



Product Recommendation

Team 7 | Vaishali, Nishant, Arpit

04/23/2018

1 Goals

2 Project
Set Up

3 Data
Modelling

4 Testing

5 User
Interface

6 Summary

Prof. Robin Hillyard

Academic semester- Spring 2018
CSYE 7200 Big Data System Engineering with Scala

Goals

- Learn and enhance skills on Scala programming, Apache Spark, Data Analysis
- Meet deadlines of each sprint and activities associated with it
- Collaborative learning and knowledge sharing, to have good team work
- Sharpen presentation skills in a group
- Achieve the stated acceptance criteria

Acceptance
Criteria

Use
Cases

Approach

Timeline

Acceptance Criteria



- Precision for Predictive model will be approx. 0.55

Use Cases

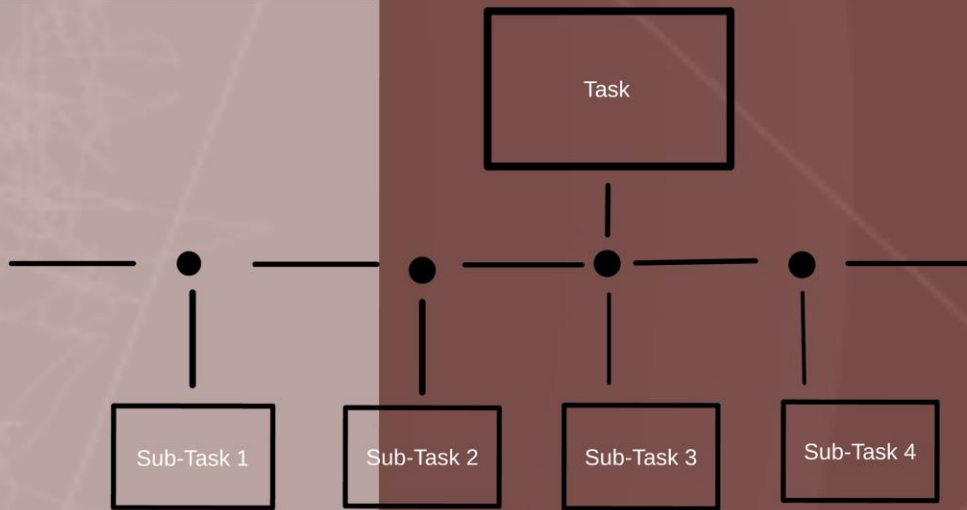


Actor – Customer

- As a new customer [customer for less than 6 months], customer will see top rated banking products
- As an existing customer [customer for more than 6 months], customer will see recommendation for relevant products on the basis of previous months product consumption of a customer

Approach

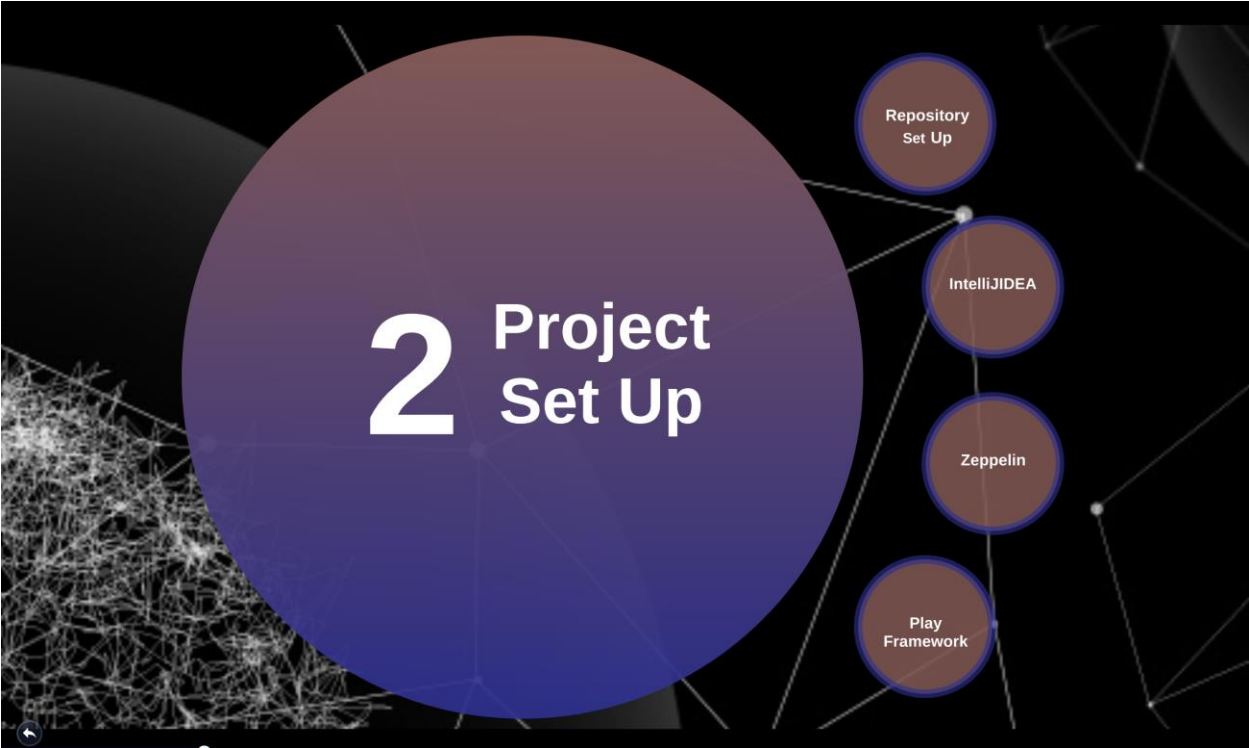
Divide Task in Multiple Sub- Tasks



Timeline

Sprint#	Timeline	Tasks
Sprint 1	03/19/2018 - 03/25/2018	Environment Set Up, Data Cleaning, Data Visualization, Unit Test Cases, Testing
Sprint 2	03/26/2018 - 04/01/2018	ML Spark, Data Modelling, Integration with pipeline, Testing
Sprint 3	04/02/2018 - 04/08/2018	Model fitting and cross validation, Testing,
Sprint 4	04/09/2018 - 04/15/2018	Optimization, UI framework integration
Sprint 5	04/16/2018 - 04/22/2018	Re-Testing and finishing, Preparation of Final Presentation





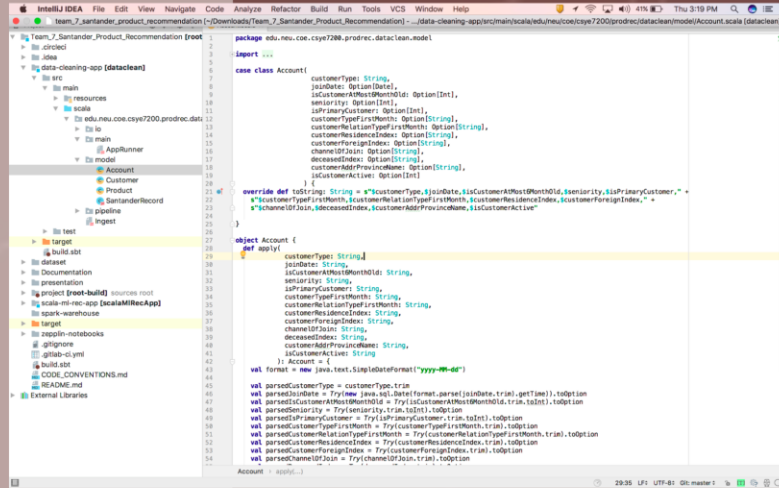
Repository Set Up

Repository View

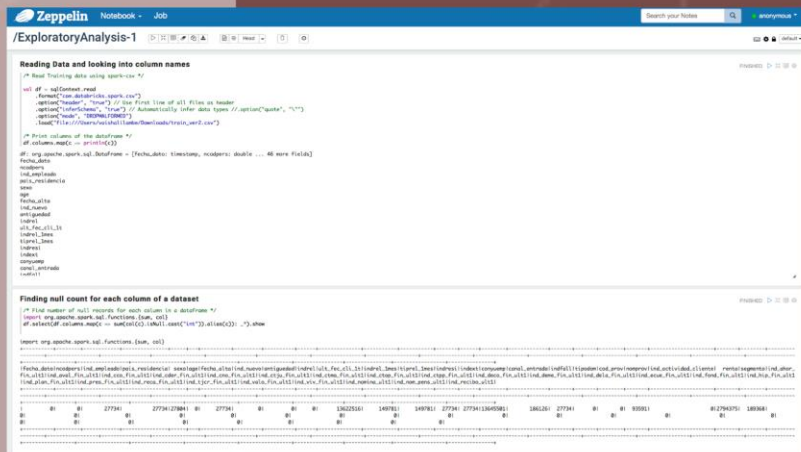
Issue Board

Issue Board

IntelliJ IDEA



Zeppelin



Play Framework

Recommendation application is ready.

3 Data Modelling

Exploratory
Analysis

Data Cleaning

Modelling

Exploratory Analysis

Data Source:
<https://www.kaggle.com/c/santander-product-recommendation/data>

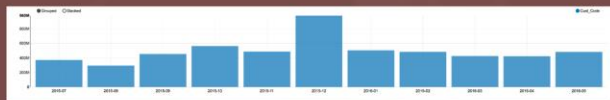
Data Size: ~2.3GB [1.36M rows]



Some Observations

- All columns names are in Spanish which need to convert in English
- Total 15 columns have Null values [Details and approach to handle it is in Repository/ Documentation folder]
- Number of Female Customers [46%] are less than Number of Male customers [54%]
- Last Date being Primary Customer the max value is 2016-05-30
- Individual Customers [58%] are more than VIP [6%] and Students [36%] of total number of customers
- Customer type has values of data type String Double and Integer
- Average gross income of VIP customers is more than Individual and Student customers
- Average gross income of female is lesser than male
- Count of Guarantees is maximum for male individual type customers with average gross income approx. 139593
- Further analysis shows that all the products are consumed maximum by male individual type of customer
- There are more university students as customer than other types [Individual and VIP]
- The Madrid region has maximum number of customers
- Number of customers as per their Join date [First_Date_Acc_Holder] is nearly steady for initial 6 months span of a year

Screenshots of plots/charts



Customer distribution as per last date being primary customer



Regionwise Distribution of Customers

Item_Identifier	Item_Identifier	Item_Identifier	Item_Identifier	Item_Identifier	Item_Identifier
1000000	1000000	1000000	1000000	1000000	1000000
1000000	1000000	1000000	1000000	1000000	1000000
1000000	1000000	1000000	1000000	1000000	1000000
1000000	1000000	1000000	1000000	1000000	1000000
1000000	1000000	1000000	1000000	1000000	1000000

Products consumption as per customer and their gross income, gender, identification

Avg_Gross_Income	Cust_Identifier
175853.08854550548	01 - TOP
141558.18551324733	02 - PARTICULARES
116431.558391003	03 - UNIVERSITARIO

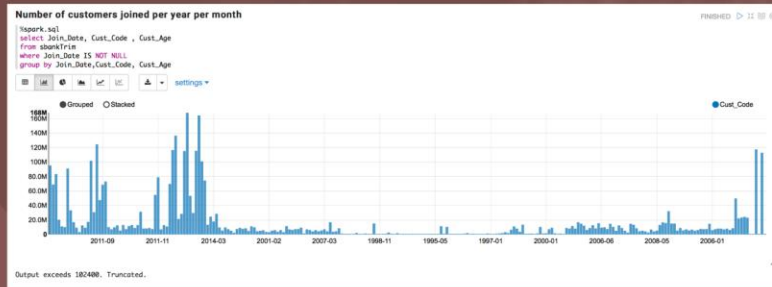
Average Gross income across customer identification



Customer distribution as per gender

Limitations of zeppelin

- Data Visualization is limited [zeppelin.interpreter.output.limit 102400]



[Total imported data was 1.36M rows out of which zeppelin can visualize only 102K, that means 10% of data is visualized]

- Performance issue [very slow with scala,spark and sql interpreter], compare to jupyter notebook zeppelin in quite unstable.

Data Cleaning



Approach:

- Try to make use of Best Practices

```
object Pipeline {  
  def run(input: String, ss: SparkSession): DataFrame = {  
    import ss.implicits._  
  
    /* Parsing */  
    val stringDS = DataParser.getStringDS(input, ss)  
    val classDS = DataParser.stringToClassDS(stringDS, ss)  
  
    /* Filtering */  
    var classDS1 = classDS.filter(d => d.customerInfo.code != None)  
    .filter(d => d.productInfo.product != "[ ]")  
    .filter(d => d.accountInfo.seniority.getOrElse(0) >= 0)  
    .map(d => if (d.customerInfo.age.getOrElse(0) > 100) {  
      new SantanderRecord(  
        Customer(d.customerInfo.code,  
          d.customerInfo.employmentStatus,  
          d.customerInfo.countryOfResidence,  
          d.customerInfo.gender,  
          None,  
          d.customerInfo.income),  
        d.accountInfo,  
        d.productInfo  
      )  
    } else d)  
  
    /* Transformation */  
    var df = TransformationLogic.classDStoDF(classDS1)  
    df = TransformationLogic.replaceEmptyToUnknown(df)  
    df = TransformationLogic.replaceEmptyToRation(df)  
    df = TransformationLogic.replaceNullWithAvg(df)  
    df = TransformationLogic.fixAge(df)  
  
    /* Format */  
    df = TransformationLogic.formatColumnDF(df)  
  
    df  
  }  
}
```

Parsing

Filtering

Transformation

Format

DataModel

```
case class SantanderRecord(customerInfo: Customer, accountInfo: Account, productInfo: Product)
```

```
case class Customer(  
  code: Option[Int],  
  employmentStatus: String,  
  countryOfResidence: String,  
  gender: String,  
  age: Option[Int],  
  income: Option[Double]  
)
```

```
case class Product(product: String)
```

```
case class Account(  
  customerType: Option[String],  
  joinDate: Option[Date],  
  isCustomerAtMost6MonthOld: Option[Int],  
  seniority: Option[Int],  
  isPrimaryCustomer: Option[Int],  
  customerTypeFirstMonth: Option[String],  
  customerRelationTypeFirstMonth: Option[String],  
  customerResidenceIndex: Option[String],  
  customerForeignIndex: Option[String],  
  channelOfJoin: Option[String],  
  deceasedIndex: Option[String],  
  customerAddrProvinceName: Option[String],  
  isCustomerActive: Option[Int]  
)
```

Approach: CommandLine Parsing

<https://github.com/scopt/scopt>

"com.github.scopt" %% "scopt" % "3.7.0"

```
case class Config(input: Option[String] = None, output: Option[String] = None)  
  
object AppRunner extends Serializable {  
  def main(args: Array[String]) {  
    val parser = new scopt.OptionParser[Config]("DataCleaningApp")  
    {  
      head("Data Cleaning App", "1.0")  
      opt[String]("i", "input") required() action  
      { (x, c) => c.copy(input = Some(x)) } text("input is the input path")  
      opt[String]("o", "output") required() action  
      { (x, c) => c.copy(output = Some(x)) } text("output is the output path")  
    }  
  
    parser.parse(args, Config()) match {  
      case Some(config) =>  
        val input = config.input.get  
        val output = config.output.get  
  
        val ss = SparkSession  
          .builder()  
          .appName("Data Cleaning App")  
          .master("local[*]")  
          .config("spark.debug.maxToStringFields", 100)  
          .getOrCreate()  
  
        ss.conf.getAll.foreach(println)  
  
        val resultdf = Pipeline.run(input, ss)  
  
        resultdf.coalesce(1).write.option("header", "true").csv(output)  
      case None =>  
    }  
  }  
}
```

Approach: Reading CSV

Problem Statement:

How to ignore comma (,) inside quoted (") string while splitting csv line? The regex with only comma does not work here.

example:

```
name,address
nishant,"Boston, MA"
Arpit,"Boston"
Vaishali,"Cambridge Ave"
```

```
val splitRow = input.split(""",(?=(("[^"]*"|'['']*'|"[^"]*"|'['']*')*"[^"]*"|'['']*')"""))
```

Approach: Testing Spark Transformation

<https://github.com/holdenk/spark-testing-base>

"com.holdenkarau" %% "spark-testing-base" % "2.2.0_0.9.0" % "test",

```
it should "work for Transformation" in {
  val sqlCx = sqlContext
  import sqlCx.implicits._

  val input1 = sc.parallelize(Seq(Some(10),Some(20),Some(30),None)).toDF(Seq("age"): _*)
  val expectedOutput1 = sc.parallelize(Seq(10,20,30,20)).toDF(Seq("age"): _*)
  val actualOutput1 = TransformationLogic.fixAge(input1)
  assertDataFrameEquals(expectedOutput1, actualOutput1)

  val input2 = sc.parallelize(Seq(Some(10.0),Some(20.0),Some(30.0),None)).toDF(Seq("income"): _*)
  val expectedOutput2 = sc.parallelize(Seq(10.0,20.0,30.0,20.0)).toDF(Seq("income"): _*)
  val actualOutput2 = TransformationLogic.replaceNullWithAvg(input2)
  assertDataFrameEquals(expectedOutput2, actualOutput2)
}
```

Challenges: Dataset Transformation

Problem Statement:

How to do Transformation operation on Dataset[SomeComplexObject] when SomeComplexObject is made of non-primitive datatype members?

```
var classDS1 = classDS.filter(d => d.customerInfo.code != None)
  .filter(d => d.productInfo.product != "[ ]")
  .filter(d => d.accountInfo.seniority.getOrElse(0) >= 0)
  .map(d => if (d.customerInfo.age.getOrElse(0) > 100) {
    new SantanderRecord(
      Customer(d.customerInfo.code,
        d.customerInfo.employmentStatus,
        d.customerInfo.countryOfResidence,
        d.customerInfo.gender,
        None,
        d.customerInfo.income),
      d.accountInfo,
      d.productInfo
    )
  } else d )
```

Challenges: Dataset to DataFrame

Problem Statement:

How to convert Dataset[SomeComplexObject] in flat DataFrame, where flat DataFrame requires more than 22 columns?

Limitation of Existing Solution:

- Every solution on the internet talks about two step proces.
 - Step one: Convert Dataset into tuple.
 - Step two: Use tuple to create DataFrame
- The solution is not applicable when the DataFrame has more than 22 columns, because as of now Scala 2.11 has only Tuple1 to Tuple22.

```
val custCol = Seq("code", "employmentStatus", "countryOfResidence", "gender", "age", "income")
for (x <- custCol) {
  df = df.withColumn(x, col("customerInfo")(x))
}
df = df.drop(col("customerInfo"))

val accCol = Seq("customerType", "joinDate", "isCustomerAtMost6MonthOld", "seniority", "isPrimaryCustomer",
  "customerTypeFirstMonth", "customerRelationTypeFirstMonth", "customerResidenceIndex", "customerForeignIndex",
  "channelOfJoin", "deceasedIndex", "customerAddrProvinceName", "isCustomerActive")
for (x <- accCol) {
  df = df.withColumn(x, col("accountInfo")(x))
}
df = df.drop(col("accountInfo"))

df = df.withColumn("product", col("productInfo")("product"))
df = df.drop(col("productInfo"))
df
```

Ref: <https://underscore.io/blog/posts/2016/10/11/twenty-two.html>

Challenges: Spark Testing

Problem Statement:

- Creating **input** DataFrame with null in Column Values
- Creating **expectedOutput** DataFrame that does not has null in Column Values
- Compare result of **actualOutput** with **expectedOutput** DataFrame without **SchemaMismatch error**

Failed Solution Approach 1:

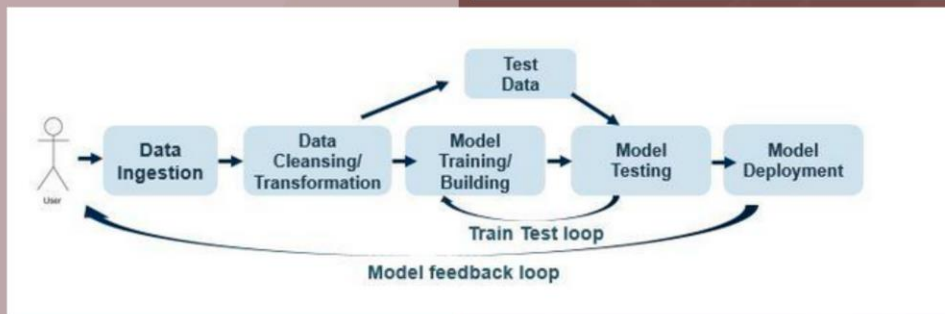
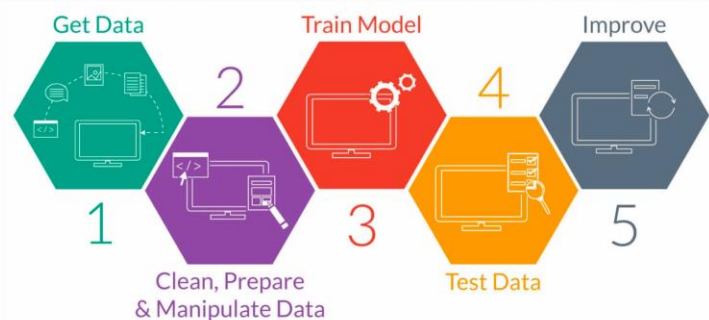
- Use DataFrameGenerator from "com.holdenkarau" %% "spark-testing-base"
- Defining own schema using "StructType" and "StructField" where you have option to make column nullable.
- You can not pass Seq of dataset with None/null. Integer datatype does not support that.

Failed Solution Approach 2:

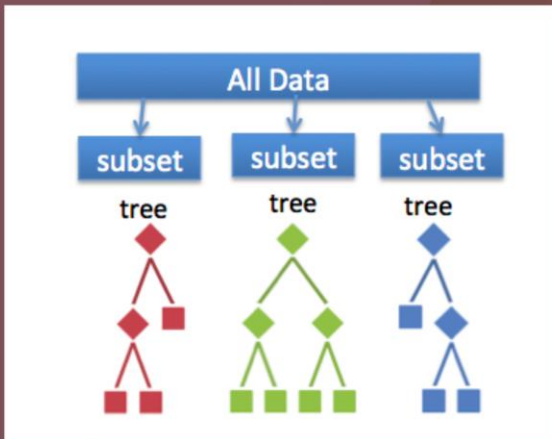
- Use native spark library to create dataframe with Option[Integer].
- The DataFrame encoder does not understand Option[Integer]/Some[Integer]

```
val input1 = sc.parallelize(Seq(Some(10),Some(20),Some(30),None)).toDF(Seq("age"): _*)
val expectedOutput1 = sc.parallelize(Seq(10,20,30,20)).toDF(Seq("age"): _*)
val actualOutput1 = TransformationLogic.fixAge(input1)
assertDataFrameEquals(expectedOutput1, actualOutput1)
```

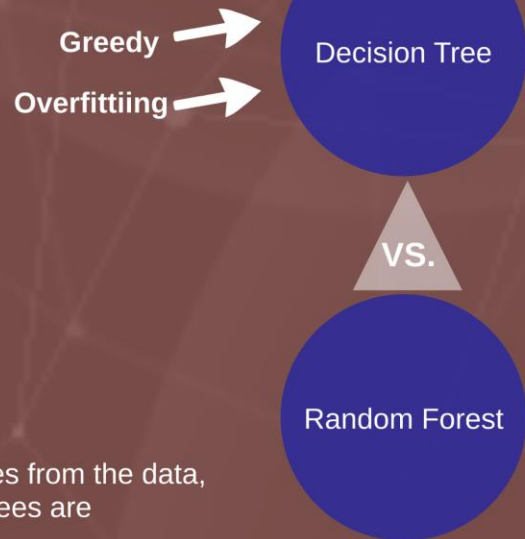
Data Modelling



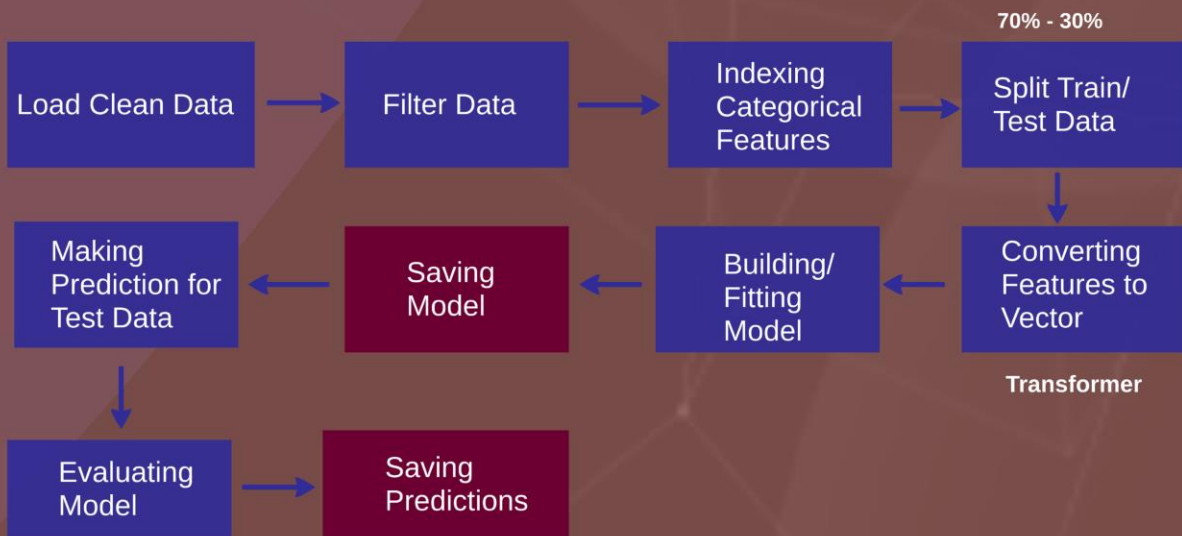
Model Selection 2 1 3



A random forest takes a random subset of features from the data, and creates n random trees from each subset. Trees are aggregated together at end.



Pipeline



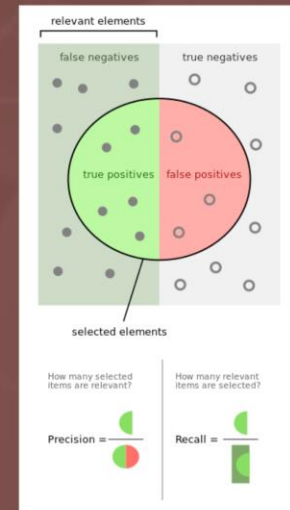
Model Evaluation Metric

predictedLabel	Features
[3]	[1013024.0,23.0,3...
[3]	[1013038.0,38.0,4...
[3]	[1013064.0,56.0,7...
[3]	[1013080.0,36.0,1...
[3]	[1013116.0,50.0,1...
[3]	[9, [0, 1, 2, 5], [101...
[3]	[1013166.0,23.0,3...
[3]	[1013169.0,31.0,1...
[3]	[1013171.0,68.0,9...
[6]	[1013173.0,7.0,95...
[3]	[1013176.0,27.0,4...
[3]	[1013177.0,25.0,1...
[3]	[9, [0, 1, 2, 5], [101...
[3]	[9, [0, 1, 2, 5], [101...
[3]	[9, [0, 1, 2, 5], [101...
[3]	[9, [0, 1, 2], [10131...
[3]	[1013198.0,27.0,7...
[3]	[1013204.0,26.0,1...
[3]	[1013215.0,29.0,8...
[3]	[1013235.0,44.0,1...

Precision - 0.63

```
18/04/22 11:54:29 INFO DAGScheduler: Job 44 finished: collec
18/04/22 11:54:29 INFO org: Precision = 0.6303460983226195
18/04/22 11:54:30 INFO SQLHadoopMapReduceCommitProtocol: Usi
```

Correct positive predictions



Challenges

- Amount of data
- Features Selection
- Tuning Random Forest Model
- Handling categorical features
- Testing Apache spark code
- Gradient Boosted Trees do not yet support multiclass classification

```
import org.scalatest.{FlatSpec, Matchers}
import org.apache.spark.sql.SparkSession

trait SparkSessionTestWrapper {

  lazy val spark: SparkSession = {
    SparkSession
      .builder()
      .master("local")
      .appName("Spark Data Model App Test")
      .getOrCreate()
  }

}

class AppSpec extends FlatSpec with SparkSess
  behavior of "load data method"
```

```
package edu.neu.coe.csye7200.prodrec.learning

import org.apache.log4j.Logger
import org.apache.spark.sql.{DataFrame, SparkSession}
import org.joda.time.{DateTime, Seconds}
import org.apache.spark.ml.classification.RandomForestClassifier
import org.apache.spark.ml.feature.{IndexToString, StringIndexer, StringIndexerModel}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
//import org.apache.spark.ml.classification.NaiveBayes
//import org.apache.spark.ml.classification.DecisionTreeClassifier

object DataModelApp extends App {

  val SANTANDER_PRODUCT_RECOMMENDATION_APP = "Santander Product Recomm
  val SET_UP_MESSAGE_COMPLETION = "Spark Set Up Complete"
  val CREATE_MODEL_MESSAGE = "Creating Random Forest Model"
  val SAVE_MODEL_MESSAGE = "Saving the Model"
  val CREATE_PIPELINE_MESSAGE = "Creating Pipeline"

  val filePath = "./dataset/clean_data.csv"
  val modelSavePath = "./dataset/spark-random-forest-model"
  val maxBin: Int = 60
  val splitSeed: Int = 5043
  val metricName = "weightedPrecision"
  val featuresColName = "Features"
  val productColName = "product"
  val productIndexedColName = "productIndexed"
  val predictColName = "prediction"
  val predictOutputColName = "predictedLabel"

  val numericColNames: Seq[String] = Seq("code", "age", "income")
  val categoricalColNames: Seq[String] = Seq(
    "gender",
    "employmentStatus",
    "customerAddrProvinceName",
    "customerRelationTypeFirstMonth",
    "customerType",
    "deceasedIndex"
  )

  val logger = getLogger()
```

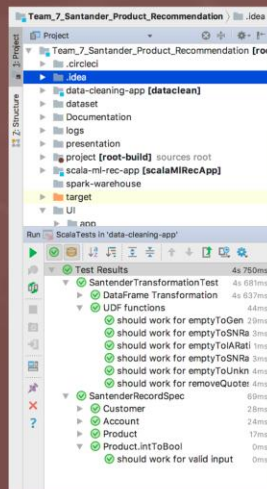
4 Testing

Coverage

Report

Coverage

Data Transformation & Cleaning



Team_7_Santander_Product_Recommendation

Project

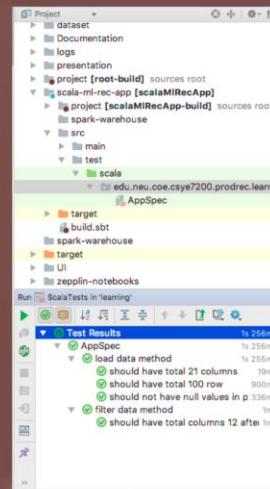
- Team_7_Santander_Product_Recommendation (root)
- data-cleaning-app [dataclean]
- dataset
- Documentation
- logs
- presentation
- project [root-build] sources root
- scala-mi-rec-app [scalaMIRecApp]
- spark-warehouse
- target
- UI
- app

Run ScalaTests in 'data-cleaning-app'

Test Results

Test	Time
SantanderTransformationTest	4s 750ms
DataFrame Transformation	4s 681ms
UDF functions	44ms
should work for emptyToGen	3ms
should work for emptyToSNRa	3ms
should work for emptyToIRa	1ms
should work for emptyToSNRa	3ms
should work for emptyToLink	4ms
should work for removeQuarter	4ms
SantanderRecordSpec	60ms
Customer	28ms
Account	24ms
Product	17ms
Product.intToBool	0ms
should work for valid input	0ms

ML App



Project

- dataset
- Documentation
- logs
- presentation
- project [root-build] sources root
- scala-mi-rec-app [scalaMIRecApp]
- project [scalaMIRecApp-build] sources root
- spark-warehouse
- src
- main
- test
- scala
- edu.neu.coe.csye7200.prodrec.learnin
- AppSpec
- target
- build.sbt
- spark-warehouse
- target
- UI
- zeppelin-notebooks

Run ScalaTests in 'learning'

Test Results

Test	Time
AppSpec	1s 216ms
load data method	1s 215ms
should have total 21 columns	18ms
should have total 100 row	900ms
should not have null values in p	336ms
filter data method	1ms
should have total columns 12 after	1ms

Coverage Report

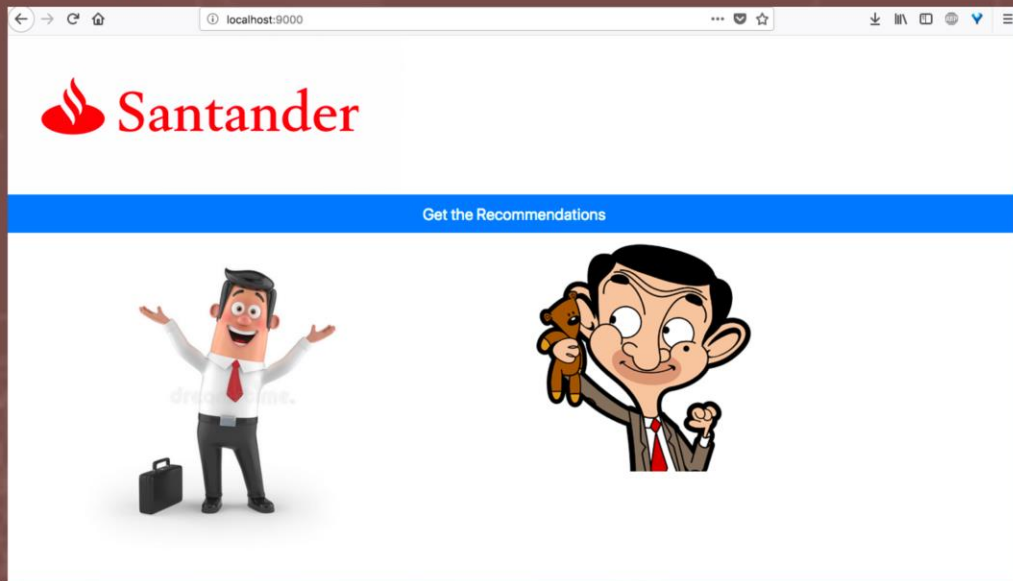
43.60%

All packages		SCoverage generated at: Sun Apr 22 19:59:44 EDT 2018									
edu.neu.coe.csy7200.prodrec.dataclean.io	0.00%										
edu.neu.coe.csy7200.prodrec.dataclean.main	0.00%	Lines of code: 629 Files: 8 Classes: 8 Methods: 26									
edu.neu.coe.csy7200.prodrec.dataclean.model	88.04%	Lines per file: 78.62 Packages: 5 Classes per package: 1.60 Methods per class: 3.25									
edu.neu.coe.csy7200.prodrec.dataclean.pipeline	31.82%	Total statements: 500 Invoked statements: 218 Total branches: 64 Invoked branches: 49									
edu.neu.coe.csy7200.prodrec.learning	12.07%	Ignored statements: 0									
Statement coverage: 43.60 %		<div><div></div></div>				Branch coverage: 76.56 %		<div><div></div></div>			
Class	Source file	Lines	Methods	Statements	Invoked	Coverage	Branches	Invoked	Coverage		
DataParser	DataParser.scala	36	2	51	0	<div><div></div></div> 0.00 %	0	0	<div><div></div></div> 0.00 %		
AppRunner	AppRunner.scala	40	1	17	0	<div><div></div></div> 0.00 %	0	0	<div><div></div></div> 0.00 %		
Account	Account.scala	54	1	33	33	<div><div></div></div> 100.00 %	0	0	<div><div></div></div> 100.00 %		
Customer	Customer.scala	36	1	19	19	<div><div></div></div> 100.00 %	2	2	<div><div></div></div> 100.00 %		
Product	Product.scala	73	2	132	110	<div><div></div></div> 83.33 %	48	37	<div><div></div></div> 77.08 %		
Pipeline	Pipeline.scala	41	1	27	0	<div><div></div></div> 0.00 %	2	0	<div><div></div></div> 0.00 %		
TransformationLogic	TransformationLogic.scala	133	12	105	42	<div><div></div></div> 40.00 %	12	10	<div><div></div></div> 83.33 %		
DataModelApp	App.scala	216	6	116	14	<div><div></div></div> 12.07 %	0	0	<div><div></div></div> 100.00 %		

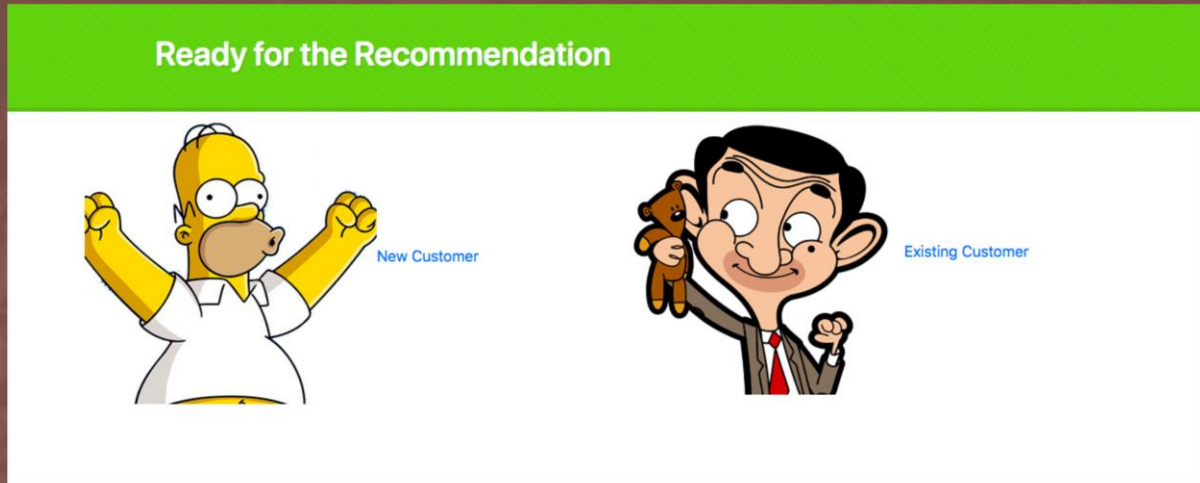
5 User Interface

Screenshots

Homepage



Manager Profile



Existing Customer Page

Recommendations

Customer No. 1050616

Products 3



New Customer Page

Recommendations

Customer No. 1050610

Products 0

Current Account

Payroll Nom

Direct Debit

Junior Account

Pension Nom

Long Term Deposit

Summary

1 . Met acceptance criteria

Achieved precision is: 0.63 [More than stated in acceptance criteria]

2. Delivered Project On Time

3. Improved Collaborative learning, Knowledge sharing

4. Improved Presentation Skills