T3S: TensorFlow Simple Stupid Server

API Documentation

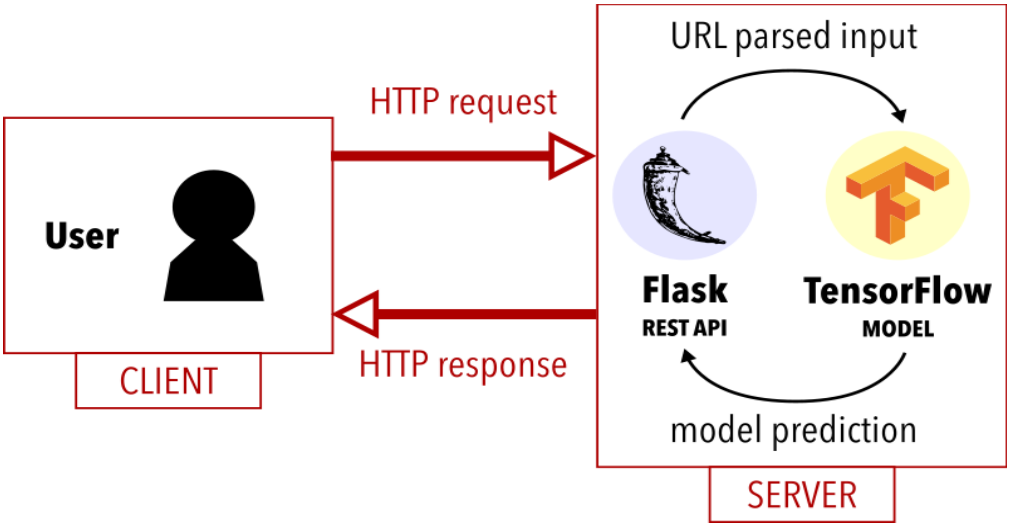
An open-source Flask/TensorFlow API – v1.1

HERETIC SAS, June 2018

A. API Presentation

The **T3S** (TensorFlow Simple Stupid Server) is an open-source project to make the development of **Flask REST APIs using a TensorFlow model** easier. The goal is to quickly setup an API to which you pass input data in the URL, that computes predictions using the TensorFlow model and outputs the results.

With this API, you can pass one or multiple examples to your model at once. It works with a simple request-response system:



When the API is asked to process a given input, it parses the URL according to certain rules (see B.3.) then it calls the model to make a prediction and answers back with the result (either numerical or categorical).

Although it was originally designed for email analysis by the **HERETIC SAS company**, the aim is to make T3S as standard as possible. In the end, it would be for everyone who is interested in linking a TensorFlow model to a web API and, hopefully, those willing to contribute to its improvement!

B. Technical Documentation

B.1. Configuring The Server

First make sure you have a TensorFlow model saved somewhere in a **`${TF_MODEL_DIR}`** directory. To learn how to prepare and export a model, you can check out the TensorFlow reference 'Wide and Deep' model tutorial.

Then, edit the **`config.py`** file to suit your needs:

1. set the **`${SERVER_NAME}`** to the address and port of your API server in the form: **`{@server:port}`** (e.g.: **`'127.0.0.1:5000'`**)
2. choose your server configuration (**`'default'`**, **`'dev'`**, **`'testing'`** or **`'production'`**) and set it in the **`configure_app()`** function

Warning: by default, the app is configured in development mode. You should change this when switching to production, or else your Flask server will still run in debug mode.

3. configure your TensorFlow model by setting the **`${TF_MODEL_DIR}`** directory
4. specify your feature computing mode:

The T3S is primarily designed only for model prediction and not feature computing, meaning you can pre-process your data and extract your feature values in whatever you wish, then send them to the API as a JSON-formatted string.

To use this raw mode, you need to set the **`TF_USE_EXTRACTOR`** variable to **`'False'`**.

However, if you would rather keep it all in the same place, you can also edit the **`tf/extractor.py`** file in the T3S folder. To have a features extraction file that is specific to your TensorFlow model, you will need to modify the **`check_data()`** and **`compute_features()`** functions. The **`extract()`** function should not be touched since it runs the process independently from your model.

To use this personalized mode, you need to set the **`TF_USE_EXTRACTOR`** variable to **`'True'`**.

B.2. Running The Server

To start the server, simply run: **`python api.py`**.

This will run the server at the **`${SERVER_NAME}`** address and port specified in your configuration file. You can now ask your model to predict outputs for given data by passing it in the URL in different forms (see B.3.).

B.3. URL Input Formatting

For now, input data can be given in two forms:

- a JSON-formatted string that holds the pre-computed features values of your examples in an array of JSON dictionaries (e.g.: **`[{"lp_length": 7, "domain": "ex.com"}, {"lp_length": 4, "domain": "ex2.com"}]`**)

- a string (with the examples separated by the **';'** character) to extract the features from, thanks to your specific extracting file (e.g.: **example@ex.com;example2@ex2.com**)

Either way, to ask the API to process the input, you simply pass it in your URL after your server root address.

For example, if using the raw mode, we could call this address:

http://127.0.0.1:5000/[{"lp_length": 7, "domain": "ex.com"}]

Or, with the personalized mode, a possible address could be:

http://127.0.0.1:5000/example@ex.com

B.4. Output Formatting

Once your model computed predictions for your examples, it will answer with a JSON dictionary containing the result for each example. For instance, if your model outputs a numerical value, you may have this type of result:

```
{
    'ex0-res': 0.43578,
    'ex1-res': 0.16733,
    'ex2-res': 0.98271
}
```

The dictionary contains one line per example (automatically prefixed by the example number) and the matching predicted value.

Note: Even though this result is displayed at the called address, it is simply printed out with no pretty-formatting because the goal of T3S is not to provide nice HTML outputting.

C. Known Issues & Perspectives

Despite our best efforts, it is complex to make an API adapted to any type of TensorFlow model. Datatypes processing, in particular, could probably be improved.

D. Development History

v1.0: initial version – Spring 2018

v1.1: multiple changes – June 2018:

- added configuration and project setup files
- standardized data types
- added in-app Python features extraction
- wrote the first version of the documentation