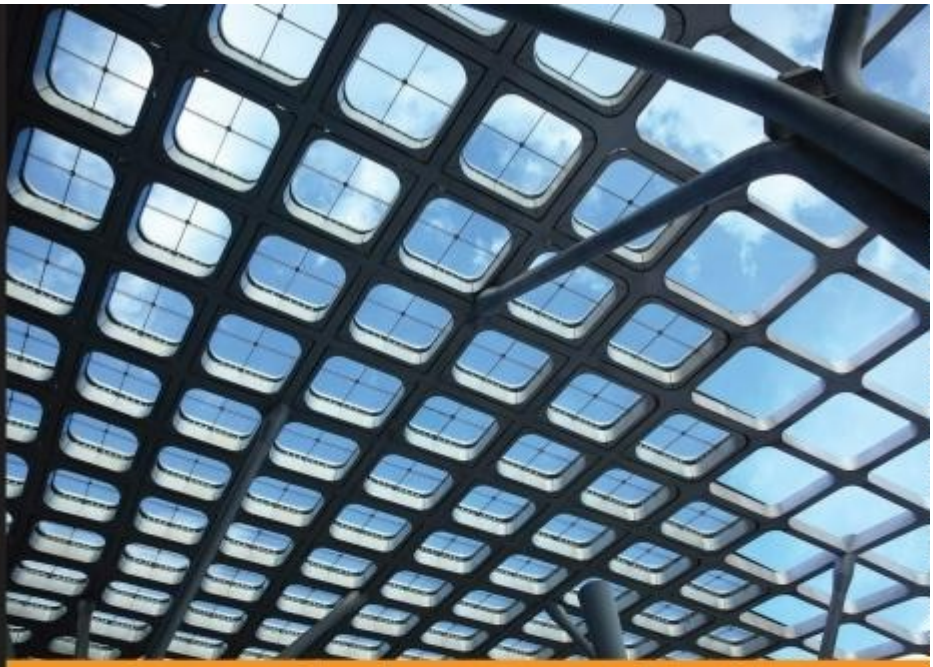


Community Experience Distilled

Learning Apache Mahout Classification

Build and personalize your own classifiers using Apache Mahout

Ashish Gupta

[PACKT] open source* PUBLISHING

Table of Contents

Learning Apache Mahout Classification
Credits
About the Author
About the Reviewers
www.PacktPub.com
    Support files, eBooks, discount offers, and more
        Why subscribe?
        Free access for Packt account holders
Preface
    What this book covers
    What you need for this book
    Who this book is for
    Conventions
    Reader feedback
    Customer support
        Downloading the example code
        Downloading the color images of this book
        Errata
        Piracy
        Questions
1. Classification in Data Analysis
    Introducing the classification
        Application of the classification system
        Working of the classification system
    Classification algorithms
    Model evaluation techniques
        The confusion matrix
        The Receiver Operating Characteristics (ROC) graph
        Area under the ROC curve
        The entropy matrix
    Summary
2. Apache Mahout
    Introducing Apache Mahout
    Algorithms supported in Mahout
    Reasons for Mahout being a good choice for classification
    Installing Mahout
        Building Mahout from source using Maven
            Installing Maven
            Building Mahout code
        Setting up a development environment using Eclipse
        Setting up Mahout for a Windows user
    Summary
3. Learning Logistic Regression / SGD Using Mahout
    Introducing regression

Understanding linear regression
Cost function
Gradient descent
Logistic regression
Stochastic Gradient Descent
Using Mahout for logistic regression
Summary
4. Learning the Naïve Bayes Classification Using Mahout
Introducing conditional probability and the Bayes rule
Understanding the Naïve Bayes algorithm
Understanding the terms used in text classification
Using the Naïve Bayes algorithm in Apache Mahout
Summary
5. Learning the Hidden Markov Model Using Mahout
Deterministic and nondeterministic patterns
The Markov process
Introducing the Hidden Markov Model
Using Mahout for the Hidden Markov Model
Summary
6. Learning Random Forest Using Mahout
Decision tree
Random forest
Using Mahout for Random forest
Steps to use the Random forest algorithm in Mahout
Summary
7. Learning Multilayer Perceptron Using Mahout
Neural network and neurons
Multilayer Perceptron
MLP implementation in Mahout
Using Mahout for MLP
Steps to use the MLP algorithm in Mahout
Summary
8. Mahout Changes in the Upcoming Release
Mahout new changes
Mahout Scala and Spark bindings
Apache Spark
Using Mahout's Spark shell
H2O platform integration
Summary
9. Building an E-mail Classification System Using Apache Mahout
Spam e-mail dataset
Creating the model using the Assassin dataset
Program to use a classifier model
Testing the program
Second use case as an exercise
The ASF e-mail dataset
Classifiers tuning
Summary

Index

Learning Apache Mahout Classification

Learning Apache Mahout Classification

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2015

Production reference: 1210215

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78355-495-9

www.packtpub.com

Credits

Author

Ashish Gupta

Reviewers

Siva Prakash

Tharindu Rusira

Vishnu Viswanath

Commissioning Editor

Akram Hussain

Acquisition Editor

Reshma Raman

Content Development Editor

Merwyn D'souza

Technical Editors

Monica John

Novina Kewalramani

Shruti Rawool

Copy Editors

Sarang Chari

Gladson Monteiro

Aarti Saldanha

Rashmi Sawant

Project Coordinator

Neha Bhatnagar

Proofreaders

Simran Bhogal

Steve Maguire

Indexer

Monica Ajmera Mehta

Graphics

Sheetal Aute

Abhinash Sahu

Production Coordinator

Conidon Miranda

Cover Work

Conidon Miranda

About the Author

Ashish Gupta has been working in the field of software development for the last 8 years. He has worked in different companies, such as SAP Labs and Caterpillar, as a software developer. While working for a start-up where he was responsible for predicting potential customers for new fashion apparels using social media, he developed an interest in the field of machine learning. Since then, he has worked on using big data technologies and machine learning for different industries, including retail, finance, insurance, and so on. He has a passion for learning new technologies and sharing the knowledge thus gained with others. He has organized many boot camps for the Apache Mahout and Hadoop ecosystem.

First of all, I would like to thank open source communities for their continuous efforts in developing great software for all. I would like to thank Merwyn D'Souza and Reshma Raman, my editors for this project. Special thanks to the reviewers of this book.

Nothing can be accomplished without the support of family, friends, and loved ones. I would like to thank my friends, family, and especially my wife and my son for their continuous support throughout the writing of this book.

About the Reviewers

Siva Prakash is working as a tech lead in Bangalore. He has extensive development experience in the analysis, design, development, implementation, and maintenance of various desktop, mobile, and web-based applications. He loves trekking, traveling, music, reading books, and blogging.

You can find him on LinkedIn at https://www.linkedin.com/in/techsivam.

Tharindu Rusira is currently a computer science and engineering undergraduate at the University of Moratuwa, Sri Lanka. As a student researcher, he has strong interests in machine learning, compilers, and high-performance computing.

Tharindu has also worked as a research and development software engineering intern at Zaizi Asia (Pvt) Ltd., where he first started using Apache Mahout during the implementation of an enterprise-level content management and information retrieval system.

He sees the potential of Apache Mahout as a scalable machine learning library for industry-level implementations and has even contributed to the Mahout 0.9 release, the latest stable release of Mahout.

He is available on LinkedIn at https://www.linkedin.com/in/trusira.

Vishnu Viswanath is a senior big data developer who has many years of industrial expertise in the arena of machine learning. He is a tech enthusiast and is passionate about big data and has expertise on most big-data-related technologies.

You can find him on LinkedIn at http://in.linkedin.com/in/vishnuviswanath25.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at <service@packtpub.com> for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



https://www2.packtpub.com/books/subscription/packtlib

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Preface

Thanks to the progress made in the hardware industries, our storage capacity has increased, and because of this, there are many organizations who want to store all types of events for analytics purposes. This has given birth to a new era of machine learning. The field of machine learning is very complex and writing these algorithms is not a piece of cake. Apache Mahout provides us with readymade algorithms in the area of machine learning and saves us from the complex task of algorithm implementation.

The intention of this book is to cover classification algorithms available in Apache Mahout. Whether you have already worked on classification algorithms using some other tool or are completely new to the field, this book will help you. So, start reading this book to explore the classification algorithms in one of the most popular open source projects which enjoys strong community support: Apache Mahout.

What this book covers

Chapter 1, *Classification in Data Analysis*, provides an introduction to the classification concept in data analysis. This chapter will cover the basics of classification, similarity matrix, and algorithms available in this area.

Chapter 2, *Apache Mahout*, provides an introduction to Apache Mahout and its installation process. Further, this chapter will talk about why it is a good choice for classification.

Chapter 3, *Learning Logistic Regression / SGD Using Mahout*, discusses logistic regression and Stochastic Gradient Descent, and how developers can use Mahout to use SGD.

Chapter 4, *Learning the Naïve Bayes Classification Using Mahout*, discusses the Bayes Theorem, Naïve Bayes classification, and how we can use Mahout to build Naïve Bayes classifier.

Chapter 5, *Learning the Hidden Markov Model Using Mahout*, covers the HMM and how to use Mahout's HMM algorithms.

Chapter 6, *Learning Random Forest Using Mahout*, discusses the Random forest algorithm in detail, and how to use Mahout's Random forest implementation.

Chapter 7, *Learning Multilayer Perceptron Using Mahout*, discusses Mahout as an early level implementation of a neural network. We will discuss

Multilayer Perceptron in this chapter. Further, we will use Mahout's implementation of MLP.

Chapter 8, *Mahout Changes in the Upcoming Release*, discusses Mahout as a work in progress. We will discuss the new major changes in the upcoming release of Mahout.

Chapter 9, *Building an E-mail Classification System Using Apache Mahout*, provides two use cases of e-mail classification—spam mail classification and e-mail classification based on the project the mail belongs to. We will create the model, and use this model in a program that will simulate the real working environment.

What you need for this book

To use the examples in this book, you should have the following software installed on your system:

- Java 1.6 or higher
- Eclipse
- Hadoop
- Mahout; we will discuss the installation in Chapter 2, *Apache Mahout,* of this book
- Maven, depending on how you install Mahout

Who this book is for

If you are a data scientist who has some experience with the Hadoop ecosystem and machine learning methods and want to try out classification on large datasets using Mahout, this book is ideal for you. Knowledge of Java is essential.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Extract the source code and ensure that the folder contains the `pom.xml` file."

A block of code is set as follows:

```
    public static Map<String, Integer>
readDictionary(Configuration conf, Path
dictionaryPath) {
        Map<String, Integer> dictionary = new
HashMap<String, Integer>();
        for (Pair<Text, IntWritable> pair : new
SequenceFileIterable<Text,
IntWritable>(dictionaryPath, true, conf)) {

 dictionary.put(pair.getFirst().toString(),
pair.getSecond().get());
        }
        return dictionary;
    }
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
    public static Map<String, Integer>
readDictionary(Configuration conf, Path
dictionaryPath) {
        Map<String, Integer> dictionary = new
HashMap<String, Integer>();
        for (Pair<Text, IntWritable> pair : new
SequenceFileIterable<Text,
IntWritable>(dictionaryPath, true, conf)) {

 dictionary.put(pair.getFirst().toString(),
pair.getSecond().get());
        }
        return dictionary;
    }
```

Any command-line input or output is written as follows:

```
hadoop fs -mkdir /user/hue/KDDTrain
hadoop fs -mkdir /user/hue/KDDTest
hadoop fs –put /tmp/KDDTrain+_20Percent.arff
 /user/hue/KDDTrain
hadoop fs –put /tmp/KDDTest+.arff
 /user/hue/KDDTest
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Now, navigate to the location for **mahout-distribution-0.9** and click on **Finish**."

> ## Note
>
> Warnings or important notes appear in a box like this.

> ## Tip
>
> Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail <feedback@packtpub.com>, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from http://www.packtpub.com/sites/default/files/downloads/4959OS_ColoredImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting http://www.packtpub.com/submit-errata, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to https://www.packtpub.com/books/content/support and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any

form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at <copyright@packtpub.com> with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at <questions@packtpub.com>, and we will do our best to address the problem.

Chapter 1. Classification in Data Analysis

In the last decade, we saw a huge growth in social networking and e-commerce sites. I am sure that you must have got information about this book on Facebook, Twitter, or some other site. Chances are also high that you are reading an e-copy of this book after ordering it on your phone or tablet.

This must give you an idea of how much data we are generating over the Internet every single day. Now, in order to obtain all necessary information from the data, we not only create data but also store this data. This data is extremely useful to get some important insights into the business. The analysis of this data can increase the customer base and create profits for the organization. Take the example of an e-commerce site. You visit the site to buy some book. You get information about books on related topics or the same topic, publisher, or writer, and this helps you to take better decisions, which also helps the site to know more about its customers. This will eventually lead to an increase in sales.

Finding related items or suggesting a new item to the user is all part of the data science in which we analyze the data and try to get useful patterns.

Data analysis is the process of inspecting historical data and creating models to get useful information that is required to help in decision making. It is helpful in many industries, such as e-commerce, banking, finance, healthcare, telecommunications, retail, oceanography, and many more.

Let's take the example of a weather forecasting system. It is a system that can predict the state of the atmosphere at a particular location. In this process, scientists collect historical data of the atmosphere of that location and try to create a model based on it to predict how the atmosphere will evolve over a period of time.

In machine learning, classification is the automation of the decision-making process that learns from examples of the past and emulates those decisions automatically. Emulating the decisions automatically is a core concept in predictive analytics. In this chapter, we will look at the following points:

- Understanding classification
- Working of classification systems
- Classification algorithms
- Model evaluation methods

Introducing the classification

The word classification always reminds us of our biology class, where we learned about the classification of animals. We learned about different categories of animals, such as mammals, reptiles, birds, amphibians, and so on.

If you remember how these categories are defined, you will realize that there were certain properties that scientists found in existing animals, and based on these properties, they categorized a new animal.

Other real-life examples of classification could be, for instance, when you visit the doctor. He/she asks you certain questions, and based on your answers, he/she is able to identify whether you have a certain disease or not.

Classification is the categorization of potential answers, and in machine learning, we want to automate this process. Biological classification is an example of **multiclass** classification and finding the disease is an example of **binary** classification.

In data analysis, we want to use machine learning concepts. To analyze the data, we want to build a system that can help us to find out which class an individual item belongs to. Usually, these classes are mutually exclusive. A related problem in this area is finding out the probability that an individual belongs to a certain class.

Classification is a supervised learning technique. In this technique, machines—based on historical data—learn and gain the capabilities to predict the unknown. In machine learning, another popular technique is unsupervised learning. In supervised learning, we already know the output categories, but in unsupervised learning, we know nothing about the output. Let's understand this with a quick example: suppose we have a fruit basket, and we want to classify fruits. When we say classify, it means that in the training data, we already have output variables, such as size and color, and we know whether the color is red and the size is from 2.3" to 3.7". We will classify that fruit as an apple. Opposite to this, in unsupervised learning, we want to separate different fruits, and we do not have any output information in the training dataset, so the learning algorithm will separate different fruits based on different features present in the dataset, but it will not be able to label them. In other words, it will not be able to tell which one is an apple and which one is a banana, although it will be able to separate them.

Application of the classification system

Classification is used for prediction. In the case of e-mail categorization, it is used to classify e-mail as spam or not spam. Nowadays, Gmail is

classifying e-mails as primary, social, and promotional as well. Classification is useful in predicting credit card frauds, to categorize customers for eligibility of loans, and so on. It is also used to predict customer churn in the insurance and telecom industries. It is useful in the healthcare industry as well. Based on historical data, it is useful in classifying particular symptoms of a disease to predict the disease in advance. Classification can be used to classify tropical cyclones. So, it is useful across all industries.

Working of the classification system

Let's understand the classification process in more detail. In the process of classification, with the dataset given to us, we try to find out informative variables using which we can reduce the uncertainty and categorize something. These informative variables are called **explanatory variables** or features.

The final categories that we are interested are called target variables or labels. Explanatory variables can be any of the following forms:

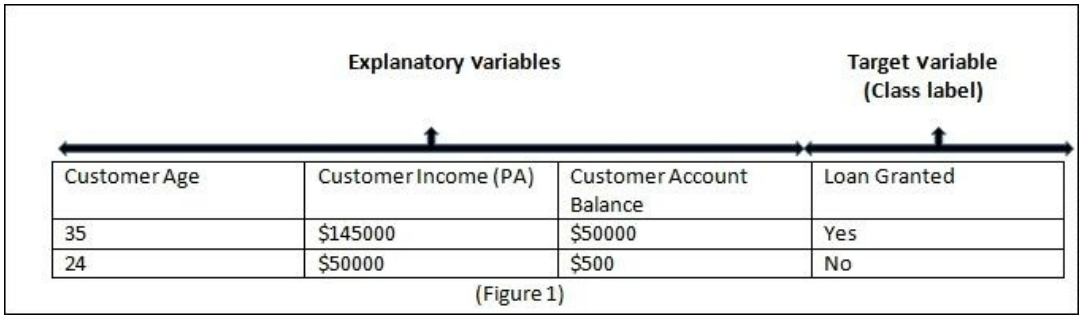
- Continuous (numeric types)
- Categorical
- Word-like
- Text-like

> **Note**
>
> If numeric types are not useful for any mathematical functions, those will be counted as categorical (zip codes, street numbers, and so on).

So, for example, we have a dataset of customer's' loan applications, and we want to build a classifier to find out whether a new customer is eligible for a loan or not. In this dataset, we can have the following fields:

- **Customer Age**
- **Customer Income (PA)**
- **Customer Account Balance**
- **Loan Granted**

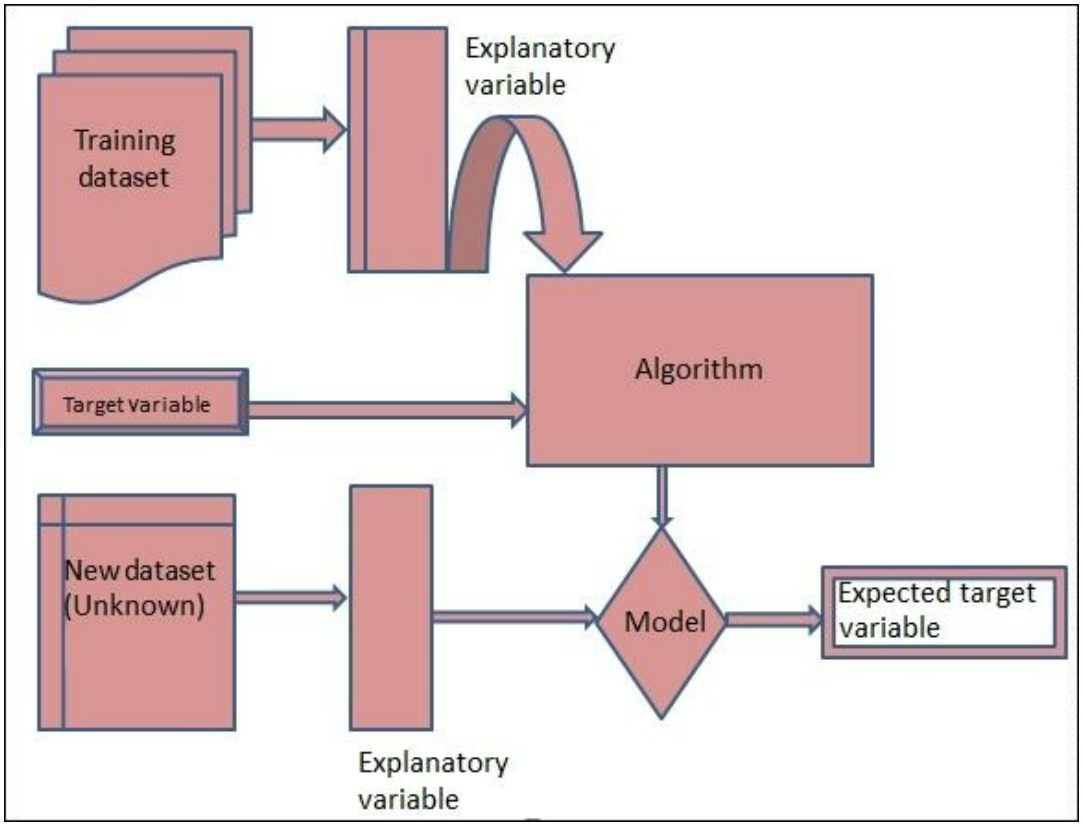
From these fields, **Customer Age**, **Customer Income (PA)** and **Customer Account Balance** will work as explanatory variables and **Loan Granted** will be the target variable, as shown in the following screenshot:



Customer Age	Customer Income (PA)	Customer Account Balance	Loan Granted
35	$145000	$50000	Yes
24	$50000	$500	No

(Figure 1)

To understand the creation of the classifier, we need to understand a few terms, as shown in the following diagram:



- **Training dataset**: From the given dataset, a portion of the data is used to create the training dataset (it could be 70 percent of the given data). This dataset is used to build the classifier. All the feature sets are used in this dataset.
- **Test dataset**: The dataset that is left after the training dataset is used to test the created model. With this data, only the feature set is used and the model is used to predict the target variables or labels.
- **Model**: This is used to understand the algorithm used to generate the target variables.

While building a classifier, we follow these steps:

- Collecting historical data

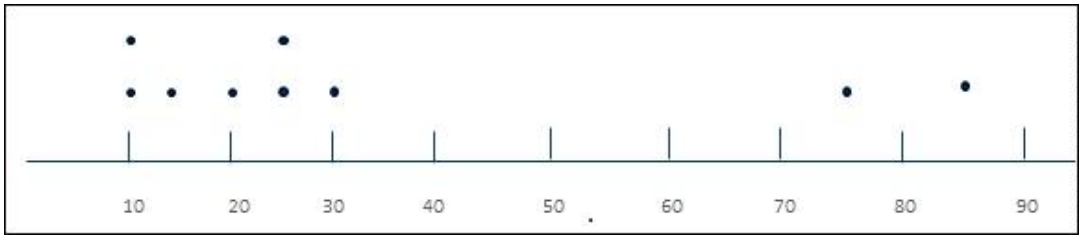
- Cleaning data (a lot of activities are involved here, such as space removal, and so on)
- Defining target variables
- Defining explanatory variables
- Selecting an algorithm
- Training the model (using the training dataset)
- Running test data
- Evaluating the model
- Adjusting explanatory variables
- Rerunning the test

While preparing the model, one should take care of outlier detection. **Outlier detection** is a method to find out items that do not conform to an expected pattern in a dataset. Outliers in an input dataset can mislead the training process of an algorithm. This can affect the model accuracy. There are algorithms to find out these outliers in the datasets. Distance-based techniques and fuzzy-logic-based methods are mostly used to find out outliers in the dataset. Let's talk about one example to understand the outliers.

We have a set of numbers, and we want to find out the mean of these numbers:

10, 75, 10, 15, 20, 85, 25, 30, 25

Just plot these numbers and the result will be as shown in the following screenshot:



Clearly, the numbers 75 and 85 are outliers (far away in the plot from the other numbers).

Mean = sum of values/number of values = 32.78

Mean without the outliers: = 19.29

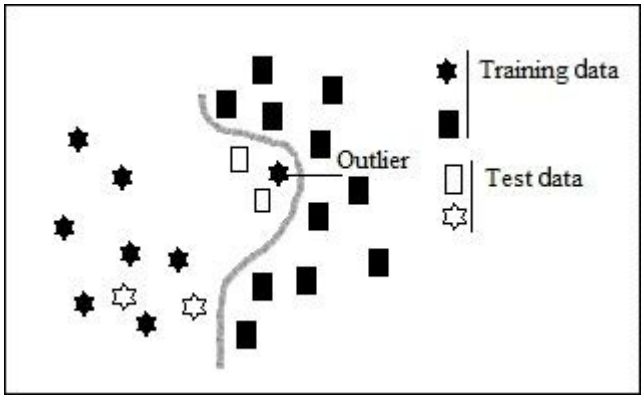
So, now you can understand how outliers can affect the results.

While creating the model, we can encounter two majorly occurring problems —**Overfitting** and **Underfitting**.

Overfitting occurs when the algorithm captures the noise of the data, and the algorithm fits the data too well. Generally, it occurs if we use all the

given data to build the model using pure memorization. Instead of finding out the generalizing pattern, the model just memorizes the pattern. Usually, in the case of overfitting, the model gets more complex, and it is allowed to pick up spurious correlations. These correlations are specific to training datasets and do not represent characteristics of the whole dataset in general.

The following diagram is an example of overfitting. An outlier is present, and the algorithm considers that and creates a model that perfectly classifies the training set, but because of this, the test data is wrongly classified (both the rectangles are classified as stars in the test data):



There is no single method to avoid overfitting; however, we have some approaches, such as a reduction in the number of features and the regularization of a few of the features. Another way is to train the model with some dataset and test with the remaining dataset. A common method called cross-validation is used to generate multiple performance measures. In this way, a single dataset is split and used for the creation of performance measures.

Underfitting occurs when the algorithm cannot capture the patterns in the data, and the data does not fit well. Underfitting is also known as high bias. It means your algorithm has such a strong bias towards its hypothesis that it does not fit the data well. For an underfitting error, more data will not help. It can increase the training error. More explanatory variables can help to deal with the underfitting problem. More explanatory fields will expand the hypothesis space and will be useful to overcome this problem.

Both overfitting and underfitting provide poor results with new datasets.

Classification algorithms

We will now discuss the following algorithms that are supported by Apache Mahout in this book:

- **Logistic regression / Stochastic Gradient Descent (SGD)**: We usually read regression along with classification, but actually, there is a difference between the two. Classification involves a categorical target variable, while regression involves a numeric target variable. Classification predicts whether something will happen, and regression predicts how much of something will happen. We will cover this algorithm in Chapter 3, *Learning Logistic Regression / SGD Using Mahout*. Mahout supports logistic regression trained via Stochastic Gradient Descent.
- **Naïve Bayes classification**: This is a very popular algorithm for text classification. Naïve Bayes uses the concept of probability to classify new items. It is based on the Bayes theorem. We will discuss this algorithm in Chapter 4, *Learning the Naïve Bayes Classification Using Mahout*. In this chapter, we will see how Mahout is useful in classifying text, which is required in the data analysis field. We will discuss vectorization, bag of words, n-grams, and other terms used in text classification.
- **Hidden Markov Model (HMM)**: This is used in various fields, such as speech recognition, parts-of-speech tagging, gene prediction, time-series analysis, and so on. In HMM, we observe a sequence of emissions but do not have a sequence of states which a model uses to generate the emission. In Chapter 5, *Learning the Hidden Markov Model Using Mahout*, we will take one more algorithm supported by Mahout Hidden Markov Model. We will discuss HMM in detail and see how Mahout supports this algorithm.
- **Random Forest**: This is the most widely used algorithm in classification. Random Forest consists of a collection of simple tree predictors, each capable of producing a response when presented with a set of explanatory variables. In Chapter 6, *Learning Random Forest Using Mahout*, we will discuss this algorithm in detail and also talk about how to use Mahout to implement this algorithm.
- **Multi-layer Perceptron (MLP)**: In Chapter 7, *Learning Multilayer Perceptron Using Mahout*, we will discuss this newly implemented algorithm in Mahout. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. It is a base for the implementation of neural networks. We will discuss neural networks a little but only after a detailed discussion on MLP in Mahout.

We will discuss all the classification algorithms supported by Apache Mahout in this book, and we will also check the model evaluation techniques provided by Apache Mahout.

Model evaluation techniques

We cannot have a single evaluation metric that can fit all the classifier models, but we can find out some common issues in evaluation, and we have techniques to deal with them. We will discuss the following techniques that are used in Mahout:

- Confusion matrix
- ROC graph
- AUC
- Entropy matrix

The confusion matrix

The confusion matrix provides us with the number of correct and incorrect predictions made by the model compared with the actual outcomes (target values) in the data. A confusion matrix is a N*N matrix, where N is the number of labels (classes). Each column is an instance in the predicted class, and each row is an instance in the actual class. Using this matrix, we can find out how one class is confused with another. Let's assume that we have a classifier that classifies three fruits: strawberries, cherries, and grapes. Assuming that we have a sample of 24 fruits: 7 strawberries, 8 cherries, and 9 grapes, the resulting confusion matrix will be as shown in the following table:

Predicted classes by model				
Actual class		**Strawberries**	**Cherries**	**Grapes**
	Strawberries	4	3	0
	Cherries	2	5	1
	Grapes	0	1	8

So, in this model, from the 8 strawberries, 3 were classified as cherries. From the 8 cherries, 2 were classified as strawberries, and 1 is classified as a grape. From the 9 grapes, 1 is classified as a cherry. From this matrix, we

will create the table of confusion. The table of confusion has two rows and two columns that report about true positive, true negative, false positive, and false negative.

So, if we build this table for a particular class, let's say for strawberries, it would be as follows:

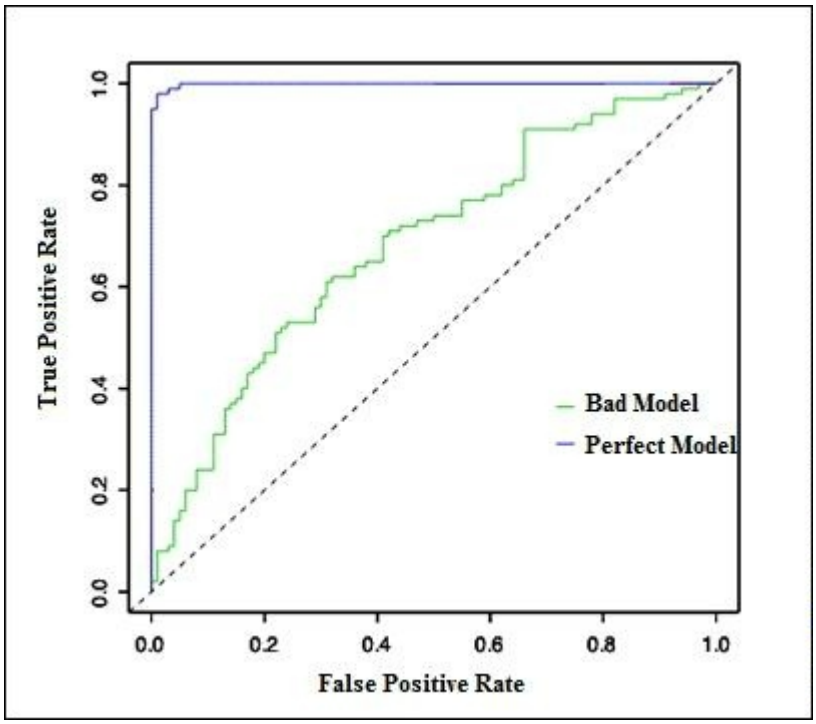
True Positive 4 (actual strawberries classified correctly) (a)	**False Positive** 2 (cherries that were classified as strawberries)(b)
False Negative 3 (strawberries wrongly classified as cherries) (c)	**True Negative** 15 (all other fruits correctly not classified as strawberries) (d)

Using this table of confusion, we can find out the following terms:

- **Accuracy**: This is the proportion of the total number of predictions that were correctly classified. It is calculated as (True Positive + True Negative) / Positive + Negative. Therefore, *accuracy = (a+d)/(a+b+c +d)*.
- **Precision or positive predictive value**: This is the proportion of positive cases that were correctly classified. It is calculated as (True Positive)/(True Positive + False Positive). Therefore, *precision = a/(a +b)*.
- **Negative predictive value**: This is the proportion of negative cases that were classified correctly. It is calculated as True Negative/(True Negative + False Negative). Therefore, *negative predictive value = d/ (c+d)*.
- **Sensitivity / true positive rate / recall**: This is the proportion of the actual positive cases that were correctly identified. It is calculated as True Positive/(True Positive + False Negative). Therefore, *sensitivity = a/(a+c)*.
- **Specificity**: This is the proportion of the actual negative cases. It is calculated as *True Negative/(False Positive + True Negative)*. Therefore, *specificity =d /(b+d)*.
- **F1 score**: This is the measure of a test's accuracy, and it is calculated as follows: *F1 = 2.((Positive predictive value (precision) * sensitivity (recall))/(Positive predictive value (precision) +sensitivity (recall)))*.

The Receiver Operating Characteristics (ROC) graph

ROC is a two-dimensional plot of a classifier with false positive rate on the *x* axis and true positive rate on the *y* axis. The lower point (0,0) in the figure represents never issuing a positive classification. Point (0,1) represents perfect classification. The diagonal from (0,0) to (1,1) divides the ROC space. Points above the diagonal represent good classification results, and points below the line represent poor results, as shown in the following diagram:



Area under the ROC curve

This is the area under the ROC curve and is also known as AUC. It is used to measure the quality of the classification model. In practice, most of the classification models have an AUC between 0.5 and 1. The closer the value is to 1, the greater is your classifier.

The entropy matrix

Before going into the details of the entropy matrix, first we need to understand **entropy**. The concept of entropy in information theory was developed by Shannon.

Entropy is a measure of disorder that can be applied to a set. It is defined as:

Entropy = -p1log(p1) – p2log(p2)- …….

Each p is the probability of a particular property within the set. Let's revisit our customer loan application dataset. For example, assuming we have a set of 10 customers from which 6 are eligible for a loan and 4 are not. Here, we have two properties (classes): eligible or not eligible.

P(eligible) = 6/10 = 0.6

P(not eligible) = 4/10 = 0.4

So, entropy of the dataset will be:

*Entropy = -[0.6*log2(0.6)+0.4*log2(0.4)]*

= -[0.6-0.74 +0.4*-1.32]*

= 0.972

Entropy is useful in acquiring knowledge of information gain. Information gain measures the change in entropy due to any new information being added in model creation. So, if entropy decreases from new information, it indicates that the model is performing well now. Information gain is calculated as:

*IG (classes , subclasses) = entropy(class) – (p(subclass1)*entropy(subclass1)+ p(subclass2)*entropy(subclass2) + …)*

Entropy matrix is basically the same as the confusion matrix defined earlier; the only difference is that the elements in the matrix are the averages of the log of the probability score for each true or estimated category combination. A good model will have small negative numbers along the diagonal and will have large negative numbers in the off-diagonal position.

Summary

We have discussed classification and its applications and also what algorithm and classifier evaluation techniques are supported by Mahout. We discussed techniques like confusion matrix, ROC graph, AUC, and entropy matrix.

Now, we will move to the next chapter and set up Apache Mahout and the developer environment. We will also discuss the architecture of Apache Mahout and find out why Mahout is a good choice for classification.

Chapter 2. Apache Mahout

In the previous chapter, we discussed classification and looked into the algorithms provided by Mahout in this area. Before going to those algorithms, we need to understand Mahout and its installation. In this chapter, we will explore the following topics:

- What is Apache Mahout?
- Algorithms supported in Mahout
- Why is it a good choice for classification problems?
- Setting up the system for Mahout development

Introducing Apache Mahout

A mahout is a person who rides and controls an elephant. Most of the algorithms in Apache Mahout are implemented on top of Hadoop, which is another Apache-licensed project and has the symbol of an elephant (http://hadoop.apache.org/). As Apache Mahout rides over Hadoop, this name is justified.

Apache Mahout is a project of Apache Software Foundation that has implementations of machine learning algorithms. Mahout was started as a subproject of the Apache Lucene project in 2008. After some time, an open source project named **Taste**, which was developed for collaborative filtering, and it was absorbed into Mahout. Mahout is written in Java and provides scalable machine learning algorithms. Mahout is the default choice for machine learning problems in which the data is too large to fit into a single machine. Mahout provides Java libraries and does not provide any user interface or server. It is a framework of tools to be used and adapted by developers.



To sum it up, Mahout provides you with implementations of the most frequently used machine learning algorithms in the area of classification, clustering, and recommendation. Instead of spending time writing algorithms, it provides us with ready-to-consume solutions.

Mahout uses Hadoop for its algorithms, but some of the algorithms can also run without Hadoop. Currently, Mahout supports the following use cases:

- **Recommendation**: This takes the user data and tries to predict items that the user might like. With this use case, you can see all the sites that are selling goods to the user. Based on your previous action, they will try to find out unknown items that could be of use. One example can be this: as soon as you select some book from Amazon, the website will show you a list of other books under the title, **Customers Who Bought This Item Also Bought**. It also shows the title, **What Other Items Do Customers Buy After Viewing This Item?** Another example of recommendation is that while playing videos on YouTube, it recommends that you listen to some other videos based on your selection. Mahout provides full API support to develop your own user-based or item-based recommendation engine.
- **Classification**: As defined in the earlier chapter, classification decides how much an item belongs to one particular category. E-mail classification for filtering out spam is a classic example of classification. Mahout provides a rich set of APIs to build your own classification model. For example, Mahout can be used to build a document classifier or an e-mail classifier.
- **Clustering**: This is a technique that tries to group items together based on some sort of similarity. Here, we find the different clusters of items based on certain properties, and we do not know the name of the cluster in advance. The main difference between clustering and classification is that in classification, we know the end class name. Clustering is useful in finding out different customer segments. Google News uses the clustering technique in order to group news. For clustering, Mahout has already implemented some of the most popular algorithms in this area, such as k-means, fuzzy k-means, canopy, and so on.
- **Dimensional reduction**: As we discussed in the previous chapter, features are called dimensions. Dimensional reduction is the process of reducing the number of random variables under consideration. This makes data easy to use. Mahout provides algorithms for dimensional reduction. Singular value decomposition and Lanczos are examples of the algorithms that Mahout provides.
- **Topic modeling**: Topic modeling is used to capture the abstract idea of a document. A topic model is a model that associates probability distribution with each document over topics. Given that a document is about a particular topic, one would expect particular words to appear in the document more or less frequently. "Football" and "goal" will

appear more in a document about sports. **Latent Dirichlet Allocation** (**LDA**) is a powerful learning algorithm for topic modeling. In Mahout, collapsed variational Bayes is implemented for LDA.

Algorithms supported in Mahout

The implementation of algorithms in Mahout can be categorized into two groups:

- **Sequential algorithms**: These algorithms are executed sequentially and do not use Hadoop scalable processing. They are usually the ones derived from Taste. For example: user-based collaborative filtering, logistic regression, Hidden Markov Model, multi-layer perceptron, singular value decomposition.
- **Parallel algorithms**: These algorithms can support petabytes of data using Hadoop's map and hence reduce parallel processing. For example, Random Forest, Naïve Bayes, canopy clustering, k-means clustering, spectral clustering, and so on.

Reasons for Mahout being a good choice for classification

In machine learning systems, the more data you use, the more accurate the system built will be. Mahout, which uses Hadoop for scalability, is way ahead of others in terms of handling huge datasets. As the number of training sets increases, Mahout's performance also increases. If the input size for training examples is from 1 million to 10 million, then Mahout is an excellent choice.

For classification problems, increased data for training is desirable as it can improve the accuracy of the model. Generally, as the number of datasets increases, memory requirement also increases, and algorithms become slow, but Mahout's scalable and parallel algorithms work better with regards to the time taken. Each new machine added decreases the training time and provides higher performance.

Installing Mahout

Now let's try the slightly challenging part of this book: Mahout installation. Based on common experiences, I have come up with the following questions or concerns that users face before installation:

- I do not know anything about Maven. How will I compile Mahout build?
- How can I set up Eclipse to write my own programs in Mahout?
- How can I install Mahout on a Windows system?

So, we will install Mahout with the help of the following steps. Each step is independent from the other. You can choose any one of these:

- Building Mahout code using Maven
- Setting up a development environment using Eclipse
- Setting up Mahout for a Windows user

Before any of the steps, some of the prerequisites are:

- You should have Java installed on your system. Wikihow is a good source for this at http://www.wikihow.com/Install-Java-on-Linux
- You should have Hadoop installed on your system from the http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleNodeSetup.html URL

Building Mahout from source using Maven

Mahout's build and release system is based on Maven.

Installing Maven

1. Create the folder `/usr/local/maven`, as follows:

```
mkdir /usr/local/maven
```

2. Download the distribution `apache-maven-x.y.z-bin.tar.gz` from the Maven site (http://maven.apache.org/download.cgi) and move this to `/usr/local/maven`, as follows:

```
mv
apache-maven-x.y.z-bin.tar.gz /usr/local/maven
```

3. Unpack to the location `/usr/local/maven`, as follows:

```
tar -xvf apache-maven-x.y.z-bin.tar.gz
```

4. Edit the `.bashrc` file, as follows:

```
export M2_HOME=/usr/local/apache-maven-x.y.z
export M2=$M2_HOME/bin
export PATH=$M2:$PATH
```

> **Note**
>
> For the Eclipse IDE, go to **Help** and select **Install new Software**. Click on the **Add** button, and in the pop up, type the name `M2Eclipse`, provide the link http://download.eclipse.org/technology/m2e/releases, and click on **OK**.

Building Mahout code

By default, Mahout assumes that Hadoop is already installed on the system. Mahout uses the `HADOOP_HOME` and `HADOOP_CONF_DIR` environment variables to access Hadoop cluster configurations. For setting up Mahout, execute the following steps:

1. Download the Mahout distribution file `mahout-distribution-0.9-src.tar.gz` from the location http://archive.apache.org/dist/mahout/0.9/.
2. Choose an installation directory for Mahout (`/usr/local/Mahout`), and place the downloaded source in the folder. Extract the source code and ensure that the folder contains the `pom.xml` file. The following is the exact location of the source:

```
tar -xvf  mahout-distribution-0.9-src.tar.gz
```

3. Install the Mahout Maven project, and skip the test cases while installing, as follows:

```
mvn install -Dmaven.test.skip=true
```

4. Set the `MAHOUT_HOME` environment variable in the `~/.bashrc` file, and update the `PATH` variable with the `Mahout bin` directory:

```
export
MAHOUT_HOME=/user/local/mahout/mahout-
distribution-0.9
export PATH=$PATH:$MAHOUT_HOME/bin
```

5. To test the Mahout installation, execute the command: `mahout`. This will list the available programs within the distribution bundle, as shown in the following screenshot:

```
MAHOUT_LOCAL is not set; adding HADOOP_CONF_DIR to classpath.
Running on hadoop, using /usr/lib/hadoop/bin/hadoop and HADOOP_CONF_DIR=/etc/hadoop/conf
MAHOUT-JOB: /usr/lib/mahout/mahout-examples-0.9.0.2.1.1.0-385-job.jar
An example program must be given as the first argument.
Valid program names are:
  arff.vector: : Generate Vectors from an ARFF file or directory
  baumwelch: : Baum-Welch algorithm for unsupervised HMM training
  canopy: : Canopy clustering
  cat: : Print a file or resource as the logistic regression models would see it
  cleansvd: : Cleanup and verification of SVD output
  clusterdump: : Dump cluster output to text
  clusterpp: : Groups Clustering Output In Clusters
  cmdump: : Dump confusion matrix in HTML or text formats
  concatmatrices: : Concatenates 2 matrices of same cardinality into a single matrix
  cvb: : LDA via Collapsed Variation Bayes (0th deriv. approx)
  cvb0_local: : LDA via Collapsed Variation Bayes, in memory locally.
  evaluateFactorization: : compute RMSE and MAE of a rating matrix factorization against probes
  fkmeans: : Fuzzy K-means clustering
  hmmpredict: : Generate random sequence of observations by given HMM
  itemsimilarity: : Compute the item-item-similarities for item-based collaborative filtering
  kmeans: : K-means clustering
  lucene.vector: : Generate Vectors from a Lucene index
  lucene2seq: : Generate Text SequenceFiles from a Lucene index
  matrixdump: : Dump matrix in CSV format
  matrixmult: : Take the product of two matrices
  parallelALS: : ALS-WR factorization of a rating matrix
  qualcluster: : Runs clustering experiments and summarizes results in a CSV
  recommendfactorized: : Compute recommendations using the factorization of a rating matrix
  recommenditembased: : Compute recommendations using item-based collaborative filtering
  regexconverter: : Convert text files on a per line basis based on regular expressions
  resplit: : Splits a set of SequenceFiles into a number of equal splits
```

Setting up a development environment using Eclipse

For this setup, you should have Maven installed on the system and the Maven plugin for Eclipse. Refer to the *Installing Maven* step explained in the previous section. This setup can be done in the following steps:

1. Download the Mahout distribution file `mahout-distribution-0.9-src.tar.gz` from the location http://archive.apache.org/dist/mahout/0.9/ and unzip this:

```
tar xzf mahout-distribution-0.9-src.tar.gz
```

2. Let's create a folder named `workspace` under `/usr/local/workspace`, as follows:

```
mkdir /usr/local/workspace
```

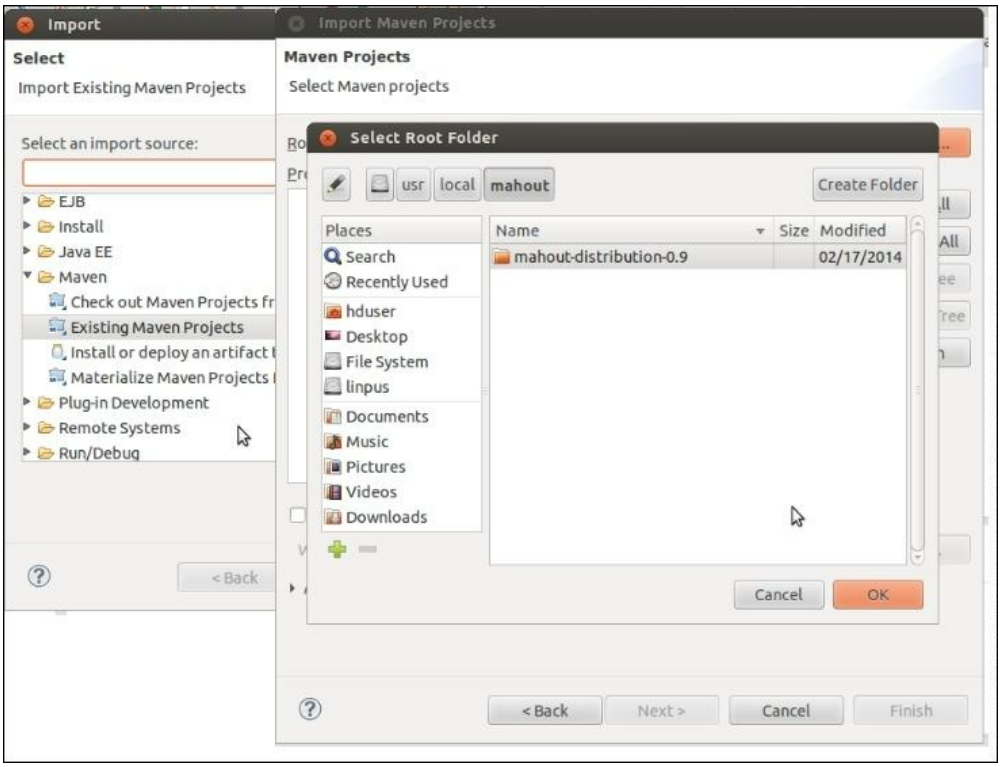
3. Move the downloaded distribution to this folder (from the downloads folder), as follows:

```
mv
mahout-distribution-0.9 /usr/local/workspace/
```

4. Move to the folder `/usr/local/workspace/mahout-distribution-0.9` and make an Eclipse project (this command can take up to an hour):

```
mvn eclipse:eclipse
```

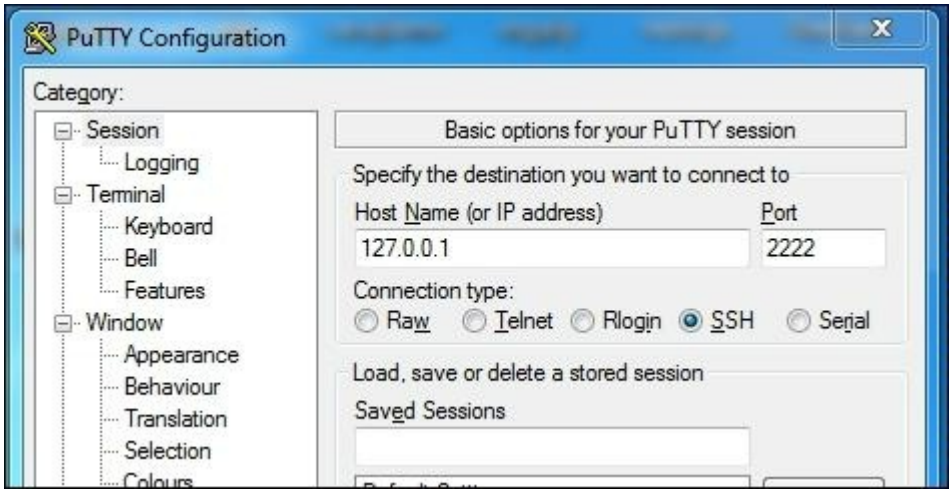
5. Set the Mahout home in the `.bashrc` file, as explained earlier in the *Building Mahout code* section.
6. Now open Eclipse. Select the file, import Maven, and **Existing Maven Projects**. Now, navigate to the location for **mahout-distribution-0.9** and click on **Finish**.



Setting up Mahout for a Windows user

A Windows user can use Cygwin (a large collection of GNU and open source tools that provides functionality similar to a Linux distribution on Windows) to set up their environment. There is also another way that is easy to use, as shown in the following steps:

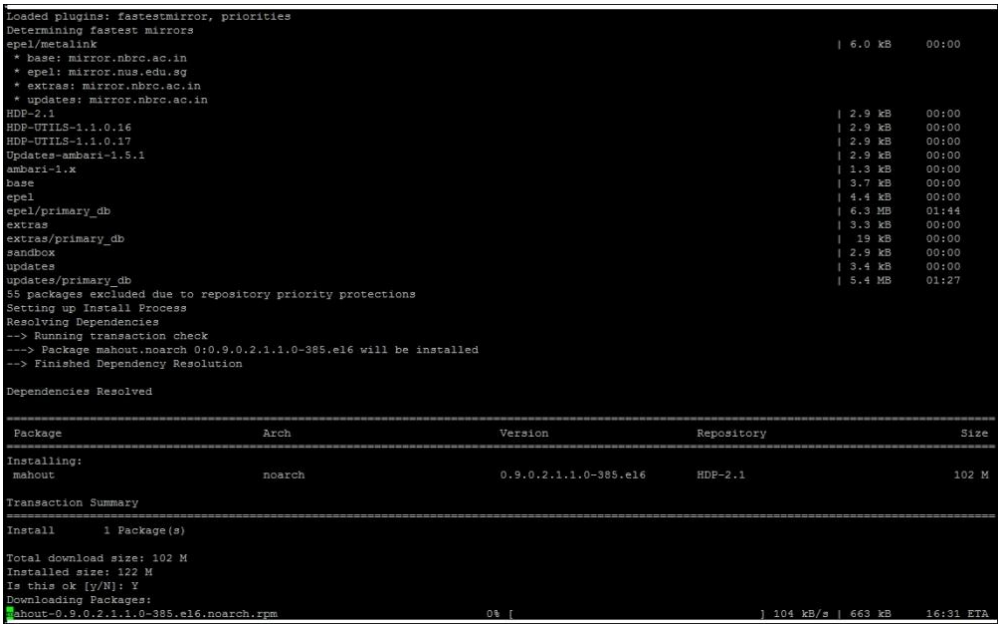
1. Download Hortonworks Sandbox for virtual box on your system from the location http://hortonworks.com/products/hortonworks-sandbox/#install. Hortonworks Sandbox on your system will be a pseudo-distributed mode of Hadoop.
2. Log in to the console. Use *Alt + F5* or alternatively download Putty and provide **127.0.0.1** as the hostname and **2222** in the port, as shown in the following figure. Log in with the username `root` and password `-hadoop`.



3. Enter the following command:

```
yum install mahout
```

Now, you will see a screen like this:



4. Enter `y`, and your Mahout will start installing. Once this is done, you can test by typing the command `mahout` and this will show you the same screen as shown in the *Setting up a development environment using Eclipse* recipe seen earlier.

Summary

We discussed Apache Mahout in detail in this chapter. We covered the process of installing Mahout on our system, along with setting up a development environment that is ready to execute Mahout algorithms. We have also taken a look at the reasons behind Mahout being considered a good choice for classification. Now, we move to the next where we will understand about logistic regression and learn about the process that needs to be followed to execute our first algorithm in Mahout.

Chapter 3. Learning Logistic Regression / SGD Using Mahout

Instead of jumping directly into logistic regression, let's try to understand a few of its concepts. In this chapter, we will explore the following topics:

- Introducing regression
- Understanding linear regression
- Cost function
- Gradient descent
- Logistic regression
- Understanding SGD
- Using Mahout for logistic regression

Introducing regression

Regression analysis is used for prediction and forecasting. It is used to find out the relationship between explanatory variables and target variables. Essentially, it is a statistical model that is used to find out the relationship among variables present in the datasets. An example that you can refer to for a better understanding of this term is this: determine the earnings of workers in a particular industry. Here, we will try to find out the factors that affect a worker's salary. These factors can be age, education, years of experience, particular skill set, location, and so on. We will try to make a model that will take all these variables into consideration and try to predict the salary. In regression analysis, we characterize the variation of the target variable around the regression function, which can be described by a probability distribution that is also of interest. There are a number of regression analysis techniques that are available. For example, linear regression, ordinary least squares regression, logistic regression, and so on.

Understanding linear regression

In linear regression, we create a model to predict the value of a target variable with the help of an explanatory variable. To understand this better, let's look at an example.

A company X that deals in selling coffee has noticed that in the month of monsoon, their sales increased to quite an extent. So they have come up with a formula to find the relation between rain and their per cup coffee sale, which is shown as follows:

$C = 1.5R+800$

So, for 2 mm of rain, there is a demand of 803 cups of coffee. Now if you go into minute details, you will realize that we have the data for rainfall and per cup coffee sale, and we are trying to build a model that can predict the demand for coffee based on the rainfall. We have data in the form of *(R1, C1), (R2, C2)…. (Ri, Ci)*. Here, we will build the model in a manner that keeps the error in the actual and predicted values at a minimum.

Cost function

In the equation *C = 1.5R+800*, the two values 1.5 and 800 are parameters and these values affect the end result. We can write this equation as *C= p0+p1R*. As we discussed earlier, our goal is to reduce the difference between the actual value and the predicted value, and this is dependent on the values of *p0* and *p1*. Let's assume that the predicted value is *Cp* and the actual value is *C* so that the difference will be *(Cp-C)*. This can be written as *(p0+p1R-C)*.To minimize this error, we define the error function, which is also called the **cost function**.

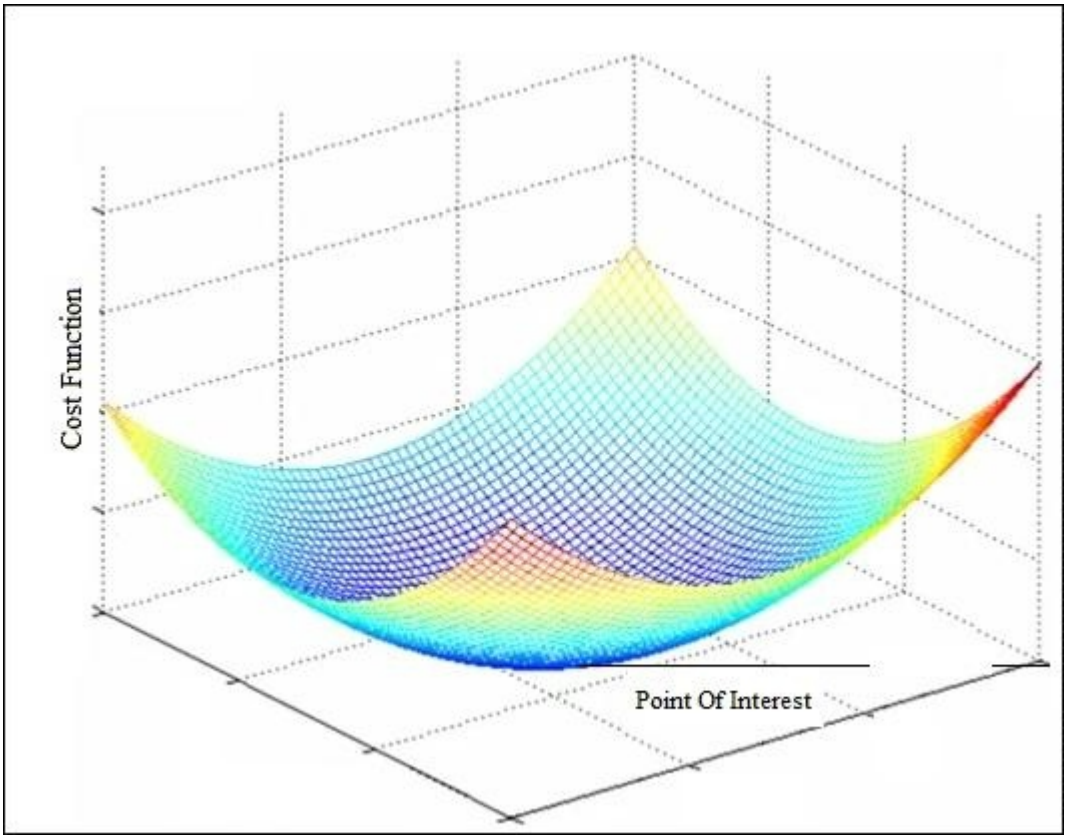
The cost function can be defined with the following formula:

$$\text{Cost Function}(p_0, p_1) = \frac{1}{N}\sum_{i=1}^{N}\left((p_0 + p_1 R_i) - C_i\right)^2$$

Here, **i** is the ith sample and **N** is the number of training examples. We calculate costs for different sets of **p0** and **p1** and finally select the **p0** and **p1** that gives the least cost (*C*). This is the model that will be used to make predictions for new input.

Gradient descent

Gradient descent starts with an initial set of parameter values, *p0* and *p1*, and iteratively moves towards a set of parameter values that minimizes the cost function. We can visualize this error function graphically, where width and length can be considered as the parameters *p0* and *p1* and height as the cost function. Our goal is to find the values for *p0* and *p1* in a way that our cost function will be minimal. We start the algorithm with some values of *p0* and *p1* and iteratively work towards the minimum value. A good way to ensure that the gradient descent is working correctly is to make sure that the cost function decreases for each iteration. In this case, the cost function surface is convex and we will try to find out the minimum value. This can be seen in the following figure:



Logistic regression

Logistic regression is used to ascertain the probability of an event. Generally, logistic regression refers to problems where the outcome is binary, for example, in building a model that is based on a customer's income, travel uses, gender, and other features to predict whether he or she will buy a particular car or not. So, the answer will be a simple yes or no. When the outcome is composed of more than one category, this is called **multinomial logistic regression**.

Logistic regression is based on the **sigmoid function**. Predictor variables are combined with linear weight and then passed to this function, which generates the output in the range of 0–1. An output close to 1 indicates that an item belongs to a certain class. Let's first understand the sigmoid or logistic function. It can be defined by the following formula:

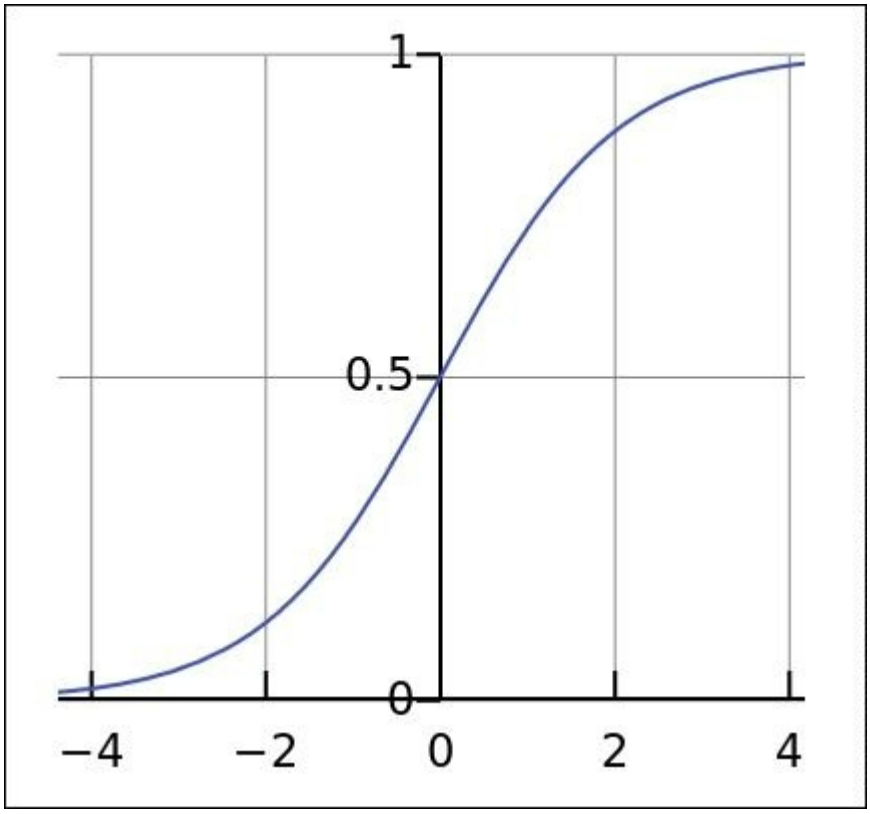
$F(z) = 1/1+e(-z)$

With a single explanatory variable, z will be defined as $z = \beta 0 + \beta 1 * x$. This equation is explained as follows:

- **z**: This is called the dependent variable. This is the variable that we would like to predict. During the creation of the model, we have this variable with us in the training set, and we build the model to predict this variable. The known values of z are called observed values.

- **x**: This is the explanatory or independent variable. These variables are used to predict the dependent variable z. For example, to predict the sales of a newly launched product at a particular location, we might include explanatory variables such as the price of the product, the average income of the people of that location, and so on.
- **β0**: This is called the regression intercept. If all explanatory variables are zero, then this parameter is equal to the dependent variable z.
- **β1**: These are values for each explanatory variable.

The graph of the logistic function is as follows:



With a little bit of mathematics, we can change this equation as follows:

*ln(F(x)/(1-F(x)) = β0 + β1*x*

In the case of linear regression, the cost function graph was convex, but here, it is not going to be convex. Finding the minimum values for parameters in a way that our predicted output is close to the actual one will be difficult. In a cost function, while calculating for logistic regression, we will replace our *Cp* value of linear regression with the function *F(z)*. To make convex logistic regression cost functions, we will replace *(p0+p1Ri-Ci)2* with one of the following:

- *log (1/1+e (-(β0 + β1*x)))* if the actual occurrence of an event is 1, this function will represent the cost.

- *log (1-(1/1+e (-(β0 + β1*x))))* if the actual occurrence of an event is 0, this function will represent the cost.

We will have to remember that in logistic regression, we calculate the class probability. So, if the probability of an event occurring (customer buying a car, being defrauded, and so on) is *p*, the probability of non-occurrence is *1-p*.

Stochastic Gradient Descent

Gradient descent minimizes the cost function. For very large datasets, gradient descent is a very expensive procedure. Stochastic Gradient Descent (SGD) is a modification of the gradient descent algorithm to handle large datasets. Gradient descent computes the gradient using the whole dataset, while SGD computes the gradient using a single sample. So, gradient descent loads the full dataset and tries to find out the local minimum on the graph and then repeat the full process again, while SGD adjusts the cost function for every sample, one by one. A major advantage that SGD has over gradient descent is that its speed of computation is a whole lot faster. Large datasets in RAM generally cannot be held as the storage is limited. In SGD, the burden on the RAM is reduced, wherein each sample or batch of samples are loaded and worked with, the results for which are stored, and so on.

Using Mahout for logistic regression

Mahout has implementations for logistic regression using SGD. It is very easy to understand and use. So let's get started.

Dataset

We will use the **Wisconsin Diagnostic Breast Cancer** (**WDBC**) dataset. This is a dataset for breast cancer tumors and data is available from 1995 onwards. It has 569 instances of breast tumor cases and has 30 features to predict the diagnosis, which is categorized as either benign or malignant.

> ## Note
>
> More details on the preceding dataset is available at http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names.

Preparing the training and test data

You can download the `wdbc.data` dataset from http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data.

Now, save it as a CSV file and include the following header line:

```
ID_Number,Diagnosis,Radius,Texture,Perimeter,Area,Smoothness,Compactness,Concavi
```

Now, we will have to perform the following steps to prepare this data to be used by the Mahout logistic regression algorithm:

1. We will make the target class numeric. In this case, the second field diagnosis is the target variable. We will change malignant to 0 and benign to 1. Use the following code snippet to introduce the changes. We can use this strategy for small datasets, but for huge datasets, we have different strategies, which we will cover in Chapter 4, *Learning the Naïve Bayes Classification Using Mahout*:

```
public void convertTargetToInteger() throws
IOException{
  //Read the data
  BufferedReader br = new BufferedReader(new
FileReader("wdbc.csv"));
  String line =null;
  //Create the file to save the resulted data
  File wdbcData = new File("<Your Destination
location for file.>");
  FileWriter fw = new FileWriter(wdbcData);
  //We are adding header to the new file

 fw.write("ID_Number"+","+"Diagnosis"+","+"Radius"+","+"Texture"+","+

  /*In the while loop we are reading line by
line and checking the last field- parts[1]
and changing it to numeric value accordingly*/
  while((line=br.readLine())!=null){
    String []parts = line.split(",");
    if(parts[1].equals("M")){

 fw.write(parts[0]+","+"0"+","+parts[2]+","+parts[3]+","+parts[4]+",
    }

    if(parts[1].equals("B")){

 fw.write(parts[0]+","+"1"+","+parts[2]+","+parts[3]+","+parts[4]+",
    }
  }
  fw.close();
  br.close();
}
```

> **Tip**
>
> **Downloading the example code**
>
> You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

2. We will have to split the dataset into training and test datasets and then shuffle the datasets so that we can mix them up, which can be done using the following code snippet:

```
public void dataPrepration() throws Exception
{
  // Reading the dataset created by earlier
method convertTargetToInteger and here we are
using google guava api's.
  List<String> result =
Resources.readLines(Resources.getResource("wdbc.csv"),
Charsets.UTF_8);
  //This is to remove header before the
randomization process. Otherwise it can
appear in the middle of dataset.
  List<String> raw = result.subList(1, 570);
  Random random = new Random();
  //Shuffling the dataset.
  Collections.shuffle(raw, random);
  //Splitting dataset into training and test
examples.
  List<String> train = raw.subList(0, 470);
  List<String> test = raw.subList(470, 569);
  File trainingData = new File("<your
Location>/ wdbcTrain.csv");
  File testData = new File("<your Location>/
wdbcTest.csv");
  writeCSV(train, trainingData);
  writeCSV(test, testData);
}
//This method is writing the list to desired
file location.
public void writeCSV(List<String> list, File
file) throws IOException{
  FileWriter fw = new FileWriter(file);

 fw.write("ID_Number"+","+"Diagnosis"+","+"Radius"+","+"Texture"+","+
  for(int i=0;i< list.size();i++){
    fw.write(list.get(i)+"\n");
  }
  fw.close();
```

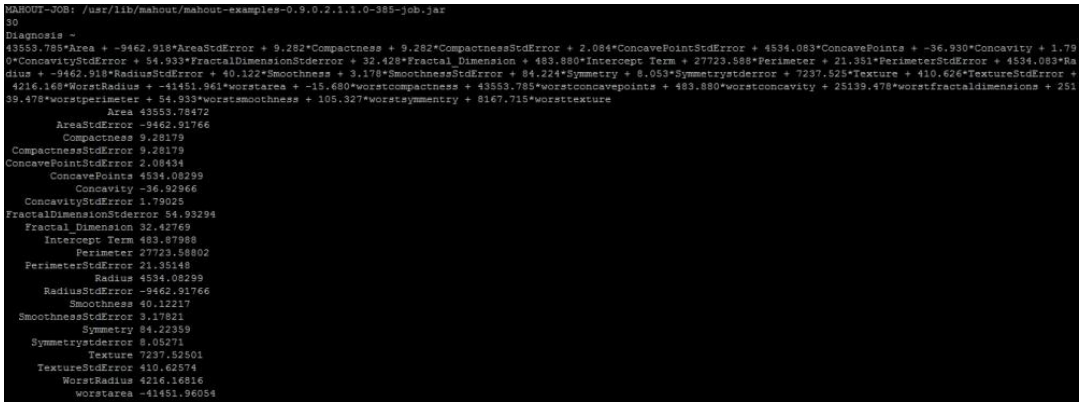
```
}
```

Training the model

We will use the training dataset and trainlogistic algorithm to prepare the model. Use the following command to create the model:

```
mahout trainlogistic --input /tmp/wdbcTrain.csv
--
output /tmp//model --target Diagnosis
--
categories 2 --predictors Radius Texture
Perimeter Area Smoothness Compactness Concavity
ConcavePoints Symmetry Fractal_Dimension
RadiusStdError TextureStdError PerimeterStdError
AreaStdError SmoothnessStdError
CompactnessStdError ConcavityStdError
ConcavePointStdError Symmetrystdderror
FractalDimensionStderror WorstRadius worsttexture
worstperimeter worstarea worstsmoothness
worstcompactness worstconcavity
worstconcavepoints worstsymmentry
worstfractaldimensions  --types numeric
--features 30 --passes 90 --rate 300
```

This command will give you the following output:



Let's understand the parameters used in this command:

- `trainlogistic`: This is the algorithm that Mahout provides to build the model using your input parameters.
- `input`: This is the location of the input file.
- `output`: This is the location of the model file.
- `target`: This is the name of the target variable that we want to predict from the dataset.
- `categories`: This refers to the number of predicted classes.

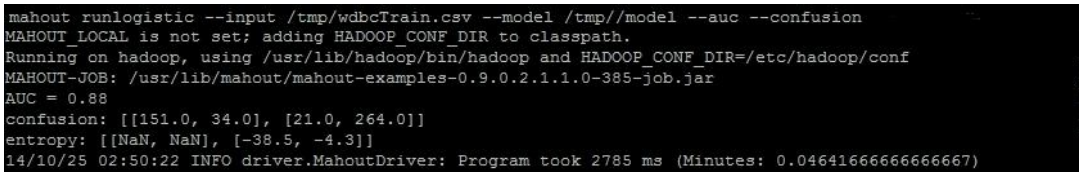
- `predictors`: This features in the dataset used to predict the target variable.
- `types`: This is a list of the types of predictor variables. (Here all are numeric but it could be word or text as well.)
- `features`: This is the size of the feature vector used to build the model.
- `passes`: This specifies the number of times the input data should be re-examined during training. Small input files may need to be examined dozens of times. Very large input files probably don't even need to be completely examined.
- `rate`: This sets the initial learning rate. This can be large if you have lots of data or use lots of passes because it decreases progressively as data is examined.

Now our model is ready to move on to the next step of evaluation. To evaluate the model further, we can use the same dataset and check the confusion and AUC matrix. The command for this will be as follows:

```
mahout runlogistic --input /tmp/wdbcTrain.csv
--model /tmp//model  --auc --confusion
```

- `runlogistic`: This is the algorithm to run the logistic regression model over an input dataset
- `model`: This is the location of the model file
- `auc`: This prints the AUC score for the model versus the input data after the data is read
- `confusion`: This prints the confusion matrix for a particular threshold

The output of the previous command is shown in the following screenshot:



Now, these matrices show that the model is not bad. Having 0.88 as the value for AUC is good, but we will check this on test data as well. The confusion matrix informs us that out of 172 malignant tumors, it has correctly classified 151 instances and that 34 benign tumors are also classified as malignant. In the case of benign tumors, out of 298, it has correctly classified 264.

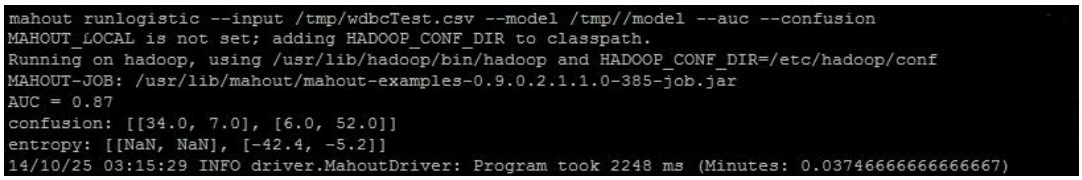
If the model does not provide good results, we have a number of options.

Change the parameters in the feature vector, increasing them if we are selecting few features. This should be done one at a time, and we should

test the result again with each generated model. We should get a model where AUC is close to 1.

Let's run the same algorithm on test data as well:

```
mahout runlogistic --input /tmp/wdbcTest.csv
--model /tmp//model  --auc –confusion
```



So this model works almost the same on test data as well. It has classified 34 out of the 40 malignant tumors correctly.

Summary

In this chapter, we discussed logistic regression and how we can use this algorithm available in Apache Mahout. We used the Wisconsin Diagnostic Breast Cancer dataset and randomly broke it into two datasets: one for training and the other for testing. We created the logistic regression model using Mahout and also ran test data over this model. Now, we will move on to the next chapter where you will learn about the Naïve Bayes classification and also the most frequently used classification technique: text classification.

Chapter 4. Learning the Naïve Bayes Classification Using Mahout

In this chapter, we will use the Naïve Bayes classification algorithm to classify a set of documents. Classifying text documents is a little tricky because of the data preparation steps involved. In this chapter, we will explore the following topics:

- Conditional probability and the Bayes rule
- Understanding the Naïve Bayes algorithm
- Understanding terms used in text classification
- Using the Naïve Bayes algorithm in Apache Mahout

Introducing conditional probability and the Bayes rule

Before learning the Naïve Bayes algorithm, you should have an understanding of conditional probability and the Bayes rule.

In very simple terms, conditional probability is the probability that something will happen, given that something else has already happened. It is expressed as *P(A/B)*, which can be read as probability of A given B, and it finds the probability of the occurrence of event A once event B has already happened.

Mathematically, it is defined as follows:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

For example, if you choose a card from a standard card deck and if you were asked about the probability for the card to be a diamond, you would quickly say 13/52 or 0.25, as there are 13 diamond cards in the deck. However, if you then look at the card and declare that it is red, then we will have narrowed the possibilities for the card to 26 possible cards, and the probability that the card is a diamond now is 13/26 = 0.5. So, if we define A as a diamond card and B as a red card, then *P(A/B)* will be the probability of the card being a diamond, given it is red.

Sometimes, for a given pair of events, conditional probability is hard to calculate, and Bayes' theorem helps us here by giving the relationship between two conditional probabilities.

Bayes' theorem is defined as follows:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

The terms in the formula are defined as follows:

- **P(A)**: This is called prior probability or prior
- **P(B/A)**: This is called conditional probability or likelihood
- **P(B)**: This is called marginal probability
- **P(A/B)**: This is called posterior probability or posterior

The following formula is derived only from the conditional probability formula. We can define *P(B/A)* as follows:

$$\frac{P(A \cap B)}{P(A)}$$

When rearranged, the formula becomes this:

$$P(A \cap B) = P(B \mid A)P(A)$$

Now, from the preceding conditional probability formula, we get the following:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \mid A)P(A)}{P(B)}$$

Let's take an example that will help us to understand how Bayes' theorem is applied.

A cancer test gives a positive result with a probability of 97 percent when the patient is indeed affected by cancer, while it gives a negative result with 99 percent probability when the patient is not affected by cancer. If a patient is drawn at random from a population where 0.2 percent of the individuals are affected by cancer and he or she is found to be positive, what is the probability that he or she is indeed affected by cancer? In probabilistic terms, what we know about this problem can be defined as follows:

P (positive| cancer) = 0.97

P (positive| no cancer) = 1-0.99 = 0.01

P (cancer) = 0.002

P (no cancer) = 1-0.002= 0.998

P (positive) = P (positive| cancer) P (cancer) + P (positive| no cancer) P (no cancer)

*= 0.97*0.002 + 0.01*0.998*

= 0.01192

Now *P (cancer| positive) = (0.97*0.002)/0.01192 = 0.1628*

So even when found positive, the probability of the patient being affected by cancer in this example is around 16 percent.

Understanding the Naïve Bayes algorithm

In Bayes' theorem, we have seen that the outcome is based only on one evidence, but in classification problems, we have multiple evidences and we have to predict the outcome. In Naïve Bayes, we uncouple multiple pieces of evidence and treat each one of them independently. It is defined as follows:

P (outcome | multiple Evidence)) = P (Evidence 1|outcome) P (Evidence 2|outcome)* P (Evidence 3|outcome) …. /P (Evidence)*

Run this formula for each possible outcome. Since we are trying to classify, each outcome will be called a class. Our task is to look at the evidence (features) to consider how likely it is for it to be of a particular class and then assign it accordingly. The class that has the highest probability gets assigned to that combination of evidences. Let's understand this with an example.

Let's say that we have data on 1,000 pieces of fruit. They happen to be bananas, apples, or some other fruit. We are aware of three characteristics of each fruit:

- **Size**: They are either long or not long
- **Taste**: They are either sweet or not sweet
- **Color**: They are either yellow or not yellow

Assume that we have a dataset like the following:

Fruit type	**Taste – sweet**	**Taste – not sweet**	**Color – yellow**	**Color – not yellow**	**Size – long**	**Size – not long**	**Total**
Banana	350	150	450	50	400	100	500
Apple	150	150	100	200	0	300	300
Other	150	50	50	150	100	100	200

Fruit type	Taste – sweet	Taste – not sweet	Color – yellow	Color – not yellow	Size – long	Size – not long	Total
Total	650	350	600	400	500	500	1000

Now let's look at the things we have:

P (Banana) = 500/1000 = 0.5

P (Apple) = 300/1000 = 0.3

P (Other) = 200/1000 = 0.2

Let's look at the probability of the features:

P (Sweet) = 650/1000 = 0.65

P (Yellow) = 600/1000 = 0.6

P (long) = 500/1000 = 0.5

P (not Sweet) = 350/1000 = 0.35

P (not yellow) = 400/1000= 0.4

P (not long) = 500/1000 = 0.5

Now we want to know what fruit we will have if it is not yellow and not long and sweet. The probability of it being an apple is as follows:

P (Apple| sweet, not long, not yellow) = P (sweet | Apple) P (not long | Apple)* P (not yellow | Apple)*P (Apple)/P (sweet)* P (not long) *P (not yellow)*

*= 0.5*1*0.67*0.3/P (Evidence)*

= 0.1005/P (Evidence)

The probability of it being a banana is this:

P (banana| sweet, not long, not yellow) = P (sweet | banana) P (not long | banana)* P (not yellow | banana)*P (banana)/P (sweet)* P (not long) *P (not yellow)*

*= 0.7*0.2*0.1*0.5/P (Evidence)*

= 0.007/P (Evidence)

The probability of it being any other fruit is as follows:

P (other fruit| sweet, not long, not yellow) = P (sweet | other fruit) P (not long | other fruit)* P (not yellow | other fruit) *P (other fruit)/P (sweet)* P (not long) *P (not yellow)*

*= 0.75*0.5*0.75*0.2/P (Evidence)*

= 0.05625/ P (Evidence)

So from the results, you can see that if the fruit is sweet, not long, and not yellow, then the highest probability is that it will be an apple. So find out the highest probability and assign the unknown item to that class.

Naïve Bayes is a very good choice for text classification. Before we move on to text classification using Naïve Bayes in Mahout, let's understand a few terms that are really useful for text classification.

Understanding the terms used in text classification

To prepare data so that it can be used by a classifier is a complex process. From raw data, we can collect explanatory and target variables and encode them as **vectors**, which is the input of the classifier.

Vectors are ordered lists of values as defined in two-dimensional space. You can take a clue from coordinate geometry as well. A point (3, 4) is a point in the x and y planes. In Mahout, it is different. Here, a vector can have (3, 4) or 10,000 dimensions.

Mahout provides support for creating vectors. There are two types of vector implementations in Mahout: sparse and dense vectors. There are a few terms that we need to understand for text classification:

- **Bag of words**: This considers each document as a collection of words. This ignores word order, grammar, and punctuation. So, if every word is a feature, then calculating the feature value of the document word is represented as a token. It is given the value 1 if it is present or 0 if not.
- **Term frequency**: This considers the word count in the document instead of 0 and 1. So the importance of a word increases with the number of times it appears in the document. Consider the following example sentence:

  Apple has launched iPhone and it will continue to launch such products. Other competitors are also planning to launch products similar to that of iPhone.

The following is the table that represents term frequency:

Term	**Count**
Apple	1
Launch	3
iPhone	2

Term	Count
Product	2
Plan	1

The following techniques are usually applied to come up with this type of table:

- **Stemming of words**: With this, the suffix is removed from the word so "launched", "launches", and "launch" are all considered as "launch".
- **Case normalization**: With this, every term is converted to lowercase.
- **Stop word removal**: There are some words that are almost present in every document. We call these words stop words. During an important feature extraction from a document, these words come into account and they will not be helpful in the overall calculation. Examples of these words are "is, are, the, that, and so on." So, while extracting, we will ignore these kind of words.
- **Inverse document frequency**: This is considered as the boost a term gets for being rare. A term should not be too common. If a term occurs in every document, it is not good for classification. The fewer documents in which a term occurs, the more significant it is likely to be for the documents it does occur in. For a term t, inverse document frequency is calculated as follows:

  IDF (t) = 1 + log (total number of documents/ number of documents containing t)

- **Term frequency and inverse term frequency**: This is one of the popular representations of the text. It is the product of term frequency and inverse document frequency, as follows:

  *TFIDF (t, d) = TF (t, d) * IDF (t)*

Each document is a feature vector and a collection of documents is a set of these feature vectors and this set works as the input for the classification. Now that we understand the basic concepts behind the vector creation of text documents, let's move on to the next section where we will classify text documents using the Naïve Bayes algorithm.

Using the Naïve Bayes algorithm in Apache Mahout

We will use a dataset of 20 newsgroups for this exercise. The 20 newsgroups dataset is a standard dataset commonly used for machine learning research. The data is obtained from transcripts of several months of postings made in 20 Usenet newsgroups from the early 1990s. This dataset consists of messages, one per file. Each file begins with header lines that specify things such as who sent the message, how long it is, what kind of software was used, and the subject. A blank line follows and then the message body follows as unformatted text.

Download the `20news-bydate.tar.gz` dataset from http://qwone.com/~jason/20Newsgroups/. The following steps are used to build the Naïve Bayes classifier using Mahout:

1. Create a `20newsdata` directory and unzip the data here:

```
mkdir /tmp/20newsdata
cd /tmp/20newsdata
tar -xzvf /tmp/20news-bydate.tar.gz
```

2. You will see two folders under `20newsdata: 20news-bydate-test` and `20news-bydate-train`. Now create another directory called `20newsdataall` and merge both the training and test data of the 20 newsgroups.
3. Come out of the directory and move to the `home` directory and execute the following:

```
mkdir /tmp/20newsdataall
cp -R /20newsdata/*/* /tmp/20newsdataall
```

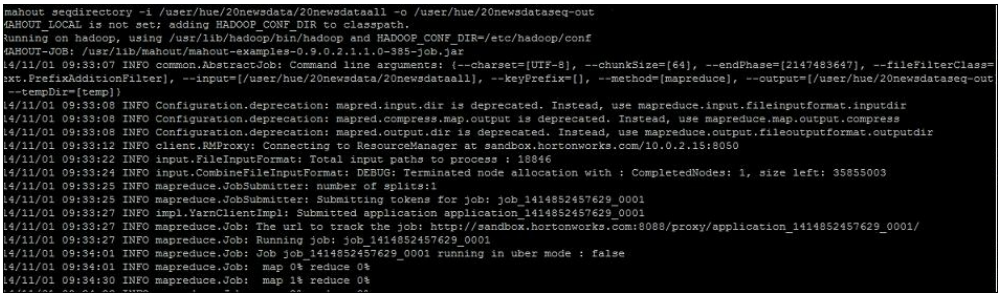
4. Create a directory in Hadoop and save this data in HDFS format:

```
hadoop fs -mkdir /user/hue/20newsdata
hadoop fs
-put /tmp/20newsdataall /user/hue/20newsdata
```

5. Convert the raw data into a sequence file. The `seqdirectory` command will generate sequence files from a directory. Sequence files are used in Hadoop. A sequence file is a flat file that consists of binary key/value pairs. We are converting the files into sequence files so that it can be processed in Hadoop, which can be done using the following command:

```
bin/mahout seqdirectory
-
i /user/hue/20newsdata/20newsdataall
-o /user/hue/20newsdataseq-out
```

The output of the preceding command can be seen in the following screenshot:



6. Convert the sequence file into a sparse vector using the following command:

```
bin/mahout seq2sparse
-i /user/hue/20newsdataseq-out/part-m-
00000 -o /user/hue/20newsdatavec -lnorm -nv
-wt tfidf
```

The terms used in the preceding command are as follows:

- `lnorm`: This is for the output vector to be log normalized
- `nv`: This refers to named vectors
- `wt`: This refers to the kind of weight to use; here, we use `tfidf`

The output of the preceding command on the console is shown in the following screenshot:

```
                CPU time spent (ms)=3450
                Physical memory (bytes) snapshot=352317440
                Virtual memory (bytes) snapshot=1801748480
                Total committed heap usage (bytes)=191889408
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=28689283
        File Output Format Counters
                Bytes Written=28689283
14/11/01 10:44:04 INFO common.HadoopUtil: Deleting /user/hue/20newsdatavec/tf-vectors-partial
14/11/01 10:44:04 INFO common.HadoopUtil: Deleting /user/hue/20newsdatavec/tf-vectors-toprune
14/11/01 10:44:05 INFO client.RMProxy: Connecting to ResourceManager at sandbox.hortonworks.com/10.0.2.15:8050
14/11/01 10:44:08 INFO input.FileInputFormat: Total input paths to process : 1
14/11/01 10:44:08 INFO mapreduce.JobSubmitter: number of splits:1
14/11/01 10:44:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1414852457629_0009
14/11/01 10:44:08 INFO impl.YarnClientImpl: Submitted application application_1414852457629_0009
14/11/01 10:44:08 INFO mapreduce.Job: The url to track the job: http://sandbox.hortonworks.com:8088/proxy/application_1414852457629_0009/
14/11/01 10:44:08 INFO mapreduce.Job: Running job: job_1414852457629_0009
14/11/01 10:44:29 INFO mapreduce.Job: Job job_1414852457629_0009 running in uber mode : false
14/11/01 10:44:29 INFO mapreduce.Job:  map 0% reduce 0%
14/11/01 10:44:51 INFO mapreduce.Job:  map 100% reduce 0%
14/11/01 10:45:23 INFO mapreduce.Job:  map 100% reduce 79%
14/11/01 10:45:25 INFO mapreduce.Job:  map 100% reduce 100%
14/11/01 10:45:27 INFO mapreduce.Job: Job job_1414852457629_0009 completed successfully
14/11/01 10:45:27 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=30327803
                FILE: Number of bytes written=57077985
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=28689430
                HDFS: Number of bytes written=28689283
                HDFS: Number of read operations=7
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
```

7. Split the set of vectors to train and test the model:

```
bin/mahout split
-i /user/hue/20newsdatavec/tfidf-
vectors
--
trainingOutput /user/hue/20newsdatatrain
--
testOutput /user/hue/20newsdatatest
--
randomSelectionPct 40 --overwrite
--sequenceFiles -xm sequential
```

The terms used in the preceding command are as follows:

- `randomSelectionPct`: This divides the percentage of data into testing and training datasets. Here, 60 percent is for testing and 40 percent for training.
- `xm`: This refers to the execution method to use: sequential or mapreduce. The default is `mapreduce`.

```
mahout split -i /user/hue/20newsdatavec/tfidf-vectors --trainingOutput /user/hue/20newsdatatrain --testOutput /user/hue/20newsdatatest --randomSelectionPct 40 -overwrite
--sequenceFiles -xm sequential
MAHOUT_LOCAL is not set; adding HADOOP_CONF_DIR to classpath.
Running on hadoop, using /usr/lib/hadoop/bin/hadoop and HADOOP_CONF_DIR=/etc/hadoop/conf
MAHOUT-JOB: /usr/lib/mahout/mahout-examples-0.9.0.2.1.1.0-385-job.jar
14/11/01 11:12:47 WARN driver.MahoutDriver: No split.props found on classpath, will use command-line arguments only
14/11/01 11:12:48 INFO common.AbstractJob: Command line arguments: {--endPhase=[2147483647], --input=[/user/hue/20newsdatavec/tfidf-vectors], --method=[sequential], --o
verwrite=null, --randomSelectionPct=[40], --sequenceFiles=null, --startPhase=[0], --tempDir=[temp], --testOutput=[/user/hue/20newsdatatest], --trainingOutput=[/user/hue
/20newsdatatrain]}
14/11/01 11:12:56 INFO utils.SplitInput: part-r-00000 has 162419 lines
14/11/01 11:12:56 INFO utils.SplitInput: part-r-00000 test split size is 64968 based on random selection percentage 40
14/11/01 11:12:56 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
14/11/01 11:12:56 INFO compress.CodecPool: Got brand-new compressor [.deflate]
14/11/01 11:12:56 INFO compress.CodecPool: Got brand-new compressor [.deflate]
14/11/01 11:13:02 INFO utils.SplitInput: file: part-r-00000, input: 162419 train: 11311, test: 7535 starting at 0
14/11/01 11:13:02 INFO driver.MahoutDriver: Program took 14552 ms (Minutes: 0.24253333333333332)
```

8. Now train the model:

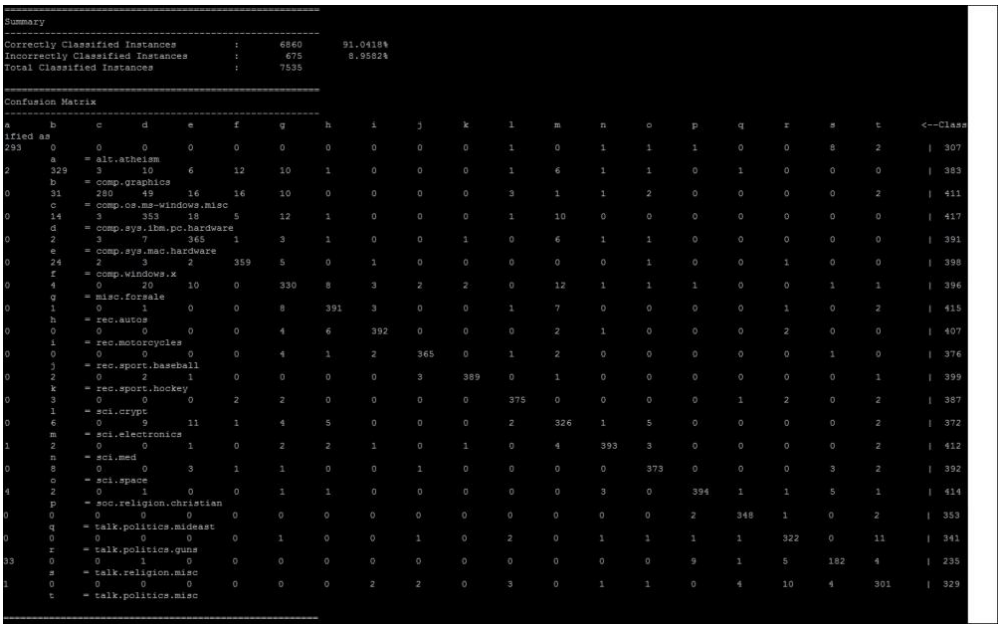
```
bin/mahout trainnb
-
i /user/hue/20newsdatatrain -el
```

```
-
o /user/hue/model -li /user/hue/labelindex
-ow -c
```

9. Test the model using the following command:

```
bin/mahout testnb -i /user/hue/20newsdatatest
-
m /user/hue/model/ -l  /user/hue/labelindex
-ow -o /user/hue/results
```

The output of the preceding command on the console is shown in the following screenshot:



We get the result of our Naïve Bayes classifier for the 20 newsgroups.

Summary

In this chapter, we discussed the Naïve Bayes algorithm. This algorithm is a simplistic yet highly regarded statistical model that is widely used in both industry and academia, and it produces good results on many occasions. We initially discussed conditional probability and the Bayes rule. We then saw an example of the Naïve Bayes algorithm. You learned about the approaches to convert text into a vector format, which is an input for classifiers. Finally, we used the 20 newsgroups dataset to build a classifier using the Naïve Bayes algorithm in Mahout. In the next chapter, we will continue our journey of exploring classification algorithms in Mahout with the Hidden Markov model implementation.

Chapter 5. Learning the Hidden Markov Model Using Mahout

In this chapter, we will cover one of the most interesting topics of classification techniques: the **Hidden Markov Model** (**HMM**). To understand the HMM, we will cover the following topics in this chapter:

- Deterministic and nondeterministic patterns
- The Markov process
- Introducing the HMM
- Using Mahout for the HMM

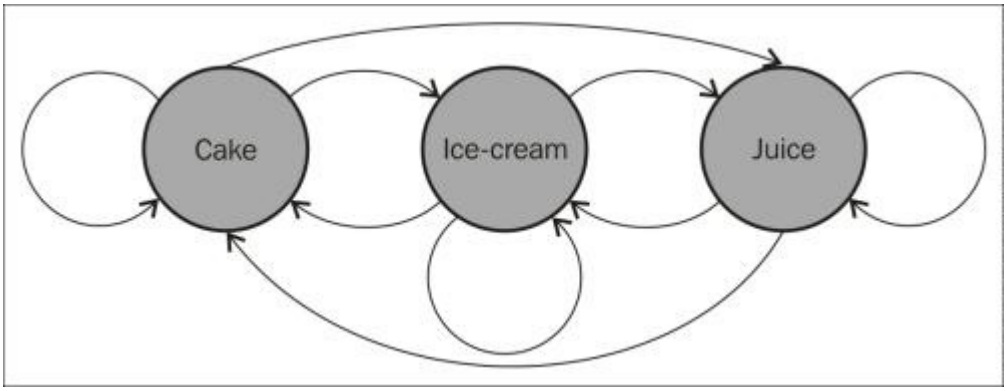
Deterministic and nondeterministic patterns

In a deterministic system, each state is solely dependent on the state it was previously in. For example, let's take the case of a set of traffic lights. The sequence of lights is red → green → amber → red. So, here we know what state will follow after the current state. Once the transitions are known, deterministic systems are easy to understand.

For nondeterministic patterns, consider an example of a person named Bob who has his snacks at 4:00 P.M. every day. Let's say he has any one of the three items from the menu: ice cream, juice, or cake. We cannot say for sure what item he will have the next day, even if we know what he had today. This is an example of a nondeterministic pattern.

The Markov process

In the Markov process, the next state is dependent on the previous states. If we assume that we have an *n* state system, then the next state is dependent on the previous *n* states. This process is called an *n* model order. In the Markov process, we make the choice for the next state probabilistically. So, considering our previous example, if Bob had juice today, he can have juice, ice cream, or cake the next day. In the same way, we can reach any state in the system from the previous state. The Markov process is shown in the following diagram:



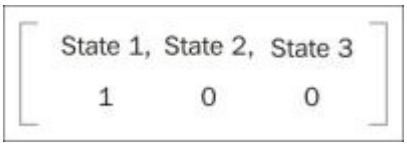
If we have *n* states in a process, then we can reach any state with n2 transitions. We have a probability of moving to any state, and hence, we will have n2 probabilities of doing this. For a Markov process, we will have the following three items:

- **States**: This refers to the states in the system. In our example, let's say there are three states: state 1, state 2, and state 3.
- **Transition matrix**: This will have the probabilities of moving from one state to any other state. An example of the transition matrix is shown in the following screenshot:

		Today		
Yesterday		State 1,	State 2,	State 3
	State 1	0.1	0.8	0.2
	State 2	0.3	0.1	0.1
	State 3	0.6	0.1	0.7

This matrix shows that if the system was in state 1 yesterday, then the probability of it to remain in the same state today will be 0.1.

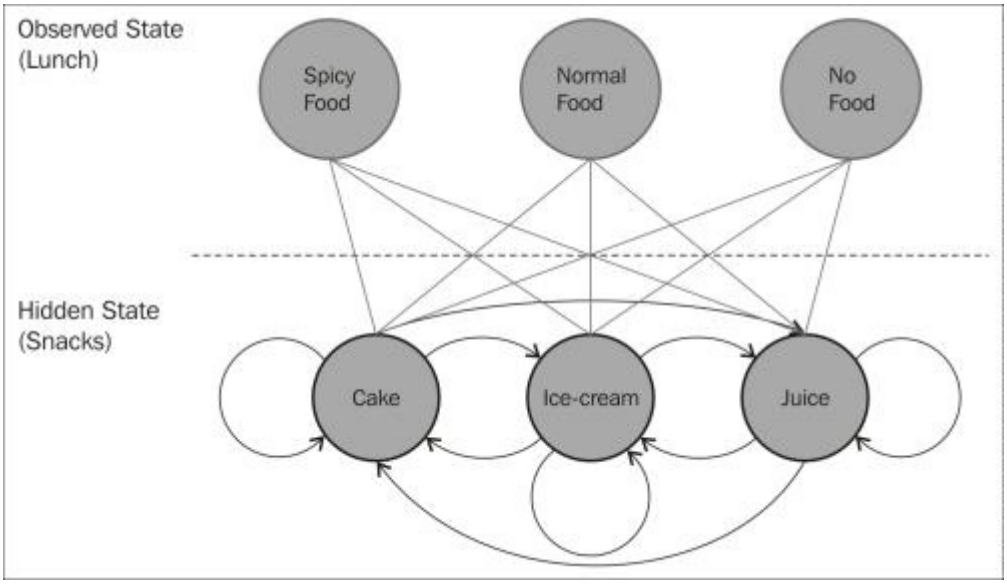
- **Initial state vector**: This is the vector of the initial state of the system. (Any one of the states will have a probability of 1 and the rest will have a probability of 0 in this vector.)



Introducing the Hidden Markov Model

The **Hidden Markov Model** (**HMM**) is a classification technique to predict the states of a system by observing the outcomes without having access to the actual states themselves. It is a Markov model in which the states are hidden.

Let's continue with Bob's snack example we saw earlier. Now assume we have one more set of events in place that is directly observable. We know what Bob has eaten for lunch and his snacks intake is related to his lunch. So, we have an observation state, which is Bob's lunch, and hidden states, which are his snacks intake. We want to build an algorithm that can forecast what would be Bob's choice of snack based on his lunch.



In addition to the transition probability matrix in the Hidden Markov Model, we have one more matrix that is called an **emission matrix**. This matrix contains the probability of the observable state, provided it is assigned a hidden state. The emission matrix is as follows:

P (observable state | one state)

So, a Hidden Markov Model has the following properties:

- **State vector**: This contains the probability of the hidden model to be in a particular state at the start
- **Transition matrix**: This has the probabilities of a hidden state, given the previous hidden state

- **Emission matrix**: Given that the hidden model is in a particular hidden state, this has the probabilities of observing a particular observable state
- **Hidden states**: This refers to the states of the system that can be defined by the Hidden Markov Model
- **Observable state**: The states that are visible in the process

Using the Hidden Markov Model, three types of problems can be solved. The first two are related to the pattern recognition problem and the third type of problem generates a Hidden Markov Model, given a sequence of observations. Let's look at these three types of problems:

- **Evaluation**: This is finding out the probability of an observed sequence, given an HMM. From the number of different HMMs that describe different systems and a sequence of observations, our goal will be to find out which HMM will most probably generate the required sequence. We use the forward algorithm to calculate the probability of an observation sequence when a particular HMM is given and find out the most probable HMM.
- **Decoding**: This is finding the most probable sequence of hidden states from some observations. We use the Viterbi algorithm to determine the most probable sequence of hidden states when you have a sequence of observations and an HMM.
- **Learning**: Learning is generating the HMM from a sequence of observations. So, if we have such a sequence, we may wonder which is the most likely model to generate this sequence. The forward-backward algorithms are useful in solving this problem.

The Hidden Markov Model is used in different applications such as speech recognition, handwritten letter recognition, genome analysis, parts of speech tagging, customer behavior modeling, and so on.

Using Mahout for the Hidden Markov Model

Apache Mahout has the implementation of the Hidden Markov Model. It is available in the `org.apache.mahout.classifier.sequencelearning.hmm` package.

The overall implementation is provided by eight different classes:

- `HMMModel`: This is the main class that defines the Hidden Markov Model.
- `HmmTrainer`: This class has algorithms that are used to train the Hidden Markov Model. The main algorithms are supervised learning, unsupervised learning, and unsupervised Baum-Welch.
- `HmmEvaluator`: This class provides different methods to evaluate an HMM model. The following use cases are covered in this class:
    - Generating a sequence of output states from a model (prediction)
    - Computing the likelihood that a given model will generate the given sequence of output states (model likelihood)
    - Computing the most likely hidden sequence for a given model and a given observed sequence (decoding)
- `HmmAlgorithms`: This class contains implementations of the three major HMM algorithms: forward, backward, and Viterbi.
- `HmmUtils`: This is a utility class and provides methods to handle HMM model objects.
- `RandomSequenceGenerator`: This is a command-line tool to generate a sequence by the given HMM.
- `BaumWelchTrainer`: This is the class to train HMM from the console.
- `ViterbiEvaluator`: This is also a command-line tool for Viterbi evaluation.

Now, let's work with Bob's example.

The following is the given matrix and the initial probability vector:

Ice cream	**Cake**	**Juice**
0.36	0.51	0.13

The following will be the state transition matrix:

	Ice cream	**Cake**	**Juice**
Ice cream	0.365	0.500	0.135
Cake	0.250	0.125	0.625
Juice	0.365	0.265	0.370

The following will be the emission matrix:

	Spicy food	**Normal food**	**No food**
Ice cream	0.1	0.2	0.7
Cake	0.5	0.25	0.25
Juice	0.80	0.10	0.10

Now we will execute a command-line-based example of this problem. We have three hidden states of what Bob's eaten for snacks: ice-cream, cake, or juice. Then, we have three observable states of what he is having at

lunch: spicy food, normal food, or no food at all. Now, the following are the steps to execute from the command line:

1. Create a directory with the name `hmm: mkdir /tmp/hmm`. Go to this directory and create the sample input file of the observed states. This will include a sequence of Bob's lunch habit: spicy food (state 0), normal food (state 1), and no food (state 2). Execute the following command:

```
echo "0 1 2 2 2 1 1 0 0 2 1 2 1 1 1 1 2 2 2 0
0 0 0 0 0 2 2 2 0 0 0 0 0 0 1 1 1 1 2 2 2 2 2
0 2 1 2 0 2 1 2 1 1 0 0 0 1 0 1 0 2 1 2 1 2 1
2 1 1 0 0 2 2 0 2 1 1 0" > hmm-input
```

2. Run the BaumWelch algorithm to train the model using the following command:

```
mahout baumwelch -i /tmp/hmm/hmm-input
-o /tmp/hmm/hmm-
model -nh 3 -no 3 -e .0001 -m 1000
```

   The parameters used in the preceding command are as follows:

   - `i`: This is the input file location
   - `o`: This is the output location for the model
   - `nh`: This is the number of hidden states. In our example, it is three (ice cream, juice, or cake)
   - `no`: This is the number of observable states. In our example, it is three (spicy, normal, or no food)
   - `e`: This is the epsilon number. This is the convergence threshold value
   - `m`: This is the maximum iteration number

   The following screenshot shows the output on executing the previous command:

```
mahout baumwelch -i /tmp/hmm/hmm-input -o /tmp/hmm/hmm-model -nh 3 -no 3 -e .0001 -m 1000
MAHOUT_LOCAL is not set; adding HADOOP_CONF_DIR to classpath.
Running on hadoop, using /usr/lib/hadoop/bin/hadoop and HADOOP_CONF_DIR=/etc/hadoop/conf
MAHOUT-JOB: /usr/lib/mahout/mahout-examples-0.9.0.2.1.1.0-385-job.jar
14/11/06 11:02:51 WARN driver.MahoutDriver: No baumwelch.props found on classpath, will use command-line arguments only
Initial probabilities:
0 1 2
1.0 1.309026126805178E-89 1.7587674488030393E-78
Transition matrix:
  0 1 2
0 4.993504845479359E-13 0.5227259091584807 0.4772740908410199
1 0.9998785172353453 3.084831951926291E-9 1.2147967982263436E-4
2 2.041684928559705lE-4 0.10221844628449037 0.8975773852226537
Emission matrix:
  0 1 2
0 0.999999999999999 1.1282788180575074E-24 9.881045303840455E-16
1 0.7514502891771425 0.2485497108228551 2.32229480690411E-15
2 0.08635564236932171 0.4108528050151504 0.502791552615528
```

3. Now we have an HMM model that can be used to build a predicted sequence. We will run the model to predict the next 15 states of the observable sequence using the following command:

```
mahout hmmpredict -m /tmp/hmm/hmm-model
-o /tmp/hmm/hmm-predictions -l 10
```

The parameters used in the preceding command are as follows:

`m`: This is the path for the HMM model

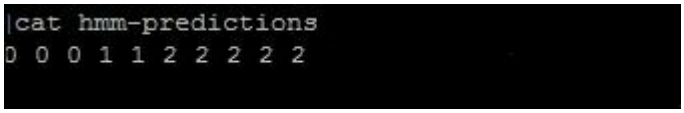
`o`: This is the output directory path

`l`: This is the length of the generated sequence

4. To view the prediction for the next 10 observable states, use the following command:

```
mahout hmmpredict -m /tmp/hmm/hmm-model
-o /tmp/hmm/hmm-predictions -l 10
```

The output of the previous command is shown in the following screenshot:



From the output, we can say that the next observable states for Bob's lunch will be spicy, spicy, spicy, normal, normal, no food, no food, no food, no food, and no food.

5. Now, we will use one more algorithm to predict the hidden state. We will use the Viterbi algorithm to predict the hidden states for a given observational state's sequence. We will first create the sequence of the observational state using the following command:

```
echo "0 1 2 0 2 1 1 0 0 1 1 2"
> /tmp/hmm/hmm-viterbi-input
```

6. We will use the Viterbi command-line option to generate the output with the likelihood of generating this sequence:

```
mahout viterbi
--input /tmp/hmm/hmm-viterbi-
input --output tmp/hmm/hmm-viterbi-output
--model /tmp/hmm/hmm-model --likelihood
```

The parameters used in the preceding command are as follows:

- `input`: This is the input location of the file
- `output`: This is the output location of the Viterbi algorithm's output
- `model`: This is the HMM model location that we created earlier
- `likelihood`: This is the computed likelihood of the observed sequence

The following screenshot shows the output on executing the previous command:

```
mahout viterbi --input /tmp/hmm/hmm-viterbi-input --output /tmp/hmm/hmm-viterbi-output --model /tmp/hmm/hmm-model --likelihood
MAHOUT_LOCAL is not set; adding HADOOP_CONF_DIR to classpath.
Running on hadoop, using /usr/lib/hadoop/bin/hadoop and HADOOP_CONF_DIR=/etc/hadoop/conf
MAHOUT-JOB: /usr/lib/mahout/mahout-examples-0.9.0.2.1.1.0-385-job.jar
14/11/06 11:42:22 WARN driver.MahoutDriver: No viterbi.props found on classpath, will use command-line arguments only
Likelihood: 1.2864746327281034E-6
14/11/06 11:42:24 INFO driver.MahoutDriver: Program took 1540 ms (Minutes: 0.025666666666666667)
```

7. Predictions from the Viterbi are saved in the output file and can be seen using the `cat` command:

```
cat /tmp/hmm/hmm-viterbi-output
```

The following output shows the predictions for the hidden state:

```
cat /tmp/hmm/hmm-viterbi-output
0 2 2 2 2 2 2 1 0 2 2 2
```

Summary

In this chapter, we discussed another classification technique: the Hidden Markov Model. You learned about deterministic and nondeterministic patterns. We also touched upon the Markov process and Hidden Markov process in general. We checked the classes implemented inside Mahout to support the Hidden Markov Model. We took up an example to create the HMM model and further used this model to predict the observational state's sequence. We used the Viterbi algorithm implemented in Mahout to predict the hidden states in the system.

Now, in the next chapter, we will cover one more interesting algorithm used in classification area: Random forest.

Chapter 6. Learning Random Forest Using Mahout

Random forest is one of the most popular techniques in classification. It starts with a machine learning technique called **decision tree**. In this chapter, we will explore the following topics:

- Decision tree
- Random forest
- Using Mahout for Random forest

Decision tree

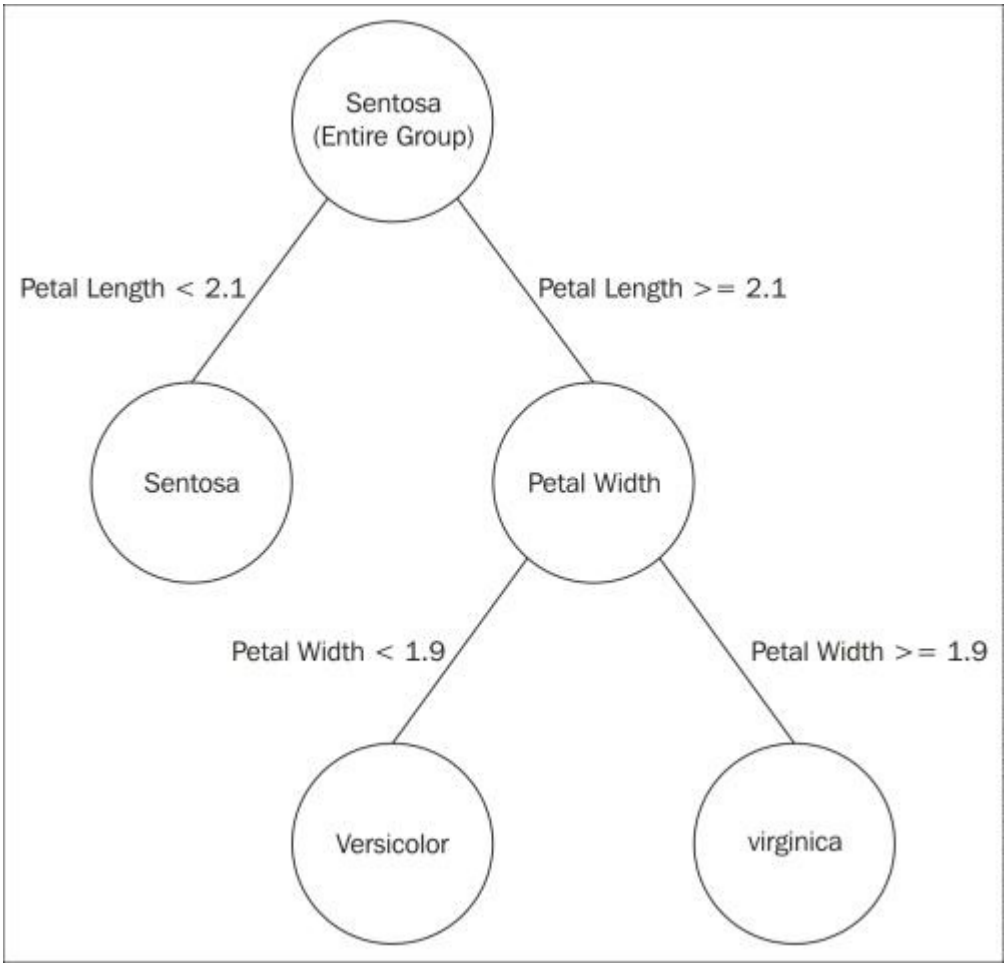
A decision tree is used for classification and regression problems. In simple terms, it is a predictive model that uses binary rules to calculate the target variable. In a decision tree, we use an iterative process of splitting the data into partitions, then we split it further on branches. As in other classification model creation processes, we start with the training dataset in which target variables or class labels are defined. The algorithm tries to break all the records in training datasets into two parts based on one of the explanatory variables. The partitioning is then applied to each new partition, and this process is continued until no more partitioning can be done. The core of the algorithm is to find out the rule that determines the initial split. There are algorithms to create decision trees, such as **Iterative Dichotomiser 3** (**ID3**), **Classification and Regression Tree** (**CART**), **Chi-squared Automatic Interaction Detector** (**CHAID**), and so on. A good explanation for ID3 can be found at http://www.cse.unsw.edu.au/~billw/cs9414/notes/ml/06prop/id3/id3.html.

Forming the explanatory variables to choose the best splitter in a node, the algorithm considers each variable in turn. Every possible split is considered and tried, and the best split is the one that produces the largest decrease in diversity of the classification label within each partition. This is repeated for all variables, and the winner is chosen as the best splitter for that node. The process is continued in the next node until we reach a node where we can make the decision.

We create a decision tree from a training dataset so it can suffer from the overfitting problem. This behavior creates a problem with real datasets. To improve this situation, a process called **pruning** is used. In this process, we remove the branches and leaves of the tree to improve the performance. Algorithms used to build the tree work best at the starting or root node since all the information is available there. Later on, with each split, data is less and towards the end of the tree, a particular node can show patterns that are related to the set of data which is used to split. These patterns create

problems when we use them to predict the real dataset. Pruning methods let the tree grow and remove the smaller branches that fail to generalize. Now take an example to understand the decision tree.

Consider we have a iris flower dataset. This dataset is hugely popular in the machine learning field. It was introduced by Sir Ronald Fisher. It contains 50 samples from each of three species of iris flower (Iris setosa, Iris virginica, and Iris versicolor). The four explanatory variables are the length and width of the sepals and petals in centimeters, and the target variable is the class to which the flower belongs.



As you can see in the preceding diagram, all the groups were earlier considered as Sentosa species and then the explanatory variable and petal length were further used to divide the groups. At each step, the calculation for misclassified items was also done, which shows how many items were wrongly classified. Moreover, the petal width variable was taken into account. Usually, items at leaf nodes are correctly classified.

Random forest

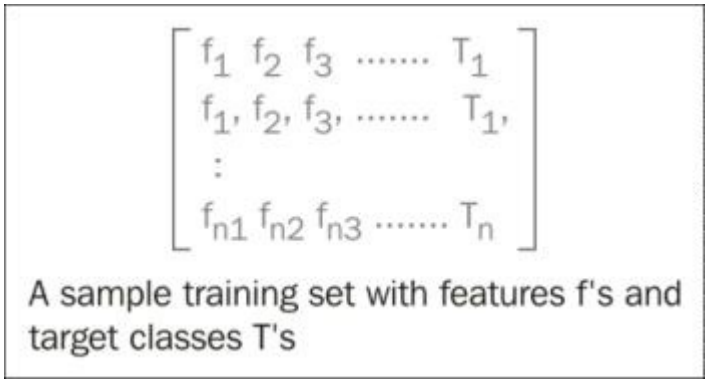
The Random forest algorithm was developed by Leo Breiman and Adele Cutler. Random forests grow many classification trees. They are an ensemble learning method for classification and regression that constructs a number of decision trees at training time and also outputs the class that is the mode of the classes outputted by individual trees.

Single decision trees show the bias–variance tradeoff. So they usually have high variance or high bias. The following are the parameters in the algorithm:

- **Bias**: This is an error caused by an erroneous assumption in the learning algorithm
- **Variance**: This is an error that ranges from sensitivity to small fluctuations in the training set

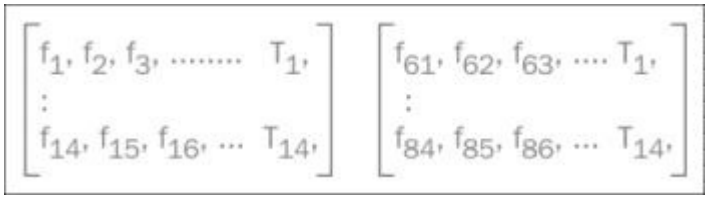
Random forests attempt to mitigate this problem by averaging to find a natural balance between two extremes. A Random forest works on the idea of bagging, which is to average noisy and unbiased models to create a model with low variance. A Random forest algorithm works as a large collection of decorrelated decision trees. To understand the idea of a Random forest algorithm, let's work with an example.

Consider we have a training dataset that has lots of features (explanatory variables) and target variables or classes:

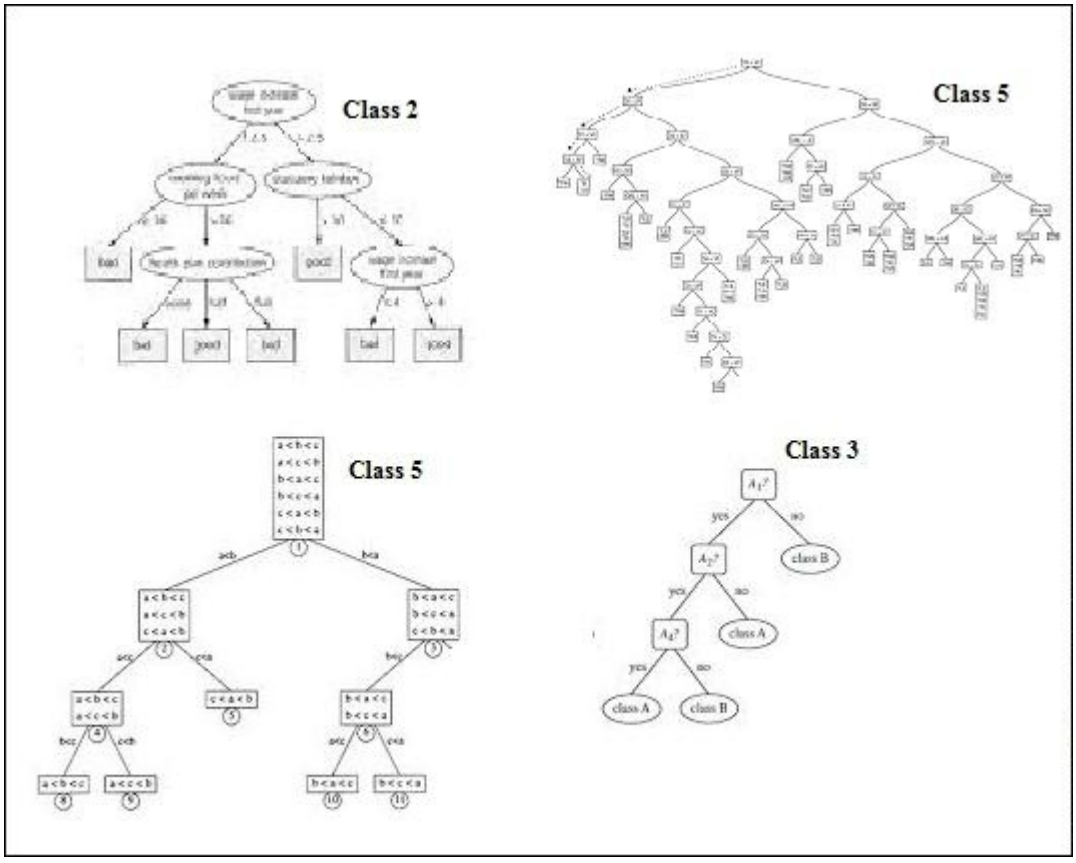


A sample training set with features f's and target classes T's

We create a sample set from the given dataset:



A different set of random features were taken into account to create the random sub-dataset. Now, from these sub-datasets, different decision trees will be created. So actually we have created a forest of the different decision trees. Using these different trees, we will create a ranking system for all the classifiers. To predict the class of a new unknown item, we will use all the decision trees and separately find out which class these trees are predicting. See the following diagram for a better understanding of this concept:



Different decision trees to predict the class of an unknown item

In this particular case, we have four different decision trees. We predict the class of an unknown dataset with each of the trees. As per the preceding figure, the first decision tree provides class 2 as the predicted class, the second decision tree predicts class 5, the third decision tree predicts class 5, and the fourth decision tree predicts class 3. Now, a Random forest will vote for each class. So we have one vote each for class 2 and class 3 and two votes for class 5. Therefore, it has decided that for the new unknown dataset, the predicted class is class 5. So the class that gets a higher vote is decided for the new dataset. A Random forest has a lot of benefits in classification and a few of them are mentioned in the following list:

- Combination of learning models increases the accuracy of the classification

- Runs effectively on large datasets as well
- The generated forest can be saved and used for other datasets as well
- Can handle a large amount of explanatory variables

Now that we have understood the Random forest theoretically, let's move on to Mahout and use the Random forest algorithm, which is available in Apache Mahout.

Using Mahout for Random forest

Mahout has implementation for the Random forest algorithm. It is very easy to understand and use. So let's get started.

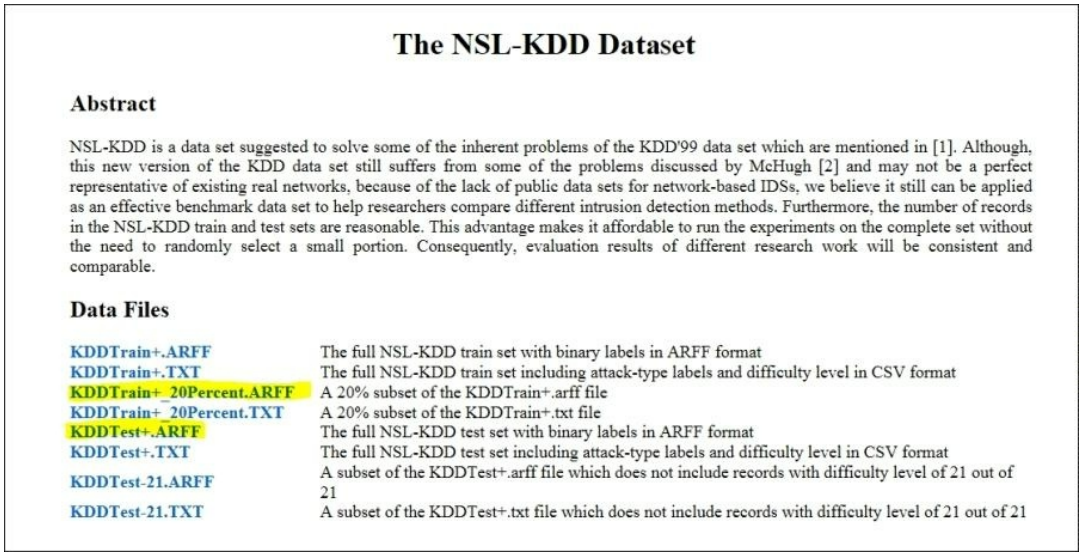
Dataset

We will use the NSL-KDD dataset. Since 1999, KDD'99 has been the most widely used dataset for the evaluation of anomaly detection methods. This dataset is prepared by S. J. Stolfo and is built based on the data captured in the DARPA'98 IDS evaluation program (*R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," discex, vol. 02, p. 1012, 2000*).

DARPA'98 is about 4 GB of compressed raw (binary) tcp dump data of 7 weeks of network traffic, which can be processed into about 5 million connection records, each with about 100 bytes. The two weeks of test data have around 2 million connection records. The KDD training dataset consists of approximately 4,900,000 single connection vectors, each of which contains 41 features and is labeled as either normal or an attack, with exactly one specific attack type.

NSL-KDD is a dataset suggested to solve some of the inherent problems of the KDD'99 dataset. You can download this dataset from http://nsl.cs.unb.ca/NSL-KDD/.

We will download the **KDDTrain+_20Percent.ARFF** and **KDDTest+.ARFF** datasets.



The NSL-KDD Dataset

Abstract

NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD'99 data set which are mentioned in [1]. Although, this new version of the KDD data set still suffers from some of the problems discussed by McHugh [2] and may not be a perfect representative of existing real networks, because of the lack of public data sets for network-based IDSs, we believe it still can be applied as an effective benchmark data set to help researchers compare different intrusion detection methods. Furthermore, the number of records in the NSL-KDD train and test sets are reasonable. This advantage makes it affordable to run the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research work will be consistent and comparable.

Data Files

KDDTrain+.ARFF	The full NSL-KDD train set with binary labels in ARFF format
KDDTrain+.TXT	The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
KDDTrain+_20Percent.ARFF	A 20% subset of the KDDTrain+.arff file
KDDTrain+_20Percent.TXT	A 20% subset of the KDDTrain+.txt file
KDDTest+.ARFF	The full NSL-KDD test set with binary labels in ARFF format
KDDTest+.TXT	The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
KDDTest-21.ARFF	A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
KDDTest-21.TXT	A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

Note

In **KDDTrain+_20Percent.ARFF** and **KDDTest+.ARFF**, remove the first 44 lines (that is, all lines starting with @attribute). If this is not done, we will not be able to generate a descriptor file.

```
@relation 'KDDTrain-20Percent'
@attribute 'duration' real
@attribute 'protocol_type' {'tcp','udp', 'icmp'}
@attribute 'service' {'aol', 'auth', 'bgp', 'courier', 'csnet_ns', 'ctf', 'daytime', 'discard', 'domain', 'domain_u', 'echo', 'eco
'finger', 'ftp', 'ftp_data', 'gopher', 'harvest', 'hostnames', 'http', 'http_2784', 'http_443', 'http_8001', 'imap4', 'IRC', 'iso_
'ldap', 'link', 'login', 'mtp', 'name', 'netbios_dgm', 'netbios_ns', 'netbios_ssn', 'netstat', 'nnsp', 'nntp', 'ntp_u', 'other', '
'printer', 'private', 'red_i', 'remote_job', 'rje', 'shell', 'smtp', 'sql_net', 'ssh', 'sunrpc', 'supdup', 'systat', 'telnet', 'tf
'urh_i', 'urp_i', 'uucp', 'uucp_path', 'vmnet', 'whois', 'X11', 'Z39_50'}
@attribute 'flag' { 'OTH', 'REJ', 'RSTO', 'RSTOS0', 'RSTR', 'S0', 'S1', 'S2', 'S3', 'SF', 'SH' }
@attribute 'src_bytes' real
@attribute 'dst_bytes' real
@attribute 'land' {'0', '1'}
@attribute 'wrong_fragment' real
@attribute 'urgent' real
@attribute 'hot' real
@attribute 'num_failed_logins' real
@attribute 'logged_in' {'0', '1'}
@attribute 'num_compromised' real
@attribute 'root_shell' real
@attribute 'su_attempted' real
@attribute 'num_root' real
@attribute 'num_file_creations' real
@attribute 'num_shells' real
@attribute 'num_access_files' real
@attribute 'num_outbound_cmds' real
@attribute 'is_host_login' {'0', '1'}
@attribute 'is_guest_login' {'0', '1'}
@attribute 'count' real
@attribute 'srv_count' real
@attribute 'serror_rate' real
@attribute 'srv_serror_rate' real
@attribute 'rerror_rate' real
@attribute 'srv_rerror_rate' real
```

Steps to use the Random forest algorithm in Mahout

The steps to implement the Random forest algorithm in Apache Mahout are as follows:

1. Transfer the test and training datasets to `hdfs` using the following commands:

```
hadoop fs -mkdir /user/hue/KDDTrain
hadoop fs -mkdir /user/hue/KDDTest
hadoop fs –put /tmp/KDDTrain+_20Percent.arff
 /user/hue/KDDTrain
hadoop fs –put /tmp/KDDTest+.arff
 /user/hue/KDDTest
```

2. Generate the descriptor file. Before you build a Random forest model based on the training data in **KDDTrain+.arff**, a descriptor file is required. This is because all information in the training dataset needs to be labeled. From the labeled dataset, the algorithm can understand which one is numerical and categorical. Use the following command to generate descriptor file:

```
hadoop jar
$MAHOUT_HOME/core/target/mahout-core-
xyz.job.jar
org.apache.mahout.classifier.df.tools.Describe
-p /user/hue/KDDTrain/KDDTrain
+_20Percent.arff
-f /user/hue/KDDTrain/KDDTrain+.info
-d N 3 C 2 N C 4 N C 8 N 2 C 19 N L
```

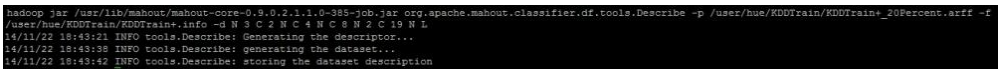
Jar: Mahout core jar (`xyz` stands for version). If you have directly installed Mahout, it can be found under the `/usr/lib/mahout` folder. The main class `Describe` is used here and it takes three parameters:

The `p` path for the data to be described.

The `f` location for the generated descriptor file.

`d` is the information for the attribute on the data. N 3 C 2 N C 4 N C 8 N 2 C 19 N L defines that the dataset is starting with a numeric (N), followed by three categorical attributes, and so on. In the last, L defines the label.

The output of the previous command is shown in the following screenshot:



3. Build the Random forest using the following command:

```
hadoop jar
$MAHOUT_HOME/examples/target/mahout-examples-
xyz-
job.jar
org.apache.mahout.classifier.df.mapreduce.BuildForest

-
Dmapred.max.split.size=1874231
-d /user/hue/KDDTrain/KDDTrain
+_20Percent.arff
-ds /user/hue/KDDTrain/KDDTrain+.info
-sl 5 -p -t 100 –o /user/hue/ nsl-forest
```

Jar: Mahout example jar (`xyz` stands for version). If you have directly installed Mahout, it can be found under the `/usr/lib/mahout` folder. The main class `build forest` is used to build the forest with other arguments, which are shown in the following list:

`Dmapred.max.split.size` indicates to Hadoop the maximum size of each partition.

`d` stands for the data path.

`ds` stands for the location of the descriptor file.

`sl` is a variable to select randomly at each tree node. Here, each tree is built using five randomly selected attributes per node.

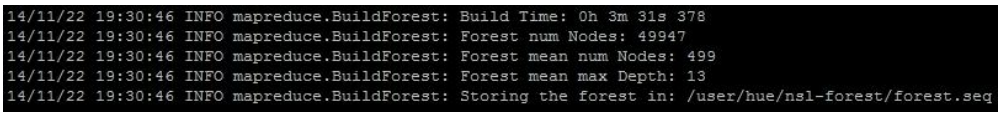
`p` uses partial data implementation.

`t` stands for the number of trees to grow. Here, the commands build 100 trees using partial implementation.

`o` stands for the output path that will contain the decision forest.



In the end, the process will show the following result:



4. Use this model to classify the new dataset:

```
hadoop jar
$MAHOUT_HOME/examples/target/mahout-examples-
xyz-
job.jar
org.apache.mahout.classifier.df.mapreduce.TestForest
-i /user/hue/KDDTest/KDDTest+.arff
-ds /user/hue/KDDTrain/KDDTrain
+.info -m /user/hue/nsl-forest -a -mr
 -o /user/hue/predictions
```

Jar: Mahout example jar (`xyz` stands for version). If you have directly installed Mahout, it can be found under the `/usr/lib/mahout` folder. The class to test the forest has the following parameters:

`I` indicates the path for the test data

`ds` stands for the location of the descriptor file

`m` stands for the location of the generated forest from the previous command

`a` informs to run the analyzer to compute the confusion matrix

`mr` informs hadoop to distribute the classification

`o` stands for the location to store the predictions in

```
hadoop jar /usr/lib/mahout/mahout-examples-0.9.0.2.1.1.0-385-job.jar org.apache.mahout.classifier.df.mapreduce.TestForest -i /user/hue/KDDTest/KDDTest+.arff -ds /user/
hue/KDDTrain/KDDTrain+.info -m /user/hue/nsl-forest -a -mr -o /user/hue/predictions
14/11/22 19:57:07 INFO mapreduce.Classifier: Adding the dataset to the DistributedCache
14/11/22 19:57:07 INFO mapreduce.Classifier: Adding the decision forest to the DistributedCache
14/11/22 19:57:07 INFO mapreduce.Classifier: Configuring the job...
14/11/22 19:57:07 INFO mapreduce.Classifier: Running the job...
14/11/22 19:57:07 INFO client.RMProxy: Connecting to ResourceManager at sandbox.hortonworks.com/10.0.2.15:8050
14/11/22 19:57:13 INFO input.FileInputFormat: Total input paths to process : 1
14/11/22 19:57:13 INFO mapreduce.JobSubmitter: number of splits:1
14/11/22 19:57:14 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1416704446466_0002
14/11/22 19:57:15 INFO impl.YarnClientImpl: Submitted application application_1416704446466_0002
14/11/22 19:57:15 INFO mapreduce.Job: The url to track the job: http://sandbox.hortonworks.com:8088/proxy/application_1416704446466_0002/
14/11/22 19:57:15 INFO mapreduce.Job: Running job: job_1416704446466_0002
14/11/22 19:58:00 INFO mapreduce.Job: Job job_1416704446466_0002 running in uber mode : false
14/11/22 19:58:00 INFO mapreduce.Job:  map 0% reduce 0%
14/11/22 19:58:29 INFO mapreduce.Job:  map 100% reduce 0%
```

The job provides the following confusion matrix:

```
=======================================================
Summary
-------------------------------------------------------
Correctly Classified Instances          :      17531       77.7635%
Incorrectly Classified Instances        :       5013       22.2365%
Total Classified Instances              :      22544

=======================================================
Confusion Matrix
-------------------------------------------------------
a       b       <--Classified as
9396    315      |  9711        a     = normal
4698    8135     |  12833       b     = anomaly

=======================================================
Statistics
-------------------------------------------------------
Kappa                                       0.57
Accuracy                                  77.7635%
Reliability                               53.3825%
Reliability (standard deviation)           0.4915
```

So, from the confusion matrix, it is clear that 9,396 instances were correctly classified and 315 normal instances were incorrectly classified as anomalies. And the accuracy percentage is 77.7635 (correctly classified instances by the model / classified instances). The output file in the prediction folder contains the list where 0 and 1. 0 defines the normal dataset and 1 defines the anomaly.

Summary

In this chapter, we discussed the Random forest algorithm. We started our discussion by understanding the decision tree and continued with an understanding of the Random forest. We took up the NSL-KDD dataset, which is used to build predictive systems for cyber security. We used Mahout to build the Random forest tree, and used it with the test dataset and generated the confusion matrix and other statistics for the output.

In the next chapter, we will look at the final classification algorithm available in Apache Mahout. So stay tuned!

Chapter 7. Learning Multilayer Perceptron Using Mahout

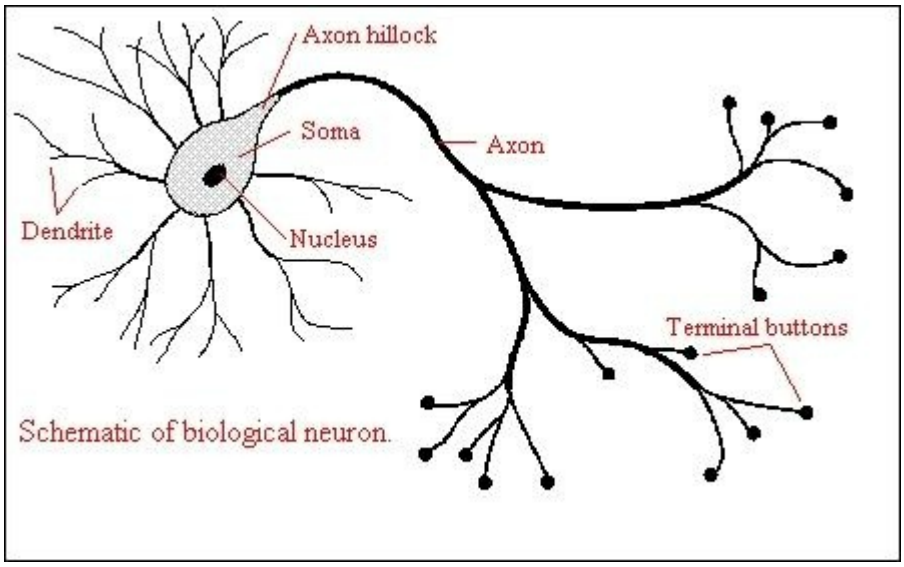
To understand a **Multilayer Perceptron** (**MLP**), we will first explore one more popular machine learning technique: **neural network**. In this chapter, we will explore the following topics:

- Neural network and neurons
- MLP
- Using Mahout for MLP implementation

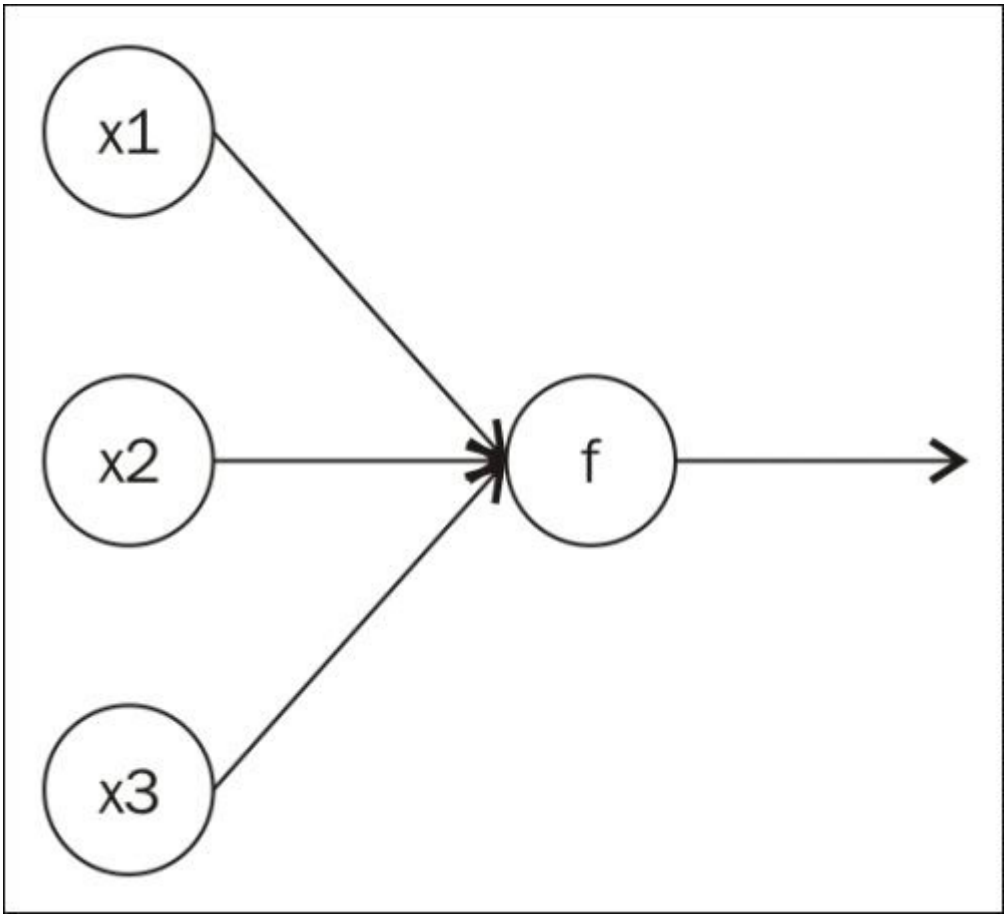
Neural network and neurons

Neural network is an old algorithm, and it was developed with a goal in mind: to provide the computer with a brain. Neural network is inspired by the biological structure of the human brain where multiple neurons are connected and form columns and layers. A **neuron** is an electrically excitable cell that processes and transmits information through electrical and chemical signals. Perceptual input enters into the neural network through our sensory organs and is then further processed into higher levels. Let's understand how neurons work in our brain.

Neurons are computational units in the brain that collect the input from input nerves, which are called **dendrites**. They perform computation on these input messages and send the output using output nerves, which are called **axons**. See the following figure (http://vv.carleton.ca/~neil/neural/neuron-a.html):



On the same lines, we develop a neural network in computers. We can represent a neuron in our algorithm as shown in the following figure:



Here, **x1**, **x2**, and **x3** are the feature vectors, and they are assigned to a function **f**, which will do the computation and provide the output. This activation function is usually chosen from the family of sigmoidal functions (as defined in Chapter 3, *Learning Logistic Regression / SGD Using Mahout*). In the case of classification problems, softmax activation functions are used. In classification problems, we want the output as the probabilities of target classes. So, it is desirable for the output to lie between 0 and 1 and the sum close to 1. Softmax function enforces these constraints. It is a generalization of the logistic function. More details on softmax function can be found at http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-12.html.

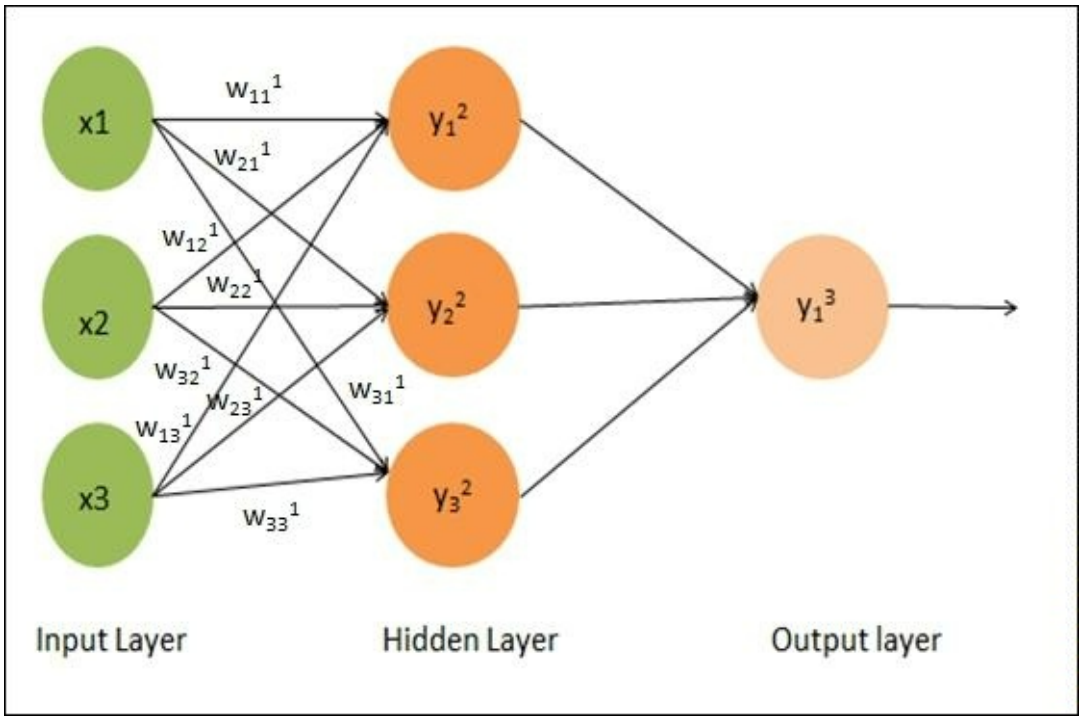
Multilayer Perceptron

A neural network or artificial neural network generally refers to an MLP network. We defined neuron as an implementation in computers in the previous section. An MLP network consists of multiple layers of these neuron units. Let's understand a perceptron network of three layers, as shown in the next figure. The first layer of the MLP represents the input and has no other purpose than routing the input to every connected unit in a feed-forward fashion. The second layer is called hidden layers, and the last layer serves the special purpose of determining the output. The activation of neurons in the hidden layers can be defined as the sum of the weight of all the input. Neuron 1 in layer 2 is defined as follows:

Y12 = g(w110x0 +w111x1+w112x2+w113x3)

The first part where *x0 = 0* is called the bias and can be used as an offset, independent of the input. Neuron 2 in layer 2 is defined as follows:

Y22 = g(w120x0 +w121x1+w122x2+w123x3)



Neuron 3 in layer 2 is defined as follows:

Y32 = g (w130x0 +w131x1+w132x2+w133x3)

Here, g is a sigmoid function, as defined in Chapter 3, *Learning Logistic Regression / SGD Using Mahout*. The function is as follows:

g(z) = 1/1+e (-z)

In this MLP network output, from each input and hidden layers, neuron units are distributed to other nodes, and this is why this type of network is called a fully connected network. In this network, no values are fed back to the previous layer. (Feed forward is another strategy and is also known as back propagation. Details on this can be found at http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html.)

An MLP network can have more than one hidden layer. To get the value of the weights so that we can get the predicted value as close as possible to the actual one is a training process of the MLP. To build an effective network, we consider a lot of items such as the number of hidden layers and neuron units in each layer, the cost function to minimize the error in predicted and actual values, and so on.

Now let's discuss two more important and problematic questions that arise when creating an MLP network:

- How many hidden layers should one use for the network?
- How many numbers of hidden units (neuron units) should one use in a hidden layer?

Zero hidden layers are required to resolve linearly separable data. Assuming your data does require separation by a non-linear technique, always start with one hidden layer. Almost certainly, that's all you will need. If your data is separable using an MLP, then this MLP probably only needs a single hidden layer. In order to select the number of units in different layers, these are the guidelines:

- **Input layer**: This refers to the number of explanatory variables in the model plus one for the bias node.
- **Output layer**: In the case of classification, this refers to the number of target variables, and in the case of regression, this is obviously one.
- **Hidden layer**: Start your network with one hidden layer and use the number of neuron units equivalent to the units in the input layer. The best way is to train several neural networks with different numbers of hidden layers and hidden neurons and measure the performance of these networks using cross-validation. You can stick with the number that yields the best-performing network. Problems that require two hidden layers are rarely encountered. However, neural networks that have more than one hidden layer can represent functions with any kind of shape. There is currently no theory to justify the use of neural networks with more than two hidden layers. In fact, for many practical problems, there is no reason to use any more than one hidden layer. A network with no hidden layer is only capable of representing linearly separable functions. Networks with one layer can approximate any

function that contains a continuous mapping from one finite space to another, and networks with two hidden layers can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy (Chapter 5 of the book *Introduction to Neural Networks for Java*).

- **Number of neurons or hidden units**: Use the number of neuron units equivalent to the units in the input layer. The number of hidden units should be less than twice the number of units in the input layer. Another rule to calculate this is *(number of input units + number of output units)* 2/3*.

Do the testing for generalization errors, training errors, bias, and variance. When a generalization error dips, then just before it begins to increase again, the numbers of nodes are usually found to be perfect at this point.

Now let's move on to the next section and explore how we can use Mahout for an MLP.

MLP implementation in Mahout

The MLP implementation is based on a more general neural network class. It is implemented to run on a single machine using Stochastic Gradient Descent, where the weights are updated using one data point at a time.

The number of layers and units per layer can be specified manually and determines the whole topology with each unit being fully connected to the previous layer. A bias unit is automatically added to the input of every layer. A bias unit is helpful for shifting the activation function to the left or right. It is like adding a coefficient to the linear function.

Currently, the logistic sigmoid is used as a squashing function in every hidden and output layer.

The command-line version does not perform iterations that lead to bad results on small datasets. Another restriction is that the CLI version of the MLP only supports classification, since the labels have to be given explicitly when executing the implementation in the command line.

A learned model can be stored and updated with new training instances using the `--update` flag. The output of the classification result is saved as a `.txt` file and only consists of the assigned labels. Apart from the command-line interface, it is possible to construct and compile more specialized neural networks using the API and interfaces in the `mrlegacy` package. (The core package is renamed as `mrlegacy`.)

In the command line, we use `TrainMultilayerPerceptron` and `RunMultilayerPerceptron` classes that are available in the `mrlegacy` package with three other classes: Neural `network.java`, `NeuralNetworkFunctions.java`, and `MultilayerPerceptron.java`. For this particular implementation, users can freely control the topology of the MLP, including the following:

- The size of the input layer
- The number of hidden layers
- The size of each hidden layer
- The size of the output layer
- The cost function
- The squashing function

The model is trained in an online learning approach, where the weights of neurons in the MLP is updated and incremented using the backPropagation algorithm proposed by *Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986), Learning representations by back-propagating errors. Nature, 323, 533-536*.

Using Mahout for MLP

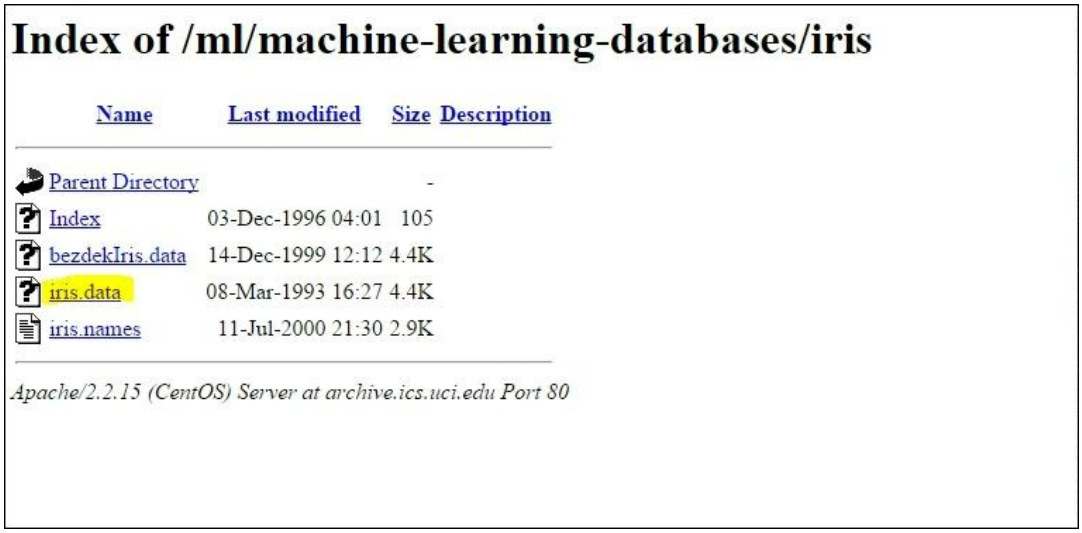
Mahout has implementation for an MLP network. The MLP implementation is currently located in the `Map-Reduce-Legacy` package. As with other classification algorithms, two separated classes are implemented to train and use this classifier. For training the classifier, the `org.apache.mahout.classifier.mlp.TrainMultilayerPerceptron` class, and for running the classifier, the `org.apache.mahout.classifier.mlp.RunMultilayerPerceptron` class is used. There are a number of parameters defined that are used with these classes, but we will discuss these parameters once we run our example on a dataset.

Dataset

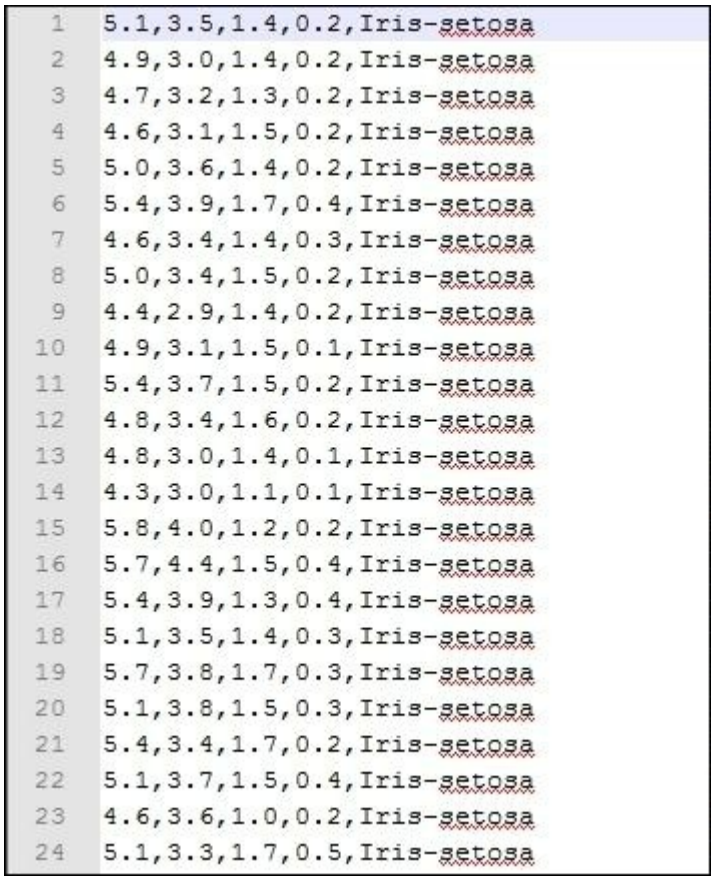
In this chapter, we will train an MLP to classify the iris dataset. The iris flower dataset contains data of three flower species, where each data point consists of four features. This dataset was introduced by Sir Ronald Fisher. It consists of 50 samples from each of three species of iris. These species are Iris setosa, Iris virginica, and Iris versicolor. Four features were measured from each sample:

- Sepal length
- Sepal width
- Petal length
- Petal width

All measurements are in centimeters. You can download this dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/iris/ and save it as a `.csv` file, as shown in the following screenshot:



This dataset will look like the the following screenshot:



```
1  5.1,3.5,1.4,0.2,Iris-setosa
2  4.9,3.0,1.4,0.2,Iris-setosa
3  4.7,3.2,1.3,0.2,Iris-setosa
4  4.6,3.1,1.5,0.2,Iris-setosa
5  5.0,3.6,1.4,0.2,Iris-setosa
6  5.4,3.9,1.7,0.4,Iris-setosa
7  4.6,3.4,1.4,0.3,Iris-setosa
8  5.0,3.4,1.5,0.2,Iris-setosa
9  4.4,2.9,1.4,0.2,Iris-setosa
10 4.9,3.1,1.5,0.1,Iris-setosa
11 5.4,3.7,1.5,0.2,Iris-setosa
12 4.8,3.4,1.6,0.2,Iris-setosa
13 4.8,3.0,1.4,0.1,Iris-setosa
14 4.3,3.0,1.1,0.1,Iris-setosa
15 5.8,4.0,1.2,0.2,Iris-setosa
16 5.7,4.4,1.5,0.4,Iris-setosa
17 5.4,3.9,1.3,0.4,Iris-setosa
18 5.1,3.5,1.4,0.3,Iris-setosa
19 5.7,3.8,1.7,0.3,Iris-setosa
20 5.1,3.8,1.5,0.3,Iris-setosa
21 5.4,3.4,1.7,0.2,Iris-setosa
22 5.1,3.7,1.5,0.4,Iris-setosa
23 4.6,3.6,1.0,0.2,Iris-setosa
24 5.1,3.3,1.7,0.5,Iris-setosa
```

Steps to use the MLP algorithm in Mahout

The steps to use the MLP algorithm in Mahout are as follows:

1. Create the MLP model.

   To create the MLP model, we will use the `TrainMultilayerPerceptron` class. Use the following command to generate the model:

```
bin/mahout
org.apache.mahout.classifier.mlp.TrainMultilayerPerceptron
-
i /tmp/irisdata.csv -labels Iris-setosa
Iris-
versicolor Iris-virginica
-
mo /tmp/model.model -ls 4 8 3 -l 0.2 -m 0.35
-r 0.0001
```

You can also run using the core jar: Mahout core jar (`xyz` stands for the version). If you have directly installed Mahout, it can be found under the `/usr/lib/mahout` folder. Execute the following command:

```
Java –cp /usr/lib/mahout/
mahout-core-xyz-
job.jar
org.apache.mahout.classifier.mlp.TrainMultilayerPerceptron
-
i /tmp/irisdata.csv -labels Iris-setosa
Iris-
versicolor Iris-virginica
-
mo /user/hue/mlp/model.model -ls 4 8 3 -l 0.2
-m 0.35 -r 0.0001
```

The `TrainMultilayerPerceptron` class is used here and it takes different parameters. Also, `i` is the path for the input dataset. Here, we have put the dataset under the `/tmp` folder (local filesystem). Additionally, labels are defined in the dataset. Here we have the following labels:

- `mo` is the output location for the created model.
- `ls` is the number of units per layer, including input, hidden, and output layers. This parameter specifies the topology of the network. Here, we have `4` as the input feature, `8` for the hidden layer, and `3` for the output class number.
- `l` is the learning rate that is used for weight updates. The default is 0.5. To approximate gradient descent, neural networks are trained with algorithms. Learning is possible either by batch or online methods. In batch training, weight changes are accumulated over an entire presentation of the training data (an epoch) before being applied, while online training updates weighs after the presentation of each training example (instance). More details can be found at http://axon.cs.byu.edu/papers/Wilson.nn03.batch.pdf.
- `m` is the momentum weight that is used for gradient descent. This must be in the range between 0–1.0.
- `r` is the regularization value for the weight vector. This must be in the range between 0–0.1. It is used to prevent overfitting.



2. To test/run the MLP classification of the trained model, we can use the following command:

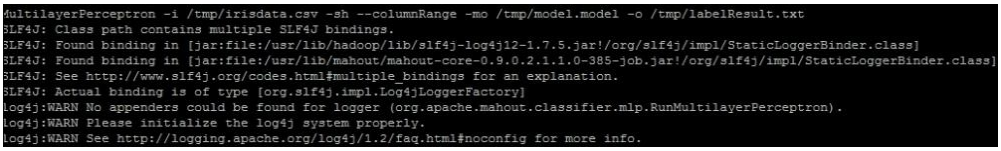
```
bin/mahout
org.apache.mahout.classifier.mlp.RunMultilayerPerceptron
-
i /tmp/irisdata.csv -cr 0 3
-mo /tmp/model.model -o /tmp/labelResult.txt
```

You can also run using the Mahout core jar (`xyz` stands for version). If you have directly installed Mahout, it can be found under the `/usr/lib/mahout` folder. Execute the following command:

```
Java -cp /usr/lib/mahout/
mahout-core-xyz-
job.jar
org.apache.mahout.classifier.mlp.RunMultilayerPerceptron
-
i /tmp/irisdata.csv -cr 0 3
-mo /tmp/model.model -o /tmp/labelResult.txt
```

The `RunMultilayerPerceptron` class is employed here to use the model. This class also takes different parameters, which are as follows:

- `i` indicates the input dataset location
- `cr` is the range of columns to use from the input file, starting with 0 (that is, `` `-cr 0 5` `` for including the first six columns only)
- `mo` is the location of the model built earlier
- `o` is the path to store labeled results from running the model



Summary

In this chapter, we discussed one of the newly implemented algorithms in Mahout: MLP. We started our discussion by understanding neural networks and neuron units and continued our discussion further to understand the MLP network algorithm. We discussed how to choose different layer units. We then moved to Mahout and used the iris dataset to test and run an MLP algorithm implemented in Mahout. With this, we have finished our discussion on classification algorithms available in Apache Mahout.

Now we move on to the next chapter of this book where we will discuss the new changes coming up in the new Mahout release.

Chapter 8. Mahout Changes in the Upcoming Release

Mahout is a community-driven project and its community is very strong. This community decided on some of the major changes in the upcoming 1.0 release. In this chapter, we will explore the upcoming changes and developments in Apache Mahout. We will look at the following topics in brief:

- New changes due in Mahout 1.0
- Apache Spark
- H20-platform-related work in Apache Mahout

Mahout new changes

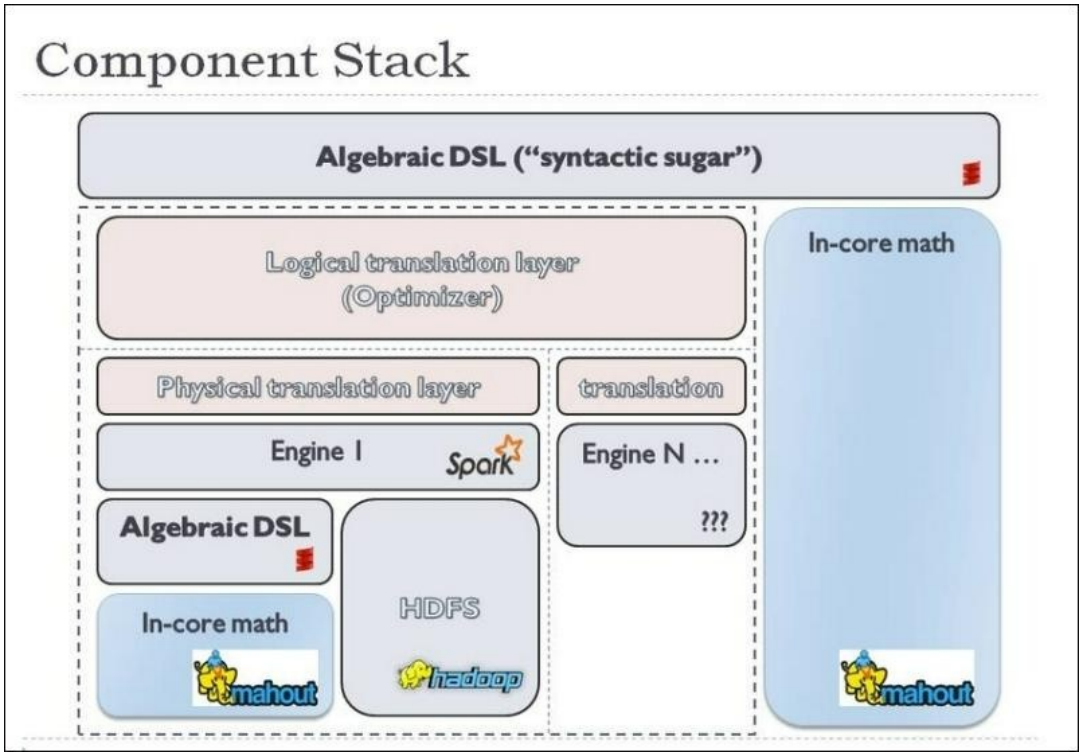
Mahout was using the map reduce programming model to handle large datasets. From the end of April 2014, the community decided to stop the implementation of the new map reduce algorithm. This decision has a valid reason. Mahout's codebase will be moving to modern data processing systems that offer a richer programming model and more efficient execution than Hadoop's MapReduce.

Mahout has started its implementation on the top of **Domain Specific Language** (**DSL**) for linear algebraic operations. Programs written in this DSL are automatically optimized and executed in parallel on Apache Spark. Scala DSL and algebraic optimizer is Scala and Spark binding for Mahout.

Mahout Scala and Spark bindings

With Mahout Scala bindings and Mahout Spark bindings for linear algebra subroutines, developers in Mahout are trying to bring semantic explicitness to Mahout's in-core and out-of-core linear algebra subroutines. They are doing this while adding the benefits of the strong programming environment of Scala and capitalizing on scalability benefits of Spark and GraphX. Scala binding is used to provide support for Scala DSL, and this will make writing machine learning programs easier.

Mahout Scala and Spark bindings are packages that aim to provide an R-like look and feel to Mahout's in-core and out-of-core Spark-backed linear algebra. An important part of Spark bindings is the expression optimizer. This optimizer looks at the entire expression and decides on how it can be simplified and which physical operators should be picked. A high-level diagram of the binding stack is shown in the following figure (https://issues.apache.org/jira/secure/attachment/12638098/BindingsStack.jpg):



The Spark binding shell has also been implemented in Mahout 1.0. Let's understand the Apache Spark project first and then we will revisit the Spark binding shell in Mahout.

Apache Spark

Apache Spark is an open source, in-memory, general-purpose computing system. Spark's in-memory technique provides performance that is 100 times faster. Instead of Hadoop-like disk-based computation, Spark uses cluster memory to upload all the data into the memory, and this data can be queried repeatedly.

Apache Spark provides high-level APIs in Java, Python, and Scala and an optimized engine that supports general execution graphs. It provides the following high-level tools:

- **Spark SQL**: This is for SQL and structured data processing.
- **MLib**: This is Spark's scalable machine learning library that consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as the underlying optimization primitives.
- **GraphX**: This is the new Spark API for graphs and graph-parallel computation.
- **Spark streaming**: This can collect data from many sources and after processing this data, it uses complex algorithms and can push the data to filesystems, databases, and live dashboards.

As Spark is gaining popularity among data scientists, the Mahout community is also quickly working on making Mahout algorithms function on Spark's execution engine to speed up its calculation 10 to 100 times faster. Mahout provides several important building blocks to create recommendations using Spark. Spark-item similarity can be used to create *other people also liked these things* kind of recommendations and when paired with a search engine can personalize recommendations for individual users. Spark-row similarity can provide non-personalized content based on recommendations and when paired with a search engine can be used to personalize content based on recommendations (http://comments.gmane.org/gmane.comp.apache.mahout.scm/6513).

Using Mahout's Spark shell

You can use Mahout's Spark shell by referring to the following steps:

1. Download Spark from http://spark.apache.org/downloads.html.
2. Create a new folder with the name `spark` using the following command and move the downloaded file there:

```
mkdir /tmp/spark
mv ~/Downloads/spark-1.1.1.tgz/tmp/spark
```

3. Unpack the archived file in a folder using the following command:

```
cd /tmp/spark
tar xzf spark-1.1.1.tgz
```

4. This will unzip the file `under/tmp/spark/spark-1.1.1`. Now, move to the newly created folder and run the following command:

```
cd /spark-1.1.1
sbt/sbt assembly
```

This will build Spark on your system as shown in the following screenshot:



5. Now create a Mahout directory and move the file to it using the following command:

```
mkdir /tmp/Mahout
```

6. Check out the master branch of Mahout from GitHub using the following command:

```
git clone https://github.com/apache/mahout
mahout
```

The output of the preceding command is shown in the following screenshot:

```
# git clone https://github.com/apache/mahout mahout
Initialized empty Git repository in /tmp/Mahout/mahout/.git/
remote: Counting objects: 79953, done.
Receiving objects: 100% (79953/79953), 39.36 MiB | 218 KiB/s, done.
remote: Total 79953 (delta 0), reused 0 (delta 0)
Resolving deltas: 100% (43010/43010), done.
```

7. Change your directory to the newly created Mahout directory and build Mahout:

```
cd mahout
mvn -DskipTests clean install
```

The output of the preceding command is shown in the following screenshot:

```
]# mvn -DskipTests clean install
/usr/local/maven/apache-maven-3.2.3/bin/mvn: line 53: uname: command not found
[INFO] Scanning for projects...
Downloading: https://repository.cloudera.com/artifactory/cloudera-repos/org/apache/apache/9/apache-9.pom
Downloading: http://repository.mapr.com/maven/org/apache/apache/9/apache-9.pom
Dec 03, 2014 9:19:33 AM org.apache.maven.wagon.providers.http.httpclient.client.protocol.ResponseProcessCookies processCookies
WARNING: Cookie rejected: "[version: 0][name: rememberMe][value: deleteMe][domain: repository.mapr.com][path: /nexus][expiry:
ch attribute "/nexus". Path of origin: "/maven/org/apache/apache/9/apache-9.pom"
Downloading: https://repo.maven.apache.org/maven2/org/apache/apache/9/apache-9.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/apache/9/apache-9.pom (15 KB at 8.1 KB/sec)
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Mahout Build Tools
[INFO] Apache Mahout
[INFO] Mahout Math
[INFO] Mahout MapReduce Legacy
[INFO] Mahout Integration
[INFO] Mahout Examples
[INFO] Mahout Release Package
[INFO] Mahout Math Scala bindings
[INFO] Mahout Spark bindings
[INFO] Mahout Spark bindings shell
[INFO] Mahout H2O backend
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Mahout Build Tools 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-remote-resources-plugin/1.1/maven-remote-reso
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-remote-resources-plugin/1.1/maven-remote-resou
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/14/maven-plugins-14.pom
```

8. Move to the directory where you unpacked Spark and type the following command to start Spark locally:

```
cd /tmp/spark/spark-1.1.1
sbin/start-all-sh
```

The output of the preceding command is shown in the following screenshot:

```
spark-1.1.1]# sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /tmp/spark/spark-1.1.
1/sbin/../logs/spark-root-org.apache.spark.deploy.master.Master-1-sandbox.horton
works.com.out
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /tmp/spark
/spark-1.1.1/sbin/../logs/spark-root-org.apache.spark.deploy.worker.Worker-1-san
dbox.hortonworks.com.out
```

9. Open a browser; point it to `http://localhost:8080/` to check whether Spark has successfully started. Copy the URL of the Spark master at the top of the page (it starts with `spark://`).
10. Define the following environment variables:

```
export MAHOUT_HOME=[directory into which you
checked out Mahout]
export SPARK_HOME=[directory where you
unpacked Spark]
export MASTER=[url of the Spark master]
```

11. Finally, change to the directory where you unpacked Mahout and type `bin/mahout spark-shell`; you should see the shell starting and get the `mahout>` prompt.

Now your Mahout Spark shell is ready and you can start playing with data. For more information on this topic, see the implementation section at https://mahout.apache.org/users/sparkbindings/play-with-shell.html.

H2O platform integration

As discussed earlier, an experimental work to integrate Mahout and the H2O platform is also in progress. The integration provides an H2O backend to the Mahout algebra DSL.

H2O makes Hadoop do math! H2O scales statistics, machine learning, and math over big data. It is extensible and users can build blocks using simple math legos in the core. H2O keeps familiar interfaces such as R, Excel, and JSON so that big data enthusiasts and experts can explore, munge, model, and score datasets using a range of simple-to-advanced algorithms. Data collection is easy, while decision making is hard. H2O makes it fast and easy to derive insights from your data through faster and better predictive modeling. It also has a vision of online scoring and modeling in a single platform (http://0xdata.com/download/).

H2O is fundamentally a peer-to-peer system. H2O nodes join together to form a cloud on which high-performance distributed math can be executed. Each node joins a cloud of a given name. Multiple clouds can exist on the same network at the same time as long as their names are different. Multiple nodes can exist on the same server as well (they can even belong to the same cloud).

The Mahout H2O integration is fit into this model by having N-1 worker nodes and one driver node, all belonging to the same cloud name. The default cloud name used for the integration is `mah2out`. Clouds have to be spun up as per their task/job.

More details can be found at https://issues.apache.org/jira/browse/MAHOUT-1500.

Summary

In this chapter, we discussed the upcoming release of Mahout 1.0, and the changes that are currently going on. We also glanced through Spark, Scala binding, and Apache Spark. We also discussed a high-level overview of H2O Mahout integration.

Now let's move on to the final chapter of this book where we will develop a production-ready classifier.

Chapter 9. Building an E-mail Classification System Using Apache Mahout

In this chapter, we will create a classifier system using Mahout. In order to build this system, we will cover the following topics:

- Getting the dataset
- Preparation of the dataset
- Preparing the model
- Training the model

In this chapter, we will target the creation of two different classifiers. The first one will be an easy one because you can both create and test it on a pseudo-distributed Hadoop installation. For the second classifier, I will provide you with all the details, so you can run it using your fully distributed Hadoop installation. I will count the second one as a hands-on exercise for the readers of this book.

First of all, let's understand the problem statement for the first use case. Nowadays, in most of the e-mail systems, we see that e-mails are classified as spam or not spam. E-mails that are not spam are delivered directly into our inbox but spam e-mails are stored in a folder called `Spam`. Usually, based on a certain pattern such as message subject, sender's e-mail address, or certain keywords in the message body, we categorize an incoming e-mail as spam. We will create a classifier using Mahout, which will classify an e-mail into spam or not spam. We will use SpamAssassin, an Apache open source project dataset for this task.

For the second use case, we will create a classifier, which can predict a group of incoming e-mails. As an open source project, there are lots of projects under the Apache software foundation, such as Apache Mahout, Apache Hadoop, Apache Solr, and so on. We will take the **Apache Software Foundation** (**ASF**) e-mail dataset and using this, we will create and train our model so that our model can predict a new incoming e-mail. So, based on certain features, we will be able to predict which group a new incoming e-mail belongs to.

In Mahout's classification problem, we will have to identify a pattern in the dataset to help us predict the group of a new e-mail. We already have a dataset, which is separated by project names. We will use the ASF public e-mail archives dataset for this use case.

Now, let's consider our first use case: spam e-mail detection classifier.

Spam e-mail dataset

As I mentioned, we will be using the Apache SpamAssassin projects dataset. Apache SpamAssassin is an open source spam filter. Download `20021010_easy_ham.tar` and `20021010_spam.tar` from http://spamassassin.apache.org/publiccorpus/, as shown in the following screenshot:



Index of /publiccorpus

Name	Last modified	Size	Description
Parent Directory		-	
20021010_easy_ham.tar.bz2	2004-06-29 03:26	1.6M	
20021010_hard_ham.tar.bz2	2004-12-16 19:49	1.0M	
20021010_spam.tar.bz2	2004-06-29 03:26	1.1M	
20030228_easy_ham.tar.bz2	2004-06-29 03:26	1.5M	
20030228_easy_ham_2.tar.bz2	2004-06-29 03:26	1.0M	
20030228_hard_ham.tar.bz2	2004-12-16 19:49	1.0M	
20030228_spam.tar.bz2	2004-06-29 03:26	1.1M	
20030228_spam_2.tar.bz2	2004-06-29 03:26	2.0M	
20050311_spam_2.tar.bz2	2005-03-11 23:55	2.0M	
obsolete/	2014-02-04 16:26	-	
readme.html	2006-01-31 20:30	4.5K	

Apache/2.4.10 (Unix) OpenSSL/1.0.1i Server at spamassassin.apache.org Port 80

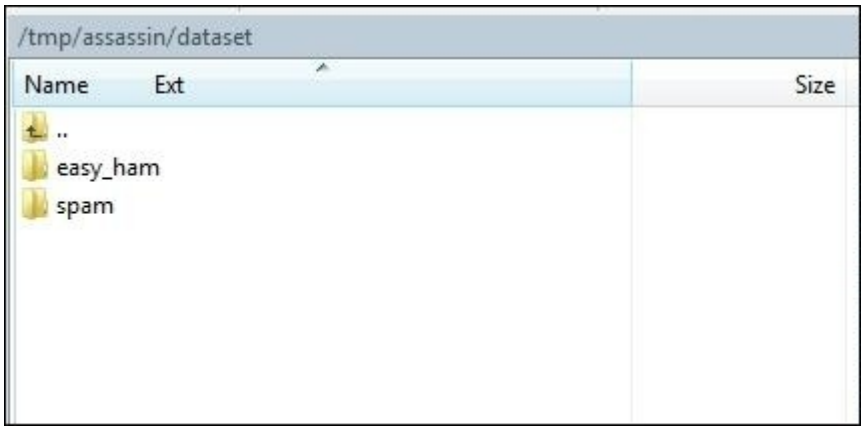
Creating the model using the Assassin dataset

We can create the model with the help of the following steps:

1. Create a folder under `tmp` with the name `dataset`, and then click on the folder and unzip the datasets using the following command:

```
mkdir /tmp/assassin/dataset
tar -xvf  /tmp/assassin/
20021010_easy_ham.tar.bz2
tar -xvf /tmp/assassin/ 20021010_spam.tar.bz2
```

   This will create two folders under the `dataset` folder, `easy _ham` and `spam`, as shown in the following screenshot:



2. Create a folder in `Hdfs` and move this dataset into Hadoop:

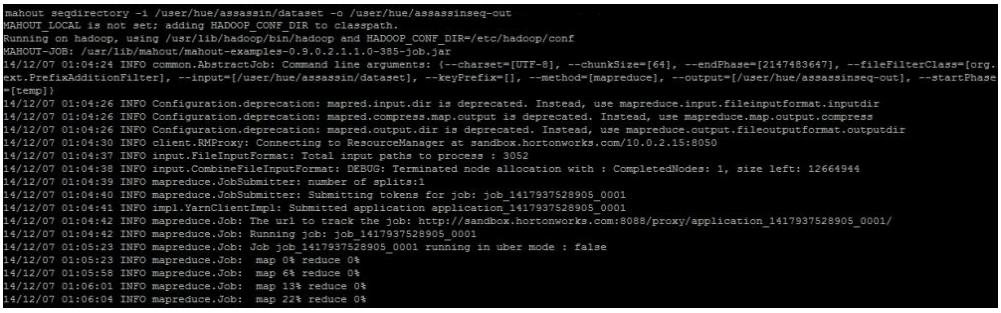
```
hadoop fs  -mkdir /user/hue/assassin/
hadoop fs -put /tmp/assassin/dataset
 /user/hue/assassin
tar -xvf /tmp/assassin/ 20021010_spam.tar.bz2
```

   Now our data preparation is done. We have downloaded the data and moved this data into `hdfs`. Let's move on to the next step.

3. Convert this data into sequence files so that we can process it using Hadoop:

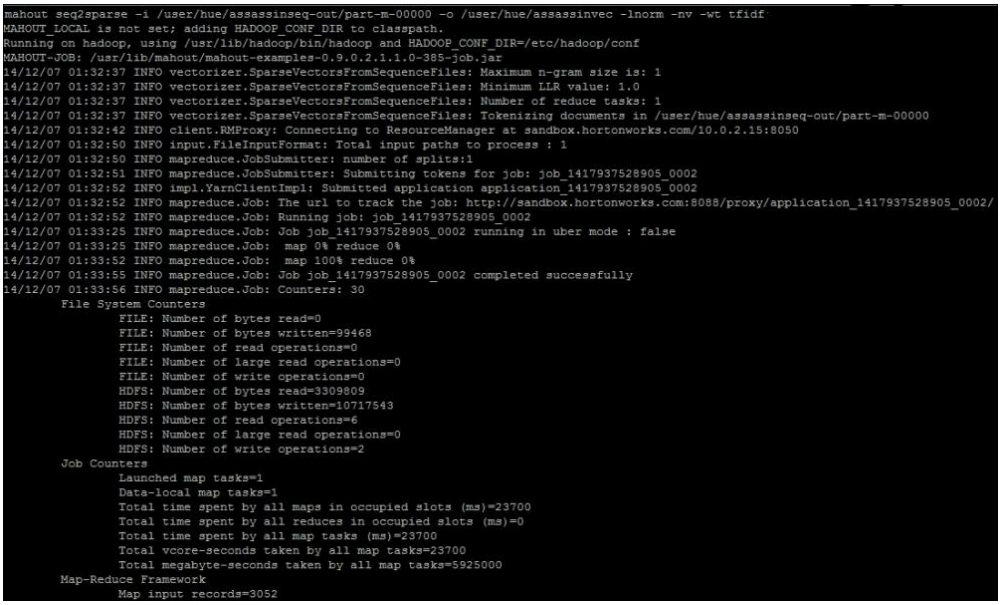
```
bin/mahout seqdirectory
-
```

```
i /user/hue/assassin/dataset
-o /user/hue/assassinseq-out
```



4. Convert the `sequence` file into sparse vector (Mahout algorithms accept input in vector format, which is why we are converting the `sequence` file into sparse vector) by using the following command:

```
bin/mahout seq2sparse
-i /user/hue/assassinseq-out/part-m-
00000 -o /user/hue/assassinvec -lnorm -nv -wt
tfidf
```



The command in the preceding screenshot is explained as follows:

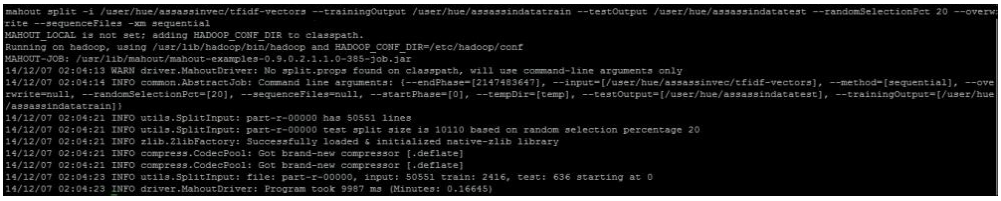
- `lnorm`: This command is used for output vector to be log normalized.
- `nv`: This command is used for named vector.
- `wt`: This command is used to identify the kind of weight to use. Here we use `tf-idf`.

5. Split the set of vectors for training and testing the model, as follows:

```
bin/mahout split
-i /user/hue/assassinvec/tfidf-
vectors
--
trainingOutput /user/hue/assassindatatrain
--
testOutput /user/hue/assassindatatest
--
randomSelectionPct 20 --overwrite
--sequenceFiles -xm sequential
```

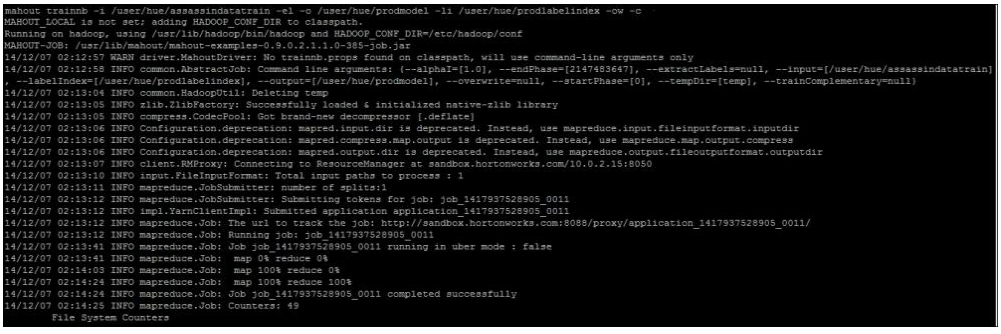
The preceding command can be explained as follows:

- The `randomSelectionPct` parameter divides the percentage of data into test and training datasets. In this case, it's 80 percent for test and 20 percent for training.
- The `xm` parameter specifies what portion of the `tf (tf-idf)` vectors is to be used expressed in times the standard deviation.
- The sigma symbol specifies the document frequencies of these vectors. It can be used to remove really high frequency terms. It is expressed as a double value. A good value to be specified is 3.0. If the value is less than `0`, no vectors will be filtered out.



6. Now, train the model using the following command:

```
bin/mahout trainnb
-
i /user/hue/assassindatatrain -el
-
o /user/hue/prodmodel
-li /user/hue/prodlabelindex -ow -c
```



7. Now, test the model using the following command:

```
bin/mahout testnb
-
i /user/hue/assassindatatest
-
m /user/hue/prodmodel/ -l
 /user/hue/prodlabelindex -ow
-o /user/hue/prodresults
```

```
14/12/07 02:22:00 INFO test.TestNaiveBayesDriver: Standard NB Results:
=======================================================
Summary
-------------------------------------------------------
Correctly Classified Instances          :        633       99.5283%
Incorrectly Classified Instances        :          3        0.4717%
Total Classified Instances              :        636

=======================================================
Confusion Matrix
-------------------------------------------------------
a       b       <--Classified as
525     3        |  528         a     = easy_ham
0       108      |  108         b     = spam

=======================================================
Statistics
-------------------------------------------------------
Kappa                                       0.9678
Accuracy                                   99.5283%
Reliability                                66.4773%
Reliability (standard deviation)            0.5757

14/12/07 02:22:00 INFO driver.MahoutDriver: Program took 47941 ms (Minutes: 0.7990333333333334)
```

You can see from the results that the output is displayed on the console. As per the matrix, the system has correctly classified 99.53 percent of the instances given.

We can use this created model to classify new documents. To do this, we can either use a Java program or create a servlet that can be deployed on our server.

Let's take an example of a Java program in continuation of this exercise.

Program to use a classifier model

We will create a Java program that will use our model to classify new e-mails. This program will take model, labelindex, dictionary-file, document frequency, and text file as input and will generate a score for the categories. The category will be decided based on the higher scores.

Let's have a look at this program step by step:

- The `.jar` files required to make a compilation of this program are as follows:
    - `Hadoop-core-x.y.x.jar`
    - `Mahout-core-xyz.jar`
    - `Mahout-integration-xyz.jar`
    - `Mahout-math-xyz.jar`
- The `import` statements are listed as follows. We are discussing this because there are lots of changes in the Mahout releases and people usually find it difficult to get the correct classes.
    - `import java.io.BufferedReader;`
    - `import java.io.FileReader;`
    - `import java.io.StringReader;`
    - `import java.util.HashMap;`
    - `import java.util.Map;`
    - `import org.apache.hadoop.conf.Configuration;`
    - `import org.apache.hadoop.fs.Path;`
    - `import org.apache.lucene.analysis.Analyzer;`
    - `import org.apache.lucene.analysis.TokenStream;`
    - `import org.apache.lucene.analysis.standard.StandardAnalyzer;`
    - `import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;`
    - `import org.apache.lucene.util.Version;`
    - `import org.apache.mahout.classifier.naivebayes.BayesUtils;`
    - `import org.apache.mahout.classifier.naivebayes.NaiveBayesModel;`
    - `import org.apache.mahout.classifier.naivebayes.StandardNaiveBayesClassifier;`
    - `import org.apache.mahout.common.Pair;`
    - `import org.apache.mahout.common.iterator.sequencefile.SequenceFileIterable;`
    - `import org.apache.mahout.math.RandomAccessSparseVector;`

- `import org.apache.mahout.math.Vector;`
- `import org.apache.mahout.math.Vector.Element;`
- `import org.apache.mahout.vectorizer.TFIDF;`
- `import org.apache.hadoop.io.*;`
- `import com.google.common.collect.ConcurrentHashMultiset;`
- `import com.google.common.collect.Multiset;`

- The supporting methods to read the dictionary are as follows:

```
public static Map<String, Integer>
readDictionary(Configuration conf, Path
dictionaryPath) {
  Map<String, Integer> dictionary = new
HashMap<String, Integer>();
  for (Pair<Text, IntWritable> pair : new
SequenceFileIterable<Text,
IntWritable>(dictionaryPath, true, conf)) {

 dictionary.put(pair.getFirst().toString(),
pair.getSecond().get());
  }
  return dictionary;
}
```

- The supporting methods to read the document frequency are as follows:

```
public static Map<Integer, Long>
readDocumentFrequency(Configuration conf,
Path documentFrequencyPath) {
  Map<Integer, Long> documentFrequency = new
HashMap<Integer, Long>();
  for (Pair<IntWritable, LongWritable> pair :
new SequenceFileIterable<IntWritable,
LongWritable>(documentFrequencyPath, true,
conf)) {

 documentFrequency.put(pair.getFirst().get(),
pair.getSecond().get());
  }
  return documentFrequency;
}
```

- The first part of the `main` method is used to perform the following actions:
  - Getting the input
  - Loading the model
  - Initializing `StandardNaiveBayesClassifier` using our created model
  - Reading `labelindex`, document frequency, and dictionary created while creating the vector from the dataset

The following code can be used for the preceding actions:

```
public static void main(String[] args)
throws Exception {
  if (args.length < 5) {
    System.out.println("Arguments:
[model] [labelindex] [dictionary]
[documentfrequency] [new file] ");
    return;
  }
  String modelPath = args[0];
  String labelIndexPath = args[1];
  String dictionaryPath = args[2];
  String documentFrequencyPath = args[3];
  String newDataPath = args[4];
  Configuration configuration = new
Configuration(); // model is a matrix
(wordId, labelId) => probability score
  NaiveBayesModel model =
NaiveBayesModel.materialize(new
Path(modelPath), configuration);
  StandardNaiveBayesClassifier
classifier = new
StandardNaiveBayesClassifier(model);
  // labels is a map label => classId
  Map<Integer, String> labels =
BayesUtils.readLabelIndex(configuration,
new Path(labelIndexPath));
  Map<String, Integer> dictionary =
readDictionary(configuration, new
Path(dictionaryPath));
  Map<Integer, Long> documentFrequency =
readDocumentFrequency(configuration, new
Path(documentFrequencyPath));
```

- The second part of the `main` method is used to extract words from the e-mail:

```
Analyzer analyzer = new
StandardAnalyzer(Version.LUCENE_CURRENT);

int labelCount = labels.size();
int documentCount =
documentFrequency.get(-1).intValue();

System.out.println("Number of labels: " +
labelCount);
System.out.println("Number of documents in
training set: " + documentCount);
BufferedReader reader = new
BufferedReader(new FileReader(newDataPath));
while(true) {
  String line = reader.readLine();
```

```
  if (line == null) {
    break;
  }

  ConcurrentHashMultiset<Object> words =
ConcurrentHashMultiset.create();
  // extract words from mail
  TokenStream ts =
analyzer.tokenStream("text", new
StringReader(line));
  CharTermAttribute termAtt =
ts.addAttribute(CharTermAttribute.class);
  ts.reset();
  int wordCount = 0;
  while (ts.incrementToken()) {
    if (termAtt.length() > 0) {
      String word =
ts.getAttribute(CharTermAttribute.class).toString();
      Integer wordId = dictionary.get(word);
      // if the word is not in the
dictionary, skip it
      if (wordId != null) {
        words.add(word);
        wordCount++;
      }
    }
  }
  ts.close();
```

- The third part of the `main` method is used to create vector of the `id` word and the `tf-idf` weights:

```
Vector vector = new
RandomAccessSparseVector(10000);
TFIDF tfidf = new TFIDF();
for (Multiset.Entry entry:words.entrySet()) {
  String word =  (String)entry.getElement();
  int count = entry.getCount();
  Integer wordId = dictionary.get(word);
  Long freq = documentFrequency.get(wordId);
  double tfIdfValue = tfidf.calculate(count,
freq.intValue(), wordCount, documentCount);
  vector.setQuick(wordId, tfIdfValue);
}
```

- In the fourth part of the `main` method, with `classifier`, we get the score for each label and assign the e-mail to the higher scored label:

```
  Vector resultVector =
classifier.classifyFull(vector);
    double bestScore = -Double.MAX_VALUE;
    int bestCategoryId = -1;
    for(int i=0 ;i<resultVector.size();i++) {
      Element e1  =
resultVector.getElement(i);
```

```
        int categoryId = e1.index();
        double score = e1.get();
        if (score > bestScore) {
          bestScore = score;
          bestCategoryId = categoryId;
        }
        System.out.print("  " +
labels.get(categoryId) + ": " + score);
      }
      System.out.println(" => " +
labels.get(bestCategoryId));
    }
}
```

Now, put all these codes under one class and create the `.jar` file of this class. We will use this `.jar` file to test our new e-mails.

Testing the program

To test the program, perform the following steps:

1. Create a folder named `assassinmodeltest` in the local directory, as follows:

```
mkdir /tmp/assassinmodeltest
```

2. To use this model, get the following files from `hdfs` to `/tmp/assassinmodeltest`:

   - For the earlier created model, use the following command:

```
hadoop fs
-
get /user/hue/prodmodel /tmp/assassinmodeltest
```

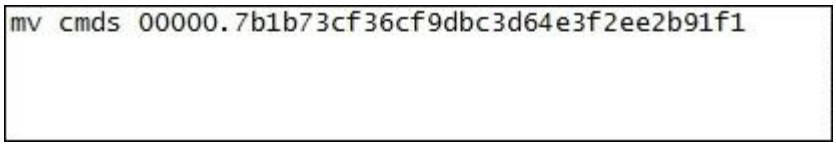
   - For `labelindex`, use the following command:

```
hadoop fs -get /user/hue/prodlabelindex
 /tmp/assassinmodeltest
```

   - For `df-counts` from the `assassinvec` folder (change the name of the `part-00000` file to `df-count`), use the following commands:

```
hadoop fs
-get /user/hue/assassinvec/df-
count  /tmp/assassinmodeltest
dictionary.file-
0 from the same assassinvec folder
hadoop fs
-
get /user/hue/assassinvec/dictionary.file-
0  /tmp/assassinmodeltest
```

3. Under `/tmp/assassinmodeltest`, create a file with the message shown in the following screenshot:



4. Now, run the program using the following command:

```
Java
-
cp /tmp/assassinmodeltest/spamclassifier.jar:/usr/lib/mahout/*
com.packt.spamfilter.TestClassifier /tmp/assassinmodeltest /tmp/assa
0 /tmp/assassinmodeltest/df-
count /tmp/assassinmodeltest/testemail
```

```
java -cp /tmp/assassinmodeltest/spamclassifier.jar:/usr/lib/mahout/* com.packt.spamfilter.TestClassifier /tmp/assassinmodeltest/ /tmp/assassinmodeltest/prodlabelindex
/tmp/assassinmodeltest/dictionary.file-0 /tmp/assassinmodeltest/df-count /tmp/assassinmodeltest/testemail
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Number of labels: 2
Number of documents in training set: 3052
  easy_ham: -91.25256617072814  spam: -60.78830055665665 => spam
```

5. Now, update the `test` e-mail file with the message shown in the following screenshot:

```
Delivered-To: exmh-workers@listman.example.com
```

6. Run the program again using the same command as given in step 4 and view the result as follows:

```
java -cp /tmp/assassinmodeltest/spamclassifier.jar:/usr/lib/mahout/* com.packt.spamfilter.TestClassifier /tmp/assassinmodeltest/ /tmp/assassinmodeltest/prodlabelindex
/tmp/assassinmodeltest/dictionary.file-0 /tmp/assassinmodeltest/df-count /tmp/assassinmodeltest/testemail
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Number of labels: 2
Number of documents in training set: 3052
  easy_ham: -88.32010546373571  spam: -159.80191214024765 => easy_ham
```

Now, we have a program ready that can use our classifier model and predict the unknown items. Let's move on to our second use case.

Second use case as an exercise

As discussed at the start of this chapter, we will now work on a second use case, where we will predict the category of a new e-mail.

The ASF e-mail dataset

The Apache Software Foundation e-mail dataset is partitioned by project. This e-mail dataset can be found at http://aws.amazon.com/datasets/7791434387204566.



A smaller dataset can be found at http://files.grantingersoll.com/ibm.tar.gz. (Refer to http://lucidworks.com/blog/scaling-mahout/). Use this data to perform the following steps:

1. Move this data to the folder of your choice (`/tmp/asfmail`) and unzip the folder:

```
mkdir /tmp/asfmail
tar -xvf  ibm.tar
```

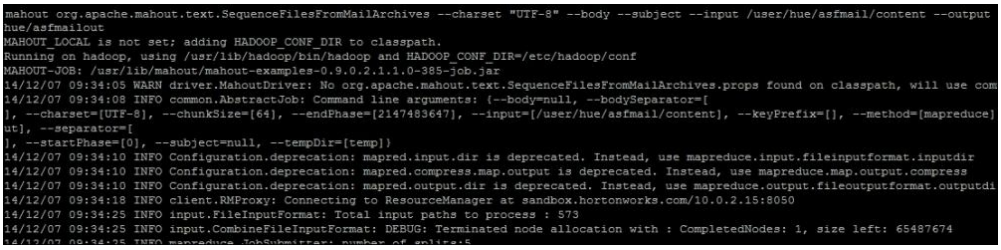
2. Move the dataset to `hdfs`:

```
hadoop fs
-
put /tmp/asfmail/ibm/content /user/hue/asfmail
```

3. Convert the `mbox` files into Hadoop's `SequenceFile` format using Mahout's `SequenceFilesFromMailArchives` as follows:

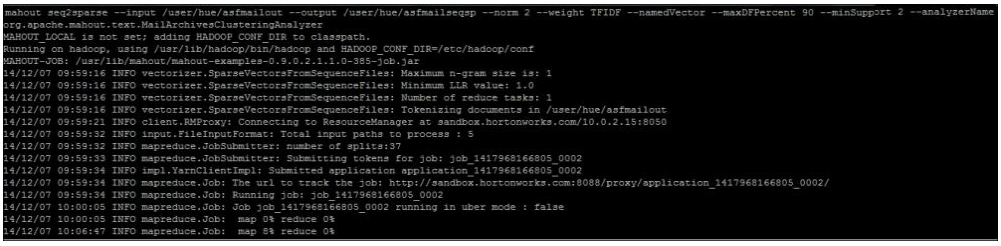
```
mahout
org.apache.mahout.text.SequenceFilesFromMailArchives
```

```
--
charset "UTF-8" --body --subject
--
input /user/hue/asfmail/content
--output /user/hue/asfmailout
```



4. Convert the `sequence` file into sparse vector:

```
mahout  seq2sparse
--
input /user/hue/asfmailout
--
output /user/hue/asfmailseqsp --norm 2
--
weight TFIDF --namedVector --maxDFPercent 90
--
minSupport 2 --analyzerName
org.apache.mahout.text.MailArchivesClusteringAnalyzer
```



5. Modify the labels:

```
mahout
org.apache.mahout.classifier.email.PrepEmailDriver
--
input /user/hue/asfmailseqsp
--
output /user/hue/asfmailseqsplabel
--maxItemsPerLabel 1000
```

Now, the next three steps are similar to the ones we performed earlier:

1. Split the dataset into `training` and `test` datasets using the following command:

```
mahout  split
--
input /user/hue/asfmailseqsplabel
--
trainingOutput /user/hue/asfmailtrain
--
testOutput /user/hue/asfmailtest
--
randomSelectionPct 20 --overwrite
--sequenceFiles
```

2. Train the model using the `training` dataset as follows:

```
mahout trainnb -i /user/hue/asfmailtrain
-
o /user/hue/asfmailmodel -extractLabels
--labelIndex /user/hue/asfmaillabels
```

3. Test the model using the `test` dataset:

```
mahout testnb -i /user/hue/asfmailtest
-
m /user/hue/asfmailmodel
--labelIndex /user/hue/asfmaillabels
```

As you may have noticed, all the steps are exactly identical to the ones we performed earlier. Hereby, I leave this topic as an exercise for you to create your own classifier system using this model. You can use hints as provided for the spam filter classifier. We now move our discussion to tuning our classifier. Let's take a brief overview of the best practices in this area.

Classifiers tuning

We already discussed classifiers' evaluation techniques in Chapter 1, *Classification in Data Analysis*. Just as a reminder, we evaluate our model using techniques such as confusion matrix, entropy matrix, area under curve, and so on.

From the explanatory variables, we create the feature vector. To check how a particular model is working, these feature vectors need to be investigated. In Mahout, there is a class available for this, `ModelDissector`. It takes the following three inputs:

- **Features**: This class takes a feature vector to use (destructively)
- **TraceDictionary**: This class takes a trace dictionary containing variables and the locations in the feature vector that are affected by them
- **Learner**: This class takes the model that we are probing to find weights on features

`ModelDissector` tweaks the feature vector and observes how the model output changes. By taking an average of the number of examples, we can determine the effect of different explanatory variables.

`ModelDissector` has a summary method, which returns the most important features with their weights, most important category, and the top few categories that they affect.

The output of `ModelDissector` is helpful in troubleshooting problems in a wrongly created model.

More details for the code can be found at https://github.com/apache/mahout/blob/master/mrlegacy/src/main/java/org/apache/mahout/cla

While improving the output of the classifier, one should take care with two commonly occurring problems: target leak, and broken feature extraction.

If the model is showing results that are too good to be true or an output beyond expectations, we could have a problem with target leak. This error comes once information from the target variable is included in the explanatory variables, which are used to train the classifier. In this instance, the classifier will work too well for the `test` dataset.

On the other hand, broken feature extraction occurs when feature extraction is broken. This type of classifier shows the opposite result from the target leak classifiers. Here, the model provides results poorer than expected.

To tune the classifier, we can use new explanatory variables, transformations of explanatory variables, and can also eliminate some of the variables. We should also try different learning algorithms to create the model and choose an algorithm, which is good in performance, training time, and speed.

More details on tuning can be found in Chapter 16, *Deploying a classifier* in the book *Mahout in Action*.

Summary

In this chapter, we discussed creating our own production ready classifier model. We took up two use cases here, one for an e-mail spam filter and the other for classifying the e-mail as per the projects. We used datasets for Apache SpamAssassin for the e-mail filter and ASF for the e-mail classifier.

We also saw how to increase the performance of your model.

So you are now ready to implement classifiers using Apache Mahout for your own real world use cases. Happy learning!

Index

A

- algorithms, classification
  - Logistic regression / Classification algorithms
  - Stochastic Gradient Descent (SGD) / Classification algorithms
  - Naïve Bayes classification / Classification algorithms
  - Hidden Markov Model (HMM) / Classification algorithms
  - random forest / Classification algorithms
  - Multi-layer perceptron (MLP) / Classification algorithms
- Apache SpamAssassin project / Spam e-mail dataset
- Apache Spark
  - about / Apache Spark
  - Spark SQL / Apache Spark
  - MLib / Apache Spark
  - GraphX / Apache Spark
  - Spark streaming / Apache Spark
- ASF e-mail dataset
  - about / The ASF e-mail dataset
  - URL / The ASF e-mail dataset
- Assassin dataset
  - used, for creating model / Creating the model using the Assassin dataset
- AUC (area under the ROC curve) / Area under the ROC curve
- axons / Neural network and neurons

B

- back propagation / Multilayer Perceptron
- Bag of words / Understanding the terms used in text classification
- BaumWelchTrainer class / Using Mahout for the Hidden Markov Model
- Bayes rule
  - about / Introducing conditional probability and the Bayes rule
- binding stack

    - URL / Mahout Scala and Spark bindings

C

- Chi-squared Automatic Interaction Detector (CHAID) / Decision tree
- classification
    - about / Introducing the classification, Introducing Apache Mahout
    - application / Application of the classification system
    - system, working / Working of the classification system
    - algorithms / Classification algorithms
- Classification and Regression Tree (CART) / Decision tree
- classifier
    - training dataset / Working of the classification system
    - test dataset / Working of the classification system
    - model / Working of the classification system
    - building / Working of the classification system
- classifier model
    - using, program for / Program to use a classifier model
- classifiers
    - tuning / Classifiers tuning
- clustering
    - about / Introducing Apache Mahout
- conditional probability
    - about / Introducing conditional probability and the Bayes rule
- confusion matrix
    - about / The confusion matrix
    - Accuracy / The confusion matrix
    - Precision or positive predictive value / The confusion matrix
    - Negative predictive value / The confusion matrix
    - Sensitivity / true positive rate / recall / The confusion matrix
    - Specificity / The confusion matrix
    - F1 score / The confusion matrix
- cost function, linear regression

    - about / Cost function

D

- DARPA'98 / Using Mahout for Random forest
- data analysis
    - classification / Introducing the classification
- decision tree
    - about / Decision tree
- dendrites / Neural network and neurons
- dependent variable / Logistic regression
- deterministic patterns / Deterministic and nondeterministic patterns
- development environment
    - setting up, Eclipse used / Setting up a development environment using Eclipse
- dimensional reduction
    - about / Introducing Apache Mahout
- Domain Specific Language (DSL) / Mahout new changes

E

- Eclipse
    - used, for building development environment / Setting up a development environment using Eclipse
- emission matrix, HMM / Introducing the Hidden Markov Model
- Entropy matrix
    - about / The entropy matrix
- explanatory variable / Logistic regression
- explanatory variables
    - about / Working of the classification system

G

- gradient descent

    - about / Gradient descent
    - sigmoid function / Logistic regression
    - logistic function / Logistic regression

- GraphX / Apache Spark

H

- H2O platform

    - integration / H2O platform integration
    - URL / H2O platform integration

- Hadoop

    - URL / Introducing Apache Mahout, Installing Mahout

- hidden layer, MLP network / Multilayer Perceptron
- hidden layers, MLP network / Multilayer Perceptron
- Hidden Markov Model (HMM) / Classification algorithms
- hidden states, HMM / Introducing the Hidden Markov Model
- HMM

    - about / Introducing the Hidden Markov Model
    - properties / Introducing the Hidden Markov Model
    - state vector / Introducing the Hidden Markov Model
    - transition matrix / Introducing the Hidden Markov Model
    - emission matrix / Introducing the Hidden Markov Model
    - hidden states / Introducing the Hidden Markov Model
    - observable state / Introducing the Hidden Markov Model
    - Mahout used / Using Mahout for the Hidden Markov Model
    - Model class / Using Mahout for the Hidden Markov Model
    - HmmTrainer class / Using Mahout for the Hidden Markov Model
    - HmmEvaluator class / Using Mahout for the Hidden Markov Model
    - HmmAlgorithms class / Using Mahout for the Hidden Markov Model
    - HmmUtils class / Using Mahout for the Hidden Markov Model
    - RandomSequencerGenerator / Using Mahout for the Hidden Markov Model
    - BaumWelchTrainer class / Using Mahout for the Hidden Markov Model
    - ViterbiEvaluator class / Using Mahout for the Hidden Markov Model
    - input command / Using Mahout for the Hidden Markov Model
    - output command / Using Mahout for the Hidden Markov Model
    - model command / Using Mahout for the Hidden Markov Model

    - likelihood command / Using Mahout for the Hidden Markov Model

- HMM, issues

    - evaluation / Introducing the Hidden Markov Model
    - decoding / Introducing the Hidden Markov Model
    - learning / Introducing the Hidden Markov Model

- HmmAlgorithms class / Using Mahout for the Hidden Markov Model
- HmmEvaluator class / Using Mahout for the Hidden Markov Model
- HMMModel class / Using Mahout for the Hidden Markov Model
- HmmTrainer class / Using Mahout for the Hidden Markov Model
- HmmUtils class / Using Mahout for the Hidden Markov Model
- Hortonworks Sandbox

    - URL / Setting up Mahout for a Windows user

I

- iInitial state vector, Markov process / The Markov process
- independent variable / Logistic regression
- input layer, MLP network / Multilayer Perceptron
- iris dataset

    - URL / Using Mahout for MLP

- Iterative Dichotomiser 3 (ID3)

    - URL / Decision tree

J

- Java

    - URL / Installing Mahout

L

- labels

    - about / Working of the classification system

- Latent Dirichlet Allocation (LDA) / Introducing Apache Mahout
- linear regression

    - about / Understanding linear regression
    - cost function / Cost function
    - gradient descent / Gradient descent

- logistic function / Logistic regression
- logistic regression / Classification algorithms

    - about / Logistic regression
    - Mahout, using for / Using Mahout for logistic regression
    - dataset / Using Mahout for logistic regression
    - training and test data, preparing / Using Mahout for logistic regression
    - model, training / Using Mahout for logistic regression
    - trainlogistic / Using Mahout for logistic regression
    - input / Using Mahout for logistic regression
    - output / Using Mahout for logistic regression
    - target / Using Mahout for logistic regression
    - categories / Using Mahout for logistic regression
    - predictors / Using Mahout for logistic regression
    - types / Using Mahout for logistic regression
    - features / Using Mahout for logistic regression
    - passes / Using Mahout for logistic regression
    - rate / Using Mahout for logistic regression
    - runlogistic / Using Mahout for logistic regression
    - model / Using Mahout for logistic regression
    - auc / Using Mahout for logistic regression
    - confusion / Using Mahout for logistic regression

M

- M2Eclipse

    - URL / Installing Maven

- Mahout

    - about / Introducing Apache Mahout
    - use cases / Introducing Apache Mahout
    - features / Reasons for Mahout being a good choice for classification
    - installing / Installing Mahout
    - prerequisites / Installing Mahout
    - building from source, Maven used / Building Mahout from source using Maven
    - Maven, installing / Installing Maven
    - code, building / Building Mahout code

    - distribution file, URL / Building Mahout code, Setting up a development environment using Eclipse
    - setting up, for Windows user / Setting up Mahout for a Windows user
    - used, for logistic regression / Using Mahout for logistic regression
    - Naïve Bayes algorithm / Using the Naïve Bayes algorithm in Apache Mahout
    - using, for HMM / Using Mahout for the Hidden Markov Model
    - using, for Random forest algorithm / Using Mahout for Random forest
    - Random forest algorithm, implementing / Steps to use the Random forest algorithm in Mahout
    - MLP, implementing / MLP implementation in Mahout
    - using, for MLP / Using Mahout for MLP
    - MLP algorithm, using / Steps to use the MLP algorithm in Mahout
    - updations / Mahout new changes
    - Scala bindings / Mahout Scala and Spark bindings
    - Spark bindings / Mahout Scala and Spark bindings
    - Spark shell, using / Using Mahout's Spark shell
    - H2O platform, integration / H2O platform integration

- Mahout, algorithms

    - about / Algorithms supported in Mahout
    - sequential algorithms / Algorithms supported in Mahout
    - parallel algorithms / Algorithms supported in Mahout

- Mahout, use cases

    - recommendation / Introducing Apache Mahout
    - classification / Introducing Apache Mahout
    - clustering / Introducing Apache Mahout
    - dimensional reduction / Introducing Apache Mahout
    - topic modeling / Introducing Apache Mahout

- Mahout Scala bindings

    - about / Mahout Scala and Spark bindings

- Mahout Spark bindings

    - about / Mahout Scala and Spark bindings

- Markov process

    - about / The Markov process
    - states / The Markov process
    - transition matrix / The Markov process

    - Transition matrix / The Markov process
    - Initial state vector / The Markov process

- Maven
    - used, for building Mahout from source / Building Mahout from source using Maven
    - installing / Installing Maven
    - URL / Installing Maven

- MLib / Apache Spark
- MLP
    - implementing, in Mahout / MLP implementation in Mahout
    - Mahout used / Using Mahout for MLP
    - iris dataset / Using Mahout for MLP

- MLP algorithm
    - using, in Mahout / Steps to use the MLP algorithm in Mahout

- MLP network
    - about / Multilayer Perceptron
    - hidden layers / Multilayer Perceptron
    - back propagation / Multilayer Perceptron
    - zero hidden layers / Multilayer Perceptron
    - input layer / Multilayer Perceptron
    - output layer / Multilayer Perceptron
    - hidden layer / Multilayer Perceptron
    - number of neurons or hidden units / Multilayer Perceptron

- model
    - creating, Assassin dataset used / Creating the model using the Assassin dataset
    - classifier model, program for using / Program to use a classifier model

- model, evaluation
    - confusion matrix / The confusion matrix
    - Receiver Operating Characteristics (ROC) graph / The Receiver Operating Characteristics (ROC) graph
    - area under the ROC curve (AUC) / Area under the ROC curve
    - Entropy matrix / The entropy matrix

- model, issues
    - overfitting / Working of the classification system

    - underfitting / Working of the classification system

- ModelDissector
    - Features class / Classifiers tuning
    - TraceDictionary class / Classifiers tuning
    - Learner class / Classifiers tuning
    - about / Classifiers tuning

- Multi-layer perceptron (MLP) / Classification algorithms

N

- Naïve Bayes algorithm
    - about / Understanding the Naïve Bayes algorithm
    - in Apache Mahout / Using the Naïve Bayes algorithm in Apache Mahout

- Naïve Bayes classification / Classification algorithms
- neural network
    - about / Neural network and neurons

- neurons
    - about / Neural network and neurons
    - URL / Neural network and neurons

- nondeterministic patterns / Deterministic and nondeterministic patterns
- NSL-KDD dataset
    - URL / Using Mahout for Random forest

O

- observable state, HMM / Introducing the Hidden Markov Model
- outlier detection
    - about / Working of the classification system

- output layer, MLP network / Multilayer Perceptron
- overfitting, model
    - issues / Working of the classification system

P

- parallel algorithms / Algorithms supported in Mahout
- program
    - testing / Testing the program
- pruning / Decision tree

R

- random forest / Classification algorithms
- Random forest algorithm
    - about / Random forest
    - Bias parameter / Random forest
    - Variance parameter / Random forest
    - Mahout used / Using Mahout for Random forest
    - NSL-KDD dataset / Using Mahout for Random forest
    - dataset / Using Mahout for Random forest
    - implementing, in Mahout / Steps to use the Random forest algorithm in Mahout
- RandomSequencerGenerator / Using Mahout for the Hidden Markov Model
- Receiver Operating Characteristics (ROC) graph
    - about / The Receiver Operating Characteristics (ROC) graph
- regression
    - about / Introducing regression
    - linear regression / Understanding linear regression
- regression intercept / Logistic regression

S

- sequential algorithms / Algorithms supported in Mahout
- sigmoid function / Logistic regression
- softmax function
    - URL / Neural network and neurons
- spam e-mail dataset classifier
    - about / Spam e-mail dataset
- Spark

    - URL / Using Mahout's Spark shell
    - binding, URL / Using Mahout's Spark shell

- Spark-item / Apache Spark
- Spark-row / Apache Spark
- Spark shell

    - using / Using Mahout's Spark shell

- Spark SQL / Apache Spark
- Spark streaming / Apache Spark
- states, Markov process / The Markov process
- state vector, HMM / Introducing the Hidden Markov Model
- Stochastic Gradient Descent (SGD) / Classification algorithms

    - about / Stochastic Gradient Descent

T

- target variables

    - about / Working of the classification system

- Term frequency / Understanding the terms used in text classification
- term frequency

    - Stemming of words / Understanding the terms used in text classification
    - Case normalization / Understanding the terms used in text classification
    - Stop word removal / Understanding the terms used in text classification
    - Inverse document frequency / Understanding the terms used in text classification
    - Term frequency and inverse term frequency / Understanding the terms used in text classification

- text classification

    - about / Understanding the terms used in text classification

- topic modeling

    - about / Introducing Apache Mahout

- transition matrix, HMM / Introducing the Hidden Markov Model
- transition matrix, Markov process / The Markov process

U

- underfitting, model
  - issues / Working of the classification system

V

- vectors
  - about / Understanding the terms used in text classification
- ViterbiEvaluator class / Using Mahout for the Hidden Markov Model

W

- Windows
  - user, Mahout setting up for / Setting up Mahout for a Windows user
- Wisconsin Diagnostic Breast Cancer (WDBC) dataset
  - URL / Using Mahout for logistic regression