Introduction to Machine Learning (67577)

IML Hackathon 2020

Teacher: Dr. Matan Gavish

TAs: Gadi Mintz, Elad Granot, Gilad Green, Asaf Yehudai

Tzars: Hadar Mulian, Elisha Kipnis

June 10, 2020

Contents

General Instructions

- The hackathon starts on Wednesday, 10.6, 7pm, and ends on Friday, 12.6, 11am.
- You may work in teams of three or four students only.
- In the following pages you will find two learning tasks. Choose **one** of them, and solve it as well as you can. All the materials for the tasks are available on the course moodle.
- You can find the data for both tasks here (login with your huji.mail account).
- If you want to use any data not provided by us, you must get a permission from the course staff.
- Your code should not access the internet while running.
- To enable us to run your code without errors, we require all submissions to use virtualenv and define an environment for the project. Then submit a requirements.txt file, generated using the freeze command, that summarizes the environment you use. Instructions on the virtualenv and the freeze command can be found here and here.

  * When activated, virtual environment will allow you to install via the command `pip install <package_name>` and use packages that are not installed on the CS computers.
- Solutions must be written in Python3 (the tester will be using Python 3.7, so please ensure your code works with this version). To use Python 3 on CS computers, use `python3` instead of `python`.
- Your code must finish running within a time limit which is determined per task.
- Submit through the course moodle a zip file named `id_number.zip`, where id_number is a 9-digits id of one team member. Submit your solution using his moodle account (it is not important which team member submits the solution). The file should include:
  - A folder named `task1` or `task2`, depending on the task you choose. This folder should include a single subfolder named `src` in which all your code and supplementary files should reside. Follow those carefully, as there will be automatic tests.
  - `README.txt` — contains a file list and a brief description of each file.
  - `USERS.txt` — contains the logins and IDs of all the team members. Use one line per student, in the format login, ID.
  - `project.pdf` -– a written description of your project, up to 2 pages long, as a PDF file. Explain your solution, describe your work process, and do your best to justify any design decisions you have made. Feel free to include supporting figures if you want.
  - `requirements.txt` — the required packages for your virtual environment, as described earlier. This file should be placed alongside your code files in the src folder of the task you've chosen, as described in page 4.
- We have provided a basic test script to help you make sure your submission follows the requirements. Nevertheless, you should **not** rely solely on this script, as it only performs basic checks. You can find it here.
  - The script receives one command-line argument, which is a path - either to your submission directory or to your submission zip file.
  - Usage: `python test_submission.py <submission-dir-or-zip-path>`
- Make sure to follow best coding practices, including good documentation of your code.
- We advise that you start by finding a basic solution that works. Then try to improve it as much as you can, until you're out of time.

Evaluation

In both tasks, your grade will be determined by two factors (in descending order of importance):

1. The quality of your prediction on the test set. The loss function will be the 0-1 loss for classification tasks and the L_2 loss for regression tasks. Your grade will be based both on the performance of your predictor on an unseen test set and by the rank between all submissions.

2. The quality and depth of your written description of your work that you wrote in the PDF. This will be based on clarity and originality.

Good Luck!

Before submission - Mandatory

IMPORTANT: Our tests have **zero tolerance** hence you have to follow the requested formats. Otherwise, you will receive no points.
Zero tolerance cases:

- You are obligated to submit files in the defined file folder structure, see below.
- Code has to be written in Python 3. Please make sure code works on python 3.7.3 or lower as the tester will be using Python 3.7.3.
- Output data has to be in the matching format.
- Your code should only call files that are in the folder you submit, and not use their full path. For example : "*weights.txt*" is good, and "$C:\backslash Users\backslash Miriam\backslash IML\backslash weights.txt$" is bad.
- You are required to submit `requirements.txt`.
- Your code cannot throw any exception or error when running.
- Your code has to finish prediction (including any necessary loading or initialization) in 5 minutes on CS computers.
- After zipping your project, it should take no more than 20 MB.
- You have to submit only **one** of the tasks (see directory structure).
- Submit the zip file only from **one** user in the team
- You must use the following directory structure in your submission zip file:

File or directory	Description
`<student-id>.zip`	Submission zip file
├── `USERS.txt`	
├── `README.txt`	
├── `project.pdf`	
├── `task1`	***Only if submitting task 1***
│ └── `src`	
│ ├── `requirements.txt`	
│ ├── `model.py`	
│ └── *`<other source files>`*	
└── `task2`	
└── `src`	***Only if submitting task 2***
├── `requirements.txt`	
├── `model.py`	
└── *`<other source files>`*	

Task 1 - Your Flight Has Been Delayed

Task

In this task you will deal with predicting flight delays. Given data of unseen flights, you are required to:

1. Predict the delay duration of those flight. A flight that arrived before scheduled time will have a negative delay.
2. For flights you've predicted as late, classify the main factor for the delay out of 4 delay factors (Multi-class classification).

Data

You are given data of domestic flights in the US during the last decade (2010-2019). The data includes the when, from-where-to-where and who (airline), of approx. 540,000 flights. Some of the flights arrived on time or even came in ahead of scheduled, while others were late. For those flights who were late, the data contains an indicator for the main cause of the delay. The metadata file, named `flight_meta_data.txt`, contains a detailed description of all the features in the data set.

In addition to the main data set, you are given a secondary data set of weather measurements. This data set contains main meteorological measurements for most of the airports in the main flight data set, during the past decade. The description of these measurements appears in the metadata file.

Notice: At test time you will be given flight data, in the same format as in the primary data set. The whole weather data set will be available for your predictor at test time, as mentioned in the next section. The weather data is given as an example of a way you may enrich your data if you find it helpful. In case you would like to use this or any other supplementary data you find helpful, you should find an efficient way to save the desired additional data in your submission. If you do so you should carefully pay attention to the volume and running time limits.

Code Requirements

You are required to submit a module named `model.py` (we've supplied a (very skim) skeleton file for you to implement). In this module implement a class named `FlightPredictor` that contains (at least) the

method `predict`. This method receives a `pandas` DataFrame `X` with m rows and 15 columns as input. The columns of `X` are as those in the training data, excluding $[ArrDelay, DelayFactor]$. The method should return an $m \times 2$ `pandas` DataFrame with columns $[PredArrDelay, PredDelayFactor]$ containing your predictions:

1. The $PredArrDelay$ column should contain your prediction of the arrival delay for flights you have predicted as late, in minutes. Negative delay numbers would represent an early arrival.

2. The $PredDelayFactor$ column should contain your prediction for the cause of the delay for flights you have predicted as late. The value should be one of the following strings:
   $[CarrierDelay, WeatherDelay, NASDelay, LateAircraftDelay]$.
   For flights you've predicted as not late the value should be `Nan`.

Notice that the API of the class `FlightPredictor` consists solely of the method `predict`. That means that in test time we will generate an instance of your `FlightPredictor` class and straight after that the method `predict` will be called with the test data. Take that into account when designing your solution, and make sure any initialization needed to perform your prediction is processed. Don't forget to take the running time limit in consideration as well, as it enforces the running-time of the class instantiating and prediction of the labels for the test data together.

Using Weather Data at Inference:
Upon instantiating an object from your `FlightPredictor` class, the path to a csv file with all the weather data you've recieved will be given as an argument to your class constructor. If your predictor uses this data set in any way, you should load the data you need from the given path.

Demo Test

If you choose to implement this task, the pre-submission test script will also check that your code complies with the API. For that you should download the `flights_demo_test_file.csv` and run the script with the command:
`python3 test_submission.py <submission-dir-or-zip-path> <path-to-demo> <path-to-weather>`
where `<path-to-demo>` should be replaced with the path to the supplied `flights_demo_test_file.csv` and `<path-to-weather>` is optional if your predictor uses the weather data set at inference. If so, it should be replaced with the path to the weather .csv file.
We've supplied an example for a submission that should pass all the basic pre-submission tests. It is called `00000018.zip` (18 is just the lowest valid id number).

Restrictions

- Your code needs to run in less than 10 minutes on a test set with 200,000 samples. The running time will be measured using a standard school computer.

Grading

Your code will be tested on unseen data. The performance of your predictor will be evaluated by the L_2 loss regarding the delay time you've predicted, and by the 0-1 loss regarding the classification of the cause for the delay. Your grade will be then based on a combination of both losses.

Task 2 - Who's Code is this?

Task

You are given code file from 7 Github projects. Your task is to construct a classifier that receives a piece of code (string) and tells which project it was taken from. The pieces of code can vary in size. That is, we can ask you to classify a string which is a code segment from the repository. A segment is defined as: a part of a line, a whole line or even several lines together.

Dataset

Files for this task can be found here. For each Github project, the zip contains a text file named `project_name_all_data.txt`. We also provide you with `extract_data.py`, the script we used to extract this text. You may edit or use it as you wish. A very similar script will also be used to fetch the test data. The test set will consist of all the new code pieces that will be committed to these repositories in the weeks following the Hackaton. Note that the balance of samples from each class may differ between the train- and test sets.
While the projects include many types of files, you will only be tested on the following types:

1. *.py
2. *.js
3. *.go
4. *.h
5. *.cc
6. *.java
7. *.dart

This is also specified in `extract_data.py`. Note that the separation between files is imaginary, since we provide you with a single file for each project. Also, these repositories are by definition written in python, so the majority of code files are *.py, despite the overly descriptive list above.

The chosen repositories are written mainly in python (i.e contain *.py files) but might contain files in any other programming language from the list specified above.

Notice: you are not provided with a function which takes `project_name_all_data.txt` and slices it into coherent strings for training or testing your model. We didn't want to restrict your thinking and

way of approaching research this challenge.
For each element in the input array, your classifier should output a single integer, according to the following scheme:

The Predicted Directory	Desired Output
building_tool	0
espnet	1
horovod	2
jina	3
PuddleHub	4
PySolFC	5
pytorch_geometric	6

Code Requirements

Your task is to implement the method classify in the skeleton `model.py` that we provided with the task files. **Do not change the signatures of the class or method in this file**, you may add methods as you see fit. The method classify will work as follows:

- **Input:** a numpy array of strings, X. Each string in the list is an instance that needs to be classified. A string will be composed of 1-5 lines of code. Small technical detail: Most strings are composed of consecutive lines of code. Sometimes we may join the last line of a file with the first line of the next file we read, so these will not be consecutive.
- **Output:** a numpy array of integers, each between 0 and 6, with the assigned predictions for the strings in X. The label for each project can be found in the method `classify`'s documentation (see `model.py`).

For your convenience we added an example of valid input and output of the `classify` function.
Input:

```
['if add_all:\n',
'load_library.load_op_library(lib_file)\n',
'assert data.val_neg_edge_index.size() == (2, 2)\n']
```

Output:

```
[3 2 6]
```

The First sequence originated from "jina" repo, the second from "horovod" and the third from "pytorch_geometric".

Demo Test

If you choose to implement this task, the pre-submission test script will also check that your code complies with the API. For that you should download the `demo_test_file.csv` under the data repository for this task, and run the script with the command:
`python3 test_submission.py <submission-dir-or-zip-path> <path-to-demo>`
We've supplied an example for a submission that should pass all the basic pre-submission tests. It is called `00000026.zip` (26 is the second lowest valid id number, after 18).

Restrictions

- Your code needs to run in less than 5 minutes on 5000 samples. The running time will be measured using a standard school computer.

Grading

Your code will be tested on new data, published after the hackathon is over. Accuracy for each element in the test set will be measured using the 0-1 loss. The mean value over all the test set elements will be computed as a final measurement of your success rate.