How to deploy Machine Learning models as a Microservice using FastAPI

Microservice imlementation using FastAPI | Ashutosh Tripathi | Data Science Duniya

As of today, FastAPI is the most popular web framework for building microservices with python 3.6+ versions. By deploying machine learning models as microservice-based architecture, we make code components re-usable, highly maintained, ease of testing, and of-course the quick response time. FastAPI is built over ASGI (Asynchronous Server Gateway Interface) instead of flask's WSGI (Web Server Gateway Interface). This is the reason it is faster as compared to flask-based APIs.

It has a data validation system that can detect **any invalid data type at the runtime** and returns the reason for bad inputs to the user in the JSON format only which frees developers from managing this exception explicitly.

In this post, the objective is to explain the machine learning model deployment as microservices with the help of FastAPI. So we will focus on that part, not on the model training.

complete source code is also available in github repository. You will get the repository link at the end of the post.

Step 1. Make your model for which you want to create the API ready

To create API for prediction we need the model ready so I have written few lines of code that train the model and save it as LRClassifier.pkl file in the local disk. I have not focused on exploratory data analysis, pre-processing or feature engineering part as that is out of the scope for this article.

```
import pandas as pdfrom sklearn.model_selection import train_test_splitfrom
sklearn.linear_model import LogisticRegressionimport pickle# Load dataseturl =
""names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class']dataset =
pd.read_csv(filepath_or_buffer=url,header=None,sep=',',names=names)# Split-out
validation datasetarray = dataset.valuesX = array[:,0:4]y = array[:,4]X_train,
X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1,
shuffle=True)classifier = LogisticRegression()classifier.fit(X_train,y_train)save the
model to diskpickle.dump(classifier, open('LRClassifier.pkl', 'wb'))load the model
from diskloaded_model = pickle.load(open('LRClassifier.pkl', 'rb'))result =
loaded_model.score(X_test, y_test)print(result)
```

Jupyter snippet of the above code:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pickle

# Load dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv(filepath_or_buffer=url,header=None,sep=',',names=names)
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1, shuffle=True)

classifier = LogisticRegression()
classifier.fit(X_train,y_train)

# save the model to disk
pickle.dump(classifier, open('LRClassifier.pkl', 'wb'))

# load the model from disk
loaded_model = pickle.load(open('LRClassifier.pkl', 'rb'))

result = loaded_model.score(X_test, y_test)
print(result)
```

```
0.9666666666666667
```

Logistic Regression python code snippet

Step 2. Create API using FastAPI framework

Start from scratch so that you don't get any error:

- Open VS code or any other editor of your choice. I use VS code
- Using file meny open the directory where you want to work
- open the terminal and create the virtual environment as below:
- python -m venv venv-name
- Activate venv using venv-name\Scripts\activate

Install Libraries:

- pip install pandas
- pip install numpy
- pip install sklearn
- pip install pickle
- pip install FastAPI

Import libraries as shown in below code.

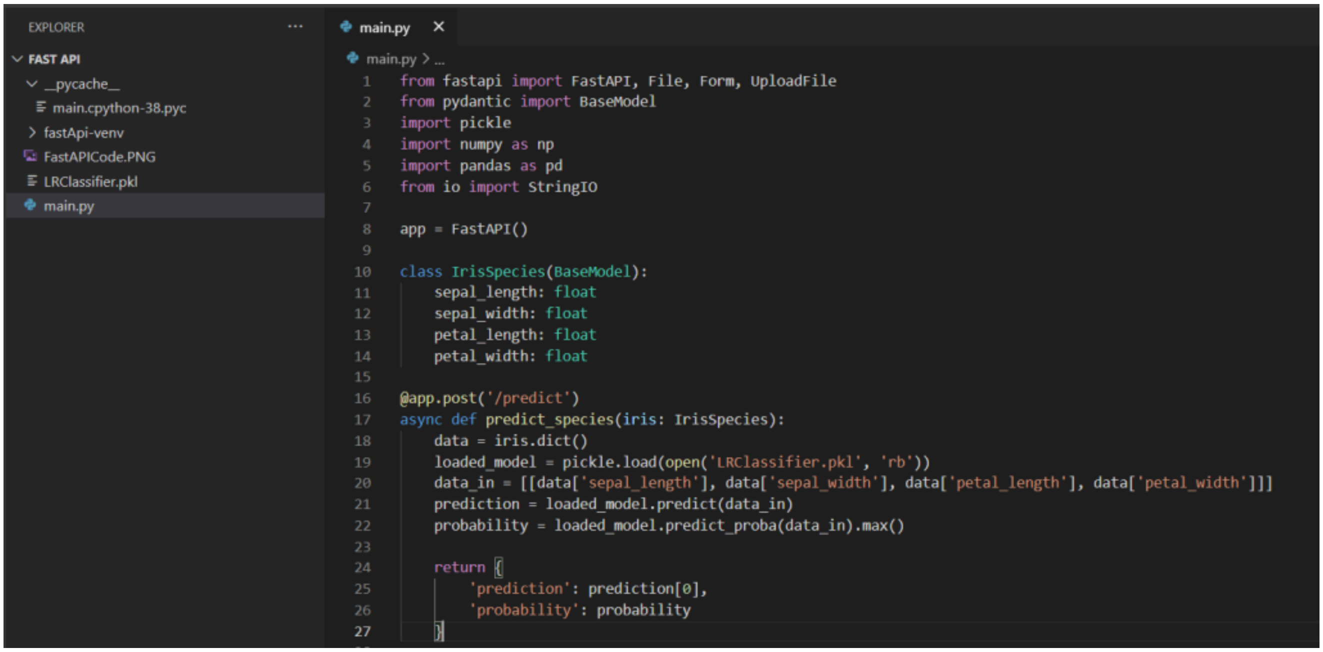
- create a `FastAPI` "instance" and assign it to app
- Here the `app` variable will be an "instance" of the class `FastAPI`.
- This will be the main point of interaction to create all your API.
- This `app` is the same one referred by `uvicorn` in the command as below:

```
$ uvicorn main:app --reload
```

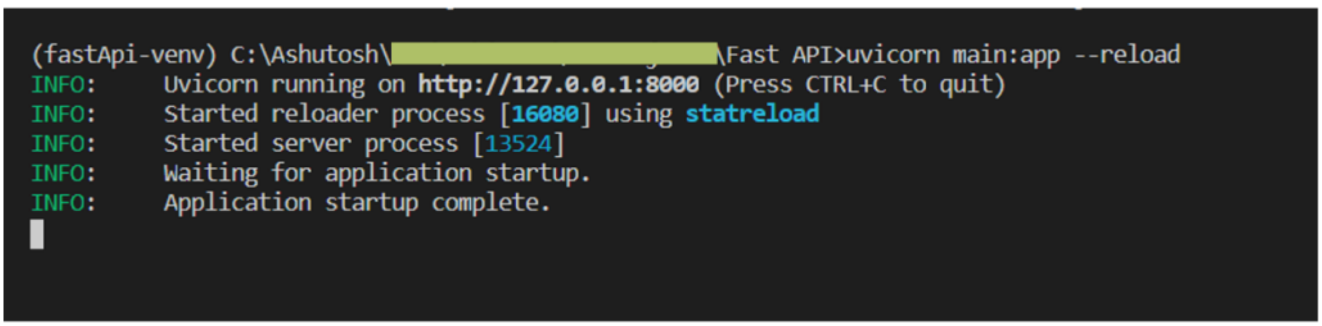
- Here main is the name of file where you are writing the code. you can give any name but same you have to use while executing in the command in place of main.
- When you need to send data from a client (let's say, a browser) to your API, you send it as a .
- A body is data sent by the client to your API. A body is the data your API sends to the client.
- Your API almost always has to send a body. But clients don't necessarily need to send bodies all the time.
- To declare a body, you use models with all their power and benefits.
- Then you declare your data model as a class that inherits from `BaseModel`.
- Use standard Python types for all the attributes.
- In our case we want to predict the Iris Species so will create a data model as class with four parameters which are the dimensions of the species.
- Now create an end point also known as route named "predict"
- Add a parameter of type data model we created which is "IrisSpecies".
- Now we can post data as json and it will be accepted in iris variable.
- Next, we will load the already saved model in a variable loaded_model.
- Now perform the prediction the same way we do in machine learning and return the results.
- now you can run the app and see the beautiful User Interface (UI) created by FastAPI which uses Swagger now known as openAPI as backend for designing the documentation and UI.
- Full code is given below you can simply copy and paste and it will work if you have followed the above steps properly.

```
from fastapi import FastAPIfrom pydantic import BaseModelimport pickleimport numpy as
npimport pandas as pdapp = FastAPI()class IrisSpecies(BaseModel):sepal_length:
floatsepal_width: floatpetal_length: floatpetal_width:
float@app.post('/predict')async def predict_species(iris: IrisSpecies):data =
iris.dict()loaded_model = pickle.load(open('LRClassifier.pkl', 'rb'))data_in =
[[data['sepal_length'], data['sepal_width'], data['petal_length'],
data['petal_width']]]prediction = loaded_model.predict(data_in)probability =
loaded_model.predict_proba(data_in).max()return {'prediction':
prediction[0],'probability': probability}
```

VS-Code snippet of the API creation:

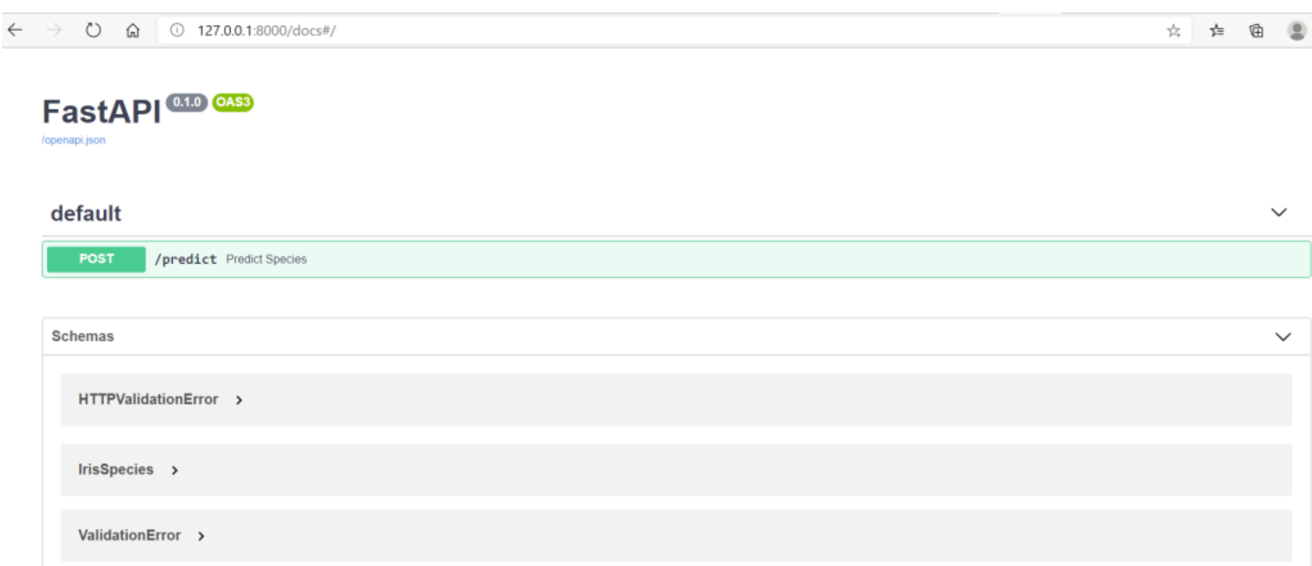


Executing the APP:

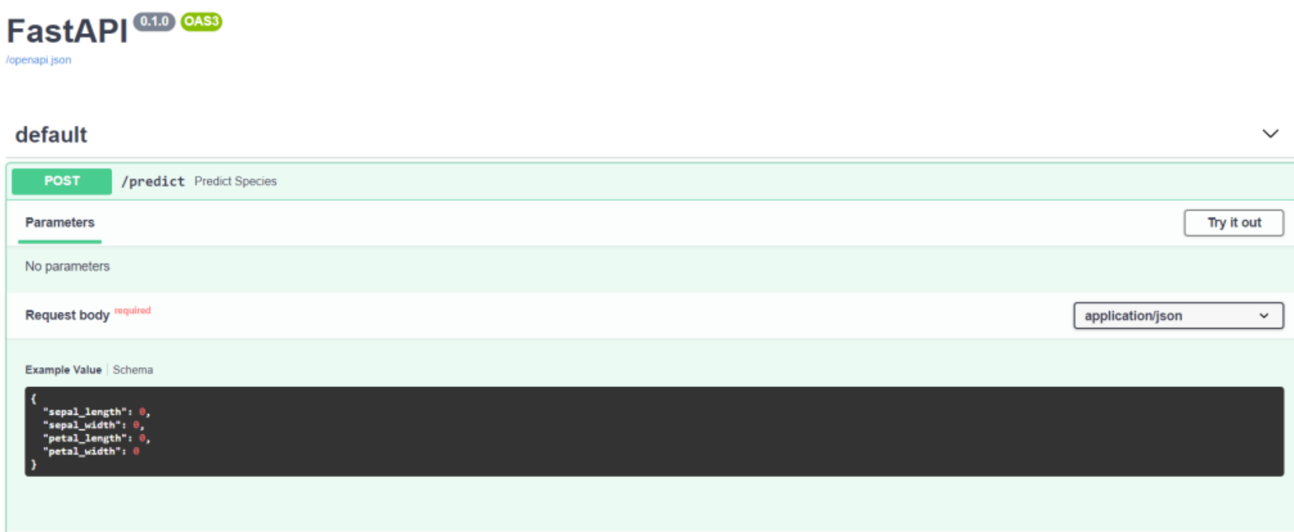


Now if you can see the nice UI created by typing the url: 127.0.0.0:8000/docs

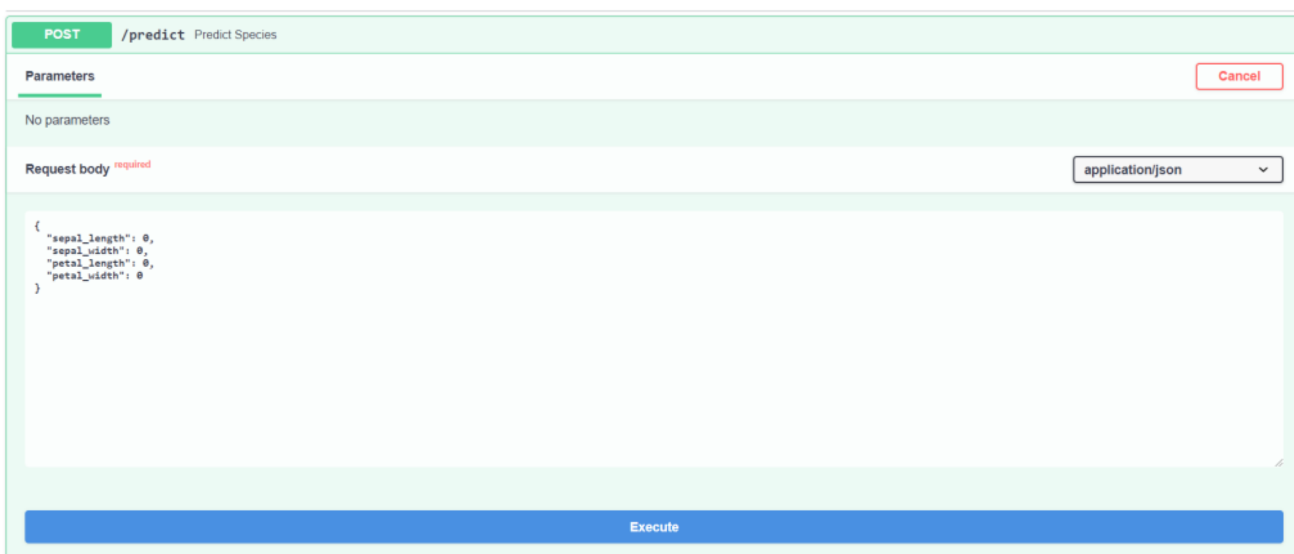
Below you see the API end point is created as POST request.



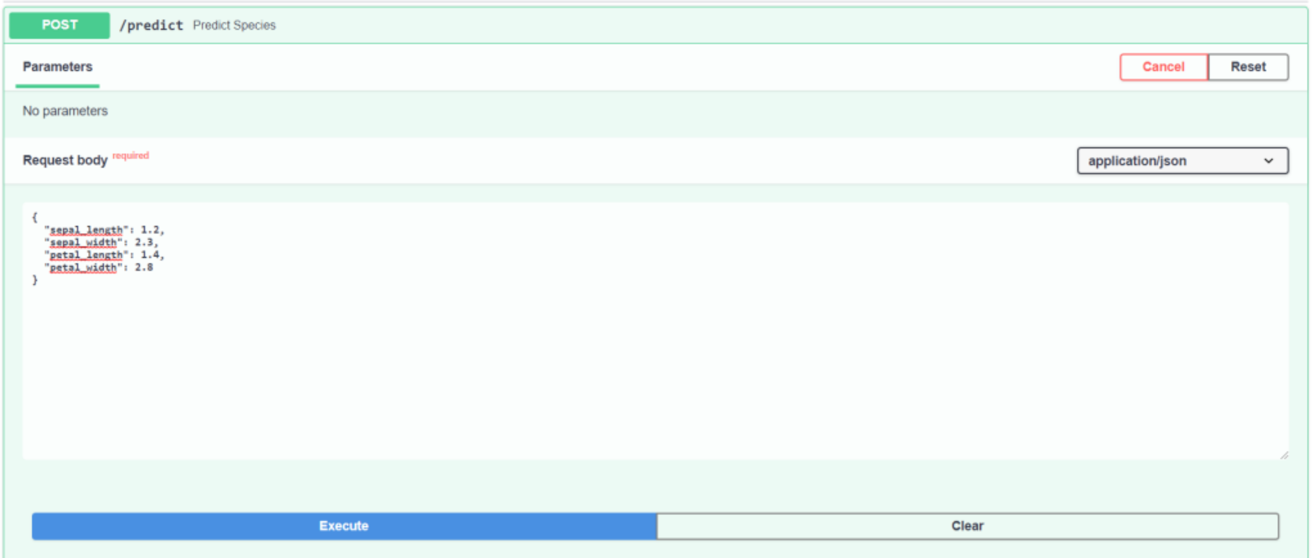
Click on the end point and it will expand as below.



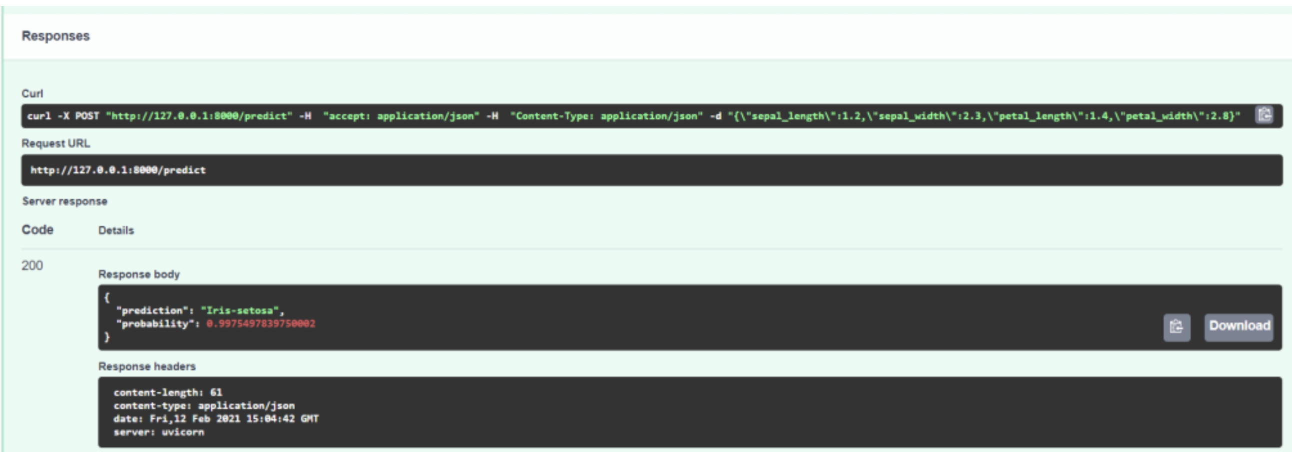
Now click on Try it out and paste the dimensions to get the prediction.



I pasted some dummy dimensions and clicked on execute.



Now you see that it has predicted it as Iris-setosa with 99% accuracy.

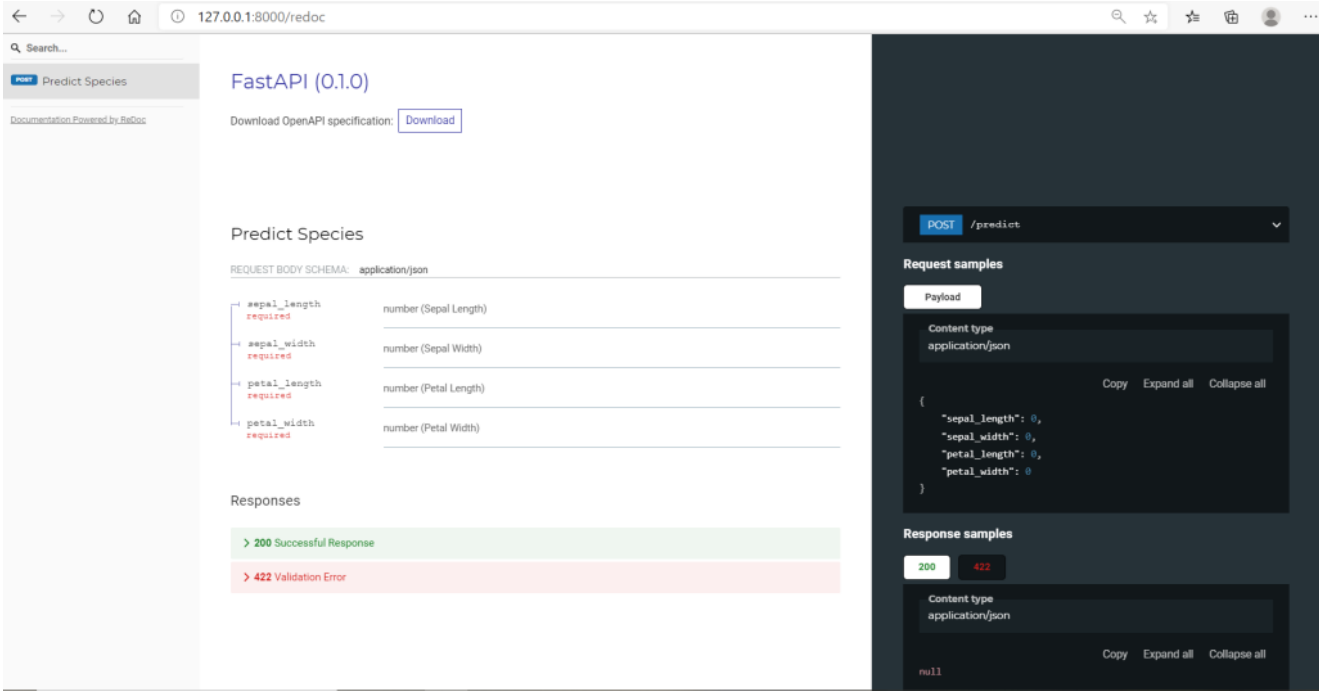


You can directly call this api from anywhere as below:

```
import requestsnew_measurement = {"sepal_length": 1.2,"sepal_width":
2.3,"petal_length": 1.4,"petal_width": 2.8}response = requests.post('',
json=new_measurement)print(response.content)>>> b'{"prediction":"Iris-
setosa","probability":0.99}'
```

So this was all about the API creation using the FastAPI.

FastAPI also provides nice documentation which gets created automatically. just type in the browser 127.0.0.0:8000/redoc



That's it for this article. Hope you enjoyed reading. Share your thoughts about your experience with FastAPI. Also, you can ask if you get any questions during implementation using the comments.