Cody Braun

Critical Success of Films as Predicted By Script Features

Studios make choices about when to release films based to some extent on the critical success they are expected to receive. It would be beneficial for them to know in advance whether a film would be well suited for awards season or if it should be dumped on audiences over the summer. Because of changing audience and critic tastes as well as artistic norms, I was by no means certain this would be entirely possible, but I hoped to be able to predict outcomes to some extent (If for no other reason than the fact that some genres are generally poorly critically received). I also hoped to consider new features of films, such as measures of sentiment that might approximate a plot arc and be useful in other situations, such as user recommendations on Netflix.

Existing Research:

This paper, predicting box office returns based on features of a script is the most relevant: http://w4.stern.nyu.edu/news/docs/hui_scripts_5.6.2010.pdf.
The authors achieved some success in predicting box office return and financial risk associated with films, however they included several subjective variables coded by viewers related to the content of the film (so their success isn't entirely based on the script).

Methods

Data Collection:

I scraped and pickled all the scripts available at IMSDB.com and SimplyScripts, combining each with the rating at Rotten Tomatoes (which is available through their API). That yielded a corpus of approximately 2,000 scripts and ratings.

In a few early attempts at prediction, many of the words with high coefficients in Naive Bayes have been words that would appear primarily in stage directions, such as "exterior", "dark", etc. Therefore, separating dialogue and stage direction would probably be useful; however formatting in the scripts was not uniform, and successfully separating character dialogue from direction proved difficult. Furthermore, the scripts are of varying quality in terms of accuracy. Some are more or less official studio versions of the script, while others are clearly amateur attempts to transcribe the film. Some manual cleanup of the scripts would definitely be useful.

In initial classification attempts, this was treated as a multiclass problem (wherein each script would receive a predicted rating of 1-100). However, outliers tended to shift the error significantly and made it difficult to find useful features. One solution was to treat this as a binary classification problem for the sake of finding solid features. To that end, I converted

each rating to a 0 (if the film was given less than 50% on Rotten Tomatoes) or 1 (if the film was given greater than 50%).

It is still possible that I would be better served by outputting a predicted rating for each script and calculating a cost based on the distance between that prediction and the actual rating. Another technique that I understand to be common is to throw away the outliers that are changing the other predictions significantly. This might be important, because it is entirely possible that most of the scripts that I'm still misclassifying have a rating very close to 50%. Therefore, calculating a realistic cost for any given classifier is something I still need to deal with going forward. And clearly it would be more useful to a content producer such as a studio to receive a fine-grained prediction that differentiates between a film with a rating of 70% and a film with a rating of 95%.

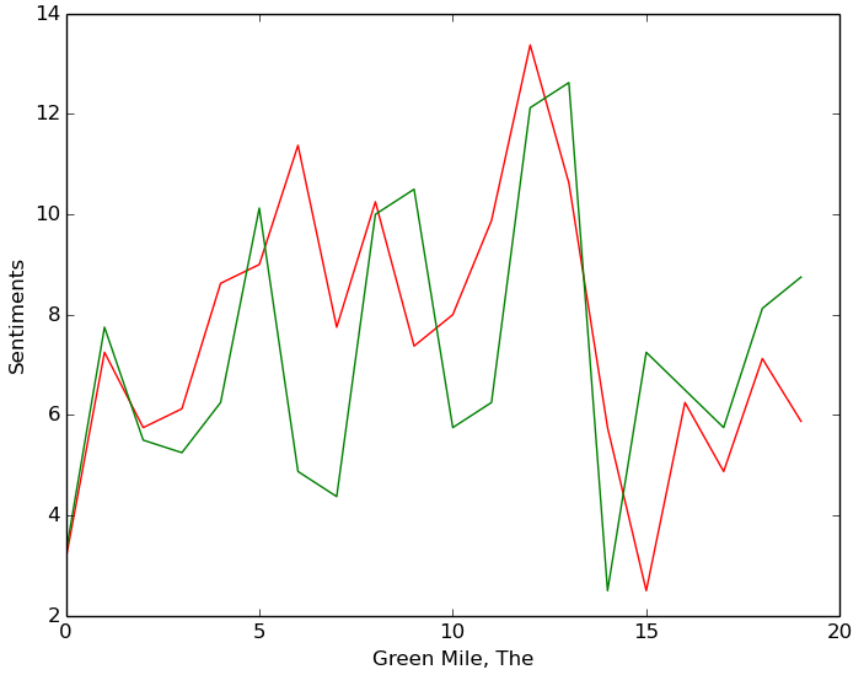
Features:

I created a wide variety of features for each script. Some of the features represent metadata about the film, which is available in a Rotten Tomatoes API query. These include title, release year, and genre. Genres are represented by a subset of all film genres, so these must be transformed into a binary feature vector, where a 1 indicates that a film belongs to a particular genre and a 0 that it does not.

Other features include lexical patterns present in the entire script, such as the length, in words, of the script, the number of unique words in the script, the number of sentences, and the average length of sentences in the film. This is calculated using nltk's punkt parser, which adapts to different sets of punctuation, which is useful because of the lack of uniform formatting in the dialogue. The "lexical complexity" was calculated to be the size of the set of words over the size of the bag of all words.

In an effort to track the action or plot arc of a film, the scripts are broken into tenths. Then, a tf-idf vector is calculated for each chunk of script and the highest tf-idf words are selected. Those words are then fed into the sentiwordnet, which essentially calculates the semantic distance of a given word from the concepts of "good" and "bad". The values are then summed.

This sentiment analysis, which will yield a good and bad value for each of the ten chunks of script, is meant to illustrate the narrative arc, as "bad" terms such as "destroy" or "hate" may tend to peak during action, while "love" or "friendship" will tend to appear more during stretches of low conflict or dramatic resolution. Many of the films have a visible arc of rising and falling action.



It is fairly easy to see that well-reviewed scripts tend to have a much more dynamic plot by simply looking at a few of these plots. Therefore, it is possible that a difference series of these plot measures will also be a useful feature. Similarly, calculating the variance of a film's "plot footprint" may have a similar effect.

Finally, a tf-idf vector is calculated for each entire script with n-grams up to trigrams. This is used to calculate the pairwise cosine distance between all scripts and also between scripts in a given year. The cosine distances between the feature vectors represents a measure of the uniqueness in general and uniqueness compared to contemporary films. (Interestingly, the list of films that are most similar to all other films includes several blockbusters).

Average overall pairwise cosine distance of films:

1623	0.972492	Independence Day
898	0.972887	Day of the Dead
986	0.972896	The Grapes of Wrath
1947	0.972906	Day of the Dead
598	0.973261	Taking of Pelham One Two Three, The
1655	0.973353	Titanic
1159	0.973359	Frequency
950	0.973418	JFK
871	0.973485	Frequency
528	0.973492	Valkyrie

The tf-idf including trigrams is also included as a feature. Without using n-grams, this feature is not particularly predictive, however with the addition of n-grams it becomes a useful feature.

The average pairwise distance did not prove to be very powerful at differentiating the scripts, largely because it is difficult to get a very large range of responses. This is largely because any given script, while it may be very similar to or different from one particular other script and generate a wider range of cosine values, is fairly distant from the aggregate of all other scripts, regardless of whether or not the year is considered. Notably, all scripts have aggregate cosine distances between .97 and .99. It will most likely be the case that selecting 10 “benchmark” scripts with which to make a comparison will yield a more useful set of distance features with broader range. To this end, I’ve begun implementing clustering to select scripts that are typical of approximately 10 clusters but have not yet attempted including it as a feature.

One important improvement I’d make with more time would be to grid search different methods of scaling and normalizing the features. Almost all are within the 0-1 range, but some features (such as average sentence length) have no upper bound and could grow to overwhelm the other features. It could also be helpful to reduce the total number of features, as some of them might be fairly useless and noisy. However, the entire feature grid take a day to build because of the number of pairwise comparisons and I haven’t had a chance to scale them differently.

Classification:

I initially expected to have some success with a simple tf-idf vector and SVM. However, this performed barely better than random. Instead, Naive Bayes and decision trees had much higher F1 and precision scores, possibly because they reflect the conditional relationships between features more fully.

Using all previously discussed features and with correctly tuned parameters, a decision tree regularly achieves close to 90 F1 score. The < 50% films are misclassified at a slightly higher rate, as seen in the confusion matrix.

```
Using Decisions tree
ACCURACY: 0.869179600887
             precision    recall  f1-score   support

          0       0.75      0.79      0.77       124
          1       0.92      0.90      0.91       327

avg / total       0.87      0.87      0.87       451


CONFUSION

     0    1
0   98   26
1   33  294
```

With a few passes of a restricted boltzmann machine, an unsupervised type of neural network to reveal hidden features, the classifiers performed slightly better. This reflects the complex conditional relationships among the various features.

For example, it may be that certain words are often found in movies that have low lexical complexity and are poorly reviewed. But these words may indicate a well reviewed movie if that script is also lexically complex. An SVM with a linear kernel fails to handle this conditional relationship, which is reflected in the coefficients of those words. However, the neural network and decision tree handles the inter-relationship of those features well.

Finally, because of the success of unsupervised neural networks in revealing features in scripts, supervised neural networks may make useful classifiers. However, the size of the feature set is prohibitive; for this reason the data dimensionality is reduced with truncated SVD. This yields a more manageable set of features for neural network learning. Though the precision of this classifier remained lower than that of decision trees, I have a fair bit of parameter adjustment yet to do (both of the SVD and the neural network itself) that may improve the promising results. It initially classifies with an F1 score of approximately .70, which is a promising start.

Implications and Future Work:

There are many possible improvements to this work, and in many ways this is just a jumping off point. Based on the relative success of this classifier, it does seem that a fair amount of a film's critical acclaim can be predicted with natural language processing. That could be useful

to studios looking for an award-worthy script. Additionally, the “narrative footprint” generated by these sentiment analysis techniques could be helpful to a company such as Netflix, which might use them as an additional feature to match users to films.

Deliverables:

- Two crawlers: return pickled dictionaries containing scripts and their Rotten Tomatoes API calls
- Consolidate script: for assembling thousands of script objects into a single data dictionary and fits a global TF-IDF vectorizer
- Script Parser script, which adds lexical features, including script length, lexical complexity, sentence length, and sentence count to data dictionary. Also handles pairwise distance and vectorizes genres
- Sentiment Finder script: breaks each script into some number of blocks, transforms each block with the global TF-IDF vectorizer, and sums the “good” and “bad” values of those those terms.
- Feature Union Builder: uses sklearn’s feature union and pipeline to assemble a block of features to train a classifier (this is where I’d like to add scaling if I have time)
- Neural Network Builder: fits a restricted Boltzmann machine to the data from feature union
- Analysis: for plotting and viewing script data
- Classifier Comparison: for loading in data, splitting it, and comparing performance of different classifiers
- SVM Grid Search: script for setting SVM parameters
- Feed Forward Network Builder Script: lowers dimensionality of data with SVD, builds and trains a feed forward network with PyBrain

Bibliography:

"Sklearn.pipeline.FeatureUnion¶." *Sklearn.pipeline.FeatureUnion — Scikit-learn 0.16.1 Documentation*. N.p., n.d. Web. 05 June 2015. http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html

Larochelle, Hugo, and Yoshua Bengio. "Classification Using Discriminative Restricted Boltzmann Machines." *Proceedings of the 25th International Conference on Machine Learning - ICML '08* (2008): n. pag. Web. http://machinelearning.org/archive/icml2008/papers/601.pdf

Pylearn2 Dev Documentation. N.p., n.d. Web. 05 June 2015. http://deeplearning.net/software/pylearn2/)

Semantic Representation of Natural Language(2013): n. pag. Web. http://w4.stern.nyu.edu/news/docs/hui_scripts_5.6.2010.pdf

Liu, Bing. "Sentiment Analysis and Opinion Mining." *Synthesis Lectures on Human Language Technologies* 5.1 (2012): 1-167. Web.
http://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf
http://www.cs.uic.edu/~liub/FBS/NLP-handbook-sentiment-analysis.pdf

"11.1. Pickle — Python Object Serialization¶." *11.1. Pickle*. N.p., n.d. Web. 05 June 2015.
https://docs.python.org/2/library/pickle.html

***Lexical Complexity and Sentence Processing*. N.p.: Chichester : Wiley, 1983. Web.**
http://pubman.mpdl.mpg.de/pubman/item/escidoc:69899:8/component/escidoc:506941/Cutler_1983_Lexical%20complexity.pdf

"Welcome to PyBrain's Documentation!¶." *Welcome to PyBrain's Documentation! — PyBrain V0.3 Documentation*. N.p., n.d. Web. 05 June 2015.
http://pybrain.org/docs/index.html

"Feedforward Neural Networks 1. What Is a Feedforward Neural Network?" *Feedforward Neural Networks 1. What Is a Feedforward Neural Network?* N.p., n.d. Web. 05 June 2015.
http://www.fon.hum.uva.nl/praat/manual/Feedforward_neural_networks_1__What_is_a_feedforward_ne.html

Work Breakdown:

- Crawler, script consolidator, and tokenizer: 300 lines of code, around 10% of time
- 2200 records downloaded, around 600 MB (once all features are extracted, this becomes several gigabytes)
- Data cleanup and classification: 400 lines of code, around 10% of time
- Reading documentation of feature combination and several other elements of sklearn that were new to me, 5% of time
- Learning about neural networks, RBM particularly: 5%
- I also started trying to use PyLearn, but ended up abandoning it: 5%
- Learning to build neural networks in PyBrain (this is surprisingly tricky) and writing neural network building scripts, 400 lines of code split across RBM and feed forward scripts: 10%
- Dimensionality reduction
- Reading comparable research on predicting film success: 5%
- Reading on sentiment analysis: 10% of time
- Parameter adjustment and parameter adjustment scripts, 200 lines of code, 10%
- Feature design and extraction, 500 lines of code split across several different scripts, 20% of time
- Plotting and analysis: 100 lines of code, 5% of time