×

-

(http://play.google.com/store/apps/details?id=com.analyticsvidhya.android)

×

-

(http://play.google.com/store/apps/details?id=com.analyticsvidhya.android)

MACHINE LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/MACHINE-LEARNING/)

PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/)

Tutorial to deploy Machine Learning models in Production as APIs (using Flask)

GUEST BLOG (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/GUEST-BLOG/), SEPTEMBER 28, 2017

Introduction

I remember initial days of my Machine Learning (ML) projects. I had put in a lot of efforts to build a really good model. I took expert advice on how to improve my model, I thought about feature engineering, I talked to domain experts to make sure their insights are captured. But, then I came across a problem!

How do I implement this model in real life? I had no idea about this. All the literature I had studied till now focussed on improving the models. But I didn't know what was the next step.

This is why, I have created this guide – so that you don't have to struggle with the question as I did. By end of this article, I will show you how to implement a machine learning model using Flask framework in Python.

Table of Contents

1. Options to implement Machine Learning models
2. What are APIs?
3. Python Environment Setup & Flask Basics
4. Creating a Machine Learning Model
5. Saving the Machine Learning Model: Serialization & Deserialization
6. Creating an API using Flask

1. Options to implement Machine Learning models

Most of the times, the real use of our Machine Learning model lies at the heart of a product – that maybe a small component of an automated mailer system or a chatbot. These are the times when the barriers seem unsurmountable.

For example, majority of ML folks use R / Python for their experiments. But consumer of those ML models would be software engineers who use a completely different stack. There are two ways via which this problem can be solved:

- **Option 1: Rewriting the whole code in the language that the software engineering folks work.** The above seems like a good idea, but the time & energy required to get those intricate models replicated would be utterly waste. Majority of languages like JavaScript, do not have great libraries to perform ML. One would be wise to stay away from it.

- **Option 2 – API-first approach –** Web APIs have made it easy for cross-language applications to work well. If a frontend developer needs to use your ML Model to create a ML powered web application, they would just need to get the URL Endpoint from where the API is being served.

2. What are APIs?

In simple words, an API is a (hypothetical) contract between 2 softwares saying if the user software provides input in a pre-defined format, the later with extend its functionality and provide the outcome to the user software.

You can read this article to understand why APIs are a popular choice amongst developers:

- History of APIs (http://apievangelist.com/2012/12/20/history-of-apis/)
- Introduction to APIs (https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-apis-application-programming-interfaces-5-apis-a-data-scientist-must-know/)

Majority of the Big Cloud providers and smaller Machine Learning focussed companies provide ready-to-use APIs. They cater to the needs of developers / businesses that don't have expertise in ML, who want to implement ML in their processes or product suites.

One such example of Web APIs offered is the Google Vision API (https://cloud.google.com/vision/)

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/09/26152341/687474703a2f2f7777772e7075626c69636b6579312e6a702f32303136

All you need is a simple REST call to the API via SDKs (Software Development Kits) provided by Google. Click here (https://github.com/GoogleCloudPlatform/cloud-vision/tree/master/python) to get an idea of what can be done using Google Vision API.

Sounds marvellous right! In this article, we'll understand how to create our own Machine Learning API using Flask, a web framework in Python.

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/09/26152627/Image2.png)

NOTE: Flask isn't the only web-framework available. There is Django, Falcon, Hug and many more. For R, we have a package called plumber (https://github.com/trestletech/plumber).

3. Python Environment Setup & Flask Basics

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/09/26152753/Image3.png)

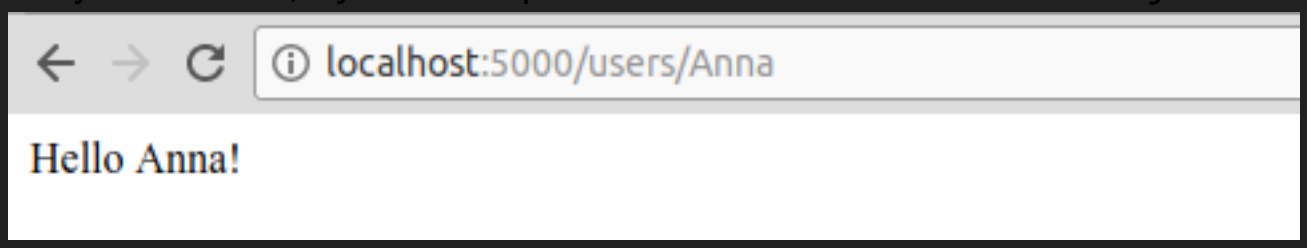
- Creating a virtual environment using Anaconda. If you need to create your workflows in Python and keep the dependencies separated out or share the environment settings, Anaconda distributions are a great option.
    - You'll find a miniconda installation for Python here (https://conda.io/miniconda.html)
    - `wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh`
    - `bash Miniconda3-latest-Linux-x86_64.sh`
    - Follow the sequence of questions.
    - `source .bashrc`
    - If you run: `conda` , you should be able to get the list of commands & help.
    - To create a new environment, run:
      `conda create --name <environment-name> python=3.6`
    - Follow the steps & once done run: `source activate <environment-name>`
    - Install the python packages you need, the two important are: `flask & gunicorn` .

- We'll try out a simple Flask Hello-World application and serve it using gunicorn:
    - Open up your favourite text editor and create `hello-world.py` file in a folder
    - Write the code below:

```
"""Filename: hello-world.py
  """

  from flask import Flask

  app = Flask(__name__)

  @app.route('/users/<string:username>')
  def hello_world(username=None):

      return("Hello {}!".format(username))
```

- Save the file and return to the terminal.
- To serve the API (to start running it), execute:
  `gunicorn --bind 0.0.0.0:8000 hello-world:app` on your terminal.
- If you get the repsonses below, you are on the right track:

  (https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/09/26153343/image41.png)
- On you browser, try out: `https://localhost:8000/users/any-name`



Viola! You wrote your first Flask application. As you have now experienced with a few simple steps, we were able to create web-endpoints that can be accessed locally.

Using Flask, we can wrap our Machine Learning models and serve them as Web APIs easily. Also, if we want to create more complex web applications (that includes JavaScript *gasps*) we just need a few modifications.

4. Creating a Machine Learning Model

- We'll be taking up the Machine Learning competition: Loan Prediction Competition (https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii). The main objective is to set a pre-processing pipeline and creating ML Models with goal towards making the ML Predictions easy while deployments.

```
import os
import json
import numpy as np
import pandas as pd
from sklearn.externals import joblib
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import make_pipeline

import warnings
warnings.filterwarnings("ignore")
```

- Saving the datasets in a folder:

```
!ls /home/pratos/Side-Project/av_articles/flask_api/data/
```

```
test.csv  training.csv
```

```
data = pd.read_csv('../data/training.csv')
```

```
list(data.columns)
```

```
['Loan_ID',
 'Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area',
 'Loan_Status']
```

```
data.shape
```

```
(614, 13)
```

- Finding out the null / Nan values in the columns:

```
for _ in data.columns:
    print("The number of null values in:{} == {}".format(_, data[_].isnull().sum()))
```

```
The number of null values in:Loan_ID == 0
The number of null values in:Gender == 13
The number of null values in:Married == 3
The number of null values in:Dependents == 15
The number of null values in:Education == 0
The number of null values in:Self_Employed == 32
The number of null values in:ApplicantIncome == 0
The number of null values in:CoapplicantIncome == 0
The number of null values in:LoanAmount == 22
The number of null values in:Loan_Amount_Term == 14
The number of null values in:Credit_History == 50
The number of null values in:Property_Area == 0
The number of null values in:Loan_Status == 0
```

- Next step is creating training and testing datasets:

```
pred_var = ['Gender','Married','Dependents','Education','Self_Employed','ApplicantInc
ome','CoapplicantIncome',\
            'LoanAmount','Loan_Amount_Term','Credit_History','Property_Area']

X_train, X_test, y_train, y_test = train_test_split(data[pred_var], data['Loan_Status
'], \
                                                    test_size=0.25, random_state=42)
```

- To make sure that the pre-processing steps are followed religiously even after we are done with experimenting and we do not miss them while predictions, we'll create a **custom pre-processing Scikit-learn estimator**.

To follow the process on how we ended up with this estimator, refer this notebook (https://github.com/pratos/flask_api/blob/master/notebooks/AnalyticsVidhya%20Article%20-%20ML%20Model%20approach.ipynb)

```
from sklearn.base import BaseEstimator, TransformerMixin


class PreProcessing(BaseEstimator, TransformerMixin):
```

```
    """Custom Pre-Processing estimator for our use-case
    """

    def __init__(self):
        pass

    def transform(self, df):
        """Regular transform() that is a help for training, validation & testing data
sets
           (NOTE: The operations performed here are the ones that we did prior to thi
s cell)
        """
        pred_var = ['Gender','Married','Dependents','Education','Self_Employed','Appl
icantIncome',\
                    'CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_Histo
ry','Property_Area']

        df = df[pred_var]

        df['Dependents'] = df['Dependents'].fillna(0)
        df['Self_Employed'] = df['Self_Employed'].fillna('No')
        df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(self.term_mean_)
        df['Credit_History'] = df['Credit_History'].fillna(1)
        df['Married'] = df['Married'].fillna('No')
        df['Gender'] = df['Gender'].fillna('Male')
        df['LoanAmount'] = df['LoanAmount'].fillna(self.amt_mean_)

        gender_values = {'Female' : 0, 'Male' : 1}
        married_values = {'No' : 0, 'Yes' : 1}
        education_values = {'Graduate' : 0, 'Not Graduate' : 1}
        employed_values = {'No' : 0, 'Yes' : 1}
        property_values = {'Rural' : 0, 'Urban' : 1, 'Semiurban' : 2}
        dependent_values = {'3+': 3, '0': 0, '2': 2, '1': 1}
        df.replace({'Gender': gender_values, 'Married': married_values, 'Education':
education_values, \
```

```
                    'Self_Employed': employed_values, 'Property_Area': property_value
s, \
                    'Dependents': dependent_values}, inplace=True)

        return df.as_matrix()

    def fit(self, df, y=None, **fit_params):
        """Fitting the Training dataset & calculating the required values from train
           e.g: We will need the mean of X_train['Loan_Amount_Term'] that will be use
d in
                transformation of X_test
        """

        self.term_mean_ = df['Loan_Amount_Term'].mean()
        self.amt_mean_ = df['LoanAmount'].mean()
        return self
```

- Convert y_train & y_test to np.array:

```
y_train = y_train.replace({'Y':1, 'N':0}).as_matrix()
y_test = y_test.replace({'Y':1, 'N':0}).as_matrix()
```

We'll create a pipeline to make sure that all the preprocessing steps that we do are just a single scikit-learn estimator.

```
pipe = make_pipeline(PreProcessing(),
                     RandomForestClassifier())
```

```
pipe
```

```
Pipeline(memory=None,
     steps=[('preprocessing', PreProcessing()), ('randomforestclassifier', RandomFore
stClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False))])
```

To search for the best hyper-parameters (degree for Polynomial Features & alpha for Ridge), we'll do a Grid Search:

- Defining param_grid:

```
param_grid = {"randomforestclassifier__n_estimators" : [10, 20, 30],
              "randomforestclassifier__max_depth" : [None, 6, 8, 10],
              "randomforestclassifier__max_leaf_nodes": [None, 5, 10, 20],
            "randomforestclassifier__min_impurity_split": [0.1, 0.2, 0.3]}
```

- Running the Grid Search:

```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=3)
```

- Fitting the training data on the pipeline estimator:

```
grid.fit(X_train, y_train)
```

```
GridSearchCV(cv=3, error_score='raise',
       estimator=Pipeline(memory=None,
     steps=[('preprocessing', PreProcessing()), ('randomforestclassifier', RandomFore
stClassifier(bootstrap=True, class_weight=None, criterion='gini',
           max_depth=None, max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impu..._jobs=1,
           oob_score=False, random_state=None, verbose=0,
           warm_start=False))]),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'randomforestclassifier__n_estimators': [10, 20, 30], 'randomfores
tclassifier__max_leaf_nodes': [None, 5, 10, 20], 'randomforestclassifier__min_impurit
y_split': [0.1, 0.2, 0.3], 'randomforestclassifier__max_depth': [None, 6, 8, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

- Let's see what parameter did the Grid Search select:

```
print("Best parameters: {}".format(grid.best_params_))
```

```
Best parameters: {'randomforestclassifier__n_estimators': 30, 'randomforestclassifier
__max_leaf_nodes': 20, 'randomforestclassifier__min_impurity_split': 0.2, 'randomfore
stclassifier__max_depth': 8}
```

- Let's score:

```
print("Validation set score: {:.2f}".format(grid.score(X_test, y_test)))
```

```
Validation set score: 0.79
```

- Load the test set:

```
test_df = pd.read_csv('../data/test.csv', encoding="utf-8-sig")
test_df = test_df.head()
```

```
grid.predict(test_df)
```

```
array([1, 1, 1, 1, 1])
```

Our pipeline is looking pretty swell & fairly decent to go the most important step of the tutorial: **Serialize the Machine Learning Model**

5. Saving Machine Learning Model : Serialization & Deserialization

> *In computer science, in the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment.*

In Python, pickling is a standard way to store objects and retrieve them as their original state. To give a simple example:

```
list_to_pickle = [1, 'here', 123, 'walker']

#Pickling the list
import pickle

list_pickle = pickle.dumps(list_to_pickle)
```

```
list_pickle
```

```
b'\x80\x03]q\x00(K\x01X\x04\x00\x00\x00hereq\x01K{X\x06\x00\x00\x00walkerq\x02e.'
```

When we load the pickle back:

```
loaded_pickle = pickle.loads(list_pickle)
```

```
loaded_pickle
```

```
[1, 'here', 123, 'walker']
```

We can save the pickled object to a file as well and use it. This method is similar to creating `.rda` files for folks who are familiar with R Programming.

NOTE: Some people also argue against using pickle for serialization(1) (https://render.githubusercontent.com/view/ipynb?commit=dac6eb737b58b73f9b9b323a9a39c0d1fe8cbfee&enc_url=68747470733a2f2f7261772e67697468 `h5py` could also be an alternative.

We have a custom Class that we need to import while running our training, hence we'll be using dill module to packup the estimator Class with our grid object.

It is advisable to create a separate `training.py` file that contains all the code for training the model (See here for example (https://github.com/pratos/flask_api/blob/master/flask_api/utils.py)).

- To install `dill`

```
!pip install dill
```

```
Requirement already satisfied: dill in /home/pratos/miniconda3/envs/ordermanagement/l
ib/python3.5/site-packages
```

```
import dill as pickle
filename = 'model_v1.pk'
```

```
with open('../flask_api/models/'+filename, 'wb') as file:
        pickle.dump(grid, file)
```

So our model will be saved in the location above. Now that the model is pickled, creating a Flask wrapper around it would be the next step.

Before that, to be sure that our pickled file works fine – let's load it back and do a prediction:

```
with open('../flask_api/models/'+filename ,'rb') as f:
    loaded_model = pickle.load(f)
```

```
loaded_model.predict(test_df)
```

```
array([1, 1, 1, 1, 1])
```

Since, we already have the preprocessing steps required for the new incoming data present as a part of the pipeline, we just have to run predict(). While working with scikit-learn, it is always easy to work with pipelines.

Estimators and pipelines save you time and headache, even if the initial implementation seems to be ridiculous. Stitch in time, saves nine!

6. Creating an API using Flask

We'll keep the folder structure as simple as possible:

```
·100% ➜ tree
.
├── flask_api.yml
├── hello-world.py
├── hello-world.pyc
├── LICENSE
├── models
│   └── model_v1.pk
├── README.md
├── requirements.txt
├── server.py
└── server.pyc

1 directory, 9 files
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/09/26155407/Image5.png)

There are three important parts in constructing our wrapper function, `apicall()` :

- Getting the `request` data (for which predictions are to be made)
- Loading our `pickled estimator`
- `jsonify` our predictions and send the response back with `status code: 200`

HTTP messages are made of a header and a body. As a standard, majority of the body content sent across are in `json` format. We'll be sending (`POST url-endpoint/`) the incoming data as batch to get predictions.

(**NOTE:** You can send plain text, XML, csv or image directly but for the sake of interchangeability of the format, it is advisable to use `json`)

```
"""Filename: server.py
"""

import os
import pandas as pd
from sklearn.externals import joblib
from flask import Flask, jsonify, request

app = Flask(__name__)
```

```
@app.route('/predict', methods=['POST'])
def apicall():
    """API Call

    Pandas dataframe (sent as a payload) from API Call
    """
    try:
        test_json = request.get_json()
        test = pd.read_json(test_json, orient='records')

        #To resolve the issue of TypeError: Cannot compare types 'ndarray(dtype=int64
)' and 'str'
        test['Dependents'] = [str(x) for x in list(test['Dependents'])]

        #Getting the Loan_IDs separated out
        loan_ids = test['Loan_ID']

    except Exception as e:
        raise e

    clf = 'model_v1.pk'

    if test.empty:
        return(bad_request())
    else:
        #Load the saved model
        print("Loading the model...")
        loaded_model = None
        with open('./models/'+clf,'rb') as f:
            loaded_model = pickle.load(f)

        print("The model has been loaded...doing predictions now...")
        predictions = loaded_model.predict(test)

        """Add the predictions as Series to a new pandas dataframe
```

```
                            OR
            Depending on the use-case, the entire test data appended with the new file
s
        """
        prediction_series = list(pd.Series(predictions))

        final_predictions = pd.DataFrame(list(zip(loan_ids, prediction_series)))

        """We can be as creative in sending the responses.
           But we need to send the response codes as well.
        """
        responses = jsonify(predictions=final_predictions.to_json(orient="records"))
        responses.status_code = 200

        return (responses)
```

Once done, run: `gunicorn --bind 0.0.0.0:8000 server:app`

Let’s generate some prediction data and query the API running locally at `https:0.0.0.0:8000/predict`

```
import json
import requests
```

```
"""Setting the headers to send and accept json responses
"""
header = {'Content-Type': 'application/json', \
                  'Accept': 'application/json'}

"""Reading test batch
"""
df = pd.read_csv('../data/test.csv', encoding="utf-8-sig")
df = df.head()

"""Converting Pandas Dataframe to json
"""
data = df.to_json(orient='records')
```

```
data
```


```
'[{"Loan_ID":"LP001015","Gender":"Male","Married":"Yes","Dependents":"0","Education":
"Graduate","Self_Employed":"No","ApplicantIncome":5720,"CoapplicantIncome":0,"LoanAmo
unt":110.0,"Loan_Amount_Term":360.0,"Credit_History":1.0,"Property_Area":"Urban"},{"L
oan_ID":"LP001022","Gender":"Male","Married":"Yes","Dependents":"1","Education":"Grad
uate","Self_Employed":"No","ApplicantIncome":3076,"CoapplicantIncome":1500,"LoanAmoun
t":126.0,"Loan_Amount_Term":360.0,"Credit_History":1.0,"Property_Area":"Urban"},{"Loa
n_ID":"LP001031","Gender":"Male","Married":"Yes","Dependents":"2","Education":"Gradua
te","Self_Employed":"No","ApplicantIncome":5000,"CoapplicantIncome":1800,"LoanAmount"
:208.0,"Loan_Amount_Term":360.0,"Credit_History":1.0,"Property_Area":"Urban"},{"Loan_
ID":"LP001035","Gender":"Male","Married":"Yes","Dependents":"2","Education":"Graduate
","Self_Employed":"No","ApplicantIncome":2340,"CoapplicantIncome":2546,"LoanAmount":1
00.0,"Loan_Amount_Term":360.0,"Credit_History":null,"Property_Area":"Urban"},{"Loan_I
D":"LP001051","Gender":"Male","Married":"No","Dependents":"0","Education":"Not Gradua
te","Self_Employed":"No","ApplicantIncome":3276,"CoapplicantIncome":0,"LoanAmount":78
.0,"Loan_Amount_Term":360.0,"Credit_History":1.0,"Property_Area":"Urban"}]'
```

```
"""POST <url>/predict
"""
resp = requests.post("http://0.0.0.0:8000/predict", \
                     data = json.dumps(data),\
                     headers= header)
```

```
resp.status_code
```

```
200
```

```
"""The final response we get is as follows:
"""
resp.json()
```

```
{'predictions': '[{"0":"LP001015","1":1},{...
```

End Notes

We have half the battle won here, with a working API that serves predictions in a way where we take one step towards integrating our ML solutions right into our products. This is a very basic API that will help with prototyping a data product, to make it as fully functional, production ready API a few more additions are required that aren't in the scope of Machine Learning.
There are a few things to keep in mind when adopting API-first approach:

- Creating APIs out of spaghetti code is next to impossible, so approach your Machine Learning workflow as if you need to create a clean, usable API as a deliverable. Will save you a lot of effort to jump hoops later.
- Try to use version control for models and the API code, Flask doesn't provide great support for version control. Saving and keeping track of ML Models is difficult, find out the least messy way that suits you. This article (https://medium.com/towards-data-science/how-to-version-control-your-machine-learning-task-cad74dce44c4) talks about ways to do it.
- Specific to sklearn models (as done in this article), if you are using custom estimators for preprocessing or any other related task make sure you keep the estimator and training code together so that the model pickled would have the estimator class tagged along.

Next logical step would be creating a workflow to deploy such APIs out on a small VM. There are various ways to do it and we'll be looking into those in the next article.

Code & Notebooks for this article: pratos/flask_api (https://github.com/pratos/flask_api)

Sources & Links:

[1]. Don't Pickle your data. (http://www.benfrederickson.com/dont-pickle-your-data/)

[2]. Building Scikit Learn compatible transformers. (http://www.dreisbach.us/articles/building-scikit-learn-compatible-transformers/)

[3]. Using jsonify in Flask. (http://flask.pocoo.org/docs/0.10/security/#json-security)

[4]. Flask-QuickStart. (http://blog.luisrei.com/articles/flaskrest.html)

About the Author

Prathamesh Sarang (https://www.linkedin.com/in/prathamesh-sarang-392b9219/) works as a Data Scientist at Lemoxo Technologies. Data Engineering is his latest love, turned towards the *nix faction recently. Strong advocate of "Markdown for everyone".

(https://www.analyticsvidhya.com/blog/2017/09/machine-learning-models-as-apis-using-flask/?share=facebook&nb=1&nb=1)

(https://www.analyticsvidhya.com/blog/2017/09/machine-learning-models-as-apis-using-flask/?share=google-plus-1&nb=1&nb=1)

(https://www.analyticsvidhya.com/blog/2017/09/machine-learning-models-as-apis-using-flask/?share=twitter&nb=1&nb=1)

(https://www.analyticsvidhya.com/blog/2017/09/machine-learning-models-as-apis-using-flask/?share=pocket&nb=1&nb=1)

(https://www.analyticsvidhya.com/blog/2017/09/machine-learning-models-as-apis-using-flask/?share=reddit&nb=1&nb=1)

Like this:

Loading...

TAGS : API (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/API/), FLASK (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/FLASK/), MACHINE LEARNING DEPLOYMENT (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/MACHINE-LEARNING-DEPLOYMENT/), PICKLE MACHINE LEARNING MODEL (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PICKLE-MACHINE-LEARNING-MODEL/)

NEXT ARTICLE

Bollinger Bands and their use in Stock Market Analysis (using Quandl & tidyverse in R)

(https://www.analyticsvidhya.com/blog/2017/10/comparative-stock-market-analysis-in-r-using-quandl-tidyverse-part-i/)

•••

PREVIOUS ARTICLE

How to build your first Machine Learning model on iPhone (Intro to Apple's CoreML)

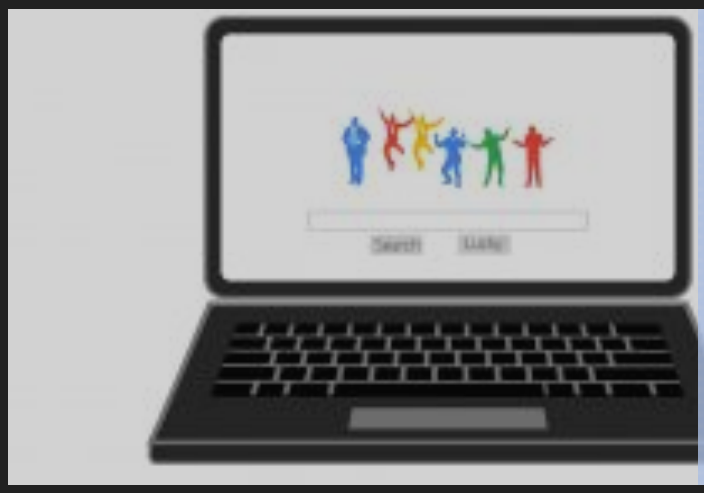
(https://www.analyticsvidhya.com/blog/2017/09/build-machine-learning-iphone-apple-coreml/)

(https://www.analyticsvidhya.com/blog/author/guest-blog/)

Guest Blog (Https://Www.analyticsvidhya.com/Blog/Author/Guest-Blog/)

RELATED ARTICLES

PRANAV DAR (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/PRANAV/) ADITYA SHARMA (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/ADITYA/) ANSHUMAN SINGH (HTTPS://WWW



(https://www.analyticsvidhya.com/blog/2018/04/deep-learning-image-dataset-using-python/)

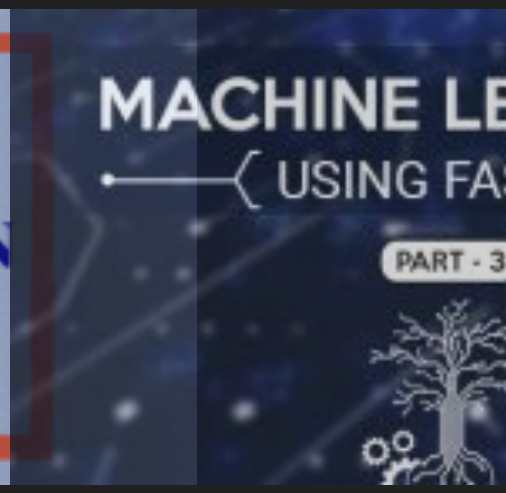
Develop Your Own Personal Deep Learning Image Dataset using this Python Script

(https://www.analyticsvidhya.com/blog/2018/04/curate-deep-learning-image-dataset-using-python/)

(https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/)

Complete Guide on DataFrame Operations in PySpark

(https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/)



(https://www.analyticsvidhya.com/blog/2016/10/spa-a-random-forest-from-scra-real-world-data-products-p-part-3/)

Building a Random Forest fr Understanding Real World D Part 3)

SHIVAM BANSAL (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/SHIVAM/) ANALYTICS VIDHYA CONTENT TEAM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/ANALYTICS-VIDHYA-CONTENT-TEAM/)

(https://www.analytics... 2017/07/30 (https://www.analytics... 2017/09/...
questions-test-data-scientist-natural-language- vs-vs-python-tool-learn/) different-methods-deal-te...
processing-solution-skilltest-nlp/) python/

30 Questions to test a data scientist on Natural Language Processing [Solution: Skilltest – NLP] (https://www.analytics... 0-questions-test-data-scientist-natural-language-processing-solution-skilltest-nlp/)

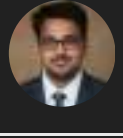
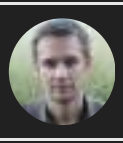
Python vs. R (vs. SAS) – which tool should I learn? (https://www.analytics... vs-vs-python-tool-learn/)

Ultimate guide to deal with T... Python) – for Data Scientists (https://www.analytics... e-different-methods-deal-tex... python/)

This article is quite old and you might not get a prompt response from the author. We request you to post this comment on Analytics Vidhya's **Discussion portal** (https://discuss.analyticsvidhya.com/) to get your queries resolved

TOP ANALYTICS VIDHYA USERS

Rank	Name	Points
1	 SRK (https://datahack.analyticsvidhya.com/user/profile/SRK)	9688
2	 Rohan Rao (https://datahack.analyticsvidhya.com/user/profile/Rohan_Rao)	9200

3		aayushmnit (https://datahack.analyticsvidhya.com/user/profile/aayushmnit)	7779
4		mark12 (https://datahack.analyticsvidhya.com/user/profile/mark12)	7212
5		sonny (https://datahack.analyticsvidhya.com/user/profile/sonny)	5947

More Rankings (http://datahack.analyticsvidhya.com/users)

POPULAR POSTS

- 24 Ultimate Data Science Projects To Boost Your Knowledge and Skills (& can be accessed freely) (https://www.analyticsvidhya.com/blog/2018/05/24-ultimate-data-science-projects-to-boost-your-knowledge-and-skills/)
- A Complete Tutorial to Learn Data Science with Python from Scratch (https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/)
- Essentials of Machine Learning Algorithms (with Python and R Codes) (https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/)
- The 25 Best Data Science and Machine Learning GitHub Repositories from 2018 (https://www.analyticsvidhya.com/blog/2018/12/best-data-science-machine-learning-projects-github/)
- 7 Types of Regression Techniques you should know! (https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/)
- Understanding Support Vector Machine algorithm from examples (along with code) (https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)
- 6 Easy Steps to Learn Naive Bayes Algorithm (with codes in Python and R) (https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/)
- The 15 Most Popular Data Science and Machine Learning Articles on Analytics Vidhya in 2018 (https://www.analyticsvidhya.com/blog/2018/12/most-popular-articles-analytics-vidhya-2018/)

RECENT POSTS

DataHack Radio #16: Kaggle Grandmaster SRK's Journey and Advice for Data Science Competitions (https://www.analyticsvidhya.com/blog/2019/01/datahack-radio-tips-crack-data-science-competitions-kaggle-grandmaster/)

JANUARY 24, 2019

Introduction to Monte Carlo Tree Search: The Game-Changing Algorithm behind DeepMind's AlphaGo (https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/)

JANUARY 24, 2019

Top 10 Presentations from rstudio::conf 2019 – The Best R Conference of the Year! (https://www.analyticsvidhya.com/blog/2019/01/top-highlights-rstudioconf-2019-best-r-conference/)

JANUARY 22, 2019

Must-Read Tutorial to Learn Sequence Modeling (deeplearning.ai Course #5) (https://www.analyticsvidhya.com/blog/2019/01/sequence-models-deeplearning/)

JANUARY 21, 2019

Analytics Vidhya
Learn everything about analytics

ANALYTICS VIDHYA

About Us (http://www.analyticsvidhya.com/about-me/)

Our Team (https://www.analyticsvidhya.com/about-me/team/)

Career (https://www.analyticsvidhya.com/career-analytics-vidhya/)

Contact Us (https://www.analyticsvidhya.com/contact/)

Write for us (https://www.analyticsvidhya.com/about-me/write/)

DATA SCIENTISTS

Blog (https://www.analyticsvidhya.com/blog/)

Hackathon (https://datahack.analyticsvidhya.com/)

Discussions (https://discuss.analyticsvidhya.com/)

Apply Jobs (https://www.analyticsvidhya.com/jobs/)

Leaderboard (https://datahack.analyticsvidhya.com/)

COMPANIES

Post Jobs (https://www.analyticsvidhya.com/corporate/)

Trainings (https://trainings.analyticsvidhya.com/)

Hiring Hackathons (https://datahack.analyticsvidhya.com/)

Advertising (https://www.analyticsvidhya.com/contact/)

Reach Us (https://www.analyticsvidhya.com/contact/)

JOIN OUR COMMUNITY :

(https://www.facebook.com/Analyticsvidhya) 48336

(https://twitter.com/Analytics_vidhya) 19433 Followers

(https://www.facebook.com/Analyticsvidhya) 3057

(https://plus.google.com/+Analyticsvidhya) Followers

(https://www.linkedin.com/groups/Analytics-vidhya) 7513

(https://www.linkedin.com/groups/Analytics-vidhya) Followers

Subscribe to emailer >

Privacy Policy (https://www.analyticsvidhya.com/privacy-policy/)

Terms of Use (https://www.analyticsvidhya.com/terms/)

Refund Policy (https://www.analyticsvidhya.com/refund-policy/)

Don't have an account? Si

cribe!