Introduction to Python for Big Data Psychology

Authors: Lyu Sihua, Song Mengyao, Mo Liuling, Li Dongqi, Zhu Tingshao, Zhu Tingshao

Date: 2022-03-18T21:27:48+00:00

Abstract

This paper introduces big data psychology research methods in detail, using Jiujiu Articles Network as a case study. Using text data collected from user experiments, word frequency features are extracted to train machine learning models, which are then utilized to predict the life satisfaction corresponding to articles crawled from Jiujiu Articles Network, thereby helping beginners in big data research gain an intuitive understanding of the entire processing workflow. Through concrete examples, this paper introduces the use of Python and sentiment dictionaries for text word frequency calculation, employs the scikit-learn library to complete training, testing, and application of machine learning models, and combines this with accompanying source code to facilitate direct operation by readers. This paper preliminarily introduces a big data research methodology based on machine learning modeling of text word frequencies; the technical introduction is relatively fundamental, primarily emphasizing how to apply the techniques, with less coverage of technical principles.

Full Text

Preamble

Python for Big Data Psychology Research

Lyu Sihua[1],[2], Song Mengyao[1],[2], Mo Liuling[1],[2], Li Dongqi[1],[2], Zhu Tingshao[1],[2]*

[1] Department of Psychology, University of Chinese Academy of Sciences, Beijing 100049
[2] Institute of Psychology, Chinese Academy of Sciences, Beijing 100101

Abstract

This paper provides a detailed introduction to big data research methods in psychology, using the Ninety-Nine Articles website as a case study. We col-

lected textual data through user experiments, extracted word frequency features, trained machine learning models, and then applied these models to predict life satisfaction scores for articles crawled from the Ninety-Nine Articles website. This approach offers beginners in big data research an intuitive understanding of the entire processing pipeline. Through concrete examples, we demonstrate how to use Python and sentiment dictionaries for text-based word frequency calculation, utilize the scikit-learn library for machine learning model training, testing, and application, and provide accompanying source code for hands-on practice. This article introduces a foundational big data research methodology based on text word frequency and machine learning modeling, emphasizing practical application over technical principles.

Keywords: big data, word frequency, machine learning, Python

1. Introduction

With the continuous development of information technology, the massive amounts of data generated through internet usage have become one of our most valuable assets. Data now permeates every aspect of our lives and grows at an exponential rate, ushering us into the era of big data [1]. Big data has spread across all sectors of society, influencing our learning, work, daily life, and social development, while also providing researchers with unprecedented convenience [2-3].

As one of the primary carriers of massive data, the internet contains vast quantities of underutilized information waiting to be mined for valuable insights. Python, as a programming language closest to natural language, is relatively easy to learn and has become a powerful tool for effective data mining. As an object-oriented programming language, Python offers rich libraries and application programming interfaces (APIs) that meet the diverse needs of data mining and analysis on web platforms, enabling efficient searching, storage, and display of massive datasets [4]. Song Tian and Huang Tianyu argue that Python is more suitable than traditional teaching languages such as C++ and VB for non-computer science students, offering broader pedagogical applications [6]. Zheng Jiming notes that Python' s simplicity, extensive class libraries, and excellent scalability and portability make it ideal for cultivating computational thinking [7]. By October 2021, Python had surpassed C and Java to become the top-ranked language on the TIOBE index.

In the context of the big data era, Python is undoubtedly a powerful assistant for big data research. Considering the current state of Python proficiency among researchers, this paper uses real data to introduce a four-step process for conducting big data psychology research with Python: (1) basic Python introduction, (2) web scraping for data download, (3) Jieba tokenization and word frequency statistics, and (4) machine learning model training, testing, and application. Combined with the accompanying source code and data, this guide helps readers gradually understand the key technologies involved in big data psychol-

ogy research and gain practical experience by running the programs. Many of these processing procedures can also be directly applied to readers' own research projects.

2. Python Overview

Python is a simple, highly visual programming language that is easy to learn yet powerful. It features efficient high-level data structures and enables simple yet effective object-oriented programming. With its concise syntax and support for dynamic input, Python is an interpreted language that serves as an ideal development language across numerous platforms and domains.

Python is free, open-source, and highly portable, capable of running on Unix-derived systems, Win32 systems, handheld platforms (PDAs/mobile phones), and even gaming consoles (PSP). It boasts an extensive standard library covering regular expressions, documentation generation, unit testing, threading, databases, web browsers, and machine learning, among others. Additional high-quality libraries such as wxPython, Twisted, and various image libraries further extend its capabilities.

Python can be downloaded and installed from the official website: http://www.python.org. Most Unix-derived systems come with Python pre-installed; typing "python" in the command line will display version information. Installation is straightforward and similar to other software, with numerous open tutorials available online. Popular Python programming resources include:

- W3School Online Tutorial: https://www.w3school.com.cn/python/index.asp
- A Byte of Python: https://www.woodpecker.org.cn/abyteofpython_{cn}/chinese/
- Python3 Tutorial: https://www.runoob.com/python3/python3-tutorial.html
- Woodpecker Community: https://wiki.woodpecker.org.cn/moin/

3. Python Programming Basics

In Python programming, data types are fundamental concepts. Variables can store different types of data, and different types support different operations. Python has six standard data types: Number, String, List, Tuple, Set, and Dictionary. Immutable data types include Number, String, and Tuple, while mutable types comprise List, Set, and Dictionary.

Basic Data Types: Numbers

Table 1: Python Basic Data Types

Type	Description
int (integer)	Positive integers, zero, and negative integers
float (floating-point)	All numbers containing decimal points

Type	Description
bool (Boolean)	Only two values: True and False, representing truth values
complex (complex number)	Real numbers plus imaginary numbers; any number with an imaginary component (e.g., 3j)
String	Any characters enclosed in quotes, such as "123" or 'hello'
List	Accessible, modifiable, ordered; forward indexing: 0, 1, 2, 3···; reverse indexing: -1, -2, -3···
Tuple	Accessible but not modifiable, ordered; indexing same as List
Set	Unordered, unmodifiable, automatically deduplicated
Dictionary (dict)	Stores data as key-value pairs; values accessed and modified via keys

The `type()` function returns the data type of any object, while the `print()` function outputs results (see `prog\simp-1-hello-world.py`).

Python operators perform operations on variables and values, including arithmetic, assignment, comparison, logical, identity, membership, and bitwise operators (see `prog\simp-5-exp.py`). The `if` statement controls program execution, using the `if` keyword to specify a condition. When the condition evaluates to true, the subsequent indented statements execute. The `elif` keyword expresses "if the previous condition was not true, then try this condition," while the `else` keyword catches all cases not captured by previous conditions (see `prog\simp-6-ifelse.py`).

The `while` statement loops through program execution, repeating a code block while a condition remains true to handle repetitive tasks (see `prog\simp-7-while.py`). Within `while` loops, the `break` statement terminates the loop even when the condition is true, while `continue` skips the current iteration and proceeds to the next.

The `for` loop iterates over sequences (lists, tuples, dictionaries, sets, or strings), which can be indexed from 0 to less than the sequence length and can be sliced (see `prog\simp-9-for.py`). Within `for` loops, `break` terminates the loop before iterating through all items, `continue` stops the current iteration and proceeds to the next, and the `else` keyword specifies a code block to execute when the loop completes.

Functions are code blocks that run only when called, can receive data (parameters) in parentheses following the function name, and can return data as results. Functions must be defined before being called (see `prog\simp-4-function.py`). In Python, the `def` keyword defines functions, which are invoked by following the function name with parentheses.

4. Web Scraping for Data Download

Web crawlers are programs that automatically extract web page information according to specified rules by simulating human operations. This section introduces the workflow of web crawling technology and explains the Python-based web data download process through practical cases.

Traditional crawlers begin with one or several initial webpage URLs, obtain URLs from these pages, and continuously extract new URLs from current pages into a queue until certain stopping conditions are met. As shown in Figure 1, the basic web crawler workflow is: (1) select one or more seed URLs; (2) place seed URLs into the pending queue; (3) sequentially retrieve URLs from the queue, resolve DNS to obtain the server IP, download and save the webpage to a database, then move the URL to the crawled queue; (4) analyze crawled URLs to extract additional URLs and return them to the pending queue, continuing the cycle [5].

Using the Ninety-Nine Articles website as an example, we crawl article titles and content from http://www.99wenzhangwang.com/article/18491.html, saving them to local text files. This case uses Python 3.8, with complete code in `prog\www-9-99wz-sample.py`. Web data download involves four steps: data acquisition, parsing, extraction, and storage.

The crawler initiates requests to servers based on provided URLs and returns data. Data acquisition uses the `requests` library, which can download webpage source code, text, images, and even audio. Since `requests` is not a Python standard library, install it via terminal: `pip install requests` (use `pip3` on Mac). The `get()` method sends requests to servers, which return HTML document packets. Sample code:

```
import requests
from bs4 import BeautifulSoup

r = requests.get('http://www.99wenzhangwang.com/article/18491.html')
r.encoding = r.apparent_{encoding}
```

HTML (Hypertext Markup Language) is the standard markup language for creating webpages. Markup languages combine text with related information such as size, height, color, and position. As shown in Figure 2, HTML has a clear hierarchical structure like Python, with three fundamental elements: the `html` element (`<html></html>`), `head` element (`<head></head>`), and `body` element (`<body></body>`). The head typically sets webpage encoding, logos, titles, and external file references, while the body defines all content within the webpage window and is our primary focus. Tags enclosed in angle brackets (`<>` and `</>`) mark text information, with attributes like `align` and `style` defining element appearance. We locate desired data using tag names and attribute values.

Webpage parsing methods include regular expressions, BeautifulSoup, and lxml, each with distinct characteristics. This case uses BeautifulSoup, which requires

separate installation. BeautifulSoup parsing requires two parameters: the text to parse (must be a string) and the parser identifier (we use Python' s built-in `html.parser`):

```
soup = BeautifulSoup(r.text, 'html.parser')
```

After parsing the webpage into a BeautifulSoup object, we locate desired data using tags and attributes. The SelectorGadget plugin for Chrome helps identify tags (see `software\chrome\install.pdf` for installation). After identifying tags, use `find()` and `find_{all}()` methods to extract data. `find()` returns the first matching item, while `find_{all}()` returns all matches as a list requiring iteration:

```
title = soup.find('h2')
contents = soup.find(class_='hl_{body}').find_{all}('p')
content = ""
for para in contents:
    if len(para) > 0:
        content += para.text
```

After extraction, data must be stored. Storage options include plain text files (txt, csv, Excel) or databases (MySQL). This case writes crawled data to text files through three steps: open, write, and close:

```
file = open('99wenzhang.txt', 'w', encoding='utf8')
file.write(title.text)
file.write('\n')
file.write(content)
file.close()
```

The saved result is shown in Figure 4. Similar methods can build site-specific crawlers: seven Python programs (`prog\www-1~7-*.py`) progressively introduce crawlers for the Institute of Psychology, Chinese Academy of Sciences announcement board, while `prog\www-8-99wzsave.py` downloads articles from specific Ninety-Nine Articles categories.

5. Jieba Tokenization and Word Frequency Statistics

A critical challenge in applying machine learning to textual data is obtaining suitable input features. Raw text is typically represented as character strings, but machine learning models generally cannot process character data directly, and variable text lengths conflict with the fixed-dimensional input requirements of most models. Therefore, raw text must be processed to obtain numerical features. The main processing techniques and workflow are shown in Figure 5.

The conversion process begins with text preprocessing. The first step is text cleaning, which removes or replaces useless symbols like spaces that would interfere with tokenization and word frequency statistics. The second step is

tokenization—splitting continuous character sequences into semantically independent word sequences according to specific rules. While English uses spaces as natural delimiters, Chinese lacks formal word boundaries, making Chinese tokenization challenging. However, multiple technologies and tools now address this, including Jieba for Python (see `prog\jieba-1-seg.py`).

The final preprocessing step is stop word removal (see `prog\jieba-2-stopwords.py`). Stop words are typically excluded from frequency statistics, usually comprising prepositions, auxiliary words, or adverbs in Chinese. Stop word lists vary depending on the task objectives.

After preprocessing, text must be converted to numerical representation through text modeling, which has three main approaches: vector space models, neural network embedding models, and topic models. Vector space models represent text as points in vector space, focusing on feature selection and weight calculation. Neural network embedding models use deep learning to map discrete text variables to continuous numerical vectors. Topic models are statistical models that cluster latent semantic structures in text through unsupervised learning. Word frequency statistics can be considered a simplified vector space model that uses a predefined dictionary to analyze word frequencies, generating frequency vectors for direct use in supervised learning or statistical analysis.

This section uses articles crawled from Ninety-Nine Articles as raw data, applying the Dalian University of Technology Emotion Dictionary and Weibo Basic Emotion Lexicon for tokenization and frequency statistics. The Dalian dictionary contains 21 word categories including happiness, peace, and respect, while the Weibo lexicon includes five categories: happiness, sadness, anger, fear, and disgust. First, we define identifiers for different word categories (see `prog\swls-2-jieba-affect-export.py`):

```
affect_{col}_{list} = ['PA', 'PE', 'PD', 'PH', 'PG', 'PB', 'PK', 'NA', 'NB', 'NJ', 'NH', 'PF
```

Before Jieba tokenization and frequency statistics, we must load the dictionary files and stop word lists:

```
def load_{affect}_{dict}(filepath):
    m_{affectdict} = []
    for m_{col} in affect_{col}_{list}:
        m_{col} = []
        m_{affectdict}.append(m_{col})
    for m_{line} in open(filepath, 'r', encoding='utf-8').readlines():
        m_{line} = m_{line}.strip()
        kwd = m_{line}.split('\t')[0].strip()
        col = m_{line}.split('\t')[1].strip()
        m_{affectdict}[affect_{col}_{list}.index(col)].append(kwd)
    return m_{affectdict}

affect_{dict}_{file} = '../data/dict-affect.txt'
```

```
affect_{dict} = load_{affect}_{dict}(affect_{dict}_{file})
jieba.load_{userdict}('../data/jieba_{load}_{affect}_{dict}.txt')
```

Stop word loading:

```
def load_{stopwords}(filepath):
    m_{stopwords} = [line.strip() for line in open(filepath, 'r', encoding='utf-8').readline
    return m_{stopwords}

stop_{word}_{file} = '../data/stop_{words}_{cn}.txt'
stopwords = load_{stopwords}(stop_{word}_{file})
```

After loading Chinese stop words and the emotion dictionaries, Jieba performs tokenization and stop word removal, then counts frequencies for different word categories, generating emotion dictionary frequency vectors for each text:

```
def read_{swls}_{file}(fname):
    fr_{swls} = open(fname, 'r', encoding='UTF-8-sig')
    x_{swls}_{strs} = fr_{swls}.readlines()
    fr_{score} = int(x_{swls}_{strs}[0].strip('\n'))
    fr_{gender} = x_{swls}_{strs}[1].strip('\n')
    fr_{desc} = ''
    x_{swls}_{strs}.pop(0)
    x_{swls}_{strs}.pop(0)
    for swls_{str} in x_{swls}_{strs}:
        fr_{desc} += swls_{str}.strip().strip('\n') + ' '
    fr_{swls}.close()
    return fr_{score}, fr_{gender}, fr_{desc}

for fdata in swls_{files}:
    x_{score}, x_{gender}, x_{desc} = read_{swls}_{file}(swls_{dir} + fdata)
    str_{export} = str(x_{score}) + ','
    if (x_{gender} == 'M') or (x_{gender} == 'm') or (x_{gender} == '男'):
        str_{export} += '0'
    else:
        str_{export} += '1'
    for idx, g_{col} in enumerate(affect_{col}_{list}):
        r_{affect} = cntkws_{jieba}_{seg}_{wrds}(x_{desc}, affect_{dict}[idx])
        str_{export} += ',' + str(r_{affect})
    dstfp.write(str_{export} + '\n')
    dstfp.flush()
dstfp.close()
```

Each text is thus converted into a numerical vector of consistent dimensions, where each dimension carries linguistic meaning, enabling subsequent supervised learning or statistical analysis.

6. Machine Learning Model Training, Testing, and Application

Machine learning methods can uncover hidden patterns in large datasets and predict future inputs based on these patterns. Machine learning models divide into supervised and unsupervised learning (see `prog\ml-1-kmeans.py`). Common supervised models include classification (see `prog\ml-5-svm.py`) and regression (see `prog\ml-4-svr.py`). Using `prog\ml-5-svm.py` as an example, we introduce the main processes of model training and application:

```
from sklearn import svm

training_x = [[0, 0], [1, 1], [3, 2], [2, 2]]
training_y = [0, 1, 2, 1]
clf = svm.SVC(gamma='scale')
clf.fit(training_x, training_y)

testing_x = [[2, 2], [1, 2]]
result = clf.predict(testing_x)
print("Predict: ", result)
```

Generally, predicting new samples requires first training a model. Training data comprises independent variables (features) and dependent variables (labels) for each sample. Features may be multiple, forming a feature vector, while dependent variables are called annotations or outputs. Model training teaches the computer the mapping between independent and dependent variables, enabling prediction of new samples' outputs from their inputs alone.

The above illustrates basic training and prediction, but supervised learning practice typically involves four stages: training, testing, saving, and importing/application. Training and testing involve debugging different algorithms on data to find optimal performance, including feature extraction, selection, model training, and performance evaluation. This process is iterative—test results can inform adjustments to any previous step. For performance testing, data is randomly split into non-overlapping training and test sets. Models train on the training set and are evaluated on the test set. Based on results, appropriate algorithms and parameters are selected, final models are trained on the full dataset and saved for future import and application.

Using life satisfaction prediction as an example, we need participants' self-descriptive texts and life satisfaction scores (collected via questionnaire; see `data\生活满意度练习.pdf`). We treat self-descriptions as input variables and questionnaire scores as dependent variables, aiming to build a model that predicts life satisfaction from text. Through word frequency statistics, we extract textual features as independent variables and life satisfaction scores as dependent variables. `prog\swls-5-train-save.py` uses emotion dictionaries for feature extraction (see Section 5 for details). The code assigns individual word frequency features to `x_{kws}` and life satisfaction scores to `y_{score}`:

```
x_{kws} = []
y_{score} = []
dirs = '../data/swls/'
subdir = os.listdir(dirs)
for f in subdir:
    x_{score}, x_{gender}, x_{desc} = read_{swls}_{file}(dirs + f)
    item = feature_{extraction}(x_{desc})
    x_{kws}.append(item)
    y_{score}.append((x_{score} - 5) / 30)
```

Python' s scikit-learn library is an open-source machine learning module (https://scikit-learn.org.cn/) offering various classification, regression, and clustering algorithms. It enables model training through simple function calls, reducing mathematical and computational requirements. Model training involves two steps: specifying the model and fitting it to training data. Using LassoCV as an example:

```
clf_{lasso} = LassoCV()
clf_{lasso}.fit(x_{kws}, y_{score})
```

This basic training process must be extended because different algorithms and hyperparameters produce varying performance. Test sets, containing both input data and true values, evaluate model performance. The correlation coefficient between predictions and true values indicates performance—higher correlations are better. In the life satisfaction project, `prog\swls-3-train-test-score.py` implements model testing. Using Lasso regression as an example, the program trains on the training set, predicts life satisfaction from test set word frequencies, and calculates the correlation coefficient:

```
clf_{lasso} = LassoCV()
clf_{lasso}.fit(training_x, training_y)
result = clf_{lasso}.predict(testing_x)
ab = np.array([testing_y, result])
print('Lasso: ', np.corrcoef(ab))
```

By comparing different hyperparameter settings on the test set, we select the best-performing model, train the final model on all data, and export it for future use. `prog\swls-5-train-save.py` saves the trained model:

```
mod_{file} = '../data/swls.mod'
joblib.dump(clf_{lasso}, mod_{file})
print('SWLS model saved!')
```

After model creation and export, we can predict life satisfaction for new texts. Using Ninety-Nine Articles texts as examples, we extract features and make predictions. After obtaining word frequency features, we load the saved model and predict satisfaction scores, implemented in `prog\swls-6-99wz-apply.py`:

```
testdirs = '../data/99wz/'
testsubdir = os.listdir(testdirs)
```

```
apply_{kws} = []
wz_{list} = []
for f in testsubdir:
    buf = open(testdirs + f, 'r', encoding='utf-8').read()
    item = feature_{extraction}(buf)
    apply_{kws}.append(item)
    wz_{list}.append(f)

mod_{file} = '../data/swls.mod'
clf = joblib.load(mod_{file})
result = clf.predict(apply_{kws})
print(result)
```

7. Conclusion

This paper introduced the main processes of conducting big data psychology research using Python, using life satisfaction prediction from Ninety-Nine Articles as a case study to demonstrate the complete workflow of text-based word frequency machine learning modeling. Building on these techniques, researchers can investigate topics such as internet users' sleep patterns [2] and coming-out decisions among sexual minorities [3]. Researchers can use web scraping for targeted data collection, combine Jieba tokenization with existing dictionaries for text feature extraction, and employ supervised machine learning algorithms through Python to build and save models for automatic prediction of psychological indicators, enabling broader psychological research applications.

References

[1] Rupa Mahanti. Data Governance and Data Management[M]. Strathfield: Springer, p1-3.
[2] Song Mengyao, Jin Yuanyuan, Bai Bingyu, Xu Ying, Ma Yan, Ding Xiaoqing, Zhu Tingshao, Zhao Nan. The Relationship Between Staying Up Late, Urban Development Level, and Life Satisfaction—A Study Based on Weibo Big Data[DB/OL]. (2021-12-29) [2022-03-05]. http://www.chinaxiv.org/abs/202112.00163
[3] Wang Xinshu, Zhao Menghan, Pan Chao, Zhao Nan, Zhu Tingshao. The Impact of Coming Out on Sexual Minorities' Psychology—A Study Based on Weibo and Zhihu Data[DB/OL]. (2021-12-18) [2022-03-05]. http://www.chinaxiv.org/abs/202112.00120
[4] Li Yuxiang, Wang Mengyu, Tu Yuxi. Research on Web Crawler Technology Based on Python[J]. Information Technology and Informatization, 2019, (12): 143-145
[5] Song Tian, Huang Tianyu, Li Xin. Python Language: An Ideal Choice for Programming Course Teaching Reform[J]. China University Teaching, 2016, (2): 42-47.
[6] Zheng Jiming. Application of Computational Thinking in Python Program-

ming Courses[J]. University Education, 2016, (8): 127-129.
[7] Zhang Yan, Wu Yuquan. Design of Network Data Crawler Program Based on Python[J]. Computer Programming Skills & Maintenance, 2020(4): 26-27. DOI:10.3969/j.issn.1006-4052.2020.04.010.
[8] Tianguiyuyu. K-Fold Cross Validation[DB/OL]. (2018-06-14) [2022-03-05]. https://blog.csdn.net/tianguiyuyu/article/details/80697223

Note: Figure translations are in progress. See original paper for figures.