COMP4332 / RMBI 4310 Project 1 Report (Sentiment Analysis)

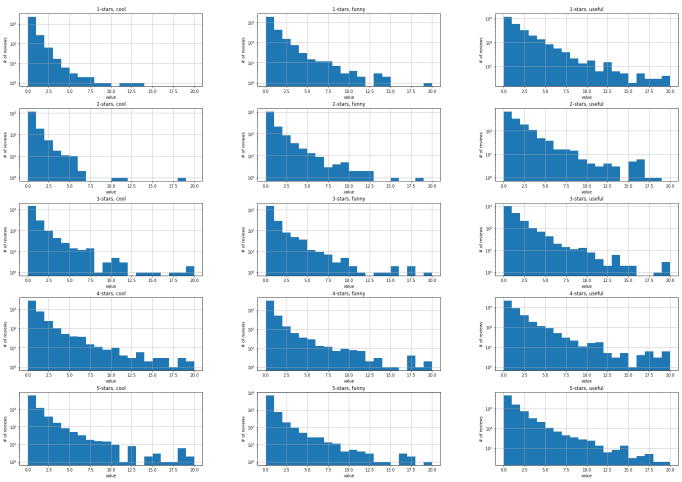
Team Members: SETHI Aryan (20634962), CHOW Hau Cheung Jasper (20589533), KAUSHAL Kaustubh (20634039), PAREKH Yashasvi Gopalbahi (20634089)

Team Name: AKJY **Group ID:** 05

Introduction

Sentiment analysis is a natural language processing technique for extracting and identifying the emotion expressed towards a particular topic in text. Typical applications of sentiment analysis are for analysing survey responses, customer feedback and product reviews. While sentiment analysis is rarely perfectly accurate due to the highly subjective nature of language, fine-grained sentiment analysis can provide an insightful and valuable picture of the general sentiment surrounding a certain topic, and which particular words and phrases convey that sentiment.

Preprocessing and Exploratory Data Analysis



In this project, we have been given data of reviews from 18000 users for various businesses and services. The data contains the following information about each user: *business_id, review_id, user_id, date* (the date the review was posted), *text* (the actual review given by the user), *stars* (an integer rating from reviewer in the range [1,5]), and the reactions the review received (*cool*, *funny*, *useful*). Our objective is to use this data to build a model that will predict the **rating** of each reviewer (i.e. the stars column) based on their review (i.e. the text column). A preliminary analysis of the dataset suggests that prediction of the star rating may be improved by considering *cool, funny,* and *useful* features. However, close examination of the distribution of each of these features for each star rating (figure on left) shows no discernible difference between star levels (column-wise), hence we did not consider them as useful features for our models. We also computed the pairwise correlation between the features and observed that they have almost no correlation with the number of stars.

	cool	funny	useful	stars
cool	1.000000	0.682619	0.665464	0.054584
funny	0.682619	1.000000	0.564596	-0.061035
useful	0.665464	0.564596	1.000000	-0.083490
stars	0.054584	-0.061035	-0.083490	1.000000

Our preprocessing approach initially involved punctuation and stopword removal, text tokenization and stemming the tokens. However, after running all our models and removing each preprocessing technique one by one, we discovered that simply preprocessing the data by tokenizing the text (and NOT applying stemming/lemmatization) showed superior performance (higher average macro F1-score) across all models. Thus, all data was preprocessed by only tokenizing the text. We developed different types of models for sentiment analysis, including RNN, CNN, Logistic Regression, Naive Bayes, SVM, Gradient Boosting, and Random Forest. However, only Logistic Regression and RNN turned out to be our best performing models out of all the models we tried on our validation set, so we only focus on these two data mining models in further depth.

Data Modelling

Recurrent Neural Networks (RNN)

A vanilla RNN suffers from the vanishing/exploding gradient problem, since repeated multiplications are required to compute the gradient (and thereby update the weights) at very early timesteps. Therefore, to avoid this, we used the LSTM neural network architecture with 2 dense layers followed by a softmax to get a 5-d vector representing the prediction probabilities for each star rating. Initially, we believed that deploying an LSTM to be the most effective way of handling such text data, which is sequential in nature. Lots of experiments were carried out under the RNN Classifier Model, from using various loss functions (LDAM, Macro Soft F1, categorical cross entropy), changing the number of dense layers, the number of stacked RNN layers, and various hyperparameters such as the learning rate and dropout rate were adjusted too to see if the model's performance would improve. Experimentation revealed that neither increasing the number of dense layers nor increasing the number of stacked RNNs (even with residual connections) improved the performance. However, the most significant performance improvement came from utilising different word embeddings. Our most noteworthy results are summarised in the table below.

LSTM Size	Loss	Dimensionality of word embedding	Other parameters	Validation average macro F1

COMP4332 / RMBI 4310 Project 1 Report (Sentiment Analysis)

Team Members: SETHI Aryan (20634962), CHOW Hau Cheung Jasper (20589533), KAUSHAL Kaustubh (20634039), PAREKH Yashasvi Gopalbahi (20634089)

Team Name: AKJY **Group ID:** 05

Bidirectional LSTM with 64 cells	Categorical Cross-Entropy	128 (Randomly generated)	Learning rate = 2e-5 Dense layer L2 regularisation = 0.0001	0.41
Bidirectional LSTM with 64 cells	Categorical Cross-Entropy	128 (embeddings trained with Fasttext (GloVe))	Learning rate = 2e-5 Dense layer L2 regularisation = 0.0001	0.44
Standard LSTM with 128 cells	LDAM	300 (the pretrained Google News Word2Vec)	Learning rate = 2e-5 Dense layer L2 regularisation = 0.0001 Recurrent dropout rate = 0.2 Embedding layer frozen	0.47
Standard LSTM with 128 cells	LDAM	300 (the pretrained Google News Word2Vec)	Learning rate = 2e-5 Dense layer L2 regularisation = 0.0001 Recurrent dropout rate = 0.2 Embedding layer able to be fine-tuned	0.53

We found that the best results in the average macro F1 score came from fine tuning the pretrained word embeddings.

Logistic Regression

After tokensing the text in the datasets, TFIDF Vectorizer was used to transform the text into feature vectors to use as input to the estimator. From this, GridSearch was used on the training set to determine the optimal hyperparameter values for max_df, penalty, and c. Through GridSearch, the optimal estimators found for the parameters were: Best Penalty = l2, Best C = $10^{2.204}$, Best max_df = 0.75. L2 Regularisation was used as our default penalty since the lbfgs solver was used. Once the optimal hyperparameter values were found, they were used in our Logistic Regression Model and the default n-gram range in the TFIDF Vectorizer was changed to (1,2). This was so that the model would consider both unigrams and bigrams of text to make a potentially more accurate analysis on the sentiment of the review. Further ranges of n-grams (i.e. n=3 or more) were not explored due to the limited computing power and time we had to build our model and using GridSearch to find the optimal n-gram range did not provide any timely results which could be used in our model. We fit our Logistic Regression Model into the general pipeline and first used it on the training dataset to ensure the model can learn how to extract and analyse certain text before deploying it. Once the model was trained, the model's performance (rating prediction) on the validation dataset is shown on the right along with the benchmark of a strong performing model. Our Logistic Regression Model with TFIDF Vectorizer was able to outperform the strong baseline model on all metrics, suggesting this model's performance was strong enough to be deployed. As such, we used this model then to predict the ratings of the reviews on our test set, and generated ratings for 4000 new reviews.

Logistic Regression Model Performance	F1-Score	Precision	Recall	Accuracy
Macro Average	0.5575	0.5641	0.5569	0.6625

Strong Model Baseline Performance	F1-Score	Precision	Recall	Accuracy
Macro Average	0.5370	0.5509	0.5327	0.6500

Conclusion

In general, all RNN experiments revealed that the validation loss and macro F1 scores stabilise relatively quickly (within a few epochs) while the training loss continues to decrease. This was a symptom of overfitting, and is why no further experimentation with more complex RNN structures was done. Additionally, all of the models we tested (RNN, Naive Bayes, Random Forest, Gradient Boosting, SVM) showed poor macro F1 scores of the underweighted classes (2-star and 3-star), while performing relatively well on 1-star and 5-star reviews, despite use of the customised LDAM loss to specifically counter class imbalance with RNN. There is a possibility that we simply have insufficient data to capture a signal significant enough from the minority classes, because it is difficult to quantify the difference between a 2-star (middling-bad) and 3-star (middling review). For instance, the *review_id* 'gQZAmO59ZMVyEC0XOJ57rg' in the test set primarily uses positive language to describe their experience but the user only gave a rating of 3 stars, suggesting there may be a discrepancy between what reviewers write in their reviews and the rating they give, something which most NLP models may not be able to distinguish. As most NLP tasks nowadays utilise pretrained whole models (with the exception of the final layer for a downstream task like sentiment analysis) to develop good parameter initialisations, we believe that using transformers such as BERT may help further augment the RNN model performance. However, for the given dataset, we generated *pred.csv* from Logistic Regression with TFIDF Vectorizer as it was the best performing model and the only model out of all the models we tried that outperformed the 'strong' baseline benchmark on the validation dataset.