User Document

Praxyk (The Praxyk Group)
2015/11/30

For a User

First Section, describe the project to an outsider who is not a developer.

Praxyk.com is a scalable, easy to use machine learning on demand service. There are currently three prediction-on-demand (POD) models available. Facial recognition will recognize faces in pictures that you upload and return a JSON object with their coordinates. Optical character recognition (OCR) will look at your picture of text for you and return the text that it recognizes from the picture. Spam detection will take your text and determine whether or not it can be classified as a spam message.

To use the service, you have to create an account on the website, which requires a valid email address for account confirmation. Once you have run through some transactions, analytics and charts are available to see on your requests.

There is also a direct API and client bindings for the more technically minded consumers, as well as developers, which can be accessed by cloning the repositories in our Github.

For a New Developer

The project we are running is Praxyk.com. It consists of several main components - the backend machine learning library, the Master, the API server, client bindings, and the website frontend.

All the components are built with their own build.sh file inside their respective subdirectory.

The backend library implements our Prediction-on-Demand (POD) functionality. The primary library is written in C++, which abstracts away calls to other dependent libraries: Tesseract, CLandmark, and MLPack. The C++ library interfaces with the rest of the Python-based architecture through SWIG. This backend is distributed among many virtual machines hosted through DigitalOcean. Each virtual machine has several sandboxed worker processes which handle requests. To import and call the library in python, just “import praxyk”.

Master (also called controller-daemon) is the component that distributes requests among the workers. It uses python-rq to accomplish this. The queue created by controller-daemon is not used by the daemon, but rather is monitored for length and compared against the number of available workers.

The algorithm for load compensation is simple: If there are more jobs in the queue than there are worker containers, and creating another VM is allowed, create and set up a new VM. This is run every second.

In order to run controller-daemon, the user must be logged into a devops account capable of starting new VM's, and a docker account used to pull Praxyk docker images. The code for this module is written in Python as a daemon that write to stdout. To start controller-daemon, the user must provide the following arguments:

- max_vms - Maximum allowable number of virtual machines. controller-daemon will not spin up more than this amount.
- queue_host - The address of the queue host or redis server
- queue_password - The password used to access the queue host
- queue_port - the port used to access the queue host
- db_file - name of the file used to store the list of virtual machines started.

These arguments are provided to the script controller.py, the driver of controller-daemon on startup. One must also specify, in addition to the parameters above, start, stop, or restart.

The API server handles the authentication of requests. Once a request is authenticated, the server passes it on to the master server for distribution. This component also stores user accounts.

Our website, praxyk.com, is the primary frontend. It allows users to create an account and use the Praxyk service.

The front end is the website server. It serves the website, and functions as a wrapper to the API.