Week 9 - Online Prediction

In previous weeks your team built multiple models and by now have identified the best performing one for your team. After sharing your initial results, you've been asked to make this model available for recurring use and future predictions. While there are many ways to share models, a common approach is to build a simple web app using a tool like Flask. These applications expose model inferencing abilities behind a web interface where an HTTP request is used to trigger a prediction from your model and returns an HTTP response with the prediction result. This week, your team will build three Flask applications (one per team member) that each host a model. One model should be the best one from Phase 2 and the other two should use the two starter notebooks provided to you in the starter code for this week.

Learning Objectives

LO1. Prepare a machine learning model for inference

LO2. Identify model input requirements

LO3. Compare HTTP methods and their utility for online inference

LO4. Define and run a Flask application responding to an HTTP request

LO5. Define an API end point using one of the above request types

Tasks

Instructions:

1. `[Ind]` Each member of your team should follow the following steps to prepare your environment for this week's tasks.

   i. Open your capstone directory in VSCode. Ensure you activate your team's virtual environment as follows: (Linux/Mac) `source .venv/bin/activate` (Windows) `.venv\Scripts\activate`

   ii. Create and checkout a new branch

   iii. Create a new directory for a Flask application using the naming convention specified below based on the model prediction service you are building for your team.

      a. Best Phase 2 Model - `mkdir marketing_pred && cd marketing_pred`

      b. Computer Vision Model - `mkdir cv_pred && cd cv_pred`

      c. Natural Language Model - `mkdir nlp_pred && cd nlp_pred`

   iv. In that directory, create a new file titled `README.md`. You will answer a number of questions from this and future weeks in this README file.

   v. Open the integrated terminal window and run: `pip install Flask==2.3.2`

   vi. Create a new file named `requirements.txt` in your project's top level directory (`capstone`)

   vii. In this new file, add the following line: `Flask==2.3.2`

   viii. Verify your project directory looks like this (substituting marketing_pred with the directory name you created above):

```
capstone/
├── .venv/
├── .gitignore
├── requirements.txt
├── README.md
└── marketing_pred/
    └── README.md
```

   ix. Stage, commit, and push your changes to your team's directory

2. `[Ind]` Prepare a model for inference

   For each of three models (one per team member), serialize your model. Note: Each of you should name your model something unique.

   i. `[Ind]` Best Model from Phase 2. In the notebook you used to train your model import joblib and use the `dump` method to serialize your model.

   ii. `[Ind]` Computer Vision Model. Open the notebook titled, `cv.ipynb`. At the end of the notebook, import joblib and use the `dump` method to serialize your model.
   iii. `[Ind]` Time Series Model. Open the notebook titled, `nlp.ipynb`. At the end of the notebook, import joblib and use the `dump` method to serialize your model.

3. `[Ind]` Save the serialized model in your newly created directory.
4. `[Ind]` Stage, commit, and push your changes to your team's repository. Your newly created directory should now have two files (README.md and your model file.)
5. `[Ind]` Create a minimal Flask application.
   i. Create a new file in your new directory titled `app.py`.
   ii. Import Flask and define a route that listens for a `GET` request at the path `heartbeat` and responds with a success message as its response.
   iii. Run your Flask application to test its functionality. Add a screenshot of your test in the README.md file in your app's directory. (Hint: You can test a web server using curl in your terminal or shell.)
   iv. Once your app is successfully responding to HTTP requests at the heartbeat endpoint, stage, commit, and push your changes to your team's repository.
6. `[Ind]` Identify your model's input requirements. Review your team's (or the provided notebook if you are working on the computer vision or natural language processing models) notebook and identify the required inputs for your model to perform an inference. Describe those requirements in the README.md file in your app's directory. Be sure to include the following pieces of information:
   i. What method do you call for your model to perform a prediction?
   ii. What is the input type of that method?
   iii. What limitations exist for that input?
   iv. What does the input represent?
   v. Is the input pre-processed before it is passed as a paramter to the model's prediction method?
7. `[Ind]` With the answers identified above in mind, create a new endpoint in your Flask application. It must respond to HTTP requests at the path, `predict`. Use this new endpoint to expose your model's prediction method. A few things to keep in mind as you create this endpoint.
   i. You will need to dynamically access the input you identified in the previous list of questions. In Flask you do this by accessing the request object.
   ii. This may require you to adjust which HTTP method your `predict` endpoint responds to. You can learn more about the methods available at the Mozilla Developer Network (MDN) website. (Hint: the two methods relevant for this project are the GET and POST methods.) In the README.md file in your app's directory, write a short paragraph comparing and contrasting the GET and POST methods and their use cases. Include which one you are using for your app and why you chose it.
   iii. Since you deserialized your model previously using joblib, you will now need to import joblib in your `app.py` file and install it in your virtual environment. When you do, be sure to update your project's `requirements.txt` file as well.
   iv. Once you have created an endpoint that responds to the proper method at the correct path, you will need to consider how to load your model into memory when your application starts or when a prediction request is made. In your app's README.md file, include a short paragraph describing why you chose to load your model at prediction time or when your app starts. Compare and contrast the benefits of each approach. (Hint: Speed and memory are generally considered part of a tradeoff space when you make system design decisions like this.)
   v. Start your Flask application to test its functionality. Include a screenshot of your test in the README.md file in your app's directory. (Hint: curl can be configured with options to send files with a request or to use a specific request method)
   vi. Once your application is successfully responding to prediction requests at the `predict` endpoint, stage, commit, and push your changes to your team's repository.
8. `[Ind]` Now that you have finished creating a prediction application, navigate to your team's remote git repository on GitHub and open a pull request. When you do, follow the following guidelines.

i. Request for one of your teammates to "review" your changes. Once they have checked your work, you can merge your changes into dev.

ii. If you have properly completed all the previous steps for this week, after the first one of you merges your changes you will find your next merge requests are blocked due to a merge conflict. You will need to resolve the merge conflict present in your `requirements.txt` file before you can proceed to merge all your team's work. While they should be rare in this project, merge conflicts can occur and it is good practice for you to resolve them. Be sure when you do that all packages your project explicitly depends on (all those you "import" in your source files) are listed in your project's `requirements.txt` file.

9. by the end of this week, your team's repository should look like the following. If you have additional files, clean up your repository by removing them before starting on week 10.

```
capstone/
├── .venv/
├── .gitignore
├── requirements.txt
├── README.md
├── marketing_pred/
│   ├── README.md
│   ├── app.py
│   └── <your best Phase 2 model>
├── cv_pred/
│   ├── README.md
│   ├── app.py
│   └── <your serialized cv model>
└── nlp_pred/
    ├── README.md
    ├── app.py
    └── <your serialized nlp model>
```