16.6.4

Pipeline Setup to Run the Model

Great, you have all the steps of the pipeline down! Now it's time to put it all together and create a pipeline that will link everything and allow for the process to be reproduced.

Set Up the Pipeline

A pipeline enables us to store all of the functions we have created in different stages and run only once. Each stage that is passed in won't run until the previous stage has been completed, which means the output of one stage is passed on to the next one.

We'll put the pipeline to use by employing a sample Yelp dataset. This will allow you to practice with a smaller set of reviews using similar data that you'll be working on for your client.

Open up a new notebook and run the usual setup.

```
import os
# Find the latest version of spark 3.0 from http://www.apache.org/dist/spark/ and enter as the spark version
# For example:
# spark_version = 'spark-3.0.3'
spark_version = 'spark-3.<enter version>'
os.environ['SPARK_VERSION']=spark_version

# Install Spark and Java
!apt-get update
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q http://www.apache.org/dist/spark/$SPARK_VERSION/$SPARK_VERSION-bin-hadoop2.7.tgz
!tar xf $SPARK_VERSION-bin-hadoop2.7.tgz
!pip install -q findspark

# Set Environment Variables
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
```

```
os.environ["SPARK_HOME"] = f"/content/{spark_version}-bin-hadoop2.7"

# Start a SparkSession
import findspark
findspark.init()
```

Then start a new Spark session.

```
# Start Spark session
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Yelp_NLP").getOrCreate()
```

Then load in the following dataset into a DataFrame by entering the following code.

```
# Read in data from S3 Buckets
from pyspark import SparkFiles
url ="https://s3.amazonaws.com/dataviz-curriculum/day_2/yelp_reviews.csv"
spark.sparkContext.addFile(url)
df = spark.read.csv(SparkFiles.get("yelp_reviews.csv"), sep=",", header=True)

# Show DataFrame
df.show()
```

Type the code to import all the functions that will be used in our NLP process, or pipeline, as follows:

```
# Import functions
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF, StringIndexer
```

Next, create a new column that uses the `length` function to create a `future feature` with the length of each row. This is similar to the `tokenizer` phase when we created our own `udf` to do the same thing. A `udf` could still be used here, but PySpark makes it easier by supplying a ready-to-use function.

Type and run the following code.

```
from pyspark.sql.functions import length
# Create a length column to be used as a future feature
data_df = df.withColumn('length', length(df['text']))
data_df.show()
```

Now we'll create all the transformations to be applied in our pipeline. Note that the `StringIndexer` encodes a string column to a column of table indexes. Here we are working with positive and negative game reviews, which will be converted to 0 and 1. This will form our labels, which we'll delve into in the ML unit. The label is what we're trying to predict: will the review's given text let us know if it was positive or negative?

Also note that we don't need to run all of these completely as we did before. By creating all the functions now, we can then use them all in the pipeline later.

Type and run the following code.

```
# Create all the features to the data set
pos_neg_to_num = StringIndexer(inputCol='class',outputCol='label')
tokenizer = Tokenizer(inputCol="text", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
hashingTF = HashingTF(inputCol="stop_tokens", outputCol='hash_token')
idf = IDF(inputCol='hash_token', outputCol='idf_token')
```

We'll create a feature vector containing the output from the IDFModel (the last stage in the pipeline) and the length. This will combine all the raw features to train the ML model that we'll be using. Enter the code for this as follows:

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vector
# Create feature vectors
clean_up = VectorAssembler(inputCols=['idf_token', 'length'], outputCol='features')
```

Now it's time to create our pipeline, the easiest step. We'll import the pipeline from `pyspark.ml`, and then store a list of the stages created earlier. It's important to list the stages in the order they need to be executed. As we mentioned before, the output from one stage will then be passed off to another stage.

Run the following code:

```
# Create and run a data processing Pipeline
from pyspark.ml import Pipeline
data_prep_pipeline = Pipeline(stages=[pos_neg_to_num, tokenizer, stopremove, hashingTF, idf, clean_up])
```

Nice work! All the stages of the pipeline have been set up. You may have noticed that each individual step didn't need to be set up, which is the perk of setting up the pipeline! Now your data is ready to be run through the pipeline. Then we can run it through a machine learning model.