|  |
| --- |
|  |
| CS 1632 – FINAL DELIVERABLE |
| Unit, Regression, and Pinning tests on Logistic Regression analysis of Yelp! reviews |

|  |
| --- |
|  |

Github URL: https://github.com/kzh4ng/CS1671-Final\_Project

Colin Swan – CRS110@pitt.edu

April 19th, 2016

# Text Classification of Yelp reviews into Seasons.

For this project I chose to create unit, regression, and pinning tests for my final project for CS1671. The project is being written in Python, and the file under test was already written before work began on this deliverable, so it could be considered Legacy testing. The project as a whole is being developed with the goal of determining whether certain words or phrases might indicate certain seasons of the year, particularly in the context of reviews for businesses on Yelp. We are hoping to accomplish this task by using common natural language processing Python libraries, such as NLTK and SKLearn, to classify a set of Yelp reviews that Yelp provides as part of their dataset challenge. We will then look at the vocabularies used by the models to determine what words were most relevant in their classifications of the text.

For this assignment our group is splitting up the work by having each person work on implementing a different model. Once the models are complete we will compare the accuracies of the classifications the models provide to determine which one performed the best in regards to precision and recall. My role in the development for this tool has been to implement Logistic Regression categorization for the reviews, which can be found in logreg.py. The tests were developed in the test\_logreg.py file.

I had several reasons for choosing this project to test. The biggest reason was because the project uses Python, and I felt it would be more interesting to try creating tests in a language other than Java for this deliverable. I have some experience programming in Python during my co-op experiences, so it’s a language that I’ve become comfortable with and that I enjoy using due to the straightforward way in which it allows developers to program. Many of the aspects of Python that allow for this can make creating good tests a bit more difficult, however, as often times tests will simply involve mocking out a large section of code. So much of Python relies on packages with simple logic connecting them that it can be hard to see what exactly needs to be tested. This is something I noticed at my last co-op; my co-worker and I had a lot of tests with good code coverage, but they were incredibly flimsy and would break after minor modifications. This resulted in both of us spending more time on maintaining tests than on actually developing the software, which defeats the point of the fast development time that Python is meant to facilitate. As a result my co-worker decided to basically scrap and redo all the tests half way through my last rotation there (most of which were his, as he had been working on the software for a year. So it was not simply due to my inexperience)! This experience turned out to be the prime motivator for my decision to take Quality Assurance, as I realized this is a topic I should have a formal understanding of to try and prevent such incidents in the future.

# Stages of Grief

Unfortunately, I again quickly ran into some of the difficulties I mentioned previously. As with my experience on co-op, much of the work I have done for this project simply involves some basic logic parsing and transferring data between different methods and objects, which are doing virtually all of the heavy lifting. This makes it more difficult to create good, useful tests as one of the principles behind good test design is avoiding testing code that is outside of the scope of the test suite. I should not be testing that a certain model returns the expected prediction for a given input; the tests for that model should already confirm that. Instead I need to mock out the object and test to make sure that the logic is correct for the current scope. However, many times the only side effect of a method for the current scope is just setting a variable for the current object, and it seems trivial to test that the variable was simply set as expected.

Another big challenge when considering the tests for this object are the uncertainties lying behind many of the methods under test. This project is a work in progress, so the methods themselves could change drastically. This makes it harder to reliably mock the correct portions of code, as a call to, say, a fit\_transform method for a vectorizer could completely change during the next iteration of the project, resulting in the tests failing. This is the issue that caused the flimsy tests during my co-op, and I believe I may be falling into the same trap with these tests as well. The train method might call the model’s fit method with a certain set of expected arguments now, but if it is determined that a different method for the model would be more appropriate for training the data then the entire test for that method could go out the window. If this happens frequently enough then the tests could become more of a hindrance to development than a benefit.

Looking forward, I think many of the same obstacles that have already been seen will continue to cause issues with test development down the line. The last major feature that needs to be implemented is determining how to retrieve a vocabulary from a model that has classified Yelp data correctly. This will involve function calls and other code structures that are hard to predict and test for beforehand. Property based testing could be done ahead of time, but if all we expect from a method’s output is a list of strings is that really worth testing? And how can the logic before a method call be tested if that logic simply involves parsing through a list of tuples? It does not seem particularly useful to mock out a method call that we expect to be performed last in the method, and then assert that that mocked out call is returning what we told it to. The only way that would help with development is if we add logic after the mocked method call, which seems unlikely to happen.

# Quality Assessment – logreg.py

Currently there are no areas of the code under test that are causing major issues. The current project runs but does not meet requirements. In particular, the following areas are still being developed:

|  |  |  |
| --- | --- | --- |
| Category | Description | Severity |
| Vocabulary | The project has not yet completed work on obtaining a vocabulary of relevant words for a season from a classification model. | Blocking |
| Classification | The model is able to classify a review, however it does not exceed the baseline expected in the project’s requirements. It at best reaches the baseline, and at worst is a few points below the baseline (which is usually 27% accuracy). | Critical |
| Testing - Code Coverage | Currently the tests for the logistic regression model cover 98% of the code, which meets the 80% expected by requirements. This is likely to change upon further development of the project, and will need to be maintained accordingly. | Normal |
| Testing – Unit and Regression Tests | While the current tests pass, there are concerns that they may not fully test the project, and new ones will need to be added upon further development of the project | Normal |
| Documentation | The tests and the project itself are documented. Documentation should be maintained as development proceeds. | Trivial |

This project is not ready to be released, as major features are not yet developed or are not yet meeting requirements. However, the project should be on track to release at the expected go-live time on April 28th, 2016.

# Tests and Coverage for logreg.py

