Learning Apache Spark with Python

Wenqiang Feng

April 04, 2019

CONTENTS

1 Preface **3**
- 1.1 About . . . 3
- 1.2 Motivation for this tutorial . . . 4
- 1.3 Copyright notice and license info . . . 4
- 1.4 Acknowledgement . . . 4
- 1.5 Feedback and suggestions . . . 5

2 Why Spark with Python ? **7**
- 2.1 Why Spark? . . . 7
- 2.2 Why Spark with Python (PySpark)? . . . 8

3 Configure Running Platform **11**
- 3.1 Run on Databricks Community Cloud . . . 11
- 3.2 Configure Spark on Mac and Ubuntu . . . 16
- 3.3 Configure Spark on Windows . . . 19
- 3.4 PySpark With Text Editor or IDE . . . 19
- 3.5 PySparkling Water: Spark + H2O . . . 20
- 3.6 Set up Spark on Cloud . . . 24
- 3.7 Demo Code in this Section . . . 24

4 An Introduction to Apache Spark **27**
- 4.1 Core Concepts . . . 27
- 4.2 Spark Components . . . 27
- 4.3 Architecture . . . 30
- 4.4 How Spark Works? . . . 30

5 Programming with RDDs **31**
- 5.1 Create RDD . . . 31
- 5.2 Spark Operations . . . 35
- 5.3 `rdd.DataFrame` vs `pd.DataFrame` . . . 37

6 Statistics Preliminary **53**
- 6.1 Notations . . . 53
- 6.2 Measurement Formula . . . 53
- 6.3 Statistical Tests . . . 54

7 Data Exploration **55**
7.1 Univariate Analysis . . . 55
7.2 Multivariate Analysis . . . 68

8 Regression **75**
8.1 Linear Regression . . . 75
8.2 Generalized linear regression . . . 85
8.3 Decision tree Regression . . . 92
8.4 Random Forest Regression . . . 99
8.5 Gradient-boosted tree regression . . . 105

9 Regularization **113**
9.1 Ordinary least squares regression . . . 113
9.2 Ridge regression . . . 113
9.3 Least Absolute Shrinkage and Selection Operator (LASSO) . . . 114
9.4 Elastic net . . . 114

10 Classification **115**
10.1 Binomial logistic regression . . . 115
10.2 Multinomial logistic regression . . . 126
10.3 Decision tree Classification . . . 137
10.4 Random forest Classification . . . 146
10.5 Gradient-boosted tree Classification . . . 156
10.6 XGBoost: Gradient-boosted tree Classification . . . 156
10.7 Naive Bayes Classification . . . 158

11 Clustering **171**
11.1 K-Means Model . . . 171

12 RFM Analysis **181**
12.1 RFM Analysis Methodology . . . 182
12.2 Demo . . . 184
12.3 Extension . . . 190

13 Text Mining **197**
13.1 Text Collection . . . 197
13.2 Text Preprocessing . . . 205
13.3 Text Classification . . . 207
13.4 Sentiment analysis . . . 214
13.5 N-grams and Correlations . . . 221
13.6 Topic Model: Latent Dirichlet Allocation . . . 221

14 Social Network Analysis **239**
14.1 Introduction . . . 239
14.2 Co-occurrence Network . . . 239
14.3 Appendix: matrix multiplication in PySpark . . . 243
14.4 Correlation Network . . . 246

15 ALS: Stock Portfolio Recommendations **247**

15.1 Recommender systems . . . 248
15.2 Alternating Least Squares . . . 249
15.3 Demo . . . 249

16 Monte Carlo Simulation **257**
16.1 Simulating Casino Win . . . 257
16.2 Simulating a Random Walk . . . 259

17 Markov Chain Monte Carlo **269**
17.1 Metropolis algorithm . . . 269
17.2 A Toy Example of Metropolis . . . 270
17.3 Demos . . . 271

18 Neural Network **279**
18.1 Feedforward Neural Network . . . 279

19 My PySpark Package **283**
19.1 Hierarchical Structure . . . 283
19.2 Set Up . . . 284
19.3 ReadMe . . . 284

20 My Cheat Sheet **287**

21 PySpark API **291**
21.1 Stat API . . . 291
21.2 Regression API . . . 297
21.3 Classification API . . . 318
21.4 Clustering API . . . 341
21.5 Recommendation API . . . 357
21.6 Pipeline API . . . 362
21.7 Tuning API . . . 364
21.8 Evaluation API . . . 369

22 Main Reference **375**

Bibliography **377**

Python Module Index **379**

Index **381**

Welcome to my **Learning Apache Spark with Python** note! In this note, you will learn a wide array of concepts about **PySpark** in Data Mining, Text Mining, Machine Learning and Deep Learning. The PDF version can be downloaded from HERE.

CHAPTER

ONE

PREFACE

1.1 About

1.1.1 About this note

This is a shared repository for Learning Apache Spark Notes. The PDF version can be downloaded from HERE. The first version was posted on Github in ChenFeng ([Feng2017]). This shared repository mainly contains the self-learning and self-teaching notes from Wenqiang during his IMA Data Science Fellowship. The reader is referred to the repository https://github.com/runawayhorse001/LearningApacheSpark for more details about the `dataset` and the `.ipynb` files.

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: von198@gmail.com

- **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, UTK 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 Copyright notice and license info

This Learning Apache Spark with Python PDF file is supposed to be a free and living document, which is why its source is available online at https://runawayhorse001.github.io/LearningApacheSpark/pyspark.pdf. But this document is licensed according to both MIT License and Creative Commons Attribution-NonCommercial 2.0 Generic (CC BY-NC 2.0) License.

When you plan to use, copy, modify, merge, publish, distribute or sublicense, Please see the terms of those licenses for more details and give the corresponding credits to the author.

1.4 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.

A special thank you goes to Dr. Haiping Lu, Lecturer in Machine Learning at Department of Computer Science, University of Sheffield, for recommending and heavily using my tutorial in his teaching class and for the valuable suggestions.

1.5 Feedback and suggestions

Your comments and suggestions are highly appreciated. I am more than happy to receive corrections, suggestions or feedbacks through email (von198@gmail.com) for improvements.

CHAPTER

TWO

WHY SPARK WITH PYTHON ?

Chinese proverb

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 from Apache Spark™ official website are good enough to convince you to use Spark.

1. Speed

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

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

Fig. 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.

Fig. 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.

2.2 Why Spark with Python (PySpark)?

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

Fig. 3: The Spark platform

Fig. 4: KDnuggets Analytics/Data Science 2017 Software Poll from kdnuggets.

CHAPTER

THREE

CONFIGURE RUNNING PLATFORM

Chinese proverb

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.

File Edit View History Tools People Help
Databricks
https://community.cloud.databricks.com/?o=4622560542654492
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments
Upgrade
Community Edition (2.45)
Welcome to databricks
Featured Notebooks
Introduction to Apache Spark on Databricks
Databricks for Data Scientists
Introduction to Structured Streaming
New
Notebook
Job
Cluster
Table
Library
Documentation
Databricks Guide
Python, R, Scala, SQL
Importing Data
Open Recent
Databricks for Data Scientists
linearRegression
What's new?
Automatic termination of clusters
Autoscaling local storage with EBS volumes (beta)
Latest release notes
Send Feedback
Home
Workspace
Recent
Tables
Clusters
Jobs
Search

File Edit View History Tools People Help
Create Cluster - Dat
https://community.cloud.databricks.com/?o=4622560542654492#create/cluster
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments
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
Home
Workspace
Recent
Tables
Clusters
Jobs
Search

File Edit View History Tools People Help
Create Table - Data
https://community.cloud.databricks.com/?o=4622560542654492#create/table
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments
Create Table
Search Tables
advertising_csv
crab
praedicat
raw_classified_txt
Create Table
Data Import
Data Source
File
File(s)
Drop file or click here to upload
Send Feedback

File Edit View History Tools People Help
Create Table - Data
https://community.cloud.databricks.com/?o=4622560542654492#create/table
Apps Bookmarks job Foreign National The FORTRAN Pr Fortran Tutorials Using the cluster Attachments Renegade Army
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
Previewing table
Table name
heart
File type
CSV
Column Delimiter
First row is header
Create Table
_c0 Age Sex ChestPain RestBP
STRING STRING STRING STRING STRING
1 63 1 typical 145
2 67 1 asymptomatic 160
Send Feedback

Databricks
https://community.cloud.databricks.com/?o=4622560542654492#
Upgrade
Community Edition (2.45)
Create Notebook
Name
test
Language
Python
Cluster
MLmachine (6 GB, Failed to atta
Cancel
Create
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

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

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:

a. Open Ubuntu Software Center

b. Search for python

c. 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.

a. Download: You can get the Pre-built Apache Spark™ from Download Apache Spark™.

b. Unpack: Unpack the Apache Spark™ to the path where you want to install the Spark.

c. 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

a. **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
```

b. **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.

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 PySparkling Water: Spark + H2O

1. Download `Sparkling Water` from: https://s3.amazonaws.com/h2o-release/sparkling-water/rel-2.4/5/index.html
2. Test PySparking

localhost:8080/#/notebook/2CTAHCNZD - Google Chrome
Zeppelin
Notebook
Job
Search your Notes
anonymous
test
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);
Took 0 sec. Last updated by anonymous at September 24 2017, 4:03:16 PM. (outdated)
%spark.pyspark
df.show(4)
FINISHED
only showing top 4 rows
Took 1 sec. Last updated by anonymous at September 24 2017, 4:14:43 PM.
%spark.pyspark
df.registerTempTable("bank")
Took 0 sec. Last updated by anonymous at September 24 2017, 4:03:32 PM. (outdated)
%sql
select age, count(1) value
from bank
where age < ${maxAge=20}
group by age
order by age
maxAge
30
settings
Took 0 sec. Last updated by anonymous at September 24 2017, 4:11:05 PM.
%sql
select age, count(1) value
from bank
where age < 30
group by age
order by age
Took 1 sec. Last updated by anonymous at September 24 2017, 4:03:35 PM.
%sql
select age, count(1) value
from bank
where marital="${marital=single,single|divorced|married}"
group by age
marital
single
Stacked
Stream
Expanded
value
Took 0 sec. Last updated by anonymous at September 24 2017, 4:07:39 PM.

~/Spark/Code/test_pyspark.py - Sublime Text
test_pyspark.py
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.cs
df.show(5)
df.printSchema()
Line 16, Column 38
Tab Size: 4
Python
feng@feng-ThinkPad: ~/Spark/Code
to bind to another address
17/05/21 19:12:47 WARN Utils: Service 'SparkUI' could not bin
d on port 4040. Attempting port 4041.
17/05/21 19:12:47 WARN Utils: Service 'SparkUI' could not bin
d on port 4041. Attempting port 4042.
|_c0| TV|Radio|Newspaper|Sales|
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
only showing top 5 rows
root
-- _c0: integer (nullable = true)
-- TV: double (nullable = true)
-- Radio: double (nullable = true)
-- Newspaper: double (nullable = true)
-- Sales: double (nullable = true)
feng@feng-ThinkPad:~/Spark/Code$

PyDev - Eclipse
Quick Access
Java
PyDev
PyDev Package Explorer
IMA
Leetcode
math563
PySpark
testPy
pytho
redhat
stat537
STAT57
Theano
dataMi
matplo
tensorf
New
Go Into
Copy
Paste
Delete
Move...
Rename...
Remove from Context
Import...
Export...
Refresh
F5
Close Project
Close Unrelated Projects
Run As
Debug As
Team
Compare With
Restore from Local History...
PyDev
Configure
Properties
Alt+Enter
project1
testPySpark
conf
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()
Console
Expressions
PyUnit
No consoles to display at this time.
1 item selected

Properties for
type filter text
Resource
Builders
Project References
PyDev - Interpreter/(
PyDev - PYTHONPAT
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

```
unzip sparkling-water-2.4.5.zip
cd  ~/sparkling-water-2.4.5/bin
./pysparkling
```

If you have a correct setup for PySpark, then you will get the following results:

```
Using Spark defined in the SPARK_HOME=/Users/dt216661/spark environmental␣
↪property

Python 3.7.1 (default, Dec 14 2018, 13:28:58)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
2019-02-15 14:08:30 WARN  NativeCodeLoader:62 - Unable to load native-hadoop␣
↪library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
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).
2019-02-15 14:08:31 WARN  Utils:66 - Service 'SparkUI' could not bind on port␣
↪4040. Attempting port 4041.
2019-02-15 14:08:31 WARN  Utils:66 - Service 'SparkUI' could not bind on port␣
↪4041. Attempting port 4042.
17/08/30 13:30:12 WARN NativeCodeLoader: Unable to load native-hadoop
```

(continues on next page)

(continued from previous page)

```
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 2.4.0
      /_/

Using Python version 3.7.1 (default, Dec 14 2018 13:28:58)
SparkSession available as 'spark'.
```

3. Setup `pysparkling` with Jupyter notebook

Add the following alias to your `bashrc` (Linux systems) or `bash_profile` (Mac system)

```
alias sparkling="PYSPARK_DRIVER_PYTHON="ipython" PYSPARK_DRIVER_PYTHON_OPTS=
↪  "notebook" /~/~/sparkling-water-2.4.5/bin/pysparkling"
```

4. Open `pysparkling` in terminal

```
sparkling
```

3.6 Set up Spark on Cloud

Following 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.

3.7 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()
```

(continues on next page)

(continued from previous page)

```
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

Chinese proverb

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

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

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

2. Executors

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

3. Cluster Manager

- Mesos
- YARN
- Spark Standalone

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

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

  - backend interface for scheduling systems that allows plugging in different implementations(Mesos, YARN, Standalone, local)

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

4.3 Architecture

4.4 How Spark Works?

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

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

CHAPTER

FIVE

PROGRAMMING WITH RDDS

Chinese proverb

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()

+----+----+----+-----+
```

(continues on next page)

(continued from previous page)

```
|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

   a. **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.

   b. **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") \
```

(continues on next page)

(continued from previous page)

```
            .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.

C. **Read dataset from HDFS**

```
from pyspark.conf import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import HiveContext
```

(continues on next page)

(continued from previous page)

```
sc= SparkContext('local','example')
hc = HiveContext(sc)
tf1 = sc.textFile("hdfs://cdhstltest/user/data/demo.CSV")
print(tf1.first())

hc.sql("use intg_cme_w")
spf = hc.sql("SELECT * FROM spf LIMIT 100")
print(spf.show(5))
```

5.2 Spark Operations

Warning: All the figures below are from Jeffrey Thompson. The interested reader is referred to pyspark pictures

There are two main types of Spark operations: Transformations and Actions [Karau2015].

Note: Some people defined three types of operations: Transformations, Actions and Shuffles.

5.2.1 Spark Transformations

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

5.2.2 Spark Actions

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

5.3 rdd.DataFrame vs pd.DataFrame

5.3.1 Create DataFrame

1. From List

```
my_list = [['a', 1, 2], ['b', 2, 3],['c', 3, 4]]
col_name = ['A', 'B', 'C']
```

:: Python Code:

```
# caution for the columns=
pd.DataFrame(my_list,columns= col_name)
#
spark.createDataFrame(my_list, col_name).show()
```

:: Comparison:

```
                    +---+---+---+
                    |  A|  B|  C|
   A  B  C          +---+---+---+
0  a  1  2          |  a|  1|  2|
1  b  2  3          |  b|  2|  3|
2  c  3  4          |  c|  3|  4|
                    +---+---+---+
```

Attention: Pay attentation to the parameter `columns=` in `pd.DataFrame`. Since the default value will make the list as rows.

:: Python Code:

```
# caution for the columns=
pd.DataFrame(my_list, columns= col_name)
#
pd.DataFrame(my_list, col_name)
```

:: Comparison:

```
   A  B  C              0  1  2
0  a  1  2          A   a  1  2
1  b  2  3          B   b  2  3
2  c  3  4          C   c  3  4
```

2. From Dict

```
d = {'A': [0, 1, 0],
     'B': [1, 0, 1],
     'C': [1, 0, 0]}
```

:: Python Code:

```
pd.DataFrame(d)for
# Tedious for PySpark
spark.createDataFrame(np.array(list(d.values())).T.tolist(),list(d.keys())).
↪show()
```

:: Comparison:

```
                    +---+---+---+
                    |  A|  B|  C|
   A  B  C          +---+---+---+
0  0  1  1          |  0|  1|  1|
1  1  0  0          |  1|  0|  0|
2  0  1  0          |  0|  1|  0|
                    +---+---+---+
```

5.3.2 Load DataFrame

1. From DataBase

Most of time, you need to share your code with your colleagues or release your code for Code Review or Quality assurance(QA). You will definitely do not want to have your `User Information` in the code. So you can save them in login.txt:

```
runawayhorse001
PythonTips
```

and use the following code to import your `User Information`:

```
#User Information
try:
    login = pd.read_csv(r'login.txt', header=None)
    user = login[0][0]
    pw = login[0][1]
    print('User information is ready!')
except:
    print('Login information is not available!!!')

#Database information
host = '##.###.###.##'
db_name = 'db_name'
table_name = 'table_name'
```

:: Comparison:

```
conn = psycopg2.connect(host=host, database=db_name, user=user, password=pw)
cur = conn.cursor()

sql = """
      select *
      from {table_name}
      """.format(table_name=table_name)
dp = pd.read_sql(sql, conn)
```

```
# connect to database
url = 'jdbc:postgresql://'+host+':5432/'+db_name+'?user='+user+'&password='+pw
properties ={'driver': 'org.postgresql.Driver', 'password': pw,'user': user}
ds = spark.read.jdbc(url=url, table=table_name, properties=properties)
```

Attention: Reading tables from Database with PySpark 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.

2. From `.csv`

:: Comparison:

```
# pd.DataFrame dp: DataFrame pandas
dp = pd.read_csv('Advertising.csv')
#rdd.DataFrame. dp: DataFrame spark
ds = spark.read.csv(path='Advertising.csv',
#                 sep=',',
#                 encoding='UTF-8',
#                 comment=None,
                header=True,
                inferSchema=True)
```

3. From `.json`

Data from: http://api.luftdaten.info/static/v1/data.json

```
dp = pd.read_json("data/data.json")
ds = spark.read.json('data/data.json')
```

:: Python Code:

```
dp[['id','timestamp']].head(4)
#
ds[['id','timestamp']].show(4)
```

:: Comparison:

```
                                                    +----------+-----------------
↪-+
                                                    |        id|          ␣
↪timestamp|
            id  timestamp                           +----------+-----------------
↪-+
0   2994551481  2019-02-28 17:23:52                 |2994551481|2019-02-28␣
↪17:23:52|
1   2994551482  2019-02-28 17:23:52                 |2994551482|2019-02-28␣
↪17:23:52|
2   2994551483  2019-02-28 17:23:52                 |2994551483|2019-02-28␣
↪17:23:52|
3   2994551484  2019-02-28 17:23:52                 |2994551484|2019-02-28␣
↪17:23:52|
                                                    +----------+-----------------
↪-+
                                                    only showing top 4 rows
```

5.3.3 First n Rows

:: Python Code:

```
dp.head(4)
#
ds.show(4)
```

:: Comparison:

```
                                        +-----+-----+---------+-----+
                                        |   TV|Radio|Newspaper|Sales|
      TV  Radio  Newspaper  Sales       +-----+-----+---------+-----+
0  230.1   37.8       69.2   22.1       |230.1| 37.8|     69.2| 22.1|
1   44.5   39.3       45.1   10.4       | 44.5| 39.3|     45.1| 10.4|
2   17.2   45.9       69.3    9.3       | 17.2| 45.9|     69.3|  9.3|
3  151.5   41.3       58.5   18.5       |151.5| 41.3|     58.5| 18.5|
                                        +-----+-----+---------+-----+
                                        only showing top 4 rows
```

5.3.4 Column Names

:: Python Code:

```
dp.columns
#
ds.columns
```

:: Comparison:

```
Index(['TV', 'Radio', 'Newspaper', 'Sales'], dtype='object')
['TV', 'Radio', 'Newspaper', 'Sales']
```

5.3.5 Data types

:: Python Code:

```
dp.dtypes
#
ds.dtypes
```

:: Comparison:

```
TV           float64                    [('TV', 'double'),
Radio        float64                     ('Radio', 'double'),
Newspaper    float64                     ('Newspaper', 'double'),
Sales        float64                     ('Sales', 'double')]
dtype: object
```

5.3.6 Fill Null

```
my_list = [['a', 1, None], ['b', 2, 3],['c', 3, 4]]
dp = pd.DataFrame(my_list,columns=['A', 'B', 'C'])
ds = spark.createDataFrame(my_list, ['A', 'B', 'C'])
#
dp.head()
ds.show()
```

:: Comparison:

```
                                        +------+---+----+
                                        |     A|  B|   C|
        A  B    C                       +------+---+----+
0    male  1  NaN                       |  male|  1|null|
1  female  2  3.0                       |female|  2|   3|
2    male  3  4.0                       |  male|  3|   4|
                                        +------+---+----+
```

:: Python Code:

```
dp.fillna(-99)
#
ds.fillna(-99).show()
```

:: Comparison:

```
                                        +------+---+----+
                                        |     A|  B|   C|
```

(continues on next page)

(continued from previous page)

```
        A  B    C                          +------+---+----+
0    male  1  -99                          |  male|  1| -99|
1  female  2  3.0                          |female|  2|   3|
2    male  3  4.0                          |  male|  3|   4|
                                           +------+---+----+
```

5.3.7 Replace Values

:: Python Code:

```
# caution: you need to chose specific col
dp.A.replace(['male', 'female'],[1, 0], inplace=True)
dp
#caution: Mixed type replacements are not supported
ds.na.replace(['male','female'],['1','0']).show()
```

:: Comparison:

```
                                   +---+---+----+
                                   |  A|  B|   C|
   A  B    C                       +---+---+----+
0  1  1  NaN                       |  1|  1|null|
1  0  2  3.0                       |  0|  2|   3|
2  1  3  4.0                       |  1|  3|   4|
                                   +---+---+----+
```

5.3.8 Rename Columns

1. Rename all columns

:: Python Code:

```
dp.columns = ['a','b','c','d']
dp.head(4)
#
ds.toDF('a','b','c','d').show(4)
```

:: Comparison:

```
                                                  +-----+----+----+----+
                                                  |    a|   b|   c|   d|
       a     b     c     d                        +-----+----+----+----+
0  230.1  37.8  69.2  22.1                        |230.1|37.8|69.2|22.1|
1   44.5  39.3  45.1  10.4                        | 44.5|39.3|45.1|10.4|
2   17.2  45.9  69.3   9.3                        | 17.2|45.9|69.3| 9.3|
3  151.5  41.3  58.5  18.5                        |151.5|41.3|58.5|18.5|
                                                  +-----+----+----+----+
                                                  only showing top 4 rows
```

2. Rename one or more columns

```
mapping = {'Newspaper':'C','Sales':'D'}
```

:: Python Code:

```
dp.rename(columns=mapping).head(4)
#
new_names = [mapping.get(col,col) for col in ds.columns]
ds.toDF(*new_names).show(4)
```

:: Comparison:

```
      TV  Radio     C     D
0  230.1   37.8  69.2  22.1
1   44.5   39.3  45.1  10.4
2   17.2   45.9  69.3   9.3
3  151.5   41.3  58.5  18.5
```

```
+-----+-----+----+----+
|   TV|Radio|   C|   D|
+-----+-----+----+----+
|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|
+-----+-----+----+----+
only showing top 4 rows
```

Note: You can also use `withColumnRenamed` to rename one column in PySpark.

:: Python Code:

```
ds.withColumnRenamed('Newspaper','Paper').show(4
```

:: Comparison:

```
+-----+-----+-----+-----+
|   TV|Radio|Paper|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|
+-----+-----+-----+-----+
only showing top 4 rows
```

5.3.9 Drop Columns

```
drop_name = ['Newspaper','Sales']
```

:: Python Code:

```
dp.drop(drop_name,axis=1).head(4)
#
ds.drop(*drop_name).show(4)
```

:: Comparison:

```
      TV  Radio
0  230.1   37.8
1   44.5   39.3
2   17.2   45.9
3  151.5   41.3
```

```
+-----+-----+
|   TV|Radio|
+-----+-----+
|230.1| 37.8|
| 44.5| 39.3|
| 17.2| 45.9|
|151.5| 41.3|
+-----+-----+
only showing top 4 rows
```

5.3.10 Filter

```
dp = pd.read_csv('Advertising.csv')
#
ds = spark.read.csv(path='Advertising.csv',
                    header=True,
                    inferSchema=True)
```

:: Python Code:

```
dp[dp.Newspaper<20].head(4)
#
ds[ds.Newspaper<20].show(4)
```

:: Comparison:

```
       TV  Radio  Newspaper  Sales
7   120.2   19.6       11.6   13.2
8     8.6    2.1        1.0    4.8
11  214.7   24.0        4.0   17.4
13   97.5    7.6        7.2    9.7
```

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|120.2| 19.6|     11.6| 13.2|
|  8.6|  2.1|      1.0|  4.8|
|214.7| 24.0|      4.0| 17.4|
| 97.5|  7.6|      7.2|  9.7|
+-----+-----+---------+-----+
only showing top 4 rows
```

:: Python Code:

```
dp[(dp.Newspaper<20)&(dp.TV>100)].head(4)
#
ds[(ds.Newspaper<20)&(ds.TV>100)].show(4)
```

:: Comparison:

```
                                                  +-----+-----+---------+-----+
                                                  |   TV|Radio|Newspaper|Sales|
       TV  Radio  Newspaper  Sales                +-----+-----+---------+-----+
7   120.2   19.6       11.6   13.2                |120.2| 19.6|     11.6| 13.2|
11  214.7   24.0        4.0   17.4                |214.7| 24.0|      4.0| 17.4|
19  147.3   23.9       19.1   14.6                |147.3| 23.9|     19.1| 14.6|
25  262.9    3.5       19.5   12.0                |262.9|  3.5|     19.5| 12.0|
                                                  +-----+-----+---------+-----+
                                                  only showing top 4 rows
```

5.3.11 With New Column

:: Python Code:

```
dp['tv_norm'] = dp.TV/sum(dp.TV)
dp.head(4)
#
ds.withColumn('tv_norm', ds.TV/ds.groupBy().agg(F.sum("TV")).collect()[0][0]).
↪show(4)
```

:: Comparison:

```
                                                  +-----+-----+---------+-----+-
↪-------------------+
                                                  |   TV|Radio|Newspaper|Sales|␣
↪            tv_norm|
      TV  Radio  Newspaper  Sales   tv_norm      +-----+-----+---------+-----+-
↪-------------------+
0  230.1   37.8       69.2   22.1  0.007824      |230.1| 37.8|     69.2| 22.
↪1|0.007824268493802813|
1   44.5   39.3       45.1   10.4  0.001513      | 44.5| 39.3|     45.1| 10.
↪4|0.001513167961643...|
2   17.2   45.9       69.3    9.3  0.000585      | 17.2| 45.9|     69.3|  9.
↪3|5.848649200061207E-4|
3  151.5   41.3       58.5   18.5  0.005152      |151.5| 41.3|     58.5| 18.
↪5|0.005151571824472517|
                                                  +-----+-----+---------+-----+-
↪-------------------+
                                                  only showing top 4 rows
```

:: Python Code:

```
dp['cond'] = dp.apply(lambda c: 1 if ((c.TV>100)&(c.Radio<40)) else 2 if c.
↪Sales> 10 else 3,axis=1)
#
ds.withColumn('cond',F.when((ds.TV>100)&(ds.Radio<40),1)\
                      .when(ds.Sales>10, 2)\
                      .otherwise(3)).show(4)
```

:: Comparison:

```
                                                    +-----+-----+---------+-----+-
↪---+
                                                    |  ␣
↪TV|Radio|Newspaper|Sales|cond|
      TV  Radio  Newspaper  Sales  cond             +-----+-----+---------+-----+-
↪---+
0  230.1   37.8       69.2   22.1     1             |230.1| 37.8|     69.2| 22.1|␣
↪  1|
1   44.5   39.3       45.1   10.4     2             | 44.5| 39.3|     45.1| 10.4|␣
↪  2|
2   17.2   45.9       69.3    9.3     3             | 17.2| 45.9|     69.3|  9.3|␣
↪  3|
3  151.5   41.3       58.5   18.5     2             |151.5| 41.3|     58.5| 18.5|␣
↪  2|
                                                    +-----+-----+---------+-----+-
↪---+
                                                    only showing top 4 rows
```

:: Python Code:

```
dp['log_tv'] = np.log(dp.TV)
dp.head(4)
#
ds.withColumn('log_tv',F.log(ds.TV)).show(4)
```

:: Comparison:

```
                                                    +-----+-----+---------+-----+-
↪-----------------+
                                                    |   TV|Radio|Newspaper|Sales|␣
↪           log_tv|
      TV  Radio  Newspaper  Sales    log_tv         +-----+-----+---------+-----+-
↪-----------------+
0  230.1   37.8       69.2   22.1  5.438514         |230.1| 37.8|     69.2| 22.1|␣
↪ 5.43851399704132|
1   44.5   39.3       45.1   10.4  3.795489         | 44.5| 39.3|     45.1| 10.
↪4|3.7954891891721947|
2   17.2   45.9       69.3    9.3  2.844909         | 17.2| 45.9|     69.3|  9.
↪3|2.8449093838194073|
3  151.5   41.3       58.5   18.5  5.020586         |151.5| 41.3|     58.5| 18.5|␣
↪5.020585624949423|
                                                    +-----+-----+---------+-----+-
↪-----------------+
                                                    only showing top 4 rows
```

:: Python Code:

```
dp['tv+10'] = dp.TV.apply(lambda x: x+10)
dp.head(4)
#
ds.withColumn('tv+10', ds.TV+10).show(4)
```

:: Comparison:

```
                                                    +-----+-----+---------+-----+-
↪----+
                                                    |  ␣
↪TV|Radio|Newspaper|Sales|tv+10|
      TV  Radio  Newspaper  Sales  tv+10            +-----+-----+---------+-----+-
↪----+
0  230.1   37.8       69.2   22.1  240.1            |230.1| 37.8|     69.2| 22.
↪1|240.1|
1   44.5   39.3       45.1   10.4   54.5            | 44.5| 39.3|     45.1| 10.4|␣
↪54.5|
2   17.2   45.9       69.3    9.3   27.2            | 17.2| 45.9|     69.3|  9.3|␣
↪27.2|
3  151.5   41.3       58.5   18.5  161.5            |151.5| 41.3|     58.5| 18.
↪5|161.5|
                                                    +-----+-----+---------+-----+-
↪----+
                                                    only showing top 4 rows
```

5.3.12 Join

```
leftp = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                      'B': ['B0', 'B1', 'B2', 'B3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']},
                      index=[0, 1, 2, 3])

rightp = pd.DataFrame({'A': ['A0', 'A1', 'A6', 'A7'],
                        'F': ['B4', 'B5', 'B6', 'B7'],
                        'G': ['C4', 'C5', 'C6', 'C7'],
                        'H': ['D4', 'D5', 'D6', 'D7']},
                        index=[4, 5, 6, 7])

lefts = spark.createDataFrame(leftp)
rights = spark.createDataFrame(rightp)
```

```
    A   B   C   D                    A   F   G   H
0  A0  B0  C0  D0               4  A0  B4  C4  D4
1  A1  B1  C1  D1               5  A1  B5  C5  D5
2  A2  B2  C2  D2               6  A6  B6  C6  D6
3  A3  B3  C3  D3               7  A7  B7  C7  D7
```

1. Left Join

   :: Python Code:

```
leftp.merge(rightp,on='A',how='left')
#
lefts.join(rights,on='A',how='left')
     .orderBy('A',ascending=True).show()
```

   :: Comparison:

```
                                        +---+---+---+---+----+---
↪-+----+
                                        |  A|  B|  C|  D|   F|  ␣
↪G|   H|
    A   B   C   D    F    G    H        +---+---+---+---+----+---
↪-+----+
0  A0  B0  C0  D0   B4   C4   D4        | A0| B0| C0| D0|  B4| ␣
↪C4|  D4|
1  A1  B1  C1  D1   B5   C5   D5        | A1| B1| C1| D1|  B5| ␣
↪C5|  D5|
2  A2  B2  C2  D2  NaN  NaN  NaN        | A2| B2| C2|␣
↪D2|null|null|null|
3  A3  B3  C3  D3  NaN  NaN  NaN        | A3| B3| C3|␣
↪D3|null|null|null|
                                        +---+---+---+---+----+---
↪-+----+
```

2. Right Join

```
::  Python Code:
```

```
leftp.merge(rightp,on='A',how='right')
#
lefts.join(rights,on='A',how='right')
     .orderBy('A',ascending=True).show()
```

```
::  Comparison:
```

```
                                        +---+----+----+----+---+-
↪--+---+
                                        |  A|   B|   C|   D|  F|␣
↪ G|  H|
    A    B    C    D   F   G   H        +---+----+----+----+---+-
↪--+---+
0  A0   B0   C0   D0  B4  C4  D4        | A0|  B0|  C0|  D0| B4|␣
↪C4| D4|
1  A1   B1   C1   D1  B5  C5  D5        | A1|  B1|  C1|  D1| B5|␣
↪C5| D5|
2  A6  NaN  NaN  NaN  B6  C6  D6        | A6|null|null|null| B6|␣
↪C6| D6|
3  A7  NaN  NaN  NaN  B7  C7  D7        | A7|null|null|null| B7|␣
↪C7| D7|
                                        +---+----+----+----+---+-
↪--+---+
```

3. Inner Join

```
::  Python Code:
```

```
leftp.merge(rightp,on='A',how='inner')
#
lefts.join(rights,on='A',how='inner')
     .orderBy('A',ascending=True).show()
```

:: Comparison:

```
                                +---+---+---+---+---+---+---+
                                |  A|  B|  C|  D|  F|  G|  H|
    A   B   C   D   F   G   H   +---+---+---+---+---+---+---+
0  A0  B0  C0  D0  B4  C4  D4   | A0| B0| C0| D0| B4| C4| D4|
1  A1  B1  C1  D1  B5  C5  D5   | A1| B1| C1| D1| B5| C5| D5|
                                +---+---+---+---+---+---+---+
```

4. Full Join

:: Python Code:

```
leftp.merge(rightp,on='A',how='full')
#
lefts.join(rights,on='A',how='full')
     .orderBy('A',ascending=True).show()
```

:: Comparison:

```
                                        +---+----+----+----+----
↪+----+----+
                                        |  A|   B|   C|   D|  ␣
↪F|   G|   H|
    A    B    C    D    F    G    H     +---+----+----+----+----
↪+----+----+
0  A0   B0   C0   D0   B4   C4   D4     | A0|  B0|  C0|  D0| ␣
↪B4|  C4|  D4|
1  A1   B1   C1   D1   B5   C5   D5     | A1|  B1|  C1|  D1| ␣
↪B5|  C5|  D5|
2  A2   B2   C2   D2  NaN  NaN  NaN     | A2|  B2|  C2| ␣
↪D2|null|null|null|
3  A3   B3   C3   D3  NaN  NaN  NaN     | A3|  B3|  C3| ␣
↪D3|null|null|null|
4  A6  NaN  NaN  NaN   B6   C6   D6     | A6|null|null|null| ␣
↪B6|  C6|  D6|
5  A7  NaN  NaN  NaN   B7   C7   D7     | A7|null|null|null| ␣
↪B7|  C7|  D7|
                                        +---+----+----+----+----
↪+----+----+
```

5.3.13 Concat Columns

```
my_list = [('a', 2, 3),
           ('b', 5, 6),
           ('c', 8, 9),
           ('a', 2, 3),
           ('b', 5, 6),
           ('c', 8, 9)]
col_name = ['col1', 'col2', 'col3']
#
```

(continues on next page)

(continued from previous page)

```
dp = pd.DataFrame(my_list,columns=col_name)
ds = spark.createDataFrame(my_list,schema=col_name)
```

```
  col1  col2  col3
0    a     2     3
1    b     5     6
2    c     8     9
3    a     2     3
4    b     5     6
5    c     8     9
```

:: Python Code:

```
dp['concat'] = dp.apply(lambda x:'%s%s'%(x['col1'],x['col2']),axis=1)
dp
#
ds.withColumn('concat',F.concat('col1','col2')).show()
```

:: Comparison:

```
  col1  col2  col3 concat
0    a     2     3     a2
1    b     5     6     b5
2    c     8     9     c8
3    a     2     3     a2
4    b     5     6     b5
5    c     8     9     c8
```

```
+----+----+----+------+
|col1|col2|col3|concat|
+----+----+----+------+
|   a|   2|   3|    a2|
|   b|   5|   6|    b5|
|   c|   8|   9|    c8|
|   a|   2|   3|    a2|
|   b|   5|   6|    b5|
|   c|   8|   9|    c8|
+----+----+----+------+
```

5.3.14 GroupBy

:: Python Code:

```
dp.groupby(['col1']).agg({'col2':'min','col3':'mean'})
#
ds.groupBy(['col1']).agg({'col2': 'min', 'col3': 'avg'}).show()
```

:: Comparison:

```
      col2  col3
col1
a        2     3
b        5     6
c        8     9
```

```
+----+---------+---------+
|col1|min(col2)|avg(col3)|
+----+---------+---------+
|   c|        8|      9.0|
|   b|        5|      6.0|
|   a|        2|      3.0|
+----+---------+---------+
```

5.3.15 Pivot

:: Python Code:

```
pd.pivot_table(dp, values='col3', index='col1', columns='col2', aggfunc=np.
↪sum)
#
ds.groupBy(['col1']).pivot('col2').sum('col3').show()
```

:: Comparison:

```
                                  +----+----+----+----+
col2    2     5     8             |col1|   2|   5|   8|
col1                              +----+----+----+----+
a     6.0   NaN   NaN             |   c|null|null|  18|
b     NaN  12.0   NaN             |   b|null|  12|null|
c     NaN   NaN  18.0             |   a|   6|null|null|
                                  +----+----+----+----+
```

5.3.16 Window

```
d = {'A':['a','b','c','d'],'B':['m','m','n','n'],'C':[1,2,3,6]}
dp = pd.DataFrame(d)
ds = spark.createDataFrame(dp)
```

:: Python Code:

```
dp['rank'] = dp.groupby('B')['C'].rank('dense',ascending=False)
#
from pyspark.sql.window import Window
w = Window.partitionBy('B').orderBy(ds.C.desc())
ds = ds.withColumn('rank',F.rank().over(w))
```

:: Comparison:

```
                        +---+---+---+----+
                        |  A|  B|  C|rank|
   A  B  C  rank        +---+---+---+----+
0  a  m  1   2.0        |  b|  m|  2|   1|
1  b  m  2   1.0        |  a|  m|  1|   2|
2  c  n  3   2.0        |  d|  n|  6|   1|
3  d  n  6   1.0        |  c|  n|  3|   2|
                        +---+---+---+----+
```

CHAPTER

SIX

STATISTICS PRELIMINARY

Chinese proverb

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
- $\bar{y} = \frac{1}{m}\sum_{i=1}^{n} y_i$: 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.

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

- Residual Sum of Squares

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

- Coefficient of determination R^2

$$R^2 := 1 - \frac{\text{RSS}}{\text{TSS}}.$$

6.3 Statistical Tests

6.3.1 Correlational Test

- Pearson correlation: Tests for the strength of the association between two continuous variables.
- Spearman correlation: Tests for the strength of the association between two ordinal variables (does not rely on the assumption of normal distributed data).
- Chi-square: Tests for the strength of the association between two categorical variables.

6.3.2 Comparison of Means test

- Paired T-test: Tests for difference between two related variables.
- Independent T-test: Tests for difference between two independent variables.
- ANOVA: Tests the difference between group means after any other variance in the outcome variable is accounted for.

6.3.3 Non-parametric Test

- Wilcoxon rank-sum test: Tests for difference between two independent variables - takes into account magnitude and direction of difference.
- Wilcoxon sign-rank test: Tests for difference between two related variables - takes into account magnitude and direction of difference.
- Sign test: Tests if two related variables are different – ignores magnitude of change, only takes into account direction.

CHAPTER

SEVEN

DATA EXPLORATION

Chinese proverb

A journey of a thousand miles begins with a single step – idiom, from Laozi.

I wouldn't say that understanding your dataset is the most difficult thing in data science, but it is really important and time-consuming. Data Exploration is about describing the data by means of statistical and visualization techniques. We explore data in order to understand the features and bring important features to our models.

7.1 Univariate Analysis

In mathematics, univariate refers to an expression, equation, function or polynomial of only one variable. "Uni" means "one", so in other words your data has only one variable. So you do not need to deal with the causes or relationships in this step. Univariate analysis takes data, summarizes that variables (attributes) one by one and finds patterns in the data.

There are many ways that can describe patterns found in univariate data include central tendency (mean, mode and median) and dispersion: range, variance, maximum, minimum, quartiles (including the interquartile range), coefficient of variation and standard deviation. You also have several options for visualizing and describing data with univariate data. Such as `frequency Distribution Tables`, `bar Charts`, `histograms`, `frequency Polygons`, `pie Charts`.

The variable could be either categorical or numerical, I will demostrate the different statistical and visulization techniques to investigate each type of the variable.

- The Jupyter notebook can be download from Data Exploration.
- The data can be downloaf from German Credit.

7.1.1 Numerical Variables

- Describe

The desctibe function in pandas and spark will give us most of the statistical results, such as min, median, max, quartiles and standard deviation. With the help of the user defined function, you can get even more statistical results.

```
# selected varables for the demonstration
num_cols = ['Account Balance','No of dependents']
df.select(num_cols).describe().show()
```

```
+-------+------------------+-------------------+
|summary|   Account Balance|   No of dependents|
+-------+------------------+-------------------+
|  count|              1000|               1000|
|   mean|             2.577|              1.155|
| stddev|1.2576377271108936|0.36208577175319395|
|    min|                 1|                  1|
|    max|                 4|                  2|
+-------+------------------+-------------------+
```

You may find out that the default function in PySpark does not include the quartiles. The following function will help you to get the same results in Pandas

```
def describe_pd(df_in, columns, deciles=False):
    '''
    Function to union the basic stats results and deciles
    :param df_in: the input dataframe
    :param columns: the cloumn name list of the numerical variable
    :param deciles: the deciles output

    :return : the numerical describe info. of the input dataframe

    :author: Ming Chen and Wenqiang Feng
    :email:  von198@gmail.com
    '''

    if deciles:
        percentiles = np.array(range(0, 110, 10))
    else:
        percentiles = [25, 50, 75]

    percs = np.transpose([np.percentile(df_in.select(x).collect(),␣
↪percentiles) for x in columns])
    percs = pd.DataFrame(percs, columns=columns)
    percs['summary'] = [str(p) + '%' for p in percentiles]

    spark_describe = df_in.describe().toPandas()
    new_df = pd.concat([spark_describe, percs],ignore_index=True)
    new_df = new_df.round(2)
    return new_df[['summary'] + columns]
```

```
describe_pd(df,num_cols)
```

```
+-------+------------------+-----------------+
|summary|   Account Balance| No of dependents|
+-------+------------------+-----------------+
|  count|            1000.0|           1000.0|
|   mean|             2.577|            1.155|
| stddev|1.2576377271108936|0.362085771753194|
|    min|               1.0|              1.0|
|    max|               4.0|              2.0|
|    25%|               1.0|              1.0|
|    50%|               2.0|              1.0|
|    75%|               4.0|              1.0|
+-------+------------------+-----------------+
```

Sometimes, because of the confidential data issues, you can not deliver the real data and your clients may ask more statistical results, such as `deciles`. You can apply the follwing function to achieve it.

```
describe_pd(df,num_cols,deciles=True)
```

```
+-------+------------------+-----------------+
|summary|   Account Balance| No of dependents|
+-------+------------------+-----------------+
|  count|            1000.0|           1000.0|
|   mean|             2.577|            1.155|
| stddev|1.2576377271108936|0.362085771753194|
|    min|               1.0|              1.0|
|    max|               4.0|              2.0|
|     0%|               1.0|              1.0|
|    10%|               1.0|              1.0|
|    20%|               1.0|              1.0|
|    30%|               2.0|              1.0|
|    40%|               2.0|              1.0|
|    50%|               2.0|              1.0|
|    60%|               3.0|              1.0|
|    70%|               4.0|              1.0|
|    80%|               4.0|              1.0|
|    90%|               4.0|              2.0|
|   100%|               4.0|              2.0|
+-------+------------------+-----------------+
```

- Skewness and Kurtosis

  This subsection comes from Wikipedia Skewness.

  In probability theory and statistics, skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive or negative, or undefined.For a unimodal distribution, negative skew commonly indicates that the tail is on the left side of the distribution, and positive skew indicates that the tail is on the right.

  Consider the two distributions in the figure just below. Within each graph, the values on the right side of the distribution taper differently from the values on the left side. These tapering sides are called tails, and they provide a visual means to determine which of the two kinds of skewness a distribution has:

1. negative skew: The left tail is longer; the mass of the distribution is concentrated on the right of the figure. The distribution is said to be left-skewed, left-tailed, or skewed to the left, despite the fact that the curve itself appears to be skewed or leaning to the right; left instead refers to the left tail being drawn out and, often, the mean being skewed to the left of a typical center of the data. A left-skewed distribution usually appears as a right-leaning curve.
2. positive skew: The right tail is longer; the mass of the distribution is concentrated on the left of the figure. The distribution is said to be right-skewed, right-tailed, or skewed to the right, despite the fact that the curve itself appears to be skewed or leaning to the left; right instead refers to the right tail being drawn out and, often, the mean being skewed to the right of a typical center of the data. A right-skewed distribution usually appears as a left-leaning curve.

This subsection comes from Wikipedia Kurtosis.

In probability theory and statistics, kurtosis (kyrtos or kurtos, meaning "curved, arching") is a measure of the "tailedness" of the probability distribution of a real-valued random variable. In a similar way to the concept of skewness, kurtosis is a descriptor of the shape of a probability distribution and, just as for skewness, there are different ways of quantifying it for a theoretical distribution and corresponding ways of estimating it from a sample from a population.

```
from pyspark.sql.functions import col, skewness, kurtosis
df.select(skewness(var),kurtosis(var)).show()
```

```
+---------------------+---------------------+
|skewness(Age (years))|kurtosis(Age (years))|
+---------------------+---------------------+
|   1.0231743160548064|   0.6114371688367672|
+---------------------+---------------------+
```

Warning: Sometimes the statistics can be misleading!

F. J. Anscombe once said that make both calculations and graphs. Both sorts of output should be studied; each will contribute to understanding. These 13 datasets in Figure *Same Stats, Different Graphs* (the Datasaurus, plus 12 others) each have the same summary statistics (x/y mean, x/y standard deviation, and Pearson's correlation) to two decimal places, while being drastically different in appearance. This work

describes the technique we developed to create this dataset, and others like it. More details and interesting results can be found in Same Stats Different Graphs.

Fig. 1: Same Stats, Different Graphs

- Histogram

Warning: Histograms are often confused with Bar graphs!

The fundamental difference between histogram and bar graph will help you to identify the two easily is that there are gaps between bars in a bar graph but in the histogram, the bars are adjacent to each other. The interested reader is referred to Difference Between Histogram and Bar Graph.

```
var = 'Age (years)'
x = data1[var]
bins = np.arange(0, 100, 5.0)

plt.figure(figsize=(10,8))
# the histogram of the data
plt.hist(x, bins, alpha=0.8, histtype='bar', color='gold',
         ec='black',weights=np.zeros_like(x) + 100. / x.size)
```

(continues on next page)

(continued from previous page)

```
plt.xlabel(var)
plt.ylabel('percentage')
plt.xticks(bins)
plt.show()

fig.savefig(var+".pdf", bbox_inches='tight')
```

```
var = 'Age (years)'
x = data1[var]
bins = np.arange(0, 100, 5.0)


######################################################################
hist, bin_edges = np.histogram(x,bins,
                               weights=np.zeros_like(x) + 100. / x.size)
# make the histogram

fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)
```

(continues on next page)

(continued from previous page)

```
# Plot the histogram heights against integers on the x axis
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black', color='gold')
# # Set the ticks to the middle of the bars
ax.set_xticks([0.5+i for i,j in enumerate(hist)])
# Set the xticklabels to a string that tells us what the bin edges were
labels =['{}'.format(int(bins[i+1])) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
plt.xlabel(var)
plt.ylabel('percentage')

#######################################################################

hist, bin_edges = np.histogram(x,bins) # make the histogram

ax = fig.add_subplot(1, 2, 2)
# Plot the histogram heights against integers on the x axis
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black', color='gold')

# # Set the ticks to the middle of the bars
ax.set_xticks([0.5+i for i,j in enumerate(hist)])

# Set the xticklabels to a string that tells us what the bin edges were
labels =['{}'.format(int(bins[i+1])) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
plt.xlabel(var)
plt.ylabel('count')
plt.suptitle('Histogram of {}: Left with percentage output;Right with count␣
↪output'
             .format(var), size=16)
plt.show()

fig.savefig(var+".pdf", bbox_inches='tight')
```

Sometimes, some people will ask you to plot the unequal width (invalid argument for histogram) of the bars. YOu can still achieve it by the follwing trick.

```
var = 'Credit Amount'
plot_data = df.select(var).toPandas()
x= plot_data[var]

bins =[0,200,400,600,700,800,900,1000,2000,3000,4000,5000,6000,10000,25000]

hist, bin_edges = np.histogram(x,bins,weights=np.zeros_like(x) + 100. / x.
↪size) # make the histogram

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(1, 1, 1)
# Plot the histogram heights against integers on the x axis
```

(continues on next page)

(continued from previous page)

```
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black',color = 'gold')

# # Set the ticks to the middle of the bars
ax.set_xticks([0.5+i for i,j in enumerate(hist)])

# Set the xticklabels to a string that tells us what the bin edges were
#labels =['{}k'.format(int(bins[i+1]/1000)) for i,j in enumerate(hist)]
labels =['{}'.format(bins[i+1]) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
#plt.text(-0.6, -1.4,'0')
plt.xlabel(var)
plt.ylabel('percentage')
plt.show()
```

- Box plot and violin plot

Note that although violin plots are closely related to Tukey's (1977) box plots, the violin plot can show more information than box plot. When we perform an exploratory analysis, nothing about the samples could be known. So the distribution of the samples can not be assumed to a normal distribution and usually when you get a big data, the normal distribution will show some out liars in box plot.

However, the violin plots are potentially misleading for smaller sample sizes, where the density plots can appear to show interesting features (and group-differences therein) even when produced for standard normal data. Some poster suggested the sample size should larger that 250. The sample sizes (e.g. n>250 or ideally even larger), where the kernel density plots provide a reasonably accurate representation of the distributions, potentially showing nuances such as bimodality or other forms of non-normality that would be invisible or less clear in box plots. More details can be found in A simple comparison of box plots and violin plots.

```
x = df.select(var).toPandas()
```

(continues on next page)

30
25
20
15
10
5
0
percentage
0
200
400
600
700
800
900
1000
2000
3000
4000
5000
6000
10000
25000
Credit Amount

(continued from previous page)

```
fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)
ax = sns.boxplot(data=x)

ax = fig.add_subplot(1, 2, 2)
ax = sns.violinplot(data=x)
```

7.1.2 Categorical Variables

Compared with the numerical variables, the categorical variables are much more easier to do the exploration.

- Frequency table

```
from pyspark.sql import functions as F
from pyspark.sql.functions import rank,sum,col
from pyspark.sql import Window

window = Window.rowsBetween(Window.unboundedPreceding,Window.
↪unboundedFollowing)
# withColumn('Percent %',F.format_string("%5.0f%%\n",col('Credit_num')*100/
↪col('total'))).\
tab = df.select(['age_class','Credit Amount']).\
   groupBy('age_class').\
   agg(F.count('Credit Amount').alias('Credit_num'),
       F.mean('Credit Amount').alias('Credit_avg'),
       F.min('Credit Amount').alias('Credit_min'),
       F.max('Credit Amount').alias('Credit_max')).\
   withColumn('total',sum(col('Credit_num')).over(window)).\
   withColumn('Percent',col('Credit_num')*100/col('total')).\
   drop(col('total'))
```

```
+---------+----------+------------------+----------+----------+-------+
|age_class|Credit_num|        Credit_avg|Credit_min|Credit_max|Percent|
```

(continues on next page)

(continued from previous page)

```
+---------+----------+------------------+----------+----------+-------+
|    45-54|       120|3183.0666666666666|       338|     12612|   12.0|
|      <25|       150| 2970.733333333333|       276|     15672|   15.0|
|    55-64|        56| 3493.660714285714|       385|     15945|    5.6|
|    35-44|       254| 3403.771653543307|       250|     15857|   25.4|
|    25-34|       397| 3298.823677581864|       343|     18424|   39.7|
|      65+|        23|3210.1739130434785|       571|     14896|    2.3|
+---------+----------+------------------+----------+----------+-------+
```

- Pie plot

```
# Data to plot
labels = plot_data.age_class
sizes =  plot_data.Percent
colors = ['gold', 'yellowgreen', 'lightcoral','blue', 'lightskyblue','green',
↪'red']
explode = (0, 0.1, 0, 0,0,0)  # explode 1st slice

# Plot
plt.figure(figsize=(10,8))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```

- Bar plot

```
labels = plot_data.age_class
missing = plot_data.Percent
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, missing, width=0.8, label='missing', color='gold')

plt.xticks(ind, labels)
plt.ylabel("percentage")

plt.show()
```

```
labels = ['missing', '<25', '25-34', '35-44', '45-54','55-64','65+']
missing = np.array([0.000095, 0.024830, 0.028665, 0.029477, 0.031918,0.037073,
↪0.026699])
man = np.array([0.000147, 0.036311, 0.038684, 0.044761, 0.051269, 0.059542, 0.
↪054259])
women = np.array([0.004035, 0.032935, 0.035351, 0.041778, 0.048437, 0.056236,
↪0.048091])
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, women, width=0.8, label='women', color='gold',␣
↪bottom=man+missing)
```

(continues on next page)

45-54
55-64
65+
12.0%
5.6%
2.3%
35-44
<25
25.4%
15.0%
39.7%
25-34

40
35
30
25
20
15
10
5
0
percentage
<25
25-34
35-44
45-54
55-64
65+

(continued from previous page)

```
plt.bar(ind, man, width=0.8, label='man', color='silver', bottom=missing)
plt.bar(ind, missing, width=0.8, label='missing', color='#CD853F')

plt.xticks(ind, labels)
plt.ylabel("percentage")
plt.legend(loc="upper left")
plt.title("demo")

plt.show()
```

7.2 Multivariate Analysis

In this section, I will only demostrate the bivariate analysis. Since the multivariate analysis is the generation of the bivariate.

7.2.1 Numerical V.S. Numerical

- Correlation matrix

```
from pyspark.mllib.stat import Statistics
import pandas as pd

corr_data = df.select(num_cols)

col_names = corr_data.columns
features = corr_data.rdd.map(lambda row: row[0:])
corr_mat=Statistics.corr(features, method="pearson")
corr_df = pd.DataFrame(corr_mat)
corr_df.index, corr_df.columns = col_names, col_names

print(corr_df.to_string())
```

```
+--------------------+--------------------+
|     Account Balance|    No of dependents|
+--------------------+--------------------+
|                 1.0|-0.01414542650320914|
|-0.01414542650320914|                 1.0|
+--------------------+--------------------+
```

- Scatter Plot

```
import seaborn as sns
sns.set(style="ticks")

df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
plt.show()
```

7.2.2 Categorical V.S. Categorical

- Pearson's Chi-squared test

> **Warning:** `pyspark.ml.stat` is only available in Spark 2.4.0.

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import ChiSquareTest

data = [(0.0, Vectors.dense(0.5, 10.0)),
        (0.0, Vectors.dense(1.5, 20.0)),
        (1.0, Vectors.dense(1.5, 30.0)),
        (0.0, Vectors.dense(3.5, 30.0)),
        (0.0, Vectors.dense(3.5, 40.0)),
        (1.0, Vectors.dense(3.5, 40.0))]
```

(continues on next page)

sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica

(continued from previous page)

```
df = spark.createDataFrame(data, ["label", "features"])

r = ChiSquareTest.test(df, "features", "label").head()
print("pValues: " + str(r.pValues))
print("degreesOfFreedom: " + str(r.degreesOfFreedom))
print("statistics: " + str(r.statistics))
```

```
pValues: [0.687289278791,0.682270330336]
degreesOfFreedom: [2, 3]
statistics: [0.75,1.5]
```

- Cross table

```
df.stat.crosstab("age_class", "Occupation").show()
```

```
+--------------------+---+---+---+---+
|age_class_Occupation|  1|  2|  3|  4|
+--------------------+---+---+---+---+
|                 <25|  4| 34|108|  4|
|               55-64|  1| 15| 31|  9|
|               25-34|  7| 61|269| 60|
|               35-44|  4| 58|143| 49|
|                 65+|  5|  3|  6|  9|
|               45-54|  1| 29| 73| 17|
+--------------------+---+---+---+---+
```

- Stacked plot

```
labels = ['missing', '<25', '25-34', '35-44', '45-54','55-64','65+']
missing = np.array([0.000095, 0.024830, 0.028665, 0.029477, 0.031918,0.037073,
↪0.026699])
man = np.array([0.000147, 0.036311, 0.038684, 0.044761, 0.051269, 0.059542, 0.
↪054259])
women = np.array([0.004035, 0.032935, 0.035351, 0.041778, 0.048437, 0.056236,
↪0.048091])
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, women, width=0.8, label='women', color='gold',␣
↪bottom=man+missing)
plt.bar(ind, man, width=0.8, label='man', color='silver', bottom=missing)
plt.bar(ind, missing, width=0.8, label='missing', color='#CD853F')

plt.xticks(ind, labels)
plt.ylabel("percentage")
plt.legend(loc="upper left")
plt.title("demo")

plt.show()
```

demo
women
man
missing
percentage
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
missing
<25
25-34
35-44
45-54
55-64
65+

7.2.3 Numerical V.S. Categorical

CHAPTER

EIGHT

REGRESSION

Chinese proverb

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).

8.1 Linear Regression

8.1.1 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 j independent variables: x_{ij}, the formula is given by:

$$y_i = \beta_0 + \beta_j x_{ij}, \text{where}, i = 1, \cdots m, j = 1, \cdots n.$$

In matrix notation, the data set is written as $\mathbf{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n]$ with $\boldsymbol{x}_j = \{x_{ij}\}_{i=1}^m$, $\boldsymbol{y} = \{y_i\}_{i=1}^m$ (see Fig. *Feature matrix and label*) and $\boldsymbol{\beta}^\top = \{\beta_j\}_{j=1}^n$. Then the matrix format equation is written as

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta}. \tag{8.1}$$

8.1.2 How to solve it?

1. Direct Methods (For more information please refer to my Prelim Notes for Numerical Analysis)

Fig. 1: Feature matrix and label

- For squared or rectangular matrices
  - Singular Value Decomposition
  - Gram-Schmidt orthogonalization
  - QR Decomposition
- For squared matrices
  - LU Decomposition
  - Cholesky Decomposition
  - Regular Splittings

2. Iterative Methods
   - Stationary cases iterative method
     - Jacobi Method
     - Gauss-Seidel Method
     - Richardson Method
     - Successive Over Relaxation (SOR) Method
   - Dynamic cases iterative method
     - Chebyshev iterative Method
     - Minimal residuals Method
     - Minimal correction iterative method
     - Steepest Descent Method
     - Conjugate Gradients Method

8.1.3 Ordinary Least Squares

In mathematics, (8.1) is a overdetermined system. The method of ordinary least squares can be used to find an approximate solution to overdetermined systems. For the system overdetermined system (8.1), the least squares formula is obtained from the problem

$$\min_x ||\mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}||, \tag{8.2}$$

the solution of which can be written with the normal equations:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y} \tag{8.3}$$

where T indicates a matrix transpose, provided $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ exists (that is, provided $\mathbf{X}$ has full column rank).

Note: Actually, (8.3) is derivated by the following way: multiply $\mathbf{X}^T$ on side of (8.1) and then multiply $(\mathbf{X}^T\mathbf{X})^{-1}$ on both side of the former result.

8.1.4 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|
+-------+-----------------+------------------+------------------+-------------
↪-----+
```

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):
```

(continues on next page)

Fig. 2: Sales distribution

(continued from previous page)

```
#     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'])
```

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]
```

(continues on next page)

(continued from previous page)

```
    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])
```

(continues on next page)

(continued from previous page)

```
    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

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 supervised machine learning algorithms in Spark are based on the **features** and **label** (unsupervised machine learning algorithms in Spark are based on the **features**). That is to say, you can play with all of the machine learning algorithms in Spark when you get ready the **features** and **label** in pipeline architecture.

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')
('##', '---')
```

(continues on next page)

(continued from previous page)

```
('##', '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.

8.2 Generalized linear regression

8.2.1 Introduction

8.2.2 How to solve it?

8.2.3 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|
```

(continues on next page)

(continued from previous page)

```
+-----+-----+---------+-----+
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**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]
```

(continues on next page)

(continued from previous page)

```
    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])
```

(continues on next page)

(continued from previous page)

```
    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

```
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'])
```

(continues on next page)

(continued from previous page)

```
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

```
# 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

```
# 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.

```
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)):
```

(continues on next page)

(continued from previous page)

```
        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",
```

(continues on next page)

(continued from previous page)

```
                                    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
```

8.3 Decision tree Regression

8.3.1 Introduction

8.3.2 How to solve it?

8.3.3 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', \
```

(continues on next page)

(continued from previous page)

```
                        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**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''
```

(continues on next page)

(continued from previous page)

```
    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

```
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|
```

(continues on next page)

(continued from previous page)

```
|[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})
```

8.4 Random Forest Regression

8.4.1 Introduction

8.4.2 How to solve it?

8.4.3 Demo

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

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark RandomForest 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);

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

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

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

(continues on next page)

(continued from previous page)

```
+-------+-----------------+------------------+------------------+-------------
↪-----+
|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**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
```

(continues on next page)

(continued from previous page)

```
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

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

# convert the data to dense vector
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
```

(continues on next page)

(continued from previous page)

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

4. Convert the data to dense vector

```
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
```

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

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

data = featureIndexer.transform(transformed)
data.show(5,True)
```

```
+-----------------+-----+-----------------+
|         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) = data.randomSplit([0.6, 0.4])
```

(continues on next page)

(continued from previous page)

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

```
+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|   [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]|
|[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]|
+----------------+-----+----------------+
only showing top 5 rows

+---------------+-----+---------------+
|       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]|
| [8.4,27.2,2.1]|  5.7| [8.4,27.2,2.1]|
+---------------+-----+---------------+
only showing top 5 rows
```

7. Fit RandomForest Regression Model

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

# Define LinearRegression algorithm
rf = RandomForestRegressor() # featuresCol="indexedFeatures",numTrees=2,␣
↪maxDepth=2, seed=42
```

Note: If you decide to use the `indexedFeatures` features, you need to add the parameter `featuresCol="indexedFeatures"`.

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, rf])
model = pipeline.fit(trainingData)
```

9. Make predictions

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

(continues on next page)

(continued from previous page)

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

```
+---------------+-----+------------------+
|       features|label|        prediction|
+---------------+-----+------------------+
| [4.1,11.6,5.7]|  3.2| 8.155439814814816|
| [5.4,29.9,9.4]|  5.3|10.412769901394899|
|[7.3,28.1,41.4]|  5.5| 12.13735648148148|
|[7.8,38.9,50.6]|  6.6|11.321796703296704|
| [8.4,27.2,2.1]|  5.7|12.071421957671957|
+---------------+-----+------------------+
only showing top 5 rows
```

10. Evaluation

```
# 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)
```

```
Root Mean Squared Error (RMSE) on test data = 2.35912
```

```
import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {:4.3f}'.format(r2_score))
```

```
r2_score: 0.831
```

11. Feature importances

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

```
SparseVector(3, {0: 0.4994, 1: 0.3196, 2: 0.181})
```

```
model.stages[-1].trees
```

```
[DecisionTreeRegressionModel (uid=dtr_c75f1c75442c) of depth 5 with 43 nodes,
 DecisionTreeRegressionModel (uid=dtr_70fc2d441581) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_bc8464f545a7) of depth 5 with 31 nodes,
 DecisionTreeRegressionModel (uid=dtr_a8a7e5367154) of depth 5 with 59 nodes,
 DecisionTreeRegressionModel (uid=dtr_3ea01314fcbc) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_be9a04ac22a6) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_38610d47328a) of depth 5 with 51 nodes,
 DecisionTreeRegressionModel (uid=dtr_bf14aea0ad3b) of depth 5 with 49 nodes,
 DecisionTreeRegressionModel (uid=dtr_cde24ebd6bb6) of depth 5 with 39 nodes,
 DecisionTreeRegressionModel (uid=dtr_a1fc9bd4fbeb) of depth 5 with 57 nodes,
 DecisionTreeRegressionModel (uid=dtr_37798d6db1ba) of depth 5 with 41 nodes,
```

(continues on next page)

(continued from previous page)

```
 DecisionTreeRegressionModel (uid=dtr_c078b73ada63) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_fd00e3a070ad) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_9d01d5fb8604) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_8bd8bdddf642) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_e53b7bae30f8) of depth 5 with 49 nodes,
 DecisionTreeRegressionModel (uid=dtr_808a869db21c) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_64d0916bceb0) of depth 5 with 33 nodes,
 DecisionTreeRegressionModel (uid=dtr_0891055fff94) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_19c8bbad26c2) of depth 5 with 51 nodes]
```

8.5 Gradient-boosted tree regression

8.5.1 Introduction

8.5.2 How to solve it?

8.5.3 Demo

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

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark GBTRegressor 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);

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

```
+-----+-----+---------+-----+
|   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|
```

(continues on next page)

(continued from previous page)

```
|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)
```

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

+-------+-----------------+------------------+------------------+------------
↪-----+
|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**)

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in comple dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]
```

(continues on next page)

(continued from previous page)

```
    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])
```

(continues on next page)

(continued from previous page)

```
    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

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

# convert the data to dense vector
#def transData(row):
#    return Row(label=row["Sales"],
#               features=Vectors.dense([row["TV"],
#                                       row["Radio"],
#                                       row["Newspaper"]]))
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF([
↪'features','label'])
```

4. Convert the data to dense vector

```
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
```

5. Deal with the Categorical variables

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

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

data = featureIndexer.transform(transformed)
data.show(5,True)
```

```
+-----------------+-----+-----------------+
|         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) = data.randomSplit([0.6, 0.4])

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

```
+----------------+-----+----------------+
|        features|label| indexedFeatures|
+----------------+-----+----------------+
|  [0.7,39.6,8.7]|  1.6|  [0.7,39.6,8.7]|
|   [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]|
|[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]|
+----------------+-----+----------------+
only showing top 5 rows

+---------------+-----+---------------+
|       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]|
| [8.4,27.2,2.1]|  5.7| [8.4,27.2,2.1]|
+---------------+-----+---------------+
only showing top 5 rows
```

7. Fit RandomForest Regression Model

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

# Define LinearRegression algorithm
rf = GBTRegressor() #numTrees=2, maxDepth=2, seed=42
```

Note: If you decide to use the `indexedFeatures` features, you need to add the parameter

```
featuresCol="indexedFeatures".
```

8. Pipeline Architecture

```
# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, rf])
model = pipeline.fit(trainingData)
```

9. Make predictions

```
predictions = model.transform(testData)

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

```
+----------------+-----+------------------+
|        features|label|        prediction|
+----------------+-----+------------------+
| [7.8,38.9,50.6]|  6.6| 6.836040343319862|
|   [8.6,2.1,1.0]|  4.8| 5.652202764688849|
| [8.7,48.9,75.0]|  7.2| 6.908750296855572|
| [13.1,0.4,25.6]|  5.3| 5.784020210692574|
|[19.6,20.1,17.0]|  7.6|6.8678921062629295|
+----------------+-----+------------------+
only showing top 5 rows
```

10. Evaluation

```
# 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)
```

```
Root Mean Squared Error (RMSE) on test data = 1.36939
```

```
import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {:4.3f}'.format(r2_score))
```

```
r2_score: 0.932
```

11. Feature importances

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

```
SparseVector(3, {0: 0.3716, 1: 0.3525, 2: 0.2759})
```

```
model.stages[-1].trees
```

```
[DecisionTreeRegressionModel (uid=dtr_7f5cd2ef7cb6) of depth 5 with 61 nodes,
 DecisionTreeRegressionModel (uid=dtr_ef3ab6baeac9) of depth 5 with 39 nodes,
 DecisionTreeRegressionModel (uid=dtr_07c6e3cf3819) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_ce724af79a2b) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_d149ecc71658) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_d3a79bdea516) of depth 5 with 43 nodes,
 DecisionTreeRegressionModel (uid=dtr_7abc1a337844) of depth 5 with 51 nodes,
 DecisionTreeRegressionModel (uid=dtr_480834b46d8f) of depth 5 with 33 nodes,
 DecisionTreeRegressionModel (uid=dtr_0cbd1eaa3874) of depth 5 with 39 nodes,
 DecisionTreeRegressionModel (uid=dtr_8088ac71a204) of depth 5 with 57 nodes,
 DecisionTreeRegressionModel (uid=dtr_2ceb9e8deb45) of depth 5 with 47 nodes,
 DecisionTreeRegressionModel (uid=dtr_cc334e84e9a2) of depth 5 with 57 nodes,
 DecisionTreeRegressionModel (uid=dtr_a665c562929e) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_2999b1ffd2dc) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_29965cbe8cfc) of depth 5 with 55 nodes,
 DecisionTreeRegressionModel (uid=dtr_731df51bf0ad) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_354cf33424da) of depth 5 with 51 nodes,
 DecisionTreeRegressionModel (uid=dtr_4230f200b1c0) of depth 5 with 41 nodes,
 DecisionTreeRegressionModel (uid=dtr_3279cdc1ce1d) of depth 5 with 45 nodes,
 DecisionTreeRegressionModel (uid=dtr_f474a99ff06e) of depth 5 with 55 nodes]
```

CHAPTER

NINE

REGULARIZATION

In mathematics, statistics, and computer science, particularly in the fields of machine learning and inverse problems, regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting (Wikipedia Regularization).

Due to the sparsity within our data, our training sets will often be ill-posed (singular). Applying regularization to the regression has many advantages, including:

1. Converting ill-posed problems to well-posed by adding additional information via the penalty parameter λ
2. Preventing overfitting
3. Variable selection and the removal of correlated variables (Glmnet Vignette). The Ridge method shrinks the coefficients of correlated variables while the LASSO method picks one variable and discards the others. The elastic net penalty is a mixture of these two; if variables are correlated in groups then $\alpha = 0.5$ tends to select the groups as in or out. If α is close to 1, the elastic net performs much like the LASSO method and removes any degeneracies and wild behavior caused by extreme correlations.

9.1 Ordinary least squares regression

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^n} \frac{1}{n}\|\mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}\|^2$$

When $\lambda = 0$ (i.e. `regParam` $= 0$), then there is no penalty.

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
↪"prediction", maxIter=100,
regParam=0.0, elasticNetParam=0.0, tol=1e-6, fitIntercept=True,␣
↪standardization=True, solver="auto",
weightCol=None, aggregationDepth=2)
```

9.2 Ridge regression

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^n} \frac{1}{n}\|\mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{\beta}\|_2^2$$

When $\lambda > 0$ (i.e. `regParam` > 0) and $\alpha = 0$ (i.e. `elasticNetParam` $= 0$) , then the penalty is an L2 penalty.

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
↪"prediction", maxIter=100,
regParam=0.1, elasticNetParam=0.0, tol=1e-6, fitIntercept=True,␣
↪standardization=True, solver="auto",
weightCol=None, aggregationDepth=2)
```

9.3 Least Absolute Shrinkage and Selection Operator (LASSO)

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^n} \frac{1}{n}\|\mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{\beta}\|_1$$

When $\lambda > 0$ (i.e. `regParam` > 0) and $\alpha = 1$ (i.e. `elasticNetParam` $= 1$), then the penalty is an L1 penalty.

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
↪"prediction", maxIter=100,
regParam=0.0, elasticNetParam=0.0, tol=1e-6, fitIntercept=True,␣
↪standardization=True, solver="auto",
weightCol=None, aggregationDepth=2)
```

9.4 Elastic net

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^n} \frac{1}{n}\|\mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda(\alpha\|\boldsymbol{\beta}\|_1 + (1-\alpha)\|\boldsymbol{\beta}\|_2^2), \alpha \in (0,1)$$

When $\lambda > 0$ (i.e. `regParam` > 0) and `elasticNetParam` $\in (0,1)$ (i.e. $\alpha \in (0,1)$) , then the penalty is an L1 + L2 penalty.

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
↪"prediction", maxIter=100,
regParam=0.0, elasticNetParam=0.0, tol=1e-6, fitIntercept=True,␣
↪standardization=True, solver="auto",
weightCol=None, aggregationDepth=2)
```

CHAPTER

TEN

CLASSIFICATION

Chinese proverb

Birds of a feather folock together. – old Chinese proverb

10.1 Binomial logistic regression

10.1.1 Introduction

10.1.2 Demo

- The Jupyter notebook can be download from Logistic Regression.
- For more details, please visit Logistic Regression API .

Note: In this demo, I introduced a new function `get_dummy` to deal with the categorical data. I highly recommend you to use my `get_dummy` function in the other cases. This function will save a lot of time for you.

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

        spark = SparkSession \
            .builder \
            .appName("Python Spark Logistic 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') \
```

(continues on next page)

(continued from previous page)

```
            .load("./data/bank.csv",header=True);
df.drop('day','month','poutcome').show(5)
```

```
+---+------------+-------+---------+-------+-------+-------+----+-------+-----
↪---+--------+-----+--------+---+
|age|         ␣
↪job|marital|education|default|balance|housing|loan|contact|duration|campaign|pdays|previo
↪ y|
+---+------------+-------+---------+-------+-------+-------+----+-------+-----
↪---+--------+-----+--------+---+
| 58|  management|married| tertiary|     no|   2143|    yes|  no|unknown|    ␣
↪261|       1|   -1|       0| no|
| 44|  technician| single|secondary|     no|     29|    yes|  no|unknown|    ␣
↪151|       1|   -1|       0| no|
| 33|entrepreneur|married|secondary|     no|      2|    yes| yes|unknown|    ␣
↪ 76|       1|   -1|       0| no|
| 47| blue-collar|married|  unknown|     no|   1506|    yes|  no|unknown|    ␣
↪ 92|       1|   -1|       0| no|
| 33|     unknown| single|  unknown|     no|      1|     no|  no|unknown|    ␣
↪198|       1|   -1|       0| no|
+---+------------+-------+---------+-------+-------+-------+----+-------+-----
↪---+--------+-----+--------+---+
only showing top 5 rows
```

```
df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- y: string (nullable = true)
```

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in complex dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
```

(continues on next page)

(continued from previous page)

```
                    outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                    for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

```
def get_dummy(df,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder␣
↪in encoders]
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select('features','label')
```

3. Deal with categorical data and Convert the data to dense vector

```
catcols = ['job','marital','education','default',
           'housing','loan','contact','poutcome']

num_cols = ['balance', 'duration','campaign','pdays','previous',]
```

(continues on next page)

(continued from previous page)

```
labelCol = 'y'

data = get_dummy(df,catcols,num_cols,labelCol)
data.show(5)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|(29,[1,11,14,16,1...|   no|
|(29,[2,12,13,16,1...|   no|
|(29,[7,11,13,16,1...|   no|
|(29,[0,11,16,17,1...|   no|
|(29,[12,16,18,20,...|   no|
+--------------------+-----+
only showing top 5 rows
```

4. Deal with Categorical Label and Variables

```
from pyspark.ml.feature import StringIndexer
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(5, True)
```

```
+--------------------+-----+------------+
|            features|label|indexedLabel|
+--------------------+-----+------------+
|(29,[1,11,14,16,1...|   no|         0.0|
|(29,[2,12,13,16,1...|   no|         0.0|
|(29,[7,11,13,16,1...|   no|         0.0|
|(29,[0,11,16,17,1...|   no|         0.0|
|(29,[12,16,18,20,...|   no|         0.0|
+--------------------+-----+------------+
only showing top 5 rows
```

```
from pyspark.ml.feature import VectorIndexer
# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as␣
↪continuous.
featureIndexer =VectorIndexer(inputCol="features", \
                                  outputCol="indexedFeatures", \
                                  maxCategories=4).fit(data)
featureIndexer.transform(data).show(5, True)
```

```
+--------------------+-----+--------------------+
|            features|label|     indexedFeatures|
+--------------------+-----+--------------------+
|(29,[1,11,14,16,1...|   no|(29,[1,11,14,16,1...|
|(29,[2,12,13,16,1...|   no|(29,[2,12,13,16,1...|
|(29,[7,11,13,16,1...|   no|(29,[7,11,13,16,1...|
```

(continues on next page)

(continued from previous page)

```
|(29,[0,11,16,17,1...|   no|(29,[0,11,16,17,1...|
|(29,[12,16,18,20,...|   no|(29,[12,16,18,20,...|
+--------------------+-----+--------------------+
only showing top 5 rows
```

5. Split the data to training and test data sets

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

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

```
+--------------------------------------------------------------------------
↪--------------------+-----+
|features                                                                  ↩
↪                     |label|
+--------------------------------------------------------------------------
↪--------------------+-----+
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪731.0,401.0,4.0,-1.0])|no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪723.0,112.0,2.0,-1.0])|no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪626.0,205.0,1.0,-1.0])|no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪498.0,357.0,1.0,-1.0])|no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪477.0,473.0,2.0,-1.0])|no   |
+--------------------------------------------------------------------------
↪--------------------+-----+
only showing top 5 rows

+--------------------------------------------------------------------------
↪--------------------+-----+
|features                                                                  ↩
↪                     |label|
+--------------------------------------------------------------------------
↪--------------------+-----+
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪648.0,280.0,2.0,-1.0])|no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪596.0,147.0,1.0,-1.0])|no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪529.0,416.0,4.0,-1.0])|no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪518.0,46.0,5.0,-1.0]) |no   |
|(29,[0,11,13,16,17,18,19,21,24,25,26,27],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
↪470.0,275.0,2.0,-1.0])|no   |
+--------------------------------------------------------------------------
↪--------------------+-----+
```

(continues on next page)

(continued from previous page)

```
only showing top 5 rows
```

6. Fit Logistic Regression Model

```
from pyspark.ml.classification import LogisticRegression
logr = LogisticRegression(featuresCol='indexedFeatures', labelCol=
↪'indexedLabel')
```

7. Pipeline Architecture

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

```
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, logr,
↪labelConverter])
```

```
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

8. Make predictions

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

```
+--------------------+-----+--------------+
|            features|label|predictedLabel|
+--------------------+-----+--------------+
|(29,[0,11,13,16,1...|   no|            no|
|(29,[0,11,13,16,1...|   no|            no|
|(29,[0,11,13,16,1...|   no|            no|
|(29,[0,11,13,16,1...|   no|            no|
|(29,[0,11,13,16,1...|   no|            no|
+--------------------+-----+--------------+
only showing top 5 rows
```

9. Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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

```
Test Error = 0.0987688
```

```
lrModel = model.stages[2]
trainingSummary = lrModel.summary

# Obtain the objective per iteration
# objectiveHistory = trainingSummary.objectiveHistory
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)

# Obtain the receiver-operating characteristic as a dataframe and␣
↪areaUnderROC.
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

# Set the model threshold to maximize F-Measure
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').
↪head(5)
# bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-
↪Measure)']) \
#     .select('threshold').head()['threshold']
# lr.setThreshold(bestThreshold)
```

You can use `z.show()` to get the data and plot the ROC curves:

You can also register a TempTable `data.registerTempTable('roc_data')` and then use `sql` to plot the ROC curve:

10. visualization

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
```

(continues on next page)

(continued from previous page)

```
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
```

(continues on next page)

(continued from previous page)

```
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

```
['no', 'yes']
```

```
from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix
```

```
array([[15657,   379],
       [ 1410,   667]])
```

```
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[15657   379]
 [ 1410   667]]
```

```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[ 0.97636568  0.02363432]
 [ 0.67886375  0.32113625]]
```

Confusion matrix, without normalization
15657
379
1410
667
no
yes
True label
no
yes
Predicted label
14000
12000
10000
8000
6000
4000
2000

Normalized confusion matrix
0.98
0.02
0.68
0.32
no
yes
True label
no
yes
Predicted label
0.8
0.6
0.4
0.2

10.2 Multinomial logistic regression

10.2.1 Introduction

10.2.2 Demo

- The Jupyter notebook can be download from Logistic Regression.
- For more details, please visit Logistic Regression API .

Note: In this demo, I introduced a new function `get_dummy` to deal with the categorical data. I highly recommend you to use my `get_dummy` function in the other cases. This function will save a lot of time for you.

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark MultinomialLogisticRegression classification") \
    .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/WineData2.csv",header=True);
df.show(5)
```

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

```
df.printSchema()
```

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

```
# 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"))

df.show(5,True)
```

```
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-----
↪--+-------+
|fixed|volatile|citric|sugar|chlorides|free|total|density| ␣
↪pH|sulphates|alcohol|quality|
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-----
↪--+-------+
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|   ␣
↪9.4| medium|
|  7.8|    0.88|   0.0|  2.6|    0.098|25.0| 67.0| 0.9968| 3.2|     0.68|   ␣
↪9.8| medium|
|  7.8|    0.76|  0.04|  2.3|    0.092|15.0| 54.0|  0.997|3.26|     0.65|   ␣
↪9.8| medium|
```

(continues on next page)

(continued from previous page)

```
| 11.2|    0.28|  0.56|  1.9|    0.075|17.0| 60.0|  0.998|3.16|     0.58|
↪9.8| medium|
|  7.4|     0.7|   0.0|  1.9|    0.076|11.0| 34.0| 0.9978|3.51|     0.56|
↪9.4| medium|
+-----+--------+------+-----+---------+----+-----+-------+----+---------+-----
↪--+-------+
only showing top 5 rows
```

```
df.printSchema()
```

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

3. Deal with categorical data and Convert the data to dense vector

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in complex dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()
↪for encoder in encoders]
```

(continues on next page)

(continued from previous page)

```
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

```
def get_dummy(df,categoricalCols,continuousCols,labelCol):
```

(continues on next page)

(continued from previous page)

```
    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder␣
↪in encoders]
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select('features','label')
```

4. Transform the dataset to DataFrame

```
from pyspark.ml.linalg import Vectors # !!!!caution: not from pyspark.mllib.
↪linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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|
+--------------------+------+
|[7.4,0.7,0.0,1.9,...|medium|
|[7.8,0.88,0.0,2.6...|medium|
|[7.8,0.76,0.04,2....|medium|
|[11.2,0.28,0.56,1...|medium|
|[7.4,0.7,0.0,1.9,...|medium|
```

(continues on next page)

(continued from previous page)

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

4. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+------------+
|            features| label|indexedLabel|
+--------------------+------+------------+
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
|[7.8,0.88,0.0,2.6...|medium|         0.0|
|[7.8,0.76,0.04,2....|medium|         0.0|
|[11.2,0.28,0.56,1...|medium|         0.0|
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
+--------------------+------+------------+
only showing top 5 rows
```

```
# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as␣
↪continuous.
featureIndexer =VectorIndexer(inputCol="features", \
                              outputCol="indexedFeatures", \
                              maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+--------------------+
|            features| label|     indexedFeatures|
+--------------------+------+--------------------+
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+--------------------+------+--------------------+
only showing top 5 rows
```

5. Split the data to training and test data sets

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

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

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

(continues on next page)

(continued from previous page)

```
+-------------------------------------------------------+------+
|[4.7,0.6,0.17,2.3,0.058,17.0,106.0,0.9932,3.85,0.6,12.9] |medium|
|[5.0,0.38,0.01,1.6,0.048,26.0,60.0,0.99084,3.7,0.75,14.0]|medium|
|[5.0,0.4,0.5,4.3,0.046,29.0,80.0,0.9902,3.49,0.66,13.6]  |medium|
|[5.0,0.74,0.0,1.2,0.041,16.0,46.0,0.99258,4.01,0.59,12.5]|medium|
|[5.1,0.42,0.0,1.8,0.044,18.0,88.0,0.99157,3.68,0.73,13.6]|high  |
+-------------------------------------------------------+------+
only showing top 5 rows

+-------------------------------------------------------+------+
|features                                               |label |
+-------------------------------------------------------+------+
|[4.6,0.52,0.15,2.1,0.054,8.0,65.0,0.9934,3.9,0.56,13.1]  |low   |
|[4.9,0.42,0.0,2.1,0.048,16.0,42.0,0.99154,3.71,0.74,14.0]|high  |
|[5.0,0.42,0.24,2.0,0.06,19.0,50.0,0.9917,3.72,0.74,14.0] |high  |
|[5.0,1.02,0.04,1.4,0.045,41.0,85.0,0.9938,3.75,0.48,10.5]|low   |
|[5.0,1.04,0.24,1.6,0.05,32.0,96.0,0.9934,3.74,0.62,11.5] |medium|
+-------------------------------------------------------+------+
only showing top 5 rows
```

6. Fit Multinomial logisticRegression Classification Model

```
from pyspark.ml.classification import LogisticRegression
logr = LogisticRegression(featuresCol='indexedFeatures', labelCol=
↪'indexedLabel')
```

7. Pipeline Architecture

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

```
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, logr,
↪labelConverter])
```

```
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

8. Make predictions

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

```
+--------------------+------+--------------+
|            features| label|predictedLabel|
+--------------------+------+--------------+
```

(continues on next page)

(continued from previous page)

```
|[4.6,0.52,0.15,2....|   low|        medium|
|[4.9,0.42,0.0,2.1...|  high|          high|
|[5.0,0.42,0.24,2....|  high|          high|
|[5.0,1.02,0.04,1....|   low|        medium|
|[5.0,1.04,0.24,1....|medium|        medium|
+--------------------+------+--------------+
only showing top 5 rows
```

9. Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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

```
Test Error = 0.181287
```

```
lrModel = model.stages[2]
trainingSummary = lrModel.summary

# Obtain the objective per iteration
# objectiveHistory = trainingSummary.objectiveHistory
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)

# Obtain the receiver-operating characteristic as a dataframe and␣
↪areaUnderROC.
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

# Set the model threshold to maximize F-Measure
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').
↪head(5)
# bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-
↪Measure)']) \
#     .select('threshold').head()['threshold']
# lr.setThreshold(bestThreshold)
```

You can use `z.show()` to get the data and plot the ROC curves:

You can also register a TempTable `data.registerTempTable('roc_data')` and then use `sql` to plot the ROC curve:

10. visualization

%pyspark
z.show(roc_data)
FINISHED
settings
TPR
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
Took 0 sec. Last updated by anonymous at October 03 2017, 10:53:50 AM. (outdated)

%pyspark
z.show(roc_data)
FINISHED
settings
Stacked
Stream
Expanded
TPR
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
Took 0 sec. Last updated by anonymous at October 03 2017, 10:55:26 AM. (outdated)

%sql
select *
from roc_data
FINISHED
settings
TPR
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Took 0 sec. Last updated by anonymous at October 03 2017, 10:30:13 AM. (outdated)

%sql
select *
from roc_data
FINISHED
settings
Stacked
Stream
Expanded
TPR
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Took 0 sec. Last updated by anonymous at October 03 2017, 10:38:18 AM. (outdated)

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

```
['medium', 'high', 'low']
```

```
from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()
```

(continues on next page)

(continued from previous page)

```
y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix
```

```
array([[526,  11,   2],
       [ 73,  33,   0],
       [ 38,   0,   1]])
```

```
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[526  11   2]
 [ 73  33   0]
 [ 38   0   1]]
```

```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
```

(continues on next page)

(continued from previous page)

```
                        title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[0.97588126 0.02040816 0.00371058]
 [0.68867925 0.31132075 0.        ]
 [0.97435897 0.         0.02564103]]
```

10.3 Decision tree Classification

10.3.1 Introduction

10.3.2 Demo

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

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession
```

(continues on next page)

(continued from previous page)

```
        spark = SparkSession \
            .builder \
            .appName("Python Spark Decision Tree classification") \
            .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/WineData2.csv",header=True);
df.show(5,True)
```

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

```
# 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
```

(continues on next page)

(continued from previous page)

```
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())
```

```
df = df.withColumn("quality", quality_udf("quality"))
df.show(5,True)
df.printSchema()
```

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

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

3. Convert the data to dense vector

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in complex dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):
```

(continues on next page)

(continued from previous page)

```
    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
```

(continues on next page)

(continued from previous page)

```
                    for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

```
# !!!!caution: not from pyspark.mllib.linalg import Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
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|
+--------------------+------+
|[7.4,0.7,0.0,1.9,...|medium|
|[7.8,0.88,0.0,2.6...|medium|
|[7.8,0.76,0.04,2....|medium|
|[11.2,0.28,0.56,1...|medium|
|[7.4,0.7,0.0,1.9,...|medium|
+--------------------+------+
only showing top 5 rows
```

5. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+------------+
|            features| label|indexedLabel|
+--------------------+------+------------+
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
|[7.8,0.88,0.0,2.6...|medium|         0.0|
|[7.8,0.76,0.04,2....|medium|         0.0|
|[11.2,0.28,0.56,1...|medium|         0.0|
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
+--------------------+------+------------+
only showing top 5 rows
```

```
    # Automatically identify categorical features, and index them.
    # Set maxCategories so features with > 4 distinct values are treated as␣
↪continuous.
    featureIndexer =VectorIndexer(inputCol="features", \
                                  outputCol="indexedFeatures", \
                                  maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+--------------------+
|            features| label|     indexedFeatures|
+--------------------+------+--------------------+
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+--------------------+------+--------------------+
only showing top 5 rows
```

6. Split the data to training and test data sets

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

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

```
+--------------------+------+
|            features| label|
+--------------------+------+
|[4.6,0.52,0.15,2....|   low|
|[4.7,0.6,0.17,2.3...|medium|
|[5.0,1.02,0.04,1....|   low|
|[5.0,1.04,0.24,1....|medium|
|[5.1,0.585,0.0,1....|  high|
+--------------------+------+
only showing top 5 rows

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

(continues on next page)

(continued from previous page)

```
+--------------------+------+
|[4.9,0.42,0.0,2.1...|  high|
|[5.0,0.38,0.01,1....|medium|
|[5.0,0.4,0.5,4.3,...|medium|
|[5.0,0.42,0.24,2....|  high|
|[5.0,0.74,0.0,1.2...|medium|
+--------------------+------+
only showing top 5 rows
```

7. Fit Decision Tree Classification Model

```
from pyspark.ml.classification import DecisionTreeClassifier

# Train a DecisionTree model
dTree = DecisionTreeClassifier(labelCol='indexedLabel', featuresCol=
↪'indexedFeatures')
```

8. Pipeline Architecture

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

```
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dTree,
↪labelConverter])
```

```
# Train model.  This also runs the indexers.
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)
```

```
+--------------------+------+--------------+
|            features| label|predictedLabel|
+--------------------+------+--------------+
|[4.9,0.42,0.0,2.1...|  high|          high|
|[5.0,0.38,0.01,1....|medium|        medium|
|[5.0,0.4,0.5,4.3,...|medium|        medium|
|[5.0,0.42,0.24,2....|  high|        medium|
|[5.0,0.74,0.0,1.2...|medium|        medium|
+--------------------+------+--------------+
only showing top 5 rows
```

10. Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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

rfModel = model.stages[-2]
print(rfModel)  # summary only
```

```
Test Error = 0.45509
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_
↪4545ac8dca9c8438ef2a)
of depth 5 with 59 nodes
```

11. visualization

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

(continues on next page)

(continued from previous page)

```
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

```
['medium', 'high', 'low']
```

```
from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix
```

```
array([[497,  29,   7],
       [ 40,  42,   0],
       [ 22,   0,   2]])
```

```
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[497  29   7]
 [ 40  42   0]
 [ 22   0   2]]
```

```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[ 0.93245779  0.05440901  0.01313321]
```

(continues on next page)

(continued from previous page)

```
 [ 0.48780488  0.51219512  0.        ]
 [ 0.91666667  0.          0.08333333]]
```

10.4 Random forest Classification

10.4.1 Introduction

10.4.2 Demo

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

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

        spark = SparkSession \
            .builder \
            .appName("Python Spark Decision Tree classification") \
            .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/WineData2.csv",header=True);
df.show(5,True)
```

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

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

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

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
```

(continues on next page)

(continued from previous page)

```
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

3. Convert the data to dense vector

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in complex dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
```

(continues on next page)

(continued from previous page)

```
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

```
# !!!!caution: not from pyspark.mllib.linalg import Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
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|
+--------------------+------+
```

(continues on next page)

(continued from previous page)

```
|[7.4,0.7,0.0,1.9,...|medium|
|[7.8,0.88,0.0,2.6...|medium|
|[7.8,0.76,0.04,2....|medium|
|[11.2,0.28,0.56,1...|medium|
|[7.4,0.7,0.0,1.9,...|medium|
+--------------------+------+
only showing top 5 rows
```

5. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+------------+
|            features| label|indexedLabel|
+--------------------+------+------------+
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
|[7.8,0.88,0.0,2.6...|medium|         0.0|
|[7.8,0.76,0.04,2....|medium|         0.0|
|[11.2,0.28,0.56,1...|medium|         0.0|
|[7.4,0.7,0.0,1.9,...|medium|         0.0|
+--------------------+------+------------+
only showing top 5 rows
```

```
    # Automatically identify categorical features, and index them.
    # Set maxCategories so features with > 4 distinct values are treated as␣
↪continuous.
    featureIndexer =VectorIndexer(inputCol="features", \
                                  outputCol="indexedFeatures", \
                                  maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+------+--------------------+
|            features| label|     indexedFeatures|
+--------------------+------+--------------------+
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+--------------------+------+--------------------+
only showing top 5 rows
```

6. Split the data to training and test data sets

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

(continues on next page)

(continued from previous page)

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

```
+--------------------+------+
|            features| label|
+--------------------+------+
|[4.6,0.52,0.15,2....|   low|
|[4.7,0.6,0.17,2.3...|medium|
|[5.0,1.02,0.04,1....|   low|
|[5.0,1.04,0.24,1....|medium|
|[5.1,0.585,0.0,1....|  high|
+--------------------+------+
only showing top 5 rows

+--------------------+------+
|            features| label|
+--------------------+------+
|[4.9,0.42,0.0,2.1...|  high|
|[5.0,0.38,0.01,1....|medium|
|[5.0,0.4,0.5,4.3,...|medium|
|[5.0,0.42,0.24,2....|  high|
|[5.0,0.74,0.0,1.2...|medium|
+--------------------+------+
only showing top 5 rows
```

7. Fit Random Forest Classification Model

```
from pyspark.ml.classification import RandomForestClassifier

# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol=
↪"indexedFeatures", numTrees=10)
```

8. Pipeline Architecture

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

```
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf,labelConverter])
```

```
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

9. Make predictions

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

(continues on next page)

(continued from previous page)

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

```
+--------------------+------+--------------+
|            features| label|predictedLabel|
+--------------------+------+--------------+
|[4.9,0.42,0.0,2.1...|  high|          high|
|[5.0,0.38,0.01,1....|medium|        medium|
|[5.0,0.4,0.5,4.3,...|medium|        medium|
|[5.0,0.42,0.24,2....|  high|        medium|
|[5.0,0.74,0.0,1.2...|medium|        medium|
+--------------------+------+--------------+
only showing top 5 rows
```

10. Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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

rfModel = model.stages[-2]
print(rfModel)  # summary only
```

```
Test Error = 0.173502
RandomForestClassificationModel (uid=rfc_a3395531f1d2) with 10 trees
```

11. visualization

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
```

(continues on next page)

(continued from previous page)

```
    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

```
['medium', 'high', 'low']
```

```
from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix
```

```
array([[502,   9,   0],
       [ 73,  22,   0],
       [ 28,   0,   0]])
```

```
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[502   9   0]
 [ 73  22   0]
 [ 28   0   0]]
```

```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[ 0.98238748  0.01761252  0.        ]
 [ 0.76842105  0.23157895  0.        ]
 [ 1.          0.          0.        ]]
```

10.5 Gradient-boosted tree Classification

10.5.1 Introduction

10.5.2 Demo

- 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.

10.6 XGBoost: Gradient-boosted tree Classification

10.6.1 Introduction

10.6.2 Demo

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

Warning: Unfortunately, I didn't find a good way to setup the XGBoost directly in Spark. But I do get the XGBoost work with `pysparkling` on my machine.

1. Start H2O cluster inside the Spark environment

```
from pysparkling import *
hc = H2OContext.getOrCreate(spark)
```

```
Connecting to H2O server at http://192.168.0.102:54323... successful.
H2O cluster uptime:     07 secs
H2O cluster timezone:   America/Chicago
H2O data parsing timezone:      UTC
H2O cluster version:    3.22.1.3
H2O cluster version age:        20 days
H2O cluster name:       sparkling-water-dt216661_local-1550259209801
H2O cluster total nodes:        1
H2O cluster free memory:        848 Mb
H2O cluster total cores:        8
H2O cluster allowed cores:      8
H2O cluster status:     accepting new members, healthy
H2O connection url:     http://192.168.0.102:54323
H2O connection proxy:   None
H2O internal security:  False
H2O API Extensions:     XGBoost, Algos, AutoML, Core V3, Core V4
Python version: 3.7.1 final

Sparkling Water Context:
 * H2O name: sparkling-water-dt216661_local-1550259209801
 * cluster size: 1
 * list of used nodes:
  (executorId, host, port)
  ------------------------
  (driver,192.168.0.102,54323)
  ------------------------

  Open H2O Flow in browser: http://192.168.0.102:54323 (CMD + click in Mac␣
↪OSX)
```

2. Parse the data using H2O and convert them to Spark Frame

```
import h2o
frame = h2o.import_file("https://raw.githubusercontent.com/h2oai/sparkling-
↪water/master/examples/smalldata/prostate/prostate.csv")
spark_frame = hc.as_spark_frame(frame)
```

```
Parse progress: |[redacted]| 100%
```

```
spark_frame.show(4)
```

```
+---+-------+---+----+-----+-----+----+----+-------+
| ID|CAPSULE|AGE|RACE|DPROS|DCAPS| PSA| VOL|GLEASON|
+---+-------+---+----+-----+-----+----+----+-------+
|  1|      0| 65|   1|    2|    1| 1.4| 0.0|      6|
|  2|      0| 72|   1|    3|    2| 6.7| 0.0|      7|
|  3|      0| 70|   1|    1|    2| 4.9| 0.0|      6|
|  4|      0| 76|   2|    2|    1|51.2|20.0|      7|
+---+-------+---+----+-----+-----+----+----+-------+
only showing top 4 rows
```

3. Train the model

```
from pysparkling.ml import H2OXGBoost
estimator = H2OXGBoost(predictionCol="AGE")
model = estimator.fit(spark_frame)
```

4. Run Predictions

```
predictions = model.transform(spark_frame)
predictions.show(4)
```

```
+---+-------+---+----+-----+-----+----+----+-------+-------------------+
| ID|CAPSULE|AGE|RACE|DPROS|DCAPS| PSA| VOL|GLEASON|  prediction_output|
+---+-------+---+----+-----+-----+----+----+-------+-------------------+
|  1|      0| 65|   1|    2|    1| 1.4| 0.0|      6|[64.85852813720703]|
|  2|      0| 72|   1|    3|    2| 6.7| 0.0|      7| [72.0611801147461]|
|  3|      0| 70|   1|    1|    2| 4.9| 0.0|      6|[70.26496887207031]|
|  4|      0| 76|   2|    2|    1|51.2|20.0|      7|[75.26521301269531]|
+---+-------+---+----+-----+-----+----+----+-------+-------------------+
only showing top 4 rows
```

10.7 Naive Bayes Classification

10.7.1 Introduction

10.7.2 Demo

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

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark  Naive Bayes classification") \
    .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/WineData2.csv",header=True);
df.show(5)
```

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

```
df.printSchema()
```

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

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

#
def condition(r):
    if (0<= r <= 6):
        label = "low"
```

(continues on next page)

(continued from previous page)

```
    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"))

df.show(5,True)
```

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

```
df.printSchema()
```

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
 |-- sugar: double (nullable = true)
 |-- chlorides: double (nullable = true)
 |-- free: double (nullable = true)
 |-- total: double (nullable = true)
 |-- density: double (nullable = true)
 |-- pH: double (nullable = true)
 |-- sulphates: double (nullable = true)
 |-- alcohol: double (nullable = true)
 |-- quality: string (nullable = true)
```

3. Deal with categorical data and Convert the data to dense vector

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in complex dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
```

(continues on next page)

(continued from previous page)

```
                    for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                    outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                    for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

```
def get_dummy(df,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder␣
↪in encoders]
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select('features','label')
```

4. Transform the dataset to DataFrame

```
from pyspark.ml.linalg import Vectors # !!!!caution: not from pyspark.mllib.
↪linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString,StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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|
+--------------------+-----+
|[7.4,0.7,0.0,1.9,...|  low|
|[7.8,0.88,0.0,2.6...|  low|
|[7.8,0.76,0.04,2....|  low|
|[11.2,0.28,0.56,1...|  low|
|[7.4,0.7,0.0,1.9,...|  low|
+--------------------+-----+
only showing top 5 rows
```

4. Deal with Categorical Label and Variables

```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+--------------------+-----+------------+
|            features|label|indexedLabel|
+--------------------+-----+------------+
|[7.4,0.7,0.0,1.9,...|  low|         0.0|
|[7.8,0.88,0.0,2.6...|  low|         0.0|
|[7.8,0.76,0.04,2....|  low|         0.0|
|[11.2,0.28,0.56,1...|  low|         0.0|
|[7.4,0.7,0.0,1.9,...|  low|         0.0|
+--------------------+-----+------------+
only showing top 5 rows
```

```
# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as␣
↪continuous.
featureIndexer =VectorIndexer(inputCol="features", \
                              outputCol="indexedFeatures", \
                              maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+--------------------+-----+--------------------+
|            features|label|     indexedFeatures|
+--------------------+-----+--------------------+
|[7.4,0.7,0.0,1.9,...|  low|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|  low|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|  low|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|  low|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|  low|[7.4,0.7,0.0,1.9,...|
+--------------------+-----+--------------------+
only showing top 5 rows
```

5. Split the data to training and test data sets

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

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

```
+-------------------------------------------------------------+-----+
|features                                                     |label|
+-------------------------------------------------------------+-----+
|[5.0,0.38,0.01,1.6,0.048,26.0,60.0,0.99084,3.7,0.75,14.0]|low  |
|[5.0,0.42,0.24,2.0,0.06,19.0,50.0,0.9917,3.72,0.74,14.0] |high |
|[5.0,0.74,0.0,1.2,0.041,16.0,46.0,0.99258,4.01,0.59,12.5]|low  |
|[5.0,1.02,0.04,1.4,0.045,41.0,85.0,0.9938,3.75,0.48,10.5]|low  |
|[5.0,1.04,0.24,1.6,0.05,32.0,96.0,0.9934,3.74,0.62,11.5] |low  |
+-------------------------------------------------------------+-----+
only showing top 5 rows

+-------------------------------------------------------------+-----+
|features                                                     |label|
+-------------------------------------------------------------+-----+
|[4.6,0.52,0.15,2.1,0.054,8.0,65.0,0.9934,3.9,0.56,13.1]  |low  |
|[4.7,0.6,0.17,2.3,0.058,17.0,106.0,0.9932,3.85,0.6,12.9] |low  |
|[4.9,0.42,0.0,2.1,0.048,16.0,42.0,0.99154,3.71,0.74,14.0]|high |
|[5.0,0.4,0.5,4.3,0.046,29.0,80.0,0.9902,3.49,0.66,13.6]  |low  |
|[5.2,0.49,0.26,2.3,0.09,23.0,74.0,0.9953,3.71,0.62,12.2] |low  |
+-------------------------------------------------------------+-----+
only showing top 5 rows
```

6. Fit Naive Bayes Classification Model

```
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(featuresCol='indexedFeatures', labelCol='indexedLabel')
```

7. Pipeline Architecture

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

```
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, nb,labelConverter])
```

```
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

8. Make predictions

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

```
+--------------------+-----+--------------+
|            features|label|predictedLabel|
+--------------------+-----+--------------+
|[4.6,0.52,0.15,2....|  low|           low|
|[4.7,0.6,0.17,2.3...|  low|           low|
|[4.9,0.42,0.0,2.1...| high|           low|
|[5.0,0.4,0.5,4.3,...|  low|           low|
|[5.2,0.49,0.26,2....|  low|           low|
+--------------------+-----+--------------+
only showing top 5 rows
```

9. Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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

```
Test Error = 0.307339
```

```
lrModel = model.stages[2]
trainingSummary = lrModel.summary

# Obtain the objective per iteration
# objectiveHistory = trainingSummary.objectiveHistory
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)

# Obtain the receiver-operating characteristic as a dataframe and␣
↪areaUnderROC.
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))
```

(continues on next page)

(continued from previous page)

```
# Set the model threshold to maximize F-Measure
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').
↪head(5)
# bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-
↪Measure)']) \
#     .select('threshold').head()['threshold']
# lr.setThreshold(bestThreshold)
```

You can use `z.show()` to get the data and plot the ROC curves:

You can also register a TempTable `data.registerTempTable('roc_data')` and then use `sql` to plot the ROC curve:

10. visualization

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
```

(continues on next page)

(continued from previous page)

```
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
class_temp = predictions.select("label").groupBy("label")\
                        .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

```
['low', 'high']
```

```
from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred,labels=class_names)
cnf_matrix
```

```
array([[392, 169],
       [ 32,  61]])
```

```
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[392 169]
 [ 32  61]]
```

```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[0.69875223 0.30124777]
 [0.34408602 0.65591398]]
```

Normalized confusion matrix
low
high
True label
0.70
0.30
0.34
0.66
low
high
Predicted label
0.65
0.60
0.55
0.50
0.45
0.40
0.35

CHAPTER

ELEVEN

CLUSTERING

Chinese proverb

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

The above figure was generated by the code from: Python Data Science Handbook.

11.1 K-Means Model

11.1.1 Introduction

11.1.2 Demo

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark K-means 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/iris.csv",header=True);
```

check the data set

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

Then you will get

```
+------------+-----------+------------+-----------+-------+
|sepal_length|sepal_width|petal_length|petal_width|species|
+------------+-----------+------------+-----------+-------+
|         5.1|        3.5|         1.4|        0.2| setosa|
|         4.9|        3.0|         1.4|        0.2| setosa|
|         4.7|        3.2|         1.3|        0.2| setosa|
|         4.6|        3.1|         1.5|        0.2| setosa|
|         5.0|        3.6|         1.4|        0.2| setosa|
+------------+-----------+------------+-----------+-------+
only showing top 5 rows

root
 |-- sepal_length: double (nullable = true)
 |-- sepal_width: double (nullable = true)
 |-- petal_length: double (nullable = true)
 |-- petal_width: double (nullable = true)
 |-- species: string (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|      sepal_length|        sepal_width|      petal_length|      ␣
↪petal_width|  species|
+-------+------------------+-------------------+------------------+-----------
↪-------+---------+
|  count|               150|                150|               150|          ␣
↪    150|      150|
|   mean| 5.843333333333335| 3.0540000000000007|3.7586666666666693|1.
↪1986666666666672|     null|
| stddev|0.8280661279778637|0.43359431136217375| 1.764420419952262|0.
↪7631607417008414|     null|
|    min|               4.3|                2.0|               1.0|          ␣
↪    0.1|   setosa|
|    max|               7.9|                4.4|               6.9|          ␣
↪    2.5|virginica|
```

(continues on next page)

(continued from previous page)

```
+-------+-----------------+-------------------+-----------------+-----------
↪-------+---------+
```

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

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

Note: You are strongly encouraged to try my `get_dummy` function for dealing with the categorical data in complex dataset.

Supervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,␣
↪VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol()␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select(indexCol,'features','label')
```

Unsupervised learning version:

```
def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    Get dummy variables and concat with continuous variables for␣
↪unsupervised learning.
```

(continues on next page)

(continued from previous page)

```
    :param df: the dataframe
    :param categoricalCols: the name list of the categorical data
    :param continuousCols:  the name list of the numerical data
    :return k: feature matrix

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol(),␣
↪for encoder in encoders]
                                + continuousCols, outputCol="features
↪")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol,'features')
```

4. Transform the dataset to DataFrame

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

```
+-----------------+
|features         |
+-----------------+
|[5.1,3.5,1.4,0.2]|
|[4.9,3.0,1.4,0.2]|
|[4.7,3.2,1.3,0.2]|
|[4.6,3.1,1.5,0.2]|
|[5.0,3.6,1.4,0.2]|
+-----------------+
only showing top 5 rows
```

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|  indexedFeatures|
+-----------------+-----------------+
|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|
|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|
|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|
|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|
|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|
+-----------------+-----------------+
only showing top 5 rows
```

6. Elbow method to determine the optimal number of clusters for k-means clustering

```
import numpy as np
cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans()\
            .setK(k)\
            .setSeed(1) \
            .setFeaturesCol("indexedFeatures")\
            .setPredictionCol("cluster")

    model = kmeans.fit(data)
    cost[k] = model.computeCost(data) # requires Spark 2.0 or later
```

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sbs
from matplotlib.ticker import MaxNLocator
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20])
ax.set_xlabel('k')
ax.set_ylabel('cost')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

In my opinion, sometimes it's hard to choose the optimal number of the clusters by using the `elbow method`. As shown in the following Figure, you can choose 3, 5 or even 8. I will choose `3` in this demo.

- Silhouette analysis

```
#PySpark libraries
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce  # For Python 3.x
```

(continues on next page)

70
60
50
40
30
20
10
0
cost
2
4
6
8
10
12
14
16
18
k

(continued from previous page)

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

def optimal_k(df_in,index_col,k_min, k_max,num_runs):
    '''
    Determine optimal number of clusters by using Silhoutte Score Analysis.
    :param df_in: the input dataframe
    :param index_col: the name of the index column
    :param k_min: the train dataset
    :param k_min: the minmum number of the clusters
    :param k_max: the maxmum number of the clusters
    :param num_runs: the number of runs for each fixed clusters

    :return k: optimal number of the clusters
    :return silh_lst: Silhouette score
    :return r_table: the running results table

    :author: Wenqiang Feng
    :email:  von198@gmail.com
    '''

    start = time.time()
    silh_lst = []
    k_lst = np.arange(k_min, k_max+1)

    r_table = df_in.select(index_col).toPandas()
    r_table = r_table.set_index(index_col)
    centers = pd.DataFrame()

    for k in k_lst:
        silh_val = []
        for run in np.arange(1, num_runs+1):

            # Trains a k-means model.
            kmeans = KMeans()\
                    .setK(k)\
                    .setSeed(int(np.random.randint(100, size=1)))
            model = kmeans.fit(df_in)

            # Make predictions
            predictions = model.transform(df_in)
            r_table['cluster_{k}_{run}'.format(k=k, run=run)]= predictions.
↪select('prediction').toPandas()

            # Evaluate clustering by computing Silhouette score
            evaluator = ClusteringEvaluator()
            silhouette = evaluator.evaluate(predictions)
            silh_val.append(silhouette)

        silh_array=np.asanyarray(silh_val)
        silh_lst.append(silh_array.mean())
```

(continues on next page)

(continued from previous page)

```
    elapsed =  time.time() - start

    silhouette = pd.DataFrame(list(zip(k_lst,silh_lst)),columns = ['k',
↪'silhouette'])

    print('+-------------------------------------------------------------+')
    print("|          The finding optimal k phase took %8.0f s.        |"
↪%(elapsed))
    print('+-------------------------------------------------------------+')


    return k_lst[np.argmax(silh_lst, axis=0)], silhouette , r_table
```

```
k, silh_lst, r_table = optimal_k(scaledData,index_col,k_min, k_max,num_runs)

+-------------------------------------------------------------+
|          The finding optimal k phase took     1783 s.       |
+-------------------------------------------------------------+
```

```
spark.createDataFrame(silh_lst).show()

+---+------------------+
|  k|        silhouette|
+---+------------------+
|  3|0.8045154385557953|
|  4|0.6993528775512052|
|  5|0.6689286654221447|
|  6|0.6356184024841809|
|  7|0.7174102265711756|
|  8|0.6720861758298997|
|  9| 0.601771359881241|
| 10|0.6292447334578428|
+---+------------------+
```

From the silhouette list, we can choose `3` as the optimal number of the clusters.

Warning: `ClusteringEvaluator` in `pyspark.ml.evaluation` requires Spark 2.4 or later!!

7. Pipeline Architecture

```
from pyspark.ml.clustering import KMeans, KMeansModel

kmeans = KMeans() \
          .setK(3) \
          .setFeaturesCol("indexedFeatures")\
          .setPredictionCol("cluster")

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

(continues on next page)

(continued from previous page)

```
model = pipeline.fit(transformed)

cluster = model.transform(transformed)
```

8. k-means clusters

```
cluster = model.transform(transformed)
```

```
+-----------------+-----------------+-------+
|         features|  indexedFeatures|cluster|
+-----------------+-----------------+-------+
|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|      1|
|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|      1|
|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|      1|
|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|      1|
|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|      1|
|[5.4,3.9,1.7,0.4]|[5.4,3.9,1.7,0.4]|      1|
|[4.6,3.4,1.4,0.3]|[4.6,3.4,1.4,0.3]|      1|
|[5.0,3.4,1.5,0.2]|[5.0,3.4,1.5,0.2]|      1|
|[4.4,2.9,1.4,0.2]|[4.4,2.9,1.4,0.2]|      1|
|[4.9,3.1,1.5,0.1]|[4.9,3.1,1.5,0.1]|      1|
|[5.4,3.7,1.5,0.2]|[5.4,3.7,1.5,0.2]|      1|
|[4.8,3.4,1.6,0.2]|[4.8,3.4,1.6,0.2]|      1|
|[4.8,3.0,1.4,0.1]|[4.8,3.0,1.4,0.1]|      1|
|[4.3,3.0,1.1,0.1]|[4.3,3.0,1.1,0.1]|      1|
|[5.8,4.0,1.2,0.2]|[5.8,4.0,1.2,0.2]|      1|
|[5.7,4.4,1.5,0.4]|[5.7,4.4,1.5,0.4]|      1|
|[5.4,3.9,1.3,0.4]|[5.4,3.9,1.3,0.4]|      1|
|[5.1,3.5,1.4,0.3]|[5.1,3.5,1.4,0.3]|      1|
|[5.7,3.8,1.7,0.3]|[5.7,3.8,1.7,0.3]|      1|
|[5.1,3.8,1.5,0.3]|[5.1,3.8,1.5,0.3]|      1|
+-----------------+-----------------+-------+
only showing top 20 rows
```

CHAPTER

TWELVE

RFM ANALYSIS

Segment	RFM	Description	Marketing
Best Customers	111	Bought most recently and most often, and spend the most	No price incentives, new products, and loyalty programs
Loyal Customers	X1X	Buy most frequently	Use R and M to further segment
Big Spenders	XX1	Spend the most	Market your most expensive products
Almost Lost	311	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Customers	411	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Cheap Customers	444	Last purchased long ago, purchased few, and spent little	Don't spend too much trying to re-acquire

The above figure source: Blast Analytics Marketing

RFM is a method used for analyzing customer value. It is commonly used in database marketing and direct marketing and has received particular attention in retail and professional services industries. More details

can be found at Wikipedia RFM_wikipedia.

RFM stands for the three dimensions:

- Recency – How recently did the customer purchase? i.e. Duration since last purchase
- Frequency – How often do they purchase? i.e. Total number of purchases
- Monetary Value – How much do they spend? i.e. Total money this customer spent

12.1 RFM Analysis Methodology

RFM Analysis contains three main steps:

12.1.1 1. Build the RFM features matrix for each customer

```
+----------+-------+---------+---------+
|CustomerID|Recency|Frequency| Monetary|
+----------+-------+---------+---------+
|     14911|      1|      248|132572.62|
|     12748|      0|      224|  29072.1|
|     17841|      1|      169| 40340.78|
|     14606|      1|      128| 11713.85|
|     15311|      0|      118| 59419.34|
+----------+-------+---------+---------+
only showing top 5 rows
```

12.1.2 2. Determine cutting points for each feature

```
+----------+-------+---------+--------+-----+-----+-----+
|CustomerID|Recency|Frequency|Monetary|r_seg|f_seg|m_seg|
+----------+-------+---------+--------+-----+-----+-----+
|     17420|     50|        3|  598.83|    2|    3|    2|
|     16861|     59|        3|  151.65|    3|    3|    1|
|     16503|    106|        5| 1421.43|    3|    2|    3|
|     15727|     16|        7| 5178.96|    1|    1|    4|
|     17389|      0|       43|31300.08|    1|    1|    4|
+----------+-------+---------+--------+-----+-----+-----+
only showing top 5 rows
```

12.1.3 3. Determine the RFM scores and summarize the corresponding business value

```
+----------+-------+---------+--------+-----+-----+-----+--------+
|CustomerID|Recency|Frequency|Monetary|r_seg|f_seg|m_seg|RFMScore|
+----------+-------+---------+--------+-----+-----+-----+--------+
```

(continues on next page)

(continued from previous page)

```
|     17988|     11|        8|  191.17|    1|    1|    1|     111|
|     16892|      1|        7|  496.84|    1|    1|    2|     112|
|     16668|     15|        6|  306.72|    1|    1|    2|     112|
|     16554|      3|        7|  641.55|    1|    1|    2|     112|
|     16500|      4|        6|  400.86|    1|    1|    2|     112|
+----------+-------+---------+--------+-----+-----+-----+--------+
only showing top 5 rows
```

The corresponding business description and marketing value:

Segment	RFM	Description	Marketing
Best Customers	111	Bought most recently and most often, and spend the most	No price incentives, new products, and loyalty programs
Loyal Customers	X1X	Buy most frequently	Use R and M to further segment
Big Spenders	XX1	Spend the most	Market your most expensive products
Almost Lost	311	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Customers	411	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Cheap Customers	444	Last purchased long ago, purchased few, and spent little	Don't spend too much trying to re-acquire

Fig. 1: Source: Blast Analytics Marketing

12.2 Demo

- The Jupyter notebook can be download from Data Exploration.
- The data can be downloaf from German Credit.

12.2.1 Load and clean data

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

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

2. Load dataset

```
df_raw = spark.read.format('com.databricks.spark.csv').\
                      options(header='true', \
                      inferschema='true').\
            load("Online Retail.csv",header=True);
```

check the data set

```
df_raw.show(5)
df_raw.printSchema()
```

Then you will get

```
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+
|InvoiceNo|StockCode|          Description|Quantity|␣
↪InvoiceDate|UnitPrice|CustomerID|        Country|
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/10 8:26|     2.55|  ␣
↪ 17850|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/10 8:26|     3.39|  ␣
↪ 17850|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/10 8:26|     2.75|  ␣
↪ 17850|United Kingdom|
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/10 8:26|     3.39|  ␣
↪ 17850|United Kingdom|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/10 8:26|     3.39|  ␣
↪ 17850|United Kingdom|
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+
only showing top 5 rows
```

(continues on next page)

(continued from previous page)

```
root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: integer (nullable = true)
 |-- Country: string (nullable = true)
```

3. Data clean and data manipulation

- check and remove the `null` values

```
from pyspark.sql.functions import count

def my_count(df_in):
    df_in.agg( *[ count(c).alias(c) for c in df_in.columns ] ).show()
```

```
my_count(df_raw)
```

```
+---------+---------+-----------+--------+-----------+---------+----------+---
↪----+
|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---------+---------+-----------+--------+-----------+---------+----------+---
↪----+
|   541909|   541909|     540455|  541909|     541909|   541909|    406829|␣
↪541909|
+---------+---------+-----------+--------+-----------+---------+----------+---
↪----+
```

Since the count results are not the same, we have some null value in the `CustomerID` column. We can drop these records from the dataset.

```
df = df_raw.dropna(how='any')
my_count(df)
```

```
+---------+---------+-----------+--------+-----------+---------+----------+---
↪----+
|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---------+---------+-----------+--------+-----------+---------+----------+---
↪----+
|   406829|   406829|     406829|  406829|     406829|   406829|    406829|␣
↪406829|
+---------+---------+-----------+--------+-----------+---------+----------+---
↪----+
```

- Dealwith the InvoiceDate

```
from pyspark.sql.functions import to_utc_timestamp, unix_timestamp, lit,␣
↪datediff, col

timeFmt = "MM/dd/yy HH:mm"

df = df.withColumn('NewInvoiceDate'
                  , to_utc_timestamp(unix_timestamp(col('InvoiceDate'),
↪timeFmt).cast('timestamp')
                  , 'UTC'))
```

```
df.show(5)

+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+--------------------+
|InvoiceNo|StockCode|         Description|Quantity|␣
↪InvoiceDate|UnitPrice|CustomerID|       Country|      NewInvoiceDate|
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+--------------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/10 8:26|     2.55|   ␣
↪ 17850|United Kingdom|2010-12-01 08:26:...|
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/10 8:26|     3.39|   ␣
↪ 17850|United Kingdom|2010-12-01 08:26:...|
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/10 8:26|     2.75|   ␣
↪ 17850|United Kingdom|2010-12-01 08:26:...|
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/10 8:26|     3.39|   ␣
↪ 17850|United Kingdom|2010-12-01 08:26:...|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/10 8:26|     3.39|   ␣
↪ 17850|United Kingdom|2010-12-01 08:26:...|
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+--------------------+
only showing top 5 rows
```

Warning: The spark is pretty sensitive to the date format!

- calculate total price

```
from pyspark.sql.functions import round

df = df.withColumn('TotalPrice', round( df.Quantity * df.UnitPrice, 2 ) )
```

- calculate the time difference

```
from pyspark.sql.functions import mean, min, max, sum, datediff, to_date

date_max = df.select(max('NewInvoiceDate')).toPandas()
current = to_utc_timestamp( unix_timestamp(lit(str(date_max.iloc[0][0])), \
                            'yy-MM-dd HH:mm').cast('timestamp'), 'UTC' )

# Calculatre Duration
df = df.withColumn('Duration', datediff(lit(current), 'NewInvoiceDate'))
```

- build the Recency, Frequency and Monetary

```
recency = df.groupBy('CustomerID').agg(min('Duration').alias('Recency'))
frequency = df.groupBy('CustomerID', 'InvoiceNo').count()\
                        .groupBy('CustomerID')\
                        .agg(count("*").alias("Frequency"))
monetary = df.groupBy('CustomerID').agg(round(sum('TotalPrice'), 2).alias(
↪'Monetary'))
rfm = recency.join(frequency,'CustomerID', how = 'inner')\
             .join(monetary,'CustomerID', how = 'inner')
```

```
rfm.show(5)

+----------+-------+---------+--------+
|CustomerID|Recency|Frequency|Monetary|
+----------+-------+---------+--------+
|     17420|     50|        3|  598.83|
|     16861|     59|        3|  151.65|
|     16503|    106|        5| 1421.43|
|     15727|     16|        7| 5178.96|
|     17389|      0|       43|31300.08|
+----------+-------+---------+--------+
only showing top 5 rows
```

12.2.2 RFM Segmentation

4. Determine cutting points

In this section, you can use the techniques (statistical results and visualizations) in *Data Exploration* section to help you determine the cutting points for each attribute. In my opinion, the cutting points are mainly depend on the business sense. You's better talk to your makrting people and get feedback and suggestion from them. I will use the quantile as the cutting points in this demo.

```
cols = ['Recency','Frequency','Monetary']
describe_pd(rfm,cols,1)
```

```
+-------+-----------------+-----------------+------------------+
|summary|          Recency|        Frequency|          Monetary|
+-------+-----------------+-----------------+------------------+
|  count|           4372.0|           4372.0|            4372.0|
|   mean|91.58119853613907| 5.07548032936871|1898.4597003659655|
| stddev|100.7721393138483|9.338754163574727| 8219.345141139722|
|    min|              0.0|              1.0|          -4287.63|
|    max|            373.0|            248.0|         279489.02|
|    25%|             16.0|              1.0|293.36249999999995|
|    50%|             50.0|              3.0|           648.075|
|    75%|            143.0|              5.0|          1611.725|
+-------+-----------------+-----------------+------------------+
```

The user defined function by using the cutting points:

```
def RScore(x):
    if  x <= 16:
        return 1
    elif x<= 50:
        return 2
    elif x<= 143:
        return 3
    else:
        return 4

def FScore(x):
    if  x <= 1:
        return 4
    elif x <= 3:
        return 3
    elif x <= 5:
        return 2
    else:
        return 1

def MScore(x):
    if  x <= 293:
        return 4
    elif x <= 648:
        return 3
    elif x <= 1611:
        return 2
    else:
        return 1

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

R_udf = udf(lambda x: RScore(x), StringType())
F_udf = udf(lambda x: FScore(x), StringType())
M_udf = udf(lambda x: MScore(x), StringType())
```

5. RFM Segmentation

```
rfm_seg = rfm.withColumn("r_seg", R_udf("Recency"))
rfm_seg = rfm_seg.withColumn("f_seg", F_udf("Frequency"))
rfm_seg = rfm_seg.withColumn("m_seg", M_udf("Monetary"))
rfm_seg.show(5)
```

```
+----------+-------+---------+--------+-----+-----+-----+
|CustomerID|Recency|Frequency|Monetary|r_seg|f_seg|m_seg|
+----------+-------+---------+--------+-----+-----+-----+
|     17420|     50|        3|  598.83|    2|    3|    2|
|     16861|     59|        3|  151.65|    3|    3|    1|
|     16503|    106|        5| 1421.43|    3|    2|    3|
|     15727|     16|        7| 5178.96|    1|    1|    4|
|     17389|      0|       43|31300.08|    1|    1|    4|
```

(continues on next page)

(continued from previous page)

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

```
rfm_seg = rfm_seg.withColumn('RFMScore',
                             F.concat(F.col('r_seg'),F.col('f_seg'), F.col('m_
↪seg')))
rfm_seg.sort(F.col('RFMScore')).show(5)
```

```
+----------+-------+---------+--------+-----+-----+-----+--------+
|CustomerID|Recency|Frequency|Monetary|r_seg|f_seg|m_seg|RFMScore|
+----------+-------+---------+--------+-----+-----+-----+--------+
|     17988|     11|        8|  191.17|    1|    1|    1|     111|
|     16892|      1|        7|  496.84|    1|    1|    2|     112|
|     16668|     15|        6|  306.72|    1|    1|    2|     112|
|     16554|      3|        7|  641.55|    1|    1|    2|     112|
|     16500|      4|        6|  400.86|    1|    1|    2|     112|
+----------+-------+---------+--------+-----+-----+-----+--------+
only showing top 5 rows
```

12.2.3 Statistical Summary

6. Statistical Summary

- simple summary

```
rfm_seg.groupBy('RFMScore')\
       .agg({'Recency':'mean',
             'Frequency': 'mean',
             'Monetary': 'mean'} )\
        .sort(F.col('RFMScore')).show(5)
```

```
+--------+-----------------+------------------+------------------+
|RFMScore|     avg(Recency)|     avg(Monetary)|    avg(Frequency)|
+--------+-----------------+------------------+------------------+
|     111|             11.0|            191.17|               8.0|
|     112|              8.0|          505.9775|               7.5|
|     113|7.237113402061856|1223.3604123711339| 7.752577319587629|
|     114|6.035123966942149| 8828.888595041324|18.882231404958677|
|     121|              9.6|            207.24|               4.4|
+--------+-----------------+------------------+------------------+
only showing top 5 rows
```

- complex summary

```
grp = 'RFMScore'
num_cols = ['Recency','Frequency','Monetary']
df_input = rfm_seg

quantile_grouped = quantile_agg(df_input,grp,num_cols)
```

(continues on next page)

(continued from previous page)

```
quantile_grouped.toPandas().to_csv(output_dir+'quantile_grouped.csv')

deciles_grouped = deciles_agg(df_input,grp,num_cols)
deciles_grouped.toPandas().to_csv(output_dir+'deciles_grouped.csv')
```

12.3 Extension

You can also apply the K-means clustering in *Clustering* section to do the segmentation.

12.3.1 Build feature matrix

1. build dense feature matrix

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

# 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: [r[0],Vectors.dense(r[1:])]).toDF([
↪'CustomerID','rfm'])
```

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

```
+----------+-------------------+
|CustomerID|                rfm|
+----------+-------------------+
|     17420|  [50.0,3.0,598.83]|
|     16861|  [59.0,3.0,151.65]|
|     16503|[106.0,5.0,1421.43]|
|     15727| [16.0,7.0,5178.96]|
|     17389|[0.0,43.0,31300.08]|
+----------+-------------------+
only showing top 5 rows
```

2. Scaler the feature matrix

```
from pyspark.ml.feature import MinMaxScaler

scaler = MinMaxScaler(inputCol="rfm",\
```

(continues on next page)

(continued from previous page)

```
         outputCol="features")
scalerModel =  scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)
scaledData.show(5,False)
```

```
+----------+-------------------+-----------------------------------------------
↪----------------+
|CustomerID|rfm                |features                                       ␣
↪                |
+----------+-------------------+-----------------------------------------------
↪----------------+
|17420     |[50.0,3.0,598.83]  |[0.13404825737265416,0.008097165991902834,0.
↪01721938714830836]|
|16861     |[59.0,3.0,151.65]  |[0.1581769436997319,0.008097165991902834,0.
↪01564357039241953] |
|16503     |[106.0,5.0,1421.43]|[0.28418230563002683,0.016194331983805668,0.
↪02011814573186342]|
|15727     |[16.0,7.0,5178.96] |[0.04289544235924933,0.024291497975708502,0.
↪03335929858922501]|
|17389     |[0.0,43.0,31300.08]|[0.0,0.1700404858299595,0.12540746393334334] ␣
↪                |
+----------+-------------------+-----------------------------------------------
↪----------------+
only showing top 5 rows
```

12.3.2 K-means clustering

3. Find optimal number of cluster

I will present two popular ways to determine the optimal number of the cluster.

- elbow analysis

```
#PySpark libraries
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce  # For Python 3.x

from pyspark.ml.clustering import KMeans
#from pyspark.ml.evaluation import ClusteringEvaluator  # requires Spark 2.4␣
↪or later

import numpy as np
cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans()\
```

(continues on next page)

(continued from previous page)

```
            .setK(k)\
            .setSeed(1) \
            .setFeaturesCol("features")\
            .setPredictionCol("cluster")

    model = kmeans.fit(scaledData)
    cost[k] = model.computeCost(scaledData) # requires Spark 2.0 or later
```

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sbs
from matplotlib.ticker import MaxNLocator

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20], marker = "o")
ax.set_xlabel('k')
ax.set_ylabel('cost')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

In my opinion, sometimes it's hard to choose the number of the clusters. As shown in Figure *Cost v.s. the number of the clusters*, you can choose 3, 5 or even 8. I will choose 3 in this demo.

- Silhouette analysis

```
#PySpark libraries
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce  # For Python 3.x

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

def optimal_k(df_in,index_col,k_min, k_max,num_runs):
    '''
    Determine optimal number of clusters by using Silhoutte Score Analysis.
    :param df_in: the input dataframe
    :param index_col: the name of the index column
    :param k_min: the train dataset
    :param k_min: the minmum number of the clusters
    :param k_max: the maxmum number of the clusters
    :param num_runs: the number of runs for each fixed clusters

    :return k: optimal number of the clusters
    :return silh_lst: Silhouette score
    :return r_table: the running results table
```

(continues on next page)

Fig. 2: Cost v.s. the number of the clusters

(continued from previous page)

```
    :author: Wenqiang Feng
    :email:  von198@gmail.com.com
    '''

    start = time.time()
    silh_lst = []
    k_lst = np.arange(k_min, k_max+1)

    r_table = df_in.select(index_col).toPandas()
    r_table = r_table.set_index(index_col)
    centers = pd.DataFrame()

    for k in k_lst:
        silh_val = []
        for run in np.arange(1, num_runs+1):

            # Trains a k-means model.
            kmeans = KMeans()\
                    .setK(k)\
                    .setSeed(int(np.random.randint(100, size=1)))
            model = kmeans.fit(df_in)

            # Make predictions
            predictions = model.transform(df_in)
            r_table['cluster_{k}_{run}'.format(k=k, run=run)]= predictions.
↪select('prediction').toPandas()

            # Evaluate clustering by computing Silhouette score
            evaluator = ClusteringEvaluator()
            silhouette = evaluator.evaluate(predictions)
            silh_val.append(silhouette)

        silh_array=np.asanyarray(silh_val)
        silh_lst.append(silh_array.mean())

    elapsed =  time.time() - start

    silhouette = pd.DataFrame(list(zip(k_lst,silh_lst)),columns = ['k',
↪'silhouette'])

    print('+-------------------------------------------------------------+')
    print("|          The finding optimal k phase took %8.0f s.       |"
↪%(elapsed))
    print('+-------------------------------------------------------------+')


    return k_lst[np.argmax(silh_lst, axis=0)], silhouette , r_table
```

```
k, silh_lst, r_table = optimal_k(scaledData,index_col,k_min, k_max,num_runs)
```

(continues on next page)

(continued from previous page)

```
+-----------------------------------------------------------+
|          The finding optimal k phase took     1783 s.       |
+-----------------------------------------------------------+
```

```
spark.createDataFrame(silh_lst).show()

+---+------------------+
|  k|        silhouette|
+---+------------------+
|  3|0.8045154385557953|
|  4|0.6993528775512052|
|  5|0.6689286654221447|
|  6|0.6356184024841809|
|  7|0.7174102265711756|
|  8|0.6720861758298997|
|  9| 0.601771359881241|
| 10|0.6292447334578428|
+---+------------------+
```

From the silhouette list, we can choose `3` as the optimal number of the clusters.

> **Warning:** `ClusteringEvaluator` in `pyspark.ml.evaluation` requires Spark 2.4 or later!!

4. K-means clustering

```
k = 3
kmeans = KMeans().setK(k).setSeed(1)
model = kmeans.fit(scaledData)
# Make predictions
predictions = model.transform(scaledData)
predictions.show(5,False)
```

```
+----------+------------------+--------------------+----------+
|CustomerID|               rfm|            features|prediction|
+----------+------------------+--------------------+----------+
|     17420|  [50.0,3.0,598.83]|[0.13404825737265...|         0|
|     16861|  [59.0,3.0,151.65]|[0.15817694369973...|         0|
|     16503|[106.0,5.0,1421.43]|[0.28418230563002...|         2|
|     15727| [16.0,7.0,5178.96]|[0.04289544235924...|         0|
|     17389|[0.0,43.0,31300.08]|[0.0,0.1700404858...|         0|
+----------+------------------+--------------------+----------+
only showing top 5 rows
```

12.3.3 Statistical summary

5. statistical summary

```
results = rfm.join(predictions.select('CustomerID','prediction'),'CustomerID',
↪how='left')
results.show(5)
```

```
+----------+-------+---------+--------+----------+
|CustomerID|Recency|Frequency|Monetary|prediction|
+----------+-------+---------+--------+----------+
|     13098|      1|       41|28658.88|         0|
|     13248|    124|        2|  465.68|         2|
|     13452|    259|        2|   590.0|         1|
|     13460|     29|        2|  183.44|         0|
|     13518|     85|        1|  659.44|         0|
+----------+-------+---------+--------+----------+
only showing top 5 rows
```

- simple summary

```
results.groupBy('prediction')\
       .agg({'Recency':'mean',
             'Frequency': 'mean',
             'Monetary': 'mean'} )\
        .sort(F.col('prediction')).show(5)
```

```
+----------+------------------+------------------+------------------+
|prediction|      avg(Recency)|     avg(Monetary)|    avg(Frequency)|
+----------+------------------+------------------+------------------+
|         0|30.966337980278816|2543.0355321319284| 6.514450867052023|
|         1|296.02403846153845|407.16831730769206|1.5592948717948718|
|         2|154.40148698884758| 702.5096406443623| 2.550185873605948|
+----------+------------------+------------------+------------------+
```

- complex summary

```
grp = 'RFMScore'
num_cols = ['Recency','Frequency','Monetary']
df_input = results

quantile_grouped = quantile_agg(df_input,grp,num_cols)
quantile_grouped.toPandas().to_csv(output_dir+'quantile_grouped.csv')

deciles_grouped = deciles_agg(df_input,grp,num_cols)
deciles_grouped.toPandas().to_csv(output_dir+'deciles_grouped.csv')
```

CHAPTER

THIRTEEN

TEXT MINING

Chinese proverb

Articles showed more than intended. – Xianglong Shen

13.1 Text Collection

13.1.1 Image to text

- My `img2txt` function

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

(continues on next page)

(continued from previous page)

```
    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.......
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------
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␣
↪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 the DST Applied␣
↪Analytics Group, Dr.
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 Computational
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
```

13.1.2 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
```

(continues on next page)

(continued from previous page)

```
    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")
```

(continues on next page)

(continued from previous page)

```
    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.

13.1.3 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
```

(continues on next page)

(continued from previous page)

```
    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␣
↪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 the DST Applied␣
↪Analytics Group, Dr.
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 Computational
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
```

13.1.4 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 in Command-line `python record.py "demo2.wav"`:

```
import sys, getopt
```

(continues on next page)

(continued from previous page)

```
import speech_recognition as sr

audio_filename = sys.argv[1]

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())
```

13.2 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
```

(continues on next page)

(continued from previous page)

```
    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'
```

(continues on next page)

(continued from previous page)

```
    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())
```

13.3 Text Classification

Theoretically speaking, you may apply any classification algorithms to do classification. I will only present Naive Bayes method is the following.

13.3.1 Introduction

13.3.2 Demo

1. create spark contexts

```
import pyspark
from pyspark.sql import SQLContext

# create spark contexts
sc = pyspark.SparkContext()
sqlContext = SQLContext(sc)
```

2. load dataset

```
# Load a text file and convert each line to a Row.
data_rdd = sc.textFile("../data/raw_data.txt")
parts_rdd = data_rdd.map(lambda l: l.split("\t"))

# Filter bad rows out
garantee_col_rdd = parts_rdd.filter(lambda l: len(l) == 3)
typed_rdd = garantee_col_rdd.map(lambda p: (p[0], p[1], float(p[2])))

#Create DataFrame
data_df = sqlContext.createDataFrame(typed_rdd, ["text", "id", "label"])

# get the raw columns
raw_cols = data_df.columns

#data_df.show()
data_df.printSchema()
```

```
root
 |-- text: string (nullable = true)
 |-- id: string (nullable = true)
 |-- label: double (nullable = true)
```

```
+--------------------+------------------+-----+
|                text|                id|label|
+--------------------+------------------+-----+
|Fresh install of ...|        1018769417|  1.0|
|Well. Now I know ...|       10284216536|  1.0|
|"Literally six we...|       10298589026|  1.0|
|Mitsubishi i MiEV...|109017669432377344|  1.0|
+--------------------+------------------+-----+
only showing top 4 rows
```

3. setup pyspark udf function

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

# Register all the functions in Preproc with Spark Context
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())
```

4. language identification

```
lang_df = data_df.withColumn("lang", check_lang_udf(data_df["text"]))
en_df = lang_df.filter(lang_df["lang"] == "en")
en_df.show(4)
```

```
+--------------------+------------------+-----+----+
|                text|                id|label|lang|
+--------------------+------------------+-----+----+
|RT @goeentertain:...|665305154954989568|  1.0|  en|
|Teforia Uses Mach...|660668007975268352|  1.0|  en|
|   Apple TV or Roku?|       25842461136|  1.0|  en|
|Finished http://t...|        9412369614|  1.0|  en|
+--------------------+------------------+-----+----+
only showing top 4 rows
```

5. remove stop words

```
rm_stops_df = en_df.select(raw_cols)\
                   .withColumn("stop_text", remove_stops_udf(en_df["text"]))
rm_stops_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+
|                text|                id|label|           stop_text|
+--------------------+------------------+-----+--------------------+
|RT @goeentertain:...|665305154954989568|  1.0|RT @goeentertain:...|
|Teforia Uses Mach...|660668007975268352|  1.0|Teforia Uses Mach...|
```

(continues on next page)

(continued from previous page)

```
|   Apple TV or Roku?|       25842461136|  1.0|      Apple TV Roku?|
|Finished http://t...|        9412369614|  1.0|Finished http://t...|
+--------------------+------------------+-----+--------------------+
only showing top 4 rows
```

6. remove irrelevant features

```
rm_features_df = rm_stops_df.select(raw_cols+["stop_text"])\
                            .withColumn("feat_text", \
                            remove_features_udf(rm_stops_df["stop_text"]))
rm_features_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+----------
↪----------+
|                text|                id|label|           stop_text|         ␣
↪ feat_text|
+--------------------+------------------+-----+--------------------+----------
↪----------+
|RT @goeentertain:...|665305154954989568|  1.0|RT @goeentertain:...|  future␣
↪blase   ...|
|Teforia Uses Mach...|660668007975268352|  1.0|Teforia Uses Mach...|teforia␣
↪uses mach...|
|   Apple TV or Roku?|       25842461136|  1.0|      Apple TV Roku?|        ␣
↪apple  roku|
|Finished http://t...|        9412369614|  1.0|Finished http://t...|         ␣
↪  finished|
+--------------------+------------------+-----+--------------------+----------
↪----------+
only showing top 4 rows
```

7. tag the words

```
tagged_df = rm_features_df.select(raw_cols+["feat_text"]) \
                          .withColumn("tagged_text", \
                           tag_and_remove_udf(rm_features_df.feat_text))

tagged_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+----------
↪----------+
|                text|                id|label|           feat_text|        ␣
↪tagged_text|
+--------------------+------------------+-----+--------------------+----------
↪----------+
|RT @goeentertain:...|665305154954989568|  1.0|  future blase   ...| future␣
↪blase vic...|
|Teforia Uses Mach...|660668007975268352|  1.0|teforia uses mach...| teforia␣
↪uses mac...|
|   Apple TV or Roku?|       25842461136|  1.0|         apple  roku|        ␣
↪apple roku |
|Finished http://t...|        9412369614|  1.0|            finished|         ␣
↪ finished |
```

(continues on next page)

(continued from previous page)

```
+--------------------+------------------+-----+--------------------+----------
↪----------+
only showing top 4 rows
```

8. lemmatization of words

```
lemm_df = tagged_df.select(raw_cols+["tagged_text"]) \
                    .withColumn("lemm_text", lemmatize_udf(tagged_df["tagged_
↪text"]))
lemm_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+----------
↪----------+
|                text|                id|label|         tagged_text|         ␣
↪ lemm_text|
+--------------------+------------------+-----+--------------------+----------
↪----------+
|RT @goeentertain:...|665305154954989568|  1.0| future blase vic...|future␣
↪blase vice...|
|Teforia Uses Mach...|660668007975268352|  1.0| teforia uses mac...|teforia␣
↪use machi...|
|   Apple TV or Roku?|       25842461136|  1.0|         apple roku |         ␣
↪apple roku|
|Finished http://t...|        9412369614|  1.0|           finished |         ␣
↪    finish|
+--------------------+------------------+-----+--------------------+----------
↪----------+
only showing top 4 rows
```

9. remove blank rows and drop duplicates

```
    check_blanks_df = lemm_df.select(raw_cols+["lemm_text"])\
                             .withColumn("is_blank", check_blanks_udf(lemm_df[
↪"lemm_text"]))
    # remove blanks
    no_blanks_df = check_blanks_df.filter(check_blanks_df["is_blank"] ==
↪"False")

# drop duplicates
dedup_df = no_blanks_df.dropDuplicates(['text', 'label'])

    dedup_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+--------+
|                text|                id|label|           lemm_text|is_blank|
+--------------------+------------------+-----+--------------------+--------+
|RT @goeentertain:...|665305154954989568|  1.0|future blase vice...|   False|
|Teforia Uses Mach...|660668007975268352|  1.0|teforia use machi...|   False|
|   Apple TV or Roku?|       25842461136|  1.0|          apple roku|   False|
|Finished http://t...|        9412369614|  1.0|              finish|   False|
+--------------------+------------------+-----+--------------------+--------+
```

(continues on next page)

(continued from previous page)

```
only showing top 4 rows
```

10. add unieuq ID

```
from pyspark.sql.functions import monotonically_increasing_id
# Create Unique ID
dedup_df = dedup_df.withColumn("uid", monotonically_increasing_id())
dedup_df.show(4)
```

```
+--------------------+------------------+-----+--------------------+--------+-
↪-----------+
|                text|                id|label|           lemm_text|is_blank|␣
↪        uid|
+--------------------+------------------+-----+--------------------+--------+-
↪-----------+
|              dragon|        1546813742|  1.0|              dragon|   False|␣
↪85899345920|
|           hurt much|        1558492525|  1.0|           hurt much|  ␣
↪False|111669149696|
|seth blog word se...|383221484023709697|  1.0|seth blog word se...|  ␣
↪False|128849018880|
|teforia use machi...|660668007975268352|  1.0|teforia use machi...|  ␣
↪False|137438953472|
+--------------------+------------------+-----+--------------------+--------+-
↪-----------+
only showing top 4 rows
```

11. create final dataset

```
data = dedup_df.select('uid','id', 'text','label')
data.show(4)
```

```
+------------+------------------+--------------------+-----+
|         uid|                id|                text|label|
+------------+------------------+--------------------+-----+
| 85899345920|        1546813742|              dragon|  1.0|
|111669149696|        1558492525|           hurt much|  1.0|
|128849018880|383221484023709697|seth blog word se...|  1.0|
|137438953472|660668007975268352|teforia use machi...|  1.0|
+------------+------------------+--------------------+-----+
only showing top 4 rows
```

12. Create taining and test sets

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

13. NaiveBayes Pipeline

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml import Pipeline
from pyspark.ml.classification import NaiveBayes, RandomForestClassifier
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 nb.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol=
↪"rawFeatures")
# vectorizer = CountVectorizer(inputCol= "words", outputCol="rawFeatures")
idf = IDF(minDocFreq=3, inputCol="rawFeatures", outputCol="features")

# Naive Bayes model
nb = NaiveBayes()

# Pipeline Architecture
pipeline = Pipeline(stages=[tokenizer, hashingTF, idf, nb])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

14. Make predictions

```
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("text", "label", "prediction").show(5,False)
```

```
+-----------------------------------------------+-----+----------+
|text                                           |label|prediction|
+-----------------------------------------------+-----+----------+
|finish                                         |1.0  |1.0       |
|meet rolo dogsofthinkgeek happy nationaldogday |1.0  |1.0       |
|pumpkin family                                 |1.0  |1.0       |
|meet jet dogsofthinkgeek happy nationaldogday  |1.0  |1.0       |
|meet vixie dogsofthinkgeek happy nationaldogday|1.0  |1.0       |
+-----------------------------------------------+-----+----------+
only showing top 5 rows
```

15. evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
evaluator.evaluate(predictions)
```

```
0.912655971479501
```

13.4 Sentiment analysis

13.4.1 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

Fig. 1: Sentiment Analysis Pipeline

13.4.2 Pipeline

13.4.3 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
```

(continues on next page)

(continued from previous page)

```
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)
```

(continues on next page)

(continued from previous page)

```
    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'
```

(continues on next page)

(continued from previous page)

```
    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|             removed|
+--------------------+----------+-------+--------------------+----------------
↪----+--------------------+
|10 Things Missing...|2602860537|  18536|10 Things Missing...|10 things␣
↪missing...|things missing in...|
|RT @_NATURALBWINN...|2602850443|  18536|RT @_NATURALBWINN...|@_
↪naturalbwinner ...|oh and do not lik...|
|RT @HBO24 yo the ...|2602761852|  18535|RT @HBO24 yo the ...|@hbo24 yo the
↪#ne...|yo the newtwitter...|
|Aaaaaaaand I have...|2602738438|  18535|Aaaaaaaand I have...|aaaaaaaand i␣
↪have...|aaaaaaaand have t...|
|can I please have...|2602684185|  18535|can I please have...|can i please␣
↪have...|can please have t...|
+--------------------+----------+-------+--------------------+----------------
↪----+--------------------+
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):
```

(continues on next page)

(continued from previous page)

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

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|
```

(continues on next page)

(continued from previous page)

```
|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
```

13.5 N-grams and Correlations

13.6 Topic Model: Latent Dirichlet Allocation

13.6.1 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.

13.6.2 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|              review|
+-----+---------------+---------+--------+------+--------+-----+-----
↪------+--------------------+
|10001|Delta Air Lines|21-Jun-14|Thailand|     7| Economy|    4|    ␣
↪   YES|Flew Mar 30 NRT t...|
|10002|Delta Air Lines|19-Jun-14|     USA|     0| Economy|    2|    ␣
↪    NO|Flight 2463 leavi...|
|10003|Delta Air Lines|18-Jun-14|     USA|     0| Economy|    1|    ␣
↪    NO|Delta Website fro...|
|10004|Delta Air Lines|17-Jun-14|     USA|     9|Business|    4|    ␣
↪   YES|"I just returned ...|
|10005|Delta Air Lines|17-Jun-14| Ecuador|     7| Economy|    3|    ␣
↪   YES|"Round-trip fligh...|
+-----+---------------+---------+--------+------+--------+-----+-----
↪------+--------------------+
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', 'review']
```

```
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
```

(continues on next page)

(continued from previous page)

```
            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)
```

(continues on next page)

(continued from previous page)

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

```
rawdata.show(5)
```

```
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+
|   id|           airline|      date|location|rating|  ␣
↪cabin|value|recommended|              review|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+
only showing top 5 rows
```

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

```
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+
|   id|           airline|      date|location|rating|  ␣
↪cabin|value|recommended|              review|            non_asci|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|Flight 3246 from ...|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|Flight from Manch...|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|I'm Executive Pla...|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|MSP-JFK-MXP and r...|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|Worst airline I h...|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+
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|              review|            non_asci|␣
↪       fixed_abbrev|
+-----+-----------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|Flight 3246 from ...|flight 3246␣
↪from ...|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|Flight from Manch...|flight from␣
↪manch...|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|I'm Executive Pla...|i'm executive␣
↪pla...|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|MSP-JFK-MXP and r...|msp-jfk-mxp␣
↪and r...|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|Worst airline I h...|worst airline␣
↪i h...|
+-----+-----------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
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|              review|        fixed_abbrev|␣
↪          stop_text|
+-----+-----------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|flight 3246 from ...|flight 3246␣
↪chica...|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|flight from manch...|flight␣
↪manchester...|
```

(continues on next page)

(continued from previous page)

```
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|i'm executive pla...|i'm executive␣
↪pla...|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|msp-jfk-mxp and r...|msp-jfk-mxp␣
↪retur...|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|worst airline i h...|worst airline␣
↪eve...|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
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|              review|           stop_text|␣
↪          feat_text|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|flight 3246 chica...|flight␣
↪chicago mi...|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|flight manchester...|flight␣
↪manchester...|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|i'm executive pla...|executive␣
↪platinu...|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|msp-jfk-mxp retur...|msp jfk mxp␣
↪retur...|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|worst airline eve...|worst airline␣
↪eve...|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
only showing top 5 rows
```

```
rawdata = rawdata.select(raw_cols+['feat_text'])\
                 .withColumn('tagged_text',tag_and_remove_
↪udf(rawdata['feat_text']))
```

(continues on next page)

(continued from previous page)

```
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
|   id|           airline|      date|location|rating|  ␣
↪cabin|value|recommended|              review|           feat_text|␣
↪        tagged_text|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|flight chicago mi...| flight␣
↪chicago m...|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|flight manchester...| flight␣
↪mancheste...|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|executive platinu...| executive␣
↪platin...|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|msp jfk mxp retur...| msp jfk mxp␣
↪retu...|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|worst airline eve...| worst␣
↪airline ua...|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
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|              review|         tagged_text|␣
↪          lemm_text|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...| flight chicago m...|flight␣
↪chicago mi...|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...| flight mancheste...|flight␣
↪manchester...|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...| executive platin...|executive␣
↪platinu...|
```

(continues on next page)

(continued from previous page)

```
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...| msp jfk mxp retu...|msp jfk mxp␣
↪retur...|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...| worst airline ua...|worst airline␣
↪ual...|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------------
↪------+
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|              review|           lemm_
↪text|is_blank|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|flight chicago mi...|   False|
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|flight manchester...|   False|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|executive platinu...|   False|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|msp jfk mxp retur...|   False|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|worst airline ual...|   False|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------+
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|              review|           lemm_
↪text|is_blank|uid|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------+---+
|10551|Southwest Airlines|2013-11-06|     USA|   1.0|Business|    2|␣
↪        NO|Flight 3246 from ...|flight chicago mi...|   False|  0|
```

(continues on next page)

(continued from previous page)

```
|10298|        US Airways|2014-03-31|      UK|   1.0|Business|    0|␣
↪        NO|Flight from Manch...|flight manchester...|   False|  1|
|10564|Southwest Airlines|2013-09-06|     USA|  10.0| Economy|    5|␣
↪       YES|I'm Executive Pla...|executive platinu...|   False|  2|
|10134|   Delta Air Lines|2013-12-10|     USA|   8.0| Economy|    4|␣
↪       YES|MSP-JFK-MXP and r...|msp jfk mxp retur...|   False|  3|
|10912|   United Airlines|2014-04-07|     USA|   3.0| Economy|    1|␣
↪        NO|Worst airline I h...|worst airline ual...|   False|  4|
+-----+------------------+----------+--------+------+--------+-----+-
↪----------+--------------------+--------------------+--------+---+
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 nb.
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...|
```

(continues on next page)

(continued from previous page)

```
|    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, lga, schedule, get, memphis, arrive]         ␣
↪      |
|1    |[363, 29, 187, 55, 48, 647, 30, 9, 204, 457] |[dublin,␣
↪class, th, sit, entertainment, express, say, delay, dl, son]     ␣
↪         |
|2    |[46, 107, 672, 274, 92, 539, 23, 27, 279, 8] |[economy,␣
↪sfo, milwaukee, decent, comfortable, iad, return, united, average,␣
↪airline]|
```

(continues on next page)

(continued from previous page)

```
|3    |[76, 43, 285, 152, 102, 34, 300, 113, 24, 31]   |[didn, pay,␣
↪lose, different, extra, bag, mile, baggage, leave, day]           ␣
↪      |
|4    |[201, 13, 372, 692, 248, 62, 211, 187, 105, 110]|[houston,␣
↪crew, heathrow, louisville, london, great, denver, th, land, jfk] ␣
↪        |
|5    |[122, 103, 181, 48, 434, 10, 121, 147, 934, 169]|[lhr, serve,␣
↪screen, entertainment, ny, delta, excellent, atl, sin, newark]    ␣
↪     |
|6    |[14, 270, 18, 74, 70, 37, 16, 450, 3, 20]       |[check,␣
↪employee, gate, line, change, wait, take, fll, time, tell]        ␣
↪          |
|7    |[111, 36, 341, 10, 320, 528, 844, 19, 195, 524] |[atlanta,␣
↪first, toilet, delta, washington, card, global, staff, route,␣
↪amsterdam]   |
|8    |[477, 266, 297, 185, 1, 33, 22, 783, 17, 908]   |[fuel, group,
↪ pas, boarding, seat, trip, minute, orleans, make, select]        ␣
↪     |
|9    |[10, 73, 46, 1, 248, 302, 213, 659, 48, 228]    |[delta, lax,␣
↪economy, seat, london, detroit, comfo, weren, entertainment, wife]␣
↪     |
|10   |[57, 29, 411, 10, 221, 121, 661, 19, 805, 733]  |[business,␣
↪class, fra, delta, lounge, excellent, syd, staff, nov, mexico]    ␣
↪       |
|11   |[293, 119, 385, 481, 503, 69, 13, 87, 176, 545] |[march, ua,␣
↪manchester, phx, envoy, drink, crew, american, aa, canada]        ␣
↪      |
|12   |[116, 218, 256, 156, 639, 20, 365, 18, 22, 136] |[san, clt,␣
↪francisco, second, text, tell, captain, gate, minute, available]  ␣
↪       |
|13   |[433, 171, 176, 339, 429, 575, 10, 26, 474, 796]|[daughter,␣
↪small, aa, ba, segment, proceed, delta, passenger, size, similar] ␣
↪       |
|14   |[74, 84, 45, 108, 342, 111, 315, 87, 52, 4]     |[line, agent,
↪ next, hotel, standby, atlanta, dallas, american, book, get]      ␣
↪     |
|15   |[669, 215, 14, 58, 561, 59, 125, 179, 93, 5]    |[fit, carry,␣
↪check, people, bathroom, ask, thing, row, don, fly]               ␣
↪     |
|16   |[198, 21, 98, 164, 57, 141, 345, 62, 121, 174]  |[ife, good,␣
↪nice, much, business, lot, dfw, great, excellent, carrier]        ␣
↪      |
|17   |[96, 29, 569, 444, 15, 568, 21, 103, 657, 505]  |[phl, class,␣
↪diego, lady, food, wheelchair, good, serve, miami, mia]           ␣
↪     |
|18   |[18, 60, 140, 64, 47, 40, 31, 35, 2, 123]       |[gate, pm,␣
↪phoenix, connection, cancel, connect, day, airpo, hour, charlotte]␣
↪       |
|19   |[33, 178, 95, 2, 9, 284, 42, 4, 89, 31]         |[trip,␣
↪counter, philadelphia, hour, delay, stay, way, get, southwest,␣
↪day]           |
+-----+-------------------------------------------------+-------------
↪-------------------------------------------------------
↪-----+
```

(continues on next page)

(continued from previous page)

- LDA results

```
+-----+------------------+----------+-----------+------+-------------
↪-------+--------------------+--------------------+
|   id|           airline|      date|      cabin|rating|             ␣
↪  words|            features|   topicDistribution|
+-----+------------------+----------+-----------+------+-------------
↪-------+--------------------+--------------------+
|10551|Southwest Airlines|2013-11-06|   Business|   1.0|[flight,␣
↪chicago,...|(4695,[0,2,3,6,11...|[0.03640342580508...|
|10298|        US Airways|2014-03-31|   Business|   1.0|[flight,␣
↪manchest...|(4695,[0,1,2,6,7,...|[0.01381306271470...|
|10564|Southwest Airlines|2013-09-06|    Economy|  10.0|[executive,␣
↪plati...|(4695,[0,1,6,7,11...|[0.05063554352934...|
|10134|   Delta Air Lines|2013-12-10|    Economy|   8.0|[msp, jfk,␣
↪mxp, r...|(4695,[0,1,3,10,1...|[0.01494708959842...|
|10912|   United Airlines|2014-04-07|    Economy|   3.0|[worst,␣
↪airline, ...|(4695,[0,1,7,8,13...|[0.04421751181232...|
|10089|   Delta Air Lines|2014-02-18|    Economy|   2.0|[dl, mia,␣
↪lax, im...|(4695,[2,4,5,7,8,...|[0.02158861273876...|
|10385|        US Airways|2013-10-21|    Economy|  10.0|[flew, gla,␣
↪phl, ...|(4695,[0,1,3,5,14...|[0.03343845991816...|
|10249|        US Airways|2014-06-17|    Economy|   1.0|[friend,␣
↪book, fl...|(4695,[0,2,3,4,5,...|[0.02362432562165...|
|10289|        US Airways|2014-04-12|    Economy|  10.0|[flew, air,␣
↪rome,...|(4695,[0,1,5,8,13...|[0.01664012816210...|
|10654|Southwest Airlines|2012-07-10|    Economy|   8.0|[lhr, jfk,␣
↪think,...|(4695,[0,4,5,6,8,...|[0.01526072330297...|
|10754| American Airlines|2014-05-04|    Economy|  10.0|[san, diego,␣
↪moli...|(4695,[0,2,8,15,2...|[0.03571177612496...|
|10646|Southwest Airlines|2012-08-17|    Economy|   7.0|[toledo, co,␣
↪stop...|(4695,[0,2,3,4,7,...|[0.02394775146271...|
|10097|   Delta Air Lines|2014-02-03|First Class|  10.0|[honolulu,␣
↪la, fi...|(4695,[0,4,6,7,13...|[0.02008375619661...|
|10132|   Delta Air Lines|2013-12-16|    Economy|   7.0|[manchester,␣
↪uk, ...|(4695,[0,1,2,3,5,...|[0.01463126146601...|
|10560|Southwest Airlines|2013-09-20|    Economy|   9.0|[first, time,
↪ sou...|(4695,[0,3,7,8,9,...|[0.04934836409896...|
|10579|Southwest Airlines|2013-07-25|    Economy|   0.0|[plane, land,
↪ pm,...|(4695,[2,3,4,5,7,...|[0.06106959241722...|
|10425|        US Airways|2013-08-06|    Economy|   3.0|[airway, bad,
↪ pro...|(4695,[2,3,4,7,8,...|[0.01770471771322...|
|10650|Southwest Airlines|2012-07-27|    Economy|   9.0|[flew, jfk,␣
↪lhr, ...|(4695,[0,1,6,13,1...|[0.02676226245086...|
|10260|        US Airways|2014-06-03|    Economy|   1.0|[february,␣
↪air, u...|(4695,[0,2,4,17,2...|[0.02887390875079...|
|10202|   Delta Air Lines|2013-09-14|    Economy|  10.0|[aug, lhr,␣
↪jfk, b...|(4695,[1,2,4,7,10...|[0.02377704988307...|
+-----+------------------+----------+-----------+------+-------------
↪-------+--------------------+--------------------+
```

(continues on next page)

(continued from previous page)

```
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 anc 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 staf 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 (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

FOURTEEN

SOCIAL NETWORK ANALYSIS

Chinese proverb

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

14.1 Introduction

14.2 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.

14.2.1 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'emma']
```

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.]])
```

14.2.2 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:

  Output:

Fig. 1: Word frequency

```
+----------+------+-------+
|    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:

Then you will get Figure *Co-occurrence network*

14.3 Appendix: matrix multiplication in PySpark

1. load test matrix

```
df = spark.read.csv("matrix1.txt",sep=",",inferSchema=True)
df.show()
```

Fig. 2: Co-occurrence network

```
+---+---+---+---+
|_c0|_c1|_c2|_c3|
+---+---+---+---+
|1.2|3.4|2.3|1.1|
|2.3|1.1|1.5|2.2|
|3.3|1.8|4.5|3.3|
|5.3|2.2|4.5|4.4|
|9.3|8.1|0.3|5.5|
|4.5|4.3|2.1|6.6|
+---+---+---+---+
```

2. main function for matrix multiplication in PySpark

```
from pyspark.sql import functions as F
from functools import reduce
# reference: https://stackoverflow.com/questions/44348527/matrix-
↪multiplication-at-a-in-pyspark
# do the sum of the multiplication that we want, and get
# one data frame for each column
colDFs = []
for c2 in df.columns:
    colDFs.append( df.select( [ F.sum(df[c1]*df[c2]).alias("op_{0}".
↪format(i)) for i,c1 in enumerate(df.columns) ] ) )
# now union those separate data frames to build the "matrix"
mtxDF = reduce(lambda a,b: a.select(a.columns).union(b.select(a.columns)),␣
↪colDFs )
mtxDF.show()
```

```
+------------------+------------------+------------------+------------------+
|              op_0|              op_1|              op_2|              op_3|
+------------------+------------------+------------------+------------------+
|            152.45|118.88999999999999|             57.15|121.44000000000001|
|118.88999999999999|104.94999999999999|             38.93|             94.71|
|             57.15|             38.93|52.540000000000006|             55.99|
|121.44000000000001|             94.71|             55.99|110.10999999999999|
+------------------+------------------+------------------+------------------+
```

3. Validation with python version

```
import numpy as np
a = np.genfromtxt("matrix1.txt",delimiter=",")
np.dot(a.T, a)
```

```
array([[152.45, 118.89,  57.15, 121.44],
       [118.89, 104.95,  38.93,  94.71],
       [ 57.15,  38.93,  52.54,  55.99],
       [121.44,  94.71,  55.99, 110.11]])
```

14.4 Correlation Network

TODO ..

CHAPTER

FIFTEEN

ALS: STOCK PORTFOLIO RECOMMENDATIONS

Chinese proverb

Don't put all your eggs in one basket.

Code for the above figure:

```
import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(figsize=(10, 8), subplot_kw=dict(aspect="equal"))

recipe = ["375 k U.S. Large Cap Blend",
          "300 k U.S. Large Cap Value",
          "75 k U.S. Short-Term Bonds",
          "50 k U.S. Small Cap Blend",
          "55 k U.S. Small Cap Value",
          "95 k U.S. Real Estate",
          "250 k Intermediate-Term Bonds"]

data = [float(x.split()[0]) for x in recipe]
ingredients = [' '.join(x.split()[2:]) for x in recipe]

print(data)
print(ingredients)
def func(pct, allvals):
    absolute = int(pct/100.*np.sum(allvals))
    return "{:.1f}%\n({:d} k)".format(pct, absolute)

explode = np.empty(len(data))#(0.1, 0.1, 0.1,  0.1, 0.1, 0.1)  # explode 1st␣
↪slice
explode.fill(0.1)

wedges, texts, autotexts = ax.pie(data, explode=explode, autopct=lambda pct:␣
↪func(pct, data),
                                  textprops=dict(color="w"))
ax.legend(wedges, ingredients,
          #title="Stock portfolio",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))

plt.setp(autotexts, size=8, weight="bold")

#ax.set_title("Stock portfolio")

plt.show()
```

15.1 Recommender systems

Recommender systems or recommendation systems (sometimes replacing "system" with a synonym such as platform or engine) are a subclass of information filtering system that seek to predict the "rating" or "preference" that a user would give to an item."

The main idea is to build a matrix users `R` items rating values and try to factorize it, to recommend main products rated by other users. A popular approach for this is matrix factorization is Alternating Least Squares (ALS)

15.2 Alternating Least Squares

Apache Spark ML implements ALS for collaborative filtering, a very popular algorithm for making recommendations.

ALS recommender is a matrix factorization algorithm that uses Alternating Least Squares with Weighted-Lamda-Regularization (ALS-WR). It factors the user to item matrix `A` into the user-to-feature matrix `U` and the item-to-feature matrix `M`: It runs the ALS algorithm in a parallel fashion. The ALS algorithm should uncover the latent factors that explain the observed user to item ratings and tries to find optimal factor weights to minimize the least squares between predicted and actual ratings.

https://www.elenacuoco.com/2016/12/22/alternating-least-squares-als-spark-ml/

15.3 Demo

- The Jupyter notebook can be download from ALS Recommender systems.
- The data can be downloaf from German Credit.

15.3.1 Load and clean data

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

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

2. Load dataset

```
df_raw = spark.read.format('com.databricks.spark.csv').\
                        options(header='true', \
                        inferschema='true').\
            load("Online Retail.csv",header=True);
```

check the data set

```
df_raw.show(5)
df_raw.printSchema()
```

Then you will get

```
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+
|InvoiceNo|StockCode|          Description|Quantity|␣
↪InvoiceDate|UnitPrice|CustomerID|        Country|
```

(continues on next page)

(continued from previous page)

```
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/10 8:26|     2.55|  ␣
↪ 17850|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/10 8:26|     3.39|  ␣
↪ 17850|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/10 8:26|     2.75|  ␣
↪ 17850|United Kingdom|
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/10 8:26|     3.39|  ␣
↪ 17850|United Kingdom|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/10 8:26|     3.39|  ␣
↪ 17850|United Kingdom|
+---------+---------+--------------------+--------+------------+---------+----
↪------+--------------+
only showing top 5 rows

root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: integer (nullable = true)
 |-- Country: string (nullable = true)
```

3. Data clean and data manipulation

- check and remove the `null` values

```
from pyspark.sql.functions import count

def my_count(df_in):
    df_in.agg( *[ count(c).alias(c) for c in df_in.columns ] ).show()
```

```
import pyspark.sql.functions as F
from pyspark.sql.functions import round
df_raw = df_raw.withColumn('Asset',round( F.col('Quantity') * F.col('UnitPrice
↪'), 2 ))
df = df_raw.withColumnRenamed('StockCode', 'Cusip')\
           .select('CustomerID','Cusip','Quantity','UnitPrice','Asset')
```

```
my_count(df)
```

```
+----------+------+--------+---------+------+
|CustomerID| Cusip|Quantity|UnitPrice| Asset|
+----------+------+--------+---------+------+
|    406829|541909|  541909|   541909|541909|
+----------+------+--------+---------+------+
```

Since the count results are not the same, we have some null value in the `CustomerID` column. We can

drop these records from the dataset.

```
df =  df.filter(F.col('Asset')>=0)
df = df.dropna(how='any')
my_count(df)
```

```
+----------+------+--------+---------+------+
|CustomerID| Cusip|Quantity|UnitPrice| Asset|
+----------+------+--------+---------+------+
|    397924|397924|  397924|   397924|397924|
+----------+------+--------+---------+------+
```

```
df.show(3)

+----------+------+--------+---------+-----+
|CustomerID| Cusip|Quantity|UnitPrice|Asset|
+----------+------+--------+---------+-----+
|     17850|85123A|       6|     2.55| 15.3|
|     17850| 71053|       6|     3.39|20.34|
|     17850|84406B|       8|     2.75| 22.0|
+----------+------+--------+---------+-----+
only showing top 3 rows
```

- Convert the `Cusip` to consistent format

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

def toUpper(s):
    return s.upper()

upper_udf = udf(lambda x: toUpper(x), StringType())
```

- Find the most top `n` stockes

```
pop = df.groupBy('Cusip')\
  .agg(F.count('CustomerID').alias('Customers'),F.round(F.sum('Asset'),2).
↪alias('TotalAsset'))\
  .sort([F.col('Customers'),F.col('TotalAsset')],ascending=[0,0])

pop.show(5)
```

```
+------+---------+----------+
| Cusip|Customers|TotalAsset|
+------+---------+----------+
|85123A|     2035|  100603.5|
| 22423|     1724| 142592.95|
|85099B|     1618|  85220.78|
| 84879|     1408|  56580.34|
| 47566|     1397|  68844.33|
+------+---------+----------+
only showing top 5 rows
```

15.3.2 Build feature matrix

- Fetch the top n cusip list

```
top = 10
cusip_lst = pd.DataFrame(pop.select('Cusip').head(top)).astype('str').iloc[:,␣
↪0].tolist()
cusip_lst.insert(0,'CustomerID')
```

```

```

- Create the portfolio table for each customer

```
pivot_tab = df.groupBy('CustomerID').pivot('Cusip').sum('Asset')
pivot_tab = pivot_tab.fillna(0)
```

- Fetch the most n stock's portfolio table for each customer

```
selected_tab  = pivot_tab.select(cusip_lst)
selected_tab.show(4)
```

```
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|CustomerID|85123A|22423|85099B|84879|47566|20725|22720|20727|POST|23203|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|     16503|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0| 33.0| 0.0|  0.0|
|     15727| 123.9| 25.5|   0.0|  0.0|  0.0| 33.0| 99.0|  0.0| 0.0|  0.0|
|     14570|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0|  0.0| 0.0|  0.0|
|     14450|   0.0|  0.0|  8.32|  0.0|  0.0|  0.0| 49.5|  0.0| 0.0|  0.0|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
only showing top 4 rows
```

- Build the `rating` matrix

```
def elemwiseDiv(df_in):
    num = len(df_in.columns)
    temp = df_in.rdd.map(lambda x: list(flatten([x[0],[x[i]/float(sum(x[1:]))
                                                        if sum(x[1:])>0 else␣
↪x[i]
                                                        for i in range(1,
↪num)]])))
    return spark.createDataFrame(temp,df_in.columns)

ratings = elemwiseDiv(selected_tab)
```

```
ratings.show(4)

+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|CustomerID|85123A|22423|85099B|84879|47566|20725|22720|20727|POST|23203|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
|     16503|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0|  1.0| 0.0|  0.0|
|     15727|  0.44| 0.09|   0.0|  0.0|  0.0| 0.12| 0.35|  0.0| 0.0|  0.0|
```

(continues on next page)

(continued from previous page)

```
|     14570|   0.0|  0.0|   0.0|  0.0|  0.0|  0.0|  0.0|  0.0| 0.0|  0.0|
|     14450|   0.0|  0.0|  0.14|  0.0|  0.0|  0.0| 0.86|  0.0| 0.0|  0.0|
+----------+------+-----+------+-----+-----+-----+-----+-----+----+-----+
```

- Convert `rating` matrix to long table

```
from pyspark.sql.functions import array, col, explode, struct, lit

def to_long(df, by):
        """
        reference: https://stackoverflow.com/questions/37864222/transpose-
↪column-to-row-with-spark
        """

    # Filter dtypes and split into column names and type description
    cols, dtypes = zip(*((c, t) for (c, t) in df.dtypes if c not in by))
    # Spark SQL supports only homogeneous columns
    assert len(set(dtypes)) == 1, "All columns have to be of the same type"

    # Create and explode an array of (column_name, column_value) structs
    kvs = explode(array([
      struct(lit(c).alias("Cusip"), col(c).alias("rating")) for c in cols
    ])).alias("kvs")
```

```
df_all = to_long(ratings,['CustomerID'])
df_all.show(5)
```

```
+----------+------+------+
|CustomerID| Cusip|rating|
+----------+------+------+
|     16503|85123A|   0.0|
|     16503| 22423|   0.0|
|     16503|85099B|   0.0|
|     16503| 84879|   0.0|
|     16503| 47566|   0.0|
+----------+------+------+
only showing top 5 rows
```

- Convert the string `Cusip` to numerical index

```
from pyspark.ml.feature import StringIndexer
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='Cusip',
                             outputCol='indexedCusip').fit(df_all)
df_all = labelIndexer.transform(df_all)

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

```
+----------+------+------+------------+
```

(continues on next page)

(continued from previous page)

```
|CustomerID| Cusip|rating|indexedCusip|
+----------+------+------+------------+
|     16503|85123A|   0.0|         6.0|
|     16503| 22423|   0.0|         9.0|
|     16503|85099B|   0.0|         5.0|
|     16503| 84879|   0.0|         1.0|
|     16503| 47566|   0.0|         0.0|
+----------+------+------+------------+
only showing top 5 rows

root
 |-- CustomerID: long (nullable = true)
 |-- Cusip: string (nullable = false)
 |-- rating: double (nullable = true)
 |-- indexedCusip: double (nullable = true)
```

15.3.3 Train model

- build `train` and `test` dataset

```
train, test = df_all.randomSplit([0.8,0.2])

train.show(5)
test.show(5)
```

```
+----------+-----+------------+-------------------+
|CustomerID|Cusip|indexedCusip|             rating|
+----------+-----+------------+-------------------+
|     12940|20725|         2.0|                0.0|
|     12940|20727|         4.0|                0.0|
|     12940|22423|         9.0|0.49990198000392083|
|     12940|22720|         3.0|                0.0|
|     12940|23203|         7.0|                0.0|
+----------+-----+------------+-------------------+
only showing top 5 rows

+----------+-----+------------+------------------+
|CustomerID|Cusip|indexedCusip|            rating|
+----------+-----+------------+------------------+
|     12940|84879|         1.0|0.1325230346990786|
|     13285|20725|         2.0|0.2054154995331466|
|     13285|20727|         4.0|0.2054154995331466|
|     13285|47566|         0.0|               0.0|
|     13623|23203|         7.0|               0.0|
+----------+-----+------------+------------------+
only showing top 5 rows
```

- train model

```
import itertools
from math import sqrt
from operator import add
import sys
from pyspark.ml.recommendation import ALS

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
def computeRmse(model, data):
    """
    Compute RMSE (Root mean Squared Error).
    """
    predictions = model.transform(data)
    rmse = evaluator.evaluate(predictions)
    print("Root-mean-square error = " + str(rmse))
    return rmse

#train models and evaluate them on the validation set

ranks = [4,5]
lambdas = [0.05]
numIters = [30]
bestModel = None
bestValidationRmse = float("inf")
bestRank = 0
bestLambda = -1.0
bestNumIter = -1

val = test.na.drop()
for rank, lmbda, numIter in itertools.product(ranks, lambdas, numIters):
    als = ALS(rank=rank, maxIter=numIter, regParam=lmbda, numUserBlocks=10,␣
↪numItemBlocks=10, implicitPrefs=False,
              alpha=1.0,
              userCol="CustomerID", itemCol="indexedCusip", seed=1, ratingCol=
↪"rating", nonnegative=True)
    model=als.fit(train)

    validationRmse = computeRmse(model, val)
    print("RMSE (validation) = %f for the model trained with " %␣
↪validationRmse + \
            "rank = %d, lambda = %.1f, and numIter = %d." % (rank, lmbda,␣
↪numIter))
    if (validationRmse, bestValidationRmse):
        bestModel = model
        bestValidationRmse = validationRmse
        bestRank = rank
        bestLambda = lmbda
        bestNumIter = numIter

model = bestModel
```

15.3.4 Make prediction

- make prediction

```
topredict=test[test['rating']==0]

predictions=model.transform(topredict)
predictions.filter(predictions.prediction>0)\
           .sort([F.col('CustomerID'),F.col('Cusip')],ascending=[0,0]).show(5)
```

```
+----------+------+------------+------+------------+
|CustomerID| Cusip|indexedCusip|rating|  prediction|
+----------+------+------------+------+------------+
|     18283| 47566|         0.0|   0.0|  0.01625076|
|     18282|85123A|         6.0|   0.0| 0.057172246|
|     18282| 84879|         1.0|   0.0| 0.059531752|
|     18282| 23203|         7.0|   0.0| 0.010502596|
|     18282| 22720|         3.0|   0.0| 0.053893942|
+----------+------+------------+------+------------+
only showing top 5 rows
```

CHAPTER

SIXTEEN

MONTE CARLO SIMULATION

Monte Carlo simulations are just a way of estimating a fixed parameter by repeatedly generating random numbers. More details can be found at A Zero Math Introduction to Markov Chain Monte Carlo Methods.

Monte Carlo simulation is a technique used to understand the impact of risk and uncertainty in financial, project management, cost, and other forecasting models. A Monte Carlo simulator helps one visualize most or all of the potential outcomes to have a better idea regarding the risk of a decision. More details can be found at The house always wins.

16.1 Simulating Casino Win

We assume that the player John has the 49% chance to win the game and the wager will be $5 per game.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

start_m =100
wager = 5
bets = 100
trials = 1000

trans = np.vectorize(lambda t: -wager if t <=0.51 else wager)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1,1,1)

end_m = []

for i in range(trials):
    money = reduce(lambda c, x: c + [c[-1] + x], trans(np.random.
↪random(bets)), [start_m])
    end_m.append(money[-1])
    plt.plot(money)

plt.ylabel('Player Money in $')
plt.xlabel('Number of bets')
plt.title(("John starts the game with $ %.2f and ends with $ %.2f")%(start_m,
↪sum(end_m)/len(end_m)))
plt.show()
```

16.2 Simulating a Random Walk

16.2.1 Fetch the histrical stock price

1. Fecth the data. If you need the code for this piece, you can contact with me.

```
stock.tail(4)

+----------+----------+----------+----------+----------+----------+--------+
|      Date|      Open|      High|       Low|     Close| Adj Close|  Volume|
+----------+----------+----------+----------+----------+----------+--------+
|2018-12-07|155.399994|158.050003|151.729996|153.059998|153.059998|17447900|
|2018-12-10|150.389999|152.809998|147.479996|151.429993|151.429993|15525500|
|2018-12-11|155.259995|156.240005|150.899994|151.830002|151.830002|13651900|
|2018-12-12|155.240005|156.169998|151.429993|     151.5|     151.5|16597900|
+----------+----------+----------+----------+----------+----------+--------+
```

2. Convert the `str` type date to date type

```
stock['Date'] = pd.to_datetime(stock['Date'])
```

3. Data visualization

```
# Plot everything by leveraging the very powerful matplotlib package
width = 10
```

(continues on next page)

(continued from previous page)

```
height = 6
data = stock
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)
ax.plot(data.Date, data.Close, label='Close')
ax.plot(data.Date, data.High, label='High')
# ax.plot(data.Date, data.Low, label='Low')
ax.set_xlabel('Date')
ax.set_ylabel('price ($)')
ax.legend()
ax.set_title('Stock price: ' + ticker, y=1.01)
#plt.xticks(rotation=70)
plt.show()
# Plot everything by leveraging the very powerful matplotlib package
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)
ax.plot(data.Date, data.Volume, label='Volume')
#ax.plot(data.Date, data.High, label='High')
# ax.plot(data.Date, data.Low, label='Low')
ax.set_xlabel('Date')
ax.set_ylabel('Volume')
ax.legend()
ax.set_title('Stock volume: ' + ticker, y=1.01)
#plt.xticks(rotation=70)
plt.show()
```

16.2.2 Calulate the Compound Annual Growth Rate

The formula for Compound Annual Growth Rate (CAGR) is very useful for investment analysis. It may also be referred to as the annualized rate of return or annual percent yield or effective annual rate, depending on the algebraic form of the equation. Many investments such as stocks have returns that can vary wildly. The CAGR formula allows you to calculate a "smoothed" rate of return that you can use to compare to other investments. The formula is defined as (more details can be found at CAGR Calculator and Formula)

$$\text{CAGR} = \left(\frac{\text{End Value}}{\text{Start Value}}\right)^{\frac{365}{\text{Days}}} - 1$$

```
days =  (stock.Date.iloc[-1] - stock.Date.iloc[0]).days
cagr = ((((stock['Adj Close'].iloc[-1]) / stock['Adj Close'].iloc[0])) **␣
↪(365.0/days)) - 1
print ('CAGR =',str(round(cagr,4)*100)+"%")
mu = cagr
```

16.2.3 Calulate the annual volatility

A stock's volatility is the variation in its price over a period of time. For example, one stock may have a tendency to swing wildly higher and lower, while another stock may move in much steadier, less turbulent

Fig. 1: Historical Stock Price

way. Both stocks may end up at the same price at the end of day, but their path to that point can vary wildly. First, we create a series of percentage returns and calculate the annual volatility of returns Annualizing volatility. To present this volatility in annualized terms, we simply need to multiply our daily standard deviation by the square root of 252. This assumes there are 252 trading days in a given year. More details can be found at How to Calculate Annualized Volatility.

```
stock['Returns'] = stock['Adj Close'].pct_change()
vol = stock['Returns'].std()*np.sqrt(252)
```

16.2.4 Create matrix of daily returns

1. Create matrix of daily returns using random normal distribution Generates an RDD matrix comprised of i.i.d. samples from the uniform distribution U(0.0, 1.0).

```
S = stock['Adj Close'].iloc[-1] #starting stock price (i.e. last available␣
↪real stock price)
T = 5 #Number of trading days
mu = cagr #Return
vol = vol #Volatility
trials = 10000

mat = RandomRDDs.normalVectorRDD(sc, trials, T, seed=1)
```

Fig. 2: Historical Stock Volume

2. Transform the distribution in the generated RDD from U(0.0, 1.0) to U(a, b), use RandomRDDs.uniformRDD(sc, n, p, seed) .map(lambda v: a + (b - a) * v)

```
a = mu/T
b = vol/math.sqrt(T)
v = mat.map(lambda x: a +  (b - a)* x)
```

3. Convert Rdd mstrix to dataframe

```
df = v.map(lambda x: [round(i,6)+1 for i in x]).toDF()
df.show(5)
```

```
+--------+--------+--------+--------+--------+
|      _1|      _2|      _3|      _4|      _5|
+--------+--------+--------+--------+--------+
|0.935234|1.162894| 1.07972|1.238257|1.066136|
|0.878456|1.045922|0.990071|1.045552|0.854516|
|1.186472|0.944777|0.742247|0.940023|1.220934|
|0.872928|1.030882|1.248644|1.114262|1.063762|
| 1.09742|1.188537|1.137283|1.162548|1.024612|
+--------+--------+--------+--------+--------+
only showing top 5 rows
```

```
from pyspark.sql.functions import lit
S = stock['Adj Close'].iloc[-1]
price = df.withColumn('init_price' ,lit(S))
```

```
price.show(5)

+--------+--------+--------+--------+--------+----------+
|      _1|      _2|      _3|      _4|      _5|init_price|
+--------+--------+--------+--------+--------+----------+
|0.935234|1.162894| 1.07972|1.238257|1.066136|     151.5|
|0.878456|1.045922|0.990071|1.045552|0.854516|     151.5|
|1.186472|0.944777|0.742247|0.940023|1.220934|     151.5|
|0.872928|1.030882|1.248644|1.114262|1.063762|     151.5|
| 1.09742|1.188537|1.137283|1.162548|1.024612|     151.5|
+--------+--------+--------+--------+--------+----------+
only showing top 5 rows
```

```
price = price.withColumn('day_0', col('init_price'))
price.show(5)
```

```
+--------+--------+--------+--------+--------+----------+-----+
|      _1|      _2|      _3|      _4|      _5|init_price|day_0|
+--------+--------+--------+--------+--------+----------+-----+
|0.935234|1.162894| 1.07972|1.238257|1.066136|     151.5|151.5|
|0.878456|1.045922|0.990071|1.045552|0.854516|     151.5|151.5|
|1.186472|0.944777|0.742247|0.940023|1.220934|     151.5|151.5|
|0.872928|1.030882|1.248644|1.114262|1.063762|     151.5|151.5|
| 1.09742|1.188537|1.137283|1.162548|1.024612|     151.5|151.5|
```

(continues on next page)

(continued from previous page)

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

16.2.5 Monte Carlo Simulation

```
from pyspark.sql.functions import round
for name in price.columns[:-2]:
    price = price.withColumn('day'+name, round(col(name)*col('init_price'),2))
    price = price.withColumn('init_price',col('day'+name))
```

```
price.show(5)

+--------+--------+--------+--------+--------+----------+-----+------+------+-
↪-----+------+------+
|      _1|      _2|      _3|      _4|      _5|init_price|day_0| day_1| day_2|␣
↪day_3| day_4| day_5|
+--------+--------+--------+--------+--------+----------+-----+------+------+-
↪-----+------+------+
|0.935234|1.162894| 1.07972|1.238257|1.066136|    234.87|151.5|141.69|164.
↪77|177.91| 220.3|234.87|
|0.878456|1.045922|0.990071|1.045552|0.854516|    123.14|151.5|133.09| 139.
↪2|137.82| 144.1|123.14|
|1.186472|0.944777|0.742247|0.940023|1.220934|    144.67|151.5|179.75|169.
↪82|126.05|118.49|144.67|
|0.872928|1.030882|1.248644|1.114262|1.063762|    201.77|151.5|132.25|136.
↪33|170.23|189.68|201.77|
| 1.09742|1.188537|1.137283|1.162548|1.024612|     267.7|151.5|166.26|197.
↪61|224.74|261.27| 267.7|
+--------+--------+--------+--------+--------+----------+-----+------+------+-
↪-----+------+------+
only showing top 5 rows
```

16.2.6 Summary

```
selected_col = [name for name in price.columns if 'day_' in name]

simulated = price.select(selected_col)
simulated.describe().show()
```

```
+-------+----------+------------------+------------------+------------------+-
↪-----------------+------------------+
|summary|2018-12-12|        2018-12-13|        2018-12-14|        2018-12-17|␣
↪       2018-12-18|        2018-12-19|
+-------+----------+------------------+------------------+------------------+-
↪-----------------+------------------+
|  count|   10000.0|           10000.0|           10000.0|           10000.0|␣
↪          10000.0|           10000.0|
```

(continues on next page)

(continued from previous page)

```
|   mean|     151.5|155.11643700000002|        158.489058|162.23713200000003|␣
↪       166.049375|        170.006525|
|    std|       0.0|18.313783237787845|26.460919262517276| 33.
↪37780495150803|39.369101074463416|45.148120695490846|
|    min|     151.5|              88.2|             74.54|             65.87|␣
↪            68.21|             58.25|
|    25%|     151.5|           142.485|            140.15|            138.72|␣
↪           138.365|            137.33|
|    50%|     151.5|            154.97|           157.175|            159.82|␣
↪            162.59|165.04500000000002|
|    75%|     151.5|           167.445|175.48499999999999|          182.8625|␣
↪           189.725|           196.975|
|    max|     151.5|            227.48|            275.94|            319.17|␣
↪            353.59|            403.68|
+-------+----------+------------------+------------------+------------------+-
↪-----------------+------------------+
```

```
data_plt = simulated.toPandas()
days = pd.date_range(stock['Date'].iloc[-1], periods= T+1,freq='B').date

width = 10
height = 6
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)

days = pd.date_range(stock['Date'].iloc[-1], periods= T+1,freq='B').date

for i in range(trials):
    plt.plot(days, data_plt.iloc[i])
ax.set_xlabel('Date')
ax.set_ylabel('price ($)')
ax.set_title('Simulated Stock price: ' + ticker, y=1.01)
plt.show()
```

16.2.7 One-year Stock price simulation

Fig. 3: Simulated Stock Price

Fig. 4: Simulated Stock Price distribution

CHAPTER

SEVENTEEN

MARKOV CHAIN MONTE CARLO

Chinese proverb

A book is known in time of need.

Monte Carlo simulations are just a way of estimating a fixed parameter by repeatedly generating random numbers. More details can be found at A Zero Math Introduction to Markov Chain Monte Carlo Methods.

Markov Chain Monte Carlo (MCMC) methods are used to approximate the posterior distribution of a parameter of interest by random sampling in a probabilistic space. More details can be found at A Zero Math Introduction to Markov Chain Monte Carlo Methods.

The following theory and demo are from Dr. Rebecca C. Steorts's Intro to Markov Chain Monte Carlo. More details can be found at Dr. Rebecca C. Steorts's STA 360/601: Bayesian Methods and Modern Statistics class at Duke.

17.1 Metropolis algorithm

The Metropolis algorithm takes three main steps:

1. Sample $\theta^* \sim J(\theta|\theta^{(s)})$

2. Compute the acceptance ratio (r)

$$r = \frac{p(\theta^*|y)}{p(\theta^{(s)}|y)} = \frac{p(y|\theta^*)p(\theta^*)}{p(y|\theta^{(s)})p(\theta^{(s)})}$$

3. Let

$$\theta^{(s+1)} = \begin{cases} \theta^* & \text{with prob } \min(r, 1) \\ \theta^{(s)} & \text{otherwise} \end{cases} \tag{17.1}$$

Note: Actually, the (17.1) in Step 3 can be replaced by sampling $u \sim \text{Uniform}(0, 1)$ and setting $\theta^{(s+1)} = \theta^*$ if $u < r$ and setting $\theta^{(s+1)} = \theta^{(s)}$ otherwise.

17.2 A Toy Example of Metropolis

The following example is going to test out the Metropolis algorithm for the conjugate Normal-Normal model with a known variance situation.

17.2.1 Conjugate Normal-Normal model

$$\begin{aligned} X_1, \cdots, X_n \quad & \theta \overset{iid}{\sim} \text{Normal}(\theta, \sigma^2) \\ & \theta \sim \text{Normal}(\mu, \tau^2) \end{aligned}$$

Recall that the posterior of θ is Normal(μ_n, τ_n^2), where

$$\mu_n = \bar{x}\frac{n/\sigma^2}{n/\sigma^2 + 1/\tau^2} + \mu\frac{1/\tau^2}{n/\sigma^2 + 1/\tau^2}$$

and

$$\tau_n^2 = \frac{1}{n/\sigma^2 + 1/\tau^2}$$

17.2.2 Example setup

The rest of the parameters are $\sigma^2 = 1$, $\tau^2 = 10$, $\mu = 5$, $n = 5$ and

$$y = [9.37, 10.18, 9.16, 11.60, 10.33]$$

For this setup, we get that $\mu_n = 10.02745$ and $\tau_n^2 = 0.1960784$.

17.2.3 Essential mathematical derivation

In the *Metropolis algorithm*, we need to compute the acceptance ratio r, i.e.

$$\begin{aligned} r &= \frac{p(\theta^*|x)}{p(\theta^{(s)}|x)} \\ &= \frac{p(x|\theta^*)p(\theta^*)}{p(x|\theta^{(s)})p(\theta^{(s)})} \\ &= \left(\frac{\prod_i \text{dnorm}(x_i, \theta^*, \sigma)}{\prod_i \text{dnorm}(x_i, \theta^{(s)}, \sigma)}\right)\left(\frac{\text{dnorm}(\theta^*, \mu, \tau)}{\text{dnorm}(\theta^{(s)}, \mu, \tau)}\right) \end{aligned}$$

In many cases, computing the ratio r directly can be numerically unstable, however, this can be modified by taking $logr$. i.e.

$$\begin{aligned} logr = &\sum_i \Big(log[\text{dnorm}(x_i, \theta^*, \sigma)] - log[\text{dnorm}(x_i, \theta^{(s)}, \sigma)]\Big) \\ &+ \sum_i \Big(log[\text{dnorm}(\theta^*, \mu, \tau)] - log[\text{dnorm}(\theta^{(s)}, \mu, \tau)]\Big) \end{aligned}$$

Then the criteria of the acceptance becomes: if $logu < logr$, where u is sample form the Uniform$(0, 1)$.

17.3 Demos

Now, We generate S iterations of the Metropolis algorithm starting at $\theta^{(0)} = 0$ and using a normal proposal distribution, where

$$\theta^{(s+1)} \sim \text{Normal}(\theta^{(s)}, 2).$$

17.3.1 R results

```
# setting values
set.seed(1)
s2<-1
t2<-10
mu<-5; n<-5


# rounding the rnorm to 2 decimal places
y<-round(rnorm(n,10,1),2)
# mean of the normal posterior
mu.n<-( mean(y)*n/s2 + mu/t2 )/( n/s2+1/t2)
# variance of the normal posterior
t2.n<-1/(n/s2+1/t2)
# defining the data
y<-c(9.37, 10.18, 9.16, 11.60, 10.33)

####metropolis part####
##S = total num of simulations
theta<-0 ; delta<-2 ; S<-10000 ; THETA<-NULL ; set.seed(1)
for(s in 1:S){
  ## simulating our proposal
  #the new value of theta
  #print(theta)
  theta.star<-rnorm(1,theta,sqrt(delta))
  ##taking the log of the ratio r
  log.r<-( sum(dnorm(y,theta.star,sqrt(s2),log=TRUE))+
                dnorm(theta.star,mu,sqrt(t2),log=TRUE))-
          ( sum(dnorm(y,theta,sqrt(s2),log=TRUE))+
                 dnorm(theta,mu,sqrt(t2),log=TRUE))
  #print(log.r)
  if(log(runif(1))<log.r) { theta<-theta.star }
  ##updating THETA
  #print(log(runif(1)))
  THETA<-c(THETA,theta)
}

##two plots: trace of theta and comparing the empirical distribution
##of simulated values to the true posterior
par(mar=c(3,3,1,1),mgp=c(1.75,.75,0))
par(mfrow=c(1,2))
# creating a sequence
skeep<-seq(10,S,by=10)
# making a trace place
plot(skeep,THETA[skeep],type="l",
     xlab="iteration",ylab=expression(theta))
# making a histogram
hist(THETA[-(1:50)],prob=TRUE,main="",
     xlab=expression(theta),ylab="density")
th<-seq(min(THETA),max(THETA),length=100)
lines(th,dnorm(th,mu.n,sqrt(t2.n)) )
```

Figure. *Histogram for the Metropolis algorithm with r* shows a trace plot for this run as well as a histogram for the Metropolis algorithm compared with a draw from the true normal density.

Fig. 1: Histogram for the Metropolis algorithm with r

17.3.2 Python results

```
# coding: utf-8

# In[1]:

import numpy as np


# In[2]:

from scipy.stats import norm

def rnorm(n,mean,sd):
    """
    same functions as rnorm in r
    r: rnorm(n, mean=0, sd=1)
    py: rvs(loc=0, scale=1, size=1, random_state=None)
    """
    return norm.rvs(loc=mean,scale=sd,size=n)

def dnorm(x,mean,sd, log=False):
    """
    same functions as dnorm in r
```

(continues on next page)

(continued from previous page)

```
    dnorm(x, mean=0, sd=1, log=FALSE)
    pdf(x, loc=0, scale=1)
    """
    if log:
        return np.log(norm.pdf(x=x,loc=mean,scale=sd))
    else:
        return norm.pdf(x=x,loc=mean,scale=sd)

def runif(n,min=0, max=1):
    """
    r: runif(n, min = 0, max = 1)
    py: random.uniform(low=0.0, high=1.0, size=None)
    """
    return np.random.uniform(min,max,size=n)



# In[3]:

s2 = 1
t2 = 10
mu = 5
n = 5


# In[4]:

y = rnorm(n,10,1)
y


# In[5]:

# mean of the normal posterior
mu_n = (np.mean(y)*n/s2 + mu/float(t2))/(n/float(s2)+1/float(t2))
mu_n


# In[6]:

# variance of the normal posterior
# t2.n<-1/(n/s2+1/t2)

t2_n = 1.0/(n/float(s2)+1.0/t2)
t2_n


# In[7]:

# defining the data
# y<-c(9.37, 10.18, 9.16, 11.60, 10.33)
```

(continues on next page)

(continued from previous page)

```
y = [9.37, 10.18, 9.16, 11.60, 10.33]


# In[8]:

mu_n = (np.mean(y)*n/s2 + mu/float(t2))/(n/float(s2)+1/float(t2))
mu_n


# In[9]:

####metropolis part####
##S = total num of simulations
# theta<-0 ; delta<-2 ; S<-10000 ; THETA<-NULL ; set.seed(1)

theta = 0
delta = 2

S = 10000

theta_v = []


# In[ ]:

for s in range(S):
    theta_star = norm.rvs(theta,np.sqrt(delta),1)
    logr = (sum(dnorm(y,theta_star,np.sqrt(s2),log=True)) +
            sum(dnorm(theta_star,mu,np.sqrt(t2),log=True)))-
            (sum(dnorm(y,theta,np.sqrt(s2),log=True)) +
             sum(dnorm([theta],mu,np.sqrt(t2),log=True)))
    #print(logr)
    if np.log(runif(1))<logr:
        theta = theta_star
    #print(theta)
    theta_v.append(theta)


# In[ ]:

import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 8))

plt.subplot(1, 2, 1)
plt.plot(theta_v,'b-.')

plt.subplot(1, 2, 2)
#bins = np.arange(0, S, 10)
plt.hist(theta_v, density=True,bins='auto')
x = np.linspace(min(theta_v),max(theta_v),100)
```

(continues on next page)

(continued from previous page)

```
y = norm.pdf(x,mu_n,np.sqrt(t2_n))
plt.plot(x,y,'y-.')
plt.xlim(right=12)  # adjust the right leaving left unchanged
plt.xlim(left=8)  # adjust the left leaving right unchanged
plt.show()


# In[ ]:
```

Fig. 2: Histogram for the Metropolis algorithm with python

Figure. *Histogram for the Metropolis algorithm with python* shows a trace plot for this run as well as a histogram for the Metropolis algorithm compared with a draw from the true normal density.

17.3.3 PySpark results

TODO…

Figure. *Histogram for the Metropolis algorithm with PySpark* shows a trace plot for this run as well as a histogram for the Metropolis algorithm compared with a draw from the true normal density.

Fig. 3: Histogram for the Metropolis algorithm with PySpark

CHAPTER

EIGHTEEN

NEURAL NETWORK

Chinese proverb

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

18.1 Feedforward Neural Network

18.1.1 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.

Fig. 1: MultiLayer Neural Network

18.1.2 Demo

1. Set up spark context and SparkSession

```
from pyspark.sql import SparkSession

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

2. Load dataset

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

3. change categorical variable size

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

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

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

(continues on next page)

(continued from previous page)

```
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

NINETEEN

MY PYSPARK PACKAGE

It's super easy to wrap your own package in Python. I packed some functions which I frequently used in my daily work. You can download and install it from My PySpark Package. The hierarchical structure and the directory structure of this package are as follows.

19.1 Hierarchical Structure

```
|-- build
|   |-- bdist.linux-x86_64
|   |-- lib.linux-x86_64-2.7
|       |-- PySparkTools
|           |-- __init__.py
|           |-- Manipulation
|           |   |-- DataManipulation.py
|           |   |-- __init__.py
|           |-- Visualization
|               |-- __init__.py
|               |-- PyPlots.py
|-- dist
|   |-- PySParkTools-1.0-py2.7.egg
|-- __init__.py
|-- PySparkTools
|   |-- __init__.py
|   |-- Manipulation
|   |   |-- DataManipulation.py
|   |   |-- __init__.py
|   |-- Visualization
|       |-- __init__.py
|       |-- PyPlots.py
|       |-- PyPlots.pyc
|-- PySParkTools.egg-info
|   |-- dependency_links.txt
|   |-- PKG-INFO
|   |-- requires.txt
|   |-- SOURCES.txt
|   |-- top_level.txt
|-- README.md
|-- requirements.txt
```

(continues on next page)

(continued from previous page)

```
|-- setup.py
|-- test
    |-- spark-warehouse
    |-- test1.py
    |-- test2.py
```

From the above hierarchical structure, you will find that you have to have `__init__.py` in each directory. I will explain the `__init__.py` file with the example below:

19.2 Set Up

```
from setuptools import setup, find_packages

try:
    with open("README.md") as f:
        long_description = f.read()
except IOError:
    long_description = ""

try:
    with open("requirements.txt") as f:
        requirements = [x.strip() for x in f.read().splitlines() if x.strip()]
except IOError:
    requirements = []

setup(name='PySParkTools',
          install_requires=requirements,
      version='1.0',
      description='Python Spark Tools',
      author='Wenqiang Feng',
      author_email='von198@gmail.com',
      url='https://github.com/runawayhorse001/PySparkTools',
      packages=find_packages(),
      long_description=long_description
     )
```

19.3 ReadMe

```
# PySparkTools

This is my PySpark Tools. If you want to colne and install it, you can use

- clone

```{bash}
git clone git@github.com:runawayhorse001/PySparkTools.git
```
```

(continues on next page)

(continued from previous page)

```
```
- install

```{bash}
cd PySparkTools
pip install -r requirements.txt
python setup.py install
```

- test

```{bash}
cd PySparkTools/test
python test1.py
```
```
```

CHAPTER TWENTY

MY CHEAT SHEET

You can download the PDF version: PySpark Cheat Sheet and pdDataFrame vs rddDataFrame.

Cheat Sheet for PySpark

Wenqiang Feng
E-mail: von198@gmail.com, Web: http://web.utk.edu/~wfeng1; https://runawayhorse001.github.io/LearningApacheSpark

Spark Configuration

```
from pyspark.sql import SparkSession
spark = SparkSession.builder
        .appName("Python Spark regression example")
        .config("config.option", "value").getOrCreate()
```

Loading Data

From RDDs

```
# Using parallelize( )
df = spark.sparkContext.parallelize([('1','Joe','70000','1'),
                                     ('2', 'Henry', '80000', None)])
          .toDF(['Id', 'Name', 'Sallary','DepartmentId'])
# Using createDataFrame( )
df = spark.createDataFrame([('1', 'Joe',   '70000', '1'),
                            ('2', 'Henry', '80000', None)],
                   ['Id','Name','Sallary','DepartmentId'])
+----+-----+-------+------------+
| Id| Name|Sallary|DepartmentId|
+----+-----+-------+------------+
|  1|  Joe|  70000|           1|
|  2|Henry|  80000|        null|
+----+-----+-------+------------+
```

From Data Sources

▷ **From .csv**

```
ds = spark.read.csv(path='Advertising.csv',
                    sep=',',encoding='UTF-8',comment=None,
                    header=True,inferSchema=True)
+-----+-----+---------+-----+
|    TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
+-----+-----+---------+-----+
```

▷ **From .json**

```
df = spark.read.json('/home/feng/Desktop/data.json')
+---------+-----------------+-------------------+
|       id|         location|          timestamp|
+---------+-----------------+-------------------+
|2957256202|[72.1,DE,8086,52....|2019-02-23 22:36:52|
|2957256203|[598.5,BG,3963,42...|2019-02-23 22:36:52|
+---------+-----------------+-------------------+
```

▷ **From Database**

```
user = 'username'; pw ='password'
table_name = 'table_name'
url='jdbc:postgresql://##.###.###.##:5432/dataset?user='
        +user+'&password='+pw
p='driver':'org.postgresql.Driver','password':pw,'user':user
df = spark.read.jdbc(url=url,table=table_name,properties=p)
+-----+-----+---------+-----+
|    TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
+-----+-----+---------+-----+
```

▷ **From HDFS**

```
from pyspark.conf import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import HiveContext
sc= SparkContext('local','example')
hc = HiveContext(sc)
tf1 = sc.textFile("hdfs://###/user/data/file_name")
+-----+-----+---------+-----+
|    TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
+-----+-----+---------+-----+
```

©All Rights Reserved by Dr.Wenqiang Feng. Powered by LaTeX. Updated:02-26-2019. von198@gmail.com

Auditing Data

Checking schema

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

Checking missing value

```
from pyspark.sql.functions import count
def my_count(df):
    df.agg(*[count(c).alias(c) for c in df_in.columns]).show()
my_count(df_raw)
+--------+---------+-----------+--------+-----------+----------+-------+
|InvoiceNo|StockCode|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+--------+---------+-----------+--------+-----------+----------+-------+
|  541909|   541909|     541909|  541909|     541909|    406829| 541909|
+--------+---------+-----------+--------+-----------+----------+-------+
```

Checking statistical results

```
# function form my pyspark
df_raw.describe().show()
+-------+------------------+------------------+------------------+
|summary|                TV|             Radio|         Newspaper|
+-------+------------------+------------------+------------------+
|  count|               200|               200|               200|
|   mean|          147.0425|23.264000000000024|30.553999999999995|
| stddev|85.85423631490805|14.846809176168728| 21.778620838552283|
|    min|               0.7|               0.0|               0.3|
|    max|             296.4|              49.6|             114.0|
+-------+------------------+------------------+------------------+
```

Manipulating Data (More details on next page)

Fixing missing value

Function	Description
df.na.fill()	#Replace null values
df.na.drop()	#Dropping any rows with null values.

Joining data

Description	Function
#Data join	left.join(right,key, how='*') * = left,right,inner,full

Wrangling with UDF

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
# user defined function
def complexFun(x):
    return results
Fn = F.udf(lambda x: complexFun(x), DoubleType())
df.withColumn('2col', Fn(df.col))
```

Reducing features

```
df.select(featureNameList)
```

Modeling Pipeline

Deal with categorical feature and label data

```
 # Deal with categorical feature data
from pyspark.ml.feature import VectorIndexer
featureIndexer = VectorIndexer(inputCol="features",
                               outputCol="indexedFeatures",
                               maxCategories=4).fit(data)
featureIndexer.transform(data).show(2, True)
+--------------------+-----+--------------------+
|            features|label|     indexedFeatures|
+--------------------+-----+--------------------+
|(29,[1,11,14,16,1...|   mo|(29,[1,11,14,16,1...|
+--------------------+-----+--------------------+


# Deal with categorical label data
labelIndexer=StringIndexer(inputCol='label',
                           outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(2, True)
+--------------------+-----+------------+
|            features|label|indexedLabel|
+--------------------+-----+------------+
|(29,[1,11,14,16,1...|   mo|         0.0|
+--------------------+-----+------------+
```

Spliting the data to training and test data sets

```
(trainingData, testData) = data.randomSplit([0.6, 0.4])
```

Importing the model

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='indexedFeatures',
                        labelCol='indexedLabel')
```

Converting indexed labels back to original labels

```
from pyspark.ml.feature import IndexToString
labelConverter = IndexToString(inputCol="prediction",
                               outputCol="predictedLabel",
                               labels=labelIndexer.labels)
```

Wrapping Pipeline

```
pipeline = Pipeline(stages=[labelIndexer, featureIndexer,
                            lr,labelConverter])
```

Training model and making predictions

```
model = pipeline.fit(trainingData)
predictions = model.transform(testData)
predictions.select("features","label","predictedLabel").show(2)
+--------------------+-----+---------------+
|            features|label|predictedLabel|
+--------------------+-----+---------------+
|(29,[0,11,13,16,1...|   mo|             mo|
+--------------------+-----+---------------+
```

Evaluating

```
from pyspark.ml.evaluation import *
evaluator = MulticlassClassificationEvaluator(
            labelCol="indexedLabel",
            predictionCol="prediction", metricName="accuracy")
accu = evaluator.evaluate(predictions)
print("Test Error: %g, AUC: %g"%(1-accu,Summary.areaUnderROC))
Test Error: 0.0986395, AUC: 0.886664269877
```

Data Wrangling: Combining DataFrame

Mutating Joins

A

X1	X2
a	1
b	2
c	3

B

X1	X3
a	T
b	F
d	T

Result	Function

X1 X2 X3
a 1 T
b 2 F
c 3 T
	#Join matching rows from B to A
	#dplyr::left_join(A, B, by = "x1")
	A.join(B,'X1',how='left')
	.orderBy('X1', ascending=True).show()

X1 X2 X3
a 1 T
b 2 F
d NA T
	#Join matching rows from A to B
	#dplyr::right_join(A, B, by = "x1")
	A.join(B,'X1',how='right')
	.orderBy('X1', ascending=True).show()

X1 X2 X3
a 1 T
b 2 F
	#Retain only rows in both sets
	#dplyr::inner_join(A, B, by = "x1")
	A.join(B,'X1',how='inner')
	.orderBy('X1', ascending=True).show()

X1 X2 X3
a 1 T
b 2 F
c 3 NA
d NA T
	#Retain all values,all rows
	#dplyr::full_join(A, B, by = "x1")
	A.join(B,'X1',how='full')
	.orderBy('X1', ascending=True).show()

Filtering Joins

X1 X2
a 1
b 2

```
#All rows in A that have a match in B
#dplyr::semi_join(A, B, by = "x1")
a.join(b,'X1',how='left_semi')
 .orderBy('X1', ascending=True).show()
```

X1 X2
c 3

```
#All rows in A, don't have a match in B
#dplyr::anti_join(A, B, by = "x1")
A.join(B,'X1',how='left_anti')
 .orderBy('X1', ascending=True).show()
```

DataFrame Operations

Y

X1	X2
a	1
b	2
c	3

Z

X1	X2
b	2
c	3
d	4

Result	Function

X1 X2
a 1
b 2
c 3
	#Rows that appear in both Y and Z
	#dplyr::intersect(Y, Z)
	Y.intersect(Z).show()

X1 X2
a 1
b 2
c 3
d 4
	#Rows that appear in either or both Y and Z
	#dplyr::union(Y, Z)
	Y.union(Z).dropDuplicates()
	.orderBy('X1', ascending=True).show()

X1 X2
a 1
	#Rows that appear in Y but not Z
	#dplyr::setdiff(Y, Z)
	Y.subtract(Z).show()

Binding

X1 X2
a 1
b 2
c 3
b 2
c 3
d 4

```
#Append Z to Y as new rows
#dplyr::bind_rows(Y, Z)
Y.union(Z)
 .orderBy('X1', ascending=True).show()
```

X1 X2 X1 X2
a 1 N 2
b 2 O 3
c 3 P 4

```
#Append Z to Y as new columns
#Caution: zipDataFrames form my package
#dplyr::bind_cols(Y, Z)
zipDataFrames(Y,Z).show()
```

©All Rights Reserved by Dr.Wenqiang Feng. Powered by LaTeX. Updated:02-26-2019. von198@gmail.com

Data Wrangling: Reshaping Data

Spliting

Change	Function

key value
a 1,2,3
b 2,3,4
	#ArrayType() tidyr::separate ::separate one column into several
	df.select("key", df.value[0], df.value[1], df.value[2]).show()
	#StructType()
	df2.select('key', 'value.*').show()

key value a 1,2,3 b 2,3,4 → key value a 1 a 2 a 3 b 2 b 3 b 4	#Splitting one column into rows df.select("key",F.split("values", ",").alias("values"), F.posexplode(F.split("values",",")).alias("pos", "val")).drop("val") .select("key",F.expr("values[pos]").alias("val")).show()

key value
a 1
a 2
a 3
b 2
b 3
b 4 → key value
a 1
a 2
a 3
b 4
b 6
	#Gather columns into rows
	def to_long(df, by):
	cols, dtypes = zip(*((c, t) for (c, t) in df.dtypes if c not in by))
	# Spark SQL supports only homogeneous columns
	assert len(set(dtypes))==1,"All columns have to be of the same type"
	# Create and explode an array of (column_name, column_value) structs
	kvs = explode(array([
	struct(lit(c).alias("key"), col(c).alias("val")) for c in cols
	]).alias("kvs")
	return df.select(by + [kvs]).select(by + ["kvs.key", "kvs.val"])

Pivot

key value
a 1
a 2
a 3
b 2
b 3
b 4

→

key	col1	col2
a	1	2
b	2	3

```
#Spread rows into columns
df.groupBy(['key'])
 .pivot('col1').sum('col1').show()
```

Subset Observations (Rows)

Function	Description
df.na.drop()	#Omitting rows with null values
df.where()	#Filters rows using the given condition
df.filter()	#Filters rows using the given condition
df.distinct()	#Returns distinct rows in this DataFrame
df.sample()	#Returns a sampled subset of this DataFrame
df.sampleBy()	#Returns a stratified sample without replacement

Subset Variables (Columns)

Function	Description
df.select()	#Applys expressions and returns a new DataFrame

Make New Vaiables

Function	Examples
df.withColumn()	df.withColumn('new',1/df.col)
	df.withColumn('new',F.log(df.col))
	df.withColumn('id', psf.monotonically_increasing_id())
	df.withColumn("new", Fn('col')) #Fn:F.udf()
	df.withColumn('new', F.when((df.c1>1)&(df.c2<2),1) .when((df.c3>3),2).otherwise(3))

Data Wrangling: Reshaping Data

Summarise Data

Function	Description
df.describe()	#Computes simple statistics
Correlation.corr(df)	#Computes the correlation matrix
df.count()	#Count the number of rows

Description	Demo
#Sum	df.agg(F.max(df.C)).head()[0]#Similar for: F.min,max,avg,stddev

Group Data

A	min_b	max_b	avg_c
m	1	2	4.5
n	3	4	7.5

```
df.groupBy(['A'])
 .agg(F.min('B').alias('min_b'),
      F.max('B').alias('max_b'),
      F.avg('C').alias('avg_c')).show()
```

A	min_b	max_b	list_c
m	1	2	[4.0,5.0]
n	3	4	[7.0,7.5]

```
def quant_pd(val_list):
    quant = np.round(np.percentile(val_list,
                    [20,50,75]),2)
    return list(map(float,quant))
Fn = F.udf(quant_pd,ArrayType(FloatType()))
#GroupBy and aggregate
df.groupBy(['A'])
 .agg(F.min('B').alias('min_b'),
      F.max('B').alias('max_b'),
      Fn(F.collect_list(col('C'))).alias('list_c'))
```

Windows

Result	Function

A B C D
a m 1 0
m 2 1
c n 3 0
d n 6 1
	from pyspark.sql import Window
	#Define windows for difference
	w = Window.partitionBy(df.B)
	D = df.C - F.max(df.C).over(w)
	df.withColumn('D',D).show()

A B C D
a m 1 1
m 2 2
c n 3 3
d n 6 4
	df = df.withColumn("D",
	F.monotonically_increasing_id())
	#Define windows for row_num
	w = Window.orderBy("D")
	df.withColumn("D", F.row_number().over(w))

A B C D
b m 2 1
m 1 2
d n 6 1
c n 3 2
	#Define windows for rank
	w = Window.partitionBy('B')
	.orderBy(df.C.desc())
	df.withColumn("D",rank().over(w)).show()

Rename Vaiables

Function	Description
df.withColumnRenamed()	#Renaming an existing column

pd.DataFrame vs rdd.DataFrame

Wenqiang Feng
E-mail: von198@gmail.com, Web: http://web.utk.edu/~wfeng1; https://runawayhorse001.github.io/LearningApacheSpark

Loading DataFrame

Creating DataFrame

▷ **From List**

```
pd.DataFrame(my_list,columns= col_name)
spark.createDataFrame(my_list, col_name).show()
```

▷ **From Dictionary**

```
pd.DataFrame(d)
spark.createDataFrame(np.array(list(d.values())).T.tolist(),
                      list(d.keys())).show()
```

From Data Sources

▷ **From Database**

```
conn = psycopg2.connect(host=host, database=db_name,
                        user=user, password=pw)
cur = conn.cursor()
sql = """select * from table_name
      """.format(table_name=table_name)
dp = pd.read_sql(sql, conn)
-----------------------------------------------------------
url='jdbc:postgresql://'+host+':5432/'+db_name+'?user='+user
      +'&password='+pw
p='driver':'org.postgresql.Driver','password':pw,'user':user
ds=spark.read.jdbc(url=url, table=table_name, properties=p)
```

▷ **From .csv**

```
dp = pd.read_csv('Advertising.csv')
ds = spark.read.csv(path='Advertising.csv',
                    header=True,inferSchema=True)
```

▷ **From .json**

```
dp = pd.read_json("data/data.json")
ds = spark.read.json('data/data.json')
```

Basic Manipulation

Data Types

```
dp.dtypes
ds.dtypes
```

Count

```
dp.count()[1]
ds.count()
```

Column Names

```
dp.columns
ds.columns
```

Select Columns

```
dp[name_list].head()
ds[name_list].show()
```

Rename Columns

```
dp.columns = name_list
ds.toDF(*name_list).show()
```

Drop Columns

```
dp.drop(name_list,axis=1)
ds.drop(*name_list).show()
```

Distinct Rows

```
dp.drop_duplicates()
ds.drop_duplicates()
```

Cross Table

```
pd.crosstab(dp.col1,dp.col2)
ds.crosstab('col1','col2')
```

Replace Values

```
dp.A.replace(['male', 'female'],[1, 0], inplace=True)
ds.na.replace(['male','female'],['1','0']).show()
```

©All Rights Reserved by Dr.Wenqiang Feng. Powered by LaTeX. Updated:03-02-2019. von198@gmail.com

Basic Manipulation

Rename one or more columns

```
mapping = 'key1':'val1','key2':'val2'
dp.rename(columns=mapping).head(4)
new_names = [mapping.get(col,col) for col in ds.columns]
ds.toDF(*new_names).show(4)
```

Replace one or more data types

```
d = 'col2': 'str','col3':'str' # 'string' for pyspark
dp = dp.astype(d)
ds =  ds.select(*list(set(ds.columns)-set(d.keys())),
                *(col(c[0]).astype(c[1]).alias(c[0]) for c in d.items()))
```

Random Split

```
from sklearn.model_selection import train_test_split
a, b = train_test_split(dp, test_size=0.8)
a, b = ds.randomSplit([0.2,0.8])
```

Unixtime to Date

```
dp['date']=pd.to_datetime(dp['ts'],unit='s').dt.tz_localize('UTC')
spark.conf.set("spark.sql.session.timeZone", "UTC")
ds.withColumn('date', F.from_unixtime('ts'))
```

Make New Vaiables

```
dp['tv_norm'] = dp.TV/sum(dp.TV)
ds.withColumn('tv_norm', ds.TV/ds.groupBy()
                        .agg(F.sum("TV")).collect()[0][0]).show(4)
dp['cond'] = dp.apply(lambda c: 1 if ((c.TV>100)&(c.Radio<40))
                                  else 2 if c.Sales> 10
                                  else 3,axis=1)
ds.withColumn('cond',F.when((ds.TV>100)&(ds.Radio<40),1)
                      .when(ds.Sales>10, 2)
                      .otherwise(3)).show(4)
dp['log_tv'] = np.log(dp.TV)
ds.withColumn('log_tv',F.log(ds.TV)).show(4)
dp['tv+10'] = dp.TV.apply(lambda x: x+10)
ds.withColumn('tv+10', ds.TV+10).show(4)
```

Summarise Data

```
dp.describe()
ds.describe().show()
dp.corr(method='pearson')
mat=Statistics.corr(ds.rdd.map(lambda r: r[0:]),method='pearson')
pd.DataFrame(mat,columns=ds.columns,index=ds.columns)
```

```
dp.C.max() #Similar for: min,max,mean,std
ds.agg(F.max(df.C)).head()[0] #Similar for: min,max,avg,stddev
```

Mutating Joins

Result	Function

X1 X2 X3
a 1 T
b 2 F
c 3 T
	`#Join matching rows from B to A`
	`A.merge(B,on='X1',how='left')`
	`A.join(B,'X1',how='left')`
	`.orderBy('X1', ascending=True).show()`

X1 X2 X3
a 1 .
b 2 F
c null T
	`#Join matching rows from A to B`
	`A.merge(B,on='X1',how='right')`
	`A.join(B,'X1',how='right')`
	`.orderBy('X1', ascending=True).show()`

X1 X2 X3
a 1 T
b 2 F
	`#Retain only rows in both sets`
	`A.merge(B,on='X1',how='inner')`
	`A.join(B,'X1',how='inner')`
	`.orderBy('X1', ascending=True).show()`

X1 X2 X3
a 1 .
b 2 F
c 3 null
d null T
	`#Retain all values,all rows`
	`A.merge(B,on='X1',how='full')`
	`A.join(B,'X1',how='full')`
	`.orderBy('X1', ascending=True).show()`

Group Data

```
dp.groupby(['A']).agg('B':'min','C':'mean')
ds.groupBy(['A']).agg('B':'min','C':'avg').show()
```

Pivot

```
pd.pivot_table(dp, values='col1', index='key',
               columns='col2', aggfunc=np.sum)
-----------------------------------------------------------
df.groupBy(['key'])
  .pivot('col1').sum('col2').show()
```

Windows

```
dp['rank'] = dp.groupby('B')['C'].rank('dense', ascending=False)
-----------------------------------------------------------
w = Window.partitionBy('B').orderBy(ds.C.desc())
ds  = ds.withColumn('rank',F.rank().over(w))
```

SQL

```
sql = """
      SELECT * FROM table_name
      """.format(table_name=table_name)
dp = pd.read_sql(sql, conn)
#
ds.registerTempTable('ds')
spark.sql("SELECT * FROM ds").show()
```

CHAPTER

TWENTYONE

PYSPARK API

Those APIs are automatically generated from `PySpark` package, so all the CopyRights belong to Spark.

21.1 Stat API

class pyspark.ml.stat.**ChiSquareTest**

Note: Experimental

Conduct Pearson's independence test for every feature against the label. For each feature, the (feature, label) pairs are converted into a contingency matrix for which the Chi-squared statistic is computed. All label and feature values must be categorical.

The null hypothesis is that the occurrence of the outcomes is statistically independent.

New in version 2.2.0.

static test(*dataset*, *featuresCol*, *labelCol*)

Perform a Pearson's independence test using dataset.

Parameters

- **dataset** – DataFrame of categorical labels and categorical features. Real-valued features will be treated as categorical for each distinct value.
- **featuresCol** – Name of features column in dataset, of type $Vector$ ($VectorUDT$).
- **labelCol** – Name of label column in dataset, of any numerical type.

Returns DataFrame containing the test result for every feature against the label. This DataFrame will contain a single Row with the following fields: - $pValues$: $Vector$ - $degreesOfFreedom$: $Array[Int]$ - $statistics$: $Vector$ Each of these fields has one value per feature.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.stat import ChiSquareTest
>>> dataset = [[0, Vectors.dense([0, 0, 1])],
...            [0, Vectors.dense([1, 0, 1])],
...            [1, Vectors.dense([2, 1, 1])],
...            [1, Vectors.dense([3, 1, 1])]]
>>> dataset = spark.createDataFrame(dataset, ["label", "features"])
>>> chiSqResult = ChiSquareTest.test(dataset, 'features', 'label')
>>> chiSqResult.select("degreesOfFreedom").collect()[0]
Row(degreesOfFreedom=[3, 1, 0])
```

New in version 2.2.0.

class pyspark.ml.stat.**Correlation**

Note: Experimental

Compute the correlation matrix for the input dataset of Vectors using the specified method. Methods currently supported: $pearson$ (default), $spearman$.

Note: For Spearman, a rank correlation, we need to create an RDD[Double] for each column and sort it in order to retrieve the ranks and then join the columns back into an RDD[Vector], which is fairly costly. Cache the input Dataset before calling corr with $method =' spearman'$ to avoid recomputing the common lineage.

New in version 2.2.0.

static corr(*dataset*, *column*, *method='pearson'*)

Compute the correlation matrix with specified method using dataset.

Parameters

- **dataset** – A Dataset or a DataFrame.
- **column** – The name of the column of vectors for which the correlation coefficient needs to be computed. This must be a column of the dataset, and it must contain Vector objects.
- **method** – String specifying the method to use for computing correlation. Supported: $pearson$ (default), $spearman$.

Returns A DataFrame that contains the correlation matrix of the column of vectors. This DataFrame contains a single row and a single column of name '$METHODNAME($COLUMN)'.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.stat import Correlation
>>> dataset = [[Vectors.dense([1, 0, 0, -2])],
...            [Vectors.dense([4, 5, 0, 3])],
```

(continues on next page)

(continued from previous page)

```
...             [Vectors.dense([6, 7, 0, 8])],
...             [Vectors.dense([9, 0, 0, 1])]]
>>> dataset = spark.createDataFrame(dataset, ['features'])
>>> pearsonCorr = Correlation.corr(dataset, 'features', 'pearson').
↪collect()[0][0]
>>> print(str(pearsonCorr).replace('nan', 'NaN'))
DenseMatrix([[ 1.        ,  0.0556...,         NaN,  0.4004...],
             [ 0.0556...,  1.        ,         NaN,  0.9135...],
             [        NaN,         NaN,  1.        ,         NaN],
             [ 0.4004...,  0.9135...,         NaN,  1.        ]])
>>> spearmanCorr = Correlation.corr(dataset, 'features', method=
↪'spearman').collect()[0][0]
>>> print(str(spearmanCorr).replace('nan', 'NaN'))
DenseMatrix([[ 1.        ,  0.1054...,         NaN,  0.4       ],
             [ 0.1054...,  1.        ,         NaN,  0.9486... ],
             [        NaN,         NaN,  1.        ,         NaN],
             [ 0.4       ,  0.9486... ,         NaN,  1.        ]])
```

New in version 2.2.0.

class pyspark.ml.stat.**KolmogorovSmirnovTest**

Note: Experimental

Conduct the two-sided Kolmogorov Smirnov (KS) test for data sampled from a continuous distribution.

By comparing the largest difference between the empirical cumulative distribution of the sample data and the theoretical distribution we can provide a test for the the null hypothesis that the sample data comes from that theoretical distribution.

New in version 2.4.0.

static test(*dataset*, *sampleCol*, *distName*, **params*)

Conduct a one-sample, two-sided Kolmogorov-Smirnov test for probability distribution equality. Currently supports the normal distribution, taking as parameters the mean and standard deviation.

Parameters

- **dataset** – a Dataset or a DataFrame containing the sample of data to test.
- **sampleCol** – Name of sample column in dataset, of any numerical type.
- **distName** – a *string* name for a theoretical distribution, currently only support "norm".
- **params** – a list of *Double* values specifying the parameters to be used for the theoretical distribution. For "norm" distribution, the parameters includes mean and variance.

Returns A DataFrame that contains the Kolmogorov-Smirnov test result for the input sampled data. This DataFrame will contain a single Row with the following fields: - $pValue$: $Double$ - $statistic$: $Double$

```
>>> from pyspark.ml.stat import KolmogorovSmirnovTest
>>> dataset = [[-1.0], [0.0], [1.0]]
>>> dataset = spark.createDataFrame(dataset, ['sample'])
>>> ksResult = KolmogorovSmirnovTest.test(dataset, 'sample', 'norm',␣
↪0.0, 1.0).first()
>>> round(ksResult.pValue, 3)
1.0
>>> round(ksResult.statistic, 3)
0.175
>>> dataset = [[2.0], [3.0], [4.0]]
>>> dataset = spark.createDataFrame(dataset, ['sample'])
>>> ksResult = KolmogorovSmirnovTest.test(dataset, 'sample', 'norm',␣
↪3.0, 1.0).first()
>>> round(ksResult.pValue, 3)
1.0
>>> round(ksResult.statistic, 3)
0.175
```

New in version 2.4.0.

class pyspark.ml.stat.**Summarizer**

Note: Experimental

Tools for vectorized statistics on MLlib Vectors. The methods in this package provide various statistics for Vectors contained inside DataFrames. This class lets users pick the statistics they would like to extract for a given column.

```
>>> from pyspark.ml.stat import Summarizer
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> summarizer = Summarizer.metrics("mean", "count")
>>> df = sc.parallelize([Row(weight=1.0, features=Vectors.dense(1.0, 1.0,
↪ 1.0)),
...                      Row(weight=0.0, features=Vectors.dense(1.0, 2.0,
↪ 3.0))]).toDF()
>>> df.select(summarizer.summary(df.features, df.weight)).
↪show(truncate=False)
+-----------------------------------+
|aggregate_metrics(features, weight)|
+-----------------------------------+
|[[1.0,1.0,1.0], 1]                 |
+-----------------------------------+
<BLANKLINE>
>>> df.select(summarizer.summary(df.features)).show(truncate=False)
+--------------------------------+
```

(continues on next page)

(continued from previous page)

```
|aggregate_metrics(features, 1.0)|
+--------------------------------+
|[[1.0,1.5,2.0], 2]              |
+--------------------------------+
<BLANKLINE>
>>> df.select(Summarizer.mean(df.features, df.weight)).
↪show(truncate=False)
+--------------+
|mean(features)|
+--------------+
|[1.0,1.0,1.0] |
+--------------+
<BLANKLINE>
>>> df.select(Summarizer.mean(df.features)).show(truncate=False)
+--------------+
|mean(features)|
+--------------+
|[1.0,1.5,2.0] |
+--------------+
<BLANKLINE>
```

New in version 2.4.0.

static count(*col*, *weightCol=None*)

return a column of count summary

New in version 2.4.0.

static max(*col*, *weightCol=None*)

return a column of max summary

New in version 2.4.0.

static mean(*col*, *weightCol=None*)

return a column of mean summary

New in version 2.4.0.

static metrics(**metrics*)

Given a list of metrics, provides a builder that it turns computes metrics from a column.

See the documentation of [[Summarizer]] for an example.

The following metrics are accepted (case sensitive):

- mean: a vector that contains the coefficient-wise mean.
- variance: a vector tha contains the coefficient-wise variance.
- count: the count of all vectors seen.
- numNonzeros: a vector with the number of non-zeros for each coefficients
- max: the maximum for each coefficient.
- min: the minimum for each coefficient.

- normL2: the Euclidian norm for each coefficient.
- normL1: the L1 norm of each coefficient (sum of the absolute values).

Parameters **metrics** – metrics that can be provided.

Returns an object of *pyspark.ml.stat.SummaryBuilder*

Note: Currently, the performance of this interface is about 2x~3x slower then using the RDD interface.

New in version 2.4.0.

static min(*col*, *weightCol=None*)
return a column of min summary

New in version 2.4.0.

static normL1(*col*, *weightCol=None*)
return a column of normL1 summary

New in version 2.4.0.

static normL2(*col*, *weightCol=None*)
return a column of normL2 summary

New in version 2.4.0.

static numNonZeros(*col*, *weightCol=None*)
return a column of numNonZero summary

New in version 2.4.0.

static variance(*col*, *weightCol=None*)
return a column of variance summary

New in version 2.4.0.

class pyspark.ml.stat.**SummaryBuilder**(*jSummaryBuilder*)

Note: Experimental

A builder object that provides summary statistics about a given column.

Users should not directly create such builders, but instead use one of the methods in *pyspark.ml.stat.Summarizer*

New in version 2.4.0.

summary(*featuresCol*, *weightCol=None*)
Returns an aggregate object that contains the summary of the column with the requested metrics.

Parameters

- **featuresCol** – a column that contains features Vector object.

- **weightCol** – a column that contains weight value. Default weight is 1.0.

Returns an aggregate column that contains the statistics. The exact content of this structure is determined during the creation of the builder.

New in version 2.4.0.

21.2 Regression API

class pyspark.ml.regression.**AFTSurvivalRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', fitIntercept=True, maxIter=100, tol=1e-06, censorCol='censor', quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2*)

Note: Experimental

Accelerated Failure Time (AFT) Model Survival Regression

Fit a parametric AFT survival regression model based on the Weibull distribution of the survival time.

See also:

AFT Model

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0), 1.0),
...     (1e-40, Vectors.sparse(1, [], []), 0.0)], ["label", "features",
↪"censor"])
>>> aftsr = AFTSurvivalRegression()
>>> model = aftsr.fit(df)
>>> model.predict(Vectors.dense(6.3))
1.0
>>> model.predictQuantiles(Vectors.dense(6.3))
DenseVector([0.0101, 0.0513, 0.1054, 0.2877, 0.6931, 1.3863, 2.3026, 2.
↪9957, 4.6052])
>>> model.transform(df).show()
+-------+---------+------+----------+
|  label| features|censor|prediction|
+-------+---------+------+----------+
|    1.0|    [1.0]|   1.0|       1.0|
|1.0E-40|(1,[],[])|   0.0|       1.0|
```

(continues on next page)

(continued from previous page)

```
+-------+---------+------+----------+
...
>>> aftsr_path = temp_path + "/aftsr"
>>> aftsr.save(aftsr_path)
>>> aftsr2 = AFTSurvivalRegression.load(aftsr_path)
>>> aftsr2.getMaxIter()
100
>>> model_path = temp_path + "/aftsr_model"
>>> model.save(model_path)
>>> model2 = AFTSurvivalRegressionModel.load(model_path)
>>> model.coefficients == model2.coefficients
True
>>> model.intercept == model2.intercept
True
>>> model.scale == model2.scale
True
```

New in version 1.6.0.

getCensorCol()

Gets the value of censorCol or its default value.

New in version 1.6.0.

getQuantileProbabilities()

Gets the value of quantileProbabilities or its default value.

New in version 1.6.0.

getQuantilesCol()

Gets the value of quantilesCol or its default value.

New in version 1.6.0.

setCensorCol(*value*)

Sets the value of `censorCol`.

New in version 1.6.0.

setParams(*featuresCol='features', labelCol='label', predictionCol='prediction', fitIntercept=True, maxIter=100, tol=1e-06, censorCol='censor', quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2*)

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", fitIntercept=True, maxIter=100, tol=1E-6, censorCol="censor", quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2):

New in version 1.6.0.

setQuantileProbabilities(*value*)

Sets the value of `quantileProbabilities`.

New in version 1.6.0.

setQuantilesCol(*value*)
Sets the value of `quantilesCol`.

New in version 1.6.0.

class pyspark.ml.regression.**AFTSurvivalRegressionModel**(*java_model=None*)

Note: Experimental

Model fitted by `AFTSurvivalRegression`.

New in version 1.6.0.

coefficients
Model coefficients.

New in version 2.0.0.

intercept
Model intercept.

New in version 1.6.0.

predict(*features*)
Predicted value

New in version 2.0.0.

predictQuantiles(*features*)
Predicted Quantiles

New in version 2.0.0.

scale
Model scale parameter.

New in version 1.6.0.

class pyspark.ml.regression.**DecisionTreeRegressor**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', seed=None, varianceCol=None*)
Decision tree learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> dt = DecisionTreeRegressor(maxDepth=2, varianceCol="variance")
>>> model = dt.fit(df)
>>> model.depth
1
>>> model.numNodes
3
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
↪"features"])
>>> model.transform(test1).head().prediction
1.0
>>> dtr_path = temp_path + "/dtr"
>>> dt.save(dtr_path)
>>> dt2 = DecisionTreeRegressor.load(dtr_path)
>>> dt2.getMaxDepth()
2
>>> model_path = temp_path + "/dtr_model"
>>> model.save(model_path)
>>> model2 = DecisionTreeRegressionModel.load(model_path)
>>> model.numNodes == model2.numNodes
True
>>> model.depth == model2.depth
True
>>> model.transform(test1).head().variance
0.0
```

New in version 1.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="variance"*, *seed=None*, *varianceCol=None*)

Sets params for the DecisionTreeRegressor.

New in version 1.4.0.

class pyspark.ml.regression.**DecisionTreeRegressionModel**(*java_model=None*)

Model fitted by `DecisionTreeRegressor`.

New in version 1.4.0.

featureImportances

Estimate of the importance of each feature.

This generalizes the idea of "Gini" importance to other losses, following the explanation of Gini importance from "Random Forests" documentation by Leo Breiman and Adele Cutler, and following the implementation from scikit-learn.

This feature importance is calculated as follows:

- importance(feature j) = sum (over nodes which split on feature j) of the gain, where gain is scaled by the number of instances passing through node
- Normalize importances for tree to sum to 1.

Note: Feature importance for single decision trees can have high variance due to correlated predictor variables. Consider using a `RandomForestRegressor` to determine feature importance instead.

New in version 2.0.0.

class pyspark.ml.regression.**GBTRegressor**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, subsamplingRate=1.0, checkpointInterval=10, lossType='squared', maxIter=20, stepSize=0.1, seed=None, impurity='variance', featureSubsetStrategy='all'*)

Gradient-Boosted Trees (GBTs) learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> gbt = GBTRegressor(maxIter=5, maxDepth=2, seed=42)
>>> print(gbt.getImpurity())
variance
>>> print(gbt.getFeatureSubsetStrategy())
all
>>> model = gbt.fit(df)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
```

(continues on next page)

(continued from previous page)

```
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
↪"features"])
>>> model.transform(test1).head().prediction
1.0
>>> gbtr_path = temp_path + "gbtr"
>>> gbt.save(gbtr_path)
>>> gbt2 = GBTRegressor.load(gbtr_path)
>>> gbt2.getMaxDepth()
2
>>> model_path = temp_path + "gbtr_model"
>>> model.save(model_path)
>>> model2 = GBTRegressionModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
>>> model.treeWeights == model2.treeWeights
True
>>> model.trees
[DecisionTreeRegressionModel (uid=...) of depth...,␣
↪DecisionTreeRegressionModel...]
>>> validation = spark.createDataFrame([(0.0, Vectors.dense(-1.0))],
...             ["label", "features"])
>>> model.evaluateEachIteration(validation, "squared")
[0.0, 0.0, 0.0, 0.0, 0.0]
```

New in version 1.4.0.

getLossType()

Gets the value of lossType or its default value.

New in version 1.4.0.

setFeatureSubsetStrategy(*value*)

Sets the value of `featureSubsetStrategy`.

New in version 2.4.0.

setLossType(*value*)

Sets the value of `lossType`.

New in version 1.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *subsamplingRate=1.0*, *checkpointInterval=10*, *lossType="squared"*, *maxIter=20*, *stepSize=0.1*, *seed=None*, *impurity="variance"*, *featureSubsetStrategy="all"*)

Sets params for Gradient Boosted Tree Regression.

New in version 1.4.0.

class pyspark.ml.regression.**GBTRegressionModel**(*java_model=None*)

Model fitted by *GBTRegressor*.

New in version 1.4.0.

evaluateEachIteration(*dataset*, *loss*)

Method to compute error or loss for every iteration of gradient boosting.

Parameters

- **dataset** – Test dataset to evaluate model on, where dataset is an instance of `pyspark.sql.DataFrame`
- **loss** – The loss function used to compute error. Supported options: squared, absolute

New in version 2.4.0.

featureImportances

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie, Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also:

`DecisionTreeRegressionModel.featureImportances`

New in version 2.0.0.

trees

These have null parent Estimators.

New in version 2.0.0.

Type Trees in this ensemble. Warning

class pyspark.ml.regression.**GeneralizedLinearRegression**(*labelCol='label', featuresCol='features', predictionCol='prediction', family='gaussian', link=None, fitIntercept=True, maxIter=25, tol=1e-06, regParam=0.0, weightCol=None, solver='irls', linkPredictionCol=None, variancePower=0.0, linkPower=None, offsetCol=None*)

Note: Experimental

Generalized Linear Regression.

Fit a Generalized Linear Model specified by giving a symbolic description of the linear predictor (link function) and a description of the error distribution (family). It supports "gaussian", "binomial", "poisson", "gamma" and "tweedie" as family. Valid link functions for each family is listed below. The first link function of each family is the default one.

- "gaussian" -> "identity", "log", "inverse"
- "binomial" -> "logit", "probit", "cloglog"
- "poisson" -> "log", "identity", "sqrt"
- "gamma" -> "inverse", "identity", "log"
- "tweedie" -> power link function specified through "linkPower". The default link power in the tweedie family is 1 - variancePower.

See also:

GLM

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(0.0, 0.0)),
...     (1.0, Vectors.dense(1.0, 2.0)),
```

(continues on next page)

(continued from previous page)

```
...     (2.0, Vectors.dense(0.0, 0.0)),
...     (2.0, Vectors.dense(1.0, 1.0)),], ["label", "features"])
>>> glr = GeneralizedLinearRegression(family="gaussian", link="identity",
↪ linkPredictionCol="p")
>>> model = glr.fit(df)
>>> transformed = model.transform(df)
>>> abs(transformed.head().prediction - 1.5) < 0.001
True
>>> abs(transformed.head().p - 1.5) < 0.001
True
>>> model.coefficients
DenseVector([1.5..., -1.0...])
>>> model.numFeatures
2
>>> abs(model.intercept - 1.5) < 0.001
True
>>> glr_path = temp_path + "/glr"
>>> glr.save(glr_path)
>>> glr2 = GeneralizedLinearRegression.load(glr_path)
>>> glr.getFamily() == glr2.getFamily()
True
>>> model_path = temp_path + "/glr_model"
>>> model.save(model_path)
>>> model2 = GeneralizedLinearRegressionModel.load(model_path)
>>> model.intercept == model2.intercept
True
>>> model.coefficients[0] == model2.coefficients[0]
True
```

New in version 2.0.0.

getFamily()

Gets the value of family or its default value.

New in version 2.0.0.

getLink()

Gets the value of link or its default value.

New in version 2.0.0.

getLinkPower()

Gets the value of linkPower or its default value.

New in version 2.2.0.

getLinkPredictionCol()

Gets the value of linkPredictionCol or its default value.

New in version 2.0.0.

getOffsetCol()

Gets the value of offsetCol or its default value.

New in version 2.3.0.

getVariancePower()
Gets the value of variancePower or its default value.

New in version 2.2.0.

setFamily(*value*)
Sets the value of `family`.

New in version 2.0.0.

setLink(*value*)
Sets the value of `link`.

New in version 2.0.0.

setLinkPower(*value*)
Sets the value of `linkPower`.

New in version 2.2.0.

setLinkPredictionCol(*value*)
Sets the value of `linkPredictionCol`.

New in version 2.0.0.

setOffsetCol(*value*)
Sets the value of `offsetCol`.

New in version 2.3.0.

setParams(*self*, *labelCol="label"*, *featuresCol="features"*, *predictionCol="prediction"*, *family="gaussian"*, *link=None*, *fitIntercept=True*, *maxIter=25*, *tol=1e-6*, *regParam=0.0*, *weightCol=None*, *solver="irls"*, *linkPredictionCol=None*, *variancePower=0.0*, *linkPower=None*, *offsetCol=None*)
Sets params for generalized linear regression.

New in version 2.0.0.

setVariancePower(*value*)
Sets the value of `variancePower`.

New in version 2.2.0.

class pyspark.ml.regression.**GeneralizedLinearRegressionModel**(*java_model=None*)

Note: Experimental

Model fitted by `GeneralizedLinearRegression`.

New in version 2.0.0.

coefficients
Model coefficients.

New in version 2.0.0.

evaluate (*dataset*)

Evaluates the model on a test dataset.

Parameters **dataset** – Test dataset to evaluate model on, where dataset is an instance of `pyspark.sql.DataFrame`

New in version 2.0.0.

hasSummary

Indicates whether a training summary exists for this model instance.

New in version 2.0.0.

intercept

Model intercept.

New in version 2.0.0.

summary

Gets summary (e.g. residuals, deviance, pValues) of model on training set. An exception is thrown if $trainingSummaryisNone$.

New in version 2.0.0.

class `pyspark.ml.regression.`**GeneralizedLinearRegressionSummary** (*java_obj=None*)

Note: Experimental

Generalized linear regression results evaluated on a dataset.

New in version 2.0.0.

aic

Akaike's "An Information Criterion"(AIC) for the fitted model.

New in version 2.0.0.

degreesOfFreedom

Degrees of freedom.

New in version 2.0.0.

deviance

The deviance for the fitted model.

New in version 2.0.0.

dispersion

The dispersion of the fitted model. It is taken as 1.0 for the "binomial" and "poisson" families, and otherwise estimated by the residual Pearson's Chi-Squared statistic (which is defined as sum of the squares of the Pearson residuals) divided by the residual degrees of freedom.

New in version 2.0.0.

nullDeviance
The deviance for the null model.

New in version 2.0.0.

numInstances
Number of instances in DataFrame predictions.

New in version 2.2.0.

predictionCol
Field in *predictions* which gives the predicted value of each instance. This is set to a new column name if the original model's $predictionCol$ is not set.

New in version 2.0.0.

predictions
Predictions output by the model's $transform$ method.

New in version 2.0.0.

rank
The numeric rank of the fitted linear model.

New in version 2.0.0.

residualDegreeOfFreedom
The residual degrees of freedom.

New in version 2.0.0.

residualDegreeOfFreedomNull
The residual degrees of freedom for the null model.

New in version 2.0.0.

residuals(*residualsType='deviance'*)
Get the residuals of the fitted model by type.

Parameters **residualsType** – The type of residuals which should be returned. Supported options: deviance (default), pearson, working, and response.

New in version 2.0.0.

class pyspark.ml.regression.**GeneralizedLinearRegressionTrainingSummary**(*java_obj=None*)

Note: Experimental

Generalized linear regression training results.

New in version 2.0.0.

coefficientStandardErrors
Standard error of estimated coefficients and intercept.

If `GeneralizedLinearRegression.fitIntercept` is set to True, then the last element returned corresponds to the intercept.

New in version 2.0.0.

numIterations
Number of training iterations.

New in version 2.0.0.

pValues
Two-sided p-value of estimated coefficients and intercept.

If `GeneralizedLinearRegression.fitIntercept` is set to True, then the last element returned corresponds to the intercept.

New in version 2.0.0.

solver
The numeric solver used for training.

New in version 2.0.0.

tValues
T-statistic of estimated coefficients and intercept.

If `GeneralizedLinearRegression.fitIntercept` is set to True, then the last element returned corresponds to the intercept.

New in version 2.0.0.

class pyspark.ml.regression.**IsotonicRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', weightCol=None, isotonic=True, featureIndex=0*)

Currently implemented using parallelized pool adjacent violators algorithm. Only univariate (single feature) algorithm supported.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> ir = IsotonicRegression()
>>> model = ir.fit(df)
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> model.boundaries
DenseVector([0.0, 1.0])
>>> ir_path = temp_path + "/ir"
>>> ir.save(ir_path)
>>> ir2 = IsotonicRegression.load(ir_path)
>>> ir2.getIsotonic()
True
```

(continues on next page)

(continued from previous page)

```
>>> model_path = temp_path + "/ir_model"
>>> model.save(model_path)
>>> model2 = IsotonicRegressionModel.load(model_path)
>>> model.boundaries == model2.boundaries
True
>>> model.predictions == model2.predictions
True
```

New in version 1.6.0.

getFeatureIndex()
Gets the value of featureIndex or its default value.

getIsotonic()
Gets the value of isotonic or its default value.

setFeatureIndex(*value*)
Sets the value of `featureIndex`.

setIsotonic(*value*)
Sets the value of `isotonic`.

setParams(*featuresCol='features', labelCol='label', predictionCol='prediction', weightCol=None, isotonic=True, featureIndex=0*)
setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", weightCol=None, isotonic=True, featureIndex=0): Set the params for IsotonicRegression.

class pyspark.ml.regression.**IsotonicRegressionModel**(*java_model=None*)
Model fitted by *IsotonicRegression*.

New in version 1.6.0.

boundaries
Boundaries in increasing order for which predictions are known.

New in version 1.6.0.

predictions
Predictions associated with the boundaries at the same index, monotone because of isotonic regression.

New in version 1.6.0.

class pyspark.ml.regression.**LinearRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, standardization=True, solver='auto', weightCol=None, aggregationDepth=2, loss='squaredError', epsilon=1.35*)
Linear regression.

The learning objective is to minimize the specified loss function, with regularization. This supports two kinds of loss:

- squaredError (a.k.a squared loss)
- huber (a hybrid of squared error for relatively small errors and absolute error for relatively large ones, and we estimate the scale parameter from training data)

This supports multiple types of regularization:

- none (a.k.a. ordinary least squares)
- L2 (ridge regression)
- L1 (Lasso)
- L2 + L1 (elastic net)

Note: Fitting with huber loss only supports none and L2 regularization.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, 2.0, Vectors.dense(1.0)),
...     (0.0, 2.0, Vectors.sparse(1, [], []))], ["label", "weight",
↪"features"])
>>> lr = LinearRegression(maxIter=5, regParam=0.0, solver="normal",␣
↪weightCol="weight")
>>> model = lr.fit(df)
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> abs(model.transform(test0).head().prediction - (-1.0)) < 0.001
True
>>> abs(model.coefficients[0] - 1.0) < 0.001
True
>>> abs(model.intercept - 0.0) < 0.001
True
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
↪"features"])
>>> abs(model.transform(test1).head().prediction - 1.0) < 0.001
True
>>> lr.setParams("vector")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> lr_path = temp_path + "/lr"
>>> lr.save(lr_path)
>>> lr2 = LinearRegression.load(lr_path)
>>> lr2.getMaxIter()
5
>>> model_path = temp_path + "/lr_model"
>>> model.save(model_path)
>>> model2 = LinearRegressionModel.load(model_path)
>>> model.coefficients[0] == model2.coefficients[0]
True
>>> model.intercept == model2.intercept
True
```

(continues on next page)

(continued from previous page)

```
>>> model.numFeatures
1
>>> model.write().format("pmml").save(model_path + "_2")
```

New in version 1.4.0.

getEpsilon()

Gets the value of epsilon or its default value.

New in version 2.3.0.

setEpsilon(*value*)

Sets the value of `epsilon`.

New in version 2.3.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxIter=100*, *regParam=0.0*, *elasticNetParam=0.0*, *tol=1e-6*, *fitIntercept=True*, *standardization=True*, *solver="auto"*, *weightCol=None*, *aggregationDepth=2*, *loss="squaredError"*, *epsilon=1.35*)

Sets params for linear regression.

New in version 1.4.0.

class pyspark.ml.regression.**LinearRegressionModel**(*java_model=None*)

Model fitted by `LinearRegression`.

New in version 1.4.0.

coefficients

Model coefficients.

New in version 2.0.0.

evaluate(*dataset*)

Evaluates the model on a test dataset.

Parameters **dataset** – Test dataset to evaluate model on, where dataset is an instance of `pyspark.sql.DataFrame`

New in version 2.0.0.

hasSummary

Indicates whether a training summary exists for this model instance.

New in version 2.0.0.

intercept

Model intercept.

New in version 1.4.0.

scale

The value by which $\|y - X'w\|$ is scaled down when loss is "huber", otherwise 1.0.

New in version 2.3.0.

summary

Gets summary (e.g. residuals, mse, r-squared) of model on training set. An exception is thrown if $trainingSummaryisNone$.

New in version 2.0.0.

class pyspark.ml.regression.**LinearRegressionSummary**(*java_obj=None*)

Note: Experimental

Linear regression results evaluated on a dataset.

New in version 2.0.0.

coefficientStandardErrors

Standard error of estimated coefficients and intercept. This value is only available when using the "normal" solver.

If `LinearRegression.fitIntercept` is set to True, then the last element returned corresponds to the intercept.

See also:

`LinearRegression.solver`

New in version 2.0.0.

degreesOfFreedom

Degrees of freedom.

New in version 2.2.0.

devianceResiduals

The weighted residuals, the usual residuals rescaled by the square root of the instance weights.

New in version 2.0.0.

explainedVariance

Returns the explained variance regression score. explainedVariance = $1 - \frac{variance(y-\hat{y})}{variance(y)}$

See also:

Wikipedia explain variation

Note: This ignores instance weights (setting all to 1.0) from $LinearRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

featuresCol

Field in "predictions" which gives the features of each instance as a vector.

New in version 2.0.0.

labelCol

Field in "predictions" which gives the true label of each instance.

New in version 2.0.0.

meanAbsoluteError

Returns the mean absolute error, which is a risk function corresponding to the expected value of the absolute error loss or l1-norm loss.

Note: This ignores instance weights (setting all to 1.0) from $LinearRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

meanSquaredError

Returns the mean squared error, which is a risk function corresponding to the expected value of the squared error loss or quadratic loss.

Note: This ignores instance weights (setting all to 1.0) from $LinearRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

numInstances

Number of instances in DataFrame predictions

New in version 2.0.0.

pValues

Two-sided p-value of estimated coefficients and intercept. This value is only available when using the "normal" solver.

If `LinearRegression.fitIntercept` is set to True, then the last element returned corresponds to the intercept.

See also:

`LinearRegression.solver`

New in version 2.0.0.

predictionCol

Field in "predictions" which gives the predicted value of the label at each instance.

New in version 2.0.0.

predictions

Dataframe outputted by the model's $transform$ method.

New in version 2.0.0.

r2

Returns R^2, the coefficient of determination.

See also:

Wikipedia coefficient of determination

Note: This ignores instance weights (setting all to 1.0) from $LinearRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

r2adj

Returns Adjusted R^2, the adjusted coefficient of determination.

See also:

Wikipedia coefficient of determination, Adjusted R^2

Note: This ignores instance weights (setting all to 1.0) from $LinearRegression.weightCol$. This will change in later Spark versions.

New in version 2.4.0.

residuals

Residuals (label - predicted value)

New in version 2.0.0.

rootMeanSquaredError

Returns the root mean squared error, which is defined as the square root of the mean squared error.

Note: This ignores instance weights (setting all to 1.0) from $LinearRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

tValues

T-statistic of estimated coefficients and intercept. This value is only available when using the "normal" solver.

If `LinearRegression.fitIntercept` is set to True, then the last element returned corresponds to the intercept.

See also:

`LinearRegression.solver`

New in version 2.0.0.

class pyspark.ml.regression.**LinearRegressionTrainingSummary**(*java_obj=None*)

Note: Experimental

Linear regression training results. Currently, the training summary ignores the training weights except for the objective trace.

New in version 2.0.0.

objectiveHistory

Objective function (scaled loss + regularization) at each iteration. This value is only available when using the "l-bfgs" solver.

See also:

`LinearRegression.solver`

New in version 2.0.0.

totalIterations

Number of training iterations until termination. This value is only available when using the "l-bfgs" solver.

See also:

`LinearRegression.solver`

New in version 2.0.0.

class pyspark.ml.regression.**RandomForestRegressor**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', subsamplingRate=1.0, seed=None, numTrees=20, featureSubsetStrategy='auto'*)

Random Forest learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> rf = RandomForestRegressor(numTrees=2, maxDepth=2, seed=42)
>>> model = rf.fit(df)
>>> model.featureImportances
```

(continues on next page)

(continued from previous page)

```
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 1.0])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> model.numFeatures
1
>>> model.trees
[DecisionTreeRegressionModel (uid=...) of depth..., DecisionTreeRegressionModel...]
>>> model.getNumTrees
2
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
0.5
>>> rfr_path = temp_path + "/rfr"
>>> rf.save(rfr_path)
>>> rf2 = RandomForestRegressor.load(rfr_path)
>>> rf2.getNumTrees()
2
>>> model_path = temp_path + "/rfr_model"
>>> model.save(model_path)
>>> model2 = RandomForestRegressionModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

New in version 1.4.0.

setFeatureSubsetStrategy(*value*)

Sets the value of `featureSubsetStrategy`.

New in version 2.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="variance"*, *subsamplingRate=1.0*, *seed=None*, *numTrees=20*, *featureSubsetStrategy="auto"*)

Sets params for linear regression.

New in version 1.4.0.

class pyspark.ml.regression.**RandomForestRegressionModel**(*java_model=None*)

Model fitted by *RandomForestRegressor*.

New in version 1.4.0.

featureImportances

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie,

Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also:

DecisionTreeRegressionModel.featureImportances

New in version 2.0.0.

trees

These have null parent Estimators.

New in version 2.0.0.

Type Trees in this ensemble. Warning

21.3 Classification API

class pyspark.ml.classification.**LinearSVC**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2*)

Note: Experimental

Linear SVM Classifier

This binary classifier optimizes the Hinge Loss using the OWLQN optimizer. Only supports L2 regularization currently.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = sc.parallelize([
...     Row(label=1.0, features=Vectors.dense(1.0, 1.0, 1.0)),
...     Row(label=0.0, features=Vectors.dense(1.0, 2.0, 3.0))]).toDF()
>>> svm = LinearSVC(maxIter=5, regParam=0.01)
>>> model = svm.fit(df)
>>> model.coefficients
DenseVector([0.0, -0.2792, -0.1833])
>>> model.intercept
1.0206118982229047
>>> model.numClasses
2
>>> model.numFeatures
3
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, -1.0, -1.
↪0))]).toDF()
```

(continues on next page)

(continued from previous page)

```
>>> result = model.transform(test0).head()
>>> result.prediction
1.0
>>> result.rawPrediction
DenseVector([-1.4831, 1.4831])
>>> svm_path = temp_path + "/svm"
>>> svm.save(svm_path)
>>> svm2 = LinearSVC.load(svm_path)
>>> svm2.getMaxIter()
5
>>> model_path = temp_path + "/svm_model"
>>> model.save(model_path)
>>> model2 = LinearSVCModel.load(model_path)
>>> model.coefficients[0] == model2.coefficients[0]
True
>>> model.intercept == model2.intercept
True
```

New in version 2.2.0.

setParams(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxIter=100*, *regParam=0.0*, *tol=1e-06*, *rawPredictionCol='rawPrediction'*, *fitIntercept=True*, *standardization=True*, *threshold=0.0*, *weightCol=None*, *aggregationDepth=2*)

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, regParam=0.0, tol=1e-6, rawPredictionCol="rawPrediction", fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2): Sets params for Linear SVM Classifier.

New in version 2.2.0.

class pyspark.ml.classification.**LinearSVCModel**(*java_model=None*)

Note: Experimental

Model fitted by LinearSVC.

New in version 2.2.0.

coefficients

Model coefficients of Linear SVM Classifier.

New in version 2.2.0.

intercept

Model intercept of Linear SVM Classifier.

New in version 2.2.0.

class pyspark.ml.classification.**LogisticRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol='probability', rawPredictionCol='rawPrediction', standardization=True, weightCol=None, aggregationDepth=2, family='auto', lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None*)

Logistic regression. This class supports multinomial logistic (softmax) and binomial logistic regression.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> bdf = sc.parallelize([
...     Row(label=1.0, weight=1.0, features=Vectors.dense(0.0, 5.0)),
...     Row(label=0.0, weight=2.0, features=Vectors.dense(1.0, 2.0)),
...     Row(label=1.0, weight=3.0, features=Vectors.dense(2.0, 1.0)),
...     Row(label=0.0, weight=4.0, features=Vectors.dense(3.0, 3.0))]).
↪toDF()
>>> blor = LogisticRegression(regParam=0.01, weightCol="weight")
>>> blorModel = blor.fit(bdf)
>>> blorModel.coefficients
DenseVector([-1.080..., -0.646...])
>>> blorModel.intercept
3.112...
>>> data_path = "data/mllib/sample_multiclass_classification_data.txt"
>>> mdf = spark.read.format("libsvm").load(data_path)
>>> mlor = LogisticRegression(regParam=0.1, elasticNetParam=1.0, family=
↪"multinomial")
>>> mlorModel = mlor.fit(mdf)
>>> mlorModel.coefficientMatrix
SparseMatrix(3, 4, [0, 1, 2, 3], [3, 2, 1], [1.87..., -2.75..., -0.50...
↪], 1)
>>> mlorModel.interceptVector
```

(continues on next page)

(continued from previous page)

```
DenseVector([0.04..., -0.42..., 0.37...])
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, 1.0))]).
↪toDF()
>>> result = blorModel.transform(test0).head()
>>> result.prediction
1.0
>>> result.probability
DenseVector([0.02..., 0.97...])
>>> result.rawPrediction
DenseVector([-3.54..., 3.54...])
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(2, [0], [1.
↪0]))]).toDF()
>>> blorModel.transform(test1).head().prediction
1.0
>>> blor.setParams("vector")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> lr_path = temp_path + "/lr"
>>> blor.save(lr_path)
>>> lr2 = LogisticRegression.load(lr_path)
>>> lr2.getRegParam()
0.01
>>> model_path = temp_path + "/lr_model"
>>> blorModel.save(model_path)
>>> model2 = LogisticRegressionModel.load(model_path)
>>> blorModel.coefficients[0] == model2.coefficients[0]
True
>>> blorModel.intercept == model2.intercept
True
>>> model2
LogisticRegressionModel: uid = ..., numClasses = 2, numFeatures = 2
```

New in version 1.3.0.

getFamily()

Gets the value of `family` or its default value.

New in version 2.1.0.

getLowerBoundsOnCoefficients()

Gets the value of `lowerBoundsOnCoefficients`

New in version 2.3.0.

getLowerBoundsOnIntercepts()

Gets the value of `lowerBoundsOnIntercepts`

New in version 2.3.0.

getThreshold()

Get threshold for binary classification.

If `thresholds` is set with length 2 (i.e., binary classification), this returns the equivalent

threshold: $\frac{1}{1+\frac{thresholds(0)}{thresholds(1)}}$. Otherwise, returns `threshold` if set or its default value if unset.

New in version 1.4.0.

getThresholds()

If `thresholds` is set, return its value. Otherwise, if `threshold` is set, return the equivalent thresholds for binary classification: (1-threshold, threshold). If neither are set, throw an error.

New in version 1.5.0.

getUpperBoundsOnCoefficients()

Gets the value of `upperBoundsOnCoefficients`

New in version 2.3.0.

getUpperBoundsOnIntercepts()

Gets the value of `upperBoundsOnIntercepts`

New in version 2.3.0.

setFamily(*value*)

Sets the value of `family`.

New in version 2.1.0.

setLowerBoundsOnCoefficients(*value*)

Sets the value of `lowerBoundsOnCoefficients`

New in version 2.3.0.

setLowerBoundsOnIntercepts(*value*)

Sets the value of `lowerBoundsOnIntercepts`

New in version 2.3.0.

setParams(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxIter=100*, *regParam=0.0*, *elasticNetParam=0.0*, *tol=1e-06*, *fitIntercept=True*, *threshold=0.5*, *thresholds=None*, *probabilityCol='probability'*, *rawPredictionCol='rawPrediction'*, *standardization=True*, *weightCol=None*, *aggregationDepth=2*, *family='auto'*, *lowerBoundsOnCoefficients=None*, *upperBoundsOnCoefficients=None*, *lowerBoundsOnIntercepts=None*, *upperBoundsOnIntercepts=None*)

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-6, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol="probability", rawPredictionCol="rawPrediction", standardization=True, weightCol=None, aggregationDepth=2, family="auto", lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None): Sets params for logistic regression. If the threshold and thresholds Params are both set, they must be equivalent.

New in version 1.3.0.

setThreshold(*value*)

Sets the value of `threshold`. Clears value of `thresholds` if it has been set.

New in version 1.4.0.

setThresholds(*value*)
Sets the value of `thresholds`. Clears value of `threshold` if it has been set.

New in version 1.5.0.

setUpperBoundsOnCoefficients(*value*)
Sets the value of `upperBoundsOnCoefficients`

New in version 2.3.0.

setUpperBoundsOnIntercepts(*value*)
Sets the value of `upperBoundsOnIntercepts`

New in version 2.3.0.

class `pyspark.ml.classification.`**LogisticRegressionModel**(*java_model=None*)
Model fitted by LogisticRegression.

New in version 1.3.0.

coefficientMatrix
Model coefficients.

New in version 2.1.0.

coefficients
Model coefficients of binomial logistic regression. An exception is thrown in the case of multinomial logistic regression.

New in version 2.0.0.

evaluate(*dataset*)
Evaluates the model on a test dataset.

Parameters **dataset** – Test dataset to evaluate model on, where dataset is an instance of `pyspark.sql.DataFrame`

New in version 2.0.0.

hasSummary
Indicates whether a training summary exists for this model instance.

New in version 2.0.0.

intercept
Model intercept of binomial logistic regression. An exception is thrown in the case of multinomial logistic regression.

New in version 1.4.0.

interceptVector
Model intercept.

New in version 2.1.0.

summary
Gets summary (e.g. accuracy/precision/recall, objective history, total iterations) of model trained on the training set. An exception is thrown if $trainingSummaryisNone$.

New in version 2.0.0.

class pyspark.ml.classification.**LogisticRegressionSummary**(*java_obj=None*)

Note: Experimental

Abstraction for Logistic Regression Results for a given model.

New in version 2.0.0.

accuracy

Returns accuracy. (equals to the total number of correctly classified instances out of the total number of instances.)

New in version 2.3.0.

fMeasureByLabel(*beta=1.0*)

Returns f-measure for each label (category).

New in version 2.3.0.

falsePositiveRateByLabel

Returns false positive rate for each label (category).

New in version 2.3.0.

featuresCol

Field in "predictions" which gives the features of each instance as a vector.

New in version 2.0.0.

labelCol

Field in "predictions" which gives the true label of each instance.

New in version 2.0.0.

labels

Returns the sequence of labels in ascending order. This order matches the order used in metrics which are specified as arrays over labels, e.g., truePositiveRateByLabel.

Note: In most cases, it will be values {0.0, 1.0, ..., numClasses-1}, However, if the training set is missing a label, then all of the arrays over labels (e.g., from truePositiveRateByLabel) will be of length numClasses-1 instead of the expected numClasses.

New in version 2.3.0.

precisionByLabel

Returns precision for each label (category).

New in version 2.3.0.

predictionCol

Field in "predictions" which gives the prediction of each class.

New in version 2.3.0.

predictions
Dataframe outputted by the model's $transform$ method.

New in version 2.0.0.

probabilityCol
Field in "predictions" which gives the probability of each class as a vector.

New in version 2.0.0.

recallByLabel
Returns recall for each label (category).

New in version 2.3.0.

truePositiveRateByLabel
Returns true positive rate for each label (category).

New in version 2.3.0.

weightedFMeasure(*beta=1.0*)
Returns weighted averaged f-measure.

New in version 2.3.0.

weightedFalsePositiveRate
Returns weighted false positive rate.

New in version 2.3.0.

weightedPrecision
Returns weighted averaged precision.

New in version 2.3.0.

weightedRecall
Returns weighted averaged recall. (equals to precision, recall and f-measure)

New in version 2.3.0.

weightedTruePositiveRate
Returns weighted true positive rate. (equals to precision, recall and f-measure)

New in version 2.3.0.

class pyspark.ml.classification.**LogisticRegressionTrainingSummary**(*java_obj=None*)

Note: Experimental

Abstraction for multinomial Logistic Regression Training results. Currently, the training summary ignores the training weights except for the objective trace.

New in version 2.0.0.

objectiveHistory
Objective function (scaled loss + regularization) at each iteration.

New in version 2.0.0.

totalIterations

Number of training iterations until termination.

New in version 2.0.0.

class pyspark.ml.classification.**BinaryLogisticRegressionSummary**(*java_obj=None*)

Note: Experimental

Binary Logistic regression results for a given model.

New in version 2.0.0.

areaUnderROC

Computes the area under the receiver operating characteristic (ROC) curve.

Note: This ignores instance weights (setting all to 1.0) from $LogisticRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

fMeasureByThreshold

Returns a dataframe with two fields (threshold, F-Measure) curve with beta = 1.0.

Note: This ignores instance weights (setting all to 1.0) from $LogisticRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

pr

Returns the precision-recall curve, which is a Dataframe containing two fields recall, precision with (0.0, 1.0) prepended to it.

Note: This ignores instance weights (setting all to 1.0) from $LogisticRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

precisionByThreshold

Returns a dataframe with two fields (threshold, precision) curve. Every possible probability obtained in transforming the dataset are used as thresholds used in calculating the precision.

Note: This ignores instance weights (setting all to 1.0) from $LogisticRegression.weightCol$.

This will change in later Spark versions.

New in version 2.0.0.

recallByThreshold

Returns a dataframe with two fields (threshold, recall) curve. Every possible probability obtained in transforming the dataset are used as thresholds used in calculating the recall.

Note: This ignores instance weights (setting all to 1.0) from $LogisticRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

roc

Returns the receiver operating characteristic (ROC) curve, which is a Dataframe having two fields (FPR, TPR) with (0.0, 0.0) prepended and (1.0, 1.0) appended to it.

See also:

Wikipedia reference

Note: This ignores instance weights (setting all to 1.0) from $LogisticRegression.weightCol$. This will change in later Spark versions.

New in version 2.0.0.

class pyspark.ml.classification.**BinaryLogisticRegressionTrainingSummary**(*java_obj=None*

Note: Experimental

Binary Logistic regression training results for a given model.

New in version 2.0.0.

class pyspark.ml.classification.**DecisionTreeClassifier**(*featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', seed=None*)

Decision tree learning algorithm for classification. It supports both binary and multiclass labels, as well as both continuous and categorical features.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> dt = DecisionTreeClassifier(maxDepth=2, labelCol="indexed")
>>> model = dt.fit(td)
>>> model.numNodes
3
>>> model.depth
1
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> model.numClasses
2
>>> print(model.toDebugString)
DecisionTreeClassificationModel (uid=...) of depth 1 with 3 nodes...
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
```

(continues on next page)

(continued from previous page)

```
>>> result.probability
DenseVector([1.0, 0.0])
>>> result.rawPrediction
DenseVector([1.0, 0.0])
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
↪"features"])
>>> model.transform(test1).head().prediction
1.0
```

```
>>> dtc_path = temp_path + "/dtc"
>>> dt.save(dtc_path)
>>> dt2 = DecisionTreeClassifier.load(dtc_path)
>>> dt2.getMaxDepth()
2
>>> model_path = temp_path + "/dtc_model"
>>> model.save(model_path)
>>> model2 = DecisionTreeClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

New in version 1.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="gini"*, *seed=None*)

Sets params for the DecisionTreeClassifier.

New in version 1.4.0.

class pyspark.ml.classification.**DecisionTreeClassificationModel**(*java_model=None*)

Model fitted by DecisionTreeClassifier.

New in version 1.4.0.

featureImportances

Estimate of the importance of each feature.

This generalizes the idea of "Gini" importance to other losses, following the explanation of Gini importance from "Random Forests" documentation by Leo Breiman and Adele Cutler, and following the implementation from scikit-learn.

This feature importance is calculated as follows:

- importance(feature j) = sum (over nodes which split on feature j) of the gain, where gain is scaled by the number of instances passing through node
- Normalize importances for tree to sum to 1.

Note: Feature importance for single decision trees can have high variance due to correlated predictor variables. Consider using a `RandomForestClassifier` to determine feature im-

portance instead.

New in version 2.0.0.

class pyspark.ml.classification.**GBTClassifier**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, lossType='logistic', maxIter=20, stepSize=0.1, seed=None, subsamplingRate=1.0, featureSubsetStrategy='all'*)

Gradient-Boosted Trees (GBTs) learning algorithm for classification. It supports binary labels, as well as both continuous and categorical features.

The implementation is based upon: J.H. Friedman. "Stochastic Gradient Boosting." 1999.

Notes on Gradient Boosting vs. TreeBoost: - This implementation is for Stochastic Gradient Boosting, not for TreeBoost. - Both algorithms learn tree ensembles by minimizing loss functions. - TreeBoost (Friedman, 1999) additionally modifies the outputs at tree leaf nodes based on the loss function, whereas the original gradient boosting method does not. - We expect to implement TreeBoost in the future: SPARK-4240

Note: Multiclass labels are not currently supported.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> gbt = GBTClassifier(maxIter=5, maxDepth=2, labelCol="indexed",␣
↪seed=42)
>>> gbt.getFeatureSubsetStrategy()
'all'
>>> model = gbt.fit(td)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
```

(continues on next page)

(continued from previous page)

```
>>> model.transform(test0).head().prediction
0.0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
↪"features"])
>>> model.transform(test1).head().prediction
1.0
>>> model.totalNumNodes
15
>>> print(model.toDebugString)
GBTClassificationModel (uid=...)...with 5 trees...
>>> gbtc_path = temp_path + "gbtc"
>>> gbt.save(gbtc_path)
>>> gbt2 = GBTClassifier.load(gbtc_path)
>>> gbt2.getMaxDepth()
2
>>> model_path = temp_path + "gbtc_model"
>>> model.save(model_path)
>>> model2 = GBTClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
>>> model.treeWeights == model2.treeWeights
True
>>> model.trees
[DecisionTreeRegressionModel (uid=...) of depth...,␣
↪DecisionTreeRegressionModel...]
>>> validation = spark.createDataFrame([(0.0, Vectors.dense(-1.0),)],
...             ["indexed", "features"])
>>> model.evaluateEachIteration(validation)
[0.25..., 0.23..., 0.21..., 0.19..., 0.18...]
>>> model.numClasses
2
```

New in version 1.4.0.

getLossType()

Gets the value of lossType or its default value.

New in version 1.4.0.

setFeatureSubsetStrategy(*value*)

Sets the value of `featureSubsetStrategy`.

New in version 2.4.0.

setLossType(*value*)

Sets the value of `lossType`.

New in version 1.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *lossType="logistic"*, *maxIter=20*, *stepSize=0.1*, *seed=None*, *subsamplingRate=1.0*, *featureSubsetStrategy="all"*)

Sets params for Gradient Boosted Tree Classification.

New in version 1.4.0.

class pyspark.ml.classification.**GBTClassificationModel**(*java_model=None*)

Model fitted by GBTClassifier.

New in version 1.4.0.

evaluateEachIteration(*dataset*)

Method to compute error or loss for every iteration of gradient boosting.

Parameters **dataset** – Test dataset to evaluate model on, where dataset is an instance of pyspark.sql.DataFrame

New in version 2.4.0.

featureImportances

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie, Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also:

DecisionTreeClassificationModel.featureImportances

New in version 2.0.0.

trees

These have null parent Estimators.

New in version 2.0.0.

Type Trees in this ensemble. Warning

class pyspark.ml.classification.**RandomForestClassifier**(*featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', numTrees=20, featureSubsetStrategy='auto', seed=None, subsamplingRate=1.0*)

Random Forest learning algorithm for classification. It supports both binary and multiclass labels, as well as both continuous and categorical features.

```
>>> import numpy
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> rf = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="indexed
↪", seed=42)
>>> model = rf.fit(td)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 1.0, 1.0])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
```

(continues on next page)

(continued from previous page)

```
>>> numpy.argmax(result.probability)
0
>>> numpy.argmax(result.rawPrediction)
0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
↪"features"])
>>> model.transform(test1).head().prediction
1.0
>>> model.trees
[DecisionTreeClassificationModel (uid=...) of depth..., ␣
↪DecisionTreeClassificationModel...]
>>> rfc_path = temp_path + "/rfc"
>>> rf.save(rfc_path)
>>> rf2 = RandomForestClassifier.load(rfc_path)
>>> rf2.getNumTrees()
3
>>> model_path = temp_path + "/rfc_model"
>>> model.save(model_path)
>>> model2 = RandomForestClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

New in version 1.4.0.

setFeatureSubsetStrategy(*value*)

Sets the value of `featureSubsetStrategy`.

New in version 2.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *seed=None*, *impurity="gini"*, *numTrees=20*, *featureSubsetStrategy="auto"*, *subsamplingRate=1.0*)

Sets params for linear classification.

New in version 1.4.0.

class pyspark.ml.classification.**RandomForestClassificationModel**(*java_model=None*)

Model fitted by RandomForestClassifier.

New in version 1.4.0.

featureImportances

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie, Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also:

DecisionTreeClassificationModel.featureImportances

New in version 2.0.0.

trees

These have null parent Estimators.

New in version 2.0.0.

Type Trees in this ensemble. Warning

class pyspark.ml.classification.**NaiveBayes**(*featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', smoothing=1.0, modelType='multinomial', thresholds=None, weightCol=None*)

Naive Bayes Classifiers. It supports both Multinomial and Bernoulli NB. Multinomial NB can handle finitely supported discrete data. For example, by converting documents into TF-IDF vectors, it can be used for document classification. By making every vector a binary (0/1) data, it can also be used as Bernoulli NB. The input feature values must be nonnegative.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     Row(label=0.0, weight=0.1, features=Vectors.dense([0.0, 0.0])),
...     Row(label=0.0, weight=0.5, features=Vectors.dense([0.0, 1.0])),
...     Row(label=1.0, weight=1.0, features=Vectors.dense([1.0, 0.0]))])
>>> nb = NaiveBayes(smoothing=1.0, modelType="multinomial", weightCol=
↪"weight")
>>> model = nb.fit(df)
>>> model.pi
DenseVector([-0.81..., -0.58...])
>>> model.theta
DenseMatrix(2, 2, [-0.91..., -0.51..., -0.40..., -1.09...], 1)
>>> test0 = sc.parallelize([Row(features=Vectors.dense([1.0, 0.0]))]).
↪toDF()
>>> result = model.transform(test0).head()
>>> result.prediction
1.0
>>> result.probability
DenseVector([0.32..., 0.67...])
>>> result.rawPrediction
DenseVector([-1.72..., -0.99...])
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(2, [0], [1.
↪0]))]).toDF()
>>> model.transform(test1).head().prediction
1.0
>>> nb_path = temp_path + "/nb"
>>> nb.save(nb_path)
>>> nb2 = NaiveBayes.load(nb_path)
```

(continues on next page)

(continued from previous page)

```
>>> nb2.getSmoothing()
1.0
>>> model_path = temp_path + "/nb_model"
>>> model.save(model_path)
>>> model2 = NaiveBayesModel.load(model_path)
>>> model.pi == model2.pi
True
>>> model.theta == model2.theta
True
>>> nb = nb.setThresholds([0.01, 10.00])
>>> model3 = nb.fit(df)
>>> result = model3.transform(test0).head()
>>> result.prediction
0.0
```

New in version 1.5.0.

getModelType()

Gets the value of modelType or its default value.

New in version 1.5.0.

getSmoothing()

Gets the value of smoothing or its default value.

New in version 1.5.0.

setModelType(*value*)

Sets the value of `modelType`.

New in version 1.5.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *smoothing=1.0*, *modelType="multinomial"*, *thresholds=None*, *weightCol=None*)

Sets params for Naive Bayes.

New in version 1.5.0.

setSmoothing(*value*)

Sets the value of `smoothing`.

New in version 1.5.0.

class pyspark.ml.classification.**NaiveBayesModel**(*java_model=None*)

Model fitted by NaiveBayes.

New in version 1.5.0.

pi

log of class priors.

New in version 2.0.0.

theta

log of class conditional probabilities.

New in version 2.0.0.

class pyspark.ml.classification.**MultilayerPerceptronClassifier**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, tol=1e-06, seed=None, layers=None, blockSize=128, stepSize=0.03, solver='l-bfgs', initialWeights=None, probability-Col='probability', rawPredictionCol='rawPrediction'*)

Classifier trainer based on the Multilayer Perceptron. Each layer has sigmoid activation function, output layer has softmax. Number of inputs has to be equal to the size of feature vectors. Number of outputs has to be equal to the total number of labels.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (0.0, Vectors.dense([0.0, 0.0])),
...     (1.0, Vectors.dense([0.0, 1.0])),
...     (1.0, Vectors.dense([1.0, 0.0])),
...     (0.0, Vectors.dense([1.0, 1.0]))], ["label", "features"])
>>> mlp = MultilayerPerceptronClassifier(maxIter=100, layers=[2, 2, 2],␣
↪blockSize=1, seed=123)
>>> model = mlp.fit(df)
>>> model.layers
[2, 2, 2]
```

(continues on next page)

(continued from previous page)

```
>>> model.weights.size
12
>>> testDF = spark.createDataFrame([
...     (Vectors.dense([1.0, 0.0]),),
...     (Vectors.dense([0.0, 0.0]),)], ["features"])
>>> model.transform(testDF).select("features", "prediction").show()
+---------+----------+
| features|prediction|
+---------+----------+
|[1.0,0.0]|       1.0|
|[0.0,0.0]|       0.0|
+---------+----------+
...
>>> mlp_path = temp_path + "/mlp"
>>> mlp.save(mlp_path)
>>> mlp2 = MultilayerPerceptronClassifier.load(mlp_path)
>>> mlp2.getBlockSize()
1
>>> model_path = temp_path + "/mlp_model"
>>> model.save(model_path)
>>> model2 = MultilayerPerceptronClassificationModel.load(model_path)
>>> model.layers == model2.layers
True
>>> model.weights == model2.weights
True
>>> mlp2 = mlp2.setInitialWeights(list(range(0, 12)))
>>> model3 = mlp2.fit(df)
>>> model3.weights != model2.weights
True
>>> model3.layers == model.layers
True
```

New in version 1.6.0.

getBlockSize()

Gets the value of blockSize or its default value.

New in version 1.6.0.

getInitialWeights()

Gets the value of initialWeights or its default value.

New in version 2.0.0.

getLayers()

Gets the value of layers or its default value.

New in version 1.6.0.

getStepSize()

Gets the value of stepSize or its default value.

New in version 2.0.0.

setBlockSize(*value*)
Sets the value of `blockSize`.

New in version 1.6.0.

setInitialWeights(*value*)
Sets the value of `initialWeights`.

New in version 2.0.0.

setLayers(*value*)
Sets the value of `layers`.

New in version 1.6.0.

setParams(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, tol=1e-06, seed=None, layers=None, blockSize=128, stepSize=0.03, solver='l-bfgs', initialWeights=None, probabilityCol='probability', rawPredictionCol='rawPrediction'*)
setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, tol=1e-6, seed=None, layers=None, blockSize=128, stepSize=0.03, solver="l-bfgs", initialWeights=None, probabilityCol="probability", rawPredictionCol="rawPrediction"): Sets params for MultilayerPerceptronClassifier.

New in version 1.6.0.

setStepSize(*value*)
Sets the value of `stepSize`.

New in version 2.0.0.

class pyspark.ml.classification.**MultilayerPerceptronClassificationModel**(*java_model=No*
Model fitted by MultilayerPerceptronClassifier.

New in version 1.6.0.

layers
array of layer sizes including input and output layers.

New in version 1.6.0.

weights
the weights of layers.

New in version 2.0.0.

class pyspark.ml.classification.**OneVsRest**(*featuresCol='features', labelCol='label', predictionCol='prediction', classifier=None, weightCol=None, parallelism=1*)

Note: Experimental

Reduction of Multiclass Classification to Binary Classification. Performs reduction using one against all strategy. For a multiclass classification with k classes, train k models (one per class). Each example is scored against all k models and the model with highest score is picked to label the example.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> data_path = "data/mllib/sample_multiclass_classification_data.txt"
>>> df = spark.read.format("libsvm").load(data_path)
>>> lr = LogisticRegression(regParam=0.01)
>>> ovr = OneVsRest(classifier=lr)
>>> model = ovr.fit(df)
>>> model.models[0].coefficients
DenseVector([0.5..., -1.0..., 3.4..., 4.2...])
>>> model.models[1].coefficients
DenseVector([-2.1..., 3.1..., -2.6..., -2.3...])
>>> model.models[2].coefficients
DenseVector([0.3..., -3.4..., 1.0..., -1.1...])
>>> [x.intercept for x in model.models]
[-2.7..., -2.5..., -1.3...]
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, 0.0, 1.0, 1.
↪0))]).toDF()
>>> model.transform(test0).head().prediction
0.0
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(4, [0], [1.
↪0]))]).toDF()
>>> model.transform(test1).head().prediction
2.0
>>> test2 = sc.parallelize([Row(features=Vectors.dense(0.5, 0.4, 0.3, 0.
↪2))]).toDF()
>>> model.transform(test2).head().prediction
0.0
>>> model_path = temp_path + "/ovr_model"
>>> model.save(model_path)
>>> model2 = OneVsRestModel.load(model_path)
>>> model2.transform(test0).head().prediction
0.0
```

New in version 2.0.0.

copy (*extra=None*)

Creates a copy of this instance with a randomly generated uid and some extra params. This creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters **extra** – Extra parameters to copy to the new instance

Returns Copy of this instance

New in version 2.0.0.

setParams (*featuresCol='features', labelCol='label', predictionCol='prediction', classifier=None, weightCol=None, parallelism=1*)

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", classifier=None, weightCol=None, parallelism=1): Sets params for OneVsRest.

New in version 2.0.0.

class pyspark.ml.classification.**OneVsRestModel**(*models*)

Note: Experimental

Model fitted by OneVsRest. This stores the models resulting from training k binary classifiers: one for each class. Each example is scored against all k models, and the model with the highest score is picked to label the example.

New in version 2.0.0.

copy(*extra=None*)

Creates a copy of this instance with a randomly generated uid and some extra params. This creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters **extra** – Extra parameters to copy to the new instance

Returns Copy of this instance

New in version 2.0.0.

21.4 Clustering API

class pyspark.ml.clustering.**BisectingKMeans**(*featuresCol='features', predictionCol='prediction', maxIter=20, seed=None, k=4, minDivisibleClusterSize=1.0, distanceMeasure='euclidean'*)

A bisecting k-means algorithm based on the paper "A comparison of document clustering techniques" by Steinbach, Karypis, and Kumar, with modification to fit Spark. The algorithm starts from a single cluster that contains all points. Iteratively it finds divisible clusters on the bottom level and bisects each of them using k-means, until there are k leaf clusters in total or no leaf clusters are divisible. The bisecting steps of clusters on the same level are grouped together to increase parallelism. If bisecting all divisible clusters on the bottom level would result more than k leaf clusters, larger clusters get higher priority.

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.dense([0.0, 0.0]),), (Vectors.dense([1.0, 1.0]),),
...         (Vectors.dense([9.0, 8.0]),), (Vectors.dense([8.0, 9.0]),)]
>>> df = spark.createDataFrame(data, ["features"])
>>> bkm = BisectingKMeans(k=2, minDivisibleClusterSize=1.0)
>>> model = bkm.fit(df)
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> model.computeCost(df)
```

(continues on next page)

(continued from previous page)

```
2.000...
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
2
>>> summary.clusterSizes
[2, 2]
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[0].prediction == rows[1].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> bkm_path = temp_path + "/bkm"
>>> bkm.save(bkm_path)
>>> bkm2 = BisectingKMeans.load(bkm_path)
>>> bkm2.getK()
2
>>> bkm2.getDistanceMeasure()
'euclidean'
>>> model_path = temp_path + "/bkm_model"
>>> model.save(model_path)
>>> model2 = BisectingKMeansModel.load(model_path)
>>> model2.hasSummary
False
>>> model.clusterCenters()[0] == model2.clusterCenters()[0]
array([ True,  True], dtype=bool)
>>> model.clusterCenters()[1] == model2.clusterCenters()[1]
array([ True,  True], dtype=bool)
```

New in version 2.0.0.

getDistanceMeasure()

Gets the value of $distanceMeasure$ or its default value.

New in version 2.4.0.

getK()

Gets the value of k or its default value.

New in version 2.0.0.

getMinDivisibleClusterSize()

Gets the value of $minDivisibleClusterSize$ or its default value.

New in version 2.0.0.

setDistanceMeasure(*value*)

Sets the value of `distanceMeasure`.

New in version 2.4.0.

setK(*value*)

Sets the value of `k`.

New in version 2.0.0.

setMinDivisibleClusterSize(*value*)
Sets the value of `minDivisibleClusterSize`.

New in version 2.0.0.

setParams(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *maxIter=20*, *seed=None*, *k=4*, *minDivisibleClusterSize=1.0*, *distanceMeasure="euclidean"*)
Sets params for BisectingKMeans.

New in version 2.0.0.

class pyspark.ml.clustering.**BisectingKMeansModel**(*java_model=None*)
Model fitted by BisectingKMeans.

New in version 2.0.0.

clusterCenters()
Get the cluster centers, represented as a list of NumPy arrays.

New in version 2.0.0.

computeCost(*dataset*)
Computes the sum of squared distances between the input points and their corresponding cluster centers.

New in version 2.0.0.

hasSummary
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

summary
Gets summary (e.g. cluster assignments, cluster sizes) of the model trained on the training set. An exception is thrown if no summary exists.

New in version 2.1.0.

class pyspark.ml.clustering.**BisectingKMeansSummary**(*java_obj=None*)

Note: Experimental

Bisecting KMeans clustering results for a given model.

New in version 2.1.0.

class pyspark.ml.clustering.**KMeans**(*featuresCol='features'*, *predictionCol='prediction'*, *k=2*, *initMode='k-means||'*, *initSteps=2*, *tol=0.0001*, *maxIter=20*, *seed=None*, *distanceMeasure='euclidean'*)
K-means clustering with a k-means++ like initialization mode (the k-means|| algorithm by Bahmani

et al).

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.dense([0.0, 0.0]),), (Vectors.dense([1.0, 1.0]),),
...         (Vectors.dense([9.0, 8.0]),), (Vectors.dense([8.0, 9.0]),)]
>>> df = spark.createDataFrame(data, ["features"])
>>> kmeans = KMeans(k=2, seed=1)
>>> model = kmeans.fit(df)
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> model.computeCost(df)
2.000...
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[0].prediction == rows[1].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
2
>>> summary.clusterSizes
[2, 2]
>>> summary.trainingCost
2.000...
>>> kmeans_path = temp_path + "/kmeans"
>>> kmeans.save(kmeans_path)
>>> kmeans2 = KMeans.load(kmeans_path)
>>> kmeans2.getK()
2
>>> model_path = temp_path + "/kmeans_model"
>>> model.save(model_path)
>>> model2 = KMeansModel.load(model_path)
>>> model2.hasSummary
False
>>> model.clusterCenters()[0] == model2.clusterCenters()[0]
array([ True,  True], dtype=bool)
>>> model.clusterCenters()[1] == model2.clusterCenters()[1]
array([ True,  True], dtype=bool)
```

New in version 1.5.0.

getDistanceMeasure()

Gets the value of $distanceMeasure$

New in version 2.4.0.

getInitMode()

Gets the value of $initMode$

New in version 1.5.0.

getInitSteps()
Gets the value of $initSteps$

New in version 1.5.0.

getK()
Gets the value of k

New in version 1.5.0.

setDistanceMeasure(*value*)
Sets the value of `distanceMeasure`.

New in version 2.4.0.

setInitMode(*value*)
Sets the value of `initMode`.

New in version 1.5.0.

setInitSteps(*value*)
Sets the value of `initSteps`.

New in version 1.5.0.

setK(*value*)
Sets the value of `k`.

New in version 1.5.0.

setParams(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *initMode="k-means||"*, *initSteps=2*, *tol=1e-4*, *maxIter=20*, *seed=None*, *distanceMeasure="euclidean"*)
Sets params for KMeans.

New in version 1.5.0.

class pyspark.ml.clustering.**KMeansModel**(*java_model=None*)
Model fitted by KMeans.

New in version 1.5.0.

clusterCenters()
Get the cluster centers, represented as a list of NumPy arrays.

New in version 1.5.0.

computeCost(*dataset*)
Return the K-means cost (sum of squared distances of points to their nearest center) for this model on the given data.

..note:: Deprecated in 2.4.0. It will be removed in 3.0.0. Use ClusteringEvaluator instead.
You can also get the cost on the training dataset in the summary.

New in version 2.0.0.

hasSummary
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

summary

Gets summary (e.g. cluster assignments, cluster sizes) of the model trained on the training set. An exception is thrown if no summary exists.

New in version 2.1.0.

class pyspark.ml.clustering.**GaussianMixture**(*featuresCol='features', predictionCol='prediction', k=2, probabilityCol='probability', tol=0.01, maxIter=100, seed=None*)

GaussianMixture clustering. This class performs expectation maximization for multivariate Gaussian Mixture Models (GMMs). A GMM represents a composite distribution of independent Gaussian distributions with associated "mixing" weights specifying each's contribution to the composite.

Given a set of sample points, this class will maximize the log-likelihood for a mixture of k Gaussians, iterating until the log-likelihood changes by less than convergenceTol, or until it has reached the max number of iterations. While this process is generally guaranteed to converge, it is not guaranteed to find a global optimum.

Note: For high-dimensional data (with many features), this algorithm may perform poorly. This is due to high-dimensional data (a) making it difficult to cluster at all (based on statistical/theoretical arguments) and (b) numerical issues with Gaussian distributions.

```
>>> from pyspark.ml.linalg import Vectors
```

```
>>> data = [(Vectors.dense([-0.1, -0.05 ]),),
...         (Vectors.dense([-0.01, -0.1]),),
...         (Vectors.dense([0.9, 0.8]),),
...         (Vectors.dense([0.75, 0.935]),),
...         (Vectors.dense([-0.83, -0.68]),),
...         (Vectors.dense([-0.91, -0.76]),)]
>>> df = spark.createDataFrame(data, ["features"])
>>> gm = GaussianMixture(k=3, tol=0.0001,
...                      maxIter=10, seed=10)
>>> model = gm.fit(df)
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
3
>>> summary.clusterSizes
[2, 2, 2]
>>> summary.logLikelihood
8.14636...
>>> weights = model.weights
>>> len(weights)
3
>>> model.gaussiansDF.select("mean").head()
```

(continues on next page)

(continued from previous page)

```
Row(mean=DenseVector([0.825, 0.8675]))
>>> model.gaussiansDF.select("cov").head()
Row(cov=DenseMatrix(2, 2, [0.0056, -0.0051, -0.0051, 0.0046], False))
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[4].prediction == rows[5].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> gmm_path = temp_path + "/gmm"
>>> gm.save(gmm_path)
>>> gm2 = GaussianMixture.load(gmm_path)
>>> gm2.getK()
3
>>> model_path = temp_path + "/gmm_model"
>>> model.save(model_path)
>>> model2 = GaussianMixtureModel.load(model_path)
>>> model2.hasSummary
False
>>> model2.weights == model.weights
True
>>> model2.gaussiansDF.select("mean").head()
Row(mean=DenseVector([0.825, 0.8675]))
>>> model2.gaussiansDF.select("cov").head()
Row(cov=DenseMatrix(2, 2, [0.0056, -0.0051, -0.0051, 0.0046], False))
```

New in version 2.0.0.

getK()

Gets the value of k

New in version 2.0.0.

setK(*value*)

Sets the value of `k`.

New in version 2.0.0.

setParams(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *probabilityCol="probability"*, *tol=0.01*, *maxIter=100*, *seed=None*)

Sets params for GaussianMixture.

New in version 2.0.0.

class pyspark.ml.clustering.**GaussianMixtureModel**(*java_model=None*)

Model fitted by GaussianMixture.

New in version 2.0.0.

gaussiansDF

Retrieve Gaussian distributions as a DataFrame. Each row represents a Gaussian Distribution. The DataFrame has two columns: mean (Vector) and cov (Matrix).

New in version 2.0.0.

hasSummary
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

summary
Gets summary (e.g. cluster assignments, cluster sizes) of the model trained on the training set. An exception is thrown if no summary exists.

New in version 2.1.0.

weights
Weight for each Gaussian distribution in the mixture. This is a multinomial probability distribution over the k Gaussians, where weights[i] is the weight for Gaussian i, and weights sum to 1.

New in version 2.0.0.

class pyspark.ml.clustering.**GaussianMixtureSummary**(*java_obj=None*)

Note: Experimental

Gaussian mixture clustering results for a given model.

New in version 2.1.0.

logLikelihood
Total log-likelihood for this model on the given data.

New in version 2.2.0.

probability
DataFrame of probabilities of each cluster for each training data point.

New in version 2.1.0.

probabilityCol
Name for column of predicted probability of each cluster in $predictions$.

New in version 2.1.0.

class pyspark.ml.clustering.**LDA**(*featuresCol='features', maxIter=20, seed=None, checkpointInterval=10, k=10, optimizer='online', learningOffset=1024.0, learningDecay=0.51, subsamplingRate=0.05, optimizeDocConcentration=True, docConcentration=None, topicConcentration=None, topicDistributionCol='topicDistribution', keepLastCheckpoint=True*)

Latent Dirichlet Allocation (LDA), a topic model designed for text documents.

Terminology:

- "term" = "word": an element of the vocabulary

- "token": instance of a term appearing in a document
- "topic": multinomial distribution over terms representing some concept
- "document": one piece of text, corresponding to one row in the input data

Original LDA paper (journal version): Blei, Ng, and Jordan. "Latent Dirichlet Allocation." JMLR, 2003.

Input data (featuresCol): LDA is given a collection of documents as input data, via the featuresCol parameter. Each document is specified as a `Vector` of length vocabSize, where each entry is the count for the corresponding term (word) in the document. Feature transformers such as `pyspark.ml.feature.Tokenizer` and `pyspark.ml.feature.CountVectorizer` can be useful for converting text to word count vectors.

```
>>> from pyspark.ml.linalg import Vectors, SparseVector
>>> from pyspark.ml.clustering import LDA
>>> df = spark.createDataFrame([[1, Vectors.dense([0.0, 1.0])],
...      [2, SparseVector(2, {0: 1.0})],], ["id", "features"])
>>> lda = LDA(k=2, seed=1, optimizer="em")
>>> model = lda.fit(df)
>>> model.isDistributed()
True
>>> localModel = model.toLocal()
>>> localModel.isDistributed()
False
>>> model.vocabSize()
2
>>> model.describeTopics().show()
+-----+-----------+--------------------+
|topic|termIndices|         termWeights|
+-----+-----------+--------------------+
|    0|     [1, 0]|[0.50401530077160...|
|    1|     [0, 1]|[0.50401530077160...|
+-----+-----------+--------------------+
...
>>> model.topicsMatrix()
DenseMatrix(2, 2, [0.496, 0.504, 0.504, 0.496], 0)
>>> lda_path = temp_path + "/lda"
>>> lda.save(lda_path)
>>> sameLDA = LDA.load(lda_path)
>>> distributed_model_path = temp_path + "/lda_distributed_model"
>>> model.save(distributed_model_path)
>>> sameModel = DistributedLDAModel.load(distributed_model_path)
>>> local_model_path = temp_path + "/lda_local_model"
>>> localModel.save(local_model_path)
>>> sameLocalModel = LocalLDAModel.load(local_model_path)
```

New in version 2.0.0.

getDocConcentration()

Gets the value of `docConcentration` or its default value.

New in version 2.0.0.

getK()
Gets the value of `k` or its default value.

New in version 2.0.0.

getKeepLastCheckpoint()
Gets the value of `keepLastCheckpoint` or its default value.

New in version 2.0.0.

getLearningDecay()
Gets the value of `learningDecay` or its default value.

New in version 2.0.0.

getLearningOffset()
Gets the value of `learningOffset` or its default value.

New in version 2.0.0.

getOptimizeDocConcentration()
Gets the value of `optimizeDocConcentration` or its default value.

New in version 2.0.0.

getOptimizer()
Gets the value of `optimizer` or its default value.

New in version 2.0.0.

getSubsamplingRate()
Gets the value of `subsamplingRate` or its default value.

New in version 2.0.0.

getTopicConcentration()
Gets the value of `topicConcentration` or its default value.

New in version 2.0.0.

getTopicDistributionCol()
Gets the value of `topicDistributionCol` or its default value.

New in version 2.0.0.

setDocConcentration(*value*)
Sets the value of `docConcentration`.

```
>>> algo = LDA().setDocConcentration([0.1, 0.2])
>>> algo.getDocConcentration()
[0.1..., 0.2...]
```

New in version 2.0.0.

setK(*value*)
Sets the value of `k`.

```
>>> algo = LDA().setK(10)
>>> algo.getK()
10
```

New in version 2.0.0.

setKeepLastCheckpoint(*value*)

Sets the value of `keepLastCheckpoint`.

```
>>> algo = LDA().setKeepLastCheckpoint(False)
>>> algo.getKeepLastCheckpoint()
False
```

New in version 2.0.0.

setLearningDecay(*value*)

Sets the value of `learningDecay`.

```
>>> algo = LDA().setLearningDecay(0.1)
>>> algo.getLearningDecay()
0.1...
```

New in version 2.0.0.

setLearningOffset(*value*)

Sets the value of `learningOffset`.

```
>>> algo = LDA().setLearningOffset(100)
>>> algo.getLearningOffset()
100.0
```

New in version 2.0.0.

setOptimizeDocConcentration(*value*)

Sets the value of `optimizeDocConcentration`.

```
>>> algo = LDA().setOptimizeDocConcentration(True)
>>> algo.getOptimizeDocConcentration()
True
```

New in version 2.0.0.

setOptimizer(*value*)

Sets the value of `optimizer`. Currently only support 'em' and 'online'.

```
>>> algo = LDA().setOptimizer("em")
>>> algo.getOptimizer()
'em'
```

New in version 2.0.0.

setParams(*self*, *featuresCol="features"*, *maxIter=20*, *seed=None*, *checkpointInterval=10*, *k=10*, *optimizer="online"*, *learningOffset=1024.0*, *learningDecay=0.51*, *subsamplingRate=0.05*, *optimizeDocConcentration=True*, *docConcentration=None*, *topicConcentration=None*, *topicDistributionCol="topicDistribution"*, *keepLastCheckpoint=True*)

Sets params for LDA.

New in version 2.0.0.

setSubsamplingRate(*value*)

Sets the value of `subsamplingRate`.

```
>>> algo = LDA().setSubsamplingRate(0.1)
>>> algo.getSubsamplingRate()
0.1...
```

New in version 2.0.0.

setTopicConcentration(*value*)

Sets the value of `topicConcentration`.

```
>>> algo = LDA().setTopicConcentration(0.5)
>>> algo.getTopicConcentration()
0.5...
```

New in version 2.0.0.

setTopicDistributionCol(*value*)

Sets the value of `topicDistributionCol`.

```
>>> algo = LDA().setTopicDistributionCol("topicDistributionCol")
>>> algo.getTopicDistributionCol()
'topicDistributionCol'
```

New in version 2.0.0.

class pyspark.ml.clustering.**LDAModel**(*java_model=None*)

Latent Dirichlet Allocation (LDA) model. This abstraction permits for different underlying representations, including local and distributed data structures.

New in version 2.0.0.

describeTopics(*maxTermsPerTopic=10*)

Return the topics described by their top-weighted terms.

New in version 2.0.0.

estimatedDocConcentration()

Value for `LDA.docConcentration` estimated from data. If Online LDA was used and `LDA.optimizeDocConcentration` was set to false, then this returns the fixed (given) value for the `LDA.docConcentration` parameter.

New in version 2.0.0.

isDistributed()

Indicates whether this instance is of type DistributedLDAModel

New in version 2.0.0.

logLikelihood(*dataset*)

Calculates a lower bound on the log likelihood of the entire corpus. See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of `DistributedLDAModel` (produced when `optimizer` is set to "em"), this involves collecting a large `topicsMatrix()` to the driver. This implementation may be changed in the future.

New in version 2.0.0.

logPerplexity(*dataset*)

Calculate an upper bound on perplexity. (Lower is better.) See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of `DistributedLDAModel` (produced when `optimizer` is set to "em"), this involves collecting a large `topicsMatrix()` to the driver. This implementation may be changed in the future.

New in version 2.0.0.

topicsMatrix()

Inferred topics, where each topic is represented by a distribution over terms. This is a matrix of size vocabSize x k, where each column is a topic. No guarantees are given about the ordering of the topics.

WARNING: If this model is actually a `DistributedLDAModel` instance produced by the Expectation-Maximization ("em") *optimizer*, then this method could involve collecting a large amount of data to the driver (on the order of vocabSize x k).

New in version 2.0.0.

vocabSize()

Vocabulary size (number of terms or words in the vocabulary)

New in version 2.0.0.

class pyspark.ml.clustering.**LocalLDAModel**(*java_model=None*)

Local (non-distributed) model fitted by `LDA`. This model stores the inferred topics only; it does not store info about the training dataset.

New in version 2.0.0.

class pyspark.ml.clustering.**DistributedLDAModel**(*java_model=None*)

Distributed model fitted by `LDA`. This type of model is currently only produced by Expectation-Maximization (EM).

This model stores the inferred topics, the full training dataset, and the topic distribution for each training document.

New in version 2.0.0.

getCheckpointFiles()

If using checkpointing and `LDA.keepLastCheckpoint` is set to true, then there may be saved checkpoint files. This method is provided so that users can manage those files.

Note: Removing the checkpoints can cause failures if a partition is lost and is needed by certain *DistributedLDAModel* methods. Reference counting will clean up the checkpoints when this model and derivative data go out of scope.

:return List of checkpoint files from training

New in version 2.0.0.

logPrior()

Log probability of the current parameter estimate: log P(topics, topic distributions for docs | alpha, eta)

New in version 2.0.0.

toLocal()

Convert this distributed model to a local representation. This discards info about the training dataset.

WARNING: This involves collecting a large `topicsMatrix()` to the driver.

New in version 2.0.0.

trainingLogLikelihood()

Log likelihood of the observed tokens in the training set, given the current parameter estimates: log P(docs | topics, topic distributions for docs, Dirichlet hyperparameters)

Notes:

- This excludes the prior; for that, use *logPrior()*.
- Even with *logPrior()*, this is NOT the same as the data log likelihood given the hyperparameters.
- This is computed from the topic distributions computed during training. If you call `logLikelihood()` on the same training dataset, the topic distributions will be computed again, possibly giving different results.

New in version 2.0.0.

class pyspark.ml.clustering.**PowerIterationClustering**(*k=2, maxIter=20, initMode='random', srcCol='src', dstCol='dst', weightCol=None*)

Note: Experimental

Power Iteration Clustering (PIC), a scalable graph clustering algorithm developed by Lin and Cohen. From the abstract: PIC finds a very low-dimensional embedding of a dataset using truncated power iteration on a normalized pair-wise similarity matrix of the data.

This class is not yet an Estimator/Transformer, use `assignClusters()` method to run the PowerIterationClustering algorithm.

See also:

Wikipedia on Spectral clustering

```
>>> data = [(1, 0, 0.5),
...         (2, 0, 0.5), (2, 1, 0.7),
...         (3, 0, 0.5), (3, 1, 0.7), (3, 2, 0.9),
...         (4, 0, 0.5), (4, 1, 0.7), (4, 2, 0.9), (4, 3, 1.1),
...         (5, 0, 0.5), (5, 1, 0.7), (5, 2, 0.9), (5, 3, 1.1), (5, 4, 1.
↪3)]
>>> df = spark.createDataFrame(data).toDF("src", "dst", "weight")
>>> pic = PowerIterationClustering(k=2, maxIter=40, weightCol="weight")
>>> assignments = pic.assignClusters(df)
>>> assignments.sort(assignments.id).show(truncate=False)
+---+-------+
|id |cluster|
+---+-------+
|0  |1      |
|1  |1      |
|2  |1      |
|3  |1      |
|4  |1      |
|5  |0      |
+---+-------+
...
>>> pic_path = temp_path + "/pic"
>>> pic.save(pic_path)
>>> pic2 = PowerIterationClustering.load(pic_path)
>>> pic2.getK()
2
>>> pic2.getMaxIter()
40
```

New in version 2.4.0.

assignClusters(*dataset*)

Run the PIC algorithm and returns a cluster assignment for each input vertex.

Parameters **dataset** – A dataset with columns src, dst, weight representing the affinity matrix, which is the matrix A in the PIC paper. Suppose the src column value is i, the dst column value is j, the weight column value is similarity s,,ij,, which must be nonnegative. This is a symmetric matrix and hence s,,ij,, = s,,ji,,. For any (i, j) with nonzero similarity, there should be either (i, j, s,,ij,,) or (j, i, s,,ji,,) in the input. Rows with i = j are ignored, because we assume s,,ij,, = 0.0.

Returns A dataset that contains columns of vertex id and the corresponding cluster for the id. The schema of it will be: - id: Long - cluster: Int

New in version 2.4.0.

New in version 2.4.0.

getDstCol()
Gets the value of `dstCol` or its default value.

New in version 2.4.0.

getInitMode()
Gets the value of `initMode` or its default value.

New in version 2.4.0.

getK()
Gets the value of `k` or its default value.

New in version 2.4.0.

getSrcCol()
Gets the value of `srcCol` or its default value.

New in version 2.4.0.

setDstCol(*value*)
Sets the value of `dstCol`.

New in version 2.4.0.

setInitMode(*value*)
Sets the value of `initMode`.

New in version 2.4.0.

setK(*value*)
Sets the value of `k`.

New in version 2.4.0.

setParams(*self*, *k=2*, *maxIter=20*, *initMode="random"*, *srcCol="src"*, *dstCol="dst"*, *weightCol=None*)
Sets params for PowerIterationClustering.

New in version 2.4.0.

setSrcCol(*value*)
Sets the value of `srcCol`.

New in version 2.4.0.

21.5 Recommendation API

class pyspark.ml.recommendation.**ALS**(*rank=10, maxIter=10, regParam=0.1, numUserBlocks=10, numItemBlocks=10, implicitPrefs=False, alpha=1.0, userCol='user', itemCol='item', seed=None, ratingCol='rating', nonnegative=False, checkpointInterval=10, intermediateStorageLevel='MEMORY_AND_DISK', finalStorageLevel='MEMORY_AND_DISK', coldStartStrategy='nan'*)

Alternating Least Squares (ALS) matrix factorization.

ALS attempts to estimate the ratings matrix R as the product of two lower-rank matrices, X and Y, i.e. $X * Yt = R$. Typically these approximations are called 'factor' matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix.

This is a blocked implementation of the ALS factorization algorithm that groups the two sets of factors (referred to as "users" and "products") into blocks and reduces communication by only sending one copy of each user vector to each product block on each iteration, and only for the product blocks that need that user's feature vector. This is achieved by pre-computing some information about the ratings matrix to determine the "out-links" of each user (which blocks of products it will contribute to) and "in-link" information for each product (which of the feature vectors it receives from each user block it will depend on). This allows us to send only an array of feature vectors between each user block and product block, and have the product block find the users' ratings and update the products based on these messages.

For implicit preference data, the algorithm used is based on "Collaborative Filtering for Implicit Feedback Datasets",, adapted for the blocked approach used here.

Essentially instead of finding the low-rank approximations to the rating matrix R, this finds the approximations for a preference matrix P where the elements of P are 1 if r > 0 and 0 if r <= 0. The ratings then act as 'confidence' values related to strength of indicated user preferences rather than explicit ratings given to items.

```
>>> df = spark.createDataFrame(
...     [(0, 0, 4.0), (0, 1, 2.0), (1, 1, 3.0), (1, 2, 4.0), (2, 1, 1.0),
↪ (2, 2, 5.0)],
...     ["user", "item", "rating"])
>>> als = ALS(rank=10, maxIter=5, seed=0)
>>> model = als.fit(df)
>>> model.rank
10
>>> model.userFactors.orderBy("id").collect()
[Row(id=0, features=[...]), Row(id=1, ...), Row(id=2, ...)]
>>> test = spark.createDataFrame([(0, 2), (1, 0), (2, 0)], ["user", "item
↪"])
>>> predictions = sorted(model.transform(test).collect(), key=lambda r:␣
↪r[0])
```

(continues on next page)

(continued from previous page)

```
>>> predictions[0]
Row(user=0, item=2, prediction=-0.13807615637779236)
>>> predictions[1]
Row(user=1, item=0, prediction=2.6258413791656494)
>>> predictions[2]
Row(user=2, item=0, prediction=-1.5018409490585327)
>>> user_recs = model.recommendForAllUsers(3)
>>> user_recs.where(user_recs.user == 0)        .select("recommendations.
↪item", "recommendations.rating").collect()
[Row(item=[0, 1, 2], rating=[3.910..., 1.992..., -0.138...])]
>>> item_recs = model.recommendForAllItems(3)
>>> item_recs.where(item_recs.item == 2)        .select("recommendations.
↪user", "recommendations.rating").collect()
[Row(user=[2, 1, 0], rating=[4.901..., 3.981..., -0.138...])]
>>> user_subset = df.where(df.user == 2)
>>> user_subset_recs = model.recommendForUserSubset(user_subset, 3)
>>> user_subset_recs.select("recommendations.item", "recommendations.
↪rating").first()
Row(item=[2, 1, 0], rating=[4.901..., 1.056..., -1.501...])
>>> item_subset = df.where(df.item == 0)
>>> item_subset_recs = model.recommendForItemSubset(item_subset, 3)
>>> item_subset_recs.select("recommendations.user", "recommendations.
↪rating").first()
Row(user=[0, 1, 2], rating=[3.910..., 2.625..., -1.501...])
>>> als_path = temp_path + "/als"
>>> als.save(als_path)
>>> als2 = ALS.load(als_path)
>>> als.getMaxIter()
5
>>> model_path = temp_path + "/als_model"
>>> model.save(model_path)
>>> model2 = ALSModel.load(model_path)
>>> model.rank == model2.rank
True
>>> sorted(model.userFactors.collect()) == sorted(model2.userFactors.
↪collect())
True
>>> sorted(model.itemFactors.collect()) == sorted(model2.itemFactors.
↪collect())
True
```

New in version 1.4.0.

getAlpha()

Gets the value of alpha or its default value.

New in version 1.4.0.

getColdStartStrategy()

Gets the value of coldStartStrategy or its default value.

New in version 2.2.0.

getFinalStorageLevel()
Gets the value of finalStorageLevel or its default value.

New in version 2.0.0.

getImplicitPrefs()
Gets the value of implicitPrefs or its default value.

New in version 1.4.0.

getIntermediateStorageLevel()
Gets the value of intermediateStorageLevel or its default value.

New in version 2.0.0.

getItemCol()
Gets the value of itemCol or its default value.

New in version 1.4.0.

getNonnegative()
Gets the value of nonnegative or its default value.

New in version 1.4.0.

getNumItemBlocks()
Gets the value of numItemBlocks or its default value.

New in version 1.4.0.

getNumUserBlocks()
Gets the value of numUserBlocks or its default value.

New in version 1.4.0.

getRank()
Gets the value of rank or its default value.

New in version 1.4.0.

getRatingCol()
Gets the value of ratingCol or its default value.

New in version 1.4.0.

getUserCol()
Gets the value of userCol or its default value.

New in version 1.4.0.

setAlpha(*value*)
Sets the value of `alpha`.

New in version 1.4.0.

setColdStartStrategy(*value*)
Sets the value of `coldStartStrategy`.

New in version 2.2.0.

setFinalStorageLevel(*value*)
Sets the value of `finalStorageLevel`.

New in version 2.0.0.

setImplicitPrefs(*value*)
Sets the value of `implicitPrefs`.

New in version 1.4.0.

setIntermediateStorageLevel(*value*)
Sets the value of `intermediateStorageLevel`.

New in version 2.0.0.

setItemCol(*value*)
Sets the value of `itemCol`.

New in version 1.4.0.

setNonnegative(*value*)
Sets the value of `nonnegative`.

New in version 1.4.0.

setNumBlocks(*value*)
Sets both `numUserBlocks` and `numItemBlocks` to the specific value.

New in version 1.4.0.

setNumItemBlocks(*value*)
Sets the value of `numItemBlocks`.

New in version 1.4.0.

setNumUserBlocks(*value*)
Sets the value of `numUserBlocks`.

New in version 1.4.0.

setParams(*self*, *rank=10*, *maxIter=10*, *regParam=0.1*, *numUserBlocks=10*, *numItemBlocks=10*, *implicitPrefs=False*, *alpha=1.0*, *userCol="user"*, *itemCol="item"*, *seed=None*, *ratingCol="rating"*, *nonnegative=False*, *checkpointInterval=10*, *intermediateStorageLevel="MEMORY_AND_DISK"*, *finalStorageLevel="MEMORY_AND_DISK"*, *coldStartStrategy="nan"*)
Sets params for ALS.

New in version 1.4.0.

setRank(*value*)
Sets the value of `rank`.

New in version 1.4.0.

setRatingCol(*value*)
Sets the value of `ratingCol`.

New in version 1.4.0.

setUserCol(*value*)
Sets the value of `userCol`.

New in version 1.4.0.

class `pyspark.ml.recommendation.`**ALSModel**(*java_model=None*)
Model fitted by ALS.

New in version 1.4.0.

itemFactors
id and $features$

New in version 1.4.0.

Type a DataFrame that stores item factors in two columns

rank
rank of the matrix factorization model

New in version 1.4.0.

recommendForAllItems(*numUsers*)
Returns top $numUsers$ users recommended for each item, for all items.

Parameters **numUsers** – max number of recommendations for each item

Returns a DataFrame of (itemCol, recommendations), where recommendations are stored as an array of (userCol, rating) Rows.

New in version 2.2.0.

recommendForAllUsers(*numItems*)
Returns top $numItems$ items recommended for each user, for all users.

Parameters **numItems** – max number of recommendations for each user

Returns a DataFrame of (userCol, recommendations), where recommendations are stored as an array of (itemCol, rating) Rows.

New in version 2.2.0.

recommendForItemSubset(*dataset*, *numUsers*)
Returns top $numUsers$ users recommended for each item id in the input data set. Note that if there are duplicate ids in the input dataset, only one set of recommendations per unique id will be returned.

Parameters

- **dataset** – a Dataset containing a column of item ids. The column name must match $itemCol$.
- **numUsers** – max number of recommendations for each item

Returns a DataFrame of (itemCol, recommendations), where recommendations are stored as an array of (userCol, rating) Rows.

New in version 2.3.0.

recommendForUserSubset(*dataset*, *numItems*)

Returns top $numItems$ items recommended for each user id in the input data set. Note that if there are duplicate ids in the input dataset, only one set of recommendations per unique id will be returned.

Parameters

- **dataset** – a Dataset containing a column of user ids. The column name must match $userCol$.
- **numItems** – max number of recommendations for each user

Returns a DataFrame of (userCol, recommendations), where recommendations are stored as an array of (itemCol, rating) Rows.

New in version 2.3.0.

userFactors

id and $features$

New in version 1.4.0.

Type a DataFrame that stores user factors in two columns

21.6 Pipeline API

class pyspark.ml.pipeline.**Pipeline**(*stages=None*)

A simple pipeline, which acts as an estimator. A Pipeline consists of a sequence of stages, each of which is either an `Estimator` or a `Transformer`. When `Pipeline.fit()` is called, the stages are executed in order. If a stage is an `Estimator`, its `Estimator.fit()` method will be called on the input dataset to fit a model. Then the model, which is a transformer, will be used to transform the dataset as the input to the next stage. If a stage is a `Transformer`, its `Transformer.transform()` method will be called to produce the dataset for the next stage. The fitted model from a `Pipeline` is a `PipelineModel`, which consists of fitted models and transformers, corresponding to the pipeline stages. If stages is an empty list, the pipeline acts as an identity transformer.

New in version 1.3.0.

copy(*extra=None*)

Creates a copy of this instance.

Parameters **extra** – extra parameters

Returns new instance

New in version 1.4.0.

getStages()

Get pipeline stages.

New in version 1.3.0.

classmethod read()

Returns an MLReader instance for this class.

New in version 2.0.0.

setParams(*self*, *stages=None*)
Sets params for Pipeline.

New in version 1.3.0.

setStages(*value*)
Set pipeline stages.

Parameters **value** – a list of transformers or estimators

Returns the pipeline instance

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

New in version 2.0.0.

class pyspark.ml.pipeline.**PipelineModel**(*stages*)
Represents a compiled pipeline with transformers and fitted models.

New in version 1.3.0.

copy(*extra=None*)
Creates a copy of this instance.

Parameters **extra** – extra parameters

Returns new instance

New in version 1.4.0.

classmethod read()
Returns an MLReader instance for this class.

New in version 2.0.0.

write()
Returns an MLWriter instance for this ML instance.

New in version 2.0.0.

class pyspark.ml.pipeline.**PipelineModelReader**(*cls*)
(Private) Specialization of `MLReader` for *`PipelineModel`* types

load(*path*)
Load the ML instance from the input path.

class pyspark.ml.pipeline.**PipelineModelWriter**(*instance*)
(Private) Specialization of `MLWriter` for *`PipelineModel`* types

saveImpl(*path*)
save() handles overwriting and then calls this method. Subclasses should override this method to implement the actual saving of the instance.

class pyspark.ml.pipeline.**PipelineReader**(*cls*)

(Private) Specialization of `MLReader` for `Pipeline` types

load(*path*)

Load the ML instance from the input path.

class pyspark.ml.pipeline.**PipelineSharedReadWrite**

Note: DeveloperApi

Functions for `MLReader` and `MLWriter` shared between `Pipeline` and `PipelineModel`

New in version 2.3.0.

static getStagePath(*stageUid*, *stageIdx*, *numStages*, *stagesDir*)

Get path for saving the given stage.

static load(*metadata*, *sc*, *path*)

Load metadata and stages for a `Pipeline` or `PipelineModel`

Returns (UID, list of stages)

static saveImpl(*instance*, *stages*, *sc*, *path*)

Save metadata and stages for a `Pipeline` or `PipelineModel` - save metadata to path/metadata - save stages to stages/IDX_UID

static validateStages(*stages*)

Check that all stages are Writable

class pyspark.ml.pipeline.**PipelineWriter**(*instance*)

(Private) Specialization of `MLWriter` for `Pipeline` types

saveImpl(*path*)

save() handles overwriting and then calls this method. Subclasses should override this method to implement the actual saving of the instance.

21.7 Tuning API

class pyspark.ml.tuning.**ParamGridBuilder**

Builder for a param grid used in grid search-based model selection.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression()
>>> output = ParamGridBuilder() \
...     .baseOn({lr.labelCol: 'l'}) \
...     .baseOn([lr.predictionCol, 'p']) \
...     .addGrid(lr.regParam, [1.0, 2.0]) \
...     .addGrid(lr.maxIter, [1, 5]) \
...     .build()
>>> expected = [
```

(continues on next page)

(continued from previous page)

```
...     {lr.regParam: 1.0, lr.maxIter: 1, lr.labelCol: 'l', lr.
↪predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 1, lr.labelCol: 'l', lr.
↪predictionCol: 'p'},
...     {lr.regParam: 1.0, lr.maxIter: 5, lr.labelCol: 'l', lr.
↪predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 5, lr.labelCol: 'l', lr.
↪predictionCol: 'p'}]
>>> len(output) == len(expected)
True
>>> all([m in expected for m in output])
True
```

New in version 1.4.0.

addGrid(*param*, *values*)

Sets the given parameters in this grid to fixed values.

New in version 1.4.0.

baseOn(**args*)

Sets the given parameters in this grid to fixed values. Accepts either a parameter dictionary or a list of (parameter, value) pairs.

New in version 1.4.0.

build()

Builds and returns all combinations of parameters specified by the param grid.

New in version 1.4.0.

class pyspark.ml.tuning.**CrossValidator**(*estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *numFolds=3*, *seed=None*, *parallelism=1*, *collectSubModels=False*)

K-fold cross validation performs model selection by splitting the dataset into a set of non-overlapping randomly partitioned folds which are used as separate training and test datasets e.g., with k=3 folds, K-fold cross validation will generate 3 (training, test) dataset pairs, each of which uses 2/3 of the data for training and 1/3 for testing. Each fold is used as the test set exactly once.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.ml.linalg import Vectors
>>> dataset = spark.createDataFrame(
...     [(Vectors.dense([0.0]), 0.0),
...      (Vectors.dense([0.4]), 1.0),
...      (Vectors.dense([0.5]), 0.0),
...      (Vectors.dense([0.6]), 1.0),
...      (Vectors.dense([1.0]), 1.0)] * 10,
...     ["features", "label"])
>>> lr = LogisticRegression()
>>> grid = ParamGridBuilder().addGrid(lr.maxIter, [0, 1]).build()
```

(continues on next page)

(continued from previous page)

```
>>> evaluator = BinaryClassificationEvaluator()
>>> cv = CrossValidator(estimator=lr, estimatorParamMaps=grid,␣
↪evaluator=evaluator,
...     parallelism=2)
>>> cvModel = cv.fit(dataset)
>>> cvModel.avgMetrics[0]
0.5
>>> evaluator.evaluate(cvModel.transform(dataset))
0.8333...
```

New in version 1.4.0.

copy(*extra=None*)

Creates a copy of this instance with a randomly generated uid and some extra params. This copies creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters **extra** – Extra parameters to copy to the new instance

Returns Copy of this instance

New in version 1.4.0.

getNumFolds()

Gets the value of numFolds or its default value.

New in version 1.4.0.

classmethod read()

Returns an MLReader instance for this class.

New in version 2.3.0.

setNumFolds(*value*)

Sets the value of `numFolds`.

New in version 1.4.0.

setParams(*estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *numFolds=3*, *seed=None*, *parallelism=1*, *collectSubModels=False*)

setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, numFolds=3, seed=None, parallelism=1, collectSubModels=False): Sets params for cross validator.

New in version 1.4.0.

write()

Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

class pyspark.ml.tuning.**CrossValidatorModel**(*bestModel*, *avgMetrics=[]*, *subModels=None*)

CrossValidatorModel contains the model with the highest average cross-validation metric across folds and uses this model to transform input data. CrossValidatorModel also tracks the metrics for each param map evaluated.

New in version 1.4.0.

avgMetrics = None
Average cross-validation metrics for each paramMap in CrossValidator.estimatorParamMaps, in the corresponding order.

bestModel = None
best model from cross validation

copy(*extra=None*)
Creates a copy of this instance with a randomly generated uid and some extra params. This copies the underlying bestModel, creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over. It does not copy the extra Params into the subModels.

Parameters **extra** – Extra parameters to copy to the new instance

Returns Copy of this instance

New in version 1.4.0.

classmethod read()
Returns an MLReader instance for this class.

New in version 2.3.0.

subModels = None
sub model list from cross validation

write()
Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

class pyspark.ml.tuning.**TrainValidationSplit**(*estimator=None, estimatorParamMaps=None, evaluator=None, trainRatio=0.75, parallelism=1, collectSubModels=False, seed=None*)

Note: Experimental

Validation for hyper-parameter tuning. Randomly splits the input dataset into train and validation sets, and uses evaluation metric on the validation set to select the best model. Similar to *CrossValidator*, but only splits the set once.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.ml.linalg import Vectors
>>> dataset = spark.createDataFrame(
...     [(Vectors.dense([0.0]), 0.0),
...      (Vectors.dense([0.4]), 1.0),
...      (Vectors.dense([0.5]), 0.0),
...      (Vectors.dense([0.6]), 1.0),
```

(continues on next page)

(continued from previous page)

```
...       (Vectors.dense([1.0]), 1.0)] * 10,
...      ["features", "label"])
>>> lr = LogisticRegression()
>>> grid = ParamGridBuilder().addGrid(lr.maxIter, [0, 1]).build()
>>> evaluator = BinaryClassificationEvaluator()
>>> tvs = TrainValidationSplit(estimator=lr, estimatorParamMaps=grid,␣
↪evaluator=evaluator,
...     parallelism=2)
>>> tvsModel = tvs.fit(dataset)
>>> evaluator.evaluate(tvsModel.transform(dataset))
0.8333...
```

New in version 2.0.0.

copy(*extra=None*)

Creates a copy of this instance with a randomly generated uid and some extra params. This copies creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters **extra** – Extra parameters to copy to the new instance

Returns Copy of this instance

New in version 2.0.0.

getTrainRatio()

Gets the value of trainRatio or its default value.

New in version 2.0.0.

classmethod read()

Returns an MLReader instance for this class.

New in version 2.3.0.

setParams(*estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *trainRatio=0.75*, *parallelism=1*, *collectSubModels=False*, *seed=None*)

setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, trainRatio=0.75, parallelism=1, collectSubModels=False, seed=None): Sets params for the train validation split.

New in version 2.0.0.

setTrainRatio(*value*)

Sets the value of `trainRatio`.

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

class pyspark.ml.tuning.**TrainValidationSplitModel**(*bestModel*, *validationMetrics=[]*, *subModels=None*)

Note: Experimental

Model from train validation split.

New in version 2.0.0.

bestModel = None
best model from train validation split

copy(*extra=None*)
Creates a copy of this instance with a randomly generated uid and some extra params. This copies the underlying bestModel, creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over. And, this creates a shallow copy of the validationMetrics. It does not copy the extra Params into the subModels.

Parameters **extra** – Extra parameters to copy to the new instance

Returns Copy of this instance

New in version 2.0.0.

classmethod read()
Returns an MLReader instance for this class.

New in version 2.3.0.

subModels = None
sub models from train validation split

validationMetrics = None
evaluated validation metrics

write()
Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

21.8 Evaluation API

class pyspark.ml.evaluation.**Evaluator**
Base class for evaluators that compute metrics from predictions.

New in version 1.4.0.

evaluate(*dataset*, *params=None*)
Evaluates the output with optional parameters.

Parameters

- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns metric

New in version 1.4.0.

isLargerBetter()

Indicates whether the metric returned by *evaluate()* should be maximized (True, default) or minimized (False). A given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

class pyspark.ml.evaluation.**BinaryClassificationEvaluator**(*rawPredictionCol='rawPrediction', labelCol='label', metricName='areaUnderROC'*)

Note: Experimental

Evaluator for binary classification, which expects two input columns: rawPrediction and label. The rawPrediction column can be of type double (binary 0/1 prediction, or probability of label 1) or of type vector (length-2 vector of raw predictions, scores, or label probabilities).

```
>>> from pyspark.ml.linalg import Vectors
>>> scoreAndLabels = map(lambda x: (Vectors.dense([1.0 - x[0], x[0]]),␣
↪x[1]),
...     [(0.1, 0.0), (0.1, 1.0), (0.4, 0.0), (0.6, 0.0), (0.6, 1.0), (0.6,
↪ 1.0), (0.8, 1.0)])
>>> dataset = spark.createDataFrame(scoreAndLabels, ["raw", "label"])
...
>>> evaluator = BinaryClassificationEvaluator(rawPredictionCol="raw")
>>> evaluator.evaluate(dataset)
0.70...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "areaUnderPR"})
0.83...
>>> bce_path = temp_path + "/bce"
>>> evaluator.save(bce_path)
>>> evaluator2 = BinaryClassificationEvaluator.load(bce_path)
>>> str(evaluator2.getRawPredictionCol())
'raw'
```

New in version 1.4.0.

getMetricName()

Gets the value of metricName or its default value.

New in version 1.4.0.

setMetricName(*value*)
Sets the value of `metricName`.

New in version 1.4.0.

setParams(*self*, *rawPredictionCol="rawPrediction"*, *labelCol="label"*, *metricName="areaUnderROC"*)
Sets params for binary classification evaluator.

New in version 1.4.0.

class pyspark.ml.evaluation.**RegressionEvaluator**(*predictionCol='prediction'*, *labelCol='label'*, *metricName='rmse'*)

Note: Experimental

Evaluator for Regression, which expects two input columns: prediction and label.

```
>>> scoreAndLabels = [(-28.98343821, -27.0), (20.21491975, 21.5),
...   (-25.98418959, -22.0), (30.69731842, 33.0), (74.69283752, 71.0)]
>>> dataset = spark.createDataFrame(scoreAndLabels, ["raw", "label"])
...
>>> evaluator = RegressionEvaluator(predictionCol="raw")
>>> evaluator.evaluate(dataset)
2.842...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "r2"})
0.993...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "mae"})
2.649...
>>> re_path = temp_path + "/re"
>>> evaluator.save(re_path)
>>> evaluator2 = RegressionEvaluator.load(re_path)
>>> str(evaluator2.getPredictionCol())
'raw'
```

New in version 1.4.0.

getMetricName()
Gets the value of metricName or its default value.

New in version 1.4.0.

setMetricName(*value*)
Sets the value of `metricName`.

New in version 1.4.0.

setParams(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="rmse"*)
Sets params for regression evaluator.

New in version 1.4.0.

class pyspark.ml.evaluation.**MulticlassClassificationEvaluator**(*predictionCol='prediction', labelCol='label', metricName='f1'*)

Note: Experimental

Evaluator for Multiclass Classification, which expects two input columns: prediction and label.

```
>>> scoreAndLabels = [(0.0, 0.0), (0.0, 1.0), (0.0, 0.0),
...     (1.0, 0.0), (1.0, 1.0), (1.0, 1.0), (1.0, 1.0), (2.0, 2.0), (2.0,
↪ 0.0)]
>>> dataset = spark.createDataFrame(scoreAndLabels, ["prediction", "label
↪"])
...
>>> evaluator = MulticlassClassificationEvaluator(predictionCol=
↪"prediction")
>>> evaluator.evaluate(dataset)
0.66...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "accuracy"})
0.66...
>>> mce_path = temp_path + "/mce"
>>> evaluator.save(mce_path)
>>> evaluator2 = MulticlassClassificationEvaluator.load(mce_path)
>>> str(evaluator2.getPredictionCol())
'prediction'
```

New in version 1.5.0.

getMetricName()

Gets the value of metricName or its default value.

New in version 1.5.0.

setMetricName(*value*)

Sets the value of `metricName`.

New in version 1.5.0.

setParams(*self, predictionCol="prediction", labelCol="label", metricName="f1"*)

Sets params for multiclass classification evaluator.

New in version 1.5.0.

class pyspark.ml.evaluation.**ClusteringEvaluator**(*predictionCol='prediction', featuresCol='features', metricName='silhouette', distanceMeasure='squaredEuclidean'*)

Note: Experimental

Evaluator for Clustering results, which expects two input columns: prediction and features. The metric computes the Silhouette measure using the squared Euclidean distance.

The Silhouette is a measure for the validation of the consistency within clusters. It ranges between 1 and -1, where a value close to 1 means that the points in a cluster are close to the other points in the same cluster and far from the points of the other clusters.

```
>>> from pyspark.ml.linalg import Vectors
>>> featureAndPredictions = map(lambda x: (Vectors.dense(x[0]), x[1]),
...     [([0.0, 0.5], 0.0), ([0.5, 0.0], 0.0), ([10.0, 11.0], 1.0),
...     ([10.5, 11.5], 1.0), ([1.0, 1.0], 0.0), ([8.0, 6.0], 1.0)])
>>> dataset = spark.createDataFrame(featureAndPredictions, ["features",
↪"prediction"])
...
>>> evaluator = ClusteringEvaluator(predictionCol="prediction")
>>> evaluator.evaluate(dataset)
0.9079...
>>> ce_path = temp_path + "/ce"
>>> evaluator.save(ce_path)
>>> evaluator2 = ClusteringEvaluator.load(ce_path)
>>> str(evaluator2.getPredictionCol())
'prediction'
```

New in version 2.3.0.

getDistanceMeasure()

Gets the value of $distanceMeasure$

New in version 2.4.0.

getMetricName()

Gets the value of metricName or its default value.

New in version 2.3.0.

setDistanceMeasure(*value*)

Sets the value of `distanceMeasure`.

New in version 2.4.0.

setMetricName(*value*)

Sets the value of `metricName`.

New in version 2.3.0.

setParams(*self*, *predictionCol="prediction"*, *featuresCol="features"*, *metricName="silhouette"*, *distanceMeasure="squaredEuclidean"*)

Sets params for clustering evaluator.

New in version 2.3.0.

CHAPTER

TWENTYTWO

MAIN REFERENCE

BIBLIOGRAPHY

[Feng2017] W. Feng and M. Chen. Learning Apache Spark, Github 2017.

[Karau2015] H. 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/

PYTHON MODULE INDEX

p

pyspark.ml.classification, 318
pyspark.ml.clustering, 341
pyspark.ml.evaluation, 369
pyspark.ml.pipeline, 362
pyspark.ml.recommendation, 357
pyspark.ml.regression, 297
pyspark.ml.stat, 291
pyspark.ml.tuning, 364

INDEX

A

accuracy (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
addGrid() (*pyspark.ml.tuning.ParamGridBuilder method*), 365
AFTSurvivalRegression (*class in pyspark.ml.regression*), 297
AFTSurvivalRegressionModel (*class in pyspark.ml.regression*), 299
aic (*pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute*), 307
ALS (*class in pyspark.ml.recommendation*), 357
ALSModel (*class in pyspark.ml.recommendation*), 361
areaUnderROC (*pyspark.ml.classification.BinaryLogisticRegressionSummary attribute*), 326
assignClusters() (*pyspark.ml.clustering.PowerIterationClustering method*), 355
avgMetrics (*pyspark.ml.tuning.CrossValidatorModel attribute*), 367

B

baseOn() (*pyspark.ml.tuning.ParamGridBuilder method*), 365
bestModel (*pyspark.ml.tuning.CrossValidatorModel attribute*), 367
bestModel (*pyspark.ml.tuning.TrainValidationSplitModel attribute*), 369
BinaryClassificationEvaluator (*class in pyspark.ml.evaluation*), 370
BinaryLogisticRegressionSummary (*class in pyspark.ml.classification*), 326
BinaryLogisticRegressionTrainingSummary (*class in pyspark.ml.classification*), 327
BisectingKMeans (*class in pyspark.ml.clustering*), 341
BisectingKMeansModel (*class in pyspark.ml.clustering*), 343
BisectingKMeansSummary (*class in pyspark.ml.clustering*), 343
boundaries (*pyspark.ml.regression.IsotonicRegressionModel attribute*), 310
build() (*pyspark.ml.tuning.ParamGridBuilder method*), 365

C

ChiSquareTest (*class in pyspark.ml.stat*), 291
clusterCenters() (*pyspark.ml.clustering.BisectingKMeansModel method*), 343
clusterCenters() (*pyspark.ml.clustering.KMeansModel method*), 345
ClusteringEvaluator (*class in pyspark.ml.evaluation*), 372
coefficientMatrix (*pyspark.ml.classification.LogisticRegressionModel attribute*), 323
coefficients (*pyspark.ml.classification.LinearSVCModel attribute*), 319
coefficients (*pyspark.ml.classification.LogisticRegressionModel attribute*), 323
coefficients (*pyspark.ml.regression.AFTSurvivalRegressionModel attribute*), 299
coefficients (*pys-*

park.ml.regression.GeneralizedLinearRegressionModel attribute), 306
coefficients (*pyspark.ml.regression.LinearRegressionModel attribute*), 312
coefficientStandardErrors (*pyspark.ml.regression.GeneralizedLinearRegressionTrainingSummary attribute*), 308
coefficientStandardErrors (*pyspark.ml.regression.LinearRegressionSummary attribute*), 313
computeCost() (*pyspark.ml.clustering.BisectingKMeansModel method*), 343
computeCost() (*pyspark.ml.clustering.KMeansModel method*), 345
Configure Spark on Mac and Ubuntu, 16
copy() (*pyspark.ml.classification.OneVsRest method*), 340
copy() (*pyspark.ml.classification.OneVsRestModel method*), 341
copy() (*pyspark.ml.pipeline.Pipeline method*), 362
copy() (*pyspark.ml.pipeline.PipelineModel method*), 363
copy() (*pyspark.ml.tuning.CrossValidator method*), 366
copy() (*pyspark.ml.tuning.CrossValidatorModel method*), 367
copy() (*pyspark.ml.tuning.TrainValidationSplit method*), 368
copy() (*pyspark.ml.tuning.TrainValidationSplitModel method*), 369
corr() (*pyspark.ml.stat.Correlation static method*), 292
Correlation (*class in pyspark.ml.stat*), 292
count() (*pyspark.ml.stat.Summarizer static method*), 295
CrossValidator (*class in pyspark.ml.tuning*), 365
CrossValidatorModel (*class in pyspark.ml.tuning*), 366

D

DecisionTreeClassificationModel (*class in pyspark.ml.classification*), 329
DecisionTreeClassifier (*class in pyspark.ml.classification*), 327
DecisionTreeRegressionModel (*class in pyspark.ml.regression*), 300
DecisionTreeRegressor (*class in pyspark.ml.regression*), 299
degreesOfFreedom (*pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute*), 307
degreesOfFreedom (*pyspark.ml.regression.LinearRegressionSummary attribute*), 313
describeTopics() (*pyspark.ml.clustering.LDAModel method*), 352
deviance (*pyspark.ml.regression.GeneralizedLinearRegressionSum attribute*), 307
devianceResiduals (*pyspark.ml.regression.LinearRegressionSummary attribute*), 313
dispersion (*pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute*), 307
DistributedLDAModel (*class in pyspark.ml.clustering*), 353

E

estimatedDocConcentration() (*pyspark.ml.clustering.LDAModel method*), 352
evaluate() (*pyspark.ml.classification.LogisticRegressionModel method*), 323
evaluate() (*pyspark.ml.evaluation.Evaluator method*), 369
evaluate() (*pyspark.ml.regression.GeneralizedLinearRegressionModel method*), 306
evaluate() (*pyspark.ml.regression.LinearRegressionModel method*), 312
evaluateEachIteration() (*pyspark.ml.classification.GBTClassificationModel method*), 332
evaluateEachIteration() (*pyspark.ml.regression.GBTRegressionModel method*), 302
Evaluator (*class in pyspark.ml.evaluation*), 369

explainedVariance (*pyspark.ml.regression.LinearRegressionSummary attribute*), 313

F

falsePositiveRateByLabel (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
featureImportances (*pyspark.ml.classification.DecisionTreeClassificationModel attribute*), 329
featureImportances (*pyspark.ml.classification.GBTClassificationModel attribute*), 332
featureImportances (*pyspark.ml.classification.RandomForestClassificationModel attribute*), 334
featureImportances (*pyspark.ml.regression.DecisionTreeRegressionModel attribute*), 300
featureImportances (*pyspark.ml.regression.GBTRegressionModel attribute*), 303
featureImportances (*pyspark.ml.regression.RandomForestRegressionModel attribute*), 317
featuresCol (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
featuresCol (*pyspark.ml.regression.LinearRegressionSummary attribute*), 313
fMeasureByLabel() (*pyspark.ml.classification.LogisticRegressionSummary method*), 324
fMeasureByThreshold (*pyspark.ml.classification.BinaryLogisticRegressionSummary attribute*), 326

G

GaussianMixture (*class in pyspark.ml.clustering*), 346
GaussianMixtureModel (*class in pyspark.ml.clustering*), 347
GaussianMixtureSummary (*class in pyspark.ml.clustering*), 348
gaussiansDF (*pyspark.ml.clustering.GaussianMixtureModel attribute*), 347
GBTClassificationModel (*class in pyspark.ml.classification*), 332
GBTClassifier (*class in pyspark.ml.classification*), 330
GBTRegressionModel (*class in pyspark.ml.regression*), 302
GBTRegressor (*class in pyspark.ml.regression*), 301
GeneralizedLinearRegression (*class in pyspark.ml.regression*), 303
GeneralizedLinearRegressionModel (*class in pyspark.ml.regression*), 306
GeneralizedLinearRegressionSummary (*class in pyspark.ml.regression*), 307
GeneralizedLinearRegressionTrainingSummary (*class in pyspark.ml.regression*), 308
getAlpha() (*pyspark.ml.recommendation.ALS method*), 358
getBlockSize() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 338
getCensorCol() (*pyspark.ml.regression.AFTSurvivalRegression method*), 298
getCheckpointFiles() (*pyspark.ml.clustering.DistributedLDAModel method*), 353
getColdStartStrategy() (*pyspark.ml.recommendation.ALS method*), 358
getDistanceMeasure() (*pyspark.ml.clustering.BisectingKMeans method*), 342
getDistanceMeasure() (*pyspark.ml.clustering.KMeans method*), 344
getDistanceMeasure() (*pyspark.ml.evaluation.ClusteringEvaluator method*), 373
getDocConcentration() (*pyspark.ml.clustering.LDA method*), 349
getDstCol() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
getEpsilon() (*pyspark.ml.regression.LinearRegression method*), 312

getFamily() (*pyspark.ml.classification.LogisticRegression method*), 321
getFamily() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 305
getFeatureIndex() (*pyspark.ml.regression.IsotonicRegression method*), 310
getFinalStorageLevel() (*pyspark.ml.recommendation.ALS method*), 358
getImplicitPrefs() (*pyspark.ml.recommendation.ALS method*), 359
getInitialWeights() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 338
getInitMode() (*pyspark.ml.clustering.KMeans method*), 344
getInitMode() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
getInitSteps() (*pyspark.ml.clustering.KMeans method*), 344
getIntermediateStorageLevel() (*pyspark.ml.recommendation.ALS method*), 359
getIsotonic() (*pyspark.ml.regression.IsotonicRegression method*), 310
getItemCol() (*pyspark.ml.recommendation.ALS method*), 359
getK() (*pyspark.ml.clustering.BisectingKMeans method*), 342
getK() (*pyspark.ml.clustering.GaussianMixture method*), 347
getK() (*pyspark.ml.clustering.KMeans method*), 345
getK() (*pyspark.ml.clustering.LDA method*), 349
getK() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
getKeepLastCheckpoint() (*pyspark.ml.clustering.LDA method*), 350
getLayers() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 338
getLearningDecay() (*pyspark.ml.clustering.LDA method*), 350
getLearningOffset() (*pyspark.ml.clustering.LDA method*), 350
getLink() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 305
getLinkPower() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 305
getLinkPredictionCol() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 305
getLossType() (*pyspark.ml.classification.GBTClassifier method*), 331
getLossType() (*pyspark.ml.regression.GBTRegressor method*), 302
getLowerBoundsOnCoefficients() (*pyspark.ml.classification.LogisticRegression method*), 321
getLowerBoundsOnIntercepts() (*pyspark.ml.classification.LogisticRegression method*), 321
getMetricName() (*pyspark.ml.evaluation.BinaryClassificationEvaluator method*), 370
getMetricName() (*pyspark.ml.evaluation.ClusteringEvaluator method*), 373
getMetricName() (*pyspark.ml.evaluation.MulticlassClassificationEvaluator method*), 372
getMetricName() (*pyspark.ml.evaluation.RegressionEvaluator method*), 371
getMinDivisibleClusterSize() (*pyspark.ml.clustering.BisectingKMeans method*), 342
getModelType() (*pyspark.ml.classification.NaiveBayes method*), 336
getNonnegative() (*pyspark.ml.recommendation.ALS method*), 359
getNumFolds() (*pyspark.ml.tuning.CrossValidator method*),

366
getNumItemBlocks() (*pyspark.ml.recommendation.ALS method*), 359
getNumUserBlocks() (*pyspark.ml.recommendation.ALS method*), 359
getOffsetCol() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 305
getOptimizeDocConcentration() (*pyspark.ml.clustering.LDA method*), 350
getOptimizer() (*pyspark.ml.clustering.LDA method*), 350
getQuantileProbabilities() (*pyspark.ml.regression.AFTSurvivalRegression method*), 298
getQuantilesCol() (*pyspark.ml.regression.AFTSurvivalRegression method*), 298
getRank() (*pyspark.ml.recommendation.ALS method*), 359
getRatingCol() (*pyspark.ml.recommendation.ALS method*), 359
getSmoothing() (*pyspark.ml.classification.NaiveBayes method*), 336
getSrcCol() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
getStagePath() (*pyspark.ml.pipeline.PipelineSharedReadWrite static method*), 364
getStages() (*pyspark.ml.pipeline.Pipeline method*), 362
getStepSize() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 338
getSubsamplingRate() (*pyspark.ml.clustering.LDA method*), 350
getThreshold() (*pyspark.ml.classification.LogisticRegression method*), 321
getThresholds() (*pyspark.ml.classification.LogisticRegression method*), 322
getTopicConcentration() (*pyspark.ml.clustering.LDA method*), 350
getTopicDistributionCol() (*pyspark.ml.clustering.LDA method*), 350
getTrainRatio() (*pyspark.ml.tuning.TrainValidationSplit method*), 368
getUpperBoundsOnCoefficients() (*pyspark.ml.classification.LogisticRegression method*), 322
getUpperBoundsOnIntercepts() (*pyspark.ml.classification.LogisticRegression method*), 322
getUserCol() (*pyspark.ml.recommendation.ALS method*), 359
getVariancePower() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 305

H

hasSummary (*pyspark.ml.classification.LogisticRegressionModel attribute*), 323
hasSummary (*pyspark.ml.clustering.BisectingKMeansModel attribute*), 343
hasSummary (*pyspark.ml.clustering.GaussianMixtureModel attribute*), 347
hasSummary (*pyspark.ml.clustering.KMeansModel attribute*), 345
hasSummary (*pyspark.ml.regression.GeneralizedLinearRegressionModel attribute*), 307
hasSummary (*pyspark.ml.regression.LinearRegressionModel attribute*), 312

I

intercept (*pyspark.ml.classification.LinearSVCModel attribute*), 319
intercept (*pyspark.ml.classification.LogisticRegressionModel attribute*), 323
intercept (*pyspark.ml.regression.AFTSurvivalRegressionModel attribute*), 299

intercept (*pyspark.ml.regression.GeneralizedLinearRegressionModel attribute*), 307
intercept (*pyspark.ml.regression.LinearRegressionModel attribute*), 312
interceptVector (*pyspark.ml.classification.LogisticRegressionModel attribute*), 323
isDistributed() (*pyspark.ml.clustering.LDAModel method*), 352
isLargerBetter() (*pyspark.ml.evaluation.Evaluator method*), 370
IsotonicRegression (*class in pyspark.ml.regression*), 309
IsotonicRegressionModel (*class in pyspark.ml.regression*), 310
itemFactors (*pyspark.ml.recommendation.ALSModel attribute*), 361

K

KMeans (*class in pyspark.ml.clustering*), 343
KMeansModel (*class in pyspark.ml.clustering*), 345
KolmogorovSmirnovTest (*class in pyspark.ml.stat*), 293

L

labelCol (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
labelCol (*pyspark.ml.regression.LinearRegressionSummary attribute*), 313
labels (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
layers (*pyspark.ml.classification.MultilayerPerceptronClassificationModel attribute*), 339
LDA (*class in pyspark.ml.clustering*), 348
LDAModel (*class in pyspark.ml.clustering*), 352
LinearRegression (*class in pyspark.ml.regression*), 310
LinearRegressionModel (*class in pyspark.ml.regression*), 312
LinearRegressionSummary (*class in pyspark.ml.regression*), 313
LinearRegressionTrainingSummary (*class in pyspark.ml.regression*), 315
LinearSVC (*class in pyspark.ml.classification*), 318
LinearSVCModel (*class in pyspark.ml.classification*), 319
load() (*pyspark.ml.pipeline.PipelineModelReader method*), 363
load() (*pyspark.ml.pipeline.PipelineReader method*), 364
load() (*pyspark.ml.pipeline.PipelineSharedReadWrite static method*), 364
LocalLDAModel (*class in pyspark.ml.clustering*), 353
LogisticRegression (*class in pyspark.ml.classification*), 319
LogisticRegressionModel (*class in pyspark.ml.classification*), 323
LogisticRegressionSummary (*class in pyspark.ml.classification*), 324
LogisticRegressionTrainingSummary (*class in pyspark.ml.classification*), 325
logLikelihood (*pyspark.ml.clustering.GaussianMixtureSummary attribute*), 348
logLikelihood() (*pyspark.ml.clustering.LDAModel method*), 353
logPerplexity() (*pyspark.ml.clustering.LDAModel method*), 353
logPrior() (*pyspark.ml.clustering.DistributedLDAModel method*), 354

M

max() (*pyspark.ml.stat.Summarizer static method*), 295
mean() (*pyspark.ml.stat.Summarizer static method*), 295
meanAbsoluteError (*pyspark.ml.regression.LinearRegressionSummary attribute*), 314
meanSquaredError (*pyspark.ml.regression.LinearRegressionSummary attribute*), 314
metrics() (*pyspark.ml.stat.Summarizer static method*), 295

min() (*pyspark.ml.stat.Summarizer static method*), 296
MulticlassClassificationEvaluator (*class in pyspark.ml.evaluation*), 371
MultilayerPerceptronClassificationModel (*class in pyspark.ml.classification*), 339
MultilayerPerceptronClassifier (*class in pyspark.ml.classification*), 337

N

NaiveBayes (*class in pyspark.ml.classification*), 335
NaiveBayesModel (*class in pyspark.ml.classification*), 336
normL1() (*pyspark.ml.stat.Summarizer static method*), 296
normL2() (*pyspark.ml.stat.Summarizer static method*), 296
nullDeviance (*pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute*), 307
numInstances (*pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute*), 308
numInstances (*pyspark.ml.regression.LinearRegressionSummary attribute*), 314
numIterations (*pyspark.ml.regression.GeneralizedLinearRegressionTrainingSummary attribute*), 309
numNonZeros() (*pyspark.ml.stat.Summarizer static method*), 296

O

objectiveHistory (*pyspark.ml.classification.LogisticRegressionTrainingSummary attribute*), 325
objectiveHistory (*pyspark.ml.regression.LinearRegressionTrainingSummary attribute*), 316
OneVsRest (*class in pyspark.ml.classification*), 339
OneVsRestModel (*class in pyspark.ml.classification*), 341

P

ParamGridBuilder (*class in pyspark.ml.tuning*), 364
pi (*pyspark.ml.classification.NaiveBayesModel attribute*), 336
Pipeline (*class in pyspark.ml.pipeline*), 362
PipelineModel (*class in pyspark.ml.pipeline*), 363
PipelineModelReader (*class in pyspark.ml.pipeline*), 363
PipelineModelWriter (*class in pyspark.ml.pipeline*), 363
PipelineReader (*class in pyspark.ml.pipeline*), 363
PipelineSharedReadWrite (*class in pyspark.ml.pipeline*), 364
PipelineWriter (*class in pyspark.ml.pipeline*), 364
PowerIterationClustering (*class in pyspark.ml.clustering*), 354
pr (*pyspark.ml.classification.BinaryLogisticRegressionSummary attribute*), 326
precisionByLabel (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
precisionByThreshold (*pyspark.ml.classification.BinaryLogisticRegressionSummary attribute*), 326
predict() (*pyspark.ml.regression.AFTSurvivalRegressionModel method*), 299
predictionCol (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
predictionCol (*pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute*), 308
predictionCol (*pyspark.ml.regression.LinearRegressionSummary attribute*), 314
predictions (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 324
predictions (*pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute*), 308
predictions (*pyspark.ml.regression.IsotonicRegressionModel attribute*), 310
predictions (*pyspark.ml.regression.LinearRegressionSummary

attribute), 314
predictQuantiles() (pyspark.ml.regression.AFTSurvivalRegressionModel method), 299
probability (pyspark.ml.clustering.GaussianMixtureSummary attribute), 348
probabilityCol (pyspark.ml.classification.LogisticRegressionSummary attribute), 325
probabilityCol (pyspark.ml.clustering.GaussianMixtureSummary attribute), 348
pValues (pyspark.ml.regression.GeneralizedLinearRegressionTrainingSummary attribute), 309
pValues (pyspark.ml.regression.LinearRegressionSummary attribute), 314
pyspark.ml.classification (module), 318
pyspark.ml.clustering (module), 341
pyspark.ml.evaluation (module), 369
pyspark.ml.pipeline (module), 362
pyspark.ml.recommendation (module), 357
pyspark.ml.regression (module), 297
pyspark.ml.stat (module), 291
pyspark.ml.tuning (module), 364

R

r2 (pyspark.ml.regression.LinearRegressionSummary attribute), 314
r2adj (pyspark.ml.regression.LinearRegressionSummary attribute), 315
RandomForestClassificationModel (class in pyspark.ml.classification), 334
RandomForestClassifier (class in pyspark.ml.classification), 332
RandomForestRegressionModel (class in pyspark.ml.regression), 317
RandomForestRegressor (class in pyspark.ml.regression), 316
rank (pyspark.ml.recommendation.ALSModel attribute), 361
rank (pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute), 308
read() (pyspark.ml.pipeline.Pipeline class method), 362
read() (pyspark.ml.pipeline.PipelineModel class method), 363
read() (pyspark.ml.tuning.CrossValidator class method), 366
read() (pyspark.ml.tuning.CrossValidatorModel class method), 367
read() (pyspark.ml.tuning.TrainValidationSplit class method), 368
read() (pyspark.ml.tuning.TrainValidationSplitModel class method), 369
recallByLabel (pyspark.ml.classification.LogisticRegressionSummary attribute), 325
recallByThreshold (pyspark.ml.classification.BinaryLogisticRegressionSummary attribute), 327
recommendForAllItems() (pyspark.ml.recommendation.ALSModel method), 361
recommendForAllUsers() (pyspark.ml.recommendation.ALSModel method), 361
recommendForItemSubset() (pyspark.ml.recommendation.ALSModel method), 361
recommendForUserSubset() (pyspark.ml.recommendation.ALSModel method), 361
RegressionEvaluator (class in pyspark.ml.evaluation), 371
residualDegreeOfFreedom (pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute), 308
residualDegreeOfFreedomNull (pyspark.ml.regression.GeneralizedLinearRegressionSummary attribute), 308
residuals (pyspark.ml.regression.LinearRegressionSummary attribute), 315
residuals() (pyspark.ml.regression.GeneralizedLinearRegressionSummary method), 308
roc (pyspark.ml.classification.BinaryLogisticRegressionSummary attribute), 327
rootMeanSquaredError (pyspark.ml.regression.LinearRegressionSummary attribute), 315
Run on Databricks Community Cloud, 11

S

saveImpl() (*pyspark.ml.pipeline.PipelineModelWriter method*), 363
saveImpl() (*pyspark.ml.pipeline.PipelineSharedReadWrite static method*), 364
saveImpl() (*pyspark.ml.pipeline.PipelineWriter method*), 364
scale (*pyspark.ml.regression.AFTSurvivalRegressionModel attribute*), 299
scale (*pyspark.ml.regression.LinearRegressionModel attribute*), 312
Set up Spark on Cloud, 20
setAlpha() (*pyspark.ml.recommendation.ALS method*), 359
setBlockSize() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 338
setCensorCol() (*pyspark.ml.regression.AFTSurvivalRegression method*), 298
setColdStartStrategy() (*pyspark.ml.recommendation.ALS method*), 359
setDistanceMeasure() (*pyspark.ml.clustering.BisectingKMeans method*), 342
setDistanceMeasure() (*pyspark.ml.clustering.KMeans method*), 345
setDistanceMeasure() (*pyspark.ml.evaluation.ClusteringEvaluator method*), 373
setDocConcentration() (*pyspark.ml.clustering.LDA method*), 350
setDstCol() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
setEpsilon() (*pyspark.ml.regression.LinearRegression method*), 312
setFamily() (*pyspark.ml.classification.LogisticRegression method*), 322
setFamily() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 306
setFeatureIndex() (*pyspark.ml.regression.IsotonicRegression method*), 310
setFeatureSubsetStrategy() (*pyspark.ml.classification.GBTClassifier method*), 331
setFeatureSubsetStrategy() (*pyspark.ml.classification.RandomForestClassifier method*), 334
setFeatureSubsetStrategy() (*pyspark.ml.regression.GBTRegressor method*), 302
setFeatureSubsetStrategy() (*pyspark.ml.regression.RandomForestRegressor method*), 317
setFinalStorageLevel() (*pyspark.ml.recommendation.ALS method*), 359
setImplicitPrefs() (*pyspark.ml.recommendation.ALS method*), 360
setInitialWeights() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 339
setInitMode() (*pyspark.ml.clustering.KMeans method*), 345
setInitMode() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
setInitSteps() (*pyspark.ml.clustering.KMeans method*), 345
setIntermediateStorageLevel() (*pyspark.ml.recommendation.ALS method*), 360
setIsotonic() (*pyspark.ml.regression.IsotonicRegression method*), 310
setItemCol() (*pyspark.ml.recommendation.ALS method*), 360
setK() (*pyspark.ml.clustering.BisectingKMeans method*), 342
setK() (*pyspark.ml.clustering.GaussianMixture method*), 347
setK() (*pyspark.ml.clustering.KMeans method*), 345
setK() (*pyspark.ml.clustering.LDA method*), 350

setK() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
setKeepLastCheckpoint() (*pyspark.ml.clustering.LDA method*), 351
setLayers() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 339
setLearningDecay() (*pyspark.ml.clustering.LDA method*), 351
setLearningOffset() (*pyspark.ml.clustering.LDA method*), 351
setLink() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 306
setLinkPower() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 306
setLinkPredictionCol() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 306
setLossType() (*pyspark.ml.classification.GBTClassifier method*), 331
setLossType() (*pyspark.ml.regression.GBTRegressor method*), 302
setLowerBoundsOnCoefficients() (*pyspark.ml.classification.LogisticRegression method*), 322
setLowerBoundsOnIntercepts() (*pyspark.ml.classification.LogisticRegression method*), 322
setMetricName() (*pyspark.ml.evaluation.BinaryClassificationEvaluator method*), 370
setMetricName() (*pyspark.ml.evaluation.ClusteringEvaluator method*), 373
setMetricName() (*pyspark.ml.evaluation.MulticlassClassificationEvaluator method*), 372
setMetricName() (*pyspark.ml.evaluation.RegressionEvaluator method*), 371
setMinDivisibleClusterSize() (*pyspark.ml.clustering.BisectingKMeans method*), 343
setModelType() (*pyspark.ml.classification.NaiveBayes method*), 336
setNonnegative() (*pyspark.ml.recommendation.ALS method*), 360
setNumBlocks() (*pyspark.ml.recommendation.ALS method*), 360
setNumFolds() (*pyspark.ml.tuning.CrossValidator method*), 366
setNumItemBlocks() (*pyspark.ml.recommendation.ALS method*), 360
setNumUserBlocks() (*pyspark.ml.recommendation.ALS method*), 360
setOffsetCol() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 306
setOptimizeDocConcentration() (*pyspark.ml.clustering.LDA method*), 351
setOptimizer() (*pyspark.ml.clustering.LDA method*), 351
setParams() (*pyspark.ml.classification.DecisionTreeClassifier method*), 329
setParams() (*pyspark.ml.classification.GBTClassifier method*), 331
setParams() (*pyspark.ml.classification.LinearSVC method*), 319
setParams() (*pyspark.ml.classification.LogisticRegression method*), 322
setParams() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 339
setParams() (*pyspark.ml.classification.NaiveBayes method*), 336
setParams() (*pyspark.ml.classification.OneVsRest method*), 340
setParams() (*pyspark.ml.classification.RandomForestClassifier method*), 334

setParams() (*pyspark.ml.clustering.BisectingKMeans method*), 343
setParams() (*pyspark.ml.clustering.GaussianMixture method*), 347
setParams() (*pyspark.ml.clustering.KMeans method*), 345
setParams() (*pyspark.ml.clustering.LDA method*), 351
setParams() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
setParams() (*pyspark.ml.evaluation.BinaryClassificationEvaluator method*), 371
setParams() (*pyspark.ml.evaluation.ClusteringEvaluator method*), 373
setParams() (*pyspark.ml.evaluation.MulticlassClassificationEvaluator method*), 372
setParams() (*pyspark.ml.evaluation.RegressionEvaluator method*), 371
setParams() (*pyspark.ml.pipeline.Pipeline method*), 363
setParams() (*pyspark.ml.recommendation.ALS method*), 360
setParams() (*pyspark.ml.regression.AFTSurvivalRegression method*), 298
setParams() (*pyspark.ml.regression.DecisionTreeRegressor method*), 300
setParams() (*pyspark.ml.regression.GBTRegressor method*), 302
setParams() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 306
setParams() (*pyspark.ml.regression.IsotonicRegression method*), 310
setParams() (*pyspark.ml.regression.LinearRegression method*), 312
setParams() (*pyspark.ml.regression.RandomForestRegressor method*), 317
setParams() (*pyspark.ml.tuning.CrossValidator method*), 366
setParams() (*pyspark.ml.tuning.TrainValidationSplit method*), 368
setQuantileProbabilities() (*pyspark.ml.regression.AFTSurvivalRegression method*), 298
setQuantilesCol() (*pyspark.ml.regression.AFTSurvivalRegression method*), 298
setRank() (*pyspark.ml.recommendation.ALS method*), 360
setRatingCol() (*pyspark.ml.recommendation.ALS method*), 360
setSmoothing() (*pyspark.ml.classification.NaiveBayes method*), 336
setSrcCol() (*pyspark.ml.clustering.PowerIterationClustering method*), 356
setStages() (*pyspark.ml.pipeline.Pipeline method*), 363
setStepSize() (*pyspark.ml.classification.MultilayerPerceptronClassifier method*), 339
setSubsamplingRate() (*pyspark.ml.clustering.LDA method*), 352
setThreshold() (*pyspark.ml.classification.LogisticRegression method*), 322
setThresholds() (*pyspark.ml.classification.LogisticRegression method*), 322
setTopicConcentration() (*pyspark.ml.clustering.LDA method*), 352
setTopicDistributionCol() (*pyspark.ml.clustering.LDA method*), 352
setTrainRatio() (*pyspark.ml.tuning.TrainValidationSplit method*), 368
setUpperBoundsOnCoefficients() (*pyspark.ml.classification.LogisticRegression method*), 323
setUpperBoundsOnIntercepts() (*pys-*

park.ml.classification.LogisticRegression method), 323
setUserCol() (*pyspark.ml.recommendation.ALS method*), 360
setVariancePower() (*pyspark.ml.regression.GeneralizedLinearRegression method*), 306
solver (*pyspark.ml.regression.GeneralizedLinearRegressionTrainingSummary attribute*), 309
subModels (*pyspark.ml.tuning.CrossValidatorModel attribute*), 367
subModels (*pyspark.ml.tuning.TrainValidationSplitModel attribute*), 369
Summarizer (*class in pyspark.ml.stat*), 294
summary (*pyspark.ml.classification.LogisticRegressionModel attribute*), 323
summary (*pyspark.ml.clustering.BisectingKMeansModel attribute*), 343
summary (*pyspark.ml.clustering.GaussianMixtureModel attribute*), 348
summary (*pyspark.ml.clustering.KMeansModel attribute*), 346
summary (*pyspark.ml.regression.GeneralizedLinearRegressionModel attribute*), 307
summary (*pyspark.ml.regression.LinearRegressionModel attribute*), 312
summary() (*pyspark.ml.stat.SummaryBuilder method*), 296
SummaryBuilder (*class in pyspark.ml.stat*), 296

T

test() (*pyspark.ml.stat.ChiSquareTest static method*), 291
test() (*pyspark.ml.stat.KolmogorovSmirnovTest static method*), 293
theta (*pyspark.ml.classification.NaiveBayesModel attribute*), 336
toLocal() (*pyspark.ml.clustering.DistributedLDAModel method*), 354
topicsMatrix() (*pyspark.ml.clustering.LDAModel method*), 353
totalIterations (*pyspark.ml.classification.LogisticRegressionTrainingSummary attribute*), 326
totalIterations (*pyspark.ml.regression.LinearRegressionTrainingSummary attribute*), 316
trainingLogLikelihood() (*pyspark.ml.clustering.DistributedLDAModel method*), 354
TrainValidationSplit (*class in pyspark.ml.tuning*), 367
TrainValidationSplitModel (*class in pyspark.ml.tuning*), 368
trees (*pyspark.ml.classification.GBTClassificationModel attribute*), 332
trees (*pyspark.ml.classification.RandomForestClassificationModel attribute*), 335
trees (*pyspark.ml.regression.GBTRegressionModel attribute*), 303
trees (*pyspark.ml.regression.RandomForestRegressionModel attribute*), 318
truePositiveRateByLabel (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 325
tValues (*pyspark.ml.regression.GeneralizedLinearRegressionTrainingSummary attribute*), 309
tValues (*pyspark.ml.regression.LinearRegressionSummary attribute*), 315

U

userFactors (*pyspark.ml.recommendation.ALSModel attribute*), 362

V

validateStages() (*pyspark.ml.pipeline.PipelineSharedReadWrite static method*), 364
validationMetrics (*pyspark.ml.tuning.TrainValidationSplitModel attribute*), 369
variance() (*pyspark.ml.stat.Summarizer static method*), 296
vocabSize() (*pyspark.ml.clustering.LDAModel method*), 353

W

weightedFalsePositiveRate (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 325

`weightedFMeasure()` (*pyspark.ml.classification.LogisticRegressionSummary method*), 325
`weightedPrecision` (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 325
`weightedRecall` (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 325
`weightedTruePositiveRate` (*pyspark.ml.classification.LogisticRegressionSummary attribute*), 325
`weights` (*pyspark.ml.classification.MultilayerPerceptronClassificationModel attribute*), 339
`weights` (*pyspark.ml.clustering.GaussianMixtureModel attribute*), 348
`write()` (*pyspark.ml.pipeline.Pipeline method*), 363
`write()` (*pyspark.ml.pipeline.PipelineModel method*), 363
`write()` (*pyspark.ml.tuning.CrossValidator method*), 366
`write()` (*pyspark.ml.tuning.CrossValidatorModel method*), 367
`write()` (*pyspark.ml.tuning.TrainValidationSplit method*), 368
`write()` (*pyspark.ml.tuning.TrainValidationSplitModel method*), 369