In [1]:

```python
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

Out[1]:

```
'C:\\Spark\\spark-3.2.1-bin-hadoop3.2'
```

In [2]:

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder.config("spark.driver.memory", "15g").appName("Titanic").getOrCreate()

spark
```

Out[2]:

SparkSession - in-memory

SparkContext

Spark UI

Version	v3.2.1
Master	local[*]
AppName	Titanic

In [3]:

```python
import pandas as pd
df = (spark.read.format("csv").options(header="true")
    .load(r"C:\Users\Rajesh Kumar\Documents\train.csv"))
```

In [4]:

```python
df.limit(20).toPandas()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	None	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925	None	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05	None	S
5	6	0	3	Moran, Mr. James	male	None	0	0	330877	8.4583	None	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075	None	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333	None	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708	None	C
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1	1	PP 9549	16.7	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
12	13	0	3	Saundercock, Mr. William Henry	male	20	0	0	A/5. 2151	8.05	None	S
13	14	0	3	Andersson, Mr. Anders Johan	male	39	1	5	347082	31.275	None	S
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14	0	0	350406	7.8542	None	S
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55	0	0	248706	16	None	S
16	17	0	3	Rice, Master. Eugene	male	2	4	1	382652	29.125	None	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	None	0	0	244373	13	None	S
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female	31	1	0	345763	18	None	S
19	20	1	3	Masselmani, Mrs. Fatima	female	None	0	0	2649	7.225	None	C

In [5]:

```python
df.count()
```

Out[5]:

```
891
```

In [6]:

```python
df.printSchema()
```

```
root
 |-- PassengerId: string (nullable = true)
 |-- Survived: string (nullable = true)
 |-- Pclass: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- SibSp: string (nullable = true)
 |-- Parch: string (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: string (nullable = true)
 |-- Cabin: string (nullable = true)
 |-- Embarked: string (nullable = true)
```

In [7]:

```python
df.groupBy("Sex").count().show()

df.groupBy("Survived").count().show()

df.groupBy('Sex','Survived').count().orderBy('Sex').show()

df.groupBy('Pclass','Survived').count().orderBy('Pclass').show()
```

```
+------+-----+
|   Sex|count|
+------+-----+
|female|  314|
|  male|  577|
+------+-----+

+--------+-----+
|Survived|count|
+--------+-----+
|       0|  549|
|       1|  342|
+--------+-----+

+------+--------+-----+
|   Sex|Survived|count|
+------+--------+-----+
|female|       0|   81|
|female|       1|  233|
|  male|       1|  109|
|  male|       0|  468|
+------+--------+-----+

+------+--------+-----+
|Pclass|Survived|count|
+------+--------+-----+
|     1|       0|   80|
|     1|       1|  136|
|     2|       0|   97|
|     2|       1|   87|
|     3|       1|  119|
|     3|       0|  372|
+------+--------+-----+
```

In [8]:

```python
from pyspark.sql.functions import *

def null_value_calc(df):
    null_columns_counts = []
    numRows = df.count()
    for k in df.columns:
        nullRows = df.where(col(k).isNull()).count()
        if(nullRows > 0):
            temp = k,nullRows,(nullRows/numRows)*100
            null_columns_counts.append(temp)
    return(null_columns_counts)

null_columns_calc_list = null_value_calc(df)
spark.createDataFrame(null_columns_calc_list, ['Column_Name', 'Null_Values_Count','Null_Value_Percent']).show()
```

```
+-----------+-----------------+-------------------+
|Column_Name|Null_Values_Count| Null_Value_Percent|
+-----------+-----------------+-------------------+
|        Age|              177| 19.865319865319865|
|      Cabin|              687|  77.10437710437711|
|   Embarked|                2|0.22446689113355783|
+-----------+-----------------+-------------------+
```

In [9]:

```python
df.select(mean('age')).show()
```

```
+-----------------+
|         avg(age)|
+-----------------+
|29.69911764705882|
+-----------------+
```

In [10]:

```python
df.select('name').show(10,False)
```

```
+---------------------------------------------------+
|name                                               |
+---------------------------------------------------+
|Braund, Mr. Owen Harris                            |
|Cumings, Mrs. John Bradley (Florence Briggs Thayer)|
|Heikkinen, Miss. Laina                             |
|Futrelle, Mrs. Jacques Heath (Lily May Peel)       |
|Allen, Mr. William Henry                           |
|Moran, Mr. James                                   |
|McCarthy, Mr. Timothy J                            |
|Palsson, Master. Gosta Leonard                     |
|Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)  |
|Nasser, Mrs. Nicholas (Adele Achem)                |
+---------------------------------------------------+
only showing top 10 rows
```

In [11]:

```python
df = df.withColumn('initial', regexp_extract(col('name'), '([a-zA-Z]+)\.',1))


df = df.replace(['Mlle','Mme', 'Ms', 'Dr','Major','Lady','Countess','Jonkheer','Col','Rev','Capt','Sir','Don'],
               ['Miss','Miss','Miss','Mr','Mr',  'Mrs',  'Mrs',  'Other',  'Other','Other','Mr','Mr','Mr'])
```

In [13]:

```python
df = df.withColumn('Age', when((df['initial'] == 'Miss') & (df['Age'].isNull()), 22).otherwise(df['Age']))
df = df.withColumn('Age', when((df['initial'] == 'Other') & (df['Age'].isNull()), 45).otherwise(df['Age']))
df = df.withColumn('Age', when((df['initial'] == 'Master') & (df['Age'].isNull()), 6).otherwise(df['Age']))
df = df.withColumn('Age', when((df['initial'] == 'Mr') & (df['Age'].isNull()), 32).otherwise(df['Age']))
df = df.withColumn('Age', when((df['initial'] == 'Mrs') & (df['Age'].isNull()), 33).otherwise(df['Age']))

df = df.na.fill({'Embarked' : 'S'})
df = df.drop('Cabin')


df.select('initial','name','age').filter("initial == 'Miss'").show(100,False)
```

```
+-------+----------------------------------------------+----+
|initial|name                                          |age |
+-------+----------------------------------------------+----+
|Miss   |Heikkinen, Miss. Laina                        |26  |
|Miss   |Sandstrom, Miss. Marguerite Rut               |4   |
|Miss   |Bonnell, Miss. Elizabeth                      |58  |
|Miss   |Vestrom, Miss. Hulda Amanda Adolfina          |14  |
|Miss   |"McGowan, Miss. Anna ""Annie"""               |15  |
|Miss   |Palsson, Miss. Torborg Danira                 |8   |
|Miss   |"O'Dwyer, Miss. Ellen ""Nellie"""             |22  |
|Miss   |Glynn, Miss. Mary Agatha                      |22  |
|Miss   |Vander Planke, Miss. Augusta Maria            |18  |
|Miss   |Nicola-Yarred, Miss. Jamila                   |14  |
|Miss   |Laroche, Miss. Simonne Marie Anne Andree      |3   |
|Miss   |Devaney, Miss. Margaret Delia                 |19  |
|Miss   |O'Driscoll, Miss. Bridget                     |22  |
|Miss   |Rugg, Miss. Emily                             |21  |
|Miss   |West, Miss. Constance Mirium                  |5   |
|Miss   |Icard, Miss. Amelie                           |38  |
|Miss   |Andersson, Miss. Erna Alexandra               |17  |
|Miss   |Goodwin, Miss. Lillian Amy                    |16  |
|Miss   |Dowdell, Miss. Elizabeth                      |30  |
|Miss   |McDermott, Miss. Brigdet Delia                |22  |
|Miss   |Ilett, Miss. Bertha                           |17  |
|Miss   |Fortune, Miss. Mabel Helen                    |23  |
|Miss   |Petranec, Miss. Matilda                       |28  |
|Miss   |Salkjelsvik, Miss. Anna Kristine              |21  |
|Miss   |Moran, Miss. Bertha                           |22  |
|Miss   |Zabour, Miss. Hileni                          |14.5|
|Miss   |Jussila, Miss. Katriina                       |20  |
|Miss   |Attalah, Miss. Malake                         |17  |
|Miss   |Andersson, Miss. Ellis Anna Maria             |2   |
|Miss   |Webber, Miss. Susan                           |32.5|
|Miss   |Peter, Miss. Anna                             |22  |
|Miss   |Newsom, Miss. Helen Monypeny                  |19  |
|Miss   |Nysten, Miss. Anna Sofia                      |22  |
|Miss   |"Ford, Miss. Robina Maggie ""Ruby"""          |9   |
|Miss   |"Gilnagh, Miss. Katherine ""Katie"""          |16  |
|Miss   |Johnson, Miss. Eleanor Ileen                  |1   |
|Miss   |Isham, Miss. Ann Elizabeth                    |50  |
|Miss   |Sage, Miss. Constance Gladys                  |22  |
|Miss   |Kink-Heilmann, Miss. Luise Gretchen           |4   |
|Miss   |Andersen-Jensen, Miss. Carla Christine Nielsine|19  |
|Miss   |Lurette, Miss. Elise                          |58  |
|Miss   |"Madigan, Miss. Margaret ""Maggie"""          |22  |
|Miss   |"Yrois, Miss. Henriette (""Mrs Harbeck"")"    |24  |
|Miss   |Strom, Miss. Telma Matilda                    |2   |
|Miss   |"Carr, Miss. Helen ""Ellen"""                 |16  |
|Miss   |Cameron, Miss. Clear Annie                    |35  |
|Miss   |Newell, Miss. Madeleine                       |31  |
|Miss   |Honkanen, Miss. Eliina                        |27  |
|Miss   |Bazzani, Miss. Albina                         |32  |
|Miss   |Lefebre, Miss. Mathilde                       |22  |
|Miss   |Asplund, Miss. Lillian Gertrud                |5   |
|Miss   |Harknett, Miss. Alice Phoebe                  |22  |
|Miss   |"Collyer, Miss. Marjorie ""Lottie"""          |8   |
|Miss   |Zabour, Miss. Thamine                         |22  |
|Miss   |"Murphy, Miss. Katherine ""Kate"""            |22  |
|Miss   |Lindahl, Miss. Agda Thorilda Viktoria         |25  |
|Miss   |Cherry, Miss. Gladys                          |30  |
|Miss   |Ward, Miss. Anna                              |35  |
|Miss   |Henry, Miss. Delia                            |22  |
|Miss   |Bissette, Miss. Amelia                        |35  |
|Miss   |"Healy, Miss. Hanora ""Nora"""                |22  |
|Miss   |Andrews, Miss. Kornelia Theodosia             |63  |
|Miss   |Lindblom, Miss. Augusta Charlotta             |45  |
|Miss   |Connolly, Miss. Kate                          |22  |
|Miss   |"Barber, Miss. Ellen ""Nellie"""              |26  |
|Miss   |Haas, Miss. Aloisia                           |24  |
|Miss   |Allison, Miss. Helen Loraine                  |2   |
|Miss   |"Kelly, Miss. Anna Katherine ""Annie Kate"""  |22  |
|Miss   |Keane, Miss. Nora A                           |22  |
|Miss   |Fleming, Miss. Margaret                       |22  |
|Miss   |Francatelli, Miss. Laura Mabel                |30  |
|Miss   |Hays, Miss. Margaret Bechstein                |24  |
|Miss   |Ryerson, Miss. Emily Borie                    |18  |
|Miss   |Nilsson, Miss. Helmina Josefina               |26  |
|Miss   |Wick, Miss. Mary Natalie                      |31  |
|Miss   |Slayter, Miss. Hilda Mary                     |30  |
|Miss   |Young, Miss. Marie Grice                      |36  |
|Miss   |Hippach, Miss. Jean Gertrude                  |16  |
|Miss   |McCoy, Miss. Agnes                            |22  |
|Miss   |Burns, Miss. Elizabeth Margaret               |41  |
|Miss   |Fortune, Miss. Alice Elizabeth                |24  |
|Miss   |"Brown, Miss. Amelia ""Mildred"""             |24  |
|Miss   |Smith, Miss. Marion Elsie                     |40  |
|Miss   |Bowerman, Miss. Elsie Edith                   |22  |
|Miss   |Funk, Miss. Annie Clemmer                     |38  |
|Miss   |McGovern, Miss. Mary                          |22  |
|Miss   |"Mockler, Miss. Helen Mary ""Ellie"""         |22  |
|Miss   |Jermyn, Miss. Annie                           |22  |
|Miss   |Aubart, Mme. Leontine Pauline                 |24  |
|Miss   |Palsson, Miss. Stina Viola                    |3   |
|Miss   |Landergren, Miss. Aurora Adelia               |22  |
|Miss   |Bidois, Miss. Rosalie                         |42  |
|Miss   |"Nakid, Miss. Maria (""Mary"")"               |1   |
|Miss   |Buss, Miss. Kate                              |36  |
|Miss   |Lehmann, Miss. Bertha                         |17  |
|Miss   |Newell, Miss. Marjorie                        |23  |
|Miss   |Olsson, Miss. Elina                           |31  |
|Miss   |Jussila, Miss. Mari Aina                      |21  |
|Miss   |Oreskovic, Miss. Marija                       |20  |
|Miss   |Lefebre, Miss. Ida                            |22  |
+-------+----------------------------------------------+----+
only showing top 100 rows
```

In [14]:

```python
df.printSchema()

df = df.withColumn('Family_size',col('Parch') + col('SibSp'))
df.groupBy('Family_size').count().show()
```

```
root
 |-- PassengerId: string (nullable = true)
 |-- Survived: string (nullable = true)
 |-- Pclass: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- SibSp: string (nullable = true)
 |-- Parch: string (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: string (nullable = true)
 |-- Embarked: string (nullable = false)
 |-- initial: string (nullable = true)

+-----------+-----+
|Family_size|count|
+-----------+-----+
|        0.0|  537|
|        7.0|    6|
|        1.0|  161|
|        4.0|   15|
|        3.0|   29|
|        2.0|  102|
|       10.0|    7|
|        6.0|   12|
|        5.0|   22|
+-----------+-----+
```

In [15]:

```python
df = df.withColumn('Alone', lit('0'))
```

In [16]:

```python
df = df.withColumn('Alone', when(df['Age'] == 0, 1).otherwise(df['Alone']))
```

In [17]:

```python
df = df.drop("PassengerId","Name","Ticket","Cabin","Embarked","Sex","initial")
```

In [18]:

```python
# Data Prep function
def MLClassifierDFPrep(df,input_columns,dependent_var,treat_outliers=True,treat_neg_values=True):

    # change label (class variable) to string type to prep for reindexing
    # Pyspark is expecting a zero indexed integer for the label column.
    # Just incase our data is not in that format... we will treat it by using the StringIndexer built in method
    renamed = df.withColumn("label_str", df[dependent_var].cast(StringType())) #Rename and change to string type
    indexer = StringIndexer(inputCol="label_str", outputCol="label") #Pyspark is expecting the this naming convention
    indexed = indexer.fit(renamed).transform(renamed)

    # Convert all string type data in the input column list to numeric
    # Otherwise the Algorithm will not be able to process it
    numeric_inputs = []
    string_inputs = []
    for column in input_columns:
        if str(indexed.schema[column].dataType) == 'StringType':
            indexer = StringIndexer(inputCol=column, outputCol=column+"_num")
            indexed = indexer.fit(indexed).transform(indexed)
            new_col_name = column+"_num"
            string_inputs.append(new_col_name)
        else:
            numeric_inputs.append(column)

    if treat_outliers == True:
        print("We are correcting for non normality now!")
        # empty dictionary d
        d = {}
        # Create a dictionary of quantiles
        for col in numeric_inputs:
            d[col] = indexed.approxQuantile(col,[0.01,0.99],0.25) #if you want to make it go faster increase the last number
        #Now fill in the values
        for col in numeric_inputs:
            skew = indexed.agg(skewness(indexed[col])).collect() #check for skewness
            skew = skew[0][0]
            # This function will floor, cap and then log+1 (just in case there are 0 values)
            if skew > 1:
                indexed = indexed.withColumn(col, \
                log(when(df[col] < d[col][0],d[col][0])\
                .when(indexed[col] > d[col][1], d[col][1])\
                .otherwise(indexed[col] ) +1).alias(col))
                print(col+" has been treated for positive (right) skewness. (skew =)",skew,")")
            elif skew < -1:
                indexed = indexed.withColumn(col, \
                exp(when(df[col] < d[col][0],d[col][0])\
                .when(indexed[col] > d[col][1], d[col][1])\
                .otherwise(indexed[col] )).alias(col))
                print(col+" has been treated for negative (left) skewness. (skew =",skew,")")


    # Produce a warning if there are negative values in the dataframe that Naive Bayes cannot be used.
    # Note: we only need to check the numeric input values since anything that is indexed won't have negative values
    minimums = df.select([min(c).alias(c) for c in df.columns if c in numeric_inputs]) # Calculate the mins for all columns in the df
    min_array = minimums.select(array(numeric_inputs).alias("mins")) # Create an array for all mins and select only the input cols
    df_minimum = min_array.select(array_min(min_array.mins)).collect() # Collect golobal min as Python object
    df_minimum = df_minimum[0][0] # Slice to get the number itself

    features_list = numeric_inputs + string_inputs
    assembler = VectorAssembler(inputCols=features_list,outputCol='features')
    output = assembler.transform(indexed).select('features','label')

#     final_data = output.select('features','label') #drop everything else

    # Now check for negative values and ask user if they want to correct that?
    if df_minimum < 0:
        print(" ")
        print("WARNING: The Naive Bayes Classifier will not be able to process your dataframe as it contains negative values")
        print(" ")

    if treat_neg_values == True:
        print("You have opted to correct that by rescaling all your features to a range of 0 to 1")
        print(" ")
        print("We are rescaling you dataframe....")
        scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

        # Compute summary statistics and generate MinMaxScalerModel
        scalerModel = scaler.fit(output)

        # rescale each feature to range [min, max].
        scaled_data = scalerModel.transform(output)
        final_data = scaled_data.select('label','scaledFeatures')
        final_data = final_data.withColumnRenamed('scaledFeatures','features')
        print("Done!")

    else:
        print("You have opted not to correct that therefore you will not be able to use to Naive Bayes classifier")
        print("We will return the dataframe unscaled.")
        final_data = output

    return final_data
```

In [19]:

```python
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import MinMaxScaler

# input_columns = df.columns # Collect the column names as a list
# input_columns = input_columns[8:] # keep only relevant columns: from column 8 until the end
input_columns = df.columns[1:]
dependent_var = 'Survived'

final_data = MLClassifierDFPrep(df,input_columns,dependent_var,False, False)
final_data.limit(5).toPandas()
```

```
You have opted not to correct that therefore you will not be able to use to Naive Bayes classifier
We will return the dataframe unscaled.
```

Out[19]:

	features	label
0	[1.0, 0.0, 1.0, 1.0, 0.0, 11.0, 0.0]	0.0
1	[1.0, 1.0, 25.0, 1.0, 0.0, 224.0, 0.0]	1.0
2	(0.0, 0.0, 12.0, 0.0, 0.0, 6.0, 0.0)	1.0
3	[1.0, 1.0, 14.0, 1.0, 0.0, 37.0, 0.0]	1.0
4	(0.0, 0.0, 14.0, 0.0, 0.0, 0.0, 0.0)	0.0

In [20]:

```python
final_data.columns
```

Out[20]:

```
['features', 'label']
```

In [21]:

```python
train,test = final_data.randomSplit([0.8,0.2])
```

In [22]:

```python
from pyspark.ml.classification import *
classifier = LogisticRegression()
fitModel = classifier.fit(train)
```

In [23]:

```python
from pyspark.ml.evaluation import *

predictionAndLabels = fitModel.transform(test)
# predictionAndLabels = predictionAndLabels.predictions.select('label','prediction')
Bin_evaluator = BinaryClassificationEvaluator(rawPredictionCol='prediction') #labelCol='label'
auc = Bin_evaluator.evaluate(predictionAndLabels)
print("AUC:",auc)
```

```
AUC: 0.5793408086986068
```

In [24]:

```python
predictions = fitModel.transform(test)
# predictions.show()
MC_evaluator = MulticlassClassificationEvaluator(metricName="accuracy") # predictionCol="prediction",
accuracy = (MC_evaluator.evaluate(predictions))*100
print("Accuracy: {0:.2f}".format(accuracy),"%") #     print("Test Error = %g " % (1.0 - accuracy))
print(" ")
```

```
Accuracy: 67.48 %
```

In [25]:

```python
def ClassTrainEval(classifier,features,classes,train,test):

    def FindMtype(classifier):
        # Intstantiate Model
        M = classifier
        # Learn what it is
        Mtype = type(M).__name__

        return Mtype

    Mtype = FindMtype(classifier)

    def InstanceFitModel(Mtype,classifier,classes,features,train):

        if Mtype == "OneVsRest":
            # instantiate the base classifier.
            lr = LogisticRegression()
            # instantiate the One Vs Rest Classifier.
            OVRclassifier = OneVsRest(classifier=lr)
#             fitModel = OVRclassifier.fit(train)
            # Add parameters of your choice here:
            paramGrid = ParamGridBuilder() \
                .addGrid(lr.regParam, [0.1, 0.01]) \
                .build()
            #Cross Validator requires the following parameters:
            crossval = CrossValidator(estimator=OVRclassifier,
                                      estimatorParamMaps=paramGrid,
                                      evaluator=MulticlassClassificationEvaluator(),
                                      numFolds=2) # 3 is best practice
            # Run cross-validation, and choose the best set of parameters.
            fitModel = crossval.fit(train)
            return fitModel
        if Mtype == "MultilayerPerceptronClassifier":
            # specify layers for the neural network:
            # input layer of size features, two intermediate of features+1 and same size as features
            # and output of size number of classes
            # Note: crossvalidator cannot be used here
            features_count = len(features[0][0])
            layers = [features_count, features_count+1, features_count, classes]
            MPC_classifier = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)
            fitModel = MPC_classifier.fit(train)
            return fitModel
        if Mtype in("LinearSVC","GBTClassifier") and classes != 2: # These classifiers currently only accept binary classification
            print(Mtype," could not be used because PySpark currently only accepts binary classification data for this algorithm")
            return
        if Mtype in("LogisticRegression","NaiveBayes","RandomForestClassifier","GBTClassifier","LinearSVC","DecisionTreeClassifier"):

            # Add parameters of your choice here:
            if Mtype in("LogisticRegression"):
                paramGrid = (ParamGridBuilder() \
#                              .addGrid(classifier.regParam, [0.1, 0.01]) \
                             .addGrid(classifier.maxIter, [10, 15,20])
                             .build())

            # Add parameters of your choice here:
            if Mtype in("NaiveBayes"):
                paramGrid = (ParamGridBuilder() \
                             .addGrid(classifier.smoothing, [0.0, 0.2, 0.4, 0.6]) \
                             .build())

            # Add parameters of your choice here:
            if Mtype in("RandomForestClassifier"):
                paramGrid = (ParamGridBuilder() \
                               .addGrid(classifier.maxDepth, [2, 5, 10])
#                                .addGrid(classifier.maxBins, [5, 10, 20])
#                                .addGrid(classifier.numTrees, [5, 20, 50])
                             .build())

            # Add parameters of your choice here:
            if Mtype in("GBTClassifier"):
                paramGrid = (ParamGridBuilder() \
#                              .addGrid(classifier.maxDepth, [2, 5, 10, 20, 30]) \
#                              .addGrid(classifier.maxBins, [10, 20, 40, 80, 100]) \
                             .addGrid(classifier.maxIter, [10, 15,50,100])
                             .build())

            # Add parameters of your choice here:
            if Mtype in("LinearSVC"):
                paramGrid = (ParamGridBuilder() \
                             .addGrid(classifier.maxIter, [10, 15]) \
                             .addGrid(classifier.regParam, [0.1, 0.01]) \
                             .build())

            # Add parameters of your choice here:
            if Mtype in("DecisionTreeClassifier"):
                paramGrid = (ParamGridBuilder() \
#                              .addGrid(classifier.maxDepth, [2, 5, 10, 20, 30]) \
                             .addGrid(classifier.maxBins, [10, 20, 40, 80, 100]) \
                             .build())

            #Cross Validator requires all of the following parameters:
            crossval = CrossValidator(estimator=classifier,
                                      estimatorParamMaps=paramGrid,
                                      evaluator=MulticlassClassificationEvaluator(),
                                      numFolds=3) # 3 + is best practice
            # Fit Model: Run cross-validation, and choose the best set of parameters.
            fitModel = crossval.fit(train)
            return fitModel

    fitModel = InstanceFitModel(Mtype,classifier,classes,features,train)

    # Print feature selection metrics
    if fitModel is not None:

        if Mtype in("OneVsRest"):
            # Get Best Model
            BestModel = fitModel.bestModel
            print(" ")
            print('\033[1m' + Mtype + '\033[0m')
            # Extract list of binary models
            models = BestModel.models
            for model in models:
                print('\033[1m' + 'Intercept: '+ '\033[0m',model.intercept,'\033[1m' + '\nCoefficients:'+ '\033[0m',model.coefficients)

        if Mtype == "MultilayerPerceptronClassifier":
            print("")
            print('\033[1m' + Mtype," Weights"+ '\033[0m')
            print('\033[1m' + "Model Weights: "+ '\033[0m',fitModel.weights.size)
            print("")

        if Mtype in("DecisionTreeClassifier", "GBTClassifier","RandomForestClassifier"):
            # FEATURE IMPORTANCES
            # Estimate of the importance of each feature.
            # Each feature’s importance is the average of its importance across all trees
            # in the ensemble The importance vector is normalized to sum to 1.
            # Get Best Model
            BestModel = fitModel.bestModel
            print(" ")
            print('\033[1m' + Mtype," Feature Importances"+ '\033[0m')
            print("(Scores add up to 1)")
            print("Lowest score is the least important")
            print(" ")
            featureimportances = BestModel.featureImportances.toArray()
            print(featureimportances)

            if Mtype in("DecisionTreeClassifier"):
                global DT_featureimportances
                DT_featureimportances = BestModel.featureImportances.toArray()
                global DT_BestModel
                DT_BestModel = BestModel
            if Mtype in("GBTClassifier"):
                global GBT_featureimportances
                GBT_featureimportances = BestModel.featureImportances.toArray()
                global GBT_BestModel
                GBT_BestModel = BestModel
            if Mtype in("RandomForestClassifier"):
                global RF_featureimportances
                RF_featureimportances = BestModel.featureImportances.toArray()
                global RF_BestModel
                RF_BestModel = BestModel

        if Mtype in("LogisticRegression"):
            # Get Best Model
            BestModel = fitModel.bestModel
            print(" ")
            print('\033[1m' + Mtype," Coefficient Matrix"+ '\033[0m')
            print("You should compares these relative to eachother")
            print("Coefficients: \n" + str(BestModel.coefficientMatrix))
            print("Intercept: " + str(BestModel.interceptVector))
            global LR_coefficients
            LR_coefficients = BestModel.coefficientMatrix.toArray()
            global LR_BestModel
            LR_BestModel = BestModel

        if Mtype in("LinearSVC"):
            # Get Best Model
            BestModel = fitModel.bestModel
            print(" ")
            print('\033[1m' + Mtype," Coefficients"+ '\033[0m')
            print("You should compares these relative to eachother")
            print("Coefficients: \n" + str(BestModel.coefficients))
            global LSVC_coefficients
            LSVC_coefficients = BestModel.coefficients.toArray()
            global LSVC_BestModel
            LSVC_BestModel = BestModel


    # Set the column names to match the external results dataframe that we will join with later:
    columns = ['Classifier', 'Result']

    if Mtype in("LinearSVC","GBTClassifier") and classes != 2:
        Mtype = [Mtype] # make this a list
        score = ["N/A"]
        result = spark.createDataFrame(zip(Mtype,score), schema=columns)
    else:
        predictions = fitModel.transform(test)
        MC_evaluator = MulticlassClassificationEvaluator(metricName="accuracy") # predictionCol="prediction",
        accuracy = (MC_evaluator.evaluate(predictions))*100
        Mtype = [Mtype] # make this a string
        score = [str(accuracy)] #make this a string and convert to a list
        result = spark.createDataFrame(zip(Mtype,score), schema=columns)
        result = result.withColumn('Result',result.Result.substr(0, 5))

    return result
    #Also returns the fit model important scores or p values
```

In [32]:

```python
from pyspark.ml.classification import *
from pyspark.ml.evaluation import *
from pyspark.sql import functions
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

classifiers = [
#                 LogisticRegression()
#                ,OneVsRest()
#                ,LinearSVC()
#                ,NaiveBayes()
#               RandomForestClassifier()
#                ,GBTClassifier()
#                ,DecisionTreeClassifier()
                MultilayerPerceptronClassifier()
              ]

train,test = final_data.randomSplit([0.7,0.3])
features = final_data.select(['features']).collect()
# Learn how many classes there are in order to specify evaluation type based on binary or multi and turn the df into an object
class_count = final_data.select(countDistinct("label")).collect()
classes = class_count[0][0]

#set up your results table
columns = ['Classifier', 'Result']
vals = [("Place Holder","N/A")]
results = spark.createDataFrame(vals, columns)

for classifier in classifiers:
    new_result = ClassTrainEval(classifier,features,classes,train,test)
    results = results.union(new_result)
results = results.where("Classifier!='Place Holder'")
results.show(100,False)
```

MultilayerPerceptronClassifier Weights
Model Weights: 143

```
+------------------------------+------+
|Classifier                    |Result|
+------------------------------+------+
|MultilayerPerceptronClassifier|69.31 |
+------------------------------+------+
```

In [35]:

```python
# predictions.show()

MC_evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
accuracy = MC_evaluator.evaluate(predictions)
print(accuracy)
```

```
0.6748466257668712
```