Overview

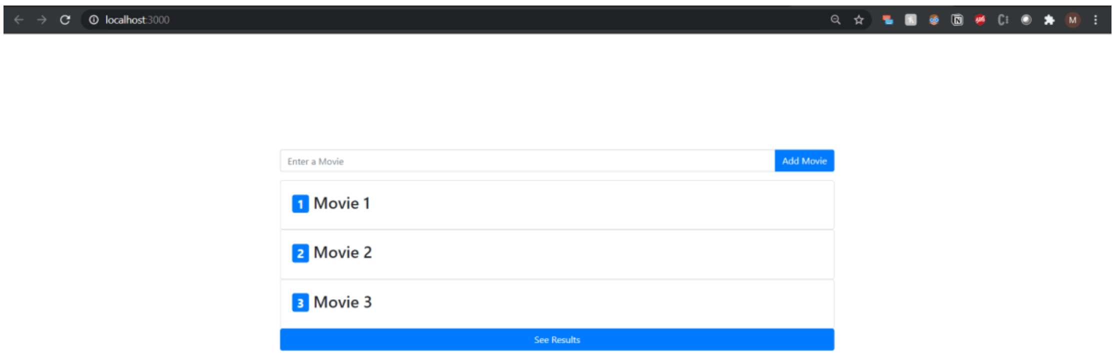
We are creating a web app that allows users to rank their favorite movies. We will then do sentiment analysis on reviews of those movies to show the user how similar their ranking is to how the general public would rank those movies.

There are 4 components to this project

- Frontend
- Backend REST Endpoints
- Review Scraping
- Sentiment Analysis

Frontend

The frontend is finished. It's also been integrated with a mock endpoint until we create the real endpoint. See image below for screenshot of frontend.



Backend REST Endpoints

The web frontend will do the sentiment analysis by sending the user data to the backend which will run our python scripts. Right now, the frontend is communicating with a backend endpoint that returns fake data. We just need to drop in the real script once it's done.

Review Scraping

The scraping of data was done using BeautifulSoup and Python Requests. We scrape the Metacritic website for user reviews as well as critic reviews then combine them into one single array containing all the arrays as strings.

To Do: Due to the lack of api, I have to create a function that will translate the input movie title, into a url-appropriate title. For example: "The Matrix" -> "the-matrix"

Sentiment Analysis

Sentiment analysis (also known as opinion mining is a text analysis technique that detects polarity (e.g. a positive or negative opinion) within text, whether a whole document, paragraph, sentence, or clause. Understanding people’s emotions is essential for businesses since customers express their thoughts and feelings more openly than ever before.

In sentiment analysis, we use some packages like nltk.classify, nltk.corpus, nltk.sentiment, nltk.sentiment.util. Each document is represented by a tuple (sentence, label). The sentence is tokenized, so it is represented by a list of strings, like the following example (['smart', 'and', 'alert', ',', 'thirteen', 'conversations', 'about', 'one', 'thing', 'is', 'a', 'small', 'gem', '.'], 'subj'). We separately split subjective and objective instances to keep a balanced uniform class distribution in both train and test sets like the following train_subj_docs = subj_docs[:80]; test_subj_docs = subj_docs[80:100]. Then, we use simple unigram word features, handling negation: sentim_analyzer.add_feat_extractor(extract_unigram_feats, unigrams=unigram_feats) and apply features to obtain a feature-value representation of our datasets: training_set = sentim_analyzer.apply_features(training_docs); test_set = sentim_analyzer.apply_features(testing_docs). We can now train our classifier on the training set, and subsequently output the evaluation results, which is shown as the follows: Accuracy: 0.8, F-measure [obj]: 0.8, F-measure [subj]: 0.8, Precision [obj]: 0.8, Precision [subj]: 0.8, Recall [obj]: 0.8, Recall [subj]: 0.8

Future Steps

- We need to decide how we want to display the result to users
- There may be some small changes to the frontend for showing results