

**MSBD 5003 Course Project Report**  
**Semantic-based clothing size and fitting recommendation System**  
**Group 2**

Mingjun Guo  
20527755

Brian LAM  
20576299

## **1. Abstract**

In order to leverage a competitive advantage in highly competitive E-Commerce market, exploiting data for clothing recommendation and size prediction to customers is crucial asset for e-commerce giants, which can increase income by enhancing the shopping experience and saving cost by reducing return rates. Therefore, In this project, we make full use of ***RentTheRunway*** public dataset to do knowledge discovery, which can give insights for companies such as eBay and Amazon. We implemented a group of cutting edge technologies to develop a robust Spark ecosystem, which is composed of three parts: data storage, backend system and front-end application(in future development). In general, there are three parts of our system architecture. The first part is a database or data warehouse for data storage. To be more specific, we used MongoDB as our database structure, Spark SQL for data preprocessing and exploratory data analysis (EDA), Spark MLLib for recommendation system and size prediction, and AWS Cloud to implement cloud computing.

Key MileStone:

1. Predicting size based on the feature extracted from customer reviews by NLP
2. Implement the Recommendation System based on the rating data and clothing data by ALS
3. Build Spark ecosystem based on three parts: data storage, backend system and front-end application by making full use of Spark technologies.

## **2. Datasets**

The datasets contain self-reported information and feedback from shoppers, such as reviews, rating, size purchased, fitting reviews and customers measurements (weight, height, and body size measurement, etc) from ***RentTheRunway***. ***RentTheRunway*** is an online platform for clothing rentals. There are 3 areas of features, customer features (ie. customer information), product features (ie. product information), and text features (ie. written reviews). The fit review is categorical data of small, fit, and large.

The original dataset of rent the runway is a size of 192,544 x 15 matrix with our users, items, and the feedback data. We can use the user and item features to approximate the interaction between the users and items relationship, which is the weights on how each user and item relates to each feature. By using the matrix factorization, our goal is to predict the missing values .

### Product features from RentTheRunway

age	body type	bust size	category	fit	height	item_id	rating	rented for	review_date	review_summary	review_text	size	user_id	weight
30.0	hourglass	32ddd/e	dress	small	5' 9"	248116	10.0	everyday	December 12, 2017	Great dress - sizing is arbitrary. I wore the ...	XL was fitted and the slit came to mid thigh. ...	26	906037	155lbs
26.0	hourglass	32d	dress	fit	5' 7"	248116	8.0	party	November 21, 2017	I wore this for Friendsgiving with my fashion ...	I really liked this outfit! I ended up renting...	20	73673	NaN
38.0	athletic	32c	dress	fit	5' 6"	248116	10.0	everyday	November 30, 2017	So COMFY!	This dress is like wearing pajamas, but so muc...	14	967716	134lbs
35.0	athletic	34dd	dress	large	5' 3"	248116	8.0	party	December 4, 2017	Husband loved it!	The large was just a tad too big... A medium w...	20	488736	160lbs
41.0	full bust	38ddd/e	dress	fit	5' 7"	248116	10.0	vacation	December 30, 2017	Cute, comfortable and incredibly versatile.	I rented this dress for a holiday trip to Sout...	14	153908	160lbs

The project focus on text features, the table above is the review of an item with 5 reviews, given 3 fits, 1 small and 1 large.

The first customer rates the fitting run **small** with a written review,

*"XL was fitted and the slit came to midhigh. Very flattering doesn't even though I typically am a medium!"*

The second customer rate the fitting as **fitted** and gave a review,

*"I really liked this outfit! I ended up renting the XL and L, but actually could have gotten away with a Medium potentially. However, the dress hugged all the right curves - I got a lot of compliments on it throughout the night. It's a bit of a 90s throwback with the metallic and the stripe and worked with my look that night."*

The fourth customer rates the fitting as **large** and wrote,

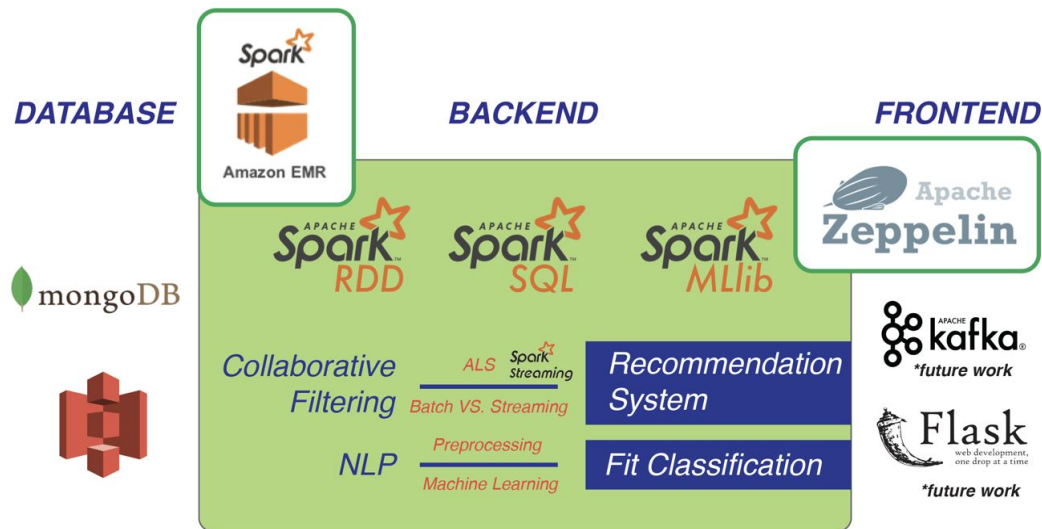
*"The large was just a tad too big ... A medium would have been more flattering and less baggy, but still loved it"*

The above reviews showed one single item, 3 different perspectives on size fittings and all of the written reviews discussed on size fittings.

In addition to the text and rating relationships, we may use the recommending products that have been rated in the past by the shopper. A common approach to solve this is through matrix factorization, given the ratings that user have rated certain items and predicting their rating for the rest. The interactive user-item matrix is constructed as, given n users and m items. The (n, m) matrix has a high sparsity which represent the missing ratings.

### 3. System Architecture and Technology

In this project, we implemented a group of cutting edge technologies to build big data architecture by taking advantage of each component to develop a robust Spark ecosystem. In general, there are three parts of our system architecture. The first part is a database or data warehouse for data storage. The second part is the backend system, where we implement our Spark mainframe, and the last part is the front-end application for data visualization.



#### 3.1. MongoDB

For this project, MongoDB is used as our database structures. MongoDB is open source document-oriented schema-less NoSQL database with the advantage of high flexibility and scalability and performance on large scale dataset, which stores data in the form of key-value pair. MongoDB overcomes the static and scalability challenge in the relational database. The key feature of MongoDB includes the minimize the input-output operation to achieve a high performance like faster queries, which can be ten times faster than MySQL in operation. Another key feature of MongoDB is the auto replication feature. As a result, the database has high availability. The auto replication feature provides an automatic failover mechanism. Data is restored through a backup copy if any failures. MongoDB is useful when working with big data when the data's nature does not require a relational model. The database objective is to store and retrieve large quantities of data, not the relationship between the elements. Compare with SQL/RDBM, MongoDB use different terms, such as collection in MongoDB is similar to table in SQL, document is similar to row, field is similar to column, it has a schema-less structure while RDBM is schema-oriented.

In summary, the following are the key advantage of MongoDB

1. MongoDB is a schema-less database, therefore no schema migrations needed, and the code defines the schema.

2. MongoDB stores data in binary JSON (BSON) format, which helps to store data in a rich way which capable of holding arrays and other documents and derive document-based data model.
3. The document query language is a crucial factor in supporting dynamic queries.
4. Compare with a relational database, performance tuning in MongoDB is more manageable, and faster access to data due to the use of internal memory for storage.

```
# mongoimport --db dbName --collection collectionName --file fileName.json
mongoimport --db fittings --collection renttherunway --file renttherunway_final_data.json
```

localhost:27017 (STANDALONE) MongoDB 4.0.3 Community

fittings.renttherunway DOCUMENTS 192.5k TOTAL SIZE 118.2MB AVG. SIZE 643B INDEXES 1 TOTAL SIZE 1.7MB AVG. SIZE 1.7MB

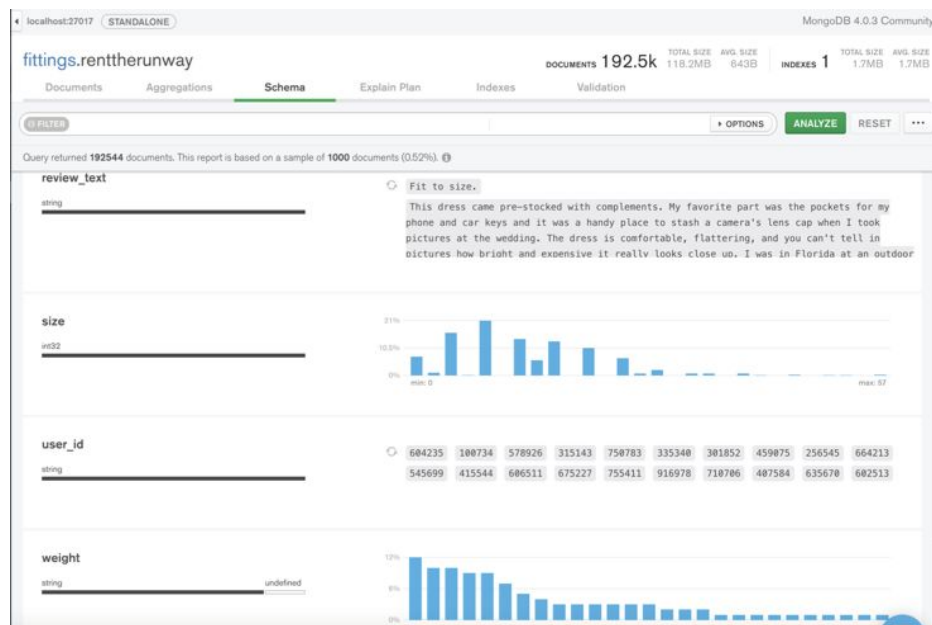
Documents Aggregations Schema Explain Plan Indexes Validation

0 FILTER OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 20 of 192544

```
{
  "_id": "ObjectId('5cc5689900de574e7486d26d')",
  "fit": "fit",
  "age": "116",
  "size": 4,
  "item_id": "18637651",
  "rating": "10",
  "rented_for": "party",
  "review_text": "This hugged in all the right places! It was a perfect dress for my eve...",
  "review_summary": "It was a great time to celebrate the (almost) completion of my first y...",
  "category": "shoath",
  "height": "5' 4\"",
  "user_id": "368448",
  "review_date": "December 14, 2015"
}
```

```
{
  "_id": "ObjectId('5cc5689900de574e7486d26d')",
  "fit": "fit",
  "user_id": "989926",
  "bust_size": "34c",
  "item_id": "126335",
  "weight": "135lbs",
  "rating": "8",
  "rented_for": "formal affair",
  "review_text": "I rented this for my company's black tie awards banquet. I liked that...",
  "body_type": "pear",
  "review_summary": "Dress arrived on time and in perfect condition. ",
  "category": "dress",
  "height": "5' 5\"",
  "size": 8,
  "age": "34",
  "review_date": "February 12, 2014"
}
```



MongoDB is an excellent place for data storage along with high flexibility in data modeling and scaling capabilities, in addition, they provide MongoDB Connector for Apache Spark to integrate data into another system, such that we can connect our database to the Spark ecosystem seamlessly. With the connector, all of the Spark libraries in SparkSQL, Spark streaming and Spark MLLib API can be used with MongoDB dataset. As a result, we can leverage the power of MongoDB and take advantage of MongoDB's aggregation pipeline and secondary indexes to extract, filter, and process the range of data we need. Another key advantage is the MongoDB Connector can co-locate RDD with the source MongoDB node. Therefore data movement across the cluster are, and the latency is reduced, and the performance is maximized across large, distributed datasets.

```
my_spark = SparkSession.builder.appName("myApp") .config("spark.mongodb.input.uri",
    "mongodb://localhost:27017/fitting.renttherunway") .config("spark.mongodb.output.uri",
    "mongodb://localhost:27017/fitting.renttherunway") .getOrCreate()

df_renttherunway = spark.read.format("com.mongodb.spark.sql.DefaultSource").load()
```

### 3.2. Exploratory Data Analysis

There are 105,508 unique user and 5850 unique item in the rent the runway dataset, then we create the interactive matrix to see the relationship between the users and items. The red dot below are the low rating (2~6), while the white dot are the high rating review (6 ~ 10). When we check the violinplot, we see that over 80% of the rating are 8 and 10, which is a unbalanced distribution.

```
# read the data
items_file_name = 'items.csv'
ratings_file_name = 'ratings.csv'

items = spark.read.csv(movies_file_name, header=True, inferSchema=True).repartition(10).cache()
ratings = spark.read.csv(ratings_file_name, header=True, inferSchema=True).repartition(10).cache()

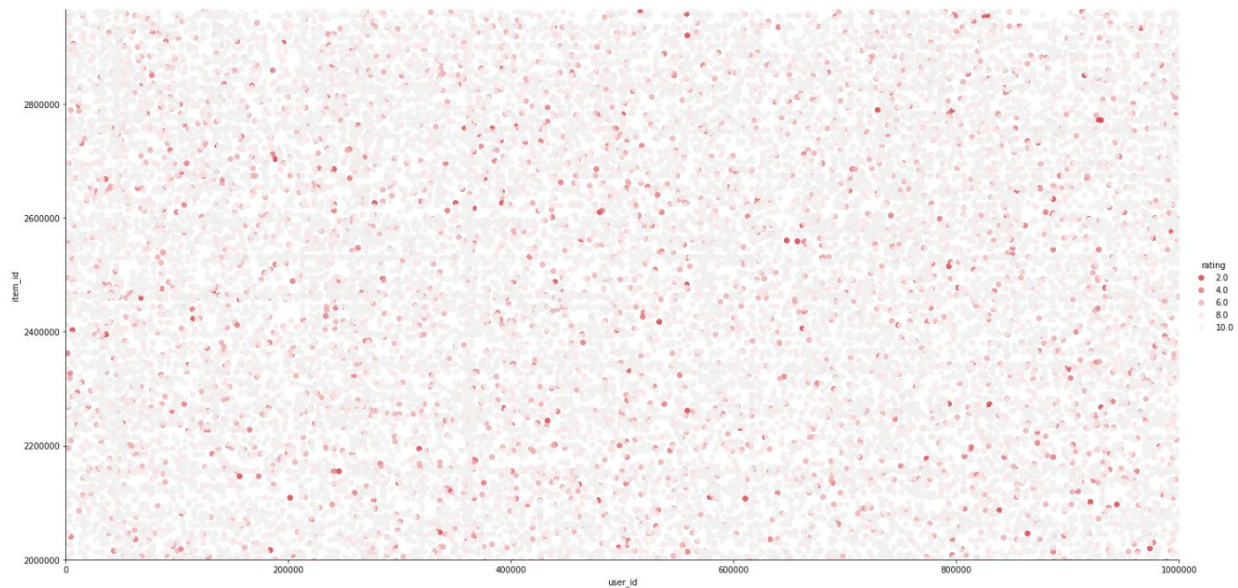
# use specific methods from the DataFrame API to compute any statistic:
print("Number of different users: " + str(ratings.select('user_id').distinct().count()))
print("Number of different items: " + str(ratings.select('item_id').distinct().count()))
print("Number of items with at least one rating strictly higher than 4: " + str(ratings.filter('rating >
4').select('item_id').distinct().count()))
```

```
Number of different users: 105508
Number of different items: 5850
Number of items with at least one rating strictly higher than 4: 5840
```

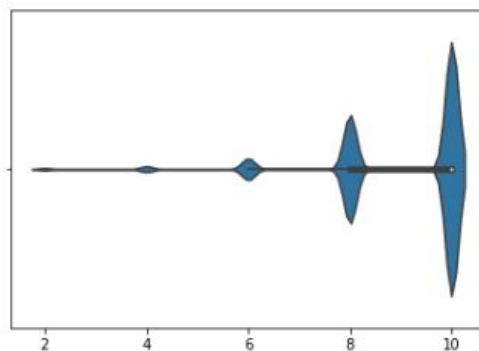
```
# Create a graph of the items reviewed by users:
# to do some visualization:
import seaborn as sns
%matplotlib inline

ratingsPandas = ratings.toPandas()
sns.lmplot(x='user_id', y='item_id', data=ratingsPandas, fit_reg=False)

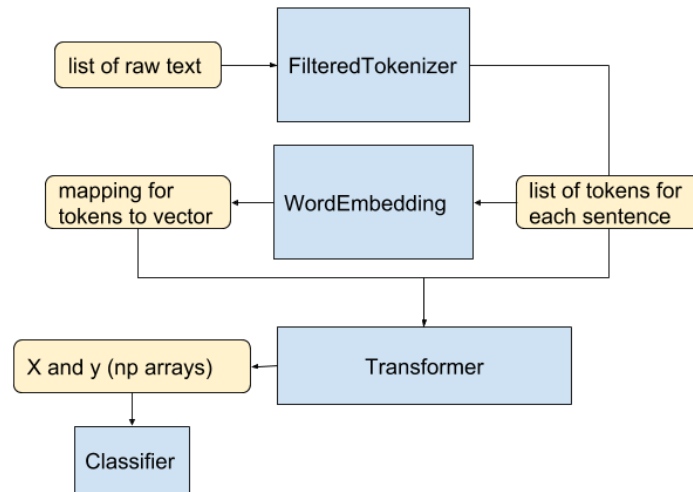
# Create the graph on a larger scale with the color palette:
lm = sns.lmplot(x='user_id', y='item_id', hue='rating', data=ratingsPandas, fit_reg=False, size=10,
aspect=2, palette=sns.diverging_palette(10, 133, sep=80, n=10))
axes = lm.axes
axes[0,0].set_ylim(2000000,2966087) # max movieId is 163949
axes[0,0].set_xlim(9,999997) # max userId is 999997
lm
```



```
sns.violinplot([ratingsPandas.rating])
```



### 3.3. Natural Language Processing and Review Prediction



The figure above shows the process flow of the preprocessing steps to process text and convert it into embedding vectors for the sentences. First, sentences in the form of list are filtered and tokenized. Each sentence is broken down into words that are called “tokens” with some words are filtered out including stop words, and some words with very high frequency (e.g. ‘this’). Then, each token is passed to the CountVectorizer function to retrieve its corresponding word vector in the word embedding. It is noticed that not every token is within the vocabulary of the embedding and therefore the embedding coverage percentage is recorded so as to know how many tokens can be represented by vectors and fed to the machine learning models.

After this process, the embedding coverage and the mapping of tokens to vectors are obtained. Then, a sentence vector from a group of token vectors for each sentence (data) is aggregated. This is done by TF-IDF aggregation. The final output will be a one-hot vector  $y$  for the class label and a matrix with each row representing a data item (sentence), each column representing one dimension in the word embedding. Thus  $X$  and  $y$  are ready to be used for machine learning prediction.

StringIndexer encodes a string column of labels to a column of label indices. The indices are in  $[0, \text{numLabels})$ , ordered by label frequencies, so the most frequent label gets index 0. In our case, the label column (fitting) will be encoded to label indices, as 0, 1 and 2, which represent small, fit, and large.

The Spark Action then fit into the pipeline and then fit into tokenizer, remove stopword, TFIDF and clean up.

```
tokenizer = Tokenizer(inputCol = 'review_text', outputCol = 'token_text')
stopremove = StopWordsRemover(inputCol = 'token_text', outputCol = 'stop_tokens')
count_vec = CountVectorizer(inputCol = 'stop_tokens', outputCol = 'c_vec')
idf = IDF(inputCol = 'c_vec', outputCol = 'tf_idf')
fitting = StringIndexer(inputCol = 'fit', outputCol = 'label')
```

```

clean_up = VectorAssembler(inputCols = ['tf_idf', 'rating'], outputCol = 'features')
data_prep_pipe = Pipeline(stages = [fitting, tokenizer, stopremove, count_vec, idf, clean_up])
cleaner = data_prep_pipe.fit(reduced_rtf)
clean_data = cleaner.transform(reduced_rtf)
clean_data = clean_data.select(['label', 'features'])

```

We use the Naive Bayes algorithm with default parameters from Spark MLlib API to obtain an accuracy of 65%. The experiment gives us the simple machine learning prediction for large, fit and small.

```

nb = NaiveBayes()
(training, testing) = clean_data.randomSplit([0.8, 0.2])
fit_predictor = nb.fit(training)
test_results = fit_predictor.transform(testing)

```

label	features	rawPrediction	probability	prediction
0.0	(107013,[0,1,2,3,...])	[-4210.1948825103...	[2.07011284677129...	1.0
0.0	(107013,[0,1,2,3,...])	[-3004.6612366149...	[0.42612663852947...	1.0
0.0	(107013,[0,1,2,3,...])	[-3203.7236270279...	[1.0,7.1125810876...	0.0
0.0	(107013,[0,1,2,3,...])	[-2314.8327388251...	[1.96536040484580...	2.0
0.0	(107013,[0,1,2,3,...])	[-1458.7802218494...	[0.05671706275691...	1.0
0.0	(107013,[0,1,2,3,...])	[-4745.5781881733...	[1.36727904145003...	2.0
0.0	(107013,[0,1,2,3,...])	[-4446.9327243278...	[0.00108476900342...	1.0
0.0	(107013,[0,1,2,3,...])	[-3583.7059927837...	[0.05956898171349...	1.0
0.0	(107013,[0,1,2,3,...])	[-6087.9267073282...	[1.0,7.3184471040...	0.0
0.0	(107013,[0,1,2,3,...])	[-1465.2141909514...	[0.99178983710788...	0.0
0.0	(107013,[0,1,2,3,...])	[-2592.7496378523...	[0.99999807441492...	0.0
0.0	(107013,[0,1,2,3,...])	[-1713.9047599549...	[6.36535660787685...	1.0
0.0	(107013,[0,1,2,3,...])	[-2612.3930840804...	[6.84794990056387...	2.0
0.0	(107013,[0,1,2,3,...])	[-1803.3229045179...	[0.99999991451658...	0.0
0.0	(107013,[0,1,2,3,...])	[-1048.7762927945...	[1.41766953614844...	2.0
0.0	(107013,[0,1,2,3,...])	[-2469.4718800336...	[0.99999999999996...	0.0
0.0	(107013,[0,1,2,3,...])	[-5233.3105022071...	[0.00604081270022...	1.0
0.0	(107013,[0,1,2,3,...])	[-4131.1857950066...	[8.35894885847986...	1.0
0.0	(107013,[0,1,2,3,...])	[-3114.0678477844...	[0.99999846561018...	0.0
0.0	(107013,[0,1,2,3,...])	[-1115.2084327021...	[5.25732109795948...	2.0

only showing top 20 rows

### 3.4. Recommendation System

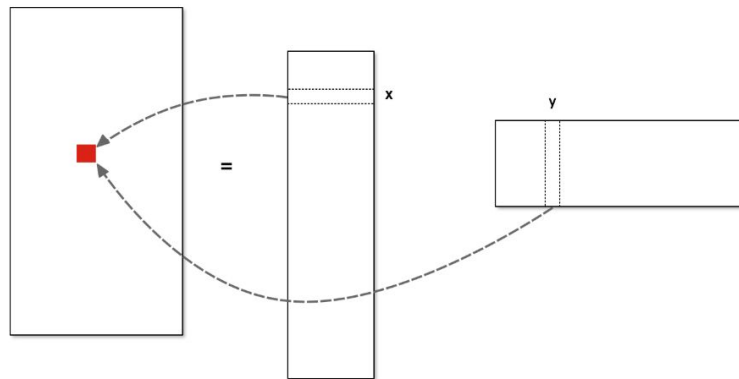
Recommendation engines are a popular method to match users, products and historical data on user behaviour. In most cases, when referring to the Recommendation system, we will think about the collaborative filtering. The “collaborative” aspect refers to the fact that we are using collective information from a group of users and “filtering” is simply a synonym for “prediction”. So, we use collaborative filtering quite frequently in our daily life and it really seems like common sense. One of the most popular collaborative filtering methods is Alternating Least Squares (ALS). In ALS we assume that the available rating data can be represented in a



sparse matrix form, that is, we will assume a sequential ordering of both users and products. Each entry of the matrix will then represent the rating for a unique pair of user and products. This will look something like the matrix represented in the figure below.

$$R = \begin{matrix} & \text{user 1} & \text{user 2} & \text{user 3} & \cdots & \text{user } N \\ \begin{bmatrix} 1 \\ ? \\ 5 \\ \vdots \\ 2 \end{bmatrix} & & 4.5 & ? & \cdots & 3 \\ & & 3 & 3 & \cdots & 4 \\ & & 3 & ? & \cdots & ? \\ & & \vdots & \vdots & \ddots & \vdots \\ & & 4 & 1 & \cdots & ? \end{bmatrix} & \begin{matrix} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product } M \end{matrix} \end{matrix}$$

The way that ALS works is, in simplified terms, to find the factors U and P, which when multiplied together provide an approximation of our ratings matrix R. Once we have the factors U and P, we can then predict the missing values in R by using the approximation value.

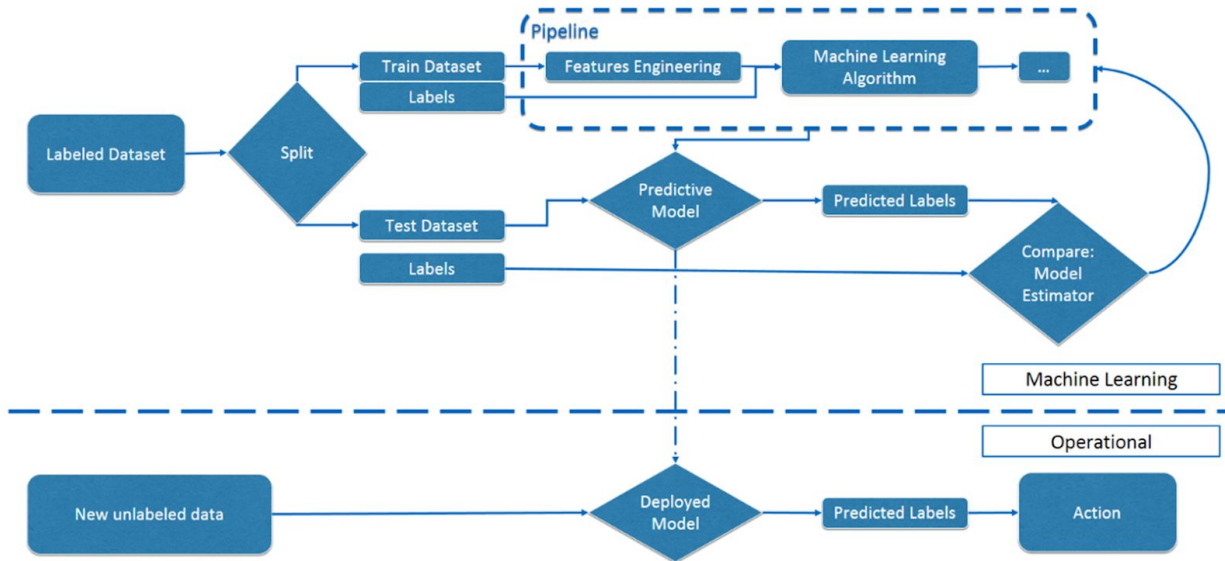


It is clear that in a real world scenario we would have many missing ratings, simply due to the assumption that no user rates all products. ALS is designed to deal with sparse matrices and to fill the blanks using predicted values. After factorization, our approximated ratings matrix will look something like this:

$$R = \begin{matrix} & \text{user 1} & \text{user 2} & \text{user 3} & \cdots & \text{user } N \\ \begin{bmatrix} 1 \\ 3.2 \\ 5 \\ \vdots \\ 2 \end{bmatrix} & & 4.5 & 3.8 & \cdots & 3 \\ & & 3 & 3 & \cdots & 4 \\ & & 3 & 3.4 & \cdots & 3.1 \\ & & \vdots & \vdots & \ddots & \vdots \\ & & 4 & 1 & \cdots & 2.7 \end{bmatrix} & \begin{matrix} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product } M \end{matrix} \end{matrix}$$

In summary, we use Alternating Least Square (ALS) to implement our recommendation system. And the following are the key advantages of ALS:

1. A matrix factorization algorithm to deal with sparse ratings data
2. Runs itself in a parallel fashion: ALS is implemented in Apache Spark ML and built for a larges-scale collaborative filtering problems
3. ALS is doing a pretty good job at solving scalability and it's simple and scales well to very large datasets.



The figure above is our learning process. Firstly, we train our data into training and testing dataset. Secondly, for training dataset, we do feature engineering and implement the machine learning algorithm, and select the optimal model with lowest RMSE(evaluation index). So, in prediction stage, we can deploy our model and predict for user. The result shown that the best model was train with rank 24.

```

seed = 5L
iterations = 10
regularization_parameter = 0.1
ranks = [16, 20, 24]
errors = [0, 0, 0]
err = 0
tolerance = 0.02

min_error = float('inf')
best_rank = -1
best_iteration = -1
for rank in ranks:
    model = ALS.train(training_RDD, rank, seed=seed, iterations=iterations,
                      lambda_=regularization_parameter)
    predictions = model.predictAll(validation_for_predict_RDD).map(lambda r: ((r[0], r[1]), r[2]))
    rates_and_preds = validation_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
    error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())
    errors[err] = error
    err += 1
    print 'For rank %s the RMSE is %s' % (rank, error)
    if error < min_error:
        min_error = error
        best_rank = rank
  
```

```
print 'The best model was trained with rank %s' % best_rank
```

```
For rank 16 the RMSE is 6.49279681528
For rank 20 the RMSE is 6.33951908492
For rank 24 the RMSE is 6.06042819457
The best model was trained with rank 24
```

```
# adding new user ratings
new_user_ID = 0

# The format of each line is (userID, itemID, rating)
new_user_ratings = [
    (0,2140811,9),
    (0,2281848,9),
    (0,2703625,9),
    (0,2357523,8),
    (0,1861964,9),
    (0,472333,9),
    (0,131117,9),
    (0,322434,9),
    (0,1532310,10) ,
    (0,267405,10)
]

new_user_ratings_RDD = sc.parallelize(new_user_ratings)
new_ratings_model = ALS.train(complete_data_with_new_ratings_RDD, best_rank, seed=seed,
                               iterations=iterations, lambda_=regularization_parameter)

# get just item IDs
new_user_ratings_ids = map(lambda x: x[1], new_user_ratings)
# keep just those not on the ID list
new_user_unrated_items_RDD = (complete_items_data.filter(lambda x: x[0] not in
new_user_ratings_ids).map(lambda x: (new_user_ID, x[0])))

# predict new ratings for the items
new_user_recommendations_RDD = new_ratings_model.predictAll(new_user_unrated_items_RDD)
# Transform new_user_recommendations_RDD into pairs of the form (item ID, Predicted Rating)
new_user_recommendations_rating_RDD = new_user_recommendations_RDD.map(lambda x:
(x.product, x.rating))
new_user_recommendations_rating_title_and_count_RDD = \

new_user_recommendations_rating_RDD.join(complete_items_titles).join(item_rating_counts_RDD.
map(lambda x: (int(x[0]), x[1])))
new_user_recommendations_rating_title_and_count_RDD.take(3)
```

```
[(466944, ((4.123172974617845, u'gown'), 25)),
 (2107392, ((8.42363048717481, u'down'), 32)),
 (2576388, ((10.833968349772954, u'jumpsuit'), 8))]
```

Finally, we create a new user data and apply the recommendation system to predict the best recommended clothing to him/her, filtering out the items with more than 25 reviews. To make the result more readable, we transform the recommendation result into DataFrame.

For example, for the first recommend clothing with the highest rating prediction 14.18 , it belongs to gown with 25 review which have been rated by other users.

```
# Finally, get the highest rated recommendations for the new user, filtering out items with less than 25 ratings.
```

```
top_items = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r:
r[3]>=25).takeOrdered(25, key=lambda x: -x[2])
```

```
df_rating = df_rating.withColumnRenamed('_1', 'user_id').withColumnRenamed('_2',
'item_id').withColumnRenamed('_3', 'rating')
```

```
# transform the recommendation result into DataFrame to read easily.
```

```
df_top_items = spark.createDataFrame(top_items)
```

```
df_top_items = df_top_items.withColumnRenamed('_1', 'item_id').withColumnRenamed('_2',
'category')
```

```
.withColumnRenamed('_3', 'rating').withColumnRenamed('_4', 'num_review')
```

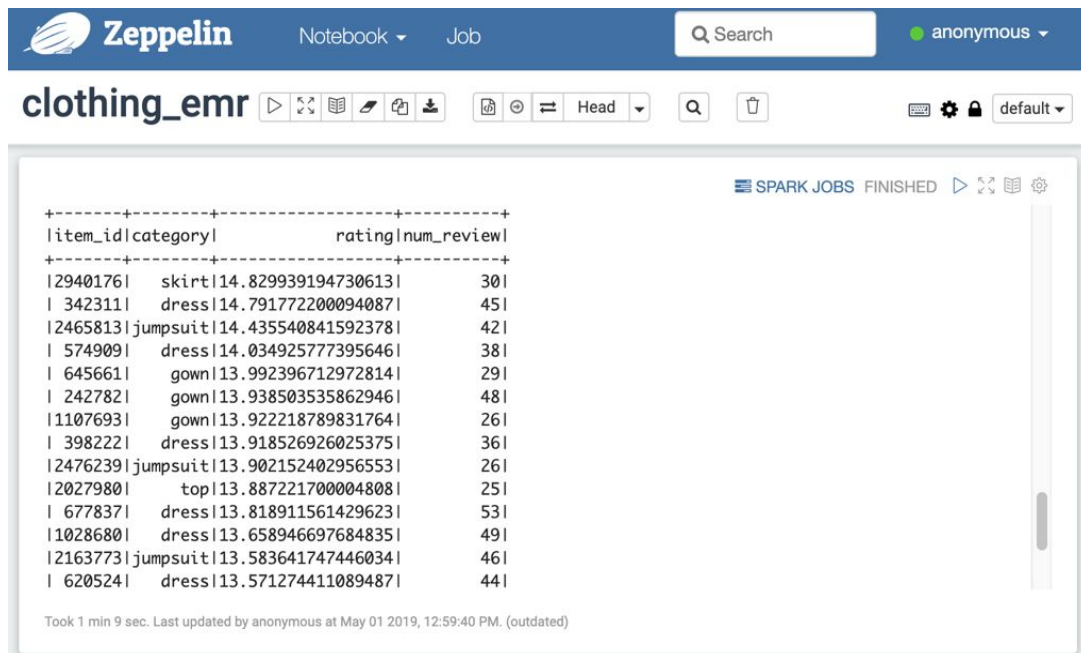
```
df_top_items.show()
```

```
+-----+-----+-----+-----+
|item_id|category|      rating|num_review|
+-----+-----+-----+-----+
|1420896|  gown|14.184187605082625|    25|
|1606923|  dress|13.804242349538914|    31|
|2315001|jumpsuit|13.729496587544082|    34|
|1584094|  dress|13.375911178826284|    43|
|1269149|  gown|13.316707606593885|    25|
|2931769| romper|13.289865236963289|    34|
|1440124|  dress|13.035235093499773|    26|
| 507766|  dress|13.021977989160643|    35|
| 386314| sheath|12.990129769619886|    37|
|2429745| romper|12.962967679329056|    42|
|1207360| sheath|12.952243789810975|    64|
|2635020|   top|12.94348618124923|    25|
|1069265|  maxi|12.865889015865037|    40|
| 709521|  dress|12.863461735252645|    40|
|1155639|  dress|12.820026419765345|    29|
|1937688|  dress|12.811998256465639|    38|
|1519172| sheath|12.797419319325227|    28|
|1846462|  gown|12.708789813709211|    25|
|1901762|  dress|12.703336415619257|    36|
|1944337|  dress|12.65806273008108|    39|
+-----+-----+-----+-----+
```

only showing top 20 rows

### 3.5. Cloud Computing

We used Amazon Web Services for this project to implement cloud computing. AWS does not support MongoDB. However, they provide an alternative database similar to MongoDB, documentDB. Also, AWS provides the dynamic DB for streaming data and S3 for object storage. For this project, we use S3 as our data storage. Then we created an Amazon Elastic Map Reduce (EMR) cluster, which is an AWS proprietary product works similar to Apache Hadoop and Apache Spark to manage the cluster and use for cloud computing. Instead of Jupyter notebook, we ran our query and machine learning algorithm on Apache Zeppelin. It is a web-based cloud notebook allowed us to write programs that can run on the underlying Spark framework. The interactive and collaborative features provide the flexibility to share the data analysis results with other team members.



The screenshot shows the Apache Zeppelin Notebook interface. The top navigation bar includes the Zeppelin logo, 'Notebook' and 'Job' tabs, a search bar, and a user profile 'anonymous'. Below the navigation bar, the notebook title 'clothing\_emr' is displayed with various action icons. The main content area shows the output of a Spark job, which is a table of clothing items with their categories, ratings, and review counts. The status 'SPARK JOBS FINISHED' is visible in the top right of the output area. At the bottom, a timestamp indicates the job was completed on May 01, 2019, at 12:59:40 PM.

item_id	category	rating	num_review
2940176	skirt	14.829939194730613	30
342311	dress	14.79177220094087	45
2465813	jumpsuit	14.435540841592378	42
574909	dress	14.034925777395646	38
645661	gown	13.992396712972814	29
242782	gown	13.938503535862946	48
1107693	gown	13.922218789831764	26
398222	dress	13.918526926025375	36
2476239	jumpsuit	13.902152402956553	26
2027980	top	13.887221700004808	25
677837	dress	13.818911561429623	53
1028680	dress	13.658946697684835	49
2163773	jumpsuit	13.58364174446034	46
620524	dress	13.571274411089487	44

Took 1 min 9 sec. Last updated by anonymous at May 01 2019, 12:59:40 PM. (outdated)

### 4. Future Work

In the future, we can make good use of SparkStreaming make Streaming Recommendation system. The ALS matrix factorization API provided by Spark MLlib is naive approach for collaborative filtering. We can also use Spark Streaming to create a small window size and design a mini-batch approach to achieve a faster update. The Spark Streaming can also use as real time data streaming, so that the user reviews and ratings can be constantly updated.

## 5. Summary

In this project, we build the spark ecosystem to do knowledge discovery from our data, not only make full use of Spark technologies what we have learned in class, but also be familiar with building the pipeline to tackle the big data problem in Spark, for example, from data storage to backend system, and finally the front-end application to do the prediction\classification and data visualization. In the future, we will improve our framework and involve more technologies to make it a full stack application on cloud, so our ecosystem can be used to tackle more challenging problem and more complex data.

## 6. Reference

- <https://www.kaggle.com/rmisra/clothing-fit-dataset-for-size-recommendation>
- <https://www.modelcloth.com/>
- <https://www.renttherunway.com/>
- [Hive vs.HBase–Different Technologies that work Better Together](#)
- <https://spark.apache.org/docs/2.2.0/mllib-feature-extraction.html>
- <https://www.fitanalytics.com/>
- <https://medium.com/@jaceklaskowski/why-is-spark-sql-so-obsessed-with-hive-after-just-a-single-day-with-hive-289e75fa6f2b>
- <https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>
- <https://sujitpal.blogspot.com/2018/08/collaborative-filtering-recommender.html>
- <https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1>
- <https://ruivieira.github.io/a-streaming-als-implementation.html>
- <https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>