ML Model Management using Python Django Web Framework

Mr. P. R. Ranjith
Senior Software Engineer, SAP Ariba, Bangalore, India
ranjith.pavanje.raja.rao@sap.com

Why Machine learning:

Machine learning is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

For Any Industry to be a market leader in the current competitive world, it needs to quickly and automatically produce models that can analyze bigger, more complex data and deliver faster, more accurate results – even on a very large scale. And by building precise models, an organization has a better chance of identifying profitable opportunities – or avoiding unknown risks.

Python for Machine Learning:

Python is widely considered as the preferred language for teaching and learning ML. Few Reasons are as below:

1. **Simple to Learn:**
   You can quickly pick up on the language and start using it for AI development rather than wasting too much time/effort in learning the language.

2. **Great Library Support:**
   A great choice of libraries is one of the main reasons Python is the most popular programming language used for AI. ML requires continuous data processing, and Python's libraries let you access, handle and transform data. The Most popular libraries include Scikit-learn, Pandas, Keras, Tensorflow, Matplotlib, NLTK.

3. **Platform independent:**
   Python is not only comfortable to use and easy to learn but also very versatile. It can run on any platform including Windows, MacOS, Linux, Unix and many more.

4. **Readability:**
   Python is very easy to read so every Python developer can understand the code of their peers and update them accordingly.

5. **Open Source:**
   Python is open source and hence anyone can pick it and start using.

6. **Strong Community Support:**
   Since Python is opensource, a lot of Python documentation is available online as well as in Python communities and forums, where programmers and machine learning developers discuss errors, solve problems, and help each other out

All these factors make Python the preferred language for ML.

Challenges of Hosting an ML based Solution:

We have the data, A good Python based ML algorithm which does the inference/Prediction with a very good confidence. Great!

Now comes the next question. How do we host this solution for prediction? we should be able to send a request (typically a web request) and get a prediction/recommendation as a response.

A web application or a Rest framework is usually written in java/dot net or any other web technologies and the major challenge is the cross-language communication between the web technologies and the python.

Let us consider a prediction application written in java-based web technology. Every time a request comes to the web server for Training/Prediction, the java thread needs to communicate with the Python ML algorithm - which is a standalone process, for the following:

- Provide Training data
- Wait for the python algorithm to finish to get the result
- Preloading the model before the prediction request comes
- Once trained, Manage the generated Models (storing, sending, uploading etc.)
- Retrieve the prediction outcome and any other response parameters returned by python.

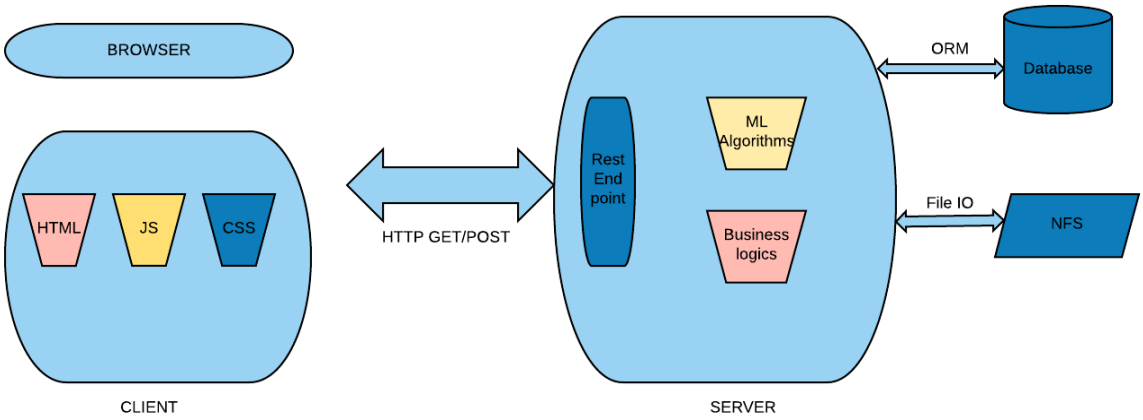
The overhead involved in managing the above tasks can be quite a lot and hence we might need a better technology to manage these barriers.

Python Django To the Rescue!

What if we had a web-based technology that is completely written in python. Since there would be no cross-language communication, the above overheads could be managed more efficiently. That exactly is what Django does.

Django Framework:

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel.



As shown in the Diagram above, Django is based on MVC architecture. Below are the main components:

1. **View:**

   It is the Front end of the framework which sits on the client browser. Django supports HTML, CSS, Javascript as the main UI technologies.

   We could also integrate Angular framework into Django with few simple changes.

2. **Model:**

   The Model is responsible for managing the Data of the application. Django is capable of connecting to various databases through its ORM, thus managing in storage and retrieval of the data.

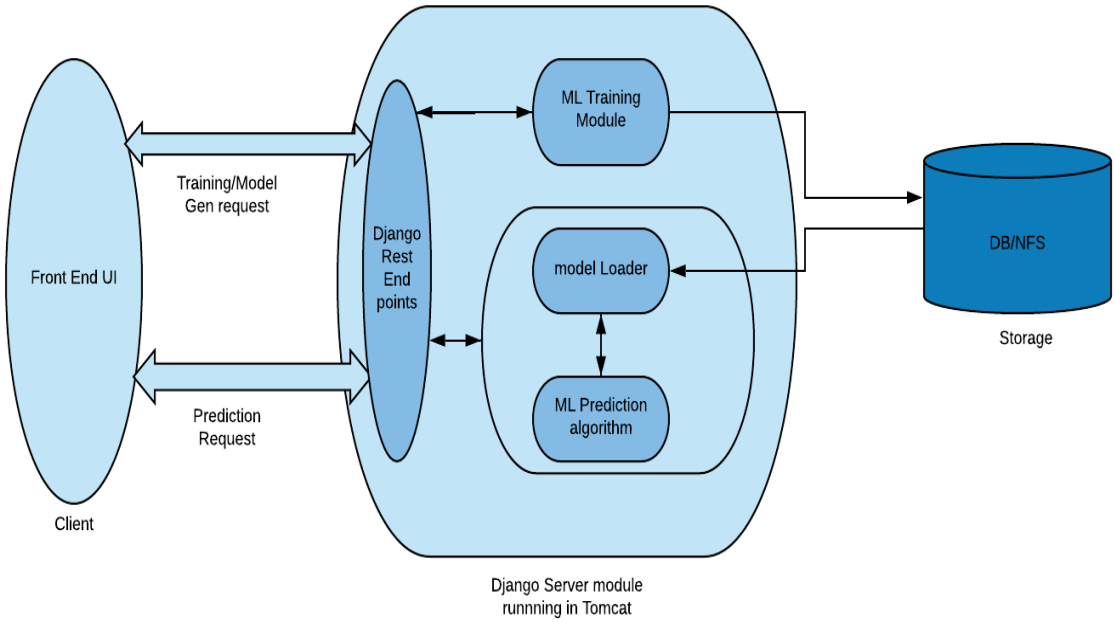
   In a microservice Environment with NFS, Django can also mount and store/retrieve data from NFS.

3. **Controller:**

   Controller is responsible for all the business logic and transformation/manipulation of the data. In our ML use case, this is where all the ML based algorithm should sit.

Django from a ML Use Case:

Having known about Django, let's see how we can leverage it for an ML use case.



As shown above, the two main Aspects of ML use case would be:

Training/Model Generation:

- The Request for training (single or multi-tenant) would be intercepted by a Rest endpoint controller and handed to an ML based training module algorithm.
- The algorithm generates a model (a protobuf or a pickle file) as output.
- This Model is stored in either Database, NFS, or sent via rest call to any storage server.

Inference/Prediction:

- The Request for Prediction is again intercepted by a Rest handler and is sent to Prediction algorithm.
- The module shall first load the relevant model from the Storage.
- Once the Model is loaded to memory, the Prediction runs, and the outcome is sent back in the response

Model Management in Django:

The backbone of any ML solution is managing the created ML Models. The success of any realtime prediction engine depends a lot on how efficiently we manage the models (in a single or multi-tenant environment). Model management involves, storing, retrieving, updating, deleting models from a pool of models available.

Storing the models:

Here are some of the ways we can achieve this using Django.

Store in a Database:

Django can Directly interact with an underlying Database with minimal configurations using its ORM.
Hence DB is a good choice for storing and retrieving the models.
We can store the model in a table as a payload with some metadata information (model configurations).
In a multi-tenant environment, it could be stored with tenant information as well. While retrieving, we could retrieve the relevant model from the tenantId passed.
We could also have a list of (pool of) models for a tenant and choose the best model based on the criteria selected.

Store in a Disk/NFS:

We could also store the models in the server File system and access through fileIO.
If the Django is running on a microservice infrastructure with NFS mounted and auto mirroring, then there is hardly any risk of disk crash/ data loss.

In a multi-tenant environment, one strategy of storing the models would be by creating a folder for each tenant with the folder name as the tenantId. This way we could go and fetch the exact model for the tenantId passed.

Store in a Storage Server:

The Model could also be stored/Retrieved from a dedicated Storage Server. Django can make an API call through its Django Rest Framework and store/retrieve the relevant model.

Loading the Model:

The Models could be either preloaded to memory when the Django server starts or Loaded to memory as and when a request for prediction comes.
Preloading the model to memory is a good idea to use when the models are bulky and takes some time to compute/load. Also, if there are only few models to be loaded, then preloading would greatly help in reducing the prediction time (the overhead of loading the model for each request would not be there).
When a Django server comes up, it reads its **urls.py** for all the routing patterns and initializations. If the model loading call is plugged in here, it gets executed during the server start-up.

If working in a multi-tenant environment with a pool of models for each tenant (and the models are light), the models could be loaded when a request comes for prediction. Django also provides Caching mechanism which could be leveraged for loading the models.

A Django Admin page:

In addition to all the above, we could also build a simple admin UI using Django Template or by integrating Angular with Django. The UI could list down all the models for a tenant.
We could also provide few actions like deleting, uploading, downloading, activating models etc.
These actions are not hard to implement since the Django UI can easily make an Api call to Django server for the details.

Django Alternatives:

As Python is gaining a lot of popularity with growing ML use cases, lots of python-based web frameworks are being developed and are available to choose from. The major once include:

- Tornado
- Flask
- Pyramid
- TurboGears

Each one of them have their own advantages and disadvantages. Most of the framework listed above are Microframeworks. Hence, they may not be suitable for bigger full-stack web development.

Django stands out among these frameworks for the rich features it has to offer. Following are some of them:

- Django is a Full-stack web framework
- It has the best community support among the lot with extensive documentation and examples. Hence it is the most preferred language for bigger applications.
- It follows the standard MVC pattern.
- one managing script (“manage.py”) that can be used for performing most of the framework specific actions
- custom object-relational mapping (ORM) for communicating with the database
- a large number of external modules, e.g. Django REST Framework, Django CMS, Django Channels (websockets).

When not to Use Django:

Even though there are lots of advantages of using Django as the preferred web technology for ML usecases, there are also few cases where It is advisable not to use them.

- If your app is UI heavy. Although we could integrate Django with Angular by few tweaks, it is not quite straight forward. Also, we may not be able to use the full strength of Angular or any other powerful third-party UI frameworks
- Django needs the developer to have the complete knowledge of the framework. Not all the Data science members would be comfortable with that.

- Django is usually preferred for bigger applications. If your use case is very simple, then a lightweight web framework like Flask would be better than Django.
- Even though Django is opensource and has a good community support, it may not be as good as other web technologies(java/j2ee) which have very robust framework and community support.
- If you are running in a docker environment, there might be chances that the Django is not supported.

References:

Complete Django Documentation: https://docs.djangoproject.com

About the author

Ranjith holds around 12 years of experience building enterprise and cloud applications for various domains. He has worked in companies like cognizant, IBM. He is currently working as senior software engineer for Ariba Data Enrichment Team in SAP Ariba.

His interests are towards building java/j2ee based solutions in docker, machine learning concepts. He loves travelling, sports and movies.

Connect @ https://www.linkedin.com/in/ranjith-p-r-64499b9