Titanic: Machine Learning from Disaster

This competition involves prediction of people who survived or died, during the sinking of the mighty Titanic ship.

This report involves using of the Spark's Python API – pyspark, coded using the databricks platform, to perform state-of-art machine learning algorithms on the titanic dataset. Logistic Regression, Decision Tree and Random Forest Classifiers have been used to predict the number of people who survived/died.

1.0: Data Preparation:

Dataset Description:

The dataset has 12 columns and 891 observations.

The target variable is Survived. 1 is survived and 0 is died.

These are the following variables/attributes:

Variable Name	Data Type	Variable Description	Variable Type	Target Variable
PassengerId	int	Passenger's Unique ID	Scale	N
Survived	int	Survived (1) or not (0)	Categorical	Y
Pclass	int	Class of Travel	Categorical	N
Name	string	Passenger's name	Character	N
Sex	string	Passenger's gender	Categorical	N
Age	int	Passenger's age	Scale	N
SibSp	int	Number of Sibling/Spouse aboard	Scale	N
Parch	int	Number of Parent/Child aboard	Scale	N
Ticket	string	Ticket number	Character	N
Fare	double	Ticket price	Scale	N
Cabin	string	Alotted cabin	Character	N
Embarked	string	Port aboarded at	Character	N

First five rows of the data:

```
1 display(training.head(5))
```

(4) Spark Jobs

passengerid	survived	pclass	name	sex	age	sibsp	parch	fare
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	7.25
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	71.2833
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	7.925
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	53.1
5	0	3	Allen, Mr. William Henry	male	35	0	0	8.05

Count of people survived/died:

We can see that only 342 people survived and 549 people died.

```
1 training.groupBy("survived").count().show()
```

▸ (5) Spark Jobs

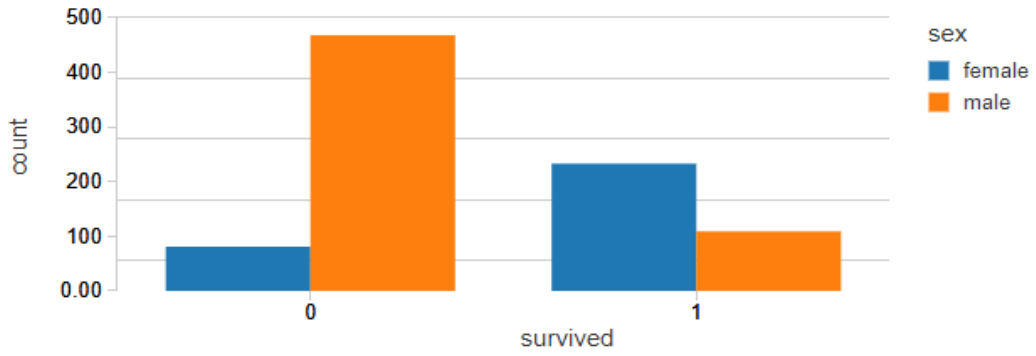
```
+--------+-----+
|survived|count|
+--------+-----+
|       0|  549|
|       1|  342|
+--------+-----+
```

Survival rate amongst gender:

We can see a stark difference in the numbers of survived/died in terms of gender, that females survived more than males and the number of females who died is also very less, compared to the astonishing number of male deaths.

```
1 display(training.groupBy("sex","survived").count())
```

▸ (5) Spark Jobs



Count of males and females on the ship:

```
1 training.groupBy("sex").count().show()
```

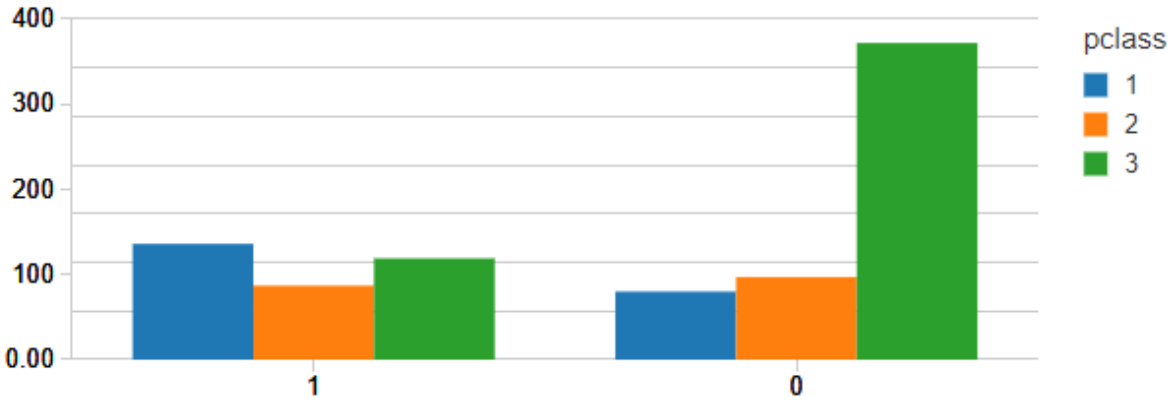
▸ (5) Spark Jobs

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

Survival rate among different passenger classes:

```
#class
display(training.groupBy("pclass","survived").count())
```

(5) Spark Jobs



We can see that, passengers of class 3 died the most.

Checking for null values and imputing:

There are 177 null values in the age column

```
from pyspark.sql.functions import isnan, when, count, col

training.select([count(when(col(c).isNull(), c)).alias(c) for c in training.columns]).show()
```

(2) Spark Jobs

passengerid	survived	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked
0	0	0	0	0	177	0	0	0	0	687	2

The columns age, cabin and embarked have missing values. I will be dropping the columns cabin and embarked at a later stage, hence will not be dealing with those columns right now.

However, the age column is a very important determining factor for a person to survive, hence we will see how we can impute the missing values.

As there were many people of different age ranges, we should not impute the age with just the mean of the column. One idea is to find out the age ranges from the titles of their name. We can see that titles such as ‘Mr’, ‘Mrs, ‘Ms’, ‘Master’ can be found from the name column.

Performing a regex function on the name column and extracting the titles into a new column “title”.

```
training = training.withColumn("title",regexp_extract(col("name"),"([A-Za-z]+)(\\.)",1))
```

Distinct titles are retrieved as following,

```
1  training.select('title').distinct().show()
```

▸ (5) Spark Jobs

```
+--------+
|   title|
+--------+
|     Don|
|    Miss|
|Countess|
|     Col|
|     Rev|
|    Lady|
|  Master|
|     Mme|
|    Capt|
|      Mr|
|      Dr|
|     Mrs|
|     Sir|
|Jonkheer|
|    Mlle|
|   Major|
|      Ms|
+--------+
```

We can rename the rarer titles to the more commonly used titles as:

```
training = training.replace(['Mlle','Ms','Mme'],
                            ['Miss','Miss','Miss'])
```

```
training:  pyspark.sql.dataframe.DataFrame = [passengerid: string, survived: string ... 11 more fields]
mand took 0.06 seconds -- by d17129392@mydit.ie at 4/7/2019, 11:33:29 PM on c2
```

```
10
training = training.replace(['Dr','Major','Jonkheer','Col','Rev','Capt','Sir','Don'],
                            ['Mr','Mr',  'Mr',  'Mr',  'Mr',  'Mr','Mr','Mr'])
```

```
training:  pyspark.sql.dataframe.DataFrame = [passengerid: string, survived: string ... 11 more fields]
mand took 0.10 seconds -- by d17129392@mydit.ie at 4/7/2019, 11:33:29 PM on c2
```

```
11
training = training.replace(['Lady','Countess'],
                            ['Mrs','Mrs'])
```

Getting the mean age of passengers by their title and imputing the missing age values based on the mean value of their title:

```
1  display(training.groupby('title').mean('age'))
```

▸ (5) Spark Jobs

title	avg(age)
Miss	21.84
Master	4.472222222222222
Mr	33.004784688995215
Mrs	35.981818181818184

```
#Imputing with means based on titles
training=training.withColumn("age",when((training["title"] == "Miss")
                                        & (training["age"].isNull()), 22).otherwise(training["age"]))
training=training.withColumn("age",when((training["title"] == "Master")
                                        & (training["age"].isNull()), 5).otherwise(training["age"]))
training=training.withColumn("age",when((training["title"] == "Mr")
                                        & (training["age"].isNull()), 33).otherwise(training["age"]))
training=training.withColumn("age",when((training["title"] == "Mrs")
                                        & (training["age"].isNull()), 36).otherwise(training["age"]))
```

Recoding column ‘sex’ as gender into an integer variable:

```
indexer = StringIndexer(inputCol="sex", outputCol="gender")
training = indexer.fit(training).transform(training)
```

Dropping of columns:

Dropped the following columns and the reasons for dropping:

passengerid – These are unique index integer values, which will provide no insight for our target label

name, title – These are strings which won’t be useful in regression.

Sex – This variable has been recoded as a dummy integer variable ‘gender’.

Ticket – This is the ticket ID, a string mixed with numbers and alphabets, which will again not add value to our classification task.

Cabin – This has a lot of missing values, and imputing them will be a little hard.

Embarked – This is the port where a passenger boarded the ship and a location of boarding won’t add a value to our classification task.

Hypothesis Testing:

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import ChiSquareTest

r = ChiSquareTest.test(training, "predictor_variables", "survived").head()
print("pValues: " + str(r.pValues))
print("degreesOfFreedom: " + str(r.degreesOfFreedom))
print("statistics: " + str(r.statistics))
```

▸ (2) Spark Jobs

```
pValues: [0.0,7.95127653941e-07,1.55858104656e-06,9.70352642105e-05,9.70352642105e-05,0.0]
degreesOfFreedom: [2, 70, 6, 6, 6, 1]
statistics: [102.888988757,142.091522441,37.2717929152,27.9257840602,27.9257840602,263.050574071]
```

The variables pclass and gender are statistically significant to the variable survived, as the p-value of those variables are less than 0.05. The other variables seem to have non-statistically significant correlation.

Correlation:

```
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=training.columns, outputCol=vector_col)
df_vector = assembler.transform(training).select(vector_col)
# get correlation matrix
r1 = Correlation.corr(df_vector, vector_col)
r1.collect()[0]["pearson({})".format(vector_col)].values
```

```
Out[207]:
array([ 1.        , -0.33848104, -0.09147608, -0.0353225 ,  0.08162941,
        0.54335138, -0.33848104,  1.        , -0.33967052,  0.08308136,
        0.01844267, -0.13190049, -0.09147608, -0.33967052,  1.        ,
       -0.26747835, -0.19894736, -0.11998362, -0.0353225 ,  0.08308136,
       -0.26747835,  1.        ,  0.4148377 ,  0.11463081,  0.08162941,
        0.01844267, -0.19894736,  0.4148377 ,  1.        ,  0.24548896,
        0.54335138, -0.13190049, -0.11998362,  0.11463081,  0.24548896,  1.        ])
```

As we can see, no two variables have a correlation more than 0.8, which means the variables are not similar.

Summary statistics:

```
#Summary stats
training.describe().show()
```

▸ (1) Spark Jobs

summary	survived	pclass	age	sibsp	parch	fare	gender
count	891	891	891	891	891	891	891
mean	0.3838383838383838	2.308641975308642	29.826038159371492	0.5230078563411896	0.38159371492704824	0.38159371492704824	0.35241301907968575
stddev	0.48659245426485753	0.8360712409770491	13.290447524811878	1.1027434322934315	0.8060572211299488	0.8060572211299488	0.4779900708960982
min	0	1	0	0	0	0.0	0.0
max	1	3	80	8	6	6.0	1.0

Normalization:

```
feature = VectorAssembler(inputCols=training.columns[1:], outputCol="predictor_variables")
vector = feature.transform(training)
training= feature.transform(training)
```

v result

21

```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol="predictor_variables", outputCol="scaledFeatures",
                        withStd=True, withMean=False)

# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(training)

# Normalize each feature to have unit standard deviation.
training = scalerModel.transform(training)
```

The independent variables are scaled using the StandardScaler function.

Scaling is done to normalize the variables, so that they have standard deviation of one and mean of value zero.

Splitting of dataset:

```
training=training.select('survived','scaledFeatures')

survived=training.where(col('survived')==1)

died=training.where(col('survived')==0)

(trainSurvived,testSurvived) = survived.randomSplit([0.8,0.2])

(trainDied,testDied) = died.randomSplit([0.8,0.2])

trainData = trainSurvived.unionByName(trainDied)

#trainData = DataFrame.unionAll([trainSurvived,trainDied])
testData = testSurvived.unionByName(testDied)
```

- Splitting the dataset initially as survived and died by the target variable ‘survived’, where survived has a value of one and died with a value of zero.
- Splitting the data of survived into trainSurvived and testSurvived in an 80/20 split.
- Splitting the data of died into trainDied and testDied in an 80/20 split.
- Then, combining the trainSurvived and trainDied as the training dataset ‘trainData’ and combining the testSurvived and testDied as the test dataset ‘testData’.

2.0: Implementation:

Logistic Regression:

lr = LogisticRegression(labelCol = 'survived', featuresCol = 'scaledFeatures')

lrModel = lr.fit(trainData)

lr_prediction = lrModel.transform(testData)

lr_prediction.select("prediction", "survived", "scaledFeatures").show()

evaluator = MulticlassClassificationEvaluator(labelCol="survived", predictionCol="prediction", metricName="accuracy")

```
+----------+--------+--------------------+
|prediction|survived|      scaledFeatures|
+----------+--------+--------------------+
|       1.0|       1|(6,[0,1],[1.19607...|
|       1.0|       1|(6,[0,1],[1.19607...|
|       1.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[2.39214...|
|       0.0|       1|(6,[0,1],[2.39214...|
```

Decision Tree classifier:

from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(labelCol="survived", featuresCol="scaledFeatures")

dt_model = dt.fit(trainData)

dt_prediction = dt_model.transform(testData)

dt_prediction.select("prediction", "survived", "scaledFeatures").show()

```
+----------+--------+--------------------+
|prediction|survived|      scaledFeatures|
+----------+--------+--------------------+
|       1.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[2.39214...|
|       0.0|       1|(6,[0,1],[3.58821...|
|       0.0|       1|(6,[0,1],[3.58821...|
```

Random forest classifier:

```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(labelCol="survived", featuresCol="scaledFeatures", numTrees=10)

rf_model = rf.fit(trainData)

rf_prediction = rf_model.transform(testData)

rf_prediction.select("prediction", "survived", "scaledFeatures").show()
```

```
+----------+--------+--------------------+
|prediction|survived|      scaledFeatures|
+----------+--------+--------------------+
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[1.19607...|
|       0.0|       1|(6,[0,1],[2.39214...|
|       0.0|       1|(6,[0,1],[3.58821...|
```

3.0: Evaluation:

Evaluation of Logistic Regression:

Accuracy:

```
lr_accuracy = evaluator.evaluate(lr_prediction)
print("Accuracy of LogisticRegression is = %g"% (lr_accuracy))
print("Test Error of LogisticRegression = %g " % (1.0 - lr_accuracy))
```

```
▸ (2) Spark Jobs
Accuracy of LogisticRegression is = 0.775862
Test Error of LogisticRegression = 0.224138
```

```
#Summary

trainingSummary = lrModel.summary

# Obtain the objective per iteration

objectiveHistory = trainingSummary.objectiveHistory

print("objectiveHistory:")

for objective in objectiveHistory:    print(objective)
```

```
objectiveHistory:
0.6644316205084055
0.5405384332468853
0.49139387473553037
0.47046995028356897
0.46248932752265753
```

ROC:

```
# Obtain the receiver-operating characteristic as a dataframe and areaUnderROC.
trainingSummary.roc.show()
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))
```

(7) Spark Jobs

```
+--------------------+--------------------+
|                 FPR|                 TPR|
+--------------------+--------------------+
|                 0.0|                 0.0|
|                 0.0|0.014652014652014652|
|                 0.0|0.040293040293040296|
|                 0.0|0.054945054945054944|
|                 0.0| 0.06593406593406594|
```

Evaluation measures:

```
falsePositiveRate = trainingSummary.weightedFalsePositiveRate
truePositiveRate = trainingSummary.weightedTruePositiveRate
fMeasure = trainingSummary.weightedFMeasure()
precision = trainingSummary.weightedPrecision
recall = trainingSummary.weightedRecall
print("FPR: %s\nTPR: %s\nF-measure: %s\nPrecision: %s\nRecall: %s"
      % (falsePositiveRate, truePositiveRate, fMeasure, precision, recall))
```

```
FPR: 0.22948720019849728
TPR: 0.810320781032078
F-measure: 0.8084561184136699
Precision: 0.8085146077281986
Recall: 0.810320781032078
```

Decision tree classifier evaluation:

```
dt_accuracy = evaluator.evaluate(dt_prediction)
print("Accuracy of DecisionTreeClassifier is = %g"% (dt_accuracy))
print("Test Error of DecisionTreeClassifier = %g " % (1.0 - dt_accuracy))
```

▸ (2) Spark Jobs

```
Accuracy of DecisionTreeClassifier is = 0.810345
Test Error of DecisionTreeClassifier = 0.189655
```

Random Forest classifier Evaluation:

```
#Evaluating accuracy of RandomForestClassifier.
rf_accuracy = evaluator.evaluate(rf_prediction)
print("Accuracy of RandomForestClassifier is = %g"% (rf_accuracy))
print("Test Error of RandomForestClassifier  = %g " % (1.0 - rf_accuracy))
```

▸ (2) Spark Jobs

```
Accuracy of RandomForestClassifier is = 0.804598
Test Error of RandomForestClassifier  = 0.195402
```

4.0: Discussion:

Logistic regression was chosen because it is one of the state-of-art algorithms, primarily used for binary classification.

Decision Tree generally performs well in binary classifications and is very easy to understand the concept behind how the result was arrived at.

Random Forest Classifier is nothing but an ensemble of Decision Trees, which splits the data into test and train randomly (hence the name random forest). This was chosen because, it arrives at the result in multiple different ways.

From the three models, we can clearly see that the Decision Tree classifier seems to be doing a good job at predicting, with an accuracy of 0.81, while the Logistic Regression and Random Forest Classifier have an accuracy of 0.79 and 0.8 respectively.

Thus, based on the accuracy obtained by the different models, the Decision Tree Classifier performs the best for predicting the number of people who survived/died from the Titanic disaster dataset.