Learning Apache Spark with Python

Release v1.0

Wenqiang Feng

September 27, 2017

CONTENTS

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 Leanring and Deep Learning.

CHAPTER

ONE

PREFACE

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**
  - Data Scientist and Phd in Mathematics
  - University of Tennessee at Knoxville
  - Email: wfeng1@utk.edu

- Biography

Wenqiang Feng is Data Scientist within DST's Applied Analytics Group. Dr. Feng's responsibilities include providing DST clients with access to cutting-edge skills and technologies, including Big Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning algorithms, business intelligence, and applying Big Data tools to strategically solve industry problems in a cross-functional business. Before joining DST, Dr. Feng was an IMA Data Science Fellow at The Institute for Mathematics and its Applications (IMA) at the University of Minnesota. While there, he helped startup companies make marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville, with Ph.D. in Computational Mathematics and Master's degree in Statistics. He also holds Master's degree in Computational Mathematics from Missouri University of Science and Technology (MST) and Master's degree in Applied Mathematics from the University of Science and Technology of China (USTC).

- Declaration

The work of Wenqiang Feng was supported by the IMA, while working at IMA. However, any opinion, finding, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the IMA and DST.

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 disscussion 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.

CHAPTER

TWO

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 from the following two parts:

2.1 Why Spark?

I think the following four main reasons form 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.

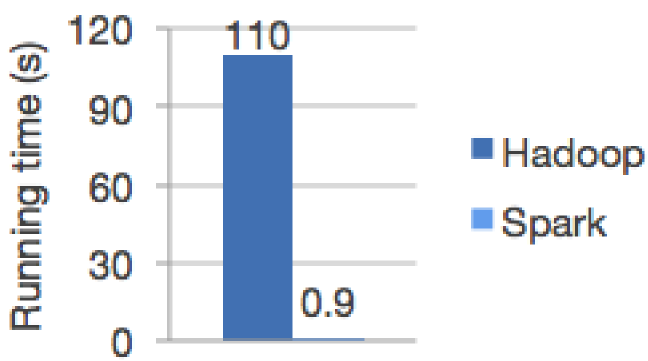
   

   Figure 2.1: Logistic regression in Hadoop and Spark

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.

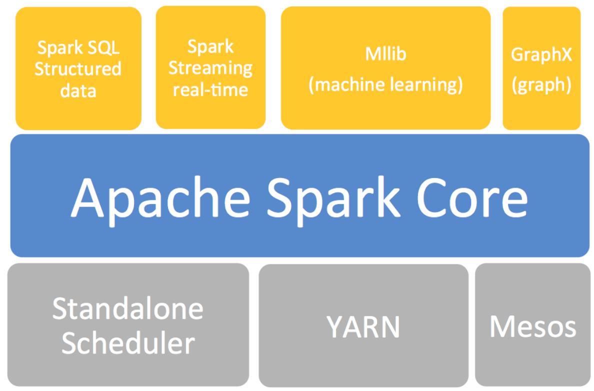


Figure 2.2: The Spark stack

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.

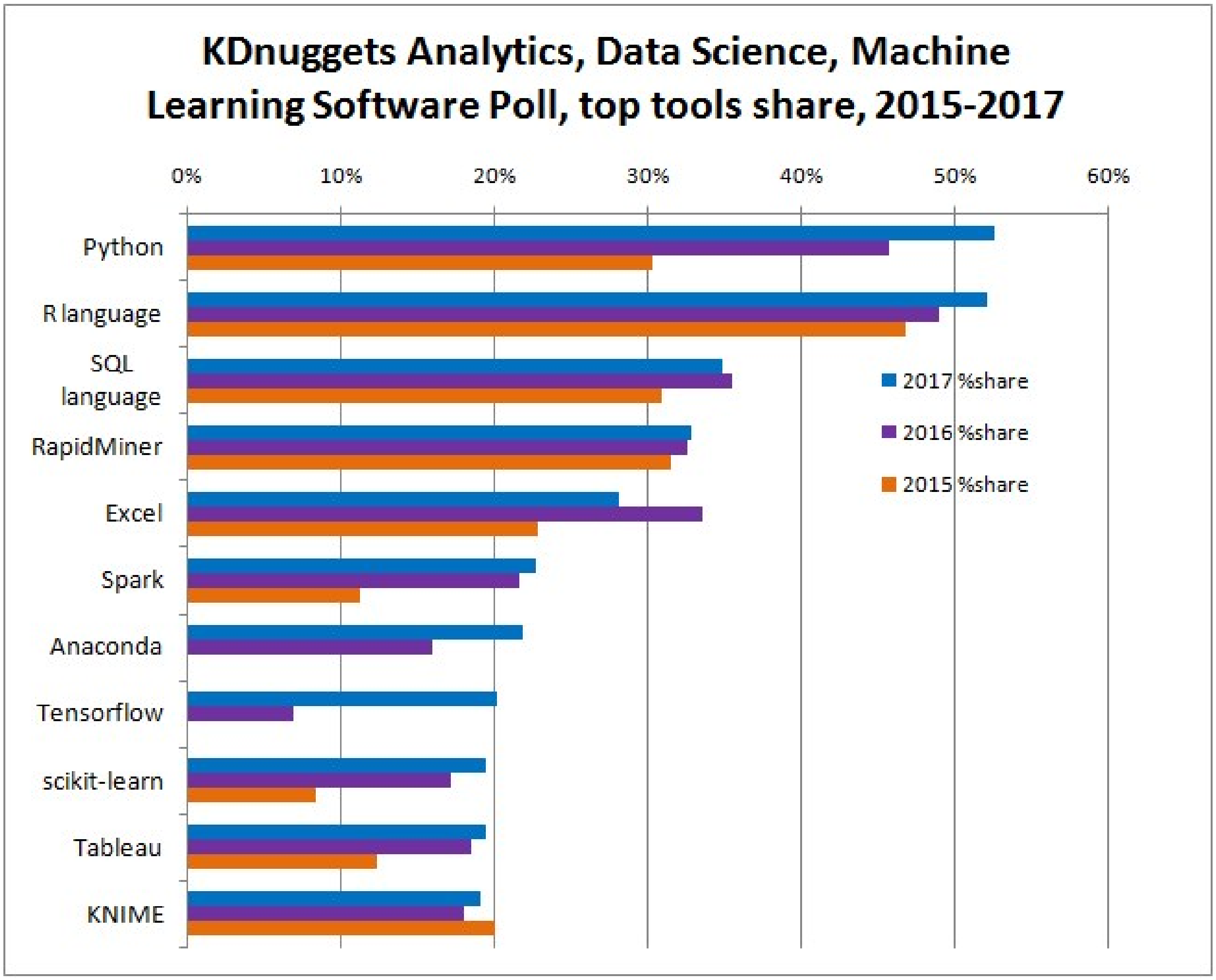


Figure 2.4: KDnuggets Analytics/Data Science 2017 Software Poll from kdnuggets.

CHAPTER

THREE

CONFIGURE RUNNING PLATFORM

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

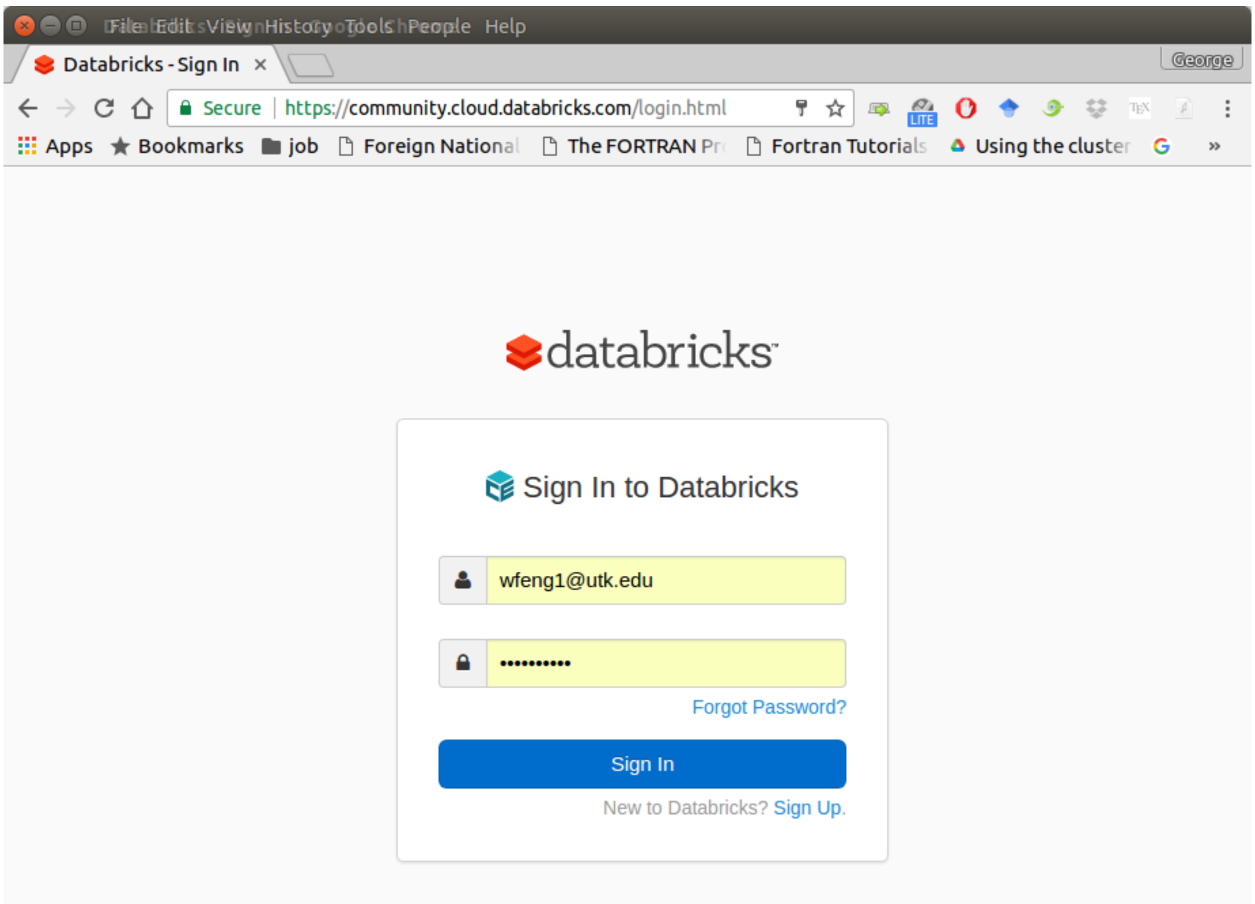
2. Sign in with your account, then you can creat your cluster(machine), table(dataset) and notebook(code).

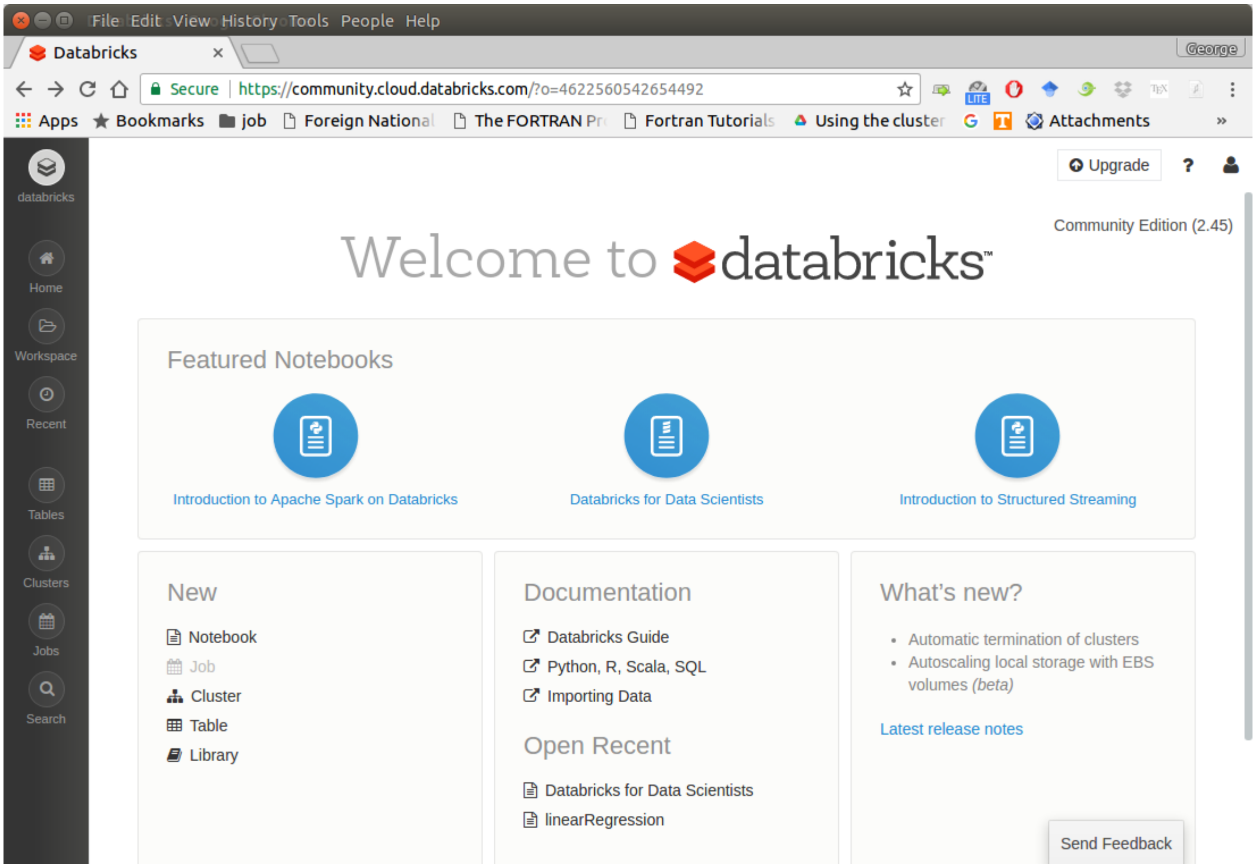
3. Create your cluster where your code will run

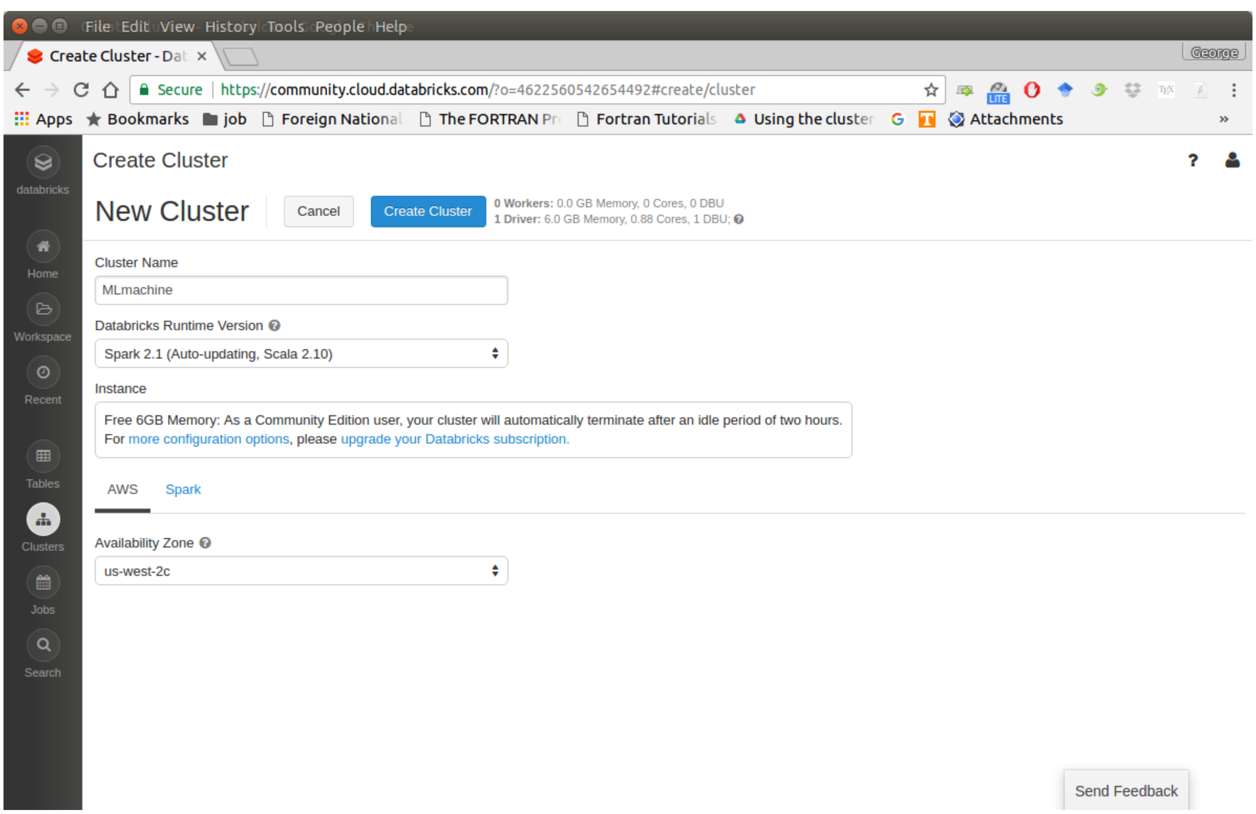
4. Import your dataset

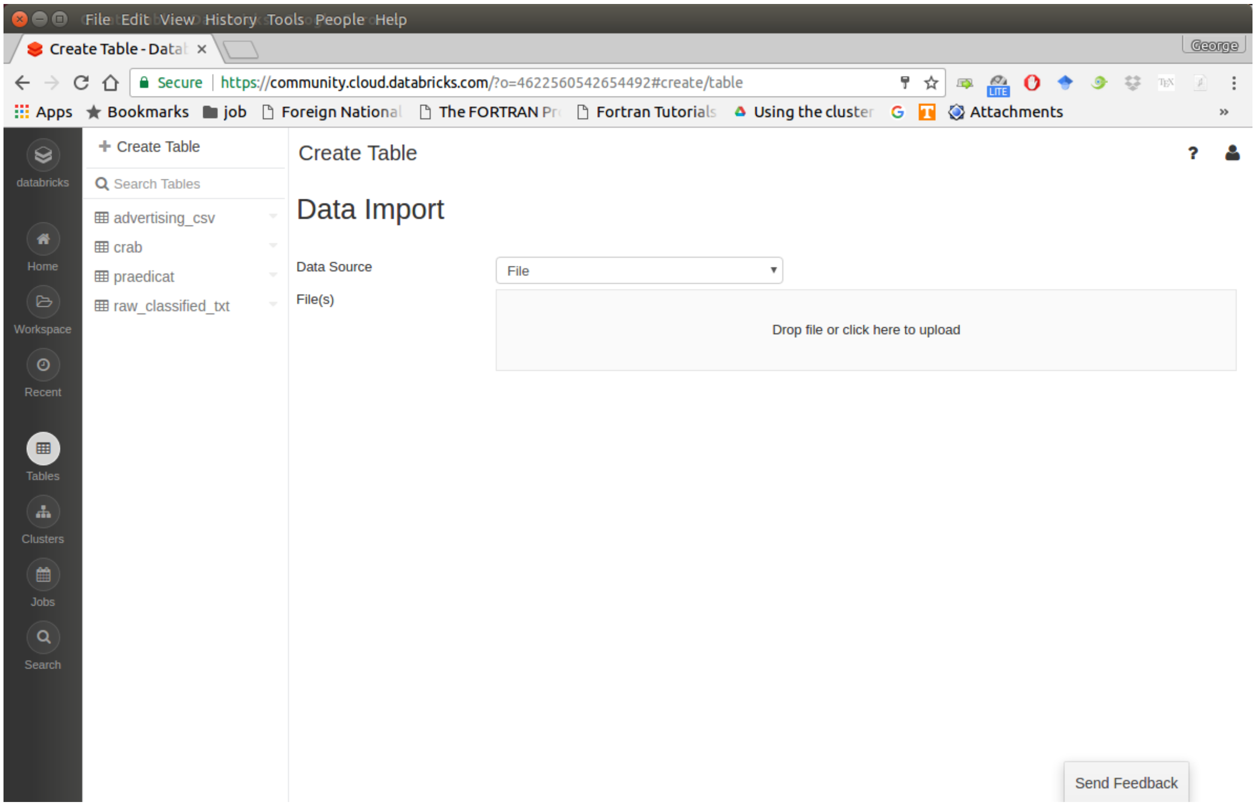
Note: You need to save the path which appears at Uploaded to DBFS: /FileStore/tables/05rmhuqv1489687378010/. Since we will use this path to load the dataset.

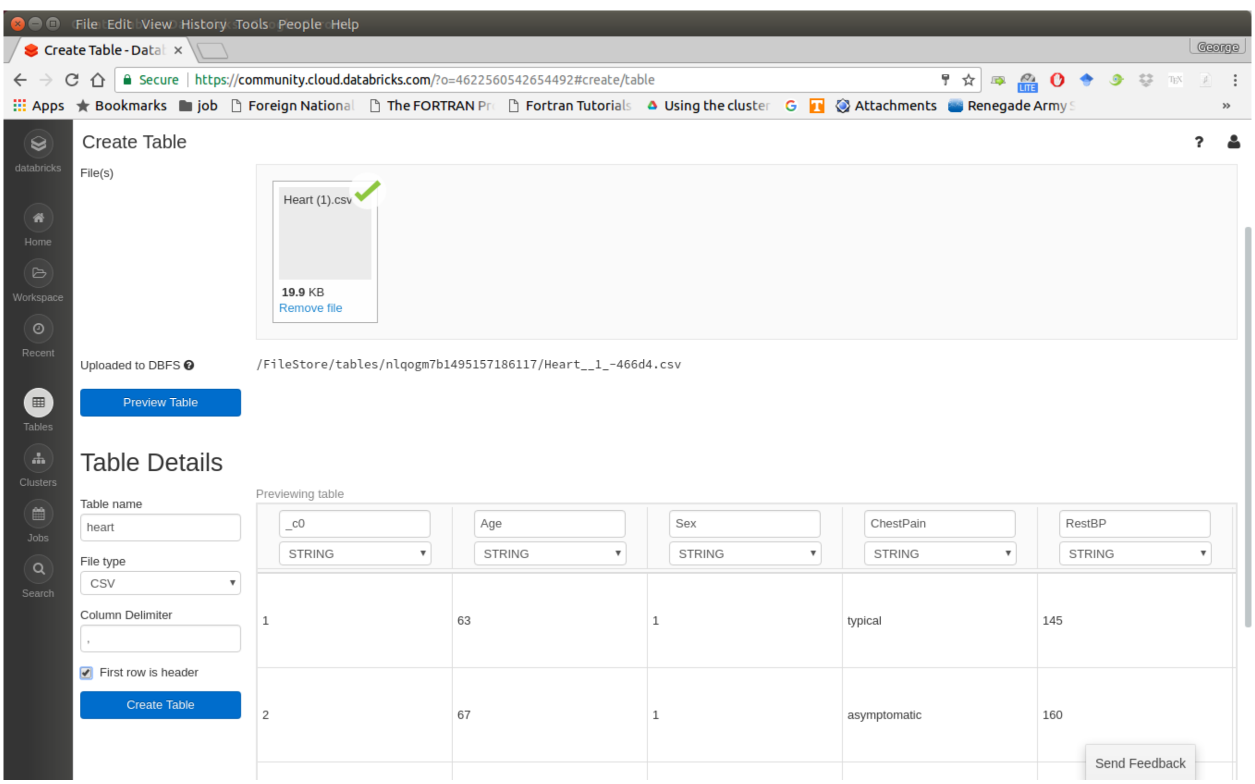
5. Creat your notebook

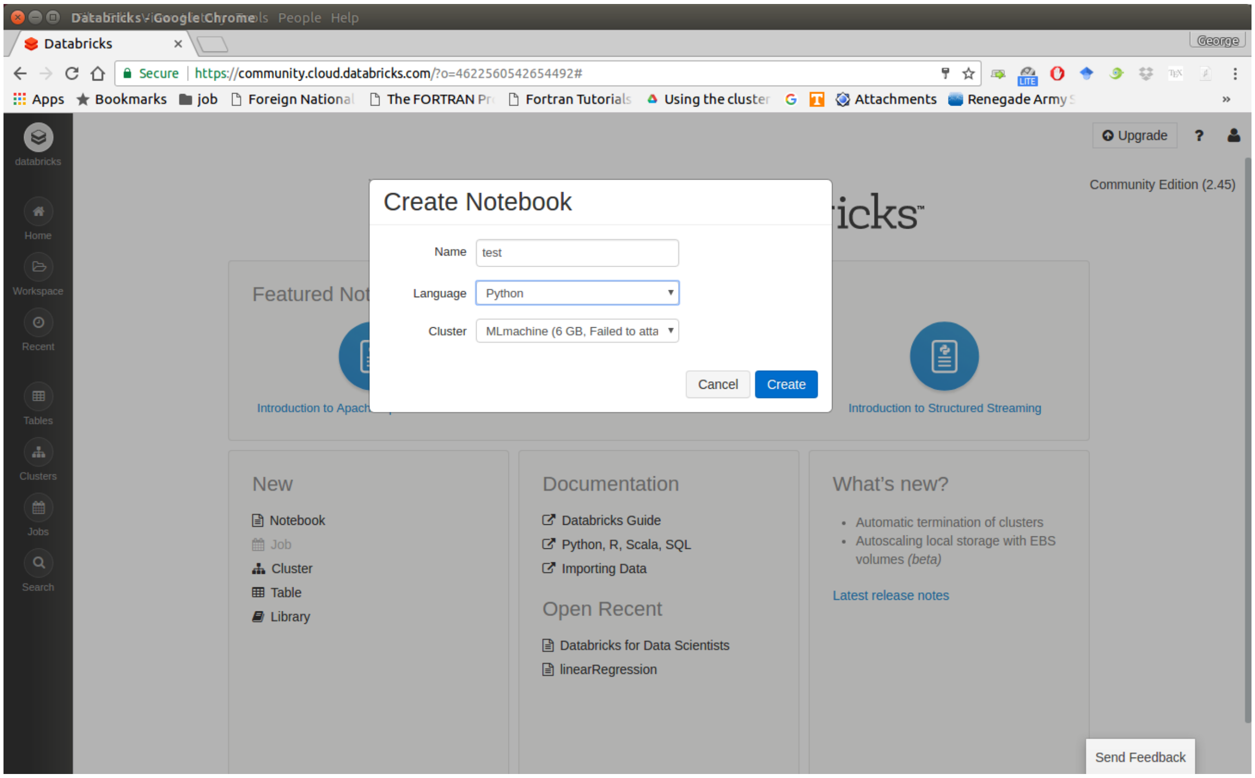

File Edit View History Tools People Help
Databricks - Sign In
George
Secure https://community.cloud.databricks.com/login.html
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster
databricks
Sign In to Databricks
wfeng1@utk.edu
Forgot Password?
Sign In
New to Databricks? Sign Up.

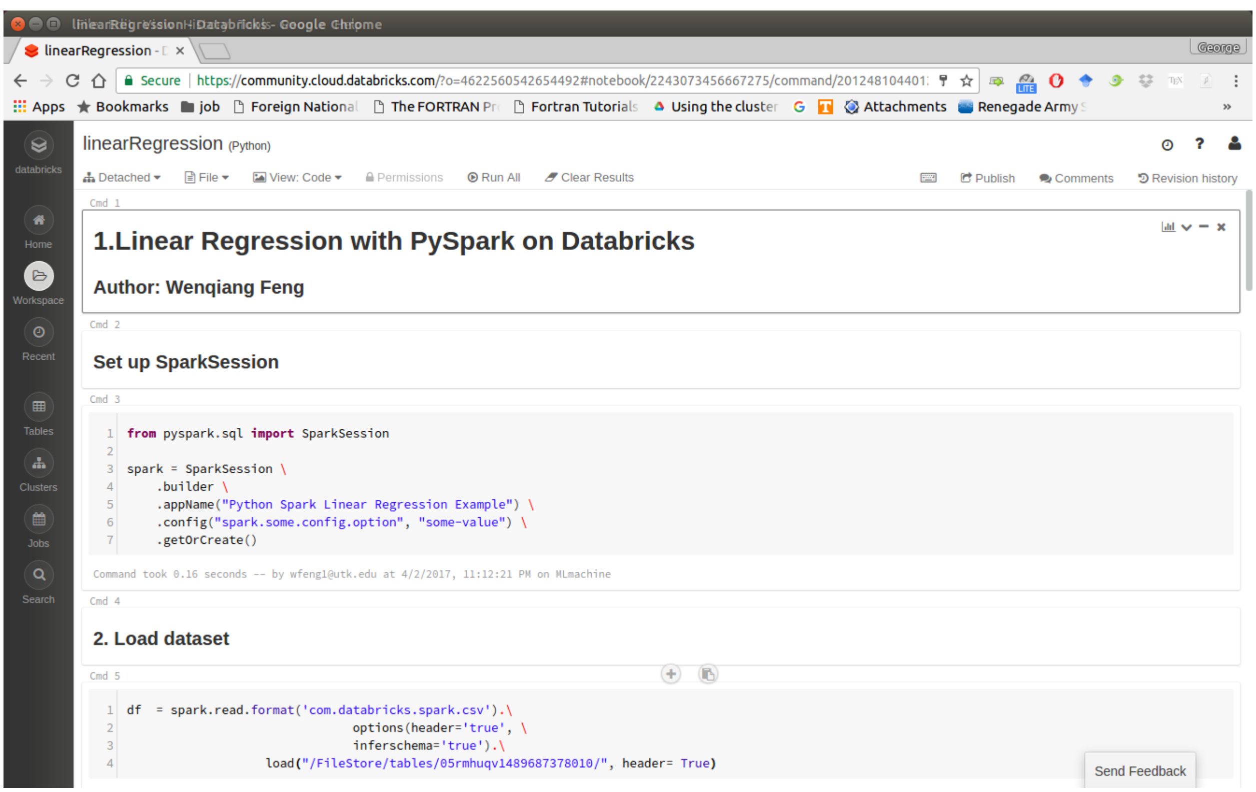

Welcome to databricks
Community Edition (2.45)
Featured Notebooks
Introduction to Apache Spark on Databricks
Databricks for Data Scientists
Introduction to Structured Streaming
New
Notebook
Job
Cluster
Table
Library
Documentation
Databricks Guide
Python, R, Scala, SQL
Importing Data
Open Recent
Databricks for Data Scientists
linearRegression
What's new?
Automatic termination of clusters
Autoscaling local storage with EBS volumes (beta)
Latest release notes
Send Feedback


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


Create Table
Data Import
Data Source
File
File(s)
Drop file or click here to upload
Send Feedback


Create Table
File(s)
Heart (1).csv
19.9 KB
Remove file
Uploaded to DBFS
/FileStore/tables/nlqogm7b1495157186117/Heart__1_-466d4.csv
Preview Table
Table Details
Table name
heart
File type
CSV
Column Delimiter
First row is header
Create Table
Previewing table
Send Feedback


Databricks - Google Chrome
Databricks
Secure https://community.cloud.databricks.com/?o=4622560542654492#
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments Renegade Army
Upgrade
Community Edition (2.45)
Create Notebook
Name test
Language Python
Cluster MLmachine (6 GB, Failed to atta
Cancel Create
Featured Not
Introduction to Apach
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
databricks
Home
Workspace
Recent
Tables
Clusters
Jobs
Search


linearRegression - Databricks - Google Chrome
linearRegression
Secure https://community.cloud.databricks.com/?o=4622560542654492#notebook/2243073456667275/command/201248104401
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments Renegade Army
linearRegression (Python)
Detached File View: Code Permissions Run All Clear Results
Publish Comments Revision history
Cmd 1
1.Linear Regression with PySpark on Databricks
Author: Wenqiang Feng
Cmd 2
Set up SparkSession
Cmd 3
from pyspark.sql import SparkSession
spark = SparkSession \
.builder \
.appName("Python Spark Linear Regression Example") \
.config("spark.some.config.option", "some-value") \
.getOrCreate()
Command took 0.16 seconds -- by wfeng1@utk.edu at 4/2/2017, 11:12:21 PM on MLmachine
Cmd 4
2. Load dataset
Cmd 5
df = spark.read.format('com.databricks.spark.csv').\
options(header='true', \
inferschema='true').\
load("/FileStore/tables/05rmhuqv1489687378010/", header= True)
Send Feedback
databricks
Home
Workspace
Recent
Tables
Clusters
Jobs
Search

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 Configure Spark on Mac and Ubuntu

3.2.1 Installing Prerequisites

I will strongly recommend you to install Anaconda, since it contains most of the prerequisites and support multiple Operator Systems.

1. **Install 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
```

3.2.2 Install Java

Java is used by many other softwares. So it is quite possible that you have already installed it. You can by using the following command in Command Prompt:

```
java -version
```

Otherwise, you can follow the steps in How do I install Java for my Mac? to install java on Mac and use the following command in Command Prompt to install on Ubuntu:

```
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

3.2.3 Install Java SE Runtime Environment

I installed ORACLE Java JDK.

> **Warning: Installing Java and Java SE Runtime Environment steps are very important, since Spark is a domain-specific language written in Java.**

You can check if your Java is available and find it's version by using the following command in Command Prompt:

```
java -version
```

If your Java is installed successfully, you will get the similar results as follows:

```
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

3.2.4 Install Apache Spark

Actually, the Pre-build version doesn't need installation. You can use it when you unpack it.

1. Download: You can get the Pre-built Apache Spark™ from Download Apache Spark™.
2. Unpack: Unpack the Apache Spark™ to the path where you want to install the Spark.
3. Test: Test the Prerequisites: change the direction `spark-#.#.#-bin-hadoop#.#/bin` and run

```
./pyspark
```

```
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default, Dec 20 2016, 23:05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR,
use setLogLevel(newLevel).
17/08/30 13:30:12 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
17/08/30 13:30:17 WARN ObjectStore: Failed to get database global_temp,
returning NoSuchObjectException
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Python version 2.7.13 (default, Dec 20 2016 23:05:08)
SparkSession available as 'spark'.
```

3.2.5 Configure the Spark

1. **Mac Operator System:** open your `bash_profile` in Terminal

```
vim ~/.bash_profile
```

And add the following lines to your `bash_profile` (remember to change the path)

```
# add for spark
export SPARK_HOME=your_spark_installation_path
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_DRIVE_PYTHON="jupyter"
export PYSPARK_DRIVE_PYTHON_OPTS="notebook"
```

At last, remember to source your `bash_profile`

```
source ~/.bash_profile
```

2. **Ubuntu Operator Sysytem:** open your `bashrc` in Terminal

```
vim ~/.bashrc
```

And add the following lines to your `bashrc` (remember to change the path)

```
# add for spark
export SPARK_HOME=your_spark_installation_path
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_DRIVE_PYTHON="jupyter"
export PYSPARK_DRIVE_PYTHON_OPTS="notebook"
```

At last, remember to source your `bashrc`

```
source ~/.bashrc
```

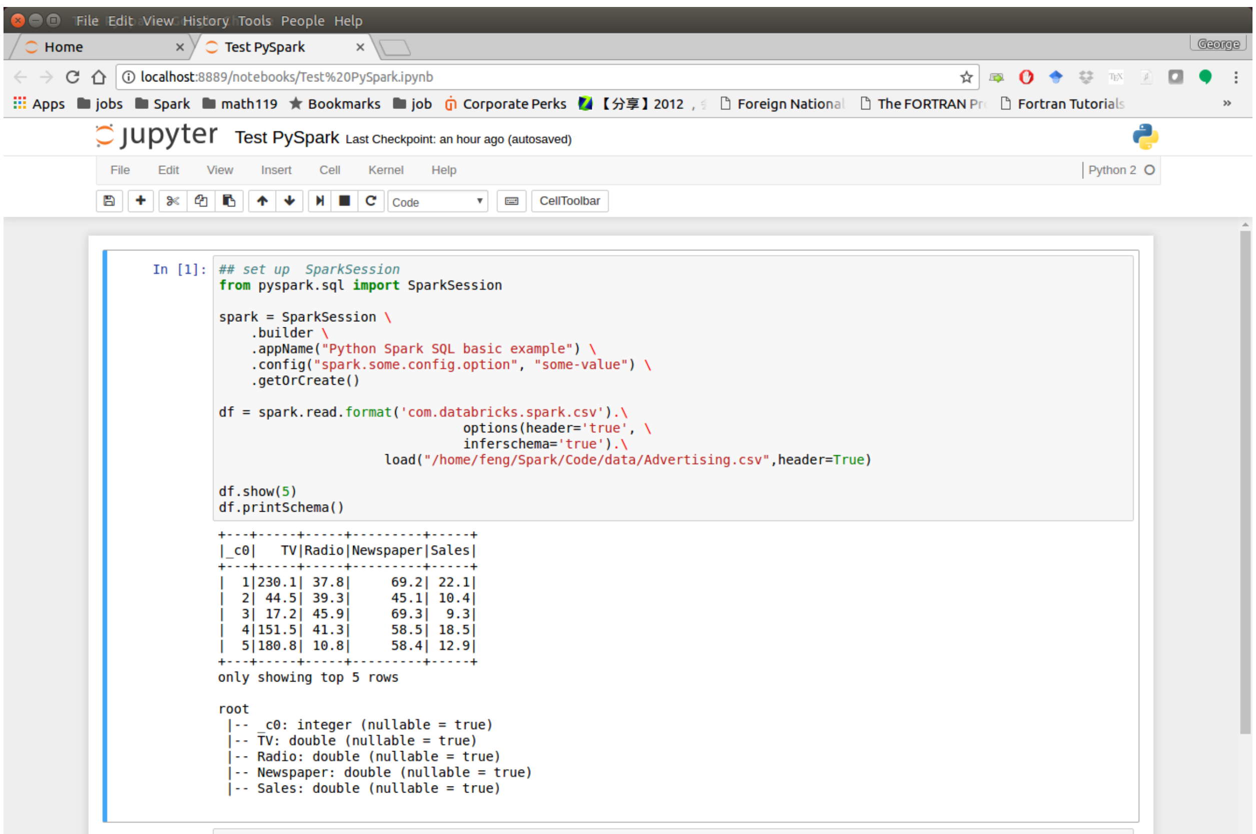
3.3 Configure Spark on Windows

Installing open source software on Windows is always a nightmare for me. Thanks for Deelesh Mandloi. You can follow the detailed procedures in the blog Getting Started with PySpark on Windows to install the Apache Spark™ on your Windows Operator System.

3.4 PySpark With Text Editor or IDE

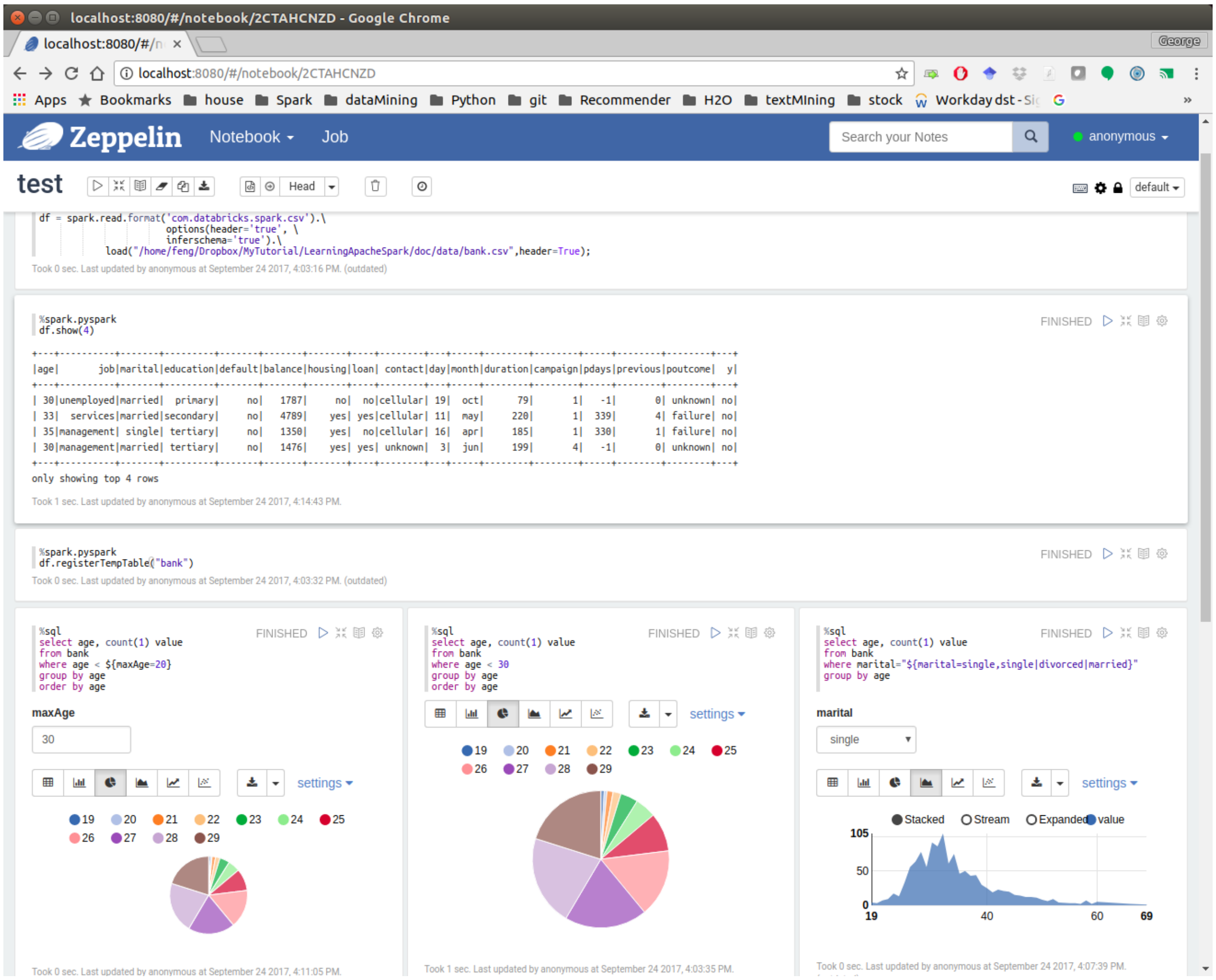
3.4.1 PySpark With Jupyter Notebook

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



3.4.2 PySpark With Apache Zeppelin

After you finishing the above setup steps in *Configure Spark on Mac and Ubuntu*, then you should be good to write and run your PySpark Code in Apache Zeppelin.

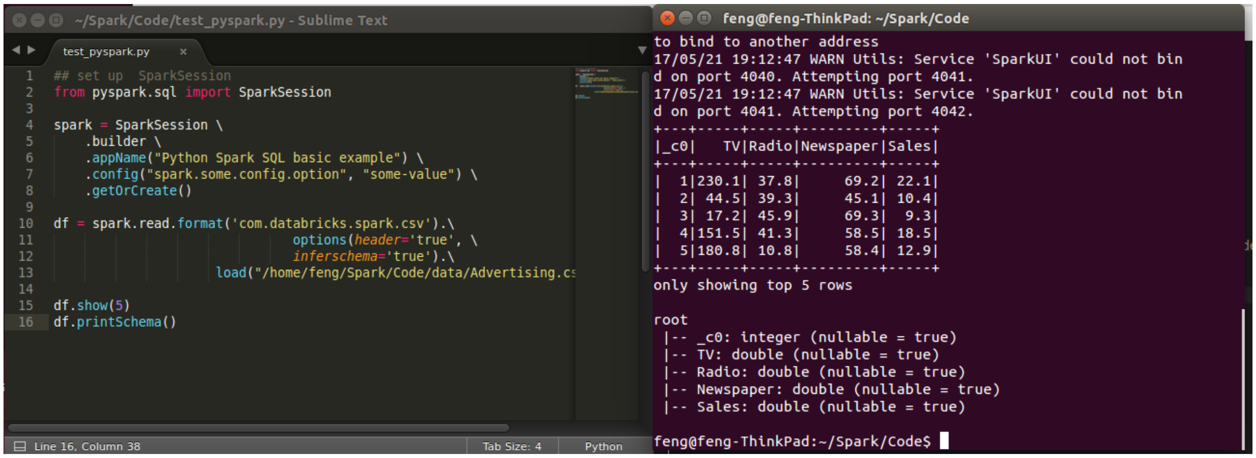

localhost:8080/#/notebook/2CTAHCNZD - Google Chrome
Zeppelin
Notebook
Job
Search your Notes
anonymous
test
Head
default
df = spark.read.format('com.databricks.spark.csv').\
options(header='true', \
inferschema='true').\
load("/home/feng/Dropbox/MyTutorial/LearningApacheSpark/doc/data/bank.csv",header=True);
%spark.pyspark
df.show(4)
FINISHED
only showing top 4 rows
%spark.pyspark
df.registerTempTable("bank")
%sql
select age, count(1) value
from bank
where age < ${maxAge=20}
group by age
order by age
maxAge
30
settings
where age < 30
where marital="${marital=single,single|divorced|married}"
marital
single
Stacked
Stream
Expanded
value

3.4.3 PySpark With Sublime Text

After you finishing the above setup steps in *Configure 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 output results in your terminal.



3.4.4 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:

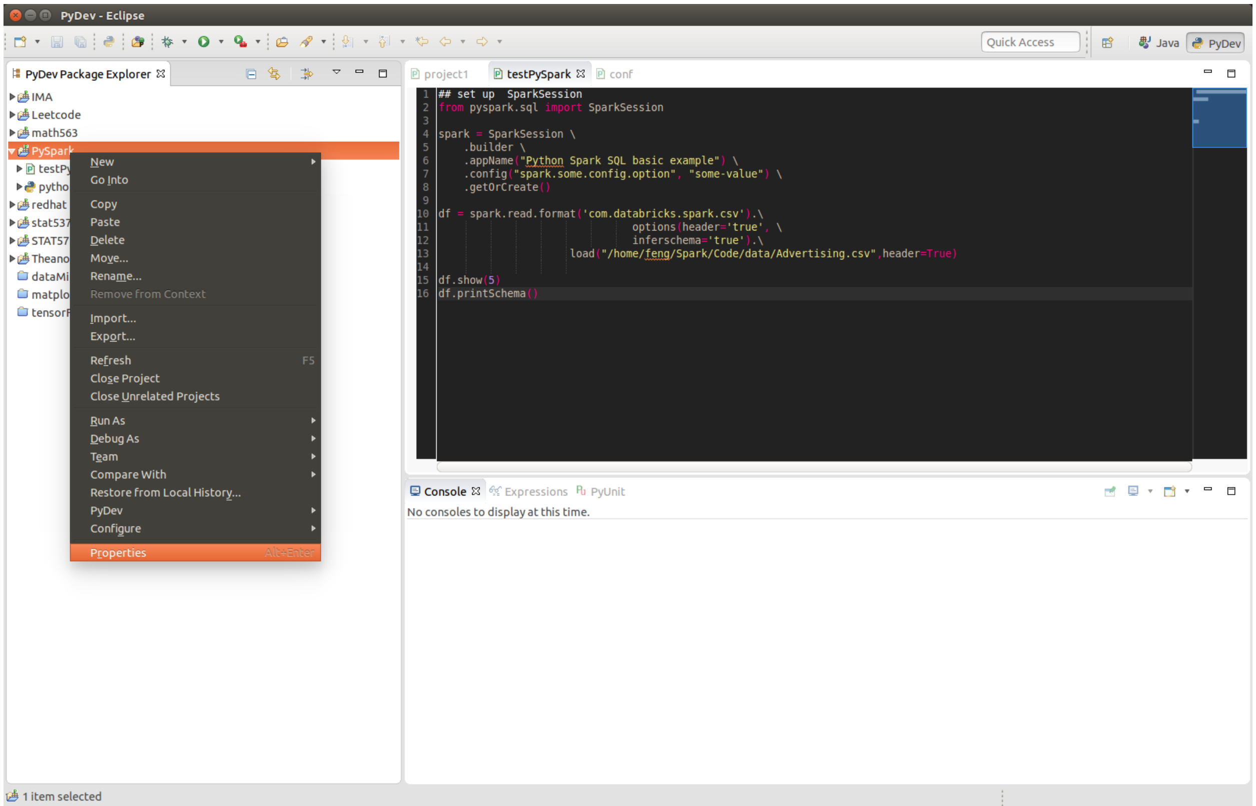
1. Open the properties of your project

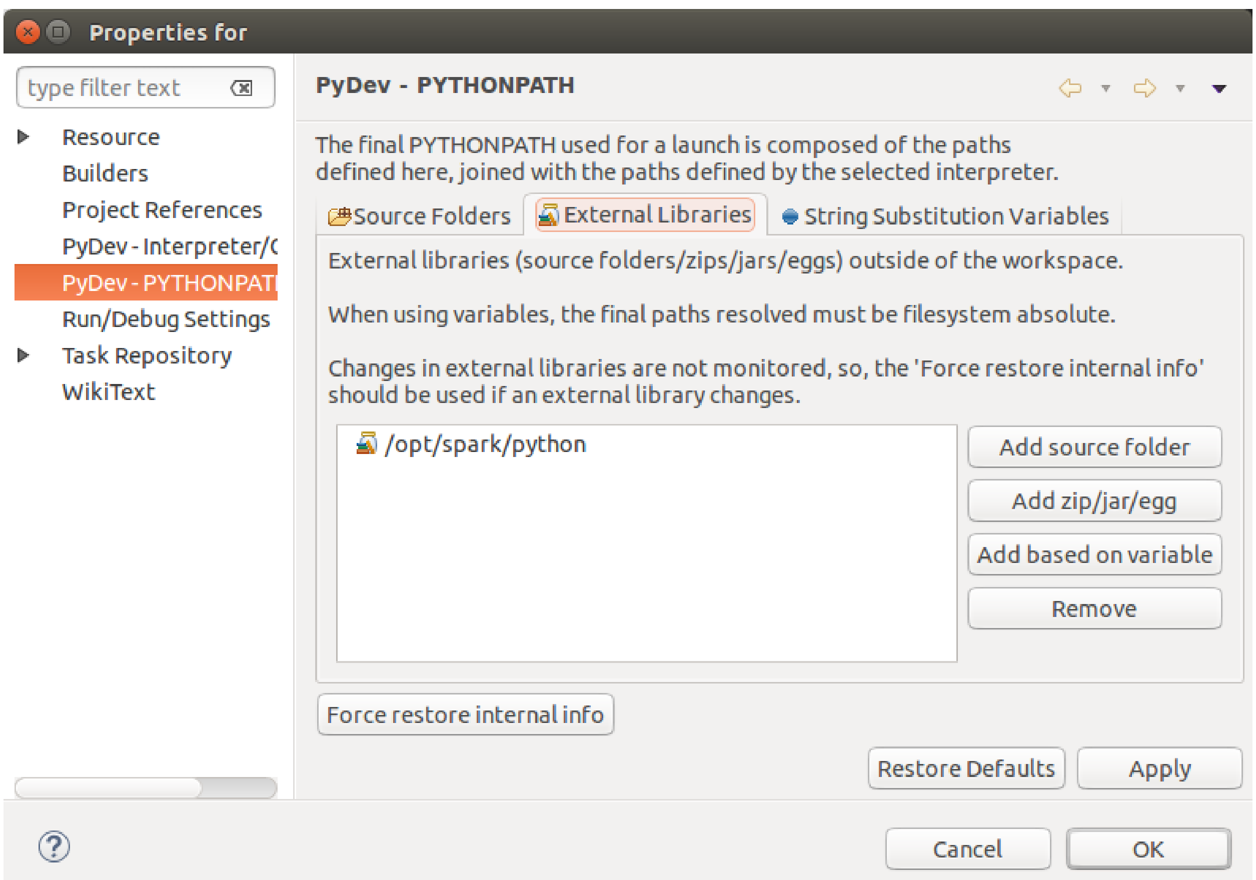
2. Add the paths for the **External Libraries**

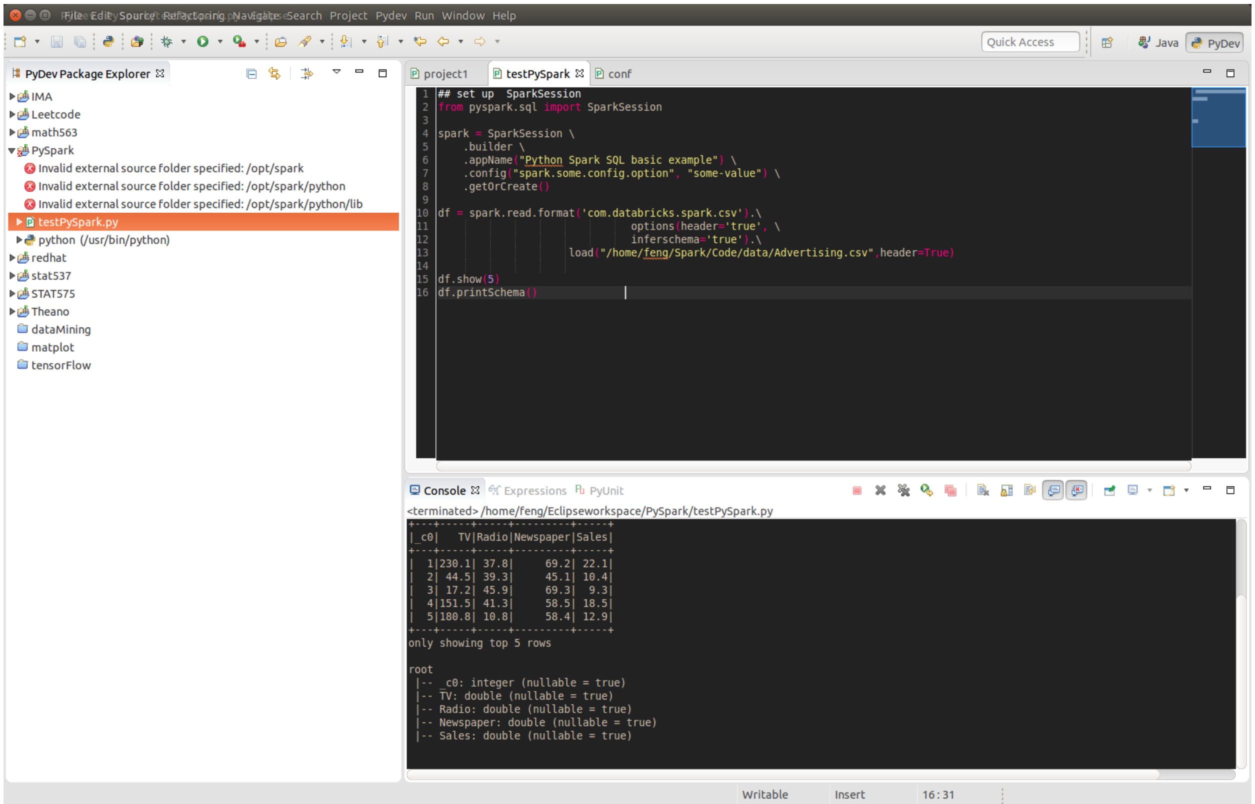
And then you should be good to run your code on Eclipse with PyDev.

3.5 Set up Spark on Cloud

Folloing the setup steps in *Configure 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 whitout any setting just like Databricks Community Cloud. If you want more details, please feel free to contact with me.


PyDev - Eclipse
PyDev Package Explorer
testPySpark
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()
New
Go Into
Copy
Paste
Delete
Move...
Rename...
Remove from Context
Import...
Export...
Refresh
Close Project
Close Unrelated Projects
Run As
Debug As
Team
Compare With
Restore from Local History...
PyDev
Configure
Properties
Console
No consoles to display at this time.
1 item selected


Properties for
type filter text
Resource
Builders
Project References
PyDev - Interpreter/C
PyDev - PYTHONPATH
Run/Debug Settings
Task Repository
WikiText
PyDev - PYTHONPATH
The final PYTHONPATH used for a launch is composed of the paths
defined here, joined with the paths defined by the selected interpreter.
Source Folders
External Libraries
String Substitution Variables
External libraries (source folders/zips/jars/eggs) outside of the workspace.
When using variables, the final paths resolved must be filesystem absolute.
Changes in external libraries are not monitored, so, the 'Force restore internal info'
should be used if an external library changes.
/opt/spark/python
Add source folder
Add zip/jar/egg
Add based on variable
Remove
Force restore internal info
Restore Defaults
Apply
Cancel
OK



3.6 Demo Code in this Section

The code for this section is available for download test_pyspark, and the Jupyter notebook can be download from test_pyspark_ipynb.

- 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()
```

CHAPTER

FOUR

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

Most of the following content comes from [Kirillov2016]. So the copyright belongs to **Anton Kirillov**. I will refer you to get more details from Apache Spark core concepts, architecture and internals.

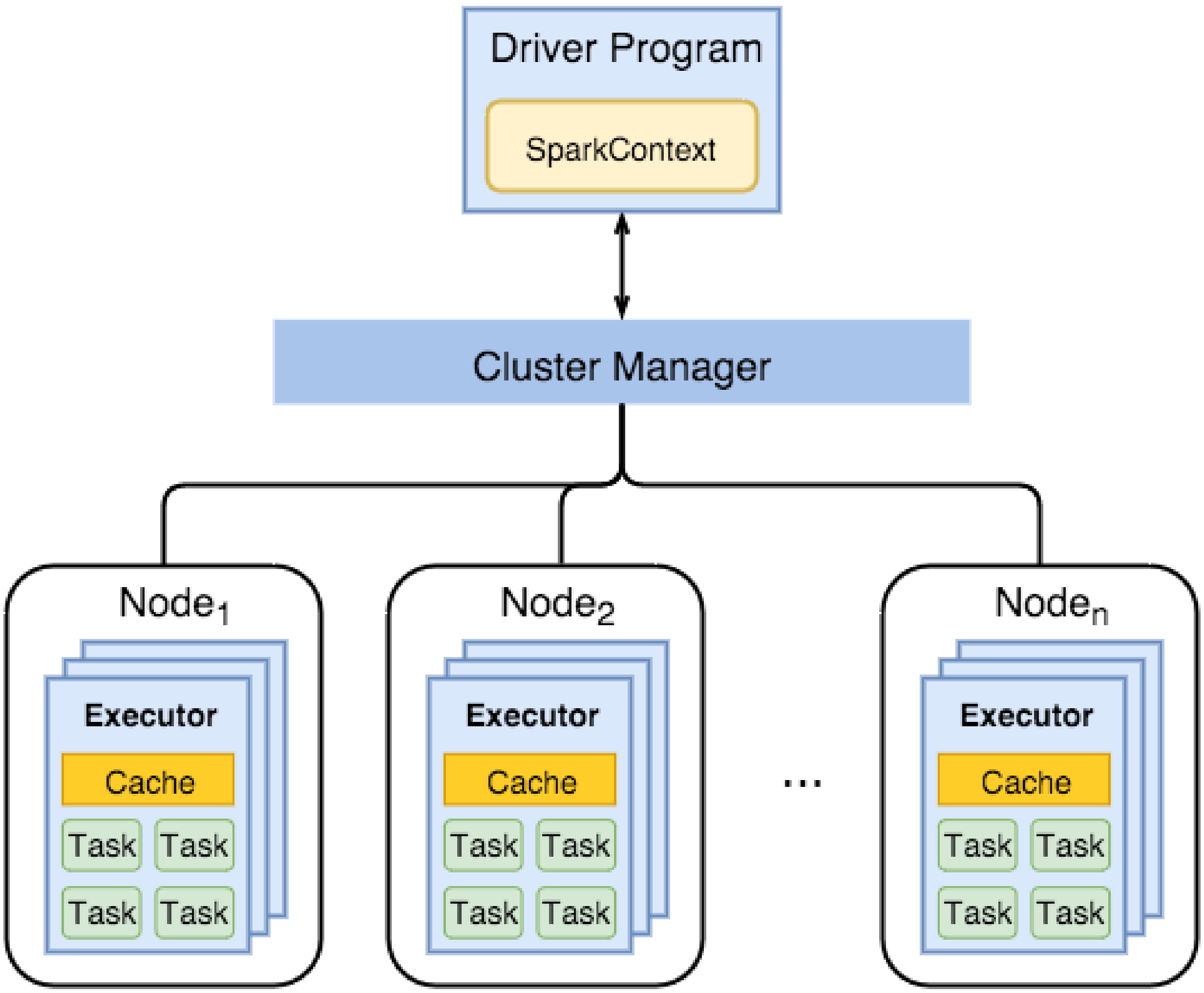
Before diving deep into how Apache Spark works, lets understand the jargon of Apache Spark

- 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 (Its 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 its 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


Driver Program
SparkContext
Cluster Manager
Node1
Executor
Cache
Task
Task
Task
Task
Node2
Executor
Cache
Task
Task
Task
Task
...
Noden
Executor
Cache
Task
Task
Task
Task

- creates SparkContext to schedule jobs execution and negotiate with cluster manager

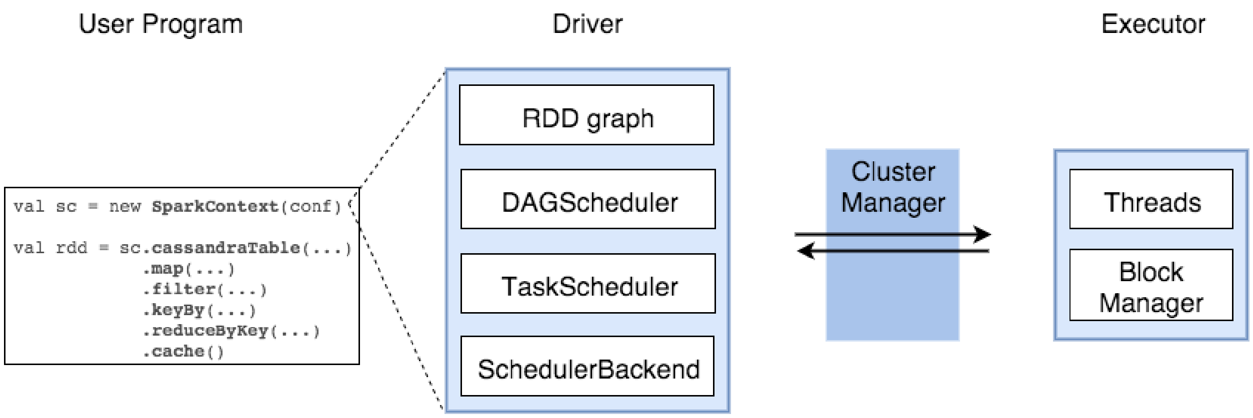
2. Executors

- 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)

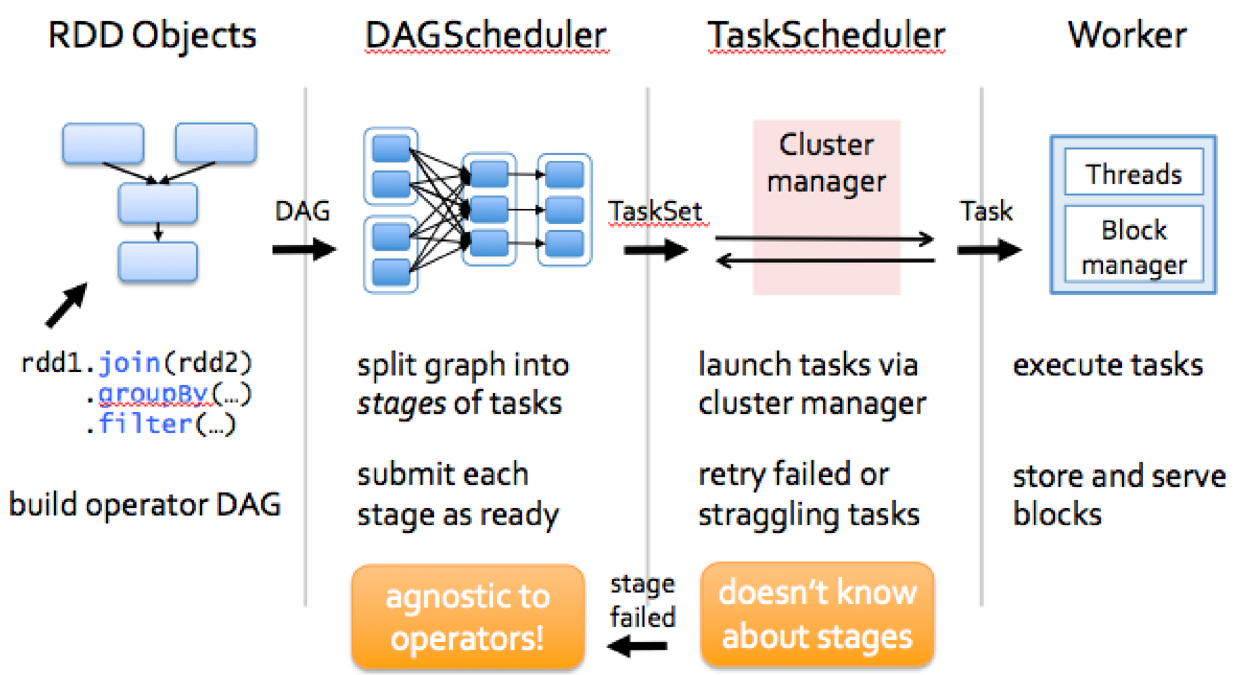
- BlockManager
  - provides interfaces for putting and retrieving blocks both locally and remotely into various stores (memory, disk, and off-heap)

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.



CHAPTER

FIVE

PROGRAMMING WITH RDDS

Note: If you only know yourself, but not your opponent, you may win or may lose. If you know neither yourself nor your enemy, you will always endanger yourself – idiom, from Sunzi's Art of War

RDD represents **Resilient Distributed Dataset**. An RDD in Spark is simply an immutable distributed collection of objects sets. Each RDD is split into multiple partitions (similar pattern with smaller sets), which may be computed on different nodes of the cluster.

5.1 Create RDD

Usually, there are two popular way to create the RDDs: loading an external dataset, or distributing a set of collection of objects. The following examples show some simplest ways to create RDDs by using `parallelize()` fucntion which takes an already existing collection in your program and pass the same to the Spark Context.

1. By using `parallelize( )` fucntion

```
from pyspark.sql import SparkSession

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

df = spark.sparkContext.parallelize([(1, 2, 3, 'a b c'),
             (4, 5, 6, 'd e f'),
             (7, 8, 9, 'g h i')]).toDF(['col1', 'col2', 'col3','col4'])
```

Then you will get the RDD data:

```
df.show()

+----+----+----+-----+
|col1|col2|col3| col4|
+----+----+----+-----+
|   1|   2|   3|a b c|
|   4|   5|   6|d e f|
```

```
|   7|   8|   9|g h i|
+----+----+----+-----+
```

```
from pyspark.sql import SparkSession

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

myData = spark.sparkContext.parallelize([(1,2), (3,4), (5,6), (7,8), (9,10)])
```

Then you will get the RDD data:

```
myData.collect()

[(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)]
```

2. By using `createDataFrame( )` function

```
from pyspark.sql import SparkSession

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

Employee = spark.createDataFrame([
                        ('1', 'Joe',   '70000', '1'),
                        ('2', 'Henry', '80000', '2'),
                        ('3', 'Sam',   '60000', '2'),
                        ('4', 'Max',   '90000', '1')],
                        ['Id', 'Name', 'Sallary','DepartmentId']
                       )
```

Then you will get the RDD data:

```
+---+-----+-------+------------+
| Id| Name|Sallary|DepartmentId|
+---+-----+-------+------------+
|  1|  Joe|  70000|           1|
|  2|Henry|  80000|           2|
|  3|  Sam|  60000|           2|
|  4|  Max|  90000|           1|
+---+-----+-------+------------+
```

3. By using `read` and `load` functions

    1. **Read dataset from .csv file**

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

```
spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD 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()
```

Then you will get the RDD data:

```
+---+-----+-----+---------+-----+
|_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)
```

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

2. **Read dataset from DataBase**

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

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

## User information
user = 'your_username'
pw   = 'your_password'

## Database information
table_name = 'table_name'
url = 'jdbc:postgresql://##.###.###.##:5432/dataset?user='+user+'&password='+pw
```

```
properties ={'driver': 'org.postgresql.Driver', 'password': pw,'user': user}

df = spark.read.jdbc(url=url, table=table_name, properties=properties)

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

Then you will get the RDD data:

```
+---+-----+-----+---------+-----+
|_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)
```

Note: Reading tables from Database needs the proper drive for the corresponding Database. For example, the above demo needs `org.postgresql.Driver` and **you need to download it and put it in ''jars'' folder of your spark installation path.** I download `postgresql-42.1.1.jar` from the official website and put it in `jars` folder.

5.2 Spark Transformations

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

5.3 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).

CHAPTER

SIX

STATISTICS PRELIMINARY

Note: If you only know yourself, but not your opponent, you may win or may lose. If you know neither yourself nor your enemy, you will always endanger yourself – idiom, from Sunzi's Art of War

6.1 Notations

- m : the number of the samples
- n : the number of the features
- y_i : i-th label
- $\bar{y}$: the mean of y.

6.2 Measurement Formula

- Mean squared error

In statistics, the **MSE** (Mean Squared Error) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated.

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

- Root Mean squared error

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}$$

- Total sum of squares

In statistical data analysis the **TSS** (Total Sum of Squares) is a quantity that appears as part of a standard way of presenting results of such analyses. It is defined as being the sum, over all observations, of the squared differences of each observation from the overall mean.

$$\text{TSS} = \sum_{i=1}^{m} (y_i - \bar{y})^2$$

CHAPTER

SEVEN

REGRESSION

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

In statistical modeling, regression analysis focuses on investigating the relationship between a dependent variable and one or more independent variables. Wikipedia Regression analysis

In data mining, Regression is a model to represent the relationship between the value of lable (or target, it is numerical variable) and on one or more features (or predictors they can be numerical and categorical variables).

7.1 Linear Regression

7.1.1 Introduction

Given that a data set $\{\, x_{i1},\ldots,x_{in},y_{i}\}_{i=1}^{m}$ which contains n features (variables) and m samples (data points), in simple linear regression model for modeling m data points with one independent variable: x_{i1}, the formula is given by:

$$y_i = \beta_0 + \beta_1 x_{i1}, \text{where}, i = 1, \cdots m.$$

In matrix notation, the data set is written as $\mathbf{X} = [\mathbf{X}_1, \cdots, \mathbf{X}_n]$ with $\mathbf{X}_i = \{x_{\cdot i}\}_{i=1}^{n}$, $\mathbf{y} = \{y_i\}_{i=1}^{m}$ and $\boldsymbol{\beta}^\top = \{\beta_i\}_{i=1}^{m}$. Then the normal equations are written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}.$$

7.1.2 How to solve it?

1. Direct Methods
2. Iterative Methods

7.1.3 Demo

- The Jupyter notebook can be download from Linear Regression which was implemented without using Pipeline.
- The Jupyter notebook can be download from Linear Regression with Pipeline which was implemented with using Pipeline.
- I will only present the code with pipeline style in the following.
- For more details about the parameters, please visit Linear Regression API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

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

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                       options(header='true', \
                       inferschema='true').\
            load("../data/Advertising.csv",header=True);
```

check the data set

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

Then you will get

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

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

You can also get the Statistical resutls from the data frame (Unfortunately, it only works for numerical).

```
df.describe().show()
```

Then you will get

```
+-------+-----------------+------------------+------------------+------------------+
|summary|               TV|             Radio|         Newspaper|             Sales|
+-------+-----------------+------------------+------------------+------------------+
|  count|              200|               200|               200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|    min|              0.7|               0.0|               0.3|               1.6|
|    max|            296.4|              49.6|             114.0|              27.0|
+-------+-----------------+------------------+------------------+------------------+
```

Figure 7.1: Sales distribution

3. Convert the data to dense vector (**features** and **label**)

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

# I provide two ways to build the features and labels

# method 1 (good for small feature):
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))

# Method 2 (good for large features):
```

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

4. Transform the dataset to DataFrame

```
transformed= transData(df)
transformed.show(5)

+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

Note: You will find out that all of the machine learning algorithms in Spark are based on the **features** and **label**. That is to say, you can play with all of the machine learning algorithms in Spark when you get ready the **features** and **label**.

---

5. Deal With Categorical Variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4 distinct values are treated as continuous.

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

Now you check your dataset with

```
data.show(5,True)
```

you will get

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
```

```
+-----------------+-----+-----------------+
only showing top 5 rows
```

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

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

You can check your train and test data as follows (In my opinion, it is always to good to keep tracking your data during prototype pahse):

```
trainingData.show(5)
testData.show(5)
```

Then you will get

```
+---------------+-----+---------------+
|       features|label|indexedFeatures|
+---------------+-----+---------------+
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
| [5.4,29.9,9.4]|  5.3| [5.4,29.9,9.4]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
|[7.8,38.9,50.6]|  6.6|[7.8,38.9,50.6]|
|  [8.6,2.1,1.0]|  4.8|  [8.6,2.1,1.0]|
+---------------+-----+---------------+
only showing top 5 rows

+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|  [8.4,27.2,2.1]|  5.7|  [8.4,27.2,2.1]|
|[11.7,36.9,45.2]|  7.3|[11.7,36.9,45.2]|
|[13.2,15.9,49.6]|  5.6|[13.2,15.9,49.6]|
|[16.9,43.7,89.4]|  8.7|[16.9,43.7,89.4]|
+----------------+-----+----------------+
only showing top 5 rows
```

7. Fit Ordinary Least Square Regression Model

For more details about the parameters, please visit Linear Regression API .

```
# Import LinearRegression class
from pyspark.ml.regression import LinearRegression

# Define LinearRegression algorithm
lr = LinearRegression()
```

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, lr])

model = pipeline.fit(trainingData)
```

9. Summary of the Model

Spark has a poor summary function for data and model. I wrote a summary function which has similar format as **R** output for the linear regression in PySpark.

```
def modelsummary(model):
    import numpy as np
    print ("Note: the last rows are the information for Intercept")
    print ("##","-------------------------------------------------")
    print ("##","  Estimate   |   Std.Error | t Values   |  P-value")
    coef = np.append(list(model.coefficients),model.intercept)
    Summary=model.summary

    for i in range(len(Summary.pValues)):
        print ("##",'{:10.6f}'.format(coef[i]),\
        '{:10.6f}'.format(Summary.coefficientStandardErrors[i]),\
        '{:8.3f}'.format(Summary.tValues[i]),\
        '{:10.6f}'.format(Summary.pValues[i]))

    print ("##",'---')
    print ("##","Mean squared error: % .6f" \
           % Summary.meanSquaredError, ", RMSE: % .6f" \
           % Summary.rootMeanSquaredError )
    print ("##","Multiple R-squared: %f" % Summary.r2, ", \
            Total iterations: %i"% Summary.totalIterations)

modelsummary(model.stages[-1])
```

You will get the following summary results:

```
Note: the last rows are the information for Intercept
('##', '-------------------------------------------------')
('##', '  Estimate   |   Std.Error | t Values   |  P-value')
('##', '  0.044186', '  0.001663', '  26.573', '  0.000000')
('##', '  0.206311', '  0.010846', '  19.022', '  0.000000')
('##', '  0.001963', '  0.007467', '   0.263', '  0.793113')
('##', '  2.596154', '  0.379550', '   6.840', '  0.000000')
('##', '---')
('##', 'Mean squared error:  2.588230', ', RMSE:  1.608798')
('##', 'Multiple R-squared: 0.911869', ',             Total iterations: 1')
```

10. Make predictions

```
# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+----------------+-----+------------------+
|        features|label|        prediction|
+----------------+-----+------------------+
|  [0.7,39.6,8.7]|  1.6| 10.81405928637388|
|  [8.4,27.2,2.1]|  5.7| 8.583086404079918|
```

```
|[11.7,36.9,45.2]|  7.3|10.814712818232422|
|[13.2,15.9,49.6]|  5.6| 6.557106943899219|
|[16.9,43.7,89.4]|  8.7|12.534151375058645|
+-----------------+-----+------------------+
only showing top 5 rows
```

9. Evaluation

```
from pyspark.ml.evaluation import RegressionEvaluator
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(labelCol="label",
                                predictionCol="prediction",
                                metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

The final Root Mean Squared Error (RMSE) is as follows:

```
Root Mean Squared Error (RMSE) on test data = 1.63114
```

You can also check the R^2 value for the test data:

```
y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
```

Then you will get

```
r2_score: 0.854486655585
```

Warning: You should know most softwares are using different formula to calculate the R^2 value when no intercept is included in the model. You can get more information from the disscussion at StackExchange.

7.2 Generalized linear regression

7.2.1 Introduction

7.2.2 How to solve it?

7.2.3 Demo

- The Jupyter notebook can be download from Generalized Linear Regression.
- For more details about the parameters, please visit Generalized Linear Regression API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

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

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                       options(header='true', \
                       inferschema='true').\
            load("../data/Advertising.csv",header=True);
```

check the data set

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

Then you will get

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

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

You can also get the Statistical resutls from the data frame (Unfortunately, it only works for numerical).

```
df.describe().show()
```

Then you will get

```
+-------+-----------------+------------------+------------------+------------------+
|summary|               TV|             Radio|         Newspaper|             Sales|
+-------+-----------------+------------------+------------------+------------------+
|  count|              200|               200|               200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|    min|              0.7|               0.0|               0.3|               1.6|
```

```
|    max|             296.4|              49.6|             114.0|              27.0|
+-------+------------------+------------------+------------------+------------------+
```

3. Convert the data to dense vector (**features** and **label**)

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

# I provide two ways to build the features and labels

# method 1 (good for small feature):
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))

# Method 2 (good for large features):
def transData(data):
return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label'])

transformed= transData(df)
transformed.show(5)

+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

Note: You will find out that all of the machine learning algorithms in Spark are based on the **features** and **label**. That is to say, you can play with all of the machine learning algorithms in Spark when you get ready the **features** and **label**.

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. Deal with the Categorical variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4
# distinct values are treated as continuous.

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

When you check you data at this point, you will get

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----------------+-----+-----------------+
only showing top 5 rows
```

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

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

You can check your train and test data as follows (In my opinion, it is always to good to keep tracking your data during prototype pahse):

```
trainingData.show(5)
testData.show(5)
```

Then you will get

```
+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [5.4,29.9,9.4]|  5.3|  [5.4,29.9,9.4]|
| [7.8,38.9,50.6]|  6.6| [7.8,38.9,50.6]|
|  [8.4,27.2,2.1]|  5.7|  [8.4,27.2,2.1]|
| [8.7,48.9,75.0]|  7.2| [8.7,48.9,75.0]|
|[11.7,36.9,45.2]|  7.3|[11.7,36.9,45.2]|
+----------------+-----+----------------+
only showing top 5 rows

+---------------+-----+---------------+
|       features|label|indexedFeatures|
```

```
+---------------+-----+---------------+
| [0.7,39.6,8.7]|  1.6| [0.7,39.6,8.7]|
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
|  [8.6,2.1,1.0]|  4.8|  [8.6,2.1,1.0]|
|[17.2,4.1,31.6]|  5.9|[17.2,4.1,31.6]|
+---------------+-----+---------------+
only showing top 5 rows
```

7. Fit Generalized Linear Regression Model

```python
# Import LinearRegression class
from pyspark.ml.regression import GeneralizedLinearRegression

# Define LinearRegression algorithm
glr = GeneralizedLinearRegression(family="gaussian", link="identity",\
                                  maxIter=10, regParam=0.3)
```

8. Pipeline Architecture

```python
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, glr])

model = pipeline.fit(trainingData)
```

9. Summary of the Model

Spark has a poor summary function for data and model. I wrote a summary function which has similar format as **R** output for the linear regression in PySpark.

```python
def modelsummary(model):
    import numpy as np
    print ("Note: the last rows are the information for Intercept")
    print ("##","-------------------------------------------------")
    print ("##","  Estimate   |   Std.Error | t Values   |  P-value")
    coef = np.append(list(model.coefficients),model.intercept)
    Summary=model.summary

    for i in range(len(Summary.pValues)):
        print ("##",'{:10.6f}'.format(coef[i]),\
        '{:10.6f}'.format(Summary.coefficientStandardErrors[i]),\
        '{:8.3f}'.format(Summary.tValues[i]),\
        '{:10.6f}'.format(Summary.pValues[i]))

    print ("##",'---')
#     print ("##","Mean squared error: % .6f" \
#            % Summary.meanSquaredError, ", RMSE: % .6f" \
#            % Summary.rootMeanSquaredError )
#     print ("##","Multiple R-squared: %f" % Summary.r2, ", \
#             Total iterations: %i"% Summary.totalIterations)

modelsummary(model.stages[-1])
```

You will get the following summary results:

```
Note: the last rows are the information for Intercept
('##', '-------------------------------------------------')
('##', '  Estimate   |   Std.Error | t Values   |  P-value')
('##', '  0.042857', '  0.001668', '  25.692', '  0.000000')
('##', '  0.199922', '  0.009881', '  20.232', '  0.000000')
('##', ' -0.001957', '  0.006917', '  -0.283', '  0.777757')
('##', '  3.007515', '  0.406389', '   7.401', '  0.000000')
('##', '---')
```

10. Make predictions

```
# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+---------------+-----+------------------+
|       features|label|        prediction|
+---------------+-----+------------------+
| [0.7,39.6,8.7]|  1.6|10.937383732327625|
| [4.1,11.6,5.7]|  3.2| 5.491166258750164|
|[7.3,28.1,41.4]|  5.5|   8.8571603947873|
|  [8.6,2.1,1.0]|  4.8| 3.793966281660073|
|[17.2,4.1,31.6]|  5.9| 4.502507124763654|
+---------------+-----+------------------+
only showing top 5 rows
```

11. Evaluation

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import RegressionEvaluator
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(labelCol="label",
                                predictionCol="prediction",
                                metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

The final Root Mean Squared Error (RMSE) is as follows:

```
Root Mean Squared Error (RMSE) on test data = 1.89857
```

```
y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
```

Then you will get the R^2 value:

```
r2_score: 0.87707391843
```

7.3 Decision tree Regression

7.3.1 Introduction

7.3.2 How to solve it?

7.3.3 Demo

- The Jupyter notebook can be download from Decision Tree Regression.
- For more details about the parameters, please visit Decision Tree Regressor API .

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

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

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                       options(header='true', \
                       inferschema='true').\
            load("../data/Advertising.csv",header=True);
```

check the data set

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

Then you will get

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

root
 |-- TV: double (nullable = true)
```

```
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

You can also get the Statistical resutls from the data frame (Unfortunately, it only works for numerical).

```
df.describe().show()
```

Then you will get

```
+-------+-----------------+-----------------+------------------+------------------+
|summary|               TV|            Radio|         Newspaper|             Sales|
+-------+-----------------+-----------------+------------------+------------------+
|  count|              200|              200|               200|               200|
|   mean|         147.0425|23.26400000000024|30.553999999999995|14.022500000000003|
| stddev|85.85423631490805|14.84680917616872| 21.77862083852283| 5.217456565710477|
|    min|              0.7|              0.0|               0.3|               1.6|
|    max|            296.4|             49.6|             114.0|              27.0|
+-------+-----------------+-----------------+------------------+------------------+
```

3. Convert the data to dense vector (**features** and **label**)

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

# I provide two ways to build the features and labels

# method 1 (good for small feature):
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))

# Method 2 (good for large features):
def transData(data):
return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF(['features','label'])

transformed= transData(df)
transformed.show(5)

+-----------------+-----+
|         features|label|
+-----------------+-----+
|[230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
|[151.5,41.3,58.5]| 18.5|
|[180.8,10.8,58.4]| 12.9|
+-----------------+-----+
only showing top 5 rows
```

Note: You will find out that all of the machine learning algorithms in Spark are based on the **features** and

label. That is to say, you can play with all of the machine learning algorithms in Spark when you get ready the **features** and **label**.

---

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'])

transformed = transData(df)
transformed.show(5)
```

5. Deal with the Categorical variables

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4
# distinct values are treated as continuous.

featureIndexer = VectorIndexer(inputCol="features", \
                               outputCol="indexedFeatures",\
                               maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

When you check you data at this point, you will get

```
+-----------------+-----+-----------------+
|         features|label|  indexedFeatures|
+-----------------+-----+-----------------+
|[230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4| [44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3| [17.2,45.9,69.3]|
|[151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
|[180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----------------+-----+-----------------+
only showing top 5 rows
```

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

```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

You can check your train and test data as follows (In my opinion, it is always to good to keep tracking your data during prototype pahse):

```
trainingData.show(5)
testData.show(5)
```

Then you will get

```
+----------------+-----+----------------+
|        features|label|indexedFeatures|
+----------------+-----+----------------+
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7]|
|[7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4]|
| [8.4,27.2,2.1]|  5.7| [8.4,27.2,2.1]|
|  [8.6,2.1,1.0]|  4.8|  [8.6,2.1,1.0]|
|[8.7,48.9,75.0]|  7.2|[8.7,48.9,75.0]|
+----------------+-----+----------------+
only showing top 5 rows

+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|  [5.4,29.9,9.4]|  5.3|  [5.4,29.9,9.4]|
| [7.8,38.9,50.6]|  6.6| [7.8,38.9,50.6]|
|[17.2,45.9,69.3]|  9.3|[17.2,45.9,69.3]|
|[18.7,12.1,23.4]|  6.7|[18.7,12.1,23.4]|
+----------------+-----+----------------+
only showing top 5 rows
```

7. Fit Decision Tree Regression Model

```
from pyspark.ml.regression import DecisionTreeRegressor

# Train a DecisionTree model.
dt = DecisionTreeRegressor(featuresCol="indexedFeatures")
```

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, dt])

model = pipeline.fit(trainingData)
```

9. Make predictions

```
# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
+----------+-----+----------------+
|prediction|label|        features|
+----------+-----+----------------+
|       7.2|  1.6|  [0.7,39.6,8.7]|
|       7.3|  5.3|  [5.4,29.9,9.4]|
|       7.2|  6.6| [7.8,38.9,50.6]|
|      8.64|  9.3|[17.2,45.9,69.3]|
|      6.45|  6.7|[18.7,12.1,23.4]|
```

```
+----------+-----+----------------+
only showing top 5 rows
```

10. Evaluation

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import RegressionEvaluator
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(labelCol="label",
                                predictionCol="prediction",
                                metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

The final Root Mean Squared Error (RMSE) is as follows:

```
Root Mean Squared Error (RMSE) on test data = 1.50999
```

```
y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
```

Then you will get the R^2 value:

```
r2_score: 0.911024318967
```

You may also check the importance of the features:

```
model.stages[1].featureImportances
```

The you will get the weight for each features

```
SparseVector(3, {0: 0.6811, 1: 0.3187, 2: 0.0002})
```

7.4 Random Forest Regression

7.4.1 Introduction

7.4.2 How to solve it?

7.4.3 Demo

- The Jupyter notebook can be download from Random Forest Regression.
- For more details about the parameters, please visit Random Forest Regressor API .

7.5 Gradient-boosted tree regression

7.5.1 Introduction

7.5.2 How to solve it?

7.5.3 Demo

- The Jupyter notebook can be download from Gradient-boosted tree regression.
- For more details about the parameters, please visit Gradient boosted tree API .

CHAPTER

EIGHT

CLASSIFICATION

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

8.1 Logistic regression

8.1.1 Binomial logistic regression

8.1.2 Multinomial logistic regression

8.2 Decision tree Classification

- The Jupyter notebook can be download from Decision Tree Classification.
- For more details, please visit DecisionTreeClassifier API .

8.3 Random forest Classification

- The Jupyter notebook can be download from Random forest Classification.
- For more details, please visit RandomForestClassifier API .

8.4 Gradient-boosted tree Classification

- The Jupyter notebook can be download from Gradient boosted tree Classification.
- For more details, please visit GBTClassifier API .

Warning: Unfortunately, the GBTClassifier currently only supports binary labels.

8.5 Naive Bayes Classification

- The Jupyter notebook can be download from Naive Bayes Classification.
- For more details, please visit NaiveBayes API .

8.6 Support Vector Machines Classification

CHAPTER

NINE

CLUSTERING

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

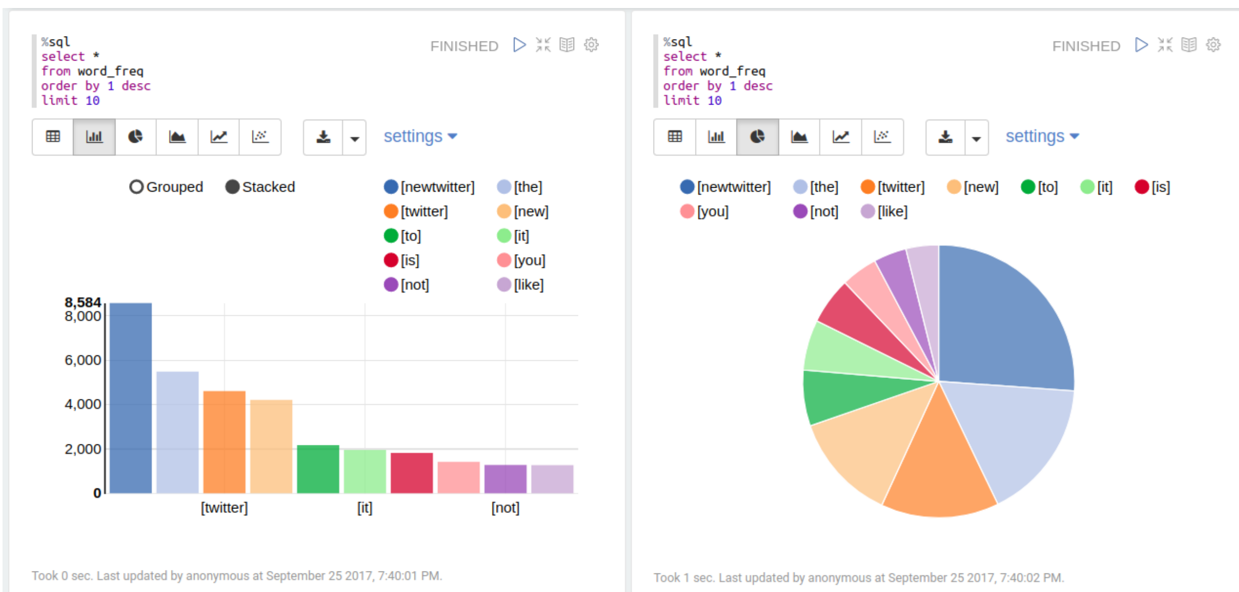
9.1 K-Means Model

CHAPTER

TEN

TEXT MINING

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



10.1 Text Collection

10.1.1 Image to text

- My `img2txt` function

```
def img2txt(img_dir):
    """
    convert images to text
    """
    import os, PythonMagick
    from datetime import datetime
    import PyPDF2

    from PIL import Image
```

```
import pytesseract

f = open('doc4img.txt','wa')
for img in [img_file for img_file in os.listdir(img_dir)
            if (img_file.endswith(".png") or
                img_file.endswith(".jpg") or
                img_file.endswith(".jpeg"))]:

    start_time = datetime.now()

    input_img = img_dir + "/" + img

    print('--------------------------------------------------------------------')
    print(img)
    print('Converting ' + img +'.......')
    print('--------------------------------------------------------------------')

    # extract the text information from images
    text = pytesseract.image_to_string(Image.open(input_img))
    print(text)

    # ouput text file
    f.write( img + "\n")
    f.write(text.encode('utf-8'))


    print "CPU Time for converting" + img +":"+ str(datetime.now() - start_time) +"\n"
    f.write( "\n-------------------------------------------------------------\n")

f.close()
```

- Demo

I applied my `img2txt` function to the image in Image folder.

```
image_dir = r"Image"

img2txt(image_dir)
```

Then I got the following results:

```
--------------------------------------------------------------------
feng.pdf_0.png
Converting feng.pdf_0.png.......
--------------------------------------------------------------------
l I l w

Wenqiang Feng
Data Scientist
DST APPLIED ANALYTICS GROUP



Wenqiang Feng is Data Scientist for DST's Applied Analytics Group. Dr. Feng's responsibilit
```

```
include providing DST clients with access to cutting--edge skills and technologies, includi
Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning
algorithms, business intelligence, and applying Big Data tools to strategically solve indus
problems in a cross--functional business. Before joining the DST Applied Analytics Group, D
Feng holds a MA Data Science Fellow at The Institute for Mathematics and Its Applications
{IMA) at the University of Minnesota. While there, he helped startup companies make
marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville with PhD in Computational
mathematics and Master's degree in Statistics. He also holds Master's degree in Computation
Mathematics at Missouri University of Science and Technology (MST) and Master's degree in
Applied Mathematics at University of science and technology of China (USTC).
CPU Time for convertingfeng.pdf_0.png:0:00:02.061208
```

10.1.2 Image Enhnaced to text

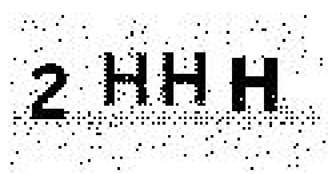
- My `img2txt_enhance` function

```
def img2txt_enhance(img_dir,scaler):
    """
    convert images files to text
    """

    import numpy as np
    import os, PythonMagick
    from datetime import datetime
    import PyPDF2

    from PIL import Image, ImageEnhance, ImageFilter
    import pytesseract

    f = open('doc4img.txt','wa')
    for img in [img_file for img_file in os.listdir(img_dir)
                if (img_file.endswith(".png") or
                    img_file.endswith(".jpg") or
                    img_file.endswith(".jpeg"))]:

        start_time = datetime.now()

        input_img = img_dir + "/" + img
        enhanced_img = img_dir + "/" +"Enhanced" + "/"+ img

        im = Image.open(input_img) # the second one
        im = im.filter(ImageFilter.MedianFilter())
        enhancer = ImageEnhance.Contrast(im)
        im = enhancer.enhance(1)
        im = im.convert('1')
        im.save(enhanced_img)

        for scale in np.ones(scaler):
```

```
            im = Image.open(enhanced_img) # the second one
            im = im.filter(ImageFilter.MedianFilter())
            enhancer = ImageEnhance.Contrast(im)
            im = enhancer.enhance(scale)
            im = im.convert('1')
            im.save(enhanced_img)


        print('--------------------------------------------------------------------')
        print(img)
        print('Converting ' + img +'.......')
        print('--------------------------------------------------------------------')

        # extract the text information from images
        text = pytesseract.image_to_string(Image.open(enhanced_img))
        print(text)

        # ouput text file
        f.write( img + "\n")
        f.write(text.encode('utf-8'))


        print "CPU Time for converting" + img +":"+ str(datetime.now() - start_time) +"\n"
        f.write( "\n-------------------------------------------------------------\n")

    f.close()
```

- Demo

I applied my `img2txt_enhance` function to the following noised image in Enhance folder.




```
image_dir = r"Enhance"

pdf2txt_enhance(image_dir)
```

Then I got the following results:

```
--------------------------------------------------------------------
noised.jpg
Converting noised.jpg.......
--------------------------------------------------------------------
zHHH
CPU Time for convertingnoised.jpg:0:00:00.135465
```

while the result from `img2txt` function is

```
--------------------------------------------------------------------
noised.jpg
Converting noised.jpg.......
--------------------------------------------------------------------
,2 WW
CPU Time for convertingnoised.jpg:0:00:00.133508
```

which is not correct.

10.1.3 PDF to text

- My pdf2txt function

```
def pdf2txt(pdf_dir,image_dir):
    """
    convert PDF to text
    """

    import os, PythonMagick
    from datetime import datetime
    import PyPDF2

    from PIL import Image
    import pytesseract

    f = open('doc.txt','wa')
    for pdf in [pdf_file for pdf_file in os.listdir(pdf_dir) if pdf_file.endswith(".pdf")]

        start_time = datetime.now()

        input_pdf = pdf_dir + "/" + pdf

        pdf_im = PyPDF2.PdfFileReader(file(input_pdf, "rb"))
        npage = pdf_im.getNumPages()

        print('--------------------------------------------------------------------')
        print(pdf)
        print('Converting %d pages.' % npage)
        print('--------------------------------------------------------------------')

        f.write( "\n--------------------------------------------------------------------\n"

        for p in range(npage):

            pdf_file = input_pdf + '[' + str(p) +']'
            image_file =  image_dir  + "/" + pdf+ '_' + str(p)+ '.png'

            # convert PDF files to Images
            im = PythonMagick.Image()
            im.density('300')
            im.read(pdf_file)
            im.write(image_file)
```

```
            # extract the text information from images
            text = pytesseract.image_to_string(Image.open(image_file))

            #print(text)

            # ouput text file
            f.write( pdf + "\n")
            f.write(text.encode('utf-8'))


        print "CPU Time for converting" + pdf +":"+ str(datetime.now() - start_time) +"\n"

    f.close()
```

- Demo

I applied my `pdf2txt` function to my scaned bio pdf file in pdf folder.

```
pdf_dir = r"pdf"
image_dir = r"Image"

pdf2txt(pdf_dir,image_dir)
```

Then I got the following results:

```
--------------------------------------------------------------------
feng.pdf
Converting 1 pages.
--------------------------------------------------------------------
l I l w

Wenqiang Feng
Data Scientist
DST APPLIED ANALYTICS GROUP



Wenqiang Feng is Data Scientist for DST's Applied Analytics Group. Dr. Feng's responsibilit
include providing DST clients with access to cutting--edge skills and technologies, includi
Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning
algorithms, business intelligence, and applying Big Data tools to strategically solve indus
problems in a cross--functional business. Before joining the DST Applied Analytics Group, D
Feng holds a MA Data Science Fellow at The Institute for Mathematics and Its Applications
{IMA) at the University of Minnesota. While there, he helped startup companies make
marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville with PhD in Computational
mathematics and Master's degree in Statistics. He also holds Master's degree in Computation
Mathematics at Missouri University of Science and Technology (MST) and Master's degree in
Applied Mathematics at University of science and technology of China (USTC).
CPU Time for convertingfeng.pdf:0:00:03.143800
```

10.1.4 Audio to text

- My `audio2txt` function

```
def audio2txt(audio_dir):
    ''' convert audio to text'''

    import speech_recognition as sr
    r = sr.Recognizer()

    f = open('doc.txt','wa')
    for audio_n in [audio_file for audio_file in os.listdir(audio_dir) \
                    if audio_file.endswith(".wav")]:

        filename = audio_dir + "/" + audio_n

        # Read audio data
        with sr.AudioFile(filename) as source:
            audio = r.record(source)  # read the entire audio file

        # Google Speech Recognition
        text = r.recognize_google(audio)

        # ouput text file
        f.write( audio_n + ": ")
        f.write(text.encode('utf-8'))
        f.write("\n")

        print('You said: ' + text)

    f.close()
```

- Demo

I applied my `audio2txt` function to my audio records in audio folder.

```
audio_dir = r"audio"

audio2txt(audio_dir)
```

Then I got the following results:

```
You said: hello this is George welcome to my tutorial
You said: mathematics is important in daily life
You said: call me tomorrow
You said: do you want something to eat
You said: I want to speak with him
You said: nice to see you
You said: can you speak slowly
You said: have a good day
```

By the way, you can use my following python code to record your own audio and play with `audio2txt` function:

```
import speech_recognition as sr

audio_filename = "test9.wav"

r = sr.Recognizer()
with sr.Microphone() as source:
    r.adjust_for_ambient_noise(source)
    print("Hey there, say something, I am recording!")
    audio = r.listen(source)
    print("Done listening!")

with open(audio_filename, "wb") as f:
    f.write(audio.get_wav_data())
```

10.2 Text Preprocessing

- check to see if a row only contains whitespace

```
def check_blanks(data_str):
    is_blank = str(data_str.isspace())
    return is_blank
```

- Determine whether the language of the text content is english or not: Use langid module to classify the language to make sure we are applying the correct cleanup actions for English langid

```
def check_lang(data_str):
    predict_lang = langid.classify(data_str)
    if predict_lang[1] >= .9:
        language = predict_lang[0]
    else:
        language = 'NA'
    return language
```

- Remove features

```
def remove_features(data_str):
    # compile regex
    url_re = re.compile('https?://(www.)?\w+\.\w+(/\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # convert to lowercase
    data_str = data_str.lower()
    # remove hyperlinks
    data_str = url_re.sub(' ', data_str)
    # remove @mentions
    data_str = mention_re.sub(' ', data_str)
    # remove puncuation
    data_str = punc_re.sub(' ', data_str)
    # remove numeric 'words'
```

```
    data_str = num_re.sub(' ', data_str)
    # remove non a-z 0-9 characters and words shorter than 3 characters
    list_pos = 0
    cleaned_str = ''
    for word in data_str.split():
        if list_pos == 0:
            if alpha_num_re.match(word) and len(word) > 2:
                cleaned_str = word
            else:
                cleaned_str = ' '
        else:
            if alpha_num_re.match(word) and len(word) > 2:
                cleaned_str = cleaned_str + ' ' + word
            else:
                cleaned_str += ' '
        list_pos += 1
    return cleaned_str
```

- removes stop words

```
def remove_stops(data_str):
    # expects a string
    stops = set(stopwords.words("english"))
    list_pos = 0
    cleaned_str = ''
    text = data_str.split()
    for word in text:
        if word not in stops:
            # rebuild cleaned_str
            if list_pos == 0:
                cleaned_str = word
            else:
                cleaned_str = cleaned_str + ' ' + word
            list_pos += 1
    return cleaned_str
```

- tagging text

```
def tag_and_remove(data_str):
    cleaned_str = ' '
    # noun tags
    nn_tags = ['NN', 'NNP', 'NNP', 'NNPS', 'NNS']
    # adjectives
    jj_tags = ['JJ', 'JJR', 'JJS']
    # verbs
    vb_tags = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    nltk_tags = nn_tags + jj_tags + vb_tags

    # break string into 'words'
    text = data_str.split()

    # tag the text and keep only those with the right tags
    tagged_text = pos_tag(text)
    for tagged_word in tagged_text:
```

```
        if tagged_word[1] in nltk_tags:
            cleaned_str += tagged_word[0] + ' '

    return cleaned_str
```

- lemmatization

```
def lemmatize(data_str):
    # expects a string
    list_pos = 0
    cleaned_str = ''
    lmtzr = WordNetLemmatizer()
    text = data_str.split()
    tagged_words = pos_tag(text)
    for word in tagged_words:
        if 'v' in word[1].lower():
            lemma = lmtzr.lemmatize(word[0], pos='v')
        else:
            lemma = lmtzr.lemmatize(word[0], pos='n')
        if list_pos == 0:
            cleaned_str = lemma
        else:
            cleaned_str = cleaned_str + ' ' + lemma
        list_pos += 1
    return cleaned_str
```

define the preprocessing function in PySpark

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import preproc as pp

check_lang_udf = udf(pp.check_lang, StringType())
remove_stops_udf = udf(pp.remove_stops, StringType())
remove_features_udf = udf(pp.remove_features, StringType())
tag_and_remove_udf = udf(pp.tag_and_remove, StringType())
lemmatize_udf = udf(pp.lemmatize, StringType())
check_blanks_udf = udf(pp.check_blanks, StringType())
```

10.3 Text Classification

```
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import string
import re
import langid
```

10.4 Sentiment analysis

10.4.1 Introduction

Sentiment analysis (sometimes known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

Generally speaking, sentiment analysis aims to **determine the attitude** of a speaker, writer, or other subject with respect to some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event. The attitude may be a judgment or evaluation (see appraisal theory), affective state (that is to say, the emotional state of the author or speaker), or the intended emotional communication (that is to say, the emotional effect intended by the author or interlocutor).

Sentiment analysis in business, also known as opinion mining is a process of identifying and cataloging a piece of text according to the tone conveyed by it. It has broad application:

- Sentiment Analysis in Business Intelligence Build up
- Sentiment Analysis in Business for Competitive Advantage
- Enhancing the Customer Experience through Sentiment Analysis in Business

10.4.2 Pipeline

Figure 10.1: Sentiment Analysis Pipeline

10.4.3 Demo

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

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

2. Load dataset

```
df = spark.read.format('com.databricks.spark.csv').\
                                options(header='true', \
                                inferschema='true').\
            load("../data/newtwitter.csv",header=True);
```

```
+--------------------+----------+-------+
|                text|        id|pubdate|
+--------------------+----------+-------+
|10 Things Missing...|2602860537|  18536|
|RT @_NATURALBWINN...|2602850443|  18536|
|RT @HBO24 yo the ...|2602761852|  18535|
|Aaaaaaaand I have...|2602738438|  18535|
|can I please have...|2602684185|  18535|
+--------------------+----------+-------+
only showing top 5 rows
```

3. Text Preprocessing

- remove non ASCII characters

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import string
import re

# remove non ASCII characters
def strip_non_ascii(data_str):
    ''' Returns the string without non ASCII characters'''
    stripped = (c for c in data_str if 0 < ord(c) < 127)
    return ''.join(stripped)
# setup pyspark udf function
strip_non_ascii_udf = udf(strip_non_ascii, StringType())
```

check:

```
df = df.withColumn('text_non_asci',strip_non_ascii_udf(df['text']))
df.show(5,True)
```

ouput:

```
+--------------------+----------+-------+--------------------+
|                text|        id|pubdate|       text_non_asci|
+--------------------+----------+-------+--------------------+
|10 Things Missing...|2602860537|  18536|10 Things Missing...|
|RT @_NATURALBWINN...|2602850443|  18536|RT @_NATURALBWINN...|
|RT @HBO24 yo the ...|2602761852|  18535|RT @HBO24 yo the ...|
|Aaaaaaaand I have...|2602738438|  18535|Aaaaaaaand I have...|
|can I please have...|2602684185|  18535|can I please have...|
+--------------------+----------+-------+--------------------+
only showing top 5 rows
```

- fixed abbreviation

```
# fixed abbreviation
def fix_abbreviation(data_str):
    data_str = data_str.lower()
    data_str = re.sub(r'\bthats\b', 'that is', data_str)
    data_str = re.sub(r'\bive\b', 'i have', data_str)
    data_str = re.sub(r'\bim\b', 'i am', data_str)
    data_str = re.sub(r'\bya\b', 'yeah', data_str)
    data_str = re.sub(r'\bcant\b', 'can not', data_str)
    data_str = re.sub(r'\bdont\b', 'do not', data_str)
    data_str = re.sub(r'\bwont\b', 'will not', data_str)
    data_str = re.sub(r'\bid\b', 'i would', data_str)
    data_str = re.sub(r'wtf', 'what the fuck', data_str)
    data_str = re.sub(r'\bwth\b', 'what the hell', data_str)
    data_str = re.sub(r'\br\b', 'are', data_str)
    data_str = re.sub(r'\bu\b', 'you', data_str)
    data_str = re.sub(r'\bk\b', 'OK', data_str)
    data_str = re.sub(r'\bsux\b', 'sucks', data_str)
    data_str = re.sub(r'\bno+\b', 'no', data_str)
    data_str = re.sub(r'\bcoo+\b', 'cool', data_str)
    data_str = re.sub(r'rt\b', '', data_str)
    data_str = data_str.strip()
    return data_str

fix_abbreviation_udf = udf(fix_abbreviation, StringType())
```

check:

```
df = df.withColumn('fixed_abbrev',fix_abbreviation_udf(df['text_non_asci']))
df.show(5,True)
```

ouput:

```
+--------------------+----------+-------+--------------------+--------------------+
|                text|        id|pubdate|       text_non_asci|        fixed_abbrev|
+--------------------+----------+-------+--------------------+--------------------+
|10 Things Missing...|2602860537|  18536|10 Things Missing...|10 things missing...|
|RT @_NATURALBWINN...|2602850443|  18536|RT @_NATURALBWINN...|@_naturalbwinner ...|
|RT @HBO24 yo the ...|2602761852|  18535|RT @HBO24 yo the ...|@hbo24 yo the #ne...|
|Aaaaaaaand I have...|2602738438|  18535|Aaaaaaaand I have...|aaaaaaaand i have...|
|can I please have...|2602684185|  18535|can I please have...|can i please have...|
+--------------------+----------+-------+--------------------+--------------------+
only showing top 5 rows
```

- remove irrelevant features

```
def remove_features(data_str):
    # compile regex
    url_re = re.compile('https?://(www.)?\w+\.\w+(/\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # convert to lowercase
```

```
    data_str = data_str.lower()
    # remove hyperlinks
    data_str = url_re.sub(' ', data_str)
    # remove @mentions
    data_str = mention_re.sub(' ', data_str)
    # remove puncuation
    data_str = punc_re.sub(' ', data_str)
    # remove numeric 'words'
    data_str = num_re.sub(' ', data_str)
    # remove non a-z 0-9 characters and words shorter than 1 characters
    list_pos = 0
    cleaned_str = ''
    for word in data_str.split():
        if list_pos == 0:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = word
            else:
                cleaned_str = ' '
        else:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = cleaned_str + ' ' + word
            else:
                cleaned_str += ' '
        list_pos += 1
    # remove unwanted space, *.split() will automatically split on
    # whitespace and discard duplicates, the " ".join() joins the
    # resulting list into one string.
    return " ".join(cleaned_str.split())
# setup pyspark udf function
remove_features_udf = udf(remove_features, StringType())
```

check:

```
df = df.withColumn('removed',remove_features_udf(df['fixed_abbrev']))
df.show(5,True)
```

ouput:

```
+--------------------+----------+-------+--------------------+--------------------+--------
|                text|        id|pubdate|      text_non_asci|        fixed_abbrev|
+--------------------+----------+-------+--------------------+--------------------+--------
|10 Things Missing...|2602860537|  18536|10 Things Missing...|10 things missing...|things m
|RT @_NATURALBWINN...|2602850443|  18536|RT @_NATURALBWINN...|@_naturalbwinner ...|oh and c
|RT @HBO24 yo the ...|2602761852|  18535|RT @HBO24 yo the ...|@hbo24 yo the #ne...|yo the n
|Aaaaaaaand I have...|2602738438|  18535|Aaaaaaaand I have...|aaaaaaaand i have...|aaaaaaaa
|can I please have...|2602684185|  18535|can I please have...|can i please have...|can plea
+--------------------+----------+-------+--------------------+--------------------+--------
only showing top 5 rows
```

4. Sentiment Analysis main function

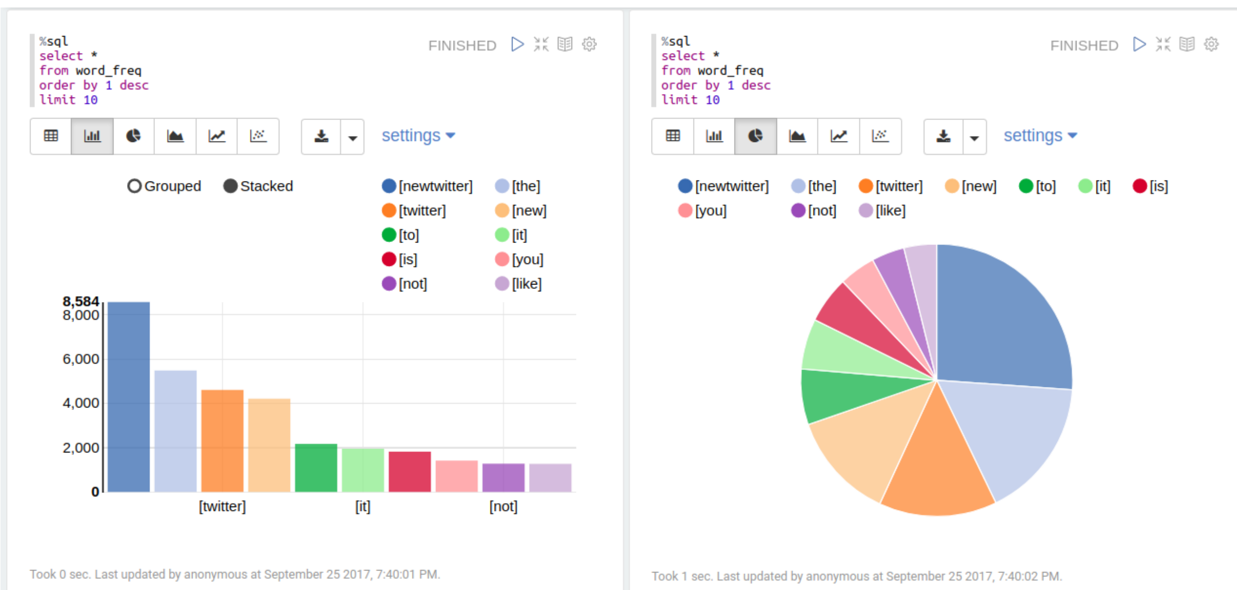
```
from pyspark.sql.types import FloatType

from textblob import TextBlob
```

```
def sentiment_analysis(text):
    return TextBlob(text).sentiment.polarity

sentiment_analysis_udf = udf(sentiment_analysis , FloatType())

df  = df.withColumn("sentiment_score", sentiment_analysis_udf( df['removed'] ))
df.show(5,True)
```

- Sentiment score

```
+--------------------+---------------+
|             removed|sentiment_score|
+--------------------+---------------+
|things missing in...|    -0.03181818|
|oh and do not lik...|    -0.03181818|
|yo the newtwitter...|      0.3181818|
|aaaaaaaand have t...|     0.11818182|
|can please have t...|     0.13636364|
+--------------------+---------------+
only showing top 5 rows
```

- Words frequency



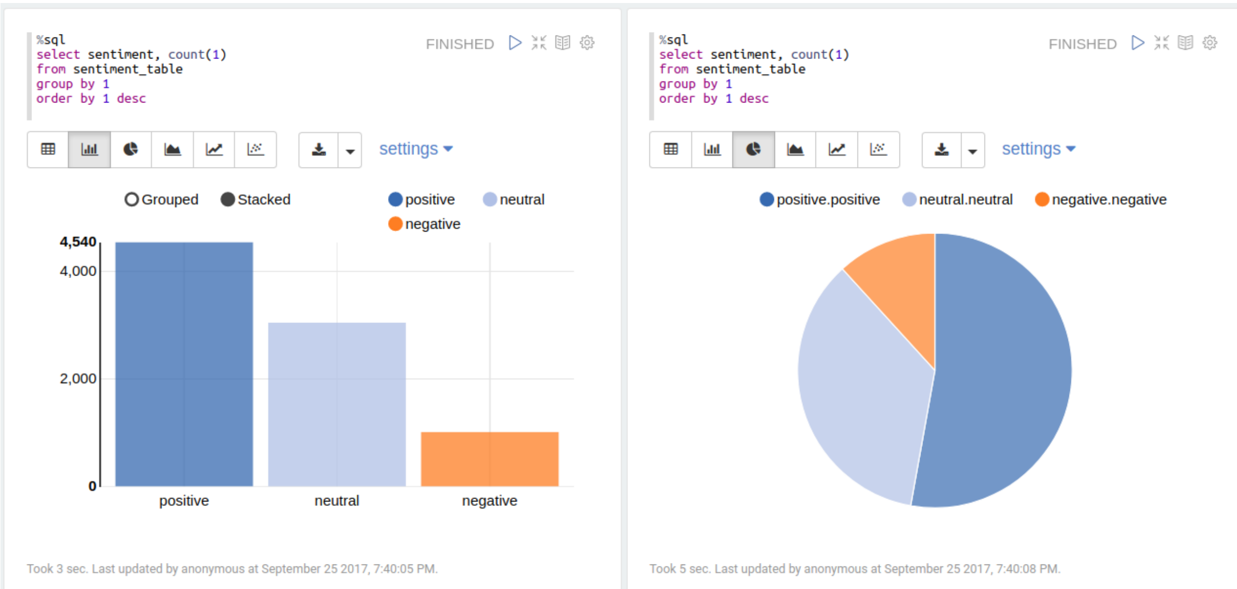
- Sentiment Classification

```
def condition(r):
    if (r >=0.1):
        label = "positive"
    elif(r <= -0.1):
        label = "negative"
    else:
        label = "neutral"
    return label
```

```python
sentiment_udf = udf(lambda x: condition(x), StringType())
```

5. Output

- Sentiment Class



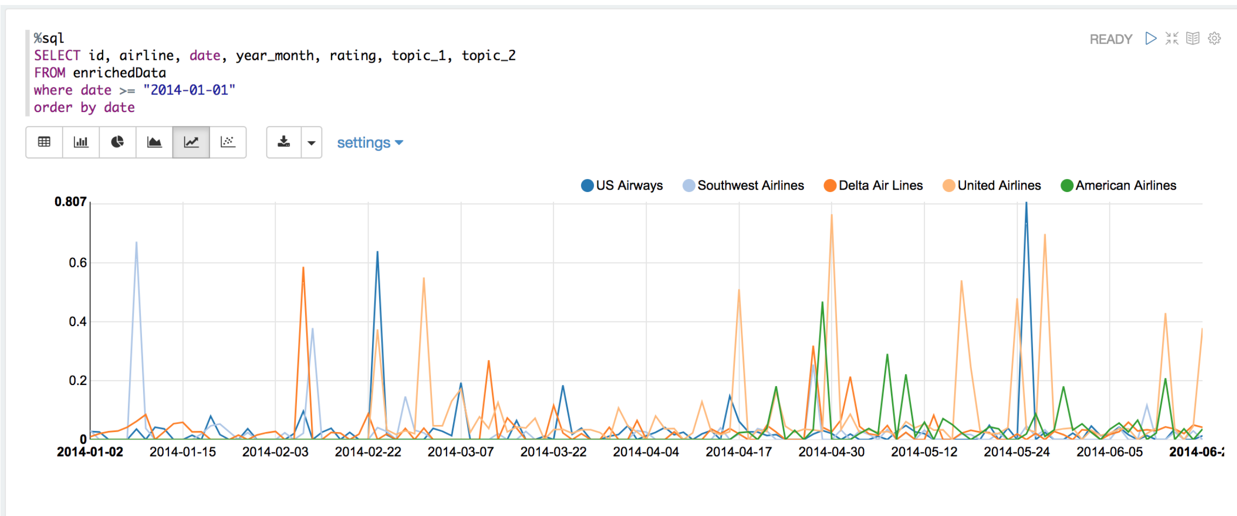
- Top tweets from each sentiment class

```
+--------------------+---------------+---------+
|                text|sentiment_score|sentiment|
+--------------------+---------------+---------+
|and this #newtwit...|            1.0| positive|
|"RT @SarahsJokes:...|            1.0| positive|
|#newtwitter using...|            1.0| positive|
|The #NewTwitter h...|            1.0| positive|
|You can now undo ...|            1.0| positive|
+--------------------+---------------+---------+
only showing top 5 rows

+--------------------+---------------+---------+
|                text|sentiment_score|sentiment|
+--------------------+---------------+---------+
|Lists on #NewTwit...|           -0.1|  neutral|
|Too bad most of m...|           -0.1|  neutral|
|the #newtwitter i...|           -0.1|  neutral|
|Looks like our re...|           -0.1|  neutral|
|i switched to the...|           -0.1|  neutral|
+--------------------+---------------+---------+
only showing top 5 rows

+--------------------+---------------+---------+
|                text|sentiment_score|sentiment|
+--------------------+---------------+---------+
|oh. #newtwitter i...|           -1.0| negative|
```

```
|RT @chqwn: #NewTw...|          -1.0| negative|
|Copy that - its W...|          -1.0| negative|
|RT @chqwn: #NewTw...|          -1.0| negative|
|#NewTwitter has t...|          -1.0| negative|
+--------------------+--------------+---------+
only showing top 5 rows
```

10.5 N-grams and Correlations

10.6 Topic Model: Latent Dirichlet Allocation



10.6.1 Introduction

In text mining, a topic model is a unsupervised model for discovering the abstract "topics" that occur in a collection of documents.

Latent Dirichlet Allocation (LDA) is a mathematical method for estimating both of these at the same time: finding the mixture of words that is associated with each topic, while also determining the mixture of topics that describes each document.

10.6.2 Demo

1. Load data

```
rawdata = spark.read.load("../data/airlines.csv", format="csv", header=True)
rawdata.show(5)
```

```
+-----+---------------+---------+--------+------+--------+-----+-----------+----------
|   id|        airline|     date|location|rating|   cabin|value|recommended|
```

```
+-----+---------------+---------+--------+------+--------+-----+-----------+----------
|10001|Delta Air Lines|21-Jun-14|Thailand|     7| Economy|    4|        YES|Flew Mar 3
|10002|Delta Air Lines|19-Jun-14|     USA|     0| Economy|    2|         NO|Flight 246
|10003|Delta Air Lines|18-Jun-14|     USA|     0| Economy|    1|         NO|Delta Webs
|10004|Delta Air Lines|17-Jun-14|     USA|     9|Business|    4|        YES|"I just re
|10005|Delta Air Lines|17-Jun-14| Ecuador|     7| Economy|    3|        YES|"Round-tri
+-----+---------------+---------+--------+------+--------+-----+-----------+----------
only showing top 5 rows
```

1. Text preprocessing

I will use the following raw column names to keep my table concise:

```
raw_cols =  rawdata.columns
raw_cols

['id', 'airline', 'date', 'location', 'rating', 'cabin', 'value', 'recommended', 'revi

rawdata = rawdata.dropDuplicates(['review'])

from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType, DoubleType, DateType

from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import langid
import string
import re
```

- remove non ASCII characters

```
# remove non ASCII characters
def strip_non_ascii(data_str):
    ''' Returns the string without non ASCII characters'''
    stripped = (c for c in data_str if 0 < ord(c) < 127)
    return ''.join(stripped)
```

- check it blank line or not

```
# check to see if a row only contains whitespace
def check_blanks(data_str):
    is_blank = str(data_str.isspace())
    return is_blank
```

- check the language (a little bit slow, I skited this step)

```
# check the language (only apply to english)
def check_lang(data_str):
    from langid.langid import LanguageIdentifier, model
    identifier = LanguageIdentifier.from_modelstring(model, norm_probs=True)
    predict_lang = identifier.classify(data_str)

    if predict_lang[1] >= .9:
```

```
        language = predict_lang[0]
    else:
        language = predict_lang[0]
    return language
```

- fixed abbreviation

```
# fixed abbreviation
def fix_abbreviation(data_str):
    data_str = data_str.lower()
    data_str = re.sub(r'\bthats\b', 'that is', data_str)
    data_str = re.sub(r'\bive\b', 'i have', data_str)
    data_str = re.sub(r'\bim\b', 'i am', data_str)
    data_str = re.sub(r'\bya\b', 'yeah', data_str)
    data_str = re.sub(r'\bcant\b', 'can not', data_str)
    data_str = re.sub(r'\bdont\b', 'do not', data_str)
    data_str = re.sub(r'\bwont\b', 'will not', data_str)
    data_str = re.sub(r'\bid\b', 'i would', data_str)
    data_str = re.sub(r'wtf', 'what the fuck', data_str)
    data_str = re.sub(r'\bwth\b', 'what the hell', data_str)
    data_str = re.sub(r'\br\b', 'are', data_str)
    data_str = re.sub(r'\bu\b', 'you', data_str)
    data_str = re.sub(r'\bk\b', 'OK', data_str)
    data_str = re.sub(r'\bsux\b', 'sucks', data_str)
    data_str = re.sub(r'\bno+\b', 'no', data_str)
    data_str = re.sub(r'\bcoo+\b', 'cool', data_str)
    data_str = re.sub(r'rt\b', '', data_str)
    data_str = data_str.strip()
    return data_str
```

- remove irrelevant features

```
# remove irrelevant features
def remove_features(data_str):
    # compile regex
    url_re = re.compile('https?://(www.)?\w+\.\w+(/\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # convert to lowercase
    data_str = data_str.lower()
    # remove hyperlinks
    data_str = url_re.sub(' ', data_str)
    # remove @mentions
    data_str = mention_re.sub(' ', data_str)
    # remove puncuation
    data_str = punc_re.sub(' ', data_str)
    # remove numeric 'words'
    data_str = num_re.sub(' ', data_str)
    # remove non a-z 0-9 characters and words shorter than 1 characters
    list_pos = 0
    cleaned_str = ''
    for word in data_str.split():
```

```
            if list_pos == 0:
                if alpha_num_re.match(word) and len(word) > 1:
                    cleaned_str = word
                else:
                    cleaned_str = ' '
            else:
                if alpha_num_re.match(word) and len(word) > 1:
                    cleaned_str = cleaned_str + ' ' + word
                else:
                    cleaned_str += ' '
            list_pos += 1
    # remove unwanted space, *.split() will automatically split on
    # whitespace and discard duplicates, the " ".join() joins the
    # resulting list into one string.
    return " ".join(cleaned_str.split())
```

- removes stop words

```
# removes stop words
def remove_stops(data_str):
    # expects a string
    stops = set(stopwords.words("english"))
    list_pos = 0
    cleaned_str = ''
    text = data_str.split()
    for word in text:
        if word not in stops:
            # rebuild cleaned_str
            if list_pos == 0:
                cleaned_str = word
            else:
                cleaned_str = cleaned_str + ' ' + word
            list_pos += 1
    return cleaned_str
```

- Part-of-Speech Tagging

```
# Part-of-Speech Tagging
def tag_and_remove(data_str):
    cleaned_str = ' '
    # noun tags
    nn_tags = ['NN', 'NNP', 'NNP', 'NNPS', 'NNS']
    # adjectives
    jj_tags = ['JJ', 'JJR', 'JJS']
    # verbs
    vb_tags = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    nltk_tags = nn_tags + jj_tags + vb_tags

    # break string into 'words'
    text = data_str.split()

    # tag the text and keep only those with the right tags
    tagged_text = pos_tag(text)
    for tagged_word in tagged_text:
```

```
            if tagged_word[1] in nltk_tags:
                cleaned_str += tagged_word[0] + ' '

    return cleaned_str
```

- lemmatization

```
# lemmatization
def lemmatize(data_str):
    # expects a string
    list_pos = 0
    cleaned_str = ''
    lmtzr = WordNetLemmatizer()
    text = data_str.split()
    tagged_words = pos_tag(text)
    for word in tagged_words:
        if 'v' in word[1].lower():
            lemma = lmtzr.lemmatize(word[0], pos='v')
        else:
            lemma = lmtzr.lemmatize(word[0], pos='n')
        if list_pos == 0:
            cleaned_str = lemma
        else:
            cleaned_str = cleaned_str + ' ' + lemma
        list_pos += 1
    return cleaned_str
```

- setup pyspark udf function

```
# setup pyspark udf function
strip_non_ascii_udf = udf(strip_non_ascii, StringType())
check_blanks_udf = udf(check_blanks, StringType())
check_lang_udf = udf(check_lang, StringType())
fix_abbreviation_udf = udf(fix_abbreviation, StringType())
remove_stops_udf = udf(remove_stops, StringType())
remove_features_udf = udf(remove_features, StringType())
tag_and_remove_udf = udf(tag_and_remove, StringType())
lemmatize_udf = udf(lemmatize, StringType())
```

1. Text processing

- correct the data schema

```
rawdata = rawdata.withColumn('rating', rawdata.rating.cast('float'))

rawdata.printSchema()

root
|-- id: string (nullable = true)
|-- airline: string (nullable = true)
|-- date: string (nullable = true)
|-- location: string (nullable = true)
|-- rating: float (nullable = true)
|-- cabin: string (nullable = true)
```

```
 |-- value: string (nullable = true)
 |-- recommended: string (nullable = true)
 |-- review: string (nullable = true)

from datetime import datetime
from pyspark.sql.functions import col

# https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior
# 21-Jun-14 <----> %d-%b-%y
to_date =  udf (lambda x: datetime.strptime(x, '%d-%b-%y'), DateType())

rawdata = rawdata.withColumn('date', to_date(col('date')))

rawdata.printSchema()

root
 |-- id: string (nullable = true)
 |-- airline: string (nullable = true)
 |-- date: date (nullable = true)
 |-- location: string (nullable = true)
 |-- rating: float (nullable = true)
 |-- cabin: string (nullable = true)
 |-- value: string (nullable = true)
 |-- recommended: string (nullable = true)
 |-- review: string (nullable = true)

rawdata.show(5)

+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows

rawdata = rawdata.withColumn('non_asci', strip_non_ascii_udf(rawdata['review']))


+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows
```

```
rawdata = rawdata.select(raw_cols+['non_asci'])\
                 .withColumn('fixed_abbrev',fix_abbreviation_udf(rawdata['non_asci']))

+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows

 rawdata = rawdata.select(raw_cols+['fixed_abbrev'])\
                  .withColumn('stop_text',remove_stops_udf(rawdata['fixed_abbrev']))

+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows

rawdata = rawdata.select(raw_cols+['stop_text'])\
                 .withColumn('feat_text',remove_features_udf(rawdata['stop_text']))

+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows

 rawdata = rawdata.select(raw_cols+['feat_text'])\
                  .withColumn('tagged_text',tag_and_remove_udf(rawdata['feat_text']))

+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
```

```
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows

 rawdata = rawdata.select(raw_cols+['tagged_text']) \
                  .withColumn('lemm_text',lemmatize_udf(rawdata['tagged_text'])


+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows

 rawdata = rawdata.select(raw_cols+['lemm_text']) \
                  .withColumn("is_blank", check_blanks_udf(rawdata["lemm_text"]))


+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows

from pyspark.sql.functions import monotonically_increasing_id
# Create Unique ID
rawdata = rawdata.withColumn("uid", monotonically_increasing_id())
data = rawdata.filter(rawdata["is_blank"] == "False")

+-----+------------------+----------+--------+------+--------+-----+-----------+------
|   id|           airline|      date|location|rating|   cabin|value|recommended|
+-----+------------------+----------+--------+------+--------+-----+-----------+------
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|         NO|Flight
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|         NO|Flight
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|        YES|I'm Ex
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|        YES|MSP-JF
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|         NO|Worst
+-----+------------------+----------+--------+------+--------+-----+-----------+------
only showing top 5 rows
```

Pipeline for LDA model

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml import Pipeline
```

```
from pyspark.ml.classification import NaiveBayes, RandomForestClassifier
from pyspark.ml.clustering import LDA
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.feature import CountVectorizer

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and n
tokenizer = Tokenizer(inputCol="lemm_text", outputCol="words")
#data = tokenizer.transform(data)
vectorizer = CountVectorizer(inputCol= "words", outputCol="rawFeatures")
idf = IDF(inputCol="rawFeatures", outputCol="features")
#idfModel = idf.fit(data)

lda = LDA(k=20, seed=1, optimizer="em")

pipeline = Pipeline(stages=[tokenizer, vectorizer,idf, lda])


model = pipeline.fit(data)
```

1. Results presentation

- Topics

```
+-----+--------------------+--------------------+
|topic|         termIndices|         termWeights|
+-----+--------------------+--------------------+
|    0|[60, 7, 12, 483, ...|[0.01349507958269...|
|    1|[363, 29, 187, 55...|[0.01247250144447...|
|    2|[46, 107, 672, 27...|[0.01188684264641...|
|    3|[76, 43, 285, 152...|[0.01132638300115...|
|    4|[201, 13, 372, 69...|[0.01337529863256...|
|    5|[122, 103, 181, 4...|[0.00930415977117...|
|    6|[14, 270, 18, 74,...|[0.01253817708163...|
|    7|[111, 36, 341, 10...|[0.01269584954257...|
|    8|[477, 266, 297, 1...|[0.01017486869509...|
|    9|[10, 73, 46, 1, 2...|[0.01050875237546...|
|   10|[57, 29, 411, 10,...|[0.01777350667863...|
|   11|[293, 119, 385, 4...|[0.01280305149305...|
|   12|[116, 218, 256, 1...|[0.01570714218509...|
|   13|[433, 171, 176, 3...|[0.00819684813575...|
|   14|[74, 84, 45, 108,...|[0.01700630002172...|
|   15|[669, 215, 14, 58...|[0.00779310974971...|
|   16|[198, 21, 98, 164...|[0.01030577084202...|
|   17|[96, 29, 569, 444...|[0.01297142577633...|
|   18|[18, 60, 140, 64,...|[0.01306356985169...|
|   19|[33, 178, 95, 2, ...|[0.00907425683229...|
+-----+--------------------+--------------------+
```

- Topic terms

```
from pyspark.sql.types import ArrayType, StringType

def termsIdx2Term(vocabulary):
    def termsIdx2Term(termIndices):
        return [vocabulary[int(index)] for index in termIndices]
    return udf(termsIdx2Term, ArrayType(StringType()))

vectorizerModel = model.stages[1]
vocabList = vectorizerModel.vocabulary
final = ldatopics.withColumn("Terms", termsIdx2Term(vocabList)("termIndices"))
```

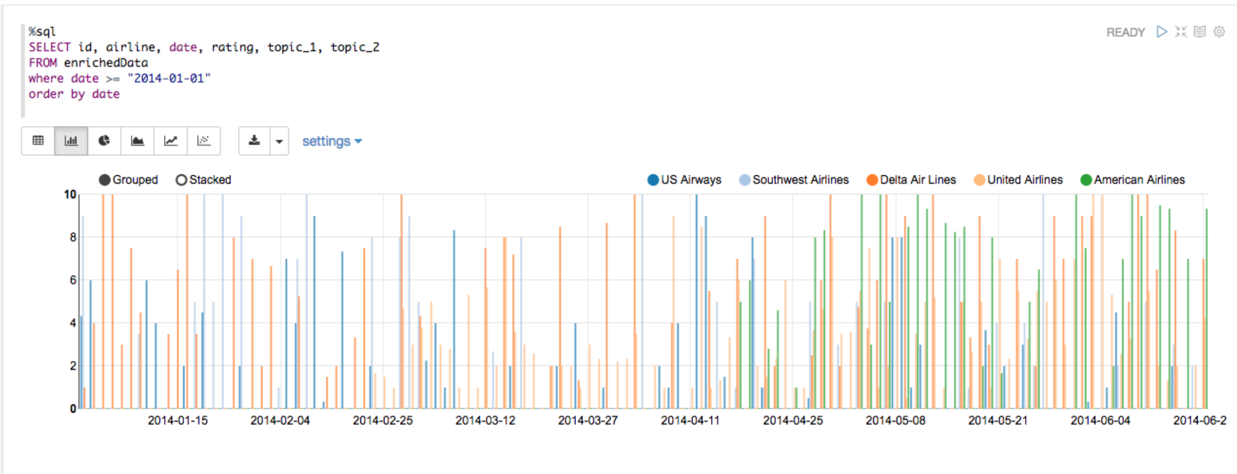
```
+-----+------------------------------------------------+------------------------------
|topic|termIndices                                     |Terms
+-----+------------------------------------------------+------------------------------
|0    |[60, 7, 12, 483, 292, 326, 88, 4, 808, 32]      |[pm, plane, board, kid, online
|1    |[363, 29, 187, 55, 48, 647, 30, 9, 204, 457]    |[dublin, class, th, sit, enter
|2    |[46, 107, 672, 274, 92, 539, 23, 27, 279, 8]    |[economy, sfo, milwaukee, dece
|3    |[76, 43, 285, 152, 102, 34, 300, 113, 24, 31]   |[didn, pay, lose, different, e
|4    |[201, 13, 372, 692, 248, 62, 211, 187, 105, 110]|[houston, crew, heathrow, loui
|5    |[122, 103, 181, 48, 434, 10, 121, 147, 934, 169]|[lhr, serve, screen, entertain
|6    |[14, 270, 18, 74, 70, 37, 16, 450, 3, 20]       |[check, employee, gate, line,
|7    |[111, 36, 341, 10, 320, 528, 844, 19, 195, 524] |[atlanta, first, toilet, delta
|8    |[477, 266, 297, 185, 1, 33, 22, 783, 17, 908]   |[fuel, group, pas, boarding, s
|9    |[10, 73, 46, 1, 248, 302, 213, 659, 48, 228]    |[delta, lax, economy, seat, lc
|10   |[57, 29, 411, 10, 221, 121, 661, 19, 805, 733]  |[business, class, fra, delta,
|11   |[293, 119, 385, 481, 503, 69, 13, 87, 176, 545] |[march, ua, manchester, phx, e
|12   |[116, 218, 256, 156, 639, 20, 365, 18, 22, 136] |[san, clt, francisco, second,
|13   |[433, 171, 176, 339, 429, 575, 10, 26, 474, 796]|[daughter, small, aa, ba, segm
|14   |[74, 84, 45, 108, 342, 111, 315, 87, 52, 4]     |[line, agent, next, hotel, sta
|15   |[669, 215, 14, 58, 561, 59, 125, 179, 93, 5]    |[fit, carry, check, people, ba
|16   |[198, 21, 98, 164, 57, 141, 345, 62, 121, 174]  |[ife, good, nice, much, busine
|17   |[96, 29, 569, 444, 15, 568, 21, 103, 657, 505]  |[phl, class, diego, lady, food
|18   |[18, 60, 140, 64, 47, 40, 31, 35, 2, 123]       |[gate, pm, phoenix, connection
|19   |[33, 178, 95, 2, 9, 284, 42, 4, 89, 31]         |[trip, counter, philadelphia,
+-----+------------------------------------------------+------------------------------
```

- LDA results

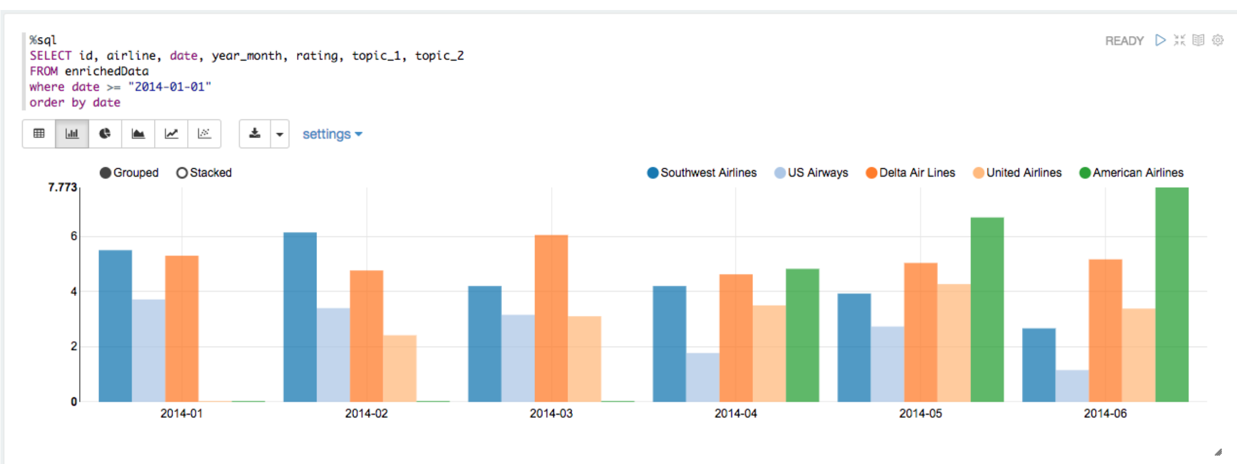
```
+-----+------------------+----------+-----------+------+--------------------+---------
|   id|           airline|      date|      cabin|rating|               words|
+-----+------------------+----------+-----------+------+--------------------+---------
|10551|Southwest Airlines|2013-11-06|   Business|   1.0|[flight, chicago,...|(4695,[0,
|10298|        US Airways|2014-03-31|   Business|   1.0|[flight, manchest...|(4695,[0,
|10564|Southwest Airlines|2013-09-06|    Economy|  10.0|[executive, plati...|(4695,[0,
|10134|   Delta Air Lines|2013-12-10|    Economy|   8.0|[msp, jfk, mxp, r...|(4695,[0,
|10912|   United Airlines|2014-04-07|    Economy|   3.0|[worst, airline, ...|(4695,[0,
|10089|   Delta Air Lines|2014-02-18|    Economy|   2.0|[dl, mia, lax, im...|(4695,[2,
|10385|        US Airways|2013-10-21|    Economy|  10.0|[flew, gla, phl, ...|(4695,[0,
|10249|        US Airways|2014-06-17|    Economy|   1.0|[friend, book, fl...|(4695,[0,
|10289|        US Airways|2014-04-12|    Economy|  10.0|[flew, air, rome,...|(4695,[0,
|10654|Southwest Airlines|2012-07-10|    Economy|   8.0|[lhr, jfk, think,...|(4695,[0,
|10754| American Airlines|2014-05-04|    Economy|  10.0|[san, diego, moli...|(4695,[0,
|10646|Southwest Airlines|2012-08-17|    Economy|   7.0|[toledo, co, stop...|(4695,[0,
|10097|   Delta Air Lines|2014-02-03|First Class|  10.0|[honolulu, la, fi...|(4695,[0,
```

```
|10132|   Delta Air Lines|2013-12-16|    Economy|   7.0|[manchester, uk, ...|(4695,[0,
|10560|Southwest Airlines|2013-09-20|    Economy|   9.0|[first, time, sou...|(4695,[0,
|10579|Southwest Airlines|2013-07-25|    Economy|   0.0|[plane, land, pm,...|(4695,[2,
|10425|        US Airways|2013-08-06|    Economy|   3.0|[airway, bad, pro...|(4695,[2,
|10650|Southwest Airlines|2012-07-27|    Economy|   9.0|[flew, jfk, lhr, ...|(4695,[0,
|10260|        US Airways|2014-06-03|    Economy|   1.0|[february, air, u...|(4695,[0,
|10202|   Delta Air Lines|2013-09-14|    Economy|  10.0|[aug, lhr, jfk, b...|(4695,[1,
+-----+------------------+----------+-----------+------+--------------------+---------
only showing top 20 rows
```

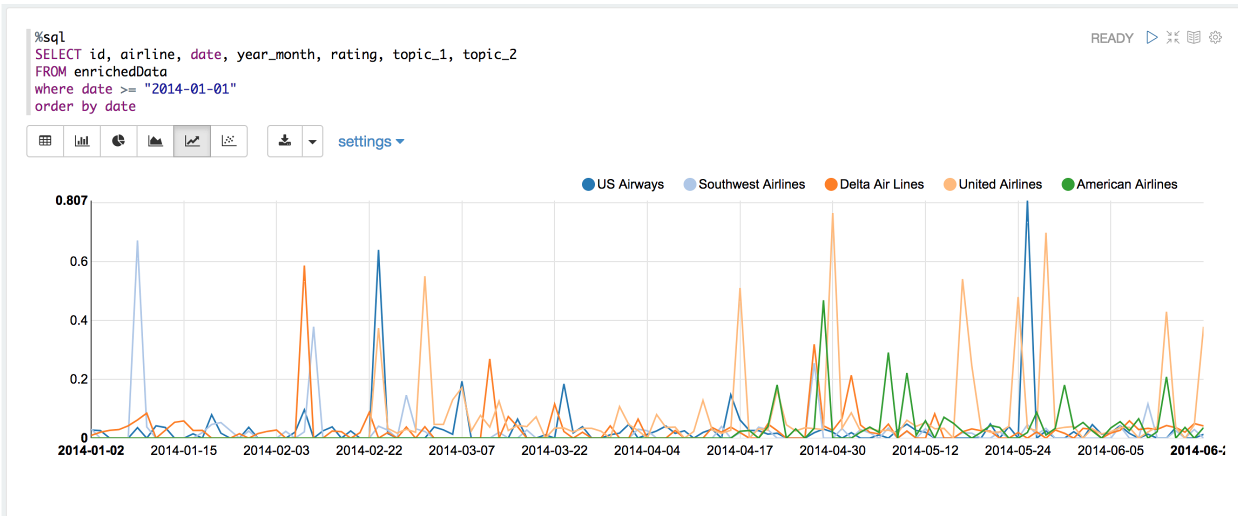
- Average rating and airlines for each day

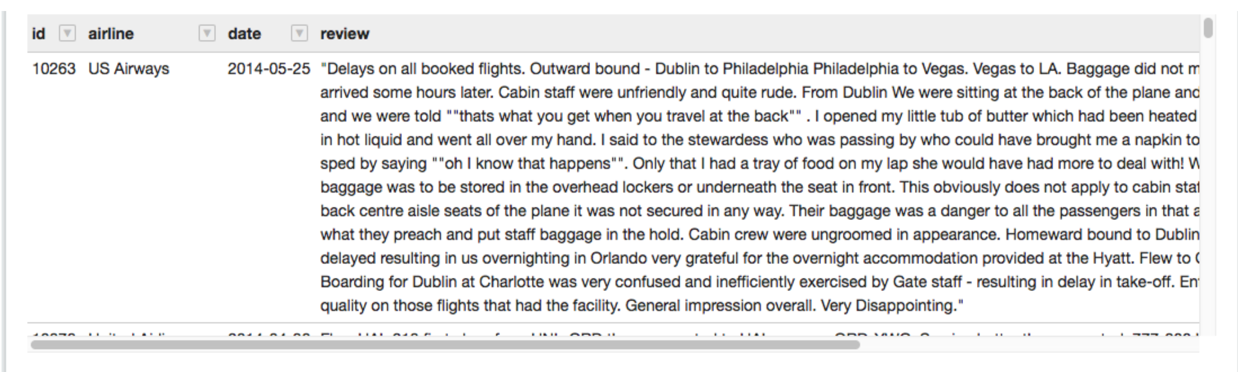


- Average rating and airlines for each month



- Topic 1 corresponding to time line
- reviews (documents) relate to topic 1





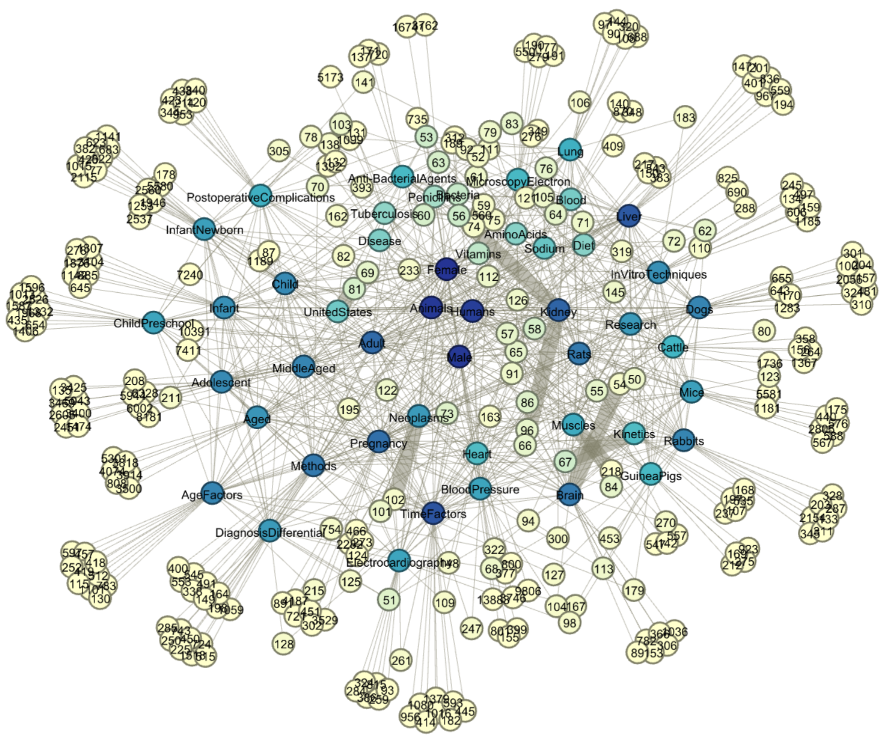
id	airline	date	review
10263	US Airways	2014-05-25	"Delays on all booked flights. Outward bound - Dublin to Philadelphia Philadelphia to Vegas. Vegas to LA. Baggage did not m arrived some hours later. Cabin staff were unfriendly and quite rude. From Dublin We were sitting at the back of the plane and and we were told ""thats what you get when you travel at the back"" . I opened my little tub of butter which had been heated in hot liquid and went all over my hand. I said to the stewardess who was passing by who could have brought me a napkin to sped by saying ""oh I know that happens"". Only that I had a tray of food on my lap she would have had more to deal with! W baggage was to be stored in the overhead lockers or underneath the seat in front. This obviously does not apply to cabin stat back centre aisle seats of the plane it was not secured in any way. Their baggage was a danger to all the passengers in that a what they preach and put staff baggage in the hold. Cabin crew were ungroomed in appearance. Homeward bound to Dublin delayed resulting in us overnighting in Orlando very grateful for the overnight accommodation provided at the Hyatt. Flew to C Boarding for Dublin at Charlotte was very confused and inefficiently exercised by Gate staff - resulting in delay in take-off. En quality on those flights that had the facility. General impression overall. Very Disappointing."

CHAPTER

ELEVEN

SOCIAL NETWORK ANALYSIS

Note: A Touch of Cloth,linked in countless ways. – old Chinese proverb



11.1 Co-occurrence Network

Co-occurrence networks are generally used to provide a graphic visualization of potential relationships between people, organizations, concepts or other entities represented within written material. The generation and visualization of co-occurrence networks has become practical with the advent of electronically stored text amenable to text mining.

11.1.1 Methodology

- Build Corpus C
- Build Document-Term matrix D based on Corpus C
- Compute Term-Document matrix D^T
- Adjacency Matrix $A = D^T \cdot D$

There are four main components in this algorithm in the algorithm: Corpus C, Document-Term matrix D, Term-Document matrix D^T and Adjacency Matrix A. In this demo part, I will show how to build those four main components.

Given that we have three groups of friends, they are

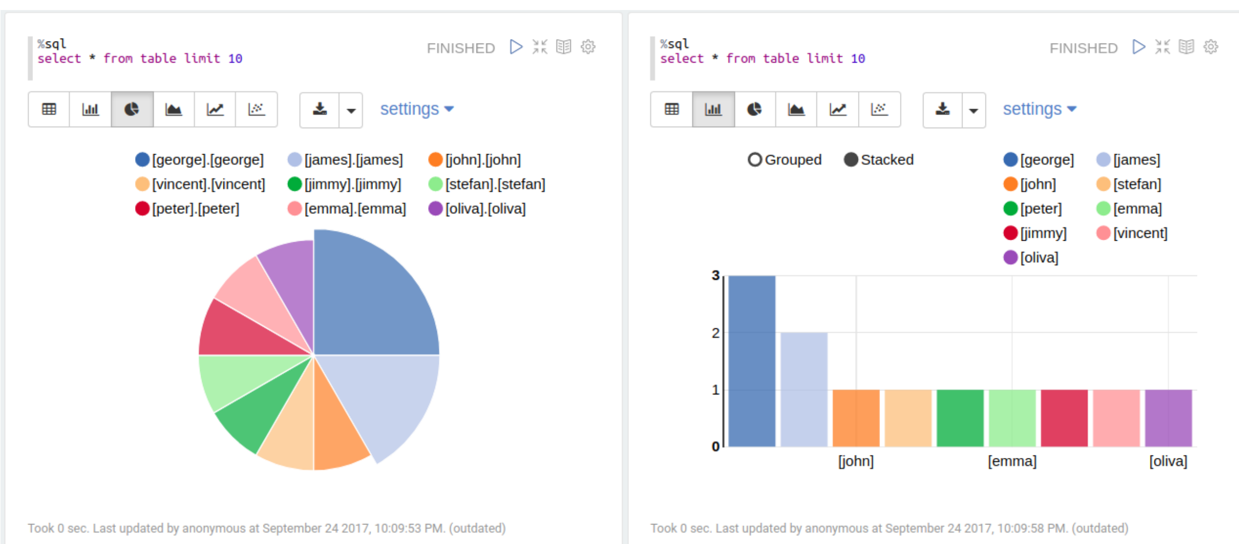
```
+-------------------------------------+
|words                                |
+-------------------------------------+
|[[george] [jimmy] [john] [peter]]    |
|[[vincent] [george] [stefan] [james]]|
|[[emma] [james] [olivia] [george]]   |
+-------------------------------------+
```

1. Corpus C

Then we can build the following corpus based on the unique elements in the given group data:

```
[u'george', u'james', u'jimmy', u'peter', u'stefan', u'vincent', u'olivia', u'john', u
```

The corresponding elements frequency:



2. Document-Term matrix D based on Corpus C (CountVectorizer)

```
from pyspark.ml.feature import CountVectorizer
count_vectorizer_wo = CountVectorizer(inputCol='term', outputCol='features')
```

```
# with total unique vocabulary
countVectorizer_mod_wo = count_vectorizer_wo.fit(df)
countVectorizer_twitter_wo = countVectorizer_mod_wo.transform(df)
# with truncated unique vocabulary (99%)
count_vectorizer = CountVectorizer(vocabSize=48,inputCol='term',outputCol='features')
countVectorizer_mod = count_vectorizer.fit(df)
countVectorizer_twitter = countVectorizer_mod.transform(df)
```

```
+-------------------------------+
|features                       |
+-------------------------------+
|(9,[0,2,3,7],[1.0,1.0,1.0,1.0])|
|(9,[0,1,4,5],[1.0,1.0,1.0,1.0])|
|(9,[0,1,6,8],[1.0,1.0,1.0,1.0])|
+-------------------------------+
```

- Term-Document matrix D^T

  RDD:

  ```
  [array([ 1.,  1.,  1.]), array([ 0.,  1.,  1.]), array([ 1.,  0.,  0.]),
   array([ 1.,  0.,  0.]), array([ 0.,  1.,  0.]), array([ 0.,  1.,  0.]),
   array([ 0.,  0.,  1.]), array([ 1.,  0.,  0.]), array([ 0.,  0.,  1.])]
  ```

  Matrix:

  ```
  array([[ 1.,  1.,  1.],
         [ 0.,  1.,  1.],
         [ 1.,  0.,  0.],
         [ 1.,  0.,  0.],
         [ 0.,  1.,  0.],
         [ 0.,  1.,  0.],
         [ 0.,  0.,  1.],
         [ 1.,  0.,  0.],
         [ 0.,  0.,  1.]])
  ```

3. Adjacency Matrix $A = D^T \cdot D$

   RDD:

   ```
   [array([ 1.,  1.,  1.]), array([ 0.,  1.,  1.]), array([ 1.,  0.,  0.]),
    array([ 1.,  0.,  0.]), array([ 0.,  1.,  0.]), array([ 0.,  1.,  0.]),
    array([ 0.,  0.,  1.]), array([ 1.,  0.,  0.]), array([ 0.,  0.,  1.])]
   ```

   Matrix:

   ```
   array([[ 3.,  2.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
          [ 2.,  2.,  0.,  0.,  1.,  1.,  1.,  0.,  1.],
          [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
          [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
          [ 1.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.],
          [ 1.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.],
          [ 1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.],
          [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
          [ 1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.]])
   ```

11.1.2 Coding Puzzle from my interview

- Problem

The attached utf-8 encoded text file contains the tags associated with an online biomedical scientific article formatted as follows (size: 100000). Each Scientific article is represented by a line in the file delimited by carriage return.

```
+--------------------+
|               words|
+--------------------+
|[ACTH Syndrome, E...|
|[Antibody Formati...|
|[Adaptation, Phys...|
|[Aerosol Propella...|
+--------------------+
only showing top 4 rows
```

Write a program that, using this file as input, produces a list of pairs of tags which appear TOGETHER in any order and position in at least fifty different Scientific articles. For example, in the above sample, [Female] and [Humans] appear together twice, but every other pair appears only once. Your program should output the pair list to stdout in the same form as the input (eg tag 1, tag 2n).

- My solution

  The corresponding words frequency:

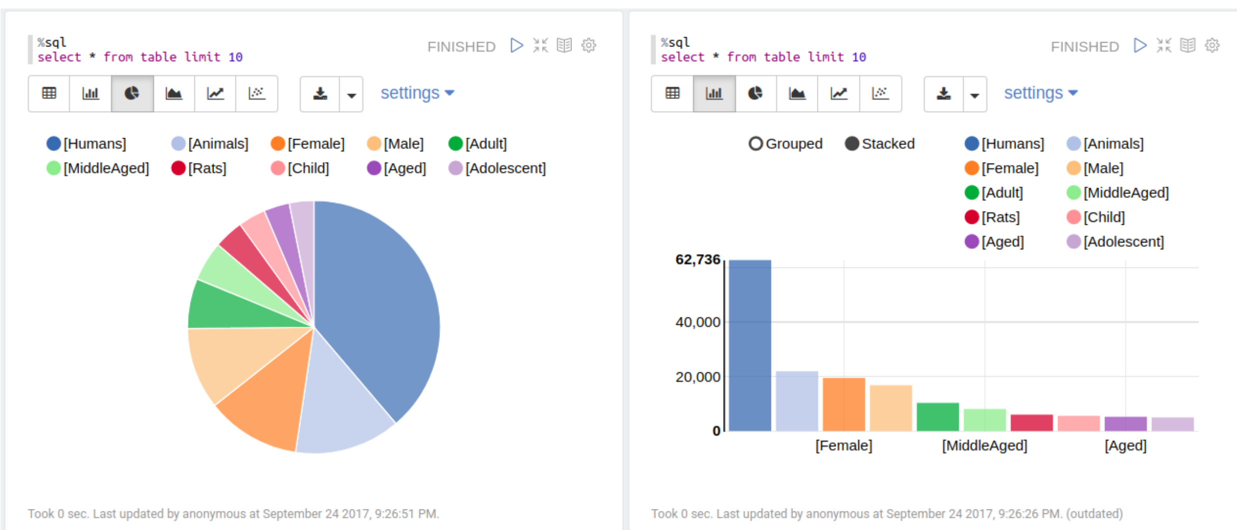


Figure 11.1: Word frequency

Output:

```
+----------+------+-------+
|    term.x|term.y|   freq|
+----------+------+-------+
|    Female|Humans|16741.0|
```

```
|      Male|Humans|13883.0|
|     Adult|Humans|10391.0|
|      Male|Female| 9806.0|
|MiddleAged|Humans| 8181.0|
|     Adult|Female| 7411.0|
|     Adult|  Male| 7240.0|
|MiddleAged|  Male| 6328.0|
|MiddleAged|Female| 6002.0|
|MiddleAged| Adult| 5944.0|
+----------+------+-------+
only showing top 10 rows
```

The corresponding Co-occurrence network:

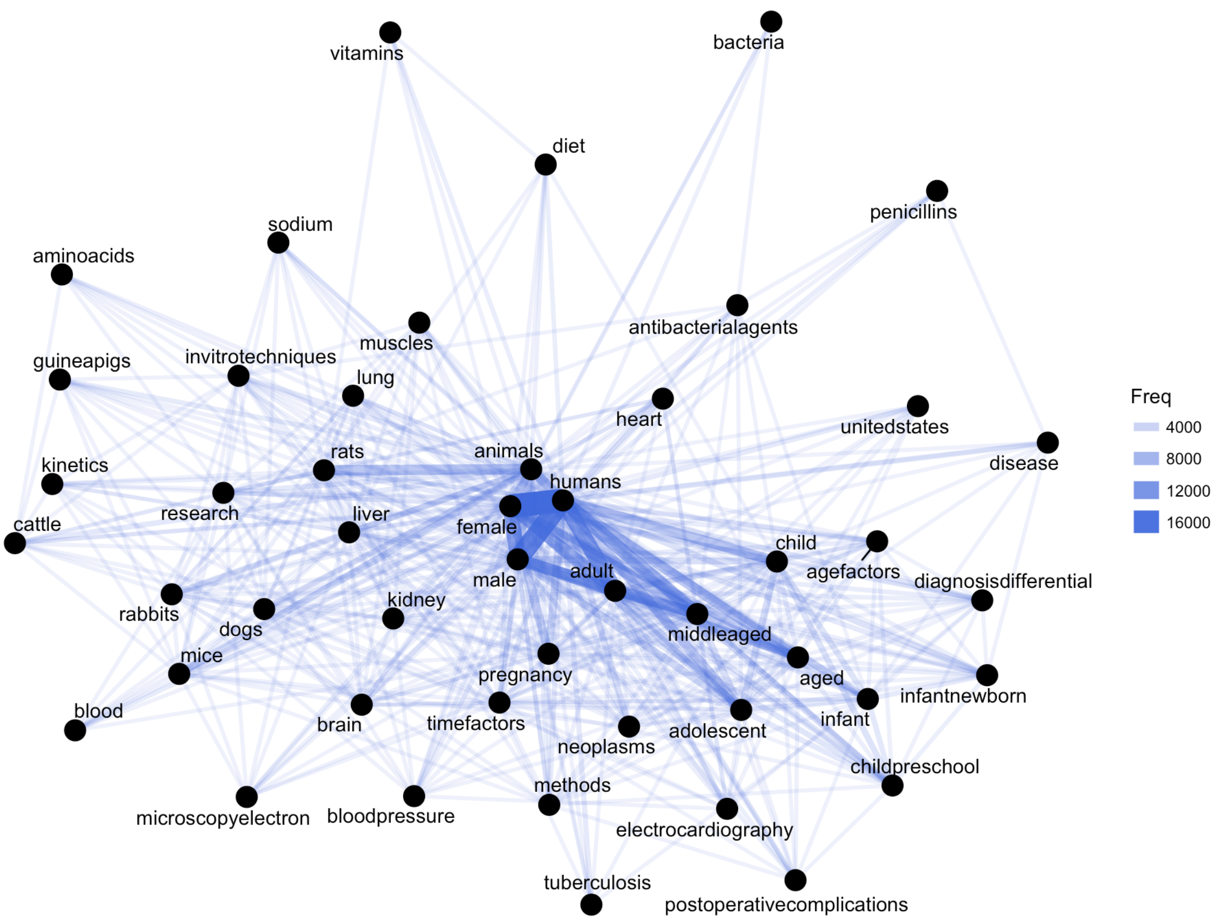


Figure 11.2: Co-occurrence network

Then you will get Figure *Co-occurrence network*

11.2 Correlation Network

CHAPTER

TWELVE

NEURAL NETWORK

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

12.1 Feedforward Neural Network

12.1.1 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.

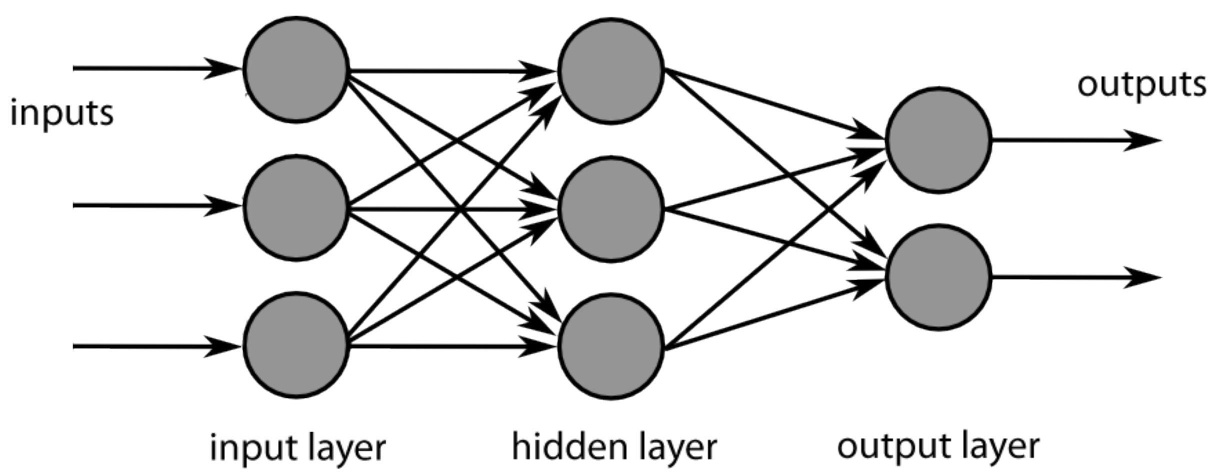


Figure 12.1: MultiLayer Neural Network

12.1.2 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|alcohcl|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

THIRTEEN

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

[Kirillov2016] Anton Kirillov. Apache Spark: core concepts, architecture and internals. http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/

INDEX