System and Unit Test Report

Praxyk (Praxyk Group)
2015/11/30

Unit Testing

This section describes how the individual components of the Praxyk system were tested.

API Server

The API server has limited unit-testing, this is because the API server contains many different sub-parts that interact in complicated ways. For this reason, the majority of the API server testing happens at the system level. See the System Testing section for more on API server testing.

Controller Daemon

In order to test the efficacy of the controller daemon, a small procedure would be run. First, a redis server is started on the redis server, and the daemon is initialized and connected to this redis server. This will create a worker VM, which should connect to the redis database using a provided configuration file packaged with the container.

Now that the environment is set up, a simple function, which simply returns the integer 15, is enqueued and, theoretically, passed to the worker. If the result is passed back to the host computer, the worker is operational. Then, another 10,000 identical tasks are enqueued simultaneously. If the controller daemon is functioning correctly, it will spin up another worker VM to process the load. If this is a success, then the test passes. Otherwise, it fails.

POD Container Build Framework

Testing was quite simple for this particular component. The script build_master.sh would be executed, and if it exited with a nonzero exit status, the system had an issue somewhere, and it would need to be found. In development, a branch that was known to compile correctly would be used to test build system functionality, so that any errors produced would most likely be from the build and not from the source code being built.

Prediction on Demand (POD)

There are three POD models, each with their own testing. All of these tests can be started at once, once the `pod/build.sh` script has built Praxyk. Navigate to `pod/build`, and run `make test`. As a whole, running these tests makes sure that our Praxyk C++ library can import its dependent libraries without error, and individual scripts test each POD functionality; the details of these tests are described below:

Optical Character Recognition

Our OCR function has two possible paths: either the given image has no text, or it has text; a function runs to test for each possibility:

- The first function takes in an image known to have no text and makes sure that the function indeed detects no text.
- The second function takes in an image known to contain capital letters A-Z and makes sure that the library detects this. Note that due to the inherent limitations of the library, as well as OCR algorithms in general, results are not 100% accurate, so our test simply makes sure the string is 75% accurate.

The referenced test images can be found here:
https://github.com/Praxyk/Praxyk-data/tree/master/pod-test-images

Facial Feature Detection

Our facial feature detection has two possible paths: either the image has zero faces, or the image has at least one face. The first path is tested once, and the second path is tested twice:

- The first function takes in an image known to have zero faces and makes sure that the function indeed detects zero faces.
- The second function takes in an image known to have a single face and makes sure that the function indeed detects a single face.
- The third function takes in an image known to have two faces and makes sure that the function indeed detects two faces.

The referenced test images can be found here:
https://github.com/Praxyk/Praxyk-data/tree/master/pod-test-images

Spam Classification

There is only one function responsible for the spam classification, and it’s tested with some text. Testing for bad input is not handled because bad input is caught at another level. Test coverage is close to 100% since there are basically no branches in the function.

System Testing

https://travis-ci.org/Praxyk/Praxyk

On top of unit testing, we set up a system to test all of the components together. This allowed us to find bugs that passed through the individual unit tests unseen.

The system testing is done via the Travis-CI **continuous integration** system (linked above). What this means is that every time we push to Github, Travis-CI detects this via a git-hook and kicks off an **automated** build process on a virtual machine and reports the results directly to us. The **continuous integration** system begins by calling the top-level build.sh script in our Praxyk/Praxyk repository. This script in turn calls all of the other build-scripts in the subdirectories of our repository, which builds our entire service in a sequential fashion.

If all of the build and installation instructions pass, then the system-testing sequence is kicked off. This is done via a call to the the top-level test.sh file. This file starts a miniature replica of our entire system on the single Travis-CI virtual machine, including a local MySQL and Redis database, our API server listening on localhost port 5000, a local task-queue, and more.

Sprint 1 :

- As a public-facing office worker, I want to be able to upload an image and receive a transcription of the image's text (OCR) as a string for handwritten form processing.
    - Navigate to test.praxyk.com:5000.
    - Click on "Register for Free".
    - Enter desired credentials.
    - Click on the link sent to given email.
    - Navigate back to Praxyk website and click on "Services".
    - Input a name in the "Transaction Name" field.
    - Select "OCR" in the "Choose a Model" field.
    - Click on the "Browse…" button and load a picture.
    - Click on "Make Request".
    - On the left, click on "Transactions" and then "Overview". The OCR transaction will appear.
    - Click on "View" and then "Predictions". The string in the "result_string" field is the transcription of the image's text.
- As a developer I want to be able to use a portable web-api so I can access these services from many different locations and programs without having to worry about library dependencies and training data.
    - Download the praxyk client bindings.
    - Start the praxyk_client.py script that exists in the praxyk client directory.
    - Use this interactive script to interact with the full range of features exposed by the Praxyk API.

    - Now you can access any of our services from anywhere without worrying about training data or library dependencies.Download our client bindings for python or Javascript(your favorite languages).

Sprint 2 :

- As a data analyst I would like to upload a picture or series of pictures and receive coordinates of faces within the photo so that I can process bulk image data more easily.
    - Navigate to test.praxyk.com:5000.
    - Click on “Register for Free”.
    - Enter desired credentials.
    - Click on the link sent to given email.
    - Navigate back to Praxyk website and click on “Services”.
    - Input a name in the “Transaction Name” field.
    - Select “Facial Detection” in the “Choose a Model” field.
    - Click on the “Browse…” button and load a picture.
    - Click on “Make Request”.
    - On the left, click on “Transactions” and then “Overview”. The Facial Detection transaction will appear.
    - Click on “View” and then “Predictions”. The string in the “faces” field is the lists the coordinates of features in each detected face.
- As a user, I want a website to be available to use the Praxyk services through a graphical environment, so I don’t have to touch any code.
    - Navigate to test.praxyk.com:5000
    - Enjoy your graphical environment.

Sprint 3 :

- As a software developer I want higher-level bindings to the API in my favorite languages (like Python, Javascript, and C++).
    - Clone the Praxyk-Client repo from https://github.com/Praxyk/Praxyk-Clients
    - cd into the root folder and run the client script using ‘python praxyk_client.py’
    - To use the Python libraries add the path to the ‘libs’ directory within the Praxyk-Clients directory to your path, then you can use them by using ‘from libs.python.praxyk import Praxyk’ in your Python program.
- As a user, I want a group of specialists to handle training and management of the most popular machine learning models.
    - See above.