PSL Project 3

Team

Problem Statement

We are solving a binary classification problem using IMDB movies reviews. A model is built using the review text classified into positive or negative sentiments/classes, this model can be used to predict the sentiment of a future movie based on the review text alone. The effectiveness of the model will be determined using AUC. The training set contains 50,000 reviews, which in part or full to be used for training and testing the model. Stress has been given to build a simple model with as few features/words as possible which gives an AUC of at least 0.96. Additionally, the model needs to be interpretable, in other words its output can be reviewed and explained.

Model Building

Data Bootstrapping

All training and test data were input to for building the vocabulary set. Using partial data for building vocabulary didn't result in AUC > 0.96.

Data Preprocessing

HTML characters were removed, all characters were converted to lower case, standard English stop words were removed and 4 grams were created. Furthermore, words were removed which whose term frequency/count is less than 10, which appeared in less than 0.1% of documents and which appeared in >50% of documents. This is similar to TF-IDF weighting where terms/words with higher frequency in a document are preferred and terms which are very frequent across documents are given less preference.

Stopword removal - We used the English stopwords provided by the 'stopwords' library. The stopwords are common English words which are used very frequently. The probability of their appearance is very high so they dominate the model without giving any meaningful outcome, so they are removed. Example - words like "his", "herself","yours", "could", "should" would provide little to no value to the overall sentiment of the review

Lemmatization and Stemming - We have also tried Lemmatization and Stemming during pre-processing using textstem::lemmatize_words and SnowballC::wordStem respectively. Lemmatization gets the root word of a given word so it reduces the total number of words/features without changing the meaning of a text. Example - "bad" and "worse" will turn into "bad". Stemming chops off the begining or end of words to give them uniformity. Example - "run" and "running" will turn into "run". We tested using Lemmatization and Stemming to build the vocabulary and also apply then before evaluating the "test". Neither of them increased the

AUC, and decreased it to something like - 0.9592,0.9574,0.9582,0.9576 – for five different splits. Therefore, we ended up not using it for the final vocabulary set.

Customized Vocabulary

We used 'Two Sample T-test" to choose 2,000 statistically significant words and then Lasso to reduce that to below 1,000 words. Four grams are used which makes serial combinations of 4 words as often full meaning of a word is understood in the context of adjacent words. Four-gram is better than unigram but its effectiveness reduces as we increase the grams. More details on our custom word list is described in the submitted mymain.html file.

Training process

Data is randomly split into 5 splits of training and testing samples. Using the same vocabulary built in the 1st step and training data for each split, a document term matrix (DTM) was built. Same preprocessing and four-gram are applied to training data. This DTM was fed into glmnet with alpha=0 (Ridge) and family=binomial (Logistic Regression) algorithm. Lambda value is obtained using cross validation lambda min.

Test Process

Area under the curve (AUC) was used to test the model against the 5 test splits. A value of greater than 0.96 across all splits was considered acceptable.

Tuning Parameter

There were not many tuning parameters used except cross validation to use lambda min, Stop word removal, four-gram and selection of vocabulary using Lasso on bootstrap data. These parameters were explained in detail in the myvocab implementation html doc.

Model and Performance

Our resulting model uses a vocabulary list of 979 words – which roughly uses predefined algorithms from create_vocabulary and create_dtm – and performs relatively well on the test data. Across all splits the AUC was determined to be greater than 0.96 and each split ran less than 1.14 minutes.

Results

Split #	**1**	**2**	**3**	**4**	**5**
AUC	**0.9618**	**0.9601**	**0.9612**	**0.9607**	**0.9606**
Run time	**1.128 min**	**1.126 min**	**1.137 min**	**1.137 min**	**1.121 min**

Average AUC: 0.96088

Tech Specs

We used a Windows machine with following specs.
Windows 10 Machine – XPS 15 9560
Processor: Intel Core i7-7700HQ CPU @2.8 GHz
RAM: 32 GBs

Interpretability

We have built a white box model which is easy to explain as we have the full list of vocabularies and their beta values which explains the quantum of their impact on the final out of sentiment. Please see below the top positive and negative words obtained from the model. These can be reviewed by anybody and get a sense of the working logic of this mode. If there are social biased terms then those can be simply removed from vocabulary set or even during pre-processing.

Top Negative Words		Top Positive Words	
word	beta	word	beta
4_10	-2.2153769576	7_10	2.3053767449
3_10	-1.9592734762	8_10	1.6122188514
2_10	-1.4924090589	good_worth	1.2200363381
waste	-1.0956340282	10_10	1.1985150092
1_10	-1.0805169366	9_10	1.0634345481
give_4	-1.0722373946	highly_recommend	0.9945289030
awful	-0.9386258110	pleasantly_surprise	0.8731489725
disappointment	-0.8797359182	refresh	0.8514656640
poorly	-0.8614272629	must_see	0.8405578391
laughable	-0.8346029607	give_7	0.7689263823
miscast	-0.7498957579	definitely_worth	0.7421755112
stinker	-0.7346834694	excellent	0.7314106012
skip_one	-0.7342579406	one_good	0.6981994469
want_like	-0.7179098470	keep_guess	0.6614083837
forgettable	-0.7175608994	wonderfully	0.6578005616
dull	-0.7029568074	superb	0.6336692600
go_motion	-0.6972747314	first_rate	0.5824469483
plus_side	-0.6920535252	hilarious	0.5688551070
tedious	-0.6841550219	brilliantly	0.5402792536
dreadful	-0.6809846476	wonderful	0.5386543523
unfunny	-0.6806985269	back_enjoy	0.5345871213
may_enjoy	-0.6674910166	favorite	0.5281088378
embarrass	-0.6483974669	gem	0.5229080186
terrible	-0.6416818644		

Model Limitation

This model being ‘logistic regression’ is prone to a rigid decision boundary, there is no difference between probability of 0.51 and 0.99. I think some other model like SVM which has a margin between classes can be more resilient to outliers.

There is no way to apply weights to certain terms to suite a special need. For example, we couldn’t specialize this model for a genre of movies. Words may mean different for different genre, example “violence” could be a negative word in general but for an action movie it may not be as negative. Although our model uses vocabulary list of 979 words, we think it may not be fully optimized. Finding similar words and unifying these similar meaning words in train and test data to produce optimized vocabulary list and utilizing in our model could produce better performance. N-gram doesn’t seem to be completely figured out; sometime larger grams are better but not always. I guess this is a general issue. We had to use both train and test data to build vocabulary list to create the best fit model we could create with the given data set.

Error Analysis

The example review to the right is given a probability of 0.507 which is positive but in fact should be classified as a negative review. The sentence “they don't make for a very satisfying movie” is negative but four-gram “a very satisfying movie” sounds positive. There are negative words like depressed, yelling, problem and withdrawn which don’t have high beta value to make a significant impact in our model.

> “18878- This movie is about a depressed and emotionally constricted man has a distant relative move in with him in his apartment in Istanbul. As time passes, their relationship becomes more and more strained until finally he begins yelling at his house guest--who is out of work and doesn't appear all that eager to find work. That's most of the movie in fact. The problem is that although emotionally constricted and depressed people are VERY withdrawn and non-communicative, they don't make for a very satisfying movie. That's because most of the time he (and his roomie) just stare into space and say nothing. I think all these flat moments could have been shortened to make a 30-minute movie--I certainly wouldn't have minded.”

Interesting Observation

Some of the top negative words seem ambiguous and can potentially misclassify some reviews, like skip-one, want_like, plus_side and may_enjoy. Similarly, some positive words also sound ambiguous like refresh, keep_guess etc.

Conclusion

We built binary classification model for IMDB movies reviews by performing following:
We utilized ‘Two Sample T-test” to choose 2,000 statistically significant words and subsequently used Lasso to reduce the vocabulary words size to 979 words. This vocabulary was vectorized and used to build the Document Term Matrix (DTM). The DTM was fed into glmnet with alpha=0 (Ridge) and family=binomial (Logistic Regression) algorithm. Lambda value was obtained using cross validation lambda min.

The model achieved AUC over the 5 splits above 0.96. Runtime was just above 1 minute for each split. We were able to achieve AUC higher than 0.96 by sacrificing the size of vocabulary; by bringing to 2K size which we did not keep this in our final model.

Future Steps

The data bootstrapping could have been better, instead of using all training and test data we could have randomly sampled fewer data several times to extract effective features. As elaborated in error analysis, our model may fail to capture the sentiments of the review and could misclassify it. We think deep learning based sentiment classification model might perform better. Other classification model such as Stochastic Gradient Descent (SGD), and Support Vector Machine (SVM) can be explored and compared.

Acknowledgement

1. Piazza Posts
2. Class lecture videos
3. Text2Vec: http://text2vec.org/vectorization.html