MIE1628 Final Project Report

Illicit Bitcoin transaction analysis using graphs

Group 6

Da Da 1002115515

Aug.28th 2020

1. Introduction

Bitcoin is a digital currency created in 2009. It can be transferred using apps. Every transaction is recorded in the decentralized authority known as blockchain. Bitcoin is a highly scalable peer-to-peer network that has no responsible authority or central management; it is accessible to anyone with an Internet connection. [1] In recent years, Bitcoin and cryptocurrency system has been subjected to criminal cases. According to a study conducted earlier in 2018, approximately one-quarter of Bitcoin users and one-half of Bitcoin transactions are associated with illicit activity. Around $72 billion of unlawful activity per year involves Bitcoin, which is close to the scale of the U.S. and European markets for illegal drugs. [2]

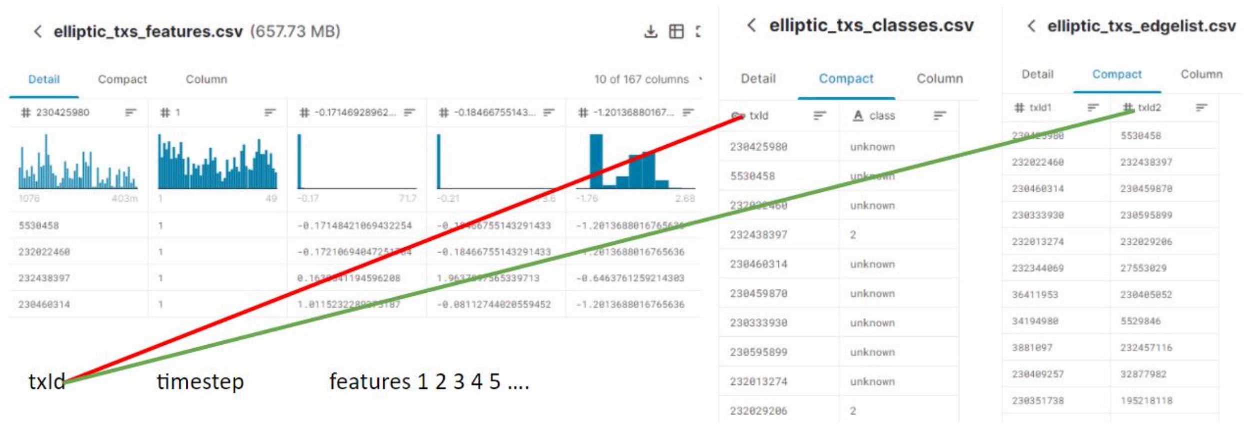
The task of this project is to train a model to classify the Bitcoin transactions into licit and illicit by using the features of each transactions. Specifically, graph-based analysis will be used. The Elliptic Data Set is used for this project, which is available on www.kaggle.com .

The Elliptic Data Set maps Bitcoin transactions to real entities belonging to licit categories (exchanges, wallet providers, miners, licit services, etc.) versus illicit ones (scams, malware, terrorist organizations, ransomware, Ponzi schemes, etc.). The task on the dataset is to classify the illicit and licit nodes in the graph. [3]

2. Dataset Description

This anonymized data set is a transaction graph collected from the Bitcoin blockchain. A node in the graph represents a transaction, an edge can be viewed as a flow of Bitcoins between one transaction and the other. Each node has 166 features and has been labeled as being created by a "licit", "illicit" or "unknown" entity.

The entire dataset contains three CSV files, each include features, classes (labels) and edge list. The dataset relational, where the transaction IDs are the primary keys, and is the node in the graph. It can be seen in figure 1.



elliptic_txs_features.csv (657.73 MB)

230425980	1	-0.17146928962...	-0.18466755143...	-1.20136880167...
5530458	1	-0.17148421069432254	-0.18466755143291433	-1.201368801676563[illegible]
232022460	1	-0.172106940472517[illegible]	-0.18466755143291433	-1.201368801676563 6
232438397	1	0.16[illegible]41194596208	1.96[illegible]565339713	-0.6463761259214303
230460314	1	1.0115232[illegible]187	-0.08112744020559452	-1.201368801676563 6

elliptic_txs_classes.csv

txId	class
230425980	unknown
5530458	unknown
232022460	unknown
232438397	2
230460314	unknown
230459870	unknown
230333930	unknown
230595899	unknown
232013274	unknown
232029206	2

elliptic_txs_edgelist.csv

txId1	txId2
230425980	5530458
232022460	232438397
230460314	230459870
230333930	230595899
232013274	232029206
232344069	27553029
36411953	230405052
34194980	5529846
3881097	232457116
230409257	32877982
230351738	195218118

Figure 1: CSV files and relations

Nodes and edges

The graph is made of 203,769 nodes and 234,355 edges. Two percent (4,545) of the nodes are labelled class1 (illicit). Twenty-one percent (42,019) are labelled class2 (licit). The remaining transactions are not labelled regarding licit versus illicit.

Features

There are 166 features associated with each node. Due to intellectual property issues, exact description of all the features is not in the dataset. There is a time step associated to each node, representing a measure of the time when a transaction was broadcasted to the Bitcoin network. The time steps, running from 1 to 49, are evenly spaced with an interval of about two weeks. Each time step contains a single connected component of transactions that appeared on the blockchain within less than three hours between each other; there are no edges connecting the different time steps.

3. Background Information

Graph Analysis

Generally, graph is a data structure which has nodes and edges, and edges carries relationships between nodes. The following figure 2 is an example of Facebook social network graph. Each node represents a person, and the edges between them conveys their relationships. This is an example of directed graph, which the directions of "flows" between nodes are specified. Graphs provide a better way of dealing with abstract concepts like relationships and interactions. They also offer an intuitively visual way of thinking about these concepts. Graphs also form a natural basis for analyzing relationships in a Social context. [4]

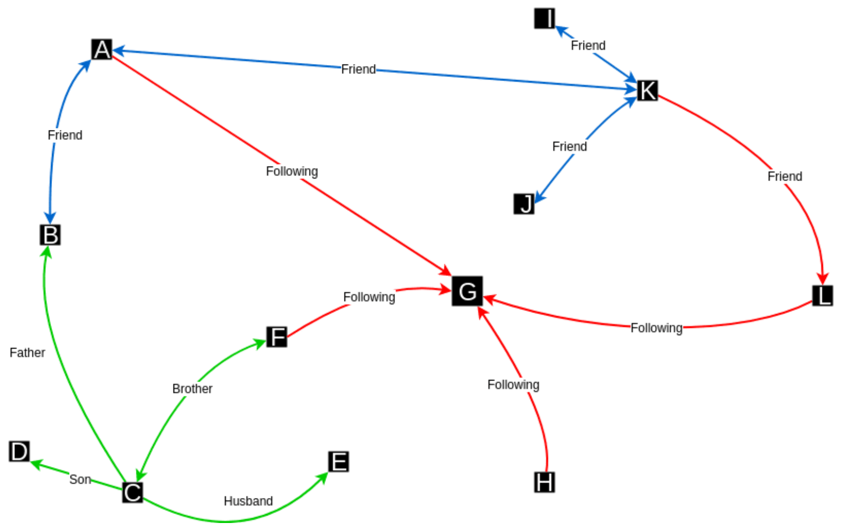


Figure 2: Facebook social network graph

Degree

In graph analysis, degree of centrality is the simplest concept. It measures how many edges are connected to a node. In a directed graph, degree centrality includes inflow and outflow centrality.

Page Rank

Page Rank is a well-known algorithm developed by Larry Page and Sergey Brin in 1996. They were studying at Stanford University and it was part of a research project to develop a new kind of search engine. They then successfully founded Google Inc. This algorithm assigns a numerical weighting to every node of a connected network. This measure represents the relative importance of a node within the graph (its rank). [5]

Label Propagation

The Label Propagation algorithm (LPA) is a fast algorithm for finding communities in a graph. It detects these communities using network structure alone as its guide and does not require a pre-defined objective function or prior information about the communities.

One interesting feature of LPA is that nodes can be assigned preliminary labels to narrow down the range of solutions generated. This means that it can be used as semi-supervised way of finding communities where we hand-pick some initial communities. [6]

4. Model Development

Label Propagation

Use the label propagation technique to predict unlabelled transactions based on the communities they are in. If a community is consisted of mostly illicit transactions, the likelihood of the remaining unlabelled ids belong to this community are also likely to be illicit. This method does not involve any features, only uses the edge list. From the following figure 3, imaging there is a three-class classification problem, the triangles are labelled, and the circled grey ones are unlabelled. The label propagation technique marks the same members within the community to the corresponding labelled class.

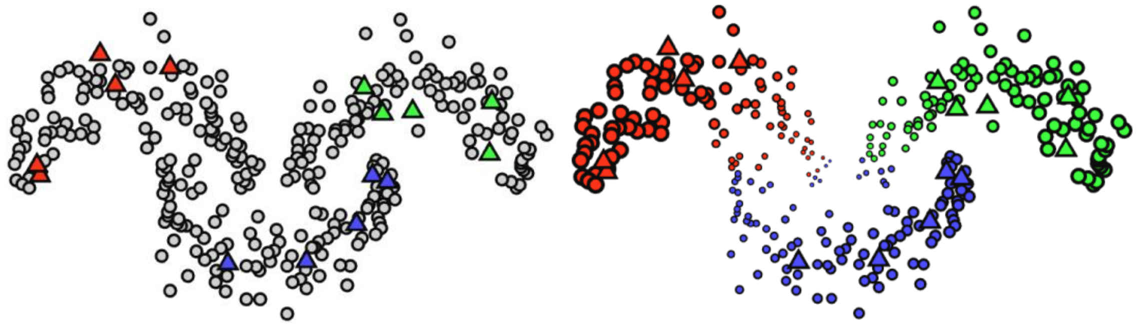


Figure 3: Label Propagation Method

Since Label Propagation method only predict on an existing community which has learnt by the labeled transactions, it is inevitable to miss those transactions in independent communities.

For the labeled nodes, this method reaches exceptionally good result as shown in the following tables.

Table 1: Label Propagation results on labeled nodes only:

Metrics	Scores
Precision	0.98314222
Recall	0.98959994
Accuracy	0.97530281
F1	0.98636051

Table 2: Label Propagation confusion matrix on labeled nodes only:

		Actual	
		Positive	Negative
Predicted	Positive	41582	713
	Negative	437	3832

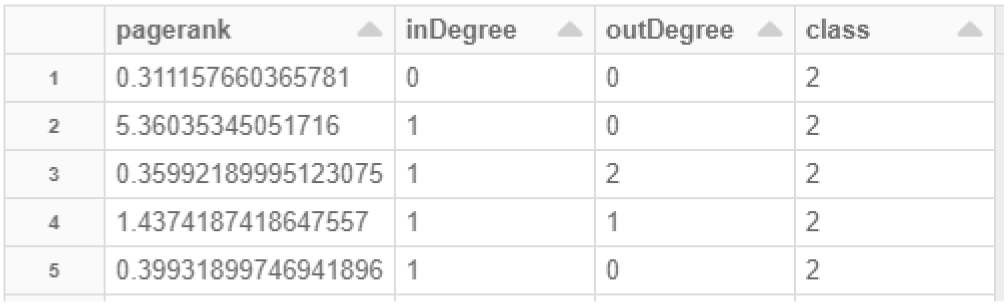
The model has successfully predicted 119,802 nodes out of total 203,769 data points (83967 records cannot be predicted). Although the results on labeled nodes are good, it still has limitations. About 55% of the unknown transactions cannot be predicted based solely on Label Propagation Method. The reason is in their communities, there is no single existing labelled data. Either more labelled data could be provided, or regular ML algorithm can be used on the remaining dataset.

Machine Learning on Graph Features

Since label propagation can only classify 55% of the unknown nodes, classic machine learning algorithms should be applied. Some key features are drawn from the graph analysis: page rank, inflow degree and outflow degree. Based on these three graph-based features, logistic regression and random forest are applied to the classification tasks. The data frame and results are shown below.

Table 3: ML on graph features

Models	F1 Score on labeled dataset	Accuracy on labeled dataset
Label Propagation (Graph)	98.6%	97.53%
Logistic classifier	89.9%	85.11%
Random Forest classifier	89.9%	85.11%



	pagerank	inDegree	outDegree	class
1	0.311157660365781	0	0	2
2	5.36035345051716	1	0	2
3	0.35992189995123075	1	2	2
4	1.4374187418647557	1	1	2
5	0.39931899746941896	1	0	2

Figure 4: Graph features data frame

Logistic regression and random forest on solely three graph-based features can have good Performance. Although the accuracy and F1 score is lower than label propagation, they can classify all the unknown nodes. The above two models demonstrate the power of graph analysis. By using only, the edge list, graph-based analysis can provide accurate models. However, the performance of models can be further improved if more features are included.

Machine Learning on Graph Features and Node Features

To further improve the model's performance, graph features can be combined with other features given in the data set. The original data set offers 165 features for each transaction ID (node). If all the 165 features are used to do the classification, it can give remarkably high accuracy and F1 score, but with graph features together the performance could be even better. The following figure demonstrate the concept of the model. There is a total of 167 features, which include the original 165 node features and the additional graph features: degree (include inflow and outflow) and PageRank. Furthermore, hyperparameter tuning by grid search is applied to improve the model.

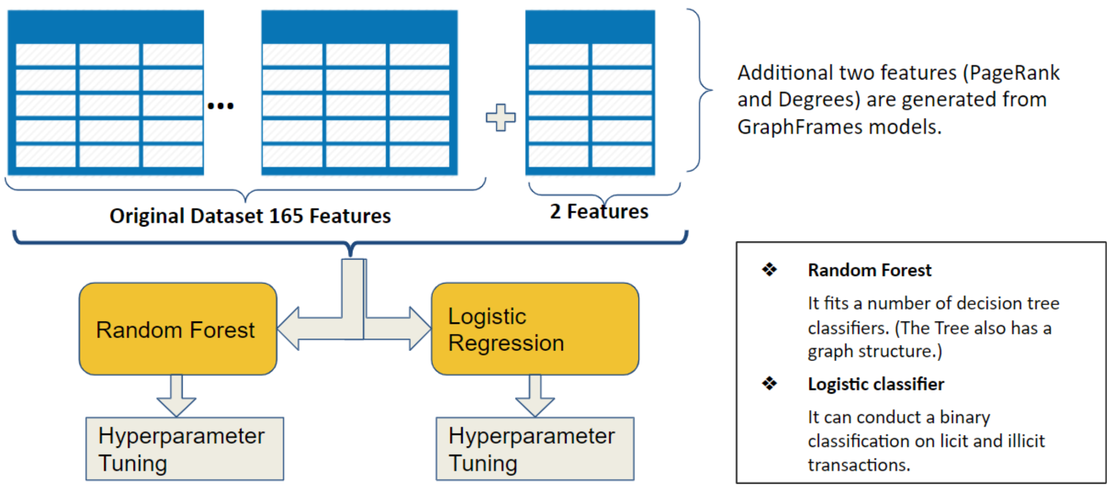

Additional two features (PageRank and Degrees) are generated from GraphFrames models.
Original Dataset 165 Features
2 Features
Random Forest
Logistic Regression
Hyperparameter Tuning
Hyperparameter Tuning
Random Forest
It fits a number of decision tree classifiers. (The Tree also has a graph structure.)
Logistic classifier
It can conduct a binary classification on licit and illicit transactions.

Figure 5: Aggregate model explanation

The performance of the model is extremely good. For logistic regression, adding graph features increases both scores by 5% and reaches 99%. Overall, it is clear how powerful the Graph features are for improving the model accuracy. It would be a great practice to combine both Graph techniques and ML algorithms for the elliptic dataset.

Table 4: Results comparisons among all models

Models	F1 Score on labeled dataset (without additional Graph features)	Accuracy on labeled dataset (without additional Graph features)	**Updated Accuracy after adding Graph features***	**Updated F1 after adding Graph features***	Percentage of unknown label data have been predicted?
Label Propagation (Graph)	98.6%	97.53%	Not Applicable	Not Applicable	45%
Logistic Regression	94%	94%	99%	99%	100%
Random Forest	97%	97%	97%	97%	100%

5. Discussion

In this project, we firstly tried using label propagation and found out its limitations. About 55% of the unknown nodes cannot be predicted. Label propagation uses pre-given partial labels as constraints to predict the labels of unlabeled data. It only uses label information as clustering constraints. Real applications not only contain partial label information but pairwise constraints on a dataset. [7] For future improvement on this project, we could use the new label propagation algorithm with pairwise constraints, which makes use of pairwise relations of labels as constraints to construct an optimization model for spreading labels.

Due to the limitations of label propagation, we used classic machine learning algorithms (logistic regression and random forest) to train the labeled data. Although we got good results on modeling the labeled data, the unlabeled data are not involved in the training process. The task is reduced to supervised learning from semi-supervised learning. On the other hand, training 167 features is computational expensive. For further improvement, we could do feature selection to filter out the less important features and add more informative features from graph analysis. The graphframes library in Databrick does not provide betweenness and closeness centrality. If we can include more graph-based features and reduce the original node features, it will accelerate the training.

The task of the project is binary classification, but the labeled data are imbalanced. Obviously, the more important part is the minority (illicit). For further improvement, we should focus more on the performance of models detecting illicit transaction. A weighted cross entropy loss could be used to provide higher importance to the illicit samples.

Lastly, more advanced semi-supervised learning algorithm other than benchmark methods should be applied to this dataset. Deep learning on graph structured data is a subject of rapidly increasing interest. Dealing with combinatorial complexity inherent to graph structures poses scalability challenges for practical applications, and significant strides have been made in addressing these challenges. [8] Methods like graph convolutional networks are suitable for this dataset.

[1] Elliptic, www.elliptic.co

[2] Tracing Illegal Activity Through The Bitcoin Blockchain To Combat Cryptocurrency-Related Crimes, https://www.forbes.com/sites/rachelwolfson/2018/11/26/tracing-illegal-activity-through-the-bitcoin-blockchain-to-combat-cryptocurrency-related-crimes/#6a09036b33a9

[3] Chen Zhao, Yong Guan. A GRAPH-BASED INVESTIGATION OF BITCOIN TRANSACTIONS. 11th IFIP International Conference on Digital Forensics (DF), Jan 2015, Orlando, FL, United States. pp.79-95, ff10.1007/978-3-319-24123-4_5ff. ffhal-01449078

[4] An Introduction to Graph Theory and Network Analysis, https://www.analyticsvidhya.com/blog/2018/04/introduction-to-graph-theory-network-analysis-python-codes/

[5] How to Perform Fraud Detection with Personalized Page Rank,

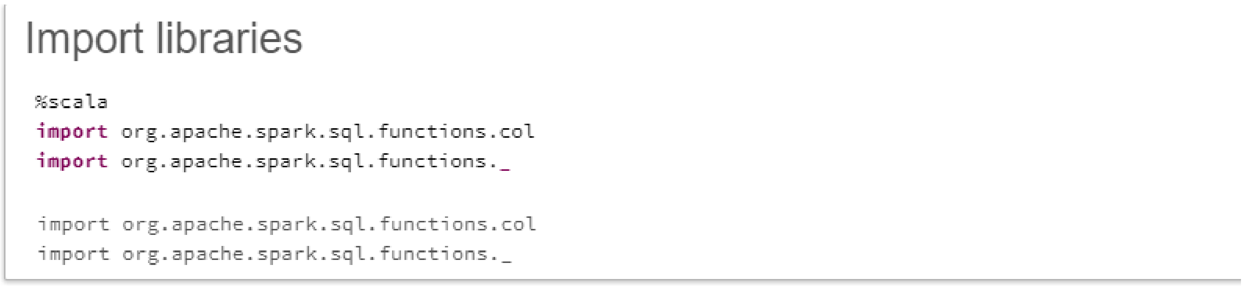
https://medium.com/sicara/fraud-detection-personalized-page-rank-networkx-15bd52ba2bf6

[6] Graph Algorithms: Practical Examples in Apache Spark and Neo4j, by Mark Needham & Amy E. Hodler

[7] arXiv:1904.04717

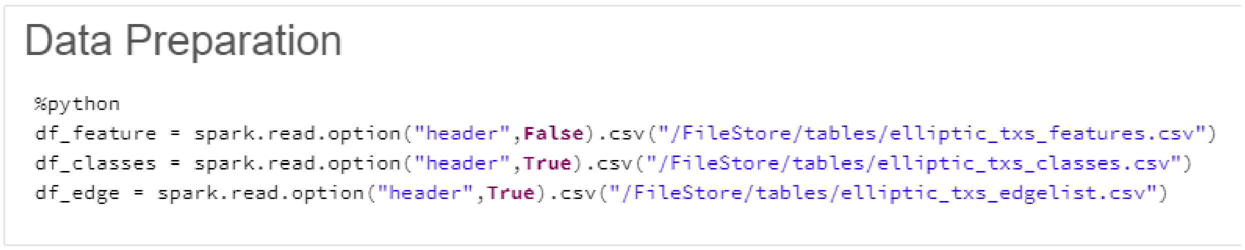
Code link: https://drive.google.com/drive/folders/1lkMuqPhdI9-6dKADNX9v0803-qBKUzs2?usp=sharing

Data Cleaning and Graph Preparing:



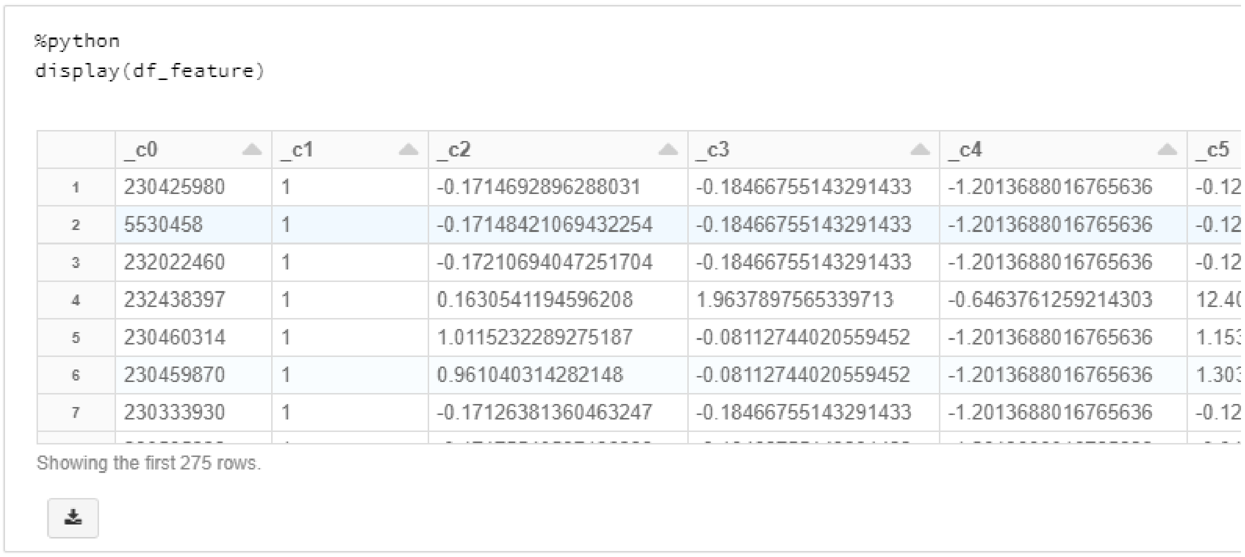
Import libraries

```
%scala
import org.apache.spark.sql.functions.col
import org.apache.spark.sql.functions._

import org.apache.spark.sql.functions.col
import org.apache.spark.sql.functions._
```



Data Preparation

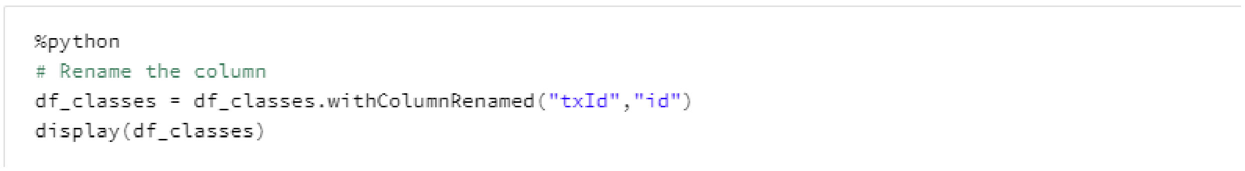
```
%python
df_feature = spark.read.option("header",False).csv("/FileStore/tables/elliptic_txs_features.csv")
df_classes = spark.read.option("header",True).csv("/FileStore/tables/elliptic_txs_classes.csv")
df_edge = spark.read.option("header",True).csv("/FileStore/tables/elliptic_txs_edgelist.csv")
```



```
%python
display(df_feature)
```

	_c0	_c1	_c2	_c3	_c4	_c5
1	230425980	1	-0.1714692896288031	-0.18466755143291433	-1.2013688016765636	-0.12
2	5530458	1	-0.17148421069432254	-0.18466755143291433	-1.2013688016765636	-0.12
3	232022460	1	-0.17210694047251704	-0.18466755143291433	-1.2013688016765636	-0.12
4	232438397	1	0.1630541194596208	1.9637897565339713	-0.6463761259214303	12.4
5	230460314	1	1.0115232289275187	-0.08112744020559452	-1.2013688016765636	1.15
6	230459870	1	0.961040314282148	-0.08112744020559452	-1.2013688016765636	1.30
7	230333930	1	-0.17126381360463247	-0.18466755143291433	-1.2013688016765636	-0.12

Showing the first 275 rows.



```
%python
# Rename the column
df_classes = df_classes.withColumnRenamed("txId","id")
display(df_classes)
```

	id	class
1	230425980	unknown
2	5530458	unknown
3	232022460	unknown
4	232438397	2
5	230460314	unknown
6	230459870	unknown
7	230333930	unknown

Showing the first 1000 rows.

```
%python
# Rename the column
df_edge = df_edge.withColumnRenamed("txId","id")
display(df_edge)
```

	txId1	txId2
1	230425980	5530458
2	232022460	232438397
3	230460314	230459870
4	230333930	230595899
5	232013274	232029206
6	232344069	27553029
7	36411953	230405052

Showing the first 1000 rows.

```
%python
# Rename the columns of df_feature
oldColumns = df_feature.schema.names
newColumns = ['id', 'time step'] + [f'trans_feat_{i}' for i in range(93)] + [f'agg_feat_{i}' for i in range(72)]

from functools import reduce

df_feature_cleaned = reduce(lambda df_feature, idx: df_feature.withColumnRenamed(oldColumns[idx], newColumns[idx]), range(len(oldColumns)), df_feature)
df_feature_cleaned.printSchema()
```

```
 |-- trans_feat_13: string (nullable = true)
 |-- trans_feat_14: string (nullable = true)
 |-- trans_feat_15: string (nullable = true)
 |-- trans_feat_16: string (nullable = true)
 |-- trans_feat_17: string (nullable = true)
 |-- trans_feat_18: string (nullable = true)
 |-- trans_feat_19: string (nullable = true)
 |-- trans_feat_20: string (nullable = true)
 |-- trans_feat_21: string (nullable = true)
 |-- trans_feat_22: string (nullable = true)
 |-- trans_feat_23: string (nullable = true)
 |-- trans_feat_24: string (nullable = true)
 |-- trans_feat_25: string (nullable = true)
 |-- trans_feat_26: string (nullable = true)
 |-- trans_feat_27: string (nullable = true)
 |-- trans_feat_28: string (nullable = true)
 |-- trans_feat_29: string (nullable = true)
 |-- trans_feat_30: string (nullable = true)
 |-- trans_feat_31: string (nullable = true)
 |-- trans_feat_32: string (nullable = true)
 |-- trans_feat_33: string (nullable = true)
 |-- trans_feat_34: string (nullable = true)
```

Check NULLS

```
#Missing value analysis on the entire dataframe
from pyspark.sql.functions import isnan, when, count, col
df_feature.select([count(when(isnan(c), c)).alias(c) for c in df_feature.columns]).show()
```

```
+---+---+---+---+---+---+---+---+---+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+-
--+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
```

```
#Missing value analysis on the df_classes
df_classes.select([count(when(isnan(c), c)).alias(c) for c in df_classes.columns]).show()
```

```
+---+-----+
| id|class|
+---+-----+
|  0|    0|
+---+-----+
```

```
#Missing value analysis on the df_edge
df_edge.select([count(when(isnan(c), c)).alias(c) for c in df_edge.columns]).show()
```

```
+-----+-----+
|txId1|txId2|
+-----+-----+
|    0|    0|
+-----+-----+
```

Encoding Labels

```
# Data join
from pyspark.sql.functions import udf, col
df_data = df_feature_cleaned.join(df_classes, on=['id'], how='left')

encoding= udf(lambda x: '0' if x == "unknown" else x)
df_data = df_data.withColumn("class",encoding(col("class")))
df_data.registerTempTable("df_data_temp")
#display(df_data)
```

```
/databricks/spark/python/pyspark/sql/dataframe.py:142: DeprecationWarning: Deprecated in 2.0, use createOrReplaceTempView instead.
  "Deprecated in 2.0, use createOrReplaceTempView instead.", DeprecationWarning)
```

```
# Check for distinct class
from pyspark.sql.functions import countDistinct
df_data.select("class").groupBy('class').count().show()

+-----+------+
|class| count|
+-----+------+
|    0|157205|
|    1|  4545|
|    2| 42019|
+-----+------+
```

GARPH

```
from pyspark.sql import *
from graphframes import *
from pyspark.sql.functions import col
from pyspark.sql.functions import *
```

```
# all transactions, but only selected columns
df_vertices_all = spark.sql("SELECT T1.txId, T2.class from elliptic_txs_features_1_csv T1 inner join elliptic_txs_classes_csv T2 on T1.txId = T2.txId")
df_vertices_all = df_vertices_all.withColumnRenamed("txId","id")
df_edges_all = spark.sql("SELECT T3.txId1 as src, T3.txID2 as dst from elliptic_txs_edgelist_csv T3")
g_all = GraphFrame(df_vertices_all, df_edges_all)
```

Page Rank

```
#PageRank: Identify important vertices in a graph
ranks_all = g_all.pageRank(maxIter = 5)
#display(ranks_all.vertices.select("id","pagerank").orderBy(desc("pagerank")))
```

Degrees

```
degrees_all = g_all.degrees.sort(desc("degree"))
degrees_all.registerTempTable("degrees_all_temp")
#display( degrees_all )
```

Append graph features to the original dataset

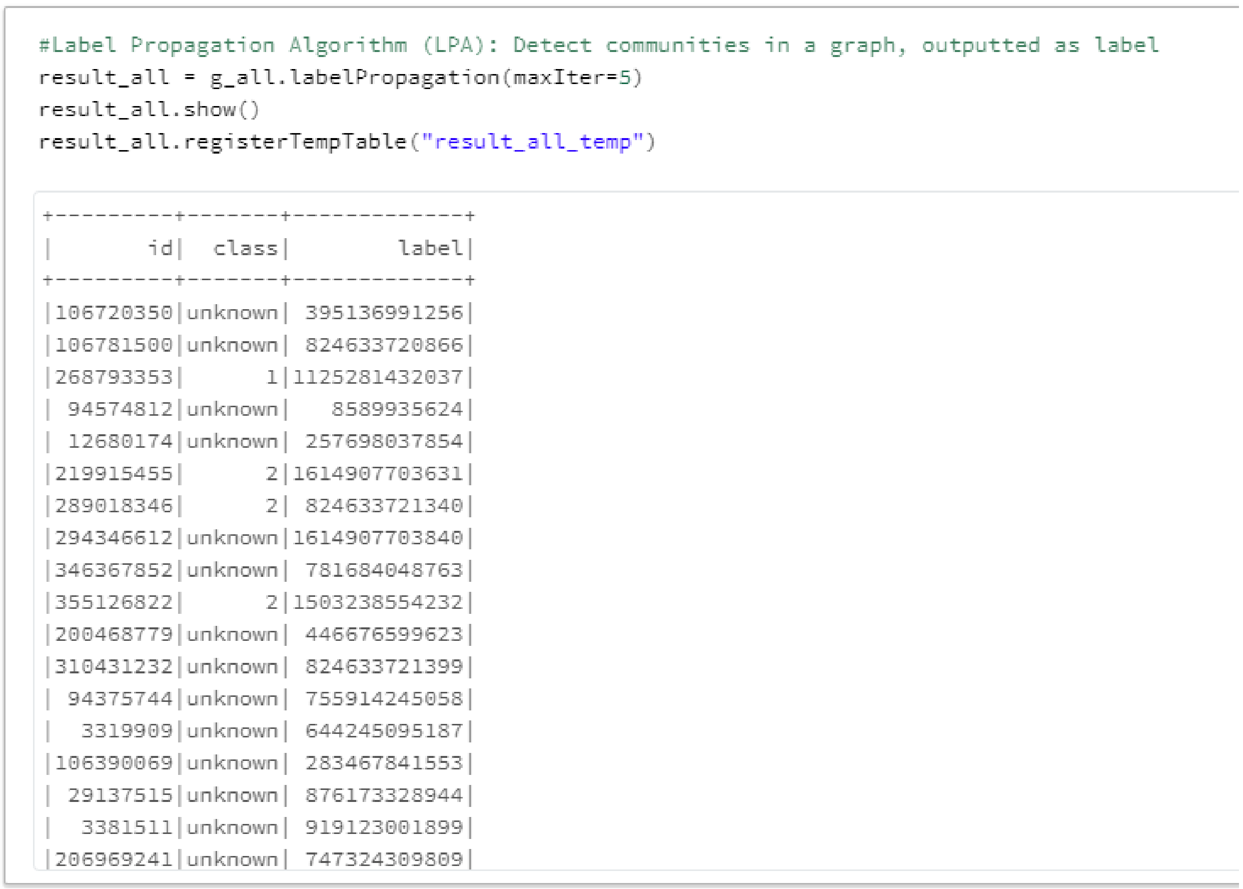
```
# Append graph featres to the orginal dataset
Data_combined = spark.sql("SELECT T2.pagerank, T3.degree, T1.* from df_data_temp T1 left join PageRank_temp T2 on T1.id = T2.id left join degrees_all_temp T3 on T2.id = T3.id ")
```

```
# Checke for the appended columns
display(Data_combined)
```

	pagerank	degree	id	time step	trans_feat_0	trans_feat_1	trans_feat_2	trans_feat_3	trans_feat_4	trans_feat_
1	0.31115766036575415	1	100617351	48	-0.17260438503505138	-0.11400142552026857	-0.09138345016629715	-0.046932091385977995	-0.04387454791734898	-0.0291397
2	0.32217782750370794	1	101208888	25	-0.17241193786795864	-0.16960246524933928	0.46360922558883605	-0.12196959975910057	-0.04387454791734898	-0.1130020
3	0.31115766036575415	1	101620257	5	-0.17253761932527448	-0.18466755143291433	-1.2013688016765636	-0.12196959975910057	-0.04387454791734898	-0.1130020
4	0.31115766036575415	3	105411194	44	0.12976582554188143	0.40286822122373117	1.5735945770991024	0.25321794210651233	0.07522560401157881	0.30630918
5	0.31115766036575415	2	105426769	44	-0.1716551380035851	-0.11385388086176963	1.018601901343969	-0.12196959975910057	-0.04387454791734898	-0.1130020
6	0.5756416716766452	3	105594237	44	-0.1666119906854753	-0.05052616033236017	1.018601901343969	-0.046932091385977995	-0.04387454791734898	-0.0291397
7	0.5756416716766452	3	105594840	44	-0.06282842939279747	-0.11385129235898896	1.018601901343969	-0.12196959975910057	-0.04387454791734898	-0.1130020

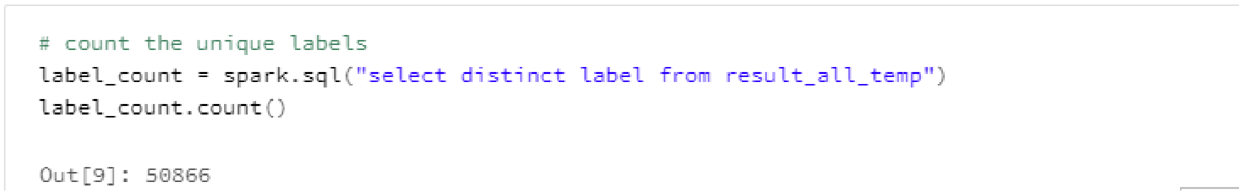
Showing the first 273 rows.

Label Propagation:



```
#Label Propagation Algorithm (LPA): Detect communities in a graph, outputted as label
result_all = g_all.labelPropagation(maxIter=5)
result_all.show()
result_all.registerTempTable("result_all_temp")
```

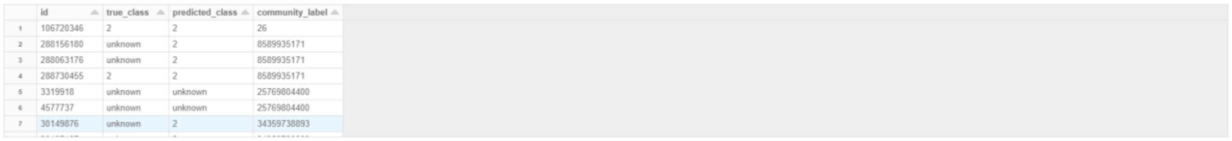
```
+---------+-------+-------------+
|       id|  class|        label|
+---------+-------+-------------+
|106720350|unknown| 395136991256|
|106781500|unknown| 824633720866|
|268793353|      1|1125281432037|
| 94574812|unknown|   8589935624|
| 12680174|unknown| 257698037854|
|219915455|      2|1614907703631|
|289018346|      2| 824633721340|
|294346612|unknown|1614907703840|
|346367852|unknown| 781684048763|
|355126822|      2|1503238554232|
|200468779|unknown| 446676599623|
|310431232|unknown| 824633721399|
| 94375744|unknown| 755914245058|
|  3319909|unknown| 644245095187|
|106390069|unknown| 283467841553|
| 29137515|unknown| 876173328944|
|  3381511|unknown| 919123001899|
|206969241|unknown| 747324309809|
```



```
# count the unique labels
label_count = spark.sql("select distinct label from result_all_temp")
label_count.count()

Out[9]: 50866
```

```
# prediction algorithm
# select label and the highest counted id classes for that label
# if unknown counts > other counts (class 1 and class 2), we prioritize and rank other counts first (class 1 and class 2)

result_classify = spark.sql("select distinct label, class, count(class) as count ,ROW_NUMBER() OVER (PARTITION BY label order by label) as Row from result_all_temp group by label, class order by label, count desc")
result_classify.registerTempTable("result_classify_temp")
step1 = result_classify.count()
result_classify_1 = spark.sql("select distinct label, class, count, ROW_NUMBER() OVER (PARTITION BY label order by label) as Row from result_classify_temp where class <> 'unknown' order by label, count desc")
result_classify_1.registerTempTable("result_classify_1_temp")
step1_1 = result_classify_1.count()
result_classify_1 = spark.sql("select distinct label, class as predicted_class from result_classify_1_temp where Row = '1'")
result_classify_1.registerTempTable("result_classify_1_temp")
step1_2 = result_classify_1.count()
result_classify = spark.sql("select distinct label, class as predicted_class from result_classify_temp where Row = '1'")
result_classify.registerTempTable("result_classify_temp")
step2 = result_classify.count()
```

```
# join back the ids based on community_label and write the predicted_class too
# some predict_class is unknown because there is no single id in this community that is being labelled.

predicted_results = spark.sql("select t1.id, t1.class as true_class, case when t3.predicted_class is not null then t3.predicted_class else t2.predicted_class end as predicted_class, t1.label as community_label from result_all_temp t1 left 
join result_classify_temp t2 on t1.label = t2.label left join result_classify_1_temp t3 on t1.label = t3.label")
predicted_results.registerTempTable("predicted_results_temp")
display(predicted_results)
```



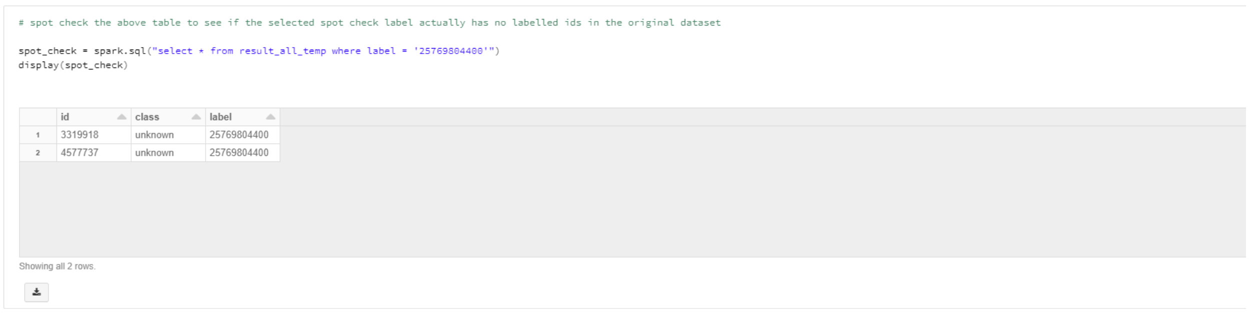
	id	true_class	predicted_class	community_label
1	106720346	2	2	26
2	288156180	unknown	2	8589935171
3	288063176	unknown	2	8589935171
4	288730455	2	2	8589935171
5	3319918	unknown	unknown	25769804400
6	4577737	unknown	unknown	25769804400
7	30149876	unknown	2	34359738893

Showing the first 1000 rows.

```
# find the number of predictable results
predicted_results = spark.sql("select count(*) as predicted_results from predicted_results_temp where predicted_class <> 'unknown'")
predicted_results.show()

+-----------------+
|predicted_results|
+-----------------+
|           119802|
+-----------------+
```

```
# find the number of unpredictable results
unpredictable_results = spark.sql("select count(*) as unpredictable_results from predicted_results_temp where predicted_class = 'unknown'")
unpredictable_results.show()

+---------------------+
|unpredictable_results|
+---------------------+
|                83967|
+---------------------+
```

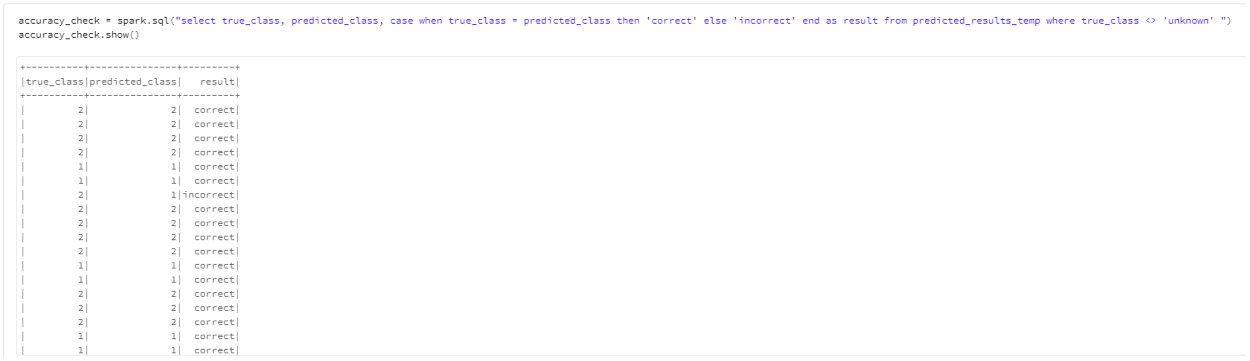
```
# get the details of the unpredictable results
unpredictable_results_details = spark.sql("select distinct * from predicted_results_temp where predicted_class = 'unknown'")
unpredictable_results_details.show()

+---------+----------+---------------+---------------+
|       id|true_class|predicted_class|community_label|
+---------+----------+---------------+---------------+
|  3319918|   unknown|        unknown|    25769804400|
|  4577737|   unknown|        unknown|    25769804400|
|235844286|   unknown|        unknown|    42949673366|
|234864427|   unknown|        unknown|    42949673366|
|234591938|   unknown|        unknown|    60129542543|
|234590508|   unknown|        unknown|    60129542543|
|121711130|   unknown|        unknown|    68719476798|
| 94675680|   unknown|        unknown|    68719477722|
| 94192698|   unknown|        unknown|    68719477722|
| 94192384|   unknown|        unknown|    68719477722|
| 54714209|   unknown|        unknown|    85899346665|
| 54709642|   unknown|        unknown|    85899346665|
| 54710284|   unknown|        unknown|    85899346665|
```



```
# spot check the above table to see if the selected spot check label actually has no labelled ids in the original dataset

spot_check = spark.sql("select * from result_all_temp where label = '25769804400'")
display(spot_check)
```

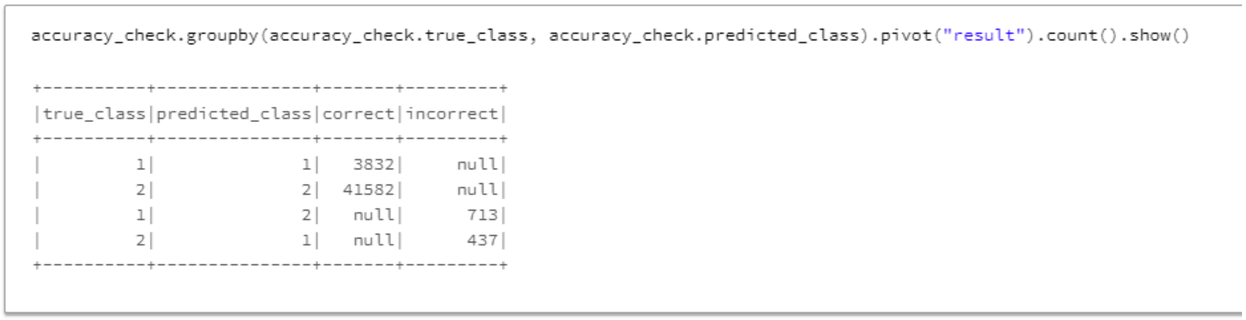
	id	class	label
1	3319918	unknown	25769804400
2	4577737	unknown	25769804400

Showing all 2 rows.

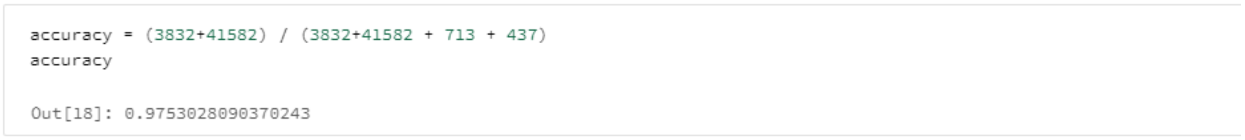


```
accuracy_check = spark.sql("select true_class, predicted_class, case when true_class = predicted_class then 'correct' else 'incorrect' end as result from predicted_results_temp where true_class <> 'unknown' ")
accuracy_check.show()
```

```
+----------+---------------+---------+
|true_class|predicted_class|   result|
+----------+---------------+---------+
|         2|              2|  correct|
|         2|              2|  correct|
|         2|              2|  correct|
|         2|              2|  correct|
|         1|              1|  correct|
|         1|              1|  correct|
|         2|              1|incorrect|
|         2|              2|  correct|
|         2|              2|  correct|
|         2|              2|  correct|
|         2|              2|  correct|
|         1|              1|  correct|
|         1|              1|  correct|
|         2|              2|  correct|
|         2|              2|  correct|
|         2|              2|  correct|
|         1|              1|  correct|
|         1|              1|  correct|
```

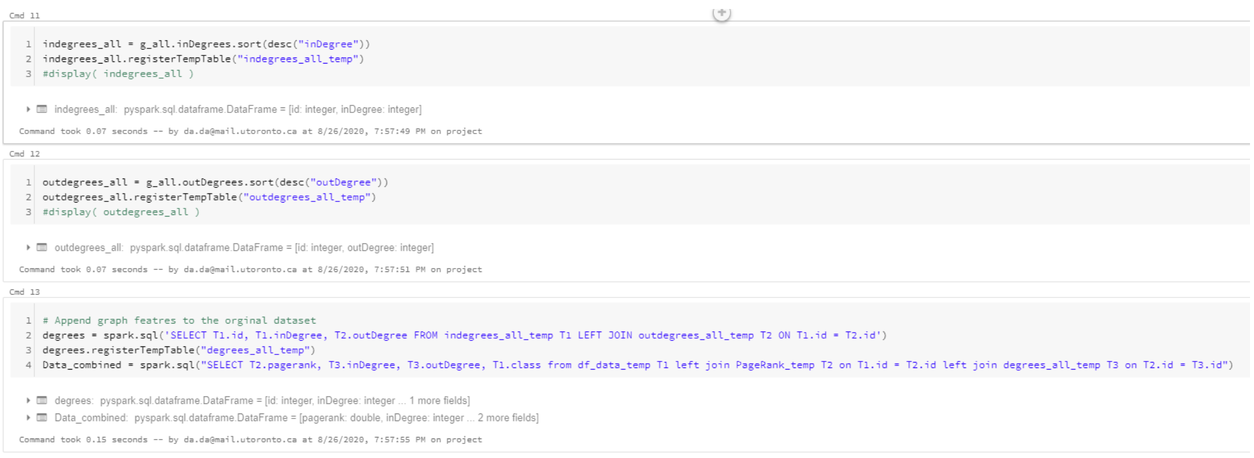


```
accuracy_check.groupby(accuracy_check.true_class, accuracy_check.predicted_class).pivot("result").count().show()

+----------+---------------+-------+---------+
|true_class|predicted_class|correct|incorrect|
+----------+---------------+-------+---------+
|         1|              1|   3832|     null|
|         2|              2|  41582|     null|
|         1|              2|   null|      713|
|         2|              1|   null|      437|
+----------+---------------+-------+---------+
```



```
accuracy = (3832+41582) / (3832+41582 + 713 + 437)
accuracy

Out[18]: 0.9753028090370243
```

Graph Features Only:



Cmd 11

```
indegrees_all = g_all.inDegrees.sort(desc("inDegree"))
indegrees_all.registerTempTable("indegrees_all_temp")
#display( indegrees_all )
```

```
indegrees_all:  pyspark.sql.dataframe.DataFrame = [id: integer, inDegree: integer]
Command took 0.07 seconds -- by da.da@mail.utoronto.ca at 8/26/2020, 7:57:49 PM on project
```

Cmd 12

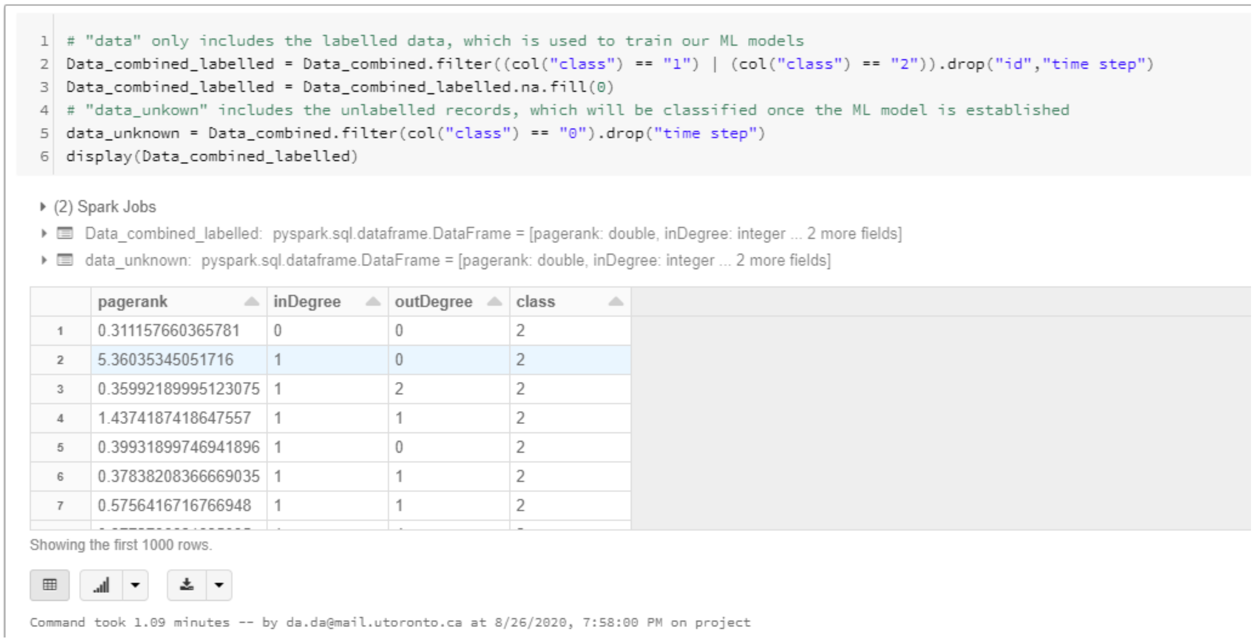
```
outdegrees_all = g_all.outDegrees.sort(desc("outDegree"))
outdegrees_all.registerTempTable("outdegrees_all_temp")
#display( outdegrees_all )
```

```
outdegrees_all:  pyspark.sql.dataframe.DataFrame = [id: integer, outDegree: integer]
Command took 0.07 seconds -- by da.da@mail.utoronto.ca at 8/26/2020, 7:57:51 PM on project
```

Cmd 13

```
# Append graph featres to the orginal dataset
degrees = spark.sql('SELECT T1.id, T1.inDegree, T2.outDegree FROM indegrees_all_temp T1 LEFT JOIN outdegrees_all_temp T2 ON T1.id = T2.id')
degrees.registerTempTable("degrees_all_temp")
Data_combined = spark.sql("SELECT T2.pagerank, T3.inDegree, T3.outDegree, T1.class from df_data_temp T1 left join PageRank_temp T2 on T1.id = T2.id left join degrees_all_temp T3 on T2.id = T3.id")
```

```
degrees:  pyspark.sql.dataframe.DataFrame = [id: integer, inDegree: integer ... 1 more fields]
Data_combined:  pyspark.sql.dataframe.DataFrame = [pagerank: double, inDegree: integer ... 2 more fields]
Command took 0.15 seconds -- by da.da@mail.utoronto.ca at 8/26/2020, 7:57:55 PM on project
```



```
# "data" only includes the labelled data, which is used to train our ML models
Data_combined_labelled = Data_combined.filter((col("class") == "1") | (col("class") == "2")).drop("id","time step")
Data_combined_labelled = Data_combined_labelled.na.fill(0)
# "data_unkown" includes the unlabelled records, which will be classified once the ML model is established
data_unknown = Data_combined.filter(col("class") == "0").drop("time step")
display(Data_combined_labelled)
```

```
(2) Spark Jobs
Data_combined_labelled:  pyspark.sql.dataframe.DataFrame = [pagerank: double, inDegree: integer ... 2 more fields]
data_unknown:  pyspark.sql.dataframe.DataFrame = [pagerank: double, inDegree: integer ... 2 more fields]
```

	pagerank	inDegree	outDegree	class
1	0.311157660365781	0	0	2
2	5.36035345051716	1	0	2
3	0.35992189995123075	1	2	2
4	1.4374187418647557	1	1	2
5	0.39931899746941896	1	0	2
6	0.37838208366669035	1	1	2
7	0.5756416716766948	1	1	2

Showing the first 1000 rows.

```
Command took 1.09 minutes -- by da.da@mail.utoronto.ca at 8/26/2020, 7:58:00 PM on project
```

Cmd 18

```python
# train and test data split
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2], seed =2020)

# Fit Model
rf = RandomForestClassifier(labelCol='class', featuresCol='features', maxDepth=5)
model = rf.fit(training_data)
rf_predictions = model.transform(test_data)
```

▸ (5) Spark Jobs

▸ training_data: pyspark.sql.dataframe.DataFrame = [pagerank: float, inDegree: float ... 3 more fields]

▸ test_data: pyspark.sql.dataframe.DataFrame = [pagerank: float, inDegree: float ... 3 more fields]

▸ rf_predictions: pyspark.sql.dataframe.DataFrame = [pagerank: float, inDegree: float ... 6 more fields]

```
Command took 4.79 minutes -- by da.da@mail.utoronto.ca at 8/26/2020, 1:39:46 PM on Project
```

Cmd 19

```python
# Evaluate the random forest classifer model
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
#from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol = 'class', metricName = 'accuracy')
print('Random Forest classifier Accuracy:', evaluator.evaluate(rf_predictions))
```

▸ (1) Spark Jobs

```
Random Forest classifier Accuracy: 0.898960983884648
```

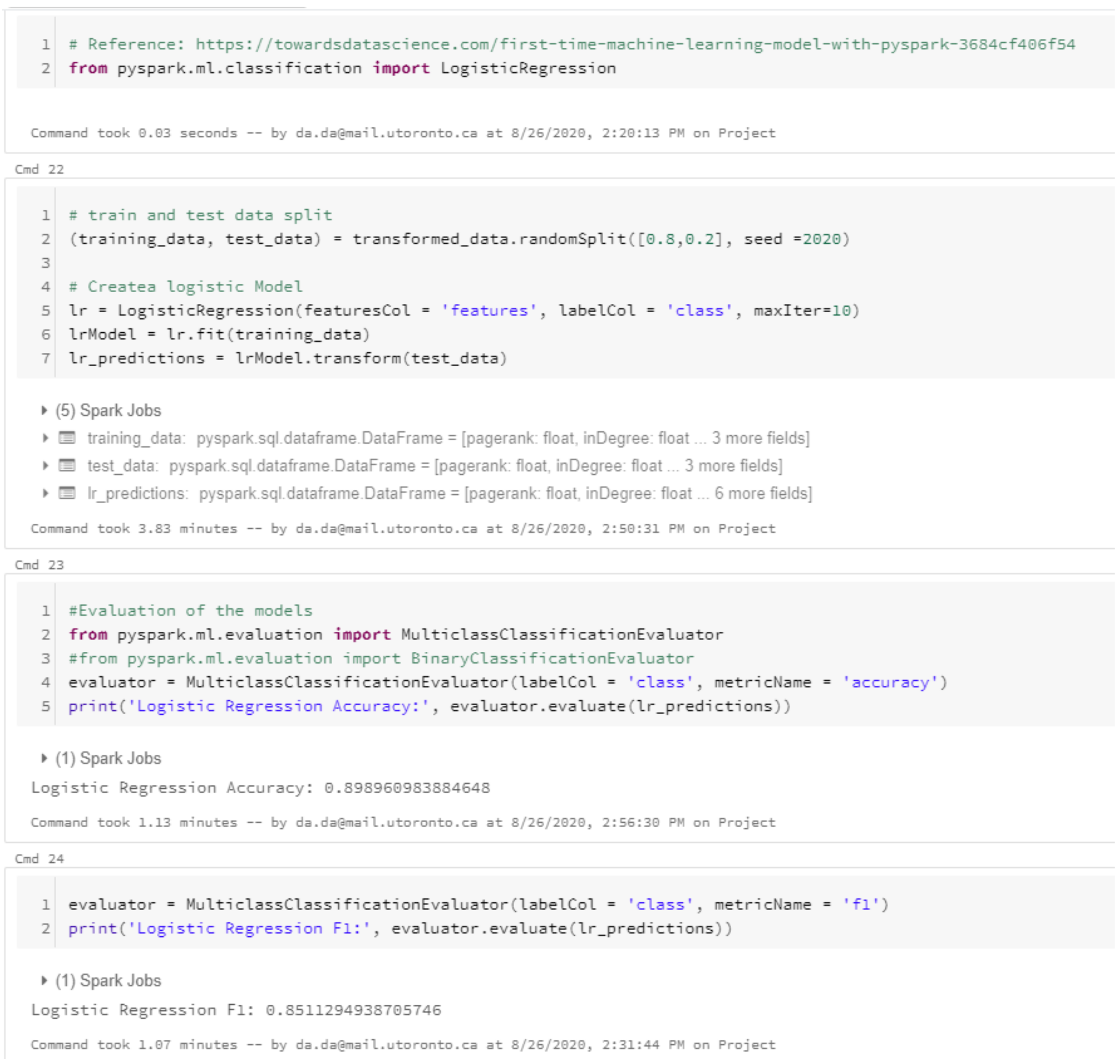
```
Command took 1.24 minutes -- by da.da@mail.utoronto.ca at 8/26/2020, 1:45:01 PM on Project
```

Cmd 20

```python
evaluator = MulticlassClassificationEvaluator(labelCol = 'class', metricName = 'f1')
print('Random Forest classifier F1:', evaluator.evaluate(rf_predictions))
```

▸ (1) Spark Jobs

```
Random Forest classifier F1: 0.8511294938705746
```



```
# Reference: https://towardsdatascience.com/first-time-machine-learning-model-with-pyspark-3684cf406f54
from pyspark.ml.classification import LogisticRegression
```

```
Command took 0.03 seconds -- by da.da@mail.utoronto.ca at 8/26/2020, 2:20:13 PM on Project
```

Cmd 22

```
# train and test data split
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2], seed =2020)

# Createa logistic Model
lr = LogisticRegression(featuresCol = 'features', labelCol = 'class', maxIter=10)
lrModel = lr.fit(training_data)
lr_predictions = lrModel.transform(test_data)
```

```
(5) Spark Jobs
training_data:  pyspark.sql.dataframe.DataFrame = [pagerank: float, inDegree: float ... 3 more fields]
test_data:  pyspark.sql.dataframe.DataFrame = [pagerank: float, inDegree: float ... 3 more fields]
lr_predictions:  pyspark.sql.dataframe.DataFrame = [pagerank: float, inDegree: float ... 6 more fields]
Command took 3.83 minutes -- by da.da@mail.utoronto.ca at 8/26/2020, 2:50:31 PM on Project
```

Cmd 23

```
#Evaluation of the models
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
#from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol = 'class', metricName = 'accuracy')
print('Logistic Regression Accuracy:', evaluator.evaluate(lr_predictions))
```

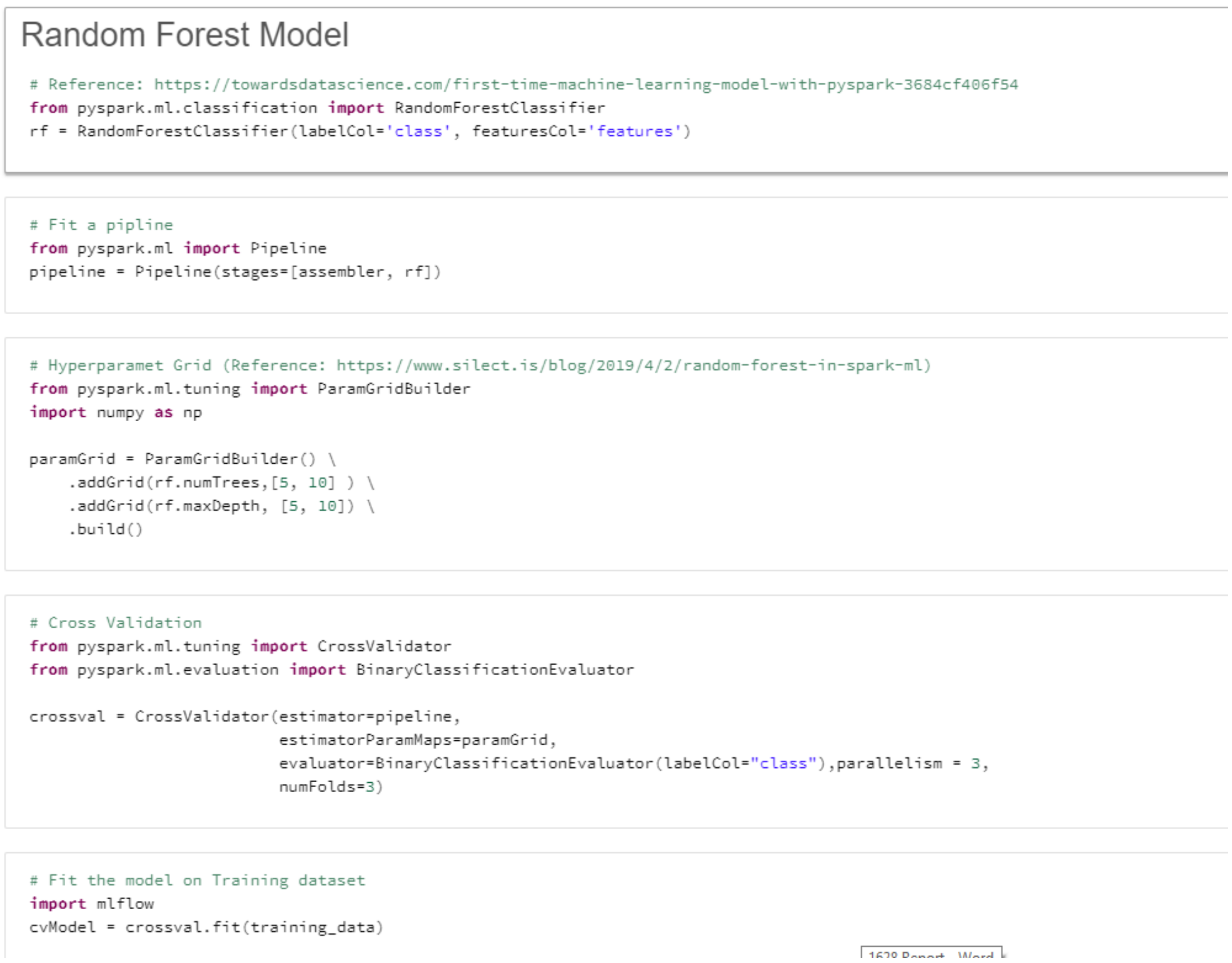
```
(1) Spark Jobs
Logistic Regression Accuracy: 0.898960983884648
Command took 1.13 minutes -- by da.da@mail.utoronto.ca at 8/26/2020, 2:56:30 PM on Project
```

Cmd 24

```
evaluator = MulticlassClassificationEvaluator(labelCol = 'class', metricName = 'f1')
print('Logistic Regression F1:', evaluator.evaluate(lr_predictions))
```

```
(1) Spark Jobs
Logistic Regression F1: 0.8511294938705746
Command took 1.07 minutes -- by da.da@mail.utoronto.ca at 8/26/2020, 2:31:44 PM on Project
```

Node Features Only:



Random Forest Model

```python
# Reference: https://towardsdatascience.com/first-time-machine-learning-model-with-pyspark-3684cf406f54
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol='class', featuresCol='features')
```

```python
# Fit a pipline
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[assembler, rf])
```

```python
# Hyperparamet Grid (Reference: https://www.silect.is/blog/2019/4/2/random-forest-in-spark-ml)
from pyspark.ml.tuning import ParamGridBuilder
import numpy as np

paramGrid = ParamGridBuilder() \
    .addGrid(rf.numTrees,[5, 10] ) \
    .addGrid(rf.maxDepth, [5, 10]) \
    .build()
```

```python
# Cross Validation
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(labelCol="class"),parallelism = 3,
                          numFolds=3)
```

```python
# Fit the model on Training dataset
import mlflow
cvModel = crossval.fit(training_data)
```

1628 Report - Word

```python
# Make precision on testing dataset
rf_predictions = cvModel.transform(test_data)
```

Evaluation Metrics Report

```python
# Evualtaion matrix report
import sklearn
y_true = rf_predictions.select(['class']).collect()
y_pred = rf_predictions.select(['prediction']).collect()

from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.99      0.72      0.83       889
         2.0       0.97      1.00      0.98      8330

    accuracy                           0.97      9219
   macro avg       0.98      0.86      0.91      9219
weighted avg       0.97      0.97      0.97      9219
```

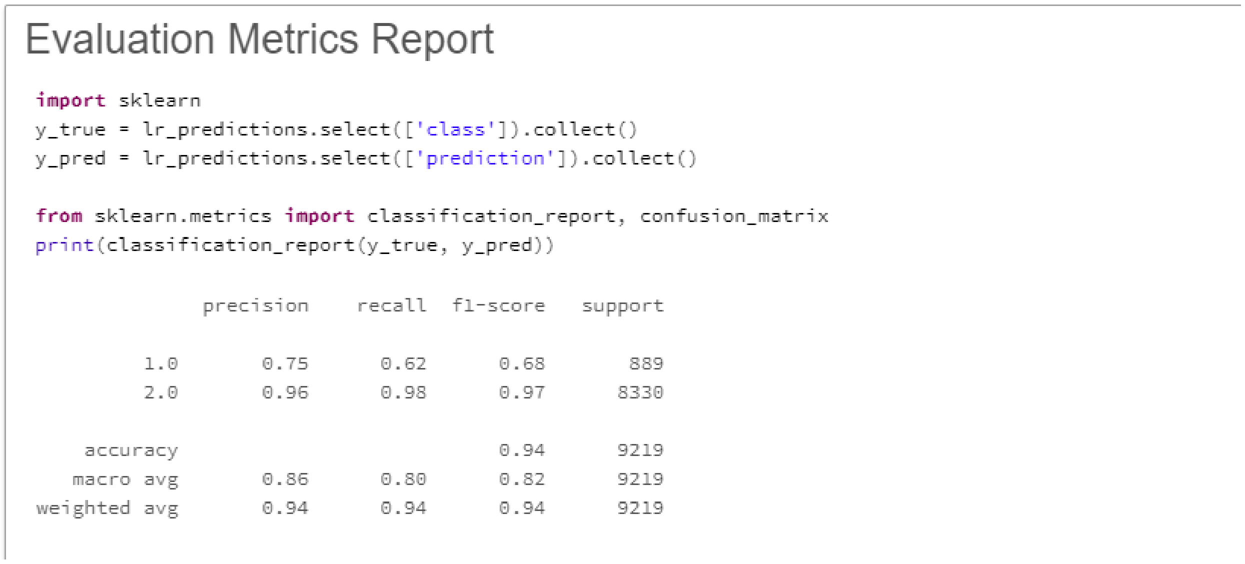
Logistic Regression Model

```python
# Reference: https://towardsdatascience.com/first-time-machine-learning-model-with-pyspark-3684cf406f54
from pyspark.ml.classification import LogisticRegression

# Createa logistic Model
lr = LogisticRegression(featuresCol = 'features', labelCol = 'class', maxIter=10)
```

```python
# Hyperparamet Grid (Reference: https://www.silect.is/blog/2019/4/2/random-forest-in-spark-ml)
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# train and test data split
(training_data, test_data) = data_float.randomSplit([0.8,0.2], seed =2020)

# Fit a pipline
pipeline_lr = Pipeline(stages=[assembler, lr])

# Define a Hyperparamter Grid
paramGrid_lr = ParamGridBuilder().addGrid(lr.regParam, [0,0.01, 0.5]).build()

#Cross Validation
cv_lr = CrossValidator(estimator=pipeline_lr,estimatorParamMaps=paramGrid_lr,evaluator=BinaryClassificationEvaluator(labelCol="class"), numFolds=3)
```

```python
# Fit the logistic mdoel on training dataset
cvModel_lr = cv_lr.fit(training_data)
```

```python
# Make precision on testing dataset
lr_predictions = cvModel_lr.transform(test_data)
```

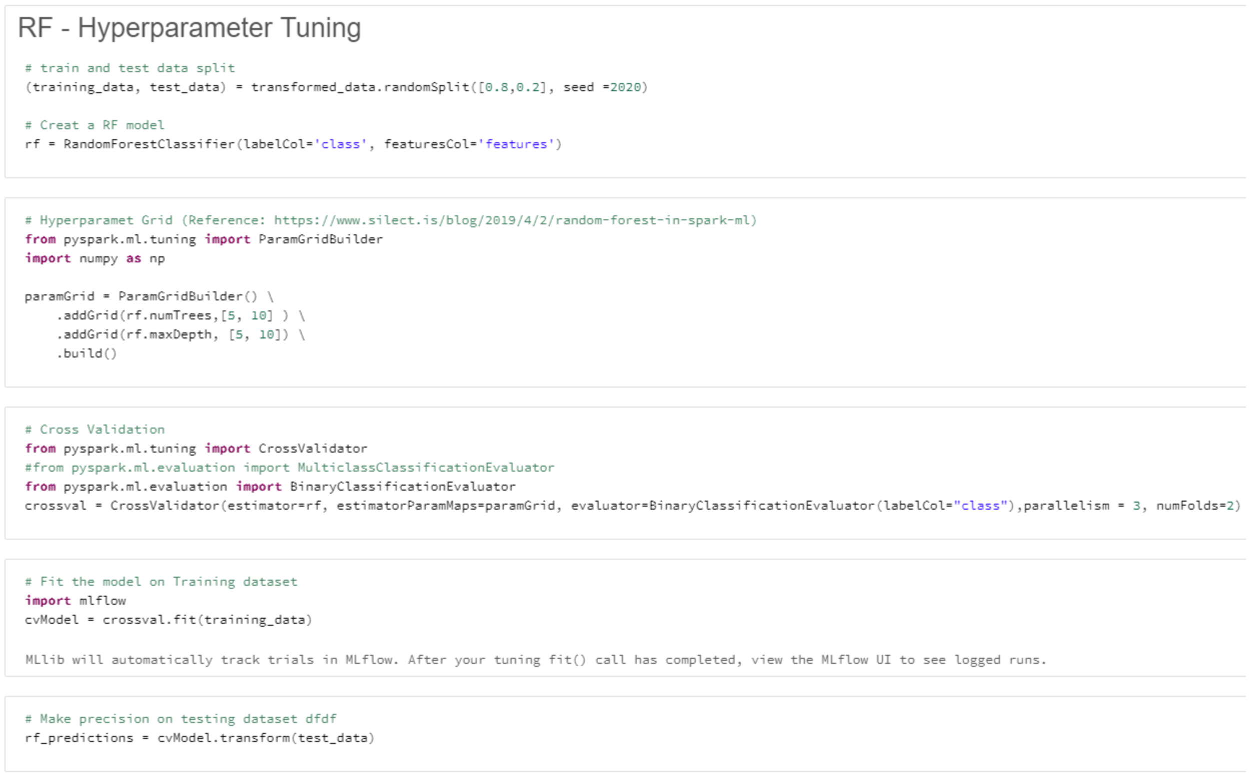


Evaluation Metrics Report

```python
import sklearn
y_true = lr_predictions.select(['class']).collect()
y_pred = lr_predictions.select(['prediction']).collect()

from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.75      0.62      0.68       889
         2.0       0.96      0.98      0.97      8330

    accuracy                           0.94      9219
   macro avg       0.86      0.80      0.82      9219
weighted avg       0.94      0.94      0.94      9219
```

Aggregate model with both node and graph features:



RF - Hyperparameter Tuning

```python
# train and test data split
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2], seed =2020)

# Creat a RF model
rf = RandomForestClassifier(labelCol='class', featuresCol='features')
```

```python
# Hyperparamet Grid (Reference: https://www.silect.is/blog/2019/4/2/random-forest-in-spark-ml)
from pyspark.ml.tuning import ParamGridBuilder
import numpy as np

paramGrid = ParamGridBuilder() \
    .addGrid(rf.numTrees,[5, 10] ) \
    .addGrid(rf.maxDepth, [5, 10]) \
    .build()
```

```python
# Cross Validation
from pyspark.ml.tuning import CrossValidator
#from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.evaluation import BinaryClassificationEvaluator
crossval = CrossValidator(estimator=rf, estimatorParamMaps=paramGrid, evaluator=BinaryClassificationEvaluator(labelCol="class"),parallelism = 3, numFolds=2)
```

```python
# Fit the model on Training dataset
import mlflow
cvModel = crossval.fit(training_data)
```

```
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
```

```python
# Make precision on testing dataset dfdf
rf_predictions = cvModel.transform(test_data)
```

Evaluation Metrics Report

```
# Evualtaion matrix report
import sklearn
y_true = rf_predictions.select(['class']).collect()
y_pred = rf_predictions.select(['prediction']).collect()

from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.99      0.74      0.85       949
         2.0       0.97      1.00      0.99      8505

    accuracy                           0.97      9454
   macro avg       0.98      0.87      0.92      9454
weighted avg       0.97      0.97      0.97      9454
```

```
# Extract the hyperparameters
bestPipeline_RF = cvModel.bestModel
bestParams_RF = cvModel.extractParamMap()
bestPipeline_RF
```

```
Out[34]: RandomForestClassificationModel: uid=RandomForestClassifier_2734116d485d, numTrees=10, numClasses=3, numFeatures=168
```

Logistic Regression Model

```
# Reference: https://towardsdatascience.com/first-time-machine-learning-model-with-pyspark-3684cf406f54
from pyspark.ml.classification import LogisticRegression
```

Logistic Regression - Hyperparameter Tuning

```
# Hyperparamet Grid (Reference: https://www.silect.is/blog/2019/4/2/random-forest-in-spark-ml)
# train and test data split
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2], seed =2020)

# Createa logistic Model
lr = LogisticRegression(featuresCol = 'features', labelCol = 'class', maxIter=10)
```

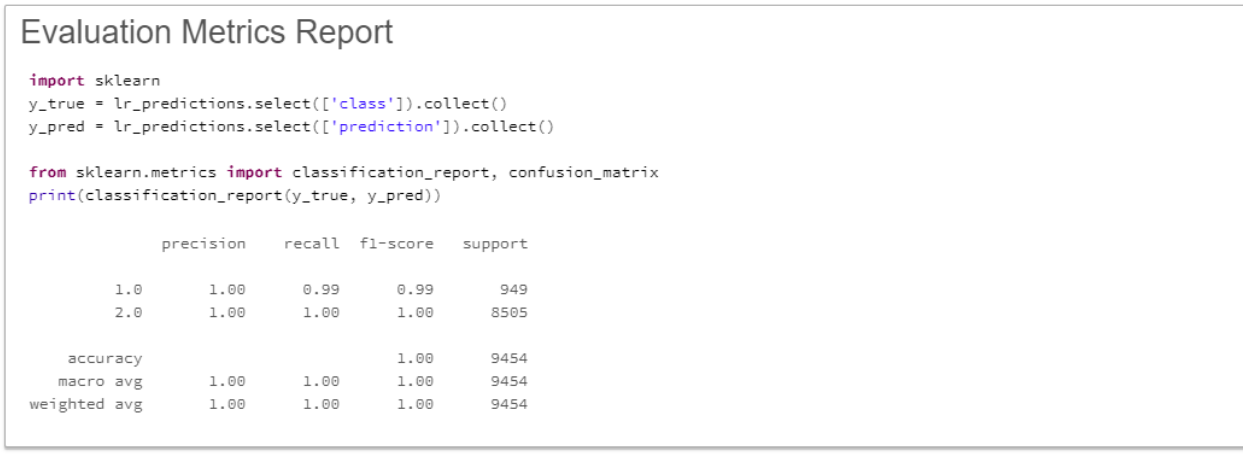
```
# Define a Hyperparamter Grid
paramGrid_lr = ParamGridBuilder().addGrid(lr.regParam, [0,0.01, 0.5]).build() # regParam corresponds to lambda (regularization rate) When λ = 0, the pe
least squares
```

```
#Cross Validation
from pyspark.ml.evaluation import BinaryClassificationEvaluator
cv_lr = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid_lr,evaluator=BinaryClassificationEvaluator(labelCol="class"),numFolds=3)
```

```
# Fit the logistic mdoel on training dataset
cvModel_lr = cv_lr.fit(training_data)
```

```
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
```

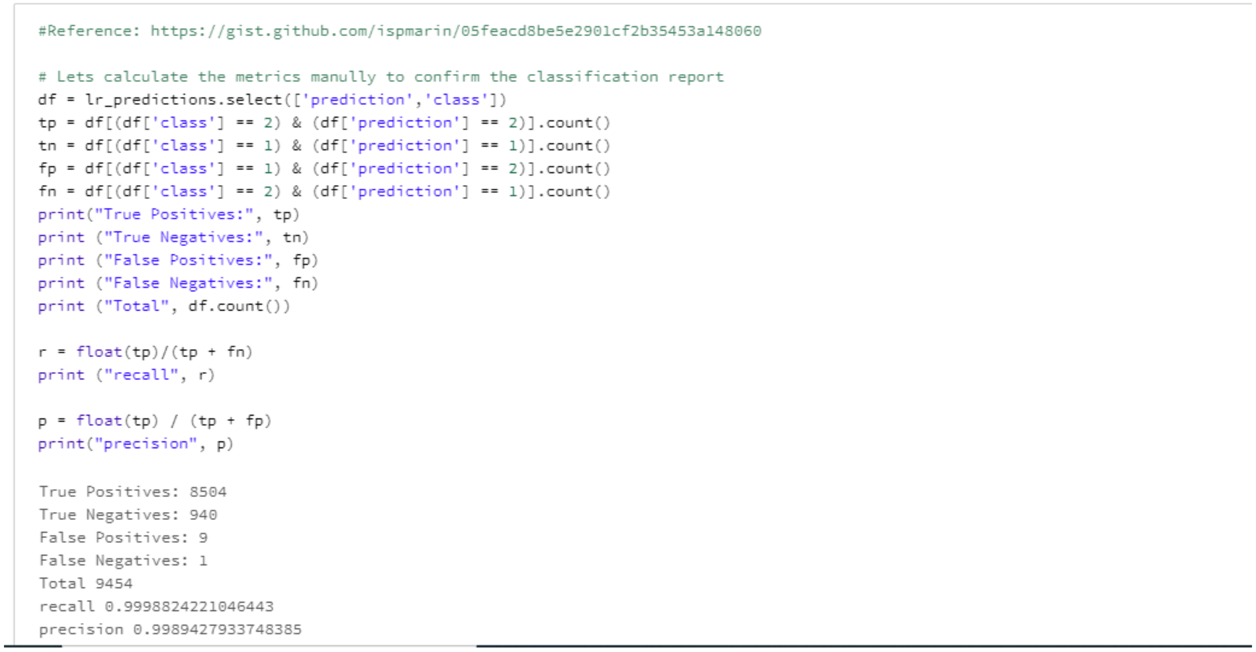
```
# Make precision on testing dataset
lr_predictions = cvModel_lr.transform(test_data)
```



Evaluation Metrics Report

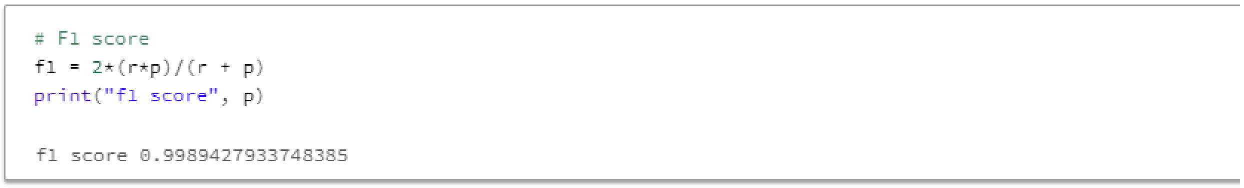
```
import sklearn
y_true = lr_predictions.select(['class']).collect()
y_pred = lr_predictions.select(['prediction']).collect()

from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       1.00      0.99      0.99       949
         2.0       1.00      1.00      1.00      8505

    accuracy                           1.00      9454
   macro avg       1.00      1.00      1.00      9454
weighted avg       1.00      1.00      1.00      9454
```



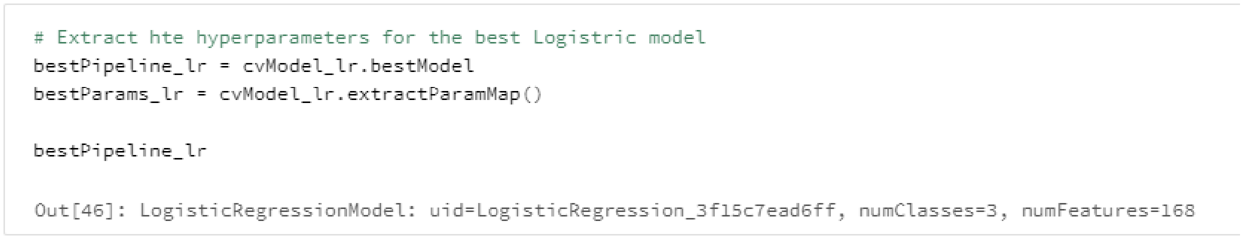
```
#Reference: https://gist.github.com/ispmarin/05feacd8be5e2901cf2b35453a148060

# Lets calculate the metrics manully to confirm the classification report
df = lr_predictions.select(['prediction','class'])
tp = df[(df['class'] == 2) & (df['prediction'] == 2)].count()
tn = df[(df['class'] == 1) & (df['prediction'] == 1)].count()
fp = df[(df['class'] == 1) & (df['prediction'] == 2)].count()
fn = df[(df['class'] == 2) & (df['prediction'] == 1)].count()
print("True Positives:", tp)
print ("True Negatives:", tn)
print ("False Positives:", fp)
print ("False Negatives:", fn)
print ("Total", df.count())

r = float(tp)/(tp + fn)
print ("recall", r)

p = float(tp) / (tp + fp)
print("precision", p)
```

```
True Positives: 8504
True Negatives: 940
False Positives: 9
False Negatives: 1
Total 9454
recall 0.9998824221046443
precision 0.9989427933748385
```



```
# F1 score
f1 = 2*(r*p)/(r + p)
print("f1 score", p)
```

```
f1 score 0.9989427933748385
```



```
# Extract hte hyperparameters for the best Logistric model
bestPipeline_lr = cvModel_lr.bestModel
bestParams_lr = cvModel_lr.extractParamMap()

bestPipeline_lr
```

```
Out[46]: LogisticRegressionModel: uid=LogisticRegression_3f15c7ead6ff, numClasses=3, numFeatures=168
```