Milestone3-Task4

April 16, 2022

1 Task 4 (Guided Exercise)

This notebook is part of Milestone 3, task 3 and is a guided exercise. I have put guidelines and helpful links (as comments) along with this notebook to take you through this.

In this exercise you will be using Spark's MLlib. The idea is to tune some hyperparameters of a Random Forest to find an optimum model. Once we know the optimum settings, we'll train a Random Forest in sklearn (task 4)and save it with joblib (task 5) (so that we can use it next week to deploy).

Here consider MLlib as another python package that you are using, like the scikit-learn. You will be seeing many scikit-learn similar classes and methods available in MLlib for various ML related tasks, you might also notice that some of them are not yet implimented in MLlib. What you write using pyspark package will be using the spark engine to run your code, and hence all the benefits of distributed computing what we discussed in class.

NOTE: Here whenever you use spark makes sure that you refer to the right documentation based on the version what you will be using. Here you can select the version of the spark and go to the correct documentation. In our case we are using spark 3.1.2, and here is the link to spark documetation that you can refer to, - MLlib Documentation - MLlib API Reference

You may notice that there are RDD-based API and DataFrame-based (Main Guide) API available in the documentation. You want to focus on DataFrame based API as no one these days use RDD based API. We will discuss the difference in class.

Before you start this notebook make sure that you are using EMR jupyterHub and the kernal that you selected is PySpark.

1.1 Import necessary libraries

[1]:
```
from pyspark.ml import Pipeline
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.ml.feature import VectorAssembler, UnivariateFeatureSelector
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.regression import RandomForestRegressor as sparkRFR
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import pandas as pd
```

```
Starting Spark application
```

```
<IPython.core.display.HTML object>
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
 ↪layout=Layout(height='25px', width='50%'),…
```

```
SparkSession available as 'spark'.
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
 ↪layout=Layout(height='25px', width='50%'),…
```

1.2 Read the data

To start with; read 100 data points for development purpose. Once your code is ready then try on the whole dataset.

[2]:
```
## Depending on the permissions that you provided to your bucket you might need␣
 ↪to provide your aws credentials
## to read from the bucket, if so provide with your credentials and pass as␣
 ↪storage_options=aws_credentials
# aws_credentials = {"key": "","secret": "","token":""}
## here 100 data points for testing the code
#pandas_df = pd.read_csv("s3://mds-s3-group18/output/ml_data_SYD.csv",␣
 ↪index_col=0, parse_dates=True).iloc[:100].dropna()
pandas_df = pd.read_csv("s3://mds-s3-group18/output/ml_data_SYD.csv",␣
 ↪index_col=0, parse_dates=True).dropna()
feature_cols = list(pandas_df.drop(columns="observed_rainfall").columns)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
 ↪layout=Layout(height='25px', width='50%'),…
```

1.3 Preparing dataset for ML

[3]:
```
# Load dataframe and coerce features into a single column called "Features"
# This is a requirement of MLlib
# Here we are converting your pandas dataframe to a spark dataframe,
# Here "spark" is a spark session I will discuss this in our Wed class.
# It is automatically created for you in this notebook.
# read more  here https://blog.knoldus.com/spark-createdataframe-vs-todf/
training = spark.createDataFrame(pandas_df)
assembler = VectorAssembler(inputCols=feature_cols, outputCol="Features")
training = assembler.transform(training).select("Features", "observed_rainfall")
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
 ↪layout=Layout(height='25px', width='50%'),…
```

1.4 Find best hyperparameter settings

Official Documentation of MLlib, Random forest regression here.

Here we will be mainly using following classes and methods;

- RandomForestRegressor
- ParamGridBuilder
  - addGrid
  - build
- CrossValidator
  - fit

Use these parameters for coming up with ideal parameters, you could try more parameters, but make sure you have enough power to do it. But you are required to try only following parameters. This will take around 15 min on entire dataset….

```
- Use numTrees as [10, 50,100]
- maxDepth as [5, 10]
- bootstrap as [False, True]
- In the CrossValidator use evaluator to be RegressionEvaluator(labelCol="Observed")
```

Additional reference: You can refer to here and here. Some additional reading here

[4]:
```
##Once you finish testing the model on 100 data points, then load entire␣
 ↪dataset and run , this could take ~15 min.
## write code here.
rf = sparkRFR(labelCol="observed_rainfall", featuresCol="Features")

rfparamGrid = (ParamGridBuilder()
               .addGrid(rf.maxDepth, [5, 10])
               .addGrid(rf.numTrees, [10, 50, 100])
               .addGrid(rf.bootstrap, [False, True])
             .build())

rfevaluator = RegressionEvaluator(predictionCol="prediction",␣
 ↪labelCol="observed_rainfall", metricName="rmse")

rfcv = CrossValidator(estimator = rf,
                          estimatorParamMaps = rfparamGrid,
                          evaluator = rfevaluator,
                          numFolds = 5)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
 ↪layout=Layout(height='25px', width='50%'),…
```

[5]:
```
cvModel = rfcv.fit(training)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
 ↪layout=Layout(height='25px', width='50%'),…
```

[6]:
```
# Print run info
print("\nBest model")
print("==========")
print(f"\nCV Score: {min(cvModel.avgMetrics):.2f}")
```

```
print(f"numTrees: {cvModel.bestModel.getNumTrees}")
print(f"maxDepth: {cvModel.bestModel.getMaxDepth()}")
print(f"bootstrap: {cvModel.bestModel.getBootstrap()}")
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
 ↪layout=Layout(height='25px', width='50%'),…


Best model
==========

CV Score: 8.17
numTrees: 100
maxDepth: 5
bootstrap: False
```