BDA Project

RFM model-based customer segmentation

Simona Scala

University of Bologna

July 2023

The Aim Of The Project

The implementation of this project involves exploring data-mining techniques for an online retailer using the PySpark library, following the methodology outlined by Chen et al..

Their article primarily aims to enhance business understanding of its customers and to utilize data mining techniques to segment customers based on the **Recency-Frequency-Monetary** (RFM) model.

The Dataset

- Online Retail II data set
- Transactions from an online retail business in the UK
- Period between 01/12/2009 and 09/12/2011
- Company selling unique gifts for all occasions
- Many customers are wholesalers
- Number of features: 8
- Number of instances: 525461

The Changes To The Dataset

The old dataset

- 11 features
  - Invoice, StockCode, Description, Quantity, Price, InvoiceDate, **Address Line 1**, **Address Line 2**, **Address Line 3**, **PostCode**, Country

The new dataset

- 8 features
  - Invoice, StockCode, Description, Quantity, Price, InvoiceDate, **Customer ID**, Country

Data Pre-Processing

The dataset underwent pre-processing steps before using ML algorithms:

Data Pre-Processing

The dataset underwent pre-processing steps before using ML algorithms:

- Delete all rows that contain null values in the columns `Quantity`, `InvoiceDate`, `Price`, `Customer ID`, and `Country`.
  - ```
    df = df.dropna(subset=['Quantity', 'InvoiceDate',
    'Price', 'Customer ID', 'Country'])
    ```
    ```

Data Pre-Processing

The dataset underwent pre-processing steps before using ML algorithms:

- Delete all rows that contain null values in the columns `Quantity`, `InvoiceDate`, `Price`, `Customer ID`, and `Country`.
  - `df = df.dropna(subset=['Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country'])`
- Delete all rows where the `Quantity` value is less than 0
  - `df = df.filter(df.Quantity > 0)`

Data Pre-Processing

The dataset underwent pre-processing steps before using ML algorithms:

- Delete all rows that contain null values in the columns `Quantity`, `InvoiceDate`, `Price`, `Customer ID`, and `Country`.
  - `df = df.dropna(subset=['Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country'])`
- Delete all rows where the `Quantity` value is less than 0
  - `df = df.filter(df.Quantity > 0)`
- Compute the total amount of money spent on each product by multiplying the values of `Quantity` with `Price`
  - `df = df.withColumn('Amount', (col('Quantity')*col('Price')).cast('float'))`

Data Pre-Processing

The dataset underwent pre-processing steps before using ML algorithms:

- Delete all rows that contain null values in the columns `Quantity`, `InvoiceDate`, `Price`, `Customer ID`, and `Country`.
  - `df = df.dropna(subset=['Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country'])`
- Delete all rows where the `Quantity` value is less than 0
  - `df = df.filter(df.Quantity > 0)`
- Compute the total amount of money spent on each product by multiplying the values of `Quantity` with `Price`
  - `df = df.withColumn('Amount', (col('Quantity')*col('Price')).cast('float'))`
- Separate the variable `InvoiceDate` into two variables `Date` and `Time`
  - `df = df.withColumn('Date', date_format('InvoiceDate', 'yyyy-MM-dd'))`
  - `df = df.withColumn('Time', date_format('InvoiceDate', 'HH:mm:ss'))`
  - `df = df.drop('InvoiceDate')`

Data Pre-Processing

The dataset underwent pre-processing steps before using ML algorithms:

- Filter out all transactions that are not linked to the United Kingdom
  - `df = df.filter(df.Country.like("United Kingdom"))`

Data Pre-Processing

The dataset underwent pre-processing steps before using ML algorithms:

- Filter out all transactions that are not linked to the United Kingdom
  - `df = df.filter(df.Country.like("United Kingdom"))`
- Sort the dataset by `Customer ID` and create three fundamental aggregated variables: `Recency`, `Frequency`, and `Monetary`.
  - `df_agg = df.groupby("Customer ID").agg(datediff(current_date(), max("Date")).alias("Recency"), count("Invoice").alias("Frequency"), sum("Amount").cast('float').alias("Monetary"))`

Customer ID	Recency	Frequency	Monetary
12346.0	4777	33	372.86
12608.0	4652	16	415.79
12745.0	4734	22	723.85
12746.0	4788	17	254.55
12747.0	4617	154	5080.5303
12748.0	4613	2634	22879.66
12749.0	4647	139	2806.48
12777.0	4705	26	519.45
12819.0	4706	19	540.52

The Clustering Formula

At this point, the `RFormula` can be utilized to define the clustering formula, which includes specifying the features to be employed in the process. These features will be assembled into a vector, enabling clustering based on the selected variables.

```
rf = RFormula(formula="∼ Recency + Frequency + Monetary")
rf_fit = rf.fit(df_agg)
rf_transfd = rf_fit.transform(df_agg)
```

Customer ID	Recency	Frequency	Monetary	features
12346.0	4777	33	372.86	[4777.0,33.0,372....
12608.0	4652	16	415.79	[4652.0,16.0,415....
12745.0	4734	22	723.85	[4734.0,22.0,723....
12746.0	4788	17	254.55	[4788.0,17.0,254....
12747.0	4617	154	5080.5303	[4617.0,154.0,508...
12748.0	4613	2634	22879.66	[4613.0,2634.0,22...
12749.0	4647	139	2806.48	[4647.0,139.0,280...
12777.0	4705	26	519.45	[4705.0,26.0,519....
12819.0	4706	19	540.52	[4706.0,19.0,540....
12820.0	4645	101	1747.18	[4645.0,101.0,174...
12821.0	4859	7	128.08	[4859.0,7.0,128.0...
12823.0	4642	13	4742.0	[4642.0,13.0,4742.0]
12825.0	4769	24	518.63	[4769.0,24.0,518....
12826.0	4613	75	1481.03	[4613.0,75.0,1481...
12829.0	4798	8	92.299995	[4798.0,8.0,92.29...
12831.0	4711	13	236.06	[4711.0,13.0,236....
12835.0	4675	620	6043.31	[4675.0,620.0,604...
12836.0	4637	239	3972.76	[4637.0,239.0,397...
12837.0	4817	80	554.31	[4817.0,80.0,554....
12838.0	4621	300	2715.35	[4621.0,300.0,271...

only showing top 20 rows

Scaling

In the context of clustering, it is essential to scale the features to ensure they have comparable ranges. Scaling helps prevent any particular feature from dominating the distance calculations used in clustering algorithms such as the k-means algorithm. A commonly used method to standardize the features is by employing the `StandardScaler`.

The result of applying the `StandardScaler` is that each feature will have a mean of zero and a standard deviation of one.

It means that the data points are equally distributed above and below zero, resulting in a balanced distribution around the zero point on the number line and that the spread or dispersion of data points is equal to one, which indicates that the data points are distributed closely around the mean with little variability from the average value.

Scaling

```
scaler = StandardScaler(inputCol="features",
outputCol="scaledFeatures")
rf_transfd = scaler.fit(rf_transfd).transform(rf_transfd)
```

```
+-----------+-------+---------+---------+--------------------+--------------------+
|Customer ID|Recency|Frequency| Monetary|            features|      scaledFeatures|
+-----------+-------+---------+---------+--------------------+--------------------+
|    12346.0|   4777|       33|   372.86|[4777.0,33.0,372....|[49.1650914958890...|
|    12608.0|   4652|       16|   415.79|[4652.0,16.0,415....|[47.8785860663336...|
|    12745.0|   4734|       22|   723.85|[4734.0,22.0,723....|[48.7225336281219...|
|    12746.0|   4788|       17|   254.55|[4788.0,17.0,254....|[49.2783039736899...|
|    12747.0|   4617|      154|5080.5303|[4617.0,154.0,508...|[47.5183645460581...|
|    12748.0|   4613|     2634| 22879.66|[4613.0,2634.0,22...|[47.4771963723123...|
|    12749.0|   4647|      139|  2806.48|[4647.0,139.0,280...|[47.8271258491514...|
|    12777.0|   4705|       26|   519.45|[4705.0,26.0,519....|[48.4240643684651...|
|    12819.0|   4706|       19|   540.52|[4706.0,19.0,540....|[48.4343564119015...|
|    12820.0|   4645|      101|  1747.18|[4645.0,101.0,174...|[47.8065417622785...|
|    12821.0|   4859|        7|   128.08|[4859.0,7.0,128.0...|[50.0090390576773...|
|    12823.0|   4642|       13|   4742.0|[4642.0,13.0,4742.0]|[47.7756656319692...|
|    12825.0|   4769|       24|   518.63|[4769.0,24.0,518....|[49.0827551483974...|
|    12826.0|   4613|       75|  1481.03|[4613.0,75.0,1481...|[47.4771963723123...|
|    12829.0|   4798|        8|92.299995|[4798.0,8.0,92.29...|[49.3812244080543...|
|    12831.0|   4711|       13|   236.06|[4711.0,13.0,236....|[48.4858166290837...|
|    12835.0|   4675|      620|  6043.31|[4675.0,620.0,604...|[48.1153030653718...|
|    12836.0|   4637|      239|  3972.76|[4637.0,239.0,397...|[47.7242054147869...|
|    12837.0|   4817|       80|   554.31|[4817.0,80.0,554....|[49.5767732333467...|
|    12838.0|   4621|      300|  2715.35|[4621.0,300.0,271...|[47.5595327198039...|
+-----------+-------+---------+---------+--------------------+--------------------+
only showing top 20 rows
```

Clustering

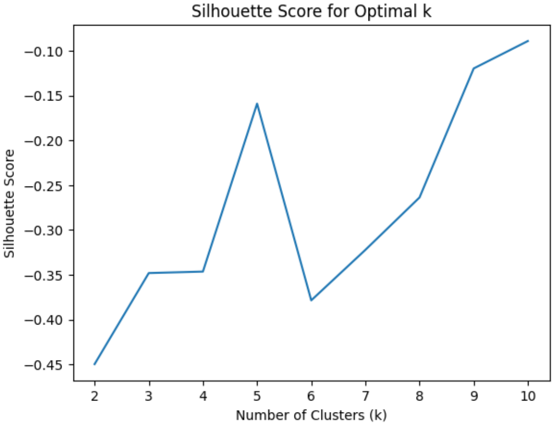
The clustering algorithm adopted in this project was the k-means algorithm.

The first step dealt with the selection of the optimal value for `k`, the number of clusters.

The silhouette score serves as a metric to evaluate how well each object within the dataset belongs to its respective cluster compared to other clusters. A higher silhouette score indicates that the clusters are well-defined and distinct.

By computing the silhouette scores for different `k` values, it is possible to identify the one that maximizes the score, indicating the most suitable number of clusters for our dataset.

Clustering



- Research paper: `k = 5`
- Computing the silhouette score: `k = 10`

Clustering

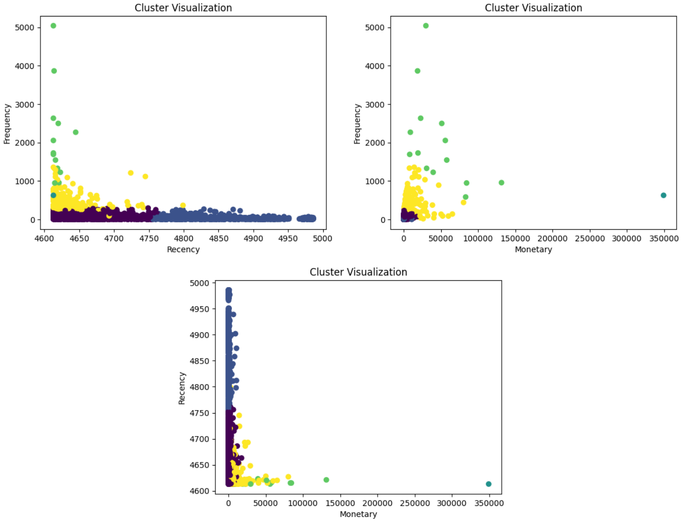


Figure: Clustering result with k = 5

Clustering

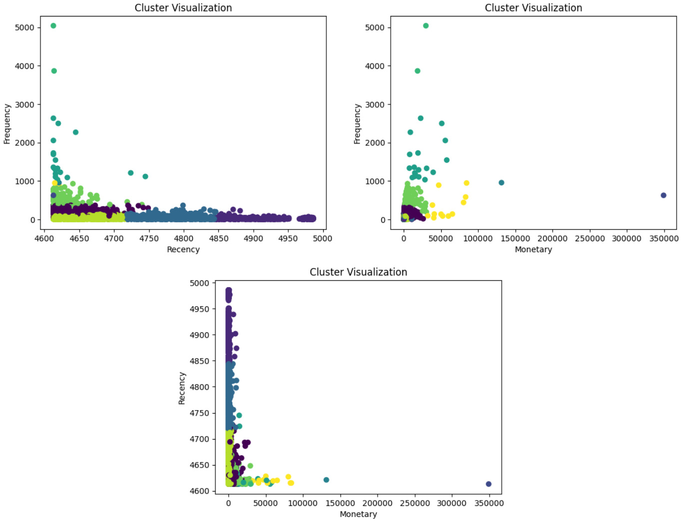


Figure: Clustering result with k = 10

Classification Models

Once the data had been labelled, three different classification models were trained:

- Decision Tree Model
- Random Forest Classifier
- Multi-Layer Perceptron

Decision Tree Model

```
tree = DecisionTreeClassifier()
tree_model = tree.fit(in_train)
tree_pred = tree_model.transform(in_test)
# Evaluation
tree_evaluator =
MulticlassClassificationEvaluator(labelCol='label',
predictionCol='prediction', metricName='accuracy')
tree_accuracy = tree_evaluator.evaluate(tree_pred) * 100
```

- Accuracy with `k = 5`: 97.07%
- Accuracy with `k = 10`: 95.91%

Random Forest Classifier

```
rf = RandomForestClassifier()
rf_model = rf.fit(in_train))
rf_pred = rf_model.transform(in_test))
# Evaluation
rf_evaluator =
MulticlassClassificationEvaluator(labelCol='label',
predictionCol='prediction', metricName='accuracy')
rf_accuracy = rf_evaluator.evaluate(rf_pred) * 100
```

- Accuracy with `k = 5`: 97.07%
- Accuracy with `k = 10`: 95.83%

Multi-Layer Perceptron

```
layers = [3, 5, 5, 10]
nn = MultilayerPerceptronClassifier(layers=layers,
seed=seed)
nn_model = nn.fit(in_train)
nn_pred = nn_model.transform(in_test)
# Evaluation
nn_evaluator =
MulticlassClassificationEvaluator(labelCol='label',
predictionCol='prediction', metricName='accuracy')
nn_accuracy = nn_evaluator.evaluate(nn_pred) * 100
```

- Accuracy with `k = 5`: 70.43%
- Accuracy with `k = 10`: 62.70%