



FLASK WITH EMBEDDED MACHINE LEARNING I : SERIALIZING WITH PICKLE AND DB SETUP



(<http://www.addthis.com/bookmark.php?v=250&username=khhong7>)



K Hong
google.com/+KHongSanF
Francisc...

 **Follow**

4,892 followers

Ph.D. / Golden Gate Ave, San Francisco / Seoul National Univ / Carnegie Mellon / UC Berkeley / DevOps / Deep Learning / Visualization

Sponsor Open Source development activities and free contents for everyone.



Thank you.

- K Hong (http://bogotobogo.com/about_us.php)

bogotobogo.com site search:

Introduction

Machine learning models in web applications include spam detection in submission forms, shopping portals, search engines, recommendation systems for media, and so on.

Throughout the series of articles we'll see how to embed a machine learning model into a web application that not only makes classification but also learns from data in real-time.

Basically, via this series, we can extend our knowledge of machine learning theory. We are going to see how to serialize a model after training and how to load it for later use cases. We also create a SQLite database and create a Flask web application that our movie classifier available to the user.

Serializing fitted scikit-learn estimators

Because training a machine learning model requires expensive computational resources, we don't want our model to learn the model from the training data all over again.

Fortunately, Python's pickle (<https://docs.python.org/3/library/pickle.html#pickle.dump>) allows us to serialize and de-serialize object so that we can save our classifier in its current state and reload it if we want to classify new samples without needing to learn the model from the training data again.

In the following code, we create a **movieclassifier** directory where we will later store the files and data for our web application. Within this **movieclassifier** directory, we create **pkl_objects** subdirectory to save the serialized Python objects to our local drive.

```
import pickle
import os
dest = os.path.join('movieclassifier', 'pkl_objects')
if not os.path.exists(dest):
    os.makedirs(dest)

pickle.dump(stop, open(os.path.join(dest, 'stopwords.pkl'),'wb'), protocol=None)

pickle.dump(clf, open(os.path.join(dest, 'classifier.pkl'), 'wb'), protocol=None)
```

Using pickle's **dump()** method, we then serialized the trained logistic regression model as well as the **stop word** set from the NLTK library so that we can avoid installing the NLTK vocabulary on our server.

Here is the API definition of the **dump()** method:

```
pickle.dump(obj, file, protocol=None, *, fix_imports=True)
```

The **dump()** method writes a pickled representation of **obj** to the open file object **file**.

In the code, we used the "wb" argument within the open function: ('b') - binary mode for pickle.

movieclassifier/vectorizer.py

Since **HashingVectorizer** does not need to be fitted, we don't need to pickle it.

Instead, we can create a new Python script file, from which we can import the **vectorizer** into our current Python session. Here is the code (**movieclassifier/vectorizer.py**):

```
# movieclassifier/vectorizer.py

from sklearn.feature_extraction.text import HashingVectorizer
import re
import os
import pickle

cur_dir = os.path.dirname(os.path.abspath('__file__'))
stop = pickle.load(open(os.path.join(cur_dir, 'pkl_objects','stopwords.pkl'), 'rb'))

def tokenizer(text):
    text = re.sub('<[^\>]*>', '', text)
    emoticons = re.findall('(?:[;|=](?:-)?(?:\)|\{|\D|P))',
        text.lower())
    text = re.sub('[\W]+', ' ', text.lower()) \
        + ' '.join(emoticons.replace('-', ' '))
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized

vect = HashingVectorizer(decode_error='ignore', n_features=2**21,
    preprocessor=None, tokenizer=tokenizer)
```

Now we have pickled the Python objects, and let's deserialize and test if it is really working.

Flask

Deploying Flask Hello World App with Apache WSGI on Ubuntu 14
(/python/Flask/Python_Flask_Hello_World)

Flask Micro blog "Admin App" with Postgresql
(/python/Flask/Python_Flask_Blog)

Flask "Blog App" with MongoDB - Part 1 (Local via Flask server)
(/python/Flask/Python_Flask_Blog)

Flask "Blog App" with MongoDB on Ubuntu 14 - Part 2 (Local Apache WSGI)
(/python/Flask/Python_Flask_Blog)

Flask "Blog App" with MongoDB on CentOS 7 - Part 3 (Production Apache WSGI)
(/python/Flask/Python_Flask_Blog)

Flask word count app 1 with PostgreSQL and Flask-SQLAlchemy
(/python/Flask/Python_Flask_App)

Flask word count app 2 via BeautifulSoup, and Natural Language Toolkit (NLTK) with Unicorn/PM2/Apache
(/python/Flask/Python_Flask_App)

Flask word count app 3 with Redis task queue
(/python/Flask/Python_Flask_App)

Flask word count app 4 with AngularJS polling the back-end
(/python/Flask/Python_Flask_App/end.php)

Flask word count app 5 with AngularJS front-end updates and submit error handling
(/python/Flask/Python_Flask_App)

Deserializing

Now we may want to test if we can deserialize the objects without any error.

Let's restart Jupyter Notebook kernel from a terminal, navigate to the **movieclassifier** directory, start a new session and execute the following code to verify that we can **import the vectorizer** and **unpickle the classifier**:

```
import pickle
import re
import os
from vectorizer import vect
clf = pickle.load(open(os.path.join('pkl_objects', 'classifier.pkl'), 'rb'))

clf

SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
eta0=0.0, fit_intercept=True, l1_ratio=0.15,
learning_rate='optimal', loss='log', n_iter=1, n_jobs=1,
penalty='l2', power_t=0.5, random_state=1, shuffle=True, verbose=0,
warm_start=False)
```

Once we have successfully loaded the vectorizer and unpickled the classifier, we can use these objects to pre-process document samples and make predictions about their sentiment.

Since our classifier returns the class labels as integers, we defined a simple Python dictionary to map those integers to their sentiment:

```
import numpy as np
label = {0:'negative', 1:'positive'}
```

We then used the **HashingVectorizer** to transform the simple example document into a word vector XX :

```
example = ['I love this movie a lot']
X = vect.transform(example)
X

<1x2097152 sparse matrix of type '<type 'numpy.float64'>'
with 3 stored elements in Compressed Sparse Row format>
```

Finally, we used the predict method of the logistic regression classifier to predict the class label as well as the **predict_proba()** method to return the corresponding probability of our prediction.

```
clf.predict(X)
array([1])

clf.predict(X)[0]
1

clf.predict_proba(X)
array([[ 0.1817794,  0.8182206]])

label[clf.predict(X)[0]], np.max(clf.predict_proba(X))*100
('positive', 81.822059513044678)
```

Note that the **predict_proba()** method returns an array with a probability value for each unique class label.

Since the class label with the largest probability corresponds to the class label that is returned by the **predict()**, we used the **np.max()** to return the probability of the predicted class.

Here are additional sentiment analysis when we changed the example with more negative comments input to the classification prediction:

```
import numpy as np
label = {0:'negative', 1:'positive'}
example = ['I hate this movie']
X = vect.transform(example)
clf.predict_proba(X)

array([[ 0.67270865,  0.32729135]])

label[clf.predict(X)[0]], np.max(clf.predict_proba(X))*100

('negative', 67.270864651747914)

import numpy as np
label = {0:'negative', 1:'positive'}
example = ['The movie was really terrible and awful']
X = vect.transform(example)

clf.predict_proba(X)

array([[ 0.99336116,  0.00663884]])

label[clf.predict(X)[0]], np.max(clf.predict_proba(X))*100

('negative', 99.336116282043989)
```

Favorable review:

```
import numpy as np
label = {0:'negative', 1:'positive'}
example = ['Amazing, this movie is the best ever']
X = vect.transform(example)

clf.predict_proba(X)

array([[ 0.04785583,  0.95214417]])

label[clf.predict(X)[0]], np.max(clf.predict_proba(X))*100

('positive', 95.21441701567484)
```

SQLite setup

Now we want to set up a SQLite database to collect app user's feedback that we can use to update our classification model.

SQLite is a self-contained SQL database engine, and it is ideal for a simple web applications.

There is already an API in the Python standard library, **sqlite3**, which allows us to work with SQLite databases.

With the following code, we create a connection (**conn**) to an SQLite database file by calling sqlite3's **connect()** method, which creates the new database file **reviews.sqlite** in the **movieclassifier** directory:

7 - Like button
(/python/Flask/Python_Flask_Blo

Flask blog app with Dashboard
8 - Deploy
(/python/Flask/Python_Flask_Blo

Flask blog app with Dashboard
- Appendix (tables and mysql
stored procedures/functions
(/python/Flask/Python_Flask_Blo

Sponsor Open Source development activities and free contents for everyone.



- K Hong (http://bogotobogo.com/about_us.php)

Python
tutorial

```
import sqlite3
import os

conn = sqlite3.connect('reviews.sqlite')
c = conn.cursor()

c.execute('CREATE TABLE review_db\'
\' (review TEXT, sentiment INTEGER, date TEXT)')

example1 = 'I love this movie'
c.execute("INSERT INTO review_db"\'
\' (review, sentiment, date) VALUES"\'
\' (?, ?, DATETIME('now'))", (example1, 1))

example2 = 'I disliked this movie'
c.execute("INSERT INTO review_db"\'
\' (review, sentiment, date) VALUES"\'
\' (?, ?, DATETIME('now'))", (example2, 0))

conn.commit()
conn.close()
```

Next, we created a cursor via the **cursor()** method, which allows us to traverse over the database records.

By calling **execute()**, we created a new database table, **review_db**. We used this to store and access database entries. We also created three columns in this database table: **review**, **sentiment**, and **date**.

We're using these to store example movie reviews and respective class labels (sentiments).

Using the **DATETIME('now')** command, we also added **date** and **timestamps** to our entries.

In addition to the timestamps, we used the question mark(?) to pass the movie review texts and the corresponding class labels (1 and 0) as positional arguments to the **execute()** method as members of a tuple.

Finally, we called the **commit()** to save the changes we made to the database and closed the connection via the **close()** method.

To check if the entries have been stored in the database table correctly, we need to reopen the connection to the database and use the **SELECT** command to fetch all rows in the database table that have been committed between the beginning of the Nov.1, 2016 and today:

```
# Reconenct to check if the reviews have been successfully stored

conn = sqlite3.connect('reviews.sqlite')
c = conn.cursor()
c.execute("SELECT * FROM review_db WHERE date"\'
\' BETWEEN \'2016-11-01 00:00:00\' AND DATETIME('now')")
results = c.fetchall()
conn.close()
results

[(u'I love this movie', 1, u'2016-11-03 15:55:12'),
 (u'I disliked this movie', 0, u'2016-11-03 15:55:12')]
```

SQLite manager on Firefox

We may want to use the free Firefox browser plugin SQLite Manager (<https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/>):

(/python/pytut.php)

Introduction
(/python/python_introduction.php)

Running Python Programs (os, sys, import)
(/python/python_running.php)

Modules and IDLE (Import, Reload, exec)
(/python/python_modules_idle.p

Object Types - Numbers, Strings, and None
(/python/python_numbers_string

Strings - Escape Sequence, Raw String, and Slicing
(/python/python_strings.php)

Strings - Methods
(/python/python_strings_method

Formatting Strings - expressions and method calls
(/python/python_string_format

Files and os.path
(/python/python_files.php)

Traversing directories recursively
(/python/python_traversing_dire

Subprocess Module
(/python/python_subprocess_mo

Regular Expressions with Python
(/python/python_regularExpress

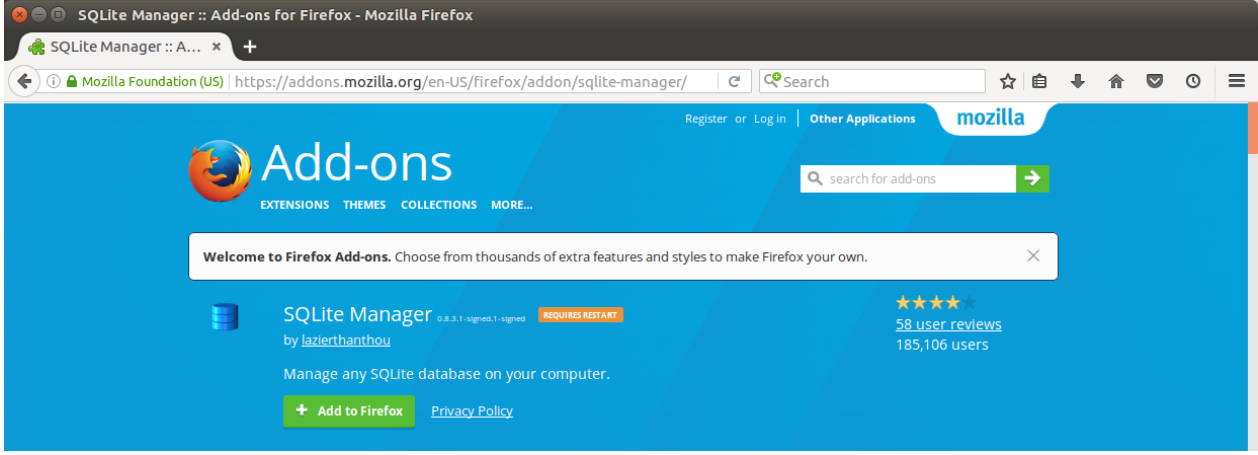
Object Types - Lists
(/python/python_lists.php)

Object Types - Dictionaries and Tuples
(/python/python_dictionaries_tu

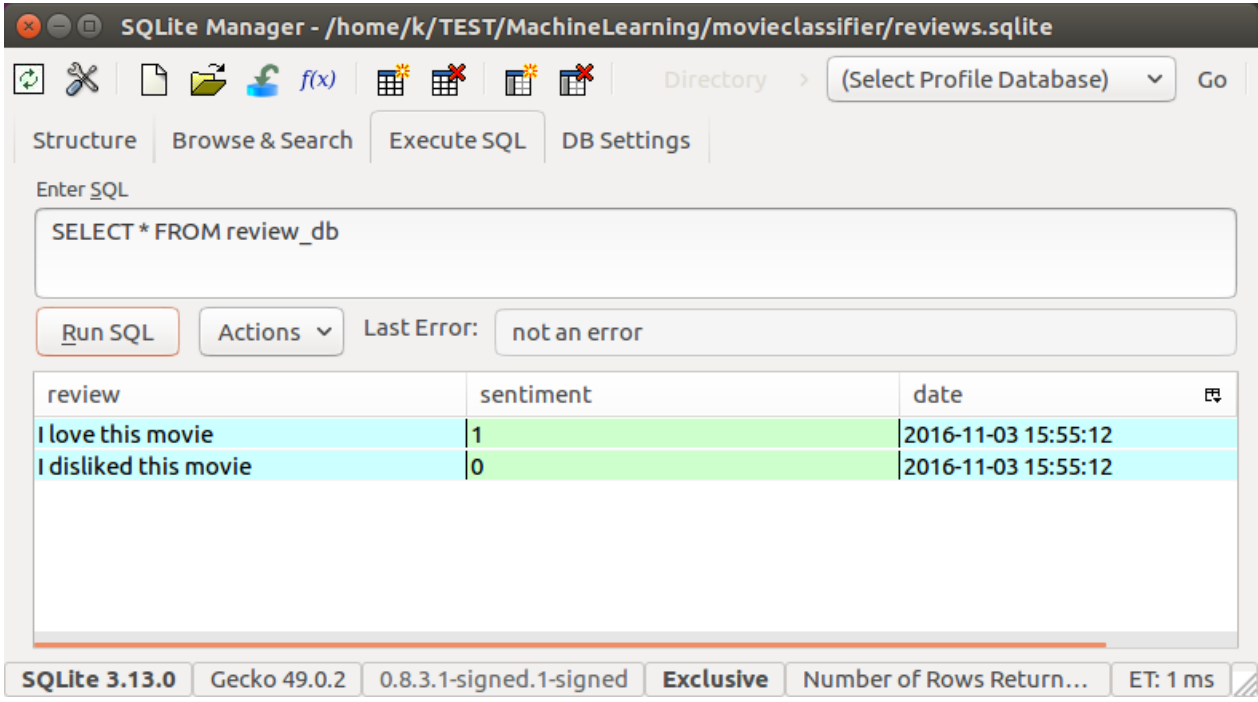
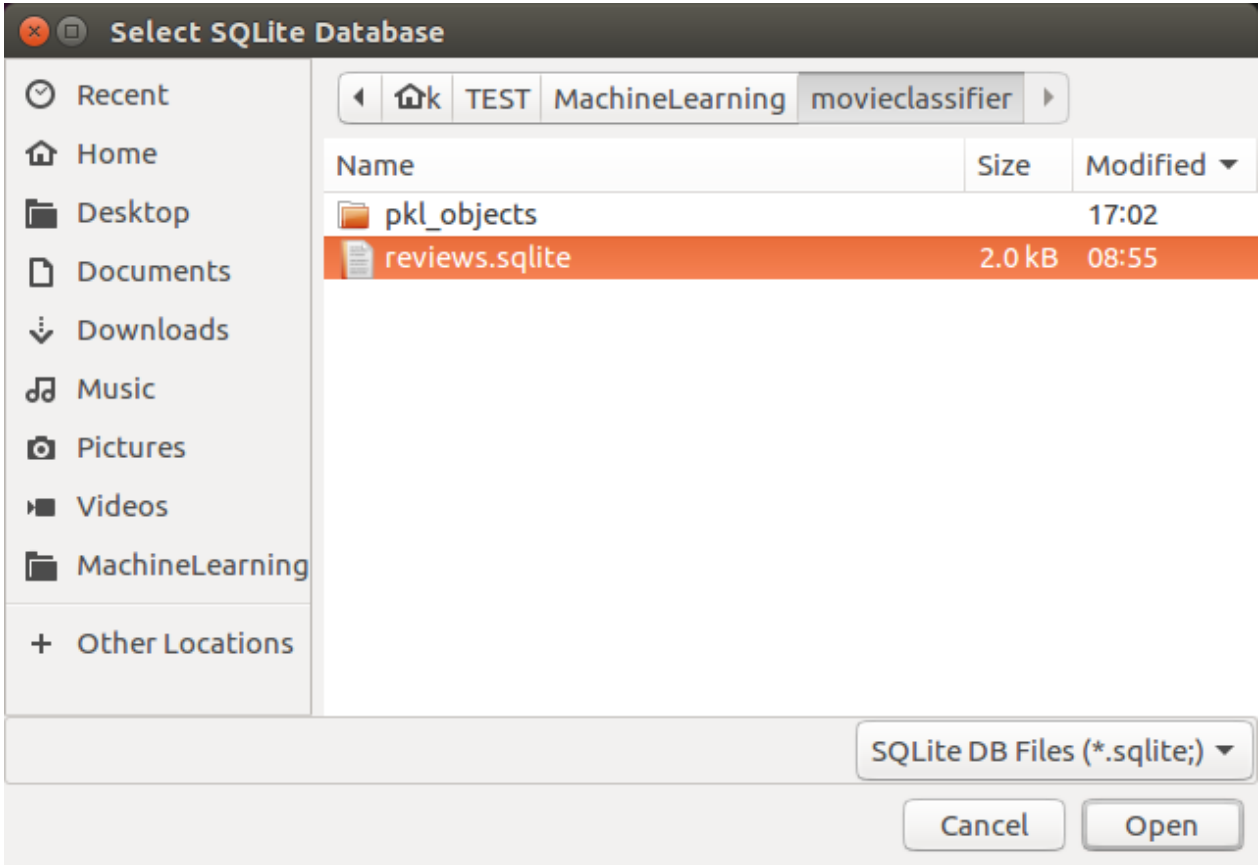
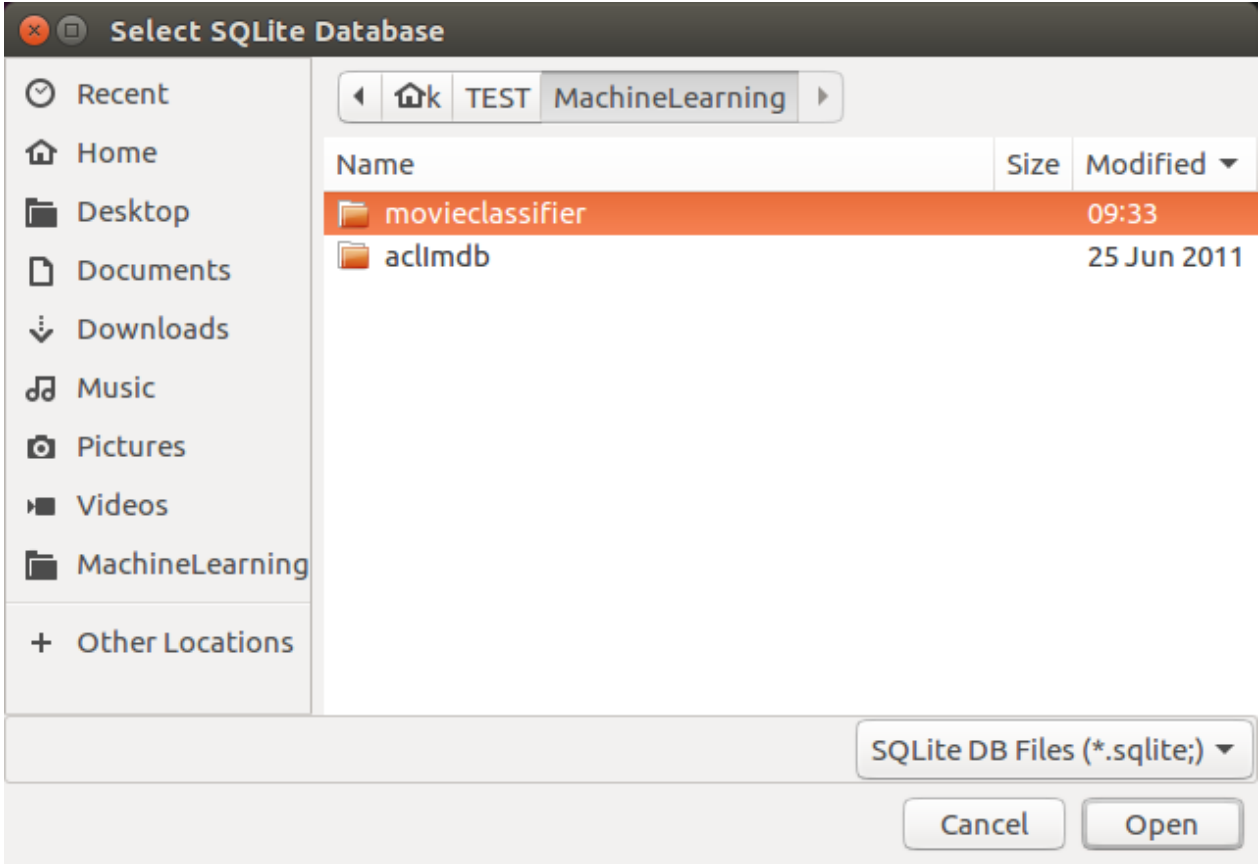
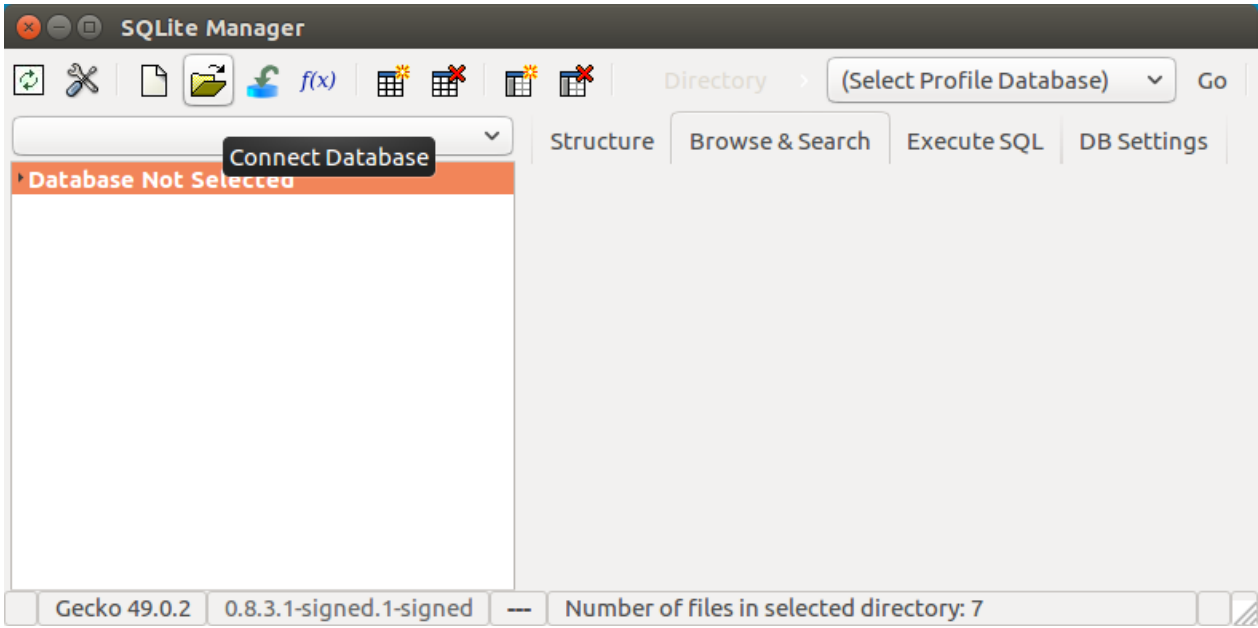
Functions def, *args, **kargs
(/python/python_functions_def.p

Functions lambda
(/python/python_functions_lamb

Built-in Functions
(/python/python_functions_built



After restarting the browser, select "SQLite Manager" under "Tools" topmenu.



map, filter, and reduce
(/python/python_fncls_map_filter)

Decorators
(/python/python_decorators.php)

List Comprehension
(/python/python_list_comprehen)

Sets (union/intersection) and
itertools - Jaccard coefficient
and shingling to check
plagiarism
(/python/python_sets_union_inte)

Hashing (Hash tables and
hashlib)
(/python/python_hash_tables_ha)

Dictionary Comprehension
with zip
(/python/python_dictionary_com)

The yield keyword
(/python/python_function_with_y)

Generator Functions and
Expressions
(/python/python_generators.php)

generator.send() method
(/python/python_function_with_)

Iterators
(/python/python_iterators.php)

Classes and Instances (__init__,
__call__, etc.)
(/python/python_classes_instanc)

if __name__ == '__main__':
(/python/python_if_name_equ)

argparse
(/python/python_argparse.php)

Exceptions
(/python/python_try_except_fina)

@static method vs class
method
(/python/python_differences_be)

Private attributes and private
methods
(/python/python_private_attribu)

bits, bytes, bitstring, and

SQLite shell

We can access sqlite via shell.

To access our **reviews.sqlite**:

```
$ sqlite3 reviews.sqlite
```

Then, we'll get a sqlite prompt. To list tables, we can type **.tables**:

```
sqlite> .tables
```

To see what's in the table:

```
sqlite> select * from review_db;
I love this movie|1|2016-11-03 15:55:12
I disliked this movie|0|2016-11-03 15:55:12
The movie is the best ever, and the most intriguing film.
|1|2016-11-04 16:07:21
The movie is the best ever, and the most intriguing film.|1|2016-11-04 16:37:22
This film served as great entertainment with its colorful cast and numerous plot twists.

The gorgeous action scenes and impressive dialogue really held the audience's attention
and kept them on the edge of their seats.|1|2016-11-05 05:43:22
This movie is the best ever.|1|2016-11-05 06:13:37
Overall, this is a great movie with a mix of a ton of laughs and a love story all rolled
into one.
If you're looking for a pick-me-up or to laugh hysterically, this is undoubtedly the mov
ie for you. |1|2016-11-05 19:36:00
sqlite>
```

To list databases:

```
sqlite> .databases
seq  name                file
---  -----
0    main                  /home/sfvue/MySites/ahaman/reviews.sqlite
```

We can exit from sqlite with the command **Ctrl + D**.

Github Jupyter Notebook source

Github Jupyter notebook is available from
FlaskAppWithEmbeddedMachineLearningSentimentAnalysis.ipynb
(<https://github.com/Einsteinish/bogotobogo-Machine-Learning.git>)

constBitStream
(/python/python_bits_bytes_bits)

json.dump(s) and json.load(s)
(/python/python-json-dumps-loads-file-read-write.php)

Python Object Serialization - pickle and json
(/python/python_serialization_pi)

Python Object Serialization - yaml and json
(/python/python_yaml_json_con)

Priority queue and heap queue data structure
(/python/python_PriorityQueue_)

Graph data structure
(/python/python_graph_data_str)

Dijkstra's shortest path algorithm
(/python/python_Dijkstras_Short)

Prim's spanning tree algorithm
(/python/python_Prims_Spanning)

Closure
(/python/python_closure.php)

Functional programming in Python
(/python/python_functional_pro)

Remote running a local file using ssh
(/python/python_ssh_remote_ru)

SQLite 3 - A. Connecting to DB, create/drop table, and insert data into a table
(/python/python_sqlite_connect)

SQLite 3 - B. Selecting, updating and deleting data
(/python/python_sqlite_select_up)

MongoDB with PyMongo I - Installing MongoDB ...
(/python/MongoDB_PyMongo/py)

Python HTTP Web Services - urllib, httplib2
(/python/python_http_web_servi)

Web scraping with Selenium

for checking domain
availability
(/python/python_Web_scraping_

REST API : Http Requests for
Humans with Flask
(/python/python-REST-API-
Http-Requests-for-Humans-
with-Flask.php)

Blog app with Tornado
(/python/Tornado/Python_Torna

Multithreading ...
(/python/Multithread/python_m

Python Network Programming
I - Basic Server / Client : A
Basics
(/python/python_network_progr

Python Network Programming
I - Basic Server / Client : B File
Transfer
(/python/python_network_progr

Python Network Programming
II - Chat Server / Client
(/python/python_network_progr

Python Network Programming
III - Echo Server using
socketserver network
framework
(/python/python_network_progr

Python Network Programming
IV - Asynchronous Request
Handling : ThreadingMixIn and
ForkingMixIn
(/python/python_network_progr

Python Interview Questions I
(/python/python_interview_ques

Python Interview Questions II
(/python/python_interview_ques

Python Interview Questions III
(/python/python_interview_ques

Python Interview Questions IV
(/python/python_interview_ques

Python Interview Questions V
(/python/python_interview_ques

Image processing with Python

Github source

Source is available from ahaman-Flask-with-Machine-Learning-Sentiment-Analysis
(<https://github.com/Einsteinish/ahaman-Flask-with-Machine-Learning-Sentiment-Analysis.git>)

Refs

Python Machine Learning, Sebastian Raschka

Next

Flask with Embedded Machine Learning II : Basic Flask App
(/python/Flask/Python_Flask_Embedding_Machine_Learning_2.php)

image library Pillow
(/python/python_image_process

Python and C++ with SIP
(/python/python_cpp_sip.php)

PyDev with Eclipse
(/python/pydev_eclipse_plugin_i

Matplotlib
(/python/python_matplotlib.php

Redis with Python
(/python/python_redis_with_pytl

NumPy array basics A
(/python/python_numpy_array_t

NumPy Matrix and Linear
Algebra
(/python/python_numpy_matrix_

Pandas with NumPy and
Matplotlib
(/python/python_Pandas_NumPy

Celluar Automata
(/python/python_cellular_autom

Batch gradient descent
algorithm
(/python/python_numpy_batch_

Longest Common Substring
Algorithm
(/python/python_longest_comm

Python Unit Test - TDD using
unittest.TestCase class
(/python/python_unit_testing.ph

Simple tool - Google page
ranking by keywords
(/python/python_site_page_rank

Google App Hello World
(/python/GoogleApp/python_Go

Google App webapp2 and
WSGI
(/python/GoogleApp/python_Go

Uploading Google App Hello
World
(/python/GoogleApp/python_Go

Python 2 vs Python 3
(/python/python_differences_Py

virtualenv and
virtualenvwrapper
(/python/python_virtualenv_virtu

Uploading a big file to AWS S3
using boto module
(/DevOps/AWS/aws_S3_uploadin

Scheduled stopping and
starting an AWS instance
(/DevOps/AWS/aws_stopping_sta

Cloudera CDH5 - Scheduled
stopping and starting services
(/Hadoop/BigData_hadoop_CDH

Removing Cloud Files -
Rackspace API with curl and
subprocess
(/python/python_Rackspace_API

Checking if a process is
running/hanging and stop/run
a scheduled task on Windows
(/python/python-Windows-
Check-if-a-Process-is-Running-
Hanging-Schtasks-Run-
Stop.php)

Apache Spark 1.3 with PySpark
(Spark Python API) Shell
(/Hadoop/BigData_hadoop_Apac

Apache Spark 1.2 Streaming
(/Hadoop/BigData_hadoop_Apac

bottle 0.12.7 - Fast and simple
WSGI-micro framework for
small web-applications ...
(/python/Bottle/Python_Bottle_F

Flask app with Apache WSGI
on Ubuntu14/CentOS7 ...
(/python/Flask/Python_Flask_Blo

Selenium WebDriver
(/python/python_Selenium_Web

Fabric - streamlining the use of
SSH for application
deployment
(/python/Fabric/python_Fabric.p

Ansible Quick Preview - Setting
up web servers with Nginx,
configure enviroments, and
deploy an App

(/DevOps/Ansible/Ansible_Setting

Neural Networks with
backpropagation for XOR using
one hidden layer
(/python/python_Neural_Network

NLP - NLTK (Natural Language
Toolkit) ...
(/python/NLTK/NLTK_install.php

RabbitMQ(Message broker
server) and Celery(Task queue)
...
(/python/RabbitMQ_Celery/pytho

OpenCV3 and Matplotlib ...
(/python/OpenCV_Python/pytho

Simple tool - Concatenating
slides using FFmpeg ...
(/FFMpeg/ffmpeg_fade_in_fade_o

iPython - Signal Processing
with NumPy
(/python/OpenCV_Python/pytho

iPython and Jupyter - Install
Jupyter, iPython Notebook,
drawing with Matplotlib, and
publishing it to Github
(/python/IPython/IPython_Jupyte

iPython and Jupyter Notebook
with Embedded D3.js
(/python/IPython/iPython_Jupyte

Downloading YouTube videos
using youtube-dl embedded
with Python
(/VideoStreaming/YouTube/yout
dl-embedding.php)

Machine Learning : scikit-learn
... (/python/scikit-
learn/scikit_machine_learning_S

Django 1.6/1.8 Web
Framework ...
(/python/Django/Python_Django

CONTACT

BogoToBogo
contactus@bogotobogo.com (mailto:#)

FOLLOW BOGOTOBOGO

f (<https://www.facebook.com/KHongSanFrancisco>) **🐦**
(<https://twitter.com/KHongTwit>) **g⁺**
(<https://plus.google.com/u/0/+KHongSanFrancisco/posts>)

ABOUT US (/ABOUT_US.PHP)

contactus@bogotobogo.com (mailto:#)

Golden Gate Ave, San Francisco, CA 94115

Golden Gate Ave, San Francisco, CA 94115

Loading [MathJax]/jax/output/CommonHTML/jax.js