



Learning Apache Spark with Python

Release v1.0

Wenqiang Feng

May 23, 2017

CONTENTS

1	Preface	3
1.1	About	3
1.2	Motivation for this tutorial	3
1.3	Acknowledgement	4
1.4	Feedback and suggestions	4
2	Why Spark with Python ?	5
2.1	Why Spark?	5
2.2	Why Spark with Python (PySpark)?	7
3	Getting Started	9
3.1	Run on Databricks Community Cloud	9
3.2	Set up Spark on Mac and Ubuntu	14
3.3	PySpark With Text Editor or IDE	14
3.4	Set up Spark on Cloud	15
3.5	Demo Code in this Section	18
4	An Introduction to Apache Spark	19
4.1	Core Concepts	19
4.2	Spark Components	19
4.3	Architecture	22
4.4	How Spark Works?	22
5	Programming with RDDs	23
5.1	Spark Transformations	23
5.2	Spark Actions	23
6	Regression	25
6.1	Linear Regression	27
6.2	Generalized linear regression	27
6.3	Decision tree Regression	27
6.4	Random Forest Regression	27
6.5	Gradient-boosted tree regression	27
7	Classification	29
7.1	Logistic regression	29

7.2	Decision tree Classification	29
7.3	Random forest Classification	29
7.4	Gradient-boosted tree Classification	29
7.5	Naive Bayes Classification	29
7.6	Support Vector Machines Classification	29
8	Clustering	31
8.1	K-Means Model	31
9	Text Mining	33
9.1	Text Preprocessing	33
9.2	Text Classification	33
9.3	Sentiment analysis	33
9.4	N-grams and Correlations	33
9.5	Topic Model: Latent Dirichlet Allocation	33
10	Social Network Analysis	35
10.1	Co-occurrence Network	35
10.2	Correlation Network	38
11	Neural Network	39
11.1	Feedforward Neural Network	39
12	Main Reference	43
	Bibliography	45
	Index	47



Welcome to our **Learning Apache Spark with Python** note! In these note, you will learn a wide array of concepts about **PySpark** in Data Mining, Text Mining, Machine Learning and Deep Learning.

1.1 About

1.1.1 About this note

This is a shared repository for Learning Apache Spark Notes. The first version was posted on Github in [\[Feng2017\]](#). This shared repository mainly contains the self-learning and self-teaching notes from Wenqiang during his [IMA Data Science Fellowship](#).

In this repository, I try to use the detailed demo code and examples to show how to use each main functions. If you find your work wasn't cited in this note, please feel free to let me know.

Although I am by no means an data mining programming and Big Data expert, I decided that it would be useful for me to share what I learned about PySpark programming in the form of easy tutorials with detailed example. I hope those tutorials will be a valuable tool for your studies.

The tutorials assume that the reader has a preliminary knowledge of programing and Linux. And this document is generated automatically by using [sphinx](#).

1.1.2 About the authors

- **Wenqiang Feng**
 - Phd in Mathematics
 - University of Tennessee at Knoxville
 - Email: wfeng1@utk.edu

1.2 Motivation for this tutorial

I was motivated by the [IMA Data Science Fellowship](#) project to learn PySpark. After that I was impressed and attracted by the PySpark. And I foud that:

1. It is no exaggeration to say that Spark is the most powerful Bigdata tool.

2. However, I still found that learning Spark was a difficult process. I have to Google it and identify which one is true. And it was hard to find detailed examples which I can easily learned the full process in one file.
3. Good sources are expensive for a graduate student.

1.3 Acknowledgement

At here, I would like to thank Ming Chen, Jian Sun and Zhongbo Li at the University of Tennessee at Knoxville for the valuable discussion and thank the generous anonymous authors for providing the detailed solutions and source code on the internet. Without those help, this repository would not have been possible to be made. Wenqiang also would like to thank the [Institute for Mathematics and Its Applications \(IMA\)](#) at [University of Minnesota, Twin Cities](#) for support during his IMA Data Scientist Fellow visit.

1.4 Feedback and suggestions

Your comments and suggestions are highly appreciated. I am more than happy to receive corrections, suggestions or feedbacks through email (wfeng1@utk.edu) for improvements.

WHY SPARK WITH PYTHON ?

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

I want to answer this question in two folders:

2.1 Why Spark?

I think the following four main reasons from [Apache Spark™](#) official website are good enough to convince you to use Spark.

1. Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Apache Spark has an advanced DAG execution engine that supports acyclic data flow and in-memory computing.

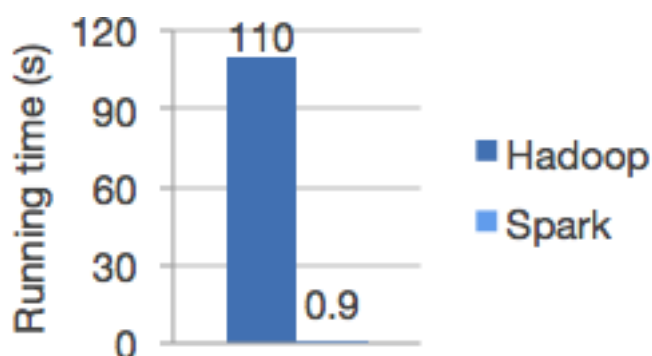


Figure 2.1: Logistic regression in Hadoop and Spark

2. Ease of Use

Write applications quickly in Java, Scala, Python, R.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.

3. Generality

Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.

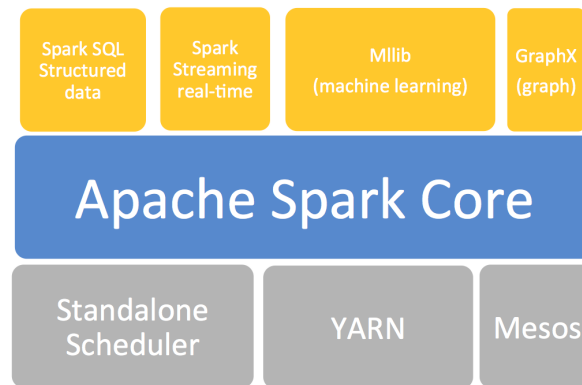


Figure 2.2: The Spark stack

4. Runs Everywhere

Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.



Figure 2.3: The Spark platform

2.2 Why Spark with Python (PySpark)?

No matter you like it or not, Python has been one of the most popular programming languages.

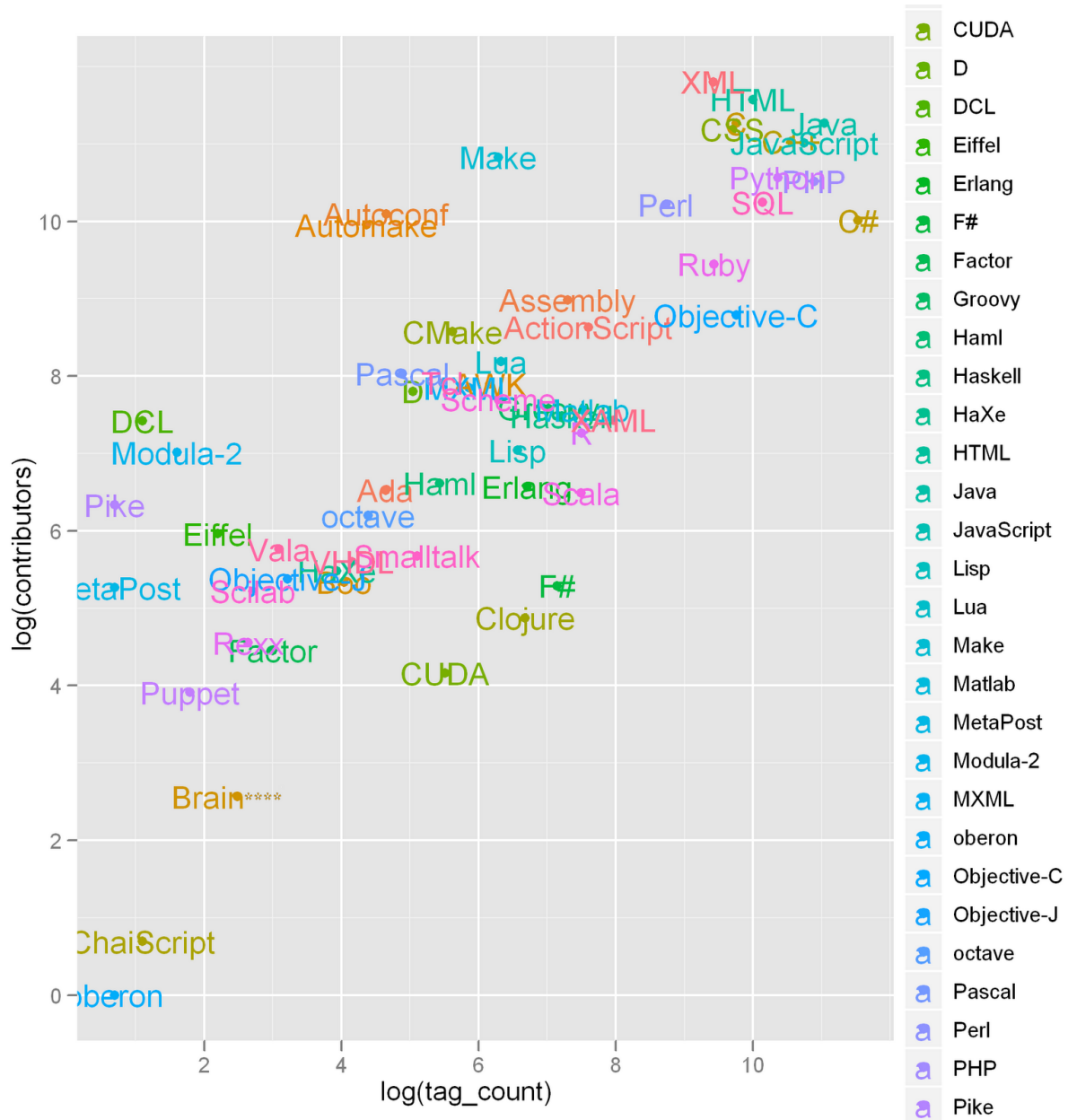


Figure 2.4: programming language popularity from R-Chart

GETTING STARTED

Note: Good tools are prerequisite to the successful execution of a job. – old Chinese proverb

A good programming platform can save you lots of troubles and time. Herein I will only present how to install my favorite programming platform and only show the easiest way which I know to set it up on Linux system. If you want to install on the other operator system, you can Google it. In this section, you may learn how to set up Pyspark on the corresponding programming platform and package.

3.1 Run on Databricks Community Cloud

If you don't have any experience with Linux or Unix operator system, I would love to recommend you to use Spark on Databricks Community Cloud. Since you do not need to setup the Spark and it's totally **free** for Community Edition. Please follow the steps listed below.

1. Sign up a account at: <https://community.cloud.databricks.com/login.html>
2. Sign in with your account, then you can creat your cluster(machine), table(dataset) and notebook(code).
3. Create your cluster where your code will run
4. Import your dataset

Note: You need to save the path which appears at Uploaded to DBFS: /File-Store/tables/05rmhuqv1489687378010/. Since we will use this path to load the dataset.

5. Creat your notebook



File Edit View History Tools People Help

Databricks

Secure | <https://community.cloud.databricks.com/?o=4622560542654492>

Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments

Upgrade ? @George

Community Edition (2.45)

Welcome to databricks™

Featured Notebooks

- Introduction to Apache Spark on Databricks
- Databricks for Data Scientists
- Introduction to Structured Streaming

New

- Notebook
- Job
- Cluster
- Table
- Library

Documentation

- Databricks Guide
- Python, R, Scala, SQL
- Importing Data

Open Recent

- Databricks for Data Scientists
- linearRegression

What's new?

- Automatic termination of clusters
- Autoscaling local storage with EBS volumes (beta)

[Latest release notes](#)

Send Feedback

File Edit View History Tools People Help

Create Cluster - Databricks

Secure | <https://community.cloud.databricks.com/?o=4622560542654492#create/cluster>

Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments

Upgrade ? @George

Create Cluster

New Cluster

Cancel Create Cluster

0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 6.0 GB Memory, 0.88 Cores, 1 DBU

Cluster Name
MLmachine

Databricks Runtime Version
Spark 2.1 (Auto-updating, Scala 2.10)

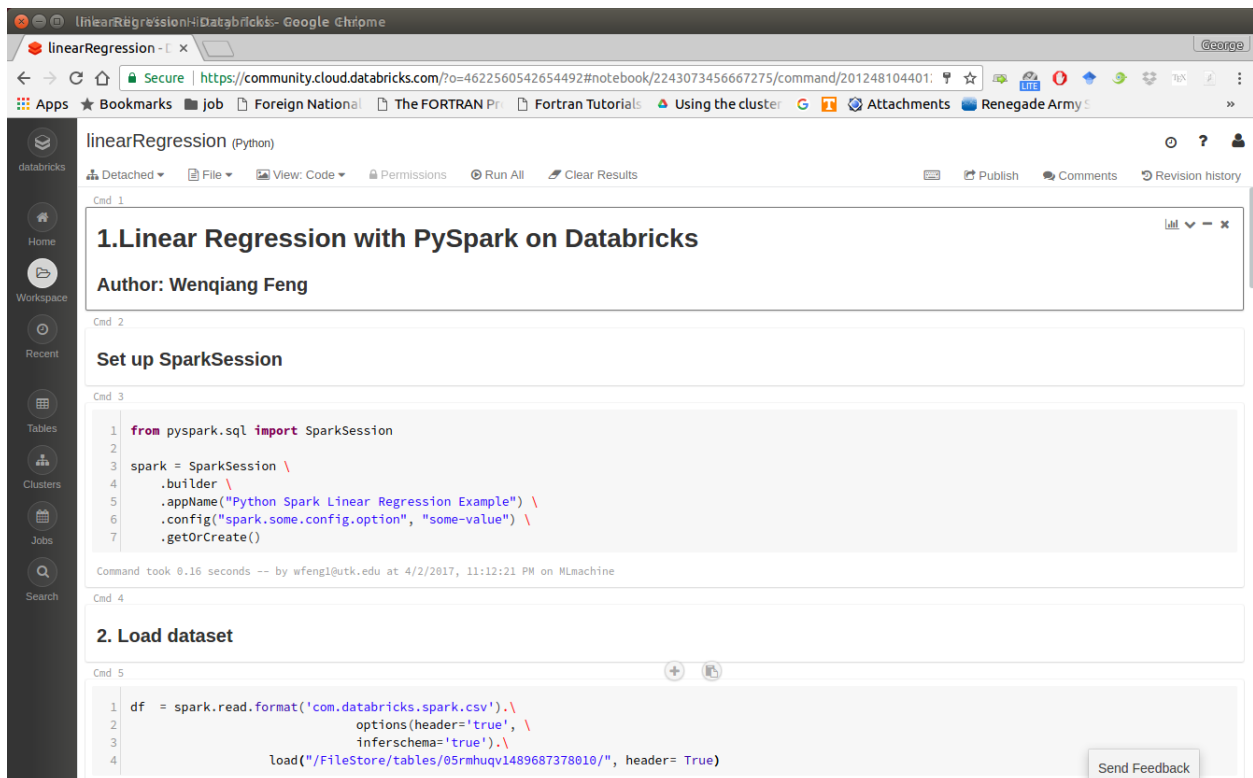
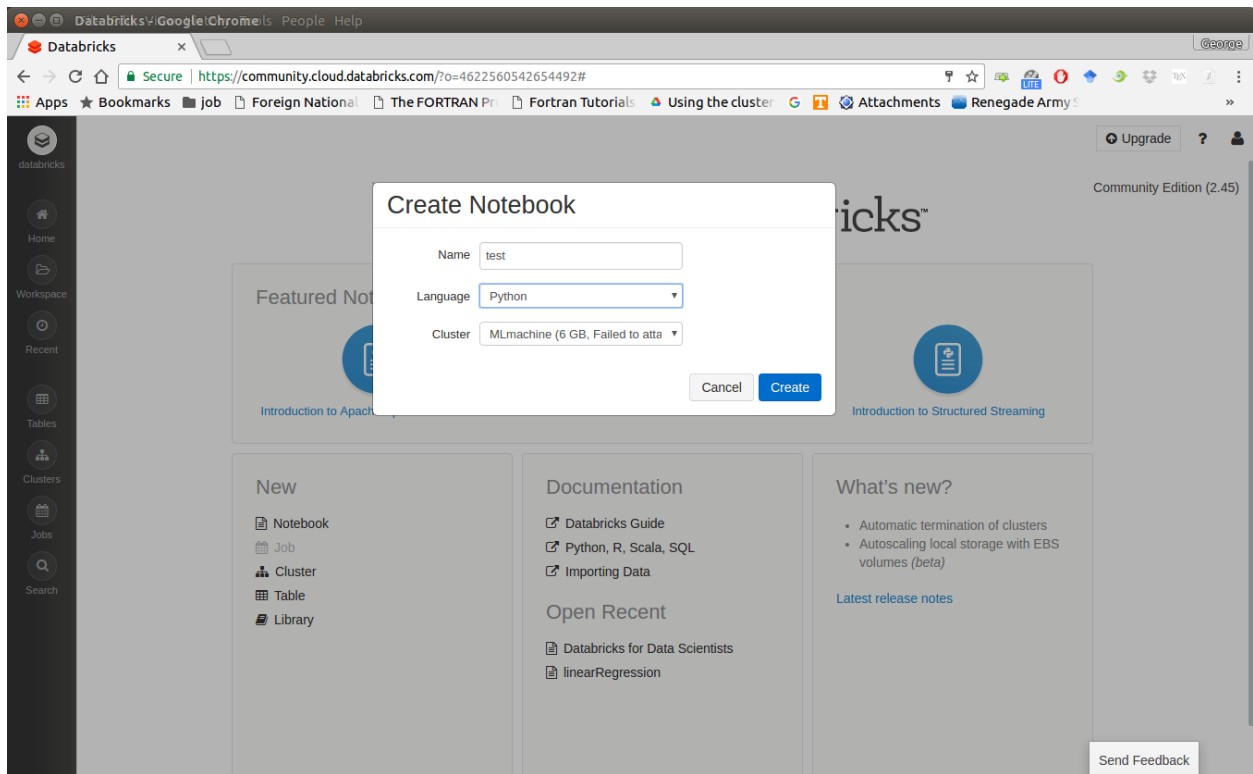
Instance
Free 6GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For [more configuration options](#), please [upgrade your Databricks subscription](#).

AWS Spark

Availability Zone
us-west-2c

Send Feedback





After finishing the above 5 steps, you are ready to run your Spark code on Databricks Community Cloud. I will run all the following demos on Databricks Community Cloud. Hopefully, when you run the demo code, you will get the following results:

```
+---+-----+-----+-----+-----+
|_c0|      TV|Radio|Newspaper|Sales|
+---+-----+-----+-----+-----+
|  1|230.1| 37.8|      69.2| 22.1|
|  2| 44.5| 39.3|      45.1| 10.4|
|  3| 17.2| 45.9|      69.3|  9.3|
|  4|151.5| 41.3|      58.5| 18.5|
|  5|180.8| 10.8|      58.4| 12.9|
+---+-----+-----+-----+-----+
only showing top 5 rows

root
|-- _c0: integer (nullable = true)
|-- TV: double (nullable = true)
|-- Radio: double (nullable = true)
|-- Newspaper: double (nullable = true)
|-- Sales: double (nullable = true)
```

3.2 Set up Spark on Mac and Ubuntu

• Installing Python

Go to Ubuntu Software Center and follow the following steps:

1. Open Ubuntu Software Center
2. Search for python
3. And click Install

Or Open your terminal and using the following command:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev
                        libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
sudo apt-get install python
sudo easy_install pip
sudo pip install ipython
```

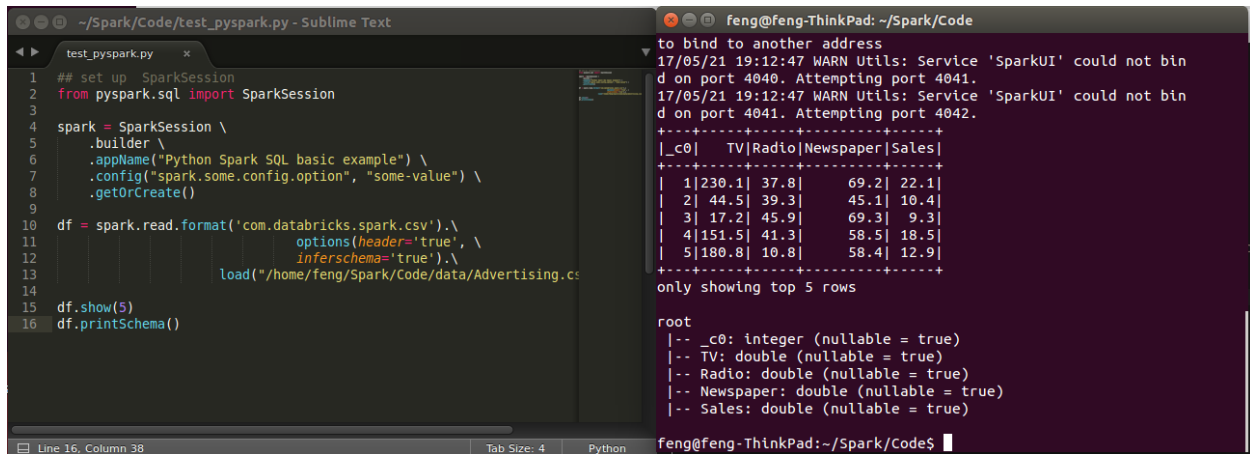
3.3 PySpark With Text Editor or IDE

3.3.1 PySpark With Sublime Text

After you finishing the above setup steps in *Set up Spark on Mac and Ubuntu*, then you should be good to use Sublime Text to write your PySpark Code and run your code as a normal python code in Terminal.

```
python test_pyspark.py
```

Then you should get the putput results in your terminal.



```
test_pyspark.py
1 ## set up SparkSession
2 from pyspark.sql import SparkSession
3
4 spark = SparkSession \
5     .builder \
6     .appName("Python Spark SQL basic example") \
7     .config("spark.some.config.option", "some-value") \
8     .getOrCreate()
9
10 df = spark.read.format('com.databricks.spark.csv').\
11     options(header='true', \
12             inferSchema='true').\
13     load("/home/feng/Spark/Code/data/Advertising.csv")
14
15 df.show(5)
16 df.printSchema()
```

```

feng@feng-ThinkPad: ~/Spark/Code
to bind to another address
17/05/21 19:12:47 WARN Utils: Service 'SparkUI' could not bind
d on port 4040. Attempting port 4041.
17/05/21 19:12:47 WARN Utils: Service 'SparkUI' could not bind
d on port 4041. Attempting port 4042.
+-----+-----+-----+-----+-----+
|_c0|  TV|Radio|Newspaper|Sales|
+-----+-----+-----+-----+
| 1|230.1| 37.8|    69.2| 22.1|
| 2| 44.5| 39.3|    45.1| 10.4|
| 3| 17.2| 45.9|    69.3|  9.3|
| 4|151.5| 41.3|    58.5| 18.5|
| 5|180.8| 10.8|    58.4| 12.9|
+-----+-----+-----+-----+
only showing top 5 rows

root
|-- _c0: integer (nullable = true)
|-- TV: double (nullable = true)
|-- Radio: double (nullable = true)
|-- Newspaper: double (nullable = true)
|-- Sales: double (nullable = true)
feng@feng-ThinkPad:~/Spark/Code$
```

3.3.2 PySpark With Eclipse

If you want to run PySpark code on Eclipse, you need to add the paths for the **External Libraries** for your **Current Project** as follows:

1. Open the properties of your project
2. Add the paths for the **External Libraries**

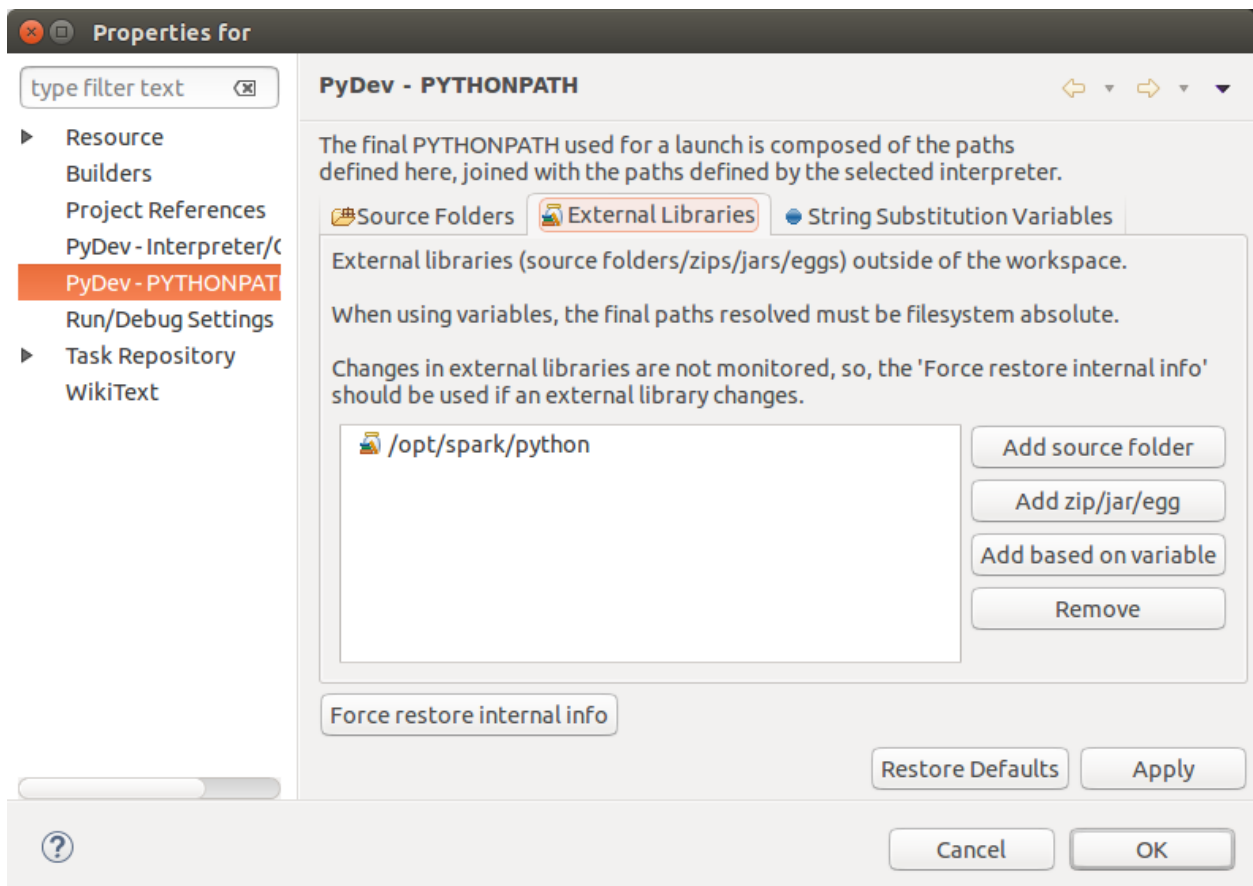
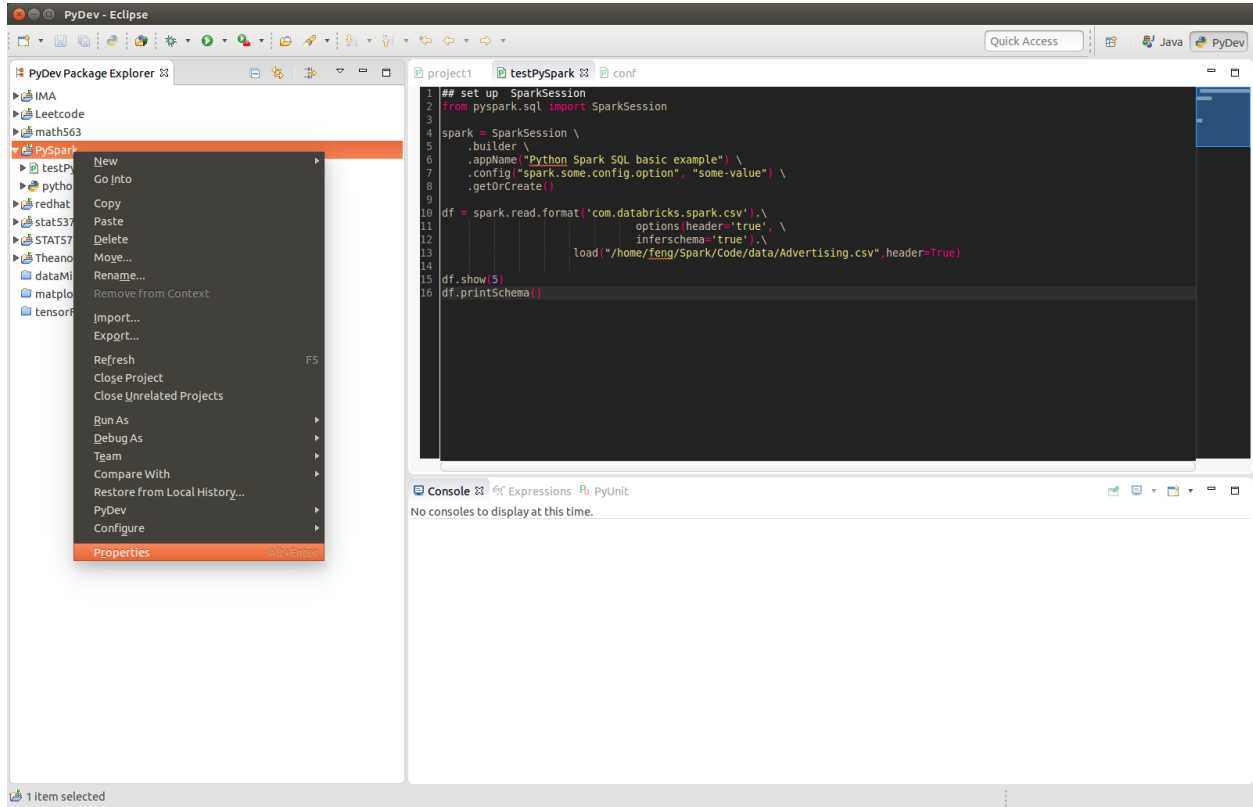
And then you should be good to run your code on Eclipse with PyDev.

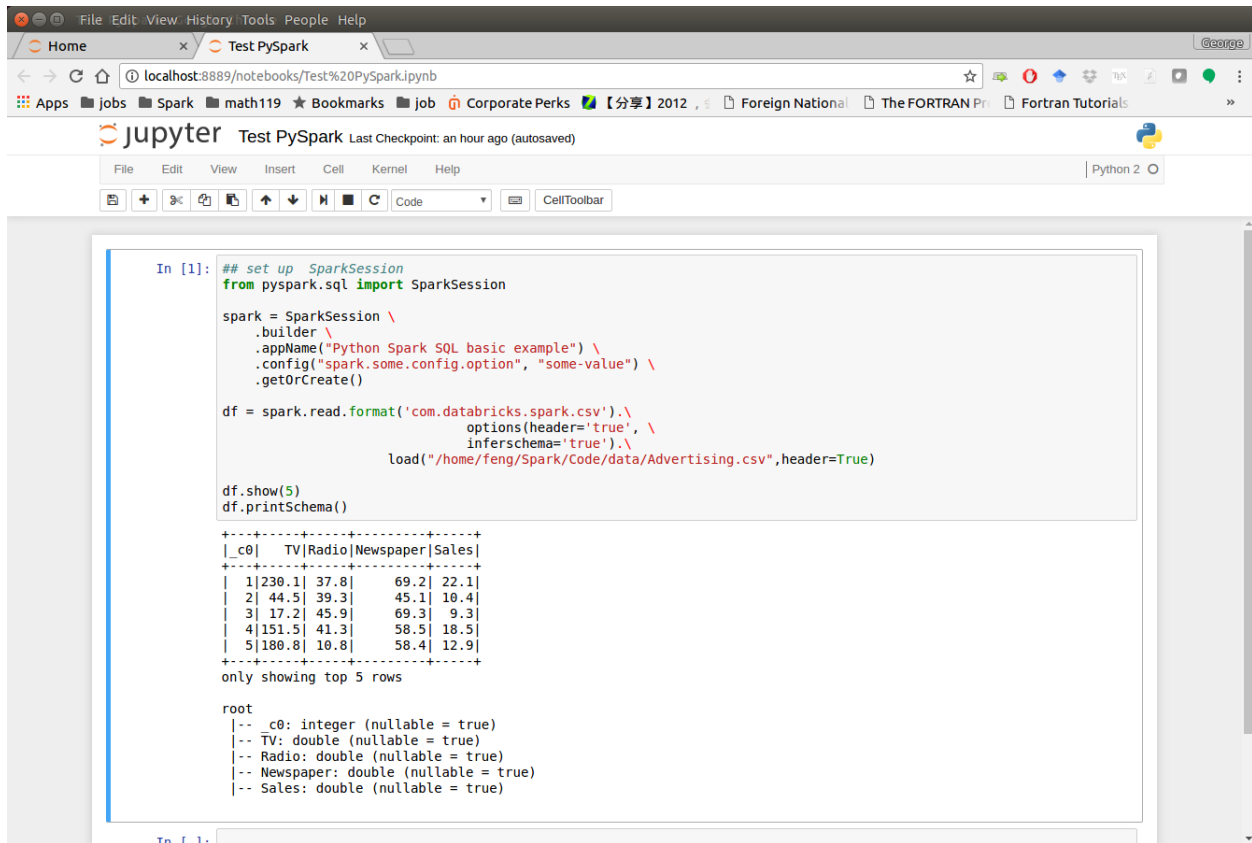
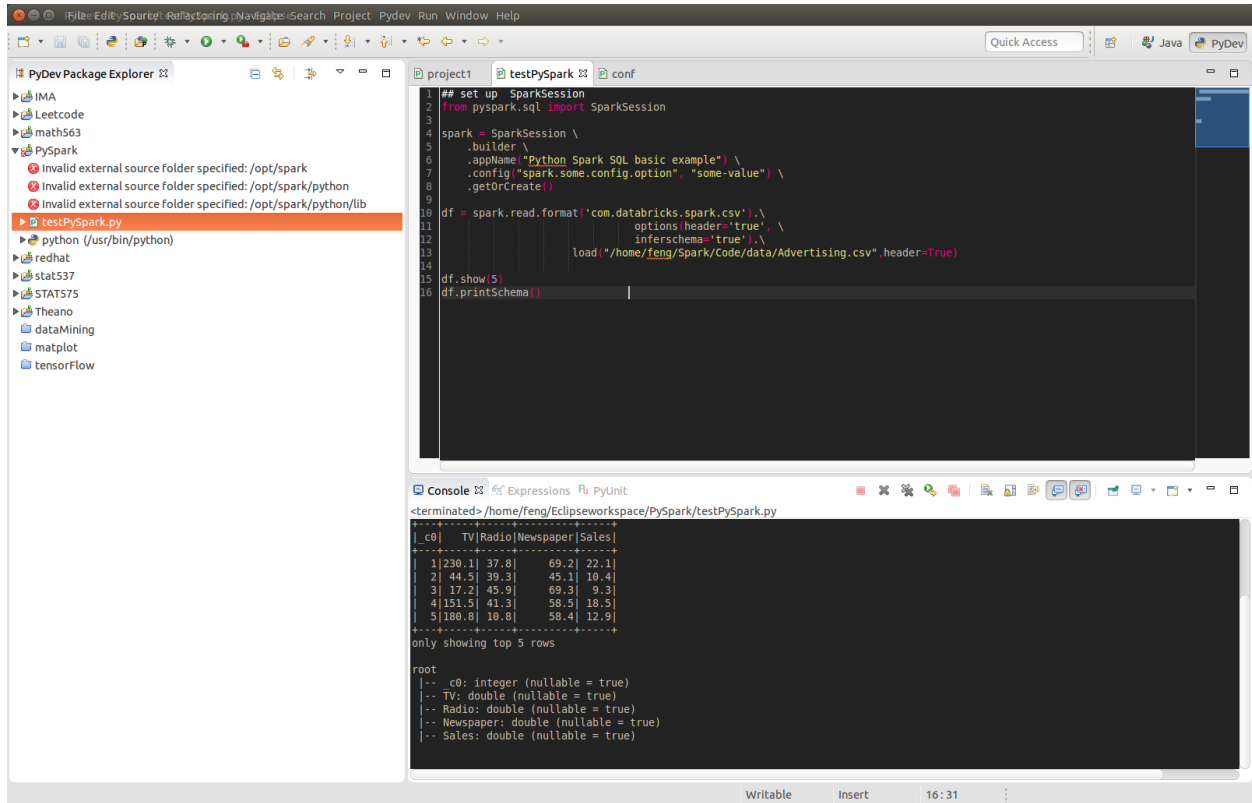
3.3.3 PySpark With Jupyter Notebook

After you finishing the above setup steps in *Set up Spark on Mac and Ubuntu*, then you should be good to use write and run your PySpark Code in Jupyter notebook.

3.4 Set up Spark on Cloud

Folloing the setup steps in *Set up Spark on Mac and Ubuntu*, you can set up your own cluster on the cloud, for example AWS, Google Cloud. Actually, for those clouds, they have their own Big Data tool. Yon can run





them directly without any setting just like Databricks Community Cloud. If you want more details, please feel free to contact with me.

3.5 Demo Code in this Section

The code for this section is available for download `test_pyspark`,

- Python Source code

```
## set up SparkSession
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.read.format('com.databricks.spark.csv').\
    options(header='true', \
             inferSchema='true').\
    load("/home/feng/Spark/Code/data/Advertising.csv", header=True)

df.show(5)
df.printSchema()
```

AN INTRODUCTION TO APACHE SPARK

Note: **Know yourself and know your enemy, and you will never be defeated** – idiom, from Sunzi's Art of War

4.1 Core Concepts

Apache Spark core concepts, architecture and internals

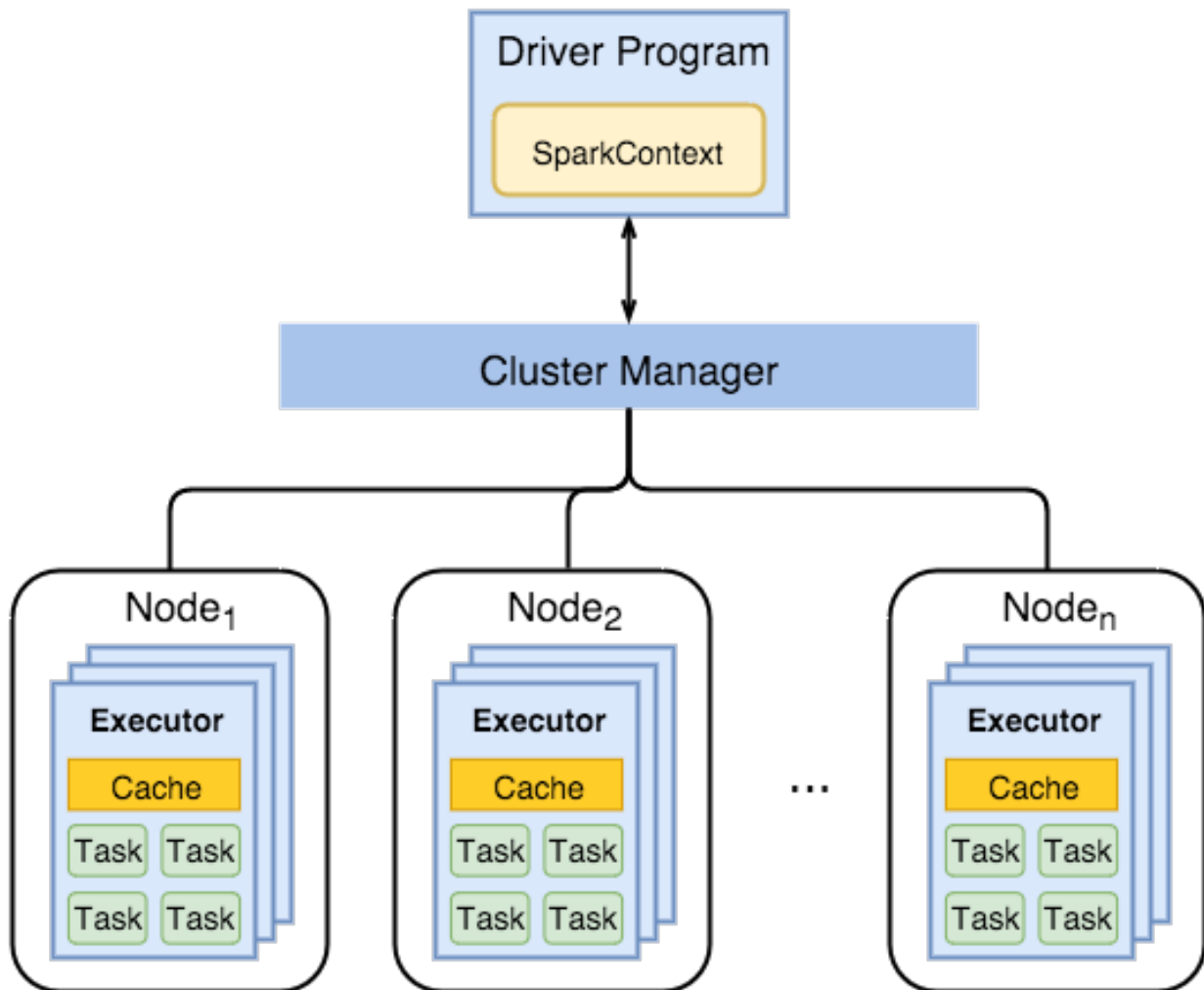
Before diving deep into how Apache Spark works, let's understand the jargon of Apache Spark

- **Job:** A piece of code which reads some input from HDFS or local, performs some computation on the data and writes some output data.
- **Stages:** Jobs are divided into stages. Stages are classified as a Map or reduce stages (It's easier to understand if you have worked on Hadoop and want to correlate). Stages are divided based on computational boundaries, all computations (operators) cannot be updated in a single Stage. It happens over many stages.
- **Tasks:** Each stage has some tasks, one task per partition. One task is executed on one partition of data on one executor (machine).
- **DAG:** DAG stands for Directed Acyclic Graph, in the present context it's a DAG of operators.
- **Executor:** The process responsible for executing a task.
- **Master:** The machine on which the Driver program runs
- **Slave:** The machine on which the Executor program runs

4.2 Spark Components

1. Spark Driver

- separate process to execute user applications
- creates SparkContext to schedule jobs execution and negotiate with cluster manager



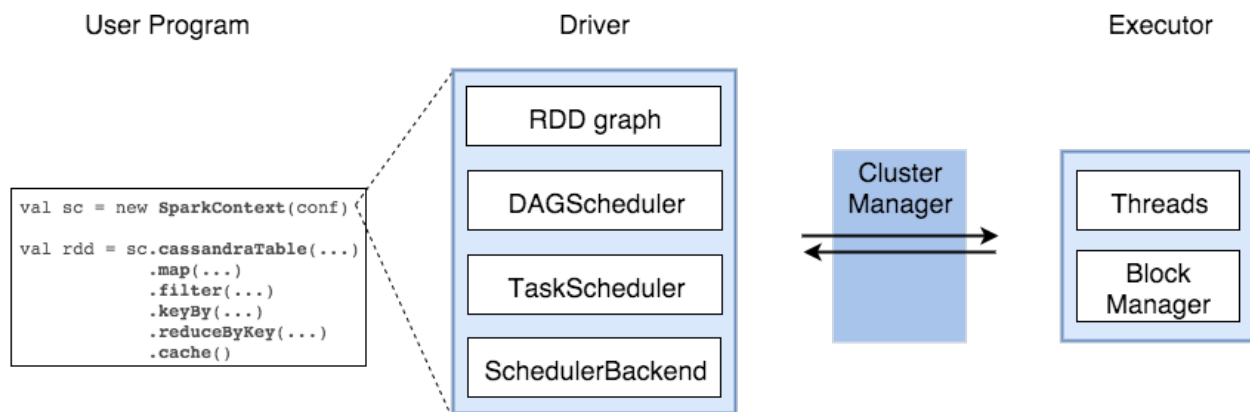
2. Executors

- run tasks scheduled by driver
- store computation results in memory, on disk or off-heap
- interact with storage systems

3. Cluster Manager

- Mesos
- YARN
- Spark Standalone

Spark Driver contains more components responsible for translation of user code into actual jobs executed on cluster:



- `SparkContext`
 - represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster
- `DAGScheduler`
 - computes a DAG of stages for each job and submits them to `TaskScheduler` determines preferred locations for tasks (based on cache status or shuffle files locations) and finds minimum schedule to run the jobs
- `TaskScheduler`
 - responsible for sending tasks to the cluster, running them, retrying if there are failures, and mitigating stragglers
- `SchedulerBackend`
 - backend interface for scheduling systems that allows plugging in different implementations (Mesos, YARN, Standalone, local)
- `BlockManager`

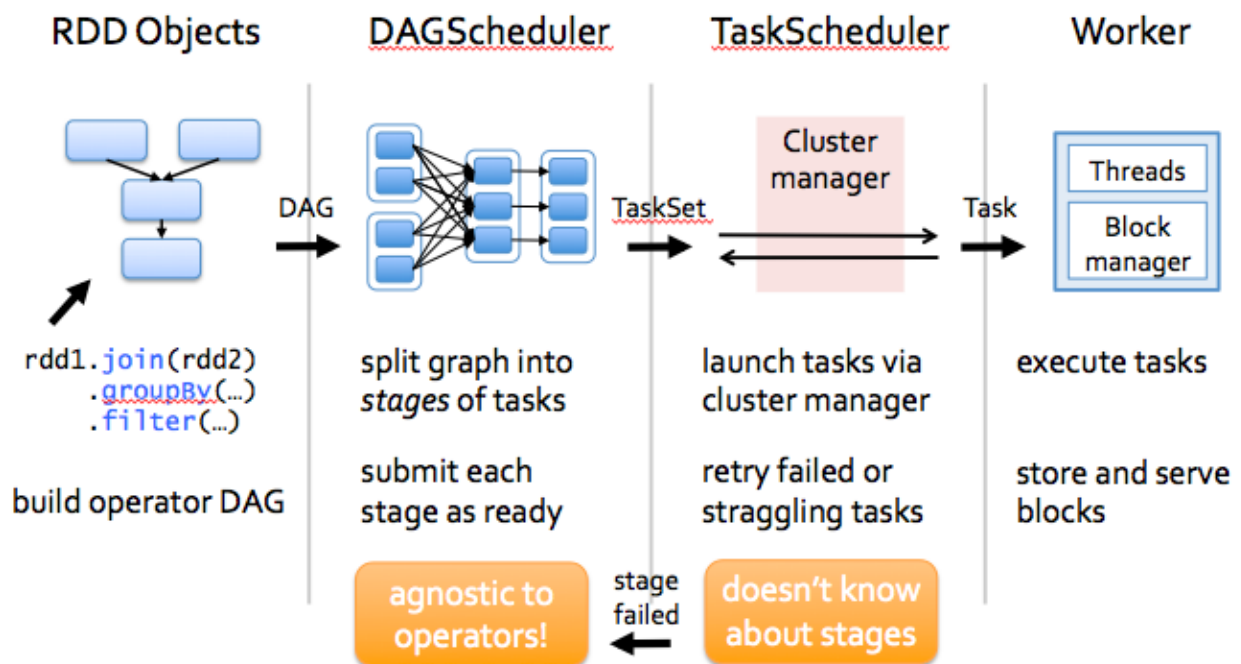
- provides interfaces for putting and retrieving blocks both locally and remotely into various stores (memory, disk, and off-heap)

4.3 Architecture

4.4 How Spark Works?

Spark has a small code base and the system is divided in various layers. Each layer has some responsibilities. The layers are independent of each other.

The first layer is the interpreter, Spark uses a Scala interpreter, with some modifications. As you enter your code in spark console (creating RDD's and applying operators), Spark creates a operator graph. When the user runs an action (like collect), the Graph is submitted to a DAG Scheduler. The DAG scheduler divides operator graph into (map and reduce) stages. A stage is comprised of tasks based on partitions of the input data. The DAG scheduler pipelines operators together to optimize the graph. For e.g. Many map operators can be scheduled in a single stage. This optimization is key to Sparks performance. The final result of a DAG scheduler is a set of stages. The stages are passed on to the Task Scheduler. The task scheduler launches tasks via cluster manager. (Spark Standalone/Yarn/Mesos). The task scheduler doesn't know about dependencies among stages.



PROGRAMMING WITH RDDS

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

An RDD in Spark is simply an immutable distributed collection of objects. Each RDD is split into multiple partitions, which may be computed on different nodes of the cluster.

Once created, RDDs offer two types of operations: transformations and actions.

5.1 Spark Transformations

Transformations construct a new RDD from a previous one. For example, one common transformation is filtering data that matches a predicate.

5.2 Spark Actions

Actions, on the other hand, compute a result based on an RDD, and either return it to the driver program or save it to an external storage system (e.g., HDFS).

REGRESSION

Note: A journey of a thousand miles begins with a single step – old Chinese proverb

The following is one of my favorite Cheat Sheet of the Data Mining Algorithms which was proposed by Laura Diane Hamilton on September 09, 2014 at <http://www.lauradhamilton.com/machine-learning-algorithm-cheat-sheet>. From this cheat sheet, you can have a basic idea of those algorithms.

Table 6.1: Cheat Sheet table of Data Mining Algorithms (Copyright belongs to Laura Diane Hamilton)

Algorithm	Pro	Cons	Good at
Linear regression	<ul style="list-style-type: none"> • Very fast(runs in constant time) • Easy to understand the model • Random Forest • Less prone to over-fitting 	<ul style="list-style-type: none"> • Unable to model complex relationships • Unable to capture nonlinear relationships without first transforming the inputs 	<ul style="list-style-type: none"> • The first look at a dataset • Numerical data with lots of features
Decision tree	<ul style="list-style-type: none"> • Fast • Robust to noise and missing values • Accurate 	<ul style="list-style-type: none"> • Complex trees are hard to interpret • Duplication within the same sub-tree is possible 	<ul style="list-style-type: none"> • Star classification • Medical diagnosis • Credit risk analysis
Neural networks	<ul style="list-style-type: none"> • Extremely powerful • Can model even very complex relationships • No need to understand the underlying data • Almost works by “magic” 	<ul style="list-style-type: none"> • Prone to overfitting • Long training time • Requires significant computing power for large datasets • Model is essentially unreadable 	<ul style="list-style-type: none"> • Images • Video • “Human-intelligence” type tasks like driving or flying • Robotics
SVM	<ul style="list-style-type: none"> • Can model complex, nonlinear relationships • Robust to noise (because they maximize margins) 	<ul style="list-style-type: none"> • Need to select a good kernel function • Model parameters are difficult to interpret • Sometimes numerical stability problems • Requires significant memory and processing power 	<ul style="list-style-type: none"> • Classifying proteins • Text classification • Image classification • Handwriting recognition
kNN	<ul style="list-style-type: none"> • Simple • Powerful • No training involved (lazy) 	<ul style="list-style-type: none"> • Expensive and slow to predict new instances • Must define a meaningful distance function 	<ul style="list-style-type: none"> • Low-dimensional datasets • Fault detection in semiconductor manufacturing
26	<ul style="list-style-type: none"> • Naturally handles multiclass classification and regression 	<ul style="list-style-type: none"> • Performs poorly on high- 	<p>Chapter 6. Regression</p> <ul style="list-style-type: none"> • Video content retrieval • Gene expression

6.1 Linear Regression

Given that $\{y_i, x_{i1}, \dots, x_{ip}\}_{i=1}^n$

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Linear Regression Example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

6.2 Generalized linear regression

6.3 Decision tree Regression

6.4 Random Forest Regression

6.5 Gradient-boosted tree regression

CLASSIFICATION

Note: *Birds of a feather folock together.* – old Chinese proverb

7.1 Logistic regression

7.1.1 Binomial logistic regression

7.1.2 Multinomial logistic regression

7.2 Decision tree Classification

7.3 Random forest Classification

7.4 Gradient-boosted tree Classification

7.5 Naive Bayes Classification

7.6 Support Vector Machines Classification

CLUSTERING

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

8.1 K-Means Model

TEXT MINING

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

I want to answer this question in two folders:

9.1 Text Preprocessing

9.2 Text Classification

9.3 Sentiment analysis

9.4 N-grams and Correlations

9.5 Topic Model: Latent Dirichlet Allocation

SOCIAL NETWORK ANALYSIS

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

10.1 Co-occurrence Network

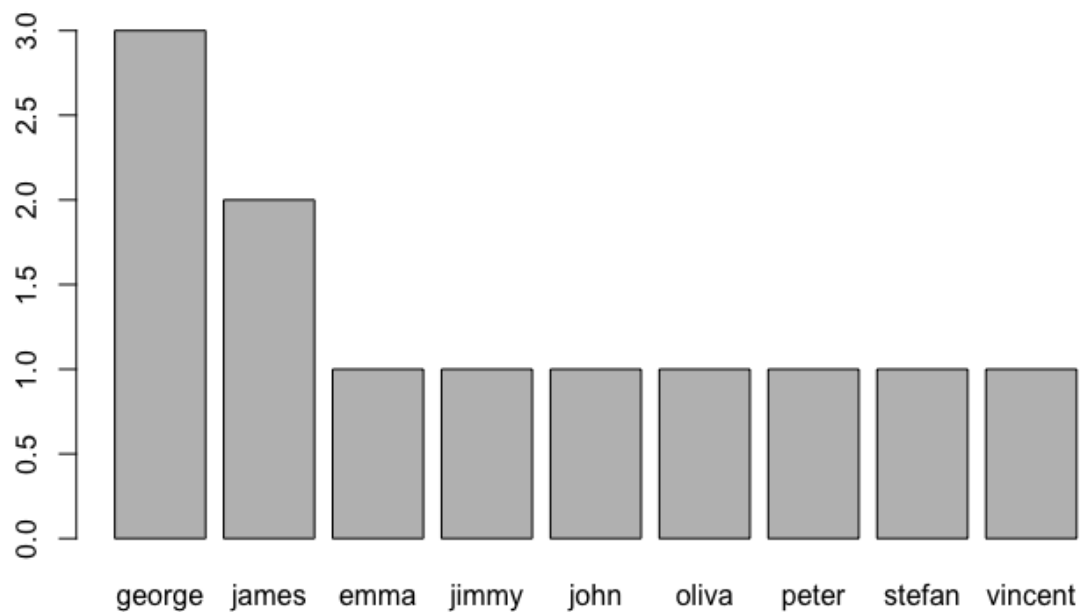


Figure 10.1: Name frequency

Then you will get Figure *Co-occurrence network*

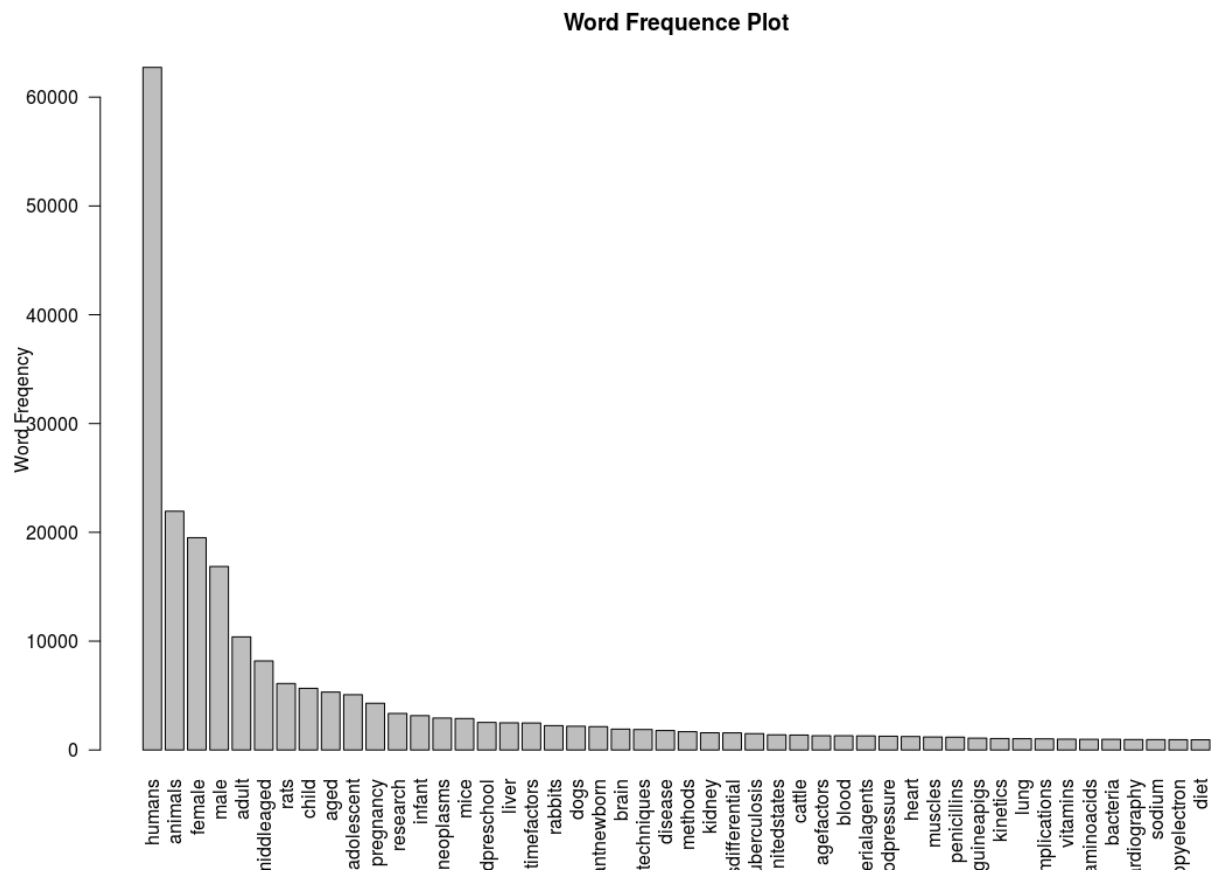
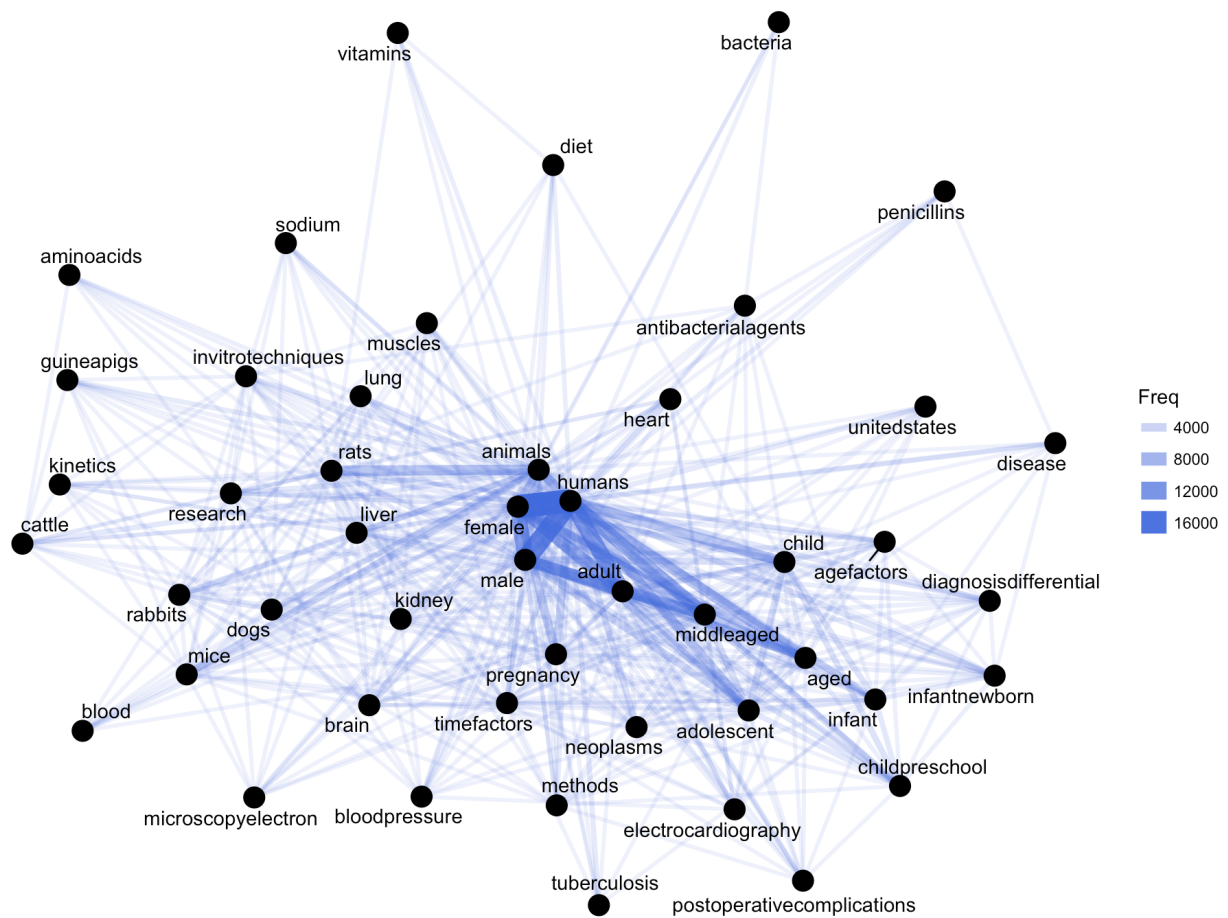


Figure 10.2: Word frequency



10.2 Correlation Network

NEURAL NETWORK

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

11.1 Feedforward Neural Network

11.1.1 Introduction

A feedforward neural network is an artificial neural network wherein connections between the units do not form a cycle. As such, it is different from recurrent neural networks.

The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward (see Fig. *MultiLayer Neural Network*), from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

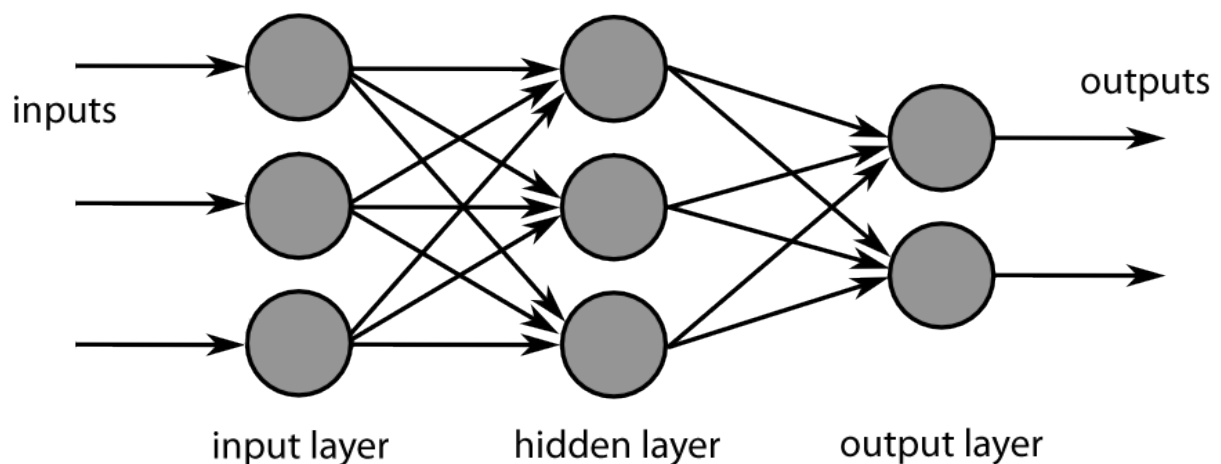


Figure 11.1: MultiLayer Neural Network

11.1.2 Demo

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Feedforward neural network example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Load dataset

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density|pH|sulphates|alcohol|quality|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| 9.4| 5|
| 7.8| 0.88| 0.0| 2.6| 0.098|25.0| 67.0| 0.9968| 3.2| 0.68| 9.8| 5|
| 7.8| 0.76| 0.04| 2.3| 0.092|15.0| 54.0| 0.997|3.26| 0.65| 9.8| 5|
| 11.2| 0.28| 0.56| 1.9| 0.075|17.0| 60.0| 0.998|3.16| 0.58| 9.8| 6|
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| 9.4| 5|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

3. change categorical variable size

```
# Convert to float format
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0<= r <= 4):
        label = "low"
    elif(4< r <= 6):
        label = "medium"
    else:
        label = "high"
    return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())
df= df.withColumn("quality", quality_udf("quality"))

4. Convert the data to dense vector

# convert the data to dense vector
def transData(data):
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
        toDF(['label', 'features'])
```

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors
```

```
data= transData(df)
data.show()
```

5. Split the data into training and test sets (40% held out for testing)

```
# Split the data into train and test
(trainingData, testData) = data.randomSplit([0.6, 0.4])
```

6. Train neural network

```
# specify layers for the neural network:
# input layer of size 11 (features), two intermediate of size 5 and 4
# and output of size 7 (classes)
layers = [11, 5, 4, 4, 3, 7]

# create the trainer and set its parameters
FNN = MultilayerPerceptronClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures",
                                     maxIter=100, layers=layers, blockSize=128, seed=1234)

# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and forest in a Pipeline
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, FNN, labelConverter])
# train the model
# Train model. This also runs the indexers.
model = pipeline.fit(trainingData)
```

7. Make predictions

```
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features", "label", "predictedLabel").show(5)
```

8. Evaluation

```
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Predictions accuracy = %g, Test Error = %g" % (accuracy, (1.0 - accuracy)))
```

**CHAPTER
TWELVE**

MAIN REFERENCE

BIBLIOGRAPHY

- [Bird2009] 19. Bird, E. Klein, and E. Loper. Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc., 2009.
- [Feng2017] 23. Feng and M. Chen. [Learning Apache Spark](#), Github 2017.
- [Karau2015] 8. Karau, A. Konwinski, P. Wendell and M. Zaharia. Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media, Inc., 2015

R

Run on Databricks Community Cloud, [9](#)

S

Set up Spark on Cloud, [15](#)

Set up Spark on Mac and Ubuntu, [14](#)