PySpark ML and XGBoost full integration tested on the Kaggle Titanic dataset

Bogdan Cojocar Follow
Jul 8, 2018 · 4 min read ★

In this tutorial we will discuss about integrating PySpark and XGBoost using a standard machine learing pipeline.

We will use data from the Titanic: Machine learning from disaster one of the many Kaggle competitions.

the many Kaggle competitions.

Before getting started please know that you should be familiar with Apache Spark and Xgboost and Python.

The code used in this tutorial is available in a Jupyther notebook on github.

Step 1: Download or build the XGBoost jars

The python code will need two scala jars dependencies in order to work. You can download them directly from maven:

- xgboost4j
- xgboost4j-spark

If you wish to build them yourself you can find out how to do it from one of my previous tutorials.

Step 2: Download the XGBoost python wrapper

You can download the PySpark XGBoost code from here. This is the interface between the part that we will write and the XGBoost scala implementation. We will see how to integrate it in the code later in the tutorial.

Step 3: Start a new Jupyter notebook

We will start a new notebook in order to be able to write our code:

```
jupyter notebook
```

Step 4: Add the custom XGBoost jars to the Spark app

Before starting Spark we need to add the jars we previously downloaded. We can do this using the `--jars` flag:

```
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars xgboost4j-spark-
0.72.jar,xgboost4j-0.72.jar pyspark-shell'
```

Step 5: Integrate PySpark into the Jupyther notebook

Easiest way to make PySpark available is using the `findspark` package:

```
import findspark
findspark.init()
```

Step 6: Start the spark session

We are now ready to start the spark session. We are creating a spark app that will run locally and will use as many threads as there are cores using `local[*]` :

```
spark = SparkSession\
        .builder\
        .appName("PySpark XGBOOST Titanic")\
        .master("local[*]")\
        .getOrCreate()
```

Step 7: Add the PySpark XGBoost wrapper code

As we have now the spark session, we can add the wrapper code we previously dowloaded:

```
spark.sparkContext.addPyFile("YOUR_PATH/sparkxgb.zip")
```

Step 8: Defining a schema

Next we define a schema of the data we read from the csv. This is usually a better practice than letting spark to infer the schema because it consumes less resources and we have total control over the fields.

```
schema = StructType(
  [StructField("PassengerId", DoubleType()),
    StructField("Survival", DoubleType()),
    StructField("Pclass", DoubleType()),
    StructField("Name", StringType()),
    StructField("Sex", StringType()),
    StructField("Age", DoubleType()),
    StructField("SibSp", DoubleType()),
    StructField("Parch", DoubleType()),
    StructField("Ticket", StringType()),
    StructField("Fare", DoubleType()),
    StructField("Cabin", StringType()),
    StructField("Embarked", StringType())
  ])
```

Step 9: Read the csv data into a dataframe

We read the csv into a `DataFrame`, making sure we mention we have a header and we also replace `null` values with 0:

```
df_raw = spark\
  .read\
  .option("header", "true")\
  .schema(schema)\
  .csv("YOUR_PATH/train.csv")

df = df_raw.na.fill(0)
```

Step 10: Convert the nominal values to numeric

Before walking through the code on this step let's go briefly through some Spark ML concepts. They introduce the concept of ML pipelines, which is a set of high level APIs build on top of the `DataFrames` which make it easier to combine multiple algorithms into a single process. The main elements of a pipeline are the `Transformer` and the `Estimator`. The first can represent an algorithm that can transform a `DataFrame` into another `DataFrame`, and the latter is an algorithm that can fit on a `DataFrame` to produce a `Transformer` .

In order to convert the nominal values into numeric ones we need to define a `Transformer` for each column:

```
sexIndexer = StringIndexer()\
  .setInputCol("Sex")\
  .setOutputCol("SexIndex")\
  .setHandleInvalid("keep")

cabinIndexer = StringIndexer()\
  .setInputCol("Cabin")\
  .setOutputCol("CabinIndex")\
  .setHandleInvalid("keep")

embarkedIndexer = StringIndexer()\
  .setInputCol("Embarked")\
  .setOutputCol("EmbarkedIndex")\
  .setHandleInvalid("keep")
```

We are using the `StringIndexer` to transform the values. For each `Transformer` we are defining the input column and the output column that will contain the modified value.

Step 11: Assemble the columns into a feature vector

We will use another `Transformer` to assemble the columns used in the classification by the XGBoost `Estimator` into a vector:

```
vectorAssembler = VectorAssembler()\
  .setInputCols(["Pclass", "SexIndex", "Age", "SibSp", "Parch",
"Fare", "CabinIndex", "EmbarkedIndex"])\
  .setOutputCol("features")
```

Step 12: Defining the XGBoostEstimator

In this step we are defining the `Estimator` that will produce the model. Most of the parameters used here are default:

```
xgboost = XGBoostEstimator(
    featuresCol="features",
    labelCol="Survival",
    predictionCol="prediction"
)
```

We only define the `feature, label` (have to match out columns from the `DataFrame`) and the new `prediction` column that contains the output of the classifier.

Step 13: Building the pipeline and the classifier

After we created all the individual steps we can define the actual pipeline and the order of the operations:

```
pipeline = Pipeline().setStages([sexIndexer, cabinIndexer,
embarkedIndexer, vectorAssembler, xgboost])
```

The input `DataFrame` will be transformed multiple times and in the end will produce the model trained with our data.

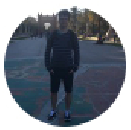
Step 14: Train the model and predict on new test data

We first split the data into train and test, then we fit the model with the train data and finally we see what predictions we have obtained for each passenger:

```
trainDF, testDF = df.randomSplit([0.8, 0.2], seed=24)

model = pipeline.fit(trainDF)

model.transform(testDF).select(col("PassengerId"),
col("prediction")).show()
```

WRITTEN BY

Bogdan Cojocar

Follow

Big data consultant. I write about the wonderful world of data.
If you enjoy my articles:

https://www.buymeacoffee.com/paxdata. Cheers!

Towards Data Science

A Medium publication sharing concepts, ideas, and codes.

Follow