**Exercice 4 - Build a Logistic Regression model**

You've already built a Decision Tree model using the flights data. Now you're going to create a Logistic Regression model on the same data.

The objective is to predict whether a flight is likely to be delayed by at least 15 minutes (label 1) or not (label 0).

Although you have a variety of predictors at your disposal, you'll only use the mon, depart and duration columns for the moment. These are numerical features which can immediately be used for a Logistic Regression model. You'll need to do a little more work before you can include categorical features. Stay tuned!

The data have been split into training and testing sets and are available as flights\_train and flights\_test.

**Instructions**

**100 XP**

* Import the class for creating a Logistic Regression classifier.
* Create a classifier object and train it on the training data.
* Make predictions for the testing data and create a confusion matrix.

# Import the logistic regression class

from pyspark.ml.\_\_\_\_ import \_\_\_\_

# Create a classifier object and train on training data

logistic = \_\_\_\_().\_\_\_\_(\_\_\_\_)

# Create predictions for the testing data and show confusion matrix

prediction = \_\_\_\_.\_\_\_\_(\_\_\_\_)

prediction.groupBy(\_\_\_\_, \_\_\_\_).\_\_\_\_().show()

###### Hint

* Import the LogisticRegression class from pyspark.ml.classification.
* Create an instance of LogisticRegression. Use the fit() method on flights\_train.
* Use the transform() method on flights\_test to generate predictions. Then, group the predictions by label and prediction before calling the count() method.

**Evaluate the Logistic Regression model**

Accuracy is generally not a very reliable metric because it can be biased by the most common target class.

There are two other useful metrics:

* *precision* and
* *recall*.

Check the slides for this lesson to get the relevant expressions.

Precision is the proportion of positive predictions which are correct. For all flights which are predicted to be delayed, what proportion is actually delayed?

Recall is the proportion of positives outcomes which are correctly predicted. For all delayed flights, what proportion is correctly predicted by the model?

The precision and recall are generally formulated in terms of the positive target class. But it's also possible to calculate *weighted* versions of these metrics which look at both target classes.

The components of the confusion matrix are available as TN, TP, FN and FP, as well as the object prediction.

**Instructions**

**100 XP**

* Find the precision and recall.
* Create a multi-class evaluator and evaluate weighted precision.
* Create a binary evaluator and evaluate AUC using the "areaUnderROC" metric.

from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClassificationEvaluator

# Calculate precision and recall

precision = \_\_\_\_

recall = \_\_\_\_

print('precision = {:.2f}\nrecall    = {:.2f}'.format(precision, recall))

# Find weighted precision

multi\_evaluator = \_\_\_\_

weighted\_precision = multi\_evaluator.\_\_\_\_(prediction, {multi\_evaluator.metricName: "\_\_\_\_"})

# Find AUC

binary\_evaluator = \_\_\_\_

auc = binary\_evaluator.\_\_\_\_(\_\_\_\_, {\_\_\_\_})

###### Hint

* Precision is TP / (TP + FP) and recall is TP / (TP + FN).
* Create an instance of MulticlassClassificationEvaluator. Call the evaluate() method. The metric name is "weightedPrecision".
* Create an instance of BinaryClassificationEvaluator. Call the evaluate() method. The metric name is "areaUnderROC". Use the metricName attribute on the evaluator to choose the right metric. Make sure to use the right evaluator!

**Punctuation, numbers and tokens**

At the end of the previous chapter you loaded a dataset of SMS messages which had been labeled as either "spam" (label 1) or "ham" (label 0). You're now going to use those data to build a classifier model.

But first you'll need to prepare the SMS messages as follows:

* remove punctuation and numbers
* tokenize (split into individual words)
* remove stop words
* apply the hashing trick
* convert to TF-IDF representation.

In this exercise you'll remove punctuation and numbers, then tokenize the messages.

The SMS data are available as sms.

**Instructions**

**100 XP**

* Import the function to replace regular expressions and the feature to tokenize.
* Replace all punctuation characters from the text column with a space. Do the same for all numbers in the text column.
* Split the text column into tokens. Name the output column words.

# Import the necessary functions

from pyspark.sql.functions import \_\_\_\_

from pyspark.ml.feature import \_\_\_\_

# Remove punctuation (REGEX provided) and numbers

wrangled = sms.withColumn('text', \_\_\_\_(sms.text, '[\_():;,.!?\\-]', \_\_\_\_))

wrangled = wrangled.withColumn(\_\_\_\_, \_\_\_\_(\_\_\_\_, \_\_\_\_, \_\_\_\_))

# Merge multiple spaces

wrangled = wrangled.withColumn('text', regexp\_replace(wrangled.text, ' +', ' '))

# Split the text into words

wrangled = \_\_\_\_(inputCol='text', outputCol=\_\_\_\_).\_\_\_\_(wrangled)

wrangled.show(4, truncate=False)

###### Hint

* Import regexp\_replace from pyspark.sql.functions and Tokenizer from pyspark.ml.feature.
* Within regexp\_replace() refer to the text column using sms.text and wrangled.text. The regular expression for numbers is '[0-9]' or '\d'..
* Use the Tokenizer class and call the transform() method.

**Stop words and hashing**

The next steps will be to remove stop words and then apply the hashing trick, converting the results into a TF-IDF.

A quick reminder about these concepts:

* The hashing trick provides a fast and space-efficient way to map a very large (possibly infinite) set of items (in this case, all words contained in the SMS messages) onto a smaller, finite number of values.
* The TF-IDF matrix reflects how important a word is to each document. It takes into account both the frequency of the word within each document but also the frequency of the word across all of the documents in the collection.

The tokenized SMS data are stored in sms in a column named words. You've cleaned up the handling of spaces in the data so that the tokenized text is neater.

**Instructions**

**100 XP**

* Import the StopWordsRemover, HashingTF and IDF classes.
* Create a StopWordsRemover object (input column words, output column terms). Apply to sms.
* Create a HashingTF object (input results from previous step, output column hash). Apply to wrangled.
* Create an IDF object (input results from previous step, output column features). Apply to wrangled.

from pyspark.ml.\_\_\_\_ import \_\_\_\_, \_\_\_\_, \_\_\_\_

# Remove stop words.

wrangled = \_\_\_\_(inputCol=\_\_\_\_, outputCol=\_\_\_\_)\

      .\_\_\_\_(sms)

# Apply the hashing trick

wrangled = \_\_\_\_(\_\_\_\_, \_\_\_\_, numFeatures=1024)\

      .\_\_\_\_(wrangled)

# Convert hashed symbols to TF-IDF

tf\_idf = \_\_\_\_(\_\_\_\_, \_\_\_\_)\

      .\_\_\_\_(wrangled).\_\_\_\_(wrangled)

tf\_idf.select('terms', 'features').show(4, truncate=False)

###### Hint

* These classes are found in pyspark.ml.feature.
* Call the transform() method to create a StopWordsRemover object.
* Call the transform() method to create a HashingTF object. The output column from StopWordsRemover becomes the input column for HashingTF.
* Call the fit() and transform() methods to create an IDF object. The output column from HashingTF becomes the input column for IDF.

**Training a spam classifier**

The SMS data have now been prepared for building a classifier. Specifically, this is what you have done:

* removed numbers and punctuation
* split the messages into words (or "tokens")
* removed stop words
* applied the hashing trick and
* converted to a TF-IDF representation.

Next you'll need to split the TF-IDF data into training and testing sets. Then you'll use the training data to fit a Logistic Regression model and finally evaluate the performance of that model on the testing data.

The data are stored in sms and LogisticRegression has been imported for you.

**Instructions**

**100 XP**

* Split the data into training and testing sets in a 4:1 ratio. Set the random number seed to 13 to ensure repeatability.
* Create a LogisticRegression object and fit it to the training data.
* Generate predictions on the testing data.
* Use the predictions to form a confusion matrix.

# Split the data into training and testing sets

sms\_train, sms\_test = sms.\_\_\_\_(\_\_\_\_, \_\_\_\_)

# Fit a Logistic Regression model to the training data

logistic = \_\_\_\_(regParam=0.2).\_\_\_\_(\_\_\_\_)

# Make predictions on the testing data

prediction = logistic.\_\_\_\_(\_\_\_\_)

# Create a confusion matrix, comparing predictions to known labels

prediction.groupBy(\_\_\_\_, \_\_\_\_).\_\_\_\_().\_\_\_\_()

###### Hint

* Use the randomSplit() method. The weights should be [0.8, 0.2]. Specify the seed argument.
* The LogisticRegression() class is defined in pyspark.ml.classification. Call the fit() method.
* Call the transform() method.
* Group the data by label and prediction then count() and show() the results.