Preliminary Design Document

Silver Screen - October 17th, 2016

Introduction

This document aims to outline the design and architecture of Silver Screen - a web application which movie-goers and industry professionals can use to assess and track sentiments expressed within relevant movie-related tweets. Within each of the following sections we hope to share the rationale behind our design decisions and provide insight into the specific platforms and structures we have chosen for this project.

System Architecture and Rationale

While users solely interface with the web front-end of our application, a large majority of our development focus is on the acquisition and processing of data in preparation for the front-end presentation. As such, we can break down our application as follows:

1. A base Python framework used to facilitate external API calls and data transfer between modules
2. Internal sentiment analysis modules which use natural language toolkits to break down, analyze, and score tweets based on their sentiment
3. A database which stores sentiment scores, external scores (ex. Rotten Tomatoes, IMDB), and other movie information for historical lookup
4. A web framework which presents the user with sentiment scores and facilitates user requests for specific movies or historical data

Python

The bulk of our backend data processing will be implemented in Python. Python was chosen because of its many well-known packages for performing data processing. Additionally, using only Python for the internal data processing will reduce the complexity of interfacing modules with one another.

The APIs we will be using include Twitter, IMDB and Rotten Tomatoes.

Natural Language Toolkit (NLTK)

As the most difficult aspect of the development of this application is the process of analysing the sentiment of tweets, we have opted to structure our design decisions around this aspect of the project. After extensive research into natural language processing, we have found that Python strikes an excellent balance between its analytical toolkits and extendability to all areas of our project (unlike statistically-focused yet fairly narrow languages such as R). In particular, we have

decided to utilize Python's Natural Language Toolkit: a collection of lexical resources which can be used to parse, analyze, and classify text (in our case, the retrieved tweets). The NLTK also provides tools for us to reason about more complicated text, such as the FreqDist (Frequency Distribution) and Sentiment packages, which allow us to better handle cases where slang, double negatives, or similar difficult constructs appear.

We will also make use of the corpora of movie reviews which are included in the package, which can be used to train our sentiment analysis, by attributing various polarities to a set of words. More specifically, we can make use of NLTK's connectivity with the WordNet lexical database to take that training into a larger set of words, whose organization opens the doors for us to include a massive lexicon of polarized words.

Sentiment Analysis

Our sentiment analysis algorithm will be composed of two large parts - the algorithm itself and a pre-built lexicon of words and their various attributes. The two will be built in tandem, with emphasis put on creating the lexicon, whose accuracy profoundly affects the ability for our algorithm to accurately interpret sentences.

By using WordNet's synset system, we can broaden our lexicon, by assuming that synset words have similar positive and negative sentiment. We will explore using that same system to ascribe other qualities to words on a massive scale as well. NLTK also allows us to make use of the SentiWordNet lexicon, which is a subset of WordNet that ascribes positivity, negativity, and objectivity scores to all of its words. There are a variety of other lexicons which specialize in parsing the meaning of texts from social media, which we will explore in our effort to create an expansive and accurate lexicon.

Within the realm of sentiment analysis, there seem to be two major archetypes in how to quantify the meanings of words, a polar system or an intensity system. The polar system simply scores a word on whether it is positive or negative, based on its context. The benefits of this system are that it offers an easy path to create a massive lexicon which is able to pull some meaning from a given sentence. The more difficult method is to create a system which measures the intensity of the positivity, negativity, and neutrality of a given word, so that we can paint a more accurate picture of what a tweet is actually trying to say.

Once our lexical model is complete, we can start building our algorithm in a modular way. By using what we've learned from the construction of the lexicon, and by starting to analyze tweets in a supervised fashion, we can learn about the various adjustments our algorithm will need to make. Thus, the testing of our sentiment algorithm will be a semi-automated process wherein we draw conclusions about how we can better eliminate the noise of our data and find a clear signal.

Data Analysis: NumPy and Pandas

In addition to featuring the results of natural language processing on the site we would like to be able to have the ability to display other interesting features of the data. For instance, displaying breakdowns of tweet analysis over time, whether tweets with higher numbers of favourites are more positive or negative, or other parameters.

Python has a number of packages that make it strong in developing modules for these analyses. Since the data analysis that we will be implementing will likely be mostly data summation and aggregation we will be primarily using the NumPy and pandas packages. However, should we need to do more statistical operations on our data we can add SciPy to our data analysis packages as well. NumPy and Pandas provide methods and objects that allow data to be formatted in ways that allow for it to be more easily processed and manipulated.

Django

To better integrate the back-end analysis with the web-front end, we have opted to use Django - a high-level Python web framework which utilizes the Model-View-Template architectural pattern. Django aims to reduce the work needed to create complex, database driven websites using a modular component structure and a don't repeat yourself (DRY) approach. Django's clean separation between its database layer and application layer also allows Django to very scalable, a key consideration when assessing the future of our site.

Model View Template Architectural Pattern

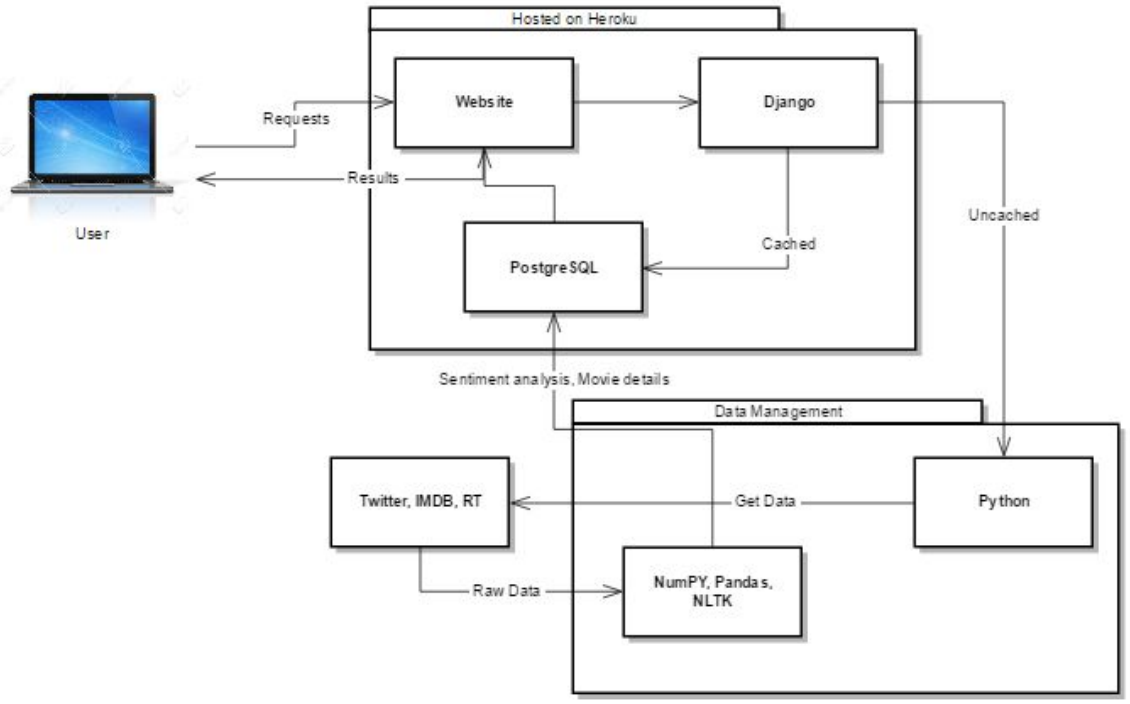
Django is based upon the Model View Template framework, a reimagining of the traditional Model View Controller framework. In this interpretation, the 'view' describes the data which is presented to the user, rather than how the data is presented to the user (a task delegated to the 'template'). In contrast with MVC, the controller (the part of the system which manipulates the model through user) is thought of as the framework itself - containing the logic to tie the model and view together.

Heroku and PostgreSQL

In order to host our application, we have opted to utilize Heroku - a cloud platform which provides a complete set of tools to build and host web applications in a fully managed runtime environment. In addition to its robustness and scalability, a major factor in our decision to use Heroku is the platform's integration with a number of development tools, including Git/GitHub.

As a result of our decision to use Heroku, we are required to use PostgreSQL - a relational database management service which supports a large majority of SQL constructs. Very similar to MySQL, PostgreSQL excels in reliability and stability and requires no licensing fees to operate.

Dynamic View and Description



The diagram above shows how the main modules of the system interact with each other. It also details how data will flow through the system after users make requests.

Data Management

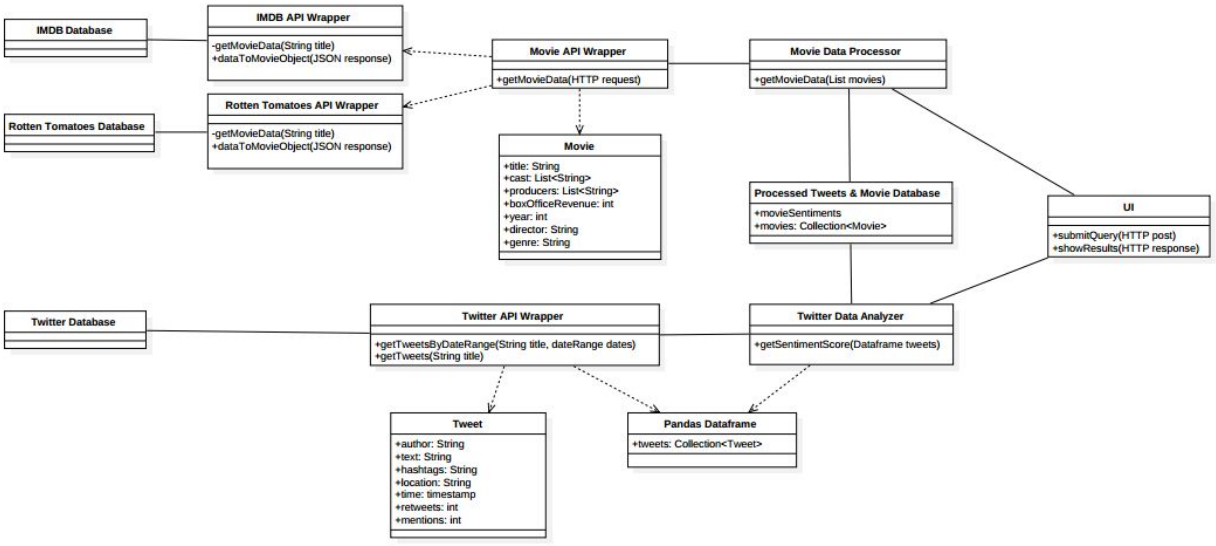
As Silver Screen is extremely statistics-focused, the importance of data structures and their management is difficult to understate. Data is being collected from three main sources: Twitter, IMDB, and Rotten Tomatoes. Data that will be needed for handling requests from the user will be stored in a PostgreSQL database.

Currently, the format of the data in the database is not finalised. The data will most likely be stored in separate tables for each of the data sources. It would be easiest to populate each table with columns corresponding to the values of each data object. For example, we can see in the figure below that after the unneeded fields of twitter data have been dropped, we are left with a relatively small number of columns to update in our database. There are a couple candidates for a primary key if we wanted to make multiple tables relating the gathered tweets. We could either assign each tweet an incrementing number id as we enter it into the table, or retain the tweet id field from the original data.

	created_at	favorite_count	lang	retweet_count	source	text	user.name	user.screen_name	user.time_zone	user.verified
1	Fri Oct 14 10:58:50 +0000 2016	75	en	40	<a href="http://twitter.com" rel="nofollow">Twitter W...	Conversation with @Potus about Star Trek, The Marti...	Joi Ito	Joi	Eastern Time (US & Canada)	TRUE
2	Fri Oct 14 15:01:07 +0000 2016	68	en	38	<a href="http://www.socialflow.com" rel="nofollow">...	Matt Damon calls 'Martian' director Ridley Scott the "ji...	Hollywood Reporter	THR	Pacific Time (US & Canada)	TRUE
3	Fri Oct 14 15:25:53 +0000 2016	42	en	24	<a href="http://twitter.com" rel="nofollow">Twitter W...	Ridley Scott will receive the American Cinemathequ...	Rebecca Ford	Beccamford	Pacific Time (US & Canada)	TRUE
4	Sun Oct 16 03:22:46 +0000 2016	NA	en	NA	<a href="http://instagram.com" rel="nofollow">Instag...	Digital Martian World Travel x Opulent Lifestyle - Thai...	A Wills	FortunePowers	Pacific Time (US & Canada)	NA

Figure: Tweets pulled from Twitter using the search term ‘The Martian’, shown after removing extraneous columns

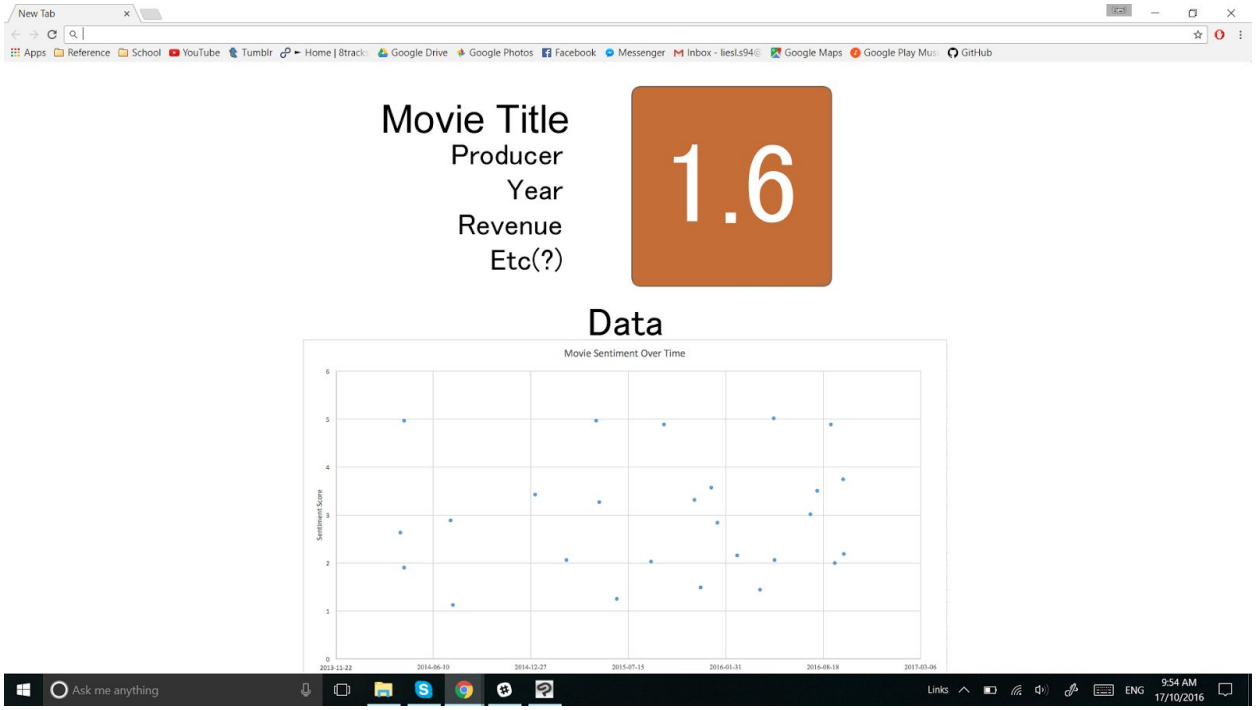
System Design

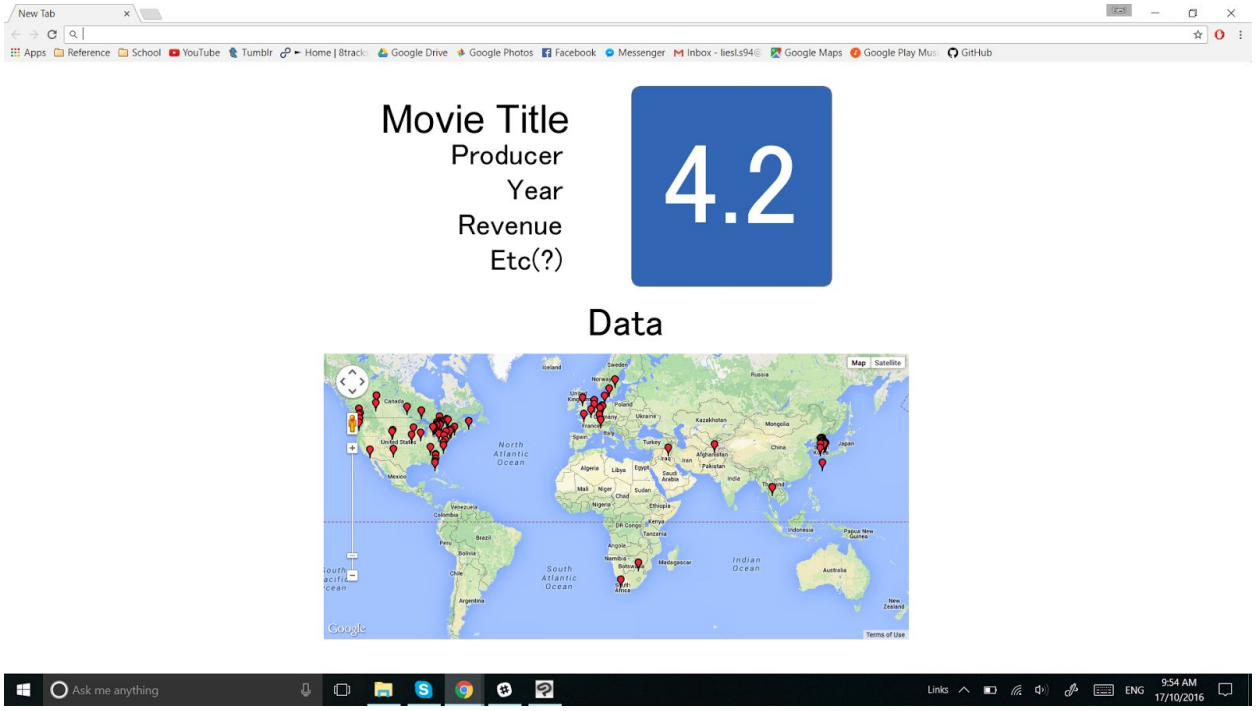


Data flows in the system is as follows, with each indentation representing a new layer:

- Client submits query
    - Query submission calls Movie Data Processor and Twitter Data Analyzer
        - Movie Data Processor calls Movie API Wrapper
            - Movie API Wrapper is made up of IMDB and Rotten Tomatoes API Wrappers that get data from IMDB and Rotten Tomatoes using their API
                - ➢ IMDB and Rotten Tomatoes return JSON responses
            - Movie API Wrapper gets JSON response and returns a collection of Movie Objects
        - Movie data processor gets critical information from movie objects, stores it in the database, and returns the relevant movie data
        - Twitter Data Analyzer calls Twitter API Wrapper
            - Twitter API wrapper makes Twitter API request
                - ➢ Twitter returns JSON response
            - Twitter API wrapper gets JSON response and returns a Pandas Dataframe of Tweet Objects
        - Twitter Data Analyzer performs sentiment analysis on the tweets, filters and reduces the data, stores it, and returns to the client
    - Data is displayed on the client side

GUI





The figures above present a mock-up of the page that a user will be brought to after they perform a search for a movie.

Our user interface will come in the form of a website, which will provide users with an easy-to-use search engine for movies and their tweet-derived sentiment scores. Our main focus lies in providing users with a movie's basic information such as the title, director, and production year as well as its tweet based score, which we will calculate during the sentiment analysis and language processing stage. We also plan on integrating information from IMDB and Rotten Tomatoes, displaying their take on a movie's score alongside our own.

Other ideas in development include creating graphs which show a movie's score with relation to time, displaying tweets made by verified accounts that may be related to the movie, and graphs which show how a movie's score changes depending on the geographical origin of the tweets.

Validation

As Silver Screen is aimed towards a large segment of the population, we must be very cautious in assuming the skillsets and expectations that our users might have. Resultantly, it is very important that we develop our application with continuous feedback from outside sources - especially from those outside of our own technical background. Time permitting, we can also consider reaching out to the alternative users described in our requirements document (mainly, industry professionals and theatre operators), although it is likely that these users will only be willing to provide feedback after a large majority of the functionality is complete.