CSEE 5590-0001/490-0003: Big Data Programming Project

Team Name : Noone

Member 1: Raju Nekadi

Class Id: 7

Member 2: Sushma Manne

Class Id: 3

Member 3: Shekar Pentakota

Class ID : 9

Member 4: Bilal Mustafa

Class ID : 15

Source Code Github Link :

https://github.com/rnekadi/Big_Data_Project_Fall_2018

Video Web Anomalies:

https://youtu.be/clsCTPlaGgE

Video Loan Prediction:

https://www.youtube.com/watch?v=kVZaC1hI5NU

Table of Content

Web Traffic Anomalies

- ## Introduction :
  Web Traffic Anomalies project framework is based on Apache Flume,Apache Spark Streaming, Spark Sql and Apache Cassandra which can detect and report abnormal HTTPS requests in seconds on from any web log server.

- **Background :**
  For any company of all types and sizes website availability and performance are very critical, not only those with revenue stream tied to web. Website or particular web page can become unpopular for any reason such as overburding , content management and etc. Search engine quickly apply a significant ranking penalty to slowly loading pages. Therefore most of companies looking for solution which can prevent long term damages.

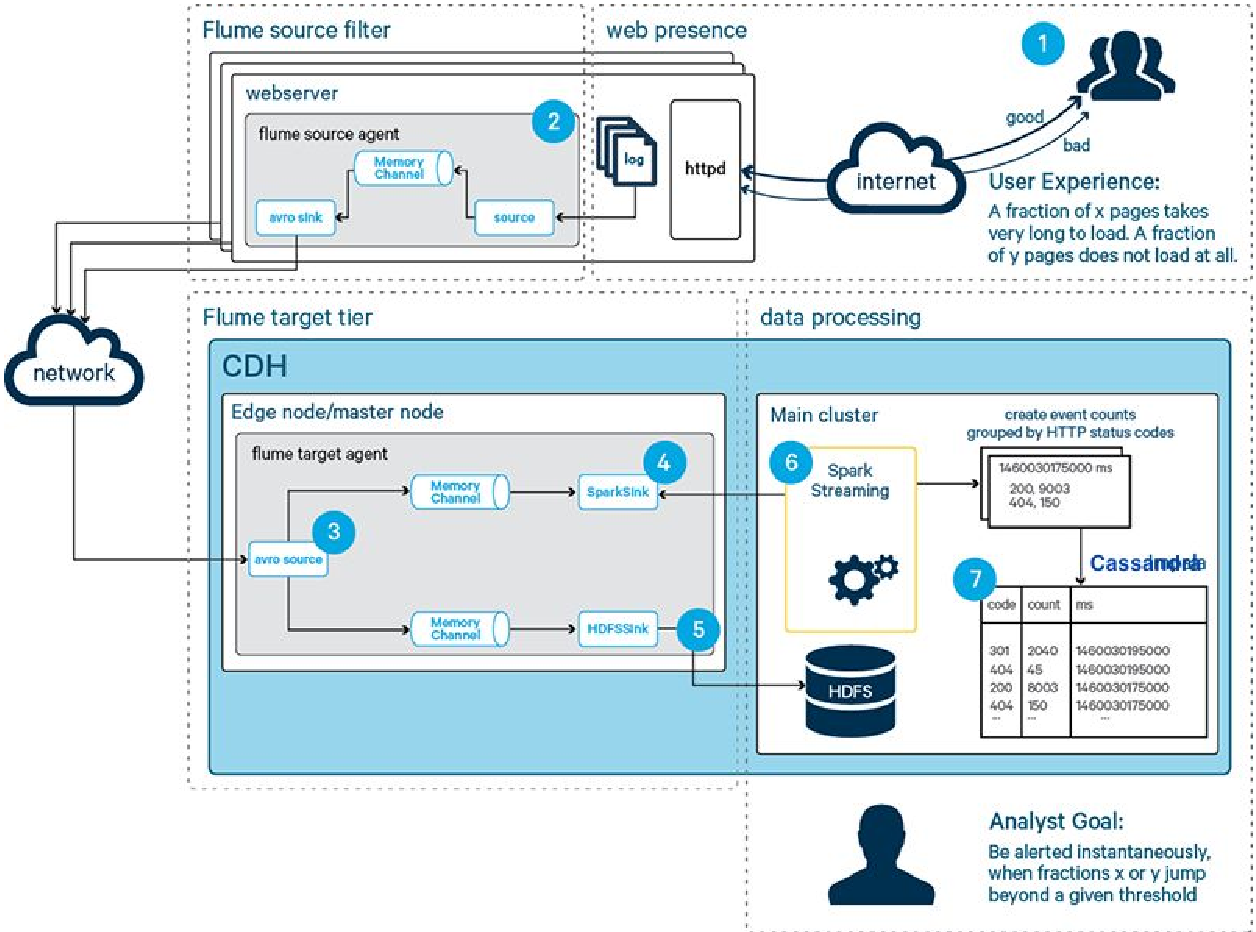
  The main idea of this project comes from Metlife, that provides the Auto and Home Insurance across all states in US. Metlife, store the web-servers access logs and use same to enhance the website performance and availability.
  Using this web logs we can can setup framework which can hub to detect and report errors as they happen in near-real time.

- **Model**
  - **Architecture Diagram with explanation**

Metlife web servers generate up to 2 million user session per day, which generate several thousands HTTP GET requests per minutes.

The approach of our framework is to use flume and Spark Streaming application to feed the Cassandra table with every n minutes with the current counts of HTTP status code within n windows.

Below pic represent the architecture and design of our framework.



Event Format Tier :

In this tier user logs to the website and creates the one line log data used by web server and save it in common log format file.

The access log line entry looks as below.

```
163.205.2.105 -- [20/Jul/1995:07:37:17 -0400] "GET/images/NASA-logosmall.gif HTTP/1.0" 200 786
```

The data format of this access log line will be explained in Dataset section in more details.

Flume Tier :
In this framework we used the Flume agents in two tiers one is Source Tier and Target Tier. The Source Tier collects the event on web server and extract log from https.
The second, Target Tier executes on hadoop cluster and events are replicated to Spark Streaming using the Flume Sink(Spark Sink).

Spark Streaming Tier:
The goal of this framework is to capture and cout all the bad requests and detect the rise in error near real time.

This Frame responsible for :

1. Consuming all log events from Flume Sink(Spark Sink) and adding them discretized streams.
2. After considerable them time counting all the event occurrence grouped by status code.
3. Writing the Status code and code to Cassandra table for monitoring.

We count all the events occurence of status code in window frame and group them as shown below in key-value pair.

Key	Value
HTTP Status Code	Count

Spark Stream creates Dsteram grouped by status code and counting all occurrences from start time to end time of that window. As result key value pair marked with end time of window in milliseconds.

Key	Value	ms
HTTP Status Code	Count	Timestamp

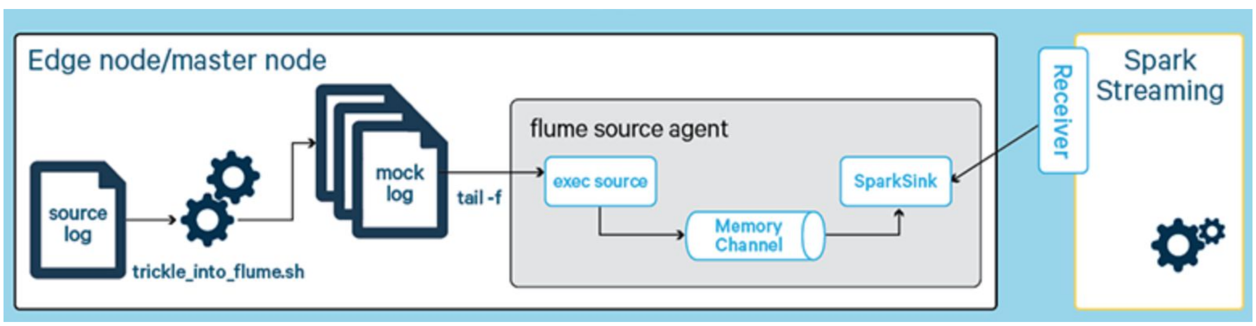
Cassandra Tier:
This framework is responsible for saving all the status code , count and time information in Cassandra table for visualization, further monitoring and analysis.

- **Workflow diagram with explanation**

The approach to transfer log data from Flume agents(Flume Sinks) to Spark Streaming is called Pull based approach.In this approach we run the custom Flume Sink that allows following.

1. Flume agent pushed the data or access log into sink and data remained buffered there.
2. Spark Streaming uses the reliable Spark Sink which pull the data from the flume agent sink.

This approach provides strong reliability and fault tolerance.



As you can see in this diagram framework Source, Flume agent and Spark Streaming as explained above.

- **Dataset:**

  Detailed Description of Dataset

  The logs are an ASCII file with one line per request, with the following columns:

1. **host** making the request. A hostname when possible, otherwise the Internet address if the name could not be looked up.
2. **timestamp** in the format "DAY MON DD HH:MM:SS YYYY", where DAY is the day of the week, MON is the name of the month, DD is the day of the month, HH:MM:SS is the time of day using a 24-hour clock, and YYYY is the year. The timezone is -0400.
3. **request** given in quotes.
4. **HTTP** reply code.
5. **bytes** in the reply.

Sample log looks like below.

```
163.200.2.108 -- [20/Jul/1995:07:37:17 -0400] "GET/images/NASA-logosmall.gif HTTP/1.0" 400 746
```

Detail design of Features with diagram

Not applicable for this framework.

- **Analysis of Data**

  As we are using the data from access log file no need of preprocessing or further analysis is required in our framework.

- **Implementation:**

  Now let's explain the implementation of each framework component in detail below.

  Flume Implementation:

  As explained in earlier section we are using pull based approach to connect Spark Streaming to Flume.Spark Streaming receiver uses the transaction to get event from the `org.apache.spark.streaming.flume.sink.SparkSink`

  Explained in http://spark.apache.org/docs/latest/streaming-flume-integration.html

The configuration for Flume conf is shown below.

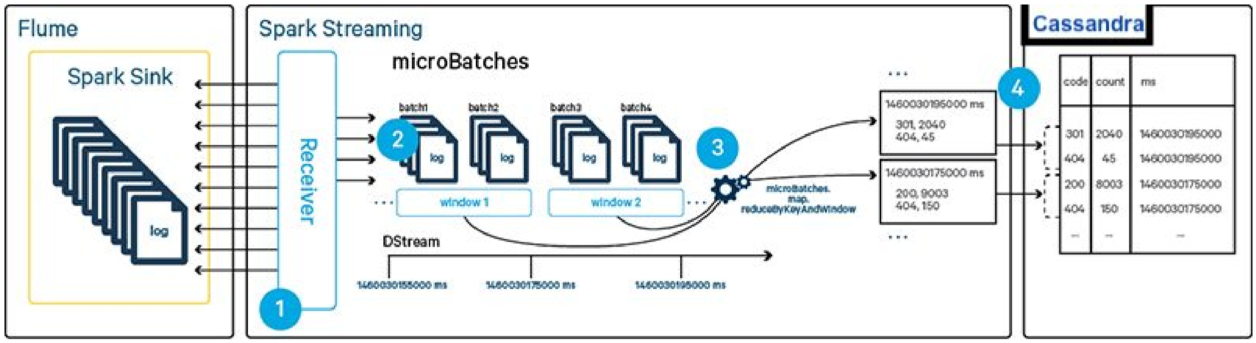
Its has one source agent called apache_server which sources the log data.

It has memory channels which act as medium of transferring data from source to sink.
It has one spark sink which buffers the log for Spark Streaming as we can we have tied this sink at **localhost at port 33333**

```
# Source
rn_myAgent.sources = apache_server

rn_myAgent.sources.apache_server.type = exec
rn_myAgent.sources.apache_server.command = tail -f
/Users/sai/Documents/GitHub/Big_Data_Project_Fall_2018/logs/logs.txt
rn_myAgent.sources.apache_server.batchSize = 1000
rn_myAgent.sources.apache_server.channels = myMemoryChannel

#Memmory


rn_myAgent.channels = myMemoryChannel
rn_myAgent.channels.myMemoryChannel.type = memory
rn_myAgent.channels.myMemoryChannel.capacity = 200000
rn_myAgent.channels.myMemoryChannel.transactionCapacity = 200000

## Send to sink that spark streaming pulls on

rn_myAgent.sinks = spark1
rn_myAgent.sinks.spark1.type= org.apache.spark.streaming.flume.sink.SparkSink
rn_myAgent.sinks.spark1.hostname = localhost
rn_myAgent.sinks.spark1.port = 33333
rn_myAgent.sinks.spark1.channel = myMemoryChannel
```

Spark Streaming Implementation:

The below diagram shows the Spark Streaming flow.



Spark Streaming Phases are described below.

Parameter:

We have used the open source **scopt** parsing library to pass and validate the parameters to our program and also **Config case class** to hold this parameter in our program.

```
case class rn_Config(
                  rn_master: String = "local[5]",
                  rn_out : String = "",
                  rn_agg: String = "",
                  rn_cp: String = "",
                  rn_flumeHost: String = "localhost",
                  rn_flumePort: Int = 33333,
                  rn_numStreams: Int = 1,
                  rn_batchSeconds: Int = 15,
                  rn_slideSeconds: Int = 300,
                  rn_windowSeconds: Int = 300,
                  rn_verbose: Boolean = false
                )
```

The below mentioned parameter are made available.

master	Defines in which mode the application will run (yarn-client/cluster/local).
out	Defines the HDFS location where the events within the window get written
agg	Defines the _required_ HDFS location where event counts grouped by error codes within the window get written to
cp	Defines the _required_ HDFS location where Spark will materialize checkpoints for stateful transformations
flumeHost	Defines the host running the flume spark sink to which the receiver will connect. Defaults to 'localhost'
flumePort	Defines the port running the flume spark sink that the receiver will connect to. Defaults to 7777
batchSeconds	Defines the interval of micro-batches
slideSeconds	Defines the interval at which the sliding window of events gets written to HDFS
windowSeconds	Defines the amount of time that the sliding window reaches back to

Extracting Status Code :

To extract the status code from the events to Spark's Sinks, we define a class called **Interrogator**

```
class rn_Interrogator {
 val rn_httpFlumeHeaderRegex = """.host.[^ ]*. uuid.[^ ]*. timestamp.[^ ]*."""
 val rn_validIpRegex = """[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}"""
 val rn_validHostnameRegex = """[^ ]*\.[^ ].*"""
 val rn_validIpORHostnameRegex = """[^ ]*\.[^ ].*"""
 val rn_validDateRegex =
"""\[[0-3]*[0-9]\/[a-zA-Z]{3}\/[0-9]{4}\:[0-9]{2}\:[0-9]{2}\:[0-9]{2} .{5}\]"""
 val rn_validPortRegex = """\:[0-9]{2}"""
 val rn_validReqRegex = """\"[A-Z]{3}.*HTTP\/.{3}\""""
 val rn_validCodeRegex = """[1-5][0-5][0-9]"""
```

We have defined getstatuscode function which will extract the HTTP code from the log data line by line as these contains regular expression.

```
def rn_getStatusCode(body : String) : String = {
 val list = List (
  ".*",
  rn_validIpORHostnameRegex,
  " ",
 rn_ validDateRegex,
  ".* ",
  rn_validReqRegex,
  " ",
  "(" + rn_validCodeRegex + ")" + " .*"
 )
```

Micro Batches Creation:

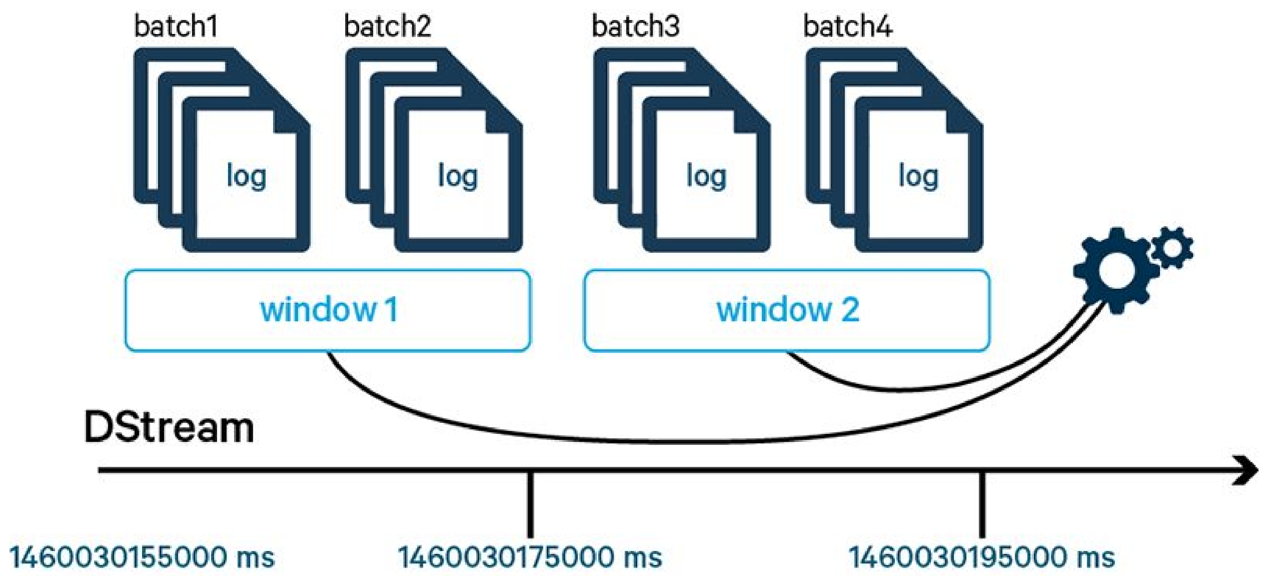
Creating a discretized stream of events from the Flume access log events via the following code:

```
val rn_flumeStreams = (0 to (myConfig.numStreams - 1)).map { i =>
FlumeUtils.createPollingStream(ssc, "localhost", 33333)}
```

Above code generates DStreaming RDDs that are managed by Spark.

The map function map the log events into key-value pair that aggregate error code and count.

```
val rn_microAgg = unifiedFlumeStream.map(e => (new String(new
Interrogator().getStatusCode(new String(e.event.getBody.array()))).toInt, 1))
```



The reduce function reduces all the log events that occured within window time and produce a new DStream on that window reduction is done by adding new events(+1) and subtracting old event as shown in code below.

```
val rn_reportedAgg = rn_microAgg.reduceByKeyAndWindow(
 {(a,b) => a + b},
 {(a,b) => a - b},
 Seconds(rn_windowSeconds),
 Seconds(rn_slideSeconds)
)
```

The resulting **reportedAgg** DStream is now being pushed to Cassandra as described below.

Storage to Cassandra:

To store the data to Cassandra we have created the destination table and then used the Spark Streaming to continuously add data to destination at each window.

The DDL is shown below.

```
CREATE TABLE web.errornrt_table (
    rn_code text PRIMARY KEY,
    rn_counts text,
    rn_ms text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class':
'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

Now to add the RRDs rows to errornrt_table table we have used the Spark SQL ,Spark to Cassandra Connectors.

We have setup new Conf which points to Cassandra connection details.

```
val rn_conf = new SparkConf(true).set("spark.cassandra.connection.host", cassandraHost)
```

We have also setup HTTP event class which maps each RDDs into format.

```
case class rn_HTTPEvent(rn_code: String, rn_counts: String, rn_ms: String)
```

Below is the FInal COde which pushes the RDDs one by one for aggregated status code along with count and window time in millisecond.

```
val sqlContext = SparkSession
   .builder()
   .appName("Spark SQL basic example")
   .config(conf =conf)
   .getOrCreate()

import sqlContext.implicits._
reportedAgg.foreachRDD((rdd, time) => {
 println("foreachRDD")
 val rowsDF = rdd.
   filter(x=> x._2 > 0).
   map(x => rn_HTTPEvent(x._1.toString, x._2.toString, time.toString().stripSuffix(" ms")))
 //rowsDF.show()


 val webDF = sqlContext.createDataFrame(rowsDF)
 webDF.write.format("org.apache.spark.sql.cassandra").options(Map( "table" ->
"errornrt_table", "keyspace" -> "web")).mode(SaveMode.Append).save()
 webDF.show()

})
```

Thus these are the steps which finally explains all the Implementation steps involved here. In next, section we will discuss about the results we got using these steps.

Results Evaluation:

As we can see we will get the data we according to our discussion.

```
cqlsh:web> select * from rn_errornrt_table;


 code | counts | ms
------+--------+---------------
  200 |      5 | 1543607715000
  302 |     15 | 1543605495000
  404 |      2 | 1543605495000
  304 |     13 | 1543605495000
```

- **Conclusion :** Flume , Spark Streaming and Cassandra provides the excellent framework for capturing and storing the data at near real time.
- **Future Work :** None

Project Management

Implementation status report

Work completed:

Description:

1. Implemented the Flume agent for generating spark sink.
2. Implemented Spark Stream which read the stream of data and write to table.
3. Storing the Streaming Data to Cassandra table.

Responsibility (Task, Person)

1. Flume Configuration and Setup : **Raju Nekadi**
2. Spark Streaming: **Raju Nekadi**
3. Storing the Streaming Data to Cassandra table: Sushma Manne

Contributions (members/percentage):
Raju Nekadi 50%
Sushma Manne 50%

Issues/Concerns:
None

References/Bibliography:

https://stdatalabs.com/2016/11/spark-streaming-flume-example-pull-approac/
http://spark.apache.org/docs/latest/streaming-flume-integration.html
https://github.com/scopt/scopt
https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlshDescribe.html
http://datastax.github.io/spark-cassandra-connector/ApiDocs/2.3.2/spark-cassandra-connector/#com.datastax.spark.connector.package
https://www.supergloo.com/fieldnotes/apache-spark-cassandra/

PREDICTING LOAN CREDIT RISK

Introduction: Machine Learning is been part of everyday life of human beings without knowing them, suppose if we look at YouTube videos, they analyse our data and sends us the videos which we cannot skip watching them. Not only YouTube it’s been used many top companies like Google, Microsoft, Apple and many others. In our project we are going to predict bank loan credit risk using Apache sparks ml Random Forests and Decision trees.

Background:

Tools and Technologies - Spark ML (Classification Algorithms Random Forests and Decision Tree), IntelliJ

Input data contains string columns which must be converted to integers.

Before conversion -

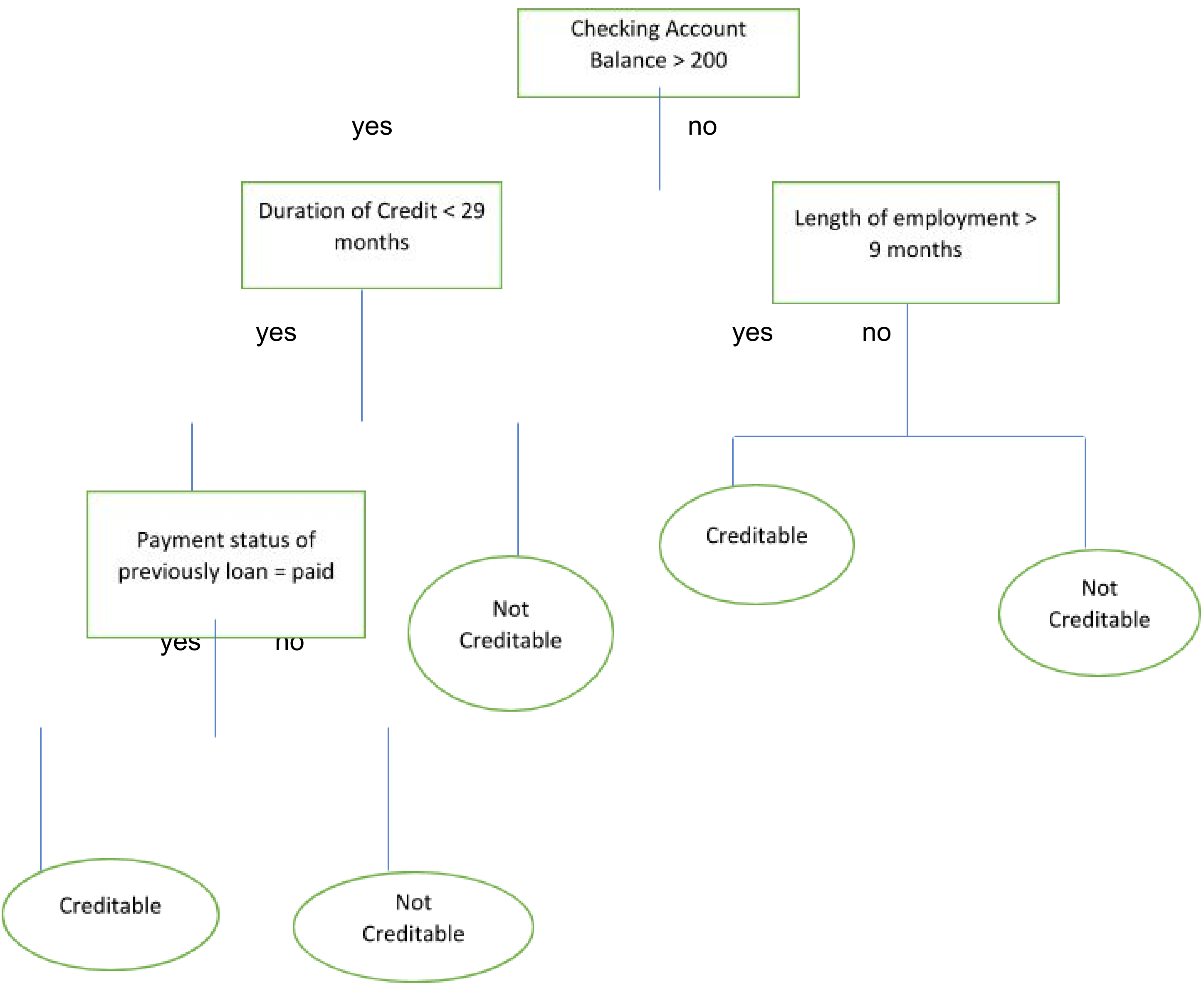
```
A11 6 A34 A43 1169 A65 A75 4 A93 A101 4 A121 67 A143 A152 2 A173 1 A192 A201 1
A12 48 A32 A43 5951 A61 A73 2 A92 A101 2 A121 22 A143 A152 1 A173 1 A191 A201 2
A14 12 A34 A46 2096 A61 A74 2 A93 A101 3 A121 49 A143 A152 1 A172 2 A191 A201 1
A11 42 A32 A42 7882 A61 A74 2 A93 A103 4 A122 45 A143 A153 1 A173 2 A191 A201 1
A11 24 A33 A40 4870 A61 A73 3 A93 A101 4 A124 53 A143 A153 2 A173 2 A191 A201 2
A14 36 A32 A46 9055 A65 A73 2 A93 A101 4 A124 35 A143 A153 1 A172 2 A192 A201 1
A14 24 A32 A42 2835 A63 A75 3 A93 A101 4 A122 53 A143 A152 1 A173 1 A191 A201 1
A12 36 A32 A41 6948 A61 A73 2 A93 A101 2 A123 35 A143 A151 1 A174 1 A192 A201 1
A14 12 A32 A43 3059 A64 A74 2 A91 A101 4 A121 61 A143 A152 1 A172 1 A191 A201 1
A12 30 A34 A40 5234 A61 A71 4 A94 A101 2 A123 28 A143 A152 2 A174 1 A191 A201 2
A12 12 A32 A40 1295 A61 A72 3 A92 A101 1 A123 25 A143 A151 1 A173 1 A191 A201 2
A11 48 A32 A49 4308 A61 A72 3 A92 A101 4 A122 24 A143 A151 1 A173 1 A191 A201 2
A12 12 A32 A43 1567 A61 A73 1 A92 A101 1 A123 22 A143 A152 1 A173 1 A192 A201 1
A11 24 A34 A40 1199 A61 A75 4 A93 A101 4 A123 60 A143 A152 2 A172 1 A191 A201 2
A11 15 A32 A40 1403 A61 A73 2 A92 A101 4 A123 28 A143 A151 1 A173 1 A191 A201 1
A11 24 A32 A43 1282 A62 A73 4 A92 A101 2 A123 32 A143 A152 1 A172 1 A191 A201 2
A14 24 A34 A43 2424 A65 A75 4 A93 A101 4 A122 53 A143 A152 2 A173 1 A191 A201 1
A11 30 A30 A49 8072 A65 A72 2 A93 A101 3 A123 25 A141 A152 3 A173 1 A191 A201 1
A12 24 A32 A41 12579 A61 A75 4 A92 A101 2 A124 44 A143 A153 1 A174 1 A192 A201 2
A14 24 A32 A43 3430 A63 A75 3 A93 A101 2 A123 31 A143 A152 1 A173 2 A192 A201 1
A14 9 A34 A40 2134 A61 A73 4 A93 A101 4 A123 48 A143 A152 3 A173 1 A192 A201 1
A11 6 A32 A43 2647 A63 A73 2 A93 A101 3 A121 44 A143 A151 1 A173 2 A191 A201 1
A11 10 A34 A40 2241 A61 A72 1 A93 A101 3 A121 48 A143 A151 2 A172 2 A191 A202 1
A12 12 A34 A41 1804 A62 A72 3 A93 A101 4 A122 44 A143 A152 1 A173 1 A191 A201 1
A14 10 A34 A42 2069 A65 A73 2 A94 A101 1 A123 26 A143 A152 2 A173 1 A191 A202 1
A11 6 A32 A42 1374 A61 A73 1 A93 A101 2 A121 36 A141 A152 1 A172 1 A192 A201 1
A14 6 A30 A43 426 A61 A75 4 A94 A101 4 A123 39 A143 A152 1 A172 1 A191 A201 1
A13 12 A31 A43 409 A64 A73 3 A92 A101 3 A121 42 A143 A151 2 A173 1 A191 A201 1
A12 7 A32 A43 2415 A61 A73 3 A93 A103 2 A121 34 A143 A152 1 A173 1 A191 A201 1
A11 60 A33 A49 6836 A61 A75 3 A93 A101 4 A124 63 A143 A152 2 A173 1 A192 A201 2
A12 18 A32 A49 1913 A64 A72 3 A94 A101 3 A121 36 A141 A152 1 A173 1 A192 A201 1
A11 24 A32 A42 4020 A61 A73 2 A93 A101 2 A123 27 A142 A152 1 A173 1 A191 A201 1
```

After conversion –

1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1	1
2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1	2
4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0	1
1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	1	1
1	24	3	49	1	3	3	4	4	53	3	2	2	1	1	1	0	1	0	0	0	0	0	1	2
4	36	2	91	5	3	3	4	4	35	3	1	2	2	1	0	0	1	0	0	0	0	1	0	1
4	24	2	28	3	5	3	4	2	53	3	1	1	1	1	0	0	1	0	0	1	0	0	1	1
2	36	2	69	1	3	3	2	3	35	3	1	1	2	1	0	1	1	0	1	0	0	0	0	1
4	12	2	31	4	4	1	4	1	61	3	1	1	1	1	0	0	1	0	0	1	0	1	0	1
2	30	4	52	1	1	4	2	3	28	3	2	1	1	1	1	0	1	0	0	1	0	0	0	2
2	12	2	13	1	2	2	1	3	25	3	1	1	1	1	1	0	1	0	1	0	0	0	1	2
1	48	2	43	1	2	2	4	2	24	3	1	1	1	1	0	0	1	0	1	0	0	0	1	2
2	12	2	16	1	3	2	1	3	22	3	1	1	2	1	0	0	1	0	0	1	0	0	1	1
1	24	4	12	1	5	3	4	3	60	3	2	1	1	1	1	0	1	0	0	1	0	1	0	2
1	15	2	14	1	3	2	4	3	28	3	1	1	1	1	1	0	1	0	1	0	0	0	1	1
1	24	2	13	2	3	2	2	3	32	3	1	1	1	1	0	0	1	0	0	1	0	1	0	2
4	24	4	24	5	5	3	4	2	53	3	2	1	1	1	0	0	1	0	0	1	0	0	1	1
1	30	0	81	5	2	3	3	3	25	1	3	1	1	1	0	0	1	0	0	1	0	0	1	1
2	24	2	126	1	5	2	2	4	44	3	1	1	2	1	0	1	1	0	0	0	0	0	0	2
4	24	2	34	3	5	3	2	3	31	3	1	2	2	1	0	0	1	0	0	1	0	0	1	1
4	9	4	21	1	3	3	4	3	48	3	3	1	2	1	1	0	1	0	0	1	0	0	1	1
1	6	2	26	3	3	3	3	1	44	3	1	2	1	1	0	0	1	0	1	0	0	0	1	1
1	10	4	22	1	2	3	3	1	48	3	2	2	1	2	1	0	1	0	1	0	0	1	0	1
2	12	4	18	2	2	3	4	2	44	3	1	1	1	1	0	1	1	0	0	1	0	0	1	1
4	10	4	21	5	3	4	1	3	26	3	2	1	1	2	0	0	1	0	0	1	0	0	1	1
1	6	2	14	1	3	3	2	1	36	1	1	1	2	1	0	0	1	0	0	1	0	1	0	1
4	6	0	4	1	5	4	4	3	39	3	1	1	1	1	0	0	1	0	0	1	0	1	0	1
3	12	1	4	4	3	2	3	1	42	3	2	1	1	1	0	0	1	0	1	0	0	0	1	1
2	7	2	24	1	3	3	2	1	34	3	1	1	1	1	0	0	0	0	0	1	0	0	1	1
1	60	3	68	1	5	3	4	4	63	3	2	1	2	1	0	0	1	0	0	1	0	0	1	2
2	18	2	19	4	2	4	3	1	36	1	1	1	2	1	0	0	1	0	0	1	0	0	1	1

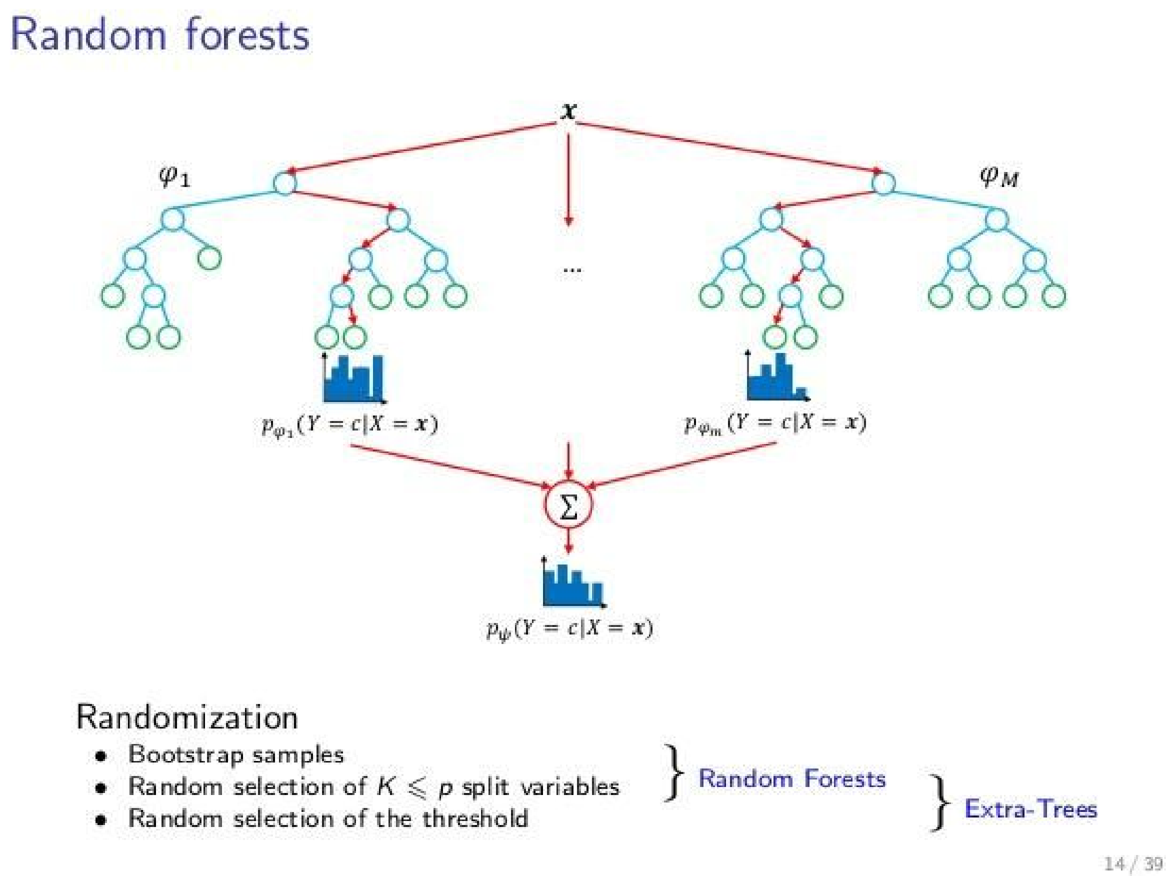
Model:

Decision Tree – A decision tree is a supervised machine learning classification algorithm that uses a tree like model of decisions and their possible outcomes. Below we can ask the first question like if checking account balance is greater than 200 and then two more questions can be derived like duration of credit greater then 29 months for partial creditability and length of employment greater than 9 months for partial non-creditability.



Random Forest:

Random forests are the best classification algorithm out of all classification algorithms and it gives the more accuracy compared to other algorithms. This algorithm is a popular ensemble learning method for both classification and regression. This model consists of multiple decision trees depending on different subsets of data at the training stage.



Source: google

Dataset:

The dataset consists of 20 attributes and a total of 1000 instances or records. Attributes include both categorical and integer values.

Below are the features of the data.

Features → {"balance", "duration", "history", "purpose", "amount", "savings", "employment", "instPercent", "sexMarried", "guarantors", "residenceDuration", "assets", "age", "concCredit", "apartment", "credits", "occupation", "dependents", "hasPhone", "foreign"}

Dataset and attribute information –

Data Set Information:

Two datasets are provided. the original dataset, in the form provided by Prof. Hofmann, contains categorical/symbolic attributes and is in the file "german.data".

For algorithms that need numerical attributes, Strathclyde University produced the file "german.data-numeric". This file has been edited and several indicator variables added to make it suitable for algorithms which cannot cope with categorical variables. Several attributes that are ordered categorical (such as attribute 17) have been coded as integer. This was the form used by StatLog.

This dataset requires use of a cost matrix (see below)

```
..... 1 2
----------------------------
1 0 1
-----------------------
2 5 0
```

(1 = Good, 2 = Bad)

The rows represent the actual classification and the columns the predicted classification.

It is worse to class a customer as good when they are bad (5), than it is to class a customer as bad when they are good (1).

Attribute Information:

Attribute 1: (qualitative)
Status of existing checking account
A11 : ... < 0 DM
A12 : 0 <= ... < 200 DM
A13 : ... >= 200 DM / salary assignments for at least 1 year
A14 : no checking account

Attribute 2: (numerical)
Duration in month

Attribute 3: (qualitative)
Credit history
A30 : no credits taken/ all credits paid back duly
A31 : all credits at this bank paid back duly
A32 : existing credits paid back duly till now
A33 : delay in paying off in the past
A34 : critical account/ other credits existing (not at this bank)

Attribute 4: (qualitative)
Purpose
A40 : car (new)
A41 : car (used)
A42 : furniture/equipment
A43 : radio/television
A44 : domestic appliances
A45 : repairs
A46 : education
A47 : (vacation - does not exist?)
A48 : retraining
A49 : business
A410 : others

Attribute 5: (numerical)
Credit amount

Attibute 6: (qualitative)
Savings account/bonds
A61 : ... < 100 DM
A62 : 100 <= ... < 500 DM

Attibute 6: (qualitative)
Savings account/bonds
A61 : ... < 100 DM
A62 : 100 <= ... < 500 DM
A63 : 500 <= ... < 1000 DM
A64 : .. >= 1000 DM
A65 : unknown/ no savings account

Attribute 7: (qualitative)
Present employment since
A71 : unemployed
A72 : ... < 1 year
A73 : 1 <= ... < 4 years
A74 : 4 <= ... < 7 years
A75 : .. >= 7 years

Attribute 8: (numerical)
Installment rate in percentage of disposable income

Attribute 9: (qualitative)
Personal status and sex
A91 : male : divorced/separated
A92 : female : divorced/separated/married
A93 : male : single
A94 : male : married/widowed
A95 : female : single

Attribute 10: (qualitative)
Other debtors / guarantors
A101 : none
A102 : co-applicant
A103 : guarantor

Attribute 11: (numerical)
Present residence since

Attribute 12: (qualitative)
Property
A121 : real estate
A122 : if not A121 : building society savings agreement/ life insurance
A123 : if not A121/A122 : car or other, not in attribute 6
A124 : unknown / no property

Attribute 13: (numerical)
Age in years

Attribute 14: (qualitative)
Other installment plans
A141 : bank
A142 : stores
A143 : none

Attribute 15: (qualitative)
Housing
A151 : rent
A152 : own
A153 : for free

Attribute 16: (numerical)
Number of existing credits at this bank

Attribute 17: (qualitative)
Job
A171 : unemployed/ unskilled - non-resident
A172 : unskilled - resident
A173 : skilled employee / official
A174 : management/ self-employed/
highly qualified employee/ officer

Attribute 18: (numerical)
Number of people being liable to provide maintenance for

Attribute 19: (qualitative)
Telephone
A191 : none
A192 : yes, registered under the customers name

Attribute 20: (qualitative)
foreign worker
A201 : yes
A202 : no

Analysis of Data:

Input data from the csv file is read as RDD and it is parsed and then converted to data frame which makes life easier to analyse. Also, we have used SparkSQL for pranalysis of data which shows the complete description about the column like mean, max and min values of that column. Finally, we have applied Machine learning models to predict whether the bank customer is credit worthy or not to give a loan.
Lectures used are Spark RDD, Spark Data Frames, SparkSQL and Spark Machine Learning.

Implementation:

Code is implemented in Spark Machine Learning using both random forest and decision tree classifiers.

Random Forest code –

```scala
import org.apache.spark._
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SQLContext
import org.apache.spark.ml.classification.RandomForestClassifier
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.tuning.{ ParamGridBuilder, CrossValidator }
import org.apache.spark.ml.{ Pipeline, PipelineStage }
import org.apache.spark.mllib.evaluation.RegressionMetrics

object Credit {


  // Creating Credit case class for holding input data structure
  case class Credit(
                   creditability: Double,
                   balance: Double, duration: Double, history: Double, purpose: Double, amount: Double,
                   savings: Double, employment: Double, instPercent: Double, sexMarried: Double, guarantors: Double,
                   residenceDuration: Double, assets: Double, age: Double, concCredit: Double, apartment: Double,
                   credits: Double, occupation: Double, dependents: Double, hasPhone: Double, foreign: Double
                  )

  // Parsing and converting all the columns to double type
  def parseCredit(line: Array[Double]): Credit = {
    Credit(
      line(0),
      line(1) - 1, line(2), line(3), line(4), line(5),
      line(6) - 1, line(7) - 1, line(8), line(9) - 1, line(10) - 1,
      line(11) - 1, line(12) - 1, line(13), line(14) - 1, line(15) - 1,
      line(16) - 1, line(17) - 1, line(18) - 1, line(19) - 1, line(20) - 1
    )
  }

  def parseRDD(rdd: RDD[String]): RDD[Array[Double]] = {
    rdd.map(_.split( regex = ",")).map(_.map(_.toDouble))
  }
```

```scala
}

def main(args: Array[String]) : Unit  {

  val conf = new SparkConf().setAppName("LoanPredictionRandomClassifier").setMaster("local")
  val sc = new SparkContext(conf)
  val sqlContext = new SQLContext(sc)

  import sqlContext.implicits._

  //Loading csv file as rdd, parsing and then converting to Dataframe
  val creditDF = parseRDD(sc.textFile( path = "src/main/scala/germancredit.csv")).map(parseCredit).toDF().cache()
  creditDF.registerTempTable( tableName = "credit")
  creditDF.printSchema

  //printing dataframe
  creditDF.show

  //calcualting avg balance, amount, duration for people who are creditable
  sqlContext.sql( sqlText = "SELECT creditability, avg(balance) as avgbalance, avg(amount) as avgamt, avg(duration) as avgdur  FROM credit GROUP BY creditability ").show

  // analysis of balance column
  creditDF.describe( cols = "balance").show
  creditDF.groupBy( col1 = "creditability").avg( colNames = "balance").show

  val featureCols = Array("balance", "duration", "history", "purpose", "amount",
    "savings", "employment", "instPercent", "sexMarried", "guarantors",
    "residenceDuration", "assets", "age", "concCredit", "apartment",
    "credits", "occupation", "dependents", "hasPhone", "foreign")
  val assembler = new VectorAssembler().setInputCols(featureCols).setOutputCol("features")
  val df2 = assembler.transform(creditDF)
  df2.show

  val labelIndexer = new StringIndexer().setInputCol("creditability").setOutputCol("label")
  val df3 = labelIndexer.fit(df2).transform(df2)
  df3.show
  val splitSeed = 5043
```

```scala
  val splitSeed = 5043
  val Array(trainingData, testData) = df3.randomSplit(Array(0.7, 0.3), splitSeed)

  //applying randomforest classifier with depth 3 and number of trees as 20
  val classifier = new RandomForestClassifier().setImpurity("gini").setMaxDepth(3).setNumTrees(20).setFeatureSubsetStrategy("auto").setSeed(5043)
  val model = classifier.fit(trainingData)

  val evaluator = new BinaryClassificationEvaluator().setLabelCol("label")
  val predictions = model.transform(testData)
  model.toDebugString

  //accuracy before adding pipeline
  val accuracy = evaluator.evaluate(predictions)
  println("accuracy before pipeline fitting" + accuracy)

  val regmet = new RegressionMetrics(
    predictions.select( col = "prediction",  cols = "label").rdd.map(x =>
      (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double]))
  )

  //printing all the errors and variance
  println("MeanSquErr: " + regmet.meanSquaredError)
  println("MeanAbsolError: " + regmet.meanAbsoluteError)
  println("RootMeanSqrErr Squared: " + regmet.rootMeanSquaredError)
  println("R Squared: " + regmet.r2)
  println("Explained Variance: " + regmet.explainedVariance + "\n")

  val paramGrid = new ParamGridBuilder()
    .addGrid(classifier.maxBins, Array(25, 31))
    .addGrid(classifier.maxDepth, Array(5, 10))
    .addGrid(classifier.numTrees, Array(20, 60))
    .addGrid(classifier.impurity, Array("entropy", "gini"))
    .build()

  val steps: Array[PipelineStage] = Array(classifier)
  val pipeline = new Pipeline().setStages(steps)
```

```
val steps: Array[PipelineStage] = Array(classifier)
val pipeline = new Pipeline().setStages(steps)

val cv = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(evaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(10)

val pipelineFittedModel = cv.fit(trainingData)

//accuracy after pipeline fitting
val predictions2 = pipelineFittedModel.transform(testData)
val accuracy2 = evaluator.evaluate(predictions2)
println("accuracy after pipeline fitting" + accuracy2)

println(pipelineFittedModel.bestModel.asInstanceOf[org.apache.spark.ml.PipelineModel].stages(0))

pipelineFittedModel
  .bestModel.asInstanceOf[org.apache.spark.ml.PipelineModel]
  .stages(0)
  .extractParamMap

val regmet2 = new RegressionMetrics(
  predictions2.select( col = "prediction",  cols = "label").rdd.map(x =>
    (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double]))
)

//printing all the errors and variance
println("MeanSquErr: " + regmet2.meanSquaredError)
println("MeanAbsolError: " + regmet2.meanAbsoluteError)
println("RootMeanSqrErr Squared: " + regmet2.rootMeanSquaredError)
println("R Squared: " + regmet2.r2)
println("Explained Variance: " + regmet2.explainedVariance + "\n")

}
}
```

Decision Tree Code –

```
  "savings", "employment", "instPercent", "sexMarried", "guarantors",
  "residenceDuration", "assets", "age", "concCredit", "apartment",
  "credits", "occupation", "dependents", "hasPhone", "foreign")
val assembler = new VectorAssembler().setInputCols(featureCols).setOutputCol("features")
val df2 = assembler.transform(creditDF)
df2.show

val labelIndexer = new StringIndexer().setInputCol("creditability").setOutputCol("label")
val df3 = labelIndexer.fit(df2).transform(df2)
df3.show
val splitSeed = 5043
val Array(trainingData, testData) = df3.randomSplit(Array(0.7, 0.3), splitSeed)

//applying Decision Tree classifier with depth 3
val classifier = new DecisionTreeClassifier().setImpurity("gini").setMaxDepth(3).setSeed(5043)
val model = classifier.fit(trainingData)

val evaluator = new BinaryClassificationEvaluator().setLabelCol("label")
val predictions = model.transform(testData)
model.toDebugString

//accuracy before adding pipeline
val accuracy = evaluator.evaluate(predictions)
println("accuracy before pipeline fitting" + accuracy)

val regmet = new RegressionMetrics(
  predictions.select( col = "prediction",  cols = "label").rdd.map(x =>
    (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double]))
)

//printing all the errors and variance
println("MeanSquErr: " + regmet.meanSquaredError)
println("MeanAbsolError: " + regmet.meanAbsoluteError)
println("RootMeanSqrErr Squared: " + regmet.rootMeanSquaredError)
println("R Squared: " + regmet.r2)
println("Explained Variance: " + regmet.explainedVariance + "\n")
```

Results Evaluation:

By comparing both the models, we can say that Random Forest model is the best of two, which has the highest accuracy among them and less error compared to the other model.

Random Forest output –

Credit × Creditd ×

```
18/12/05 10:16:40 INFO BlockManagerMasterEndpoint: Registering block mai
18/12/05 10:16:40 INFO BlockManagerMaster: Registered BlockManager Bloc
18/12/05 10:16:40 INFO BlockManager: Initialized BlockManager: BlockMan
root
 |-- creditability: double (nullable = true)
 |-- balance: double (nullable = true)
 |-- duration: double (nullable = true)
 |-- history: double (nullable = true)
 |-- purpose: double (nullable = true)
 |-- amount: double (nullable = true)
 |-- savings: double (nullable = true)
 |-- employment: double (nullable = true)
 |-- instPercent: double (nullable = true)
 |-- sexMarried: double (nullable = true)
 |-- guarantors: double (nullable = true)
 |-- residenceDuration: double (nullable = true)
 |-- assets: double (nullable = true)
 |-- age: double (nullable = true)
 |-- concCredit: double (nullable = true)
 |-- apartment: double (nullable = true)
 |-- credits: double (nullable = true)
 |-- occupation: double (nullable = true)
 |-- dependents: double (nullable = true)
 |-- hasPhone: double (nullable = true)
 |-- foreign: double (nullable = true)
```

```
|-- foreign: double (nullable = true)
```

creditability	balance	duration	history	purpose	amount	savings	employment	instPercent	sexMarried	guarantors	residenceDuration	assets	age	concCredit	apartment	credits	occupation	dependents	hasPhone
1.0	0.0	18.0	4.0	2.0	1049.0	0.0	1.0	4.0	1.0	0.0	3.0	1.0	21.0	2.0	0.0	0.0	2.0	0.0	0.0
1.0	0.0	9.0	4.0	0.0	2799.0	0.0	2.0	2.0	2.0	0.0	1.0	0.0	36.0	2.0	0.0	1.0	2.0	1.0	0.0
1.0	1.0	12.0	2.0	9.0	841.0	1.0	3.0	2.0	1.0	0.0	3.0	0.0	23.0	2.0	0.0	0.0	1.0	0.0	0.0
1.0	0.0	12.0	4.0	0.0	2122.0	0.0	2.0	3.0	2.0	0.0	1.0	0.0	39.0	2.0	0.0	1.0	1.0	1.0	0.0
1.0	0.0	12.0	4.0	0.0	2171.0	0.0	2.0	4.0	2.0	0.0	3.0	1.0	38.0	0.0	1.0	1.0	1.0	0.0	0.0
1.0	0.0	10.0	4.0	0.0	2241.0	0.0	1.0	1.0	2.0	0.0	2.0	0.0	48.0	2.0	0.0	1.0	1.0	1.0	0.0
1.0	0.0	8.0	4.0	0.0	3398.0	0.0	3.0	1.0	2.0	0.0	3.0	0.0	39.0	2.0	1.0	1.0	1.0	0.0	0.0
1.0	0.0	6.0	4.0	0.0	1361.0	0.0	1.0	2.0	2.0	0.0	3.0	0.0	40.0	2.0	1.0	0.0	1.0	1.0	0.0
1.0	3.0	18.0	4.0	3.0	1098.0	0.0	0.0	4.0	1.0	0.0	3.0	2.0	65.0	2.0	1.0	1.0	0.0	0.0	0.0
1.0	1.0	24.0	2.0	3.0	3758.0	2.0	0.0	1.0	1.0	0.0	3.0	3.0	23.0	2.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	11.0	4.0	0.0	3905.0	0.0	2.0	2.0	2.0	0.0	1.0	0.0	36.0	2.0	0.0	1.0	2.0	1.0	0.0
1.0	0.0	30.0	4.0	1.0	6187.0	1.0	3.0	1.0	3.0	0.0	3.0	2.0	24.0	2.0	0.0	1.0	2.0	0.0	0.0
1.0	0.0	6.0	4.0	3.0	1957.0	0.0	3.0	1.0	1.0	0.0	3.0	2.0	31.0	2.0	1.0	0.0	2.0	0.0	0.0
1.0	1.0	48.0	3.0	10.0	7582.0	1.0	0.0	2.0	2.0	0.0	3.0	3.0	31.0	2.0	1.0	0.0	3.0	0.0	1.0
1.0	0.0	18.0	2.0	3.0	1936.0	4.0	3.0	2.0	3.0	0.0	3.0	2.0	23.0	2.0	0.0	1.0	1.0	0.0	0.0
1.0	0.0	6.0	2.0	3.0	2647.0	2.0	2.0	2.0	2.0	0.0	2.0	0.0	44.0	2.0	0.0	0.0	2.0	1.0	0.0
1.0	0.0	11.0	4.0	0.0	3939.0	0.0	2.0	1.0	2.0	0.0	1.0	0.0	40.0	2.0	1.0	1.0	1.0	1.0	0.0
1.0	1.0	18.0	2.0	3.0	3213.0	2.0	1.0	1.0	3.0	0.0	2.0	0.0	25.0	2.0	0.0	0.0	2.0	0.0	0.0
1.0	1.0	36.0	4.0	3.0	2337.0	0.0	4.0	4.0	2.0	0.0	3.0	0.0	36.0	2.0	1.0	0.0	2.0	0.0	0.0
1.0	3.0	11.0	4.0	0.0	7228.0	0.0	2.0	1.0	2.0	0.0	3.0	1.0	39.0	2.0	1.0	1.0	1.0	0.0	0.0

only showing top 20 rows

```
|          1.0|    3.0|    11.0|    4.0|    0.0|7228.0|    0.0|       2.0|        1.0|       2.0|       0
+-------------+-------+--------+-------+-------+------+-------+----------+-----------+----------+--------
only showing top 20 rows

+-------------+------------------+------------------+------------------+
|creditability|        avgbalance|            avgamt|            avgdur|
+-------------+------------------+------------------+------------------+
|          0.0|0.9033333333333333|3938.1266666666666|             24.86|
|          1.0|1.8657142857142857| 2985.442857142857|19.207142857142856|
+-------------+------------------+------------------+------------------+

+-------+------------------+
|summary|           balance|
+-------+------------------+
|  count|              1000|
|   mean|             1.577|
| stddev|1.2576377271108938|
|    min|               0.0|
|    max|               3.0|
+-------+------------------+

+-------------+------------------+
|creditability|      avg(balance)|
+-------------+------------------+
|          0.0|0.9033333333333333|
|          1.0|1.8657142857142857|
+-------------+------------------+

+-------------+-------+--------+-------+-------+------+-------+----------+-----------+----------+--------
|creditability|balance|duration|history|purpose|amount|savings|employment|instPercent|sexMarried|guaranto
+-------------+-------+--------+-------+-------+------+-------+----------+-----------+----------+--------
|          1.0|    0.0|    18.0|    4.0|    2.0|1049.0|    0.0|       1.0|        4.0|       1.0|       0
|          1.0|    0.0|     9.0|    4.0|    0.0|2799.0|    0.0|       2.0|        2.0|       2.0|       0
```

```
|          1.0|     1.0|    24.0|     2.0|     3.0|3758.0|     2.0|        0.0|        1.
|          1.0|     0.0|    11.0|     4.0|     0.0|3905.0|     0.0|        2.0|        2.
|          1.0|     0.0|    30.0|     4.0|     1.0|6187.0|     1.0|        3.0|        1.
|          1.0|     0.0|     6.0|     4.0|     3.0|1957.0|     0.0|        3.0|        1.
|          1.0|     1.0|    48.0|     3.0|    10.0|7582.0|     1.0|        0.0|        2.
|          1.0|     0.0|    18.0|     2.0|     3.0|1936.0|     4.0|        3.0|        2.
|          1.0|     0.0|     6.0|     2.0|     3.0|2647.0|     2.0|        2.0|        2.
|          1.0|     0.0|    11.0|     4.0|     0.0|3939.0|     0.0|        2.0|        1.
|          1.0|     1.0|    18.0|     2.0|     3.0|3213.0|     2.0|        1.0|        1.
|          1.0|     1.0|    36.0|     4.0|     3.0|2337.0|     0.0|        4.0|        4.
|          1.0|     3.0|    11.0|     4.0|     0.0|7228.0|     0.0|        2.0|        1.
+-------------+-------+--------+-------+-------+------+-------+----------+----------
only showing top 20 rows

accuracy before pipeline fitting0.7266118836915278
MeanSquErr: 0.22442244224422442
MeanAbsolError: 0.22442244224422442
RootMeanSqrErr Squared: 0.47373245850820106
R Squared: -0.1840018388690956
Explained Variance: 0.09866135128364424

accuracy after pipeline fitting0.7518101367658875
RandomForestClassificationModel (uid=rfc_b41496906028) with 60 trees
MeanSquErr: 0.24092409240924087
MeanAbsolError: 0.24092409240924093
RootMeanSqrErr Squared: 0.4908401902954167
R Squared: -0.2710607976094699
Explained Variance: 0.1554640612576106


Process finished with exit code 0
```

Decision Tree output –

```
Credit   Creditd
|            1.0|   3.0|   18.0|   4.0|    3.0|1098.0|   0.0|       0.0|        4.0|
|            1.0|   1.0|   24.0|   2.0|    3.0|3758.0|   2.0|       0.0|        1.0|
|            1.0|   0.0|   11.0|   4.0|    0.0|3905.0|   0.0|       2.0|        2.0|
|            1.0|   0.0|   30.0|   4.0|    1.0|6187.0|   1.0|       3.0|        1.0|
|            1.0|   0.0|    6.0|   4.0|    3.0|1957.0|   0.0|       3.0|        1.0|
|            1.0|   1.0|   48.0|   3.0|   10.0|7582.0|   1.0|       0.0|        2.0|
|            1.0|   0.0|   18.0|   2.0|    3.0|1936.0|   4.0|       3.0|        2.0|
|            1.0|   0.0|    6.0|   2.0|    3.0|2647.0|   2.0|       2.0|        2.0|
|            1.0|   0.0|   11.0|   4.0|    0.0|3939.0|   0.0|       2.0|        1.0|
|            1.0|   1.0|   18.0|   2.0|    3.0|3213.0|   2.0|       1.0|        1.0|
|            1.0|   1.0|   36.0|   4.0|    3.0|2337.0|   0.0|       4.0|        4.0|
|            1.0|   3.0|   11.0|   4.0|    0.0|7228.0|   0.0|       2.0|        1.0|
+------------+-------+--------+-------+-------+------+-------+----------+-----------+------
only showing top 20 rows

accuracy before pipeline fitting0.6287208366854385
MeanSquErr: 0.30033003300330036
MeanAbsolError: 0.30033003300330036
RootMeanSqrErr Squared: 0.5480237522254855
R Squared: -0.5844730490748189
Explained Variance: 0.13274297726802356

accuracy after pipeline fitting0.6469084013331801
DecisionTreeClassificationModel (uid=dtc_2a017a6949bf) of depth 5 with 53 nodes
MeanSquErr: 0.3168316831683168
MeanAbsolError: 0.31683168316831684
RootMeanSqrErr Squared: 0.5628780357842335
R Squared: -0.6715320078151934
Explained Variance: 0.16682460325240392


Process finished with exit code 0
```

Conclusion:

In conclusion Random Forest gives better results than the decision tree. Also, by using ML Pipeline gives higher accuracy percentage than without the ML Pipeline. A pipeline gives a basic method to experiment with various mixes of parameters, utilizing a procedure called network look, where you set up the parameters to test, and MLLib will test every one of the mixes. Pipelines make it simple to tune a whole model building work process without a moment's delay, instead of tuning every component in the Pipeline independently.

Project Management:

Implementation status report

Work completed: 100%

Description:

German bank loan Dataset has been loaded and format and feature has been identified/extraction.

Applied ML models on the identified dataset, calculated accuracy of the models and tested them.

Responsibility (Task, Person)

Downloading and analysing Dataset: Chandra sekhar Pentakota

Worked on Random Forest: Chandra sekhar Pentakota

Worked on Decision Tree: Bilal Mustafa

Contributions (members/percentage):

Chandra sekhar Pentakota 50%

Bilal Mustafa 50%

Responsibility (Task, Person)

Apache Spark Setup / Chandra sekhar Pentakota

Reading Dataset and converting to Data Frame / Bilal Mustafa

Extract Features / Bilal Mustafa

Spark Analysis generating result / Chandra sekhar Pentakota

Training and Testing Model / Chandra sekhar Pentakota

Issues/Concerns:

None

References/Bibliography:

https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

https://mapr.com/blog/predicting-loan-credit-risk-using-apache-spark-machine-learning-random-forests/