UDACITY

Return to Classroom

Deploying a Machine Learning Model on Heroku with FastAPI

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on passing this project! 🎉

You have turned in a very good project. Your understanding of the concepts introduced in this module show in your work and you put in extra effort in many places. Very good work with your docstrings and code comments. 👍 This is great attitude and will serve you well as a part of any software engineering team. 👏👏

git and dvc

- GitHub action should run pytest and flake8 on push to main/master.
- PyTest must pass (by time the project is done there should be at least six tests) and flake8 must pass without errors.
- Include either a link to your GitHub repository or a screenshot of the CI passing called continuous_integration.png.

- **GitHub action should run pytest and flake8 on push to main/master.** ✓✓Your GitHub action runs both pytest and flake8. Well done 👍

- **PyTest must pass (by time the project is done there should be at least six tests) and flake8 must pass without errors.** ✓✓ Both Pytest and flake8 pass without errors in your Action.

- **Include either a link to your GitHub repository or a screenshot of the CI passing called continuous_integration.png.** ✓✓ Your GitHub repository shows your CI passing as required. 👍

Many times, teams want to enforce syntax and style checks before commits are made and the CI checks begin. To achieve this, they use the pre-commit tool which runs defined checks before any commit can be made locally. See Getting Started with Python Pre-commit Hooks and Introduction to Pre-Commit

- **Commit at least four files, original data, cleaned data, model, and OneHot encoder.**
- **Include a screenshot of "dvc dag" to show all tracked files and name it dvcdag.png.**

✓✓ Your screenshot of the `dvc dag` shows that you committed files including the cleaned data, model and encoder. Well done!

Remember that DVC can also be used to track experiments. See Experiment Management

Model building

- **The model should train on the provided data. The data should either be split to have a train-test split or use cross-validation on the entire dataset.**
- **Implement all stubbed functions in the starter code or create equivalents. At a minimum, there should be functions to:**
  - **train, save and load the model and any categorical encoders**
  - **model inference**
  - **determine the classification metrics.**
- **Write a script that takes in the data, processes it, trains the model, and saves it and the encoder. This script must use the functions you have written.**

✓✓ Your work here meets requirements. Consider wrapping the logic in your trainmodel.py into a function that you then call in an `if name == "__main__"` block

- **Write at least 3 unit tests. Unit testing ML can be hard due to the stochasticity -- at least test if any ML functions return the expected type.**

Your tests look great and you take advantage of Pytest Fixtures to create inputs that you pass to your

functions. Here are two articles to guide you here: Make your python tests efficient with pytest fixtures and End-To-End Tutorial For Pytest Fixtures With Examples

If you would like to reduce test times, you can consider mocking some functionalities. Mocking allows you to create replicas of objects for when we want to simply test that particular functionality is called/invoked but do not want any of its side effects (such as created files etc.). Alex Ronquillo's Understanding the Python Mock Object Library and Contravo's Effective Mocking of Unit Tests for Machine Learning provides a great introduction to this.

- **Write a function that computes performance on model slices. I.e. a function that computes the performance metrics when the value of a given feature is held fixed. E.g. for education, it would print out the model metrics for each slice of data that has a particular value for education. You should have one set of outputs for every single unique value in education.**
- **Complete the stubbed function or write a new one that for a given categorical variable computes the metrics when its value is held fixed.**
- **Write a script that runs this function (or include it as part of the training script) that iterates through the distinct values in one of the features and prints out the model metrics for each value.**
- **Output the printout to a file named slice_output.txt.**

✓✓ Your function compute the performance of your model on slices of the data as required. Well done! 👏👏 You also included this calculation as part of your training logic so that this slice performance is automatically calculated when a new model is trained.

Sometimes, we want to monitor these kinds of performance in production also. Platforms like Azure ML (Collect data from models in production) and AWS (Amazon SageMaker Model Monitor) allow us monitor our systems and make retraining/promotion decisions quicker.

- **The model card should address every section of the template.**
- **The model card should be written in complete sentences and include metrics on model performance. Please include both the metrics used and your model's performance on those metrics.**

✓✓ You addressed every section of the template and put great thought into your answers. 👍 Consider using markdown to highlight important keywords. This will draw attention to important aspects of your model card.

HuggingFace Transformers contain some of the best examples of Model Cards. Here's the model card for the BERT Base Uncased model. . It may give you great ideas for improving your work here.

API Creation

- **The API must implement GET and POST. GET must be on the root domain and give a greeting**

and POST on a different path that does model inference.
- Use Python type hints such that FastAPI creates the automatic documentation.
- Use a Pydantic model to ingest the body of the POST. This should implement an example (hint: Pydantic/FastAPI provides multiple ways to do this, see the docs for more information: https://fastapi.tiangolo.com/tutorial/schema-extra-example/).
- Include a screenshot of the docs that shows the example and name it example.png.

Great work overall here. You defined your Pydantic model appropriately and implemented GET and POST for your API. Very good use of FastAPI startup events too. 👍

Some of the field names have hyphens and you appear to have handled this by manually replacing them with underscores. The recommended way to handle this is to defined an `alias` for the relevant fields or to define an `alias_generator` for your Model.

See Exporting Models

It is a little unclear why you are using the `inference_utils` here since it only contains a None type object. You then load your model, encoder and binarizer and attach it to the module.

You should write at least three test cases -

- A test case for the GET method. This MUST test both the status code as well as the contents of the request object.
- One test case for EACH of the possible inferences (results/outputs) of the ML model.

✓✓ You included all the required tests here. Well done!

Another great test to add is one that checks that your endpoint throws the right exception when passed a malformed payload.

API Deployment

- Deploy using a GitHub repository with Continuous Delivery enabled. Include a screenshot showing that CD is enabled and label it continuous_deloyment.png.
- Include a screenshot of your browser receiving the contents of the GET you implemented on the root domain. Name this screenshot live_get.png.

✓✓ You have included both screenshots showing that you enabled continuous deployment and your browser at the welcome page of your application. Well done!

In production, we often need to ensure Continuous Monitoring for our models in addition to CI/CD which we enabled here. Continuous Monitoring allows us to monitor the quality of the decision making our ML system enables. See Why is it Important to Monitor Machine Learning Models?

- Write a script that POSTS to the API using the requests module and returns both the result of model inference and the status code. Include a screenshot of the result. Name this live_post.png.

✓✓ You fulfilled this rubric

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START