Final Project Report: Financial Capability Predictor

BUS ADM 742: Big Data in Business

University of Wisconsin Milwaukee

Snehal D. Sase

Nikhil Sunku

20-December-2021

BUS ADM 742: Big Data in Business

Introduction

Financial Capability Predictor is a system which helps us to predict the capability of a candidate to pay off the loan amount.

Dataset

We have used bank loan dataset which has 307511 rows and 122 columns So each column would be occupying a lot of space. Figure bellow shows top five rows of the dataset.

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREI
0	100002	1	Cash loans	M	N	Y	0	202500.0	40659
1	100003	0	Cash loans	F	N	N	0	270000.0	129350
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	13500
3	100006	0	Cash loans	F	N	Y	0	135000.0	31268
4	100007	0	Cash loans	M	N	Y	0	121500.0	51300

5 rows × 122 columns

Data Processing using SPARK

This document briefly describes how we used SPARK for data processing.

SPAKR Cluster

We created a SPARK cluster local on our machine by creating a multi-threaded application that had 4 threads. 1 thread was for SPARK driver and other 3 threads were for SPARK executors. These three executors performed data processing TASK in parallel which replicated a SPARK cluster like structure on our local machine.

```
conf = SparkConf().setMaster("local[4]").setAppName("BigDataProcessing")
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

BUS ADM 742: Big Data in Business

Data Loading

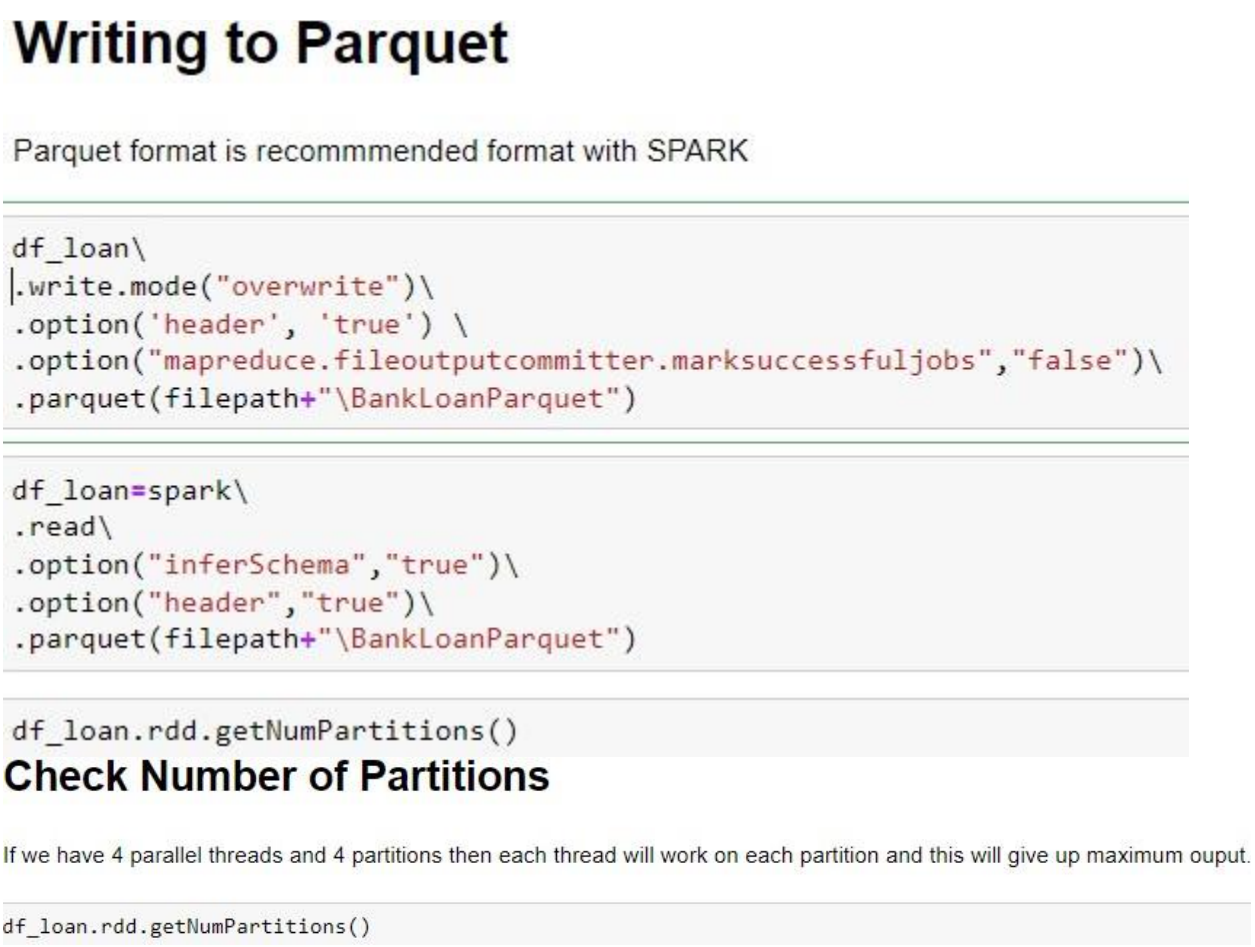
We converted a CSV file to a PARQUET file with 4 numbers of partitions. The purpose of this activity was that the recommended file format for SPARK is PARQUET because using this file format we can enhance the speed of SPARK data load.

```
filepath=r"D:\UpWork\PySpark"

df_loan = spark\
.read\
.option("inferSchema","true")\
.option("header","true")\
.csv(filepath+"\BankLoan.csv")
```

Partitions

We partitioned our file into four equal partitions. The reason for this activity was that our SPARK application is multi-threaded and has 4 threads. When we partitioned our file into 4 equal partitions then in our multi-threaded SPARK application will assign each thread to each file partition and our data processing task will execute in parallel utilizing the SPARK ability to process data in parallel.



Exploratory Data Analysis

Performed EDA on our data to understand our data in order to build our data model, we done following analysis on our data in EDA.

1. Found out the Dimension of our data.
2. Total number of rows and columns in our data.
3. Find out categorical and non-categorical columns.
4. Check that if the type of each column is correct or not by printing the schema of dataframe.
5. Find out unique values in each category in our categorical features.
6. Wrote a SPARK UDF to plot the data.
7. Find out different loan types in our data.
8. Find out different attributes of the person that is applying for loan.

Data Processing

For data processing we corrected issues in our data. Following are the activities that we performed in-order to make our data correct.

1. Correct the in-correct values by replace them we most frequent while. For example, in gender column we found a value which was XNA ad we replaced it most frequent value in gender column.

```
df_loan.select("CODE_GENDER").groupBy("CODE_GENDER").agg(count("CODE_GENDER").alias('count')).show()
```

```
+-----------+------+
|CODE_GENDER| count|
+-----------+------+
|          F|202448|
|          M|105059|
|        XNA|     4|
+-----------+------+
```

```
from pyspark.sql.functions import regexp_replace
df_loan = df_loan.withColumn('CODE_GENDER', regexp_replace('CODE_GENDER', 'XNA', 'F'))
df_loan.select("CODE_GENDER").groupBy("CODE_GENDER").agg(count("CODE_GENDER").alias('count')).show()
```

```
+-----------+------+
|CODE_GENDER| count|
+-----------+------+
|          F|202452|
|          M|105059|
```

2. In categorical features we represented the NULL values with the more understandable value the helped us in our analysis.
3. Categorical features occupies a lot of space which makes our data processing much difficult we encoded them into numerical representation that helped us to free up some space in RAM and helped us to perform in memory computation with more data which increased our speed.
4. After encoding we had a lot of columns so we dropped categorical columns and keep their numerical representation.

After applying data reduction techniques we have now 88 columns

```
print(len(df_loan_new.columns))
print(len(sparkloan_DF.columns))
```

```
122
88
```

5. Then we furthermore reduced he unnecessary columns by finding out the columns that were highly correlated, because the data points that are highly correlated give us same time of analysis, this make us process unnecessary data which is of no use.

```
Your selected dataframe has 88 columns.
There are 28 columns that have missing values.
```

	Missing Values	% of Total Values
COMMONAREA_AVG	214865	69.9
NONLIVINGAPARTMENTS_AVG	213514	69.4
FLOORSMIN_AVG	208642	67.8
YEARS_BUILD_AVG	204488	66.5
OWN_CAR_AGE	202929	66.0
LANDAREA_AVG	182590	59.4
BASEMENTAREA_AVG	179943	58.5

```
print("Initiak columns count:"+str(len(df_loan.columns)))
print("Final columns count:"+str(len(SparkDF.columns)))
```

```
Initiak columns count:122
Final columns count:56
```

6. Then we imputed the missing values in our numerical features by taking media for this we wrote a SPARK UDF.

```
ip_cols=[c for c in SparkDF.columns if c not in 'TARGET']
```

```
vector_assembler = VectorAssembler(inputCols=ip_cols,outputCol="features")
df_temp = vector_assembler.transform(SparkDF)
df_temp.select('features').show(3)
```

```
+--------------------+
|            features|
+--------------------+
|[0.26294859274717...|
|[0.62224577525550...|
|[0.55591208339044...|
+--------------------+
only showing top 3 rows
```

7. Then we selected necessary data using extra tree classifier algorithm.

Conclusion

After performing all the data processing in SAPRK we need to validate that if our data is ready for any kind of analysis after pre-processing it. We used three machine learning models

Decision Tree, Naïve Bayes and SM. We found out that all these three models were giving accuracy more than 85%. Thus we can conclude that using these data pre-processing techniques we can build a reliable and scalable models.