Configuration Manual

MSc Research Project
Data Analytics

Emmanuel Amadi
Student ID: x18178103

School of Computing
National College of Ireland

Supervisor: Dr. Cristina Muntean

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Emmanuel Amadi
Student ID:	x18178103
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Cristina Muntean
Submission Due Date:	12th December 2019
Project Title:	Configuration Manual
Word Count:	1015
Page Count:	11

Signature:	
Date:	

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Emmanuel Amadi
x18178103

1 Introduction

This configuration manual presents a detailed hardware and software requirements necessary for the successful development of the research “Analysis of wildfire risk using machine learning and distributed computing in Canadian Region”. System specification

1.1 Hardware specification

The hardware used for the execution of this research was Apple Macbook pro. The specification for the hardware used includes; macOS Mojave operating system (version 10.14.3), 2.3 GHz Intel core i7 processor, 8GB 1600 MHz DDR3 memory and Macintosh HD as the startup disk.

1.2 Software configurations

This section describes the software configuration needed to successful deploy this solution. The first step the installation of Anaconda software as it contains the Jupiter notebook used as the IDE for writing the code and supports multiple operator systems. After the installation of the Anaconda software, open the search bar and search for “Terminal” and run the code outlined in figure 1 to successfully install python.

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev
                 libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
sudo apt-get install python
sudo easy_install pip
sudo pip install ipython
```

Figure 1: Python installation (Feng, 2019).

After the installation of python, the next step is ensuring that Java exists on the hardware. Figure 2 provides the line of code used to check if Java exist on the hardware. The Java runtime environment was downloaded and installed using the Oracle Java JDK.

```
java -version
```

Figure 2: Java version information for hardware (Feng, 2019).

To enable the distributed computing platform “Apache spark”, the pre-built version was downloaded from Apache spark official web site. The download file was unpacked to the spark folder on the local disk. Figure 3 shows the code to initialize the spark session of the hardware and the out of the initialization confirming the successful initialization of the spark a session.

```
./pyspark
```

```
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default, Dec 20 2016,␣
↪23:05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more␣
↪information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.
↪properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR,
use setLogLevel(newLevel).
17/08/30 13:30:12 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where␣
↪applicable
17/08/30 13:30:17 WARN ObjectStore: Failed to get database global_
↪temp,
returning NoSuchObjectException
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Python version 2.7.13 (default, Dec 20 2016 23:05:08)
SparkSession available as 'spark'.
```

Figure 3: Initializing Spark Session (Feng, 2019).

To set up the spark session as a global variable on the hardware to allow the Jupiter notebook IDE to run with the spark session. Figure 4 describes the code required to set bash profile from the terminal of the hardware. To lunch the Jupiter IDE the “pyspark” command was run on the terminal on the spark machine learning folder on the local disk.

```
vim ~/.bash_profile
```

```
# add for spark
export SPARK_HOME=your_spark_installation_path
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_DRIVE_PYTHON="jupyter"
export PYSPARK_DRIVE_PYTHON_OPTS="notebook"
```

```
source ~/.bash_profile
```

```
vim ~/.bashrc
```

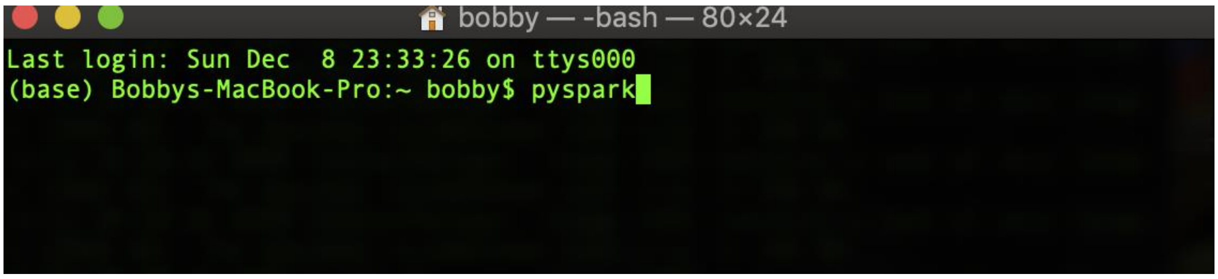
Figure 3: Initializing Spark on Jupiter Notebook (Feng, 2019).

2 Deploying the Python Script

This section describes the step by step procedure required to successfully deploy the solution on the target hardware.

2.1 Starting Jupiter Notebook with Spark

To access the spark session from Jupiter notebook, Open the terminal of the target hardware, navigate to the target folder containing the code and dataset. Enter the command “pyspark” on the terminal as shown in figure 4, this startup the Jupiter notebook with spark session running in the background. Open the file with the name “WildFireAnalysisMain” to lunch the code for the analysis.

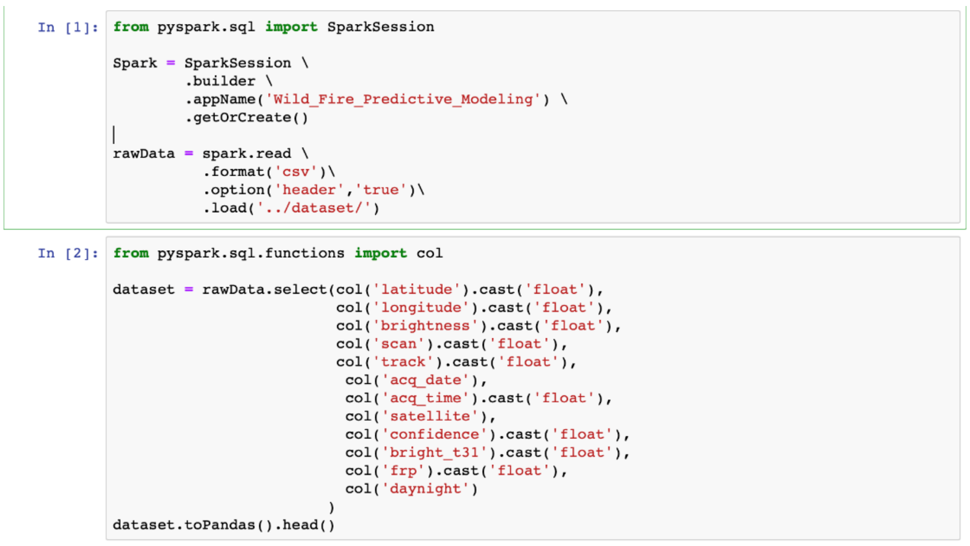


```
bobby — -bash — 80×24
Last login: Sun Dec  8 23:33:26 on ttys000
(base) Bobbys-MacBook-Pro:~ bobby$ pyspark
```

Figure 4: Running Jupiter notebook on spark session.

2.2 Loading the Dataset into the Script

To load the data into Jupiter notebook the SQL library for the spark is used as shown in figure 5 with the spark session explicitly declared and used to read the data from CSV into SQL format. The numeric columns are then cast to “float” data type as shown in figure 5 to ensure standardization of the data.



```
In [1]: from pyspark.sql import SparkSession

        Spark = SparkSession \
                .builder \
                .appName('Wild_Fire_Predictive_Modeling') \
                .getOrCreate()

        rawData = spark.read \
                  .format('csv')\
                  .option('header','true')\
                  .load('../dataset/')
```

```
In [2]: from pyspark.sql.functions import col

        dataset = rawData.select(col('latitude').cast('float'),
                                 col('longitude').cast('float'),
                                 col('brightness').cast('float'),
                                 col('scan').cast('float'),
                                 col('track').cast('float'),
                                  col('acq_date'),
                                  col('acq_time').cast('float'),
                                  col('satellite'),
                                  col('confidence').cast('float'),
                                  col('bright_t31').cast('float'),
                                  col('frp').cast('float'),
                                  col('daynight')
                                )
        dataset.toPandas().head()
```

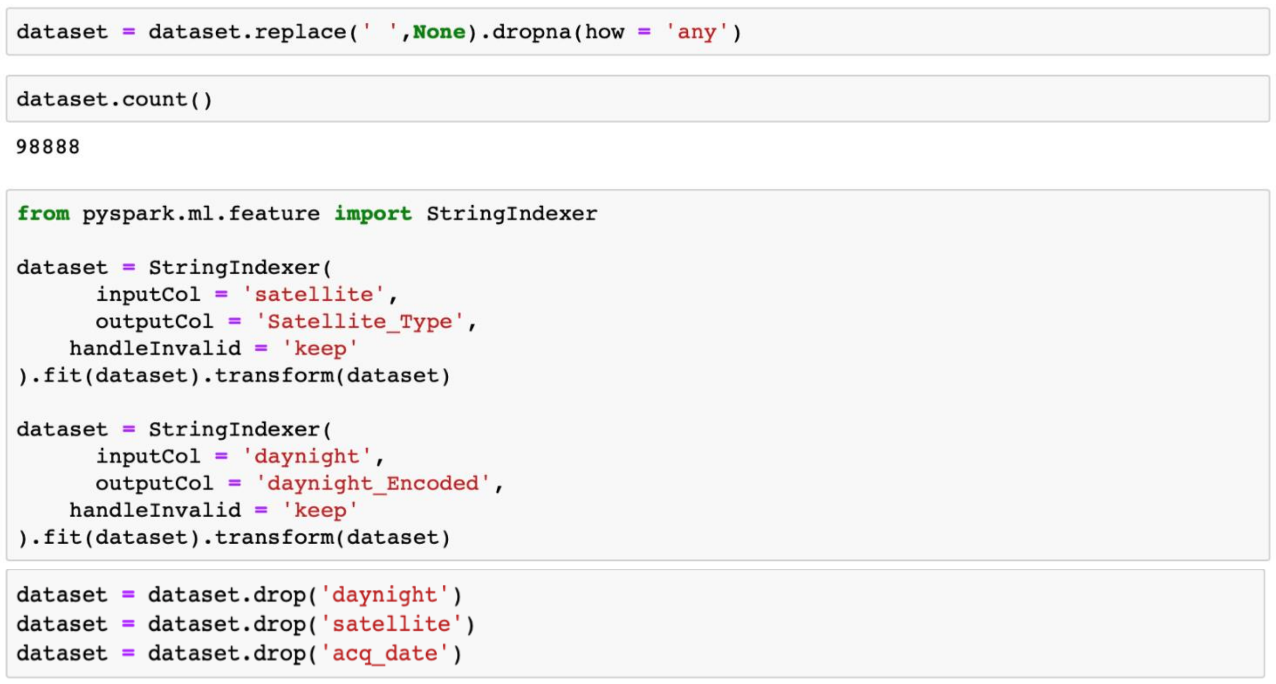
Figure 5: Running Jupiter notebook on spark session.

The lines of codes shown in figure 5 as well as other lines of code in this manual can be run in two ways:

- By running each line of code individual by using the run button at the top of the notebook and observing the output from the data.
- The entire code can be run at once by right-clicking on the cell line and selecting "Run All" from the dropdown list.

2.3 Preprocessing of Data

The next step for this research was the preprocessing phase carried out to further prepare the data for the analysis. Figure 6 shows the preprocessing tasks carried out; the first task was to ensure there were no missing variables in the data, secondly, the naming convection was regularized with the previous columns dropped from the analysis



```
dataset = dataset.replace(' ',None).dropna(how = 'any')
```

```
dataset.count()
```

```
98888
```

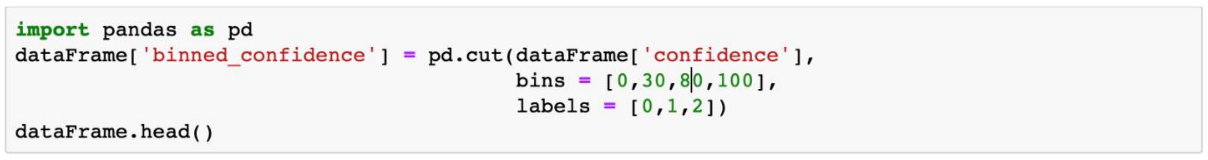
```
from pyspark.ml.feature import StringIndexer

dataset = StringIndexer(
      inputCol = 'satellite',
      outputCol = 'Satellite_Type',
    handleInvalid = 'keep'
).fit(dataset).transform(dataset)

dataset = StringIndexer(
      inputCol = 'daynight',
      outputCol = 'daynight_Encoded',
    handleInvalid = 'keep'
).fit(dataset).transform(dataset)
```

```
dataset = dataset.drop('daynight')
dataset = dataset.drop('satellite')
dataset = dataset.drop('acq_date')
```

Figure 6: Running Jupiter notebook on spark session.

2.4 Data Binning

The classification phase for this research utilizes the confidence values calculated for each fire pixel. Thus, the confidence value is binned into 3 categories stated in Giglio *et al*. (2018) guide for analysing the MODIS dataset.



```
import pandas as pd
dataFrame['binned_confidence'] = pd.cut(dataFrame['confidence'],
                                        bins = [0,30,80,100],
                                        labels = [0,1,2])
dataFrame.head()
```

Figure 7: Data Binning for Confidence Variable

2.5 Over Sampling

The binned data generated an imbalance dataset as shown in figure 8, with the majority of the data belonging to the high fire pixel class.

```
smoteDataFrame = pd.DataFrame(y, columns=['binned_confidence'])
smoteDataFrame['binned_confidence'].value_counts().plot.bar(title='Freq dist of fire pixel')
objects = ('high','norminal','low')
y_pos = np.arange(len(objects))

plt.bar(y_pos, smoteDataFrame['binned_confidence'].value_counts(), align='center')
plt.xticks(y_pos, objects)
plt.ylabel('Observation')
plt.title('Data Binning')

plt.show()
```

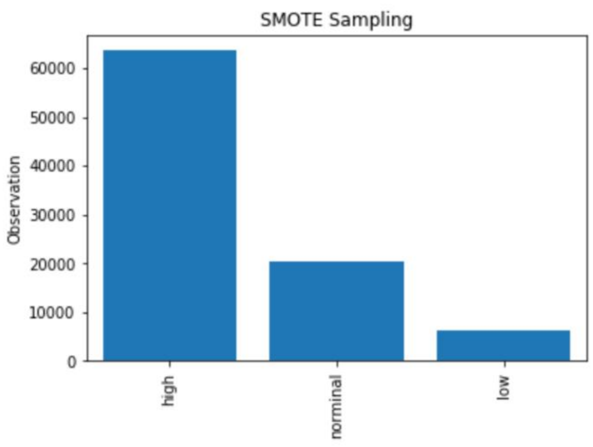


Figure 8: Imbalanced Class of the Dataset

Hence, the SMOTE (Synthentic Minority Over-sampling Technique) is used to ensure class balance to avoid biased results in the analysis. Figure 9 shows the over-sampling technique and the expected output.

```
smoteDataFrame = pd.DataFrame(y_samp, columns=['binned_confidence'])
smoteDataFrame['binned_confidence'].value_counts().plot.bar(title='Freq dist of fire pixel')
objects = ('high','norminal','low')
y_pos = np.arange(len(objects))

plt.bar(y_pos, smoteDataFrame['binned_confidence'].value_counts(), align='center')
plt.xticks(y_pos, objects)
plt.ylabel('Observation')
plt.title('SMOTE Sampling')

plt.show()
```

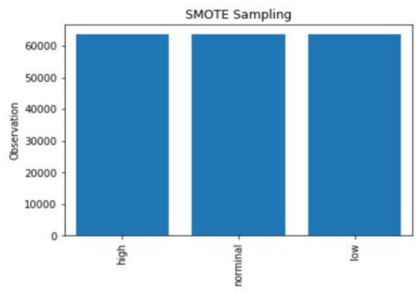


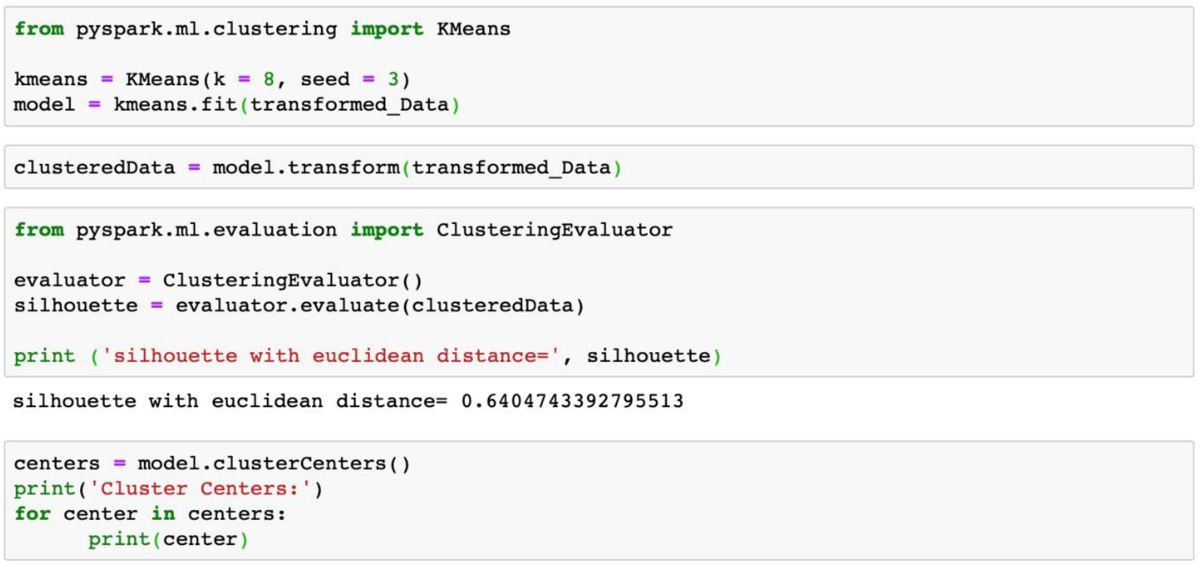
Figure 8: Imbalanced Class of the Dataset

3 Implementation

This section provides a detailed description of the algorithms used in this research. The decisions for choosing these algorithms were properly explained in the research paper.

3.1 Kmeans Clustering

The kmeans clustering was implemented as shown in figure 9 with the Euclidean distance calculation and centers calculated for each of the clusters. Figure 9 also describes the implementation of the elbow method used to evaluate the clusters for 14 clusters.



```
from pyspark.ml.clustering import KMeans

kmeans = KMeans(k = 8, seed = 3)
model = kmeans.fit(transformed_Data)
```

```
clusteredData = model.transform(transformed_Data)
```

```
from pyspark.ml.evaluation import ClusteringEvaluator

evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(clusteredData)

print ('silhouette with euclidean distance=', silhouette)
```

```
silhouette with euclidean distance= 0.6404743392795513
```

```
centers = model.clusterCenters()
print('Cluster Centers:')
for center in centers:
      print(center)
```

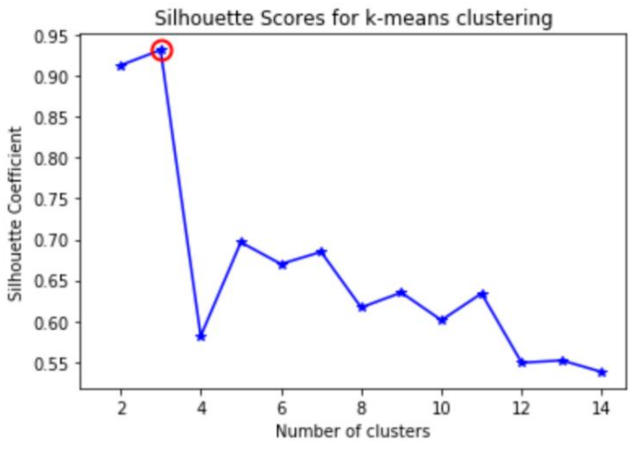
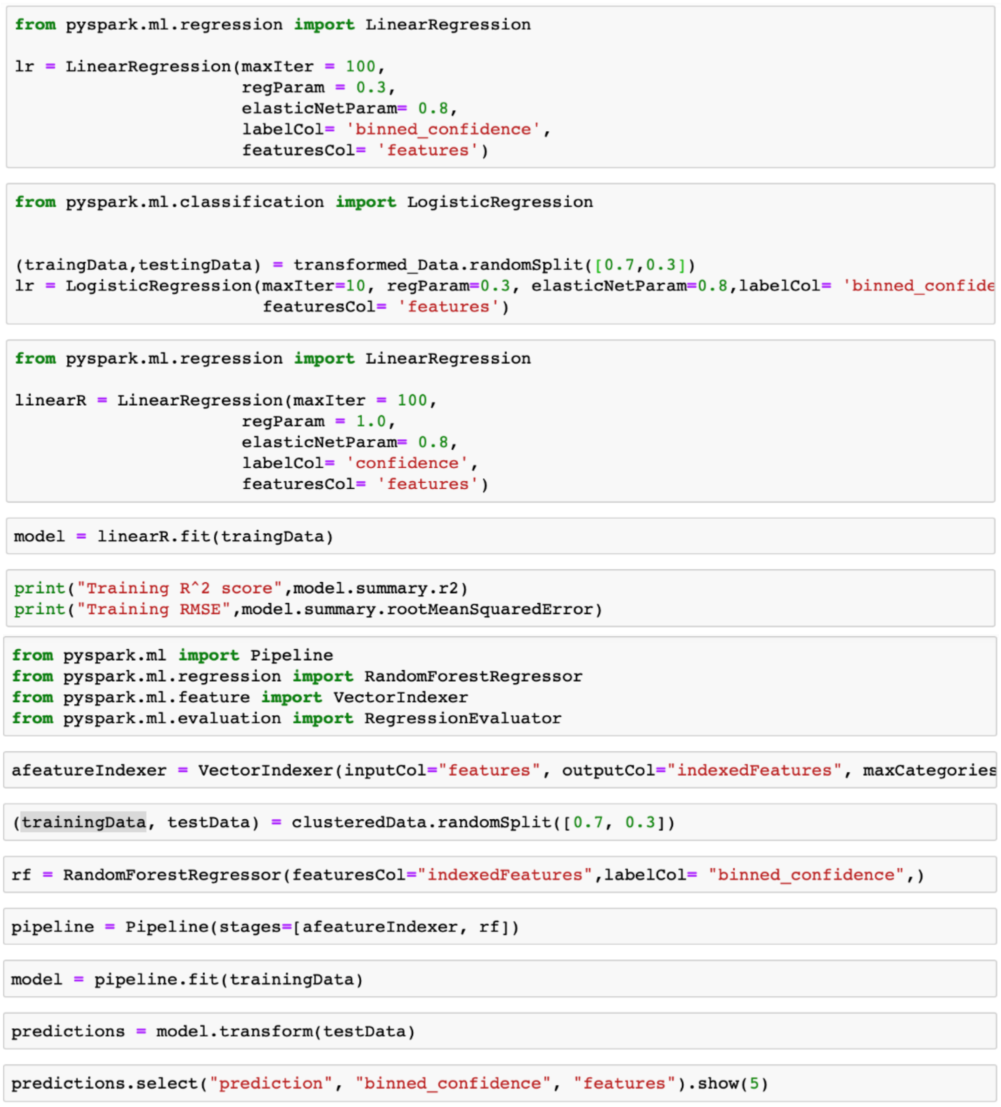


Figure 9: Kmeans Clustering

3.2 Regression

The regression for this research was applied using linear and random forest regression. Figure 10 shows the implementation of these algorithms on the dataset.



```
from pyspark.ml.regression import LinearRegression

lr = LinearRegression(maxIter = 100,
                      regParam = 0.3,
                      elasticNetParam= 0.8,
                      labelCol= 'binned_confidence',
                      featuresCol= 'features')
```

```
from pyspark.ml.classification import LogisticRegression


(traingData,testingData) = transformed_Data.randomSplit([0.7,0.3])
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8,labelCol= 'binned_confide
                        featuresCol= 'features')
```

```
from pyspark.ml.regression import LinearRegression

linearR = LinearRegression(maxIter = 100,
                      regParam = 1.0,
                      elasticNetParam= 0.8,
                      labelCol= 'confidence',
                      featuresCol= 'features')
```

```
model = linearR.fit(traingData)
```

```
print("Training R^2 score",model.summary.r2)
print("Training RMSE",model.summary.rootMeanSquaredError)
```

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
```

```
afeatureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories
```

```
(trainingData, testData) = clusteredData.randomSplit([0.7, 0.3])
```

```
rf = RandomForestRegressor(featuresCol="indexedFeatures",labelCol= "binned_confidence",)
```

```
pipeline = Pipeline(stages=[afeatureIndexer, rf])
```

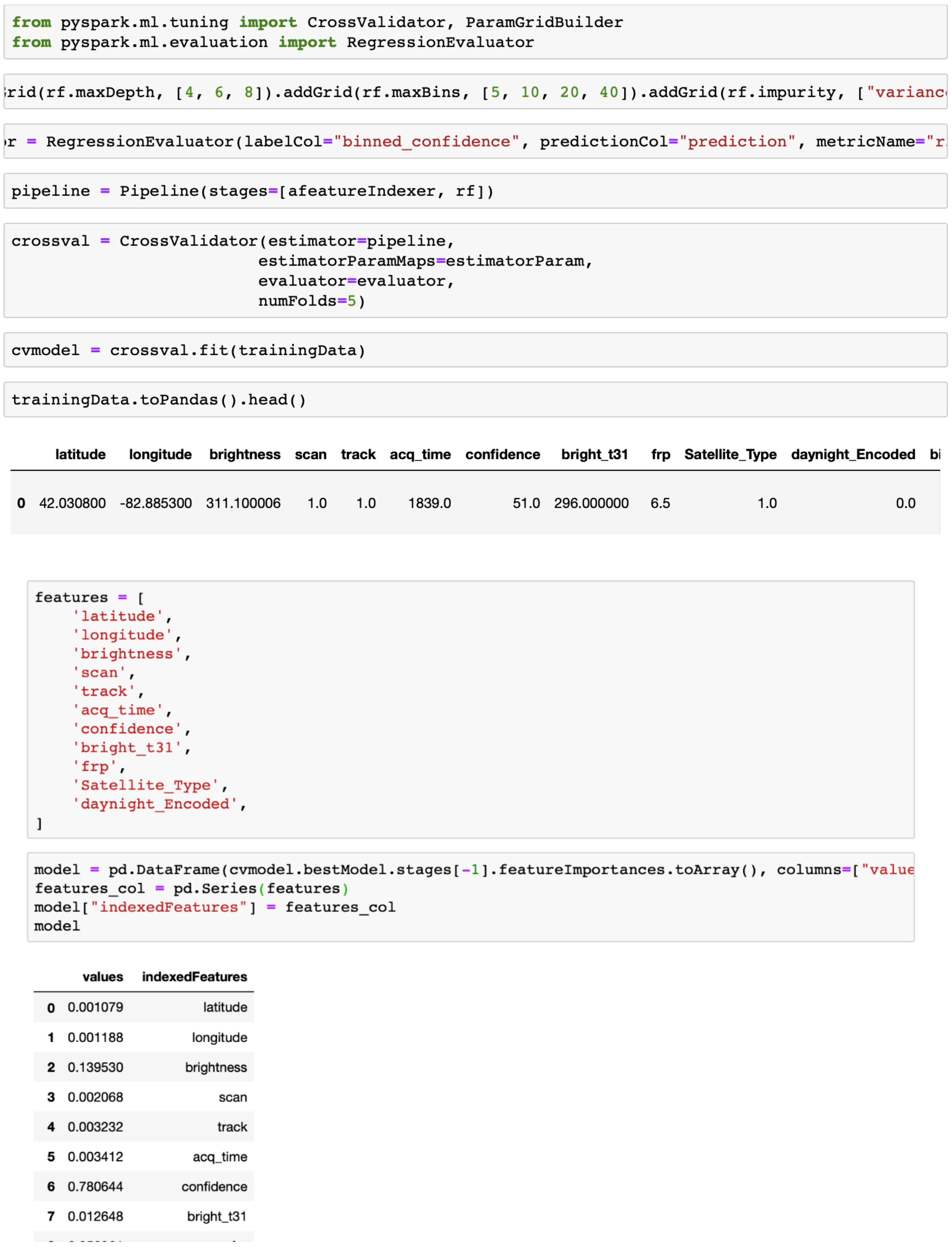
```
model = pipeline.fit(trainingData)
```

```
predictions = model.transform(testData)
```

```
predictions.select("prediction", "binned_confidence", "features").show(5)
```

Figure 10: Regression Implementation

To get the best performing model, cross-validation was used to compute the best performing model. Figure 11 shows the values for each of the feature used in the analysis.



```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import RegressionEvaluator
```

```
rid(rf.maxDepth, [4, 6, 8]).addGrid(rf.maxBins, [5, 10, 20, 40]).addGrid(rf.impurity, ["varianc
```

```
r = RegressionEvaluator(labelCol="binned_confidence", predictionCol="prediction", metricName="r
```

```
pipeline = Pipeline(stages=[afeatureIndexer, rf])
```

```
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=estimatorParam,
                          evaluator=evaluator,
                          numFolds=5)
```

```
cvmodel = crossval.fit(trainingData)
```

```
trainingData.toPandas().head()
```

	latitude	longitude	brightness	scan	track	acq_time	confidence	bright_t31	frp	Satellite_Type	daynight_Encoded	bi
0	42.030800	-82.885300	311.100006	1.0	1.0	1839.0	51.0	296.000000	6.5	1.0	0.0	

```
features = [
    'latitude',
    'longitude',
    'brightness',
    'scan',
    'track',
    'acq_time',
    'confidence',
    'bright_t31',
    'frp',
    'Satellite_Type',
    'daynight_Encoded',
]
```

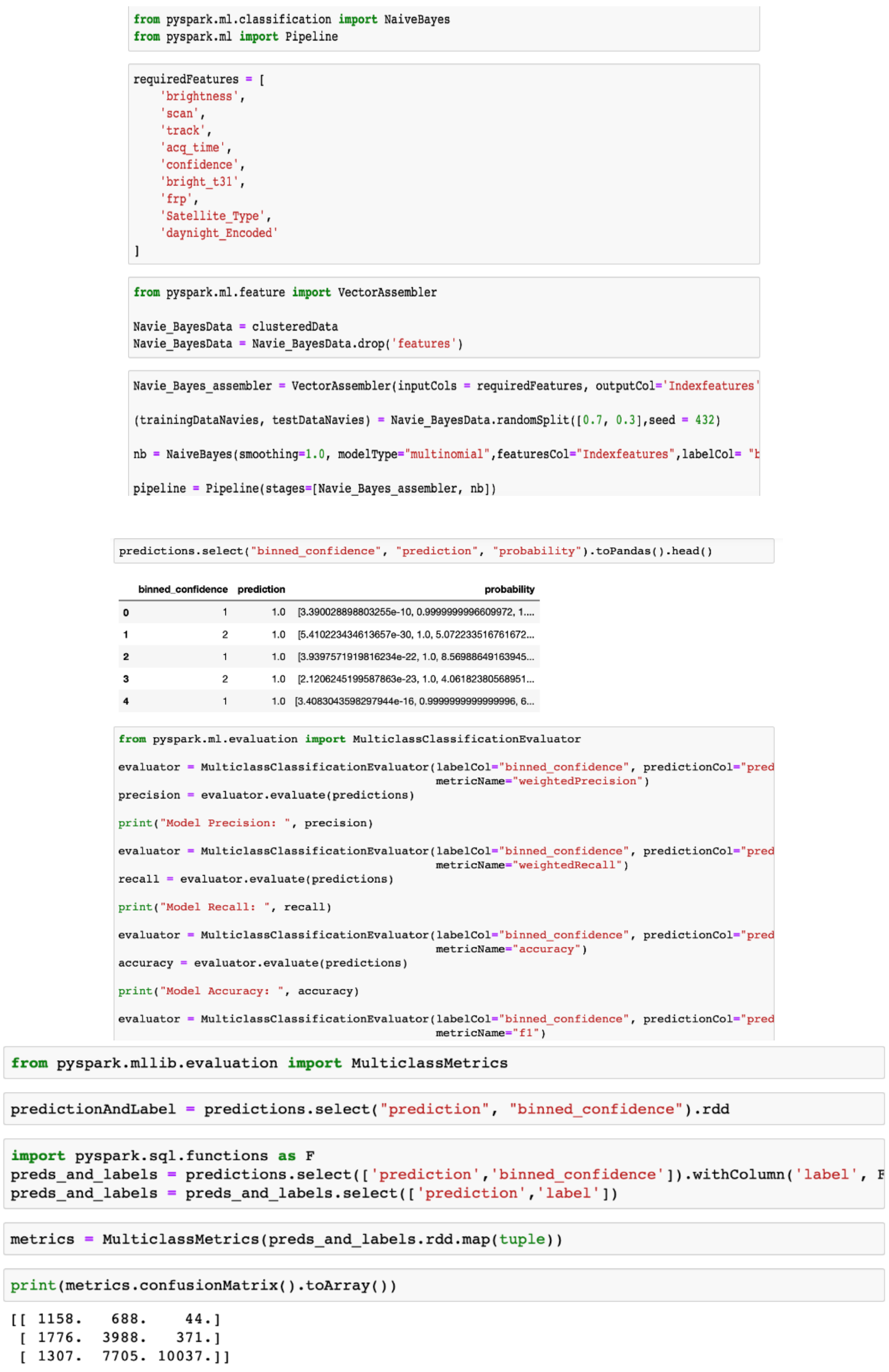
```
model = pd.DataFrame(cvmodel.bestModel.stages[-1].featureImportances.toArray(), columns=["value
features_col = pd.Series(features)
model["indexedFeatures"] = features_col
model
```

	values	indexedFeatures
0	0.001079	latitude
1	0.001188	longitude
2	0.139530	brightness
3	0.002068	scan
4	0.003232	track
5	0.003412	acq_time
6	0.780644	confidence
7	0.012648	bright_t31

Figure 10: Cross-Validation of Regression Model

3.3 Classification

The multiclass classification for this research was implemented using Random forest and navie baye's classifier. The implementation code for the navie baye's classifier is shown in figure 11 while figure 12 shows the implementation code for the Random Forest classifier with the evaluation metricsapplied for each phase of the research.



```
from pyspark.ml.classification import NaiveBayes
from pyspark.ml import Pipeline
```

```
requiredFeatures = [
    'brightness',
    'scan',
    'track',
    'acq_time',
    'confidence',
    'bright_t31',
    'frp',
    'Satellite_Type',
    'daynight_Encoded'
]
```

```
from pyspark.ml.feature import VectorAssembler

Navie_BayesData = clusteredData
Navie_BayesData = Navie_BayesData.drop('features')
```

```
Navie_Bayes_assembler = VectorAssembler(inputCols = requiredFeatures, outputCol='Indexfeatures'

(trainingDataNavies, testDataNavies) = Navie_BayesData.randomSplit([0.7, 0.3],seed = 432)

nb = NaiveBayes(smoothing=1.0, modelType="multinomial",featuresCol="Indexfeatures",labelCol= "b

pipeline = Pipeline(stages=[Navie_Bayes_assembler, nb])
```

```
predictions.select("binned_confidence", "prediction", "probability").toPandas().head()
```

	binned_confidence	prediction	probability
0	1	1.0	[3.390028898803255e-10, 0.9999999996609972, 1....
1	2	1.0	[5.410223434613657e-30, 1.0, 5.072233516761672...
2	1	1.0	[3.9397571919816234e-22, 1.0, 8.56988649163945...
3	2	1.0	[2.1206245199587863e-23, 1.0, 4.06182380568951...
4	1	1.0	[3.4083043598297944e-16, 0.9999999999999996, 6...

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="binned_confidence", predictionCol="pred
                                              metricName="weightedPrecision")
precision = evaluator.evaluate(predictions)

print("Model Precision: ", precision)

evaluator = MulticlassClassificationEvaluator(labelCol="binned_confidence", predictionCol="pred
                                              metricName="weightedRecall")
recall = evaluator.evaluate(predictions)

print("Model Recall: ", recall)

evaluator = MulticlassClassificationEvaluator(labelCol="binned_confidence", predictionCol="pred
                                              metricName="accuracy")
accuracy = evaluator.evaluate(predictions)

print("Model Accuracy: ", accuracy)

evaluator = MulticlassClassificationEvaluator(labelCol="binned_confidence", predictionCol="pred
                                              metricName="f1")
```

```
from pyspark.mllib.evaluation import MulticlassMetrics
```

```
predictionAndLabel = predictions.select("prediction", "binned_confidence").rdd
```

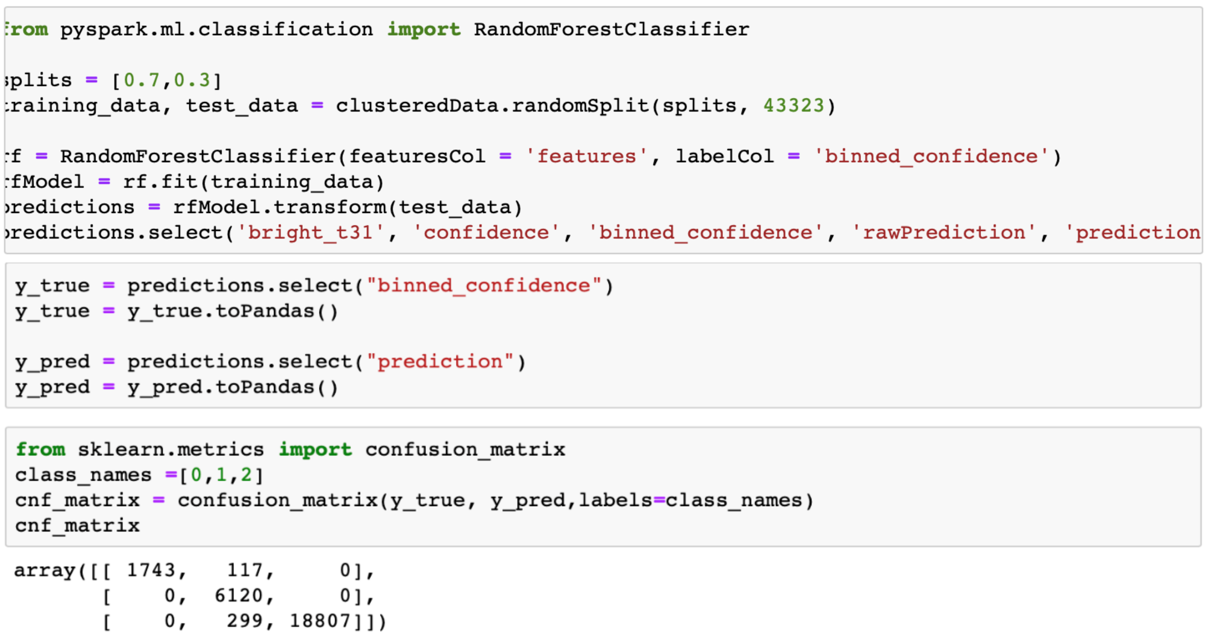
```
import pyspark.sql.functions as F
preds_and_labels = predictions.select(['prediction','binned_confidence']).withColumn('label', F
preds_and_labels = preds_and_labels.select(['prediction','label'])
```

```
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))
```

```
print(metrics.confusionMatrix().toArray())
```

```
[[ 1158.   688.    44.]
 [ 1776.  3988.   371.]
 [ 1307.  7705. 10037.]]
```

Figure 10: Navie Baye's Classifier



```
rom pyspark.ml.classification import RandomForestClassifier

plits = [0.7,0.3]
raining_data, test_data = clusteredData.randomSplit(splits, 43323)

f = RandomForestClassifier(featuresCol = 'features', labelCol = 'binned_confidence')
fModel = rf.fit(training_data)
redictions = rfModel.transform(test_data)
redictions.select('bright_t31', 'confidence', 'binned_confidence', 'rawPrediction', 'prediction
```

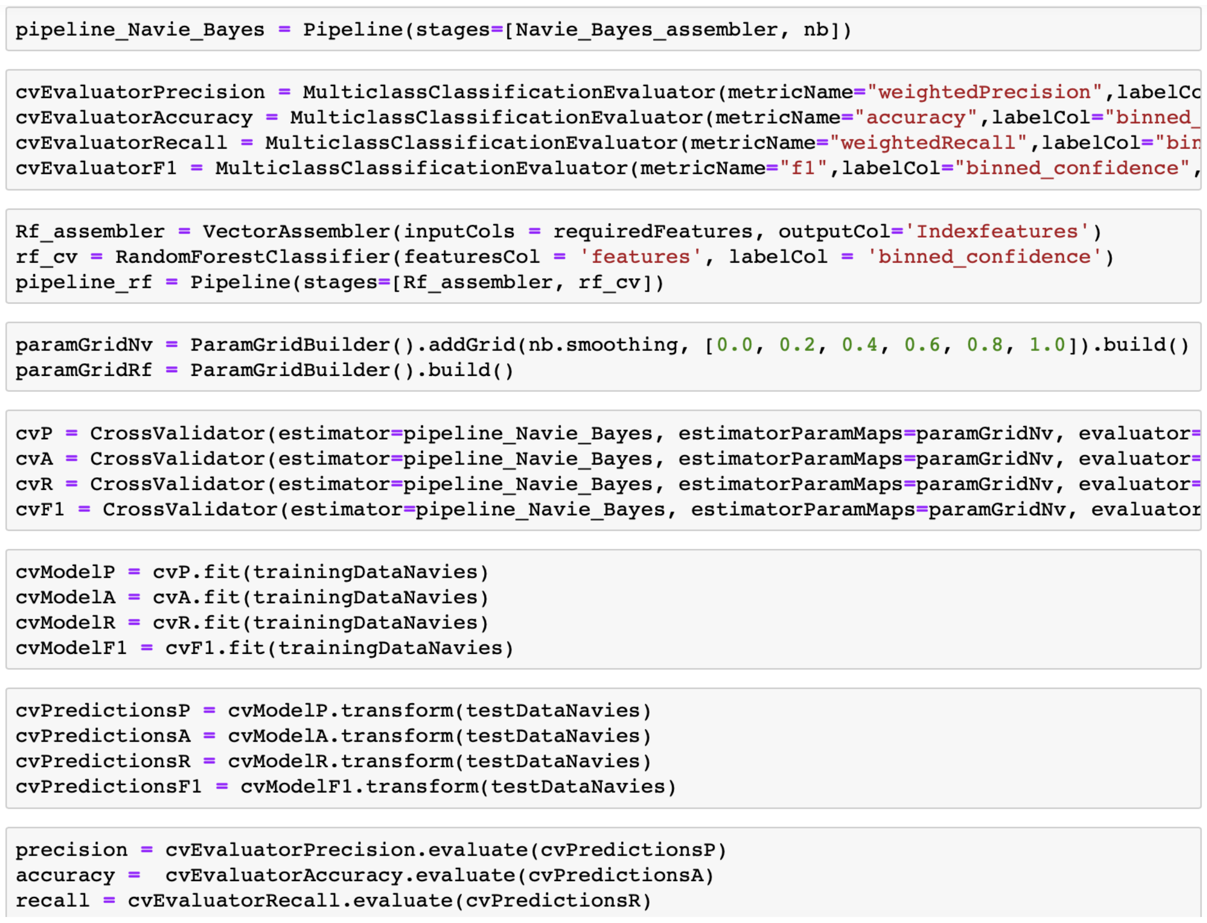
```
y_true = predictions.select("binned_confidence")
y_true = y_true.toPandas()

y_pred = predictions.select("prediction")
y_pred = y_pred.toPandas()
```

```
from sklearn.metrics import confusion_matrix
class_names =[0,1,2]
cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix
```

```
array([[ 1743,   117,     0],
       [    0,  6120,     0],
       [    0,   299, 18807]])
```

Figure 11: Random Forest Classifier

The cross validation for both models is shown in figure 12 to validate the result generated by the models.



```
pipeline_Navie_Bayes = Pipeline(stages=[Navie_Bayes_assembler, nb])
```

```
cvEvaluatorPrecision = MulticlassClassificationEvaluator(metricName="weightedPrecision",labelCo
cvEvaluatorAccuracy = MulticlassClassificationEvaluator(metricName="accuracy",labelCol="binned_
cvEvaluatorRecall = MulticlassClassificationEvaluator(metricName="weightedRecall",labelCol="bin
cvEvaluatorF1 = MulticlassClassificationEvaluator(metricName="f1",labelCol="binned_confidence",
```

```
Rf_assembler = VectorAssembler(inputCols = requiredFeatures, outputCol='Indexfeatures')
rf_cv = RandomForestClassifier(featuresCol = 'features', labelCol = 'binned_confidence')
pipeline_rf = Pipeline(stages=[Rf_assembler, rf_cv])
```

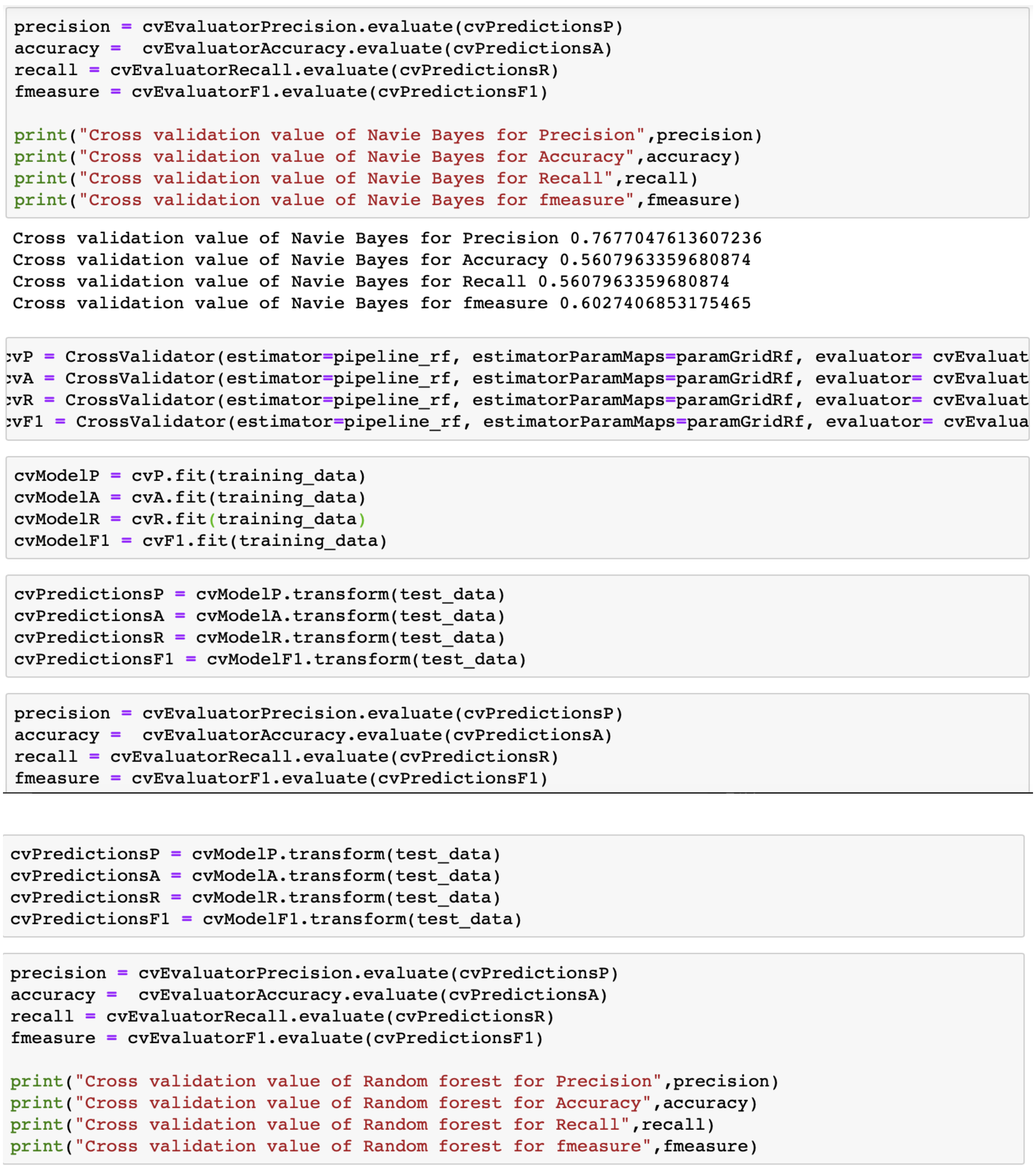
```
paramGridNv = ParamGridBuilder().addGrid(nb.smoothing, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]).build()
paramGridRf = ParamGridBuilder().build()
```

```
cvP = CrossValidator(estimator=pipeline_Navie_Bayes, estimatorParamMaps=paramGridNv, evaluator=
cvA = CrossValidator(estimator=pipeline_Navie_Bayes, estimatorParamMaps=paramGridNv, evaluator=
cvR = CrossValidator(estimator=pipeline_Navie_Bayes, estimatorParamMaps=paramGridNv, evaluator=
cvF1 = CrossValidator(estimator=pipeline_Navie_Bayes, estimatorParamMaps=paramGridNv, evaluator
```

```
cvModelP = cvP.fit(trainingDataNavies)
cvModelA = cvA.fit(trainingDataNavies)
cvModelR = cvR.fit(trainingDataNavies)
cvModelF1 = cvF1.fit(trainingDataNavies)
```

```
cvPredictionsP = cvModelP.transform(testDataNavies)
cvPredictionsA = cvModelA.transform(testDataNavies)
cvPredictionsR = cvModelR.transform(testDataNavies)
cvPredictionsF1 = cvModelF1.transform(testDataNavies)
```

```
precision = cvEvaluatorPrecision.evaluate(cvPredictionsP)
accuracy =  cvEvaluatorAccuracy.evaluate(cvPredictionsA)
recall = cvEvaluatorRecall.evaluate(cvPredictionsR)
```



```
precision = cvEvaluatorPrecision.evaluate(cvPredictionsP)
accuracy =  cvEvaluatorAccuracy.evaluate(cvPredictionsA)
recall = cvEvaluatorRecall.evaluate(cvPredictionsR)
fmeasure = cvEvaluatorF1.evaluate(cvPredictionsF1)

print("Cross validation value of Navie Bayes for Precision",precision)
print("Cross validation value of Navie Bayes for Accuracy",accuracy)
print("Cross validation value of Navie Bayes for Recall",recall)
print("Cross validation value of Navie Bayes for fmeasure",fmeasure)
```

```
Cross validation value of Navie Bayes for Precision 0.7677047613607236
Cross validation value of Navie Bayes for Accuracy 0.5607963359680874
Cross validation value of Navie Bayes for Recall 0.5607963359680874
Cross validation value of Navie Bayes for fmeasure 0.6027406853175465
```

```
:vP = CrossValidator(estimator=pipeline_rf, estimatorParamMaps=paramGridRf, evaluator= cvEvaluat
:vA = CrossValidator(estimator=pipeline_rf, estimatorParamMaps=paramGridRf, evaluator= cvEvaluat
:vR = CrossValidator(estimator=pipeline_rf, estimatorParamMaps=paramGridRf, evaluator= cvEvaluat
:vF1 = CrossValidator(estimator=pipeline_rf, estimatorParamMaps=paramGridRf, evaluator= cvEvalua
```

```
cvModelP = cvP.fit(training_data)
cvModelA = cvA.fit(training_data)
cvModelR = cvR.fit(training_data)
cvModelF1 = cvF1.fit(training_data)
```

```
cvPredictionsP = cvModelP.transform(test_data)
cvPredictionsA = cvModelA.transform(test_data)
cvPredictionsR = cvModelR.transform(test_data)
cvPredictionsF1 = cvModelF1.transform(test_data)
```

```
precision = cvEvaluatorPrecision.evaluate(cvPredictionsP)
accuracy =  cvEvaluatorAccuracy.evaluate(cvPredictionsA)
recall = cvEvaluatorRecall.evaluate(cvPredictionsR)
fmeasure = cvEvaluatorF1.evaluate(cvPredictionsF1)
```

```
cvPredictionsP = cvModelP.transform(test_data)
cvPredictionsA = cvModelA.transform(test_data)
cvPredictionsR = cvModelR.transform(test_data)
cvPredictionsF1 = cvModelF1.transform(test_data)
```

```
precision = cvEvaluatorPrecision.evaluate(cvPredictionsP)
accuracy =  cvEvaluatorAccuracy.evaluate(cvPredictionsA)
recall = cvEvaluatorRecall.evaluate(cvPredictionsR)
fmeasure = cvEvaluatorF1.evaluate(cvPredictionsF1)

print("Cross validation value of Random forest for Precision",precision)
print("Cross validation value of Random forest for Accuracy",accuracy)
print("Cross validation value of Random forest for Recall",recall)
print("Cross validation value of Random forest for fmeasure",fmeasure)
```

Figure 11: Cross validation of Classification Algorithms

References

Giglio, L., Schroeder, W., Hall, J., & Justice, C. (2018). *MODIS Collection 6 Active Fire Product User's Guide Revision B.*

Feng, W. (2019). Learning Apache Spark with Python. 1-427.