MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Autonomous Institution - UGC, Govt. of India

APPROVED BY AICTE, ACCREDITED BY NBA & NAAC – 'A' GRADE ISO 9001:2015 CERTIFIED

B.TECH	II – I	R20	2021-2022

Introduction to Data Science

[R20DS501]

DIGITAL NOTES

DEPARTMENT OF CSE (DATA SCIENCE, CYBER SECURITY, INTERNET OF THINGS)

Introduction to Datascience
(R20DS501)

LECTURE NOTES

B.TECH II YEAR – I SEM (R20)
(2021-2022)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
(DATA SCIENCE,CYBER SECURITY,INTERNET OF THINGS)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, INDIA.

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

II Year B.Tech CSE(DS) – I Sem

L/T/P/C
3 -/-/-3

(R20DS501) Introduction to Datascience

COURSE OBJECTIVES:

1. An understanding of the data operations
2. An overview of simple statistical models and the basics of machine learning techniques of regression.
3. An understanding good practices of data science
4. Skills in the use of tools such as python, IDE
5. Understanding of the basics of the Supervised learning

UNIT-1

Introduction, Toolboxes: Python, fundamental libraries for data Scientists. Integrated development environment (IDE). Data operations: Reading, selecting, filtering, manipulating, sorting, grouping, rearranging, ranking, and plotting.

UNIT-2

Descriptive statistics, data preparation. Exploratory Data Analysis data summarization, data distribution, measuring asymmetry. Sample and estimated mean, variance and standard score. Statistical Inference frequency approach, variability of estimates, hypothesis testing using confidence intervals, using p-values

UNIT-3

Supervised Learning: First step, learning curves, training-validation and test. Learning models generalities, support vector machines, random forest. Examples

UNIT-4

Regression analysis, Regression: linear regression simple linear regression, multiple & Polynomial regression, Sparse model. Unsupervised learning, clustering, similarity and distances, quality measures of clustering, case study.

UNIT-5

Network Analysis, Graphs, Social Networks, centrality, drawing centrality of Graphs, PageRank, Ego-Networks, community Detection

TEXT/REFERENCES BOOK:

1. Introduction to Data Science a Python approach to concepts, Techniques and Applications, Igual, L;Seghi', S. Springer, ISBN:978-3-319-50016-4
2. Data Analysis with Python A Modern Approach, David Taieb, Packt Publishing, ISBN-9781789950069
3. Python Data Analysis, Second Ed., Armando Fandango, Packt Publishing, ISBN: 9781787127487

COURSE OUTCOMES:

1. Describe what Data Science is and the skill sets needed to be a data scientist
2. Explain the significance of exploratory data analysis (EDA) in data science
3. Ability to learn the supervised learning, SVM
4. Apply basic machine learning algorithms (Linear Regression)
5. Explore the Networks, PageRank

UNIT-1

Introduction, Toolboxes: Python, fundamental libraries for data Scientists. Integrated development environment (IDE). Data operations: Reading, selecting, filtering, manipulating, sorting, grouping, rearranging, ranking, and plotting.

Introduction to Data Science

1.1 What is Data Science?

You have, no doubt, already experienced data science in several forms. When you are looking for information on the web by using a search engine or asking your mobile phone for directions, you are interacting with data science products. Data science has been behind resolving some of our most common daily tasks for several years.

Most of the scientific methods that power data science are not new and they havebeen out there, waiting for applications to be developed, for a long time. Statistics is an old science that stands on the shoulders of eighteenth-century giants such as Pierre Simon Laplace (1749–1827) and Thomas Bayes (1701–1761). Machine learning isyounger, but it has already moved beyond its infancy and can be considered a well- established discipline. Computer science changed our lives several decades ago andcontinues to do so; but it cannot be considered new.

So, why is data science seen as a novel trend within business reviews, in technologyblogs, and at academic conferences?

The novelty of data science is not rooted in the latest scientific knowledge, but in a disruptive change in our society that has been caused by the evolution of technology: datification. Datification is the process of rendering into data aspects of the world that have never been quantified before. At the personal level, the list of datified conceptsis very long and still growing: business networks, the lists of books we are reading,the films we enjoy, the food we eat, our physical activity, our purchases, our drivingbehavior, and so on. Even our thoughts are datified when we publish them on our favorite social network; and in a not so distant future, your gaze could be datified by wearable vision registering devices. At the business level, companies are datifying semi-structured data that were previously discarded: web activity logs, computer network activity, machinery signals, etc. Nonstructured data, such as written reports,e-mails, or voice recordings, are now being stored not only for archive purposes butalso to be analyzed.

However, datification is not the only ingredient of the data science revolution. The other ingredient is the democratization of data analysis. Large companies such as Google, Yahoo, IBM, or SAS were the only players in this field when data science had no name. At the beginning of the century, the huge computational resources of those companies allowed them to take advantage of datification by using analyticaltechniques to develop innovative products and even to take decisions about their own business. Today, the analytical gap between those companies and the rest of the world (companies and people) is shrinking. Access to cloud computing allows any individual to analyze huge amounts of data in short periods of time. Analyticalknowledge is free and most of the crucial algorithms that are needed to create a solution can be found, because open-source development is the norm in this field. Asa result, the possibility of using rich data to take evidence-based decisions is open to virtually any person or company.

Data science is commonly defined as a methodology by which actionable insights can be inferred from data. This is a subtle but important difference with respect to previous approaches to data analysis, such as business intelligence or exploratory statistics. Performing data science is a task with an ambitious objective: the produc-tion of beliefs informed by data and to be used as the basis of decision-making. In the absence of data, beliefs are uninformed and decisions, in the best of cases, are based on best practices or intuition. The representation of complex environments byrich data opens up the possibility of applying all the scientific knowledge we have regarding how to infer knowledge from data.

In general, data science allows us to adopt four different strategies to explore theworld using data:

1. *Probing reality*. Data can be gathered by passive or by active methods. In the latter case, data represents the response of the world to our actions. Analysis of those responses can be extremely valuable when it comes to taking decisions about our subsequent actions. One of the best examples of this strategy is the use of A/B testing for web development: What is the best button size and color?The best answer can only be found by probing the world.
2. *Pattern discovery*. Divide and conquer is an old heuristic used to solve complex problems; but it is not always easy to decide how to apply this common sense to problems. Datified problems can be analyzed automatically to discover useful patterns and natural clusters that can greatly simplify their solutions. The use of this technique to profile users is a critical ingredient today in such importantfields as programmatic advertising or digital marketing.
3. *Predicting future events*. Since the early days of statistics, one of the most impor-tant scientific questions has been how to build robust data models that are capa- ble of predicting future data samples. Predictive analytics allows decisions to be taken in response to future events, not only reactively. Of course, it is not possible to predict the future in any environment and there will always be unpre- dictable events; but the identification of predictable events represents valuableknowledge. For example, predictive analytics can be used

to optimize the tasks

planned for retail store staff during the following week, by analyzing data suchas weather, historic sales, traffic conditions, etc.

4. *Understanding people and the world*. This is an objective that at the moment is beyond the scope of most companies and people, but large companies and governments are investing considerable amounts of money in research areas such as understanding natural language, computer vision, psychology and neu- roscience. Scientific understanding of these areas is important for data science because in the end, in order to take optimal decisions, it is necessary to know the real processes that drive people's decisions and behavior. The development of deep learning methods for natural language understanding and for visual object recognition is a good example of this kind of research.

Toolboxes for Data Scientists

Introduction

In this chapter, first we introduce some of the tools that data scientists use. The toolbox of any data scientist, as for any kind of programmer, is an essential ingredient for success and enhanced performance. Choosing the right tools can save a lot of time and thereby allow us to focus on data analysis.

The most basic tool to decide on is which programming language we will use. Many people use only one programming language in their entire life: the first and only one they learn. For many, learning a new language is an enormous task that, if at all possible, should be undertaken only once. The problem is that some languages are intended for developing high-performance or production code, such as C, C++, or Java, while others are more focused on prototyping code, among these the best known are the so-called scripting languages: Ruby, Perl, and Python. So, depending on the first language you learned, certain tasks will, at the very least, be rather tedious. The main problem of being stuck with a single language is that many basic tools simply will not be available in it, and eventually you will have either to reimplementthem or to create a bridge to use some other language just for a specific task.

In conclusion, you either have to be ready to change to the best language for each task and then glue the results together, or choose a very flexible language with a rich ecosystem (e.g., third-party open-source libraries). In this book we have selected Python as the programming language.

Why Python?

Python[1] is a mature programming language but it also has excellent properties for newbie programmers, making it ideal for people who have never programmed before. Some of the most remarkable of those properties are easy to read code, suppression of non-mandatory delimiters, dynamic typing, and dynamic memory usage. Pythonis an interpreted language, so the code is executed immediately in the Python con- sole without needing the compilation step to machine language. Besides the Python console (which comes included with any Python installation) you can find other in-teractive consoles, such as IPython,[2] which give you a richer environment in whichto execute your Python code.

Currently, Python is one of the most flexible programming languages. One of its main characteristics that makes it so flexible is that it can be seen as a multiparadigm language. This is especially useful for people who already know how to program with other languages, as they can rapidly start programming with Python in the same way. For example, Java programmers will feel comfortable using Python as it supports the object-oriented paradigm, or C programmers could mix Python and C code using *cython*. Furthermore, for anyone who is used to programming in functional languages such as Haskell or Lisp, Python also has basic statements for functional programming in its own core library.

In this book, we have decided to use Python language because, as explained before, it is a mature language programming, easy for the newbies, and can be used as a specific platform for data scientists, thanks to its large ecosystem of scientific libraries and its high and vibrant community. Other popular alternatives to Python for data scientists are R and MATLAB/Octave.

Fundamental Python Libraries for Data Scientists

The Python community is one of the most active programming communities with a huge number of developed toolboxes. The most popular Python toolboxes for any data scientist are NumPy, SciPy, Pandas, and Scikit-Learn.

Numeric and Scientific Computation: NumPy and SciPy

NumPy[3] is the cornerstone toolbox for scientific computing with Python. NumPy provides, among other things, support for multidimensional arrays with basic oper-ations on them and useful linear algebra functions. Many toolboxes use the NumPyarray representations as an efficient basic data structure. Meanwhile, *SciPy* provides a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, and much more. Another core toolbox in SciPyis the plotting library *Matplotlib*. This toolbox has many tools for data visualization.

SCIKIT-Learn: Machine Learning in Python

Scikit-learn[4] is a machine learning library built from NumPy, SciPy, and Matplotlib. Scikit-learn offers simple and efficient tools for common tasks in data analysis such as classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

PANDAS: Python Data Analysis Library

Pandas[5] provides high-performance data structures and data analysis tools. The keyfeature of Pandas is a fast and efficient DataFrame object for data manipulation withintegrated indexing. The DataFrame structure can be seen as a spreadsheet which offers very flexible ways of working with it. You can easily transform any dataset in the way you want, by reshaping it and adding or removing columns or rows. It also provides high-performance functions for aggregating, merging, and joining dataset-
s. Pandas also has tools for importing and exporting data from different formats: comma-separated value (CSV), text files, Microsoft Excel, SQL databases, and the fast HDF5 format. In many situations, the data you have in such formats will not be complete or totally structured. For such cases, Pandas offers handling of miss-ing data and intelligent data alignment. Furthermore, Pandas provides a convenientMatplotlib interface.

Data Science Ecosystem Installation

Before we can get started on solving our own data-oriented problems, we will need toset up our programming environment. The first question we need to answer concerns

Python language itself. There are currently two different versions of Python: Python 2.X and Python 3.X. The differences between the versions are important, so there isno compatibility between the codes, i.e., code written in Python 2.X does not workin Python 3.X and vice versa. Python 3.X was introduced in late 2008; by then, a lotof code and many toolboxes were already deployed using Python 2.X (Python 2.0 was initially introduced in 2000). Therefore, much of the scientific community didnot change to Python 3.0 immediately and they were stuck with Python 2.7. By now, almost all libraries have been ported to Python 3.0; but Python 2.7 is still maintained, so one or another version can be chosen. However, those who already have a large amount of code in 2.X rarely change to Python 3.X. In our examples throughout thisbook we will use Python 2.7.

Once we have chosen one of the Python versions, the next thing to decide is whether we want to install the data scientist Python ecosystem by individual tool- boxes, or to perform a bundle installation with all the needed toolboxes (and a lot more). For newbies, the second option is recommended. If the first option is chosen,then it is only necessary to install all the mentioned toolboxes in the previous section, in exactly that order.

However, if a bundle installation is chosen, the Anaconda Python distribution[6] is then a good option. The Anaconda distribution provides integration of all the Python toolboxes and applications needed for data scientists into a single directory without mixing it with other Python toolboxes installed on the machine. It contain- s, of course, the core toolboxes and applications such as NumPy, Pandas, SciPy, Matplotlib, Scikit-learn, IPython, Spyder, etc., but also more specific tools for other related tasks such as data visualization, code optimization, and big data processing.

Integrated Development Environments (IDE)

For any programmer, and by extension, for any data scientist, the integrated de- velopment environment (IDE) is an essential tool. IDEs are designed to maximize programmer productivity. Thus, over the years this software has evolved in order tomake the coding task less complicated. Choosing the right IDE for each person is crucial and, unfortunately, there is no “one-size-fits-all” programming environment. The best solution is to try the most popular IDEs among the community and keep whichever fits better in each case.

In general, the basic pieces of any IDE are three: the editor, the compiler, (or interpreter) and the debugger. Some IDEs can be used in multiple programming languages, provided by language-specific plugins, such as Netbeans[7] or Eclipse.[8] Others are only specific for one language or even a specific programming task. In

the case of Python, there are a large number of specific IDEs, both commercial (PyCharm,[9] WingIDE[10] …) and open-source. The open-source community helps IDEs to spring up, thus anyone can customize their own environment and share it with the rest of the community. For example, Spyder[11] (Scientific Python Development EnviRonment) is an IDE customized with the task of the data scientist in mind.

Web Integrated Development Environment (WIDE): Jupyter

With the advent of web applications, a new generation of IDEs for interactive lan- guages such as Python has been developed. Starting in the academia and e- learningcommunities, web-based IDEs were developed considering how not only your codebut also all your environment and executions can be stored in a server. One of the first applications of this kind of WIDE was developed by William Stein in early 2005 using Python 2.3 as part of his SageMath mathematical software. In

SageMath, a server can be set up in a center, such as a university or school, and then students can work on their homework either in the classroom or at home, starting from exactly the same point they left off. Moreover, students can execute all the previous steps overand over again, and then change some particular *code cell* (a segment of the docu- ment that may content source code that can be executed) and execute the operation again. Teachers can also have access to student sessions and review the progress orresults of their pupils.

Nowadays, such sessions are called notebooks and they are not only used in classrooms but also used to show results in presentations or on business dashboards. The recent spread of such notebooks is mainly due to IPython. Since December 2011, IPython has been issued as a browser version of its interactive console, called IPython notebook, which shows the Python execution results very clearly and concisely by means of cells. Cells can contain content other than code. For example, markdown (a wiki text language) cells can be added to introduce algorithms. It is also possible toinsert Matplotlib graphics to illustrate examples or even web pages. Recently, some scientific journals have started to accept notebooks in order to show experimental results, complete with their code and data sources. In this way, experiments can become completely and absolutely replicable.

Since the project has grown so much, IPython notebook has been separated fromIPython software and now it has become a part of a larger project: Jupyter[12]. Jupyter (for Julia, Python and R) aims to reuse the same WIDE for all these interpreted languages and not just Python. All old IPython notebooks are automatically imported to the new version when they are opened with the Jupyter platform; but once they

Get Started with Python for Data Scientists

Throughout this book, we will come across many practical examples. In this chapter, we will see a very basic example to help get started with a data science ecosystem from scratch. To execute our examples, we will use Jupyter notebook, although anyother console or IDE can be used.

The Jupyter Notebook Environment

Once all the ecosystem is fully installed, we can start by launching the Jupyter notebook platform. This can be done directly by typing the following command onyour terminal or command line: $ jupyter notebook

If we chose the bundle installation, we can start the Jupyter notebook platform by clicking on the Jupyter Notebook icon installed by Anaconda in the start menu or onthe desktop.

The browser will immediately be launched displaying the Jupyter notebook home- page, whose URL is http://localhost:8888/tree. Note that a special port is used; by default it is 8888. As can be seen in Fig. 2.1, this initial page displays a tree view of a directory. If we use the command line, the root directory is the same directory where we launched the Jupyter notebook. Otherwise, if we use the Anaconda launcher, the root directory is the current user directory. Now, to start a new

New > Notebooks > Python 2

Table 2.1 List of most common aggregation functions

Function	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of values
median()	Arithmetic median of values
min()	Minimum
max()	Maximum
prod()	Product of values
std()	Unbiased standard deviation
var()	Unbiased variance

one of their results ends in an undefined value. A subtle feature of NaN values is that two NaN are never equal. Because of this, the only safe way to tell whether a value is missing in a DataFrame is by using the isnull() function. Indeed, this function can be used to filter rows with missing values:

In [9]:

```
edu[edu["Value"].isnull()].head()
```

Out[9]:

	TIME	GEO	Value
0	2000	European Union (28 countries)	NaN
1	2001	European Union (28 countries)	NaN
36	2000	Euro area (18 countries)	NaN
37	2001	Euro area (18 countries)	NaN
48	2000	Euro area (17 countries)	NaN

Manipulating Data

Once we know how to select the desired data, the next thing we need to know is how to manipulate data. One of the most straightforward things we can do is to operate with columns or rows using aggregation functions. Table 2.1 shows a list of the most common aggregation functions. The result of all these functions applied to a row or column is always a number. Meanwhile, if a function is applied to a DataFrame or a selection of rows and columns, then you can specify if the function should be applied to the rows for each column (setting the axis=0 keyword on the invocation of the function), or it should be applied on the columns for each row (setting the axis=1 keyword on the invocation of the function).

In [10]:

```
edu.max(axis = 0)
```

Out[10]:
```
TIME          2011
GEO          Spain
Value         8.81
dtype: object
```

Note that these are functions specific to Pandas, not the generic Python functions. There are differences in their implementation. In Python, NaN values propagate through all operations without raising an exception. In contrast, Pandas operations exclude NaN values representing missing data. For example, the pandas max function excludes NaN values, thus they are interpreted as missing values, while the standard Python max function will take the mathematical interpretation of NaN and return it as the maximum:

In [11]:
```
print "Pandas max function:", edu['Value'].max()
print "Python max function:", max(edu['Value'])
```

Out[11]:
```
Pandas max function: 8.81Python max function:
nan
```

Beside these aggregation functions, we can apply operations over all the values in rows, columns or a selection of both. The rule of thumb is that an operation between columns means that it is applied to each row in that column and an operation between rows means that it is applied to each column in that row. For example we can applyany binary arithmetical operation (+,-,*,/) to an entire row:

In [12]:
```
s = edu["Value"]/100
s.head()
```

Out[12]:
```
0          NaN
1          NaN
2       0.0500
3       0.0503
4       0.0495
Name: Value, dtype: float64
```

However, we can apply any function to a DataFrame or Series just setting its name as argument of the apply method. For example, in the following code, we apply the sqrt function from the NumPy library to perform the square root of each value in the Value column.

In [13]:
```
s = edu["Value"].apply(np.sqrt)
s.head()
```

Out[13]:
```
0            NaN
1            NaN
2       2.236068
3       2.242766
4       2.224860
Name: Value, dtype: float64
```

If we need to design a specific function to apply it, we can write an in-line function, commonly known as a λ-function. A λ-function is a function without a name. It is only necessary to specify the parameters it receives, between the lambda keyword and the colon (:). In the next example, only one parameter is needed, which will bethe value of each element in the Value column. The value the function returns will be the square of that value.

In [14]:

```
s = edu["Value"].apply(lambda d: d**2)
s.head()
```

Out[14]:

```
0            NaN
1            NaN
2        25.0000
3        25.3009
4        24.5025
Name: Value, dtype: float64
```

Another basic manipulation operation is to set new values in our DataFrame. This can be done directly using the assign operator (=) over a DataFrame. For example, to add a new column to a DataFrame, we can assign a Series to a selection of a column that does not exist. This will produce a new column in the DataFrame after all the others. You must be aware that if a column with the same name already exists, the previous values will be overwritten. In the following example, we assign the Series that results from dividing the Value column by the maximum value in the same column to a new column named ValueNorm.

In [15]:

```
edu['ValueNorm'] = edu['Value']/edu['Value'].max()
edu.tail()
```

Out[15]:

	TIME	GEO	Value	ValueNorm
379	2007	Finland	5.90	0.669694
380	2008	Finland	6.10	0.692395
381	2009	Finland	6.81	0.772985
382	2010	Finland	6.85	0.777526
383	2011	Finland	6.76	0.767310

Now, if we want to remove this column from the DataFrame, we can use the drop function; this removes the indicated rows if axis=0, or the indicated columns if axis=1. In Pandas, all the functions that change the contents of a DataFrame, such as the drop function, will normally return a copy of the modified data, instead of overwriting the DataFrame. Therefore, the original DataFrame is kept. If you do notwant to keep the old values, you can set the keyword inplaceto True. By default, this keyword is set to False, meaning that a copy of the data is returned.

In [16]:

```
edu.drop('ValueNorm', axis = 1, inplace = True)
edu.head()
```

Out[16]:

	TIME	GEO	Value
0	2000	European Union (28 countries)	NaN
1	2001	European Union (28 countries)	NaN
2	2002	European Union (28 countries)	5
3	2003	European Union (28 countries)	5.03
4	2004	European Union (28 countries)	4.95

Instead, if what we want to do is to insert a new row at the bottom of the DataFrame,we can use the Pandas appendfunction. This function receives as argumentthe new row, which is represented as a dictionary where the keys are the nameof the columns and the values are the associated value. You must be aware to setting the ignore_index flag in the append method to True, otherwise the index 0is given to this new row, which will produce an error if it already

In [17]:

```
edu = edu.append({"TIME": 2000,"Value": 5.00,"GEO": 'a'},
                 ignore_index = True)
edu.tail()
```

Out[17]:

	TIME	GEO	Value
380	2008	Finland	6.1
381	2009	Finland	6.81
382	2010	Finland	6.85
383	2011	Finland	6.76
384	2000	a	5

Finally, if we want to remove this row, we need to use the drop function again. Now we have to set the axis to 0, and specify the index of the row we want to remove. Since we want to remove the last row, we can use the max function over the indexesto determine which row is.

In [18]:

```
edu.drop(max(edu.index), axis = 0, inplace = True)
edu.tail()
```

Out[18]:

	TIME	GEO	Value
379	2007	Finland	5.9
380	2008	Finland	6.1
381	2009	Finland	6.81
382	2010	Finland	6.85
383	2011	Finland	6.76

The drop()function is also used to remove missing values by applying it overthe result of the isnull() function. This has a similar effect to filtering the NaN values, as we explained above, but here the difference is that a copy of the DataFramewithout the NaNvalues is returned, instead of a view.

In [19]:

```
eduDrop = edu.drop(edu["Value"].isnull(), axis = 0)
eduDrop.head()
```

First, let us read the data:

In[1]:

```
file = open('files/ch03/adult.data', 'r')
def chr_int(a):
    if a.isdigit(): return int(a)
    else: return 0

data = []
for line in file:
     data1 = line.split(', ')
     if len(data1) == 15:
        data.append([chr_int(data1[0]), data1[1],
                     chr_int(data1[2]), data1[3],
                     chr_int(data1[4]), data1[5],
                     data1[6], data1[7], data1[8],
                     data1[9], chr_int(data1[10]),
                     chr_int(data1[11]),
                     chr_int(data1[12]),
                     data1[13], data1[14]
                    ])
```

Checking the data, we obtain:

In[2]:

```
print data[1:2]
```

Out[2]:

```
[[50, 'Self-emp-not-inc', 83311, 'Bachelors', 13,
'Married-civ-spouse', 'Exec-managerial', 'Husband', 'White','Male', 0, 0, 13, 'United-
States', ' <= 50K']]
```

One of the easiest ways to manage data in Python is by using the DataFrame structure, defined in the *Pandas* library, which is a two-dimensional, size-mutable,potentially heterogeneous tabular data structure with labeled axes:

In[3]:

```
df = pd.DataFrame(data)
df.columns = [
    'age', 'type_employer', 'fnlwgt',
    'education', 'education_num', 'marital',
    'occupation',' relationship', 'race',
    'sex', 'capital_gain', 'capital_loss',
    'hr_per_week', 'country', 'income'
    ]
```

The command shapegives exactly the number of data samples (in rows, in this case) and features (in columns):

In[4]:

```
df.shape
```

Out[4]:

```
(32561, 15)
```

Thus, we can see that our dataset contains 32,561 data records with 15 featureseach. Let us count the number of items per country:

In[5]:

```
counts = df.groupby('country').size()
print counts.head()
```

Out[5]:

```
country
? 583
Cambodia 19
Vietnam 67
Yugoslavia 16
```

The first row shows the number of samples with unknown country, followed bythe number of samples corresponding to the first countries in the dataset.

Let us split people according to their gender into two groups: men and women.

In[6]:

```
ml = df[(df.sex == 'Male')]
```

If we focus on high-income professionals separated by sex, we can do:

In[7]:

```
ml1 = df[(df.sex == 'Male') & (df.income=='>50K\n')
    ]
fm = df[(df.sex == 'Female')]
fm1 = df[(df.sex == 'Female') & (df.income=='>50K\n
    ')]
```

Exploratory Data Analysis

The data that come from performing a particular measurement on all the subjects in a sample represent our observations for a single characteristic like country, age, education, etc. These measurements and categories represent a *sample distribution* of the variable, which in turn approximately represents the *population distribution* of the variable. One of the main goals of exploratory data analysis is to visualize and summarize the sample distribution, thereby allowing us to make tentative assumptions about the population distribution.

Summarizing the Data

The data in general can be categorical or quantitative. For categorical data, a simple tabulation of the frequency of each category is the best non-graphical exploration for data analysis. For example, we can ask ourselves what is the proportion of high-income professionals in our database:

In[8]:

```
df1 = df[(df.income=='>50K\n')]
print 'The rate of people with high income is: ',
      int(len(df1)/float(len(df))*100), '%.'
print 'The rate of men with high income is: ',
      int(len(ml1)/float(len(ml))*100), '%.'
print 'The rate of women with high income is: ',
      int(len(fm1)/float(len(fm))*100), '%.'
```

Out[8]:

```
The rate of people with high income is: 24 %.
The rate of men with high income is: 30 %. The rate of women
with high income is: 10 %.
```

Given a quantitative variable, exploratory data analysis is a way to make prelim-inary assessments about the population distribution of the variable using the data of the observed samples. The characteristics of the population distribution of a quanti- tative variable are its *mean*, *deviation*, *histograms*, *outliers*, etc. Our observed datarepresent just a finite set of samples of an often infinite number of possible samples.The characteristics of our randomly observed samples are interesting only to the degree that they represent the population of the data they came from.

Mean

One of the first measurements we use to have a look at the data is to obtain *samplestatistics* from the data, such as the sample mean [1]. Given a sample of n values,
$\{x_i\}, i = 1, \ldots, n$, the *mean*, μ, is the sum of the values divided by the number of values,[2] in other words:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i . \qquad (3.1)$$

The terms mean and *average* are often used interchangeably. In fact, the maindistinction between them is that the mean of a sample is the summary statistic com-puted by Eq. (3.1), while an average is not strictly defined and could be one of manysummary statistics that can be chosen to describe the central tendency of a sample.

In our case, we can consider what the average age of men and women samples inour dataset would be in terms of their mean:

In[9]:

```
print 'The average age of men is: ',
      ml['age'].mean()
print ' The average age  of women is: ',
      fm['age'].mean()

print ' The average age  of high-income men is: ',
      ml1['age'].mean()
print ' The average age of high - income women is : ',
      fm1['age'].mean()
```

Out[9]:

```
The average age of men is: 39.4335474989 The average age of
women is: 36.8582304336
The average age of high-income men is: 44.6257880516
The average age of high-income women is: 42.1255301103
```

This difference in the sample means can be considered initial evidence that thereare differences between men and women with high income!

Comment: Later, we will work with both concepts: the population mean and thesample mean. We should not confuse them! The first is the mean of samples takenfrom the population; the second, the mean of the whole population.

Sample Variance

The mean is not usually a sufficient descriptor of the data. We can go further by knowing two numbers: mean and *variance*. The variance σ^2 describes the spread ofthe data and it is defined as follows:

$$\sigma^2 = \frac{1}{n}\sum_{i} (x_i - \mu)^2. \qquad (3.2)$$

The term $(x_i \ \mu)$ is called the *deviation* from the mean, so the variance is the mean squared deviation. The square root of the variance, σ, is called the *standard deviation*. We consider the standard deviation, because the variance is hard to interpret (e.g., ifthe units are grams, the variance is in grams squared).

Let us compute the mean and the variance of hours per week men and women inour dataset work:

In[10]:

```
ml_mu  =  ml['age'].mean()
fm_mu  =  fm['age'].mean()
ml_var  =  ml['age'].var()
fm_var  =  fm['age'].var()
ml_std  =  ml['age'].std()
fm_std  =  fm['age'].std()
print 'Statistics of age for men: mu:',
      ml_mu, 'var:', ml_var, 'std:', ml_std
```

In[17]:
```
df2 = df.drop(df.index[
    (df.income == '>50K\n') &
    (df['age'] > df['age'].median() + 35) &
    (df['age'] > df['age'].median() -15)
    ])
ml1_age = ml1['age']
fm1_age = fm1['age']

ml2_age = ml1_age.drop(ml1_age.index[
    (ml1_age > df['age'].median() + 35) &
    (ml1_age > df['age'].median() - 15)
    ])
fm2_age = fm1_age.drop(fm1_age.index[
    (fm1_age > df['age'].median() + 35) &
    (fm1_age > df['age'].median() - 15)
    ])
```

We can check how the mean and the median changed once the data were cleaned:

In[18]:
```
mu2ml = ml2_age.mean()
std2ml = ml2_age.std()
md2ml = ml2_age.median()
mu2fm = fm2_age.mean()
std2fm = fm2_age.std()
md2fm = fm2_age.median()

print "Men statistics:"
print "Mean:", mu2ml, "Std:", std2ml
print "Median:", md2ml
print "Min:", ml2_age.min(), "Max:", ml2_age.max()

print "Women statistics:"
print "Mean:", mu2fm, "Std:", std2fm
print "Median:", md2fm
print "Min:", fm2_age.min(), "Max:", fm2_age.max()
```

Out[18]:
```
Men statistics: Mean: 44.3179821239 Std: 10.0197498572 Median:
44.0 Min: 19 Max: 72
Women statistics: Mean: 41.877028181 Std: 10.0364418073 Median:
41.0 Min: 19 Max: 72
```

Let us visualize how many outliers are removed from the whole data by:

In[19]:
```
plt.figure(figsize = (13.4, 5))
df.age[(df.income == '>50K\n')]
      .plot(alpha = .25, color = 'blue')
df2.age[(df2.income == '>50K\n')]
      .plot(alpha = .45, color = 'red')
```

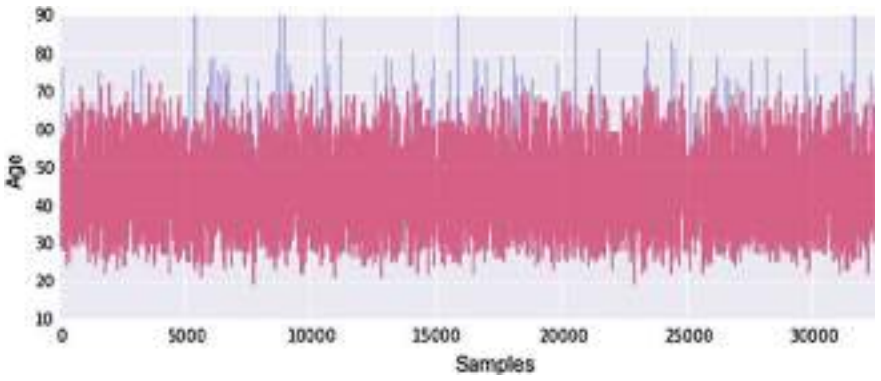


Fig. 3.4 The *red* shows the cleaned data without the considered outliers (in *blue*)

Figure 3.4 shows the outliers in blue and the rest of the data in red. Visually, wecan confirm that we removed mainly outliers from the dataset.

Next we can see that by removing the outliers, the difference between the popula-tions (men and women) actually decreased. In our case, there were more outliers inmen than women. If the difference in the mean values before removing the outliersis 2.5, after removing them it slightly decreased to 2.44:

In[20]:
```
print 'The mean difference with outliers is: %4.2f.
    '
       % (ml_age.mean() - fm_age.mean())
print 'The mean difference without outliers is:
    %4.2f.'
       % (ml2_age.mean() - fm2_age.mean())
```

Out[20]:
```
The mean difference with outliers is: 2.58.
The mean difference without outliers is: 2.44.
```

Let us observe the difference of men and women incomes in the cleaned subsetwith some more details.

In[21]:
```
countx, divisionx = np.histogram(ml2_age, normed =
    True)
county, divisiony = np.histogram(fm2_age, normed =
    True)

val = [(divisionx[i] + divisionx[i+1])/2
        for i in range(len(divisionx) - 1)]
plt.plot(val, countx - county, 'o-')
```

The results are shown in Fig. 3.5. One can see that the differences between male and female values are slightly negative before age 42 and positive after it. Hence, women tend to be promoted (receive more than 50 K) earlier than men.

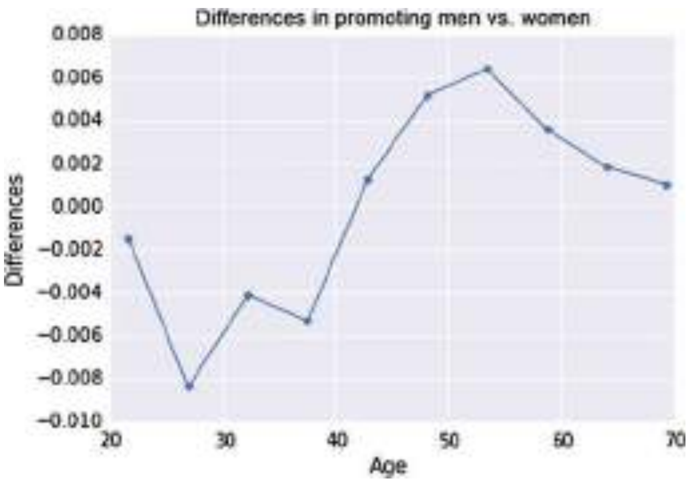


Fig. 3.5 Differences in high-income earner men versus women as a function of age

Measuring Asymmetry: Skewness and Pearson's Median Skewness Coefficient

For univariate data, the formula for *skewness* is a statistic that measures the asym-metry of the set of *n* data samples, x_i:

$$g_1 = \frac{1}{n}\frac{\sum_i (x_i - \mu^3)}{\sigma^3}, \qquad (3.3)$$

where μ is the mean, σ is the standard deviation, and *n* is the number of data points.

Negative deviation indicates that the distribution "skews left" (it extends furtherto the left than to the right). One can easily see that the skewness for a normal distribution is zero, and any symmetric data must have a skewness of zero. Note that skewness can be affected by outliers! A simpler alternative is to look at the relationship between the mean μ and the median μ_{12}.

In[22]:

```
def  skewness(x):
    res = 0
    m = x.mean()
    s = x. std ()
    for i in x:
        res += (i-m) * (i-m) * (i-m)
    res /= (len(x) * s * s * s)
    return res

print "Skewness of the male population = ",
      skewness(ml2_age)
print "Skewness of the female population is = ",
      skewness(fm2_age)
```

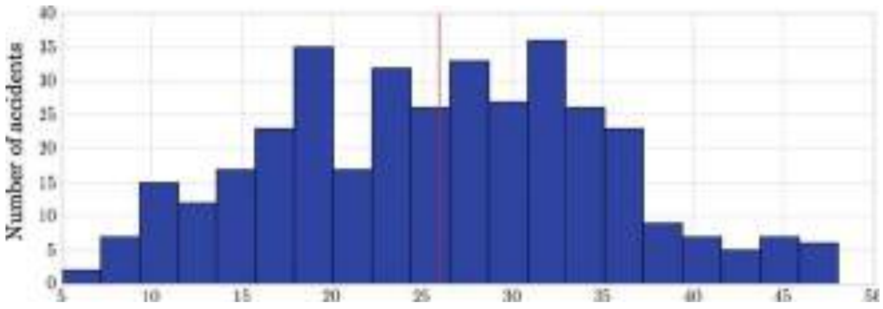


Fig. 4.2 Mean sampling distribution by bootstrapping. In *red*, the mean value of this distribution

In [4]:

```
def meanBootstrap(X, numberb):
    x = [0]*numberb
    for i in range(numberb):
        sample = [X[j]
                  for j
                  in np.random.randint(len(X), size=len(X))
                 ]
        x[i] = np.mean(sample)
    return x
m = meanBootstrap(accidents, 10000)
print "Mean estimate:", np.mean(m)
```

Out[4]:

```
Mean estimate: 25.9094
```

The basic idea of the bootstrapping method is that the observed sample contains sufficient information about the underlying distribution. So, the information we canextract from resampling the sample is a good approximation of what can be expected from resampling the population.

The bootstrapping method can be applied to other simple estimates such as the median or the variance and also to more complex operations such as estimates of censored data.[3]

Confidence Intervals

A point estimate Θ, such as the sample mean, provides a *single plausible value fora parameter*. However, as we have seen, a point estimate is rarely perfect; usually there is some error in the estimate. That is why we have suggested using the standard error as a measure of its variability.

Instead of that, a next logical step would be to provide *a plausible range of values*for the parameter. A plausible range of values for the sample parameter is called a *confidence interval*.

```
Out[8]: 2010 accident rate estimate: 24.8109
        2013 accident rate estimate: 25.9095
CI for 2013: [24.9751, 26.8440]
```

Because the 2010 accident rate estimate does not fall in the range of plausible values of 2013, we say the alternative hypothesis cannot be discarded. That is, it cannot be ruled out that in 2013 the mean rate of traffic accidents in Barcelona washigher than in 2010.

Interpreting CI Tests

Hypothesis testing is built around rejecting or failing to reject the null hypothesis. That is, we do not reject H_0 unless we have strong evidence against it. But what precisely does strong evidence mean? As a general rule of thumb, for those cases where the null hypothesis is actually true, we do not want to incorrectly reject H_0 more than 5% of the time. This corresponds to a *significance level* of $\alpha = 0.05$. In this case, the correct interpretation of our test is as follows:

> If we use a 95% confidence interval to test a problem where the null hypothesis is true, wewill make an error whenever the point estimate is at least 1.96 standard errors away from thepopulation parameter. This happens about 5% of the time (2.5% in each tail).

Testing Hypotheses Using *p*-Values

A more advanced notion of *statistical significance* was developed by R.A. Fisher in the 1920s when he was looking for a test to decide whether variation in crop yields was due to some specific intervention or merely random factors beyond experimental control.

Fisher first assumed that fertilizer caused no difference (*null hypothesis*) and thencalculated P, the probability that an observed yield in a fertilized field would occurif fertilizer had no real effect. This probability is called the *p-value*.

The *p*-value is the probability of observing data at least as favorable to the alter-native hypothesis as our current dataset, if the null hypothesis is true. We typically use a summary statistic of the data to help compute the *p*-value and evaluate the hypotheses.

Usually, if P is less than 0.05 (the chance of a fluke is less than 5%) the result is declared *statistically significant*.

It must be pointed out that this choice is rather arbitrary and should not be takenas a scientific truth.

The goal of classical hypothesis testing is to answer the question, "*Given a sample and an apparent effect, what is the probability of seeing such an effect by chance?*"Here is how we answer that question:

- The first step is to quantify the size of the apparent effect by choosing a test statistic. In our case, the apparent effect is a difference in accident rates, so a natural choicefor the test statistic is the **difference in means between the two periods**.

- The second step is to define a *null hypothesis*, which is a model of the system based on the assumption that the apparent effect is not real. In our case, the null hypothesis is that there is no difference between the two periods.
- The third step is to compute a *p-value*, which is the probability of seeing the apparent effect if the null hypothesis is true. In our case, we would compute the difference in means, then compute the probability of seeing a difference as big, orbigger, under the null hypothesis.
- The last step is to *interpret the result*. If the *p*-value is low, the effect is said to be *statistically significant*, which means that it is unlikely to have occurred by chance. In this case we infer that the effect is more likely to appear in the larger population.

In our case, the test statistic can be easily computed:

In [9]:
```
m= len(counts2010)
n= len(counts2013)
p = (counts2013.mean() - counts2010.mean())
print 'm:', m, 'n:', n
print 'mean difference: ', p
```

Out[9]:
```
m: 365 n: 365
mean difference: 1.0986
```

To approximate the *p*-value , we can follow the following procedure:

1. Pool the distributions, generate samples with size n and compute the differencein the mean.
2. Generate samples with size n and compute the difference in the mean.
3. Count how many differences are larger than the observed one.

In [10]:
```
# pooling distributions
x = counts2010
y = counts2013
pool = np.concatenate([x, y])
np.random.shuffle(pool)

#sample generation
import random
N = 10000 # number of samples
diff = range(N)
for i in range(N):
    p1 = [random.choice(pool) for _ in xrange(n)]
    p2 = [random.choice(pool) for _ in xrange(n)]
    diff[i] = (np.mean(p1) - np.mean(p2))
```

In [11]:
```
# counting differences larger than the observed one
diff2 = np.array(diff)
w1 = np.where(diff2 > p)[0]

print 'p-value (Simulation)=', len(w1)/float(N),
      '(', len(w1)/float(N)*100 ,'%)', 'Difference =', p
if (len(w1)/float(N)) < 0.05:
    print 'The effect is likely'
else:
    print 'The effect is not likely'
```

Out[11]:
```
p-value (Simulation)= 0.0485 ( 4.85%) Difference = 1.098The effect is likely
```

Interpreting *P*-Values

A *p*-value is the probability of an observed (or more extreme) result arising only from chance.

If P is less than 0.05, there are two possible conclusions: there is a real effect or the result is an improbable fluke. *Fisher's method offers no way of knowing which isthe case*.

We must not confuse the odds of getting a result (if a hypothesis is true) with the odds of favoring the hypothesis if you observe that result. If P is less than 0.05,we cannot say that this means that it is 95% certain that the observed effect is real and could not have arisen by chance. Given an observation E and a hypothesis H, $P(E|H)$ and $P(H|E)$ are not the same!

Another common error equates *statistical significance* to *practical importance/ relevance*. When working with large datasets, we can detect statistical significancefor small effects that are meaningless in practical terms.

We have defined the effect as *a difference in mean as large or larger than δ, considering the sign*. A test like this is called *one sided*.

If the relevant question is whether *accident rates are different*, then it makes senseto test the absolute difference in means. This kind of test is called two sided becauseit counts both sides of the distribution of differences.

Direct Approach

The formula for the standard error of the absolute difference in two means is similarto the formula for other standard errors. Recall that the standard error of a single mean can be approximated by:

$$SE_{\bar{x}_1} = \frac{\sigma_1}{\sqrt{n_1}}$$

The standard error of the difference of two sample means can be constructed fromthe standard errors of the separate sample means:

$$SE_{\bar{x}_1-\bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

This would allow us to define a direct test with the 95% confidence interval.

```
from sklearn import neighbors
from sklearn import datasets
# Create an instance of K-nearest neighbor classifier
knn = neighbors.KNeighborsClassifier(n_neighbors = 11)
# Train the classifier
knn.fit(x, y)
# Compute the prediction according to the model
yhat = knn.predict(x)
# Check the result on the last example
print 'Predicted value: ' + str(yhat[-1]),
      ', real target: ' + str(y[-1])
```

Out[3]: Predicted value: -1.0 , real target: -1.0

The basic measure of performance of a classifier is its *accuracy*. This is defined as the number of correctly predicted examples divided by the total amount of examples. Accuracy is related to the error as follows: $acc = 1 - err$.

$$acc = \frac{\text{Number of correct predictions}}{n}$$

Each estimator has a score()method that invokes the default scoring metric. In the case of k-nearest neighbors, this is the classification accuracy.

In [4]:

```
knn.score(x,y)
```

Out[4]: 0.83164251207729467

It looks like a really good result. But how good is it? Let us first understand a little bit more about the problem by checking the distribution of the labels.

Let us load the dataset and check the distribution of labels:

In [5]:

```
plt.pie(np.c_[np.sum(np.where(y == 1, 1, 0)),
         np.sum(np.where(y == -1, 1, 0))][0],
         labels = ['Not fully funded','Full amount'],
         colors = ['r', 'g'],shadow = False,
         autopct = '%.2f' )
plt.gcf().set_size_inches((7, 7))
```

with the result observed in Fig. 5.1.

Note that there are far more positive labels than negative ones. In this case, the dataset is referred to as *unbalanced*.[5] This has important consequences for a classifier as we will see later on. In particular, a very simple rule such as always predict the

[5]The term unbalanced describes the condition of data where the ratio between positives and negatives is a small value. In these scenarios, always predicting the majority class usually yields accurate performance, though it is not very informative. This kind of problems is very common when we want to model unusual events such as rare diseases, the occurrence of a failure in machinery, fraudulent credit card operations, etc. In these scenarios, gathering data from usual events is very easy but collecting data from unusual events is difficult and results in a comparatively small dataset.

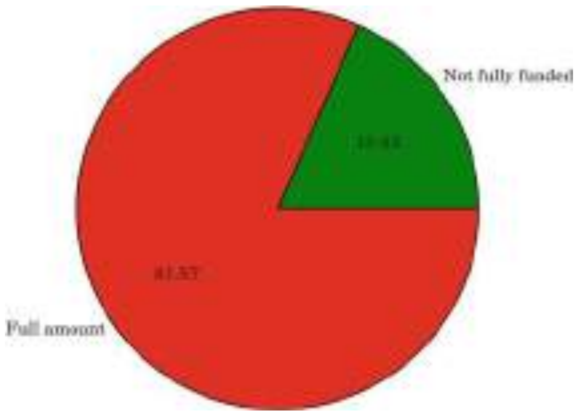


Fig. 5.1 Pie chart showing the distribution of labels inthe dataset

majority class, will give us good performance. In our problem, always predicting that the loan will be fully funded correctly predicts 81.57% of the samples. Observethat this value is very close to that obtained using the classifier.

Although accuracy is the most normal metric for evaluating classifiers, there arecases when the business value of correctly predicting elements from one class is different from the value for the prediction of elements of another class. In those cases, accuracy is not a good performance metric and more detailed analysis is needed. The *confusion matrix* enables us to define different metrics considering such scenarios. The confusion matrix considers the concepts of the classifier outcome and the actual ground truth or gold standard. In a binary problem, there are four possiblecases:

- *True positives (TP)*: When the classifier predicts a sample as positive and it really is positive.
- *False positives (FP)*: When the classifier predicts a sample as positive but in fact it is negative.
- *True negatives (TN)*: When the classifier predicts a sample as negative and it really is negative.
- *False negatives (FN)*: When the classifier predicts a sample as negative but in fact it is positive.

We can summarize this information in a matrix, namely the confusion matrix, asfollows:

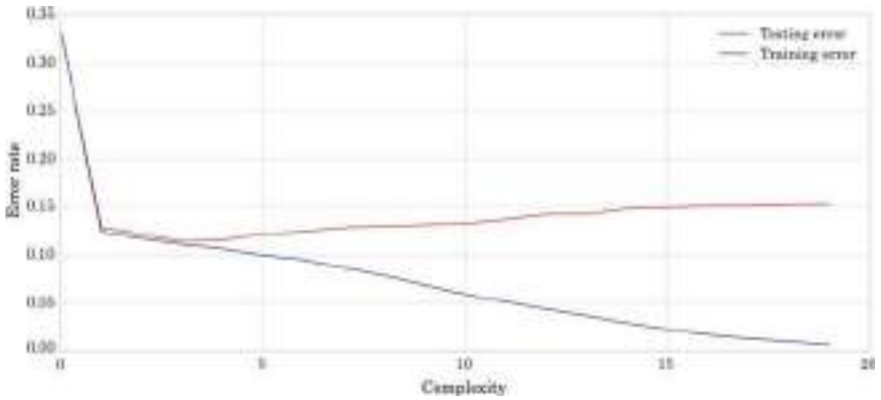


Fig. 5.7 Learning curves (training and test errors) for a fixed number of data samples, as the complexity of the decision tree increases

- With a low degree of complexity, the training and test errors converge to the biassooner/with fewer data.
- Moreover, with a low degree of complexity, the error of convergence is larger thanwith increased complexity.

The value both errors converge towards is also called the *bias*; and the difference between this value and the test error is called the *variance*. The *bias/variance* decomposition of the learning curve is an alternative approach to the training and generalization view.

Let us now plot the learning behavior for a fixed number of examples with respectto the complexity of the model. We may use the same data but now we will changethe maximum depth of the decision tree, which governs the complexity of the model.Observe in Fig. 5.7 that as the complexity increases the training error is reduced; but above a certain level of complexity, the test error also increases. This effect is called *overfitting*. We may enact several cures for overfitting:

- Observe that models are usually parameterized by some hyperparameters. Select-ing the complexity is usually governed by some such parameters. Thus, we are faced with a model selection problem. A good heuristic for selecting the model isto choose the value of the hyperparameters that yields the smallest estimated testerror. Remember that this can be done using cross-validation.
- We may also change the formulation of the objective function to penalize complex models. This is called *regularization*. Regularization accounts for estimating the value of Ω in our out-of-sample error inequality. In other words, it models the complexity of the technique. This usually becomes implicit in the algorithm but has huge consequences in real applications. The most common regularization strategies are as follows:

  - L2 weight regularization: Adding an L2 penalization term to the weights of a weight-controlled model implies looking for solutions with small weight values. Intuitively, adding an L2 penalization term can be seen as a surrogate for the notion of smoothness. In this sense, a low complexity model means a very smooth model.
  - L1 weight regularization: Adding an L1 regularization term forces sparsity in the weights of the model. In this sense, a low complexity model means a modelwith few components or few active terms.

  These terms are added to the objective function. They trade off with the error function in the objective and are governed by a hyperparameter. Thus, we still have to select this parameter by means of model selection.
- We can use "ensemble techniques". A third cure for overfitting is to use ensemble techniques. The best known are *bagging* and *boosting*.

Training, Validation and Test

Going back to our problem, we have to select a model and control its complexity according to the number of training data. In order to do this, we can start by usinga model selection technique. We have seen model selection before when we wantedto compare the performance of different classifiers. In that case, our best bet was to select the classifier with the smallest E_{out}. Analogous to model selection, we may think of selecting the best hyperparameters as choosing the classifier with parameters that performs the best. Thus, we may select a set of hyperparameter values and usecross-validation to select the best configuration.

The process of selecting the best hyperparameters is called *validation*. This intro-duces a new set into our simulation scheme; we now need to divide the data we haveinto three sets: training, validation, and test sets. As we have seen, the process of assessing the performance of the classifier by estimating the generalization error iscalled testing. And the process of selecting a model using the estimation of the gen-eralization error is called validation. There is a subtle but critical difference betweenthe two and we have to be aware of it when dealing with our problem.

- Test data is used exclusively for assessing performance at the end of the processand will never be used in the learning process.[8]
- Validation data is used explicitly to select the parameters/models with the best performance according to an estimation of the generalization error. This is a formof learning.
- Training data are used to learn the instance of the model from a model class.

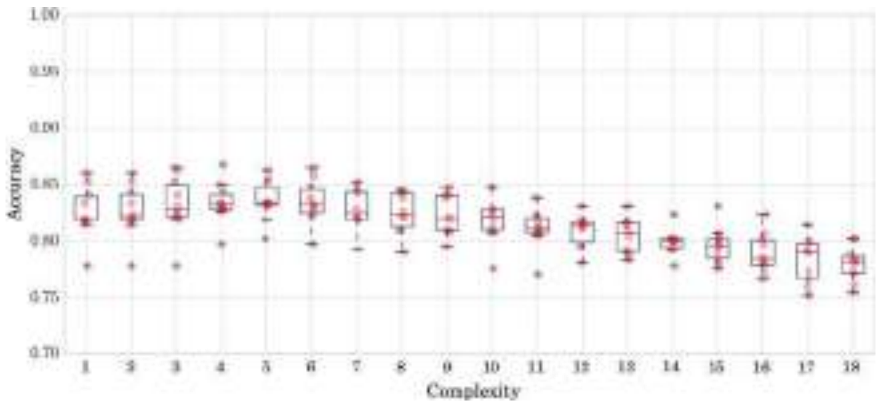


Fig. 5.8 Box plot showing accuracy for different complexities of the decision tree

In [14]:
```
# Create a 10-fold  cross-validation  set
kf  =  cross_validation.KFold(n  =  y.shape[0],
                              n_folds  = 10,
                              shuffle  = True,
                              random_state  = 0)

# Search  for  the  parameter  among  the  following:
C = np.arange(2,  20,)

acc  =  np.zeros((10,  18))
i = 0
for train_index,  val_index  in  kf:
    X_train, X_val  =  X[train_index],  X[val_index]
    y_train,  y_val  =  y[train_index],  y[val_index]
    j = 0
    for  c  in  C:
        dt  =  tree.DecisionTreeClassifier(
            min_samples_leaf  = 1,
            max_depth  = c)
        dt.fit(X_train, y_train)
        yhat  =  dt.predict(X_val)
        acc[i][j]  =  metrics.accuracy_score(yhat,  y_val)
        j = j  + 1
    i = i + 1
```

Checking Fig. 5.8, we can see that the best average accuracy is obtained by the fifth model, a maximum depth of 6. Although we can report that the best accuracy is estimated to be found with a complexity value of 6, we cannot say anything about the value it will achieve. In order to have an estimation of that value, we need to run the model on a new set of data that are completely unseen, both in training and in model selection (the model selection value is positively biased). Let us put everything together. We will be considering a simple train_test split for testing purposes and then run cross-validation for model selection.

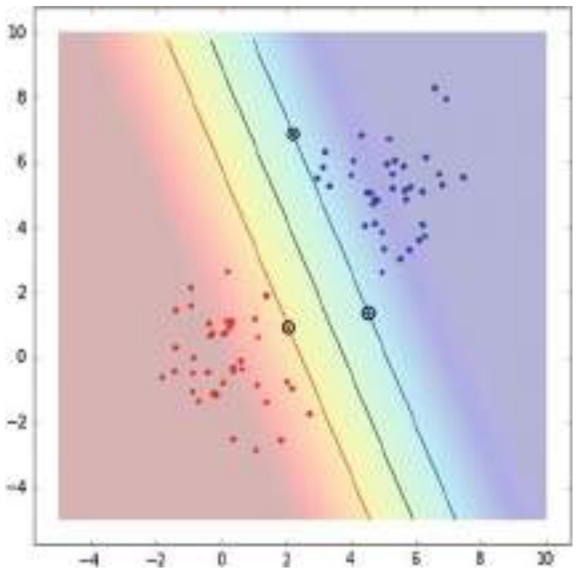


Fig. 5.9 Support vector machine decision boundaryand the support vectors

separating boundary will have a point of a class closer to it than this one. The figurealso shows the closest points of the classes to the boundary. These points are called*support vectors*. In fact, the boundary only depends on those points. If we remove any other point from the dataset, the boundary remains intact. However, in general,if any of these special points is removed the boundary will change.

A Brief Note on Deriving Hard Margin Support Vector Machines In order to understand the model, we have to be able to approximately derive its for-mulation. For this purpose it is important to understand a couple of things about basic geometry of a hyperplane. A hyperplane in $\mathbf{R}^d$ is defined as an affine combination ofthe variables: π $a^T x$ b 0. A hyperplane splits the space into two half-spaces.The evaluation of the equation of the hyperplane on any element belonging to one of the half-spaces is a positive value. It is a negative value for all the elements in theother half-space. The distance of a point $x \in \mathbf{R}^d$ to the hyperplane π is

$$d(x,\pi) = \frac{|a^T x + b|}{\|a\|_2}$$

Given a binary classification problem with training data $D = \{(x_i, y_i)\}$, $i = 1 \ldots N$, $y_i \in \{+1, -1\}$, consider $S \subseteq D$ the subset of all data points belonging to class +1, $S = \{x_i \mid y_i = +1\}$, and $R = \{x_i \mid y_i = -1\}$ its complement.

The new model becomes:

$$\begin{aligned} \text{minimize} \quad & \|a\|_2/2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i(a^T x_i + b) \geq 1 - \xi_i, \; i = 1 \ldots N \\ & \xi_i \geq 0 \end{aligned}$$

where C is the trade-off parameter that roughly balances the rates of margin and misclassification. This formulation is also called *soft-margin SVM*.

The larger the C value is, the more importance one gives to the error, i.e., the method will be more accurate according to the data at hand, at the cost of being moresensitive to variations of the data.

The decision boundary of most problems cannot be well approximated by a linearmodel. In SVM, the extension to the nonlinear case is handled by means of kernel theory. In a pragmatic way, a kernel can be referred to as any function that capturesthe similarity between any two samples in the training set. The kernel has to be a positive semi-definite function as follows:

- *Linear kernel*:
$$k(x_i, x_j) = x_i^T x_j$$
- *Polynomial kernel*:
$$k(x_i, x_j) = (1 + x_i^T x_j)^p$$
- *Radial Basis Function kernel*:
$$k(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

Note that selecting a polynomial or a Radial Basis Function kernel means that we have to adjust a second parameter p or σ, respectively. As a practical summary, the SVM method will depend on two parameters (C, γ) that have to be chosen carefully using cross-validation to obtain the best performance.

Random Forest

Random Forest (RF) is the other technique that is considered in this work. RF is an ensemble technique. Ensemble techniques rely on combining different classifiersusing some aggregation technique, such as majority voting. As pointed out earlier, ensemble techniques usually have good properties for combating overfitting. In this case, the aggregation of classifiers using a voting technique reduces the variance of the final classifier. This increases the robustness of the classifier and usually achievesa very good classification performance. A critical issue in the ensemble of classifiers is that for the combination to be successful, the errors made by the members of the ensemble should be as uncorrelated as possible. This is sometimes referred to in the satisfied and engaged with our service, so they keep using it. Analyzing the confusion matrix we can

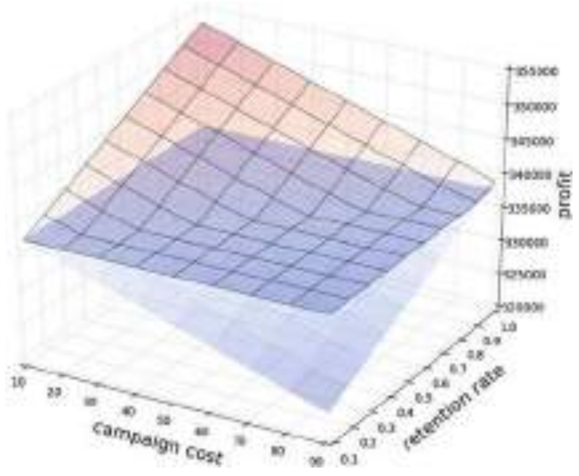


Fig. 5.10 Surfaces for two different campaign and attraction factors. The horizontal plane correspondsto the profit if no campaign is launched. The slanted plane is the profit for a certain confusion matrix

give precise meaning to different concepts in this campaign. The real positive set $(TP + FN)$ consists of the number of clients that are fully funded. According to our assumption, each of these clients generates a profit of 100 units. The total profitis 100 $(TP + FN)$. The campaign to attract investors will be cast considering all the clients we predict are not fully funded. These are those that the classifier predict as negative, i.e., $(FN + TN)$. However, the campaign will only have an effect on the investors/clients that are actually not funded, i.e., TN ; and we expect to attract a certain fraction β of them. After deploying our campaign, a simplified model of theexpected profit is as follows:

$$100 \cdot (TP + FN) - \alpha(TN + FN) + 100\beta TN$$

When optimizing the classifier for accuracy, we do not consider the business needs. In this case, optimizing an SVM using cross-validation for different parameters of the C and γ, we have an accuracy of 85.60% and a confusion matrix with the followingvalues:

$$\begin{array}{cc} 3371. & 590. \\ 6. & 173. \end{array}$$

If we check how the profit changes for different values of α and β, we obtain the plot in Fig. 5.10. The figure shows two hyperplanes. The horizontal plane is the expected profit if the campaign is not launched, i.e., 100 $(TP + FN)$. The other hyperplanerepresents the profit of the campaign for different values of α and β using a particular classifier. Remember that the cost of the campaign is given by α, and the success rate of the campaign is represented by β. For the campaign to be successful we would like to select values for both parameters so that the profit of the campaign is larger than the cost of launching it. Observe in the figure that certain costs and attraction rates result in losses.

Table 5.1 Different configurations of classifiers and their respective profit rates and accuracies

	Max profit rate (%)	Profit rate at 60% (%)	Accuracy (%)
Random forest	4.41	2.41	87.87
SVM {1 : 1}	4.59	2.54	85.60
SVM {1 : 2}	4.52	2.50	85.60
SVM {1 : 4}	4.30	2.28	83.81
SVM {1 : 8}	10.69	3.57	52.51
SVM {1 : 16}	10.68	2.88	41.40

how much a misclassification in one class counts with respect to a misclassificationin another. Figure 5.11 shows the different landscapes for different configurations of the SVM classifier and RF.

In order to frame the problem, we consider a very successful campaign with a 60% investor attraction rate. We can ask several questions in this scenario:

- What is the maximum amount to be spent on the campaign?
- How much will I gain?
- From all possible configurations of the classifier, which is the most profitable?
- Is it the one with the best accuracy?

Checking the values in Fig. 5.11, we find the results collected in Table 5.1. Observe that the most profitable campaign with 60% corresponds to a classifier that considers the cost of mistaking a sample from the non-fully funded class eight times larger than the one from the other class. Observe also that the accuracy in that case is muchworse than in other configurations.

The take-home idea of this section is that business needs are often not aligned with the notion of accuracy. In such scenarios, the confusion matrix values have specificmeanings. This must be taken into account when tuning the classifier.

may tackle many more different settings. For example, we may have different target labels for a single example; this is called multilabel learning. Or, data can come from streams or be time dependent; in these settings, sequential learning or sequence learning can be the methods of choice. Moreover, each data example can be a non-vector or have a variable size, such as a graph, a tree, or a string. In such scenarios kernel learning or structural learning may be used. During these last years we are also seeing the revival of neural networks under the name of deep learning and achieving impressive results in different domains such as computer vision or natural language processing. Nonetheless, all of these methods will behave as explained in this chapter and most of the lessons learned here can be readily applied to these techniques.

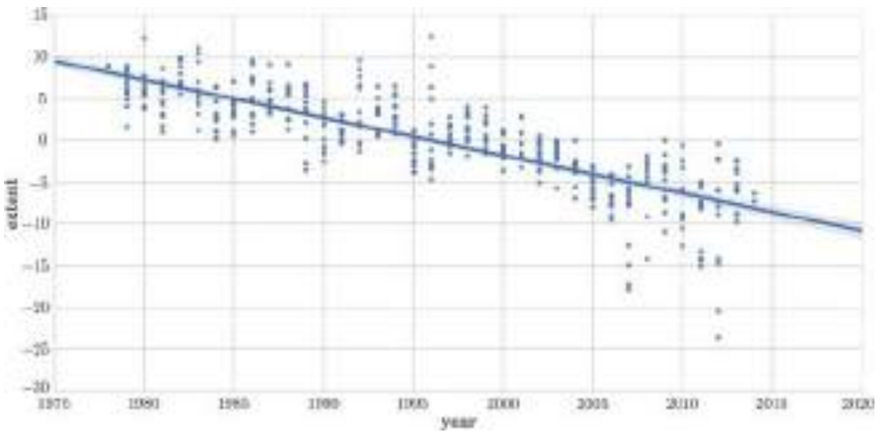


Fig. 6.4 Regression model fitting sea ice extent data for all months by year using lmplot

the two variables. The regression line is plotted with a 95% confidence band to givean impression of the uncertainty in the model.

In this figure, we can observe that the data show a long-term negative trend overyears. The negative trend can be attributed to global warming, although there is alsoa considerable amount of variation from year to year.

Up until here, we have qualitatively shown the linear regression using a useful visu- alization tool. We can also analyze the linear relationship in the data using the *Scikit- learn* library, which allows a quantitative evaluation. As was explained in the previous chapter, Scikit-learn provides an object-oriented interface centered around the con-cept of an estimator. The sklearn.linear_model.LinearRegression estimator sets the state of the estimator based on the training data using the function fit. Moreover, it allows the user to specify whether to fit an intercept term in the object construction. This is done by setting the corresponding constructor argumentsof the estimator object as follows:

In [6]:
```
from sklearn . linear_model import LinearRegression
est = LinearRegression(fit_intercept = True)
```

During the fitting process, the state of the estimator is stored in instance attributes that have a trailing underscore (‘_’). For example, the coefficients of a LinearRegression estimator are stored in the attribute coef_. We fit a regres- sion model using years as variables (**x**) and the extent values as the response (**y**).

In [7]:
```
x = ice2[['year']]
y = ice2[['extent']]
est.fit(x, y)
print "Coefficients:", est.coef_
print "Intercept:", est.intercept_
```

considering nonlinear transformations $\varphi(\cdot)$ of the variables:

$$\mathbf{y} = a_1\varphi(\mathbf{x}_1) + \cdots + a_d\varphi(\mathbf{x}_d)$$

This model is called *polynomial regression* and it is a popular nonlinear regression technique which models the relationship between the response and the variablesas an p-th order polynomial. The higher the order of the polynomial, the more complex the functions you can fit. However, using higher-order polynomial can involve *computational complexity* and *overfitting*. Overfitting occurs when a model fits the characteristics of the training data and loses the capacity to generalize fromthe seen to predict the unseen.

Sparse Model

Often, in real problems, there are uninformative variables in the data which prevent proper modeling of the problem and thus, the building of a correct regression model. In such cases, a feature selection process is crucial to select only the informative features and discard non-informative ones. This can be achieved by *sparse methods* which use a penalization approach, such as *LASSO* (least absolute shrinkage and selection operator) to set some model coefficients to zero (thereby discarding thosevariables). Sparsity can be seen as an application of Occam's razor: prefer simpler models to complex ones.

Given the set of samples $(\mathbf{X}, \mathbf{y})$, the objective of a sparse model is to minimize the SSE through a restriction (or penalty):

$$\frac{1}{2n}||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \alpha||\mathbf{w}||_1,$$

where $||\mathbf{w}||_1$ is the $L1$-norm of the parameter vector $\mathbf{w} = (a_0, \ldots, a_d)$.

Practical Case: Prediction of the Price of a New Housing Market

In this practical case we want to solve the question: Can we predict the price of a new market given any of its attributes?

We will use the Boston housing dataset from Scikit-learn, which provides recorded measurements of 13 attributes of housing markets around Boston, as well as the median house price.[3] Once we load the dataset (506 instances), the description of the dataset can easily be shown by printing the field DESCR. The data (**x**), feature names, and target (**y**) are stored in other fields of the dataset.

We first consider the task of predicting median house values in the Boston area using as the variable one of the attributes, for instance, LSTAT, defined as the "pro-portion of lower status of the population".

Seaborn visualization can be used to show this linear relationships easily:

[3]Copy of UCI ML housing dataset: http://archive.ics.uci.edu/ml/datasets/Housing.

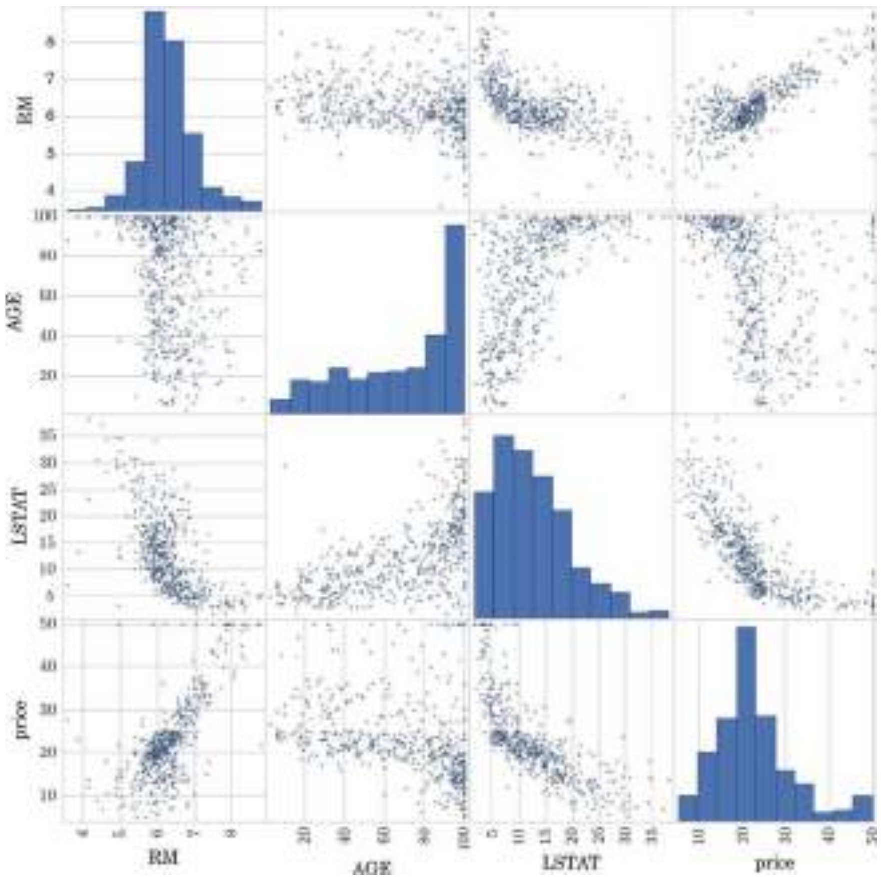


Fig. 6.8 Scatter plot of Boston housing dataset

```
Out[14]:Training and testing set sizes (253, 13) (253, 13)
        Coeff and intercept: [ 1.20133313 0.02449686 0.00999508
        0.42548672 -8.44272332 8.87767164 -0.04850422 -1.11980855
        0.20377571 -0.01597724 -0.65974775 0.01777057 -0.11480104]
        -10.0174305829
        Testing Score: -2.24420202674
        Training MSE: 9.98751732546
        Testing MSE: 302.64091133
```

We can see that all the coefficients obtained are different from zero, meaning thatno variable is discarded. Next, we try to build a sparse model to predict the price using the most important factors and discarding the non-informative ones. To do this, we can create a LASSO regressor, forcing zero coefficients.

In [15]:

```
regr_lasso = linear_model.Lasso(alpha = .3)
regr_lasso.fit(X_train, y_train) print 'Coeff and intercept:
    ',regr_lasso.coef_
print 'Tesing Score:', regr_lasso.score(X_test,
y_test) print 'Training MSE: ',
        np.mean((regr_lasso.predict(X_train)  - y_train)**2)
print 'Testing MSE: ',
        np.mean((regr_lasso.predict(X_test) - y_test)**2)
```

Out[15]:

```
Coeff and intercept: [ 0. 0.01996512 -0. 0. -0. 7.69894744
-0.03444803 -0.79380636 0.0735163 -0.0143421 -0.66768539
0.01547437 -0.22181817] -6.18324183615
Testing Score: 0.501127529021
Training MSE: 10.7343110095
Testing MSE: 46.5381680949
```

It can now be seen that the result of the model fitting for a set of sparse coefficientsis much better than before (using all the variables), with the score increasing from
–2.24 to 0.5. This demonstrates that four of the initial variables are not importantfor the prediction and in fact they confuse the regressor.

With the LASSO result, we can also emphasize the most important factors for determining the price of a new market, based on the coefficient values:

In [16]:

```
ind = np.argsort(np.abs(regr_lasso.coef_))
print 'Ordered variable (from less to more important):',
        boston.feature_names[ind]
```

Out[16]:

```
Ordered variable (from less to more important): ['CRIM' 'INDUS' 'CHAS' 'NOX' 'TAX' 'B' 'ZN' 'AGE'
'RAD' 'LSTAT' 'PTRATIO' 'DIS''RM']
```

There are also other strategies for feature selection. For instance, we can select=the k 5 best features, according to the k highest scores, using the function SelectKBestfrom Scikit-learn:

In [17]:

```
import sklearn.feature_selection as fs
selector = fs.SelectKBest(score_func = fs.f_regression,
                          k = 5)
selector.fit_transform(X_train, y_train) per
selector.fit(X_train,y_train)
print 'Selected features:',
        zip(selector.get_support(), boston.feature_names)
```

Out[17]:

```
Selected features: [(False, 'CRIM'), (False, 'ZN'), (True,
'INDUS'), (False, 'CHAS'), (False, 'NOX'), (True, 'RM'), (True,
'AGE'), (False, 'DIS'), (False, 'RAD'), (False, 'TAX'), (True,'PTRATIO'), (False, 'B'), (True, 'LSTAT')]
```

The set of selected features is now different, since the criterion has changed. However, three of the most important features: RM, PTRATIO, and LSTAT.

In order to evaluate the prediction, it could be interesting to visualize the targetand predicted responses in a scatter plot, as it is done in the next code:

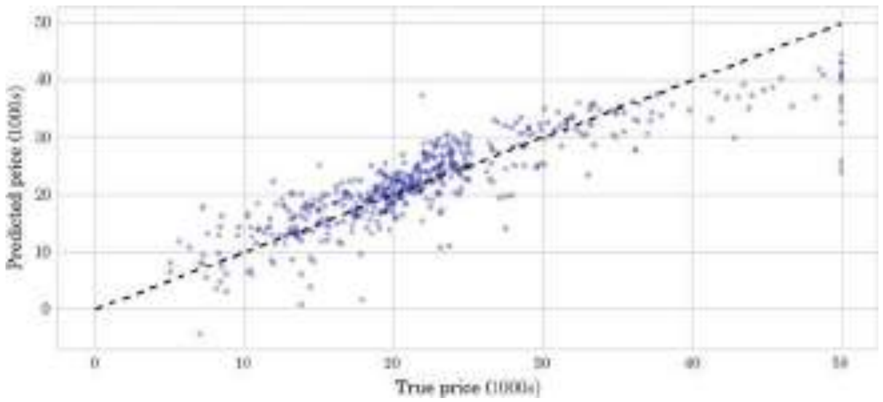


Fig. 6.9 Relation between true (x-axis) and predicted (y-axis) prices

In [18]:
```
clf = LinearRegression()
clf.fit(boston.data, boston.target)
predicted = clf.predict(boston.data)
plt.scatter(boston.target, predicted, alpha = 0.3)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.xlabel('True price ($1000s)')
plt.ylabel('Predicted price ($1000s)')
```

The output is shown in Fig. 6.9, where we can observe that the original prices are properly estimated by the predicted ones, except for the higher values, around
$50.000 (points in the top right corner).

Finally, it is worth noting that we can work with statistical evaluation of a linearregression with the *OLS* toolbox of the *Stats Model* toolbox.[4] This toolbox is usefulto study several statistics concerning the regression model. To know more about thetoolbox, go to the Documentation related to Stats Models.

Logistic Regression

Logistic regression is a type of model of probabilistic statistical classification. It is used as a binary model to predict a binary response, the outcome of a categorical dependent variable (i.e., a class label), based on one or more variables.

The form of the logistic function is:

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

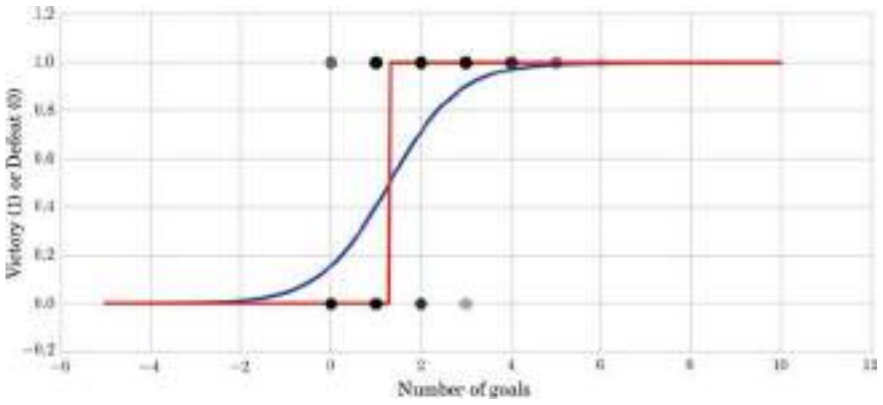


Fig. 6.12 Fitting of the logistic regression model (*blue*) and prediction of the logistic regression model (*red*) for the Spanish football league results

In [6]:
```
from sklearn import cluster

K = 3 # Assuming we have 3 clusters!
clf = cluster.KMeans(init = 'random', n_clusters = K)
clf.fit(X)
```

Out[6]:
```
KMeans(copy_x=True, init='random', max_iter=300,
n_clusters=3, n_init=10, n_jobs=1, precompute_distances=True,random_state=None,
tol=0.0001, verbose=0)
```

Each clustering algorithm in Scikit-learn is used as follows. First, an object from the clustering technique is instantiated. Then we can use the fitmethod to adjust the learning parameters. We also find the method predict that, given new data, returns the cluster they belong to. For the class, the labels over the training data canbe found in the labels_ attribute or alternatively they can be obtained using the predict method.

How many "mis-clusterings" do we have? In order to see this, we tessellate the space and color all grid points from the same cluster with the same color. Then, weoverlay the initial sample distributions (see Fig. 7.2). In the ideal case, we expect thatin each partitioned subspace the sample points are of the same color. However, as shown in Fig. 7.2, the resulting clustering, which is represented in the figure by the color subspace in gray, does not usually coincide exactly with the initial distribution, which is represented by the color of the data. For example, in the same figure, if most of the blue points belong to the same cluster, there are a few ones that belong to thespace occupied by the green data.

When computing the Rand index, we get:

In [7]:
```
print ('The Adjusted Rand index is: %.2f' %
        metrics.adjusted_rand_score(y.ravel(), clf.labels_))
```

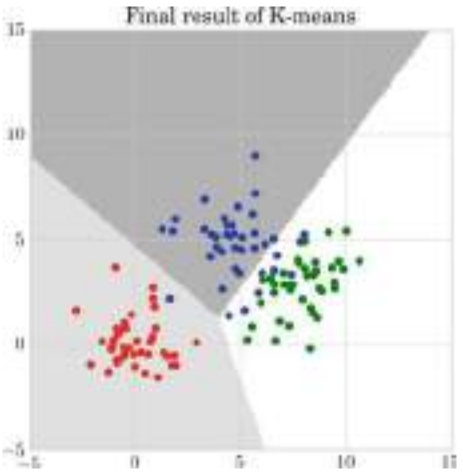


Fig. 7.2 Original samples (*dots*) generated by three distributions and the partitionof the space according to theK-means clustering

the similarity between the samples. Partitioning is computed by selecting a cut onthe tree at a certain level.

In general, there are two types of hierarchical clustering:

- *Top-down* divisive clustering applies the following algorithm:
  - Start with all the data in a single cluster.
  - Consider every possible way to divide the cluster into two.
  - Choose the best division.
  - Recursively, it operates on both sides until a stopping criterion is met. That can be something as follows: there are as much clusters as data; the predetermined number of clusters has been reached; the maximum distance between all possible partition divisions is smaller than a predetermined threshold; etc.

- *Bottom-up* agglomerative clustering applies the following algorithm:
  - Start with each data point in a separate cluster.
  - Repeatedly join the closest pair of clusters.
  - At each step, a stopping criterion is checked: there is only one cluster; a prede-termined number of clusters has been reached; the distance between the closestclusters is greater than a predetermined threshold; etc.

This process of merging forms a binary tree or hierarchy.

When merging two clusters, a question naturally arises: How to measure the similarity of two clusters? There are different ways to define this with different results for the agglomerative clustering. The linkage criterion determines the metricused for the cluster merging strategy:

- *Maximum* or *complete* linkage minimizes the maximum distance between observa-tions of pairs of clusters. Based on the similarity of the two least similar membersof the clusters, this clustering tends to give tight spherical clusters as a final result.
- *Average* linkage averages similarity between members, i.e., minimizes the averageof the distances between all observations of pairs of clusters.
- *Ward* linkage minimizes the sum of squared differences within all clusters. It is thus a variance-minimizing approach and in this sense is similar to the K-means objective function, but tackled with an agglomerative hierarchical approach.

Let us illustrate how the different linkages work with an example. Let us generatethree clusters as follows:

nectivity constraints are added between samples: it considers all the possible mergesat each step.

Adding Connectivity Constraints

Sometimes, we are interested in introducing a connectivity constraint into the clus- tering process so that merging of nonadjacent points is avoided. This can be achieved by constructing a connectivity matrix that defines which are the neighboring samples in the dataset. For instance, in the example in Fig. 7.4, we want to avoid the forma-tion of clusters of samples from the different circles. A sample code to compute agglomerative clustering with connectivity would be as follows:

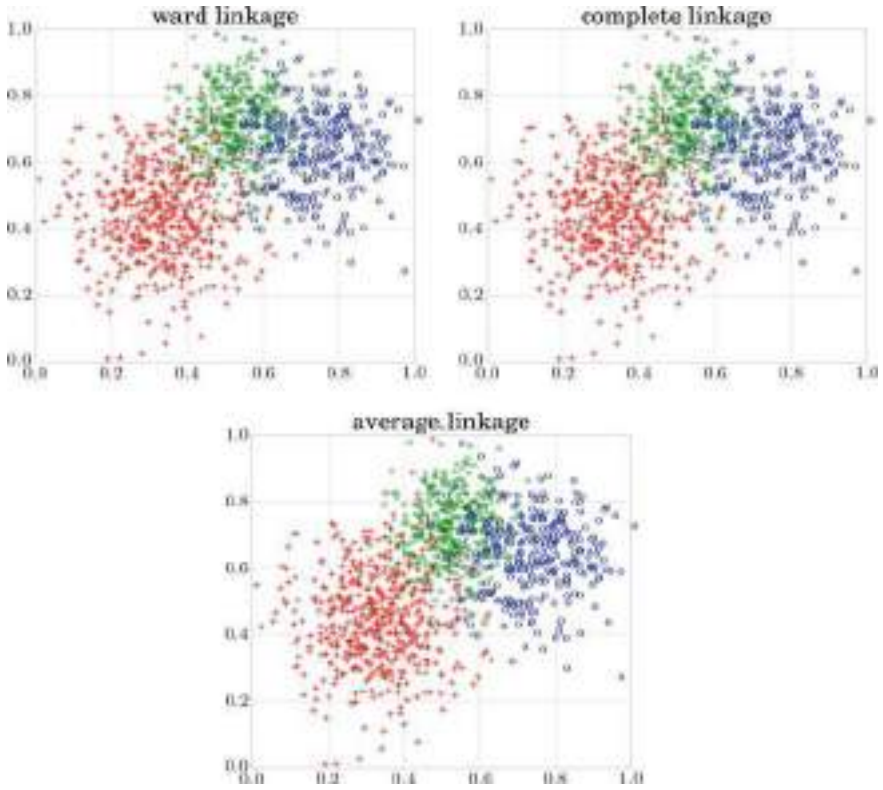


Fig. 7.3 Illustration of agglomerative clustering using different linkages: Ward, complete, and average. The symbol of each data point corresponds to the original class generated and the color corresponds to the cluster obtained

Fig. 7.4 Illustration of agglomerative clustering without (*top row*) and with (*bottom row*) a connectivity graph using the three linkages (from *left* to *right*): average, complete, and Ward. The *colors*correspond to the clusters obtained

In [10]:

```
connectivity  =  kneighbors_graph(X,  30)
model  = AgglomerativeClustering(linkage  = 'average',
    connectivity = connectivity,  n_clusters  = 8)
model.fit(X)
```

A connectivity constraint is useful to impose a certain local structure, but it also makes the algorithm faster, especially when the number of the samples is large. A connectivity constraint is imposed via a *connectivity matrix*: a sparse matrix that only has elements at the intersection of a row and a column with indexes of the dataset that should be connected. This matrix can be constructed from a priori information or can be learned from the data, for instance using kneighbors_graph to restrict merging to nearest neighbors or using image.grid_to_graphto limit mergingto neighboring pixels in an image, both from Scikit-learn. This phenomenon can beobserved in Fig. 7.4, where in the first row we see the results of the agglomerative clustering without using a connectivity graph. The clustering can join data from different circles (e.g., the black cluster). At the bottom, the three linkages use a connectivity graph and thus two of them avoid joining data points that belong to different circles (except the Ward linkage that attempts to form compact and isotropic clusters).

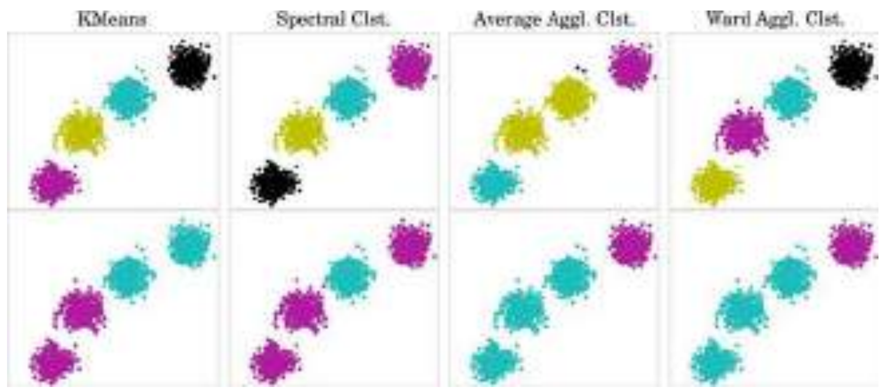


Fig. 7.5 Comparison of the different clustering techniques (from *left* to *right*): K-means, spectral clustering, and agglomerative clustering with average and Ward linkage on simple compact datasets. In the *first row*, the expected number of clusters is $k = 2$ and in the *second row*: $k = 4$

Comparison of Different Hard Partition Clustering Algorithms Let us compare the behavior of the different clustering algorithms discussed so far.For this purpose, we generate three different datasets' configurations:

(a) 4 spherical groups of data;
(b) a uniform data distribution; and
(c) a non-flat configuration of data composed of two moon-like groups of data.

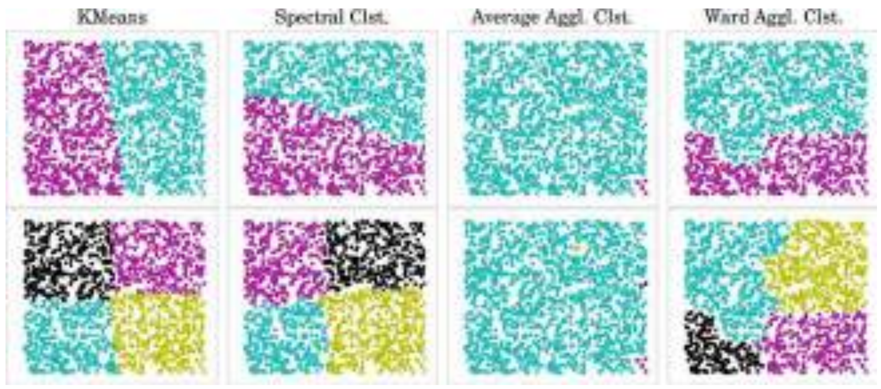


Fig. 7.6 Comparison of the different clustering techniques (from *left* to *right*): K-means, spectral clustering, and agglomerative clustering with average and Ward linkage on uniformly distributeddata. In the *first row*, the number of clusters assumed is $k = 2$ and in the *second row*: $k = 4$

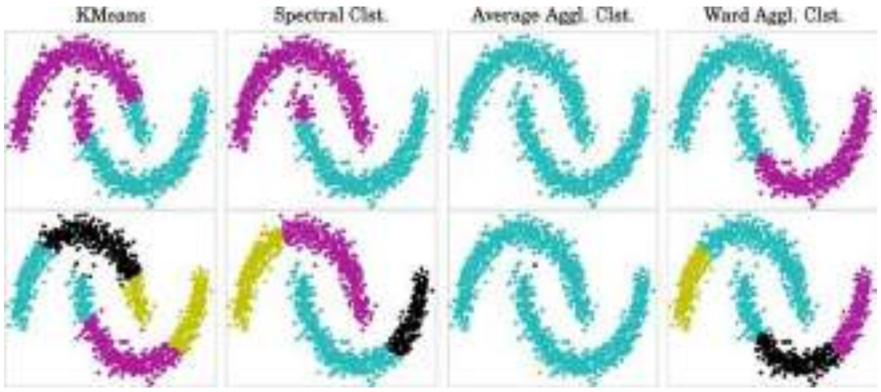


Fig. 7.7 Comparison of the different clustering techniques (from *left* to *right*): K-means, spectral clustering, and agglomerative clustering with average and Ward linkage on non-flat geometry datasets. In the *first row*, the expected number of clusters is $k = 2$ and in the *second row*: $k = 4$

second cluster of a small set of data. This behavior is observed in both cases: $k = 2$ and $k = 4$.

Regarding datasets with more complex geometry, like in the moon dataset (see Fig. 7.7), K-means and Ward linkage agglomerative clustering attempt to construct compact clusters and thus cannot separate the moons. Due to the connectivity con- straint, the spectral clustering and the average linkage agglomerative clustering sep-arated both moons in case of k 2, while in case of k 4, the average linkage agglomerative clustering clustered most of datasets correctly separating some of thenoisy data points as two separate single clusters. In the case of spectral clustering, looking for four clusters, the method splits each of the two moon datasets into two clusters.

UNIT-5

Network Analysis, Graphs, Social Networks, centrality, drawing centrality of Graphs, PageRank, Ego-Networks, community Detection

Network Analysis

Introduction

Network data are generated when we consider relationships between two or more entities in the data, like the highways connecting cities, friendships between peo- ple or their phone calls. In recent years, a huge number of network data are being generated and analyzed in different fields. For instance, in sociology there is inter-est in analyzing blog networks, which can be built based on their citations, to look for divisions in their structures between political orientations. Another example is infectious disease transmission networks, which are built in epidemiological studiesto find the best way to prevent infection of people in a territory, by isolating cer- tain areas. Other examples studied in the field of technology include interconnected computer networks or power grids, which are analyzed to optimize their functioning. We also find examples in academia, where we can build co-authorship networks and citation networks to analyze collaborations among Universities.

Structuring data as networks can facilitate the study of the data for different goals;for example, to discover the weaknesses of a structure. That could be the objective of a biologist studying a community of plants and trying to establish which of its properties promote quick transmission of a disease. A contrasting objective would be to find and exploit structures that work efficiently for the transmission of messages across the network. This may be the goal of an advertising agent trying to find the best strategy for spreading publicity.

How to analyze networks and extract the features we want to study are someof the issues we consider in this chapter. In particular, we introduce some basic concepts related with networks, such as connected components, centrality measures, ego-networks, and PageRank. We present some useful Python tools for the analysisof networks and discuss some of the visualization options. In order to motivate andillustrate the concepts, we perform social network analysis using real data. We present a practical case based on a public dataset which consists of a set of interconnected

Social Network Analysis

Social network analysis processes social data structured in graphs. It involves the extraction of several characteristics and graphics to describe the main properties of the network. Some general properties of networks, such as the shape of the network degree distribution (defined bellow) or the average path length, determine the type of network, such as a *small-world* network or a *scale-free* network. A small-world network is a type of graph in which most nodes are not neighbors of one another, butmost nodes can be reached from every other node in a small number of steps. This is the so-called *small-world phenomenon* which can be interpreted by the fact that strangers are linked by a short chain of acquaintances. In a small-world network, people usually form communities or small groups where everyone knows every- one else. Such communities can be seen as complete graphs. In addition, most the community members have a few relationships with people outside that community. However, some people are connected to a large number of communities. These maybe celebrities and such people are considered as the *hubs* that are responsible for the small-world phenomenon. Many small-world networks are also scale-free net- works. In a scale-free network the node degree distribution follows a power law (a relationship function between two quantities x and y defined as y x^n, where n isa constant). The name *scale-free* comes from the fact that power laws have the same functional form at all scales, i.e., their shape does not change on multiplication by ascale factor. Thus, by definition, a scale-free network has many nodes with a very few connections and a small number of nodes with many connections. This structure is typical of the World Wide Web and other social networks. In the following sections, we illustrate this and other graph properties that are useful in social network analysis.

In [1]:

Basics in NetworkX

NetworkX[1] is a Python toolbox for the creation, manipulation and study of the struc- ture, dynamics and functions of complex networks. After importing the toolbox, wecan create an undirected graph with 5 nodes by adding the edges, as is done in the following code. The output is the graph in Fig. 8.1.

```
import networkx as nx
G = nx.Graph()
G.add_edge('A', 'B');
G.add_edge('A', 'C');
G.add_edge('B', 'D');
G.add_edge('B', 'E');
G.add_edge('D', 'E');
nx.draw_networkx(G)
```

~~To create a directed gr~~aph we would use nx.DiGraph().

Out[4]: # connected components of Facebook network: 1

As it can be seen, there is only one connected component in the Facebook network. Thus, the Facebook network is a connected graph (see definition in Sect. 8.2). We can try to divide the graph into different connected components, which can be potentialcommunities (see Sect. 8.6). To do that, we can remove one node from the graph (this operation also involves removing the edges linking the node) and see if the number of connected components of the graph changes. In the following code, we prune the graph by removing node ‘0’ (arbitrarily selected) and compute the number of connected components of the pruned version of the graph:

In [5]:

```
fb_prun = nx.read_edgelist(
    "files/ch08/facebook_combined.txt")
fb_prun.remove_node('0')
print 'Remaining nodes:', fb_prun.number_of_nodes()
print 'New # connected components:',
       nx.number_connected_components(fb_prun)
```

Out[5]: Remaining nodes: 4038
New # connected components: 19

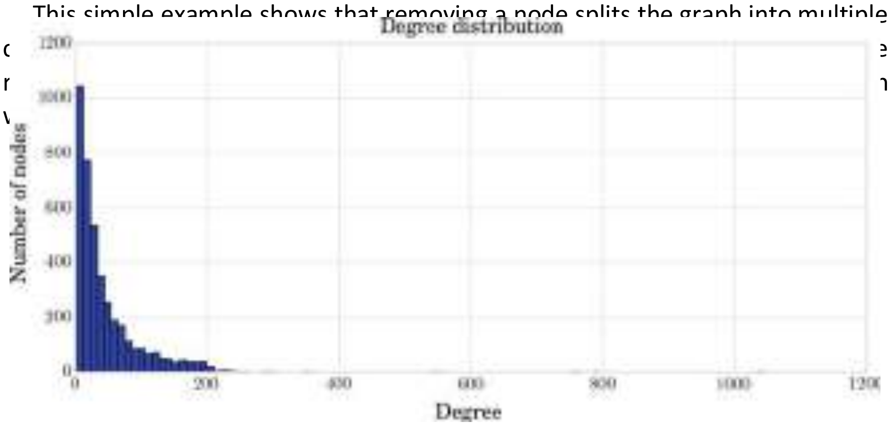
Now there are 19 connected components, but let us see how big the biggest is andhow small the smallest is:

In [6]:

```
fb_components = nx.connected_components(fb_prun)
print 'Sizes of the connected components',
       [len(c) for c in fb_components]
```

Out[6]: Sizes of the connected components [4015, 1, 3, 2, 2, 1, 1, 1,
1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1]

This simple example shows that removing a node splits the graph into multiple



do that with the matplotlib.loglog()function. Figure 8.4 shows the degree centrality histogram in linear and logarithmic scales as computed in the box

In[7]:

```
degree_cent_fb = nx.degree_centrality(fb)
print 'Facebook degree centrality: ',
      sorted(degree_cent_fb.items(),
             key = lambda x: x[1],
             reverse = True)[:10]
degree_hist = plt.hist(list(degree_cent_fb.values()), 100)
plt.loglog(degree_hist[1][1:],
           degree_hist[0], 'b', marker = 'o')
```

Out[7]:

```
Facebook degree centrality: [(u'107', 0.258791480931154),
(u'1684', 0.1961367013372957), (u'1912', 0.18697374938088163),
(u'3437', 0.13546310054482416), (u'0', 0.08593363051015354),
(u'2543', 0.07280832095096582), (u'2347', 0.07206537890044576),
(u'1888', 0.0629024269440317), (u'1800', 0.06067360079247152),
(u'1663', 0.058197127290737984)]
```

The previous plots show us that there is an interesting (large) set of nodes which corresponds to low degrees. The representation using a logarithmic scale (right-hand graphic in Fig. 8.4) is useful to distinguish the members of this set of nodes, whichare clearly visible as a straight line at low values for the x-axis (upper left-hand part of the logarithmic plot). We can conclude that most of the nodes in the graph have low degree centrality; only a few of them have high degree centrality. These latter nodes can be properly seen as the points in the bottom right-hand part of the logarithmic plot.

The next code computes the betweenness, closeness, and eigenvector centrality and prints the top 10 central nodes for each measure.

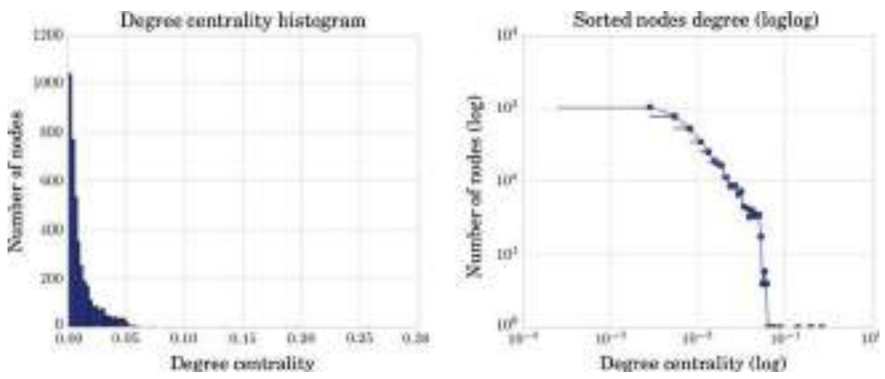


Fig. 8.4 Degree centrality histogram shown using a linear scale (*left*) and a log scale for both the x- and y-axis (*right*)

will work with a trimmed Facebook network instead of the original one. In fact, we can

pose the question: What happen if we only consider the graph nodes with more than the average degree of the network (21)? We can trim the graph using degree centrality values. To do this, in the next code, we define a function to trim the graph based on the degree centrality of the graph nodes. We set the threshold to

In[9]:
```
def trim_degree_centrality(graph, degree = 0.01):
    g = graph.copy()
    d = nx.degree_centrality(g)
    for n in g.nodes():
        if d[n] <= degree:
            g.remove_node(n)
    return g
thr = 21.0/(fb.order() - 1.0)

print 'Degree centrality threshold:', thr

fb_trimmed = trim_degree_centrality(fb, degree = thr)
print 'Remaining # nodes:', len(fb_trimmed)
```

Out[9]:
```
Degree centrality threshold: 0.00520059435364Remaining # nodes:
2226
```

The new graph is much smaller; we have removed almost half of the nodes (we have moved from 4,039 to 2,226 nodes).

The current flow betweenness centrality measure needs connected graphs, as does any betweenness centrality measure, so we should first extract a connected compo- nent from the trimmed Facebook network and then compute the

In[10]:
```
fb_subgraph = list(nx.connected_component_subgraphs(
    fb_trimed))
print '# subgraphs found:', size(fb_subgraph)
print '# nodes in the first subgraph:',
      len(fb_subgraph[0])
betweenness = nx.betweenness_centrality(fb_subgraph[0])
print 'Trimmed FB betweenness: ',
      sorted(betweenness.items(), key = lambda x: x[1],
             reverse = True)[:10]
current_flow = nx.current_flow_betweenness_centrality(
    fb_subgraph[0])
print 'Trimmed FB current flow betweenness:',
      sorted(current_flow.items(), key = lambda x: x[1],
      reverse = True)[:10]
```

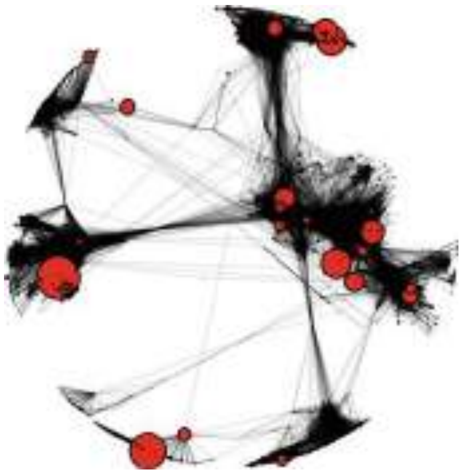


Fig. 8.7 The Facebook network drawn using theSpring layout and betweenness centrality todefine the node size

the opposite effect: nodes with high degree but relatively low betweenness. These nodes are those with redundant communication.

Changing the centrality measure to closeness and eigenvector, we obtain the graphs in Figs. 8.8 and 8.9, respectively. As can be seen, the central nodes arealso different for these measures. With this or other visualizations you will be able to discern different types of nodes. You can probably see nodes with high closeness centrality but low degree; these are essential nodes linked to a few important or active nodes. If the opposite occurs, if there are nodes with high degree centrality but lowcloseness, these can be interpreted as nodes embedded in a community that is far removed from the rest of the network.

In other examples of social networks, you could find nodes with high closeness centrality but low betweenness; these are nodes near many people, but since there may be multiple paths in the network, they are not the only ones to be near many people. Finally, it is usually difficult to find nodes with high betweenness but low closeness, since this would mean that the node in question monopolized the links from a small number of people to many others.

PageRank

PageRank is an algorithm related to the concept of eigenvector centrality in directed graphs. It is used to rate webpages objectively and effectively measure the attention devoted to them. PageRank was invented by Larry Page and Sergey Brin, and becamea Google trademark in 1998 [2].

Assigning the importance of a webpage is a subjective task, which depends on the interests and knowledge of the persons that browse the webpages. However, there are ways to objectively rank the relative importance of webpages.

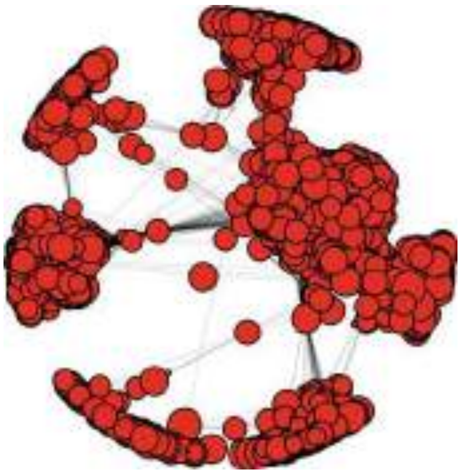


Fig. 8.8 The Facebook network drawn using the Spring layout and closeness centrality to define the nodesize

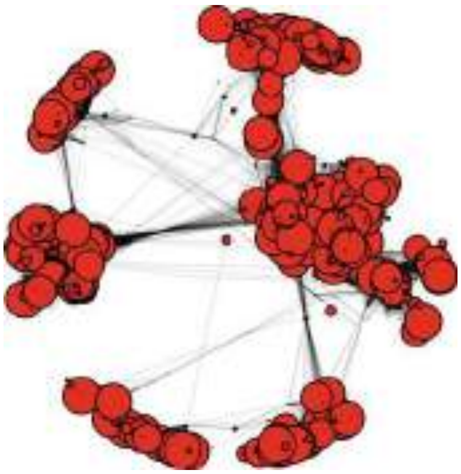


Fig. 8.9 The Facebook network drawn using theSpring layout and eigenvector centrality todefine the node size

We consider the directed graph formed by nodes corresponding to the webpagesand edges corresponding to the hyperlinks. Intuitively, a hyperlink to a page countsas a vote of support and a page has a high rank if the sum of the ranks of its incoming edges is high. This considers both cases when a page has many incoming links and when a page has a few highly ranked incoming links. Nowadays, a variant of the algorithm is used by Google. It does not only use information on the number of edges pointing into and out of a website, but uses many more variables.

We can describe the PageRank algorithm from a probabilistic point of view. The rank of page P_i is the probability that a surfer on the Internet who starts visiting a random page and follows links, visits the page P_i. With more details, we consider

```
Out[13]:# nodes of ego graph 107: 1046
        # nodes of ego graph 107 with radius up to 2: 2687
```

The ego-network size is 1,046 with a distance of 1, but when we expand the distance to 2, node ‘107’ is able to reach up to 2,687 nodes. That is quite a large ego-network, containing more than half of the total number of nodes.

Since the dataset also provides the previously labeled ego-networks, we can com-pute the actual size of the ego-network following the user labeling. We can access the ego-networks by simply importing os.path and reading the edge list corre-sponding, for instance, to node ‘107’, as in the following code:

In [14]:
```
import os.path
ego_id = 107
G_107 = nx.read_edgelist(
    os.path.join('files/ch08/facebook',
                 '{0}.edges'.format(ego_id)),
    nodetype = int)
print 'Nodes of the ego graph 107:', len(G_107)
```

```
Out[14]:Nodes of the ego graph 107: 1034
```

As can be seen, the size of the previously defined ego-network of node ‘107’ is slightly different from the ego-network automatically computed using NetworkX. This is due to the fact that the manual labeling is not necessarily referred to the subgraph of neighbors at a distance of 1.

We can now answer some other questions about the structure of the Facebook network and compare the 10 different ego-networks among them. First, we can compute which the most densely connected ego-network is from the total of 10. Todo that, in the code below, we compute the number of edges in every ego-network and select the network with the maximum number:

In [15]:

```
ego_ids = ( 0, 107, 348,
            414, 686, 698,
            1684, 1912, 3437, 3980)
ego_sizes = zeros((10, 1))
i = 0
# Fill the 'ego_sizes' vector with the size (# edges) of the
    10 ego-networks in egoids
for id in ego_ids :
    G = nx.read_edgelist(
        os.path.join('files/ch08/facebook',
                     '{0}.edges'.format(id)),
        nodetype = int)
    ego_sizes[i] = G.size()
    i = i + 1
[i_max,j] = (ego_sizes == ego_sizes.max()).nonzero()
ego_max = ego_ids[i_max]
print ' The most densely connected ego - network is \
      that of node:', ego_max

G = nx.read_edgelist(
    os.path.join('files/ch08/facebook',
                 '{0}.edges'.format(ego_max)),
    nodetype = int)
print 'Nodes: ', G.order()
print 'Edges: ', G.size()
print 'Average degree: ', G_k / G_n
```

Out[15]:

```
The most densely connected ego-network is that of node: 1912Nodes: 747
Edges: 30025
Average degree: 40
```

The most densely connected ego-network is that of node '1912', which has an average degree of 40. We can also compute which is the largest (in number of nodes) ego-network, changing the measure of sizes from G.size() by G.order(). In this case, we obtain that the largest ego-network is that of node '107', which has 1,034 nodes and an average degree of 25.

Next let us work out how much intersection exists between the ego-networks inthe Facebook network. To do this, in the code below, we add a field 'egonet' for every node and store an array with the ego-networks the node belongs to. Then, having thelength of these arrays, we compute the number of nodes that belong to 1, 2, 3, 4 andmore than 4 ego-networks:

In [16]:

```
# Add a field ' egonet ' to the nodes of the whole facebook
    network.
# Default value egonet = [], meaning that this node does not
     belong to any ego-netowrk
for i in fb.nodes() :
    fb.node[str(i)]['egonet'] = []

# Fill the 'egonet' field with one of the 10 ego values in
    ego_ids:
for id in ego_ids :
    G = nx.read_edgelist(
        os.path.join('files/ch08/facebook',
                     '{0}.edges'.format(id)),
        nodetype = int)
    print id
    for n in G.nodes() :
        if (fb.node[str(n)]['egonet'] == []) :
            fb.node[str(n)]['egonet'] = [id]
        else :
            fb.node[str(n)]['egonet'].append(id)

# Compute the intersections:
S = [len(x['egonet']) for x in fb.node.values()]


print   '#   nodes   into   0       ego-network:   ',      sum(equal(S,      0))
print   '#   nodes   into   1       ego-network:   ',      sum(equal(S,      1))
print   '#   nodes   into   2       ego-network:   ',      sum(equal(S,      2))
print   '#   nodes   into   3       ego-network:   ',      sum(equal(S,      3))
print   '#   nodes   into   4       ego-network:   ',      sum(equal(S,      4))
print '# nodes into more than 4 ego-network : ', \
     sum(greater(S, 4))
```

Out[16]:

```
# nodes into 0 ego-network: 80
# nodes into 1 ego-network: 3844
# nodes into 2 ego-network: 102
# nodes into 3 ego-network: 11
# nodes into 4 ego-network: 2
# nodes into more than 4 ego-network: 0
```

As can be seen, there is an intersection between the ego-networks in the Facebook network, since some of the nodes belong to more than 1 and up to 4 ego-networks simultaneously.

We can also try to visualize the different ego-networks. In the following code, we draw the ego-networks using different colors on the whole Facebook network and we obtain the graph in Fig. 8.12. As can be seen, the ego-networks clearly formgroups of nodes that can be seen as communities.

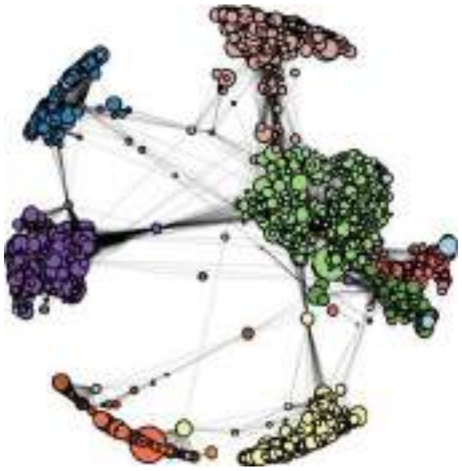


Fig. 8.12 The Facebook network drawn using the Spring layout and different colors to separate the ego-networks

In [17]:

```
# Add a field 'egocolor' to the nodes of the whole facebooknetwork.
# Default value egocolor r =0, meaning that this node
does not belong to any ego-netowrk for i in fb.nodes() :fb.node[str(i)]['egocolor']
      = 0

# Fill the 'egocolor' field with a different color numberfor each ego-network in
      ego_ids:
idColor = 1
for id in ego_ids :
      G = nx.read_edgelist(
            os.path.join('files/ch08/facebook',
                              '{0}.edges'.format(id)),nodetype = int)
      for n in G.nodes() :
            fb.node[str(n)]['egocolor'] = idColoridColor += 1

colors = [x['egocolor'] for x in fb.node.values()]

nsize = np.array([v for v in degree_cent_fb.values()])

nsize =500*(nsize - min(nsize))/(max(nsize)- min(nsize))nodes = nx.

draw_networkx_nodes(
      fb, pos = pos_fb,
      cmap = plt.get_cmap('Paired'),node_color = colors,
      node_size = nsize,
      with_labels = False)
edges=nx.draw_networkx_edges(fb, pos = pos_fb, alpha = .1)
```

However, the graph in Fig. 8.12 does not illustrate how much overlap is there between the ego-networks. To do that, we can visualize the intersection between ego-networks using a *Venn* or an *Euler* diagram. Both diagrams are useful in order to see how networks are related. Figure 8.13 shows the Venn diagram of the Facebook network. This powerful and complex graph cannot be easily built in Python tool-

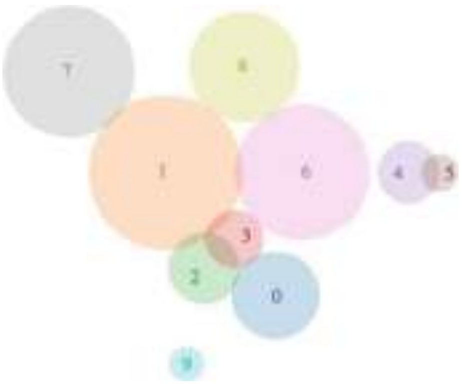


Fig. 8.13 Venn diagram. The area is weighted according to the number of friends in each ego-network and the intersection betweenego-networks is related to the number of common users

boxes like NetworkX or Matplotlib. In order to create it, we have used a JavaScript visualization library called D3.JS.[4]

Community Detection

A community in a network can be seen as a set of nodes of the network that is densely connected internally. The detection of communities in a network is a difficult task since the number and sizes of communities are usually unknown [3].

Several methods for community detection have been developed. Here, we applyone of the methods to automatically extract communities from the Facebook network. We import the Community toolbox[5] which implements the Louvain method for community detection. In the code below, we compute the best partition and plot theresulting communities in the whole Facebook network with different colors, as we did in box In [17]. The resulting graph is shown in Fig. 8.14.

In [18]:

```
import  community  partition  =  community.best_partition(fb)
     print  "#
communities  found:",  max(partition.values())  colors2  =
[partition.get(node)  for  node  in  fb.nodes()]  nsize  =  np.
     array([v
for  v  in  degree_cent_fb.values()])  nsize  =  500*(nsize   -
min(nsize))/(max(nsize)-  min(nsize))  nodes  =
nx . draw_networkx_nodes (
     fb,  pos  =  pos_fb,
     cmap  =  plt.get_cmap('Paired'),
     node_color  =  colors2,
     node_size  =  nsize,
     with_labels  =  False)
edges  =  nx.draw_networkx_edges(fb,  pos  =  pos_fb,  alpha  =  .1)
```

.

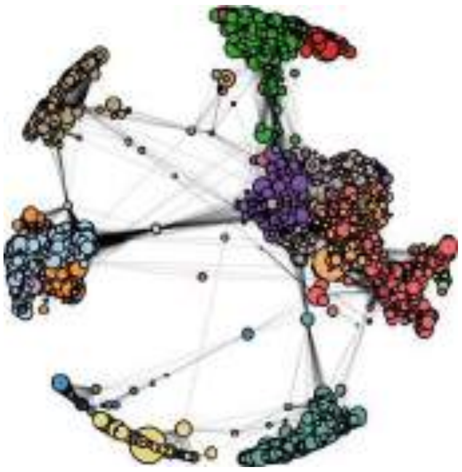


Fig. 8.14 The Facebook network drawn using the Spring layout and different colors to separate the communities found

```
Out[18]:# communities found: 15
```

As can be seen, the 15 communities found automatically are similar to the 10 ego-networks loaded from the dataset (Fig. 8.12). However, some of the 10 ego-networks are subdivided into several communities now. This discrepancy can be due to the fact that the ego-networks are manually annotated based on more properties of the nodes, whereas communities are extracted based only on the graph information.