ISSN: 2508-7894
KJAI website: http://acoms.kisti.re.kr/kjai
doi: http://dx.doi.org/10.24225/kjai.2023.11.4.29

A Case Study of Rapid AI Service Deployment - Iris Classification System

Yonghee LEE[1]

Received: October 24, 2023. Revised: November 08, 2023. Accepted: November 09, 2023.

Abstract

The flow from developing a machine learning model to deploying it in a production environment suffers challenges. Efficient and reliable deployment is critical for realizing the true value of machine learning models. Bridging this gap between development and publication has become a pivotal concern in the machine learning community. FastAPI, a modern and fast web framework for building APIs with Python, has gained substantial popularity for its speed, ease of use, and asynchronous capabilities. This paper focused on leveraging FastAPI for deploying machine learning models, addressing the potentials associated with integration, scalability, and performance in a production setting. In this work, we explored the seamless integration of machine learning models into FastAPI applications, enabling real-time predictions and showing a possibility of scaling up for a more diverse range of use cases. We discussed the intricacies of integrating popular machine learning frameworks with FastAPI, ensuring smooth interactions between data processing, model inference, and API responses. This study focused on elucidating the integration of machine learning models into production environments using FastAPI, exploring its capabilities, features, and best practices. We delved into the potential of FastAPI in providing a robust and efficient solution for deploying machine learning systems, handling real-time predictions, managing input/output data, and ensuring optimal performance and reliability.

Keywords: Machine Learning, Classification, Web Service, FastAPI

Major Classifications: Artificial Intelligence, Service Deployment

1. Introduction

The field of machine learning has demonstrated significant advancements, empowering organizations to derive fruitful insights and make informed decisions. As high quality data is more and more available, data-based machine learning systems can become more and more intelligent and be applied in real life. Conventionally, there have been rich community supports for machine learning or artificial intelligent systems from Python programming language society. Without fair use of publicly available machine learning libraries, development of smart machine learning systems appears to be unproductive and meaningless. This fact forced the machine learning developers confine themselves to Python language and its libraries. However, developing competitive machine learning systems is one thing but it is quite another to deploy developed systems (Jain & Kumar, 2023). The flow from developing a machine learning model to deploying it in a production environment is often fraught with challenges.

* This paper was supported by Shingu College.
1 First and Corresponding Author. Professor, Department of AI Software, Shingu College, South Korea. Email: leo@shingu.ac.kr

Efficient and reliable deployment is critical for realizing the true value of machine learning models. Bridging this gap between development and production has become a pivotal concern in the machine learning community (Song et al., 2023).

FastAPI, a modern and fast web framework for building APIs with Python, has gained substantial supporters due to its speed, ease of use, and asynchronous capabilities (Lathkar, 2023). This paper focuses on leveraging FastAPI for deploying machine learning models, addressing the potentials associated with integration, scalability, and performance in a production setting.

In this work, we explore the seamless integration of machine learning models into FastAPI applications, enabling real-time predictions and showing possibilities of scaling up for a more diverse range of use cases. We discuss the intricacies of integrating popular machine learning frameworks with FastAPI, ensuring smooth interactions between data processing, model inference, and API responses.

Throughout this paper, we present a comprehensive guide, illustrating step-by-step processes and providing code examples to demonstrate the deployment of machine learning models using FastAPI. Our goal is to equip machine learning learners, developers, and data engineers with the knowledge and tools necessary to deploy models effectively, providing an overall outline of a development chain for balanced perspective for development and deployment.

The aim of this paper is to serve as a valuable demonstrative case study for researchers, data scientists, machine learning engineers, and developers who seek to enhance their understanding of how FastAPI can simplify and expedite the deployment of machine learning models in a production environment, ultimately accelerating the practical adoption of machine learning across diverse industries.

At first, an iris classification problem will be discussed briefly as a representative example of machine learning system development. This case study was chosen mainly because of the simplicity of problem, diversity of suggested machine learning algorithms and high accuracy of prediction. After showing the completeness of the iris flowers classification, decision criteria for a proper deploying framework will be delved into. FastAPI has been selected to be a deploy system and necessary codes are presented (Voron, 2022). Finally, front-end part will be elaborated afterwards.

2. Iris Classification Problem

2.1. Problem Description

The iris classification problem is a classical problem in the machine learning area let alone data science and statistics field. Figure 1 shows three different species of iris flowers, they are iris Setosa, Virginica and Versicolor respectively. Based upon the given data of four measurements of separate parts, a machine learning model could figure out the correct iris species.

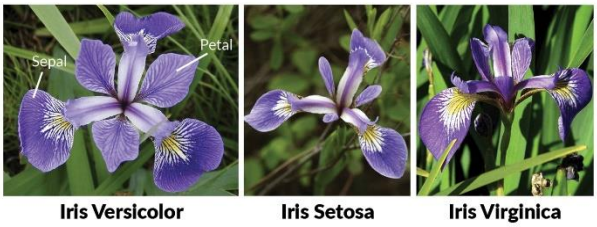

Figure 1 Three species of iris flower

Corresponding Python code begins with following code, in which measurement data are read and separated for input and label. As the original data are well preprocessed, we can go directly to training issues.

```
iris_df = pd.read_csv('iris.csv')

X = iris_df.drop('species', axis=1)
y = iris_df['species']
```

2.2. Machine Learning Model

For this problem, many solutions haven been proposed and verified. In this paper, the author chose three models for illustrative and educational purposes. Those models are Random Forest classifier, K neighbors classifier and

```
rfc = RandomForestClassifier()
model_rfc = rfc.fit(X, y)

kn = KNeighborsClassifier()
model_kn = kn.fit(X, y)\

model_tf =
    tf.keras.models.Sequential([ Dense(64,
    activation='relu', input_shape=(4,)),
    Dense(32, activation='relu'), Dense(3,
    activation='softmax'), ])
model_tf.compile(optimizer='adam',
   loss='categorical_crossentropy',
   metrics=['acc'])
model_tf.fit(train_data, validation_data=
   (valid_data), epochs=20)
```

Google's TensorFlow (Kang et al., 2022). Typical applications of the above-mentioned models are quite similar, as shown in the next code.

In typical machine learning systems, training sessions require a great amount of computing power, whereas a trained machine predicts its output based on new input data very quickly. For this reason, the training session and prediction phase should be separated, and that requirement is satisfied by saving the trained model and retrieving it for later prediction. Among several functions for the trained model saving, `joblib` library is incorporated in this case.

```
import joblib

joblib.dump(model_rfc, './model_rfc.pkl')
joblib.dump(model_kn, './model_knn.pkl')
joblib.dump(model_tf, './model_tf.pkl')

# ---
loaded_model = joblib.load('./model_knn.pkl')
…
```

Once the models are trained, saved, and reloaded, we need to check their functionality by providing sample input data. In this code, an imaginary data of [1, 4.2, 1.4, 7] was presented and Random Forest Classifier model predicts the species as 'iris setosa' and probabilities are 0.51, 0.22 and 0.26 to be Setosa, Versicolor and Virginica respectively. So far, the iris classification problem is simple, straightforward, and easy to understand. One can easily grasp the overall process of machine learning construction processes which include data gathering, preprocessing, model design, and training by reviewing this case.

```
import numpy as np

loaded_model = joblib.load('./model_rfc.pkl')
X_new = np.array([[1, 4.2, 1.4, 7]])
prediction = loaded_model.predict(X_new)
print((dataset['target_names'][prediction]))
probability = loaded_model.predict_proba(X_new)
print(probability)
```

3. Service Deployments

3.1. REST Framework

Once developing machine learning systems have performed the training session, they can be served in a single stand-alone software which can be downloaded and executed on the client's local machine. Additionally, more recent way of software deployment is on the web service. REST is one of the popular web architectural models to support the web service. It can be described as an interface between client and servers over the web with HTTP. Service suppliers provide specific services on the pre-determined APIs (Application Program Interface) whereas service users access those APIs by HTTP. As there are few dependencies between user sides (front-end) and service providers (back-end), a complicated interoperability problem is highly unlikely. The main advantages of REST over the web service architecture, such as SOAP can be summarized as follows (Kumari & Rath, 2015):

- Ease of use
- high interoperability
- flexibility
- scalability
- security

3.2. Web framework architecture

Table 1: Comparison of web framework

Features	Flask	Django	FastAPI
Open Source	Yes	Yes	Yes
Popular	Yes	Yes	Yes
Launched in	2010	2005	2018
Restful API	Flask-Marshmallow Flask-JWT Flask_RESTful	PythonAPI	RestAPI
Web application based on	WSGI	WSGI	AGI
Support for dynamic HTML	NO	Yes	Yes
Performance	Fast	Slow	Faster
Companies using framework	Reddit Netflix Mailgun	Instagram Udemy Pinterest	Uber Yogiyo
data validation	no	no	built-in data validation
convenient for project	smaller, medium	Large, complicated	small, simple
community support	rich	rich	smaller

Currently, most web application frameworks are based on Java (e.g., Spring Boot) or JavaScript (e.g., Node.js). However, as most artificial intelligence systems are built in Python language as described in the introduction section, developing a web framework with the same Python language would be highly preferred (Dani et al, 2022). Table

1 compares various web application frameworks. Except for a relatively small community, FastAPI appears to be simpler and faster than the other Python based web frameworks (Bansal & Ouda, 2022). Based on this observation, FastAPI is selected to be the framework for service deployment.

3.3. Backend Architecture

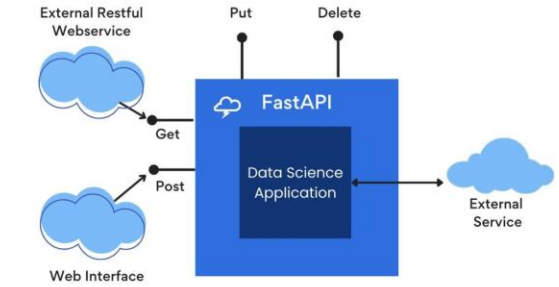

Figure 2: Diagram of bad-end system with FastAPI

FastAPI is described as a modern and high-performance web framework for developing APIs with Python. It offers high performance on par with NodeJS and Go (Turing, 2023). It is being used by top Internet service companies. To get started with FastAPI, Uvicorn is required. Uvicorn is an Asynchronous Server Gateway Interface (ASGI) server used for production (Song & Kook, 2022). FastAPI has the advantage of handling requests asynchronously. As shown in Figure 2, back-end server is providing REST services over internet with various points.

3.4. Backend Development with FastAPI

The next code is the simplest FastAPI code. Once this code starts running, the functionality of the server can be verified by accessing `http://localhost:8000/` with a web browser and receiving "`Hello":"World`" message.

```
from fastapi
import FastAPI app = FastAPI()

@app.get("/")
def read_root():
  return {"Hello": "World"}

if __name__ == '__main__':
   uvicorn.run(app, host='127.0.0.1',
      port=8000)
```

Before going further with FastAPI, the iris machine learning module should be rearranged to separate the initializing and training period from the predicting phase as follows. It is refactored mainly to let the training module be run at most one-time during the overall process.

```
class IrisVarieties(BaseModel):
  sepal_length: float
  sepal_width: float
  petal_length: float
  petal_width: float
class IrisML:
  def __init__(self):
    self.model_fname_ = 'iris_model.pkl'
    try:
      self.model = joblib.load(self.model_fname_)
      print("load ok")
    except Exception as _:
      print("training begins")
      self.model = self.train_model()
      joblib.dump(self.model, self.model_fname_)

  def train_model(self):
    iris_df = pd.read_csv('iris.csv')
    X = iris_df.drop('species', axis=1)
    y = iris_df['species']
    model = KNeighborsClassifier(n_neighbors=5)
    model = model.fit(X, y)
    return model

  def classify_species(self, sepal_length,
       sepal_width, petal_length, petal_width):
     data_in = [[sepal_length, sepal_width,
        petal_length, petal_width]]
     prediction = self.model.predict(data_in)
     probability =
        self.model.predict_proba(data_in).max()
     return prediction[0], probability
```

Class `IrisML` is designed to support overall server operation. `IrisVarieties` class is derived for the secure communication of user input data between the front-end and back-end system. FastAPI server programming should accommodate the following code so that the species prediction module can be linked by REST API pointing "`/predict`"

The method of REST architecture should be "POST" instead of "GET" for reliable and secure transmission of data between the client and the server side. Due to this

```
@app.post("/predict")
async def predict_species(iris: IrisSpecies):
  pred, prob = model.classify_species(
      iris.sepal_length, iris.sepal_width,
      iris.petal_length, iris.petal_width )
  return {"prediction": pred, "probability":
     prob.tolist()}
```

"POST" method, other API test solutions are required. A REST API test module such as Talend API Tester is utilized to approach "`/predict`" point with four input data as shown in Figure.3.

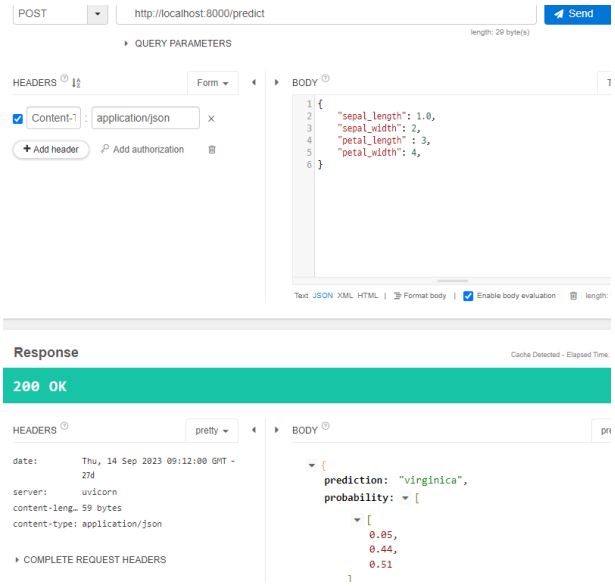

Figure 3: Output of POST operation by REST API test tool

3.5 Inner Process of FastAPI

The inner process of FastAPI regarding RESTful service involves the following steps:

1. Define the route and method for the API endpoint using decorators, such as `@app.post("/predict")`.
2. Create a function that handles the request and returns the response. This function can have input parameters that correspond to the data sent in the request.
3. Inside the function, perform the necessary operations, such as calling a machine learning model for prediction or processing the input data.
4. Return the response in JSON format, including the predicted result and any other relevant information.
5. The client sends a request to the specified API endpoint, such as `http://localhost:8000/predict`, with the required data in the request body.
6. The FastAPI server receives the request, validates the input data based on the defined data model (if any), and calls the corresponding function.
7. The function processes the request, performs the necessary computations, and generates the response.
8. The server sends the response back to the client, which can then access the predicted result and any other returned data.

This process enables the development and deployment of RESTful services using FastAPI, allowing clients to interact with the server through HTTP requests and obtain the desired results.

3.6. Front-end Development

After normal operations of web application with FastAPI are implemented, a development step of front-end is straightforward. Four separate input fields are required to accept measurements of the four parts of iris flower. By clicking the "submit" button, the corresponding JavaScript function assembles four numbers, converts those into JSON format and transfers to a pre-developed server (Snodgrass& Milkov, 2020). An essential part of the JavaScript module for REST API call with jQuery is shown as follows:

```
function Send(){
var data = {
   'sepal_length': sl.value,
  'sepal_width': sw.value,
  'petal_length': pl.value,
  'petal_width': pw.value,
  }

$.ajax({ type: "POST", url:
    'http://localhost:8000/predict',
      headers:{ "Accept" : "application/json",
    "Content-Type": "application/json", },
      data: JSON.stringify(data),
}).done(function(response)
      request done successfully
      show it to the result part of user iterface
}).fail()
  ➔  Proper error processing here
```

Figure 4. displays developed user interface by HTML. With given input, the trained machine learning model predicted its species as Virginica with 0.67 probability whereas probability of being Versicolor as 37% and its prediction results are displayed in number format and also in a graphical way. This system can easily be extended to include more machine learning models so that users can compare the performances of several models available in the market.

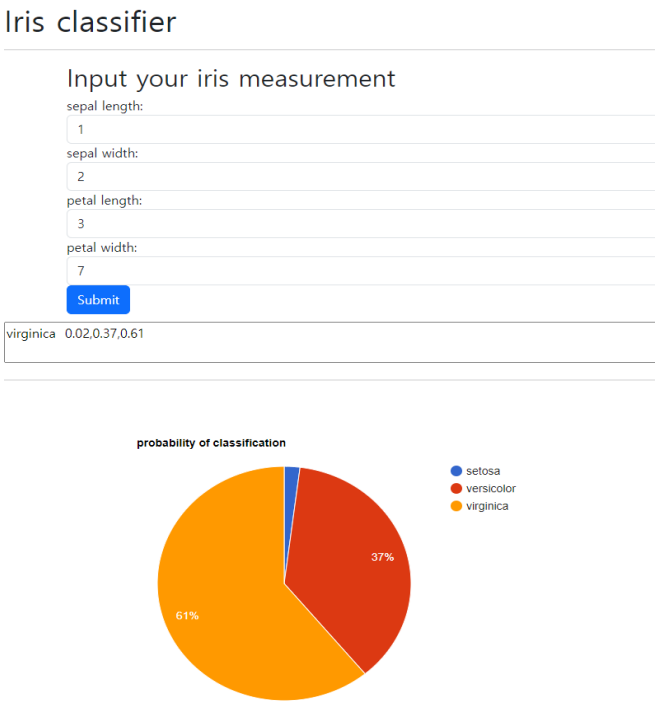

Figure 4: Design of a user interface on the web for approaching machine learning server

4. Summary

In this case study, from developing a machine learning model to publishing its service on the web, every step has been covered. To reduce unnecessary complication of each step, every phase had been simplified as much as possible. We explored the seamless integration of machine learning models into FastAPI applications, enabling efficient predictions and showing a possibility of scaling up for more diverse range of use cases. A comprehensive guide was presented, covering from the beginning of an artificial intelligence system to rapid deployment of developed systems. This case demonstrated the big picture of machine learning systems and the possibilities of swift deployment of commercial AI services with simple tools such as HTML, JavaScript, and Python. We focused on elucidating the integration of machine learning models into production environments using FastAPI, exploring its capabilities, features, and best practices. We delved into the potential of FastAPI in providing a robust and efficient solution for deploying machine learning systems.

References

Bansal, P. and Ouda, A., (2022). Study on Integration of FastAPI and Machine Learning for Continuous Authentication of Behavioral Biometrics, *2022 International Symposium on Networks, Computers and Communications (ISNCC)*, Shenzhen, China, 2022, pp. 1-6, doi: 10.1109/ISNCC55209.2022.9851790.

Dani, H., Bhople, P., Waghmare, H., Munginwar, K., Patil, A., (2022). Review on Frameworks Used for Deployment of Machine Learning Model, *International Journal for Research in Applied Science & Engineering Technology (IJRASET):* Vol. 10 Issue II Feb 2022, doi: https://doi.org/10.22214/ijraset.2022.40222

Jain, S. and Kumar, S., (2023). Cost Effective Generic Machine Learning Operation: A Case Study, 2023 *International Conference on Data Science and Network Security (ICDSNS)*, Tiptur, India, 2023, pp. 1-6, doi: 10.1109/ICDSNS58469.2023.10245408.

Kang, S., Choi, J., Kang, M., (2022). Classification Model and Crime Occurrence City Forecasting Based on Random Forest Algorithm, *Korean Journal of Artificial Intelligence, 10*(1), (2022), 21-25. doi: http://dx.doi.org/10.24225/kjai.2022.10.1.21

Kumari, S. and Rath, S. K., (2015). "Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration," *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Kochi, India, 2015, pp. 1656-1660, doi: 10.1109/ICACCI.2015.7275851.

Lathkar, M., (2023). *Getting Started with FastAPI. In: High-Performance Web Apps with FastAPI*. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-9178-8_2

Snodgrass, J. E., Milkov, A., (2020). Web-based machine learning tool that determines the origin of natural gases, *Computers & Geosciences*, *145*, December 2020, doi: https://doi.org/10.1016/j.cageo.2020.104595. (https://www.sciencedirect.com/science/article/pii/S0098300420305793)

Song, J., Cai, J., Li, R., and Li, Y., (2023). "Design and Implementation of Scientific Research Achievement Transformation System," *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, Orlando, FL, USA, 2023, pp. 407-412, doi: 10.1109/SERA57763.2023.10197696.

Song J, Kook J., (2022) Mapping Server Collaboration Architecture Design with OpenVSLAM for Mobile Devices. *Applied Sciences*. 2022; *12*(7):3653. https://doi.org/10.3390/app12073653

Turing, Python FastAPI vs Flask: A Detailed Comparison, Retrieved from https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison

Voron, F., (2022). Building Data Science Applications with FastAPI: Develop, manage, and deploy efficient machine learning applications with Python, *Packt Publishing*, 2022