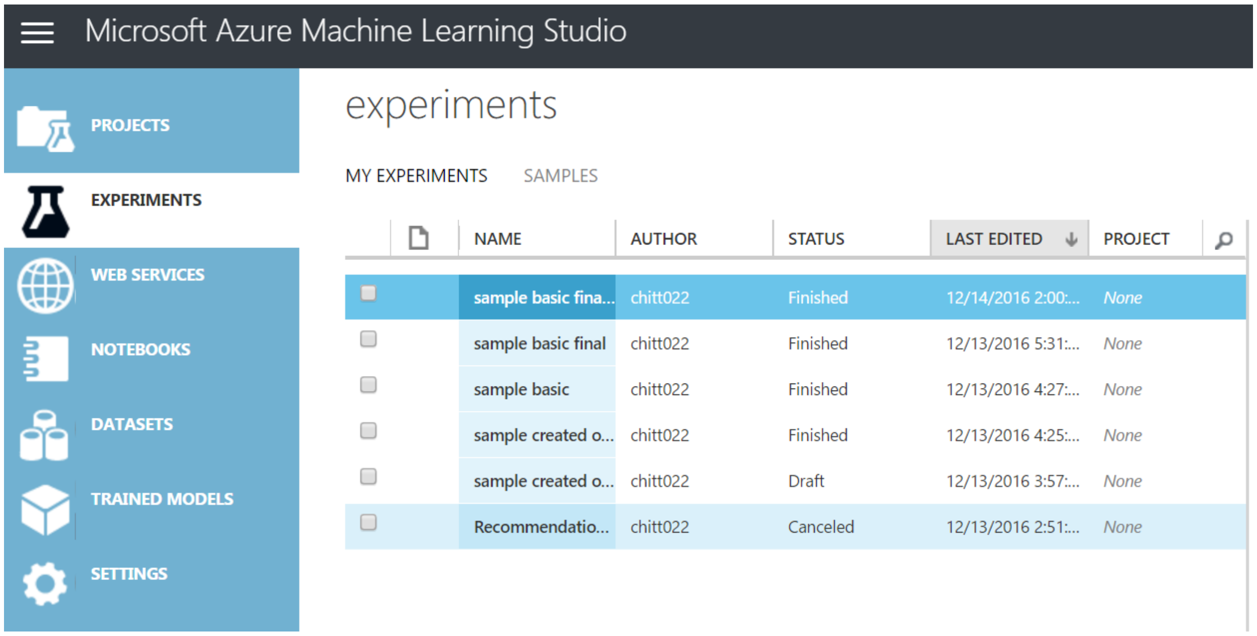


Microsoft Azure Machine Learning Studio

trained models

	NAME	SUBMITTED BY	DESCRIPTION	DATA TYPE
☐	sample basic final [trained m...	chitt022		ILearnerDotNet

fig-6



Microsoft Azure Machine Learning Studio

experiments

MY EXPERIMENTS SAMPLES

	NAME	AUTHOR	STATUS	LAST EDITED	PROJECT
☐	sample basic fina...	chitt022	Finished	12/14/2016 2:00:...	*None*
☐	sample basic final	chitt022	Finished	12/13/2016 5:31:...	*None*
☐	sample basic	chitt022	Finished	12/13/2016 4:27:...	*None*
☐	sample created o...	chitt022	Finished	12/13/2016 4:25:...	*None*
☐	sample created o...	chitt022	Draft	12/13/2016 3:57:...	*None*
☐	Recommendatio...	chitt022	Canceled	12/13/2016 2:51:...	*None*

fig-7

In *fig-7,* we see the page for the experiments where all the models built are displayed. We see metadata like Author, Status of the experiment and Name.

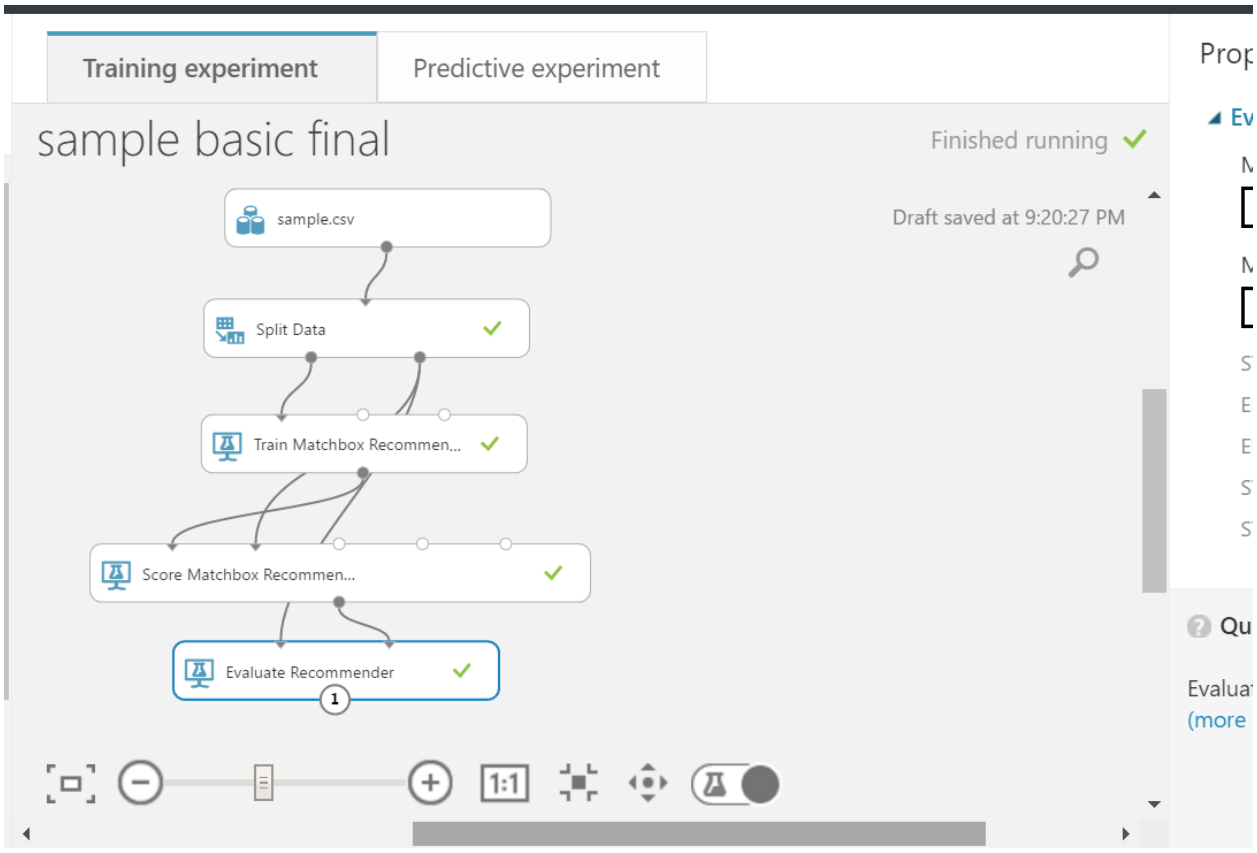
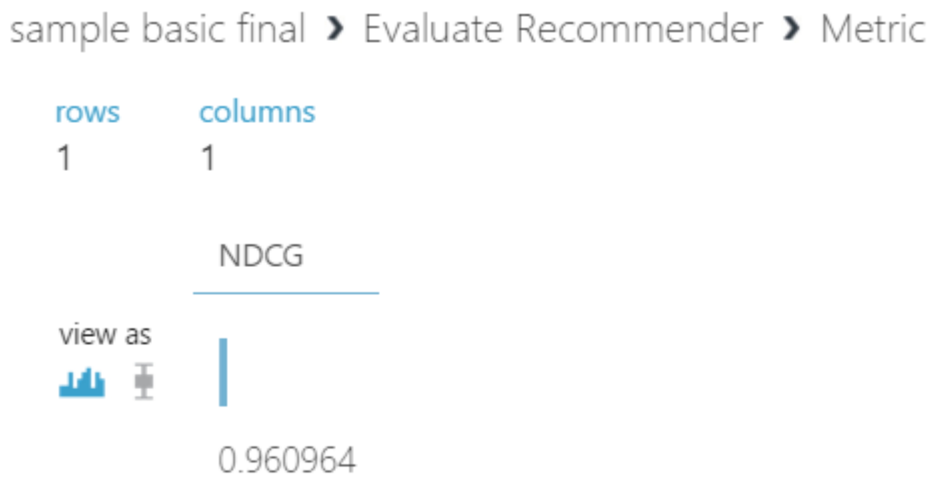


fig-8

In *fig-8,* we see a particular experiment, which has standard two tabs, one for Training experiment and the other for Predictive experiment. Training experiment tab is where we would be building the model which contains basic methodology of model building like loading the data, split the data for training and testing, fit the model, scoring and evaluation. In this particular experiment, we use item based recommendation with matchbox recommending algorithm.



sample basic final › Evaluate Recommender › Metric

rows 1 columns 1

NDCG
0.960964

fig-9

In *fig-9,* we see the metric Normalized Discounted Cumulative Gain(NDCG for top-k recommendations). Its value ranges from 0-1. The reason we got a score of 0.96 is we are testing on the data which is used to train the model. This is done just to demonstrate where in practical data mining, we would build on training dataset and test it on testing dataset

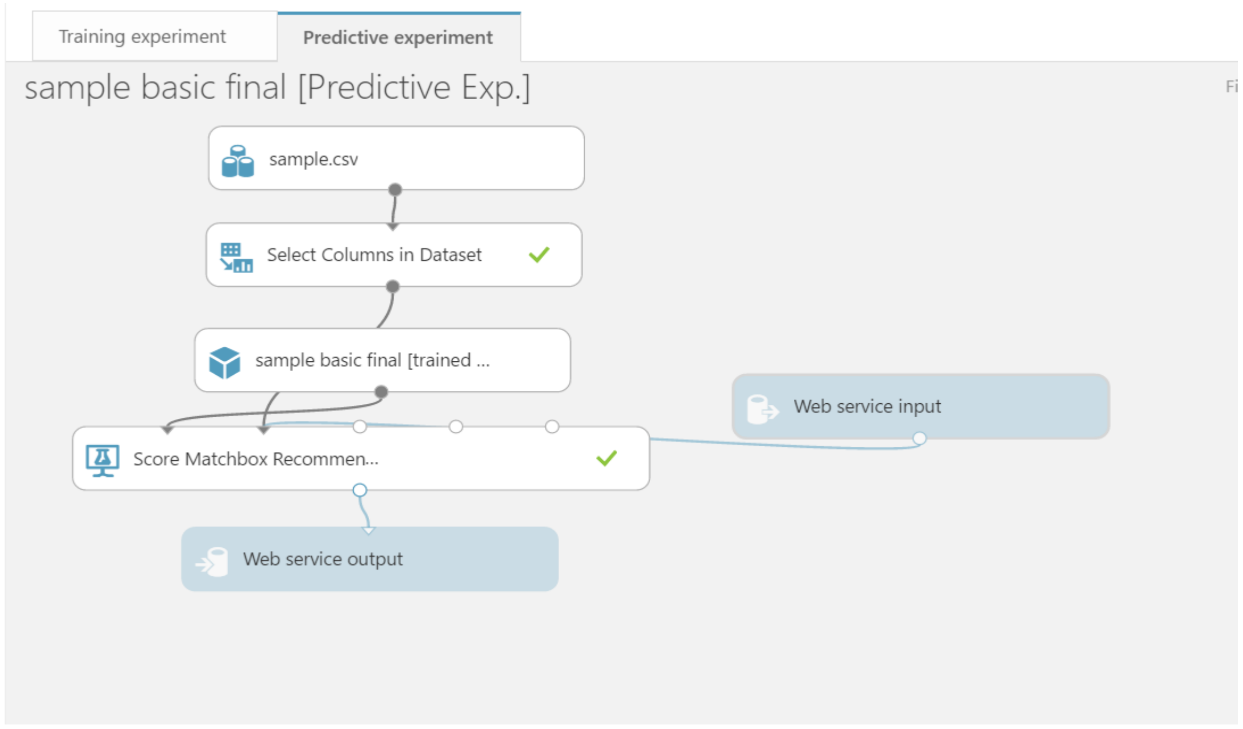
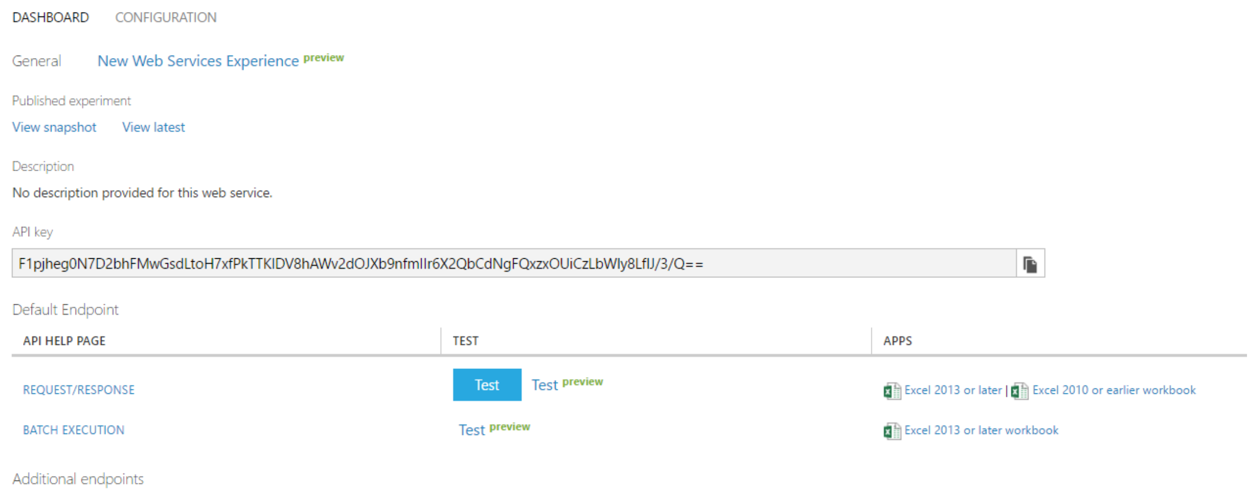

Training experiment
Predictive experiment
sample basic final [Predictive Exp.]
sample.csv
Select Columns in Dataset
sample basic final [trained ...
Web service input
Score Matchbox Recommen...
Web service output

fig-10

In *fig-10,* we test the trained model in Training experiment tab. And we can deploy the model built in the cluster. We have multiple ways to deploy the model. The one below we see in *fig-11,* is deploying as classical web service. Using the API key, we hook up the built model on any cluster we create


DASHBOARD CONFIGURATION

General New Web Services Experience preview

Published experiment

View snapshot View latest

Description

No description provided for this web service.

API key

F1pjheg0N7D2bhFMwGsdLtoH7xfPkTTKlDV8hAWv2dOJXb9nfmlIr6X2QbCdNgFQxzxOUiCzLbWly8LflJ/3/Q==

Default Endpoint

API HELP PAGE	TEST	APPS
REQUEST/RESPONSE	Test Test preview	Excel 2013 or later | Excel 2010 or earlier workbook
BATCH EXECUTION	Test preview	Excel 2013 or later workbook

Additional endpoints

fig-11

Advantages of Azure-

1. Easy to use, even a business person can work on the modeling without having to write a line of code
2. Price is competitive with other competitors like AWS, Google Cloud Platform
3. Easily scalable and has plethora of algorithms available
4. Provides cloud storage support
5. 24 * 7 Technical support available for customers
6. Azure Security Center, providing state of the art security layers for the azure tools

Conclusion

Going ahead, we believe GraphLab is one of the best tools available in market owing to low latency and at the same time has S-Frames that are column mutable as against Data Frames (that are immutable). This has been proven by the time reduction we saw for recommender system deployment which reduced by a factor of 10 from Spark Mllib and a factor of 100 from Mahout based on the number of records vs time taken plot.

We also acknowledge that GraphLab offers scalability through execution environments such as Amazon EC2 and Hadoop 2 (YARN) and these must be incorporated to handle truly big datasets i.e. datasets which cannot fit on local client disk due to SSD/HDD size limitations (~1TB).

Future Work

Content-based filtering is another approach to build recommendation system. According to Adomavicius, unlike collaborative filtering recommendation system's focus on the relationship between users and items, content-based filtering approach focus on the similarities in the items' own properties (Adomavicius, 2005). This approach regards each item as a profile, similar to how each user is regarded as a profile in user-based filtering. From an item's profile, we can obtain its key characteristics, like a movie's actors, a book's publish year, a book's description written by the manufacturer, or products' review written by the customer. Analysts can use these characteristics to calculate similarities.

The approach of content-based filtering is firstly to create target users' profiles that include their item preferences and target items' profiles that include their informative attributes and characteristics. Next, with profiles of users and items already obtained, this approach calculates the cosine distance between the vectors of the target user and the target item, and estimates how the user would be interested in the item based on that cosine distance.

Right now, our data set only includes userId, sku, rating, and click time. In the future work, as we gain more informative attributes and as our experience with recommendation systems increases, this would be a great direction to do the content-based filtering. Through this approach, we could obtain a better recommendation system. Something to note is the content attributes can be expensive to gather; however, using the Best Buy app allows for the access and collection of data easily and the costs of this method can be decreased.

Besides recommendation methods, also future work could include different machine learning environments such as Vowpal Wabbit and H2O. The Vowpal Wabbit (VM) is an open-source fast learning system that was built originally at Yahoo! Research and then matured at Microsoft Research as a fast, scalable, and useful learning algorithm. According to VM's home wiki, this learning system is fast and scalable enough to learn a "tera-feature data_set on 1000 nodes in one hour". And it is an out-of-core online learn, which means we do not need to load all data into local memory. Vowpal Wabbit is a useful machine learning system that supports many attribute reductions, including optimization algorithms and importance weighting. This system is also easy to use, as the only external source it needs is on the boost library.

Another tool for future work gaining excitement is, H2O.ai. H2O.ai is another fast and scalable machine learning system. Its goal is to provide users with analytical tool to do cloud computing. According to H2O.ai's home page, it is so scalable and flexible that users can train a model on complete datasets, instead of just a sample of it, and can connect data easily with Excel, R, and SQL. H2O can be used to analyze datasets not only in cloud computing systems and Apache Hadoop, but also in Linux, macOS, and Windows. H2O also uses iterative methods that can process data analysis quickly to give users real-time answers.

Bibliography

Analytics Vidhya. Https://www.analyticsvidhya.com/blog/author/sunil-ray/. "Tutorial – Getting Started with GraphLab For Machine Learning in Python." N.p., 19 Dec. 2015. Web. 14 Dec. 2016. <https://www.analyticsvidhya.com/blog/2015/12/started-graphlab-python/>.

Apache. "Creating a User-Based Recommender in 5 Minutes." Apache Mahout: Scalable Machine Learning and Data Mining. N.p., 2014. Web. 14 Dec. 2016. <https://mahout.apache.org/users/recommender/userbased-5-minutes.html>.

Apache. "What Is Apache Mahout?" Apache Mahout: Scalable Machine Learning and Data Mining. N.p., 6 June 2016. Web. 13 Dec. 2016.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms". GroupLens Research Group/Army HPC Research Center. 2001. http://files.grouplens.org/papers/www10_sarwar.pdf

Cook, Steve, Mahout User Recommender Tutorial with Eclipse and Maven. Perf. Steve Cook. Youtube, 16 Mar. 2014. Web. <https://www.youtube.com/watch?v=63k560Livmg>.

G Sterling, "All digital growth now coming from mobile usage - comScore", Marketing Land. 2016. http://marketingland.com/digital-growth-now-coming-mobile-usage-comscore-171505

G. Adomavicius and A. Tuzhilin, "Towards the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," IEEE Trans. on Data and Knowledge Engineering 17:6, pp. 734– 749, 2005. http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

H2O.ai, "Why H2O?". http://www.h2o.ai/h2o/

K. Cheng, "Vowpal Wabbit" https://github.com/JohnLangford/vowpal_wabbit/wiki

J. Wang, A. Vries, M. Reinders, "Unifying User-based and Item-based Collaborative Filtering Approached by Similarity Fusion". Information and Communication Theory Group. 2006. http://siplab.tudelft.nl/sites/default/files/sigir06_similarityfusion.pdf

Z-nation. "Turi-code/tutorials." GitHub. N.p., 30 July 2016. Web. 14 Dec. 2016. <https://github.com/turi-code/tutorials/blob/master/notebooks/introduction_to_sframes.ipynb>.

Matchbox algorithm

https://msdn.microsoft.com/en-us/library/azure/dn905987.aspx

APPENDIX:

MAHOUT

Mahout Loading Data:

```
CREATE TABLE ratings (user STRING, sku STRING, rating FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Load the data into the ratings table:

```
LOAD DATA LOCAL INPATH '/home/training/project/ratings_final.csv' INTO TABLE ratings;
```

Create the 10%, 20%, to 100% dataset samples:

```
INSERT INTO TABLE ten_perc
  SELECT * FROM ratings
  LIMIT 81480;
```

Create recommender and transfer the data to local:

```
mahout recommenditembased --input /user/hive/warehouse/ten_perc/* --tempDir temp_proj --
output ratings_repository --similarityClassname SIMILARITY_PEARSON_CORRELATION -n
5
```

```
hadoop fs -getmerge ratings_repository ratings_recommended.txt
```

Create ALS recommender system procedure (standard iterations set at 20):

```
mahout parallelALS --input /user/hive/warehouse/ten_perc/* --output als_ratings --lambda 0.1 --
implicitFeedback true --alpha 0.8 --numFeatures 2 --numIterations 20 --tempDir tmp
```

```
mahout recommendfactorized --input als_ratings/userRatings/* --userFeatures $als_ratings/U/ --itemFeatures $als_ratings/M/ --numRecommendations 5 --output recommendations --maxRating 5
```

Mahout Eclipse Code:

```
package com.userprediction.recommendApp;

import java.io.File;
import java.io.IOException;
import java.util.List;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;
import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
import org.apache.mahout.cf.taste.recommender.RecommendedItem;
import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;

/**
 * Hello world!
 *
 */
public class App
{
	public static void main( String[] args ) throws Exception
	{
		DataModel model = new FileDataModel(new File("Data/ratings_final.csv"));
		UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
		UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, 
similarity, model);
		UserBasedRecommender recommender = new 
GenericUserBasedRecommender(model, neighborhood, similarity);
		for(LongPrimitiveIterator users=model.getUserIDs();users.hasNext();)
		{
			long userId = users.nextLong();
			List<RecommendedItem> recommendations = 
recommender.recommend(userId,5);
			for (RecommendedItem recommendation : recommendations) {
				System.out.println(recommendation);
			}
		}
	}
}
```

<u>SPARK MLLIB</u>

Time comparison for number of records:

```
sc
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
data = sc.textFile("file:/home/training/final project/train.csv")
data1=data.filter(lambda l: not l.startswith("user")).map(lambda line:(line.split(",")))
data2=data1.map(lambda l:(float(l[0].decode("utf-8")),float(l[1].decode("utf-
8")),float(l[2].decode("utf-8"))))
train_ratings = data2.map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
x,y = train_ratings.randomSplit(0.8,0.2)
from pyspark.mllib.recommendation import ALS
import time
start_time=time.time()
rank = 10
numIterations = 20
model = ALS.train(x, rank, numIterations)

# Evaluate the model on training data1
testdata1 = x.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata1).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = x.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).reduce(lambda x, y: x + y) /
ratesAndPreds.count()
print("Mean Squared Error = " + str(MSE))
print "Time taken to run the code {:.3f} seconds".format(time.time() - start_time)
```

Time comparison for number of iterations:

```
sc
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
data = sc.textFile("file:/home/training/final project/train.csv")
data1=data.filter(lambda l: not l.startswith("user")).map(lambda line:(line.split(",")))
data2=data1.map(lambda l:(float(l[0].decode("utf-8")),float(l[1].decode("utf-
8")),float(l[2].decode("utf-8"))))
train_ratings = data2.map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
from pyspark.mllib.recommendation import ALS
import time
start_time=time.time()
rank = 10
```

```
numIterations = 20
model = ALS.train(train_ratings, rank, numIterations)

# Evaluate the model on training data1
testdata1 = train_ratings.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata1).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = train_ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).reduce(lambda x, y: x + y) /
ratesAndPreds.count()
print("Mean Squared Error = " + str(MSE))
print "Time taken to run the code {:.3f} seconds".format(time.time() - start_time)
```

<u>**GRAPHLAB**</u>

Time comparison for number of records:

```
import graphlab
#graphlab.product_key.get_product_key()
graphlab.product_key.set_product_key('XXXX')

import os
import random
os.chdir("A:/Carlson Minnesota/Fall 2016/MSBA 6330 Harvesting Big Data/Final
Project/BestBuy/factorization")

import time

fraction=range(1,11,1)
fraction = [float(x)/10 for x in fraction]

run_time=[]

for num in fraction:
   sf = graphlab.SFrame('ratings_final.csv')
   sf = sf.sample(num, seed=1)
   start_time = time.time()
   model = graphlab.recommender.factorization_recommender.create(observation_data=sf,
                                          user_id="user",
                                          item_id="sku",
                                          target="rating",
                                          solver="als",
                                          max_iterations=20,
                                          random_seed=1)
```

```
    run_time.append(time.time() - start_time)
    print num
    print "--- {:f} seconds ---".format(time.time() - start_time)
```

Time comparison for number of iterations:

```
import graphlab
#graphlab.product_key.get_product_key()
graphlab.product_key.set_product_key('891F-C8AB-72E4-3C27-2960-8C37-14E2-227A')

import os
import random
os.chdir("A:/Carlson Minnesota/Fall 2016/MSBA 6330 Harvesting Big Data/Final Project/BestBuy/factorization")

sf = graphlab.SFrame('ratings_final.csv')

import time

numIters=range(3,33,3)

run_time=[]

for numIter in numIters:
    start_time = time.time()
    model = graphlab.recommender.factorization_recommender.create(observation_data=sf,
                                                    user_id="user",
                                                    item_id="sku",
                                                    target="rating",
                                                    solver="als",
                                                    max_iterations=numIter,
                                                    random_seed=1)
    run_time.append(time.time() - start_time)
    print numIter
    print "--- {:f} seconds ---".format(time.time() - start_time)
```

--*END*--
