RESEARCH FIELD: Statistical Learning
PROJECT TITLE: Sentiment Analysis on Rotten Tomatoes Movie Reviews

1.0 - INTRODUCTION

Our project is a finished competition on Kaggle about sentiment analysis on rotten tomatoes movie reviews. The Rotten Tomatoes movie review dataset is a corpus of movie reviews used for sentiment analysis, originally collected by Pang and Lee. In their work on sentiment treebanks, Socher et al. [2] used Amazon's Mechanical Turk to create fine-grained labels for all parsed phrases in the corpus (see figure 1).

In this project, we are going to apply different machine learning technologies on the Rotten Tomatoes dataset. Our objective is to classify opinions in a text review by labeling phrases on a scale of five values: negative (0), somewhat negative (1), neutral (2), somewhat positive (3), positive (4).

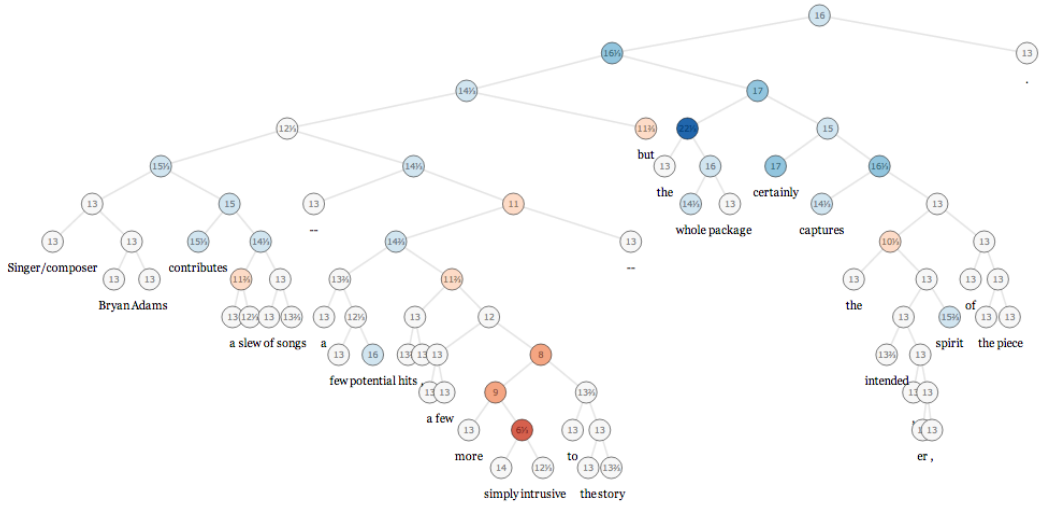


Figure 1: It shows how the phrases are parsed.

Variable Attributes

Variables	**Type**
PhraseId	Integer
SentenceId	Integer
Phrase	String
Sentiment	Integer (0 - 5)

Table 1: It includes the name and type of each of variable.

The dataset is comprised of tab-separated files with phrases from the Rotten Tomatoes dataset. Each Sentence has been parsed into many phrases by the Stanford parser. Each phrase has a PhraseId. Each sentence has a SentenceId. Phrases that are repeated (such as short/common words) are only included once in the data.

The training data set contains the phrases and their associated sentiment labels. We also have a SentenceId so that we can track which phrases belong to a single sentence.

The testing data set contains just phrases. We need to assign a sentiment label to each phrase.

2.0 - DATA PREPROCESS

Training data set consists of 156059 phrases and their sentiment values are distributed as follows,

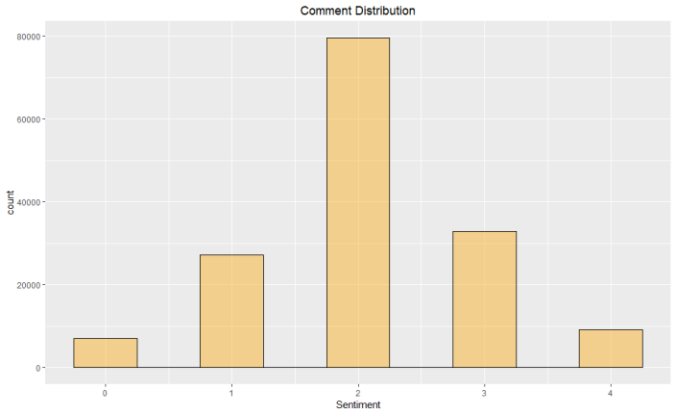


Figure 2: Distribution of Sentiment

From graph it is clear that majority of the phrases has sentiment value of 2 which corresponds to neutral sentiment. Majority of the machine learning algorithms expect input data to be vectors with numerical features. Therefore initially it is required to convert each phrase into feature vector. In this project tf-idf transformation is used in order to develop feature vectors for each phrase.

Tf means term-frequency while tf-idf means term-frequency into inverse document-frequency. This is a common term weighting scheme in information retrieval, which has also found good use in document classification. The main purpose of using tf-idf instead of using raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

Scikit python library provides several transformers for tf-idf transformation such as count vectorizer, tfidf transformer and tfidf vectorizer. There are two approaches for transforming text data to tf-idf data.
1. First use count vectorizer to build feature vectors with frequency values. Then convert them to tf-idf values using tfidf transformer.
2. Tfidf vectorizer combines the functionality of both count vectorizer and tfidf transformer. It directly converts text data to feature vectors with tf-idf values.

All tokens are stemmed and converted to lowercase to avoid duplicates when transferring text data to tf-idf format. In machine learning and data mining usually stop words (words that do not have an impact on decision making) are removed when processing text data to reduce the complexity of data. In this project, stop word removal has not performed as each phrase consists only few words and some phrases only consist of stop words.

	PhraseId	SentenceId	Phrase	Sentiment	clean_review
0	1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	1	a seri of escapad demonstr the adag that what is good for the goos is also good for the gander some of which occasion amus but none of which amount to much of a stori

Table 2: Phrase vs. clean phrase (clean_review)

N-grams
We also considered unigrams, bigrams and trigrams in tfidf since we would like to differentiate between the words "good" and "not good". This would help to understand the context better as it would take into consideration both the words together, rather than a single word.

Fixing unbalanced data
In logistic regression part, we also considered unbalanced data problem since logistic classifier could be influenced by all the data points. Unbalanced data may cause different weight to different classes. Therefore, we simulate data in another way to make sure the final training set is balanced to see whether this could help increase the accuracy.

3.0 – METHODS AND RESULTS

3.1 - Logistic Regression

Logistic Regression measures the relationship between the dependent variable (our label, what we want to predict) and the one or more independent variables (our features), by estimating probabilities using it's underlying logistic function.

These probabilities must then be transformed into binary values in order to actually make a prediction. This is the task of the logistic function, also called the sigmoid function. The Sigmoid-Function is an S-shaped curve that can take any real-valued number and map it into a value between the range of 0 and 1, but never exactly at those limits. This values between 0 and 1 will then be transformed into either 0 or 1 using a threshold classifier.

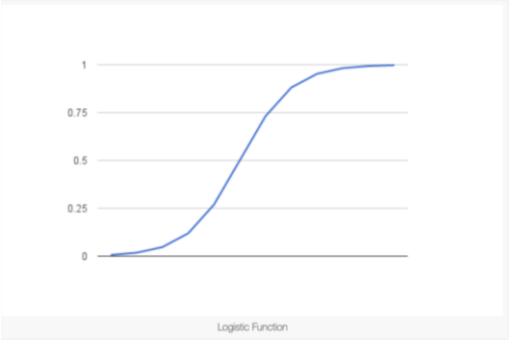


Figure 3: sigmoid function

Despite its name, it is a linear model for classification rather than regression. It is also known in the literature as logit regression, log-linear classifier and maximum-entropy classification (MaxEnt). In this

model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

For fitting the model with Logistic Regression, we have used OnevsRest classifier. This strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. This strategy is used for multilabel learning, where a classifier is used to predict multiple labels for instance, by fitting on a 2-d matrix in which cell [i, j] is 1 if sample i has label j and 0 otherwise. [3]

We also considered N-gram and unbalanced data problem into model. Here's the accuracy summary table:

	Training accuracy	Test accuracy
N-gram (1, 2)	58.55% → 56.66%	60.71% → 59.23%
Balance data	58.55% → 60.99%	60.71% → 60.86%
N-gram (1, 2) + Balance data	58.55% → 65.23%	60.71% → 60.01%

Table 3: Accuracy Summary Table

It's clear to see the accuracy of model that considered balanced data has highest accuracy 60.86%. N-gram(1,2) means the model not only considered unigram but also bigram. However, in this case, N-gram didn't help increase accuracy. And model combined with N-gram and balanced data seems over fitted since the training accuracy is high while test accuracy is not satisfying.

3.2 - Random Forest

A random forest is a Meta estimator that fits a number of decision tree classifiers on different sub samples of the dataset and use averaging method to improve the predictive accuracy and control over-fitting. Each tree in the ensemble is built from bootstrap sample from the training set. When splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases compare to the bias of a single non-random tree. It is balanced by decreasing variance through averaging hence yielding an overall better model.

We splitted training data into two parts: 80% training data and 20% test data. For the training data, we calculated tfidf with same process in the logistic regression. Then fit random forest model we got test accuracy of 62.32%, which is higher than the best result in logistic regression. However, this model took very long time to run so it's hard to say the parameters we have right now are the best ones.

3.3 - Naive Bayes

We used the multinomial Naive Bayes classifier, which takes multiple occurrences of terms into account. Training the classifier is fairly quick and simple. It estimates the prior probabilities for a class c as

$$P(c) = \frac{N_c}{N}$$

where N is that total number of documents and Nc is the number of documents that belong to class c. The classifier also estimates the conditional probabilities that a term x appears in class c:

$$P(x \mid c) = \frac{n_{x,c} + 1}{n_c + |V|}$$

where $n_{x,c}$ is the total number of times term x appears in all the documents that belong to class c, n_c is the number of terms in all the documents that belong to class c, and |V| is the size of the vocabulary. The conditional probabilities using Laplace smoothing to avoid zeros.

To classify a test document, we find $c = \arg\max_{c_j \in C} P(c_j) \prod_{i \in P} P(x_i \mid c_j)$ where P is the set of all positions in the test document that contain a term in the vocabulary, and xi is the term that occurs at position i.

We used the MultinomialNB function from sklearn package to build our model. The original data was preprocessed by procedures described in section 2. The final accuracy given on Kaggle is 57.88%.

3.4 - Support Vector Machines (SVM)

SVM usually works better than Naive Bayes. Unlike Naive Bayes, which is a probabilistic classifier, SVM attempts to find a hyperplane that divides the training documents with the largest margin. It finds the hyperplane using the SMO algorithm, and then it classifies a test document by finding

$$\sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b$$

where αi and b are the parameters for the hyperplane.

SVM works well for text classification due to its advantages such as its potential to handle large features. Another advantage is that SVM is robust when there is a sparse set of examples and because most of the problem are linearly separable.

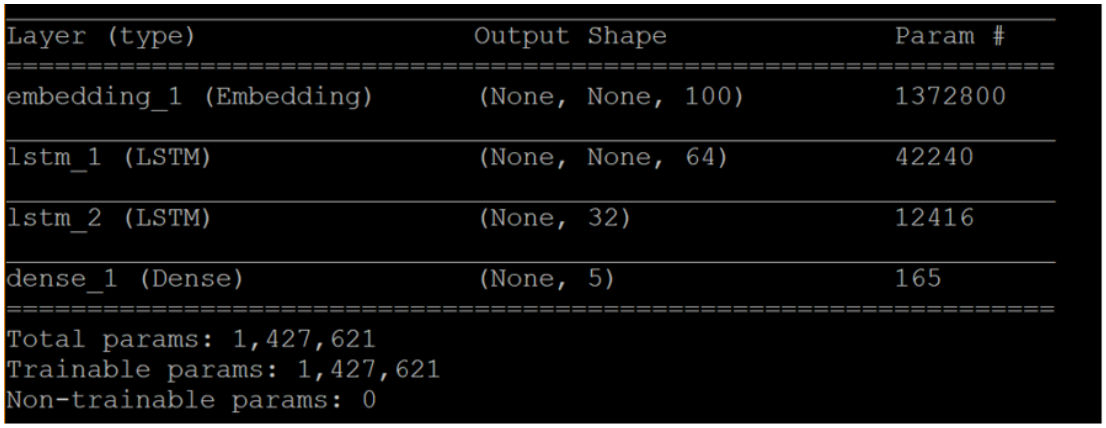
We used the LinearSVC function from sklearn package to build our model. The original data was preprocessed by procedures described in section 2. The final accuracy given on Kaggle is 60.67%.

3.5 - LSTM

Lstm is a model using three gates to control the information processing in this cell and which to circulate to the next cell. It is capable of handling "long-term dependencies."

We used validation set and categorical cross entropy to avoid overfit. In the midterm, the lstm model is constructed with 3 layers, using two ReLU functions and the last layer using the softmax activation function. Softmax activation function is often used for the last layer in classification problem. And the accuracy of validation set is about 67%.

After the midterm, we tried to use TF-IDF into the LSTM. But the matrix of TF-IDF is so large and sparse. Though, in the data processing part, we have converted the words into the stemmed word and lowercase, the number of words has decreased a lot, we still have about 15000 kinds of word in the vocabulary and we have 97000 observations in the training data. But in each sentence or phrase, it only has about 10 words, at most 20 words. So, the matrix of TF-IDF method is so sparse. And combining TF-IDF and LSTM, it costs the computer too much time. So at last, restricted by the computer power, we abandon this method. And we tried LSTM with another structure (figure 4). And we increased the accuracy of validation set is about 77%.



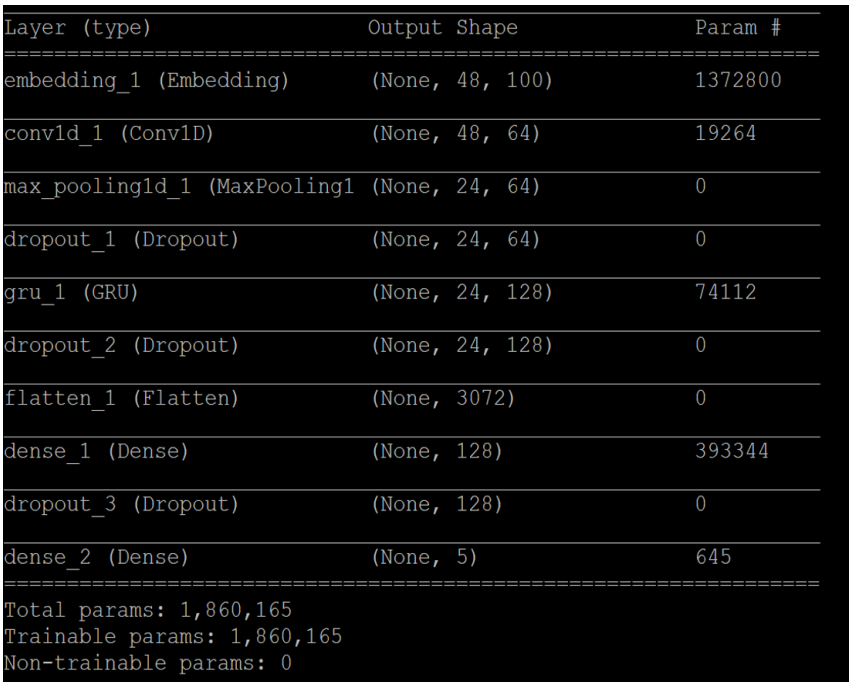
Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, None, 100)	1372800
lstm_1 (LSTM)	(None, None, 64)	42240
lstm_2 (LSTM)	(None, 32)	12416
dense_1 (Dense)	(None, 5)	165

Total params: 1,427,621
Trainable params: 1,427,621
Non-trainable params: 0

Figure 4: LSTM structure

3.6 - CNN

In the CNN model, the data processing part is as the same as the above models. And we used two convolutional layers with relu activation function, and two max pooling layers (figure 5). We have a very high validation accuracy about 85%.



Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 48, 100)	1372800
conv1d_1 (Conv1D)	(None, 48, 64)	19264
max_pooling1d_1 (MaxPooling1	(None, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 64)	0
gru_1 (GRU)	(None, 24, 128)	74112
dropout_2 (Dropout)	(None, 24, 128)	0
flatten_1 (Flatten)	(None, 3072)	0
dense_1 (Dense)	(None, 128)	393344
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645

Total params: 1,860,165
Trainable params: 1,860,165
Non-trainable params: 0

Figure 5: CNN structure

4.0 - CONCLUSION

With the addition of the improvements, Random Forest outperformed Logistic Regression. Interestingly enough, the accuracy of Logistic Regression increased only a little with the addition of improvements in N-gram and unbalanced data problem. In contrast Random Forest showed better performance with accuracy of 62.32%. For both Naive Bayes models and Support Vector Machine models, no obvious increasing accuracy was observed. Applying N-grams gives SVM models 0.1% increase on accuracy. LSTM and CNN have the good performance in prediction. Highest prediction accuracy equals to 85%.

References

[1] Pang and L. Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In ACL, pages 115–124.

[2] Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Chris Manning, Andrew Ng and Chris Potts. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013).

[3] https://medium.com/@josephroy/multi-label-text-classification-rotten-tomatoes-f998bf54f81

[4] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Appendix

Individual contributions:
Wanting Tan: built Logistic regression model and random forest model
Improved logistic regression by adding cross validation, N-gram and balanced data
Xuejun Chen: Data preprocessing, Naive Bayes and SVM models
Shuangshuang Xu: LSTM and CNN part