Local Configuration with Spark and Example Usage

Chengzhong LIU

Supervisor: Prof. WANG Yu-Hsing

Co-Supervisor: Dr. Tan Pin Siang

Abstract

This report introduces two ways in setting up local Spark with Zeppelin and Jupyter notebook on two systems: Windows and Ubuntu. Apart from this, the report also examines failed attempts in setting up local Spark to provide some insights for the future students in this UROP topic. A mini project in implementing a machine learning model with Spark MLlib will be introduced in accordance with Jupyter notebook on Windows. I do hope this project can be helpful to future students in this UROP project.

The project is available on GitHub at here. You can find more detailed instructions and more references in this repository, in case I missed out any importance. Also, due to the limitation in space and format, my report is only a representation of my work, and more information is store in this GitHub repository.

Installation of Apache Zeppelin on Ubuntu

Apache Zeppelin is one of the most powerful Spark interactive shell and widely used for research groups around the globe and currently supported with numerous popular interpreters[1]. However, the installation of Zeppelin on local computer is not a easy work and it is easy to lost for those who are new to Spark or Linux. So hereby, I would like to introduce a way to easily install Zeppelin on Ubuntu system (either on dual system or virtual machine). In my case, I have a virtual machine installed with Ubuntu, provided by VMWare[2]. If you are working on the UROP project in HKUST, a 60-days free trial should be enough. Otherwise, you might consider using VirtualBox[3] instead.

For detailed installation guide, please refer to my GitHub repository indicated in the Abstract. Note that brew is also available in MacOS. In fact, the original brew is designed for MacOS. You may refer to original tutorial mentioned in my GitHub page.
Make sure your local computer have fulfill these prerequisites:

- Ubuntu system on virtual machine / dual boot system
- Only paste the command after $ sign to your shell

Installation of Jupyter Notebook with Spark on Windows

Jupyter notebook, the interactive iPython shell, is welcomed by many coders and beginners in world. In this tutorial, I would recommended you to use Anaconda distribution[6]. You should also download Spark from Spark Official website[7] and extracted the folder to your local directory. You will also need to install and extra python package named ”findspark”.

For detailed installation guide, please refer to my GitHub repository indicated in the Abstract. Make sure your local computer have fulfill these prerequisites:

- Anaconda in your local PC
- Spark in your local directory
- Java (remember to set the environment variable)
- Access to Windows Powershell

Failed Attempts

It is not easy to finished the previous two completed tutorial. I have had so many failures before I finally figured out a way. So I'm suggesting that, if you are considering setup local Spark in the following ways, please think twice before your proceed. As I'm a primarily Windows user, many of my attempts are associated with Windows.

- Windows Jupyter scala kernel with spark
- Windows Spark-notebook
- Windows Windows Ubuntu subsystem
- Windows Run zeppelin locally
- Windows Jupyter notebook with Toree kernel
- Ubuntu Jupyter notebook with Toree kernel

Introduction to Spark Usage (on Jupyter)

As my local computer's operating system is Windows, I choose to use Jupyter notebook work with pyspark to achieve best performance. The data is available from Kaggle[4]. You can also access the original zip file or extracted csv file in my GitHub repository under directory ”data”.

Before we start our project, please make sure you have opened a Spark Session, and denoted to spark (which is the basis of the following analysis).

```
# open spark session
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
spark = SparkSession(sc)
spark
```

Reading csv is also an essential step in model formulation. Reading local csv and transform into Spark dataframe can be easily achieved by spark built-in function. You are also recommended to do a check to see if dataset has been loaded correctly: functions like count() and show().

```
# load dataset
df = spark.read.csv("path/to/my_data.csv", header = True, mode = "DROPMALFORMED")
df.count()
```

Sometimes the Spark dataframe is not powerful enough to provide sufficient dataframe operations. In this case, you can opt to use ”toPandas()” function to transform Spark dataframe into pandas dataframe. However, bear in mind that this function is only available when your data volume is small, as pandas library is not designed for data in large scale. You can check whether the transformation is successful by using ”head()” function.

```
# use pandas to get_dummies
import pandas as pd
pd_df = pd.get_dummies(df.select(*).toPandas())
pd_df.head(3)
```

The Pipeline of Analysis

The pipeline of my analysis is mainly adapted from one of the kernel[5] associated with the Kaggle dataset[4]. This kernel gets the majority votes from the community and transforming the work into Spark would be a meaningful exercise for those beginners in Spark.

The main pipeline can be found in the notebook ”attrition_comprehensive.ipynb”, which is partially a ”translation” in Spark from the original kernel (largely use pandas) and partially a liberal research in Spark exploration. The notebook ”attrition_final.ipynb” is a simplified version of the previous notebook and aims to impress people with how Spark can to on machine learning in a nutshell, while the notebook ”attrition_simple.ipynb” is an intermediated version which demonstrate how I identified a problem with Spark dataframe and how I bypass that. Ironically, I still cannot solve the problem due to my limited knowledge. However, I do suspect that this problem might be cause by the compatibility of Spark with newest Python 3.6.

The major pipeline can be described the following. For details, please refer to my workbook on GitHub.

- Data Exploration
- Feature Engineering
- Transform Categorical Data
- Apply Models
- Evaluation of Results

Model Construction with MLlib

Setting up model with Spark can be actually done in a quite standardized way. In my case, I use a decision tree model to predict employee attrition (for reference, please see the notebook in GitHub). Suppose you have a dataframe named ”df”. The first thing you need to do is to assemble the features and split the data into training and testing sets. The code is similar to this:

```
# import essiential libraries and prepare training and testing datasets
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer
assembler = VectorAssembler(inputCols = feature_col, outputCol = "features")
assembled = assembler.transform(df)
```

```
(training_data, testing_data) = assembled.randomSplit([0.8, 0.2], seed = 13234)

training_data.count(), testing_data.count()
```

Training model is actually quite fast and convenient. The Spark model would eventually produce a dataframe with some additional columns associated with the prediction.

```
# get the decision tree

dt = DecisionTreeClassifier(labelCol = "target", \

featuresCol = "features", maxDepth = 7, \

minInstancesPerNode = 20, impurity = "gini")

# train and get the model

pipeline = Pipeline(stages = [dt])

model = pipeline.fit(training_data)

# get the prediction model

predictions = model.transform(testing_data)
```

We can calculate the accuracy with number of correct cases with number of incorrect cases.

```
correct = predictions.filter(predictions["prediction"] == \

predictions["target"]).select("prediction", "target").count()

total = predictions.count()

print("The precision is: "+"{:.2%}".format(correct / total))
```

Unresolved Problem

I recorded one special observation in notebook ”attrition_simple.ipynb”. In the final version of my notebook, I use ”join” condition in pandas instead of ”join” in Spark. During the dataframe concatenation, both dataframe works fine before ”join” by Spark. However, the newly generated dataframe behaves quite abnormally. I’m able to print the schema of this new dataframe, but unable to use functions like ”show()”, ”count()”. Observed codes (both dataframe with same number of rows):

```
# no error
df1.count()
# no error
df2.count()
# error occurs here
df1.join(df2).count()
```

Acknowledgement & Future Contact

For this project, I owe my special thanks for Prof. WANG Yu-Hsing and Dr. Tan Pin Siang. Without them, I would be so lost that I could not finish the task. I have been ”fighting” with the environment for more than a month at the start of the UROP project. So I wrote as detail as I can in the GitHub page to help future students in this UROP topic to set up with local Spark faster, so that they can devote more on the Spark project itself rather than setting up the environment.

For future UROP students in this topic: My GitHub repository is always open. You are very welcomed to leave me an issue in the ”Issue” page, and I will try my best to help you at my earliest convenience.

References

[1] Foundation, T. A. (n.d.). Apache Zeppelin. Retrieved August 3, 2018 (https://zeppelin.apache.org/)

[2] VMware – Official Site. (2018, July 30). Retrieved August 3, 2018 (https://www.vmware.com/asean.html)

[3] Welcome to VirtualBox.org! (n.d.). Retrieved August 3, 2018 (https://www.virtualbox.org/)

[4] P. (2017, March 31). IBM HR Analytics Employee Attrition, Kaggle. Retrieved August 3, 2018 (https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset/home)

[5] Employee attrition via RF, GBM, Kaggle. (n.d.). Retrieved August 4, 2018 (https://www.kaggle.com/arthurtok/employee-attrition-via-rf-gbm)

[6] Anaconda. (2018, August 03). Home. Retrieved August 4, 2018 (https://www.anaconda.com/)

[7] Retrieved August 4, 2018 (https://spark.apache.org/)