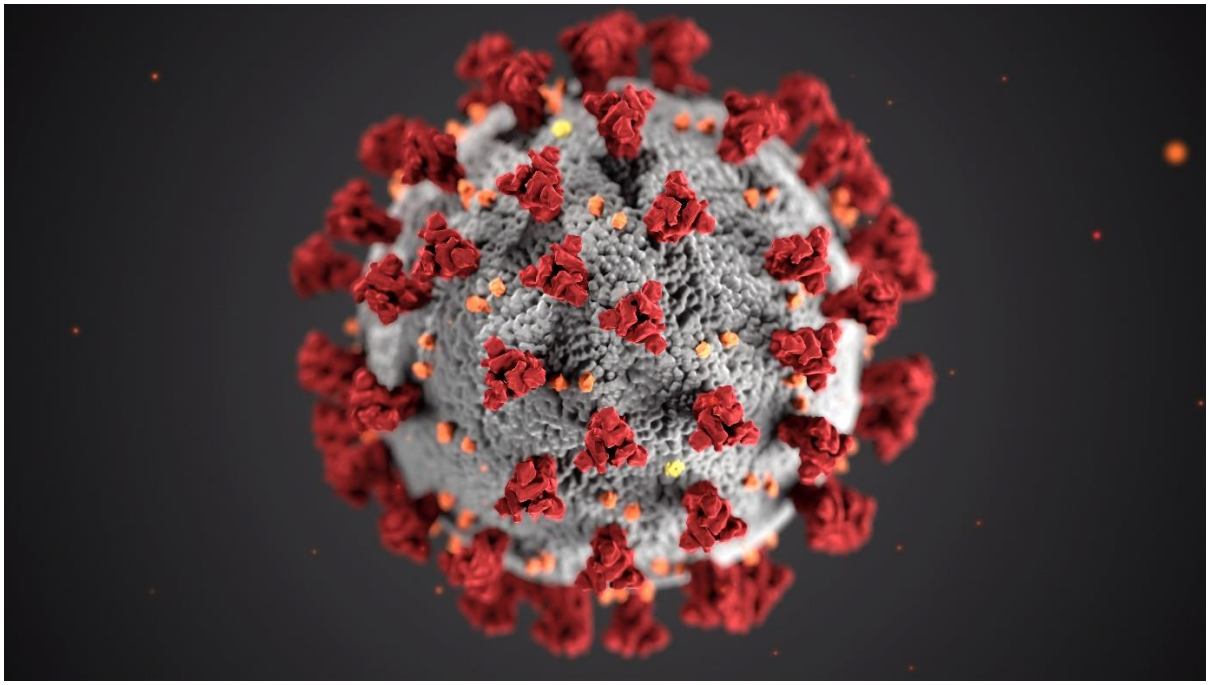
BDAT 1008 Assignment 2

To

Saber Amini



Submitted by:

Abhijeet Singh

Contents

Introduction

Pandemic has hit the world badly and we are still struggling to get on to normal life. The situation is no different in Toronto. This report walks through a machine learning model to identify the fatalities in association with few available independent features. Random Forest (RF) algorithm is employed in predicting the fatalities, but the target label *Outcome* is imbalanced. This report talks about the two approaches utilized to mitigate the bias due to imbalanced dataset while applying RF algorithm.

The dataset considered for the fatalities prediction is acquired from Toronto Open data and here is the link for the same: [COVID-19 Cases in Toronto - City of Toronto Open Data Portal]

This data set contains demographic, geographic, and severity information for all confirmed and probable cases reported to and managed by Toronto Public Health since the first case was reported in January 2020. This includes cases that are sporadic (occurring in the community) and outbreak associated. The data are extracted from the provincial Case & Contact Management System (CCM).

Dataset Exploration

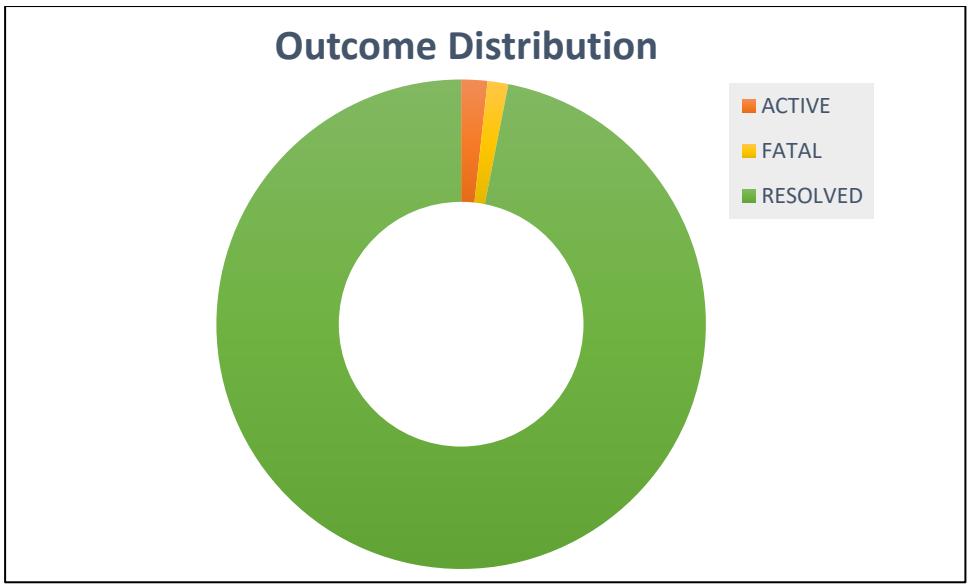
On initial observation the dataset contains 300k plus records with 18 fields namely:

Id, Assigned ID, Outbreak Associated, Age Group, Neighbourhood Name, FSA, Source of Infection, Classification, Episode Date, Reported Date, Client Gender, Outcome, Currently Hospitalized, Currently in ICU, Currently Intubated, Ever Hospitalized, Ever in ICU, Ever Intubated.

Investigating from the perspective of missing values in the dataset, each column has been examined for NULL values. The dataset is clean and observed that field *Age Group* has 300 missing values, which is very small in comparison to size of the dataset. This data has been filtered from dataset. In addition, *FSA* and *Neighbourhood* fields have 1 and 2 percent NULL values respectively, but those fields are not being considered for the analysis purpose.

On further investigation on the target field *Outcome,* it has been noticed that this field holds three unique values ACTIVE, FATAL and RESOLVED. The distribution of these values are as follows

Row Labels	Count of _id	Percentage of Total data
ACTIVE	5248	1.73
FATAL	4157	1.37
RESOLVED	294454	96.90
Grand Total	303859	100



From the above Outcome distribution chart, it can be observed that the dataset contains more than 96 percent **Resolved** records and less than 2 percent **Fatal** records. For the purpose of this analysis, **Active** records are filtered from the dataset as they yield no meaning in prediction. On careful examination after filtering out Active records, the fields ***Currently Hospitalized, Currently in ICU, Currently Intubated*** has value as No, regardless of Resolved or Fatal. Therefore, it makes no sense in considering these fields for the classification.

Even after filtering Active records, the Resolved records ratio is humungous in comparison to Fatal, which may emerge in strong bias towards one outcome.

In order to mitigate the afore mentioned imbalance concern, two different approaches are followed and the accuracy in both approaches are compared for better performance.

1) **Under Sampling**: In this approach all the records from the dataset with **Fatal** values are combined with a random sample of **Resolved** records with sample size near to Fatal values count
2) **Class Weights:** In this approach a new field termed OutcomeWeights is added to dataset and the entire records are considered for the classification. This new field holds more weight to Fatal value and less weight for resolved while applying algorithm to reduce the bias due to imbalance.

Feature Engineering

As discussed in Dataset exploration part, all Active records are filtered out and Age Group field with missing values are removed from dataset.

In above discussed both the approaches, the factors considered for classifying a record as Fatal or Resolved are as below:
Outbreak Associated, Age Group, Source of Infection, Client Gender, Ever Hospitalized, Ever in ICU, Ever Intubated.

All these were categorical variables along with our Target label which does not allow us in applying the random forest algorithm. Spark MLlib's StringIndexer class is used generate numerical values for the corresponding categorical values for each column and these values are stored in new fields, suffixing IDX to each field.

Now the new features are
Outbreak Associated IDX, Age Group IDX, Source of Infection IDX, Client Gender IDX, Ever Hospitalized IDX, Ever in ICU IDX, Ever Intubated IDX.

New Target Label: *OutcomeIDX*

Class Weights: Below is the code snippet used to add weights to the target label values.

```
def balance(data: DataFrame): DataFrame = {

   val resolved = data.filter(data("OutcomeIDX") === 0).count
   val dataSize = data.count
   val balancingFactor = (dataSize - resolved).toDouble / dataSize

   val weights = udf { d: Double =>
     if (d == 0.0) {
       1 * balancingFactor
     }
     else {
       (1 * (1.0 - balancingFactor))
     }
   }

   val covid19_weighted = covid19_df2.withColumn("OutcomeWeights",
weights(covid19_df1("OutcomeIDX")))
   covid19_weighted
 }
```

Under Sampling: Below is the code snippet utilised in downsizing the resolved records to the size of fatal records

```
val resolved_df = covid19_df2.filter(col("Outcome") === "RESOLVED")
val fatal_df = covid19_df2.filter(col("Outcome") === "FATAL")
val ratio:Double =  (resolved_df.count()/fatal_df.count())
val r:Double = (1/ratio).toDouble
val sampled_resolved_df = resolved_df.sample(false,r, 42)
val combined_sampled_df = sampled_resolved_df.unionAll(fatal_df)
```

Algorithm Application and Results Interpretartion

Random Forest algorithm is applied for in both the approaches for classification of target label Outcome based on the features selected and the performance is measured.

Accuracy is used for the performance measurement, which is calculated from the confusion matrix depicting the number of false positives and false negatives along with true predictions made.
Class Weights approach yielded in accuracy of 0.927 which is 92.7% accurate and Under Sampling approach yielded in accuracy of 0.946 which is 94.6% accurate. Both the approaches yielded in almost same accuracy. However, undersampling approach had a little edge over class weights approach.

Class Weights Accuracy

```
scala> val accuracy = print("Accuracy for Class Weights Approach: " + evaluator.evaluate(predictions))
Accuracy for Class Weights Approach: 0.9280095672825885accuracy: Unit = ()
```

Under Sampling Approach Accuracy

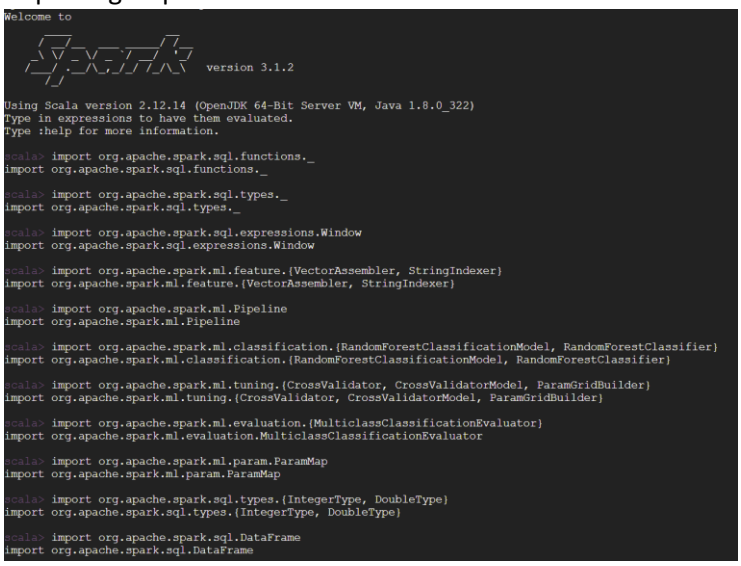
```
scala> val accuracy = print("Accuracy for Under Sampling Approach: " + evaluator.evaluate(predictions))
Accuracy for Under Sampling Approach: 0.9462809917355371accuracy: Unit = ()

scala>
```

Please refer Appendix section below for the full source code and execution screensnaps. Full source code has been provided in separate files.

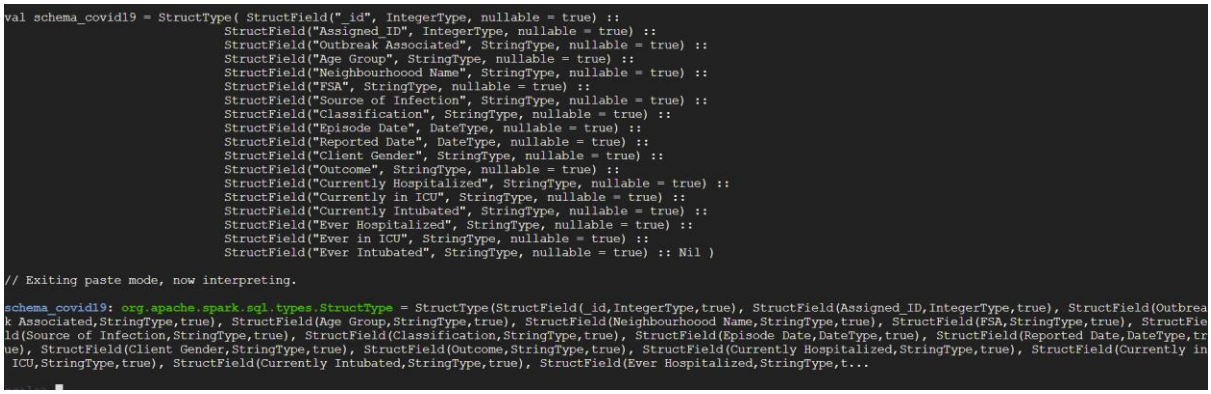
Appendix

Class Weights Approach: Here are the screenshots for the code execution to predict fatalities using class weights approach

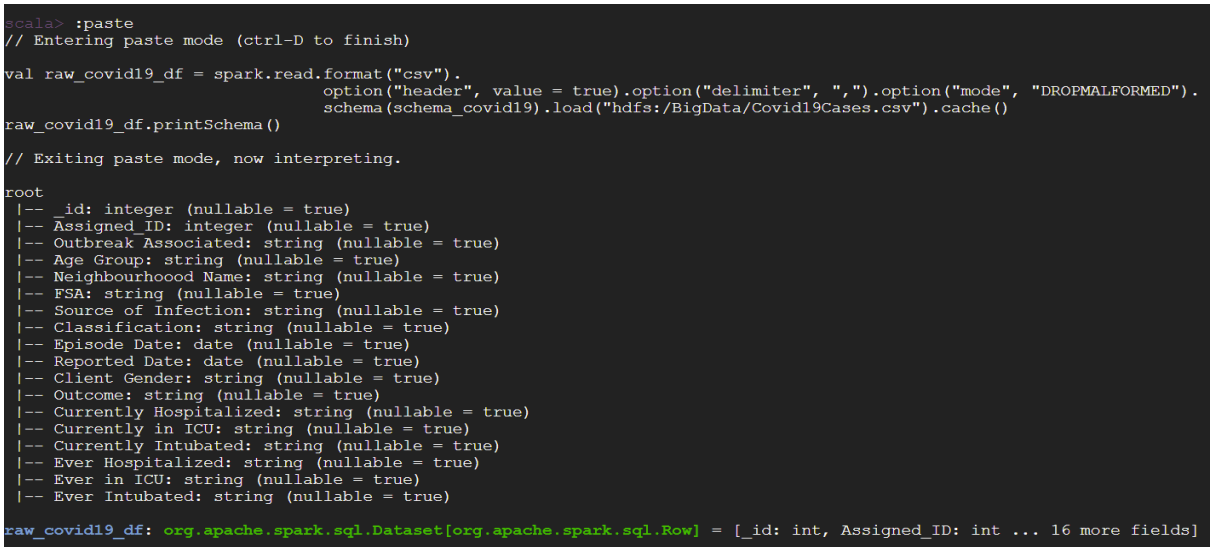
1) Importing required modules



2) Schema Definition



3) Reading file into dataframe



4) Looking at top 2 records

```
scala> raw_covid19_df.show(2)
+---+-----------+------------------+-----------------+------------------+---+-------------------+--------------+------------+-------------+-------------+--------+---
-----------------+-----------------+-----------------+----------------+------------+-------------+
|_id|Assigned_ID|Outbreak Associated|     Age Group|Neighbourhoood Name|FSA|Source of Infection|Classification|Episode Date|Reported Date|Client Gender| Outcome|Cu
rently Hospitalized|Currently in ICU|Currently Intubated|Ever Hospitalized|Ever in ICU|Ever Intubated|
+---+-----------+------------------+-----------------+------------------+---+-------------------+--------------+------------+-------------+-------------+--------+---
-----------------+-----------------+-----------------+----------------+------------+-------------+
|  1|          1|          Sporadic|50 to 59 Years|    Willowdale East|M2N|             Travel|     CONFIRMED|  2020-01-22|   2020-01-23|       FEMALE|RESOLVED|
                 No|               No|                No|              No|         No|           No|
|  2|          2|          Sporadic|50 to 59 Years|    Willowdale East|M2N|             Travel|     CONFIRMED|  2020-01-21|   2020-01-23|         MALE|RESOLVED|
                 No|               No|                No|             Yes|         No|           No|
+---+-----------+------------------+-----------------+------------------+---+-------------------+--------------+------------+-------------+-------------+--------+---
-----------------+-----------------+-----------------+----------------+------------+-------------+
only showing top 2 rows
```

5) Checking count of NULLs and their percentage

```
scala> for( i <- raw_covid19_df.columns){
     |     println(i+" "+raw_covid19_df.filter(raw_covid19_df(i).isNull || raw_covid19_df(i) === "" ).count()+" "+
     |     100*(raw_covid19_df.filter(raw_covid19_df(i).isNull || raw_covid19_df(i) === "" ).count())/(raw_covid19_df.count()))
     | }
_id 0 0
Assigned_ID 0 0
Outbreak Associated 0 0
Age Group 300 0
Neighbourhoood Name 7977 2
FSA 4296 1
Source of Infection 0 0
Classification 0 0
Episode Date 0 0
Reported Date 0 0
Client Gender 0 0
Outcome 0 0
Currently Hospitalized 0 0
Currently in ICU 0 0
Currently Intubated 0 0
Ever Hospitalized 0 0
Ever in ICU 0 0
Ever Intubated 0 0
```

6) Filtering **Outcome**: Active and **Age Group:** NULL records

```
scala> val covid19_df = raw_covid19_df.filter(col("Outcome").isin("RESOLVED","FATAL")).filter(col("Age Group").isNotNull)
covid19_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 16 more fields]
```

7) Creating a new field: **OutcomeIDX** for Target Label assigning Outcome categorical values to numeric values.

```
val indexer = new StringIndexer()
  .setInputCol("Outcome")
  .setOutputCol("OutcomeIDX")

val covid19_df1 = indexer.fit(covid19_df).transform(covid19_df)

// Exiting paste mode, now interpreting.

indexer: org.apache.spark.ml.feature.StringIndexer = strIdx_2e039218d552
covid19_df1: org.apache.spark.sql.DataFrame = [_id: int, Assigned_ID: int ... 17 more fields]
```

8) Creating equivalent numeric fields for the features being used in classification

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

var f_indexers = new Array[org.apache.spark.ml.PipelineStage](0)
val featuresList = List("Outbreak Associated","Age Group","Source of Infection","Client Gender","Ever Hospitalized",
                    "Ever in ICU","Ever Intubated")

for (feature <- featuresList){
    print(feature)
    val f_indexer = new StringIndexer().setInputCol(feature).setOutputCol(feature+ " IDX")
    print(f_indexer)
    f_indexers = f_indexers :+ f_indexer
}
f_indexers

val fpipeline =  new Pipeline()
  .setStages(f_indexers)

val covid19_df2= fpipeline.fit(covid19_df1).transform(covid19_df1)


// Exiting paste mode, now interpreting.

<pastie>:48: warning: a pure expression does nothing in statement position; multiline expressions may require enclosing parentheses
f_indexers
^
Outbreak AssociatedstrIdx_a5f9123c068bAge GroupstrIdx_8073b9745335Source of InfectionstrIdx_42f13074368eClient GenderstrIdx_34064af65eb2Ever Hospitalize
4192f961Ever in ICUstrIdx_953ab8083047Ever IntubatedstrIdx_c129afdd2188f_indexers: Array[org.apache.spark.ml.PipelineStage] = Array(strIdx_a5f9123c068b,
b9745335, strIdx_42f13074368e, strIdx_34064af65eb2, strIdx_98764192f961, strIdx_953ab8083047, strIdx_c129afdd2188)
featuresList: List[String] = List(Outbreak Associated, Age Group, Source of Infection, Client Gender, Ever Hospitalized, Ever in ICU, Ever Intubated)
fpipeline: org.apache.spark.ml.Pipeline = pipeline_7936d9e47ca5
covid19_df2: org.apache.spark.sql.DataFrame = [_id: int, Assigned_ID: int ... 24 more fields]
```

9) Checking top 2 records after the addition of new fields

```
scala> covid19_df2.filter(col("Outcome")==="FATAL").show(2)
22/04/16 12:04:25 WARN org.apache.spark.sql.catalyst.util.package: Truncated the string representation of a plan since it was too large. This behavior can be adjust
ed by setting 'spark.sql.debug.maxToStringFields'.
+---+-----------+------------------+-------------+--------------------+---+-------------------+--------------+------------+-------------+-------------+-------+----
-----------------+----------------+-------------------+------------------+------------+-------------+----------+----------------------+-------------+-----------------
---------+------------------+---------------------+----------------+-----------------+
|_id|Assigned_ID|Outbreak Associated|    Age Group|Neighbourhoood Name|FSA|Source of Infection|Classification|Episode Date|Reported Date|Client Gender|Outcome|Curr
ently Hospitalized|Currently in ICU|Currently Intubated|Ever Hospitalized|Ever in ICU|Ever Intubated|OutcomeIDX|Outbreak Associated IDX|Age Group IDX|Source of Infe
ction IDX|Client Gender IDX|Ever Hospitalized IDX|Ever in ICU IDX|Ever Intubated IDX|
+---+-----------+------------------+-------------+--------------------+---+-------------------+--------------+------------+-------------+-------------+-------+----
-----------------+----------------+-------------------+------------------+------------+-------------+----------+----------------------+-------------+-----------------
---------+------------------+---------------------+----------------+-----------------+
| 77|         80|          Sporadic|70 to 79 Years|   Victoria Village|M4A|             Travel|     CONFIRMED|  2020-03-11|   2020-03-13|         MALE|  FATAL|
                 No|              No|                 No|               Yes|         Yes|            No|       1.0|                   0.0|          6.0|
      7.0|               1.0|                1.0|            1.0|               0.0|
|263|        278|          Sporadic|60 to 69 Years|            Niagara|M5V|          Community|     CONFIRMED|  2020-03-16|   2020-03-22|         MALE|  FATAL|
                 No|              No|                 No|               Yes|         Yes|           Yes|       1.0|                   0.0|          5.0|
      1.0|               1.0|                1.0|            1.0|               1.0|
+---+-----------+------------------+-------------+--------------------+---+-------------------+--------------+------------+-------------+-------------+-------+----
-----------------+----------------+-------------------+------------------+------------+-------------+----------+----------------------+-------------+-----------------
---------+------------------+---------------------+----------------+-----------------+
only showing top 2 rows
```

10) Adding weights column for the Target label to balance

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

// Adding Weights Column to balance the 'Outcome' Weightage equally for both the values

def balance(data: DataFrame): DataFrame = {

    val resolved = data.filter(data("OutcomeIDX") === 0).count
    val dataSize = data.count
    val balancingFactor = (dataSize - resolved).toDouble / dataSize

    val weights = udf { d: Double =>
      if (d == 0.0) {
        1 * balancingFactor
      }
      else {
        (1 * (1.0 - balancingFactor))
      }
    }

    val covid19_weighted = covid19_df2.withColumn("OutcomeWeights", weights(covid19_df1("OutcomeIDX")))
    covid19_weighted
  }

val covid19_weighted = balance(covid19_df2)


// Exiting paste mode, now interpreting.

balance: (data: org.apache.spark.sql.DataFrame)org.apache.spark.sql.DataFrame
covid19_weighted: org.apache.spark.sql.DataFrame = [_id: int, Assigned_ID: int ... 25 more fields]
```

11) Checking 2 records after weights column addition

```
scala> covid19_weighted.filter(col("Outcome")==="FATAL").show(2)
+---+-----------+------------------+-------------+--------------------+---+-------------------+--------------+------------+-------------+-------------+-------+----
-----------------+----------------+-------------------+------------------+------------+-------------+----------+----------------------+-------------+-----------------
---------+------------------+---------------------+----------------+-----------------+------------------+
|_id|Assigned_ID|Outbreak Associated|    Age Group|Neighbourhoood Name|FSA|Source of Infection|Classification|Episode Date|Reported Date|Client Gender|Outcome|Curr
ently Hospitalized|Currently in ICU|Currently Intubated|Ever Hospitalized|Ever in ICU|Ever Intubated|OutcomeIDX|Outbreak Associated IDX|Age Group IDX|Source of Infe
ction IDX|Client Gender IDX|Ever Hospitalized IDX|Ever in ICU IDX|Ever Intubated IDX|    OutcomeWeights|
+---+-----------+------------------+-------------+--------------------+---+-------------------+--------------+------------+-------------+-------------+-------+----
-----------------+----------------+-------------------+------------------+------------+-------------+----------+----------------------+-------------+-----------------
---------+------------------+---------------------+----------------+-----------------+------------------+
| 77|         80|          Sporadic|70 to 79 Years|   Victoria Village|M4A|             Travel|     CONFIRMED|  2020-03-11|   2020-03-13|         MALE|  FATAL|
                 No|              No|                 No|               Yes|         Yes|            No|       1.0|                   0.0|          6.0|
      7.0|               1.0|                1.0|            1.0|               0.0|0.9860671671805872|
|263|        278|          Sporadic|60 to 69 Years|            Niagara|M5V|          Community|     CONFIRMED|  2020-03-16|   2020-03-22|         MALE|  FATAL|
                 No|              No|                 No|               Yes|         Yes|           Yes|       1.0|                   0.0|          5.0|
      1.0|               1.0|                1.0|            1.0|               1.0|0.9860671671805872|
+---+-----------+------------------+-------------+--------------------+---+-------------------+--------------+------------+-------------+-------------+-------+----
-----------------+----------------+-------------------+------------------+------------+-------------+----------+----------------------+-------------+-----------------
---------+------------------+---------------------+----------------+-----------------+------------------+
only showing top 2 rows
```

12) Assembling all features into a single vector

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val assembler = new VectorAssembler()
 .setInputCols(Array("Outbreak Associated IDX","Age Group IDX","Source of Infection IDX","Client Gender IDX",
                     "Ever Hospitalized IDX",
                    "Ever in ICU IDX","Ever Intubated IDX"))
 .setOutputCol("assembled-features")


// Exiting paste mode, now interpreting.

assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_4b8e089fd0d0, handleInvalid=error, numInputCols=7
```

13) Calling Random Forest with Weights column, Setting Pipeline stages, defining evaluator and setting hyper parameters for paramgrid and defining cross validator.

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val rf = new RandomForestClassifier()
 .setFeaturesCol("assembled-features")
 .setLabelCol("OutcomeIDX")
 .setWeightCol("OutcomeWeights")
 .setSeed(42)

val pipeline = new Pipeline()
  .setStages(Array(assembler, rf))

val evaluator = new MulticlassClassificationEvaluator()
  .setLabelCol("OutcomeIDX")
  .setPredictionCol("prediction")
  .setMetricName("accuracy")


val paramGrid = new ParamGridBuilder()
  .addGrid(rf.maxDepth, Array(3, 5))
  .addGrid(rf.impurity, Array("entropy","gini")).build()

val cross_validator = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(evaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(3)

// Exiting paste mode, now interpreting.

rf: org.apache.spark.ml.classification.RandomForestClassifier = rfc_4512c3607a12
pipeline: org.apache.spark.ml.Pipeline = pipeline_a6650ecbafe9
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=mcEval_afa1f5c47ddb,
cLabel=0.0, beta=1.0, eps=1.0E-15
paramGrid: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        rfc_4512c3607a12-impurity: entropy,
        rfc_4512c3607a12-maxDepth: 3
}, {
        rfc_4512c3607a12-impurity: gini,
        rfc_4512c3607a12-maxDepth: 3
}, {
        rfc_4512c3607a12-impurity: entropy,
        rfc_4512c3607a12-maxDepth: 5
```

14) Split Training and Test Data in 80-20 ratio, generate model using Train data, Pass Test data to the model created.

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val Array(trainingData, testData) = covid19_weighted.randomSplit(Array(0.8, 0.2), 42)

val cvModel = cross_validator.fit(trainingData)

val predictions = cvModel.transform(testData)


// Exiting paste mode, now interpreting.

trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 25 more fields]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 25 more fields]
cvModel: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_22e7cdeacb91, bestModel=pipeline_a6650ecbafe9, numFolds=3
predictions: org.apache.spark.sql.DataFrame = [_id: int, Assigned_ID: int ... 29 more fields]
```

15) Model Accuracy

```
scala> val accuracy = print("Accuracy for Class Weights Approach: " + evaluator.evaluate(predictions))
Accuracy for Class Weights Approach: 0.9280095672825885accuracy: Unit = ()
```

Under sampling Approach: Here are the screenshots for the code execution to predict fatalities using under sampling approach

1-9) Steps 1 to 9 are same as the one followed in Class Weights Approach.

10) Under Sampling Resolved cases to the size of Fatal records to attain balance to dataset

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val resolved_df = covid19_df2.filter(col("Outcome") === "RESOLVED")
val fatal_df = covid19_df2.filter(col("Outcome") === "FATAL")
val ratio:Double =  (resolved_df.count()/fatal_df.count())

val r:Double = (1/ratio).toDouble
val sampled_resolved_df = resolved_df.sample(false,r, 42)
val combined_sampled_df = sampled_resolved_df.unionAll(fatal_df)


// Exiting paste mode, now interpreting.

resolved_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 24 more fields]
fatal_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 24 more fields]
ratio: Double = 70.0
r: Double = 0.014285714285714285
sampled_resolved_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 24 more fields]
combined_sampled_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 24 more fields]
```

11) Assembler, Random Forest with no weights column, pipeline and evaluator

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val assembler = new VectorAssembler()
 .setInputCols(Array("Outbreak Associated IDX","Age Group IDX","Source of Infection IDX","Client Gender IDX","Ever Hospitalized IDX",
                     "Ever in ICU IDX","Ever Intubated IDX"))
 .setOutputCol("assembled-features")

val rf = new RandomForestClassifier()
 .setFeaturesCol("assembled-features")
 .setLabelCol("OutcomeIDX")
 .setSeed(42)

val pipeline = new Pipeline()
  .setStages(Array(assembler, rf))

val evaluator = new MulticlassClassificationEvaluator()
  .setLabelCol("OutcomeIDX")
  .setPredictionCol("prediction")
  .setMetricName("accuracy")


// Exiting paste mode, now interpreting.

assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_ef8af695c38d, handleInvalid=error, numInputCols=7
rf: org.apache.spark.ml.classification.RandomForestClassifier = rfc_740ddd088cec
pipeline: org.apache.spark.ml.Pipeline = pipeline_a0bf73458a3b
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=mcEval_1a0bc763976f, metric
cLabel=0.0, beta=1.0, eps=1.0E-15
```

12) Param Grid and Cross Validator

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val paramGrid = new ParamGridBuilder()
  .addGrid(rf.maxDepth, Array(3, 5))
  .addGrid(rf.impurity, Array("entropy","gini")).build()

val cross_validator = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(evaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(3)


// Exiting paste mode, now interpreting.

paramGrid: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        rfc_740ddd088cec-impurity: entropy,
        rfc_740ddd088cec-maxDepth: 3
}, {
        rfc_740ddd088cec-impurity: entropy,
        rfc_740ddd088cec-maxDepth: 5
}, {
        rfc_740ddd088cec-impurity: gini,
        rfc_740ddd088cec-maxDepth: 3
}, {
        rfc_740ddd088cec-impurity: gini,
        rfc_740ddd088cec-maxDepth: 5
})
cross_validator: org.apache.spark.ml.tuning.CrossValidator = cv_8383b39478ff
```

13) Split Training and Test Data in 80-20 ratio, generate model using Train data, Pass Test data to the model created.

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val Array(trainingData, testData) = combined_sampled_df.randomSplit(Array(0.8, 0.2), 40)

val cvModel = cross_validator.fit(trainingData)

val predictions = cvModel.transform(testData)


// Exiting paste mode, now interpreting.

trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 24 more fields]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [_id: int, Assigned_ID: int ... 24 more fields]
cvModel: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_8383b39478ff, bestModel=pipeline_a0bf73458a3b, numFolds=3
predictions: org.apache.spark.sql.DataFrame = [_id: int, Assigned_ID: int ... 28 more fields]
```

14) Model Accuracy

```
scala> val accuracy = print("Accuracy for Under Sampling Approach: " + evaluator.evaluate(predictions))
Accuracy for Under Sampling Approach: 0.9462809917355371accuracy: Unit = ()
```