Mortgage Approval Prediction System

Authors:
Chetan Bhanushali
Dhruv Prajapati

Table of Contents

1) About Dataset

The Home Mortgage Disclosure Act (HMDA) data provides detailed information about mortgage applications, originations, and denials in the United States. The 2020 HMDA dataset contains public data submitted by financial institutions to ensure transparency and compliance with fair lending laws.
We have used data from the year 2020 which has 25 million records and is 10 GB in size.

Source = https://ffiec.cfpb.gov/data-publication/2020

Link = gs://cs777-final-project/2020_lar.txt

1.1) Purpose of HMDA Data

- Ensure lenders serve the needs of their communities.
- Assist in identifying discriminatory lending practices.
- Provide public officials with insights for policy-making.

1.2) Key Data Elements

- Loan Information:
    - Loan purpose (e.g., purchase, refinance).
    - Loan amount.
    - Type of loan (conventional, FHA, VA).
- Applicant Information:
    - Income.
    - Ethnicity, race, and sex.
    - Credit characteristics.
- Property Information:
    - Location (e.g., census tract, county, state).
    - Type of property (e.g., single-family, multi-family).
- Action Taken:
    - Whether the loan was originated, denied, or withdrawn.
- Pricing Information:

- Interest rates, total loan costs, and discount points.

2) Data Preprocessing

Data Preprocessing is the first most important step for building a reliable model. Data can have missing values, improper data types and outliers.

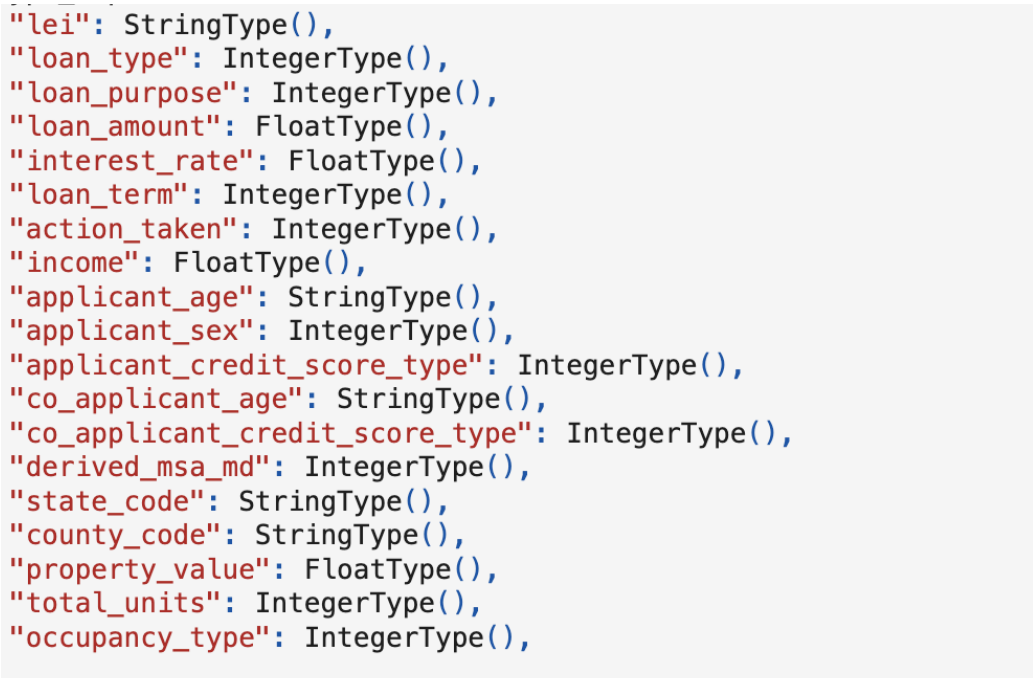
Our dataset has missing values and improper data type for some columns, we will preprocess the data before we feed it to the model.

2.1) Feature Selection

The original dataset has 25 million rows and 99 columns. Our goal for the project is to classify Loan Approval Trends depending on Loan Details , Borrower Details and Property Details. So the final dataset comes down to 25 million rows and 19 columns.

2.2) Handle Data Types

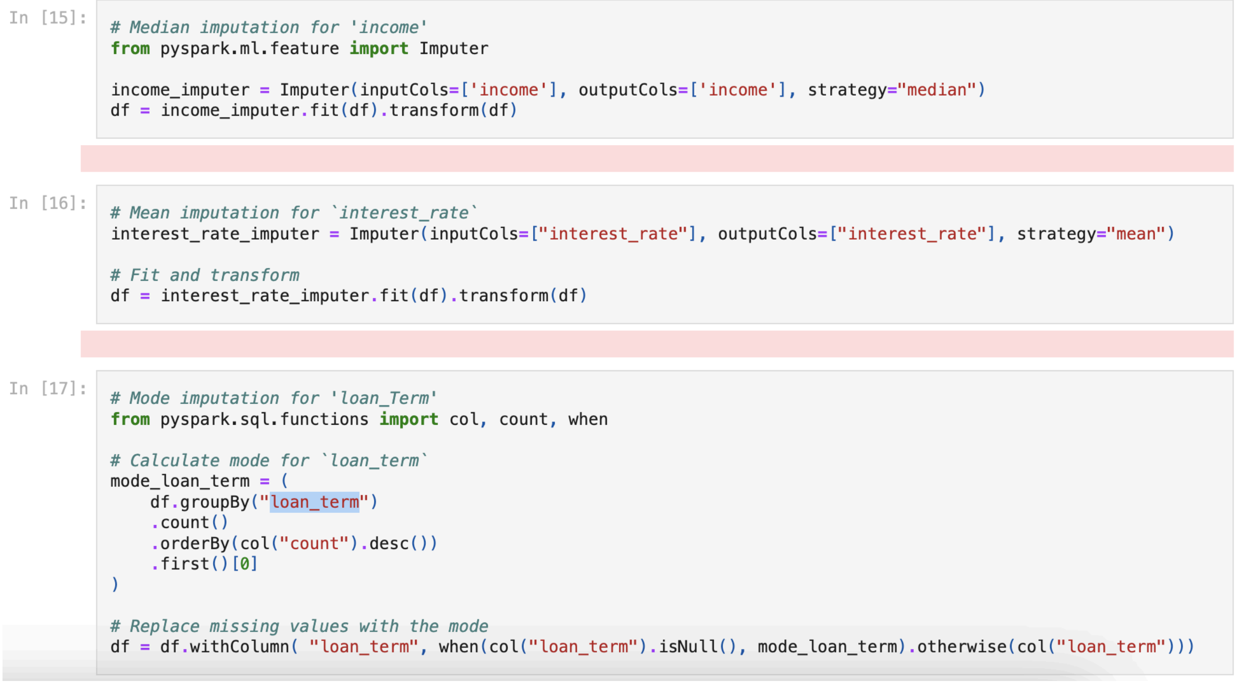
Initially all the columns data type is ‘string’, for computational reasons we have to change the data type of each column to what they belong to. Example: Income column cannot be a string but should be a float.



```
"lei": StringType(),
"loan_type": IntegerType(),
"loan_purpose": IntegerType(),
"loan_amount": FloatType(),
"interest_rate": FloatType(),
"loan_term": IntegerType(),
"action_taken": IntegerType(),
"income": FloatType(),
"applicant_age": StringType(),
"applicant_sex": IntegerType(),
"applicant_credit_score_type": IntegerType(),
"co_applicant_age": StringType(),
"co_applicant_credit_score_type": IntegerType(),
"derived_msa_md": IntegerType(),
"state_code": StringType(),
"county_code": StringType(),
"property_value": FloatType(),
"total_units": IntegerType(),
"occupancy_type": IntegerType(),
```

2.3) Handle NULL & Missing Values

To begin with, we will impute values for three important columns i.e., income, interest_rate and loan_term.Impute missing values with the mean, median or mode.



In [15]:

```
# Median imputation for 'income'
from pyspark.ml.feature import Imputer

income_imputer = Imputer(inputCols=['income'], outputCols=['income'], strategy="median")
df = income_imputer.fit(df).transform(df)
```

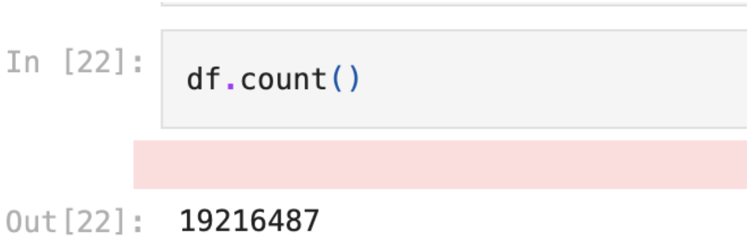
In [16]:

```
# Mean imputation for `interest_rate`
interest_rate_imputer = Imputer(inputCols=["interest_rate"], outputCols=["interest_rate"], strategy="mean")

# Fit and transform
df = interest_rate_imputer.fit(df).transform(df)
```

In [17]:

```
# Mode imputation for 'loan_Term'
from pyspark.sql.functions import col, count, when

# Calculate mode for `loan_term`
mode_loan_term = (
    df.groupBy("loan_term")
    .count()
    .orderBy(col("count").desc())
    .first()[0]
)

# Replace missing values with the mode
df = df.withColumn( "loan_term", when(col("loan_term").isNull(), mode_loan_term).otherwise(col("loan_term")))
```

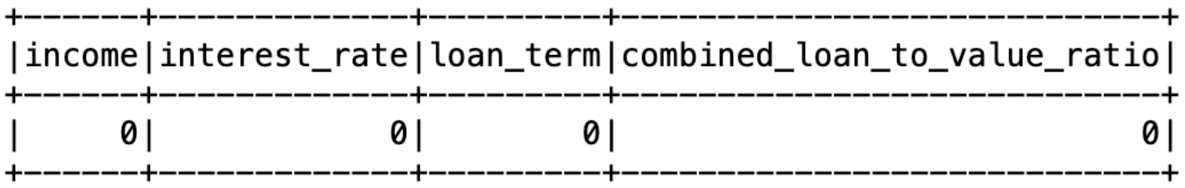
Now, we will drop rows which have no values in a subset of any selected columns. These columns are : property_value, state_code, country_code, total_units. After dropping the null values we are left with 19 million rows.



```
In [22]: df.count()

Out[22]: 19216487
```

As we can see in the above image there are 3 columns which have missing values. All these columns are numeric columns. Therefore we will replace the missing values with imputed values using the imputer function from Spark ML library.

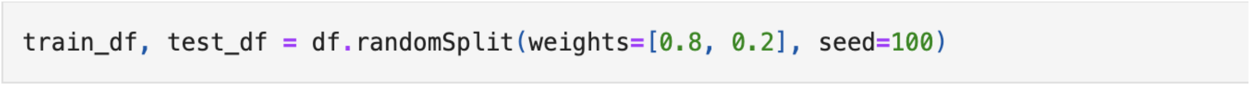
After handling missing values:



```
+------+-------------+---------+----------------------------+
|income|interest_rate|loan_term|combined_loan_to_value_ratio|
+------+-------------+---------+----------------------------+
|     0|            0|        0|                           0|
+------+-------------+---------+----------------------------+
```

3) Data Train Test Split

We will split the data into training and testing using ‘randomSplit’ in the ratio of 80% and 20% respectively.



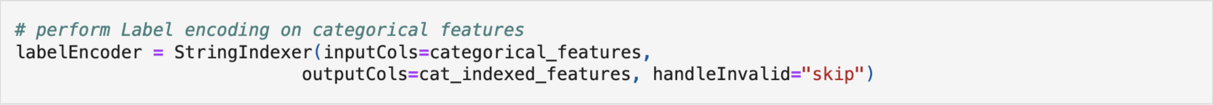
```
train_df, test_df = df.randomSplit(weights=[0.8, 0.2], seed=100)
```

4) Feature Transformation

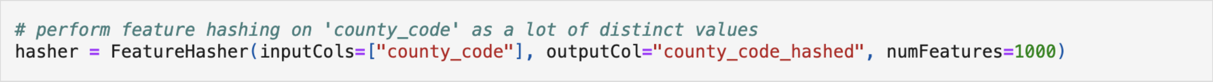
4.1) Encode Categorical Columns

We will encode categorical columns using the Pyspark ML library.

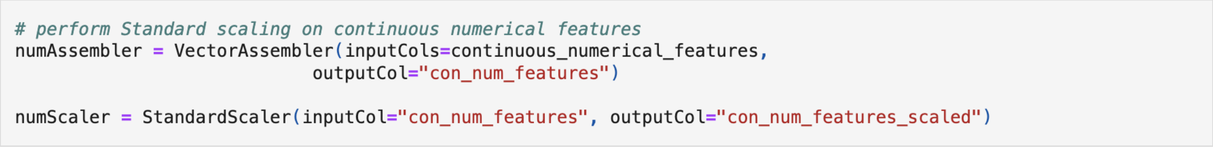
- We will perform label encoding on categorical features
- Next, we will perform scaling on continuous numerical features



```
# perform Label encoding on categorical features
labelEncoder = StringIndexer(inputCols=categorical_features,
                             outputCols=cat_indexed_features, handleInvalid="skip")
```



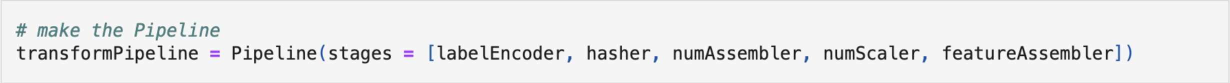
```
# perform feature hashing on 'county_code' as a lot of distinct values
hasher = FeatureHasher(inputCols=["county_code"], outputCol="county_code_hashed", numFeatures=1000)
```



```
# perform Standard scaling on continuous numerical features
numAssembler = VectorAssembler(inputCols=continuous_numerical_features,
                              outputCol="con_num_features")

numScaler = StandardScaler(inputCol="con_num_features", outputCol="con_num_features_scaled")
```

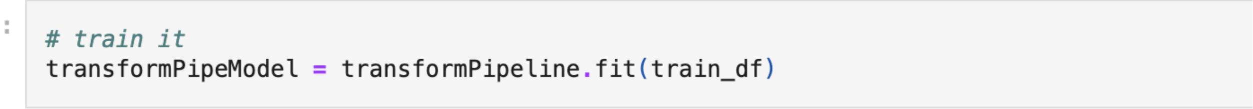
4.2) Scaling & Transformation

- Finally, assemble all the features together and build a pipeline



```
# make the Pipeline
transformPipeline = Pipeline(stages = [labelEncoder, hasher, numAssembler, numScaler, featureAssembler])
```

- Now, we train the training data set using pipeline



```
# train it
transformPipeModel = transformPipeline.fit(train_df)
```

- Finally, we apply transformation to the trained dataset.

5) Build Model & Evaluate

5.1) Choose a Predictive Model

Following are the models we used for our project:

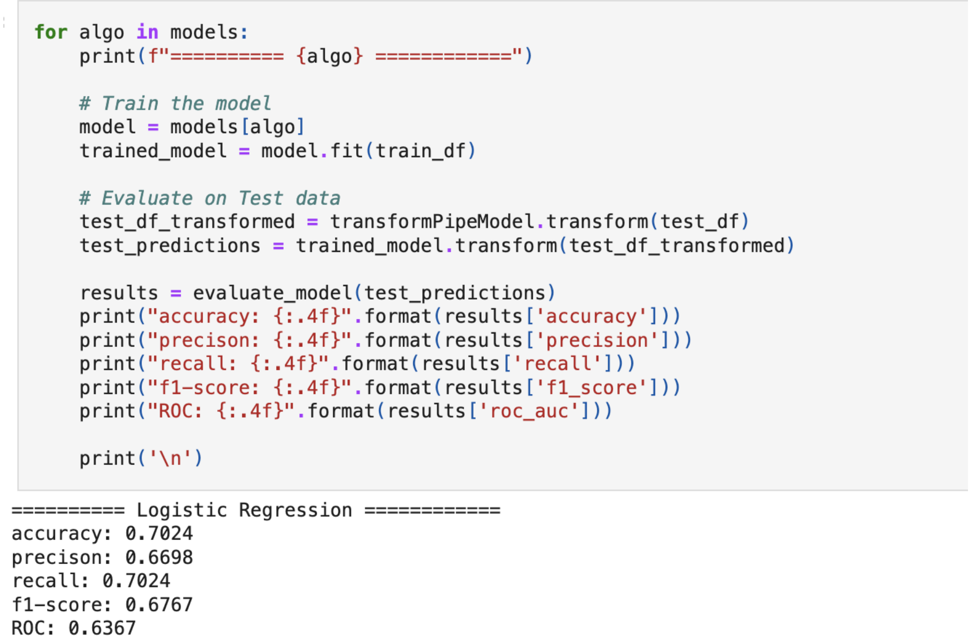
- Logistic Regression
- Linear SVM
- Factorization Machine
- Decision Tree Classifier
- Random Forest Classifier
- Gradient Boosting Trees

5.2) Evaluate the Model

To evaluate the model we run the model on test dataset.

Parameters to evaluate the models are as below:

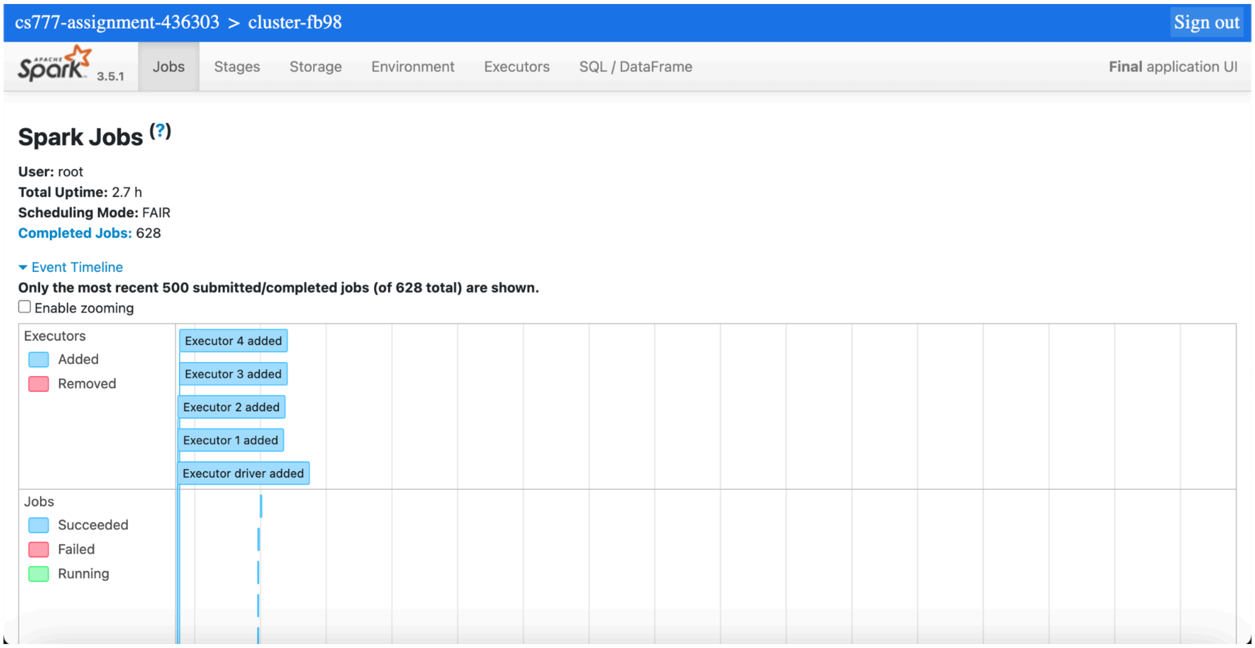
- Accuracy
- Precision
- Recall
- F1-score
- ROC



```
for algo in models:
    print(f"========== {algo} ============")

    # Train the model
    model = models[algo]
    trained_model = model.fit(train_df)

    # Evaluate on Test data
    test_df_transformed = transformPipeModel.transform(test_df)
    test_predictions = trained_model.transform(test_df_transformed)

    results = evaluate_model(test_predictions)
    print("accuracy: {:.4f}".format(results['accuracy']))
    print("precison: {:.4f}".format(results['precision']))
    print("recall: {:.4f}".format(results['recall']))
    print("f1-score: {:.4f}".format(results['f1_score']))
    print("ROC: {:.4f}".format(results['roc_auc']))

    print('\n')
```

```
========== Logistic Regression ============
accuracy: 0.7024
precison: 0.6698
recall: 0.7024
f1-score: 0.6767
ROC: 0.6367
```

Spark Server History



cs777-assignment-436303 > cluster-fb98 Sign out

Spark 3.5.1 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **Final** application UI

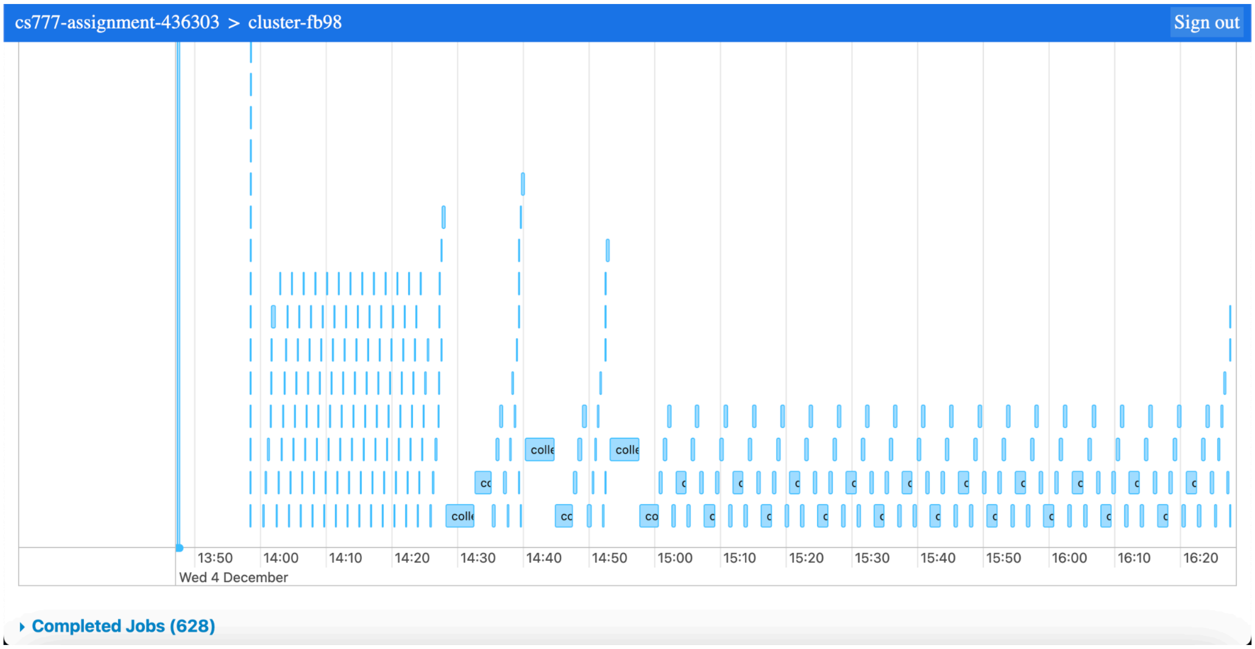
Spark Jobs (?)

User: root
Total Uptime: 2.7 h
Scheduling Mode: FAIR
Completed Jobs: 628

▾ Event Timeline

Only the most recent 500 submitted/completed jobs (of 628 total) are shown.

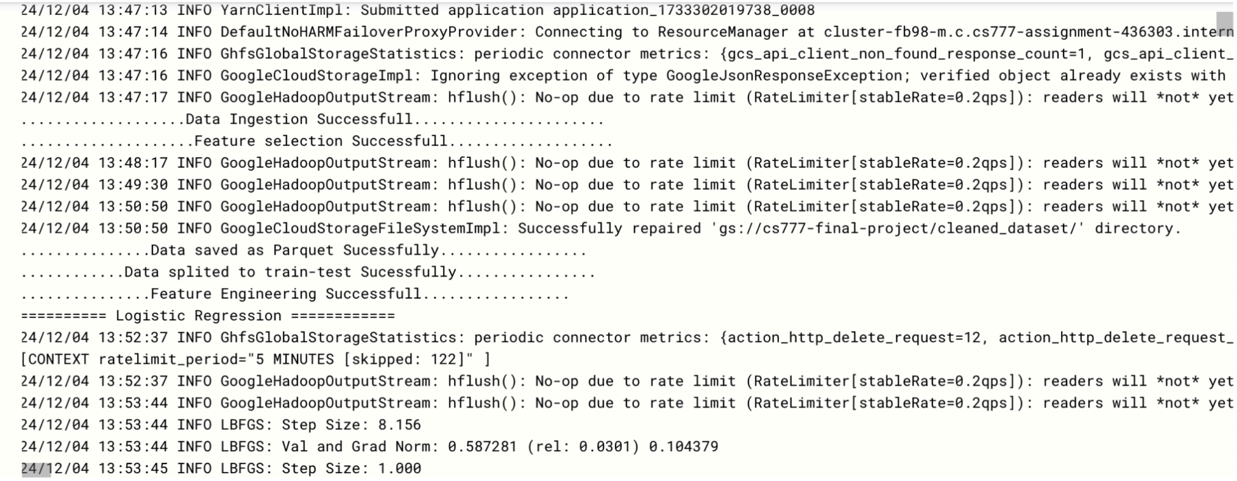
□ Enable zooming



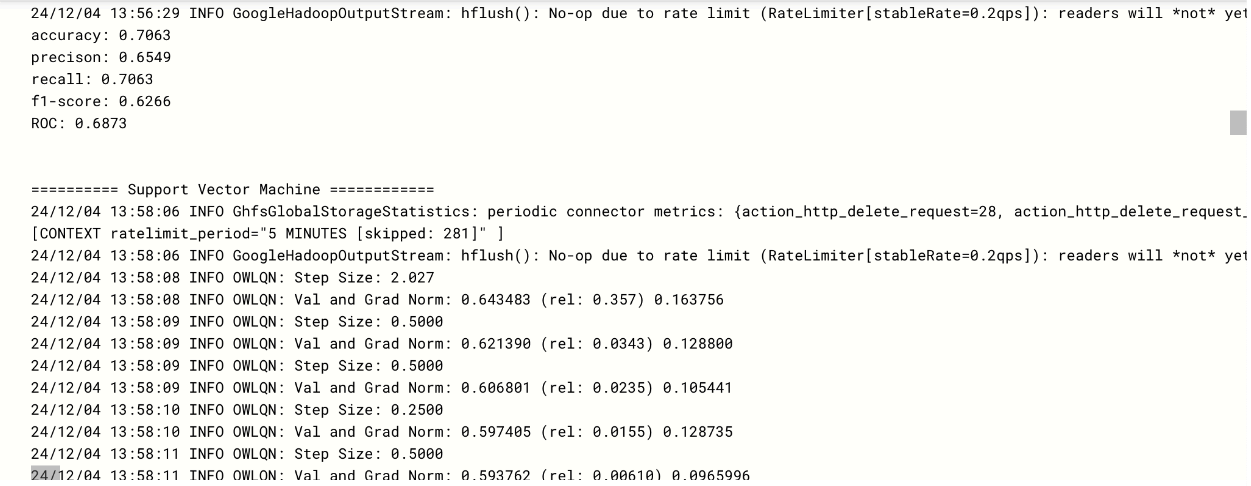
cs777-assignment-436303 > cluster-fb98 Sign out

▸ Completed Jobs (628)

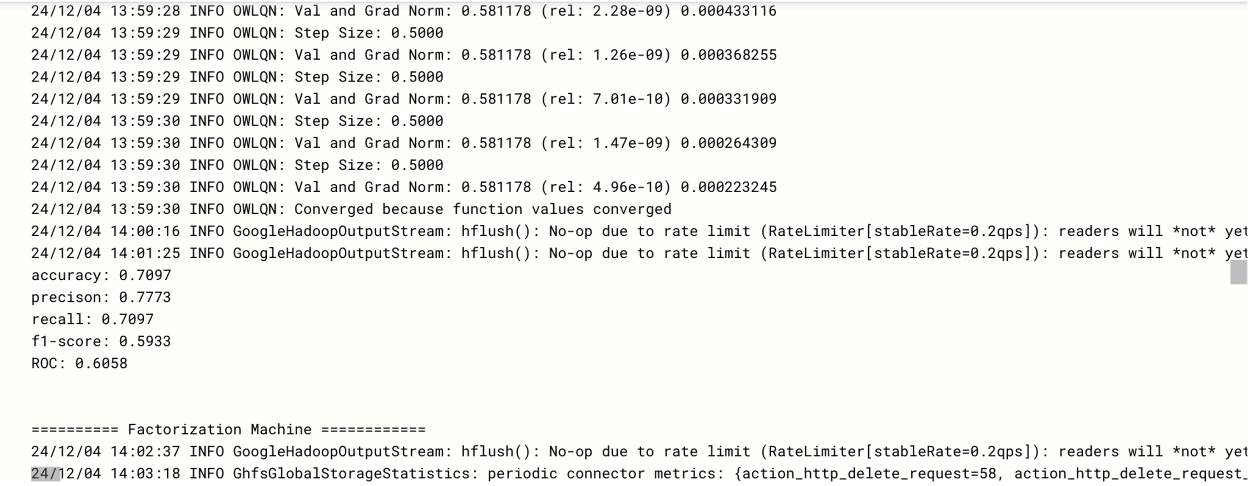
Results:



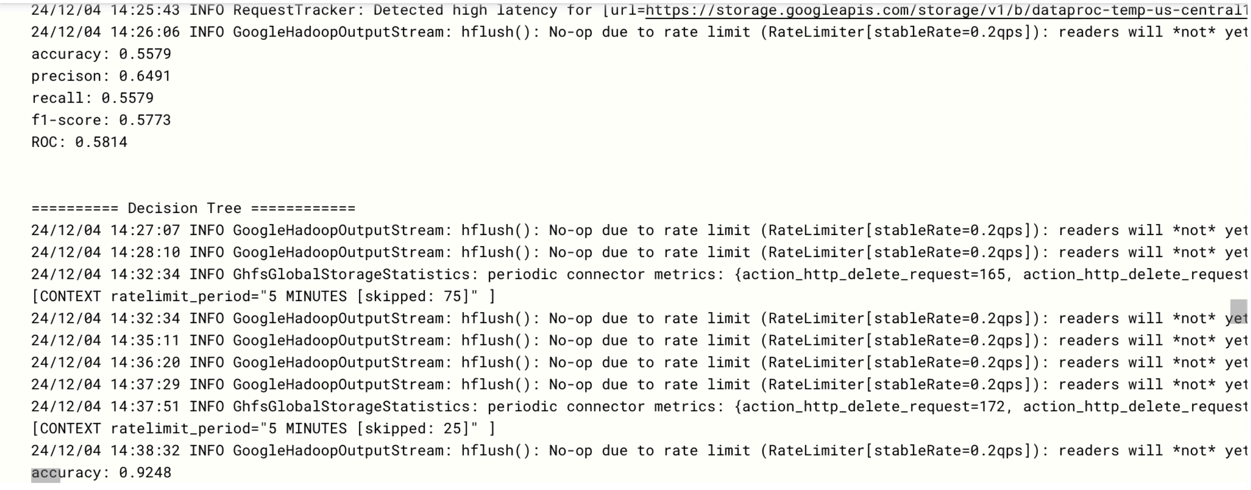
```
24/12/04 13:47:13 INFO YarnClientImpl: Submitted application application_1733302019738_0008
24/12/04 13:47:14 INFO DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at cluster-fb98-m.c.cs777-assignment-436303.intern
24/12/04 13:47:16 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {gcs_api_client_non_found_response_count=1, gcs_api_client_
24/12/04 13:47:16 INFO GoogleCloudStorageImpl: Ignoring exception of type GoogleJsonResponseException; verified object already exists with
24/12/04 13:47:17 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
...................Data Ingestion Successfull......................
....................Feature selection Successfull...................
24/12/04 13:48:17 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 13:49:30 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 13:50:50 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 13:50:50 INFO GoogleCloudStorageFileSystemImpl: Successfully repaired 'gs://cs777-final-project/cleaned_dataset/' directory.
...............Data saved as Parquet Sucessfully.................
............Data splited to train-test Sucessfully................
...............Feature Engineering Successfull.................
========== Logistic Regression ============
24/12/04 13:52:37 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=12, action_http_delete_request_
[CONTEXT ratelimit_period="5 MINUTES [skipped: 122]" ]
24/12/04 13:52:37 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 13:53:44 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 13:53:44 INFO LBFGS: Step Size: 8.156
24/12/04 13:53:44 INFO LBFGS: Val and Grad Norm: 0.587281 (rel: 0.0301) 0.104379
24/12/04 13:53:45 INFO LBFGS: Step Size: 1.000
```



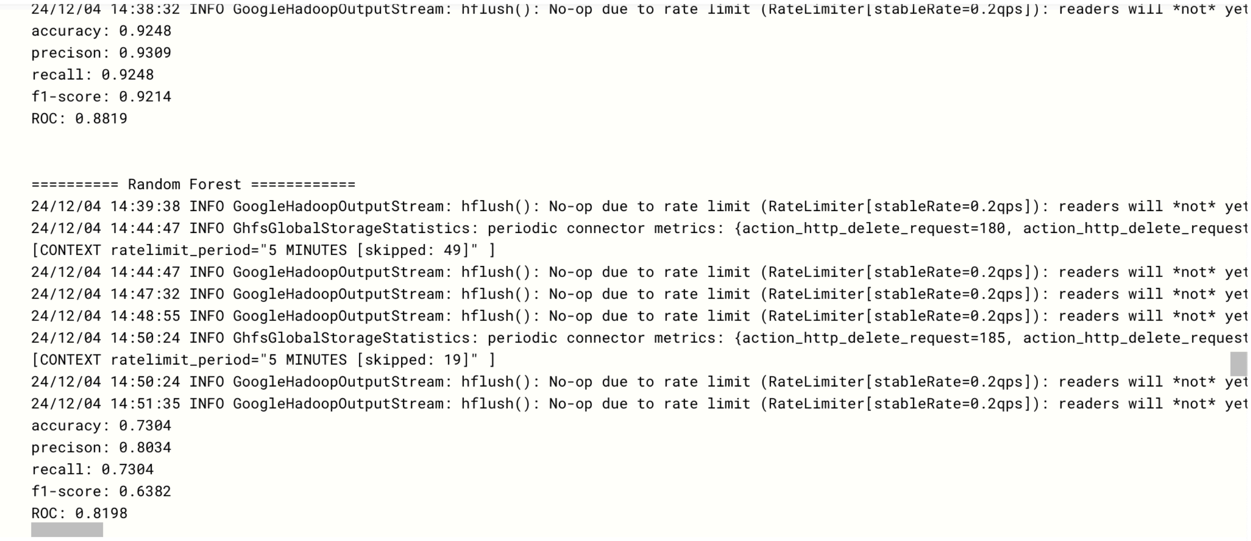
```
24/12/04 13:56:29 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
accuracy: 0.7063
precison: 0.6549
recall: 0.7063
f1-score: 0.6266
ROC: 0.6873


========== Support Vector Machine ============
24/12/04 13:58:06 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=28, action_http_delete_request_
[CONTEXT ratelimit_period="5 MINUTES [skipped: 281]" ]
24/12/04 13:58:06 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 13:58:08 INFO OWLQN: Step Size: 2.027
24/12/04 13:58:08 INFO OWLQN: Val and Grad Norm: 0.643483 (rel: 0.357) 0.163756
24/12/04 13:58:09 INFO OWLQN: Step Size: 0.5000
24/12/04 13:58:09 INFO OWLQN: Val and Grad Norm: 0.621390 (rel: 0.0343) 0.128800
24/12/04 13:58:09 INFO OWLQN: Step Size: 0.5000
24/12/04 13:58:09 INFO OWLQN: Val and Grad Norm: 0.606801 (rel: 0.0235) 0.105441
24/12/04 13:58:10 INFO OWLQN: Step Size: 0.2500
24/12/04 13:58:10 INFO OWLQN: Val and Grad Norm: 0.597405 (rel: 0.0155) 0.128735
24/12/04 13:58:11 INFO OWLQN: Step Size: 0.5000
24/12/04 13:58:11 INFO OWLQN: Val and Grad Norm: 0.593762 (rel: 0.00610) 0.0965996
```



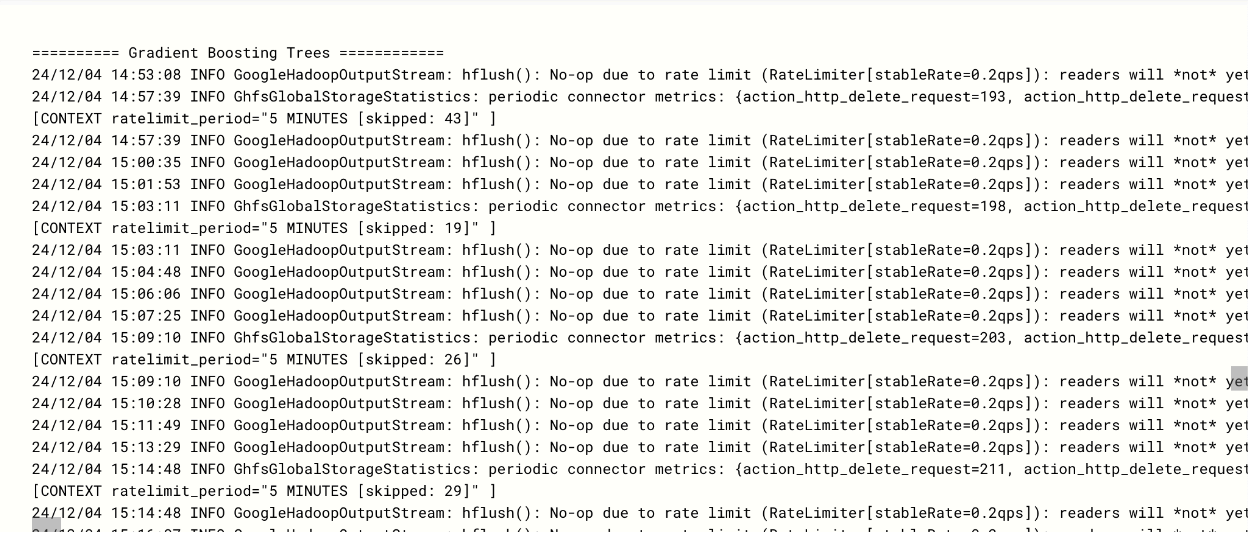
```
24/12/04 13:59:28 INFO OWLQN: Val and Grad Norm: 0.581178 (rel: 2.28e-09) 0.000433116
24/12/04 13:59:29 INFO OWLQN: Step Size: 0.5000
24/12/04 13:59:29 INFO OWLQN: Val and Grad Norm: 0.581178 (rel: 1.26e-09) 0.000368255
24/12/04 13:59:29 INFO OWLQN: Step Size: 0.5000
24/12/04 13:59:29 INFO OWLQN: Val and Grad Norm: 0.581178 (rel: 7.01e-10) 0.000331909
24/12/04 13:59:30 INFO OWLQN: Step Size: 0.5000
24/12/04 13:59:30 INFO OWLQN: Val and Grad Norm: 0.581178 (rel: 1.47e-09) 0.000264309
24/12/04 13:59:30 INFO OWLQN: Step Size: 0.5000
24/12/04 13:59:30 INFO OWLQN: Val and Grad Norm: 0.581178 (rel: 4.96e-10) 0.000223245
24/12/04 13:59:30 INFO OWLQN: Converged because function values converged
24/12/04 14:00:16 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:01:25 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
accuracy: 0.7097
precison: 0.7773
recall: 0.7097
f1-score: 0.5933
ROC: 0.6058


========== Factorization Machine ============
24/12/04 14:02:37 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:03:18 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=58, action_http_delete_request_
```



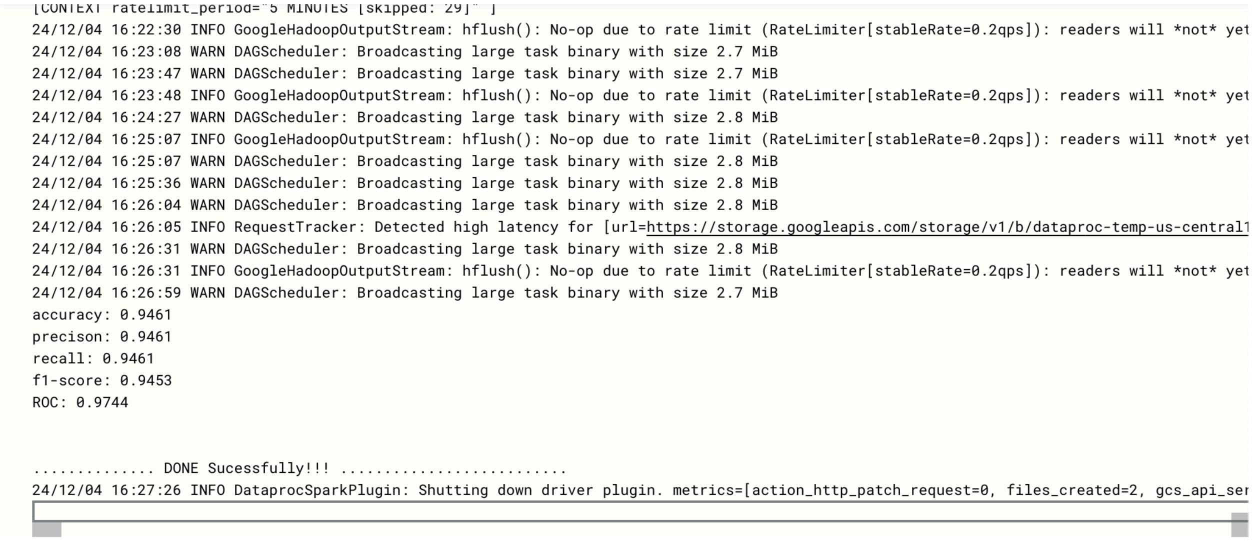
```
24/12/04 14:25:43 INFO RequestTracker: Detected high latency for [url=https://storage.googleapis.com/storage/v1/b/dataproc-temp-us-central1
24/12/04 14:26:06 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
accuracy: 0.5579
precison: 0.6491
recall: 0.5579
f1-score: 0.5773
ROC: 0.5814


========== Decision Tree ============
24/12/04 14:27:07 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:28:10 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:32:34 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=165, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 75]" ]
24/12/04 14:32:34 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:35:11 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:36:20 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:37:29 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:37:51 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=172, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 25]" ]
24/12/04 14:38:32 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
accuracy: 0.9248
```



```
24/12/04 14:38:32 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
accuracy: 0.9248
precison: 0.9309
recall: 0.9248
f1-score: 0.9214
ROC: 0.8819


========== Random Forest ============
24/12/04 14:39:38 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:44:47 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=180, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 49]" ]
24/12/04 14:44:47 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:47:32 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:48:55 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:50:24 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=185, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 19]" ]
24/12/04 14:50:24 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:51:35 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
accuracy: 0.7304
precison: 0.8034
recall: 0.7304
f1-score: 0.6382
ROC: 0.8198
```



```
========== Gradient Boosting Trees ============
24/12/04 14:53:08 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 14:57:39 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=193, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 43]" ]
24/12/04 14:57:39 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:00:35 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:01:53 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:03:11 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=198, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 19]" ]
24/12/04 15:03:11 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:04:48 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:06:06 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:07:25 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:09:10 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=203, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 26]" ]
24/12/04 15:09:10 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:10:28 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:11:49 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:13:29 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 15:14:48 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=211, action_http_delete_request
[CONTEXT ratelimit_period="5 MINUTES [skipped: 29]" ]
24/12/04 15:14:48 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
```



```
[CONTEXT ratelimit_period="5 MINUTES [skipped: 29]" ]
24/12/04 16:22:30 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 16:23:08 WARN DAGScheduler: Broadcasting large task binary with size 2.7 MiB
24/12/04 16:23:47 WARN DAGScheduler: Broadcasting large task binary with size 2.7 MiB
24/12/04 16:23:48 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 16:24:27 WARN DAGScheduler: Broadcasting large task binary with size 2.8 MiB
24/12/04 16:25:07 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 16:25:07 WARN DAGScheduler: Broadcasting large task binary with size 2.8 MiB
24/12/04 16:25:36 WARN DAGScheduler: Broadcasting large task binary with size 2.8 MiB
24/12/04 16:26:04 WARN DAGScheduler: Broadcasting large task binary with size 2.8 MiB
24/12/04 16:26:05 INFO RequestTracker: Detected high latency for [url=https://storage.googleapis.com/storage/v1/b/dataproc-temp-us-central1
24/12/04 16:26:31 WARN DAGScheduler: Broadcasting large task binary with size 2.8 MiB
24/12/04 16:26:31 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet
24/12/04 16:26:59 WARN DAGScheduler: Broadcasting large task binary with size 2.7 MiB
accuracy: 0.9461
precison: 0.9461
recall: 0.9461
f1-score: 0.9453
ROC: 0.9744


............. DONE Sucessfully!!! .........................
24/12/04 16:27:26 INFO DataprocSparkPlugin: Shutting down driver plugin. metrics=[action_http_patch_request=0, files_created=2, gcs_api_ser
```

Conclusion:

Ensemble Learning approaches are providing better accuracy out of which Gradient Boosting Machines gives highest performance of 94%. But training time for it is also quite high as compared to other algorithms.