Hands-On Big Data Analytics Using Apache Spark

Dan Chia-Tien Lo and Yong Shi

Department of Computer Science

Kennesaw State University

January, 2021

This book is partially supported by the Affordable Learning Georgia grant under R16.

Table of Contents

[Chapter 1. Introduction Machine Learning, Data Mining, and Big Data Analytics 6](#_Toc104537)

[1.1 Overview 6](#_Toc104538)

[1.2 Theory 6](#_Toc104539)

[1.3 Simple example 7](#_Toc104540)

[1.4 Exercise 7](#_Toc104541)

[1.5 Quizzes 7](#_Toc104542)

[Chapter 2. Hadoop 8](#_Toc104543)

[2.1 Overview 8](#_Toc104544)

[2.2 Theory 8](#_Toc104545)

[2.3 Simple example 9](#_Toc104546)

[2.4 Exercise 12](#_Toc104547)

[2.5 Quizzes 12](#_Toc104548)

[Chapter 3. Apache Spark 13](#_Toc104549)

[3.1 Overview 13](#_Toc104550)

[3.2 Theory 13](#_Toc104551)

[3.3 Simple example 14](#_Toc104552)

[3.4 Spark programming example 15](#_Toc104553)

[3.5 Exercise 17](#_Toc104554)

[3.6 Quizzes 17](#_Toc104555)

[Chapter 4. Decision trees 18](#_Toc104556)

[4.1 Overview 18](#_Toc104557)

[4.2 Theory 18](#_Toc104558)

[4.3 Simple example 18](#_Toc104559)

[4.4 Exercise 23](#_Toc104560)

[4.5 Quizzes 23](#_Toc104561)

[Chapter 5. Linear models 24](#_Toc104562)

[5.1 Overview 24](#_Toc104563)

[5.2 Theory 24](#_Toc104564)

[5.3 Simple example 24](#_Toc104565)

[5.4 Spark programming example 24](#_Toc104566)

[5.5 Exercise 26](#_Toc104567)

[5.6 Quizzes 26](#_Toc104568)

[Chapter 6. Linear regression 27](#_Toc104569)

[6.1 Overview 27](#_Toc104570)

[6.2 Theory 31](#_Toc104571)

[6.3 Simple example 31](#_Toc104572)

[6.4 Spark programming example 32](#_Toc104573)

[6.5 Exercise 36](#_Toc104574)

[6.6 Quizzes 36](#_Toc104575)

[Chapter 7. Regularization and Sparsity on Linear regression 37](#_Toc104576)

[7.1 Overview 37](#_Toc104577)

[7.2 Theory 37](#_Toc104578)

[7.3 Spark programming example 41](#_Toc104579)

[7.4 Exercise 43](#_Toc104580)

[7.5 Quizzes 43](#_Toc104581)

[Chapter 8. Logistic regression 44](#_Toc104582)

[8.1 Overview 44](#_Toc104583)

[8.2 Theory 44](#_Toc104584)

[8.3 Simple example 46](#_Toc104585)

[8.3 Spark programming example 47](#_Toc104586)

[8.4 Exercise 48](#_Toc104587)

[8.5 Quizzes 49](#_Toc104588)

[Chapter 9. Naïve Bayes Classifier 50](#_Toc104589)

[9.1 Overview 50](#_Toc104590)

[9.2 Theory 50](#_Toc104591)

[9.3 Simple example 52](#_Toc104592)

[9.4 Spark programming example 52](#_Toc104593)

[9.5 Exercise 54](#_Toc104594)

[9.6 Quizzes 54](#_Toc104595)

[Chapter 10. Fisher Linear Discriminant Analysis 55](#_Toc104596)

[10.1 Overview 55](#_Toc104597)

[10.2 Theory 55](#_Toc104598)

[10.3 Simple example 58](#_Toc104599)

[10.5 Exercise 60](#_Toc104600)

[10.6 Quizzes 60](#_Toc104601)

[Chapter 11. Principal Component Analysis 61](#_Toc104602)

[11.1 Overview 61](#_Toc104603)

[11.2 Theory 61](#_Toc104604)

[11.3 Simple example 64](#_Toc104605)

[11.4 Spark programming example 66](#_Toc104606)

[11.5 Exercise 66](#_Toc104607)

[11.6 Quizzes 68](#_Toc104608)

[Chapter 12. K-Nearest Neighbor Algorithm (KNN) 69](#_Toc104609)

[12.1 Overview 69](#_Toc104610)

[12.2 Theory 69](#_Toc104611)

[12.3 Simple example 70](#_Toc104612)

[12.4 Spark programming example 71](#_Toc104613)

[12.5 Exercise 72](#_Toc104614)

[12.6 Quizzes 72](#_Toc104615)

[Chapter 13. K-Means Algorithm 73](#_Toc104616)

[13.1 Overview 73](#_Toc104617)

[13.2 Theory 73](#_Toc104618)

[13.3 Simple example 73](#_Toc104619)

[13.4 Spark programming example 74](#_Toc104620)

[13.5 Exercise 75](#_Toc104621)

[13.6 Quizzes 76](#_Toc104622)

[Chapter 14. MapReduce Framework 77](#_Toc104623)

[14.1 Overview 77](#_Toc104624)

[14.2 Theory 77](#_Toc104625)

[14.3 Simple example 77](#_Toc104626)

[14.4 Exercise 82](#_Toc104627)

[14.5 Quizzes 83](#_Toc104628)

[Chapter 15. Spark SQL 84](#_Toc104629)

[15.1 Overview 84](#_Toc104630)

[15.2 Theory 84](#_Toc104631)

[15.3 Simple example 84](#_Toc104632)

[15.4 Spark programming example 87](#_Toc104633)

[15.5 Exercise 93](#_Toc104634)

# Chapter 1. Introduction Machine Learning, Data Mining, and Big Data Analytics

## 1.1 Overview

With the advances of technology, nowadays a lot of data are being generated on the daily basis from all kinds of industrial and academic sources such as e-commerce, traditional merchandize transactions, banks, and social network. The amount of data generated by large companies and organizations are increasing with a fast pace. For example, Google processed 20 PB of data each day back in 2008. Wayback Machine had 3 PB of data back in March 2009 and for just a month it produced 100 TB of data. In April 2009, Facebook already had 2.5 PB of data and it created 15 TB per day; in May 2009, eBay had 6.5 PB of user data and it produced 50 TB per day. There are various types of data generated including Relational, Text, Graph, Streaming, etc.

To analyze these data, we need to apply all sorts of process on them. For example, we use Aggregation and Statistics for Data warehouse and online analytical processing data (OLAP). We perform Indexing, Searching, and Querying for keyword based search and pattern matching. We also conduct Data Mining and Statistical Modeling for Knowledge discovery.

## 1.2 Theory

The reason we need data mining as a tool is that with the generation of huge amount of data of all kinds every day, there is often hidden information in the data, and traditional techniques are not able to detect those information for raw data. With tools provided by data mining, we are able to extract interesting patterns or knowledge from huge amount of data.

Data Mining is Confluence of Multiple Disciplines such as Machine Learning, Pattern Recognition, Statistics, Algorithms, Database, etc. It provides approaches that are highly scalable to handle large-scale data with high dimensionality and complexity. Those approaches can perform both Predictive Tasks such as Classification, Regression and Anomaly Detection, and Descriptive Tasks such as Clustering, Association Rule Discovery and Sequential Pattern Discovery. There is considerable overlap among Artificial Intelligence, Data Mining and Machine Learning, but they are also different in various ways.

Machine Learning provides algorithms that train data and improve the performance by using the knowledge, because it is often too difficult to design a set of rules by hand, and Machine Learning is about automatically extracting relevant information from data and applying it to analyze new data. It can be applied in many fields such as Face Recognition, Speech recognition and Stock prediction.

## 1.3 Simple example

Within Machine Learning, there are various types including Supervised Learning such as Classification and Regression, Unsupervised learning such as Clustering, Feature

Selection/Feature Reduction, Semi-supervised learning, and more. Oftentimes researchers use Machine Learning to minimize a loss or maximize a gain, minimize discrepancies between predictions and real results, and maximize accuracy.

With Machine Learning, we are able to develop strategies to differentiate cats and dogs from images, detect vehicle types from images, perform handwritten digit recognition, predict stock market, conduct image segmentation to break up the image into similar regions, group data sets, estimate gender and age of certain group of people, and much more.

## 1.4 Exercise

Please search online to find more applications of Machine Learning.

## 1.5 Quizzes

Q1: Why do we need data mining tools instead of traditional data analysis approaches?

Q2: What are the examples of Machine Learning algorithms?

# Chapter 2. Hadoop

## 2.1 Overview

Hadoop is an open-source software system that provides a distributed computing environment on cloud (data centers). It is a state-of-the-art environment for Big Data Computing.

Hadoop was created by Doug Cutting and Mike Cafarella in 2005, the creator of Apache Lucene, the widely used text search library. The underlying technology of Hadoop was invented by Google. Hadoop has its origins in Apache Nutch, an open source web search engine. In 2004, Google published the paper that introduced MapReduce to the world. Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year, all the major Nutch algorithms had been ported to run using MapReduce and NDFS. In February 2006, NDFS and the MapReduce moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, which provided dedicated team and the resources to turn Hadoop into a system that ran at web scale. In February 2008, Yahoo! Announced that its production search index was being generated by 10,000-core Hadoop cluster. In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community.

By the same time, Hadoop has been used by many companies besides Yahoo!, such as Last.fm, Facebook, and the New York Times. In April 2008, Hadoop broke a world record to become the fastest system to sort terabyte of data. In November 2009, Google reported that its MapReduce implementation sorted one terabyte in 68 seconds.

## 2.2 Theory

Hadoop contains two key services. One is the reliable data storage using the Hadoop Distributed File System (HDFS), the other is the distributed data processing using a technique called MapReduce.

Besides HDFS and MapReduce, the Hadoop framework also contains Hadoop Common and Hadoop YARN. Hadoop Common, also known as Hadoop Core, refers to the collection of common utilities and libraries that support other Hadoop modules. Like all other modules, Hadoop Common assumes that hardware failures are common and that these should be automatically handled in software by the Hadoop Framework. The Hadoop Common package is considered as the base/core of the framework as it provides essential services and basic processes such as abstraction of the underlying operating system and its file system. Hadoop Common also contains the necessary Java Archive (JAR) files and scripts required to start Hadoop. The Hadoop Common package also provides source code and documentation, as well as a contribution section that includes different projects from the Hadoop Community.

Hadoop YARN is the resource management and job scheduling technology in the open source Hadoop distributed processing framework. One of Apache Hadoop's core components, YARN is responsible for allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes. YARN stands for “Yet Another Resource Negotiator”, but it's commonly referred to by the acronym alone; the full name was self-deprecating humor on the part of its developers. The technology became an Apache Hadoop subproject within the Apache Software Foundation (ASF) in 2012 and was one of the key features added in Hadoop 2.0, which was released for testing that year and became generally available in October 2013.

Hadoop HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware. Some Hadoop clusters stores petabytes of data. HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern.

## 2.3 Simple example

Hadoop doesn’t require expensive, highly reliable hardware to run on. It is designed to run on clusters of commodity hardware. Hadoop was created with the assumptions that Hardware Failure happens all the time. It provides streaming Data Access for very large Data Sets. It was designed to provide portability across heterogeneous hardware and software platforms.

Files in HDFS are broken into block-sized chunks. Each chunk is stored in an independent unit. By default, the size of each block is 64 MB. There are some benefits of splitting files into blocks. First of all, a file can be larger than any single disk in the network. Second, blocks fit well with replication for providing fault tolerance and availability. To insure against corrupted blocks and disk/machine failure, each block is replicated to a small number of physically separate machines.

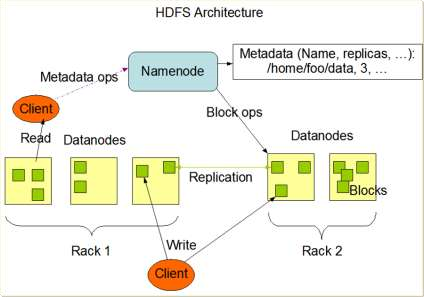
HDFS was designed with a master/slave architecture. It contains two kinds of nodes: namenodes and datanodes.

The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories. It also contains the information on the locations of blocks for a given file.

The namenode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.

Datanodes store blocks of files. They report back to the namenodes periodically. The DataNodes are responsible for serving read and write requests from the file system’s clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the namenode.

Here is a picture of the HDFS architecture:



HDFS provides the Command-Line Interface. It contains commands to perform file operations back and forth between local and HDFS environments. A

For example, in the account of a user called username, to copy a file "source.txt" from local file system to a file "destination.txt" in HDFS, we can perform:

"hadoop fs -copyFromLocal input/docs/source.txt /user/username/destination.txt"

To copy a file "source.txt" from HDFS to a file "destination.txt" in local file system, we can perform:

"hadoop fs -copyToLocal hdfs://localhost/user/username/source.txt destination.txt"

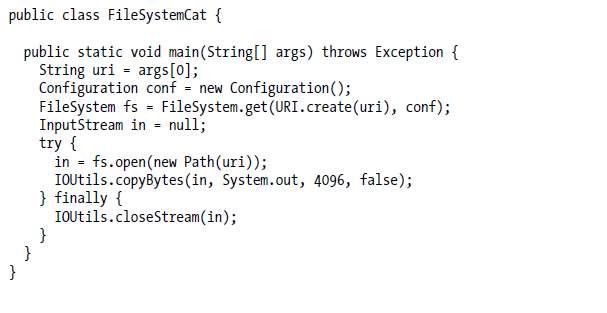
To compare these two local files "file1.txt" and "file2.txt", we can perform:

"MD5 file1.txt file2.txt"

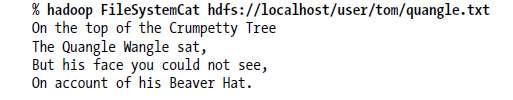
Hadoop file systems are normally implemented in Java. The Java abstract class " org.apache.hadoop.fs.FileSystem" represents a file system in Hadoop. There are several concrete implementation of this abstract class. HDFS is one of them.

Here we will see two examples. One is for how to read data from HDFS in Java programs, and the other is for how to write data to HDFS in Java programs.

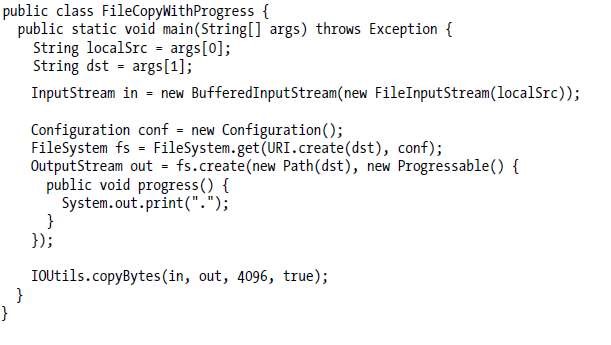
The figure below shows how to read data using the File System API:



The result is shown as:



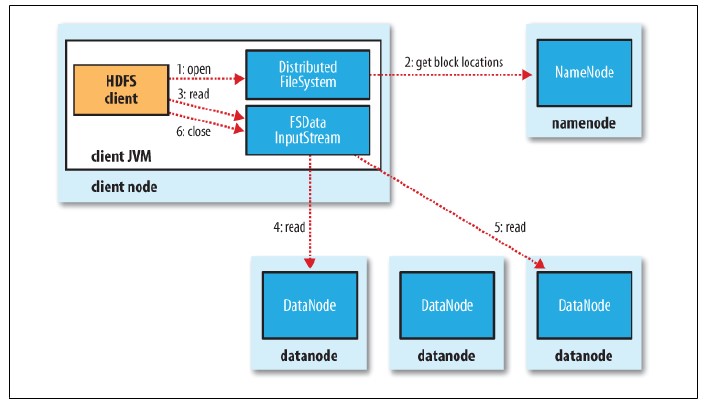
The figure below shows how to write content to a file in HDFS:



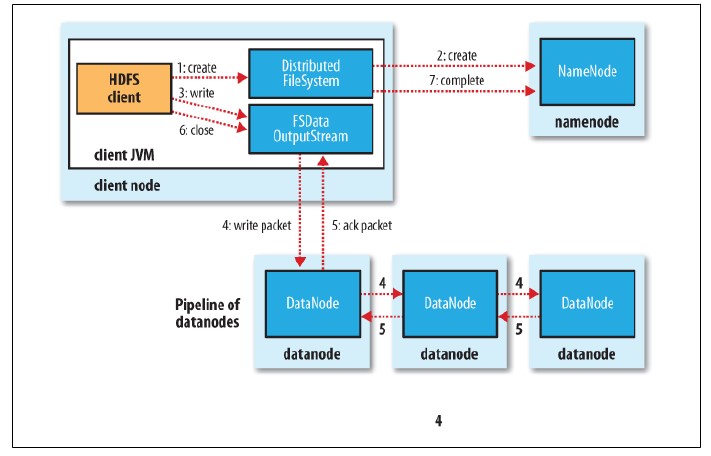
The result is shown as:



The typical data flow for a client to read data from HDFS is shown as:



The typical data flow for a client to write data to HDFS is shown as:



## 2.4 Exercise

Please search online for study more Hadoop HDFS examples.

## 2.5 Quizzes

Q1: What is Hadoop? What modules does Hadoop include?

Q2: What are the advantage of HDFS compared to traditional file systems?

Q3: How are files stored in HDFS?

# Chapter 3. Apache Spark

## 3.1 Overview

Originally developed at UC Berkeley's AMPLab in 2009, Apache Spark is an open-source distributed general-purpose cluster-computing framework, which offers high fault tolerance and high parallelism of data processing.

Spark is designed for fast, interactive, and language-integrated cluster computing. It is to extend the MapReduce model to better support two common classes of analytics apps, which are iterative algorithms in the machine learning field and interactive data mining. It is also integrated into Scala, a modern programming language, and allows interactive use from Scala interpreter.

## 3.2 Theory

Most current cluster programming models are based on data flow from stable storage to stable storage, through map and reduce stages. One drawback of current models is that the data flow is acyclic, which makes it inefficient for applications that repeatedly reuse a working set of data, because they need to load data from stable storage on each query.

One solution to this drawback is the Resilient Distributed Datasets (RDDs) that allow apps to keep working sets in memory for efficient reuse. It retains the attractive properties of MapReduce such as fault tolerance, data locality, and scalability, and also supports a wide range of applications. RDD is an efficient way to manipulate and persist intermediate datasets.

RDD is beneficial to a lot of Machine Learning algorithms that require iterative computation, and it avoids the iterations on MapReduce that cause big overhead between Map and Reduce stages as they are involved with a large amount of data replication, disk I/O and serialization that are computationally expensive, because Hadoop uses HDFS for sharing data. Since HDFS stores files as blocks across hard disks of servers across a cluster, it causes frequent file I/O, which slows down the computational process. One solution is to user RAM instead, to reduce uses of file I/O. Spark programs can be done with Python, Scala, and more.

RDD makes all data in RAM as read-only. It uses a strategy designed by Lineage and Directed Acyclic Graph (DAG), and records all history of the process of the data. When unexpected system fault happens, RDD checks the lineage of the data and easily roll back to achieve fast recovery.

RDD handles immutable and partitioned collections of objects, created through parallel transformations on data in stable storage such as map, filter, groupBy, join, etc. which can be cached for efficient reuse. Actions on RDDs include count, reduce, collect, save, and more. It is all designed to be easy to distribute in a fault-tolerant fashion.

## 3.3 Simple example

Spark provides important features for us to control data processing with better performance. Persistence, Pair RDDs and Partitioning are three of them.

In Spark programming, to use a data set multiple times without reloading them each time, we can tell Spark to persist the data, specifying different levels of persistence:

MEMORY\_ONLY, MEMORY\_ONLY\_SER, MEMORY\_AND\_DISK, MEMORY\_AND\_DISK\_SER, or DISK\_ONLY.

Spark provides special operations on RDDs that contain key/value pairs, implemented as simply tuples in Python, Scala, and Java. There are various pair RDD transformations. For example, reduceByKey(func) reduces values using func, but on a key by key basis. Transformation groupByKey() combines values with same key. Each key ends up with a list. Transformation sortByKey() returns an RDD sorted by key. Transformation mapValues(func) uses func to change values, but not key. Transformation keys() returns an RDD of only keys. Transformation values() returns an RDD of only values.

Spark distributes the data of RDDs across its resources, using key/value pairs to group data efficiently. It also allows us to create custom partitioners that outperform the default one.

Spark has two parallel programming features that make it easier and more efficient to do operations in parallel in a more explicit way: Accumulators and Broadcast variables. Accumulators are variables that allow many copies of a variable to exist on the separate worker nodes. Broadcast variables are sent by Spark to all workers so they can all access the variables.

There are numerous transformations in Spark. For example, map(func) returns a new RDD by passing each element through func. Transformation filter(func) returns a new RDD by selecting the elements for which func returns true. In flatMap(func), func can return multiple items, and generate a sequence, allowing us to “flatten” nested entries (JSON) into a list. Transformation distinct() returns an RDD with only distinct entries.

Transformation sample(…) provides various options to create a subset of the RDD. Transformation union(RDD) returns a union of the RDDs. Transformation intersection(RDD) returns an intersection of the RDDs. Transformation subtract(RDD) removes argument RDD from other. Transformation cartesian(RDD) provides cartesian product of the RDDs. parallelize(list) creates an RDD from this (Python) list (using a spark context).

There are also a lot of actions defined in Spark. For example, collect() returns all the elements from the RDD. count() returns the number of elements in RDD. Action countByValue() provides the list of times each value occurs in the RDD. Action reduce(func) aggregates the elements of the RDD by providing a function which combines any two into one (sum, min, max, …). Actions first(), take(n) return the first, or first n elements. Action top(n) returns the n highest valued elements of the RDDs. Action takeSample(…) provides various options to return a subset of the RDD. Action saveAsTextFile(path) writes the elements as a text file. Action foreach(func) runs the func on each element. Used for side-effects (updating accumulator variables) or interacting with external systems.

The complete list of transformations and actions can be found at the official website http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.

## 3.4 Spark programming example

Here is an example of log mining which loads error messages from a log into memory, then interactively search for various patterns. The key idea is to add “variables” to the “functions” in functional programming:

lines = spark.textFile(“hdfs://...”) errors = lines.filter(\_.startsWith(“ERROR”)) messages = errors.map(\_.split(‘\t’)(2)) cachedMsgs = messages.cache() cachedMsgs.filter(\_.contains(“foo”)).count cachedMsgs.filter(\_.contains(“bar”)).count

In this example we can see that RDDs maintains lineage information that can be used to reconstruct lost partitions. The statement

messages = textFile(...).filter(\_.startsWith(“ERROR”)).map(\_.split(‘\t’)(2)) shows a chained process of pattern reconstruction:



There are some examples of applications with Spark:

Example 1, Word Count

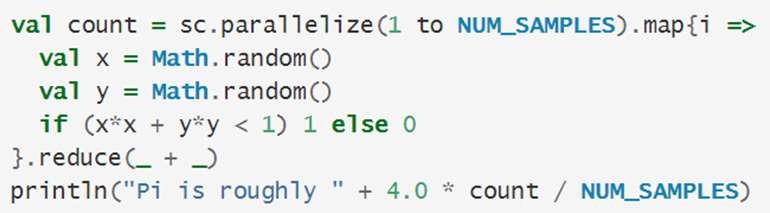
This example uses a few transformations to build a dataset of (String, int) pairs called counts and then save it to a file.



In this example, the map operation produces one output value for each input value, whereas the flatMap operation produces an arbitrary number (zero or more) values for each input value.

Example 2, Pi estimation

This example estimates π by "throwing darts" at a circle. We pick random points in the unit square ((0, 0) to (1,1)) and see how many fall in the unit circle. The fraction should be π / 4, so we use this to get our estimate.



Spark can be applied in a wide range of fields, including in-memory data mining on Hive data, predictive analytics, city traffic prediction, Twitter spam classification, collaborative filtering via matrix factorization, and more.

With modified wrapper code generation so that each line typed has references to objects for its dependencies, and distribute generated classes over the network, Spark interpreter can be modified to allow Spark to be used interactively from the command line. Spark provides a simple, efficient, and powerful programming model for a wide range of apps.

## 3.5 Exercise

Please search online for more Spark examples.

## 3.6 Quizzes

Q1: What is Spark?

Q2: What is the differences between Spark and Hadoop? Why is Spark preferred than Hadoop?

Q3: What are the transformations available in Spark?

# Chapter 4. Decision trees

## 4.1 Overview

A decision tree is a natural and simple way of inducing following kind of rules. It is powerful and perhaps most widely used modeling technique of all. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

## 4.2 Theory

There are a few terminology related to decision trees. First of all, features describe individual measurable properties being observed. When we use a decision tree, we first need to decide the number of features or distinct traits that can be used to describe each item in a quantitative manner. The term samples represents the Items to process. They can be a document, a picture, a sound, a video, or a patient. Feature vector is used to define an N-dimensional vector of numerical features that represent some objects. Each sample object has the same feature vector. Feature extraction is also called feature selection, and it is the process to transform the data in the high-dimensional space to a space of fewer dimensions.

In Machine Learning, the input Xi is normally represented as an input vector, which is an independent variable. The output Y is called response variable (dependent variable), used to predict a label when having observed some new 𝑥.

## 4.3 Simple example

Various type of variables can be used in a decision tree. Categorical variables are discrete or qualitative variables. They can be nominal, which means they can have two or more categories, which do not have an intrinsic order. They can also be dichotomous, meaning they have only two categories or levels. They can also be ordinal, with two or more categories, which can be ordered or ranked. Opposite to categorical variables, there are also continuous variables.

With large search space caused by the big size of the input data, finding the global optimal decision tree is computationally infeasible. We need efficient feasible algorithms that might not be optimal, but provide us with approximate answers. Normally we use greedy strategy, and grow the tree by making locally optimal decisions in selecting attributes.

There are numerous Decision Tree Algorithms including Hunt’s algorithm, CART, ID3, C4.5, SLIQ, and SPRINT.

Hunt’s algorithm is one of the earliest algorithms for decision tree and most decision tree induction algorithms nowadays are based on Hunt’s algorithm.

Let be the set of training data and 𝒚 be class labels, 𝒚={ 𝑦1 , 𝑦2 ,…, 𝑦c }. We set the following rules:

If contains data that belong to 𝑦k , its decision tree consists of leaf node labeled as 𝑦k If is an empty set, the decision tree is a leaf node of default class

If contains data that belong to more than one classes, perform “attribute test” to split the data into smaller and more homogenous subsets

We use the greedy strategy to split the data based on the attribute test that locally optimizes certain criterion. To determine how to split the data, we need to decide how to specify the attribute test condition and how to determine the best split. Also we need to consider when to stop splitting.

Since there are different types of input data, we use different strategies.

For nominal attributes we can either use multi-way split that uses a many partitions as distinct values, or binary split that divides values into two subsets.

For ordinal attributes, we can also either use multi-way split that uses a many partitions as distinct values, or binary split that divides values into two subsets.

For continuous attributes, we can either perform discretization to convert continuous to an ordinal categorical attribute, then do a multi-way split, or apply the binary decision that split the input data into two, and consider all possible splits and finds the best cut.

In order to find the best split, we need to find nodes with homogeneous class distribution, with the measure of node impurity. There are various ways to measure node impurity. Gini Index is the most commonly used to measure inequality of income or wealth, using a value between zero and one, where one expresses maximal inequality. Entropy is also often used, as well as misclassification error.

Let's first look at GINI Index. For a given node t:

𝐺𝐼𝑁𝐼(𝑡) = 1 − ∑[𝑝(𝑗|𝑡)]2

𝑗

in which 𝑝(𝑗|𝑡) is a conditional probability, but can be measured by the relative

frequency of class j at node t. The value of GINI(t) is between 0 when all data belong to one class only, and 1 when all data are equally distributed among all classes.

GINI index is used in CART, SLIQ and SPRINT. When a node p is split into k partitions, the quality of split is computed as,

𝑘

𝑛𝑖

𝐺𝐼𝑁𝐼𝑠𝑝𝑙𝑖𝑡 = ∑ 𝐺𝐼𝑁𝐼(𝑖)

𝑛

𝑖=1

where 𝑖 is the number of data at child i, and 𝑛 is the number of data at node p.

GINI index can be applied on binary attributes where the data are split into two partitions. Larger and Purer partitions are sought for. When it is applied on categorical attributes, for each distinct value, we count for each class, and use the count matrix to make decision.

When GINI index is applied on continuous attributes, we can either use binary decisions based on one value, with several Choices for the splitting value, and build a count matrix for each splitting value, then compute its Gini index each; or we can sort the attribute and linearly scan these values and compute gini index, choosing the split cut that has the least gini index.

Entropy strategy is adapted from a thermodynamic system that provide measure of molecular disorder within a macroscopic system. Entropy is zero when a outcome is certain.

𝐸𝑛𝑡𝑟𝑜𝑝𝑦(𝑡) = − ∑ 𝑝(𝑗|𝑡) log 𝑝(𝑗|𝑡)

𝑗

Its computations are similar to the GINI index.

Splitting can be based on various criteria. It can be based on Information Gain which is the expected reduction in entropy caused by partitioning.

𝑘

𝑛𝑖

𝐺𝐴𝐼𝑁𝑠𝑝𝑙𝑖𝑡 = 𝐸𝑛𝑡𝑟𝑜𝑝𝑦(𝑝) − (∑ 𝐸𝑛𝑡𝑟𝑜𝑝𝑦(𝑖))

𝑛

𝑖=1

Which calculates the difference between the entropy of the original collection and the expected value of the entropy after S is partitioned using attribute i. We then choose the split so that maximize gain. One of the drawbacks of this approach is that it tends to splits that result in large number of partitions, each being small but pure.

Splitting can also be based on information theory and Classification Error. The splitting is stopped when all data belong to the same class, or when all data have same (or similar) attribute values.

Decision Tree is inexpensive to construct, extremely fast at classifying new data, and easy to interpret the decision process. However, it has overfitting problem. Also, it does not perform well when it trains data with missing values.

Here is an example of how decision tree is implemented in Spark using Java.

import org.apache.spark.ml.Pipeline;

import org.apache.spark.ml.PipelineModel; import org.apache.spark.ml.PipelineStage; import org.apache.spark.ml.classification.DecisionTreeClassifier; import org.apache.spark.ml.classification.DecisionTreeClassificationModel; import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator; import org.apache.spark.ml.feature.\*; import org.apache.spark.mllib.regression.LabeledPoint; import org.apache.spark.mllib.util.MLUtils; import org.apache.spark.rdd.RDD; import org.apache.spark.sql.DataFrame;

// Load and parse the data file, converting it to a DataFrame.

RDD<LabeledPoint> rdd = MLUtils.loadLibSVMFile(sc.sc(), "data/mllib/sample\_libsvm\_data.txt");

DataFrame data = jsql.createDataFrame(rdd, LabeledPoint.class);

// Index labels, adding metadata to the label column.

// Fit on whole dataset to include all labels in index.

StringIndexerModel labelIndexer = new StringIndexer()

.setInputCol("label")

.setOutputCol("indexedLabel")

.fit(data);

// Automatically identify categorical features, and index them.

VectorIndexerModel featureIndexer = new VectorIndexer()

.setInputCol("features")

.setOutputCol("indexedFeatures")

.setMaxCategories(4) // features with > 4 distinct values are treated as continuous .fit(data);

// Split the data into training and test sets (30% held out for testing)

DataFrame[] splits = data.randomSplit(new double[] {0.7, 0.3});

DataFrame trainingData = splits[0];

DataFrame testData = splits[1];

// Train a DecisionTree model.

DecisionTreeClassifier dt = new DecisionTreeClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("indexedFeatures");

// Convert indexed labels back to original labels.

IndexToString labelConverter = new IndexToString()

.setInputCol("prediction")

.setOutputCol("predictedLabel") .setLabels(labelIndexer.labels());

// Chain indexers and tree in a Pipeline

Pipeline pipeline = new Pipeline()

.setStages(new PipelineStage[] {labelIndexer, featureIndexer, dt, labelConverter});

// Train model. This also runs the indexers.

PipelineModel model = pipeline.fit(trainingData);

// Make predictions.

DataFrame predictions = model.transform(testData);

// Select example rows to display.

predictions.select("predictedLabel", "label", "features").show(5);

// Select (prediction, true label) and compute test error

MulticlassClassificationEvaluator evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("indexedLabel")

.setPredictionCol("prediction") .setMetricName("precision"); double accuracy = evaluator.evaluate(predictions);

System.out.println("Test Error = " + (1.0 - accuracy));

DecisionTreeClassificationModel treeModel =

(DecisionTreeClassificationModel)(model.stages()[2]);

System.out.println("Learned classification tree model:\n" + treeModel.toDebugString());

## 4.4 Exercise

Please conduct research on how decision tree is implemented in Spark.

## 4.5 Quizzes

Q1: What is decision tree?

Q2: In what fields is decision tree applied to solve problems?

Q3: What are examples of decision tree algorithms?

# Chapter 5. Linear models

## 5.1 Overview

A linear model is an equation that describes a relationship between two quantities that show a constant rate of change. In Mathematics, we can describe the two quantities in terms of variable 𝑥 and variable 𝑦 by 𝑦 = 𝑓(𝑥), read as 𝑦 is a function of 𝑥. Since the relation is linear and 𝑓(𝑥) is a univariate polynomial, the linear simply means the degree of 𝑓(𝑥) is zero or one. If the degree of 𝑓(𝑥) is zero, the rate of change is zero and 𝑦 is a constant, disregard what value

𝑥 is.

The linear model is used widely in predictive analysis lately. In the function 𝑦 = 𝑓(𝑥), 𝑦 is called dependent variable or target and 𝑥 is called independent variable or predictor. If the function can be best used to describe two quantities, it simply tells about the relationship between a target that is dependent variable and a predictor using a straight line.

## 5.2 Theory

A Generalized Linear Model (GLM) consists of three components: a random component, a systematic component, and a link function. In linear model, communication is considered one way process where sender is the only one who sends message and receiver doesn't give feedback or response. The message signal is encoded and transmitted through channel in presence of noise. The sender is more prominent in linear model of communication.

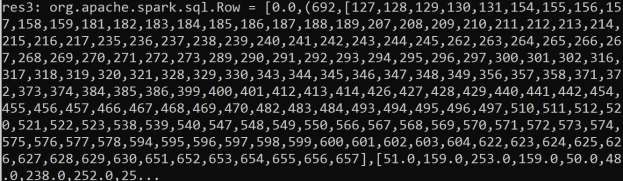
## 5.3 Simple example

A linear model is a comparison of two values, usually x and y, and the consistent change between the values. In the opening story, Jill was analyzing two values: the amount of electricity used and the total cost of her bill. The change between these two values is the cost of each kilowatt hour.

## 5.4 Spark programming example

Spark (v3.0.1) supports several linear methods including linear support vector machines, logistic regression, and generalized linear methods. The loss functions include hinge loss, logistic loss, and square loss. To regulate the model complexity, three regularizations are supported: Lasso, Ridge, and Elastic Net.

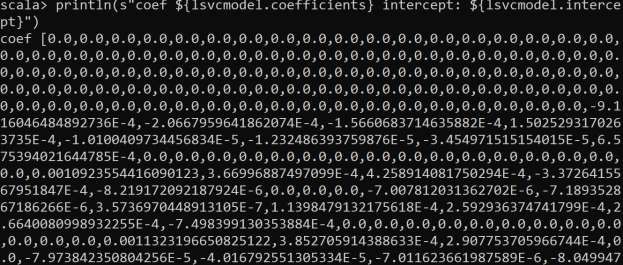
Here we are trying to analyze using linear support vector machines for the sample dataset that contains 692 features. The following shows the first record in sparse vector notation with the label 0.0 and features in sparse vector format. Note that value for the 127th feature is 51.0 and the 128th is 159.0, etc.



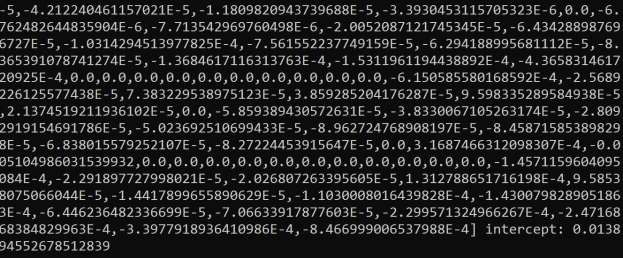
The following Scala code demonstrates applying the linear support vector machines on the dataset.

|  |
| --- |
| // import required Spark ML library module LinearSVC import org.apache.spark.ml.classification.LinearSVC  // Load the training data sample\_libsvm\_data.txt from the given folder; this file come s with the Spark installation. You need to change directory to the location where you install Spark. For example, if you install Spark in c:/spark, you may change directory to c:/spark and start spark-shell to write your program. Otherwise, there will be a f ile NOT FOUND error.  val training = spark.read.format("libsvm").load("data/mllib/sample\_libsvm\_data.txt")  // After loading the data to Spark, we will instantiate an object for LinearSVC(). To simplify the program, we just use default setting for the object.  val lsvc = new LinearSVC()  // Fit the model; The method fit() simply computes parameters for the model. Since thi s is a linear model, we have one Y-intercept and one coefficient for each feature.  val lsvcModel = lsvc.fit(training)  // Because we have 692 features, there are 692 coefficients and one Y-intercept for li near svc. We may print them out using the following statement.  println(s"Coefficients: ${lsvcModel.coefficients} Intercept: ${lsvcModel.intercept}") |

Here is the outputs for the coefficients and the Y-intercept.



…



The Y-intercept is 0.013894552678512839.

## 5.5 Exercise

Redo the Sparking programming in Section 5.4 with Lasso, Ridge, and Elastic Net regularizations.

## 5.6 Quizzes

Explain parameters used in linear models.

# Chapter 6. Linear regression

## 6.1 Overview

Regression models relation between variables where changes in some variables may “explain” or possibly “cause” changes in other variables. Explanatory variables are termed the independent variables and the variables to be explained or predicted are termed the dependent variables. Figure 6.1 shows the relation between GPA and the number of hours study weekly.

0

0.5

1

1.5

2

2.5

3

3.5

4

4.5

0

5

10

15

20

25

30

GPA

# of Hours Study Weekly

GPA vs. # of Hours Study Weekly

*Figure 6.1 The Relation between GPA and the number of Hours Study Weekly*

It can be obviously seen that the tendency of the data shows the more number of study hours weekly, the higher GPA a student will receive. You may look at the figure 10 feet away to get a macro view. This type of relation is termed positive relation. Two variables have a positive relation when above-average values of one tend to accompany above-average values of the other, and when below-average values also tend to occur together.

Another example, illustrated in Figure 6.2, shows the relation between GPA and the number of times absent. For a macro view, you will find the tendency is that the GPA is decreasing while the number of times absent is increasing. This type of relation is termed negative relation. Two variables have a negative relation when above-average values of one tend to accompany below-average values of the other.

0

0.5

1

1.5

2

2.5

3

3.5

0

5

10

15

20

25

30

35

40

GPA

# of Times Absent

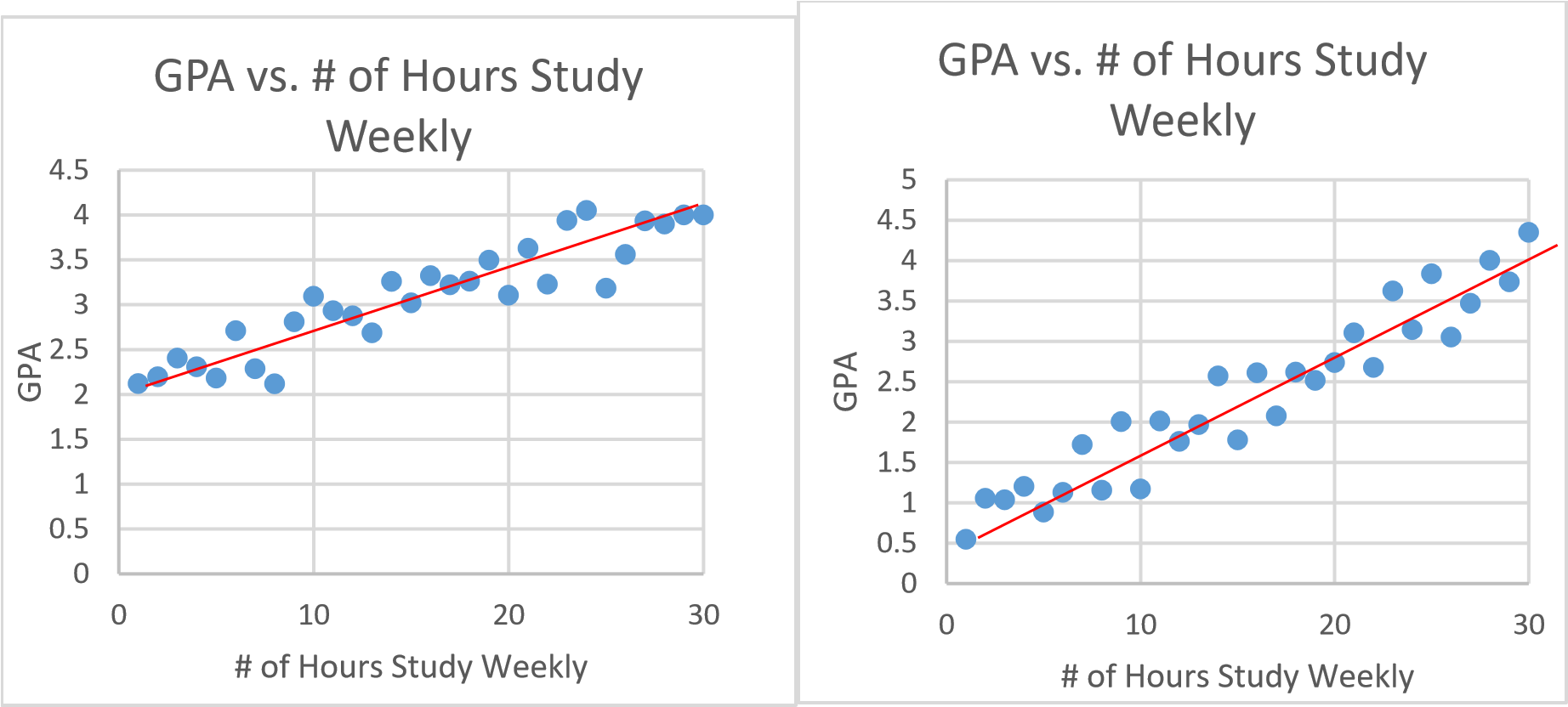
GPA vs. # of Times Absent

*Figure 6.2 The Relation between GPA and the Number of Times Absent*

The about two figures (Figure 6.1 and Figure 6.2) have a clear tendency, either positive or negative relation. The following figure shows an example that cannot be clearly identified neither. Figure 6.3 illustrates the relation between GPA and weight of a student. Obviously, we cannot tell if the relation is positive or negative. It could be non-linear, i.e., a relation cannot be described by a line. Anyhow, we can draw a conclusion if it is positive relation or negative relation. We will study more on non-linear relation later. However, we cannot draw a conclusion that GPA has no relation to a student’s weight. They may be related to each other indirectly but we just don’t have data to show. It might make sense from medical or healthy perspective that an overweight student could sleep more and thus has less time to study. In such scenario, the less time to study will result in lower GPA. Practically, to find out the relation between GPA and weight will require to collect more data with regard to potential variables that will be changed. For example, each student will be asked GPA, the number of hours study weekly, and weight in data collection.



A regression model estimates the nature of the relationship between the independent and dependent variables. Change in dependent variables that results from changes in independent variables, i.e. the number hours worked weekly of the relationship.



*Figure 6.4 The Strength of a Relation*

In Figure 6.4, the slope on the left figure is smaller than that of the right figure, if we are using the linear regression to draw a line that best represents the data. The GPA is more sensitive to the change of the number of hours study weekly in the right figure. The left figure, however, the GPA is smother with respect to the change of the number hours study weekly due to the smaller slop of the line.

A correlation coefficient measures the strength of that relationship. The Pearson correlation coefficient is defined as follow:

1 𝑥𝑖 − 𝑥̅ 𝑦𝑖 − 𝑦̅

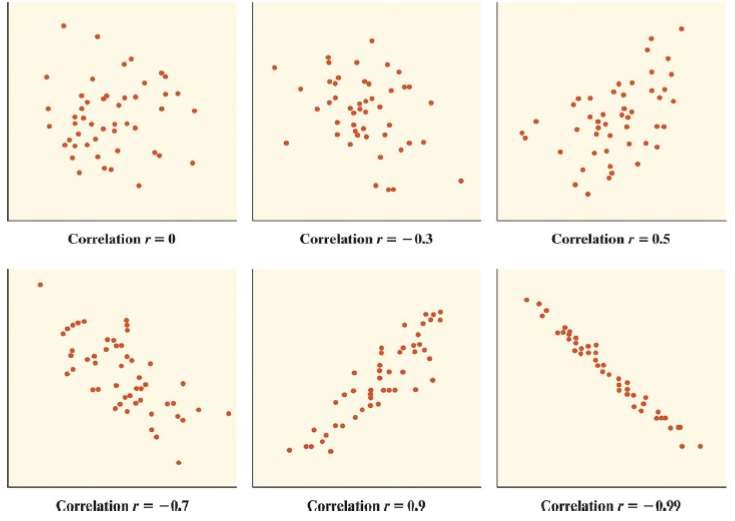
𝑟∑( )( )

 − 1 𝑆𝑥 𝑆𝑦

where the 𝑆𝑥 and 𝑆𝑦 are standard sample variance for each quantity respectively, defined as follows:

𝑆𝑥 = √∑(𝑥𝑖 − 𝑥̅)2/(𝑛 − 1) and 𝑆𝑦 = √∑(𝑦𝑖 − 𝑦̅)2/(𝑛 − 1).

Calculating a Pearson correlation coefficient requires the assumption that the relationship between the two variables is linear. The relationship between two variables is generally considered strong when their r value is larger than 0.7.



*Figure 6.5 Examples of Scatterplots with Different Correlation Coefficients*

The statistical significance indicates that changes in the independent variables correlate with shifts in the dependent variable. Correspondingly, the good R-squared value signifies that your model explains a good proportion of the variability in the dependent variable.

* Correlation coefficients have a probability (p-value), which shows the probability that the relationship between the two variables is equal to zero (null hypotheses; no relationship).
* Strong correlations have low p-values because the probability that they have no relationship is very low.
* Correlations are typically considered statistically significant if the p-value is lower than 0.05 in the social sciences, but the researcher has the liberty to decide the p-value for which he or she will consider the relationship to be significant.

## 6.2 Theory

## 6.3 Simple example

Consider oil refining process. In the first step of the refining process, crude oil is heated in a furnace until most of it vaporizes into a gas. The liquids and vapors then enter an atmospheric distillation tower, which separates the liquids and vapors into different streams, or fractions, based on differences in boiling points. Based on weight of streams, several end products are generated from the distillation tower such as liquid propane gas (LPG), gasoline, jet fuel, diesel fuel, LPG gasoline, industrial fuel, and asphalt base (form the top to bottom). In addition to the refining cost and others (shipping, packaging, etc.), gasoline retail price has something to do with the crude oil price.

To model the retail price of gasoline in Georgia with respect to the price of crude oil, we could observe two quantities, the retail prices and the price of crude oil. If we are going to predict gasoline retail price in Georgia, the dependent variable is the retail price of gasoline whereas independent variable is the price of crude oil. We could, however, exchange the role of the two quantities, if we are predicting the price of the crude oil based on the retail price of gasoline. In that case, the dependent variable is the prices of the crude oil and the independent variable is the retail price of gasoline.

Another example, if we are going to predict employment income, the dependent variable is employment income whereas independent variables might include hours of work, education, occupation, sex, age, region, years of experience, unionization status, etc.

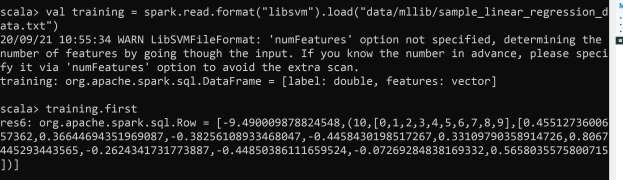
In most of the cases, each quantity could be either a dependent variable or an independent variable. For example, the price of a product has some relation to the quantity produced or sold and vice versa. Quantity sold is affected by price. Dependent variable is quantity of product sold – independent variable is price. On the other hand, price is affected by quantity offered for sale. In this case, dependent variable is price – independent variable is quantity sold. In either case, the dependent variable is the one that we are going to predict or a target in machine learning.

## 6.4 Spark programming example

Like any Spark Scala programming, we first import Spark LinearRegression library.



Then, load the data file sample\_linear\_regression\_data.txt, stored under the c:/spark/data/mslib directory.



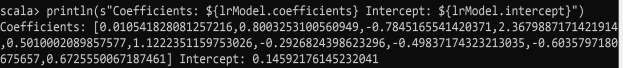
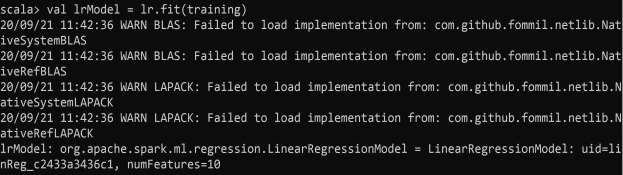
After loading the dataset, do the following:

* Create a LinearRegression model and set parameters  Set regulator parameter (𝜆) for L1 or L2 regulation.
* Set Elastic Net Parameter to 1 for L1 and 0 for L2 regulation.  Any number between 0 and 1 for Elastic Net Regulation.

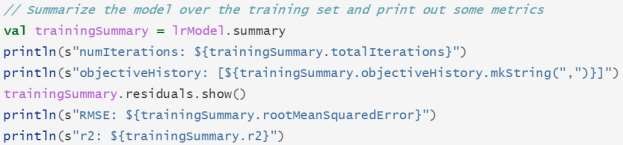


Here we create a linear regression model with the following settings:

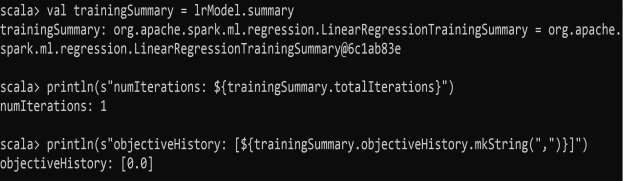
* Set L2 parameter (𝜆) to 0.3
* Choose L2 regulation by setting Elastic Net parameter to 0



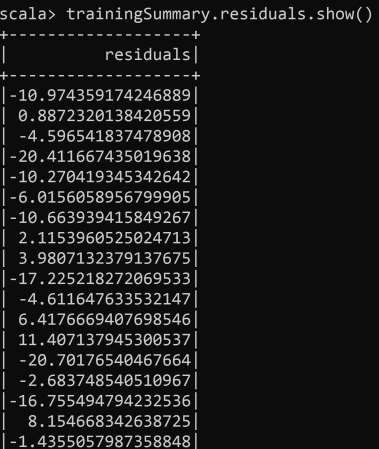
Show performance metrics over the training data such as total # of iterations, residuals, RMSE, R2, etc.



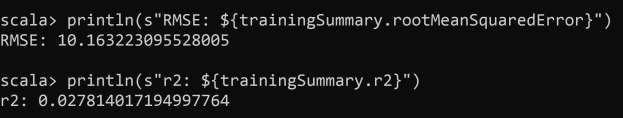
Use lrModel.summary to display performance metrics.



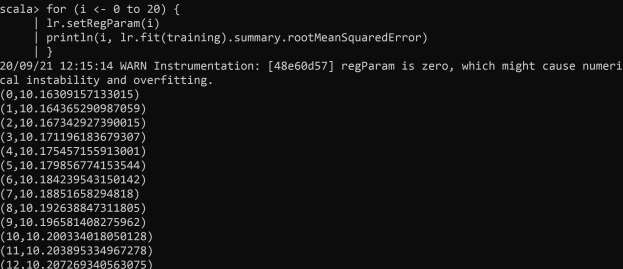
We can display residuals (𝑦𝑖 − 𝑦̂𝑖) to check how good the linear regression predicts.



Print RMSE and R2



We can iterate over different regulation parameter and find out the best result in terms of rootMeanSquaredError.



## 6.5 Exercise

Experiment other loss function settings in the programming example in Section 6.4 and find the best loss function.

## 6.6 Quizzes

Explain what is regression? What is linear regression?

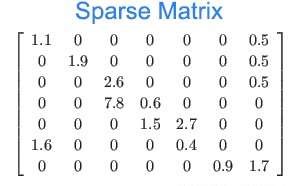
# Chapter 7. Regularization and Sparsity on Linear regression

## 7.1 Overview

What is regularization? Regularization is a technique to minimize the complexity of the model by penalizing the loss function to solve overfitting. This definition involves the following characteristics:

* Make the parameters (weights) as small as possible.
* Minimize the complexity: remove terms by in objective functions, i.e., reduce dimensions of data
* Penalize parameters in the objective function: by setting a regularizer (𝜆)
* Solve the overfitting problem: add a bias intentionally. So machine would learn too much from the training data (e.g., noise), which results in poor performance in testing data.

In Linear algebra, a sparse matrix is one that contains lot of zeros like the following:



In linear regression, we want to turn 𝛽̂ into sparse a matrix (vector) for a number of reasons:

* + Not all features are equally corrected, or unrelated at all.
  + Zero out some 𝛽̂𝑖 means remove them from the model. So simplify the model.
  + Solve overfitting problem

## 7.2 Theory

Let’s start with linear Regression using Least Squares of Residuals for its loss function. To simplify the problem, we consider a dataset with two data samples.

|  |  |
| --- | --- |
| Weekly Study Hours | GPA |
| 5 | 3 |
| 10 | 4 |

The derived linear regression model is simply the line that passes through the two data points for the reason that the line will give a zero of SSR.

y = 0.2x + 2

0

1

2

3

4

5

0

2

4

6

8

10

12

GPA

Weekly Study Hours

Now if we have a test data (15, 3.5), the predicted value is 5 and SSR is 1.5.

y = 0.05x + 3

0

1

2

3

4

5

0

5

10

15

20

GPA

However, if we consider the red dotted line with slope 0.05 (smaller than 0.2), the predicted value is 3.75 and the SSR is 0.25. Obviously, the SSR is much smaller than the original regression.

The first regression line trained from 2 data points is overfitting to the training data. So the performance is obvious very bad when predicting the test data. To overcome this problem, we add a penalty to the objective function. This mechanism is called regularization.

Ridge regularization (L2): We add a regularization term to the objective function in linear regression model as follows:

* 𝑓
* Since we are minimizing the objective function, the added term 𝜆𝛽12 tends to get 𝛽1 smaller, i.e., the slope of the regression line.
* What’s next is estimate the parameters 𝛽0, 𝛽1 again using differentiation.

To find the parameters, we take the first order partial derivative with respective to each parameter and set the derivation to zero. Then solve the parameter accordingly.

* 𝑓
* 𝜕𝑓

(

𝛽

0

,

𝛽

1

)

0

=

−

2

∑

(

𝑦

𝑖

−

𝛽

0

−

𝛽

1

𝑥

𝑖

)

=

0

𝜕𝛽

* 𝑛𝑦̅ − 𝑛𝛽0 − 𝑛𝛽1𝑥̅ = 0 ⇒ 𝛽0 = 𝑦̅ − 𝛽1𝑥̅
* Now partial differentiate on 𝛽1
* 𝜕𝑓

(

𝛽

0

,

𝛽

1

)

1

=

−

2

∑

(

𝑥

𝑖

(

𝑦

𝑖

−

𝛽

0

−

𝛽

1

𝑥

𝑖

)

)

+

2

𝜆

𝛽

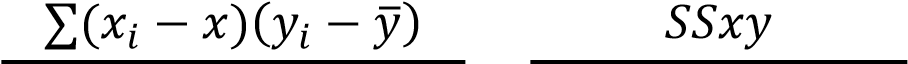
1

=

0

𝜕𝛽

* ̅ ̅
* 𝛽 ̅

̅

𝛽 

̅ ̅ 𝑆𝑆𝑥𝑥 

Let 𝜆 =

12. The red line shows that slope is 0.25 and leans toward the regression line with all three data points.

y = 0.25x + 3

R² = 0.25

0

1

2

3

4

5

5

10

15

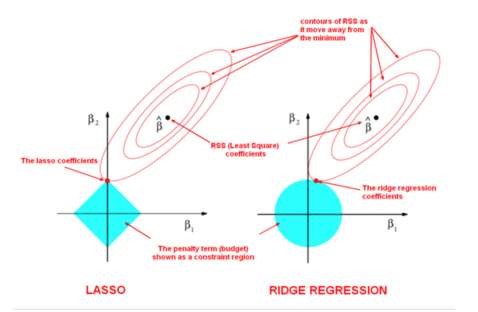
GPA

Least Absolute Shrinkage and Selection Operator (Lasso, L1) Regularization takes a different aspect to reduce the model complexity.

* L1 penalizes sum of abs(weights)
* 𝜆𝑅(𝑤) = 𝜆||𝑤||1 where ||𝑤|||𝑤𝑖|,  and 𝑤𝑖 
* It is a convex problem.
* L1 regularization serves as an approximation to L0, but has the advantage of being convex and thus efficient to compute.
* Therefore, we can use L1 regularization to encourage many of the uninformative coefficients in our model to be exactly 0, and thus reap RAM savings at inference time.
* Encourage sparsity unlike L2

With some constraints, we may plot L1 and L2 in 2 dimensional coordinates:

* For L1, the penalty can be written
* ||𝛽||1 ≤ 𝑐 ⇒ |𝛽1| + |𝛽2| ≤ 𝑐
* For L2, the penalty can be written
* ||𝛽|| 𝑐 ⇒ 𝛽12 + 𝛽22 ≤ 𝑐



Due to the constraints, L1 shows a diamond shape whereas L2 shows a circle.

Below we compare L1 and L2:

* L2 and L1 penalize weights differently:
* L2 penalizes weight
* L1 penalizes |weight|
* L2 and L1 have different derivatives
* The derivative of L2 is 2\*weight
* The derivative of L1 is k (a constant whose value is independent of weight)
* L2 removes x% of the weigh every time. It does not drive weights to zero.
* L1 subtracts some constant from the weight every time. L1 will set the weight to zero.

Practically, we may combine both L1 and L2 in an objective function and controls how much

each contributes to the objective function. This combination forms the Elastic Net Regularization.

* We may combine L1 and L2 in the penalty term.
* 𝜆𝑅(𝑤) = 𝛼𝜆1||𝑤||1 + (1 − 𝛼) 𝜆||𝑤||22, 𝛼 ∈ [0,1], 𝜆1, 𝜆2 ≥ 0
* By setting α properly, elastic net contains both L1 and L2 regularization as special cases.
* For example, if a linear regression model is trained with the elastic net parameter α set to 1, it is equivalent to a Lasso model.
* On the other hand, if α is set to 0, the trained model reduces to a ridge regression model.

## 7.3 Spark programming example

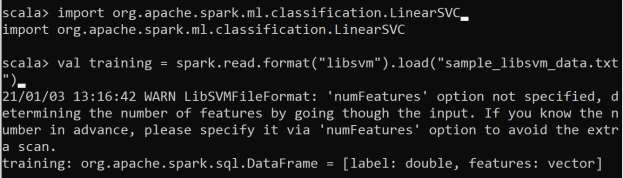
We illustrate a programming example on a liner support vector machine classifier with the ridge regularization.

|  |
| --- |
| // Import the required library module LinearSVC. import org.apache.spark.ml.classification.LinearSVC  // Load training dataset that comes with the Spark installation. We install Spark on c:/spark. So the dataset file is located at c:/spark/data/mllib/. You may change direc tory to c:/spark/data/mllib and run spark-shell there.  val training = spark.read.format("libsvm").load("sample\_libsvm\_data.txt")  // Instantiate an object for LinearSVC(). This is what’s called model.  val lsvc = new LinearSVC()  // Set the Lambda value for the ridge regularization. The Lambda value should be large r than zero. If it is set to zero, that means no regularization is applied.  // We set the lambda (regularization parameter) to 0.2 here.  lsvc.setRegParam(0.2)  // Fit the model; train the model to determine its parameters.  val lsvcModel = lsvc.fit(training) |

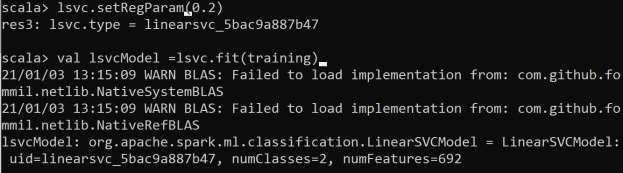
// After training, we may print the coefficients and intercept for linear svc.

println(s"Coefficients: ${lsvcModel.coefficients} Intercept: ${lsvcModel.intercept}")

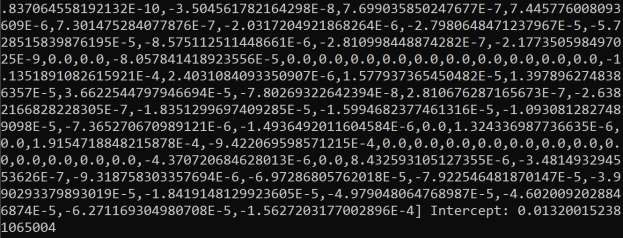
First, import the library and load the dataset.



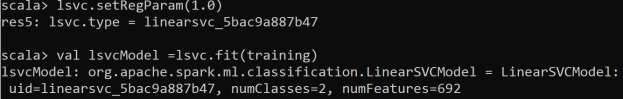
Then, set the regularization parameter to 0.2 for the ridge regularization, followed by running fit() method to train the model.



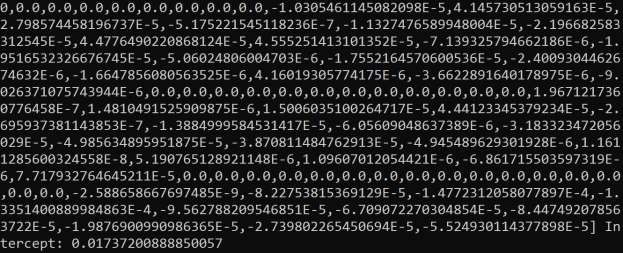
If everything runs ok, we may print out the parameters and the Y-intercept. The following screenshots only show the bottom of the output.



We can see the Y-intercept is 0.013200152381065004. We may change the regularization parameter to 1.0 and check the result.



The following result shows that the parameters have changed as well. The Y-intercept changes to 0.01737200888850057.



## 7.4 Exercise

Write a program for linear regression and set the regularization to Lasso or Elastic Net.

## 7.5 Quizzes

Which of the following statements are true?

1. L1 regularization will encourage many of the non-informative weights to be nearly (but not exactly) 0.0.
2. L1 regularization may cause informative features to get a weigh of exactly 0.0.
3. L1 regularization will encourage most of the non-informative weights to be exactly 0.0.

# Chapter 8. Logistic regression

## 8.1 Overview

Logistic Regression is one of the most commonly used Machine Learning algorithms that is used to model a binary variable that takes only 2 values – 0 and 1. The objective of Logistic

Regression is to develop a mathematical equation that can give us a score in the range of 0 to 1. This score gives us the probability of the variable taking the value 1. Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value. This value can be thought of as a probability. If the value is larger than 0.5, the predicted value is 1 (positive class). Otherwise, it is predicted to 0 (negative class).

## 8.2 Theory

Logistic regression uses an equation as the representation, very much like linear regression.

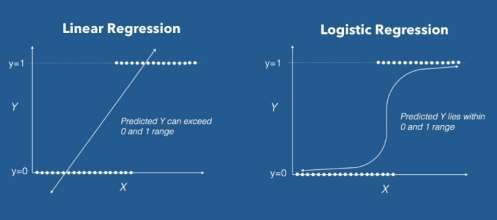
Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value. Assume the linear part is modeled as 𝑏0 + 𝑏1𝑥. The following equation describes the output of the logistic regression.

𝑒𝑏0+𝑏1𝑥

𝑦 = 1 + 𝑒𝑏0+𝑏1𝑥

In the above equation, y is the predicted output, 𝑏0 is the bias or intercept term and 𝑏1 is the coefficient for the single input value (𝑥). Each column in your input data has an associated 𝑏 coefficient (a constant real value) that must be learned from your training data. The value 𝑦 indicates the proportion of the data at 𝑥.

The following diagram compares the linear regression to the logistic regression.



So, the logistic regression is a linear method, but the predictions are transformed using the logistic function. If we replace 𝑦 with 𝑝(𝑥), the probability of a given 𝑥, we may derive the following:

𝑒𝑏0+𝑏1𝑥

𝑝(𝑥) = 1 + 𝑒𝑏0+𝑏1𝑥

𝑝(𝑥) + 𝑝(𝑥)𝑒𝑏0+𝑏1𝑥 = 𝑒𝑏0+𝑏1𝑥

𝑏0+𝑏1𝑥 = 𝑝(𝑥)

𝑒

1 − 𝑝(𝑥)

𝑝(𝑥)

𝑏0 + 𝑏1𝑥 = ln 1 − 𝑝(𝑥)

𝑝(𝑥)

The term is odds. Odds are calculated as a ratio of the probability of the event divided by

1−𝑝(𝑥)

the probability of not the event, e.g. 0.8/(1-0.8) which has the odds of 4. So we could instead write: ln(odds) = b0 + b1 \* X . So the equation becomes

𝑏0 + 𝑏1𝑥 = ln 𝑜𝑑𝑑𝑠

Because the odds are log transformed, we call this left hand side the log-odds or the probit.

𝑜𝑑𝑑𝑠 = 𝑒𝑏0+𝑏1𝑥

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation. The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximumlikelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

Maximize the joint probability of observing the data (*X*) given a specific probability distribution and its parameters (*theta*), stated formally as: 𝑝(𝑋; 𝜃) where “;” indicates “parameterized by” 𝜃 and 𝜃 is a parameter not random variable, and in fact, this is the joint probability distribution of all observations from the problem domain from 1 to *n*.

𝑝(𝑋; 𝜃) = 𝑝(𝑥1, 𝑥2, … , 𝑥𝑛; 𝜃)

This resulting probability is referred to as the likelihood of observing the data given the model parameters and written using the notation *L()* to denote the likelihood function. For example:

𝑝(𝑋; 𝜃) = 𝑝(𝑥1, 𝑥2, … , 𝑥𝑛; 𝜃) = 𝐿(𝑋; 𝜃)

The joint probability distribution can be restated as the multiplication of the probability for observing each example given the distribution parameters. Multiplying many small probabilities together can be unstable; as such, it is common to restate this problem as the sum of the log conditional probability.

ℒ(𝑋; 𝜃) = 𝑝(𝑥1, 𝑥2, … , 𝑥𝑛; 𝜃) 

𝑛



Given the frequent use of log in the likelihood function, it is referred to as a log-likelihood function. It is common in optimization problems to prefer to minimize the cost function rather than to maximize it. Therefore, the negative of the log-likelihood function is used, referred to generally as a Negative Log-Likelihood (NLL) function.

𝑛

min − log ℒ(𝑋; 𝜃) = min −log 𝑝(𝑥𝑖; 𝜃)

𝜃 𝜃

Given a sample 𝑥 , what we want is ℎ𝜃(𝑥) ∈ (0,1).

ℎ𝜃(𝑥) = 1+𝑒1−𝜃𝑇𝑥 the sigmoid or logistic function will limit output to (0,1)

Supervised learning can be framed as a conditional probability problem of predicting the probability of the output given the input: 𝑝(𝑦 = 1|𝑥; 𝜃) = ℎ𝜃(𝑥).

Similarly, 𝑝(𝑦 = 0|𝑥; 𝜃) = 1 − ℎ𝜃(𝑥) for the 𝑦 = 0 case.

Combine the two cases, we have 𝑝(𝑦|𝑥; 𝜃) = ℎ𝜃(𝑥)𝑦(1 − ℎ𝜃(𝑥))1−𝑦.

The likelihood is ℒ(𝜃) = 𝑝(𝑦⃗|𝑋; 𝜃) = 𝑝(𝑦1|𝑥1; 𝜃)𝑝(𝑦2|𝑥2; 𝜃) … 𝑝(𝑦𝑛|𝑥𝑛; 𝜃) (𝑦𝑖|𝑥𝑖; 𝜃)

The log likelihood 𝑙(𝑦𝑖|𝑥𝑖; 𝜃) 

(1 − 𝑦𝑖) log(1 − ℎ𝜃(𝑥𝑖))

Want to choose 𝜃 to  𝑙(𝜃)

𝜃

There is no closed form for 𝜃 but we may get a global maximum using batch gradient ascent method.

𝜕𝑙(𝜃)

𝜃𝑗 ≔ 𝜃𝑗 + 𝜕𝜃𝑗 Here we use + to climb up to the hill.

𝜕𝑙

(

𝜃

)

=

∑

(

𝑦

𝑖

−

ℎ

𝜃

(

𝑥

𝑖

)

)

𝑛

𝑖

𝜃𝑗𝑥𝑗 where 𝜕𝜃𝑗 𝑥𝑗 and 𝛼 is the learning rate.

## 8.3 Simple example

Given an email, we need to classify whether it is spam or not. First, we labeled the training dataset with the following features:

* Sender
* # of typos
* Occurrence of words like “offer”, “price”, “free gift”, etc.

We then train a logistic classifier, which outputs a score in [0,1]. If the score > 0.5, it is a spam. Otherwise, it is not.

## 8.3 Spark programming example

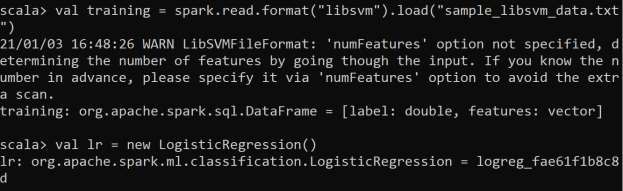
The following example shows how to train a binomial logistic regression model (can be configured to a multinomial logistic regression as well) for binary classification with elastic net regularization by setting the elasticNetParam corresponding to 𝛼 and the regParam corresponding to 𝜆.

|  |
| --- |
| // Import the required Spark machine learning library module, LogisticRegression. import org.apache.spark.ml.classification.LogisticRegression  // Load training dataset that comes with the Spark installation. If you install Spark in C:/spark, the date file is stored in c:/spark/data/mllib.  val training = spark.read.format("libsvm").load("data/mllib/sample\_libsvm\_data.txt")  // Create an object of LogisticRegression for the model. val lr = new LogisticRegression()  // Set the regularization parameter lambda to 0.3 lr.setRegParam(0.3)  // Set the Elastic Net regularization parameter alpha to 0.8. Its value ranges from 0 to 1.  lr.setElasticNetParam(0.8)  // Set the Elastic Net regularization parameter alpha to 0.8. Its value ranges from 0 to 1.  lr.setFamily(“binomial”)  // Fit the model; train the model and solve the parameters val lrModel = lr.fit(training)  // Once trained, we may print the coefficients and intercept for logistic regression println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}") |

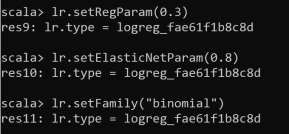
The program is straight forward. First, import the required logistic regression library module.



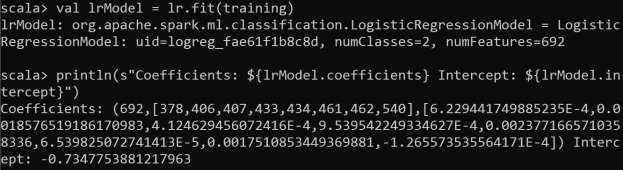
Load the dataset, sample\_libsvm\_data.txt, in libsvm format.



Now, set the hyperparameters, 𝜆, 𝛼, and “binomial” for the logistic regression family for binary classification.



We then train the model with the loaded training dataset. Once the model is trained, we may print out the parameters, i.e., weights for features and Y-intercept. Note that there are 692 coefficients because there are 692 features.



## 8.4 Exercise

Extend the programming example shown in Section 8.3 to use multinomial logistic regression model.

## 8.5 Quizzes

Explain how logistic regression can be used in multiclass classification problems.

# Chapter 9. Naïve Bayes Classifier

## 9.1 Overview

Naïve Bayes classification is a simple machine learning algorithm, typically used as a baseline for comparison purpose. It has some “naïve” assumption that each pair of data is independent, that is normally not true. However, it can be useful in some situations.

There are applications suitable for Naïve Bayes classification, listed below:

* Spam Email Classification
  + Given an email, predict whether it is spam or not
* Medical Diagnosis
  + Given a list of symptoms, predict whether a patient has disease X or not
* Weather Forecast
  + Based on temperature, humidity, etc… predict if it will rain tomorrow

## 9.2 Theory

Given a dataset with labels {𝑥𝑖, 𝑦𝑖}𝑛𝑖=1, we want to create a model and predict 𝑦 for any input 𝑥,

by computing conditional probabilities of labels given an input sample. Naïve Bayes classification is one type of generative learning:

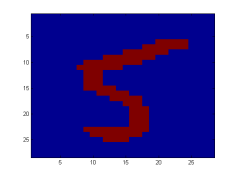
* Given x, we are learning p(x|y) and p(y) to predict p(y|x).
* Bayesian classification

The type is called discriminative learning:

* Given x, we are learning p(y|x) to predict H(p(y|x))=+1 or -1
* One example of discriminative learning is SVM.

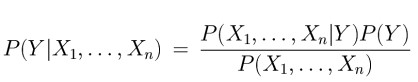
Let’s start with the handwriting recognition but we are recognizing two digits only, i.e., binary classification. The following depicts the flow of the machine learning process.

In order to compute the posterior probability, we use the Bayes theorem as follows:



Classifier

5



Prior



Likelihood



Posterior



Normalization Constant

Here we have 𝑌 represents a possible outcome (class). The posterior is the probability that an image represents 𝑌, given the data 𝑋𝑖. The likelihood is computed from the training dataset and so is the prior. Practically, we do not have to compute the normalization constant as we are looking the highest probability relative to other candidate classes.

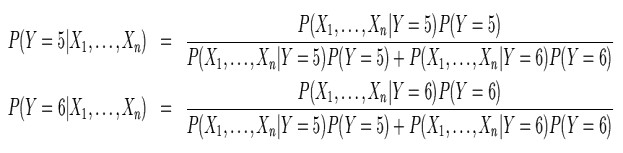
A problem with explicitly modeling P(X1,…,Xn|Y) is that there are usually way too many parameters:

* Each image is of size 28x28, 784 pixels per image. Each pixel can take on integer values in the range of 0..255 inclusive.
* Model this as a discrete probability distribution, you’d have 255^784 different possibilities.
* We’ll run out of space
* We’ll run out of time
* And we’ll need tons of training data (which is usually not available)

One of ways to conquer this is multivariate Gaussian distribution. The discussion of multivariate Gaussian process is out of scope in this chapter.

## 9.3 Simple example

Let’s say we are predicting whether a handwriting image (with resolution 28x28) is 5 or 6. Each image is stored in an array 𝑥𝑖, 𝑖 = 1, 784. A good strategy is to make a good prediction is to compute: the probability that an image represents a 5 given its pixels and the probability that the image represents a 6 given its pixels. For Naïve Bayes classification, we simply compute the following two probabilities:



The model makes its prediction by selecting the class with the larger probability. Since the probability could be very small, so log probability is used in practical computation.

## 9.4 Spark programming example

The NaiveBayes in Spark machine learning supports multinomial and Bernoulli model types. Each feature represents a term whose value is a real number (in multinomial naive Bayes) or a zero or one indicating (in Bernoulli naive Bayes). Feature values must be nonnegative. The model type is selected with an optional parameter “multinomial” or “bernoulli” with “multinomial” as the default.

NaiveBayes implements multinomial naive Bayes and takes an RDD of LabeledPoint and an optional smoothing parameter lambda as input, an optional model type parameter (default is “multinomial”), and outputs a NaiveBayesModel, which can be used for evaluation and prediction.

The following programming example shows how to use NaiveBayes and NaiveBayesModel classes in Spark.

// Import the required Spark machine learning library modules: NaiveBayes and NaiveBay esModel. Note that the special parentheses syntax allowing importing multiple library modules in an import statement.

import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}

// Import the library MLUtils, that supports loading SVM data files and other utilitie s import org.apache.spark.mllib.util.MLUtils

// Load and parse the data file in LibsVM format. The data file, sample\_libsvm\_data.tx t, is stored c:/spark/data/mllib.

val data = MLUtils.loadLibSVMFile(sc, "sample\_libsvm\_data.txt")

// Split data into training (60%) and test (40%) randomly. This is a typical process i n splitting a dataset into two sub datasets randomly.

val Array(training, test) = data.randomSplit(Array(0.6, 0.4))

// Train the model using training dataset and set the model type to multinomial.

// For some datasets, the model type can be set to bernoulli as well.

val model = NaiveBayes.train(training, lambda = 1.0, modelType = "multinomial")

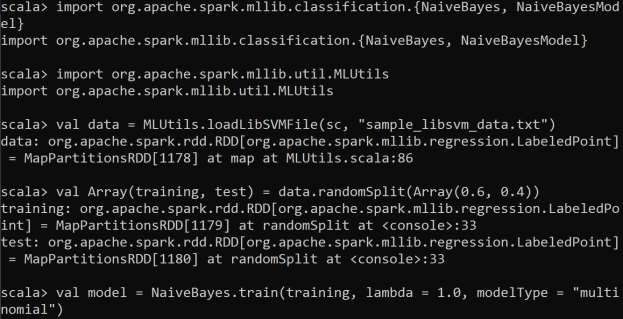
// After training, the model can be used to prediction. The result is stored in predic tionAndLabel, which can be used to compute accuracy and other performance metrics.

val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))

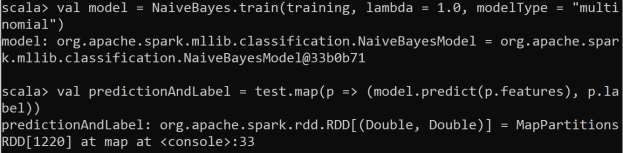
// The accuracy can be computed by the portion of the correct prediction.

val accuracy = 1.0 \* predictionAndLabel.filter(x => x.\_1 == x.\_2).count() / test.count ()

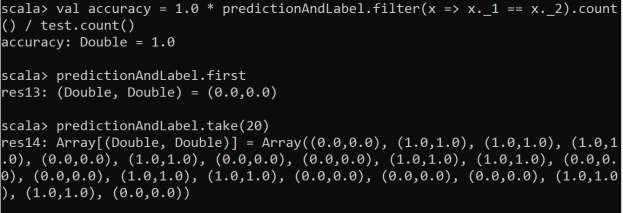
The following shows the actual execution trace.



The method train() returns a NaiveAyesModel object, which will be used for prediction.



After running the method predict(), the result is organized in an array of ordered pair, predicted value and label. We can compare if they are equal to compute the accuracy.



The result shows 100% correct for the Naïve Bayes model with respect to the sample data set. Since we randomize the training and testing data, the accuracy is reliable.

## 9.5 Exercise

Write a program that implements Naïve Bayes classification for email spam detection.

## 9.6 Quizzes

What is the assumption of Naïve Bayes classification?

# Chapter 10. Fisher Linear Discriminant Analysis

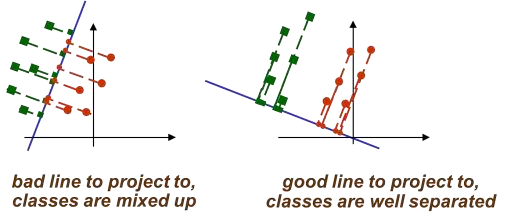
## 10.1 Overview

In data analytics, graphical representation is a great way to comprehend data to be analyzed. Yet, plotting a graph from a dataset with one feature (x) and one label (y) is easy. For 4 dimensional data, we cannot draw a 4D graph, let alone high dimensional data with thousands of features.

In linear discriminant analysis, given a dataset with labels (multiple classes), we want to reduce dimensions with a goal to maximize the separability among classes. Another dimension reduction algorithm, Principal Component Analysis (PCA), reduces dimensions by focusing on the features with the most variation. PCA finds the most accurate *data representation*

in a lower dimensional space by projecting data in the directions of maximal variance. However, the directions of maximum variance may be useless for classification. Fisher Linear Discriminant project to a line which preserves direction useful for *data classification.*

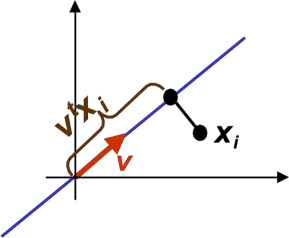
The main idea of LDA is to find projection to a line such that samples from different classes are well separated as depicted in the following diagram.



It is obvious that the projection in the left hand side of the above diagram is bad because the data points are mixed and are harder to separate green points from red points. On the other hand, the right hand side of the above diagram, the projection nicely separates green points from red points. Both of the above projections reduce the dimension from 2 to 1, but the right hand side will give a better performance in splitting green points from red ones.

## 10.2 Theory

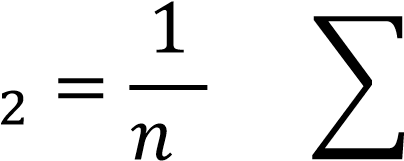
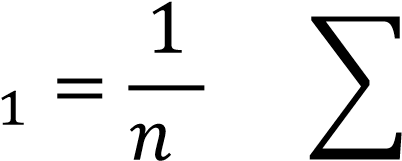
Suppose we have 2 classes and ***d***-dimensional samples ***x1***,…,***xn*** where ***n1*** samples come from the first class and ***n2*** samples come from the second class. Consider projection the data points on a line. Let the line direction be given by unit vector ***v.*** Scalar ***vtxi*** is the distance of projection of ***xi*** from the origin because 𝑣𝑡𝑥𝑖 = ‖𝑣‖‖𝑥𝑖‖ cos 𝜃 = ‖𝑥𝑖‖ cos 𝜃. Thus, ***vtxi*** is the projection of ***xi*** into a one dimensional subspace. The following diagram shows the relation.



Thus, the projection of sample ***xi*** onto a line in direction ***v*** is given by ***vtxi***. How to measure separation between projections of different classes? Let 𝜇̃1 and 𝜇̃2 be the sample means after projection for class 1 and class 2, respectively. Also, let 𝜇1 and 𝜇2 be the sample means for the

|  |  |  |
| --- | --- | --- |
| original data, respectively. |  |  |
| 𝑛1 |  | 𝑛1 |

|  |  |
| --- | --- |
| 𝑘=1, 𝑥𝑘∈𝐶1 | 𝑘=1, 𝑥𝑘∈𝐶1 |
| 𝑛2 | 𝑛2 |

𝑣𝑡

𝜇̃ 𝑣𝑡𝑥𝑘 = ∑ 𝑥𝑘 =𝑣𝑡𝜇1

1. 𝑛1

𝑣𝑡

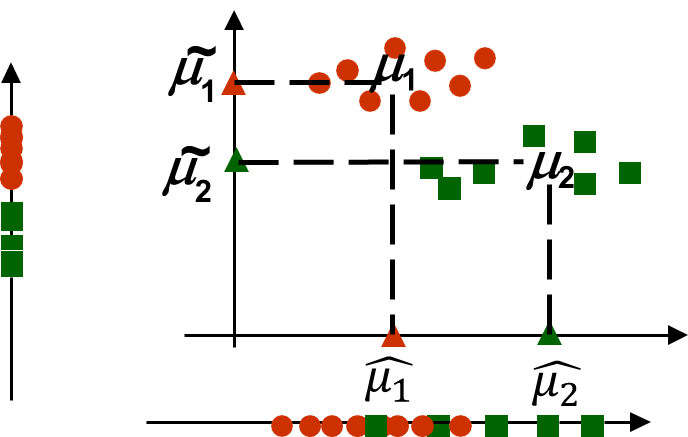
𝜇̃ 𝑣𝑡𝑥𝑘 = ∑ 𝑥𝑘 =𝑣𝑡𝜇2

1. 𝑛2

𝑘=1, 𝑥𝑘∈𝐶2 𝑘=1, 𝑥𝑘∈𝐶2

Note that the sample means are proportional to the original sample means with a factor of 𝑣𝑡. If our goal is to separate the two classes easily, one objective is to maximize the distance between 𝜇̃1 and 𝜇̃2, i.e., |𝜇̃1 − 𝜇̃2|.

However, consider the following example, the larger distance of the sample means after projecting to the horizontal line is no better than the projection along the vertical line.



The problem lies the fact that the distance of the sample means does not consider the sample variance. We need to normalize |𝜇̃1 − 𝜇̃2| by a factor, which is proportional to variance.

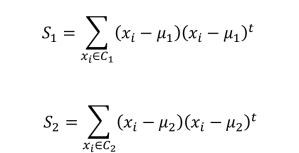
We define scatter as 𝑠12 = ∑(𝑥𝑖 − 𝜇1)2. For Mathematical convenience, we omit the factor 𝑛−11  for the sample variance. The scatter after the projection is 𝑠1̃2 = ∑(𝑣𝑡𝑥𝑖 − 𝑣𝑡𝜇1)2.

Put everything together, we have the Fisher discriminant derivation defined as follows:

(𝜇̃1 − 𝜇̃2)2

𝐽(𝑣) = 𝑠̃12 + 𝑠̃22

If we find ***v*** which makes ***J***(***v***) large, we are guaranteed that the classes are well separated. All we need to do now is to express ***J*** explicitly as a function of ***v*** and maximize it.



Now define the ***within*** the class scatter matrix 𝑠𝑤 = 𝑠1 + 𝑠2.

𝑠̃12 = ∑ (𝑣𝑡𝑥𝑖 −𝑣𝑡𝜇1)2 = ∑ (𝑣𝑡𝑥𝑖 −𝑣𝑡𝜇1)𝑡(𝑣𝑡𝑥𝑖 −𝑣𝑡𝜇1)

𝑦𝑖∈𝐶1 𝑦𝑖∈𝐶1

= ∑ (𝑣𝑡(𝑥𝑖 −𝜇1))𝑡(𝑣𝑡(𝑥𝑖 −𝜇1))

𝑦𝑖∈𝐶1

= ∑ (𝑣𝑡(𝑥𝑖 −𝜇1))𝑡(𝑣𝑡(𝑥𝑖 −𝜇1))= ∑ (𝑥𝑖 −𝜇1)𝑡𝑣(𝑣𝑡(𝑥𝑖 −𝜇1))

𝑦𝑖∈𝐶1 𝑦𝑖∈𝐶1

= ∑ (𝑣𝑡(𝑥𝑖 −𝜇1))(𝑥𝑖 −𝜇1)𝑡𝑣 =𝑣𝑡𝑆1𝑣

𝑦𝑖∈𝐶1

Therefore, we have 𝑠1̃2 + 𝑠̃22 = 𝑣𝑡𝑠1𝑣 + 𝑣𝑡𝑠2𝑣 = 𝑣𝑡(𝑠1 + 𝑠2)𝑣 = 𝑣𝑡𝑠𝑤𝑣.

Let’s now define the class scatter matrix 𝑠𝐵 = (𝜇1 − 𝜇2)(𝜇1 − 𝜇2)𝑡. 𝑠𝐵 measures separation between the means of two classes (before projection).

We may rewrite separation of the projected means as follows:

(𝜇̃1 − 𝜇̃2)2 = (𝑣𝑡𝜇1 − 𝑣𝑡𝜇̃2)2 = 𝑣𝑡(𝜇1 − 𝜇2)(𝜇1 − 𝜇2)𝑡𝑣 = 𝑣𝑡𝑠𝐵𝑣.

The objective function can be written as

𝐽(𝑣) = (𝜇̃𝑠1̃2−+𝜇̃𝑠22̃2)2 = 𝑣𝑣𝑡𝑡𝑠𝑠𝑤𝐵𝑣𝑣.

1

Maximize J(v) by taking the derivative w.r.t. v and set it to 0. We have the following

𝑡𝑆𝑊𝑣) 𝜕(𝑣𝑡𝑆𝐵𝑣) − (𝑣𝑡𝑆 𝜕(𝑣𝑡𝑆

𝜕𝐽𝜕𝑣(𝑣) = (𝑣 𝜕𝑣(𝑣𝑡𝑆𝑤𝑣)2 𝐵𝑣) 𝜕𝑣𝑊𝑣) = (𝑣𝑡𝑆𝑊𝑣)2𝑆(𝐵𝑣𝑣𝑡𝑆−𝑤𝑣(𝑣)2𝑡𝑆𝐵𝑣)2𝑆𝑊𝑣 = 0

𝑛𝑜𝑡𝑒: 𝑣𝑡𝑆𝑊𝑣 𝑖𝑠 𝑎 𝑠𝑐𝑎𝑙𝑎𝑟. 𝑆𝑜 𝑤𝑒 𝑚𝑎𝑦 𝑝𝑙𝑎𝑐𝑒 𝑖𝑡 𝑎𝑛𝑦𝑤ℎ𝑒𝑟𝑒. 𝐵𝑜𝑡ℎ 𝑆𝑤𝑎𝑛𝑑 𝑆𝐵 𝑎𝑟𝑒 𝑠𝑦𝑚𝑚𝑒𝑡𝑟𝑖𝑐.

𝜕𝑦 𝜕𝑦 𝜕𝑦 𝜕𝑦

=< , , … , > 𝑤ℎ𝑒𝑟𝑒 𝑦 𝑖𝑠 𝑎 𝑓𝑢𝑛𝑐𝑡𝑖𝑜𝑛 𝑜𝑓 𝑣.

𝜕𝑣 𝜕𝑣1 𝜕𝑣2 𝜕𝑣𝑛

Finally, we obtain the solution 𝑣 = 𝑠𝑤−1(𝜇1 − 𝜇2).

## 10.3 Simple example

Given the dataset,

Class 1 has 5 samples c1=[(1,2),(2,3),(3,3),(4,5),(5,5)]

Class 2 has 6 samples c2=[(1,0),(2,1),(3,1),(3,2),(5,3),(6,5)] First compute the mean for each class ***m1* =** ***mean* (*c1* )** **=** **[*3 3.6* ]** ***m*** ***2* =** ***mean* (*c2* )** **=** **[*3.3 2* ]** Compute scatter matrices ***S1*** and ***S2*** for each class.

𝑆1 = 4 ∗ 𝑐𝑜𝑣𝑎𝑟𝑖𝑎𝑛𝑐𝑒(𝐶1) = [810.0 87..02]

17.3 16

𝑆2 = 5 ∗ 𝑐𝑜𝑣𝑎𝑟𝑖𝑎𝑛𝑐𝑒(𝐶2) = [ 16 16]

With the class scatter:

27.3 24

𝑆𝑊 = 𝑆1 + 𝑆2 = [ 24 23.2]

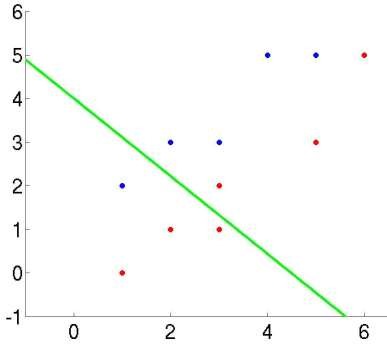
|  |  |
| --- | --- |
| The inverse is SW−1 = 𝑖𝑛𝑣(𝑆𝑊) = [−00.39.41 | −0.41  ].  0.47 |

Finally, the optimal line direction v, 𝑣 = 𝑆𝑊−1(𝜇1 − 𝜇2) = [−00.73.65].

Notice, as long as the line has the right direction, its exact position does not matter. Last step is to compute the actual 1D vector y. Let’s do it separately for each class.

Y1 = 𝑣𝑡𝐶1𝑡 = [−0.65 0.73] [12 …… 55] = [0.81 … 0.4]

Y2 = 𝑣𝑡𝐶2𝑡 = [−0.65 0.73] [10 …… 65] = [−0.65 … −0.25]



## 10.5 Exercise

Implement a Scala program in Spark for Fisher LDA for the dataset exPCA.mat. Refer to exercise in the Section 11.5.

## 10.6 Quizzes

Explain the difference between LDA and PCA.

# Chapter 11. Principal Component Analysis

## 11.1 Overview

Principal component analysis (PCA) is used to reduce dimension of a high dimensional dataset along the “principal” components, which are ranked by the variance with respect to each principal component.

PCA is a technique that can be used to simplify a dataset. It is a linear transformation that chooses a new coordinate system for the data set such that greatest variance by any projection of the data set comes to lie on the first axis (then called the first principal component), the second greatest variance on the second axis, and so on. PCA can be used for reducing dimensionality by eliminating the later principal components.

## 11.2 Theory

To understand PCA, let’s start with the theory of change of basis. Let 𝑋 and 𝑌 be 𝑚 × 𝑛 matrices and 𝑃 be a 𝑚 × 𝑚 matrix. The meaning of P𝑋 = 𝑌 is transform 𝑋 to 𝑌 using the transform matrix 𝑃. Let 𝑝𝑖 be the row vector of 𝑃, 𝑥𝑗, 𝑦𝑗 are column vector of 𝑋 and 𝑌,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| − 𝑝1 −  respectively. [ ⋮ ] [𝑥1  − 𝑝𝑚 −  𝑝1𝑥𝑗 | … | 𝑥𝑛] = [ | 𝑝1𝑥1  ⋮  𝑝𝑚𝑥1 | …  ⋮  … | 𝑝1𝑥𝑛  ⋮ ] = [𝑦1  𝑝𝑚𝑥𝑛 | … | 𝑦𝑛] |

So we have 𝑦𝑗 = [ ⋮ ]. Each component of 𝑦𝑗 is 𝑥𝑗′𝑠 projection along 𝑝𝑖.

𝑝𝑚𝑥𝑗

Covariance matrix is a matrix that represents covariance between dimensions as a matrix e.g. for 3 dimensions:

|  |  |  |
| --- | --- | --- |
| 𝑐𝑜𝑣(𝑥, 𝑥)  𝐶 = [𝑐𝑜𝑣(𝑦, 𝑥)  𝑐𝑜𝑣(𝑧, 𝑥) | 𝑐𝑜𝑣(𝑥, 𝑦)  𝑐𝑜𝑣(𝑦, 𝑦)  𝑐𝑜𝑣(𝑧, 𝑦) | 𝑐𝑜𝑣(𝑥, 𝑧)  𝑐𝑜𝑣(𝑦, 𝑧)]  𝑐𝑜𝑣(𝑧, 𝑧) |

Note that the diagonal is variance of each dimension. So the diagonal elements are the variances of x, y and z. Because cov(x,y) = cov(y,x), hence the covariance matrix is symmetrical about the diagonal. N-dimensional data will result in NxN covariance matrix.

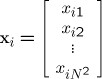
Covariance is a measure of how changes in one variable are associated with changes in a second variable. Specifically, covariance measures the degree to which two variables are linearly associated. However, it is also often used informally as a general measure of how monotonically related two variables are. Correlation Coefficient is scaled of covariance and defined as 𝜌𝑋𝑌 = 𝑐𝑜𝑟𝑟(𝑋, 𝑌) = 𝑐𝑜𝑣𝜎𝑋(𝜎𝑋𝑌,𝑌) where 𝜎𝑋 and 𝜎𝑋 are standard deviations of X and Y, respectively.

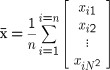
In correlation, the exact value is not as important as its sign. A positive value of covariance indicates both dimensions increase or decrease together e.g. as the number of hours studied increases, the marks in that subject increase. A negative value indicates while one increases the other decreases, or vice-versa e.g. active social life at KSU vs performance in CS dept.

If the two dimensions are independent of each other e.g. heights of students versus the marks obtained in a subject, covariance is zero. If covariance is zero, it does not imply independence!

Correlation (standardized) can tell the strength of the relationship, but covariance cannot. Covariance calculations are used to find relationships between dimensions in high dimensional data sets (usually greater than 3) where visualization is difficult.

Let x1 x2 … xn be a set of n N x 1 column vectors and let 𝑥̅ be their average:





Let X be the N x n matrix with columns

𝑥1 − 𝑥̅, 𝑥2 − 𝑥̅, ⋯ , 𝑥𝑛 − 𝑥̅



**Note that:**

1. N is the # of features and n is the # of samples.
2. subtracting the mean is equivalent to translating the coordinate system to the location of the mean.

Let Q = X XT be the N x N matrix:



Note also that

* 1. 𝑥1 − 𝑥̅ is column vector for sample 1.
  2. Q is square
  3. Q is symmetric
  4. Q is the *scatter* matrix
  5. Q can be very large (in vision, N is often the number of pixels in an image!)
  6. Covariance Matrix 𝑠𝑋 = 𝑛−11 𝑄

PCA assumes the following:

* + PCA assumes all basis vectors are orthogonal in P.
  + Transform data using 𝑃𝑋 = 𝑌.
  + The optimal covariance 𝑆𝑌 after transformation has off-diagonal terms zero.

1 T is

Objective: find some orthonormal matrix 𝑃 where 𝑌 = 𝑃𝑋 such that S𝑌 = n−1 YY diagonalized. The rows of 𝑃 are the principal components of 𝑋.

The spectral theorem is the key to PCA. Let A be an nxn symmetric real matrix. A can be

𝜆1 ⋯ 0

decomposed to 𝑉𝐷𝑉𝑇 where 𝑉 = [𝑣1 𝑣2 … 𝑣𝑛], 𝐷 = [ ⋮ ⋱ ⋮ ], and 𝑣𝑖′𝑠 are orthonormal

0 ⋯ 𝜆𝑛

vectors. Actually, 𝑣𝑖′𝑠 are eigenvectors and 𝜆′𝑖𝑠 are the eigenvalues.

To prove the spectral theorem, let’s start with the eigenvectors for a symmetric matrix is orthogonal. We can simply divide a eigenvector to its norm to get the unit vector. That is all 𝑣𝑖′𝑠

𝑎11 ⋯ 𝑎1𝑛 𝑣11

are orthonormal. We have 𝐴𝑣𝑖 = 𝜆𝑖𝑣𝑖 ∀𝑖 = 1, … , 𝑛. Expand 𝐴𝑣1 = ( ⋮ ⋱ ⋮ ) [ ⋮ ] =

𝑎𝑛1 ⋯ 𝑎𝑛𝑛 𝑣𝑛1

|  |  |  |  |
| --- | --- | --- | --- |
| 𝜆1𝑣11 𝑎11  [ ⋮ ]. We can add 𝐴𝑣2 to the above expression like this: ( ⋮  𝜆1𝑣𝑛1 𝑎𝑛1  𝜆1𝑣11 𝜆2𝑣12 𝜆1𝑣11 ⋯ 𝜆𝑛𝑣1𝑛 [ ⋮ ⋮ ]. Adding all 𝐴𝑣𝑖′𝑠 to get ( ⋮ ⋱ ⋮ ) =  𝜆1𝑣𝑛1𝜆2𝑣𝑛2 𝜆1𝑣𝑛1 ⋯ 𝜆𝑛𝑣𝑛𝑛  𝑣11 ⋯ 𝑣1𝑛 𝜆1 ⋯ 0  ( ⋮ ⋱ ⋮ ) ( ⋮ ⋱ ⋮ ) = VD  𝑣𝑛1 ⋯ 𝑣𝑛𝑛 0 ⋯ 𝜆𝑛  So we have 𝐴𝑉 = 𝑉𝐷. Since 𝑣𝑖′𝑠 are orthonormal, 𝑉𝑇𝑉 = 𝐼. 𝐴𝑉𝑉𝑇 = 𝑉𝐷𝑉𝑇 ⇒ 𝐴 = 𝑉𝐷𝑉𝑇  1 | ⋯  ⋱  ⋯ | 𝑎1𝑛 𝑣11 𝑣12  ⋮ ) [ ⋮ ⋮ ] = | |
| 𝑎𝑛𝑛 | 𝑣𝑛1𝑣𝑛2 |

Recall that we are looking for orthonormal 𝑃 such that S𝑌 = 𝑛−1 𝑌𝑌𝑇 can be diagonalized.

* + 𝑆𝑌 = 𝑛−11 (𝑃𝑋)(𝑃𝑋)𝑇 = 𝑛−11 𝑃(𝑋𝑋𝑇)𝑃𝑇 = 𝑛−11 𝑃𝑄𝑃𝑇

Since 𝑄 is symmetric, 𝑄 = 𝑉𝐷𝑉𝑇 where 𝑉 is eigenvectors of 𝑄 organized as columns. If 𝑄 has less than 𝑟 ≤ 𝑛 orthonormal eigenvectors where 𝑟 is the rank of 𝑄, we can fill up additional (𝑛 − 𝑟) orthonormal vectors in 𝑉.

* + Choose 𝑃 = 𝑉𝑇
  + ⇒ 𝑆𝑌 = 𝑛−11 𝑃𝑄𝑃𝑇 = 𝑛−11 𝑉𝑇(𝑉𝐷𝑉𝑇)𝑉 = 𝑛−11 (𝑉𝑇𝑉)𝐷(𝑉𝑇𝑉)
  + = 1 𝐼𝐷𝐼 = 1 𝐷

𝑛−1 𝑛−1

* + So the principal components of 𝑋 are the eigenvectors of 𝑋𝑋𝑇 or the rows of 𝑃.
  + The 𝑖𝑡ℎ diagonal value of SY is the variance of 𝑋 along 𝑝𝑖.

So the PCA theorem is the following:

Theorem:

Each xj can be written as:

𝑛

𝑉𝑇𝑋 = 𝑌 ⇒ 𝑋 = 𝑉𝑌 ⇒ 𝑥𝑗 − 𝑥̅ = ∑ 𝑦𝑗𝑖𝑣𝑖

𝑖=1

where 𝑣𝑖 are the n eigenvectors of Q with non-zero eigenvalues.

**Notes:**

* 1. The eigenvectors span an ***eigenspace.***
  2. 𝑣𝑖’s are N x 1 orthonormal vectors (directions in N-Dimensional space)

The scalars 𝑦𝑗𝑖 are the coordinates of xj in the space. 𝑦𝑗𝑖 = (𝑥𝑗 − 𝑥̅)𝑣𝑖

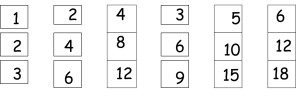
Eigenvalues in the diagonal refers to the variances along the eigenvectors. By finding the eigenvalues and eigenvectors of the covariance matrix, we find that the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset. This is the principal component.

PCA is a useful statistical technique that has found application in: • fields such as face recognition and image compression

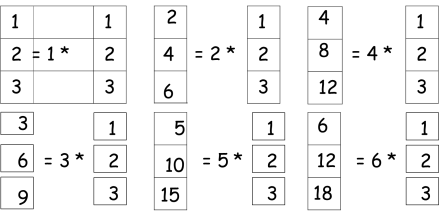
* + finding patterns in data of high dimension.

## 11.3 Simple example

Consider the following 3D points



If each component is stored in a byte, we will need 18 = 3 x 6 bytes to store them. Looking closer, we can see that all the points are related geometrically. They are all the same point, scaled by a factor.



They can ng only 9 bytes (50% savings!). Store one point (3 bytes) + the multiplying constants. We may view the points in 3D space. But in this example, all the points happen to belong to a



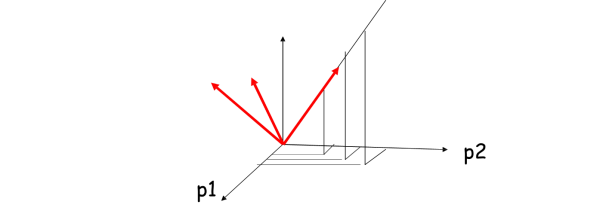
P1

P2

P3

line, a 1D subspace of the original 3D space.

For geometrical interpretation, consider a new coordinate system where one of the axes along the direction of the line.



In this coordinate system, every point has only one non-zero coordinate: we *only* need to store the direction of the line (a 3 bytes image) and the non-zero coordinate for each of the points (6 bytes).

## 11.4 Spark programming example

Spark machine learning library currently does not implement PCA. However, readers may implement a Scala program to compute PCA.

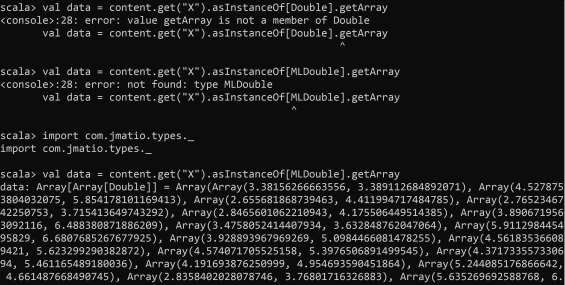
## 11.5 Exercise

In this exercise, you will implement PCA in Spark.

1. Load the data file “exPCA.mat” in MatLab format. This process requires external Java library “JMatIO.” First, download the JMatIO.jar file to your computer and store it in c:\spark\jars and the “exPCA.mat” file to your working director, say, c:\spark\dan\hw6. The following sample demonstrates how to load a MatLab file in Spark.



You need to the fully quantified path for the class. The content is a 50x2 double array but in type MLArray.



You also need to import com.jmatio.types.\_ to get MLDouble to be able to use getArray method. If everything works out, we got the 50x2 array of Doubles. Move on to the next step. The data look like:

0

1

2

3

4

5

6

7

8

0

1

2

3

4

5

6

7

exPCA

1. Normalize the data. We will need to normalize the data to have mean 0. In this exercise, you only got two features. So compute the means for each and subtract them from the data points.
2. Use the Spark PCA library to compute the top one principal component. Note that the Spark ML uses row matrix (not column matrix). Take a snapshot of the principal component.
3. Project the rows to the linear space spanned by the top one principal component. Show the 50 numbers and take a snapshot.

References

1. Spark Dimensionality Reduction – RDD-based API, <https://spark.apache.org/docs/latest/mllib-dimensionality-reduction>
2. JMatIO – Matlab’s MAT-file I/O in Java, <https://sourceforge.net/projects/jmatio/>

## 11.6 Quizzes

Which of the following are recommended applications for PCA? a\*) Data visualization: reduce data to 2D or 3D. So it can be plotted.

1. To get more features to feed into a learning algorithm
2. To prevent overfitting. Reduce the number of features. So there are fewer parameters to learn.

d\*) Data compression. Reduce the dimension of data. So it takes up less memory or disk space.

# Chapter 12. K-Nearest Neighbor Algorithm (KNN)

## 12.1 Overview

As one of the commonly used data mining algorithms, K nearest neighbor search is to find K nearest neighbors for a given query point. It is widely used in many fields such as pattern recognition, decision making, etc.

## 12.2 Theory

Let’s start with the concept of nearest neighbor search (NNS). Given a set X of points with the size of n, in d dimensional data space, the nearest neighbor search is straightforward: find the data point in X that is closest to a given query q.

Picture 1 shows an example.

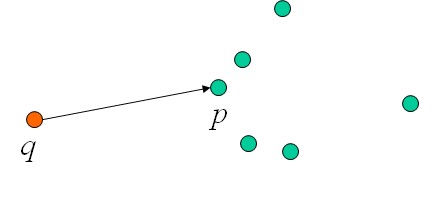


Fig 1: an example of nearest neighbor search

Nearest Neighbor Search is also called proximity search, similarity search, closest point search, and post-office problem. It can be applied in many different fields, including pattern recognition, in particular for optical character recognition. It is also applied in statistical classification, computer vision, databases such as those for content-based image retrieval, coding theory such as maximum likelihood decoding, data compression, recommendation systems, internet marketing for contextual advertising and behavioral targeting, DNA sequencing, spell checking with suggested correct spelling, plagiarism detection, contact searching algorithms in FEA, similarity scores for predicting career paths of professional athletes.

It can also be applied in cluster analysis which is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense, usually based on Euclidean distance.

Various solutions to the NNS problem have been proposed. The quality and usefulness of the algorithms are determined by the time complexity of queries as well as the space complexity of any search data structures that must be maintained. The informal observation usually referred to as the curse of dimensionality states that there is no general-purpose exact solution for NNS in high-dimensional Euclidean space using polynomial preprocessing and polylogarithmic search time.

One of the important variants of NNS is K nearest Neighbor Search (KNN). Given: a set X of n points, given a query point q, and a number K, we aim to find the K nearest neighbors of q in X.

Picture 2 shows an example:

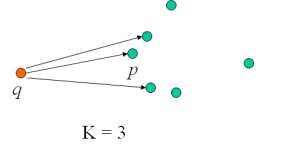


Fig 2: an example of K nearest neighbor search when K=3

KNN can be used in fields of pattern recognition, text categorization, finance, agriculture, medicine etc. For example, KNN is applied to forecast economic events. It can also help analyze heart issues. Some KNN approaches are focused on intrusion detection. In the image processing field, images are often represented as features (color histogram, etc.), and we can use KNN to fine images that are most similar to a given image. We also perform web-page search using KNN, for example: “Find 100 most similar pages for a given page”. In GIS, we can conduct search such as “find 5 closest cities to New York”.

## 12.3 Simple example

KNN is intuitive, easy to understand and implement. It renders accurate results compared to classification approaches. However, KNN also has its shortcomings. KNN needs K as the input parameter, but determining an appropriate value of K is not always an easy job. KNN also needs memory space big enough to hold all the data points, which could be problematic when we deal with big data sets. The computation time is also an issue because every time when a new query data point comes in, we need to calculate its distance (dissimilarity) with all the data points in the data set and determine K neighbors.

To overcome those shortcomings, researchers have proposed various approaches to improve the performance of KNN. For example, the algorithm "Locally Adaptive KNN" applies a unique value of K for each subset of data. With the observation that different features of data set have different importance, the algorithm "Weight Adjusted KNN" assigns a different weight for each feature before processing KNN. Due to the fact that different classes are not evenly distributed in a data set, the algorithm "Improved KNN for Text Categorization" analyzes the distribution for each class, and use this information to decide the value of K for each class. The algorithm "Adaptive KNN" uses heuristics to make the KNN process more flexible, in order to improve the time complexity. The algorithm "KNN with Shared Nearest Neighbors" applied a certain similarity measure to set a threshold to limit the group of neighbors that can participate in labeling a new data point. The algorithm "KNN with K-Means" tries to improve the computation complexity by first finding clusters, then use those clusters the help find K nearest neighbors of a new coming data point. The algorithm "KNN with Mahalanobis Metric" proposes a new distance metric called Mahalanobis distance metric that reform the data space with linear transformation, with the benefit of taking the data point correlation into consideration. The algorithm "Generalized KNN" revises the original KNN to process data sets with features of continuous values. The algorithm "Informative KNN" defines a new measure called informativeness, which analyzes the importance of each of the K neighbors. The algorithm "Bayesian KNN" combines KNN with Bayesian classifier, which assign membership probabilities to a new data point for ordering data points. The algorithm "SVM KNN" proposes combination of KNN and SVM, using KNN to train SVM classifier.

## 12.4 Spark programming example

KNN has also been combined with Neural Network, Fuzzy Logic, Rough Sets and Evolutionary Computing to solve problems. For example, In "Fuzzy KNN", gives fuzzy membership to each data point in the date set. In "Fuzzy-Rough KNN", rough set theory is fused with KNN, and it calculates the range of approximations for each distinct class. The classification of unknown tuples is based on their membership to these approximations for each class in the data set. The "KNN with Genetic Algorithm" applies genetic algorithm with KNN, using the genetic algorithm to reduce the computation time especially for large data sets. It has a wide range of applications such as heart issue diagnosis, by ranking features of a data set based on their relevance. "KNN with Ant Colony Optimization(ACO)" uses ACO algorithm to select limited number of features before KNN process. "KNN with Particle Swarm Optimization(PSO)" is applied to study coronary artery disease, and it is also used to calculate the importance value for each feature of a data set. "KNN with Artificial Bee Colony Algorithm (ABC)" uses the ABC algorithm to determine samples of classes for a data set.

KNN is known to be hard to parallelize in Spark because KNN is a "lazy learner" and the model itself is the entire dataset. The author saurfang published spark-knn which is a hybrid spill tree approach to achieve high accuracy and search efficiency on https://github.com/saurfang/spark-knn.

## 12.5 Exercise

Please implement KNN in Spark.

## 12.6 Quizzes

Q1: What is Nearest Neighbor Search?

Q2: What is KNN?

Q3: What are the advantages and disadvantages of KNN?

# Chapter 13. K-Means Algorithm

## 13.1 Overview

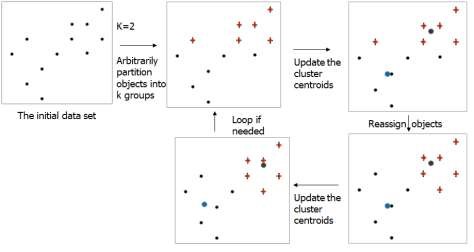
K-means Clustering is a type of partitioned clustering approach, which partitions a data set D of n objects into a set of k clusters, such that the sum of squared distances is minimized.

## 13.2 Theory

In K-means, each cluster is associated with a centroid (center point). Each point is assigned to the cluster with the closest centroid. To perform K-means on a data set, the number of clusters, K, must be specified. We first select K data points in the data set as the initial centroids, using some strategy such as random selection. Then we form K clusters by assigning all data points to their closest centroids. Then we re-compute the centroid of each cluster based on its data points. Once the centroids are updated, we assign data points to the closest centroid again, and re-compute the cluster centroids one more time. We repeat this process until the change of centroids are within a given threshold or they do not change at all.

## 13.3 Simple example

An example is shown below:



If the initial centroids are often chosen randomly, clusters produced will vary from one run to another. The centroid mi is typically the average values (mean) of the points in the cluster. The distance between a data point and a centroid is measured by Euclidean distance, cosine similarity, etc. K-means will converge for common similarity measures mentioned above. Most of the convergence happens in the first few iterations.

## 13.4 Spark programming example

Most common measure is Sum of Squared Error (SSE). For each point, the error is defined as the distance to the nearest cluster

To get SSE, we square these errors and sum them.

*K*

*SSE* *dist*2(*mi* ,*x*)

*i*1 *x**C*

*i*

in which x is a data point in cluster Ci and mi is the centroid of cluster Ci. Given two

clusters, we can choose the one with the smallest error, because the centroids of the clustering with smaller error are a better representation of points. One easy way to reduce SSE is to increase K, the number of clusters. However, a good clustering with smaller K can have a lower SSE than a poor clustering with higher K.

As we mentioned, when the initial centroids are chosen randomly, it greatly affects the quality of the clusters produces. There are solutions to this problem. One is to have multiple runs that sometimes improve the quality of clusters. Researchers also use hierarchical clustering to determine initial centroids. Another solution is to select more than k initial centroids and then select among these initial centroids. The criteria used here is always select most widely separated K candidates among the initial centroids. Another solution is bisecting K-means that is not as susceptible to initialization issues. Postprocessing is also a well known solution, which eliminates small clusters that may represent outliers, and in the meantime splits ‘loose’ clusters, i.e., clusters with relatively high SSE. It also merges clusters that are ‘close’ and that have relatively low SSE.

K-means is a very intuitive and effective clustering algorithm. However, it has problems when clusters have different sizes, densities, or non-globular shapes. Also when there are outliers existing in a data set, K-means' performance can also be affected.

To overcome its limitations, we can first generate many small clusters, and merge them together to form K clusters.

MLlib, Spark’s machine learning (ML) library, a parallelized variant of the k-means++ method called kmeans||.

A java version of Kmeans implementation in spark is as follows:

import org.apache.spark.ml.clustering.KMeansModel; import org.apache.spark.ml.clustering.KMeans; import org.apache.spark.ml.evaluation.ClusteringEvaluator; import org.apache.spark.ml.linalg.Vector; import org.apache.spark.sql.Dataset;

import org.apache.spark.sql.Row;

// Loads data.

Dataset<Row> dataset = spark.read().format("libsvm").load("data/mllib/sample\_kmeans\_data.txt");

// Trains a k-means model.

KMeans kmeans = new KMeans().setK(2).setSeed(1L);

KMeansModel model = kmeans.fit(dataset);

// Make predictions

Dataset<Row> predictions = model.transform(dataset);

// Evaluate clustering by computing Silhouette score

ClusteringEvaluator evaluator = new ClusteringEvaluator();

double silhouette = evaluator.evaluate(predictions);

System.out.println("Silhouette with squared euclidean distance = " + silhouette);

// Shows the result.

Vector[] centers = model.clusterCenters(); System.out.println("Cluster Centers: "); for (Vector center: centers) {

System.out.println(center);

}

## 13.5 Exercise

Please implement K-means in Spark using Python.

## 13.6 Quizzes

Q1: What is clustering?

Q2: What is K-means algorithm?

Q3: What are the advantages and disadvantages of K-means?

# Chapter 14. MapReduce Framework

## 14.1 Overview

MapReduce is a distributed program model for processing data with large volume. Any program that follows this model is inherently distributed and parallel. In this chapter we will illustrate how to write a program by using MapReduce framework to analyze a weather data set.

MapReduce automatic parallelization and distribution, fault-tolerance, I/O scheduling and status and monitoring.

## 14.2 Theory

MapReduce works by breaking the processing into two phases: Map phase and Reduce phase.

Correspondingly, the program using MapReduce framework will specify two functions: Map function (of a Mapper class) and Reduce function (of a Reducer class).

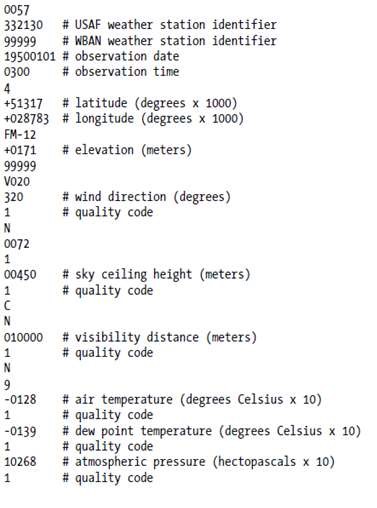
The inputs and outputs for both functions will be (key, value) pairs. Input, final output are stored on a distributed file system. The scheduler tries to schedule map tasks “close” to physical storage location of input data. The intermediate results are stored on local FS of map and reduce workers. The output is often input to another map reduce task. During the process, the idle tasks get scheduled as workers become available.

When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer. Here R is the number of reducers. Master pushes this info to reducers and also pings workers periodically to detect failures

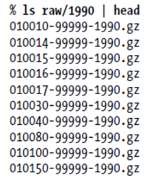
## 14.3 Simple example

We will first analyze the Sample Weather Data Set. It is generated by many weather sensors that collect data every hour at many locations across the globe. The dataset can be downloaded from National Climate Data Center (NCDC) at http://www.ncdc.noaa.gov

The data is stored using a line-oriented ASCII format. Below is a sample record of the data:

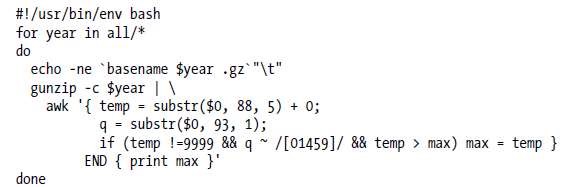


Data files are organized by date and weather station. There is a directory for each year from 1901, and each directory contains a zipped file for each weather station. Below is an example of how files are organized:



What should we do if would like to find out “what is the highest global temperature for each year”?

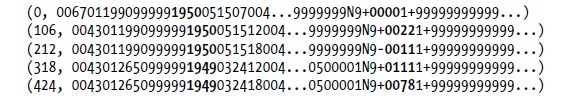
If we use the traditional way to analyze the data with Unix tools shown below:



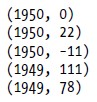
It will take 42 minutes in one run on a single EC2 High-CPU Extra Large Instance.

The alternative way is to analyze the data with Hadoop MapReduce, to take advantage of the distributed processing capability the Hadoop provides. Here we will to write our program by using MapReduce framework.

For the Map function, it processes input key/value pair, and produces set of intermediate pairs. The input could be:



and its output could be:



After the Map Function (map stage), the MapReduce Framework will sort all the output pairs and combine them into the following (key, value) pairs:



For the Reduce function, it combines all intermediate values for a particular key, and produces a set of merged output values (usually just one) . The input could be:



and the output could be:



Here is an example java code of the mapper class that extends and MapReduceBase class and implements the Mapper interface:



Here is an example java code of the reducer class that extends and MapReduceBase class and implements the Reducer interface:

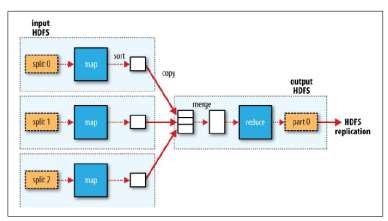


Once we designed the both mapper and reducer classes, we will implement our process in the MaxTemperature class:

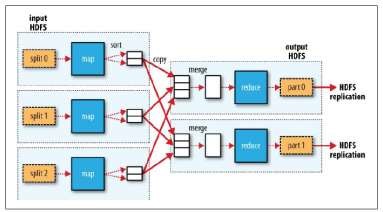


As you can see in the previous figure, there are two types of nodes that control the job execution process. One is called jobtracker and the other is tasktracker. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers, and the tasktrackers run tasks and send progress report to the jobtracker.

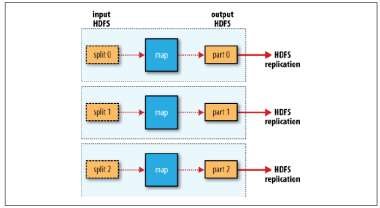
Here is an example of the MapReduce framework with just one reducer designed:



Here is an example of the MapReduce framework with two reducers designed:



Here is an example of the MapReduce framework with no reducers designed:



## 14.4 Exercise

Exercise 1: Host size.

Suppose we have a large web corpus. The metadata file contains lines of the form (URL, size, date, …). For each host, find the total number of bytes (the sum of the page sizes for all URLs from that host)

Exercise 2: Graph reversal.

Given a directed graph as an adjacency list: src1: dest11, dest12, … and src2: dest21, dest22, …, please construct the graph in which all the links are reversed.

Exercise 34: Frequent Pairs.

Given a large set of market baskets, represented by Data: Basket1, Item11, Item12, etc.

which contain a lot of transaction files. Each line of a transaction file is a list of items. Find all frequent pairs.

Exercise 5: Incoming Links

Given a set of HTML pages, compute the number of incoming hyperlinks for each URL. For example, suppose a HTML file appears in 3 pages: 3 times in page A, 3 times in page B, and 4 times in page C. Then its number of incoming hyper-links is 10.

## 14.5 Quizzes

Q1: What is MapReduce?

Q2: How does MapReduce implement parallel programming?

# Chapter 15. Spark SQL

## 15.1 Overview

Spark SQL blurs the lines between RDDs and relational tables. It intermixes SQL commands to query external data, along with complex analytics, in a single Spark program using either SQL or a familiar DataFrame API in Java, Scala, Python, or R. Furthermore, Spark SQL allows SQL extensions based on Spark MLlib. DataFrames and SQL provide a common way to access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC. You can even join data across these sources. Spark SQL supports the HiveQL syntax as well as Hive SerDes and UDFs, allowing you to access existing Hive warehouses. A server mode provides industry standard JDBC and ODBC connectivity for business intelligence tools. Spark SQL supports a costbased optimizer, columnar storage and code generation to make queries fast. Meanwhile, it scales to thousands of nodes and multi hour queries using the Spark engine, which provides full mid-query fault tolerance.

The predecessor of Spark SQL, Apache Shark was migrated to Spark SQL in 2014. Apache Shark is a large-scale data warehouse system for Spark designed to be the next generation Hive. However, it turns out that the goal does not line up with the rest of the spark projects for the reason that it only supports Hadoop and map reduce like Hive does. The succeeded project, Spark SQL is a generic runtime for processing big data with SQL. It also has a Hive context builtin for compatibility.

## 15.2 Theory

The entry point into all functionality in Spark is the SparkSession class. SparkSession in Spark 3.0 provides built-in support for Hive features including the ability to write queries using HiveQL, access to Hive User Defined Functions (UDFs), and the ability to read data from Hive tables.

To use these features, you do not need to have an existing Hive setup.

## 15.3 Simple example

Spark-shell will instantiate a SparkSession object, spark. You don’t have to create another

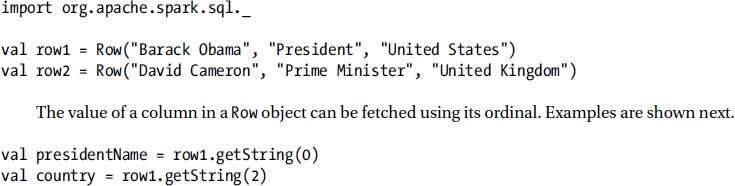
SparkSession object. Just use “spark” to the default one in Spark shell as follows,



DataFrame is Spark SQL’s primary data abstraction. Unlike RDD, DataFrame is schema aware.

It represents a distributed collection of rows organized into named columns. Conceptually, it is similar to a table in a relational database.

Row is a Spark SQL abstraction for representing a row of data. Conceptually, it is equivalent to a relational tuple or row in a table. Spark SQL provides factory methods to create Row objects. An example is shown next.



A DataFrame can be created in two ways. it can be created from a data source. It can also be created from an RDD. Spark SQL provides two methods for creating a DataFrame from an RDD: **toDF** and **createDataFrame**.

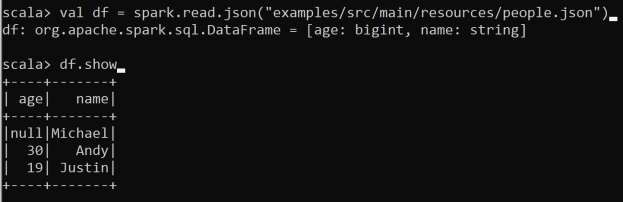
With a SparkSession, applications can create DataFrames

* from an existing RDD,
* from a Hive table, or
* from Spark data sources.

As an example, the following creates a DataFrame based on the content of a JSON file:

|  |
| --- |
| **val** df **=** spark.read.json("examples/src/main/resources/people.json")  *// Displays the content of the DataFrame to stdout* df.show()  *// +----+-------+*  *// | age| name|*  *// +----+-------+*  *// |null|Michael|*  *// | 30| Andy|*  *// | 19| Justin|*  *// +----+-------+* |

A DataFrame is created as follows:



Spark SQL provides Untyped dataset operations (aka dataframe operations). DataFrames provide a domain-specific language for structured data manipulation in Scala, Java, Python and

R. As mentioned above, in Spark 3.0, DataFrames are just Dataset of Rows in Scala and Java API.

These operations are also referred as “untyped transformations” in contrast to “typed transformations” come with strongly typed Scala/Java Datasets. Here we include some basic examples of structured data processing using Datasets: // This import is needed to use the $-notation import spark.implicits.\_

// Print the schema in a tree format df.printSchema()

// root

// |-- age: long (nullable = true)

// |-- name: string (nullable = true) // Select only the "name" column df.select("name").show()

// +-------+

// | name|

// +-------+

// |Michael|

// | Andy|

// | Justin|

// +-------+

// Select everybody, but increment the age by 1 df.select($"name", $"age" + 1).show()

// +-------+---------+

// | name|(age + 1)|

// +-------+---------+

// |Michael| null|

// | Andy| 31|

// | Justin| 20|

// +-------+---------+

// Select people older than 21 df.filter($"age" > 21).show()

// +---+----+

// |age|name|

// +---+----+

// | 30|Andy|

// +---+----+

// Count people by age df.groupBy("age").count().show()

// +----+-----+

// | age|count|

// +----+-----+ // | 19| 1|

// |null| 1|

// | 30| 1|

// +----+-----+

## 15.4 Spark programming example

A Dataset is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each Dataset also has an untyped view called a DataFrame, which is a Dataset of Row. Operations available on Datasets are divided into transformations and actions. Transformations are the ones that produce new Datasets, and actions are the ones that trigger computation and return results. Example transformations include map, filter, select, and aggregate (groupBy). Example actions count, show, or writing data out to file systems.

The SQLContext class provides a method named sql, which executes a SQL query using Spark. It takes a SQL statement as an argument and returns the result as an instance of the DataFrame class.



The sql function on a SparkSession enables applications to run SQL queries programmatically and returns the result as a DataFrame.

// Register the DataFrame as a SQL temporary view df.createOrReplaceTempView("people") val sqlDF = spark.sql("SELECT \* FROM people") sqlDF.show()

// +----+-------+

// | age| name|

// +----+-------+

// |null|Michael|

// | 30| Andy|

// | 19| Justin|

// +----+-------+

Temporary views in Spark SQL are session-scoped and will disappear if the session that creates it terminates. If you want to have a temporary view that is shared among all sessions and keep alive until the Spark application terminates, you can create a global temporary view. Global temporary view is tied to a system preserved database global\_temp, and we must use the qualified name to refer it, e.g. SELECT \* FROM global\_temp.view1.

// Register the DataFrame as a global temporary view df.createGlobalTempView("people")

// Global temporary view is tied to a system preserved database `global\_temp` spark.sql("SELECT \* FROM global\_temp.people").show()

// +----+-------+

// | age| name|

// +----+-------+

// |null|Michael|

// | 30| Andy|

// | 19| Justin|

// +----+-------+

// Global temporary view is cross-session spark.newSession().sql("SELECT \* FROM global\_temp.people").show()

// +----+-------+

// | age| name|

// +----+-------+

// |null|Michael|

// | 30| Andy|

// | 19| Justin|

// +----+-------+

Datasets are similar to RDDs, however, instead of using Java serialization or Kryo they use a specialized Encoder to serialize the objects for processing or transmitting over the network.

While both encoders and standard serialization are responsible for turning an object into bytes, encoders are code generated dynamically and use a format that allows Spark to perform many operations like filtering, sorting and hashing without deserializing the bytes back into an object.

Use toDS() function to convert an RDD to a dataset.

We may convert sequence to dataset using case class as well.

case class Person(name: String, age: Long) // Encoders are created for case classes val caseClassDS = Seq(Person("Andy", 32)).toDS() caseClassDS.show()

// +----+---+

// |name|age|

// +----+---+

// |Andy| 32| // +----+---+

Alternative way to covert sequence to dataset is using common types as follows:

// Encoders for most common types are automatically provided by importing spark.implicits.\_ val primitiveDS = Seq(1, 2, 3).toDS() primitiveDS.map(\_ + 1).collect() // Returns: Array(2, 3, 4)

DataFrames can be converted to a Dataset by providing a class. Mapping will be done by name val path = "examples/src/main/resources/people.json" val peopleDS = spark.read.json(path).as[Person] peopleDS.show()

// +----+-------+

// | age| name|

// +----+-------+

// |null|Michael|

// | 30| Andy|

// | 19| Justin| // +----+-------+

Spark SQL supports two different methods for converting existing RDDs into Datasets.

The first method uses reflection to infer the schema of an RDD that contains specific types of objects. This reflection-based approach leads to more concise code and works well when you already know the schema while writing your Spark application.

The second method for creating Datasets is through a programmatic interface that allows you to construct a schema and then apply it to an existing RDD. While this method is more verbose, it allows you to construct Datasets when the columns and their types are not known until runtime.

The Scala interface for Spark SQL supports automatically converting an RDD containing case classes to a DataFrame. The case class defines the schema of the table. The names of the arguments to the case class are read using reflection and become the names of the columns. Case classes can also be nested or contain complex types such as Seqs or Arrays. This RDD can be implicitly converted to a DataFrame and then be registered as a table. Tables can be used in subsequent SQL statements.

// For implicit conversions from RDDs to DataFrames import spark.implicits.\_

// Create an RDD of Person objects from a text file, convert it to a Dataframe val peopleDF = spark.sparkContext

.textFile("examples/src/main/resources/people.txt")

.map(\_.split(","))

.map(attributes => Person(attributes(0), attributes(1).trim.toInt))

.toDF()

// Register the DataFrame as a temporary view peopleDF.createOrReplaceTempView("people")

// SQL statements can be run by using the sql methods provided by Spark

val teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 19")

// The columns of a row in the result can be accessed by field index teenagersDF.map(teenager => "Name: " + teenager(0)).show()

// +------------+ // | value|

// +------------+

// |Name: Justin|

// +------------+

// or by field name teenagersDF.map(teenager => "Name: " + teenager.getAs[String]("name")).show()

// +------------+

// | value|

// +------------+

// |Name: Justin|

// +------------+

// No pre-defined encoders for Dataset[Map[K,V]], define explicitly implicit val mapEncoder = org.apache.spark.sql.Encoders.kryo[Map[String, Any]]

// Primitive types and case classes can be also defined as

// implicit val stringIntMapEncoder: Encoder[Map[String, Any]] = ExpressionEncoder() // row.getValuesMap[T] retrieves multiple columns at once into a Map[String, T] teenagersDF.map(teenager => teenager.getValuesMap[Any](List("name", "age"))).collect() // Array(Map("name" -> "Justin", "age" -> 19))

When case classes cannot be defined ahead of time (for example, the structure of records is encoded in a string, or a text dataset will be parsed and fields will be projected differently for different users), a DataFrame can be created programmatically with three steps.

1. Create an RDD of Rows from the original RDD;
2. Create the schema represented by a StructType matching the structure of Rows in the RDD created in Step 1.
3. Apply the schema to the RDD of Rows via createDataFrame method provided by SparkSession.

Here is an example: import org.apache.spark.sql.types.\_

// Create an RDD val peopleRDD = spark.sparkContext.textFile("examples/src/main/resources/people.txt")

// The schema is encoded in a string val schemaString = "name age"

// Generate the schema based on the string of schema

val fields = schemaString.split(" ")

.map(fieldName => StructField(fieldName, StringType, nullable = true))

val schema = StructType(fields)

// Convert records of the RDD (people) to Rows val rowRDD = peopleRDD

.map(\_.split(","))

.map(attributes => Row(attributes(0), attributes(1).trim))

// Apply the schema to the RDD val peopleDF = spark.createDataFrame(rowRDD, schema) // Creates a temporary view using the DataFrame peopleDF.createOrReplaceTempView("people")

// SQL can be run over a temporary view created using DataFrames val results = spark.sql("SELECT name FROM people")

// The results of SQL queries are DataFrames and support all the normal RDD operations // The columns of a row in the result can be accessed by field index or by field name results.map(attributes => "Name: " + attributes(0)).show()

// +-------------+

// | value|

// +-------------+

// |Name: Michael|

// | Name: Andy|

// | Name: Justin|

// +-------------+

## 15.5 Exercise

Load a DataFrame in the Apache Parquet format in Spark shell. A test Parquet file can be found at [https://www.kaggle.com/ferrarisf50/nyctaxi-parquet-cluster-test.](https://www.kaggle.com/ferrarisf50/nyctaxi-parquet-cluster-test) You may use the following statement to load the Parquet file.

spark.sqlContext.read.parquet(“taxi-test.parquet”)

Explore the dataset using spark.sqlContext.sql(“SELECT ….”).collect().foreach(println) statement.