16.6.5

Run the Model

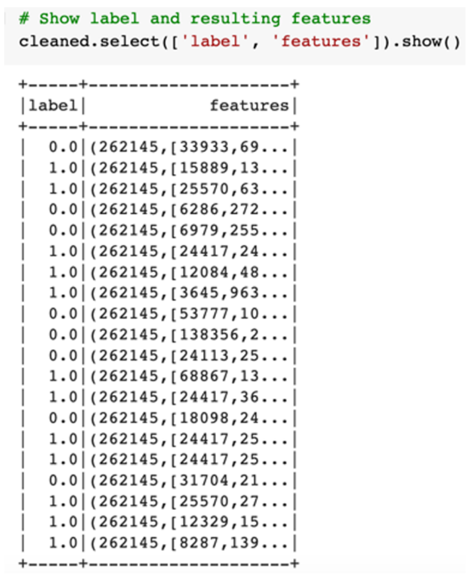
You know that ML is needed to run NLP. However, ML is its own complex topic to master. As you and Jennifer are on a deadline, you decide to put your curiosity on hold for now and work with some basics to get your NLP to work. You plan to revisit ML once this project is done.

After our pipeline has been set up, we'll fit the outcome with our original DataFrame and transform it.

Type and run the following code:

```
# Fit and transform the pipeline
cleaner = data_prep_pipeline.fit(data_df)
cleaned = cleaner.transform(data_df)
```

As you can see in the following image, our labels and features that we created early on in the process are numerical representations of positive and negative reviews. The features will be used in our model and predict whether a given review will be positive or negative. These features are the result of all the work we have been doing with the pipeline.



```
# Show label and resulting features
cleaned.select(['label', 'features']).show()
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(262145,[33933,69...|
|  1.0|(262145,[15889,13...|
|  1.0|(262145,[25570,63...|
|  0.0|(262145,[6286,272...|
|  0.0|(262145,[6979,255...|
|  1.0|(262145,[24417,24...|
|  1.0|(262145,[12084,48...|
|  1.0|(262145,[3645,963...|
|  0.0|(262145,[53777,10...|
|  0.0|(262145,[138356,2...|
|  0.0|(262145,[24113,25...|
|  1.0|(262145,[68867,13...|
|  1.0|(262145,[24417,36...|
|  0.0|(262145,[18098,24...|
|  1.0|(262145,[24417,25...|
|  1.0|(262145,[24417,25...|
|  0.0|(262145,[31704,21...|
|  1.0|(262145,[25570,27...|
|  1.0|(262145,[12329,15...|
|  1.0|(262145,[8287,139...|
+-----+--------------------+
```

IMPORTANT

You may not have seen the terms "fit," "predict," and "transform" before in this context. These terms will make more sense when we learn about machine learning:

Now let's run our ML model on the data. One of the basics of ML is that data gets broken into training data and testing data. **Training data** is the data that will be passed to our NLP model that will train our model to predict results. The **testing data** is used to test our predictions. We can do this with the `randomSplit` method, which takes in a list of the percent of data we want split into each group. Standard conventions use 70% with training and 30% with testing.

To split the data into a training set and a testing set, run the following code:

```
# Break data down into a training set and a testing set
training, testing = cleaned.randomSplit([0.7, 0.3], 21)
```

The array supplied to `randomSplit` is the percentage of the data that will be broken into training and testing respectively. So 70% to training and 30% to testing. The second number supplied is called a seed. The seed number here, 21, is arbitrary. But as long as the same seed is used, the result will be the same each time. Using a seed number ensures reproducible results.

The ML model we'll use is Naive Bayes, which we'll import and then fit the model using the training dataset. **Naive Bayes** is a group of classifier algorithms based on Bayes' theorem. Bayes theorem provides a way to determine the probability of an event based on new conditions or information that might be related to the event.

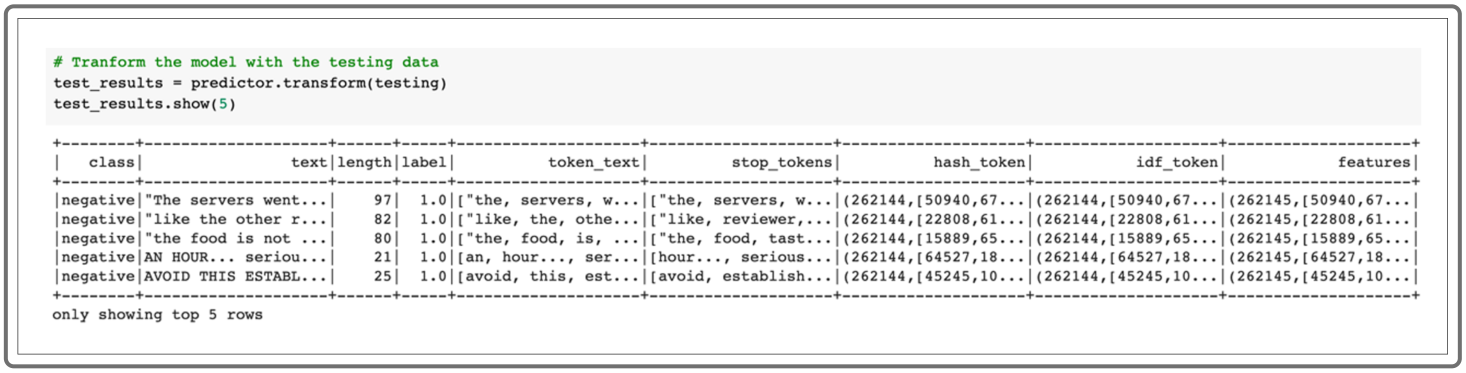
NOTE

> For a deeper dive into Naive Bayes, refer to this article on **[Naive Bayes Classifiers (https://www.geeksforgeeks.org/naive-bayes-classifiers/)](https://www.geeksforgeeks.org/naive-bayes-classifiers/)** and the **[Naive Bayes Wikipedia page (https://en.wikipedia.org/wiki/Naive_Bayes_classifier)](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)**.

Run the following code:

```
from pyspark.ml.classification import NaiveBayes
# Create a Naive Bayes model and fit training data
nb = NaiveBayes()
predictor = nb.fit(training)
```

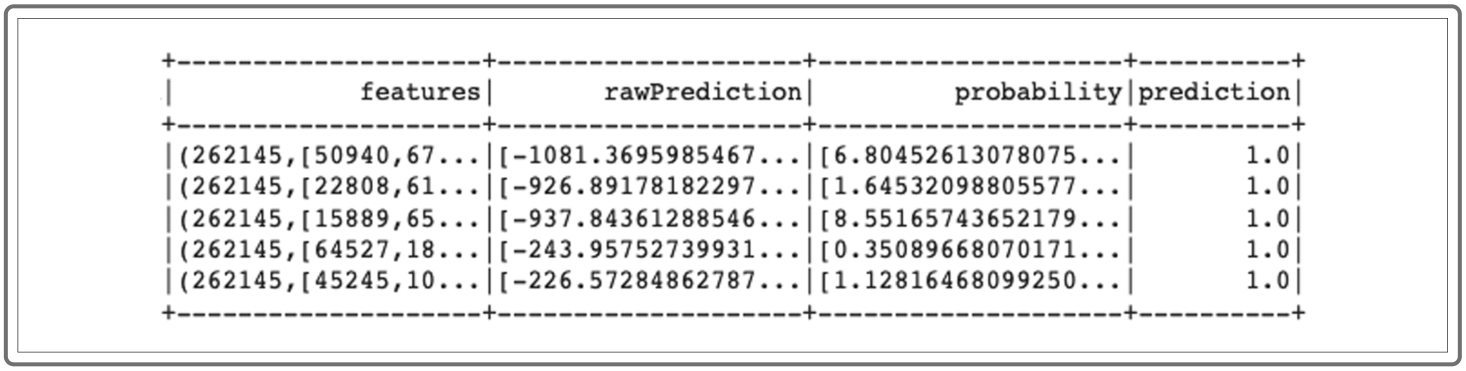
Once the model has been trained, we'll transform the model with our testing data. Run the following code, and then look at the results:



```
# Tranform the model with the testing data
test_results = predictor.transform(testing)
test_results.show(5)
```

```
+--------+--------------------+------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+
|   class|                text|length|label|          token_text|         stop_tokens|          hash_token|           idf_token|            features|
+--------+--------------------+------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+
|negative|"The servers went...|    97|  1.0|["the, servers, w...|["the, servers, w...|(262144,[50940,67...|(262144,[50940,67...|(262145,[50940,67...|
|negative|"like the other r...|    82|  1.0|["like, the, othe...|["like, reviewer,...|(262144,[22808,61...|(262144,[22808,61...|(262145,[22808,61...|
|negative|"the food is not ...|    80|  1.0|["the, food, is, ...|["the, food, tast...|(262144,[15889,65...|(262144,[15889,65...|(262145,[15889,65...|
|negative|AN HOUR... seriou...|    21|  1.0|[an, hour..., ser...|[hour..., serious...|(262144,[64527,18...|(262144,[64527,18...|(262145,[64527,18...|
|negative|AVOID THIS ESTABL...|    25|  1.0|[avoid, this, est...|[avoid, establish...|(262144,[45245,10...|(262144,[45245,10...|(262145,[45245,10...|
+--------+--------------------+------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

At first glance, it may seem like the model isn't showing us anything useful. From your output, scroll all the way to the right to view the prediction column:



```
+--------------------+--------------------+--------------------+----------+
|            features|       rawPrediction|         probability|prediction|
+--------------------+--------------------+--------------------+----------+
|(262145,[50940,67...|[-1081.3695985467...|[6.80452613078075...|       1.0|
|(262145,[22808,61...|[-926.89178182297...|[1.64532098805577...|       1.0|
|(262145,[15889,65...|[-937.84361288546...|[8.55165743652179...|       1.0|
|(262145,[64527,18...|[-243.95752739931...|[0.35089668070171...|       1.0|
|(262145,[45245,10...|[-226.57284862787...|[1.12816468099250...|       1.0|
+--------------------+--------------------+--------------------+----------+
```

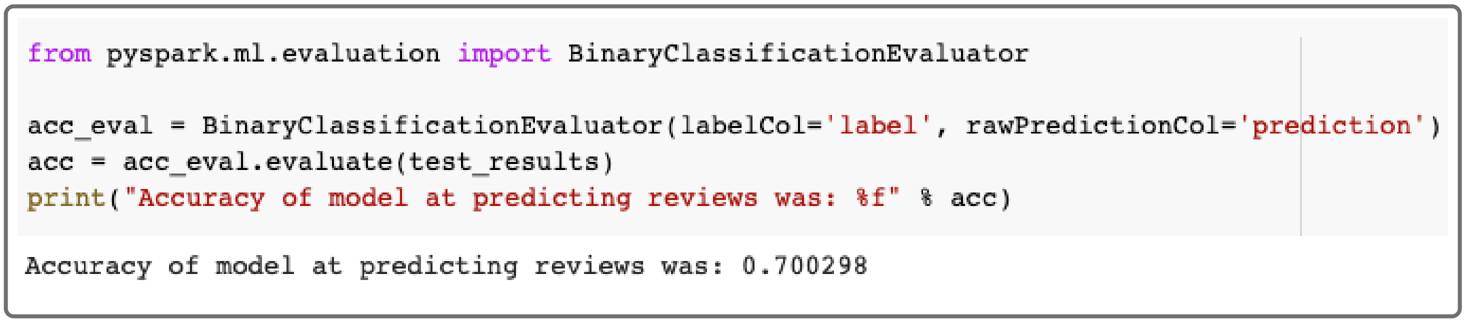
This prediction column will indicate with a 1.0 if the model thinks this review is negative and 0.0 if it thinks it's positive. Future data sets can now be run with this model and determine whether a review was positive or negative without having already supplied in the data.

How useful is this model? Should we just blindly trust that it will be right every time? There is one last step in the process to answer these questions.

It's often not enough to simply train and use a machine learning model for predictions without knowing how well the model performs at its prediction task. The last step is to import the `BinaryClassificationEvaluator`, which will display how accurate our model is in determining if a review with be positive or negative based solely on the text within a review.

The `BinaryClassificationEvaluator` uses two arguments, `labelCol` and `rawPredictionCol`. The `labelCol` takes the labels which were the result of using `StringIndexer` to convert our positive and negative strings to integers. The `rawPredictionCol` takes in numerical predictions from the output of running the Naive Bayes model.

The performance of a model can be measured based on the difference between its predicted values and actual values. This is what the `BinaryClassificationEvaluator` does. We will dive more into accuracy, precision, and sensitivity when we get to Machine Learning. Run the following code:



```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

acc_eval = BinaryClassificationEvaluator(labelCol='label', rawPredictionCol='prediction')
acc = acc_eval.evaluate(test_results)
print("Accuracy of model at predicting reviews was: %f" % acc)
```

```
Accuracy of model at predicting reviews was: 0.700298
```

The accuracy of the model isn't perfect, but it's not too low either: 0.700298. Machine learning isn't a guarantee, and tweaking our models as well as the data used is part of the process. One of the ways to do this is to add more data; when you keep adding data, eventually you grow from your local storage to something much larger—thus leading to big data!

Retake