|  |
| --- |
|  |
| Chase your right customers on right time |
| AR Overdue Predictive solution using Apache Spark (Big data) |
|  |
| **Vinnas Kuttery** |
| **4/12/2017** |

|  |
| --- |
| This articles aim is to provide a big data solution for predicting AR Overdue using Apache Spark (especially for a large organization) |

Contents

[**Overview 2**](#_Toc479798741)

[**How does analytics help here? 3**](#_Toc479798742)

[**Suggestive predictive AR model 3**](#_Toc479798743)

[**Why Apache Spark? 3**](#_Toc479798744)

[**About Data 4**](#_Toc479798745)

[**Source Code 4**](#_Toc479798746)

[**Load Data into Hadoop 4**](#_Toc479798747)

[**Import data into Spark and create pyspark dataframe 4**](#_Toc479798748)

[**Explore data and derive key features 5**](#_Toc479798749)

[**Explore and Generate relevant bins 6**](#_Toc479798750)

[**Conclusion 6**](#_Toc479798751)

# Overview

**Accounts receivable** is a legally enforceable claim for payment held by a business for goods supplied and/or services rendered that customers/clients have ordered but not paid for. These are generally in the form of invoices raised by a business and delivered to the customer for payment within an agreed time frame.

Irrespective of the size of the business, it is important to maintain an optimal account receivables to be financially strong. Hencebusinesses are required to implement an efficient monitoring and collection strategies for managing the AR.It is also an opportunity cost for the business if the receivables are not recovered before they become overdue. AR management hence becomes quint essential recipe for a successful organization and sometime is a differentiator.

In most of the organization, managing AR is done by the internal collections team and their aim is to keep it at a certain level of the sales. However, to achieve the same most business rely on traditional wisdom rather thansupported by data and insights. This is especially difficult for a large organization where there are wide variety of customers, spread over different geographies, with varied credit profile and with limited organization resources is not economical to chase all of them.

Moreover, organizations tends to be more reactive than being proactive; they spend all of their time and efforts in managing AR which has already become overdue. While a smarter organization look at predicting customer riskiness and use it for managing AR which has high probability of becoming overdue. This way the organization can effectively control AR before it becomes overdue and hence maintain the targeted AR levels and minimize opportunity cost. Modern organizations heavily depend on analytics to accomplish this.

## How does analytics help here?

In order to be proactive, the first step should be identifying the right customer invoice which is going to be overdue in future. It makes more sense if we monitor such customer invoices and take appropriate actions than chasing the customers after the AR is overdue.In fact, there are so many challenges which business should also carefully consider before implementing the such strategy. Listing few below,

* How to identify the right invoice/customer?
* What if the business concentrates on chasing low risk/low value customers?
* Is it the right time to track that customer?
* What if I chase a valuable customer continuously? Will it affect my customer relationship?

Here is where of data insights and analytics come in handy. In this scenario predictive analytics solution will help the business by suggestingthe right customer invoice which has a high probability of getting overdue.

This articles aim is to provide a big data solution for predicting AR Overdue using Apache Spark (especially for a large organization)

### Suggestive predictive AR model

Below is the sample formula which can be a accurate solution for this prediction.

|  |
| --- |
| Yt->Customer Risk Scoret + Limit Utilizationt + Overdue propensityt + Payment termt+ Pricingt+ Relationship Aget+ ……. |

### Why Apache Spark?

we all knows that programming languages like SAS,R or Python will give us the enough flexibility to develop predictive solutions, but its bit difficult or tricky when it comes to handling huge data. So in most of the Organization, they will be compromising and will use a sample data during the model development.

Since any model is the depiction of the data on which it is built, the accuracy hence can take a hit if we scale up to the large population. Hence, it is better to consider the whole of historicaldata rather than taking a portion of it. Moreover, when we talk about any data driven solution in a big organization, we must consider scalability and stable performance.

In that perspective, [apache spark](http://spark.apache.org/) is one of best predictive analytics platform which is on big data. This project has been implemented in apache spark using zeppelin platform.

# About Data

Below is the metadata of the sample which we have considered for the analysis

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Description** | **Data Type** |
| Document Number | Unique Invoice Number | Varchar |
| Payment Date | Date on which the amount is expected to pay | Date |
| Closed Date | Date on which the amount is paid for closed invoices | Date |
| Net due date |  | Date |
| Document Date | Date on which the invoice is registered | Date |
| Status | Status of the invoice (Open/Closed) | Varchar |
| Customer ID | Customer Unique ID | Varchar |
| Start Date | Date on which the customer is registered | Date |
| state | State of the customer | Varchar |
| city | City of the customer | Varchar |
| region | Region of the customer | Varchar |
| Credit Limit | Credit limit of the customer | Double |
| Payment Term | Number of days allowed to clear payment | Int |
| Amount | Amount on the Invoice | Double |

*Note: Sample data can be found @ https://www.dropbox.com/s/0u5ackm48zry9ex/Final\_withoutHeader.csv*

# Source Code

## Load Data into Hadoop

|  |
| --- |
| %sh  #remove existing file in local disc and download new data from the specified link  rm -f Final\_withoutHeader.csv  wget https://www.dropbox.com/s/0u5ackm48zry9ex/Final\_withoutHeader.csv  #remove existing copies of dataset from HDFS  hadoop fs -rm -r -f /tmp/invoiceDetails  hadoop fs -mkdir /tmp/invoiceDetails  #put data into HDFS  hadoop fs -put /home/zeppelin/Final\_withoutHeader.csv /tmp/invoiceDetails/  hadoop fs -ls -h /tmp/invoiceDetails/ |

## Import data into Spark and create pyspark dataframe

|  |
| --- |
| %pyspark  from pyspark.sql.types import StringType,DoubleType,IntegerType  from pyspark import SQLContext  from pyspark.sql.types import StructType  from datetime import datetime  from pyspark.sql.functions import col,udf, unix\_timestamp  from pyspark.sql.types import DateType  #Create Data frame from the RDD which created from the loaded csv Data  sqlContext = SQLContext(sc)  invoiceDetailRDD = sc.textFile("/tmp/invoiceDetails/Final\_withoutHeader.csv").map(lambda line: line.split(","))  columnNames=['DocumentNumber','PaymentDate','ClosedDate','NetDueDate','DocumentDate','Status','customerID','startDate','state','city','region','CreditLimit','PaymentTerm','Amount']  invoiceDetailDF = sqlc.createDataFrame(invoiceDetailRDD,columnNames)  #Functions to convert the data types.  def string\_to\_float(x):  return float(x)  convertToFloat = udf (string\_to\_float, StringType())  convertToDate = udf (lambda x: datetime.strptime(x, '%M/%d/%Y'), DateType())  #Convert data type of specified columns into date type  invoiceDetailDF = invoiceDetailDF.withColumn('PaymentDate', convertToDate(col('PaymentDate')))  invoiceDetailDF = invoiceDetailDF.withColumn('ClosedDate', convertToDate(col('ClosedDate')))  invoiceDetailDF = invoiceDetailDF.withColumn('NetDueDate', convertToDate(col('NetDueDate')))  invoiceDetailDF = invoiceDetailDF.withColumn('DocumentDate', convertToDate(col('DocumentDate')))  invoiceDetailDF = invoiceDetailDF.withColumn('startDate', convertToDate(col('startDate')))  invoiceDetailDF = invoiceDetailDF.withColumn('Amount', convertToFloat(col('Amount')))  #Convert data type of specified columns into double/integer type  invoiceDetailDF = invoiceDetailDF.withColumn("Amount", invoiceDetailDF["Amount"].cast(DoubleType()))  invoiceDetailDF = invoiceDetailDF.withColumn("CreditLimit", invoiceDetailDF["CreditLimit"].cast(DoubleType()))  invoiceDetailDF = invoiceDetailDF.withColumn("PaymentTerm", invoiceDetailDF["PaymentTerm"].cast(IntegerType()))  #register the dataframe as spark tables.  sqlc.registerDataFrameAsTable(invoiceDetailDF, "invoiceDetail")  #Create permanant tables from registered table for remaining analysis  sqlc.sql("drop table invoiceDetails")  sqlc.sql("create table invoiceDetails as select \* from invoiceDetail"); |

## Explore data and derive key features

|  |
| --- |
| %pyspark  from pyspark.sql import functions as F  from pyspark.sql.functions import datediff  from pyspark.sql import HiveContext  sqlContext = HiveContext(sc)  #Load Data from saved table (from previous session), and also generate extra variables by keeping 2017-01-01 as as of analysis data.  invoicedetailsDF = sqlContext.sql('select documentnumber,documentdate,status,region,startdate,creditlimit,paymentterm,amount,date("2017-01-01") as asOfDate,case when status="Open" then 0 else 1 end as Overdue,case when status="Clossed" then PaymentDate else date("2017-01-01") end as PaymentDate from invoicedetails')  #select only variable for analysis  tobeAnalysedDF=invoicedetailsDF.select("documentnumber","documentdate","PaymentDate","status","region","creditlimit","paymentterm","amount",  datediff( invoicedetailsDF.PaymentDate,invoicedetailsDF.documentdate).alias("invoiceAge"),  datediff( invoicedetailsDF.asOfDate,invoicedetailsDF.startdate).alias("relationshipAge"),  "Overdue")  #Save the dataframe a Spark Table  sqlc.registerDataFrameAsTable(tobeAnalysedDF, "tobeAnalysedDF1")  sqlc.sql("drop table tobeAnalysedDF");  sqlc.sql("create table tobeAnalysedDF as select \* from tobeAnalysedDF1"); |

## Explore and Generate relevant bins

|  |
| --- |
| %sql  select creditlimitBin,sum(Overdue) overdueCount,min(creditlimit) minimum,max(creditlimit) maximum from  (select creditlimit,NTILE(5) over(order by creditlimit) as creditlimitBin,Overdue from tobeAnalysedDF)creditlimitInnerQuery  group by creditlimitBin |

|  |
| --- |
| %sql  select paymenttermBin,sum(overdue) overdueCount,min(paymentterm) as minimum,max(paymentterm) as maximum from (select paymentterm,NTILE(3) over(order by paymentterm) as paymenttermBin,Overdue from tobeAnalysedDF)innerQuery group by paymenttermBin |

|  |
| --- |
| %sql  select amountBin,sum(overdue) overdueCount,min(amount) as minimum,max(amount) as maximum from (select amount,NTILE(5) over(order by amount) as amountBin,Overdue from tobeAnalysedDF)innerQuery group by amountBin |

|  |
| --- |
| %sql  select invoiceAgeBin,sum(overdue) overdueCount,min(invoiceAge) as minimum,max(invoiceAge) as maximum from (select invoiceAge,NTILE(3) over(order by invoiceAge) as invoiceAgeBin,Overdue from tobeAnalysedDF)innerQuery group by invoiceAgeBin |

|  |
| --- |
| %sql  select relationshipAgeBin,sum(overdue) overdueCount,min(relationshipAge) as minimum,max(relationshipAge) as maximum from (select relationshipAge,NTILE(3) over(order by relationshipAge) as relationshipAgeBin,Overdue from tobeAnalysedDF)innerQuery group by relationshipAgeBin |

|  |
| --- |
| %sql  select region,sum(overdue) overdueCount from tobeAnalysedDF group by region |

# Conclusion

AR Overdue Prediction using Apache Spark (Big data) is a complete scalable and highly efficient solution. The process can be fine tuned by data analysts / data scientists. Apache zeppelin implementation of this project will make easier implementation and maintenance for data experts. The entire project can be found at github link.