Static Data Analysis (   )

Feature Extraction

In order to use the data we have more efficiently you are tasked to do several tasks.

First, we will explore the demographic data and analyze it.

We would like to adjust and normalize the features:

1. For numerical variables - normalize each column according to the maximum value in that column - so the values in each numerical column would be in the range of [0,1]
2. For categorical variables (non-ordinal) - use one hot encoding to change the values of the column to a binary vector representation.

Then, combine all adjusted and normalized variables into one feature vector column that will represent each household.

Notice! The household_id number is NOT a valid data feature.

Show 7 rows from the resulting dataframe, using *df.show(7 , truncate=False)*

Visual Analysis

Now that we have one column that contains all our useful information, we can easily use many known and good algorithms to infer information from our data.

We want to see if the data is naturally divided into several groups of households. In order to do that we can use the PCA algorithm. Use the PCA algorithm to project the feature vectors from the previous part on to 2 dimensional space and then plot the result in a scatter plot.

Use the already implemented PCA algorithm from PySpark MLLib, with k=2.

In order to plot, you may use:

pca_df.toPandas().plot.scatter(x='x_col_name', y='y_col_name')

- What do you see in the scatter plot? How many different clusters do you see in the plot?
- Show 7 rows from the resulting dataframe, using *df.show(7 , truncate=False)*.

Clustering

The PCA algorithm can help us get intuition of if the data is truly divided into several clusters, however it is not a clustering algorithm and using it to cluster data can be problematic. Use the K-means algorithm to cluster the data into 6 clusters.

- Add a new column that states which household_id belongs to which cluster. The cluster names/ID-numbers do not matter.
- Add a new column that contains the distance of each household_id from it's centroid (Hint: use window functions).
- Use the already implemented K-means algorithm from PySpark MLLib, with k=6 and Set seed to 3.
- Show 7 rows from the resulting Dataframe, using *df.show(7 , truncate=False)*

Dividing households into subsets

For the calculation of the subsets, you MUST utilize PySpark Windows.

Order the households in each cluster by their distance from their cluster's centroid.

For each cluster from the demographic data observe the subset that contains every third row in that cluster (the 3th, 6th, ... rows in that cluster). Remember - the rows are ordered in each cluster by the distance to their centroid.

We'll call that subset the *"3^{rds} subset"* Notice that we have 6 such subsets - one for each cluster.

Do the same again for each cluster, now for each 17^{th} row (17th, 34th, ...). We'll call this subset the *"17^{ths} subset"*. Notice that you also have 6 such subsets.

In addition for the 2 kinds of subsets (3^{rds} and 17^{ths}), you also have the full data for each cluster.

In total, you should have 18 sets of household_ids - 3 sets per cluster.

Cluster's Viewing Analysis

Next we want to see if the clustering on the households that we've done holds any value. For that we will use the viewing data - to see whether the viewing habits of people from different demographic clusters are actually different.

In order to asses the uniqeness of each cluster's viewing habits, we will do the following procedure, that will yield us the 'diff_rank' measure:

1. For each cluster, count the amount of viewing events (rows in the viewing_data) for each station.

   The output of this stage (per cluster) - is a count of viewing rows, aggregated per station number.

2. Divide the amount of viewings per station in the total amount of views for that cluster, and then multiply by 100.

   The output of this stage is the percent of viewing events per station in this cluster. This can be treated as the cluster's popularity rating of a station.

3. Now, repeat steps 1 and 2 - but for the entire data, not on a per cluster basis. This is the popularity rating of each station among the general population.

   The output of this stage is the popularity ratings (percentages) of each station among the general public.

4. For each station and each cluster, subtract the popularity rating (percentage) of the general public from the popularity of that station for that specific cluster.

   We will name the output of this stage - the 'diff_rank' of a station among a cluster.

- For each subset calculate the top-7 highest 'diff_rank' stations. Overall you need to return 18 groups of top ranked stations, one for each subset - 3 sets of results per cluster.
- Print / Show these results, with clear titles that differentiate between subsets.
- Each cluster's 3 sets' results should be shown close to one another.
- You may take artsy liberty in choosing the exact way of presentation, while conforming to the previous bullets.

Dynamic Data Analysis - Streaming (

Use Spark Streaming to extract the viewing data from Kafka. Use the code that was provided to you in the Starter notebook in order to connect and read from the stream.

While streaming, repeat the "Cluster's Viewing Analysis" task from the previous task, for each trigger's data.

You should get outputs for each batch of viewing data that you've read from the stream.

(In case your code does not produce any prints to the cell output - check the Driver Logs - accessible via a click on your running cluster menu. You should copy the results from the streaming process to a cell in the appropriate place.)

- Read and process at least 5 batches / triggers. Show the results per batch.
- *Notice! In this part - you should only analyze the "3^{rds} subset" per each cluster*

  That means that for each trigger you would have 6 groups of top ranked stations - only 1 group of stations per cluster instead of 3.

  We are interested only in the 3^{rds} subset output in this part.