1. Problem Formulation.

1.1. Identify fraud patterns

Payment fraud is one of many types of fraud. As the name suggests, it is aimed to steal funds during payment. The payment credentials of victims are compromised, and vulnerable for fraudsters to access without consent in most of the cases. Some of the payment fraud types are credit or debit card fraud, phishing fraud, card not present (CNP) fraud, identity theft, wire fraud, push payment fraud, mobile payment fraud, account takeover and chargeback fraud (*Common Payment Fraud Scams: How to Stay Safe*, 2023).

This type of fraud has its patterns in terms of anomalies, which can be observed, detected or analyzed through big data collection. One of the biggest goals of the study is to improve the accuracy of identifying fraud in transactions, especially payment fraud. Common fraud patterns included the usage of fake email, purchasing expensive products, brief time spent on the site and performing many small transactions (Bence Jendruszak, 2017).

1.2. Big Data analytics technique

Big data is defined by the 5Vs: large data volume, high velocity of data generation, various forms, data veracity (accuracy or high quality) and valuable data. The conventional data analytics are not capable of handling such volume and complexity of the big data. Hence, big data analytics is preferred due to the ability to handle those properties with the help of big data techniques and tools, this allows hidden patterns or trends to be spotted, and gives insights for stakeholder to make decisions, as well as helping in operation or cost optimization (Duggal, 2022).

The big data tools involved for the study is Apache Spark, an open-source framework for big data which runs the process in memory rather than hard disk, therefore it is faster. It is beginner friendly in terms of the number of codes required and supports multiple programming languages, SQL included. Spark is easy to manage as it can handle multiple processing tasks on a single cluster, such as batch processing, interactive queries, machine learning, and streaming (Team, 2016).

Fraud detection is about spotting anomalies, Apache Spark able to make most contribution to achieve the expectation starting by feature engineering, required to extract and group the features related to transaction and card holder, then store them in different tables (*Real Time Credit Card Fraud Detection with Apache Spark and Event Streaming*, 2020). Then, big data techniques can be applied:

a. Data mining: To discover the hidden patterns and detect anomalies in the transactions by using Random Forest, which is an ensemble learning method with high accuracy and flexibility, with low over-fitting risk by reducing the variance through multiple decision trees. However, it is time consuming and needs more resources to store the result of large data sets (IBM, 2021).
b. Machine Learning: Split dataset into 2 parts, training, and testing. Training set is used for model building, then compare with the testing set to check the model accuracy. To enhance accuracy, parameters can be adjusted to repeat the testing until satisfaction is achieved. Since fraud records are already present in each record of data set, and fraud detection is a classification problem, therefore supervised machine learning model is required. List of models used by using Spark Machine Learning library (Spark MLlib):
   - Logistic regression: A statistical approach to identify predictors of a dichotomous outcome of binary value, where the outcome is predicted by fitting independent variables to the curve (Aletaha & Huizinga, 2009). The outcome is “is_fraud”, while predictors are the rest of the variables.
   - Support Vector Machine (SVM): Differentiate the classes or boundaries in classification problem by identifying the hyperplane that maximise the margin between closest points of distinct classes. The space dimension of where hyperplane is dependent on the number of features (IBM, 2023). The classes are 0 and 1 for “is_fraud” feature.

2. Conventional techniques

The common conventional techniques are applicable for small or medium sized data sets, which ranged from less than 10GB to 1TB (You Got It or Not? How to Tell If Big Data Is Something You Need to Worry about in Your Application | Abas, 2022). Logistic Regression and Random Forest can be used for fraud detection in this case.

Precondition:

Before running the machine learning algorithms, the cleaned data set needs to be imported into the Python script. Then, encoder (Label Encoder) is being run to convert string values into integer type for Logistic Regression and Random Forest to be run successfully. Next, split the data set into 70% for training and 30% for testing, there are now 4 sets: X_train (training features), X_test (testing features), y_train (training target) and y_test (testing target).

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('feature_fraud.csv')

# get object columns
object_col=df.select_dtypes(include='object').columns.to_list()

# Apply LabelEncoder
encoder = LabelEncoder()
for i in object_col:
    df[i] = encoder.fit_transform(df[i])

# Assume 'fraud' column is the target variable, 1 for fraud, 0 for non-fraud
X = df.drop('is_fraud', axis=1)  # Features
y = df['is_fraud']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=50)
```

Code snippet to split data into training and testing set.

```
df.info()
✓ 0.1s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14446 entries, 0 to 14445
Data columns (total 22 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   merchant      14446 non-null  object
 1   category      14446 non-null  object
 2   amt           14446 non-null  float64
 3   city          14446 non-null  object
 4   state         14446 non-null  object
 5   lat           14446 non-null  float64
 6   long          14446 non-null  float64
 7   city_pop      14446 non-null  int64
 8   job           14446 non-null  object
 9   dob           14446 non-null  object
 10  trans_num     14446 non-null  object
 11  merch_lat     14446 non-null  float64
 12  merch_long    14446 non-null  float64
 13  is_fraud      14446 non-null  int64
 14  trans_day     14446 non-null  int64
 15  trans_month   14446 non-null  int64
 16  trans_year    14446 non-null  int64
 17  trans_hour    14446 non-null  int64
 18  trans_minute  14446 non-null  int64
 19  dob_year      14446 non-null  int64
 20  age           14446 non-null  int64
 21  distance_km   14446 non-null  float64
dtypes: float64(6), int64(9), object(7)
memory usage: 2.4+ MB
```

```
df.info()
✓ 0.2s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14446 entries, 0 to 14445
Data columns (total 22 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   merchant      14446 non-null  int32
 1   category      14446 non-null  int32
 2   amt           14446 non-null  float64
 3   city          14446 non-null  int32
 4   state         14446 non-null  int32
 5   lat           14446 non-null  float64
 6   long          14446 non-null  float64
 7   city_pop      14446 non-null  int64
 8   job           14446 non-null  int32
 9   dob           14446 non-null  int32
 10  trans_num     14446 non-null  int32
 11  merch_lat     14446 non-null  float64
 12  merch_long    14446 non-null  float64
 13  is_fraud      14446 non-null  int64
 14  trans_day     14446 non-null  int64
 15  trans_month   14446 non-null  int64
 16  trans_year    14446 non-null  int64
 17  trans_hour    14446 non-null  int64
 18  trans_minute  14446 non-null  int64
 19  dob_year      14446 non-null  int64
 20  age           14446 non-null  int64
 21  distance_km   14446 non-null  float64
dtypes: float64(6), int32(7), int64(9)
memory usage: 2.0 MB
```

Before and after encoder, where object features are changed to integer.

	merchant	category	amt	city	state	lat	long	city_pop	job	dob	...	merch_long	is_fraud	trans_day	trans_month	trans_year	trans_hour	trans_minute	dob_year	age	distance_km
0	600	3	14.37	166	0	64.7556	-165.6723	145	1	47	...	-164.722603	1	4	1	2019	0	58	1939	80	109.2856
1	486	11	966.11	166	0	64.7556	-165.6723	145	1	47	...	-165.473127	1	4	1	2019	15	6	1939	80	79.8569
2	674	9	49.61	166	0	64.7556	-165.6723	145	1	47	...	-165.914542	1	4	1	2019	22	37	1939	80	66.8079
3	447	4	295.26	166	0	64.7556	-165.6723	145	1	47	...	-166.080207	1	4	1	2019	23	6	1939	80	39.6362
4	180	5	18.17	166	0	64.7556	-165.6723	145	1	47	...	-165.446843	1	4	1	2019	23	59	1939	80	77.6115
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
14441	267	12	122.00	8	9	45.8289	-118.4971	1302	38	100	...	-118.524214	0	22	1	2019	0	37	1976	43	68.2544
14442	452	9	9.07	48	9	43.7857	-124.1437	260	55	4	...	-124.995317	0	22	1	2019	0	41	1956	63	120.0584
14443	468	4	104.84	2	12	44.6873	-104.4414	110	2	81	...	-104.542117	0	22	1	2019	0	42	1973	46	94.9300
14444	53	12	268.16	166	0	64.7556	-165.6723	145	1	47	...	-165.898698	0	22	1	2019	0	48	1939	80	75.7446
14445	128	1	50.09	160	6	40.4815	-92.9951	3805	82	77	...	-92.224871	0	22	1	2019	0	55	1950	69	66.0260

String values converted into integer after applying encoder.

Logistic regression

The algorithm is a classifier for binary outcome that uses Sigmoid function (Vijay Kanade, 2022). The *LogisticRegression()* is imported under *linear_model* module; then, *.fit()* to train the model, *.predict()* for predicting the result of test data set. The *accuracy_score()* and *confusion_matrix()* are used for checking the precision of predicted values by comparing to actual values on test data set, where:

$$\text{Accuracy} = \frac{True\ positive\ +\ True\ Negative}{Total\ occurance} \times 100\%$$

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score
import time

#get the timestamp before inference in seconds
start_ts = time.time()

# Initialize and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

print(f"Accuracy: {(accuracy*100):.3f}%")
print(f"Area Under ROC Curve: {(auc*100):.3f}")
print(f"Confusion Matrix:\n{conf_matrix}")

#get the timestamp after the inference in second
end_ts = time.time()

# print the time difference in between start and end timestamps in seconds
print(f"Prediction Time [s]: {(end_ts-start_ts):.3f}")
```

✓ 0.1s

```
Accuracy: 92.293%
Area Under ROC Curve: 73.120
Confusion Matrix:
[[3738   43]
 [ 291  262]]
Prediction Time [s]: 0.119
```

Result:
Accuracy: 92.293%
AUC: 73.120
Confusion Matrix:

- True Positive: 3738
- False Negative: 43
- False Positive: 291
- True Negative: 262

Prediction time: 0.119s

Random Forest

Random Forest is an ensemble learning model that constructs many decision trees and combines their outputs. The Scikit-learn library's ensemble module is used to import *RandomForestClassifier*, which then initializes *n_estimators = 100*, means set up 100 decision trees; set random state of 42. Then, use *.fit()* to train the model with training data, to predict the outcome on the test set. For evaluation, the *accuracy_score()* is used to calculate the precision by percentage.

```python
from sklearn.ensemble import RandomForestClassifier
import time

#get the timestamp before inference in seconds
start_ts = time.time()

# Initialize and train the random forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test set
rf_pred = rf_model.predict(X_test)

# Evaluate the model
rf_accuracy = accuracy_score(y_test, rf_pred)
print(f"Random Forest Accuracy: {(rf_accuracy*100):.3f}%")

#get the timestamp after the inference in second
end_ts = time.time()

# print the time difference in between start and end timestamps in seconds
print(f"Prediction Time [s]: {(end_ts-start_ts):.3f}")
```

✓ 2.7s

```
Random Forest Accuracy: 99.838%
Prediction Time [s]: 2.717
```

Result:
Accuracy: 99.838%
Prediction time: 2.717s

3. Big Data analytics techniques

Precondition:

First, set up Apache Spark by importing the required library, *pyspark, sql* module, and *SparkSession* class. Then, initialize the Spark session and load *feature_fraud.csv.*

```python
from pyspark.sql import SparkSession

# Initialize the Spark session
spark = SparkSession.builder.appName("Spark_BigData").getOrCreate()

# Load data (CSV, Parquet, etc.)
data = spark.read.csv("feature_fraud.csv", header=True, inferSchema=True)
```

✓ 5.7s

Data reprocessing

There are many categorical variables which need to be encoded into numeric form to be sent into the classifier.

```
data.dtypes
✓ 0.0s

[('merchant', 'string'),
 ('category', 'string'),
 ('amt', 'double'),
 ('city', 'string'),
 ('state', 'string'),
 ('lat', 'double'),
 ('long', 'double'),
 ('city_pop', 'int'),
 ('job', 'string'),
 ('dob', 'string'),
 ('trans_num', 'string'),
 ('merch_lat', 'double'),
 ('merch_long', 'double'),
 ('is_fraud', 'int'),
 ('trans_day', 'int'),
 ('trans_month', 'int'),
 ('trans_year', 'int'),
 ('trans_hour', 'int'),
 ('trans_minute', 'int'),
 ('dob_year', 'int'),
 ('age', 'int'),
 ('distance_km', 'double')]
```

```
data.show()
✓ 0.4s                                                                                                      MagicPython

+--------------------+-------------+-------+--------+-----+-------+---------+--------+--------------------+----------+--------------------+---------+-
|            merchant|     category|    amt|    city|state|    lat|     long|city_pop|                 job|       dob|           trans_num|merch_lat|
+--------------------+-------------+-------+--------+-----+-------+---------+--------+--------------------+----------+--------------------+---------+-
|Stokes, Christian...|  grocery_net|  14.37|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|a3806e984cec6ac00...|65.654142|-
|        Predovic Inc| shopping_net| 966.11|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|a59185fe1b9ccf213...|65.468863|-
|    Wisozk and Sons|     misc_pos|  49.61|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|86ba3a888b42cd392...|65.347667|-
|      Murray-Smitham|  grocery_pos| 295.26|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|3a068fe1d856f0ece...|64.445035|-
|          Friesen Lt|health_fitness| 18.17|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|891cdd1191028759d...|65.447094|-
|Raynor, Reinger a...| gas_transport|  20.45|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|ef010a5f4f570d306...|64.088838|-
|      Heller-Langosh| gas_transport|  18.19|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|8e2d2fae5319d31c8...|63.917785|-
|       Padberg-Welch|  grocery_pos| 367.29|Browning|   MO| 40.029| -93.1607|     602|      Cytogeneticist|14-07-1954|5fbe827807ec9f557...|39.167065|
|   McGlynn-Heathcote|     misc_net| 768.15|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|fba83e0a3adb53025...|64.623325|-
|     Dooley-Thompson|     misc_net| 849.49|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|b87c92d4824758e70...|65.266065|-
|Gottlieb, Considi...| shopping_net|1177.79|Browning|   MO| 40.029| -93.1607|     602|      Cytogeneticist|14-07-1954|f1c51701d8b5dd6eb...|39.288305|
|Moen, Reinger and...|  grocery_pos| 307.09|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|755e4e8350ec4a3e9...|64.909145|-
|Hauck, Dietrich a...|    kids_pets|   4.58|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|8fa7880cf01e6adc9...|65.052892|-
|         Pouros-Haag| shopping_pos| 730.78|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|2396a5b8e277a4ce1...|65.233866|-
|         Goyette Inc| shopping_net| 1006.4|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|4d7e567247b6c4529...|65.220316|-
|Baumbach, Strosin...| shopping_pos| 830.72|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|773a3305db095657e...|65.710538|-
|    Pacocha-O'Reilly|  grocery_pos| 311.92|   Wales|   AK|64.7556|-165.6723|     145|Administrator, ed...|09-11-1939|191b3dcec7a6a4943...| 64.79501|-
|         Barrows PLC| shopping_pos| 762.93|Browning|   MO| 40.029| -93.1607|     602|      Cytogeneticist|14-07-1954|19b126ecf4c79997e...|40.205262|
|    Fisher-Schowalte| shopping_net| 855.88|Browning|   MO| 40.029| -93.1607|     602|      Cytogeneticist|14-07-1954|bbae703c3794b7738...|40.786018|
|  Gleason-Macejkovic| shopping_net| 909.29|Browning|   MO| 40.029| -93.1607|     602|      Cytogeneticist|14-07-1954|b98eb7183ee8a48e8...|40.977312|
+--------------------+-------------+-------+--------+-----+-------+---------+--------+--------------------+----------+--------------------+---------+-
```

Encode categorical variables with *StringIndexer*, which is equivalent to *Label Encoding*. However, errors were shown and unable to be resolved.

```python
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline

# Example data with multiple categorical columns
data = spark.read.csv("encoded_fraud.csv", header=True, inferSchema=True)
columns = ['merchant', 'category', 'city', 'state', 'job', 'dob', 'trans_num']

# Create a SparkSession
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("MultipleStringIndexer").getOrCreate()

# List of categorical columns to index
categorical_columns = ['merchant', 'category', 'city', 'state', 'job', 'dob', 'trans_num']

# Create a list to store StringIndexer stages
indexers = []

# Loop through each categorical column and create a StringIndexer for it
for col in categorical_columns:
    indexer = StringIndexer(inputCol=col, outputCol=col + "_index")
    indexers.append(indexer)

# Create a Pipeline to apply all indexers at once
pipeline = Pipeline(stages=indexers)

# Fit and transform the data
df_indexed = pipeline.fit(df).transform(df)

# Show the indexed result
df_indexed.show()
```

```
⊗ 0.4s
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
Cell In[50], line 27
     24 pipeline = Pipeline(stages=indexers)
     26 # Fit and transform the data
---> 27 df_indexed = pipeline.fit(df).transform(df)
     29 # Show the indexed result
     30 df_indexed.show()

File [redacted]\Python312\Lib\site-packages\pyspark\ml\base.py:205, in Estimator.fit(self, dataset, params)
    203         return self.copy(params)._fit(dataset)
    204     else:
--> 205         return self._fit(dataset)
    206 else:
    207     raise TypeError(
    208         "Params must be either a param map or a list/tuple of param maps, "
    209         "but got %s." % type(params)
    210     )

File [redacted]\Python\Python312\Lib\site-packages\pyspark\ml\pipeline.py:134, in Pipeline._fit(self, dataset)
    132     dataset = stage.transform(dataset)
    133 else:  # must be an Estimator
--> 134     model = stage.fit(dataset)
    135     transformers.append(model)
    136     if i < indexOfLastEstimator:
...
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:182)
        at py4j.ClientServerConnection.run(ClientServerConnection.java:106)
        at java.lang.Thread.run(Thread.java:748)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Hence, the encoded data set is label encoded in Python and saved as CSV before passing to PySpark. The categorical values are now numeric.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('feature_fraud.csv')

# get object columns
object_col=df.select_dtypes(include='object').columns.to_list()

# Apply LabelEncoder
encoder = LabelEncoder()
for i in object_col:
    df[i] = encoder.fit_transform(df[i])

df.to_csv('encoded_fraud.csv')
```
✓ 0.2s

```python
from pyspark.sql import SparkSession

# Initialize the Spark session
spark = SparkSession.builder.appName("Spark_BigData").getOrCreate()

# Load data (CSV, Parquet, etc.)
data = spark.read.csv("encoded_fraud.csv", header=True, inferSchema=True)
```
✓ 0.2s MagicPython

```python
data.show()
```
✓ 0.1s MagicPython

_c0	merchant	category	amt	city	state	lat	long	city_pop	job	dob	trans_num	merch_lat	merch_long	is_fraud	trans_day	trans_month	trans_year
0	600	3	14.37	166	0	64.7556	-165.6723	145	1	47	9134	65.654142	-164.722603	1	4	1	2019
1	486	11	966.11	166	0	64.7556	-165.6723	145	1	47	9247	65.468863	-165.473127	1	4	1	2019
2	674	9	49.61	166	0	64.7556	-165.6723	145	1	47	7511	65.347667	-165.914542	1	4	1	2019
3	447	4	295.26	166	0	64.7556	-165.6723	145	1	47	3196	64.445035	-166.080207	1	4	1	2019
4	180	5	18.17	166	0	64.7556	-165.6723	145	1	47	7643	65.447094	-165.446843	1	4	1	2019
5	498	2	20.45	166	0	64.7556	-165.6723	145	1	47	13427	64.088838	-165.104078	1	5	1	2019
6	243	2	18.19	166	0	64.7556	-165.6723	145	1	47	7928	63.917785	-165.827621	1	5	1	2019
7	472	4	367.29	18	6	40.029	-93.1607	602	37	71	5296	39.167065	-93.705245	1	5	1	2019
8	416	8	768.15	166	0	64.7556	-165.6723	145	1	47	14134	64.623325	-166.403973	1	5	1	2019
9	142	8	849.49	166	0	64.7556	-165.6723	145	1	47	10354	65.266065	-164.865352	1	5	1	2019
10	198	11	1177.79	18	6	40.029	-93.1607	602	37	71	13568	39.288305	-92.476947	1	5	1	2019
11	428	4	307.09	166	0	64.7556	-165.6723	145	1	47	6538	64.909145	-164.712087	1	5	1	2019
12	234	7	4.58	166	0	64.7556	-165.6723	145	1	47	7998	65.052892	-166.067029	1	5	1	2019
13	484	12	730.78	166	0	64.7556	-165.6723	145	1	47	1963	65.233866	-166.550779	1	5	1	2019
14	200	11	1006.4	166	0	64.7556	-165.6723	145	1	47	4277	65.220316	-165.005725	1	5	1	2019
15	33	12	830.72	166	0	64.7556	-165.6723	145	1	47	6641	65.710538	-165.986117	1	5	1	2019
16	468	4	311.92	166	0	64.7556	-165.6723	145	1	47	1385	64.79501	-165.670735	1	5	1	2019
17	22	12	762.93	18	6	40.029	-93.1607	602	37	71	1414	40.205262	-93.499211	1	5	1	2019

Splitting data into training and testing set

1. Required *PySpark* library modules and class are imported, were
    a) *VectorAssembler*: To assemble the features into a single vector column, this is the format of features needed by machine learning models in PySpark.
    b) *Pipeline*: Chain stages of data transformation, so that it is convenient for user to organize the process of data preparation into steps for sequential application.
2. Feature selection: Include the columns needed as features and exclude the target variable. In this case, the "is_fraud" is excluded in the feature list.
3. *VectorAssembler*: Assemble all the data points of feature columns, convert them into a list under a new column named "features", which would be used as input variable.
4. *Pipeline* set up and fitting data: created under one stage, *VectorAssembler.* Useful in keeping data transformation process modular and reusable. Fitting data applied transformation in pipeline to data set in vector format.
5. Split data: Data is splited to 70% train_data, and 30% test_data, where training set is used for model training, while testing set for model evaluation.

```python
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline

# Feature selection: Select features for prediction (exclude target column)
feature_columns = [col for col in data.columns if col != "is_fraud"]

# Assemble all features into a single vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Set up the pipeline
pipeline = Pipeline(stages=[assembler])
pipeline_model = pipeline.fit(data)
processed_data = pipeline_model.transform(data)

# Split the data into training and test datasets: 70% train, 30%
train_data, test_data = processed_data.randomSplit([0.7, 0.3], seed=50)
```
✓ 0.0s

```python
train_data.show()
```
✓ 0.3s MagicPython

job	dob	trans_num	merch_lat	merch_long	is_fraud	trans_day	trans_month	trans_year	trans_hour	trans_minute	dob_year	age	distance_km	features
1	47	9134	65.654142	-164.722603	1	4	1	2019	0	58	1939	80	109.2856	[0.0,600.0,3.0,14...
1	47	9247	65.468863	-165.473127	1	4	1	2019	15	6	1939	80	79.8569	[1.0,486.0,11.0,9...
1	47	7511	65.347667	-165.914542	1	4	1	2019	22	37	1939	80	66.8079	[2.0,674.0,9.0,49...
1	47	3196	64.445035	-166.080207	1	4	1	2019	23	6	1939	80	39.6362	[3.0,447.0,4.0,29...
1	47	7643	65.447094	-165.446843	1	4	1	2019	23	59	1939	80	77.6115	[4.0,180.0,5.0,18...
1	47	13427	64.088838	-165.104078	1	5	1	2019	3	15	1939	80	78.9988	[5.0,498.0,2.0,20...
1	47	7928	63.917785	-165.827621	1	5	1	2019	3	21	1939	80	93.4605	[6.0,243.0,2.0,18...
37	71	5296	39.167065	-93.705245	1	5	1	2019	11	31	1954	65	106.5952	[7.0,472.0,4.0,36...
1	47	14134	64.623325	-166.403973	1	5	1	2019	18	3	1939	80	37.7643	[8.0,416.0,8.0,76...
37	71	13568	39.288305	-92.476947	1	5	1	2019	22	5	1954	65	101.0411	[10.0,198.0,11.0,...
1	47	6538	64.909145	-164.712087	1	5	1	2019	22	12	1939	80	48.5095	[11.0,428.0,4.0,3...
1	47	7998	65.052892	-166.067029	1	5	1	2019	22	18	1939	80	37.9385	[12.0,234.0,7.0,4...
1	47	1963	65.233866	-166.550779	1	5	1	2019	22	32	1939	80	67.327	[13.0,484.0,12.0,...
1	47	6641	65.710538	-165.986117	1	5	1	2019	22	38	1939	80	107.1855	[15.0,33.0,12.0,8...
1	47	1385	64.79501	-165.670735	1	5	1	2019	23	17	1939	80	4.3828	[16.0,468.0,4.0,3...
37	71	10556	40.786018	-93.301092	1	6	1	2019	18	39	1954	65	85.0117	[18.0,175.0,11.0,...
37	71	10420	40.977312	-93.55098	1	6	1	2019	23	33	1954	65	110.4896	[19.0,193.0,11.0,...
138	118	3710	40.287778	-98.570998	1	12	1	2019	1	46	1974	45	86.4786	[20.0,358.0,2.0,7...
138	118	11101	40.834742	-99.895337	1	12	1	2019	15	36	1974	45	85.8724	[21.0,310.0,12.0,...
138	118	4942	40.341318	-99.045651	1	12	1	2019	23	9	1974	45	76.7665	[22.0,342.0,8.0,7...

Data Mining with Random Forest

- *RandomForestClassifier* class under *ml.classification* module and *MulticlassClassificationEvaluator* class under *ml.evaluation* module are are imported from *PySpark* library.
- *RandomForestClassifier* is used to train the model, where feacture vector and "is_fraud" columns, and set 100 trees as parameters.
- The evaluator is then applied to calculate the accuracy.

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
import time

#get the timestamp before inference in seconds
start_ts = time.time()

# Train the Random Forest Model on the Training Data
rf = RandomForestClassifier(featuresCol="features", labelCol="is_fraud", numTrees=100)

# Fit the model on the training data
rf_model = rf.fit(train_data)

# Make Predictions on the Test Data
predictions = rf_model.transform(test_data)

# Show some predictions
predictions.select("features", "is_fraud", "prediction").show(10)

# Step 8: Evaluate the Model
evaluator = MulticlassClassificationEvaluator(labelCol="is_fraud", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)

print(f"Accuracy: {(accuracy*100):.3f}")

#get the timestamp after the inference in second
end_ts = time.time()

# print the time difference in between start and end timestamps in seconds
print(f"Prediction Time [s]: {(end_ts-start_ts):.3f}")
```
✓ 2.8s

Result for the first 10 rows are shown, along with the accuracy and run time.

```
+--------------------+--------+----------+
|            features|is_fraud|prediction|
+--------------------+--------+----------+
|[9.0,142.0,8.0,84...|       1|       1.0|
|[14.0,200.0,11.0,...|       1|       1.0|
|[17.0,22.0,12.0,7...|       1|       1.0|
|[23.0,311.0,11.0,...|       1|       1.0|
|[29.0,531.0,12.0,...|       1|       1.0|
|[31.0,376.0,0.0,5...|       1|       1.0|
|[36.0,474.0,2.0,1...|       1|       1.0|
|[37.0,459.0,4.0,3...|       1|       1.0|
|[38.0,88.0,3.0,16...|       1|       1.0|
|[42.0,285.0,2.0,8...|       1|       1.0|
+--------------------+--------+----------+
only showing top 10 rows

Accuracy: 99.703
Prediction Time [s]: 2.865
```

Accuracy: 99.703%
Prediction time: 2.865s

Big data Machine Learning

The two machine learning techniques used for big data have overlapping classes and functions, the similarities and differences are listed:

	Logistic Regression	SVM
Classifier imported for model training	*LogisticRegression*	*LinearSVC*
Evaluation classes	• *BinaryClassificationEvaluaton* • *MulticlassClassificationEvaluator* • *col*	• *BinaryClassificationEvaluator* • *col*
Evaluation	• Confusion matrix • Accuracy	• AUC • Confusion matrix • Accuracy

Screen-shots and result for:

Logistic regression

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
from pyspark.sql.functions import col
import time

#get the timestamp before inference in seconds
start_ts = time.time()

# Train the Logistic Regression Model
lr = LogisticRegression(featuresCol="features", labelCol="is_fraud", maxIter=10, regParam=0.1)

# Fit the model on the training data
lr_model = lr.fit(train_data)

# Step 6: Make Predictions on the Test Data
predictions = lr_model.transform(test_data)

# Show some predictions
predictions.select("features", "is_fraud", "prediction").show(5)

# Evaluate the Model
evaluator_bin = BinaryClassificationEvaluator(labelCol="is_fraud", rawPredictionCol="prediction", metricName="areaUnderROC")
evaluator_class = MulticlassClassificationEvaluator(labelCol="is_fraud", predictionCol="prediction", metricName="accuracy")

#Confusion Matrix Calculation
# Create a confusion matrix by counting the number of True Positives, False Positives,
# True Negatives, and False Negatives
tp = predictions.filter((col("prediction") == 1) & (col("is_fraud") == 1)).count()
tn = predictions.filter((col("prediction") == 0) & (col("is_fraud") == 0)).count()
fp = predictions.filter((col("prediction") == 1) & (col("is_fraud") == 0)).count()
fn = predictions.filter((col("prediction") == 0) & (col("is_fraud") == 1)).count()
```

```python
    # Print the confusion matrix
    print("Confusion Matrix:")
    print(f"True Positive (TP): {tp}")
    print(f"False Negative (FN): {fn}")
    print(f"False Positive (FP): {fp}")
    print(f"True Negative (TN): {tn}")

    roc_auc = evaluator_bin.evaluate(predictions)
    accuracy = evaluator_class.evaluate(predictions)

    print(f"Area Under ROC Curve: {(roc_auc*100):.3f}")
    print(f"Accuracy: {(accuracy*100):.3f}")

    # get the timestamp after the inference in second
    end_ts = time.time()

    # print the time difference in between start and end timestamps in seconds
    print(f"Prediction Time [s]: {(end_ts-start_ts):.3f}")
```
✓ 2.3s

```
+--------------------+--------+----------+
|            features|is_fraud|prediction|
+--------------------+--------+----------+
|[9.0,142.0,8.0,84...|       1|       1.0|
|[14.0,200.0,11.0,...|       1|       1.0|
|[17.0,22.0,12.0,7...|       1|       1.0|
|[23.0,311.0,11.0,...|       1|       1.0|
|[29.0,531.0,12.0,...|       1|       1.0|
+--------------------+--------+----------+
only showing top 5 rows

Confusion Matrix:
True Positive (TP): 274
False Negative (FN): 285
False Positive (FP): 10
True Negative (TN): 3813
Area Under ROC Curve: 74.377
Accuracy: 93.268
Prediction Time [s]: 2.312
```

Accuracy: 93.268%
AUC: 74.377
Confusion Matrix:
- True Positive: 274
- False Negative: 285
- False Positive: 10
- True Negative: 3813

Prediction time: 2.312s

SVM

```python
from pyspark.ml.classification import LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
from pyspark.sql.functions import col
import time

#get the timestamp before inference in seconds
start_ts = time.time()

# Train the LinearSVC (Support Vector Machine) Model
svm = LinearSVC(featuresCol="features", labelCol="is_fraud")

# Fit the model on the training data
svm_model = svm.fit(train_data)

# Make Predictions on the Test Data
predictions = svm_model.transform(test_data)

# Show some predictions
predictions.select("features", "is_fraud", "prediction").show(5)

# Evaluate the Model
# Using BinaryClassificationEvaluator to evaluate the model performance
evaluator = BinaryClassificationEvaluator(labelCol="is_fraud", rawPredictionCol="prediction", metricName="areaUnderROC")
roc_auc = evaluator.evaluate(predictions)

print(f"Area Under ROC Curve: {(roc_auc*100):.3f}")
```

```python
# Confusion Matrix Calculation
# Create a confusion matrix by counting the number of True Positives, False Positives,
# True Negatives, and False Negatives
tp = predictions.filter((col("prediction") == 1) & (col("is_fraud") == 1)).count()
tn = predictions.filter((col("prediction") == 0) & (col("is_fraud") == 0)).count()
fp = predictions.filter((col("prediction") == 1) & (col("is_fraud") == 0)).count()
fn = predictions.filter((col("prediction") == 0) & (col("is_fraud") == 1)).count()

# Print the confusion matrix
print("Confusion Matrix:")
print(f"True Positive (TP): {tp}")
print(f"True Negative (TN): {tn}")
print(f"False Positive (FP): {fp}")
print(f"False Negative (FN): {fn}")

# Optionally, you can calculate accuracy, precision, recall, and F1 score from the confusion matrix
accuracy = (tp + tn) / (tp + tn + fp + fn)

print(f"Accuracy: {(accuracy*100):.3f}%")

# get the timestamp after the inference in second
end_ts = time.time()

# print the time difference in between start and end timestamps in seconds
print(f"Prediction Time [s]: {(end_ts-start_ts):.3f}")
```

8.6s

```
+--------------------+--------+----------+
|            features|is_fraud|prediction|
+--------------------+--------+----------+
|[0.0,600.0,3.0,14...|       1|       1.0|
|[1.0,486.0,11.0,9...|       1|       1.0|
|[4.0,180.0,5.0,18...|       1|       1.0|
|[21.0,310.0,12.0,...|       1|       1.0|
|[24.0,309.0,4.0,3...|       1|       1.0|
+--------------------+--------+----------+
only showing top 5 rows

Area Under ROC Curve: 97.785
Confusion Matrix:
True Positive (TP): 509
True Negative (TN): 3765
False Positive (FP): 4
False Negative (FN): 23
Accuracy: 99.372%
Prediction Time [s]: 8.602
```


Accuracy: 99.372%

Area Under ROC Curve:

97.785

Confusion Matrix:

- True Positive (TP): 509
- True Negative (TN): 3765
- False Positive (FP): 4
- False Negative (FN): 23

Prediction Time [s]: 8.602

4. Comparison

There is no significant difference in accuracy between conventional and big data analytics. The conventional techniques have shorter processing time compared to PySpark in general. For Logistic regression, the accuracy is slightly higher by using PySpark, however there is a big gap in the prediction time, where conventional technique is much faster than PySpark.

ML Techniques	Conventional	Big Data
Random Forest	Accuracy: 99.838% Prediction time: 2.717s	Accuracy: 99.703% Prediction time: 2.865s
Logistic Regression	Accuracy: 92.293% AUC: 73.120 Confusion Matrix: • True Positive: 3738 • False Negative: 43 • False Positive: 291 • True Negative: 262 Prediction time: 0.119s	Accuracy: 93.268% AUC: 74.377 Confusion Matrix: • True Positive: 274 • False Negative: 285 • False Positive: 10 • True Negative: 3813 Prediction time: 2.312s
SVM	-	Accuracy: 99.372% Area Under ROC Curve: 97.785 Confusion Matrix: • True Positive (TP): 509 • True Negative (TN): 3765 • False Positive (FP): 4 • False Negative (FN): 23 Prediction Time [s]: 8.602s

5. Reflection

Big data plays a key role in society and business today as it has enormous potential in shaping the future by helping in decision-making and supporting technical processes. In the case study, big data is vital in running the process of payment fraud detection through specialized big data tools and techniques, which can support its distinctive characteristics: volume, velocity, veracity, value, and variety of data.

Apache Spark is one of the prominent open-source big data tools that support multiple languages, and the Python version of Spark, PySpark, is powerful in terms of its features. PySpark can complete tasks that conventional analytics cannot. Apache Spark provides companies with the ability to optimize their operations by processing large-scale data efficiently and quickly, making it an indispensable tool for both industry professionals and learners who want to gain exposure to big data analytics.

(1690 words)

Reference

Duggal, N. (2022, January 13). *Top 7 Benefits of Big Data and Analytics and Reasons to Consider It*. Simplilearn.com; Simplilearn. https://www.simplilearn.com/benefits-of-big-data-and-analytics-article

Common Payment Fraud Scams: How to Stay Safe. (2023). DataVisor. https://www.datavisor.com/wiki/payment-fraud/

Bence Jendruszak. (2017, February 21). *Top 5 Fraud Patterns Risk Managers Should Look Out For*. SEON. https://seon.io/resources/top-five-fraudster-scam-patterns/

Team, D. (2016, September 19). *Apache Spark vs Hadoop MapReduce - Feature Wise Comparison [Infographic] - DataFlair*. DataFlair. https://data-flair.training/blogs/spark-vs-hadoop-mapreduce/

Real Time Credit Card Fraud Detection with Apache Spark and Event Streaming. (2020). Hpe.com. https://developer.hpe.com/blog/real-time-credit-card-fraud-detection-with-apache-spark-and-event-stream/#:~:text=Feature%20engineering%20to%20transform%20historical,learning%20algorithm%20and%20repeat%20tests.

https://www.facebook.com/avengapoland. (2022, November 28). *Machine Learning In Fraud Detection: An In-Depth Analysis – Avenga*. Avenga. https://www.avenga.com/magazine/fraud-detection-machine-learning/#:~:text=SVMs%2C%20advanced%20yet%20simple%20in,financial%20and%20fraud%20detection%20systems.

IBM. (2021, October 20). *Random Forest*. Ibm.com. https://www.ibm.com/topics/random-forest

You got it or not? How to tell if Big Data is something you need to worry about in your application | Abas. (2022). Abas-Erp.com. https://abas-erp.com/en/blog/you-got-it-or-not-how-tell-if-big-data-something-you-need-worry-about-your-application

Aletaha, D., & Huizinga, T. W. J. (2009). The use of data from early arthritis clinics for clinical research. *Best Practice & Research Clinical Rheumatology*, *23*(1), 117–123. https://doi.org/10.1016/j.berh.2008.11.008

IBM. (2023, December 12). *Support Vector Machine*. Ibm.com. https://www.ibm.com/topics/support-vector-machine

LogisticRegression. (2014). Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression