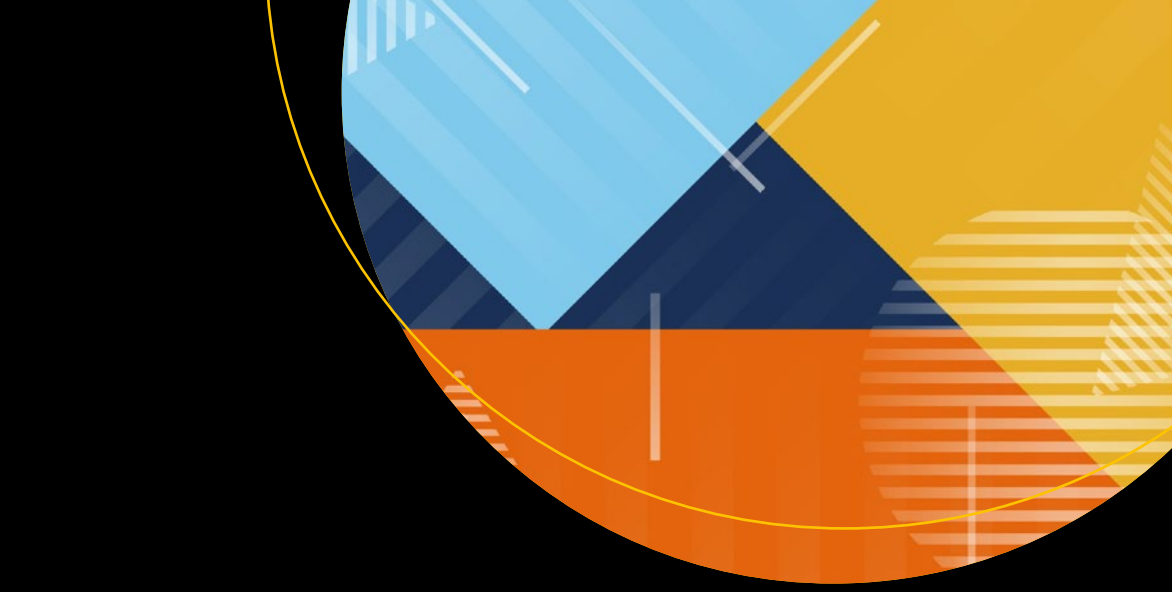


Advanced Data Analytics Using Python

With Machine Learning, Deep Learning and NLP Examples

Sayan Mukhopadhyay

Apress®

Advanced Data Analytics Using Python

With Machine Learning, Deep Learning and NLP Examples

Sayan Mukhopadhyay

Apress®

Advanced Data Analytics Using Python

Sayan Mukhopadhyay
Kolkata, West Bengal, India

ISBN-13 (pbk): 978-1-4842-3449-5 ISBN-13 (electronic): 978-1-4842-3450-1
https://doi.org/10.1007/978-1-4842-3450-1

Library of Congress Control Number: 2018937906

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin
Development Editor: Matthew Moodie
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-3449-5. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

This is dedicated to all my math teachers, especially to Kalyan Chakraborty.

Table of Contents

About the Author

Sayan Mukhopadhyay has more than 13 years of industry experience and has been associated with companies such as Credit Suisse, PayPal, CA Technologies, CSC, and Mphasis. He has a deep understanding of applications for data analysis in domains such as investment banking, online payments, online advertisement, IT infrastructure, and retail. His area of expertise is in applying high-performance computing in distributed and data-driven environments such as real-time analysis, high-frequency trading, and so on.

He earned his engineering degree in electronics and instrumentation from Jadavpur University and his master's degree in research in computational and data science from IISc in Bangalore.

About the Technical Reviewer

Sundar Rajan Raman has more than 14 years of full stack IT experience in machine learning, deep learning, and natural language processing. He has six years of big data development and architect experience, including working with Hadoop and its ecosystems as well as other NoSQL technologies such as MongoDB and Cassandra. In fact, he has been the technical reviewer of several books on these topics.

He is also interested in strategizing using Design Thinking principles and in coaching and mentoring people.

Acknowledgments

Thanks to Labonic Chakraborty (Ripa) and Kusumika Mukherjee.

CHAPTER 1

Introduction

In this book, I assume that you are familiar with Python programming. In this introductory chapter, I explain why a data scientist should choose Python as a programming language. Then I highlight some situations where Python is not a good choice. Finally, I describe some good practices in application development and give some coding examples that a data scientist needs in their day-to-day job.

Why Python?

So, why should you choose Python?

- It has versatile libraries. You always have a ready-made library in Python for any kind of application. From statistical programming to deep learning to network application to web crawling to embedded systems, you will always have a ready-made library in Python. If you learn this language, you do not have to stick to a specific use case. R has a rich set of analytics libraries, but if you are working on an Internet of Things (IoT) application and need to code in a device-side embedded system, it will be difficult in R.

S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1_1

- It is very high performance. Java is also a versatile language and has lots of libraries, but Java code runs on a Java virtual machine, which adds an extra layer of latency. Python uses high-performance libraries built in other languages. For example, SciPy uses LAPACK, which is a Fortran library for linear algebra applications. TensorFlow uses CUDA, which is a C library for parallel GPU processing.
- It is simple and gives you a lot of freedom to code. Python syntax is just like a natural language. It is easy to remember, and it does not have constraints in variables (like constants or `public`/`private`).

When to Avoid Using Python

Python has some downsides too.

- When you are writing very specific code, Python may not always be the best choice. For example, if you are writing code that deals only with statistics, R is a better choice. If you are writing MapReduce code only, Java is a better choice than Python.
- Python gives you a lot of freedom in coding. So, when many developers are working on a large application, Java/C++ is a better choice so that one developer/architect can put constraints on another developer's code using `public`/`private` and constant keywords.
- For extremely high-performance applications, there is no alternative to C/C++.

OOP in Python

Before proceeding, I will explain some features of object-oriented programming (OOP) in a Python context.

The most basic element of any modern application is an object. To a programmer or architect, the world is a collection of objects. Objects consist of two types of members: attributes and methods. Members can be private, public, or protected. Classes are data types of objects. Every object is an instance of a class. A class can be inherited in child classes. Two classes can be associated using composition.

In a Python context, Python has no keywords for public, private, or protected, so encapsulation (hiding a member from the outside world) is not implicit in Python. Like C++, it supports multilevel and multiple inheritance. Like Java, it has an `abstract` keyword. Classes and methods both can be abstract.

The following code is an example of a generic web crawler that is implemented as an airline's web crawler on the Skytrax site and as a retail crawler for the Mouthshut.com site. I'll return to the topic of web crawling in Chapter 2.

```
from abc import ABCMeta, abstractmethod
import BeautifulSoup
import urllib
import sys
import bleach
##################### Root Class (Abstract) ####################
class SkyThoughtCollector(object):
       __metaclass__ = ABCMeta

       baseURLString = "base_url"
       airlinesString = "air_lines"
       limitString = "limits"
```

```
        baseURl = ""
        airlines = []
        limit = 10

        @abstractmethod
        def collectThoughts(self):
              print "Something Wrong!! You're calling
              an abstract method"

        @classmethod
        def getConfig(self, configpath):
              #print "In get Config"
              config = {}
              conf = open(configpath)
              for line in conf:
                     if ("#" not in line):
                           words = line.strip().split('=')
                           config[words[0].strip()] = words[1].
                           strip()
              #print config
              self.baseURl = config[self.baseURLString]
              if config.has_key(self.airlinesString):
                     self.airlines = config[self.
                     airlinesString].split(',')
              if config.has_key(self.limitString):
                     self.limit = int(config[self.limitString])
              #print self.airlines

        def downloadURL(self, url):
              #print "downloading url"
              pageFile = urllib.urlopen(url)
```

```
        if pageFile.getcode() != 200:
                return "Problem in URL"
        pageHtml = pageFile.read()
        pageFile.close()
        return "".join(pageHtml)

def remove_junk(self, arg):
        f = open('junk.txt')
        for line in f:
                arg.replace(line.strip(),'')
        return arg

def print_args(self, args):
        out =''
        last = 0
        for arg in args:
                if args.index(arg) == len(args) -1:
                        last = 1
                reload(sys)
                sys.setdefaultencoding("utf-8")
                arg = arg.decode('utf8','ignore').
                encode('ascii','ignore').strip()
                arg = arg.replace('\n',' ')
                arg = arg.replace('\r','')
                arg = self.remove_junk(arg)
                if last == 0:
                        out = out + arg + '\t'
                else:
                        out = out + arg
        print out
```

```
####################### Airlines Chield ######################

class AirLineReviewCollector(SkyThoughtCollector):

      months = ['January', 'February', 'March', 'April', 'May',
'June', 'July', 'August', 'September', 'October', 'November',
'December' ]

       def __init__(self, configpath):
             #print "In Config"
       super(AirLineReviewCollector,self).getConfig(configpath)

       def parseSoupHeader(self, header):
             #print "parsing header"
             name = surname = year = month = date = country =''
             txt = header.find("h9")
             words = str(txt).strip().split(' ')
             for j in range(len(words)-1):
                    if words[j] in self.months:
                          date = words[j-1]
                          month= words[j]
                          year = words[j+1]
                          name = words[j+3]
                          surname = words[j+4]
             if ")" in words[-1]:
                    country = words[-1].split(')')[0]
             if "(" in country:
                    country = country.split('(')[1]
             else:
                    country = words[-2].split('(')[1] + country
             return (name, surname, year, month, date, country)
```

```
def parseSoupTable(self, table):
      #print "parsing table"
      images = table.findAll("img")
      over_all = str(images[0]).split("grn_bar_")[1].
      split(".gif")[0]
      money_value = str(images[1]).split("SCORE_")[1].
      split(".gif")[0]
      seat_comfort = str(images[2]).split("SCORE_")[1].
      split(".gif")[0]
      staff_service = str(images[3]).split("SCORE_")[1].
      split(".gif")[0]
      catering = str(images[4]).split("SCORE_")[1].
      split(".gif")[0]
      entertainment = str(images[4]).split("SCORE_")[1].
      split(".gif")[0]
      if 'YES' in str(images[6]):
            recommend = 'YES'
      else:
            recommend = 'NO'
      status = table.findAll("p", {"class":"text25"})
      stat = str(status[2]).split(">")[1].split("<")[0]
      return (stat, over_all, money_value, seat_comfort,
      staff_service, catering, entertainment, recomend)

def collectThoughts(self):
      #print "Collecting Thoughts"
      for al in AirLineReviewCollector.airlines:
            count = 0
            while count < AirLineReviewCollector.limit:
                  count = count + 1
                  url = ''
```

```
if count == 1:
        url = AirLineReviewCollector.
        baseURl + al + ".htm"
else:
        url = AirLineReviewCollector.
        baseURl + al + "_"+str(count)+
        ".htm"
soup = BeautifulSoup.BeautifulSoup
(super(AirLineReviewCollector,self).
downloadURL(url))
blogs = soup.findAll("p",
{"class":"text2"})
tables = soup.findAll("table",
{"width":"192"})
review_headers = soup.findAll("td",
{"class":"airport"})
for i in range(len(tables)-1):
        (name, surname, year, month,
        date, country) = self.parse
        SoupHeader(review_headers[i])
        (stat, over_all, money_value,
        seat_comfort, staff_service,
        catering, entertainment,
        recomend) = self.parseSoup
        Table(tables[i])
        blog = str(blogs[i]).
        split(">")[1].split("<")[0]
        args = [al, name, surname,
        year, month, date, country,
        stat, over_all, money_value,
        seat_comfort, staff_service,
        catering, entertainment,
        recomend, blog]
```

```
                                        super(AirLineReviewCo
                                        llector,self).print_
                                        args(args)

######################### Retail Chield ########################

class RetailReviewCollector(SkyThoughtCollector):
       def __init__(self, configpath):
             #print "In Config"
       super(RetailReviewCollector,self).getConfig(configpath)

       def collectThoughts(self):
             soup = BeautifulSoup.BeautifulSoup(super(RetailRev
             iewCollector,self).downloadURL(RetailReviewCollect
             or.baseURl))
             lines = soup.findAll("a",{"style":
             "font-size:15px;"})
             links = []
             for line in lines:
                    if ("review" in str(line)) & ("target" in
                    str(line)):
                          ln = str(line)
                          link = ln.split("href=")[-1].split
                          ("target=")[0].replace("\"","").
                          strip()
                          links.append(link)

             for link in links:
                    soup = BeautifulSoup.BeautifulSoup(
                    super(RetailReviewCollector,self).
                    downloadURL(link))
```

```
comment = bleach.clean(str(soup.findAll("di
v",{"itemprop":"description"})[0]),tags=[],
strip=True)
tables = soup.findAll("table",
{"class":"smallfont space0 pad2"})
parking = ambience = range = economy =
product = 0
for table in tables:
      if "Parking:" in str(table):
            rows = table.findAll("tbody")
            [0].findAll("tr")
            for row in rows:
                  if "Parking:" in
                  str(row):
                        parking =
                        str(row).
                        count("read-
                        barfull")
                  if "Ambience" in
                  str(row):
                        ambience =
                        str(row).
                        count("read-
                        barfull")
                  if "Store" in str(row):
                        range = str(row).
                        count("read-
                        barfull")
```

```
                                        if "Value" in str(row):
                                                economy =
                                                str(row).
                                                count("read-
                                                barfull")
                                        if "Product" in str(row):
                                                product =
                                                str(row).count
                                                ("smallratefull")

                        author = bleach.clean(soup.findAll("spa
                        n",{"itemprop":"author"})[0], tags=[],
                        strip=True)
                        date = soup.findAll("meta",{"itemprop":"dat
                        ePublished"})[0]["content"]
                        args = [date, author,str(parking),
                        str(ambience),str(range), str(economy),
                        str(product), comment]
                                        super(RetailReview
                                        Collector,self).print_
                                        args(args)

######################### Main Function ##########################

if __name__ == "__main__":
        if sys.argv[1] == 'airline':
                instance = AirLineReviewCollector(sys.argv[2])
                instance.collectThoughts()
        else:
                if sys.argv[1] == 'retail':
                        instance = RetailReviewCollector(sys.argv[2])
                        instance.collectThoughts()
```

```
            else:
                  print "Usage is"
                  print sys.argv[0], '<airline/retail>',
                  "<Config File Path>"
```

The configuration for the previous code is shown here:

```
base_url = http://www.airlinequality.com/Forum/
#base_url = http://www.mouthshut.com/product-reviews/Mega-Mart-
Bangalore-reviews-925103466
#base_url = http://www.mouthshut.com/product-reviews/Megamart-
Chennai-reviews-925104102
air_lines = emrts,brit_awys,ual,biman,flydubai
limits = 10
```

I'll now discuss the previous code in brief. It has a root class that is an abstract class. It contains essential attributes such as a base URL and a page limit; these are essential for all child classes. It also contains common logic in class method functions such as the download URL, print output, and read configuration. It also has an abstract method collectThoughts, which must be implemented in child classes. This abstract method is passing on a common behavior to every child class that all of them must collect thoughts from the Web. Implementations of this thought collection are child specific.

Calling Other Languages in Python

Now I will describe how to use other languages' code in Python. There are two examples here; one is calling R code from Python. R code is required for some use cases. For example, if you want a ready-made function for the Holt-Winter method in a time series, it is difficult to do in Python. But it is

available in R. So, you can call R code from Python using the `rpy2` module, as shown here:

```
import rpy2.robjects as ro
ro.r('data(input)')
ro.r('x <-HoltWinters(input)')
```

Sometimes you need to call Java code from Python. For example, say you are working on a name entity recognition problem in the field of natural language processing (NLP); some text is given as input, and you have to recognize the names in the text. Python's NLTK package does have a name entity recognition function, but its accuracy is not good. Stanford NLP is a better choice here, which is written in Java. You can solve this problem in two ways.

- You can call Java at the command line using Python code.

  ```
  import subprocess

  subprocess.call(['java','-cp','*','edu.
  stanford.nlp.sentiment.SentimentPipeline',
  '-file','foo.txt'])
  ```

- You can expose Stanford NLP as a web service and call it as a service.

  ```
  nlp = StanfordCoreNLP('http://127.0.0.1:9000')
  output = nlp.annotate(sentence, properties={
   "annotators": "tokenize,ssplit,parse,sentiment",
   "outputFormat": "json",
   # Only split the sentence at End Of Line.
  We assume that this method only takes in one
  single sentence.
   "ssplit.eolonly": "true",
  ```

```
 # Setting enforceRequirements to skip some
annotators and make the process faster
 "enforceRequirements": "false"
 })
```

Exposing the Python Model as a Microservice

You can expose the Python model as a microservice in the same way as your Python model can be used by others to write their own code. The best way to do this is to expose your model as a web service. As an example, the following code exposes a deep learning model using Flask:

```
from flask import Flask, request, g
from flask_cors import CORS
import tensorflow as tf
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker
import pygeoip
from pymongo import MongoClient
import json
import datetime as dt
import ipaddress
import math

app = Flask(__name__)
CORS(app)

@app.before_request
def before():
       db = create_engine('sqlite:///score.db')
       metadata = MetaData(db)
```

```
        g.scores = Table('scores', metadata, autoload=True)
        Session = sessionmaker(bind=db)
        g.session = Session()

        client = MongoClient()
        g.db = client.frequency

        g.gi = pygeoip.GeoIP('GeoIP.dat')

        sess = tf.Session()
        new_saver = tf.train.import_meta_graph('model.obj.meta')
        new_saver.restore(sess, tf.train.latest_checkpoint('./'))
        all_vars = tf.get_collection('vars')

        g.dropped_features = str(sess.run(all_vars[0]))
        g.b = sess.run(all_vars[1])[0]
        return

def get_hour(timestamp):
        return dt.datetime.utcfromtimestamp(timestamp / 1e3).hour

def get_value(session, scores, feature_name, feature_value):
        s = scores.select((scores.c.feature_name == feature_
        name) & (scores.c.feature_value == feature_value))
        rs = s.execute()
        row = rs.fetchone()
        if row is not None:
              return float(row['score'])
        else:
              return 0.0
```

```
@app.route('/predict', methods=['POST'])
def predict():
        input_json = request.get_json(force=True)

        features = ['size','domain','client_time','device',
        'ad_position','client_size', 'ip','root']
        predicted = 0
        feature_value = ''
        for f in features:
              if f not in g.dropped_features:
                    if f == 'ip':
                          feature_value = str(ipaddress.
                          IPv4Address(ipaddress.ip_address
                          (unicode(request.remote_addr))))
                    else:
                          feature_value = input_json.get(f)
                    if f == 'ip':
                          if 'geo' not in g.dropped_features:
                                geo = g.gi.country_name_by_
                                addr(feature_value)
                                predicted = predicted + get_
                                value(g.session, g.scores,
                                'geo', geo)
                          if 'frequency' not in g.dropped_
                          features:
                                res = g.db.frequency.find_
                                one({"ip" : feature_value})
                                freq = 1
                                if res is not None:
                                      freq = res['frequency']
                                predicted = predicted + get_
                                value(g.session, g.scores,
                                'frequency', str(freq))
```

```
                    if f == 'client_time':
                           feature_value = get_
                           hour(int(feature_value))
                    predicted = predicted + get_value(g.
                    session, g.scores, f, feature_value)
        return str(math.exp(predicted + g.b)-1)
app.run(debug = True, host ='0.0.0.0')
```

This code exposes a deep learning model as a Flask web service. A JavaScript client will send the request with web user parameters such as the IP address, ad size, ad position, and so on, and it will return the price of the ad as a response. The features are categorical. You will learn how to convert them into numerical scores in Chapter 3. These scores are stored in an in-memory database. The service fetches the score from the database, sums the result, and replies to the client. This score will be updated real time in each iteration of training of a deep learning model. It is using MongoDB to store the frequency of that IP address in that site. It is an important parameter because a user coming to a site for the first time is really searching for something, which is not true for a user where the frequency is greater than 5. The number of IP addresses is huge, so they are stored in a distributed MongoDB database.

High-Performance API and Concurrent Programming

Flask is a good choice when you are building a general solution that is also a graphical user interface (GUI). But if high performance is the most critical requirement of your application, then Falcon is the best choice. The following code is an example of the same model shown previously exposed by the Falcon framework. Another improvement I made in this code is that I implemented multithreading, so the code will be executed in parallel.

Except Falcon-specific changes, you should note the major changes in parallelizing the calling get_score function using a thread pool class.

```
import falcon
from falcon_cors import CORS
import json
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker
import pygeoip
from pymongo import MongoClient
import json
import datetime as dt
import ipaddress
import math
from concurrent.futures import *
from sqlalchemy.engine import Engine
from sqlalchemy import event
import sqlite3

@event.listens_for(Engine, "connect")
def set_sqlite_pragma(dbapi_connection, connection_record):
 cursor = dbapi_connection.cursor()
 cursor.execute("PRAGMA cache_size=100000")
 cursor.close()

class Predictor(object):
       def __init__(self,domain):
             db1 = create_engine('sqlite:///score_' + domain +
             'Otest.db')
             db2 = create_engine('sqlite:///probability_' +
             domain +'Otest.db')
             db3 = create_engine('sqlite:///ctr_'+ domain +
             'test.db')
```

```
        metadata1 = MetaData(db1)
        metadata2 = MetaData(db2)
        metadata3 = MetaData(db3)
        self.scores = Table('scores', metadata1, autoload=True)
        self.probabilities = Table('probabilities', metadata2,
        autoload=True)
        self.ctr = Table('ctr', metadata3, autoload=True)

      client = MongoClient(connect=False,maxPoolSize=1)
      self.db = client.frequency

      self.gi = pygeoip.GeoIP('GeoIP.dat')

            self.high = 1.2
            self.low = .8

      def get_hour(self,timestamp):
      return dt.datetime.utcfromtimestamp(timestamp / 1e3).hour

      def get_score(self, featurename, featurevalue):
            prob = 0
            pred = 0
            s = self.scores.select((self.scores.c.feature_name
            == featurename) & (self.scores.c.feature_value ==
            featurevalue))
 rs = s.execute()
 row = rs.fetchone()
 if row is not None:
      pred = pred + float(row['score'])
 s = self.probabilities.select((self.probabilities.c.feature_
name == featurename) & (self.probabilities.c.feature_value ==
featurevalue))
 rs = s.execute()
 row = rs.fetchone()
```

```
if row is not None:
     prob = prob + float(row['Probability'])
            return pred, prob

     def get_value(self, f, value):
            if f == 'ip':
                   ip = str(ipaddress.IPv4Address(ipaddress.
                   ip_address(value)))
geo = self.gi.country_name_by_addr(ip)
                   pred1, prob1 = self.get_score('geo', geo)
      res = self.db.frequency.find_one({"ip" : ip})
      freq = 1
if res is not None:
      freq = res['frequency']
                   pred2, prob2 = self.get_score('frequency',
                   str(freq))
                   return (pred1 + pred2), (prob1 + prob2)
            if f == 'root':
                   s = self.ctr.select(self.ctr.c.root == value)
                   rs = s.execute()
      row = rs.fetchone()
                   if row is not None:
                         ctr = row['ctr']
                         avv = row['avt']
                         avt = row['avv']
                         (pred1,prob1) = self.get_score
                         ('ctr', ctr)
                         (pred2,prob2) = self.get_score
                         ('avt', avt)
                         (pred3,prob3) = self.get_score
                         ('avv', avv)
```

```
                    (pred4,prob4) = self.get_score(f,
                    value)
                    return (pred1 + pred2 + pred3 + pred4),
                    (prob1 + prob2 + prob3 + prob4)
if f == 'client_time':
      value = str(self.get_hour(int(value)))
      if f == 'domain':
            conn = sqlite3.connect('multiplier.db')
            cursor = conn.execute("SELECT high,low from
            multiplier where domain='" + value + "'")
            row = cursor.fetchone()
            if row is not None:
                  self.high = row[0]
                  self.low = row[1]
      return self.get_score(f, value)

def on_post(self, req, resp):
      input_json = json.loads(req.stream.
      read(),encoding='utf-8')
      input_json['ip'] = unicode(req.remote_addr)
pred = 1
      prob = 1

      with ThreadPoolExecutor(max_workers=8) as pool:
            future_array = { pool.submit(self.get_
            value,f,input_json[f]) : f for f in
            input_json}
            for future in as_completed(future_array):
pred1, prob1 = future.result()
                  pred = pred + pred1
                  prob = prob - prob1

      resp.status = falcon.HTTP_200
```

```
        res = math.exp(pred)-1
              if res < 0:
                    res = 0
              prob = math.exp(prob)
              if(prob <= .1):
                    prob = .1
              if(prob >= .9):
                    prob = .9
              multiplier = self.low + (self.high -self.low)*prob
              pred = multiplier*pred
              resp.body = str(pred)

cors = CORS(allow_all_origins=True,allow_all_
methods=True,allow_all_headers=True)
wsgi_app = api = falcon.API(middleware=[cors.middleware])

f = open('publishers1.list')
for domain in f:
        domain = domain.strip()
        p = Predictor(domain)
        url = '/predict/' + domain
        api.add_route(url, p)
```

CHAPTER 2

ETL with Python (Structured Data)

Every data science professional has to extract, transform, and load (ETL) data from different data sources. In this chapter, I will discuss how to do ETL with Python for a selection of popular databases. For a relational database, I'll cover MySQL. As an example of a document database, I will cover Elasticsearch. For a graph database, I'll cover Neo4j, and for NoSQL, I'll cover MongoDB. I will also discuss the Pandas framework, which was inspired by R's data frame concept.

MySQL

MySQLdb is an API in Python developed at the top of the MySQL C interface.

How to Install MySQLdb?

First you need to install the Python MySQLdb module on your machine. Then run the following script:

```
#!/usr/bin/python
import MySQLdb
```


S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1_2

If you get an import error exception, that means the module is not installed properly.

The following is the instruction to install the MySQL Python module:

```
$ gunzip MySQL-python-1.2.2.tar.gz
$ tar -xvf MySQL-python-1.2.2.tar
$ cd MySQL-python-1.2.2
$ python setup.py build
$ python setup.py install
```

Database Connection

Before connecting to a MySQL database, make sure you have the following:

- You need a database called `TEST`.
- In `TEST` you need a table `STUDENT`.
- `STUDENT` needs three fields: `NAME`, `SUR_NAME`, and `ROLL_NO`.
- There needs to be a user in `TEST` that has complete access to the database.

INSERT Operation

The following code carries out the SQL `INSERT` statement for the purpose of creating a record in the `STUDENT` table:

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","user","passwd","TEST" )

# prepare a cursor object using cursor() method
cursor = db.cursor()
```

```
# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO STUDENT(NAME,
         SUR_NAME, ROLL_NO)
         VALUES ('Sayan', 'Mukhopadhyay', 1)"""
try:
   # Execute the SQL command
cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

READ Operation

The following code fetches data from the STUDENT table and prints it:

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","user","passwd","TEST" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "SELECT * FROM STUDENT "
try:
```

```
   # Execute the SQL command
cursor.execute(sql)
   # Fetch all the rows in a list of lists.
results = cursor.fetchall()
for row in results:
fname = row[0]
lname = row[1]
id = row[2]
      # Now print fetched result
print "name=%s,surname=%s,id=%d" % \
             (fname, lname, id )
except:
print "Error: unable to fecth data"

# disconnect from server
db.close()
```

DELETE Operation

The following code deletes a row from TEST with id=1:

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","test","passwd","TEST" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to DELETE required records
sql = "DELETE FROM STUDENT WHERE ROLL_NO =1"
try:
```

```
   # Execute the SQL command
cursor.execute(sql)
   # Commit your changes in the database
db.commit()
except:
   # Rollback in case there is any error
db.rollback()

# disconnect from server
db.close()
```

UPDATE Operation

The following code changes the `lastname` variable to Mukherjee, from Mukhopadhyay:

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","user","passwd","TEST" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE STUDENT SET SUR_NAME="Mukherjee"
                          WHERE SUR_NAME="Mukhopadhyay"
try:
   # Execute the SQL command
cursor.execute(sql)
   # Commit your changes in the database
db.commit()
```

```
except:
   # Rollback in case there is any error
db.rollback()

# disconnect from server
db.close()
```

COMMIT Operation

The commit operation provides its assent to the database to finalize the modifications, and after this operation, there is no way that this can be reverted.

ROLL-BACK Operation

If you are not completely convinced about any of the modifications and you want to reverse them, then you need to apply the `roll-back()` method.

The following is a complete example of accessing MySQL data through Python. It will give the complete description of data stored in a CSV file or MySQL database.

```
import MySQLdb
import sys

out = open('Config1.txt','w')
print "Enter the Data Source Type:"
print "1. MySql"
print "2. Text"
print "3. Exit"

while(1):
       data1 = sys.stdin.readline().strip()
       if(int(data1) == 1):
```

```
out.write("source begin"+"\n"+"type=mysql\n")
print "Enter the ip:"
ip = sys.stdin.readline().strip()
out.write("host=" + ip + "\n")
print "Enter the database name:"
db = sys.stdin.readline().strip()
out.write("database=" + db + "\n")
print "Enter the user name:"
usr = sys.stdin.readline().strip()
out.write("user=" + usr + "\n")
print "Enter the password:"
passwd = sys.stdin.readline().strip()
out.write("password=" + passwd + "\n")
connection = MySQLdb.connect(ip, usr, passwd, db)
cursor = connection.cursor()
query = "show tables"
cursor.execute(query)
data = cursor.fetchall()
tables = []
for row in data:
       for field in row:
              tables.append(field.strip())
for i in range(len(tables)):
       print i, tables[i]
tb = tables[int(sys.stdin.readline().strip())]
out.write("table=" + tb + "\n")
query = "describe " + tb
cursor.execute(query)
data = cursor.fetchall()
columns = []
for row in data:
       columns.append(row[0].strip())
```

```
            for i in range(len(columns)):
                  print columns[i]
            print "Not index choose the exact column names
            seperated by coma"
            cols = sys.stdin.readline().strip()
            out.write("columns=" + cols + "\n")

            cursor.close()
            connection.close()
            out.write("source end"+"\n")

            print "Enter the Data Source Type:"
            print "1. MySql"
            print "2. Text"
            print "3. Exit"

      if(int(data1) == 2):
            print "path of text file:"
            path = sys.stdin.readline().strip()
            file = open(path)
            count = 0
            for line in file:
                  print line
                  count = count + 1
                  if count > 3:
                        break
            file.close()
            out.write("source begin"+"\n"+"type=text\n")
            out.write("path=" + path + "\n")
            print "enter delimeter:"
            dlm = sys.stdin.readline().strip()
            out.write("dlm=" + dlm + "\n")
            print "enter column indexes seperated by comma:"
```

```
        cols = sys.stdin.readline().strip()
        out.write("columns=" + cols + "\n")
        out.write("source end"+"\n")

        print "Enter the Data Source Type:"
        print "1. MySql"
        print "2. Text"
        print "3. Exit"

if(int(data1) == 3):
        out.close()
        sys.exit()
```

Elasticsearch

The Elasticsearch (ES) low-level client gives a direct mapping from Python to ES REST endpoints. One of the big advantages of Elasticsearch is that it provides a full stack solution for data analysis in one place. Elasticsearch is the database. It has a configurable front end called Kibana, a data collection tool called Logstash, and an enterprise security feature called Shield.

This example has the features called cat, cluster, indices, ingest, nodes, snapshot, and tasks that translate to instances of `CatClient`, `ClusterClient`, `IndicesClient`, `CatClient`, `ClusterClient`, `IndicesClient`, `IngestClient`, `NodesClient`, `SnapshotClient`, `NodesClient`, `SnapshotClient`, and `TasksClient`, respectively. These instances are the only supported way to get access to these classes and their methods.

You can specify your own connection class, which can be used by providing the `connection_class` parameter.

```
# create connection to local host using the ThriftConnection
Es1=Elasticsearch(connection_class=ThriftConnection)
```

If you want to turn on sniffing, then you have several options (described later in the chapter).

```
# create connection that will automatically inspect the cluster to get
# the list of active nodes. Start with nodes running on 'esnode1' and
# 'esnode2'
Es1=Elasticsearch(
    ['esnode1', 'esnode2'],
# sniff before doing anything
sniff_on_start=True,
# refresh nodes after a node fails to respond
sniff_on_connection_fail=True,
# and also every 30 seconds
sniffer_timeout=30
)
```

Different hosts can have different parameters; you can use one dictionary per node to specify them.

```
# connect to localhost directly and
another node using SSL on port 443
# and an url_prefix. Note that ``port`` needs to be an int.
Es1=Elasticsearch([
{'host':'localhost'},
{'host':'othernode','port':443,'url_prefix':'es','use_ssl':True},
])
```

SSL client authentication is also supported (see `Urllib3HttpConnection` for a detailed description of the options).

```
Es1=Elasticsearch(
['localhost:443','other_host:443'],
# turn on SSL
use_ssl=True,
```

```
# make sure we verify SSL certificates (off by default)
verify_certs=True,
# provide a path to CA certs on disk
ca_certs='path to CA_certs',
# PEM formatted SSL client certificate
client_cert='path to clientcert.pem',
# PEM formatted SSL client key
client_key='path to clientkey.pem'
)
```

Connection Layer API

Many classes are responsible for dealing with the Elasticsearch cluster. Here, the default subclasses being utilized can be disregarded by handing over parameters to the `Elasticsearch` class. Every argument belonging to the client will be added onto `Transport`, `ConnectionPool`, and `Connection`.

As an example, if you want to use your own personal utilization of the `ConnectionSelector` class, you just need to pass in the `selector_class` parameter.

The entire API wraps the raw REST API with a high level of accuracy, which includes the differentiation between the required and optional arguments to the calls. This implies that the code makes a differentiation between positional and keyword arguments; I advise you to use keyword arguments for all calls to be consistent and safe. An API call becomes successful (and will return a response) if Elasticsearch returns a 2XX response. Otherwise, an instance of `TransportError` (or a more specific subclass) will be raised. You can see other exceptions and error states in exceptions. If you do not want an exception to be raised, you can always pass in an `ignore` parameter with either a single status code that should be ignored or a list of them.

```
from elasticsearch import Elasticsearch
es=Elasticsearch()
# ignore 400 cause by IndexAlreadyExistsException when creating
an index
es.indices.create(index='test-index',ignore=400)

# ignore 404 and 400
es.indices.delete(index='test-index',ignore=[400,404])
```

Neo4j Python Driver

The Neo4j Python driver is supported by Neo4j and connects with the database through the binary protocol. It tries to remain minimalistic but at the same time be idiomatic to Python.

```
pip install neo4j-driver

from neo4j.v1 import GraphDatabase, basic_auth

driver11 = GraphDatabase.driver("bolt://localhost", auth=basic_
auth("neo4j", "neo4j"))
session11 = driver11.session()

session11.run("CREATE (a:Person {name:'Sayan',
title:'Mukhopadhyay'})")

result 11= session11.run("MATCH (a:Person) WHERE a.name =
'Sayan' RETURN a.name AS name, a.title AS title")
for recordi n resul11t:
print("%s %s"% (record["title"], record["name"]))
session.close()
```

neo4j-rest-client

The main objective of neo4j-rest-client is to make sure that the Python programmers already using Neo4j locally through python-embedded are also able to access the Neo4j REST server. So, the structure of the neo4j-rest-client API is completely in sync with python-embedded. But, a new structure is brought in so as to arrive at a more Pythonic style and to augment the API with the new features being introduced by the Neo4j team.

In-Memory Database

Another important class of database is an in-memory database. It stores and processes the data in RAM. So, operation on the database is very fast, and the data is volatile. SQLite is a popular example of in-memory database. In Python you need to use the sqlalchemy library to operate on SQLite. In Chapter 1's Flask and Falcon example, I showed you how to select data from SQLite. Here I will show how to store a Pandas data frame in SQLite:

```
from sqlalchemy import create_engine
import sqlite3
conn = sqlite3.connect('multiplier.db')
conn.execute('''CREATE TABLE if not exists multiplier
       (domain        CHAR(50),
        low        REAL,
        high        REAL);''')
conn.close()
db_name = "sqlite:///" + prop + "_" + domain + str(i) + ".db"
disk_engine = create_engine(db_name)
df.to_sql('scores', disk_engine, if_exists='replace')
```

MongoDB (Python Edition)

MongoDB is an open source *document database* designed for superior performance, easy availability, and automatic scaling. MongoDB makes sure that object-relational mapping (ORM) is not required to facilitate development. A document that contains a data structure made up of field and value pairs is referred to as a *record* in MongoDB. These records are akin to JSON objects. The values of fields may be comprised of other documents, arrays, and arrays of documents.

```
{
"_id":ObjectId("01"),
"address": {
"street":"Siraj Mondal Lane",
"pincode":"743145",
"building":"129",
"coord": [ -24.97, 48.68 ]
   },
"borough":"Manhattan",
```

Import Data into the Collection

`mongoimport` can be used to place the documents into a collection in a database, within the system shell or a command prompt. If the collection preexists in the database, the operation will discard the original collection first.

```
mongoimport --DB test --collection restaurants --drop --file ~/
downloads/primer-dataset.json
```

The `mongoimport` command is joined to a MongoDB instance running on localhost on port number 27017. The `--file` option provides a way to import the data; here it's `~/downloads/primer-dataset.json`.

To import data into a MongoDB instance running on a different host or port, the hostname or port needs to be mentioned specifically in the `mongoimport` command by including the `--host` or `--port` option.

There is a similar `load` command in MySQL.

Create a Connection Using pymongo

To create a connection, do the following:

```
import MongoClient from pymongo.
Client11 = MongoClient()
```

If no argument is mentioned to `MongoClient`, then it will default to the MongoDB instance running on the localhost interface on port 27017.

A complete MongoDB URL may be designated to define the connection, which includes the host and port number. For instance, the following code makes a connection to a MongoDB instance that runs on `mongodb0.example.net` and the port of 27017:

```
Client11 = MongoClient("mongodb://myhostname:27017")
```

Access Database Objects

To assign the database named `primer` to the local variable `DB`, you can use either of the following lines:

```
Db11 = client11.primer
db11 = client11['primer']
```

Collection objects can be accessed directly by using the dictionary style or the attribute access from a database object, as shown in the following two examples:

```
Coll11 = db11.dataset
coll = db11['dataset']
```

Insert Data

You can place a document into a collection that doesn't exist, and the following operation will create the collection:

```
result=db.addrss.insert_one({<<your json >>)
```

Update Data

Here is how to update data:

```
result=db.address.update_one(
 {"building": "129",
 {"$set": {"address.street": "MG Road"}}
)
```

Remove Data

To expunge all documents from a collection, use this:

```
result=db.restaurants.delete_many({})
```

Pandas

The goal of this section is to show some examples to enable you to begin using Pandas. These illustrations have been taken from real-world data, along with any bugs and weirdness that are inherent. Pandas is a framework inspired by the R data frame concept.

To read data from a CSV file, use this:

```
import pandas as pd
broken_df=pd.read_csv('data.csv')
```

To look at the first three rows, use this:

```
broken_df[:3]
```

To select a column, use this:

```
fixed_df['Column Header']
```

To plot a column, use this:

```
fixed_df['Column Header'].plot()
```

To get a maximum value in the data set, use this:

```
MaxValue=df['Births'].max() where Births is the column header
```

Let's assume there is another column in a data set named `Name`. The command for `Name` is associated with the maximum value.

```
MaxName=df['Names'][df['Births']==df['Births'].max()].values
```

There are many other methods such as `sort`, `groupby`, and `orderby` in Pandas that are useful to play with structured data. Also, Pandas has a ready-made adapter for popular databases such as MongoDB, Google Big Query, and so on.

One complex example with Pandas is shown next. In `X` data frame for each distinct column value, find the average value of floor grouping by the `root` column.

```
for col in X.columns:
                        if col != 'root':
                                avgs = df.groupby([col,'root'],
                                as_index=False)['floor'].
                                aggregate(np.mean)
                                for i,row in avgs.iterrows():
                                        k = row[col]
                                        v = row['floor']
```

```
r = row['root']
X.loc[(X[col] == k) &
(X['root'] == r), col]
= v2.
```

ETL with Python (Unstructured Data)

Dealing with unstructured data is an important task in modern data analysis. In this section, I will cover how to parse e-mails, and I'll introduce an advanced research topic called *topical crawling*.

E-mail Parsing

See Chapter 1 for a complete example of web crawling using Python.

Like BeautifulSoup, Python has a library for e-mail parsing. The following is the example code to parse e-mail data stored on a mail server. The inputs in the configuration are the username and number of mails to parse for the user.

```
from email.parser import Parser
import os
import sys

conf = open(sys.argv[1])
config={}
users={}
for line in conf:
       if ("," in line):
             fields = line.split(",")
             key = fields[0].strip().split("=")[1].strip()
             val = fields[1].strip().split("=")[1].strip()
             users[key] = val
```

```
        else:
                if ("=" in line):
                        words = line.strip().split('=')
                        config[words[0].strip()] = words[1].strip()
conf.close()

for usr in users.keys():
        path = config["path"]+"/"+usr+"/"+config["folder"]
        files = os.listdir(path)
        for f in sorted(files):
                if(int(f) > int(users[usr])):
                        users[usr] = f
                        path1 = path + "/" + f
                        data = ""
                        with open (path1) as myfile:
                                data=myfile.read()
                        if data != "" :
                                parser = Parser()
                        email = parser.parsestr(data)
                        out = ""
                        out = out + str(email.get('From')) + ","
                        + str(email.get('To')) + "," + str(email.
                        get('Subject')) + "," + str(email.
                        get('Date')).replace(","," ")

                        if email.is_multipart():
                                for part in email.get_payload():
                                        out = out + "," + str(part.
                                        get_payload()).replace("\n","
                                        ").replace(","," ")
                        else:
```

```
                        out = out + "," + str(email.get_payload
                        ()).replace("\n"," ").replace(","," ")
                print out,"\n"

conf = open(sys.argv[1],'w')
conf.write("path=" + config["path"] + "\n")
conf.write("folder=" + config["folder"] + "\n")

for usr in users.keys():
       conf.write("name="+ usr +",value=" + users[usr] + "\n")

conf.close()
Sample config file for above code.
path=/cygdrive/c/share/enron_mail_20110402/enron_mail_20110402/
maildir
folder=Inbox
name=storey-g,value=142
name=ybarbo-p,value=775
name=tycholiz-b,value=602
```

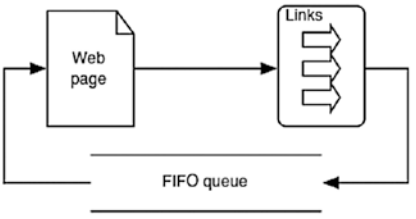
Topical Crawling

Topical crawlers are intelligent crawlers that retrieve information from anywhere on the Web. They start with a URL and then find links present in the pages under it; then they look at new URLs, bypassing the scalability limitations of universal search engines. This is done by distributing the crawling process across users, queries, and even client computers. Crawlers can use the context available to infinitely loop through the links with a goal of systematically locating a highly relevant, focused page.

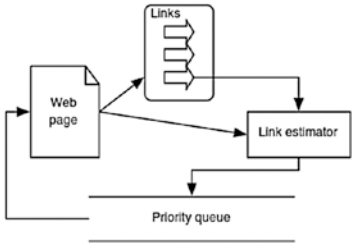
Web searching is a complicated task. A large chunk of machine learning work is being applied to find the similarity between pages, such as the maximum number of URLs fetched or visited.

Crawling Algorithms

The following diagram describes how the topical crawling algorithm works with its major components.




```
Breadth-First (starting_urls) {
   foreach link (starting_urls) {
      enqueue(frontier, link);
   }
   while (visited < MAX_PAGES) {
      link := dequeue_link(frontier);
      doc := fetch(link);
      enqueue(frontier, extract_links(doc));
      if (#frontier > MAX_BUFFER) {
         dequeue_last_links(frontier);
      }
   }
}
```




```
BFS (topic, starting_urls) {
   foreach link (starting_urls) {
      enqueue(frontier, link, 1);
   }
   while (visited < MAX_PAGES) {
      link := dequeue_top_link(frontier);
      doc := fetch(link);
      score :=  sim(topic, doc);
      enqueue(frontier, extract_links(doc), score);
      if (#frontier > MAX_BUFFER) {
         dequeue_bottom_links(frontier);
      }
   }
}
```

The starting URL of a topical crawler is known as the *seed URL*. There is another set of URLs known as the *target URLs*, which are examples of desired output.

An interesting application of topical crawling is where an HR organization is searching for a candidate from anywhere on the Web possessing a particular skill set. One easy alternative solution is to use a search engine API. The following code is an example of using the Google Search API, BeautifulSoup, and regular expressions that search the e-mail ID and phone number of potential candidates with a particular skill set from the Web.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
import pprint, json, urllib2
import nltk, sys, urllib
from bs4 import BeautifulSoup
import csv

from googleapiclient.discovery import build

def link_score(link):
 if ('cv' in link or 'resume' in link) and 'job' not in link:
 return True

def process_file():
 try:

 with open('data1.json','r') as fl:
 data = json.load(fl)
 all_links = []
 # pprint.pprint(len(data['items']))
 for item in data['items']:
 # print item['formattedUrl']
 all_links.append(item['formattedUrl'])
 return all_links
 except:
 return []

def main(istart, search_query):
 service = build("customsearch", "v1",
 developerKey="abcd")

 res = service.cse().list(
 q= search_query,
 cx='1234',
 num=10,
 gl='in', #in for india comment this for whole web
```

```
 start = istart,
 ).execute()
 import json
 with open('data1.json', 'w') as fp:
 json.dump(res, fp)
# pprint.pprint(type(res))
# pprint.pprint(res)

def get_email_ph(link_text, pdf=None):
 if pdf==True:

 from textract import process
 text = process(link_text)
 else:
 text = link_text
 # print text
 import re
 email = []
 ph = []
 valid_ph = re.compile("[789][0-9]{9}$")
 valid = re.compile("[A-Za-z]+[@]{1}[A-Za-z]+\.[a-z]+")
 for token in re.split(r'[,\s]',text):
# for token in nltk.tokenize(text):
 # print token
 a = valid.match(token)
 b = valid_ph.match(token)
 if a != None:
 print a.group()
 email.append(a.group())
 if b != None:
 print b.group()
 ph.append(b.group())
 return email, ph
```

```
def process_pdf_link(link):
 html = urllib2.urlopen(link)
 file = open("document.pdf", 'w')
 file.write(html.read())
 file.close()
 return get_email_ph("document.pdf", pdf=True)

def process_doc_link(link):
 testfile = urllib.URLopener()
 testfile.retrieve(link, "document.doc")
 return get_email_ph("document.doc", pdf=False)

def process_docx_link(link):
 testfile = urllib.URLopener()
 testfile.retrieve(link, "document.docx")
 return get_email_ph("document.docx", pdf=False)

def process_links(all_links):
 with open('email_ph.csv', 'wb') as csvfile:
 spamwriter = csv.writer(csvfile, delimiter=',')

 for link in all_links:
 if link[:4] !='http':
 link = "http://"+link
 print link
 try:
 if link[-3:] == 'pdf':
 try:
 email, ph = process_pdf_link(link)
 spamwriter.writerow([link, ' '.join(email), ' '.join(ph)])
 except:
 print "error",link
 print sys.exc_info()
```

```
 elif link[-4:] == 'docx':
 try:
 email, ph = process_docx_link(link)
 spamwriter.writerow([link, ' '.join(email), ' '.join(ph)])
 except:
 print "error",link
 print sys.exc_info()
 spamwriter.writerow([link, ' '.join(email), ' '.join(ph)])
 elif link[-3:] == 'doc':
 try:
 email, ph = process_doc_link(link)
 spamwriter.writerow([link, ' '.join(email), ' '.join(ph)])
 except:
 print "error",link
 print sys.exc_info()
 spamwriter.writerow([link, ' '.join(email), ' '.join(ph)])
 else:
 try:
 html = urllib2.urlopen(link)
 email, ph = get_email_ph(BeautifulSoup(html.read()).get_
text(), pdf=False)
 spamwriter.writerow([link, ' '.join(email), ' '.join(ph)])
 except:
 print "error",link
 print sys.exc_info()
 spamwriter.writerow([link, ' '.join(email), ' '.join(ph)])
 except:
 pass
 print "error",link
 print sys.exc_info()

if __name__ == '__main__':
```

```
#
 search_query = ' ASP .NET, C#, WebServices, HTML Chicago USA
biodata cv'
#
#
 all_links = []
# all_links.extend(links)
 for i in range(1,90,10):
 main(i, search_query)
 all_links.extend(process_file())

 process_links(all_links)
#
```

This code is used to find relevant contacts from the Web for a set of given job-searching keywords. It uses the Google Search API to fetch the links, filters them according to the presence of certain keywords in a URL, and then parses the link content and finds the e-mail ID and phone number. The content may be PDF or Word or HTML documents.

CHAPTER 3

Supervised Learning Using Python

In this chapter, I will introduce the three most essential components of machine learning.

- *Dimensionality reduction* tells how to choose the most important features from a set of features.
- *Classification* tells how to categorize data to a set of target categories with the help of a given training/example data set.
- *Regression* tells how to realize a variable as a linear or nonlinear polynomial of a set of independent variables.

Dimensionality Reduction with Python

Dimensionality reduction is an important aspect of data analysis. It is required for both numerical and categorical data. Survey or factor analysis is one of the most popular applications of dimensionality reduction. As an example, suppose that an organization wants to find out which factors are most important in influencing or bringing about changes in its operations. It takes opinions from different employees in the organization and, based on this survey data, does a factor analysis to derive a smaller set of factors in conclusion.

S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1_3

In investment banking, different indices are calculated as a weighted average of instruments. Thus, when an index goes high, it is expected that instruments in the index with a positive weight will also go high and those with a negative weight will go low. The trader trades accordingly. Generally, indices consist of a large number of instruments (more than ten). In high-frequency algorithmic trading, it is tough to send so many orders in a fraction of a second. Using principal component analysis, traders realize the index as a smaller set of instruments to commence with the trading. Singular value decomposition is a popular algorithm that is used both in principal component analysis and in factor analysis. In this chapter, I will discuss it in detail. Before that, I will cover the Pearson correlation, which is simple to use. That's why it is a popular method of dimensionality reduction. Dimensionality reduction is also required for categorical data. Suppose a retailer wants to know whether a city is an important contributor to sales volume; this can be measured by using mutual information, which will also be covered in this chapter.

Correlation Analysis

There are different measures of correlation. I will limit this discussion to the Pearson correlation only. For two variables, *x* and *y*, the Pearson correlation is as follows:

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}}$$

The value of *r* will vary from -1 to +1. The formula clearly shows that when *x* is greater than its average, then *y* is also greater, and therefore the *r* value is bigger. In other words, if *x* increases, then *y* increases, and then *r* is greater. So, if *r* is nearer to 1, it means that *x* and *y* are positively correlated. Similarly, if *r* is nearer to -1, it means that *x* and *y* are negatively correlated.

Likewise, if *r* is nearer to 0, it means that *x* and *y* are not correlated. A simplified formula to calculate *r* is shown here:

$$r=\frac{n\left(\sum xy\right)-\left(\sum x\right)\left(\sum y\right)}{\sqrt{\left[n\sum x^2-\left(\sum x\right)^2\right]\left[n\sum y^2-\left(\sum y\right)^2\right]}}$$

You can easily use correlation for dimensionality reduction. Let's say *Y* is a variable that is a weighted sum of *n* variables: `X1, X2, ... Xn`. You want to reduce this set of *X* to a smaller set. To do so, you need to calculate the correlation coefficient for each *X* pair. Now, if Xi and Xj are highly correlated, then you will investigate the correlation of *Y* with Xi and Xj. If the correlation of Xi is greater than Xj, then you remove Xj from the set, and vice versa. The following function is an example of the dropping feature using correlation:

```
from scipy.stats.stats import pearsonr

def drop_features(y_train,X_train,X,index):
        i1 = 0
        processed = 0
       while(1):
       flag = True
                for i in range(X_train.shape[1]):
       if i > processed :
              i1 = i1 + 1
                       corr = pearsonr(X_train[:,i], y_train)
               PEr= .674 * (1- corr[0]*corr[0])/ (len(X_
               train[:,i])**(1/2.0))
      if corr[0] < PEr:
      X_train = np.delete(X_train,i,1)
```

```
                                        index.append(X.
                                        columns[i1-1])
                                    processed = i - 1
                            flag = False
                    break
            if flag:
            break
    return X_train, index
```

The actual use case of this code is shown at the end of the chapter.

Now, the question is, what should be the threshold value of the previous correlation that, say, *X* and *Y* are correlated. A common practice is to assume that if $r > 0.5$, it means the variables are correlated, and if $r < 0.5$, then it means no correlation. One big limitation of this approach is that it does not consider the length of the data. For example, a 0.5 correlation in a set of 20 data points should not have the same weight as a 0.5 correlation in a set of 10,000 data points. To overcome this problem, a probable error concept has been introduced, as shown here:

$$\text{PEr} = .674 \times \frac{1 - r^2}{\sqrt{n}}$$

r is the correlation coefficient, and *n* is the sample size.

Here, $r > 6\text{PEr}$ means that *X* and *Y* are highly correlated, and if $r < \text{Per}$, this means that *X* and *Y* are independent. Using this approach, you can see that even $r = 0.1$ means a high correlation when the data size is huge.

One interesting application of correlation is in product recommendations on an e-commerce site. Recommendations can identify similar users if you calculate the correlation of their common ratings for the same products. Similarly, you can find similar products by calculating the correlation of their common ratings from the same user. This approach is known as *collaborative filtering*.

Principal Component Analysis

Theoretically correlation works well for variables with Gaussian distribution, in other words, independent variables. For other scenarios, you have to use principal component analysis. Suppose you want to reduce the dimension of *N* variables: `X1, X2, ... Xn`. Let's form a matrix of *N*×*N* dimension where the i-th column represents the observation Xi, assuming all variables have *N* number of observations. Now if *k* variables are redundant, for simplicity you assume *k* columns are the same or linear combination of each other. Then the rank of the matrix will be *N*-*k*. So, the rank of this matrix is a measure of the number of independent variables, and the eigenvalue indicates the strength of that variable. This concept is used in principal component analysis and factor analysis. To make the matrix square, a covariance matrix is chosen. Singular value decomposition is used to find the eigenvalue.

Let *Y* be the input matrix of size *p*×*q*, where *p* is the number of data rows and *q* is the number of parameters.

Then the *q*×*q* covariance matrix `Co` is given by the following:

$$\mathbf{Co = YTY/(q-1)}$$

It is a symmetric matrix, so it can be diagonalized as follows:

$$\mathbf{Co = UDU^T}$$

Each column of `U` is an eigenvector, and `D` is a diagonal matrix with eigenvalues λi in the decreasing order on the diagonal. The eigenvectors are referred to as *principal axes* or *principal directions* of the data. Projections of the data on the principal axes called principal components are also known as *PC scores*; these can be seen as new, transformed variables. The j-th principal component is given by j-th column of `YU`. The coordinates of the i-th data point in the new PC space are given by the i-th row of `YU`.

The singular value decomposition algorithm is used to find D and U. The following code is an example of factor analysis in Python.

Here is the input data:

Government Policy Changes	Competitors' Strategic Decisions	Competition	Supplier Relation	Customer Feedback	Technology Innovations
Strongly Agree	Agree	Agree	Agree	Somewhat Agree	Somewhat Disagree
Somewhat Disagree	Somewhat Disagree	Somewhat Agree	Disagree	Disagree	Agree
Somewhat Agree	Somewhat Agree	Strongly Agree	Agree	Somewhat Agree	Strongly Agree
Somewhat Disagree	Somewhat Agree	Agree	Somewhat Disagree	Somewhat Disagree	Somewhat Agree
Somewhat Disagree	Agree	Agree	Somewhat Agree	Somewhat Agree	Agree
Agree	Somewhat Disagree	Somewhat Agree	Strongly Agree	Somewhat Agree	Somewhat Agree
Agree	Agree	Strongly Agree	Somewhat Agree	Agree	Somewhat Agree
Somewhat Disagree	Agree	Somewhat Agree	Agree	Agree	Somewhat Agree
Somewhat Agree	Somewhat Agree	Agree	Agree	Agree	Somewhat Agree
Somewhat Disagree	Agree	Strongly Agree	Somewhat Disagree	Agree	Somewhat Disagree
Somewhat Agree	Agree	Somewhat Disagree	Strongly Agree	Somewhat Agree	Disagree

Somewhat Disagree	Somewhat Disagree	Somewhat Agree	Somewhat Disagree	Somewhat Disagree	Somewhat Agree
Somewhat Agree	Agree	Somewhat Agree	Agree	Somewhat Agree	Somewhat Agree
Somewhat Disagree	Agree	Strongly Agree	Somewhat Disagree	Somewhat Agree	Disagree
Somewhat Agree	Somewhat Disagree	Strongly Agree	Strongly Agree	Strongly Agree	Agree
Somewhat Agree	Somewhat Agree	Agree	Somewhat Disagree	Strongly Agree	Disagree
Somewhat Disagree	Agree	Agree	Somewhat Disagree	Agree	Somewhat Agree
Somewhat Agree	Strongly Agree	Somewhat Agree	Somewhat Agree	Agree	Somewhat Agree
Strongly Agree	Somewhat Disagree	Somewhat Disagree	Agree	Somewhat Agree	Somewhat Disagree
Somewhat Agree	Somewhat Disagree	Agree	Somewhat Agree	Strongly Agree	Somewhat Disagree
Agree	Somewhat Agree	Strongly Agree	Somewhat Disagree	Agree	Agree
Somewhat Agree	Strongly Agree	Somewhat Agree	Somewhat Disagree	Somewhat Disagree	Disagree

Before running the code, you have to enter a numeric value for categorical data, for example: Strongly Agree = 5, Agree = 4, Somewhat Agree = 3.

```
import pandas as pd
data = pd.read_csv('<input csvfile>)

from sklearn.decomposition import FactorAnalysis
factors = FactorAnalysis(n_components=6).fit(data)
print (factors.components)

from sklearn.decomposition import PCA
pcomponents = PCA(n_components=6).fit(data)
print(pcomponents.components)
```

Mutual Information

Mutual information (MI) of two random variables is a measure of the mutual dependence between the two variables. It is also used as a similarity measure of the distribution of two variables. A higher value of mutual information indicates the distribution of two variables is similar.

$$I(X;Y)=\sum_{x,y}p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

Suppose a retailer wants to investigate whether a particular city is a deciding factor for its sales volume. Then the retailer can see the distribution of sales volume across the different cities. If the distribution is the same for all cities, then a particular city is not an important factor as far as sales volume is concerned. To calculate the difference between the two probability distributions, mutual information is applied here.

Here is the sample Python code to calculate mutual information:

```
fromscipy.stats import chi2_contingency


defcalc_MI(x, y, bins):
c_xy = np.histogram2d(x, y, bins)[0]
```

```
g, p, dof, expected = chi2_contingency(c_xy, lambda_="log-
likelihood")
mi = 0.5 * g / c_xy.sum()

return mi
```

Classifications with Python

Classification is a well-accepted example of machine learning. It has a set of a target classes and training data. Each training data is labeled by a particular target class. The classification model is trained by training data and predicts the target class of test data. One common application of classification is in fraud identification in the credit card or loan approval process. It classifies the applicant as fraud or nonfraud based on data. Classification is also widely used in image recognition. From a set of images, if you recognize the image of a computer, it is classifying the image of a computer and not of a computer class.

Sentiment analysis is a popular application of text classification. Suppose an airline company wants to analyze its customer textual feedback. Then each feedback is classified according to sentiment (positive/negative/neutral) and also according to context (about staff/timing/food/price). Once this is done, the airline can easily find out what the strength of that airline's staff is or its level of punctuality or cost effectiveness or even its weakness. Broadly, there are three approaches in classification.

- *Rule-based approach*: I will discuss the decision tree and random forest algorithm.
- *Probabilistic approach*: I will discuss the Naive Bayes algorithm.
- *Distance-based approach*: I will discuss the support vector machine.

Semisupervised Learning

Classification and regression are types of supervised learning. In this type of learning, you have a set of training data where you train your model. Then the model is used to predict test data. For example, suppose you want to classify text according to sentiment. There are three target classes: positive, negative, and neutral. To train your model, you have to choose some sample text and label it as positive, negative, and neutral. You use this training data to train the model. Once your model is trained, you can apply your model to test data. For example, you may use the Naive Bayes classifier for text classification and try to predict the sentiment of the sentence "Food is good." In the training phase, the program will calculate the probability of a sentence being positive or negative or neutral when the words *Food*, *is*, and *good* are presented separately and stored in the model, and in the test phase it will calculate the joint probability when *Food*, *is*, and *good* all come together. Conversely, clustering is an example of unsupervised learning where there is no training data or target class available. The program learns from data in one shot. There is an instance of semisupervised learning also. Suppose you are classifying the text as positive and negative sentiments but your training data has only positives. The training data that is not positive is unlabeled. In this case, as the first step, you train the model assuming all unlabeled data is negative and apply the trained model on the training data. In the output, the data coming in as negative should be labeled as negative. Finally, train your model with the newly labeled data. The nearest neighbor classifier is also considered as semisupervised learning. It has training data, but it does not have the training phase of the model.

Decision Tree

A *decision tree* is a tree of rules. Each level of the tree represents a parameter, each node of the level validates a constraint for that level parameter, and each branch indicates a possible value of parent node parameter. Figure 3-1 shows an example of a decision tree.

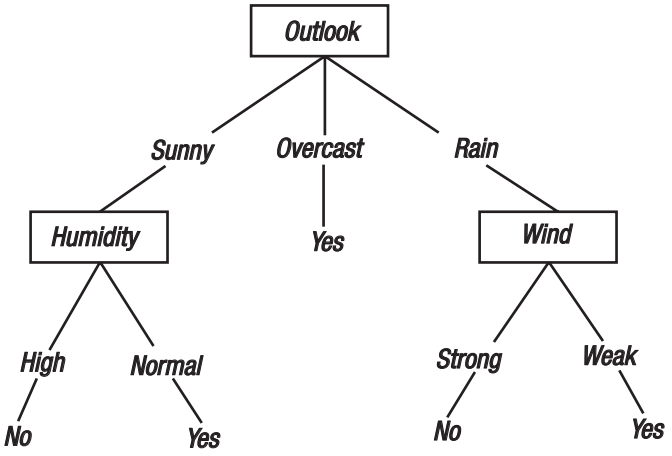


Figure 3-1. *Example of decision tree for good weather*

Which Attribute Comes First?

One important aspect of the decision tree is to decide the order of features. The entropy-based information gain measure decides it.

Entropy is a measure of randomness of a system.

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

For example, for any obvious event like the sun rises in the east, entropy is zero, P=1, and log(p)=0. More entropy means more uncertainty or randomness in the system.

Information gain, which is the expected reduction in entropy caused by partitioning the examples according to this attribute, is the measure used in this case.

Specifically, the information gain, $Gain(S,A)$, of an attribute A relative to a collection of examples S is defined as follows:

$$Gain(S,A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

So, an attribute with a higher information gain will come first in the decision tree.

```
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv('csv file path', index_col=0)
y = df[target class column ]
X = df[ col1, col2 ..]

clf= DecisionTreeClassifier()
clf.fit(X,y)
clf.predict(X_test)
```

Random Forest Classifier

A *random forest classifier* is an extension of a decision tree in which the algorithm creates N number of decision trees where each tree has M number of features selected randomly. Now a test data will be classified by all decision trees and be categorized in a target class that is the output of the majority of the decision trees.

```
from sklearn.ensemble import RandomForestClassifier

df = pd.read_csv('csv file path', index_col=0)

y = df[target class column ]
X = df[ col1, col2 ..]

clf=RandomForestClassifier(n_jobs=2,random_state=0)
clf.fit(X,y)
clf.predict(X_test)
```

Naive Bayes Classifier

X = (x1, x2, x3, ..., xn) is a vector of n dimension. The Bayesian classifier assigns each X to one of the target classes of set {C1, C2, …, Cm,}. This assignment is done on the basis of probability that X belongs to target class Ci. That is to say, X is assigned to class Ci if and only if P(Ci |X) > P(Cj |X) for all j such that $1 \leq j \leq m$.

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

In general, it can be costly computationally to compute P(X|Ci). If each component x_k of X can have one of r values, there are r^n combinations to consider for each of the m classes. To simplify the calculation, the assumption of conditional class independence is made, which means that for each class, the attributes are assumed to be independent. The classifier developing from this assumption is known as the Naive Bayes classifier. The assumption allows you to write the following:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

The following code is an example of the Naive Bayes classification of numerical data:

```
#Import Library of Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
import numpy as np

#assigning predictor and target variables
df = pd.read_csv('csv file path', index_col=0)
y = df[target class column ]
X = df[ col1, col2 ..]

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(X, y)

#Predict Output
print model.predict([input array])
```

Note You'll see another example of the Naive Bayes classifier in the "Sentiment Analysis" section.

Support Vector Machine

If you look at Figure 3-2, you can easily understand that the circle and square points are linearly separable in two dimensions (x1, x2). But they are not linearly separable in either dimension x1 or x2. The support vector machine algorithm works on this theory. It increases the dimension of the data until points are linearly separable. Once that is done, you have

to find two parallel hyperplanes that separate the data. This planes are known as the *margin*. The algorithm chose the margins in such a way that the distance between them is the maximum. That's why it is the maximum margin. The plane, which is at the middle of these two margins or at equal distance between them, is known as an optimal hyperplane that is used to classify the test data (see Figure 3-2). The separator can be nonlinear also.

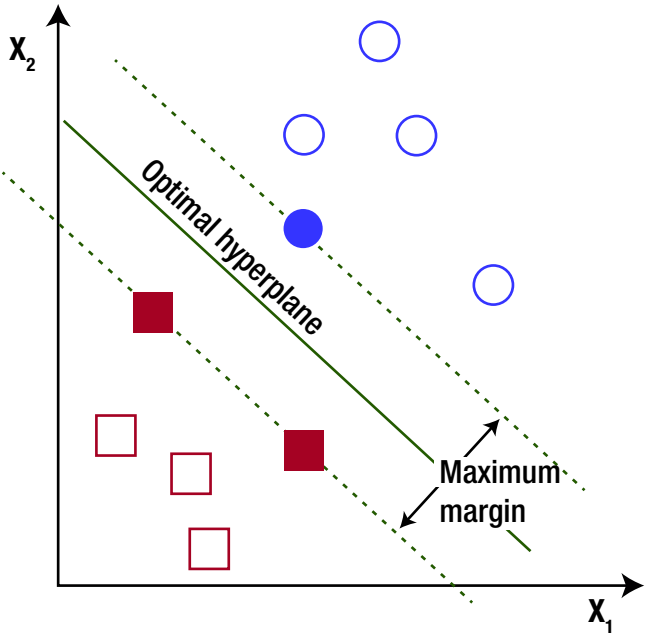


Figure 3-2. *Support vector machine*

The following code is an example of doing support vector machine classification using Python:

```
from sklearn import svm
df = pd.read_csv('csv file path', index_col=0)
y = df[target class column ]
X = df[ col1, col2 ..]
```

```
model.fit(X, y)
model.score(X, y)

print model.predict(x_test)
```

Nearest Neighbor Classifier

The nearest neighbor classifier is a simple distance-based classifier. It calculates the distance of test data from the training data and groups the distances according to the target class. The target class, which has a minimum average distance from the test instance, is selected as the class of the test data. A Python example is shown here:

```
def Distance(point1, point2, length):
      distance = 0
      for x in range(length):
            distance += pow((point1[x] -point2[x]), 2)
      return math.sqrt(distance)
def getClosePoints(trainingData, testData, k):
      distances = []
      length = len(testInstance)-1
      for x in range(len(trainingData)):
            dist = Distance(testData, trainingDatat[x], length)
            distances.append((trainingData[x], dist))
      distances.sort(key=operator.itemgetter(1))
      close= []
      for x in range(k):
            close.append(distances[x][0])
      return close
```

```
trainData = [[3,3,3,, 'A'], [5,5,5,, 'B']]
testData = [7,7,7]
k = 1
neighbors = getClosePoints(trainData, testData, 1)
print(neighbors)
```

Sentiment Analysis

Sentiment analysis is an interesting application of text classification. For example, say one airline client wants to analyze its customer feedback. It classifies the feedback according to sentiment (positive/negative) and also by aspect (food/staff/punctuality). After that, it can easily understand its strengths in business (the aspect that has the maximum positive feedback) and its weaknesses (the aspect that has the maximum negative feedback). The airline can also compare this result with its competitor. One interesting advantage of doing a comparison with the competitor is that it nullifies the impact of the accuracy of the model because the same accuracy is applied to all competitors. This is simple to implement in Python using the textblob library, as shown here:

```
from textblob.classifiers import NaiveBayesClassifier

train = [('I love this sandwich.', 'pos'),  ('this is an
amazing place!', 'pos'),('I feel very good about these
beers.', 'pos'),('this is my best work.', 'pos'),("what
an awesome view", 'pos'),('I do not like this restaurant',
'neg'),('I am tired of this stuff.', 'neg'),("I can't deal with
this", 'neg'),('he is my sworn enemy!', 'neg'),('my boss is
horrible.', 'neg')]
```

```
cl = NaiveBayesClassifier(train)
print (cl.classify("This is an amazing library!"))
```

```
output :  pos
```

```
from textblob.classifiers import NaiveBayesClassifier
```

```
train = [('Air India did a poor job of queue management
both times.', 'staff service'),  (“The 'cleaning' by flight
attendants involved regularly spraying air freshener in
the lavatories.", 'staff'),('The food tasted decent.',
'food'),('Flew Air India direct from New York to Delhi
round trip.', 'route'),('Colombo to Moscow via Delhi.',
'route'),('Flew Birmingham to Delhi with Air India.',
'route'),('Without toilet, food or anything!', 'food'),('Cabin
crew announcements included a sincere apology for the delay.',
'cabin flown')]
```

```
cl = NaiveBayesClassifier(train)
```

```
tests = ['Food is good.']
for c in tests:
      printcl.classify(c)
```

```
Output : food
```

The textblob library also supports a random forest classifier, which works best on text written in proper English such as a formal letter might be. For text that is not usually written with proper grammar, such as customer feedback, Naive Bayes works better. Naive Bayes has another advantage in real-time analytics. You can update the model without losing the previous training.

Image Recognition

Image recognition is a common example of image classification. It is easy to use in Python by applying the opencv library. Here is the sample code:

```
faceCascade=cv2.CascadeClassifier(cascPath)
image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    flags = cv2.cv.CV_HAAR_SCALE_IMAGE
)

print"Found {0} faces!".format(len(faces))
```

Regression with Python

Regression realizes a variable as a linear or nonlinear polynomial of a set of independent variables.

Here is an interesting use case: what is the sales price of a product that maximizes its profit? This is a million-dollar question for any merchant. The question is not straightforward. Maximizing the sales price may not result in maximizing the profit because increasing the sales price sometimes decreases the sales volume, which decreases the total profit. So, there will be an optimized value of sales price for which the profit will be at the maximum.

There is *N* number of records of the transaction with *M* number of features called F1, F2, ... Fm (sales price, buy price, cash back, SKU, order date, and so on). You have to find a subset of K(K<M) features that have an

impact on the profit of the merchant and suggest an optimal value of V1, V2, ... Vk for these K features that maximize the revenue.

You can calculate the profit of merchant using the following formula:

$$\text{Profit} = (SP - TCB - BP) * SV \qquad (1)$$

For this formula, here are the variables:

- SP = Sales price
- TCB = Total cash back
- BP = Buy price
- SV = Sales volume

Now using regression, you can realize SV as follows:

$$SV = a + b*SP + c*TCB + d*BP$$

Now you can express profit as a function of SP, TCB, and BP and use mathematical optimization. With constraining in all parameter values, you can get optimal values of the parameters that maximize the profit.

This is an interesting use case of regression. There are many scenarios where one variable has to be realized as the linear or polynomial function of other variables.

Least Square Estimation

Least square estimation is the simplest and oldest method for doing regression. It is also known as the *curve fitting method*. Ordinary least squares (OLS) regression is the most common technique and was invented in the 18th century by Carl Friedrich Gauss and Adrien-Marie Legendre. The following is a derivation of coefficient calculation in ordinary least square estimation:

$$Y = X\beta + \in$$
$$X'Y = X'X\beta + X' \in$$
$$X'Y = X'X\beta + 0$$
$$(X'X)^{-1} X'Y = \beta + 0$$
$$\beta = (X'X)^{-1} X'Y$$

The following code is a simple example of OLS regression:

```
fromscipyimport stats
df = pd.read_csv('csv file path', index_col=0)

y = df[target column ]
X = df[ col1, col2 ..]

X=sm.add_constant(X)


slope, intercept, r_value, p_value, std_err = stats.
linregress(X,y)
```

Logistic Regression

Logistic regression is an interesting application of regression that calculates the probability of an event. It introduces an intermediate variable that is a linear regression of linear variable. Then it passes the intermediate variable through the logistic function, which maps the intermediate variable from zero to one. The variable is treated as a probability of the event.

The following code is an example of doing logistic regression on numerical data:

```
import pandas as pd
import statsmodels.api as sm
df = pd.read_csv('csv file path', index_col=0)
```

```
y = df[target column ]
X = df[ col1, col2 ..]
X=sm.add_constant(X)
logit=sm.Logit(y,X)
result=logit.fit()
result.summary()
```

Classification and Regression

Classification and regression may be applied on the same problem. For example, when a bank approves a credit card or loan, it calculates a credit score of a candidate based on multiple parameters using regression and then sets up a threshold. Candidates having a credit score greater than the threshold are classified as potential nonfraud, and the remaining are considered as potential fraud. Likewise, any binary classification problem can be solved using regression by applying a threshold on the regression result. In Chapter 4, I discussed in detail how to choose a threshold value from the distribution of any parameter. Similarly, some binary classifications can be used in place of logistic regression. For instance, say an e-commerce site wants to predict the chances of a purchase order being converted into a successful invoice. The site can easily do so using logistic regression. The Naive Bayes classifier can be directly applied on the problem because it calculates probability when it classifies the purchase order to be in the successful or unsuccessful class. The random forest algorithm can be used in the problem as well. In that case, among the N decision tree, if the M decision tree says the purchase order will be successful, then M/N will be the probability of the purchase order being successful.

Intentionally Bias the Model to Over-Fit or Under-Fit

Sometimes you need to over- or under-predict intentionally. In an auction, when you are predicting from the buy side, it will always be good if your bid is little lower than the original. Similarly, on the sell side, it is desired that you set the price a little higher than the original. You can do this in two ways. In regression, when you are selecting the features using correlation, over-predicting intentionally drops some variable with negative correlation. Similarly, under-predicting drops some variable with positive correlation. There is another way of dealing with this. When you are predicting the value, you can predict the error in the prediction. To over-predict, when you see that the predicted error is positive, reduce the prediction value by the error amount. Similarly, to over-predict, increase the prediction value by the error amount when the error is positive.

Another problem in classification is biased training data. Suppose you have two target classes, A and B. The majority (say 90 percent) of training data is class A. So, when you train your model with this data, all your predictions will become class A. One solution is a biased sampling of training data. Intentionally remove the class A example from training. Another approach can be used for binary classification. As class B is a minority in the prediction probability of a sample, in class B it will always be less than 0.5. Then calculate the average probability of all points coming into class B. For any point, if the class B probability is greater than the average probability, then mark it as class B and otherwise class A.

The following code is an example of a tuning classification condition:

```
y_test_increasing, predicted_increasing = predict(d1, True, False)
y_test_decreasing, predicted_decreasing = predict(d2, False, False)

prob_increasing = predicted_increasing[:,1]
increasing_mean = prob_increasing.mean()
increasing_std = prob_increasing.std()
prob_decreasing = predicted_decreasing[:,0]
decreasing_mean = prob_decreasing.mean()
decreasing_std = prob_decreasing.std()
ifi> 0:
    mean_std_inc = (mean_std_inc + increasing_std)/2
    mean_std_dec = (mean_std_dec + decreasing_std)/2
else:
    mean_std_inc = increasing_std
    mean_std_dec = decreasing_std

for j in range(len(y_test_decreasing)-1):
                    ac_status = y_test_increasing[j] + y_test_
                    decreasing[j]
                    pr_status = 0
                    if True:
                           inc = (prob_increasing[j] -
                           increasing_mean + mean_std_inc)
                           dec = (prob_decreasing[j] -
                           decreasing_mean + mean_std_dec)
                           ifinc> 0 or dec> 0:
                                 ifinc>dec:
                                        pr_status = 1
                                 else:
                                        pr_status = -1
                           else:
                                 pr_status = 0
```

Dealing with Categorical Data

For algorithm-like support, vector or regression input data must be numeric. So, if you are dealing with categorical data, you need to convert to numeric data. One strategy for conversion is to use an ordinal number as the numerical score. A more sophisticated way to do this is to use an expected value of the target variable for that value. This is good for regression.

```
for col in X.columns:
       avgs = df.groupby(col, as_index=False)['floor'].
       aggregate(np.mean)
       fori,row in avgs.iterrows():
       k = row[col]
                              v = row['floor']
       X.loc[X[col] == k, col] = v
```

For logistic regression, you can use the expected probability of the target variable for that categorical value.

```
for col in X.columns:
       if str(col) != 'success':
             if str(col) not in index:
                   feature_prob = X.groupby(col).size().
                   div(len(df))
                   cond_prob = X.groupby(['success',
                   str(col)]).size().div(len(df)).div(feature_
                   prob, axis=0, level=str(col)).reset_
                   index(name="Probability")
                   cond_prob = cond_prob[cond_prob.success != '0']
                   cond_prob.drop("success",inplace=True, axis=1)
                   cond_prob['feature_value'] = cond_
                   prob[str(col)].apply(str).as_matrix()
```

```
                cond_prob.drop(str(col),inplace=True, axis=1)
                for i, row in cond_prob.iterrows():
                        k = row["feature_value"]
                        v = row["Probability"]
                        X.loc[X[col] == k, col] = v
    else:
          X.drop(str(col),inplace=True, axis=1)
```

The following example shows how to deal with categorical data and how to use correlation to select a feature. The following is the complete code of data preprocessing. The data for this code example is also available online at the Apress website.

```
def process_real_time_data(time_limit):

    df = pd.read_json(json.loads(<input>))
    df.replace('^\s+', '', regex=True, inplace=True) #front
    df.replace('\s+$', '', regex=True, inplace=True) #end

    time_limit = df['server_time'].max()

    df['floor'] = pd.to_numeric(df['floor'],
    errors='ignore')
    df['client_time'] = pd.to_numeric(df['client_time'],
    errors='ignore')
    df['client_time'] = df.apply(lambda row: get_hour(row.
    client_time), axis=1)

    y = df['floor']
    X = df[['ip','size','domain','client_time','device','ad_
    position','client_size','root']]
    X_back = df[['ip','size','domain','client_
    time','device','ad_position','client_size','root']]
```

```
for col in X.columns:
      avgs = df.groupby(col, as_index=False)['floor'].
      aggregate(np.mean)
      for index,row in avgs.iterrows():
            k = row[col]
            v = row['floor']
            X.loc[X[col] == k, col] = v

X.drop('ip', inplace=True, axis=1)
X_back.drop('ip', inplace=True, axis=1)

X_train, X_test, y_train, y_test = cross_validation.
train_test_split(X, y, test_size= 0, random_state=42)


X_train = X_train.astype(float)
y_train = y_train.astype(float)

X_train = np.log(X_train + 1)
y_train = np.log(y_train + 1)

X_train = X_train.as_matrix()
y_train = y_train.as_matrix()

index = []
i1 = 0
processed = 0
while(1):
      flag = True
      for i in range(X_train.shape[1]):
            if i > processed :
#print(i1,X_train.shape[1],X.columns[i1])
                  i1 = i1 + 1
                  corr = pearsonr(X_train[:,i], y_train)
```

```
                    PEr= .674 * (1- corr[0]*corr[0])/
                    (len(X_train[:,i])**(1/2.0))
                    if corr[0] < PEr:
                          X_train = np.delete(X_train,i,1)
                          index.append(X.columns[i1-1])
                          processed = i - 1
                          flag = False
                          break
      if flag:
            break

return y_train, X_train, y, X_back, X, time_limit,
index
```

CHAPTER 4

Unsupervised Learning: Clustering

In Chapter 3 we discussed how training data can be used to categorize customer comments according to sentiment (positive, negative, neutral), as well as according to context. For example, in the airline domain, the context can be punctuality, food, comfort, entertainment, and so on. Using this analysis, a business owner can determine the areas that his business he needs to concentrate on. For instance, if he observes that the highest percentage of negative comments has been about food, then his priority will be the quality of food being served to the customers. However, there are scenarios where business owners are not sure about the context. There are also cases where training data is not available. Moreover, the frame of reference can change with time. Classification algorithms cannot be applied where target classes are unknown. A clustering algorithm is used in these kinds of situations. A conventional application of clustering is found in the wine-making industry where each cluster represents a brand of wine, and wines are clustered according to their component ratio in wine. In Chapter 3, you learned that classification can be used to recognize a type of image, but there are scenarios where one image has multiple shapes and an algorithm is needed to separate the figures. Clustering algorithms are used in this kind of use case.

S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1_4

Clustering classifies objects into groups based on similarity or distance measure. This is an example of unsupervised learning. The main difference between clustering and classification is that the latter has well-defined target classes. The characteristics of target classes are defined by the training data and the models learned from it. That is why classification is supervised in nature. In contrast, clustering tries to define meaningful classes based on data and its similarity or distance. Figure 4-1 illustrates a document clustering process.

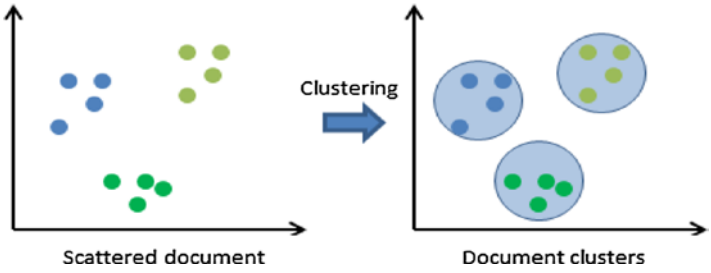


Figure 4-1. *Document clustering*

K-Means Clustering

Let's suppose that a retail distributer has an online system where local agents enter trading information manually. One of the fields they have to fill in is City. But because this data entry process is manual, people normally tend to make lots of spelling errors. For example, instead of Delhi, people enter *Dehi*, *Dehli*, *Delly*, and so on. You can try to solve this problem using clustering because the number of clusters are already known; in other words, the retailer is aware of how many cities the agents operate in. This is an example of *K-means clustering*.

The K-means algorithm has two inputs. The first one is the data X, which is a set of N number of vectors, and the second one is K, which represents the number of clusters that needs to be created. The output is a set of K centroids in each cluster as well as a label to each vector in X that indicates

the points assigned to the respective cluster. All points within a cluster are nearer in distance to their centroid than any other centroid. The condition for the K clusters C_k and the K centroids μ_k can be expressed as follows:

$$\text{minimize} \sum_{k=1}^{K} \sum_{x_n \in C_k} \left\| x_n - \mu_k \right\|^2 \text{ with respect to } C_k, \mu_k.$$

However, this optimization problem cannot be solved in polynomial time. But Lloyd has proposed an iterative method as a solution. It consists of two steps that iterate until the program converges to the solution.

1. It has a set of K centroids, and each point is assigned to a unique cluster or centroid, where the distance of the concerned centroid from that particular point is the minimum.
2. It recalculates the centroid of each cluster by using the following formula:

$$C_k = \left\{ Xn : \left\| Xn - \mu k \right\| \leq \text{all} \left\| Xn - \mu l \right\| \right\} \quad (1)$$

$$\mu_k = \frac{1}{Ck} \sum_{Xn \in Ck} Xn \quad (2)$$

The two-step procedure continues until there is no further re-arrangement of cluster points. The convergence of the algorithm is guaranteed, but it may converge to a local minima.

The following is a simple implementation of Lloyd's algorithm for performing K-means clustering in Python:

```
import random
def ED(source, target):
if source == "":
return len(target)
if target == "":
return len(source)
```

```
if source[-1] == target[-1]:
cost = 0
else:
cost = 1

res = min([ED(source[:-1], target)+1,
ED(source, target[:-1])+1,
ED(source[:-1], target[:-1]) + cost])
return res

def find_centre(x, X, mu):
min = 100
cent = 0
for c in mu:
dist = ED(x, X[c])
if dist< min:
min = dist
cent = c
return cent

def cluster_arrange(X, cent):
clusters  = {}
for x in X:
bestcent = find_centre(x, X, cent)
try:
clusters[bestcent].append(x)
exceptKeyError:
clusters[bestcent] = [x]
return clusters

def rearrange_centers(cent, clusters):
newcent = []
keys = sorted(clusters.keys())
```

```
for k in keys:
newcent.append(k)
return newcent

def has_converged(cent, oldcent):
return sorted(cent) == sorted(oldcent)

def locate_centers(X, K):
oldcent = random.sample(range(0,5), K)
cent = random.sample(range(0,5), K)
while not has_converged(cent, oldcent):
oldcent = cent
        # Assign all points in X to clusters
clusters = cluster_arrange(X, cent)
        # Reevaluate centers
cent = rearrange_centers(oldcent, clusters)
return(cent, clusters)

X = ['Delhi','Dehli', 'Delli','Kolkata','Kalkata','Kalkota']

print(locate_centers(X,2))
```

However, K-means clustering has a limitation. For example, suppose all of your data points are located in eastern India. For K=4 clustering, the initial step is that you randomly choose a center in Delhi, Mumbai, Chennai, and Kolkata. All of your points lie in eastern India, so all the points are nearest to Kolkata and are always assigned to Kolkata. Therefore, the program will converge in one step. To avoid this problem, the algorithm is run multiple times and averaged. Programmers can use various tricks to initialize the centroids in the first step.

Choosing K: The Elbow Method

There are certain cases where you have to determine the *K* in K-means clustering. For this purpose, you have to use the elbow method, which uses a percentage of variance as a variable dependent on the number of clusters. Initially, several clusters are chosen. Then another cluster is added, which doesn't make the modeling of data much better. The number of clusters is chosen at this point, which is the *elbow criterion*. This "elbow" cannot always be unambiguously identified. The percentage of variance is realized as the ratio of the between-group variance of individual clusters to the total variance. Assume that in the previous example, the retailer has four cities: Delhi, Kolkata, Mumbai, and Chennai. The programmer does not know that, so he does clustering with K=2 to K=9 and plots the percentage of variance. He will get an elbow curve that clearly indicates K=4 is the right number for K.

Distance or Similarity Measure

The measure of distance or similarity is one of the key factors of clustering. In this section, I will describe the different kinds of distance and similarity measures. Before that, I'll explain what *distance* actually means here.

Properties

The distances are measures that satisfy the following properties:

- dist(x, y) = 0 if and only if x=y.
- dist(x, y) > 0 when $x \neq y$.
- dist(x, y) = dist(x, y).
- dist(x,y) + dist(y,z) >= d(z,x) for all x, y, and z.

General and Euclidean Distance

The distance between the points p and q is the length of the geometrical line between them: ($\overline{pq}$) . This is called *Euclidean distance*.

According to Cartesian coordinates, if $p = (p_1, p_2,..., p_n)$ and $q = (q_1, q_2,..., q_n)$ are the two points in Euclidean *n*-space, then the distance (d) from q to p or from p to q is derived from the Pythagorean theorem, as shown here:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

The Euclidean vector is the position of a point in a Euclidean *n*-space. The magnitude of a vector is a measure. Its length is calculated by the following formula:

$$\|p\| = \sqrt{p_1^{\,2} + p_2^{\,2} + \cdots + p_n^{\,2}} = \sqrt{p \cdot p},$$

A vector has direction along with a distance. The distance between two points, p and q, may have a direction, and therefore, it may be represented by another vector, as shown here:

$$q - p = (q_1 - p_1, q_2 - p_2, \ldots, q_n - p_n)$$

The Euclidean distance between p and q is simply the Euclidean length of this distance (or displacement) vector.

$$\|q - p\| = \sqrt{(q - p) \cdot (q - p)}$$
$$\|q - p\| = \sqrt{\|p\|^2 + \|q\|^2 - 2p \cdot q}$$

In one dimension:

$$\sqrt{(x-y)^2} = |x-y|.$$

In two dimensions:

In the Euclidean plane, if $p = (p_1, p_2)$ and $q = (q_1, q_2)$, then the distance is given by the following:

$$d(p,q) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2}$$

Alternatively, it follows from the equation that if the polar coordinates of the point p are (r_1, θ_1) and those of q are (r_2, θ_2), then the distance between the points is as follows:

$$\sqrt{r_1^2 + r_2^2 - 2r_1r_2\cos(\theta_1 - \theta_2)}$$

In *n* dimensions:

In the general case, the distance is as follows:

$$D^2(p,q) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_i - q_i)^2 + \ldots + (p_n - q_n)^2.$$

In Chapter 3, you will find an example of Euclidian distance in the nearest neighbor classifier example.

Squared Euclidean Distance

The standard Euclidean distance can be squared to place progressively greater weight on objects that are farther apart. In this case, the equation becomes the following:

$$d^2(p,q) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_i - q_i)^2 + \ldots + (p_n - q_n)^2.$$

Squared Euclidean distance is not a metric because it does not satisfy the triangle inequality. However, it is frequently used in optimization problems in which distances are to be compared only.

Distance Between String-Edit Distance

Edit distance is a measure of dissimilarity between two strings. It counts the minimum number of operations required to make two strings identical. Edit distance finds applications in natural language processing, where automatic spelling corrections can indicate candidate corrections for a misspelled word. Edit distance is of two types.

- Levenshtein edit distance
- Needleman edit distance

Levenshtein Distance

The Levenshtein distance between two words is the least number of insertions, deletions, or replacements that need to be made to change one word into another. In 1965, it was Vladimir Levenshtein who considered this distance.

Levenshtein distance is also known as *edit distance*, although that might denote a larger family of distance metrics as well. It is affiliated with pair-wise string alignments.

For example, the Levenshtein distance between Calcutta and Kolkata is 5, since the following five edits change one into another:

Calcutta → Kalcutta (substitution of *C* for *K*)

Kalcutta → Kolcutta (substitution of *a* for *o*)

Kolcutta → Kolkutta (substitution of *c* for *k*)

Kolkutta → Kolkatta (substitution of *u* for *a*)

Kolkatta → Kolkata (deletion of *t*)

When the strings are identical, the Levenshtein distance has several simple upper bounds that are the lengths of the larger strings and the lower bounds are zero. The code example of the Levinstein distance is given in K-mean clustering code.

Needleman–Wunsch Algorithm

The Needleman–Wunsch algorithm is used in bioinformatics to align protein or nucleotide sequences. It was one of the first applications of dynamic programming for comparing biological sequences. It works using dynamic programming. First it creates a matrix where the rows and columns are alphabets. Each cell of the matrix is a similarity score of the corresponding alphabet in that row and column. Scores are one of three types: matched, not matched, or matched with insert or deletion. Once the matrix is filled, the algorithm does a backtracing operation from the bottom-right cell to the top-left cell and finds the path where the neighbor score distance is the minimum. The sum of the score of the backtracing path is the Needleman–Wunsch distance for two strings.

Pyopa is a Python module that provides a ready-made Needleman–Wunsch distance between two strings.

```
import pyopa
data = {'gap_open': -20.56,
        'gap_ext': -3.37,
        'pam_distance': 150.87,
        'scores': [[10.0]],
        'column_order': 'A',
        'threshold': 50.0}

env = pyopa.create_environment(**data)

s1 = pyopa.Sequence('AAA')
s2 = pyopa.Sequence('TTT')
print(pyopa.align_double(s1, s1, env))
print(env.estimate_pam(aligned_strings[0], aligned_strings[1]))
```

Although Levenshtein is simple in implementation and computationally less expensive, if you want to introduce a gap in string matching (for example, *New Delhi* and *NewDelhi*), then the Needleman-Wunsch algorithm is the better choice.

Similarity in the Context of Document

A similarity measure between documents indicates how identical two documents are. Generally, similarity measures are bounded in the range [-1,1] or [0,1] where a similarity score of 1 indicates maximum similarity.

Types of Similarity

To measure similarity, documents are realized as a vector of terms excluding the stop words. Let's assume that A and B are vectors representing two documents. In this case, the different similarity measures are shown here:

- **Dice**

  The Dice coefficient is denoted by the following:

  $\text{sim}(q,d_j) = D(A,B) = \frac{|A \cap B|}{\alpha|A| + (1-\alpha)|B|}$

  Also,

  $\alpha \in [0,1]$ and let $\alpha = \frac{1}{2}$

- **Overlap**

  The Overlap coefficient is computed as follows:

  $\text{Sim}(q,d_j) = O(A,B) = \frac{|A \cap B|}{\min(|A|,|B|)}$

  The Overlap coefficient is calculated using the max operator instead of min.

- **Jaccard**

  The Jaccard coefficient is given by the following:

  $$\text{Sim}(q,d_j) = J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

  The Jaccard measure signifies the degree of relevance.

- **Cosine**

  The cosine of the angle between two vectors is given by the following:

  $$\text{Sim}(q,d_j) = C(A,B) = \frac{|A \cap B|}{\sqrt{|A||B|}}$$

Distance and similarity are two opposite measures. For example, numeric data correlation is a similarity measure, and Euclidian distance is a distance measure. Generally, the value of the similarity measure is limited to between 0 and 1, but distance has no such upper boundary. Similarity can be negative, but by definition, distance cannot be negative. The clustering algorithms are almost the same as from the beginning of this field, but researchers are continuously finding new distance measures for varied applications.

What Is Hierarchical Clustering?

Hierarchical clustering is an iterative method of clustering data objects. There are two types.

- Agglomerative hierarchical algorithms, or a bottom-up approach
- Divisive hierarchical algorithms, or a top-down approach

Bottom-Up Approach

The bottom-up clustering method is called *agglomerative hierarchical clustering*. In this approach, each input object is considered as a separate cluster. In each iteration, an algorithm merges the two most similar clusters into only a single cluster. The operation is continued until all the clusters merge into a single cluster. The complexity of the algorithm is O(n^3).

In the algorithm, a set of input objects, $I = \{I_1, I_2, \ldots, I_n\}$, is given. A set of ordered triples is <D,K,S>, where D is the threshold distance, K is the number of clusters, and S is the set of clusters.

Some variations of the algorithm might allow multiple clusters with identical distances to be merged into a single iteration.

Algorithm

Input: $I=\{I_1, I_2, \ldots, I_n\}$
Output: O

```
for i = 1 to n do
        Ci ← {Ii};
end for
D ← 0;
K ← n;
S ← {C1,....., Cn};
O ← <d, k, S>;
repeat
        Dist ← CalcultedMinimumDistance(S);
        D ← ∞;
        For i = 1 to K-1 do
                For j = i+1 to K do
                        if Dist(i, j)< D then
                                D← Dist(i, j);
                                u ← i;
                                v ← j;
```

```
                    end if
                end for
            end for
            K ← K-1;
            Cnew ← Cu ∪Cv;
            S ← S∪ Cnew –Cu – Cv;
            O ← O∪<D, K, S>
Until K = 1;
```

A Python example of hierarchical clustering is given later in the chapter.

Distance Between Clusters

In hierarchical clustering, calculating the distance between two clusters is a critical step. There are three methods to calculate this.

- Single linkage method
- Complete linkage method
- Average linkage method

Single Linkage Method

In the single linkage method, the distance between two clusters is the minimum distance of all distances between pairs of objects in two clusters. As the distance is the minimum, there will be a single pair of objects that has less than equal distance between two clusters. So, the single linkage method may be given as follows:

$$\text{Dist}(C_i, C_j) = \min \text{dist}(X, Y)$$

$$X \in C_i, Y \in C_j$$

Complete Linkage Method

In the complete linkage method, the distance between two clusters is the maximum distance of all distance between pairs of objects in two clusters. The distance is the maximum, so all pairs of distances are less than equal than the distance between two clusters. So, the complete linkage method can be given by the following:

$$\text{Dist}(C_i, C_j) = \max \text{dist}(X, Y)$$

$$X \in C_i, Y \in C_j$$

Average Linkage Method

The average linkage method is a compromise between the previous two linkage methods. It avoids the extremes of large or compact clusters. The distance between clusters C_i and C_j is defined by the following:

$$\text{Dist}(C_i, C_j) = \frac{\sum_{X \in Ci} \sum_{Y \in C_j} \text{dist}(X, Y)}{|C_i| \times |C_j|}$$

$| C_k |$ is the number of data objects in cluster C_k.

The centroid linkage method is similar to the average linkage method, but here the distance between two clusters is actually the distance between the centroids. The centroid of cluster C_i is defined as follows:

$$X_c = (c_1, \ldots, c_m), \text{ with}$$

$$c_j = 1/m \sum X_{kj},$$

X_{kj} is the j-th dimension of the k-th data object in cluster C_i.

Top-Down Approach

The top-down clustering method is also called the *divisive hierarchical clustering*. It the reverse of bottom-up clustering. It starts with a single cluster consisting of all input objects. After each iteration, it splits the cluster into two parts having the maximum distance.

Algorithm

Input: I = {I1, I2, … , In}
Output: O

```
D ← ∞;
K ← 1;
S ← {I1,I2 , ... , In};
O ← <D, K, S >;
repeat
        X ← containing two data objects with the longest
            distance dist;
        Y ← ∅;
        S ← S - X;
        Xi ← data object in A with maximum D̄(Xi, X);
        X ← X − {Xi};
        Y ← Y ∪ {Xi};
        repeat
                forall data object Xj in X do
                        e(j) ← D̄(Xj, X) − D̄(Xj, Y);
                end for
                if∃e(j) > 0 then
                        Xk ← data object in X with maximum e(j);
                        X ← X − {Xk};
                        Y ← Y ∪ {Xk};
                        split ← TRUTH;
                else
```

```
                    split← FALSE;
                end if
            until split == FALSE;
            D ← dist;
            K ← K+1;
            S ← S∪ X ∪ Y
            O ← O ∪ <D, K, S>;
Until  K = n;
```

A dendrogram O is an output of any hierarchical clustering. Figure 4-2 illustrates a dendrogram.

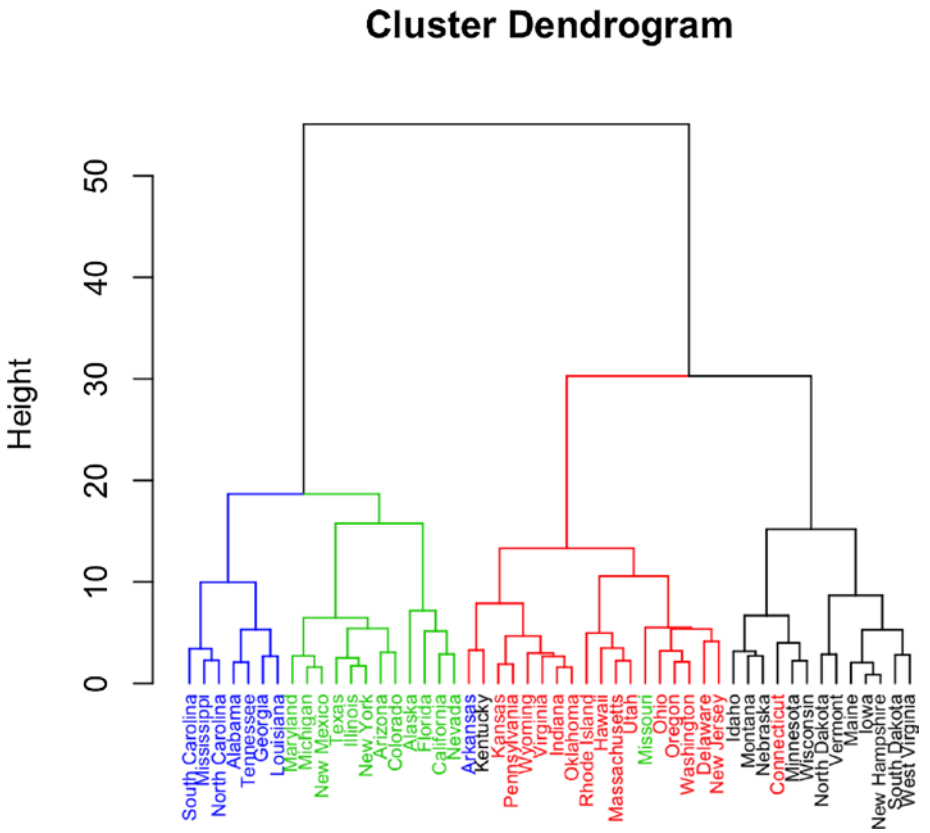


Figure 4-2. *A dendrogram*

To create a cluster from a dendrogram, you need a threshold of distance or similarity. An easy way to do this is to plot the distribution of distance or similarity and find the inflection point of the curve. For Gaussian distribution data, the inflection point is located at x = mean + n*std and x = mean – n*std, as shown Figure 4-3.

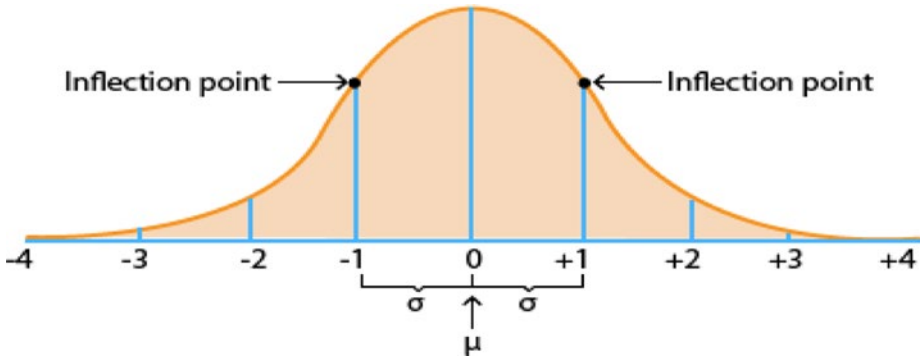


Figure 4-3. *The inflection point*

The following code creates a hierarchical cluster using Python:

```
From numpy import *
class cluster_node:
def \ __init__(self,vec1,left1=None,right1=None,distance1=0.0,i
d1=None,count1=1):
self.left1=left1
self.right1=right1
self.vec1=vec1
        self.id1=id1
self.distance1=distance1
self.count1=count1 #only used for weighted average
       def L2dist(v1,v2):
             return sqrt(sum((v1-v2)**2))
       def hcluster(features1,distanc1e=L2dist):
```

```
        #cluster the rows of the "features" matrix
        distances1={}
        currentclustid1=-1
        # clusters are initially just the individual rows
        clust1=[cluster_node(array(features1[i1]),id1=i1)
        for i1 in range(len(features1))]
        while len(clust1)>1:
              lowestpair1=(0,1)
    closest1=distance(clust1[0].vec1,clust1[1].vec1)
              # loop through every pair looking for the
              smallest distance
              for i1 in range(len(clust1)):
                    for j1 in range(i+1,len(clust)):
                          # distances is the cache of
                          distance calculations
                          if (clust1[i1].id1,clust1[j1].
                          id1) not in distances1:
                     distances[(clust1[i1].id1,clust1[j1].id1)]=\
    distance1(clust1[i1].vec1,clust1[j1].vec1)
    d1=distances1[(clust1[i1].id1,clust1[j1].id1)]
                           if d1< closest1:
               closest1=d1
               lowestpair1=(i1,j1)
               # calculate the average of the two clusters
    mergevec1=[(clust1[lowestpair1[10]].vec1[i1]\
    +clust1[lowestpair1[1]].vec1[i1])/2.0 \
               For i in range(len(clust1[0].vec1))]
               # create the new cluster
               newcluster1=cluster_node1(array(mergevec1),\
                           left1=clust1[lowestpair1[0]],\
right1=clust1[lowestpair1[1]],\
distance1=closes1t,id1=currentclustid1)
```

```
                # cluster ids that weren't in the original
                set are negative
                currentclustid1-=1
                delclust1[lowestpair1[1]]
                delclust1[lowestpair1[0]]
                clust1.append(newcluster1)
        return clust1[0]
```

The previous code will create the dendogram. Creation of the cluster from that dendogram using some threshold distance is given by the following:

```
def extract_clusters(clust1, dist1):
       # extract list of sub-tree clusters from h-cluster tree
       with distance <dist
       clusters1 = {}
       if clust.distance1<dis1:
       # we have found a cluster subtree
              return [clust1]
       else:
             # check the right and left branches
             cl1 = []
             cr1 = []
             if clust1.left1!=None:
                    cl = extract_clusters(clust1.left1,dist1=dist1)
             if clust1.right1!=None:
                    cr1 = extract_clusters(clust1.
                    right1,dist1=dist1)
             return cl1+cr1
```

Graph Theoretical Approach

The clustering problem can be mapped to a graph, where every node in the graph is an input data point. If the distance between two graphs is less than the threshold, then the corresponding nodes are connected. Now using the graph partition algorithm, you can cluster the graph. One industry example of clustering is in investment banking, where the cluster instruments depend on the correlation of their time series of price and performance trading of each cluster taken together. This is known as *basket trading* in algorithmic trading. So, by using the similarity measure, you can construct the graph where the nodes are instruments and the edges between the nodes indicate that the instruments are correlated. To create the basket, you need a set of instruments where all are correlated to each other. In a graph, this is a set of nodes or subgraphs where all the nodes in the subgraph are connected to each other. This kind of subgraph is known as a *clique*. Finding the clique of maximum size is an NP-complete problem. People use heuristic solutions to solve this problem of clustering.

How Do You Know If the Clustering Result Is Good?

After applying the clustering algorithm, verifying the result as good or bad is a crucial step in cluster analysis. Three parameters are used to measure the quality of cluster, namely, centroid, radius, and diameter.

$$\text{Centroid} = C_m = \frac{\sum_{i=1}^{N} t_{mi}}{N}$$

$$\text{Radius} = R_m = \sqrt{\frac{\sum_{i=1}^{N}(t_{mi} - C_m)^2}{N}}$$

$$\text{Diameter} = D_m = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}(t_{mi} - C_m)^2}{(N)(N-1)}}$$

If you consider the cluster as a circle in a space surrounding all member points in that cluster, then you can take the centroid as the center of the circle. Similarly, the radius and diameter of the cluster are the radius and diameter of the circle. Any cluster can be represented by using these three parameters. One measure of good clustering is that the distance between centers should be greater than the sum of radius.

General measures of the goodness of the machine learning algorithm are precision and recall. If A denotes the set of retrieved results, B denotes the set of relevant results, P denotes the precision, and R denotes the recall, then:

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

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

CHAPTER 5

Deep Learning and Neural Networks

Neural networks, specifically known as *artificial neural networks* (ANNs), were developed by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as follows:

"…a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Customarily, neutral networks are arranged in multiple layers. The layers consist of several interconnected nodes containing an activation function. The input layer, communicating to the hidden layers, delineates the patterns. The hidden layers are linked to an output layer.

Neural networks have many uses. As an example, you can cite the fact that in a passenger load prediction in the airline domain, passenger load in month t is heavily dependent on t-12 months of data rather on t-1 or t-2 data. Hence, the neural network normally produces a better result than the time-series model or even image classification. In a chatbot dialogue system, the memory network, which is actually a neural network of a bag of words of the previous conversation, is a popular approach. There are many ways to realize a neural network. In this book, I will focus only the backpropagation algorithm because it is the most popular.

S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1_5

Backpropagation

Backpropagation, which usually substitutes an optimization method like gradient descent, is a common method of training artificial neural networks. The method computes the error in the outermost layer and backpropagates up to the input layer and then updates the weights as a function of that error, input, and learning rate. The final result is to minimize the error as far as possible.

Backpropagation Approach

Problems like the noisy image to ASCII examples are challenging to solve through a computer because of the basic incompatibility between the machine and the problem. Nowadays, computer systems are customized to perform mathematical and logical functions at speeds that are beyond the capability of humans. Even the relatively unsophisticated desktop microcomputers, widely prevalent currently, can perform a massive number of numeric comparisons or combinations every second.

The problem lies in the inherent sequential nature of the computer. The "fetch-execute" cycle of the von Neumann architecture allows the machine to perform only one function at a time. In such cases, the time required by the computer to perform each instruction is so short that the average time required for even a large program is negligible to users.

A new processing system that can evaluate all the pixels in the image in parallel is referred to as the *backpropagation network* (BPN).

Generalized Delta Rule

I will now introduce the backpropagation learning procedure for knowing about internal representations. A neural network is termed a *mapping network* if it possesses the ability to compute certain functional relationships between its input and output.

Suppose a set of P vector pairs, $(x_1,y_1),(x_2,y_2),\cdots,(x_P,y_P)$, which are examples of the functional mapping $y=\phi(x): x\in R^N, y\in R^M$.

The equations for information processing in the three-layer network are shown next. An input vector is as follows:

$$net_{pj}^{h}=\sum_{i=1}^{N}\omega_{ji}^{h}x_{pi}+\theta_{j}^{h}$$

An output node is as follows:

$$i_{pj}=f_{j}^{h}\left(net_{pj}^{h}\right)$$

The equations for output nodes are as follows:

$$net_{pk}^{o}=\sum_{j=1}^{L}\omega_{kj}^{o}i_{pj}+\theta_{k}^{o}$$
$$o_{pk}=f_{k}^{o}\left(net_{pk}^{o}\right)$$

Update of Output Layer Weights

The following equation is the error term:

$$E_p=\frac{1}{2}\sum_{k}\left(y_{pk}-o_{pk}\right)^2$$
$$\frac{\partial E_p}{\partial\omega_{kj}^{o}}=-\left(y_{pk}-o_{pk}\right)\frac{\partial f_{k}^{o}}{\partial\left(net_{pk}^{o}\right)}\frac{\partial\left(net_{pk}^{o}\right)}{\partial\omega_{kj}^{o}}$$

The last factor in the equation of the output layer weight is as follows:

$$\frac{\partial\left(net_{pk}^{o}\right)}{\partial\omega_{kj}^{o}}=\left(\frac{\partial}{\partial\omega_{kj}^{o}}\sum_{j=1}^{L}\omega_{kj}^{o}i_{pj}+\theta_{k}^{o}\right)=i_{pj}$$

The negative gradient is as follows:

$$-\frac{\partial E_p}{\partial \omega_{kj}^o} = \left(y_{pk} - o_{pk}\right) f_k^{o'}\left(net_{pk}^o\right) i_{pj}$$

The weights on the output layer are updated according to the following:

$$\omega_{kj}^o\left(t+1\right) = \omega_{kj}^o\left(t\right) + \Delta_p \omega_{kj}^o\left(t\right)$$
$$\Delta_p \omega_{kj}^o = \eta\left(y_{pk} - o_{pk}\right) f_k^{o'}\left(net_{pk}^o\right) i_{pj}$$

There are two forms of the output function.

- $f_k^o\left(net_{jk}^o\right) = net_{jk}^o$
- $f_k^o\left(net_{jk}^o\right) = \left(1 + e^{-net_{jk}^o}\right)^{-1}$

Update of Hidden Layer Weights

Look closely at the network illustrated previously consisting of one layer of hidden neurons and one output neuron. When an input vector is circulated through the network, for the current set of weights there is an output prediction. Automatically, the total error somehow relates to the output values in the hidden layer. The equation is as follows:

$$\begin{aligned} E_p &= \frac{1}{2}\sum_k \left(y_{pk} - o_{pk}\right)^2 \\ &= \frac{1}{2}\sum_k \left(y_{pk} - f_k^o\left(net_{pk}^o\right)\right)^2 \\ &= \frac{1}{2}\sum_k \left(y_{pk} - f_k^o\left(\sum_j \omega_{kj}^o \boldsymbol{i}_{pj} + \theta_k^o\right)\right)^2 \end{aligned}$$

You can exploit the fact to calculate the gradient of E_p with respect to the hidden layer weights.

$$\frac{\partial E_p}{\partial \omega_{ji}^h} = \frac{1}{2}\sum_k \frac{\partial}{\partial \omega_{ji}^h}\left(y_{pk} - o_{pk}\right)^2$$

$$= -\sum_k \left(y_{pk} - o_{pk}\right)\frac{\partial o_{pk}}{\partial\left(net_{pk}^o\right)}\frac{\partial\left(net_{pk}^o\right)}{\partial i_{pj}}\frac{\partial i_{pj}}{\partial\left(net_{pj}^h\right)}\frac{\partial\left(net_{pj}^h\right)}{\partial \omega_{ji}^h}$$

Each of the factors in the equation can be calculated explicitly from the previous equation. The result is as follows:

$$\frac{\partial E_p}{\partial \omega_{ji}^h} = -\sum_k \left(y_{pk} - o_{pk}\right) f_k^{o'}\left(net_{pk}^o\right)\omega_{kj}^o f_j^{h'}\left(net_{pj}^h\right)x_{pi}$$

BPN Summary

Apply the input vector $X_p = \left(x_{p1}, x_{p2}, \ldots, x_{pN}\right)^t$ to the input units.

Calculate the net input values to the hidden layer units.

$$net_{pj}^h = \sum_{i=1}^{N} \omega_{ji}^h x_{pi} + \theta_j^h$$

Calculate the outputs from the hidden layer.

$$i_{pj} = f_j^h\left(net_{pj}^h\right)$$

Calculate the net input values to each unit.

$$net_{pk}^o = \sum_{j=1}^{L} \omega_{kj}^o i_{pj} + \theta_k^o$$

Calculate the outputs.

$$o_{pk} = f_k^o\left(net_{pk}^o\right)$$

Calculate the error terms for the output units.

$$\delta_{pk}^o = \left(y_{pk} - o_{pk}\right) f_k^{o'}\left(net_{pk}^o\right)$$

Calculate the error terms for the hidden units.

$$\delta_{pj}^h = f_j^{h'}\left(net_{pj}^h\right)\sum_k \delta_{pk}^o \omega_{kj}^o$$

Update weights on the output layer.

$$\omega_{kj}^o\left(t+1\right) = \omega_{kj}^o\left(t\right) + \eta\delta_{pk}^o i_{pj}$$

Update weights on the hidden layer.

$$\omega_{ji}^h\left(t+1\right) = \omega_{ji}^h\left(t\right) + \eta\delta_{pj}^h x_i$$

Backpropagation Algorithm

Let's see some code:

```
class NeuralNetwork(object):
       def  backpropagate(self,x,y):
              """Return a tuple "(nabla_b, nabla_w)"
              representing the
              gradient for the cost function C_x.  "nabla_b" and
              "nabla_w" are layer-by-layer lists of numpy
              arrays, similar
```

```
        to "self.biases" and "self.weights."""
        nabla_b1=[np.zeros(b1.shape)for b 1in self.biases]
        nabla_w1=[np.zeros(w1.shape)for w1in self.weights]

        # feedforward
        activation1=x1
        activations1=[x1]
        zs1=[]
        for b,winzip(self.biases,self.weights):
        z1=np.dot(w1,activation1)+b1
        zs1.append(z1)
        activation1=sigmoid(z1)
        activations1.append(activation1)
        # backward pass
        delta1=self.cost_derivative(activations1[-1],y1)* \
        sigmoid_prime(zs1[-1])
        nabla_b1[-1]=delta1
        nabla_w1[-1]=np.dot(delta,activations1[-2].
        transpose())

        for l in xrange(2,self.num_layers):
               z1=zs1[-l]
               sp1=sigmoid_prime(z1)
               delta1=np.dot(self.weights1[-l+1].
               transpose(),delta)*sp1
               nabla_b1[-l]=delta1
               nabla_w1[-l]=np.
               dot(delta,activations1[-l-1].transpose())
        return(nabla_b1,nabla_w1)

def  cost_derivative(self,output_activations,y):
        """Return the vector of partial derivatives \
        partial C_x /
```

```
            \partial a for the output activations."""
            return(output_activations1-y1)

    def sigmoid(z1):
            """The sigmoid function."""
            Return1.0/(1.0+np.exp(-z1))

    def sigmoid_prime(z1):
            """Derivative of the sigmoid function."""
            Return sigmoid(z)*(1-sigmoid(z1))
```

Other Algorithms

Many techniques are available to train neural networks besides backpropagation. One of the methods is to use common optimization algorithms such as gradient descent, Adam Optimizer, and so on. The simple perception method is also frequently applied. Hebb's postulate is another popular method. In Hebb's learning, instead of the error, the product of the input and output goes as the feedback to correct the weight.

$$w_{ij}(t+1) = w_{ij}(t) + \eta y_j(t) x_i(t)$$

TensorFlow

TensorFlow is a popular deep learning library in Python. It is a Python wrapper on the original library. It supports parallelism on the CUDA-based GPU platform. The following code is an example of simple linear regression with TensorFlow:

```
learning_rate = 0.0001

        y_t = tf.placeholder("float", [None,1])
        x_t = tf.placeholder("float", [None,X_train.shape[1]])
```

```
 W = tf.Variable(tf.random_normal([X_train.
 shape[1],1],stddev=.01))
 b = tf.constant(1.0)

 model = tf.matmul(x_t, W) + b
 cost_function = tf.reduce_sum(tf.pow((y_t - model),2))
 optimizer = tf.train.AdamOptimizer(learning_rate).
 minimize(cost_function)

 init = tf.initialize_all_variables()

with tf.Session() as sess:
        sess.run(init)
        w = W.eval(session = sess)
        of = b.eval(session = sess)
        print("Before Training ######################
        ########################")
        print(w,of)

        print("######################################
        ########################")
        step = 0
        previous = 0
        while(1):
                step = step + 1
                sess.run(optimizer, feed_dict={x_t: X_
                train.reshape(X_train.shape[0],X_train.
                shape[1]), y_t: y_train.reshape(y_
                train.shape[0],1)})
                cost = sess.run(cost_function, feed_
                dict={x_t: X_train.reshape(X_train.
                shape[0],X_train.shape[1]), y_t: y_
                train.reshape(y_train.shape[0],1)})
```

```
                    if step%1000 == 0:
                            print(cost)
                            if((previous- cost) < .0001):
                                    break
                            previous = cost
            w = W.eval(session = sess)
            of = b.eval(session = sess)
            print("Before Training ########################
            ##########################")
            print(w,of)

            print("#########################################
            ##########################")
```

With a little change, you can make it multilayer linear regression, as shown here:

```
learning_rate = 0.0001

        y_t = tf.placeholder("float", [None,1])

       if not multilayer:
              x_t = tf.placeholder("float", [None,X_train.
              shape[1]])
              W = tf.Variable(tf.random_normal([X_train.
              shape[1],1],stddev=.01))
              b = tf.constant(0.0)
              model = tf.matmul(x_t, W) + b
       else:
              x_t_user = tf.placeholder("float", [None,
              X_train_user.shape[1]])
              x_t_context = tf.placeholder("float", [None,
              X_train_context.shape[1]])
```

```
        W_user = tf.Variable(tf.random_normal([X_train_
        user.shape[1],1],stddev=.01))
        W_context = tf.Variable(tf.random_normal([X_
        train_context.shape[1],1],stddev=.01))
        W_out_user = tf.Variable(tf.random_
        normal([1,1],stddev=.01))
        W_out_context = tf.Variable(tf.random_
        normal([1,1],stddev=.01))
        model  = tf.add(tf.matmul(tf.matmul(x_t_user,
        W_user),W_out_user),tf.matmul(tf.matmul(x_t_
        context, W_context),W_out_context))

cost_function = tf.reduce_sum(tf.pow((y_t - model),2))
optimizer = tf.train.AdamOptimizer(learning_rate).
minimize(cost_function)

init = tf.initialize_all_variables()

with tf.Session() as sess:
        sess.run(init)
        print("Before Training ########################
        ########################")
        step = 0
        previous = 0
     cost = 0
     while(1):
                step = step + 1
            if not multilayer:
                        sess.run(optimizer, feed_
                        dict={x_t: X_train.reshape
                        (X_train.shape[0],X_train.
                        shape[1]), y_t: y_train.
                        reshape(y_train.shape[0],1)})
```

```
                cost = sess.run(cost_function,
                feed_dict={x_t: X_train.reshape
                (X_train.shape[0],X_train.
                shape[1]), y_t: y_train.reshape
                (y_train.shape[0],1)})
        else:
                  sess.run(optimizer, feed_
                  dict={x_t_user: X_train_
                  user.reshape(X_train_user.
                  shape[0],X_train_user.shape[1]),
                  x_t_context: X_train_context.
                  reshape(X_train_context.
                  shape[0],X_train_context.
                  shape[1]), y_t: y_train.
                  reshape(y_train.shape[0],1)})
                cost = sess.run(cost_function,
                feed_dict={x_t_user: X_train_user.
                reshape(X_train_user.shape[0],X_
                train_user.shape[1]), x_t_context:
                X_train_context.reshape(X_train_
                context.shape[0],X_train_context.
                shape[1]), y_t: y_train.reshape(y_
                train.shape[0],1)})
            if step%1000 == 0:
                    print(cost)
            if previous == cost or step > 50000:
               break
        if cost != cost :
               raise Exception("NaN value")
            previous = cost
```

```
            print("#####################################
            ############################")
        if multilayer:
              w_user = W_user.eval(session = sess)
              w_context = W_context.eval(session = sess)
              w_out_context = W_out_context.eval(session
              = sess)
              w_out_user = W_out_user.eval(session = sess)
              w_user = np.dot(w_user, w_out_user)
              w_context = np.dot(w_context, w_out_context)
        else:
              w = W.eval(session = sess)
```

You can do logistic regresson with the same code with a little change, as shown here:

```
learning_rate = 0.001
      no_of_level = 2

      y_t = tf.placeholder("float", [None,no_of_level])
      if True:
            x_t = tf.placeholder("float", [None,X_train.
            shape[1]])
             W = tf.Variable(tf.random_normal([X_train.
             shape[1],no_of_level],stddev=.01))
            model = tf.nn.softmax(tf.matmul(x_t, W))

       cost_function = tf.reduce_mean(-tf.reduce_sum(y_t*tf.
       log(model), reduction_indices=1))
       optimizer = tf.train.GradientDescentOptimizer(learni
       ng_rate).minimize(cost_function)

       init = tf.initialize_all_variables()
```

```
with tf.Session() as sess:
          sess.run(init)
          print("Before Training ########################
          ##########################")
          step = 0
          previous = 0
        cost = 0
        while(1):
                  step = step + 1
             if True:
                       sess.run(optimizer, feed_
                       dict={x_t: X_train.reshape
                       (X_train.shape[0],X_train.
                       shape[1]), y_t: y_train.
                       reshape(y_train.shape[0],
                       no_of_level)})
                     cost = sess.run(cost_function,
                     feed_dict={x_t: X_train.reshape(X_
                     train.shape[0],X_train.shape[1]),
                     y_t: y_train.reshape(y_train.
                     shape[0],no_of_level)})
                  if step%1000 == 0:
                          print(cost)
                  if previous == cost or step > 50000:
                     break
             if cost != cost :
                    raise Exception("NaN value")
                  previous = cost

          print("#####################################
          ##########################")
        if True:
             w = W.eval(session = sess)
```

Recurrent Neural Network

A recurrent neural network is an extremely popular kind of network where the output of the previous step goes to the feedback or is input to the hidden layer. It is an extremely useful solution for a problem like a sequence leveling algorithm or time-series prediction. One of the more popular applications of the sequence leveling algorithm is in an autocomplete feature of a search engine.

As an example, say one algorithmic trader wants to predict the price of a stock for trading. But his strategy requires the following criteria for prediction:

a) The predicted tick is higher than the current tick and the next tick. Win.

b) The predicted tick is lower than the current tick and the next tick. Win.

c) The predicted tick is higher than the current tick but lower than the next tick. Loss.

d) The predicted tick is lower than the current tick but higher than the next tick. Loss.

To satisfy his criteria, the developer takes the following strategy.

For generating predictions for 100 records, he is considering preceding 1,000 records as input 1, prediction errors in the last 1,000 records as input 2, and differences between two consecutive records as input 3. Using these inputs, an RNN-based engine predicts results, errors, and inter-record differences for the next 100 records.

Then he takes the following strategy:

> If predicted diff > 1 and predicted err < 1, then prediction += pred_err + 1.
>
> If predicted diff < 1 and predicted err > 1, then prediction -= pred_err -1.

In this way, prediction satisfies the developer need. The detailed code is shown next. It is using Keras, which is a wrapper above TensorFlow.

```
import matplotlib.pyplot as plt
import numpy as np
import time
import csv
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import sys
np.random.seed(1234)

def read_data(path_to_dataset,
                            sequence_length=50,
                            ratio=1.0):

    max_values = ratio * 2049280

    with open(path_to_dataset) as f:
        data = csv.reader(f, delimiter=",")
        power = []
        nb_of_values = 0
        for line in data:
            #print(line)
            #if nb_of_values == 3500:
             #   break
            try:
                power.append(float(line[1]))
                nb_of_values += 1
            except ValueError:
                pass
```

```
            # 2049280.0 is the total number of valid values,
            i.e. ratio = 1.0
            if nb_of_values >= max_values:
                break
    return power

def process_data(power, sequence_length, ratio, error):
    #print("Data loaded from csv. Formatting...")
    #fig = plt.figure()
    #plt.plot(power)
    #plt.show()
    result = []
    for index in range(len(power) - sequence_length):
        result.append(power[index: index + sequence_length])
    result = np.array(result)  # shape (2049230, 50)

    if not error:
       global result_mean, result_std
       result_mean = result.mean()
       result_std = result.std()
       result -= result_mean
       result /= result_std
    #result = np.log(result+1)
    #print result
    #exit(0)
#     print ("Shift : ", result_mean)
    print ("Data  : ", result.shape)

    row = int(round(0.9 * result.shape[0]))
    print row
    train = result[:row, :]
    np.random.shuffle(train)
    X_train = train[:, :-1]
```

```
    y_train = train[:, -1]
    X_test = result[row:, :-1]
    y_test = result[row:, -1]

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.
    shape[1], 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.
    shape[1], 1))

    return [X_train, y_train, X_test, y_test]

def build_model():
    model = Sequential()
    layers = [1, 50, 100, 1]

    model.add(LSTM(
        layers[1],
        input_shape=(None, layers[0]),
        return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
        layers[3]))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop")
    print ("Compilation Time : ", time.time() - start)
    return model
```

```
def run_network(model=None, data=None, error=False):
    global_start_time = time.time()
    epochs = 2
    ratio = 0.5
    sequence_length = 100

    X_train, y_train, X_test, y_test = process_data(
            data, sequence_length, ratio,error)

    print ('\nData Loaded. Compiling...\n')


    if model is None:
        model = build_model()

    try:
        model.fit(
            X_train, y_train,
            batch_size=512, nb_epoch=epochs, validation_
            split=0.05)
        predicted = model.predict(X_test)
        predicted = np.reshape(predicted, (predicted.size,))
    except KeyboardInterrupt:
        print ('Training duration (s) : ', time.time() -
        global_start_time)
        return model, y_test, 0

    try:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(y_test[:100]*result_max)
        plt.plot(predicted[:100]*result_max)
        plt.show()
```

```
    except Exception as e:
        print (str(e))
    print ('Training duration (s) : ', time.time() - global_
    start_time)

    return model, y_test, predicted

if __name__ == '__main__':
    path_to_dataset = '20170301_ltp.csv'
    data = read_data(path_to_dataset)
    error = []
    diff_predicted = []
    err_predicted = []
    print len(data)
    for i in range(0,len(data)-1000,89):
        d = data[i:i+1000]
        model, y_test, predicted = run_network(None,d, False)
        if i > 11 and len(error) >= 1000:
             model,err_test, err_predicted =
             run_network(None,error, True)
              error = error[90:]
              d1 = data[i:i+1001]
              diff = [0]*1000
              for k in range(1000):
                    diff[k] = d1[k+1] - d1[k]
                model,diff_test, diff_predicted =
                run_network(None,diff, True)
        print i,len(d), len(y_test)
        y_test *= result_std
        predicted *= result_std
        y_test += result_mean
        predicted += result_mean
```

```
e = (y_test - predicted)/predicted
error = np.concatenate([error, e])
 #print error
 #error.append(y_test - predicted)
if i > 11 and len(error) >= 1000 and len(err_
predicted)>=90:
        for j in range(len(y_test)-1):
              if diff_predicted[j] > 1 and err_
              predicted[j]*predicted[j] <= 1:
                    predicted[j] += abs(err_
                    predicted[j]*predicted[j]) + 1
             if diff_predicted[j] <= 1 and err_
             predicted[j]*predicted[j] > 1:
                          predicted[j] -= abs(err_
                          predicted[j]*predicted[j]) - 1
              print y_test[j], ',',predicted[j]
print "length of error",len(error)
```

CHAPTER 6

Time Series

A *time series* is a series of data points arranged chronologically. Most commonly, the time points are equally spaced. A few examples are the passenger loads of an airline recorded each month for the past two years or the price of an instrument in the share market recorded each day for the last year. The primary aim of time-series analysis is to predict the future value of a parameter based on its past data.

Classification of Variation

Traditionally time-series analysis divides the variation into three major components, namely, trends, seasonal variations, and other cyclic changes. The variation that remains is attributed to "irregular" fluctuations or error term. This approach is particularly valuable when the variation is mostly comprised of trends and seasonality.

Analyzing a Series Containing a Trend

A *trend* is a change in the mean level that is long-term in nature. For example, if you have a series like 2, 4, 6, 8 and someone asks you for the next value, the obvious answer is 10. You can justify your answer by fitting a line to the data using the simple least square estimation or any other regression method. A trend can also be nonlinear. Figure 6-1 shows an example of a time series with trends.

S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1_6

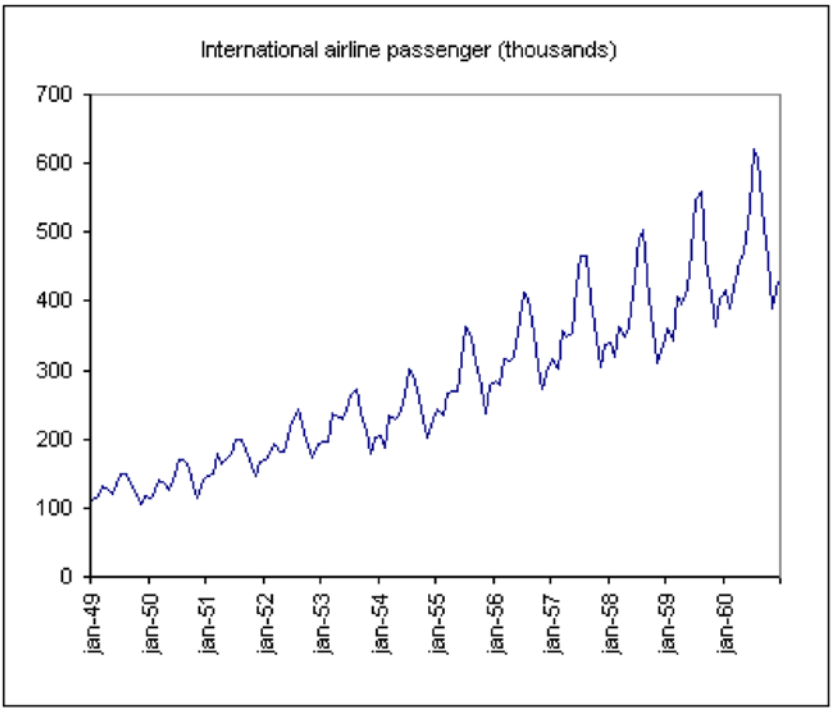


Figure 6-1. *A time series with trends*

The simplest type of time series is the familiar "linear trend plus noise" for which the observation at time t is a random variable X_t, as follows:

$$X_t = \alpha + \beta t + \varepsilon_t$$

Here, α,β are constants, and ε_t denotes a random error term with a mean of 0. The average level at time t is given by $m_t = (\alpha + \beta t)$. This is sometimes called the *trend term*.

Curve Fitting

Fitting a simple function of time such as a polynomial curve (linear, quadratic, etc.), a Gompertz curve, or a logistic curve is a well-known

method of dealing with nonseasonal data that contains a trend, particularly yearly data. The global linear trend is the simplest type of polynomial curve. The Gompertz curve can be written in the following format, where α, β, and γ are parameters with $0 < r < 1$:

$$x_t = \alpha \exp [\beta \exp(-\gamma t)]$$

This looks quite different but is actually equivalent, provided $\gamma > 0$. The logistic curve is as follows:

$$x_t = a / (1+be^{-ct})$$

Both these curves are S-shaped and approach an asymptotic value as $t \to \infty$, with the Gompertz curve generally converging slower than the logistic one. Fitting the curves to data may lead to nonlinear simultaneous equations.

For all curves of this nature, the fitted function provides a measure of the trend, and the residuals provide an estimate of local fluctuations where the residuals are the differences between the observations and the corresponding values of the fitted curve.

Removing Trends from a Time Series

Differentiating a given time series until it becomes stationary is a special type of filtering that is particularly useful for removing a trend. You will see that this is an integral part of the Box-Jenkins procedure. For data with a linear trend, a first-order differencing is usually enough to remove the trend.

Mathematically, it looks like this:

$$y(t) = a*t + c$$
$$y(t+1) = a*(t+1) + c$$
$$z(t) = y(t+1) - y(t) = a + c \text{ ; no trend present in } z(t)$$

A trend can be exponential as well. In this case, you will have to do a logarithmic transformation to convert the trend from exponential to linear.

Mathematically, it looks like this:

$$y(t) = a*\exp(t)$$
$$z(t) = \log(y(t)) = t*\log(a);\ z(t) \text{ is a linear function of } t$$

Analyzing a Series Containing Seasonality

Many time series, such as airline passenger loads or weather readings, display variations that repeat after a specific time period. For instance, in India, there will always be an increase in airline passenger loads during the holiday of Diwali. This yearly variation is easy to understand and can be estimated if seasonality is of direct interest. Similarly, like trends, if you have a series such as 1, 2, 1, 2, 1, 2, your obvious choices for the next values of the series will be 1 and 2.

The Holt-Winters model is a popular model to realize time series with seasonality and is also known as *exponential smoothing*. The Holt-Winters model has two variations: additive and multiplicative. In the additive model with a single exponential smoothing time series, seasonality is realized as follows:

$$X(t+1) = \alpha * Xt + (1 - \alpha) * St-1$$

In this model, every point is realized as a weighted average of the previous point and seasonality. So, X(t+1) will be calculated as a function X(t-1) and S(t-2) and square of α. In this way, the more you go on, the α value increases exponentially. This is why it is known as exponential smoothing. The starting value of St is crucial in this method. Commonly, this value starts with a 1 or with an average of the first four observations.

The multiplicative seasonal model time series is as follows:

$$X(t+1)= (b1 + b2*t)St + \text{noise},$$

Here, b1, often referred to as the *permanent component*, is the initial weight of the seasonality; b2 represents the trend, which is linear in this case.

However, there is no standard implementation of the Holt-Winters model in Python. It is available in R (see Chapter 1 for how R's Holt-Winters model can be called from Python code).

Removing Seasonality from a Time Series

There are two ways of removing seasonality from a time series.

- By filtering
- By differencing

By Filtering

The series $\{x_t\}$ is converted into another called $\{y_t\}$ with the linear operation shown here, where $\{a_r\}$ is a set of weights:

$$Y_t = \sum^{+s}_{r=-q} a_r x_{t+r}$$

To smooth out local fluctuations and estimate the local mean, you should clearly choose the weights so that $\sum a_r = 1$; then the operation is often referred to as a *moving average*. They are often symmetric with $s = q$ and $a_j = a_{-j}$. The simplest example of a symmetric smoothing filter is the simple moving average, for which $a_r = 1 / (2q+1)$ for $r = -q, \ldots, + q$.

The smoothed value of x_t is given by the following:

$$Sm(x_t) = \frac{1}{2q+1} \sum^{+q}_{r=-q} x_{t+r}$$

The simple moving average is useful for removing seasonal variations, but it is unable to deal well with trends.

By Differencing

Differencing is widely used and often works well. Seasonal differencing removes seasonal variation.

Mathematically, if time series y(t) contains additive seasonality S(t) with time period T, then:

$$y(t) = a*S(t) + b*t + c$$
$$y(t+T) = aS(t+T) + b*(t+T) + c$$
$$z(t) = y(t+T) - y(t) = b*T + \text{noise term}$$

Similar to trends, you can convert the multiplicative seasonality to additive by log transformation.

Now, finding time period T in a time series is the critical part. It can be done in two ways, either by using an autocorrelation function in the time domain or by using the Fourier transform in the frequency domain. In both cases, you will see a spike in the plot. For autocorrelation, the plot spike will be at lag T, whereas for FT distribution, the spike will be at frequency 1/T.

Transformation

Up to now I have discussed the various kinds of transformation in a time series. The three main reasons for making a transformation are covered in the next sections.

To Stabilize the Variance

The standard way to do this is to take a logarithmic transformation of the series; it brings closer the points in space that are widely scattered.

To Make the Seasonal Effect Additive

If the series has a trend and the volume of the seasonal effect appears to be on the rise with the mean, then it may be advisable to modify the data so as to make the seasonal effect constant from year to year. This seasonal effect is said to be additive. However, if the volume of the seasonal effect is directly proportional to the mean, then the seasonal effect is said to be multiplicative, and a logarithmic transformation is needed to make it additive again.

To Make the Data Distribution Normal

In most probability models, it is assumed that distribution of data is Gaussian or normal. For example, there can be evidence of skewness in a trend that causes "spikes" in the time plot that are all in the same direction.

To transform the data in a normal distribution, the most common transform is to subtract the mean and then divide by the standard deviation. I gave an example of this transformation in the RNN example in Chapter 5; I'll give another in the final example of the current chapter. The logic behind this transformation is it makes the mean 0 and the standard deviation 1, which is a characteristic of a normal distribution. Another popular transformation is to use the logarithm. The major advantage of a logarithm is it reduces the variation and logarithm of Gaussian distribution data that is also Gaussian. Transformation may be problem-specific or domain-specific. For instance, in a time series of an airline's passenger load data, the series can be normalized by dividing by the number of days in the month or by the number of holidays in a month.

Cyclic Variation

In some time series, seasonality is not a constant but a stochastic variable. That is known as *cyclic variation*. In this case, the periodicity first has to be predicted and then has to be removed in the same way as done for seasonal variation.

Irregular Fluctuations

A time series without trends and cyclic variations can be realized as a weekly stationary time series. In the next section, you will examine various probabilistic models to realize weekly time series.

Stationary Time Series

Normally, a time series is said to be stationary if there is no systematic change in mean and variance and if strictly periodic variations have been done away with. In real life, there are no stationary time series. Whatever data you receive by using transformations, you may try to make it somehow nearer to a stationary series.

Stationary Process

A time series is strictly stationary if the joint distribution of $X(t_1),\ldots,X(t_k)$ is the same as the joint distribution of $X(t_1 + \tau),\ldots,X(t_k + \tau)$ for all $t_1,\ldots,t_k,\tau$. If k =1, strict stationary implies that the distribution of X(t) is the same for all t, so provided the first two moments are finite, you have the following:

$$\mu(t) = \mu$$

$$\sigma^2(t) = \sigma^2$$

They are both constants, which do not depend on the value of t.

A weekly stationary time series is a stochastic process where the mean is constant and autocovariance is a function of time lag.

Autocorrelation and the Correlogram

Quantities called *sample autocorrelation coefficients* act as an important guide to the properties of a time series. They evaluate the correlation, if any, between observations at different distances apart and provide valuable descriptive information. You will see that they are also an important tool in model building and often provide valuable clues for a suitable probability model for a given set of data. The quantity lies in the range [-1,1] and measures the forcefulness of the linear association between the two variables. It can be easily shown that the value does not depend on the units in which the two variables are measured; if the variables are independent, then the ideal correlation is zero.

A helpful supplement in interpreting a set of autocorrelation coefficients is a graph called a *correlogram*. The correlogram may be alternatively called the *sample autocorrelation function*.

Suppose a stationary stochastic process X(t) has a mean μ, variance σ^2, auto covariance function (acv.f.) $\gamma(t)$, and auto correlation function (ac.f.) $\rho(\tau)$.

$$\rho(\tau) = \frac{\gamma(\tau)}{\gamma(0)} = \gamma(\tau)/\sigma^2$$

Estimating Autocovariance and Autocorrelation Functions

In the stochastic process, the autocovariance is the covariance of the process with itself at pairs of time points. Autocovariance is calculated as follows:

$$\gamma(h) = \frac{1}{n}\sum_{t=1}^{n-|h|}\left(x_{t+|h|} - \bar{x}\right)\left(x_t - \bar{x}\right), \quad -n < h < n$$

Figure 6-2 shows a sample autocorrelation distribution.

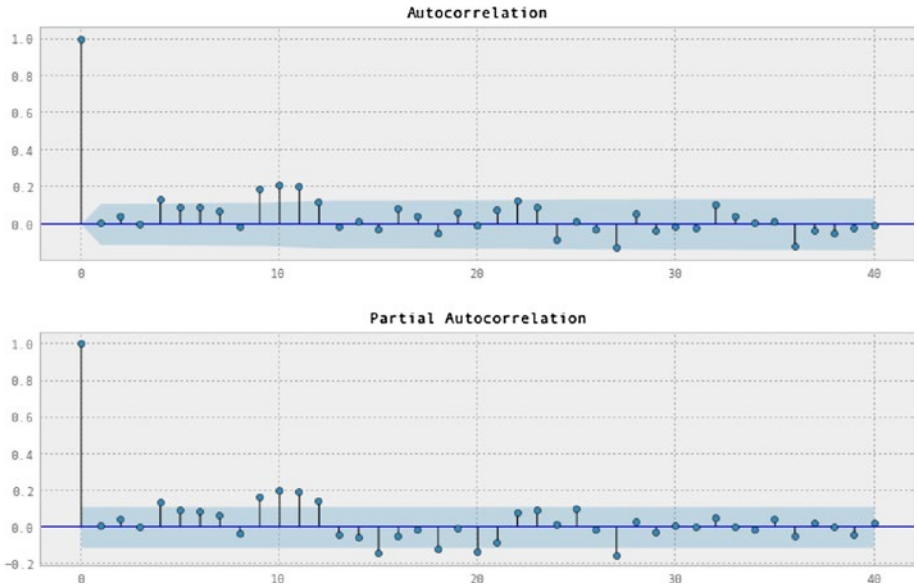


Figure 6-2. *Sample autocorrelations*

Time-Series Analysis with Python

A complement to SciPy for statistical computations including descriptive statistics and estimation of statistical models is provided by Statsmodels, which is a Python package. Besides the early models, linear regression, robust linear models, generalized linear models, and models for discrete data, the latest release of scikits.statsmodels includes some basic tools and models for time-series analysis, such as descriptive statistics, statistical tests, and several linear model classes. The linear model classes include autoregressive (AR), autoregressive moving-average (ARMA), and vector autoregressive models (VAR).

Useful Methods

Let's start with a moving average.

Moving Average Process

Suppose that $\{Z_t\}$ is a purely random process with mean 0 and variance σ_z^2. Then a process $\{X_t\}$ is said to be a moving average process of order q.

$$X_t = \beta_0 Z_t + \beta_1 Z_{t-1} + \cdots + \beta_q Z_{t-q}$$

Here, $\{\beta_i\}$ are constants. The Zs are usually scaled so that $\beta_0 = 1$.

$$E(X_t) = 0$$

$$\text{Var}(X_t) = \sigma_Z^2 \sum_{i=0}^{q} \beta_i^2$$

The Zs are independent.

$$\begin{aligned}\gamma(k) &= \text{Cov}(X_t, X_{t+k}) \\ &= \text{Cov}\left(\beta_0 Z_t + \cdots + \beta_q Z_{t-q},\ \beta_0 Z_{t+k} + \cdots + \beta_q Z_{t+k-q}\right) \\ &= \begin{cases} 0 & k > q \\ \sigma_Z^2 \sum_{i=0}^{q-k} \beta_i \beta_{i+k} & k = 0,1,\ldots,q \\ \gamma(-k) & k < 0 \end{cases}\end{aligned}$$

$$\text{Cov}(Z_s, Z_t) = \begin{cases} \sigma_Z^2 & s = t \\ 0 & s \neq t \end{cases}$$

As $\gamma(k)$ is not dependent on t and the mean is constant, the process is second-order stationary for all values of $\{\beta_i\}$.

$$\rho(k)=\begin{cases}1 & k=0\\ \sum_{i=0}^{q-k}\beta_i\beta_{i+k}\Big/\sum_{i=0}^{q}\beta_i^2 & k=1,\ldots,q\\ 0 & k>q\\ \rho(-k) & k<0\end{cases}$$

Fitting Moving Average Process

The moving-average (MA) model is a well-known approach for realizing a single-variable weekly stationary time series (see Figure 6-3). The moving-average model specifies that the output variable is linearly dependant on its own previous error terms as well as on a stochastic term. The AR model is called the Moving-Average model, which is a special case and a key component of the ARMA and ARIMA models of time series.

$$X_t=\varepsilon_t+\sum_{i=1}^{p}\varphi_i X_{t-i}+\sum_{i=1}^{q}\theta_i\varepsilon_{t-i}+\sum_{i=1}^{b}\eta_i d_{t-i}$$

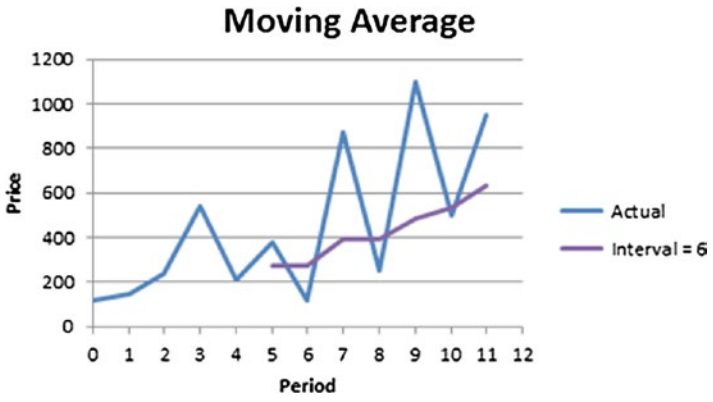


Figure 6-3. *Example of moving average*

Here's the example code for a moving average model:

```
import  numpy as np

def running_mean(l, N):
    # Also works for the(strictly invalid) cases when N is even.
    if (N//2)*2 == N:
        N = N - 1
    front = np.zeros(N//2)
    back = np.zeros(N//2)

    for i in range(1, (N//2)*2, 2):
        front[i//2] = np.convolve(l[:i], np.ones((i,))/i, mode
        = 'valid')
    for i in range(1, (N//2)*2, 2):
        back[i//2] = np.convolve(l[-i:], np.ones((i,))/i, mode
        = 'valid')
    return np.concatenate([front, np.convolve(l,
    np.ones((N,))/N, mode = 'valid'), back[::-1]])

print running_mean(2,21)
```

Autoregressive Processes

Suppose $\{Z_t\}$ is a purely random process with mean 0 and variance σ_z^2. After that, a process $\{X_t\}$ is said to be of autoregressive process of order p if you have this:

$$x_t = \alpha_1 x_{t-1} + \ldots \alpha_p x_{t-p} + z_t, \; or$$

$$x_t = \sum_{i=1}^{p} \alpha_i x_{i-1} + z_t$$

The autocovariance function is given by the following:

$$\gamma(k) = \frac{\alpha^k \sigma_z^2}{\left(1-\alpha^2\right)}, k = 0,1,2\ldots, \text{ hence}$$

$$\rho_k = \frac{\gamma(k)}{\gamma(0)} = \alpha^k$$

Figure 6-4 shows a time series and its autocorrelation plot of the AR model.

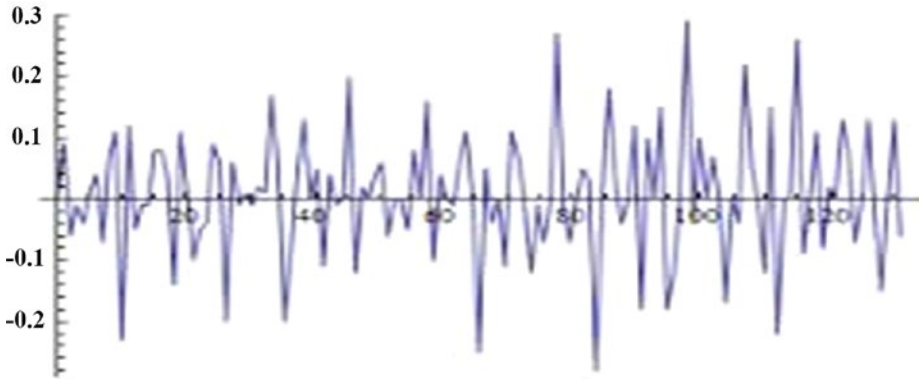


Figure 6-4. *A time series and AR model*

Estimating Parameters of an AR Process

A process is called *weakly stationary* if its mean is constant and the autocovariance function depends only on time lag. There is no weakly stationary process, but it is imposed on time-series data to do some stochastic analysis. Suppose Z(t) is a weak stationary process with mean 0 and constant variance. Then X(t) is an autoregressive process of order p if you have the following:

X(t) = a1 x X(t-1) + a2 x X(t-2) + … + ap x X(t-p) +Z(t), where a ∈ R and p ∈ I

Now, E[X(t)] is the expected value of X(t).

$$\text{Covariance}(X(t),X(t+h)) = E[(X(t) - E[X(t)]) * (X(t+h) - E[X(t+h)])]$$

$$= E[(X(t) - m) * (X(t+h) - m)]$$

If X(t) is a weak stationary process, then:

$$E[X(t)] = E[X(t+h)] = m \text{ (constant)}$$

$$= E[X(t) * X(t+h)] - m^2 = c(h)$$

Here, m is constant, and cov[X(t),X(t+h)] is the function of only h(c(h)) for the weakly stationary process. c(h) is known as autocovariance.

Similarly, the correlation (X(t),X(t+h) = ρ(h) = r(h) = c(h) = ÷ c(0) is known as autocorrelation.

If X(t) is a stationary process that is realized as an autoregressive model, then:

$$X(t) = a1 * X(t-1) + a2 * X(t-2) + ... + ap * X(t-p) + Z(t)$$

Correlation(X(t),X(t)) = a1 * correlation (X(t),X(t-1)) + ... + ap * correlation (X(t),X(t-p))+0

As covariance, (X(t),X(t+h)) is dependent only on h, so:

$$r0 = a1 * r1 + a2 * r2 + ... + ap * rp$$

$$r1 = a1 * r0 + a2 * r1 + ... + ap * r(p-1)$$

So, for an n-order model, you can easily generate the n equation and from there find the n coefficient by solving the n equation system.

In this case, realize the data sets only in the first-order and second-order autoregressive model and choose the model whose mean of residual is less. For that, the reduced formulae are as follows:

- *First order*: a1 = r1
- *Second order*: $a1 = r_1(1 - r_2) \div (1 - r_1^2), a2 = (r_2 - r_1^2) \div (1 - r_1^2)$

Here is some example code for an autoregressive model:

```
from pandas import Series
from matplotlib import pyplot
from statsmodels.tsa.ar_model import AR
from sklearn.metrics import mean_squared_error

series = series.from_csv('input.csv', header=0)

J = series.value
train, test = J[1:len(J)-10], J[len(J)-10:]

model = AR(train)
model_fit = model.fit()
print('Lag: %s' % model_fit.k_ar)
print('Coefficients: %s' % model_fit.params)

predictions = model_fit.predict(start=len(train),
end=len(train)+len(test)-1, dynamic=False)
for t in range(len(predictions)):
        print('predicted=%f, expected=%f' % (predictions[t],
test[t]))
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)

pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```

Mixed ARMA Models

Mixed ARMA models are a combination of MA and AR processes. A mixed autoregressive/moving average process containing p AR terms and q MA terms is said to be an ARMA process of order (p,q). It is given by the following:

$$X_t = \alpha_1 X_{t-1} + \cdots + \alpha_p X_{t-p} + Z_t + \beta_1 Z_{t-1} + \cdots + \beta_q Z_{t-q}$$

The following example code was taken from the stat model site to realize time-series data as an ARMA model:

```
r1,q1,p1 = sm.tsa.acf(resid.values.squeeze(), qstat=True)
data1 = np.c_[range(1,40), r1[1:], q1, p1]
table1 = pandas.DataFrame(data1, columns=['lag', "AC", "Q",
"Prob(>Q)"])
predict_sunspots1 = arma_mod40.predict('startyear', 'endyear',
dynamic=True)
```

Here is the simulated ARMA (4,1) model identification code:

```
from statsmodels. import tsa.arima_processimportarma_generate_
sample, ArmaProcess

np.random.seed(1234)
data = np.array([1, .85, -.43, -.63, .8])
parameter = np.array([1, .41]
model = ArmaProcess(data, parameter)

model.isinvertible()

True

Model.isstationary()

True
```

Here is how to estimate parameters of an ARMA model:

1. After specifying the order of a stationary ARMA process, you need to estimate the parameters.
2. Assume the following:
   - The model order (p and q) is known.
   - The data has zero mean.
3. If step 2 is not a reasonable assumption, you can subtract the sample mean Y and fit a 0 mean ARMA model, as in $\varnothing(B)X_t = \theta(B)a_t$ where $X_t = Y_t - Y$. Then use $X_t + Y$ as the model for Y_t.

Integrated ARMA Models

To fit a stationary model such as the one discussed earlier, it is imperative to remove nonstationary sources of variation. Differencing is widely used for econometric data. If X_t is replaced by $\nabla^d X_t$, then you have a model capable of describing certain types of nonstationary series.

$$Y_t = (1-L)^d X_t$$

These are the estimating parameters of an ARIMA model:

- ARIMA models are designated by the level of autoregression, integration, and moving averages.
- This does not assume any pattern uses an iterative approach of identifying a model.
- The model "fits" if residuals are generally small, randomly distributed, and, in general, contain no useful information.

Here is the example code for an ARIMA model:

```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error

def parser(p):
        return datetime.strptime('190'+p, '%Y-%m')

series = read_csv('input.csv', header=0, parse_dates=[0],
index_col=0, squeeze=True, date_parser=parser)
P = series.values
size = int(len(P) * 0.66)
train, test = P[0:size], P[size:len(P)]
history = [p for p in train]
predictions = list()
for t in range(len(test)):
        model = ARIMA(history, order=(5,1,0))
        model_fit = model.fit(disp=0)
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat)
        obs = test[t]
        history.append(obs)
        print('predicted=%f, expected=%f' % (yhat, obs))
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
# plot
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```

The Fourier Transform

The representation of nonperiodic signals by everlasting exponential signals can be accomplished by a simple limiting process, and I will illustrate that nonperiodic signals can be expressed as a continuous sum (integral) of everlasting exponential signals. Say you want to represent the nonperiodic signal g(t). Realizing any nonperiodic signal as a periodic signal with an infinite time period, you get the following:

$$g(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty}G(\omega)e^{j\omega t}\ d\omega$$

$$G(n\Delta\omega)=\lim_{T\to\infty}\int_{-T_0/2}^{T_0/2}g_p(t)e^{-jn\Delta\omega t}dt$$

$$=\int_{-\infty}^{\infty}g(t)e^{-jn\Delta\omega t}dt$$

Hence:

$$G(\omega)=\int_{-\infty}^{\infty}g(t)e^{-j\omega t}dt$$

G(w) is known as a Fourier transform of g(t).

Here is the relation between autocovariance and the Fourier transform:

$$\gamma(0)=\sigma_X^2=\int_0^{\pi}dF(\omega)=F(\pi)$$

An Exceptional Scenario

In the airline or hotel domain, the passenger load of month t is less correlated with data of t-1 or t-2 month, but it is more correlated for t-12 month. For example, the passenger load in the month of Diwali (October) is more correlated with last year's Diwali data than with the same year's August and September data. Historically, the pick-up model is used to predict this kind of data. The pick-up model has two variations.

In the additive pick-up model,

$$X(t) = X(t-1) + [X(t-12) - X(t-13)]$$

In the multiplicative pick-up model,

$$X(t) = X(t-1) * [X(t-12) / X(t-13)]$$

Studies have shown that for this kind of data the neural network–based predictor gives more accuracy than the time-series model.

In high-frequency trading in investment banking, time-series models are too time-consuming to capture the latest pattern of the instrument. So, they on the fly calculate dX/dt and d2X/dt2, where X is the price of the instruments. If both are positive, they blindly send an order to buy the instrument. If both are negative, they blindly sell the instrument if they have it in their portfolio. But if they have an opposite sign, then they do a more detailed analysis using the time series data.

As I stated earlier, there are many scenarios in time-series analysis where R is a better choice than Python. So, here is an example of time-series forecasting using R. The beauty of the auto.arima model is that it automatically finds the order, trends, and seasonality of the data and fits the model. In the forecast, we are printing only the mean value, but the

model provides the upper limit and the lower limit of the prediction in forecasting.

```
asm_weekwise<-read.csv("F:/souravda/New ASM Weekwise.
csv",header=TRUE)

asm_weekwise$Week <- NULL

library(MASS, lib.loc="F:/souravda/lib/")
library(tseries, lib.loc="F:/souravda/lib/")
library(forecast, lib.loc="F:/souravda/lib/")

asm_weekwise[is.na(asm_weekwise)] <- 0
asm_weekwise[asm_weekwise <= 0] <- mean(as.matrix(asm_weekwise))

weekjoyforecastvalues <- data.frame( "asm" = integer(), "value"
= integer(), stringsAsFactors=FALSE)

for(i in 2:ncol(asm_weekwise))
{
  asmname<-names(asm_weekwise)[i]
  temparimadata<-asm_weekwise[,i]
  m <- mean(as.matrix(temparimadata))
  #print(m)
  s <- sd(temparimadata)
  #print(s)
  temparimadata <- (temparimadata - m)
  temparimadata <- (temparimadata / s)
  temparima<-auto.arima(temparimadata, stationary = FALSE,
seasonal = TRUE, allowdrift = TRUE, allowmean = FALSE, biasadj
= FALSE)
  tempforecast<-forecast(temparima,h=12)
  #tempforecast <- (tempforecast * s)
  #print(tempforecast)
```

```
  temp_forecasted_data<-sum(data.frame(tempforecast$mean)*s + m)
  weekjoyforecastvalues[nrow(weekjoyforecastvalues) + 1, ] <-
c( asmname, temp_forecasted_data)
}

weekjoyforecastvalues$value<-as.integer(weekjoyforecastvalues$value)

#weekjoyforecastvalues

(sum(weekjoyforecastvalues$value)- 53782605)/53782605
#103000000)/103000000
```

Missing Data

One important aspect of time series and many other data analysis work is figuring out how to deal with missing data. In the previous code, you fill in the missing record with the average value. This is fine when the number of missing data instances is not very high. But if it is high, then the average of the highest and lowest values is a better alternative.

CHAPTER 7

Analytics at Scale

In recent decades, a revolutionary change has taken place in the field of analytics technology because of big data. Data is being collected from a variety of sources, so technology has been developed to analyze this data in a distributed environment, even in real time.

Hadoop

The revolution started with the development of the Hadoop framework, which has two major components, namely, MapReduce programming and the HDFS file system.

MapReduce Programming

MapReduce is a programming style inspired by functional programming to deal with large amounts of data. The programmer can process big data using MapReduce code without knowing the internals of the distributed environment. Before MapReduce, frameworks like Condor did parallel computing on distributed data. But the main advantage of MapReduce is that it is RPC based. The data does not move; on the contrary, the code jumps to different machines to process the data. In the case of big data, it is a huge savings of network bandwidth as well as computational time.

S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1_7

A MapReduce program has two major components: the mapper and the reducer. In the mapper, the input is split into small units. Generally, each line of input file becomes an input for each map job. The mapper processes the input and emits a key-value pair to the reducer. The reducer receives all the values for a particular key as input and processes the data for final output.

The following pseudocode is an example of counting the frequency of words in a document:

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");

reduce(String key, Iterator values):
 // key: a word
 // values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

Partitioning Function

Sometimes it is required to send a particular data set to a particular reduce job. The partitioning function solves this purpose. For example, in the previous MapReduce example, say the user wants the output to be stored in sorted order. Then he mentions the number of the reduce job 32 for 32

alphabets, and in the practitioner he returns 1 for the key starting with a, 2 for b, and so on. Then all the words that start with the same letters and go to the same reduce job. The output will be stored in the same output file, and because MapReduce assures that the intermediate key-value pairs are processed in increasing key order, within a given partition, the output will be stored in sorted order.

Combiner Function

The combiner is a facility in MapReduce where partial aggregation is done in the map phase. Not only does it increase the performance, but sometimes it is essential to use if the data set so huge that the reducer is throwing a stack overflow exception. Usually the reducer and combiner logic are the same, but this might be necessary depending on how MapReduce deals with the output.

To implement this word count example, we will follow a particular design pattern. There will be a root `RootBDAS` (BDAS stands for Big Data Analytic System) class that has two abstract methods: a mapper task and a reducer task. All child classes implement these mapper and reducer tasks. The main class will create an instance of the child class using reflection, and in MapReduce map functions call the mapper task of the instance and the reducer function of the reducer task. The major advantages of this pattern are that you can do unit testing of the MapReduce functionality and that it is adaptive. Any new child class addition does not require any changes in the main class or unit testing. You just have to change the configuration. Some code may need to implement combiner or partitioner logics. They have to inherit the `ICombiner` or `IPartitioner` interface.

Figure 7-1 shows a class diagram of the system.

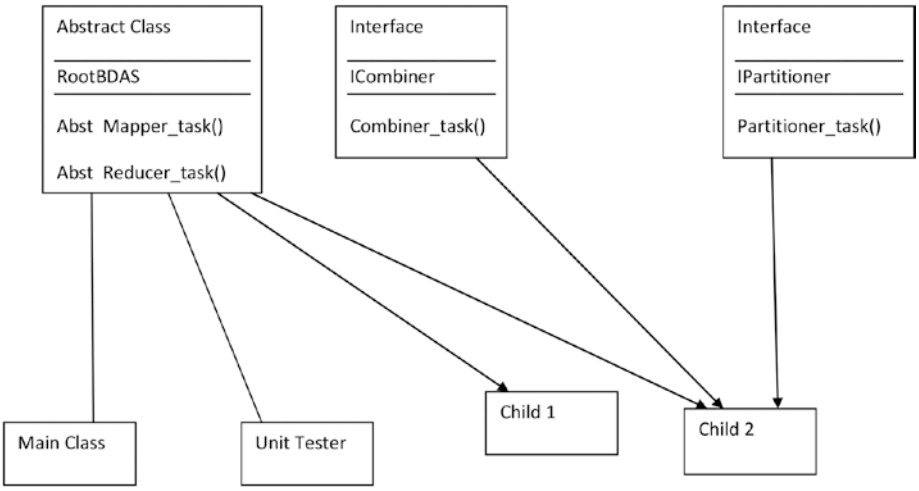


Figure 7-1. *The class diagram*

Here is the `RootBDAS` class:

```
import java.util.ArrayList;
import java.util.HashMap;

/**
 *
 */

/**
 * @author SayanM
 *
 */
```

```
public abstract class RootBDAS {
       abstract  HashMap<String, ArrayList<String>>
       mapper_task(String line);
       abstract  HashMap<String, ArrayList<String>>
       reducer_task(String key, ArrayList<String> values);

}
```

Here is the child class:

```
import java.util.ArrayList;
import java.util.HashMap;

/**
 *
 */

/**
 * @author SayanM
 *
 */

public final class WordCounterBDAS extends RootBDAS{

       @Override
       HashMap<String, ArrayList<String>> mapper_task
       (String line) {
              // TODO Auto-generated method stub
              String[] words = line.split(" ");
              HashMap<String, ArrayList<String>> result = new
              HashMap<String, ArrayList<String>>();
              for(String w : words)
              {
                     if(result.containsKey(w))
```

```
                {
                        ArrayList<String> vals = result.
                        get(w);
                        vals.add("1");
                        result.put(w, vals);
                }
                else
                {
                        ArrayList<String> vals = new
                        ArrayList<String>();
                        vals.add("1");
                        result.put(w, vals);
                }
        }
        return result;
}

@Override
HashMap<String, ArrayList<String>> reducer_task
(String key, ArrayList<String> values) {
        // TODO Auto-generated method stub
        HashMap<String, ArrayList<String>> result = new
        HashMap<String, ArrayList<String>>();
        ArrayList<String> tempres = new ArrayList
        <String>();
        tempres.add(values.size()+ "");
        result.put(key, tempres);
        return result;
}

}
```

Here is the WordCounterBDAS utility class:

```
import java.util.ArrayList;
import java.util.HashMap;

/**
 *
 */

/**
 * @author SayanM
 *
 */

public final class WordCounterBDAS extends RootBDAS{

	@Override
	HashMap<String, ArrayList<String>> mapper_task
	(String line) {
		// TODO Auto-generated method stub
		String[] words = line.split(" ");
		HashMap<String, ArrayList<String>> result = new
		HashMap<String, ArrayList<String>>();
		for(String w : words)
		{
			if(result.containsKey(w))
			{
				ArrayList<String> vals = result.
				get(w);
				vals.add("1");
				result.put(w, vals);
			}
			else
```

```
                    {
                            ArrayList<String> vals = new
                            ArrayList<String>();
                            vals.add("1");
                            result.put(w, vals);
                     }
              }
              return result;
       }

       @Override
       HashMap<String, ArrayList<String>> reducer_task
       (String key, ArrayList<String> values) {
              // TODO Auto-generated method stub
       HashMap<String, ArrayList<String>> result = new
       HashMap<String, ArrayList<String>>();
              ArrayList<String> tempres = new
              ArrayList<String>();
              tempres.add(values.size()+ "");
              result.put(key, tempres);
              return result;
       }

}
```

Here is the MainBDAS class:

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/**
 *
 */

/**
 * @author SayanM
 *
 */
public class MainBDAS {

       public static class MapperBDAS extends
       Mapper<LongWritable, Text, Text, Text> {

              protected void map(LongWritable key, Text value,
              Context context)
                           throws IOException, Interrupted
                           Exception {
                     String classname = context.
                     getConfiguration().get("classname");

                     try {
                           RootBDAS instance = (RootBDAS)
                           Class.forName(classname).
                           getConstructor().newInstance();
```

```
                        String line = value.toString();
                        HashMap<String, ArrayList<String>>
                        result = instance.mapper_task(line);
                        for(String k : result.keySet())
                        {
                               for(String v : result.get(k))
                               {
                                      context.write(new
                                      Text(k), new Text(v));
                               }
                        }
                 } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                }

}

public static class ReducerBDAS extendsReducer<Text,
Text, Text, Text> {

       protected void reduce(Text key, Iterable<Text>
       values,
                        Context context) throws IOException,
                        InterruptedException {
              String classname = context.
              getConfiguration().get("classname");
              try {
```

```
                    RootBDAS instance = (RootBDAS)
                    Class.forName(classname).
                    getConstructor().newInstance();
                    ArrayList<String> vals = new
                    ArrayList<String>();
                    for(Text v : values)
                    {
                          vals.add(v.toString());
                    }
                    HashMap<String, ArrayList<String>>
                    result = instance.reducer_task(key.
                    toString(), vals);
                    for(String k : result.keySet())
                    {
                          for(String v : result.get(k))
                          {
                                context.write(new
                                Text(k), new Text(v));
                          }
                    }
              } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }

        }

}
```

```
        public static void main(String[] args) throws Exception
{
                // TODO Auto-generated method stub

            String classname = Utility.getClassName(Utility.
            configpath);

                Configuration con = new Configuration();
                con.set("classname", classname);

                Job job = new Job(con);

                job.setJarByClass(MainBDAS.class);
                job.setJobName("MapReduceBDAS");

                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);

                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.setInputPaths(job, new
                Path(args[0]));
                FileOutputFormat.setOutputPath(job, new
                Path(args[1]));
                job.setMapperClass(MapperBDAS.class);
                job.setReducerClass(ReducerBDAS.class);

                System.out.println(job.waitForCompletion(true));

         }

}
```

To test the example, you can use this unit testing class:

```
import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.HashMap;

import org.junit.Test;


public class testBDAS {

	@Test
	public void testMapper() throws Exception{
		String classname = Utility.getClassName(Utility.
		testconfigpath);
		RootBDAS instance = (RootBDAS) Class.
		forName(classname).getConstructor().
		newInstance();
		String line = Utility.getMapperInput(Utility.
		testconfigpath);
		HashMap<String, ArrayList<String>> actualresult =
		instance.mapper_task(line);
		HashMap<String, ArrayList<String>> expectedresult
		= Utility.getMapOutput(Utility.testconfigpath);

		for(String key : actualresult.keySet())
		{
			boolean haskey = expectedresult.
			containsKey(key);
			assertEquals(true, haskey);
			ArrayList<String> actvals = actualresult.
			get(key);
			for(String v : actvals)
			{
```

```
                        boolean hasval = expectedresult.
                        get(key).contains(v);
                        assertEquals(true, hasval);
                }
        }

    }

    @Test
    public void testReducer(){
        fail();

    }

}
```

Finally, here are the interfaces:

```
import java.util.ArrayList;
import java.util.HashMap;

public interface ICombiner {
    HashMap<String, ArrayList<String>>  combiner_task(String
key, ArrayList<String> values);

}

public interface IPartitioner {

    public int  partitioner_task(String line);

}
```

HDFS File System

Other than MapReduce, HDFS is the second component in the Hadoop framework. It is designed to deal with big data in a distributed environment for general-purpose low-cost hardware. HDFS is built on top of the Unix POSSIX file system with some modifications, with the goal of dealing with streaming data.

The Hadoop cluster consists of two types of host: the name node and the data node. The name node stores the metadata, controls execution, and acts like the master of the cluster. The data node does the actual execution; it acts like a slave and performs instructions sent by the name node.

MapReduce Design Pattern

MapReduce is an archetype for processing the data that resides in hundreds of computers. There are some design patterns that are common in MapReduce programming.

Summarization Pattern

In summary, the reducer creates the summary for each key (see Figure 7-2). The practitioner can be used if you want to sort the data or for any other purpose. The word count is an example of the summarizer pattern. This pattern can be used to find the minimum, maximum, and count of data or to find the average, median, and standard deviation.

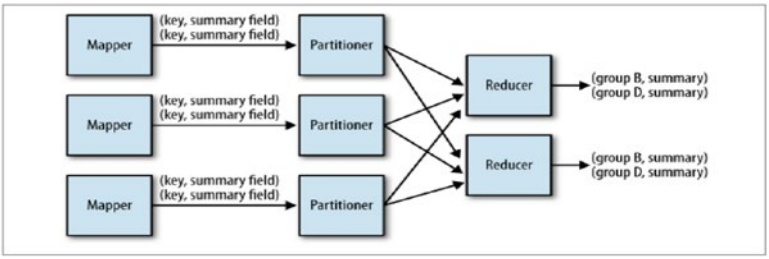


Figure 7-2. *Details of the summarization pattern*

Filtering Pattern

In MapReduce filtering is done in a divide-and-conquer way (Figure 7-3). Each mapper job filters a subset of data, and the reducer aggregates the filtered subset and produces the final output. Generating the top N records, searching data, and sampling data are the common use cases of the filtering pattern.

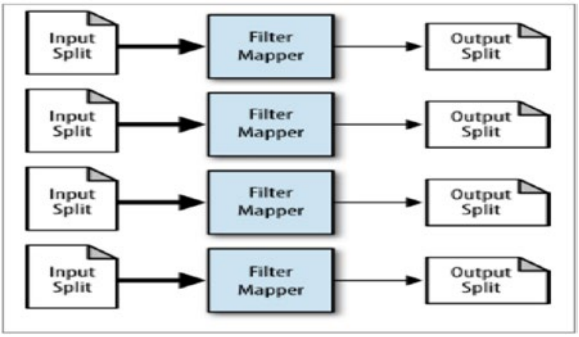


Figure 7-3. *Details of the filtering pattern*

Join Patterns

In MapReduce, joining (Figure 7-4) can be done on the map side or the reduce side. For the map side, the join data sets that will be joined should exist in the same cluster; otherwise, the reduce-side join is required. The join can be an outer join, inner join, or anti-join.

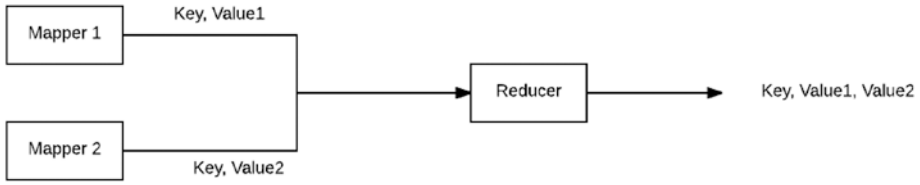


Figure 7-4. *Details of the join pattern*

The following code is an example of the reducer-side join:

```
package MapreduceJoin;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.lib.MultipleInputs;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.Reducer;
```

```
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;

@SuppressWarnings("deprecation")
public class MapreduceJoin {

////////////////////////////////////////////////////

       @SuppressWarnings("deprecation")
       public static class JoinReducer extends MapReduceBase
       implements Reducer<Text, Text, Text, Text>
       {
              public void reduce(Text key, Iterator<Text>
              values, OutputCollector<Text, Text> output,
              Reporter reporter) throws IOException
              {
                     ArrayList<String> translist = new
                     ArrayList<String>();
                     String secondvalue = "";
                     while (values.hasNext())
                     {
                           String currValue = values.next().
                           toString().trim();
                           if(currValue.contains("trans:")){
                                  String[] temp = currValue.
                                  split("trans:");
                                  if(temp.length > 1)
                                        translist.
                                        add(temp[1]);
                            }
                            if(currValue.contains("sec:"))
```

```
                    {
                            String[] temp = currValue.
                            split("sec:");
                            if(temp.length > 1)
                                    secondvalue = temp[1];
                     }
              }

              for(String trans : translist)
              {
                    output.collect(key, new Text(trans
                    +'\t' + secondvalue));
             }
       }
}

//////////////////////////////////////////////////////

@SuppressWarnings("deprecation")
public static class TransactionMapper extends
MapReduceBase implements Mapper<LongWritable, Text,
Text, Text>
{
      int index1 = 0;
      public void configure(JobConf job) {
             index1 = Integer.parseInt(job.
             get("index1"));

      }
      public void map(LongWritable key, Text value,
      OutputCollector<Text, Text> output, Reporter
      reporter) throws IOException
```

```
        {
              String line = value.toString().trim();
              if(line=="") return;
          String splitarray[] = line.split("\t");
          String id = splitarray[index1].trim();
          String ids = "trans:" + line;
          output.collect(new Text(id), new Text(ids));

      }
}

//////////////////////////////////////////////////////

@SuppressWarnings("deprecation")
public static class SecondaryMapper extends
MapReduceBase implements Mapper<LongWritable, Text,
Text, Text>
{
      int index2 = 0;
      public void configure(JobConf job) {
              index2 = Integer.parseInt(job.
              get("index2"));

      }

      public void map(LongWritable key, Text value,
      OutputCollector<Text, Text> output, Reporter
      reporter) throws IOException
      {
              String line = value.toString().trim();
              if(line=="") return;
        String splitarray[] = line.split("\t");
```

```
            String id = splitarray[index2].trim();
            String ids = "sec:" + line;
            output.collect(new Text(id), new Text(ids));
          }
    }
////////////////////////////////////////////////////////////

    @SuppressWarnings({ "deprecation", "rawtypes",
    "unchecked" })
    public static void main(String[] args)
    throws IOException, ClassNotFoundException,
    InterruptedException {
          // TODO Auto-generated method stub

          JobConf conf = new JobConf();
          conf.set("index1", args[3]);
          conf.set("index2", args[4]);
          conf.setReducerClass(JoinReducer.class);

          MultipleInputs.addInputPath(conf, new
          Path(args[0]), TextInputFormat.class, (Class<?
          extends org.apache.hadoop.mapred.Mapper>)
          TransactionMapper.class);
          MultipleInputs.addInputPath(conf, new
          Path(args[1]), TextInputFormat.class, (Class<?
          extends org.apache.hadoop.mapred.Mapper>)
          SecondaryMapper.class);
          Job job = new Job(conf);
          job.setJarByClass(MapreduceJoin.class);
          job.setJobName("MapReduceJoin");

          job.setOutputKeyClass(Text.class);
```

```
            job.setOutputValueClass(Text.class);

            FileOutputFormat.setOutputPath(job, new
            Path(args[2]));

            System.out.println(job.waitForCompletion(true));

        }

}
```

Spark

After Hadoop, Spark is the next and latest revolution in big data technology. The major advantage of Spark is that it gives a unified interface to the entire big data stack. Previously, if you needed a SQL-like interface for big data, you would use Hive. If you needed real-time data processing, you would use Storm. If you wanted to build a machine learning model, you would use Mahout. Spark brings all these facilities under one umbrella. In addition, it enables in-memory computation of big data, which makes the processing very fast. Figure 7-5 describes all the components of Spark.

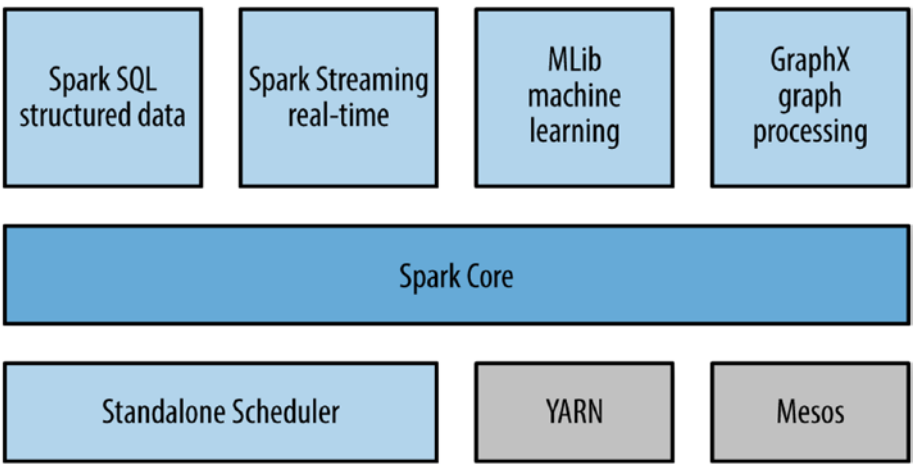


Figure 7-5. *The components of Spark*

Spark Core is the fundamental component of Spark. It can run on top of Hadoop or stand-alone. It abstracts the data set as a resilient distributed data set (RDD). RDD is a collection of read-only objects. Because it is read only, there will not be any synchronization problems when it is shared with multiple parallel operations. Operations on RDD are lazy. There are two types of operations happening on RDD: transformation and action. In transformation, there is no execution happening on a data set. Spark only stores the sequence of operations as a directed acyclic graph called a *lineage*. When an action is called, then the actual execution takes place. After the first execution, the result is cached in memory. So, when a new execution is called, Spark makes a traversal of the lineage graph and makes maximum reuse of the previous computation, and the computation for the new operation becomes the minimum. This makes data processing very fast and also makes the data fault tolerant. If any node fails, Spark looks at the lineage graph for the data in that node and easily reproduces it.

One limitation of the Hadoop framework is that it does not have any message-passing interface in parallel computation. But there are several use cases where parallel jobs need to talk with each other. Spark achieves this using two kinds of shared variable. They are the broadcast variable and the accumulator. When one job needs to send a message to all other jobs, the job uses the broadcast variable, and when multiple jobs want to aggregate their results to one place, they use an accumulator. RDD splits its data set into a unit called a *partition*. Spark provides an interface to specify the partition of the data, which is very effective for future operations like `join` or `find`. The user can specify the storage type of partition in Spark. Spark has a programming interface in Python, Java, and Scala. The following code is an example of a word count program in Spark:

```
val conf = new SparkConf().setAppName("wiki_test") // create a
spark config object
val sc = new SparkContext(conf) // Create a spark context
val data = sc.textFile("/path/to/somedir") // Read files from
"somedir" into an RDD of (filename, content) pairs.
val tokens = data.flatMap(_.split(" ")) // Split each file into
a list of tokens (words).
val wordFreq = tokens.map((_, 1)).reduceByKey(_ + _) // Add a
count of one to each token, then sum the counts per word type.
wordFreq.sortBy(s => -s._2).map(x => (x._2, x._1)).top(10)
// Get the top 10 words. Swap word and count to sort by count.
```

On top of Spark Core, Spark provides the following:

- Spark SQL, which is a SQL interface through the command line or a database connector interface. It also provides a SQL interface for the Spark data frame object.
- Spark Streaming, which enables you to process streaming data in real time.
- MLib, a machine learning library to build analytical models on Spark data.
- GraphX, a distributed graph processing framework.

Analytics in the Cloud

Like many other fields, analytics is being impacted by the cloud. It is affected in two ways. Big cloud providers are continuously releasing machine learning APIs. So, a developer can easily write a machine learning application without worrying about the underlining algorithm. For example, Google provides APIs for computer vision, natural language,

speech processing, and many more. A user can easily write code that can give the sentiment of an image of a face or voice in two or three lines of code.

The second aspect of the cloud is in the data engineering part. In Chapter 1 I gave an example of how to expose a model as a high-performance REST API using Falcon. Now if a million users are going to use it and if the load varies by much, then autoscale is a required feature of this application. If you deploy the application in Google App Engine or AWS Lambda, you can achieve the autoscale feature in 15 minutes. Once the application is autoscaled, you need to think about the database. DynamoDB from Amazon and Cloud Datastore by Google are autoscaled databases in the cloud. If you use one of them, your application is now high performance and autoscaled, but people around globe will access it, so the geographical distance will create extra latency or a negative impact on performance. You also have to make sure that your application is always available. Further, you need to deploy your application in three regions: Europe, Asia, and the United States (you can choose more regions if your budget permits). If you use an elastic load balancer with a geobalancing routing rule, which routes the traffic from a region to the app engine of that region, then it will be available across the globe. In geobalancing, you can mention a secondary app engine for each rule, which makes your application highly available. If a primary app engine is down, the secondary app engine will take care of the things.

Figure 7-6 describes this system.

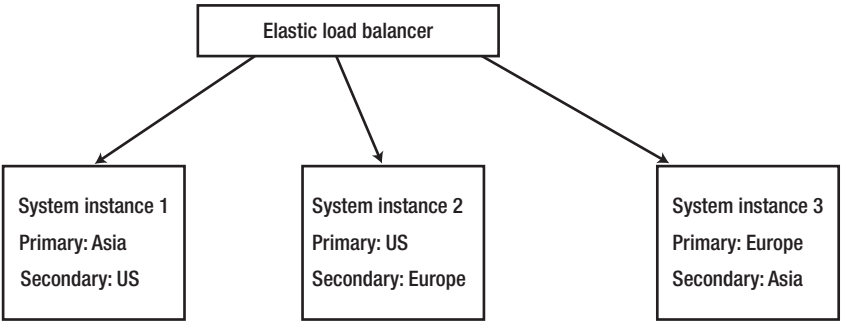


Figure 7-6. *The system*

In Chapter 1 I showed some example code of publishing a deep learning model as a REST API. The following code is the implementation of the same logic in a cloud environment where the other storage is replaced by a Google data store:

```
import falcon
from falcon_cors import CORS
import json
import pygeoip
import json
import datetime as dt
import ipaddress
import math
from concurrent.futures import *
import numpy as np
from google.cloud import datastore

def logit(x):
       return (np.exp(x) / (1 + np.exp(x)))

def is_visible(client_size, ad_position):
```

```
        y=height=0
        try:
                height  = int(client_size.split(',')[1])
                y = int(ad_position.split(',')[1])
        except:
                pass
        if y < height:
                return "1"
        else:
                return "0"

class Predictor(object):
       def __init__(self,domain,is_big):
              self.client = datastore.Client('sulvo-east')
              self.ctr = 'ctr_' + domain
              self.ip = "ip_" + domain
              self.scores = "score_num_" + domain
              self.probabilities = "probability_num_" + domain
              if is_big:
                    self.is_big = "is_big_num_" + domain
                    self.scores_big = "score_big_num_" + domain
                    self.probabilities_big = "probability_big_
                    num_" + domain
              self.gi = pygeoip.GeoIP('GeoIP.dat')
              self.big = is_big
              self.domain = domain

       def get_hour(self,timestamp):
              return dt.datetime.utcfromtimestamp(timestamp /
              1e3).hour
```

```
    def fetch_score(self, featurename, featurevalue, kind):
        pred = 0
        try:
            key = self.client.key(kind,featurename +
            "_" + featurevalue)
            res= self.client.get(key)
            if res is not None:
                pred = res['score']
        except:
            pass
        return pred

    def get_score(self, featurename, featurevalue):
        with ThreadPoolExecutor(max_workers=5) as pool:
            future_score = pool.submit(self.fetch_
            score,featurename, featurevalue,self.
            scores)
          future_prob = pool.submit(self.fetch_
          score,featurename, featurevalue,self.
          probabilities)
          if self.big:
              future_howbig = pool.submit(self.
              fetch_score,featurename,
              featurevalue,self.is_big)
              future_predbig = pool.submit(self.
              fetch_score,featurename,
              featurevalue,self.scores_big)
              future_probbig = pool.submit(self.
              fetch_score,featurename,
              featurevalue,self.probabilities_big)
          pred = future_score.result()
          prob = future_prob.result()
```

```
                if not self.big:
                        return pred, prob
                howbig = future_howbig.result()
                pred_big = future_predbig.result()
                prob_big = future_probbig.result()
                return howbig, pred, prob, pred_big, prob_big

def get_value(self, f, value):
      if f == 'visible':
                fields = value.split("_")
                value = is_visible(fields[0], fields[1])
                if f == 'ip':
                ip = str(ipaddress.IPv4Address(ipaddress.
                ip_address(value)))
                    geo = self.gi.country_name_by_addr(ip)
                if self.big:
                        howbig1,pred1, prob1, pred_big1,
                        prob_big1 = self.get_score('geo',
                        geo)
                else:
                        pred1, prob1 = self.get_score('geo',
                        geo)
                freq = '1'
                key = self.client.key(self.ip,ip)
                res = self.client.get(key)
                if res is not None:
                         freq = res['ip']
                if self.big:
                        howbig2, pred2, prob2, pred_
                        big2, prob_big2 = self.get_
                        score('frequency', freq)
```

```
                else:
                        pred2, prob2 =  self.get_
                        score('frequency', freq)
                if self.big:
                        return (howbig1 + howbig2), (pred1 +
                        pred2), (prob1 + prob2), (pred_big1
                        + pred_big2), (prob_big1 + prob_
                        big2)
                else:
                        return (pred1 + pred2), (prob1 + prob2)
        if f == 'root':
                try:
                        res = client.get('root', value)
                        if res is not None:
                                ctr = res['ctr']
                                avt = res['avt']
                                avv = res['avv']
                                if self.big:
                                   (howbig1,pred1,prob1,pred_
                                   big1,prob_big1) = self.
                                   get_score('ctr', str(ctr))
                                   (howbig2,pred2,prob2,pred_
                                   big2,prob_big2) = self.
                                   get_score('avt', str(avt))
                                   (howbig3,pred3,prob3,pred_
                                   big3,prob_big3) = self.
                                   get_score('avv', str(avv))
                                   (howbig4,pred4,prob4,pred_
                                   big4,prob_big4) = self.
                                   get_score(f, value)
```

```
                                else:
                                  (pred1,prob1) = self.get_
                                  score('ctr', str(ctr))
                                  (pred2,prob2) = self.get_
                                  score('avt', str(avt))
                                  (pred3,prob3) = self.get_
                                  score('avv', str(avv))
                                  (pred4,prob4) = self.get_
                                  score(f, value)
                          if self.big:
                                return (howbig1 + howbig2 +
                                howbig3 + howbig4), (pred1
                                + pred2 + pred3 + pred4),
                                (prob1 + prob2 + prob3 +
                                prob4),(pred_big1 + pred_
                                big2 + pred_big3 + pred_
                                big4),(prob_big1 + prob_big2
                                + prob_big3 + prob_big4)
                          else:
                                return (pred1 + pred2 + pred3
                                + pred4), (prob1 + prob2 +
                                prob3 + prob4)
                    except:
                          return 0,0
              if f == 'client_time':
                 value = str(self.get_hour(int(value)))
            return self.get_score(f, value)

      def get_multiplier(self):
            key = self.client.key('multiplier_all_num',
            self.domain)
              res = self.client.get(key)
```

```
            high = res['high']
            low = res['low']
        if self.big:
                    key = self.client.key('multiplier_
                    all_num', self.domain + "_big")
            res = self.client.get(key)
            high_big = res['high']
            low_big = res['low']
            return high, low, high_big, low_big
        return high, low

    def on_post(self, req, resp):
        if True:
            input_json = json.loads(req.stream.
            read(),encoding='utf-8')
            input_json['visible'] = input_json['client_
            size'] + "_" + input_json['ad_position']
            del input_json['client_size']
            del input_json['ad_position']
            howbig = 0
            pred = 0
            prob = 0
            pred_big = 0
            prob_big = 0
            worker = ThreadPoolExecutor(max_workers=1)
            thread = worker.submit(self.get_multiplier)
            with ThreadPoolExecutor(max_workers=8) as
            pool:
                future_array = { pool.submit(self.
                get_value,f,input_json[f]) : f for f
                in input_json}
```

```
        for future in as_completed(future_
        array):
                if self.big:
                        howbig1, pred1,
                        prob1,pred_big1,prob_
                        big1 = future.result()
                        pred = pred + pred1
                        pred_big = pred_big +
                        pred_big1
                        prob = prob + prob1
                        prob_big = prob_big +
                        prob_big1
                        howbig = howbig +
                        howbig
                else:
                        pred1, prob1 = future.
                        result()
                        pred = pred + pred1
                        prob = prob + prob1

if self.big:
        if howbig > .65:
                pred, prob = pred_big, prob_
                big

resp.status = falcon.HTTP_200


  res = math.exp(pred)-1
if res < 0.1:
        res = 0.1
if prob < 0.1 :
        prob = 0.1
```

```
                    if prob > 0.9:
                           prob = 0.9

                    if self.big:
                           high, low, high_big, low_big =
                           thread.result()
                           if howbig > 0.6:
                                  high = high_big
                                  low = low_big
                    else:
                           high, low = thread.result()

                    multiplier = low + (high -low)*prob
                    res = multiplier*res
                    resp.body = str(res)
             #except Exception,e:
             #     print(str(e))
             #     resp.status = falcon.HTTP_200
             #     resp.body = str("0.1")

cors = CORS(allow_all_origins=True,allow_all_
methods=True,allow_all_headers=True)
wsgi_app = api = falcon.API(middleware=[cors.middleware])

f = open('publishers2.list_test')
for line in f:
       if "#" not in line:
             fields = line.strip().split('\t')
             domain = fields[0].strip()
             big = (fields[1].strip() == '1')
             p = Predictor(domain, big)
             url = '/predict/' + domain
             api.add_route(url, p)
f.close()
```

You can deploy this application in the Google App Engine with the following:

```
gcloud app deploy --prject <prject id>  --version <version no>
```

Internet of Things

The IoT is simply the network of interconnected things/devices embedded with sensors, software, network connectivity, and necessary electronics that enable them to collect and exchange data, making them responsive. The field is emerging with the rise of technology just like big data, real-time analytics frameworks, mobile communication, and intelligent programmable devices. In the IoT, you can do the analysis of data on the server side using the techniques shown throughout the book; you can also put logic on the device side using the Raspberry Pi, which is an embedded system version of Python.

Index

S. Mukhopadhyay, *Advanced Data Analytics Using Python*,
https://doi.org/10.1007/978-1-4842-3450-1

D

E

F

G

H

I, J

K

L

M

N

O

P, Q

R

S

T

U, V, W, X, Y, Z