PSL Project 3 (Sentiment Analysis): Project Report

xxx xxx

Introduction

The Data

The dataset for this project was a set of 50,000 movie reviews from IMDB. For each review, the data contained the sentiment of the review (0 for negative, 1 for positive), the score out of 10 assigned by the reviewer (all reviews in the dataset had scores between 1 and 4 or between 7 and 10), and the text of the review itself.

For the purposes of evaluation, five separate train-test splits were specified, based on review ids. For each split, 25,000 reviews were used as the training data and 25,000 were used as the test data. The 'score' column containing the reviewer's scores out of 10 was not included in the training data; only the text of the review was to be used to predict the sentiment of the review.

Project Goal

The goal of this project was to build a classifier to correctly predict the sentiment of a review (0 for negative, 1 for positive) given the text of the review. In addition, we were restricted in the size of our vocabulary, with successful projects using a vocabulary of 1000 terms or fewer.

The evaluation metric for the project was AUC on the test data for each of the five splits. A successful project would achieve AUC of at least 0.96 on the test data for all five splits.

Preprocessing

The construction of the vocabulary proved to be crucial to my project's success. For details about how the review data was parsed and a vocabulary was selected, please see my other submitted file (`xxx_project3_vocab_construction.html`). My vocabulary used **exactly 1000 terms**.

Implementation

Programming Language and Libraries

This project was implemented in Python, using pandas, numpy, and sklearn.

First Modeling Attempts

Several different models were investigated for this project, for both the vocabulary construction and sentiment prediction phases.

For **vocabulary construction**, I wanted to restrict myself to a vocabulary of 1000 or fewer terms. I tried using both a Lasso and a logistic regression model for this purpose, in each case, picking the terms that were assigned the largest coefficients by the model. I found that in general, my AUC scores were higher using the vocabulary generated by the logistic regression model (sklearn's `LogisticRegressionCV`), so my final project constructs the vocabulary this way. See vocabulary construction writeup file (`xxxx_project3_vocab_construction`) for more details.

For **prediction**, I considered several different models, including a Ridge classifier (sklearn's `RidgeClassifierCV`), a random forest classifier (sklearn's `RandomForestClassifier`), and a logistic regression classifier (sklearn's `LogisticRegressionCV`). In initial tests, the Random Forest Classifier achieved AUC values around 0.92, and the Ridge Classifier and Logistic Regression models were both between 0.95 and 0.96. Both of these models in sklearn were configured to automatically choose regularization parameter values using cross-validation for each fold.

To achieve performance that passed the AUC threshold, I tried several different iterations of the vocabulary construction process. At first, I used stopwords from NLTK, but found that performance improved when I used my own custom list of stop words (which I generated by looking at the most common words across all reviews, and choosing all 'neutral'-sounding words from among these most common words). I also initially included stemming in the vocabulary generation process (using NLTK's `PorterStemmer`), but found that performance actually improved when I removed this step. After making these changes to the vocabulary generation, I found that my Logistic Regression model cleared performance benchmarks on all 5 folds, so I chose that as my final model.

Final Model: Logistic Regression

The final model that achieved the best performance was a Logistic Regression model (sklearn's `LogisticRegressionCV`), using an l2 penalty and the default lbfgs solver. The regularization parameter C was chosen automatically via cross-validation for each of the five splits. The performance of this model on each split is shown below.

Results

Performance

The AUC values achieved by the logistic regression model on each of the five splits are shown in the table below:

Split	AUC
1	0.9613
2	0.9614
3	0.9611
4	0.9603
5	0.9608

These scores all clear the benchmark of AUC >= 0.96.

Model Interpretability

One advantage of choosing a linear regression model is that it's easily interpretable: if given a specific review and a prediction for that review, we can explain how the model reached that conclusion. On a high level, the logistic regression model will predict that a review's sentiment is positive if its computed probability of being positive is greater than 0.5.

To know where this computed probability comes from, we can look at the preprocessed review text (with stopwords removed, etc.) and the coefficients assigned to each term by the logistic regression model, and see exactly how much weight each term is given (the magnitudes of the coefficients), and which terms influence the model to believe the review is positive (those with positive coefficients) vs. negative (those with negative coefficients).

Runtime

The script takes approximately 243 seconds (a little over 4 minutes) to run on a Windows 10 machine with an AMD Ryzen 7 3700X processor (4.4GHz) and 32GB of memory.

Discussion

Sources of Error: Case Study

To get an idea of sources of error for the model, I looked at a few misclassified reviews. Below are two reviews which the model misclassified, and for which the model assigned extreme probabilities to the wrong sentiment:

Review 13 (actual sentiment 0, classified as 1 with probability 0.999): ""The Love Letter" is one of those movies that could have been really clever, but they wasted it. Focusing on a letter wreaking havoc in a small town, the movie has an all-star cast with nothing to do. Tom Selleck and Alice Drummond had so recently co-starred in the super-hilarious "In & Out" (also about an upset in a small town), in which they were both great, but here they look as though they're getting drug all over the place. I can't tell what the people behind the camera are trying to do here (if anything), but they sure didn't accomplish anything. How tragic, that a potential laugh riot got so sorrowfully wasted."

This review seems to have been misclassified because, although it's a negative review, it contains a lot of positive words as well as negative: for example, "really clever", "all-star cast", "super-hilarious", "great", "laugh riot". The title of the movie itself even contained the word "love". I'm not sure what realistic steps could be taken to address reviews like this (which reference positive attributes a film *might* have had but doesn't) without taking a completely different approach that takes overall sentence structure into account, but this is an interesting problem.

Review 5 (actual sentiment 1, classified as 0 with prob 0.03): "A very accurate depiction of small time mob life filmed in New Jersey. The story, characters and script are believable but the acting drops the ball. Still, it's worth watching, especially for the strong images, some still with me even though I first viewed this 25 years ago. A young hood steps up and starts doing bigger things (tries to) but these things keep going wrong, leading the local boss to suspect that his end is being skimmed off, not a good place to be if you enjoy your health, or life. This is the film that introduced Joe Pesce to Martin Scorsese. Also present is that perennial screen wise guy, Frank Vincent. Strong on characterizations and visuals. Sound muddled and much of the acting is amateurish, but a great story."

This review seems to have a similar problem to the first–although overall it's a positive review, it also describes the film with phrases like "acting drops the ball", "muddled", and "amateurish". For this particular review, something that jumps out at me is that it contains the phrase "not a good…", which, given that I remove stopwords and then create the vocabulary, would have been translated to "not good" and interpreted as a negative opinion of the movie.

Further Steps to Improve Performance

While my final model does clear the AUC performance benchmark of 0.96 on all 5 splits, it's quite close to 0.96. I would be interested in exploring ways to improve the performance further. Some ideas that I'd like to investigate are:

- Using a TF-IDF transformation (sklearn's `TfidfVectorizer`) rather than using `CountVectorizer` to generate the document-term matrix.

- Looking for additional training data. When I look at `myvocab.txt`, the words that were chosen first (those with the largest coefficients) seem to make intuitive sense–they're words like "worst", "waste", "boring", and "excellent", which all indicate negative or positive opinions. However, closer to the end of the list, I see that the vocabulary also includes terms like "johnny depp" and "ariel", which seem to be overfitting to a certain subset of actors and movies.
- Looking into why the model performs consistently worse on negative reviews than on positive reviews. This was true across all five splits. For example, the confusion matrix for Split 1 was:

	Predicted 0	**Predicted 1**
Actual 0	11,173	1,384
Actual 1	1,145	11,328

- Trying out XGBoost, which performed very well for me on Project 1 (Housing Data), to improve prediction accuracy. This would have the downside of making the model less interpretable compared to logistic regression.
- Further tweaking the stop word list to better understand its influence on the vocabulary construction and resulting model performance (I really only attempted two different stop word lists, one from NLTK and one based on most common words from the reviews).
- Including ngrams of length 3 or 4 to see if this can improve performance (I have only tried including ngrams of length 1 and 2).