Unit 5
ML App Development

What is a web Framework?

- Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications
- The developer need not bother about low-level details such as protocols, thread management etc.

What is Flask?

- Flask is a web micro-framework developed by Armin Ronacher.
- Flask is written in Python programming language.
- Flask is made for Python applications.
- Flask helps in implementing a machine learning application in Python that can be easily plugged, extended and deployed as a web application.
- Flask is based on two key components: WSGI (Webserver Gateway Interface) toolkit and Jinja2 template engine.
- WSGI is a specification for web applications and Jinja2 renders web pages.
- Linkedin, Pinterest use the Flask framework.

Why Flask?

- Easy to use.
- Built in development server and debugger.
- Integrated unit testing support.
- RESTful request dispatching.
- Extensively documented.

Werkzeug

- It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.
- Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible.

Flask – Hello world!!! - in Python

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
    def hello():
        return "Hello World!"

if __name__ == "__main__":
    app.run()
```

```
$ python hello.py
* Running on http://localhost:5000/
```

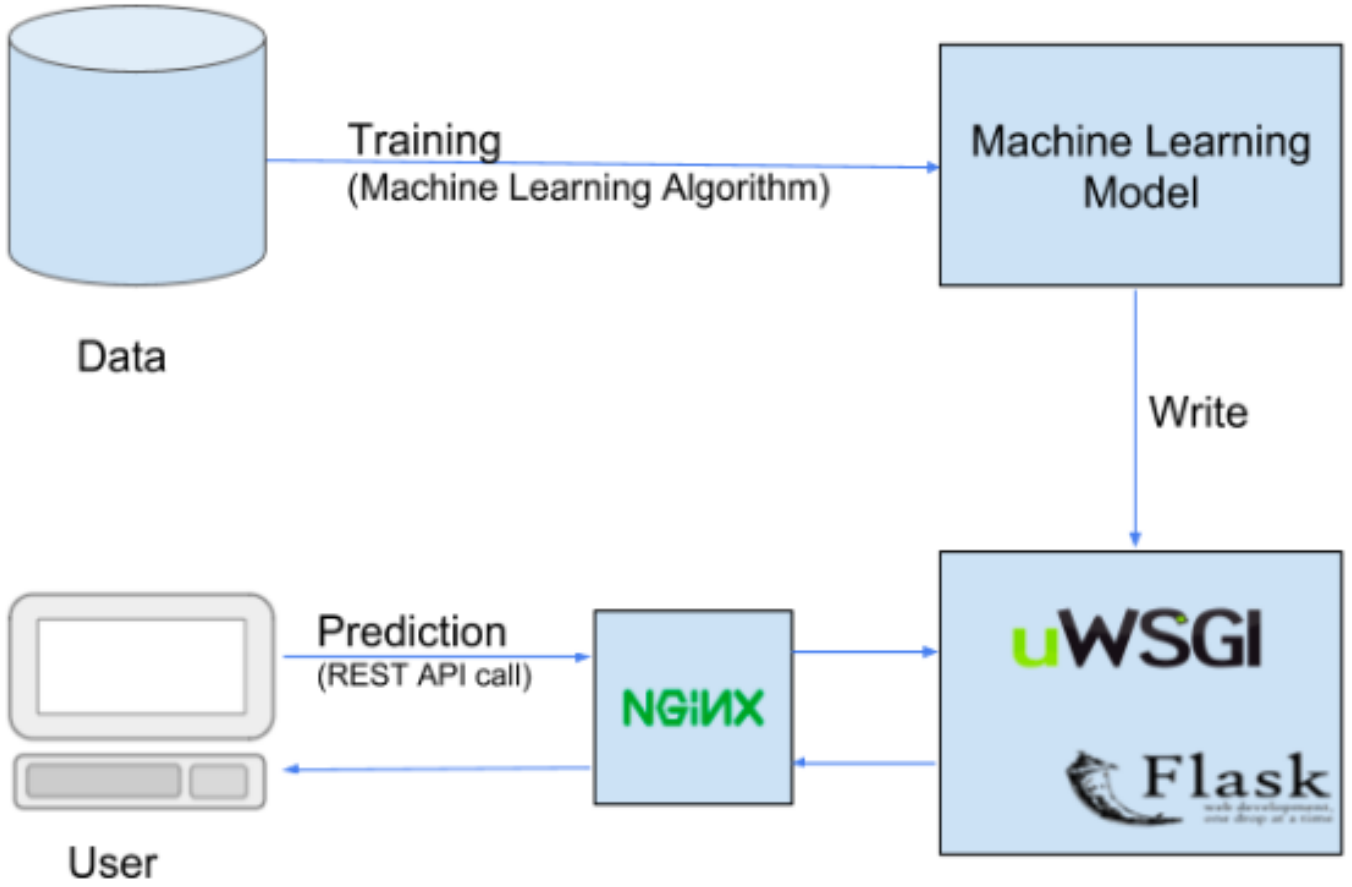
Flask -- Hello World!!! – in Jupyter

```
from werkzeug.wrappers import Request, Response
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World"

if __name__=="__main__":
    from werkzeug.serving import run_simple
    run_simple('localhost', 9000, app)
```

- The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function.
  - @app.route(rule, options)
- Finally the **run()** method of Flask class runs the application on the local development server.
  - app.run(host, port, debug, options)
- Always better to enable the debug property as True before running the app
  - app.run(debug = True)

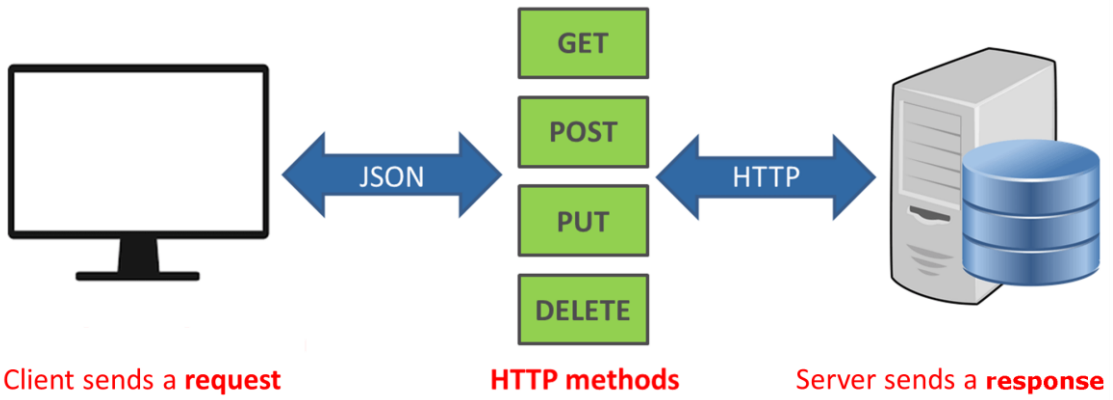
use run_simple('localhost',port,app) in jupyter

ML Model Deployment



RESTFul API

- A **RESTful API** is an application program interface (**API**) that uses HTTP requests to GET, PUT, POST and DELETE data
- **REST** is more robust Simple Object Access Protocol (SOAP) technology
- It leverages less bandwidth, making it more suitable for internet usage



Steps to Deploy ML model in Web Application

- Step 1: Build your machine learning model
- Step2: install flask using pip3 install flask
- Step 3: Create a folder structure
  - app.py: This Python file will contain the Flask specific code – which is part of the backend which is going to respond for the client requests
  - Model.pkl: This file is the dumped model which has the ability to predict
  - Templates folder: This folder will contain the HTML files. These HTML files will be rendered on the web browser.
  - You can also store js, css and other python files if they are required.

ML- Iris Application

- Client
  - Templates folder – 2 files – input html file and output html file
- Server
  - 2 files --- Python and Model File
- Prerequisite
  - Server file should be running before the client make the request (eg:app.py)
  - Routes should be properly set

Iris Client

- Input HTML file

```
<html>
   <body>
      <form action = "/predict" method="post">
        <p>Petal Length <input type = "text" name = "petal_length" /></p>
        <p>Petal Width <input type = "text" name = "petal_width" /></p>
        <p>Sepal Length<input type = "text" name = "sepal_length" /></p>
        <p>Sepal Width <input type = "text" name = "sepal_width" /></p>
        <input type="submit" name="submit" />
      </form>
   </body>
</html>
```

Iris Client

- Result HTML file

```
<html>
 <head>
 <title>Iris Prediction</title>
 </head>
 <body>
 <h3>Flower Class:</h3>
 <div>The Predicted Class is <strong>{{ pred }}</strong>)</div>
 </body>
</html>
```

Iris Server

- App.py
  - Import libraries – Flask, joblib, pandas, np
  - Create the app object
    - App=Flask(__name__)
  - Bring the model to the memory
  - Route the app as a decorator
  - Def a function which can take receives input from the client request
  - Convert the html form values as acceptable by the model
  - Predict using the model
  - Store the result in a variable
  - Pass the result to the client