Assignment 3: Logistic Regression Using Spark

Daanish Ahmed

DATA 650 9041

Spring 2018

dsahmed2334@yahoo.com

Professor Elena Gortcheva

UMUC

April 8, 2018

Part 1: Logistic Regression on Titanic Data

Discuss the method results, including the classification accuracy for training and test set.

In this part of the assignment, I implemented a logistic regression model to analyze a dataset on Titanic victims. My goal is to classify victims based on whether they survived. There are several measures I used to evaluate the model results: confusion matrices, accuracy and error rates, precision, recall, F1 scores, area under the ROC curve, and area under the PR curve. Confusion matrices show the number of predicted cases of 1 and 0 compared to the actual number of 1's and 0's. An accurate model will have a high number of true positives and true negatives, while having fewer false positives and false negatives. Accuracy and error rates indicate the percentage of correct and incorrect predictions, respectively.

Precision is the number of true positives divided by the sum of true positives and false positives, while recall is the number of true positives divided by the number of true positives and false negatives (Koehrsen, 2018). In this case, precision would be the number of predicted survivors that were actual survivors, while recall would be the number of actual survivors that were correctly predicted as survivors. The F1 score represents a balance between precision and recall, in which a higher score indicates a stronger model (Brownlee, 2014). The Receiver Operating Characteristic (ROC) curve shows a plot of the true positive rate over the false positive rate, while the Precision-Recall (PR) curve shows the precision plotted against the recall ("Differences between Receiver," 2014). A higher area under the curve is preferable for both measures. An optimal model will have an area under the ROC curve near 1.0, while a weaker model will have an area that is closer to 0.5 (Tape, n.d.).

```
# This table shows:
# 1. The number of survived passengers predicted as survived
# 2. The number of survived passengers predicted as not survived
# 3. The number of not survived passengers predicted as survived
# 4. The number of not survived passengers predicted as not survived

prediction.crosstab('label', 'prediction').show()
```

```
+----------------+---+---+
|label_prediction|0.0|1.0|
+----------------+---+---+
|             1.0| 21| 31|
|             0.0| 69| 11|
+----------------+---+---+
```

Figure 1. Confusion Matrix for Titanic Logistic Regression on Test Data.

I will first examine the confusion matrix for the test data, which shows the predictions for survivor status compared to the actual survivor statistics (see Figure 1). Based on this figure, we see that there were 31 survivors and 69 fatalities that were correctly identified. There is a total of 132 passengers in the test set, meaning that 75.8% of cases were accurately labeled. This accuracy rate is somewhat acceptable, but there is significant room for improvement. Furthermore, there are 11 false positives and 21 false negatives—the latter of which means that there was a considerable number of survivors who were classified as "dead" by this algorithm.

Next, I will compare some of the important measures within the training and test results. According to the HTML file for Part 1, this model has an accuracy rate of 83.8% for training data and 75.8% for test data. The test accuracy is lower than the training accuracy by a clear margin, which suggests that overfitting might have taken place. Likewise, the error rate is significantly higher in the test set than in the training set. The precision also decreases from 82.0% in the training set to 73.8% in the test set. This is a significant drop, but it still indicates that the model is relatively precise at identifying actual survivors. However, the recall decreases from 78.5% in the training set to only 59.6% in the test set. This is a massive decline, and it further supports the likelihood of overfitting. Also, the F1 score decreases sharply from 80.2% in the training data to

66.0% in the test data. The test F1 score leaves much room for improvement, and it has clearly been weighed down by the poor recall score.

Finally, I will examine the areas under the ROC and PR curves for the test set. The area under the ROC curve is 0.735, which is decent but unremarkable. It is approximately equidistant from the minimum area of 0.5 and the maximum area of 1.0. The area under the PR curve is 0.769, which is slightly higher than the area under the ROC curve. However, its value is still quite far from the maximum area of 1.0, which means that the current model is far from optimal.

Is logistic regression suitable for this problem? Why or Why not?

Based on my findings, the logistic regression approach used in this notebook does not produce the most effective classification results for this dataset. The decline in accuracy from 83.8% in the training set to 75.8% in the test set suggests that overfitting may have taken place. Accuracy is not the only statistic that sharply decreased—precision, recall, and the F1 score all declined by 10-20% from the training set to the test set. However, these issues might be due to the size of the dataset and the partitions used for the training and test data. The Titanic dataset used in this analysis is relatively small, since it only contains 756 cases. Furthermore, 80% of the data was partitioned to the training set and only 20% was allocated to the test set. As a result, only 132 records were assigned to the test data. It is possible that using a larger partition for the test set would reduce or eliminate the issues caused by overfitting.

One of the issues with this notebook is that no preprocessing steps were performed. Data preparation is vital for ensuring that the selected model performs well on the dataset being used. Some of the important preprocessing steps include handling missing values, outliers, and errors,

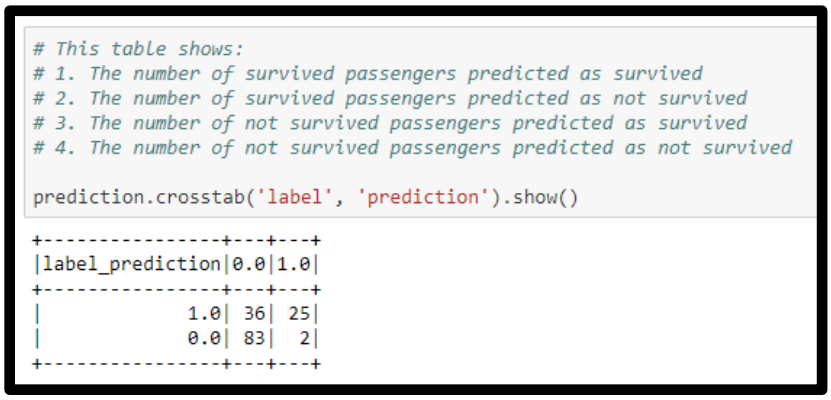
removing unnecessary variables, and checking for variable skewness. By thoroughly examining the dataset, I found that there are no missing values in any of the variables. Additionally, I noticed that the code only included the passenger class, age, and sex as independent variables when loading the data into an array. This means that the code already excluded the passenger name and ID, which are not useful for the analysis. With regards to outliers, the only remaining numeric variable to check is "age." I computed the average age as 30.4 years and the standard deviation as 14.2 years. Since the minimum age is 0.17 years and the maximum is 71 years, there are no values that are more than 3 standard deviations away from the mean. Finally, I examined the distribution of the target variable and found that there are 443 deaths and 313 survivors—indicating that the target is not heavily skewed. However, one major issue is that some of the input variables—such as passenger class and sex—are not numeric. Logistic regression requires all inputs to be numeric, which may be another reason why this algorithm does not perform well on this dataset.

What alternative machine learning methods could be suitable for this problem?

There are several other machine learning methods that could be used for this problem. Some of the possibilities include Naïve Bayes, decision trees, neural networks, support vector machines (SVM), bagging, boosting, random forests, and ensemble models. Clustering methods such as k-means clustering can be used for the dataset, but they are not well-suited for this type of problem. This is because our problem involves predicting a certain outcome, and unsupervised methods such as clustering are not used for prediction.

One alternative algorithm that I tested on this dataset was Naïve Bayes classification. This algorithm involves grouping each record into their most likely class based on the prior probabilities

for each class (James, Witten, Hastie, & Tibshirani, 2013). I imported and ran the notebook (from the ungraded exercise), which uses the Naïve Bayes algorithm instead of logistic regression. By examining the confusion matrix for the test data (see Figure 2), I noticed that the results are somewhat different from the last model. There are now 25 survivors and 83 fatalities that were correctly identified as such. This results in an overall accuracy of 74.0%, which is slightly lower than the logistic regression accuracy of 75.8%. Additionally, there are 2 false positives and 36 false negatives. This means that Naïve Bayes is more accurate than the previous model when identifying survivors, but it is much weaker when identifying non-survivors.



```
# This table shows:
# 1. The number of survived passengers predicted as survived
# 2. The number of survived passengers predicted as not survived
# 3. The number of not survived passengers predicted as survived
# 4. The number of not survived passengers predicted as not survived

prediction.crosstab('label', 'prediction').show()
```

```
+----------------+---+---+
|label_prediction|0.0|1.0|
+----------------+---+---+
|             1.0| 36| 25|
|             0.0| 83|  2|
+----------------+---+---+
```

Figure 2. Confusion Matrix for Titanic Naïve Bayes Classification on Test Data.

I will now evaluate additional results from the training and test data (see Figure 3). Firstly, the classification accuracy is 79.5% for the training set and 74.0% for the test set. Although there is a clear decline in accuracy, the difference may not be large enough to suggest that overfitting took place. Interestingly, the training precision is 93.2% and the test precision is 92.6%, which are significantly higher than the regression model's precision. Unlike the previous model, this model does not experience a significant decrease in precision from the training set to the test set. However, it does experience a large drop in recall from 54.4% in the training set to 41.0% in the test set. Although this model is more precise than the regression model, it also has a lower recall.

Still, its recall value does not decline as significantly as the previous model. Additionally, the F1 score decreases from a 68.7% training score to a 56.8% test score. The regression model had a test score of 66.0%, which is much higher than this model's F1 score.

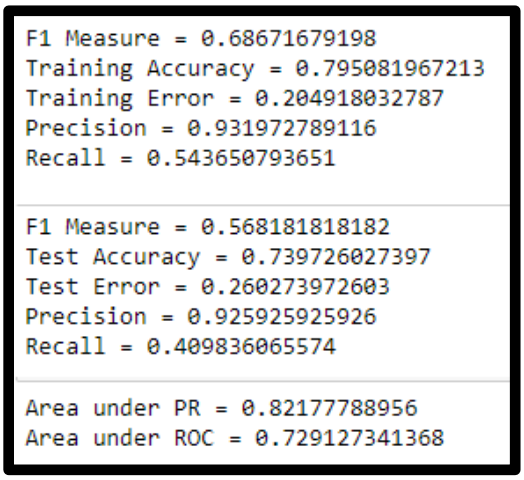


Figure 3. Naïve Bayes Classification Statistics on Training and Test Data.

According to Figure 3, the area under the PR curve is 0.822—which is clearly higher than that of the previous model (0.769). Its area is closer to 1.0 than to 0.5, which means that it is a relatively effective model. The area under the ROC curve is 0.729, which is marginally lower than the regression model's area (0.735). This value is decent but could use significant improvement. Altogether, the Naïve Bayes model is more effective than logistic regression in many ways. For instance, it has a much higher precision and a higher area under the PR curve. Additionally, it has a much smaller decline in accuracy from training to test data—suggesting that it most likely does not suffer from overfitting. Although the regression model has a higher recall and F1 score, the gap between training and test results is much higher than for the Naïve Bayes model. Of these two models, I would recommend using the Naïve Bayes approach because its training and test results are more balanced and there is less evidence for overfitting. It is likely that using a larger partition for the test data would yield even stronger results for this algorithm.

Part 2: Logistic Regression on Low Birth Weight Data

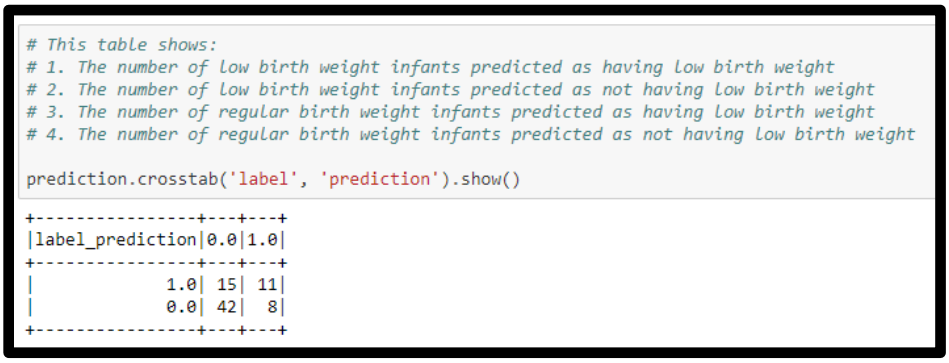
Define the purpose of the study. Explain which variable is a target variable, and which variables are the independent variables.

In this part of the assignment, I will implement a logistic regression model using Spark to analyze a dataset on low birth weight infants. Low birth weight is a major concern among newborn babies because it causes their bodies to be weaker, making it harder for them to eat, breathe, gain weight, maintain their body temperature, or fight infectious diseases ("Low Birthweight," 2014). My goal is to predict whether a baby will be born with a birth weight below 2500 grams. By identifying low birth weight cases, we will be able to study which of the included health factors (such as smoking, premature labor, and hypertension) contribute the most to these occurrences. This will allow our organization to develop solutions which will help prevent babies from being born with dangerously low birth weights.

The low birth weight dataset is very small, as it contains only 189 records and 9 variables. The target variable is low birth weight ("LOW"), which is a binary variable with the following classes: 0 indicates a normal birth weight of at least 2500 grams, and 1 indicates a low birth weight under 2500 grams. The target variable is skewed, since there are 130 cases of normal birth weight infants and only 59 cases of low birth weight infants. The independent variables include the mother's age, race, smoking status during pregnancy, history of premature labor, history of hypertension, whether she has a uterine irritability, and the number of physician visits during her first trimester (Hosmer, Lemeshow, & Sturdivant, 2013). All variables are in numeric form. The patient ID will not be included in the model because it is not useful for the study.

Discuss the method results, including the classification accuracy for training set and test set.

The logistic regression code used in this part of the assignment was slightly modified from the original notebook used in the previous part. Aside from changing the variables to match those in the new dataset, I also changed the training and test partitions to 60% and 40% respectively. This is to ensure that the test set is not too small—especially since the dataset only has 189 observations. I also used a random seed that allows me to reproduce the results. Finally, I experimented with the regression parameters to optimize the model's accuracy. I found that setting the max iterations to 50 and the "regParam" to 0.001 produced the strongest results.



```
# This table shows:
# 1. The number of low birth weight infants predicted as having low birth weight
# 2. The number of low birth weight infants predicted as not having low birth weight
# 3. The number of regular birth weight infants predicted as having low birth weight
# 4. The number of regular birth weight infants predicted as not having low birth weight

prediction.crosstab('label', 'prediction').show()
```

```
+----------------+---+---+
|label_prediction|0.0|1.0|
+----------------+---+---+
|             1.0| 15| 11|
|             0.0| 42|  8|
+----------------+---+---+
```

Figure 4. Confusion Matrix for Low Birth Weight Logistic Regression on Test Data.

First, I will examine the confusion matrix for the test results (see Figure 4). Based on this figure, there are 42 infants who were correctly categorized as having normal birth weight, and 11 infants who were correctly labeled as having low birth weight. There are 76 records in the test set, meaning that the overall test accuracy is 69.7%. This accuracy rate is not ideal, and the poor results may be due to the small dataset size as well as the skewness of the target variable. Furthermore, there are 8 false positives and 15 false negatives. This means that there were 8 normal weight babies that were falsely labeled as having low birth weight. Even worse, there were 15 low birth weight babies who were not recognized as having low birth weight.

I will now compare additional important measures within the training and test sets. Based on the HTML file for Part 2, this model has an accuracy of 78.9% in the training set and 69.7% in the test set. The error also increases drastically from 21.1% in the training set to 30.3% in the test set. Unfortunately, these findings suggest that overfitting has taken place. It seems that increasing the size of the test set was not enough to prevent overfitting, since the dataset is too small and there are too few low birth weight cases in the target variable. The precision drops extensively from 70.8% in the training set to 57.9% in the test set. Likewise, the recall decreases from 50.0% in training data to 42.3% in test data. The F1 score—which depends on both precision and recall—decreases sharply from 58.6% in the training set to 48.9% in the test set.

Lastly, I will examine the areas under the PR and ROC curves within the test set. The area under the PR curve is 0.600, which makes sense since the model suffers from weak precision and recall values. The area under the ROC curve is 0.632, which is much closer to the minimum value of 0.5 than to the maximum value of 1.0. Since the ROC curve represents the true positive rate plotted over the false positive rate, the results strongly indicate that the current model is inaccurate at predicting low birth weight cases in this dataset.

Is the logistic regression method suitable for this study? Why or why not?

Based on my findings, the current logistic regression approach did not produce desirable results. Not only did the accuracy decrease significantly from the training set to the test set, but nearly every other important measure decreased as well. However, this may not necessarily be due to the logistic regression algorithm, but it may instead be due to the dataset itself. The low birth weight dataset is extremely small, since it contains only 189 observations. Even when

allocating up to 40% of data to the test set, this only gives 76 records to the test data. Furthermore, the target variable is heavily skewed, since there are 130 normal birth weight cases and only 59 low birth weight cases. This is a problem because when data is distributed to the training and test sets, there is likely to be a different proportion of 1's and 0's in the test set than there were in the training set. One possible solution would be to sample the dataset by filtering out some of the 0's to get a more balanced distribution. However, this would lead to an even smaller dataset.

With regards to preprocessing, I examined the dataset and found that it contained no missing values. Additionally, I filtered out unimportant variables such as patient ID to ensure that only relevant inputs are included. Furthermore, all variables are already in numeric form, which is one of the requirements for using logistic regression. When checking for outliers, I found that "age" was the only variable that contained one. The average age is 23.2 years and the standard deviation is 5.3 years. The minimum age is 14 and the maximum age is 45—the latter of which is more than 3 standard deviations from the mean. But aside from this one outlier, there are no other errors in this dataset and it is otherwise well-suited for logistic regression. The key issues with this dataset are its small size and the skewness of the target variable.

Discuss the study limitations and propose the future research. Consider other models that could be suitable for this study.

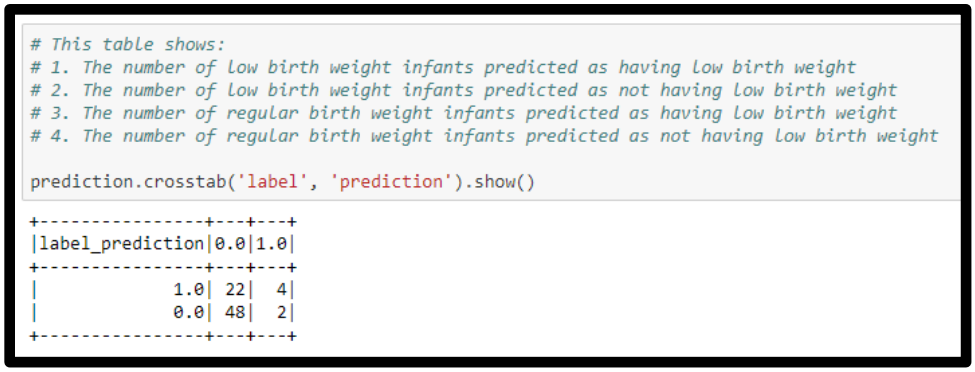
One of the main limitations in this analysis was the small size of the datasets used. Both the Titanic dataset and the low birth weight dataset have less than 800 cases—with the low birth weight data containing only 189 cases. It is difficult to obtain a good picture of the data when using very small datasets, because the data used in the training and test sets can be very different

each time. I used a random seed to ensure that the training and test sets contain the same data every time the code is run. Without this random seed, the model could produce a training accuracy of 70% and a test accuracy of 60% when executed the first time. But when executed a second time, it could have a training accuracy of 80% and a test accuracy of 55%. One recommendation would be to perform the analysis again using a larger dataset. This may require gathering a much higher volume of data. Another possible solution is to run the method multiple times without a random seed, record all important measures each time, and compute their averages at the end. This should be done until the measures converge to a certain value.

Another limitation was the skewness of the target variable. As mentioned, there were 130 cases of normal birth weight infants and only 59 cases of low birth weight infants. This skewness can cause issues when performing logistic regression and other classification methods. One proposed solution is to filter out some of the normal birth weight cases and keep all of the low birth weight cases. Although this will lead to a more balanced distribution of values, it can also drastically reduce the size of an already very small dataset. Another possible solution would be to use weighted samples when splitting the data into training and test sets. This will involve setting the rarest value (in this case '1') to take up a certain percentage of all cases, such as 40-50%. This will help produce a balanced target variable, though it will also reduce the size of the dataset. Both solutions would benefit from using a much larger dataset.

There are several other models that can be used for this problem, such as neural networks, decision trees, support vector machines, and random forests. I chose to implement two types of models: Naïve Bayes and decision tree classification. In the next few paragraphs, I will evaluate their results and compare them to my logistic regression model. First, I will discuss the results of the Naïve Bayes model on low birth weight data. This model requires loading additional packages

such as “NaiveBayes” and “MultilayerPerceptronClassifier.” To ensure that this model functions on a skewed dataset, I added the parameter “numFeatures = 32” to the “HashingTF” command. Without this parameter, the model will not identify any 1’s in the dataset. Based on the confusion matrix for test data, the model correctly identified 48 true negatives and 4 true positives (see Figure 5). This gives the model an overall accuracy of 68.4% on the test data, which is slightly lower than the logistic regression accuracy of 69.7%. The regression model already had a poor accuracy rate, meaning that this model may be even less suited to this dataset. Furthermore, the true positive rate is significantly lower than that of the previous model. Since we want to identify cases of low birth weight, this model will provide little help in its current form.



```
# This table shows:
# 1. The number of low birth weight infants predicted as having low birth weight
# 2. The number of low birth weight infants predicted as not having low birth weight
# 3. The number of regular birth weight infants predicted as having low birth weight
# 4. The number of regular birth weight infants predicted as not having low birth weight

prediction.crosstab('label', 'prediction').show()
```

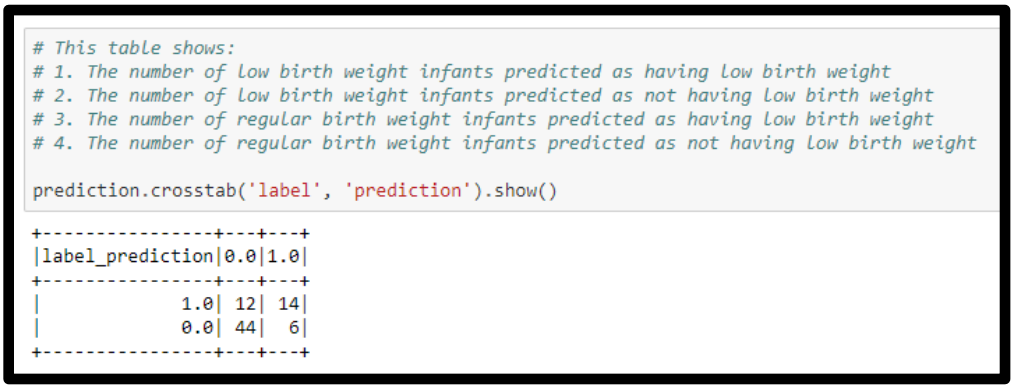
```
+----------------+---+---+
|label_prediction|0.0|1.0|
+----------------+---+---+
|             1.0| 22|  4|
|             0.0| 48|  2|
+----------------+---+---+
```

Figure 5. Confusion Matrix for Low Birth Weight Naïve Bayes Model on Test Data.

I will now compare this model’s training and test results using the HTML file for Part 2. The Naïve Bayes model’s accuracy decreased sharply from 78.1% in the training set to 68.4% in the test set. There is a similar drop when using logistic regression on this dataset, meaning that this new model is just as vulnerable to overfitting. Interestingly, the decline in accuracy when using Naïve Bayes on the Titanic dataset was not nearly as large. This indicates that the Naïve Bayes method is a better fit on the Titanic data than on the low birth weight data—which makes sense because the Titanic dataset is larger. Additionally, the precision decreases from 90.9% in

the training data to 66.7% in the test data. This is a much larger drop in precision than in the logistic regression model, although its test precision is still higher than the last model. Recall decreases from 29.4% in the training set to a dismal 15.4% in the test set, while the F1 score drops from 44.4% to 25%. Most of these results are significantly worse than those of the previous model.

Next, I will look at the areas under the PR and ROC curves for the Naïve Bayes model. The area under the PR curve is 0.555, which is clearly lower than that of the logistic regression model (0.600). Likewise, the area under the ROC curve is 0.557, which is again weaker than the regression model's area (0.632). Overall, I found that the Naïve Bayes model is weaker than the logistic regression model for almost every important measure. However, these results may once again be due to the data that was included in the training and test sets. The small size of the dataset and the skewness of the target variable can cause the data within each partition to be drastically different from one another. This means that the algorithm may have different results if the random seed was changed. This can likely be solved by using a larger, more balanced dataset.



```
# This table shows:
# 1. The number of low birth weight infants predicted as having low birth weight
# 2. The number of low birth weight infants predicted as not having low birth weight
# 3. The number of regular birth weight infants predicted as having low birth weight
# 4. The number of regular birth weight infants predicted as not having low birth weight

prediction.crosstab('label', 'prediction').show()
```

```
+----------------+---+---+
|label_prediction|0.0|1.0|
+----------------+---+---+
|             1.0| 12| 14|
|             0.0| 44|  6|
+----------------+---+---+
```

Figure 6. Confusion Matrix for Low Birth Weight Decision Tree on Test Data.

Lastly, I will experiment with using a decision tree to classify low birth weight cases. This model requires loading several packages such as "DecisionTreeClassifier," "StringIndexer," and "vectorIndexer" ("Classification and regression," n.d.). To build this model, I followed many of

the same steps that I used for the other two models. The main difference is that I created a decision tree classifier and used it as an input for the pipeline. By looking at the confusion matrix for the test set, we see that the model recognized 44 true negatives and 14 true positives (see Figure 6). This gives it a test accuracy of 76.3%, which is by far the highest of the three models. There are also 6 false positives and 12 false negatives—the fewest of any model used in this analysis. Although the model's overall accuracy could still use improvement, it appears to be more effective at classifying low birth rate cases when compared to the other models.

From here, I will examine this model's performance in both the training and test sets. According to the HTML file for Part 2, the decision tree's training accuracy is 79.8% and its test accuracy is 76.3%. Based on these figures, the decision tree appears to be the only model that does not experience overfitting on this dataset. Its precision decreases from 76.2% in the training set to 70% in the test set, which is a smaller drop-off when compared to the other two models. Its test precision is also higher than either of the previous models. Furthermore, the recall actually increases from 47.1% in the training data to 53.8% in the test data—giving it the highest test recall of any model. Likewise, the F1 score—which depends on both precision and recall—increases from 58.2% in the training set to 60.9% in the test set. Of the three models used in this analysis, the decision tree classifier has by far the highest F1 score.

Finally, I will examine the decision tree's area under the PR and ROC curves. The area under the PR curve is 0.698, which is the highest of the three models. The area under the ROC curve is 0.709, which is higher than that of the previous two models. Still, these measures are far from the maximum area of 1.0, which means that the current decision tree is far from optimal. Altogether, the decision tree is the most effective of the three algorithms for classifying low birth weight cases. Its test results are stronger than those of the other two models, and it is the only

model that did not suffer from overfitting. This model's accuracy could still use significant improvement, and its performance was likely hurt by the size of the dataset and the skewness of the target variable. Once again, the model's accuracy would likely be improved by using a larger dataset with a balanced target. In addition, I may recommend using variations of decision trees, such as bagging, boosting, or random forests. These algorithms will function more effectively when used on a much larger dataset. Furthermore, although this model appears to be the most accurate, this might be due to the random seed used for the data partition. Since the dataset is very small, it is likely that a different partition would involve very different records in the training and test sets—which may impact the performance of this algorithm.

Part 3: Research Questions

What are the key differences between HDFS and Object Storage, including the network latency?

Hadoop Distributed File System (HDFS) and object storage are intended for two distinct functions when handling big data. HDFS is mainly used for data analysis operations, such as performing calculations and manipulating files. It is designed to handle the high bandwidth patterns that are required for operating on big data (Woodie, 2015). This feature helps HDFS to effectively perform data science tasks on very large datasets. Object storage, on the other hand, is not designed for handling big data analytics tasks. Manipulating files and directories is a key component of data science, but object storage does not provide support for directories (Woodie, 2015). Instead, it is primarily used for storing large volumes of data—especially unstructured data such as images and videos. It uses erasure coding, which is an algorithm that spreads the data across multiple disks, and it builds spare drives to handle possible system failures—using the erasure coding algorithm to rebuild the data files (Woodie, 2015).

Overall, object storage is much less costly for long-term data storage when compared to HDFS. This is because HDFS stores three copies of each dataset to prevent the loss of data in the case of a system failure. For large datasets, this can use up a lot of unnecessary storage space—especially since most of this data may not even be used (Richardson, 2017). Another problem with using HDFS for file storage is that there is a greater chance of losing some of the data when storing petabytes of records (Woodie, 2015). The data is not being constantly restored—which means that if one of the three dataset copies loses some of its data, then that copy will not rebuild

the lost data and the user will most likely be unaware that the data was even lost (Woodie, 2017). Another issue with using HDFS for long-term storage relates to the master node, which stores all metadata for a single cluster. According to Richardson (2017), if the master node fails in HDFS, then that entire cluster and all of its metadata will no longer be accessible. But in object storage, the metadata can be moved anywhere in the system and does not have to depend on the master node (Richardson, 2017). Furthermore, the author claims that if a master node fails in object storage, then one of the slave nodes can easily replace it.

There is one more reason why HDFS is not intended for long-term file storage. In HDFS, computing power and storage capacity are run on the same node—meaning that you cannot perform either of these tasks independently (Richardson, 2017). Expanding the amount of storage space will use up a lot of unnecessary computing power, while increasing the computing power will likewise require additional storage space. But in object storage, you can add additional computing power or storage capacity onto separate nodes (Richardson, 2017). This is useful since it prevents wasting extra computing power or storage space when they are not needed.

Altogether, HDFS and object storage are optimized for performing specific roles when handling big data. HDFS is intended for performing analysis on the data, while object storage is designed for the long-term storage of massive amounts of data. With regards to network latency, object storage uses fast networks for moving data to the system (Woodie, 2017). This makes it very effective for transferring and storing large volumes of data. However, analyzing big data generally requires moving the computations to the data instead of moving the data to the system. Hadoop is optimized for transferring calculations to the data source, which makes HDFS much faster than object storage when performing data science tasks. Therefore, choosing which of these technologies to use will depend on whether your main goal is to store the data or analyze it.

We may use Python, Scala, and Java programming languages for Spark. Discuss the pros and cons of each language. The answer should address the level of effort to write the code, code size, and performance.

Scala, Python, and Java are programming languages that are supported by Apache Spark. These three languages each have their own strengths and weaknesses which need to be considered before choosing which language to use. With regards to difficulty and effort required to write the code, Scala is by far the hardest to learn. This is because it has a more complex structure than the other two languages, and it includes more advanced features such as tuples and macros that provide improved performance but are harder to understand (Hicks, n.d.).

Python is the easiest of these languages to learn. According to Radcliffe (2016), this is because it is a dynamically-typed programming language—which means that every variable name in the code is bound to an object but not to that object's type. The author states that this lack of type information makes the code cleaner and simpler to write, and therefore the language is easier for newcomers to learn. However, the consequence of not including the type information is that it becomes more difficult to analyze the code and understand what is going on at any point in the program (Radcliffe, 2016). If we do not know the type of a given variable, it will be hard to determine how certain functions perform using that variable.

Java, on the other hand, is a statically-typed language—meaning that every name is bound to a type rather than only to an object (Radcliffe, 2016). According to the author, this feature makes it easier to follow and analyze the code because it allows us to identify the type associated with each variable. However, this also makes Java code more difficult to write because users need to be able to manage variable types throughout the entire code (Radcliffe, 2016). But while Java requires more effort to write when compared to Python, it is still easier to use than Scala.

Interestingly, Scala is also a statically-typed programming language (Hicks, n.d.), which makes it easier to understand analytically but more difficult to write. According to Hicks (n.d.), one of the challenges of dynamically-typed languages is that it is harder to test for bugs or robustness without running the code across many different scenarios. Therefore, while static languages such as Java and Scala are more difficult to learn, they may be faster and less costly to implement.

When looking at code size, Scala programs tend to be much shorter than Java programs. Although Scala has a more complex structure, these features allow it to be much more concise since a single line of Scala code can contain the same amount of information as several lines of Java code (Hicks, n.d.). Scala was designed to overcome many of the limitations in languages such as Java, which results in the code being more complex but much less tedious (Hicks, n.d.). Likewise, Python code tends to be more concise than Java code. Since Python is a dynamically-typed language, it does not provide object type information throughout the code. This helps to reduce the size of the code when compared to programs written in Java. Both Python and Scala are relatively concise, and they can achieve similar levels of conciseness when using certain coding techniques ("6 points to compare," 2016).

Lastly, I will evaluate the performance capabilities of each programming language. Both Scala and Java are written to run in the Java Virtual Machine, or JVM (Hicks, n.d.). Code compiled in the JVM will run faster than Python code, which means that Scala and Java can provide faster execution performance when compared to Python ("6 points to compare," 2016). However, one other area to consider is the available data science libraries for each language. Both Python and Java provide many useful libraries, but there are few Java packages that are as powerful as Python's scikit-learn library ("6 points to compare," 2016). Scikit-learn is an open-source machine-learning library that provides support for data science algorithms such as classification,

regression, and clustering ("Scikit-learn," n.d.). One interesting point is that Java is primarily supported by a sole corporate sponsor, while Python receives more widespread community support (Radcliffe, 2016). This might suggest that Python's open-source libraries receive more frequent and consistent support when compared to Java's libraries.

Scala does not nearly have the same number of libraries as Python, mainly because it has not yet reached the same level of maturity or widespread use ("6 points to compare," 2016). According to this source, Scala only provides relatively basic packages for machine learning—but it is also able to access Java libraries as well as MLlib while using Spark. However, MLlib is not as easy to use as Python's libraries, and it has not yet received the same level of community support or development ("6 points to compare," 2016). Furthermore, MLlib can be used over Spark regardless of which programming language is used. Altogether, Python currently provides the most powerful data science libraries—even though it is the slowest of the three languages.

Overall, each of these languages has its own strengths and weaknesses. With regards to difficulty, Scala is the hardest to write while Python is the easiest. When looking at code size, Java programs tend to be the longest, while Scala and Python scripts can be similarly condensed. Performance can be evaluated using different measures, such as processing speed and machine learning capabilities. Python runs slower than the other two languages, but at the same time it provides some of the most well-developed data science libraries (such as scikit-learn). All three languages serve a purpose in the data science community. Python is easy to learn and can provide the desired results quickly. Java is useful for developers who want to analyze the code and gain a deeper understanding of how the program works. Finally, Scala is best for those who want more advanced functionality in their code while also providing a more streamlined coding experience.

References

6 points to compare Python and Scala for Data Science using Apache Spark. (2016, January 28). Retrieved April 8, 2018, from https://datasciencevademecum.wordpress.com/2016/01/28/6-points-to-compare-python-and-scala-for-data-science-using-apache-spark/

Brownlee, J. (2014, March 21). Classification Accuracy is Not Enough: More Performance Measures You Can Use. Retrieved April 1, 2018, from https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

Classification and regression. (n.d.). Retrieved April 3, 2018, from https://spark.apache.org/docs/2.1.0/ml-classification-regression.html#decision-tree-classifier

Dawson, R. J. (1995). The "Unusual Episode" Data Revisited. Retrieved April 1, 2018, from https://ww2.amstat.org/publications/jse/v3n3/datasets.dawson.html

Differences between Receiver Operating Characteristic AUC (ROC AUC) and Precision Recall AUC (PR AUC). (2014, April 2). Retrieved April 1, 2018, from http://www.chioka.in/differences-between-roc-auc-and-pr-auc/

Hicks, M. (n.d.). Scala vs. Java: Why Should I Learn Scala? Retrieved April 8, 2018, from https://www.toptal.com/scala/why-should-i-learn-scala

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression* (3rd ed.). New York, NY: Wiley.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning: with applications in R. Retrieved June 14, 2017.

Koehrsen, W. (2018, March 03). Beyond Accuracy: Precision and Recall. Retrieved April 1, 2018, from https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

Low Birthweight. (2014, August 24). Retrieved April 2, 2018, from http://www.chop.edu/conditions-diseases/low-birthweight

Radcliffe, T. (2016, January 26). Python vs. Java: Duck Typing, Parsing on Whitespace and Other Cool Differences. Retrieved April 8, 2018, from https://www.activestate.com/blog/2016/01/python-vs-java-duck-typing-parsing-whitespace-and-other-cool-differences

Richardson, M. A. (2017). 3 reasons to replace HDFS with Object Storage and 1 reason not to. Retrieved April 6, 2018, from https://it.toolbox.com/blogs/maryannrichardson/3-reasons-to-replace-hdfs-with-object-storage-and-1-reason-not-to-103117

Scikit-learn. (n.d.). Retrieved April 8, 2018, from http://scikit-learn.org/stable/

Tape, T. G. (n.d.). The Area Under an ROC Curve. Retrieved April 1, 2018, from http://gim.unmc.edu/dxtests/roc3.htm

Woodie, A. (2015, June 23). Data Lake Showdown: Object Store or HDFS? Retrieved April 6, 2018, from https://www.datanami.com/2015/06/23/data-lake-showdown-object-store-or-hdfs/