Configuration Manual

MSc Research Project
Data Analytics

Emmanuel Amadi
Student ID: x18178103

School of Computing
National College of Ireland

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

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author’s written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

- Attach a completed copy of this sheet to each project (including multiple copies).
- Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).
- **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<table>
<thead>
<tr>
<th>Office Use Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature:</td>
</tr>
<tr>
<td>Date:</td>
</tr>
<tr>
<td>Penalty Applied (if applicable):</td>
</tr>
</tbody>
</table>
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 operating 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
```

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
```

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

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.

![Figure 4: Running Jupiter notebook on spark session.](image)

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.

```python
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")
```

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

dataset = rawData.select(col("latitude").cast("float"),
col("longitude").cast("float"),
col("brightness").cast("float"),
col("soa")
          .cast("float"),
col("track").cast("float"),
col("wq_date"),
col("wq_time").cast("float"),
col("satellite"),
col("confidence").cast("float"),
col("bright_t3")
          .cast("float"),
col("frp")
          .cast("float"),
col("daynight")
)
dataset.toPandas().head()
```

![Figure 5: Running Jupiter notebook on spark session.](image)
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 convention was regularized with the previous columns dropped from the analysis

```python
import pandas as pd
dataFrame[‘binned_confi...confidences stated in Giglio et al. (2018) guide for analysing the MODIS dataset.
```

**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.

```python
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 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.

![SMOTE Sampling](image)

**Figure 8: Imbalanced Class of the Dataset**

Hence, the SMOTE (Synthetic 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.
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.

```python
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)
```

![Silhouette Scores for k-means clustering](image)

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

r = RandomForestRegressor(maxDepth=4, maxBins=5, maxFeatures=4, impurity='variance').fit(observations)

cv = CrossValidator(estimator=rf, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=5)
cvModel = cv.fit(observations)

train = cvModel.transform(observations)

test = spark.createDataFrame(ncols=["latitude", "longitude", "brightness", "scan", "track", "acctime", "confidence", "bright31", "frp", "satellite_type", "daynight_encoded"], numrows=1)

test = test.withColumn("latitude", lit(42.030800)).withColumn("longitude", lit(-82.885300)).withColumn("brightness", lit(311.100000)).withColumn("scan", lit(1.0)).withColumn("track", lit(1.0)).withColumn("acctime", lit(1839.0)).withColumn("confidence", lit(51.0)).withColumn("bright31", lit(296.000000)).withColumn("frp", lit(6.5)).withColumn("satellite_type", lit(1.0)).withColumn("daynight_encoded", lit(0.0)).

features = ["latitude", "longitude", "brightness", "scan", "track", "acctime", "confidence", "bright31", "frp", "satellite_type", "daynight_encoded"]

model = pd.DataFrame(cvModel.bestModel.stages[1].featureImportances.toArray(), columns=features)
model.head()
```

![Figure 10: Cross-Validation of Regression Model](image)
3.3 Classification

The multiclass classification for this research was implemented using Random forest and Naive Baye’s classifier. The implementation code for the Naive Baye’s classifier is shown in figure 11 while figure 12 shows the implementation code for the Random Forest classifier with the evaluation metrics applied for each phase of the research.

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

requiredFeatures = ['brightness', 'area', 'red', 'big_tifs', 'confidence', 'height', 'clip', 'satellite_type', 'daynight_encoded']

from pyspark.ml.feature import VectorAssembler

NavieBayesData = clusteringData
NavieBayesData = NavieBayesData.drop('features')

NavieBayes_assembler = VectorAssembler(inputCols = requiredFeatures, outputCol = 'indexFeatures')
naviBayesData = NavieBayesData.randomSplit([1.0, 0.1], seed = 431)

rb = NavieBayes(smoothcoef=1.0, modeType = 'multinomial', featureCol = 'indexFeatures', labelCol = 'label')

pipeline = Pipeline(stages=[NavieBayes_assembler, rb])

predictions.select('binned_confidence', 'prediction', 'probability').toPandas().head()

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol = 'binned_confidence', predictionCol = 'pred', metricName = 'weightedPrecision', metricMode = 'max')

print('Model Precision:', evaluator.evaluate(predictions))

evaluator = MulticlassClassificationEvaluator(labelCol = 'binned_confidence', predictionCol = 'pred', metricName = 'weightedRecall', metricMode = 'max')

print('Model Recall:', evaluator.evaluate(predictions))

evaluator = MulticlassClassificationEvaluator(labelCol = 'binned_confidence', predictionCol = 'pred', metricName = 'accuracy', metricMode = 'max')

print('Model Accuracy:', evaluator.evaluate(predictions))

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.col('label'))
preds_and_labels = preds_and_labels.select(['prediction', 'label'])

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

print(metrics.confusionMatrix().toPandas())

[[ 1158.,  688.,  44.],
 [ 1174.,  3988.,  371.],
 [ 1307.,  7705.  10037.]]

Figure 10: Naive Baye’s Classifier
Figure 11: Random Forest Classifier

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

cvEvaluatorPrecision = MulticlassClassificationEvaluator(metricName='weightedPrecision',labelCol="binned_confidence")
cvEvaluatorAccuracy = MulticlassClassificationEvaluator(metricName="accuracy",labelCol="binned_confidence")
cvEvaluatorRecall = MulticlassClassificationEvaluator(metricName="weightedRecall",labelCol="binned_confidence")
cvEvaluatorF1 = MulticlassClassificationEvaluator(metricName="f1",labelCol="binned_confidence")

pipeline_NB = Pipeline(stages=[NaiveBayesAssembler, nb])

pipeline_RF = Pipeline(stages=[RandomForestClassifier(featuresCol = 'features', labelCol = 'binned_confidence')])

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

paramGridRF = ParamGridBuilder().build() 

cvP = CrossValidator(estimator=pipeline_NB, estimatorParamMaps=paramGridDNV, evaluator=cvEvaluatorPrecision)
cvA = CrossValidator(estimator=pipeline_NB, estimatorParamMaps=paramGridDNV, evaluator=cvEvaluatorAccuracy)
cvR = CrossValidator(estimator=pipeline_NB, estimatorParamMaps=paramGridDNV, evaluator=cvEvaluatorRecall)
cvF1 = CrossValidator(estimator=pipeline_NB, estimatorParamMaps=paramGridDNV, evaluator=cvEvaluatorF1)

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)
Figure 11: Cross validation of Classification Algorithms

References
