SMTP (Mathematics) Written Report

Sorting sentiments of hotel reviews through machine learning

Group 8–01

Fu Jinghang (3i104) Tan Jia He (3i325)

Yap Hao Ming Darren (3i227, L)

2023

Hwa Chong Institution (High School)

Contents

1 Introduction and Rationale

After the Singapore government relaxed travel restrictions due to COVID-19, there has been a recent increase in the number of tourists travelling in and out of Singapore. As such, hotels have seen a rise in the number of prospective tourists to be housed, and this may encourage an increase in the number of reviews hotels may receive.

Today, it is common to use social networks, messengers, and review websites to receive data from customer opinions. This is especially true for hotels, where previous occupants may evaluate the hotel on several factors through their reviews—be it cleanliness, facilities, location and convenience, etc. These come in two forms—a quantitative review (based on stars, diamonds, hearts, etc.) and a more qualitative review through text.

However, quantitative reviews do not always paint the full picture of customers' opinions towards a certain hotel. Though it is certainly helpful to have a more objective rating system using numerical scores, eg. the Department of Tourism grading system in the Philippines, or the European Hotelstars Union system, these are given by customers subjectively and do not reflect the reasons for customers giving the rating. There is also evidence of manipulation of ratings by hotel management itself, where hotels may be compelled to forge positive or negative ratings to bias the overall rating. This made up 2.1% of the 66 million reviews submitted to TripAdvisor (TripAdvisor, 2019). Therefore, we propose using sentiment analysis to extract customers' true feedback on hotels instead.

2 Objectives and Research Questions

2.1 Objectives

1. To run sentiment analysis on individual words and quantify them on a numerical scale

2. To run sentiment analysis on paragraphs and quantify them on a numerical scale

3. To use sentiment analysis on hotel reviews to determine consumers' overall opinions of hotels

2.2 Research Questions

1. How could we quantify the sentiments of individual words on a numerical scale?

2. How could we quantify the sentiments of paragraphs on a numerical scale?

3. How could we use tokens in hotel reviews to predict the overall sentiment of the review?

2.3 Fields of Math

- Statistics and probability theory

2.4 Terminology

Below is a listing of the terminology, mostly pertaining to sentiment analysis, used in this report.

Term	**Definition**
Sentiment analysis	Process of determining the emotional tone of text
Token	A unit of meaning (usually a word) that carries sentiment
Tokenise	To split a piece of text up into its constituent tokens, to be used for further analysis.
Lemmatise	To sort, so as to group together, inflected or variant forms of the same word, eg. ‘watching’, ‘watchful’, and ‘watched’
Stop words	Tokens that carry little meaning during sentiment identification, such as ‘is’, ‘I’, ‘that’, etc.; that should be filtered out before the processing of text
Lexicon	A dictionary that maps singular tokens to a category of sentiments or sentiment scores
Polarity	Whether a piece of text is positive, negative or neutral in sentiment
Bipartite sentiment	Sentiment which is grouped into two categories, usually positive and negative
Tripartite sentiment	Sentiment which is grouped into three categories, usually positive, negative and neutral
Sentiment score	A number that shows the overall sentiment of a token or a piece of text

Lexicon-based sentiment analysis	A method of sentiment analysis which sorts tokens into categories, aided by a lexicon, then calculates the overall sentiment score of a piece of text
Corpus-based sentiment analysis	A method of sentiment analysis which relies on the co-occurrences of tokens within a piece of text itself, rather than relying on an external lexicon
True positive	A prediction made by a machine learning model that **correctly** matches up with its intended value, e.g. should a reviews sentiment be positive and a model predicts its sentiment to be positive, the model has made a true positive prediction
False positive	A prediction made by a machine learning model that **incorrectly** reflects its intended value, e.g. should a reviews sentiment be negative and a model predicts its sentiment to be positive, the model has made a false positive prediction

3 Literature Review

Sentiment analysis is a field of study which utilises computational methods to analyse text, and then categorises the text, usually into three main polarities—positive, neutral and negative. It has broad applications which range from determining consumers' opinion in sales and product analysis, to competitor research in marketing, and even detecting public opinion in social media monitoring. Sentiment analysis will be used in this project to analyse hotel occupants' reviews, and also determine the most significant upsides and

downsides of each hotel, without interference from human bias.

The stock market opinion of StockTwits communities of expert investors was predicted. Sentiment analysis was used in a Deep Learning model to extract sentiment from Big Data. A Pearson Correlation Coefficient combined the linear correlation between users' sentiment and future stock prices, which proved the accuracy of user sentiment to be 53%. It was concluded that convolutional neural networks (CNN), a type of Deep Learning artificial neural network, was able to predict stock market movement based on sentiment (Sohangir et al., 2018).

Using the social networking site Twitter, Filipinos' sentiment in response to the Philippine government's efforts at tackling COVID–19 was determined (Villavicencio et al., 2021), specifically the implementation of vaccination. Natural Language Processing (NLP) techniques such as sentiment analysis were used to extract sentiment from text in the English and Filipino languages, which was used to train a Naïve-Bayes model. A confusion matrix was produced, representing the prediction accuracy of the Naïve-Bayes model (81.77%) at classifying sentiment into positive, neutral and negative categories. It was concluded that sentiment analysis towards COVID-19 vaccines were very accurate, even helping the Philippine government better conduct budget planning and coordinate COVID-19 efforts.

Tourism quality in Spain was analysed (Borrajo-Millán et al., 2021) by extracting sentiment from reviews by Chinese people on the tourism social networking sites Baidu Travel, Ctrip, Mafengwo, and Qunar. Two sentiment analysis methods, lexicon-matching and corpus-based machine learning methods, were used. These methods allow the processing of unstructured text of comparatively longer lengths. Clustered data visualisation categorised aspects of Spanish tourism into positive and negative groups, with the majority residing with positive sentiment. It was concluded that sentiment analysis can be used to improve

tourism quality and sustainability decision-making.

SentiStrength, a tool for lexical sentiment analysis—sentiment analysis done on short, low quality texts—was used to study emotions expressed in GitHub commit comments of different open-source projects (Guzman et al., 2014). Their method involved assigning scores to each word, then calculating the net score for each comment. SentiStrength splits each comment into snippets, assigns each a score by computing the maximum and minimum scores of the sentences it contains. Following which, the average of the positive and negative scores is taken as the sentiment score of the entire commit. This study showed that Java projects warranted more negative comments, and projects which had more distributed teams tended to have a higher positive sentiment.

In conclusion, the literature reviewed showed many possible applications of sentiment analysis in quantifying the underlying emotion of feedback on online platforms. Lexicon-based sentiment analysis, which assigns each word a sentiment, then calculates a sentence's total sentiment score, can be used, due to its simplicity in implementation, and the availability of many open-source sentiment lexicons. In addition, sentiment categorisation using lexicon-based sentiment analysis makes accurate predictions upwards of 70% of the time (Khoo and Johnkhan, 2017). SentiStrength would also be useful for detecting sentiment from hotel reviews which are usually short in length quickly and efficiently, optimising the process of extracting sentiment from tourists' reviews of hotels. Using SentiStrength for sentiment generation is also rather accurate, generating both positive and negative sentiments with more than 60% accuracy (Thelwall et al., 2010). Therefore, the strategies listed above could be adopted or emulated on a smaller scale for this project.

4 Methodology

4.1 Research Question 1

How could we quantify the sentiments of individual words on a numerical scale?

A valence dictionary, which maps individual tokens to a single sentiment score, was sourced for. After collecting an open-source dataset of international hotel reviews in English, the reviews were tokenised: split into its individual tokens. Each word was assigned a sentiment score, and word clouds were generated to display the most frequent positive and negative tokens in the dataset.

4.2 Research Question 2

How could we quantify the sentiments of paragraphs on a numerical scale?

SentiStrength was used (Guzman et al., 2014b) to evaluate the total positive and negative sentiments of each review. Two scores for each review were expected: the total positive sentiment score (in the range $[1, 5]$) and the total negative sentiment (in the range $[-5, -1]$). This is similar to how GitHub commit messages were classified.

To classify the sentiments of each review into one of three categories: positive (1), neutral (0) and negative (-1), the two scores of each review were added. We define the following, where k represents the polarity of a review, and S_+ and S represent the positive and negative sentiment scores of a review:

$$k = \begin{cases} 1 & \text{if } S_+ + S_- > 0 \\ 0 & \text{if } S_+ = S_- \\ -1 & \text{if } S_+ + S_- < 0 \end{cases} \tag{1}$$

4.3 Research Question 3

How could we use tokens in hotel reviews to predict the overall sentiment of a review?

4.3.1 Logistic Regression Model

A **logistic regression model** was constructed to classify the reviews collected into two categories: those with a net positive sentiment and those with a net negative sentiment. The reviews were split into training, testing and prediction sets.

Each review was tokenised and lemmatised. A feature vector for each review was then constructed, which shows how often each word occurs in the particular review—the term frequencies for each word, where $\text{tf}(t, d)$ represents the relative term frequency of token t in review d:

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2}$$

where $f_{t,d}$ is the raw count of a token t in review d. The denominator of the above function is simply the total number of tokens in review d.

To evaluate the relevance of a token in determining the overall sentiment of the review, the **term frequency-inverse document frequency** (*tf-idf*) is found, where:

$$\text{tfidf}(t,d,D) = \text{tf}(t,d) \cdot \lg \frac{N}{1 + |\{d \in D : t \in d\}|} \tag{3}$$

also where N is the total number of documents in the set of reviews, and $|\{d \in D : t \in d\}|$ is the number of reviews where the token t occurs. 1 is added to the denominator to prevent a division-by-zero scenario, should the token not appear in the review at all.

A conventional **sigmoid function** was used to evaluate the probability of a review carrying a net positive sentiment, a value from 0 to 1:

$$h_\theta(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{4}$$

where $h_\theta(x)$ is the probability of the net sentiment of a review being positive, and $\beta_0 + \beta_1 x$ is a linear function of *tf-idf* x.

The error in the logistic regression model is given by the cost function:

$$\text{cost}\left(h_\theta(x), y\right) = \begin{cases} -\log\left(h_\theta(x)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta(x)\right) & \text{if } y = 0 \end{cases} \tag{5}$$

where y is the predicted output (a positive or negative outcome) for every x.

Lastly, the model used gradient descent to minimise this cost.

After the model was constructed, predictions were made using the aforementioned prediction dataset. Precision-recall and receiver operating characterstic curves were plotted to evaluate the precision of the model, and how well the model distinguishes between false positive and true positive predictions.

4.3.2 Random Forest Classifier

A **random forest classifier** was also constructed to predict whether hotel reviews carry a net positive or negative sentiment. The individual positive and negative sentiment scores generated by SentiStrength were used as *features*, fed into the random forest classifier.

Similarly to the logistic regression model, vector representations for each review were generated, with each token's corresponding *tf-idf* values evaluating the relative importance of each token in determining the sentiment of each review. By using the *tf-idf* values of each token as the independent variable and its corresponding sentiment polarity as the dependent variable, the classifier was trained.

As above, precision-recall and receiver operating characteristic graphs were plotted to evaluate the precision of the random forest classifier, as well as how well the model distinguishes between true positive and false positive predictions.

5 Results

5.1 RQ1: Token-Based Sentiment Analysis

After tokenising English hotel reviews from the dataset of international hotel reviews, the *Afinn* lexicon was used to assign each token a single numerical score. This score ranges from $[-5,-1]$ for negative tokens, and $[1,5]$ for positive tokens, with the magnitude representing the degree of negativity or positivity the token holds. Tokens with a sentiment score of 0 carry neutral sentiment.

While it is observed that there are more tokens with positive sentiment than their negative counterparts, the vast majority of tokens in Figure 1 hold a neutral sentiment

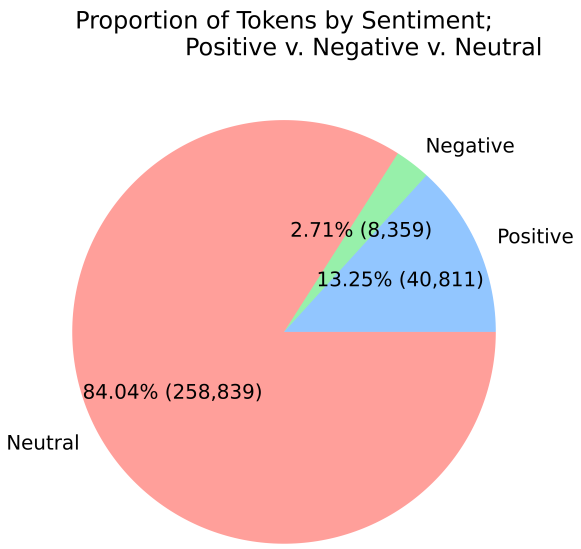

Figure 1: Proportion of tokens by sentiment (positive v. negative v. neutral)

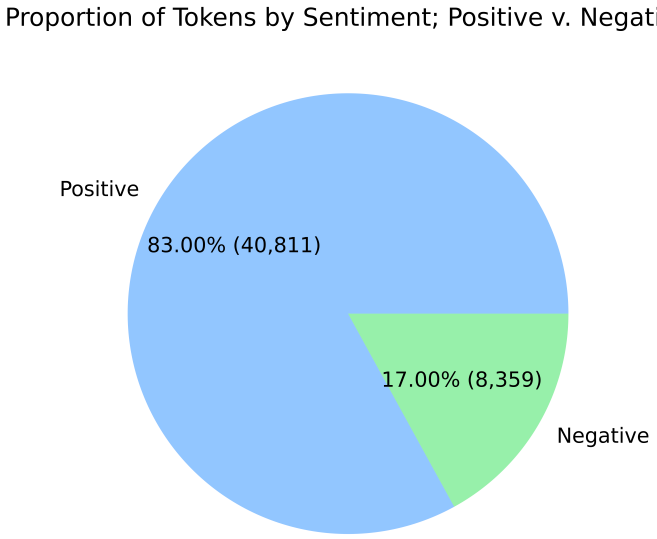

Figure 2: Proportion of tokens by sentiment (positive v. negative)

(84%), due to the presence of many standalone tokens that do not express opinion, eg. *hotel*, *food*, and *visit*.

Word clouds of positive and negative tokens were also produced. The more frequently a token appears in either the word clouds of net positive or net negative hotel reviews (that carries either a positive or negative sentiment based on the lexicon), the larger the word in the word cloud, hence the more significant the token, and the better it represents the positive or negative features of a hotel.

Figure 3: Word cloud (positive tokens)

Words of positive sentiment of greater size in the word cloud, such as *great* and *nice*, are more prominent traits that tourists associate with their experience at the hotels, while words of negative sentiment, such as *lobby* and *bad*, are more common among customers with dissatisfaction towards their hotels.

5.2 RQ2: Review-Based Sentiment Analysis

SentiStrength was used to generate two scores for each review: a positive and negative score, initialised to 1 and −1 respectively. As the reviews were analysed on a token-by-token

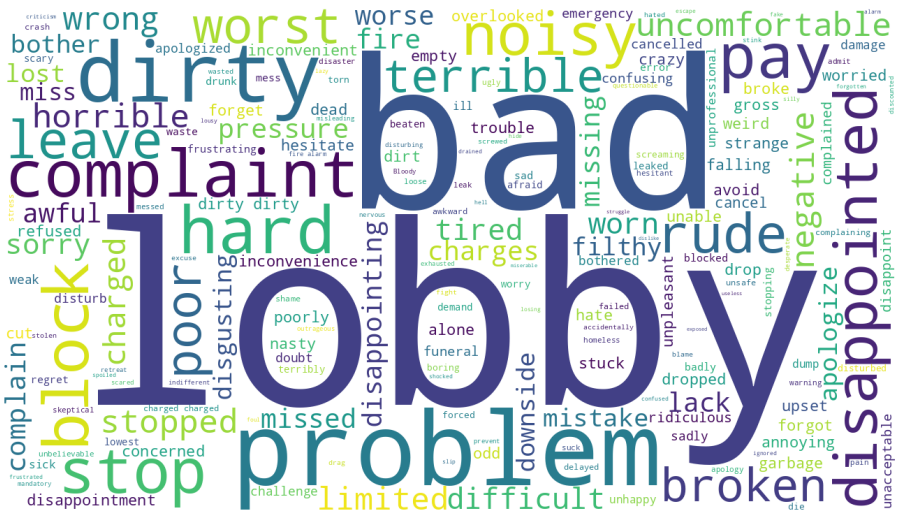

Figure 4: Word cloud (negative tokens)

basis, the polarity of the token determined which score it contributed to. The individual score of the token would then be added to either the net positive or net negative score of the review. To illustrate, the token *friendly* would contribute a positive score of 1, while the token *amazing* would contribute a positive score of 2, because it holds a stronger positive connotation. In addition, the use of punctuation was considered as well—exclamation marks for emphasis contributed to either the positive or negative score, depending on the sentiments of the tokens before and after them.

The positive and negative scores determined the extent to which the review was positive or negative. All positive scores were in the range $[1, 5]$, and the negative scores were in the range $[-5, -1]$.

The hotel reviews have been quantified by sentiment into three distinct categories: positive, neutral and negative. It was observed that, unlike in RQ1, a large proportion of the reviews (70%) are positive, with a small number of generally neutral reviews (17%) and an even smaller number of negative reviews (13%).

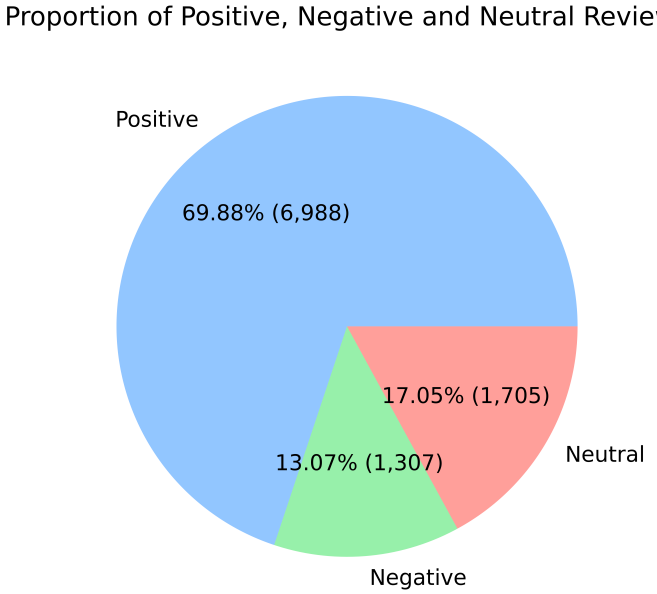

Figure 5: Proportion of reviews (positive v. negative v. neutral)

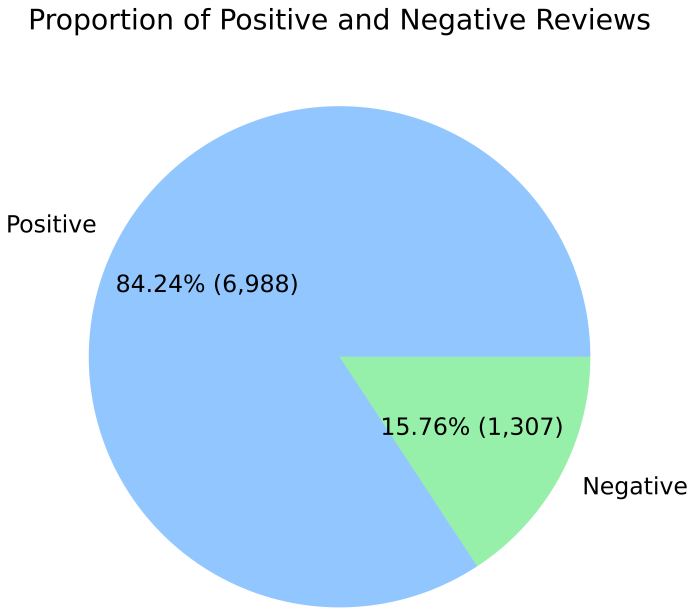

Figure 6: Proportion of reviews (positive v. negative)

5.3 RQ3: Predicting Sentiments By Token Frequency

Using the positive and negative scores SentiStrength had generated for each review, the reviews were classified into three categories: net positive, net neutral and net negative. Only net positive and net negative reviews were considered for this prediction.

A frequency table for each token was generated, which shows how often a token of each net score appears in the entire dataset of hotel reviews. The distribution of reviews (based on score and polarity) is represented in the graph below. The horizontal axis represents the overall sentiment score (a whole number, from −5 to 5) of a review, and the vertical axis represents the count of tokens of that sentiment score.

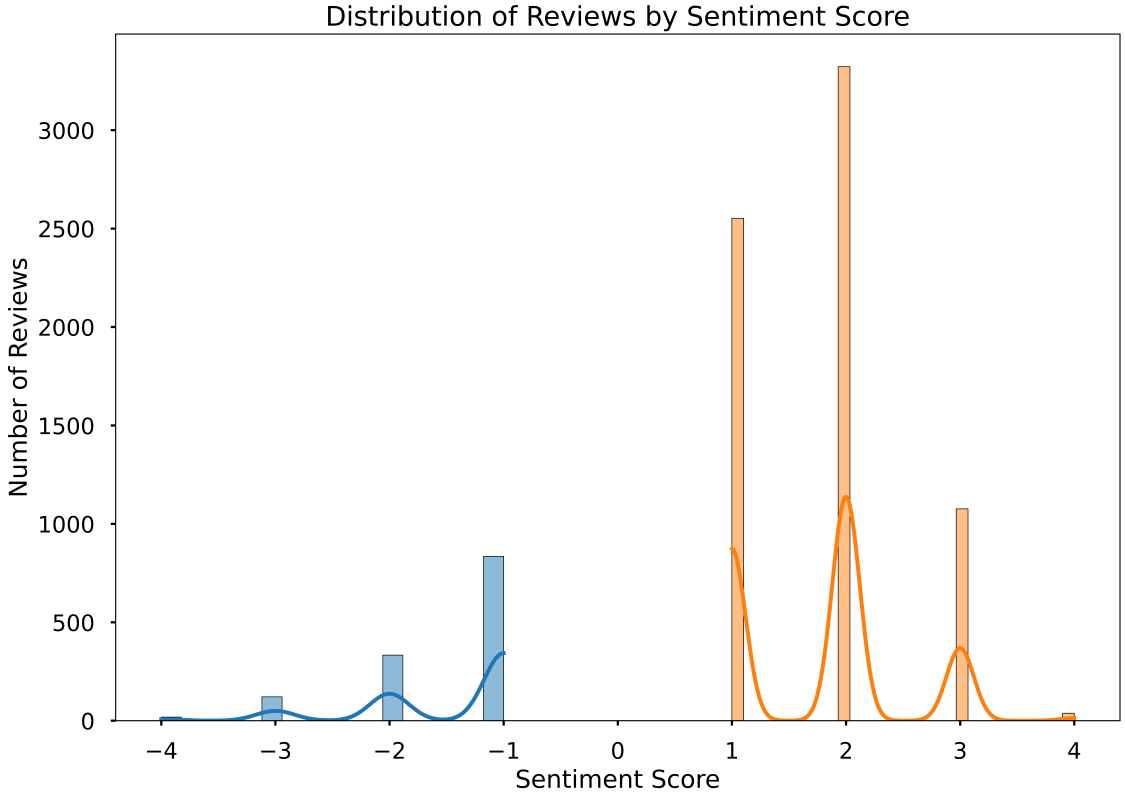

Figure 7: Frequency distribution of tokens by sentiment score

5.3.1 Logistic Regression Model

A **logistic regression model** was constructed to predict whether the sentiment of a hotel review was either positive or negative. A logistic regression model is a type of statistical model used for binary classification tasks, where the goal is to predict the probability that an instance (in this case, a review) belongs to one of two classes— in this case, net positive sentiment and net negative sentiment (1 and −1 respectively).

A precision-recall graph was plotted. The vertical axis represents the precision of the model, defined as the proportion of true positives in all predicted positives, including false positives. The horizontal axis represents recall, defined as the proportion of true positives out of all actual positives, including false negatives. It illustrates the number of positive predictions the model can accurately make.

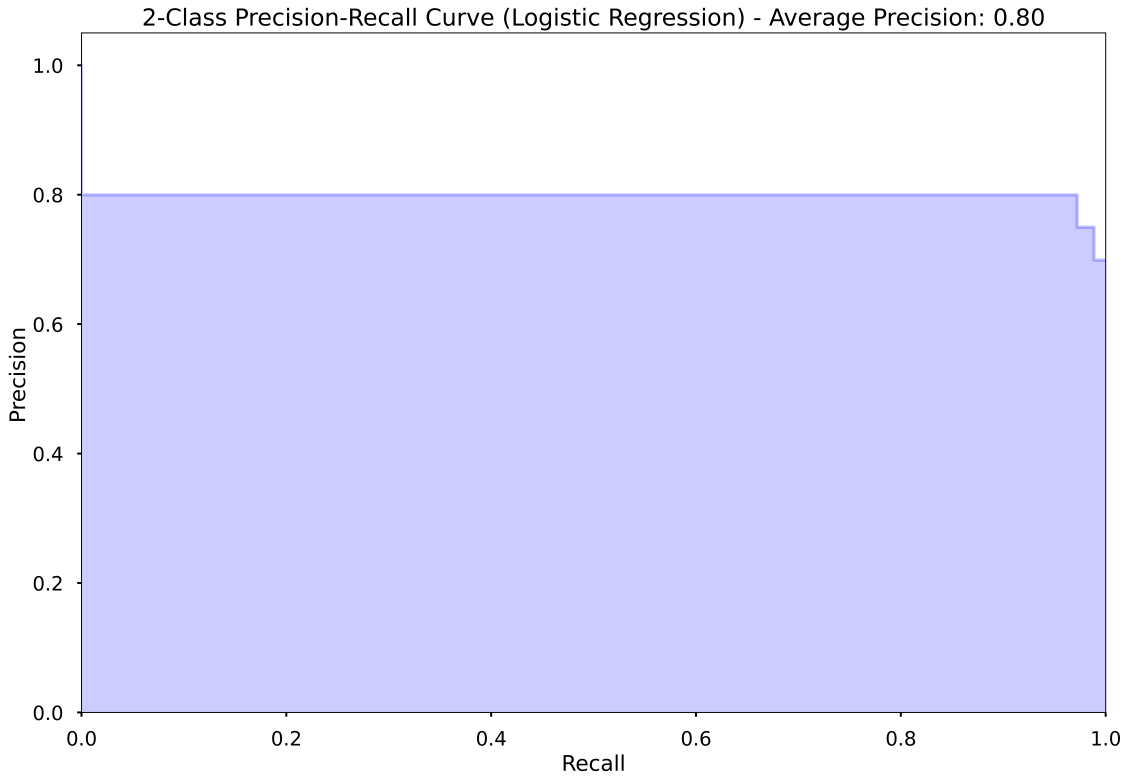

Figure 8: 2-class precision-recall curve of logistic regression model

The average precision of the model is 0.80, hence the model is quite accurate according to statistical norms.

The receiver operating characteristic (ROC) curve of the logistic regression model was also plotted. The vertical axis reflects the rate of true positive predictions being made, and the horizontal axis represents the rate of false positive predictions being made. The higher the ROC curve is positioned over the straight line, the higher the sensitivity of the model—its ability to identify true positive instances—and thus the accuracy of the model.

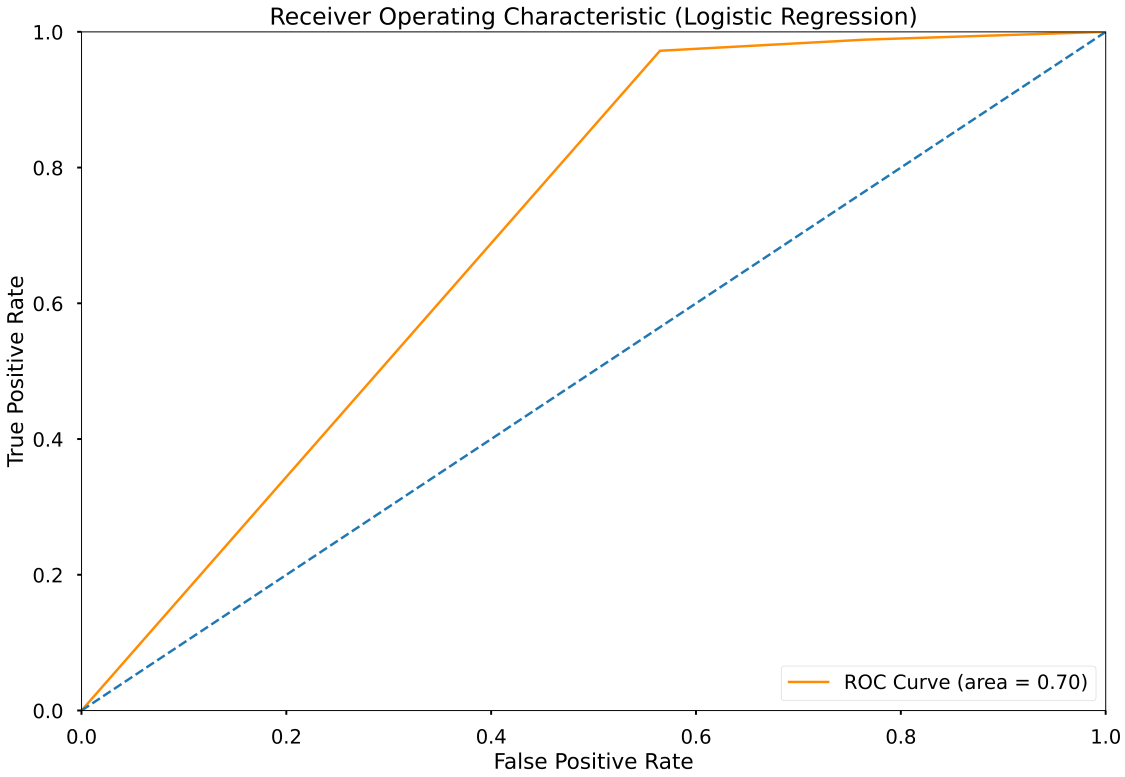

Figure 9: Receiver operating characteristic curve of logistic regression model

The area under the ROC curve (AUC) measures the ability of the model to distinguish between true positive and false positive instances. It ranges from 0 to 1, where 1 indicates the perfect separation of the two classes (true positive and false positive predictions), while 0 indicates that the two classes are indistinguishable for the model. A higher AUC value indicates more accurate model performance; it is able to distinguish between positive and

negative instances to a larger extent. For this ROC curve, the AUC is 0.70, hence the logistic regression model performs well at distinguishing true positive and false positive predictions.

5.3.2 Random Forest Classifier

A **random forest classifier** was also constructed to predict whether hotel reviews carry a net positive or negative sentiment. A random forest classifier relies on feature importance—how important a feature of the dataset, the individual tokens which carry sentiment—are in making future predictions. The model takes random samples from the dataset of hotel reviews, then generates multiple decision trees, after which it consolidates the results from each decision tree. By comparing its predictions with the results obtained, the precision of the model can be determined.

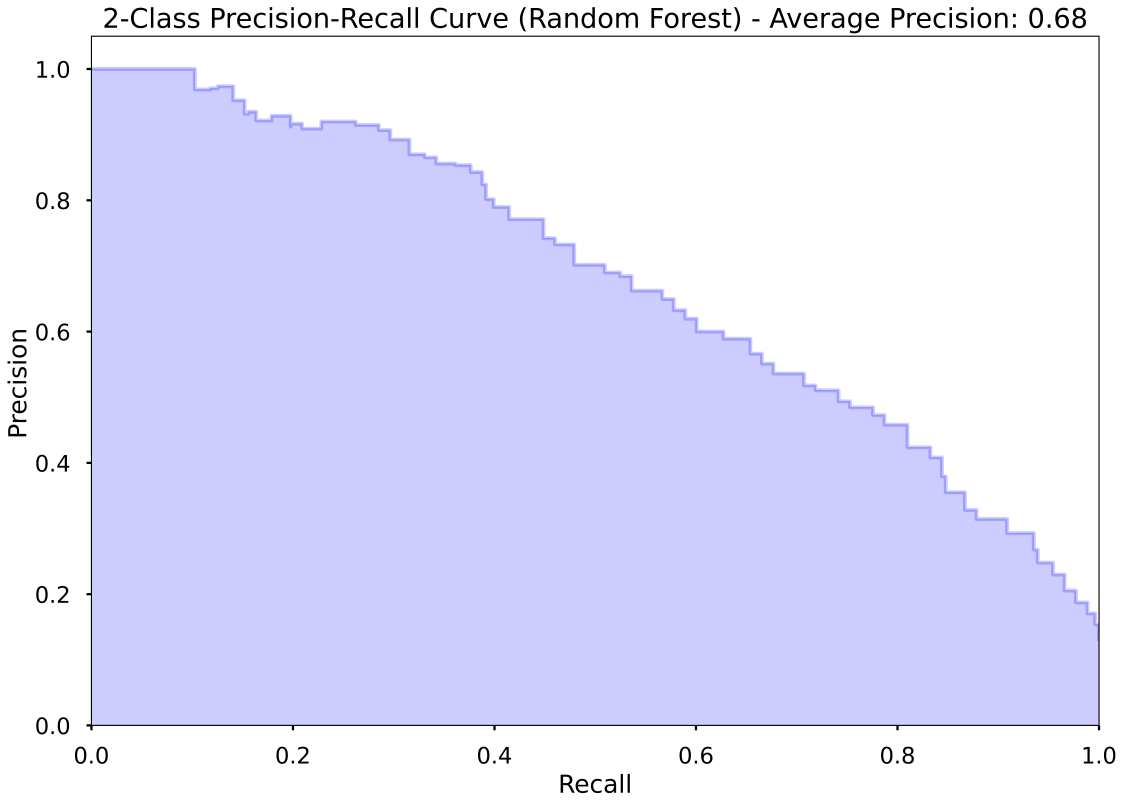

Figure 10: 2-class precision-recall curve of random forest classifier

From the precision-recall curve that was plotted, the average precision of this model is 0.68, thus the model can be said to be quite accurate.

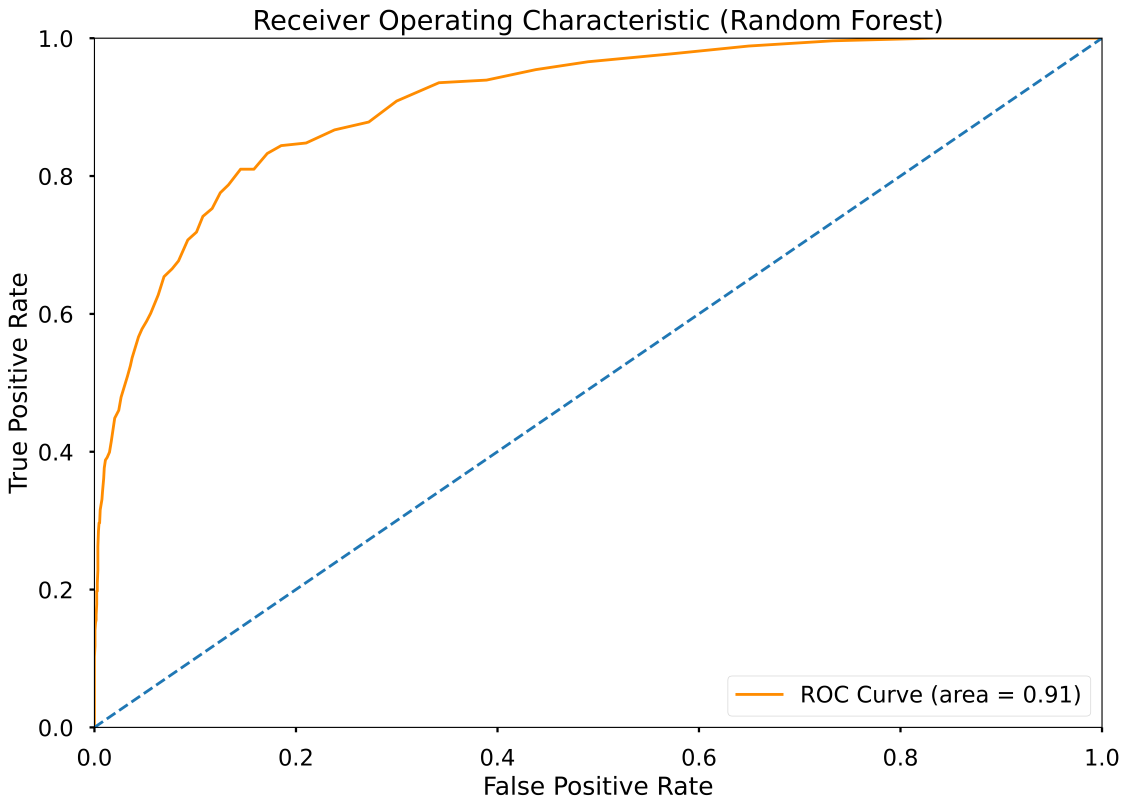

Figure 11: Receiver operating characteristic curve of random forest classifier

For the ROC curve of the random forest classifier, the AUC is 0.90, hence the random forest classifier performs well at distinguishing true and false positive predictions.

6 Discussion and Further Extension

Throughout this project, sentiment analysis was performed on a dataset of international hotel reviews in English. Hotel reviews were tokenised— split into individual words which carry their own meaning—and their sentiments determined using a sentiment lexicon. Although the majority of the lexicon contains neutral tokens, the number of tokens that

carry positive sentiment far exceeds that with negative sentiment. On a larger scale, the sentiments of individual hotel reviews were quantified using SentiStrength with two scores: a positive and a negative sentiment score. The hotel reviews were then classified into three polarities: net positive, net neutral and net negative. A logistic regression model, which relies on a sigmoid function to evaluate the probability of a review being positive, and a random forest classifier, which relies on multiple decision trees created using different random subsets of the data—in this case, the frequency of each token, were created to predict the sentiments of hotel reviews. Upon making predictions, the models were found to have average precisions of 0.80 and 0.68 respectively, and a high rate of success in distinguishing false positive and true positive predictions.

There are a few limitations to this project. Homographs—words that have the same spelling but different meanings—could not be distinguished during tokenisation (RQ1), resulting in slightly unreliable sentiment scores assigned. For example, the word *lobby* was shown to have a negative sentiment, as seen in the word cloud generated for negative tokens, although in context, the word *lobby* was used to refer to a hotel's lobby, rather than a group of people seeking to influence legislators on a particular issue. Furthermore, the data collected was limited only to English international hotel reviews, whereas some of the literature that was referred to managed to expand this to other languages. A more complete analysis of hotel reviews would include tourists of different language backgrounds, and hence sentiment analysis would need to be conducted in more than one language. Lastly, while a random forest classification generally gives more diverse results via random selection of token frequencies, it fails to discover trends that would enable it in extrapolating values that fall outside the training data, since it relies on the average of all the results of its decision trees instead. Furthermore, a logistic regression model assumes that the *tf-idf* values of tokens and the overall sentiment of a review are linearly related, which may hinder the accuracy of the predictions it makes.

Much can be done to extend this project, because of the relatively small scale on which this project was conducted. As above, sentiment analysis could be conducted in more than one language (besides English), to gain a broader understanding of tourists' preferences in hotels from different languages and cultural backgrounds. Furthermore, more prediction models could be constructed. This would create a larger variation in the predictions made, enabling the precision of the models used in this project to be compared with those of other models to evaluate the model with the highest accuracy. Constructing a wider range of prediction models also allows for the possibility of more advanced techniques, including Naïve-Bayes prediction models or Recurrent Neural Networks (RNN), both of which are widely used in analysing sentiment from textual data.

7 References

Borrajo-Millán, F., Alonso-Almeida, M.-d.-M., Escat-Cortes, M., & Yi, L. (2021). Sentiment analysis to measure quality and build sustainability in tourism destinations. *Sustainability*, *13*(11). https://doi.org/10.3390/su13116015

Guzman, E., Azócar, D., & Li, Y. (2014). Sentiment analysis of commit comments in github: An empirical study. https://www.researchgate.net/profile/Emitza_Guzman/publication/266657943_Sentiment_analysis_of_commit_comments_in_GitHub_An_empirical_study/links/5b8305ba4585151fd134f10c/Sentiment-analysis-of-commit-comments-in-GitHub-An-empirical-study.pdf

Khoo, C. S. G., & Johnkhan, S. B. (2017). Lexicon-based sentiment analysis: Comparative evaluation of six sentiment lexicons. *Journal of Information Science*. https://hdl.handle.net/10356/83570

Sohangir, S., Wang, D., Pomeranets, A., & Khoshgoftaar, T. M. (2018). Big data: Deep learning for financial sentiment analysis. *Journal of Big Data*, *5*(3). https://doi.org/10.1186/s40537-017-0111-6

Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., & Kappas, A. (2010). Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*.

TripAdvisor. (2019). 2019 tripadvisor review transparency report. https://www.tripadvisor.co.id/TripAdvisorInsights/wp-content/uploads/2019/09/2147_PR_Content_Transparency_Report_6SEP19_US.pdf

Villavicencio, C., Macrohon, J. J., Inbaraj, X. A., Jeng, J.-H., & Hsieh, J.-G. (2021). Twitter sentiment analysis towards covid-19 vaccines in the philippines using naïve bayes. *Information*, *12*(5), 204. https://doi.org/10.3390/info12050204

8 Appendices

8.1 RQ1 Source Code

```
# imports
from afinn import Afinn
from os import path
import matplotlib as mpl
import matplotlib.pyplot as plt
import nltk as nt
import pandas as pd
import wordcloud as wc

nt.download("punkt")
nt.download("stopwords")

# matplotlib
```

```
plt.style.use("seaborn-v0_8-pastel")

# define some stopwords
stop = nt.corpus.stopwords.words("english")
for i in "$-@_.&+#!*\\(),'\"?:%":
    stop.append(i)
stop.append("n\'t")

# read the data
data = pd.read_csv("./data/datafiniti_reviews.csv",
                   header=0,
                   sep=',',
                   on_bad_lines="skip")

# extract the title and body text of each review into a large list
bodies = data["reviews.text"].astype(str)
titles = data["reviews.title"].astype(str)
"""
remove extraneous words that should not be analysed:
remove '... More' from reviews (if it exists):
        '... More' (captured while web-scraping)
        'Bad', 'Good'
"""
bodies = bodies.str.replace(
    "((Bad|Good):)|(\\.\\.\\. More)",
    "",
    regex=True)

# tokenise, remove stop words and puncutation
bodies_tokens = (bodies.apply(nt.word_tokenize)).apply(
    lambda x: [token for token in x if token.lower() not in stop])

# get a large array of all tokens to be analysed
bodies_tokens_raw = []
for bodies_sentence in bodies_tokens:
    for bodies_token in bodies_sentence:
        bodies_tokens_raw.append(bodies_token)

# create a list of tuples (token, sentiment)
```

```
tokens_sentiments = []

# sentiment analysis starts here.
afn = Afinn()

# rq1: token-based sentiment analysis.
"""loop through the tokens one by one,
assign each word a score, then add it to the list."""
for token in bodies_tokens_raw:
    tokens_sentiments.append(tuple((token, afn.score(token))))
"""filter the sentiment data into three categories:
positive, neutral and negative."""
sentiments_pos, sentiments_neg, sentiments_neu = [], [], []
for token_sentiment in tokens_sentiments:
    if token_sentiment[1] > 0:
        sentiments_pos.append(token_sentiment)
    elif token_sentiment[1] < 0:
        sentiments_neg.append(token_sentiment)
    else:
        sentiments_neu.append(token_sentiment)

# generate a string of positive and negative tokens --
# these will be used for generating the wordclouds.
tokens_pos = "".join(
    token_pos[0] +
    " " for token_pos in sentiments_pos)
tokens_neg = "".join(
    token_neg[0] +
    " " for token_neg in sentiments_neg)

totals_bi = [len(sentiments_pos), len(sentiments_neg)]
totals_tri = [
    len(sentiments_pos),
    len(sentiments_neg),
    len(sentiments_neu)]
total_bi = sum(totals_bi)
total_tri = sum(totals_tri)
labels_bi = ["Positive", "Negative"]
labels_tri = ["Positive", "Negative", "Neutral"]
```

```
# plot a bar graph for bipartite sentiments (+ve, -ve)
figure, axes = plt.subplots()
bars_container = axes.bar(labels_bi, totals_bi)
axes.set_title("Sentiments (Token-Based, Bipartite)")
axes.set_xlabel("Sentiment (Bipartite)")
axes.set_ylabel("Number of Tokens")
axes.bar_label(bars_container, fmt="{:,.0f}")
plt.savefig("./results/rq1/bar_bipartite.png", dpi=600)

# plot a bar graph for tripartite sentiments (+ve, -ve)
figure, axes = plt.subplots()
bars_container = axes.bar(labels_tri, totals_tri)
axes.set_title("Sentiments (Token-Based, Tripartite)")
axes.set_xlabel("Sentiment (Tripartite)")
axes.set_ylabel("Number of Tokens")
axes.bar_label(bars_container, fmt="{:,.0f}")
plt.savefig("./results/rq1/bar_tripartite.png", dpi=600)

# pie chart for bipartite sentiments
fig_pie_bi, ax_pie_bi = plt.subplots()
ax_pie_bi.set_title(
    "Proportion of Tokens by Sentiment; Positive v. Negative")
ax_pie_bi.pie(
    totals_bi,
    labels=labels_bi,
    autopct=lambda pct: "{:.2f}% ({:,.0f})".format(
        pct,
        pct *
        total_bi /
        100),
    shadow=False)

plt.savefig(
    "./results/rq1/pie_bipartite.png",
    dpi=1200,
    bbox_inches='tight')

fig_pie_tri, ax_pie_tri = plt.subplots()
```

```
ax_pie_tri.set_title("""Proportion of Tokens by Sentiment;
                     Positive v. Negative v. Neutral""")
ax_pie_tri.pie(
    totals_tri,
    labels=labels_tri,
    autopct=lambda pct: "{:.2f}% ({:,.0f})".format(
        pct,
        pct *
        total_tri /
        100),
    shadow=False)
plt.savefig(
    "./results/rq1/pie_tripartite.png",
    dpi=1200,
    bbox_inches='tight')

# wordcloud (positive tokens)
wordcloud = wc.WordCloud(background_color="white",
                         mode="RGB",
                         width=1280,
                         height=720)
wordcloud.generate(tokens_pos)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
# plt.title("Word Cloud: Positive Tokens")
# plt.show()
wordcloud.to_file("./results/rq1/wordcloud_pos.png")

# wordcloud (negative tokens)
wordcloud.generate(tokens_neg)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
# plt.title("Word Cloud: Negative Tokens")
# plt.show()
wordcloud.to_file("./results/rq1/wordcloud_neg.png")
```

8.2 RQ2 Source Code

```
# imports
import matplotlib.pyplot as plt
import nltk as nt
import numpy as np
import pandas as pd
import wordcloud as wc

# matplotlib things
plt.style.use("seaborn-v0_8-pastel")

# nt.download('punkt')
# nt.download('stopwords')

# open the file.
df = pd.read_csv("./data/sentistrength_data.csv")
positives, negatives = df["sent.pos"], df["sent.neg"]
nets, polarities = [], []
for row in df.index:
    net_sentiment = positives[row] + negatives[row]
    nets.append(net_sentiment)
    polarity = 2
    if net_sentiment > 0:
        polarity = 1
    elif net_sentiment < 0:
        polarity = -1
    elif net_sentiment == 0:
        polarity = 0
    polarities.append(polarity)

# write the sentiments to a new csv
reviews = pd.read_csv("./data/datafiniti_reviews.csv",
                      header=0,
                      sep=',',
                      on_bad_lines="skip")
combined_data = reviews[["reviews.rating", "reviews.title",
                         "reviews.text"]].copy()
```

```
combined_data.insert(1, value=df["sent.pos"], column="sent.pos")
combined_data.insert(2, value=df["sent.neg"], column="sent.neg")
combined_data.insert(3, value=nets, column="sent.net")
combined_data.insert(4, value=polarities, column="sent.polarity")
combined_data.to_csv("./data/combined_sentiments.csv")

positive_no = sum(pol == 1 for pol in polarities)
neutral_no = sum(pol == 0 for pol in polarities)
negative_no = sum(pol == -1 for pol in polarities)

# print charts and stuff
# tripartite
fig_tri, ax_tri = plt.subplots()
labels_tri = "Positive", "Negative", "Neutral"
fracs_tri = [positive_no, negative_no, neutral_no]
total_tri = sum(fracs_tri)
ax_tri.pie(
    fracs_tri,
    labels=labels_tri,
    autopct=lambda pct: f"{pct:.2f}% ({(pct * total_tri / 100):,.0f})",
    shadow=False)
ax_tri.set_title(
    "Proportion of Positive, Negative and Neutral Reviews")
plt.savefig(
    "./results/rq2/pie_chart_3part.png",
    dpi=1200,
    bbox_inches='tight')
plt.clf()

# bipartite
fig_bi, ax_bi = plt.subplots()
labels_bi = "Positive", "Negative"
fracs_bi = [positive_no, negative_no]
total_bi = positive_no + negative_no
ax_bi.pie(
    fracs_bi,
    labels=labels_bi,
    autopct=lambda pct: f"{pct:.2f}% ({(pct * total_bi / 100):,.0f})",
    shadow=False)
```

```
ax_bi.set_title("Proportion of Positive and Negative Reviews")
plt.savefig(
    "./results/rq2/pie_chart_2part.png",
    dpi=1200,
    bbox_inches='tight')
```

8.3 RQ3 Source Code: Logistic Regression

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import string

import matplotlib.pyplot as plt
# natural language processing
import nltk
import numpy as np
import pandas as pd
import seaborn as sns
from funcsigs import signature
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk import pos_tag
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
# data processing
from sklearn.feature_extraction.text import (CountVectorizer, TfidfTransformer,
                                             TfidfVectorizer)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (auc, average_precision_score,
                             precision_recall_curve, roc_curve)
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline

nltk.data.path.append('/usr/share/nltk_data/')
```

```
# machine learning imports

# matplotlib things
plt.style.use('seaborn-v0_8-poster')


# In[2]:


# import data
df = pd.read_csv(
    './data/combined_sentiments.csv',
    header=0,
    sep=',',
    on_bad_lines='skip')

# lemmatise


def get_wordnet_pos(pos_tag):
    if pos_tag.startswith('J'):
        return wordnet.ADJ
    elif pos_tag.startswith('V'):
        return wordnet.VERB
    elif pos_tag.startswith('N'):
        return wordnet.NOUN
    elif pos_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

# check whether there is a digit or not


def check_digits(text):
    return any(i.isdigit() for i in text)
```

```
# tokenise


def clean_review(review):
    review = str(review)
    review = review.lower()  # turn into lowercase
    review = [word.strip(string.punctuation)
              for word in review.split(' ')]  # remove punctuation
    # remove digits
    review = [word for word in review if not check_digits(word)]

    # remove stop words
    stop = stopwords.words('english')
    review = [token for token in review if token not in stop]
    # remove empty tokens
    review = [token for token in review if len(token) > 0]

    # tag each token with its part of speech (pos)
    pos_tags = pos_tag(review)
    review = [
        WordNetLemmatizer().lemmatize(
            tag[0], get_wordnet_pos(
                tag[1])) for tag in pos_tags]

    # remove words with only one letter
    review = [token for token in review if len(token) > 1]
    review = ' '.join(review)
    return review


# generate a cleaned, tokenised and lemmatised version of the reviews
df['reviews.clean'] = df['reviews.text'].apply(
    lambda x: clean_review(x))

reviews = df["reviews.clean"].values.tolist()
# print(reviews[:6])

"""
* create a list with all the unique words in the whole corpus of reviews.
```

```
* construct a feature vector that contains the counts of how often each word occurs
        in the review.
"""
count_vectoriser = CountVectorizer()
wordbag = count_vectoriser.fit_transform(reviews)
print(count_vectoriser.vocabulary_)


# In[3]:


"""
raw term frequency: frequency of each token
[term frequency-inverse document frequency]
* tf - idf(t, d) = tf(t, d) idf(t, d)
> tf(t, d) is the raw term frequency;
> idf(t, d) is the inverse document frequency: log (n_d / [1 + df(d, t)])
        * n - total number of documents
  * df(t, d) - number of documents where term t appears.
"""


tfidf = TfidfTransformer(use_idf=True,
                         norm='l2',
                         smooth_idf=True)

np.set_printoptions(precision=2)

# feed the tf-idf transformer with our previously created bag
tfidf.fit_transform(wordbag).toarray()


# In[4]:


# split into train, test and validation sets
# X = df['reviews.clean']
X = reviews
y = df['sent.polarity']
```

```
X_t, X_test, y_t, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(
    X_t, y_t, test_size=0.25, random_state=1, stratify=y_t)


# create a parameter grid for the model to pick the best params
paramgrid = [{'vect__ngram_range': [(1, 1)],
              'clf__penalty': ['l1', 'l2'],
              'clf__C': [1.0, 10.0, 100.0]},
             {'vect__ngram_range': [(1, 1)],
              'vect__use_idf':[False],
              'vect__norm':[None],
              'clf__penalty': ['l1', 'l2'],
              'clf__C': [1.0, 10.0, 100.0]},
             ]

# train the model!
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=None)
leftright = Pipeline([('vect', tfidf),
                      ('clf', LogisticRegression(random_state=0))])

gridsearch = GridSearchCV(leftright, paramgrid,
                          scoring='accuracy',
                          cv=5,
                          verbose=1,
                          n_jobs=-1)
gridsearch.fit(X_train, y_train)


# In[5]:


print('best accuracy: %.3f' % gridsearch.best_score_)

clf = gridsearch.best_estimator_
print('accuracy in test: %.3f' % clf.score(X_test, y_test))
```

```
# In[6]:


# make some predictions
# print(type(X_val))
# print(X_val.shape)
# print(X_val.iloc[:,1])
# print(X_val)
preds = clf.predict(X_val)
actuals = y_val.to_numpy()
actuals[actuals == 0] = -1
print(preds[:10], actuals[:10])

false_rate, true_rate, thresholds = roc_curve(
    actuals, preds, pos_label=1)

'''
receiver operating characteristic:
the higher the curve above the diagonal baseline, the better the preds
'''
roc_auc = auc(false_rate, true_rate)

# plot the roc curve
plt.figure(1, figsize=(15, 10))
lw = 2
plt.plot(false_rate, true_rate, color='darkorange',
         lw=lw, label='ROC Curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], lw=lw, linestyle='-')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (Logistic Regression)')
plt.legend(loc="lower right")
plt.show()
plt.savefig(
    "./results/rq3/roc_logreg.png",
```

```
    dpi=800,
    bbox_inches='tight')

# area-under-curve precision-recall
average_precision = average_precision_score(
    actuals, preds, pos_label=1)
precision, recall, _ = precision_recall_curve(actuals, preds)
plt.clf()
step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})

plt.figure(1, figsize=(15, 10))
plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(
    recall,
    precision,
    alpha=0.2,
    color='b',
    **step_kwargs)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title(
    '2-Class Precision-Recall Curve (Logistic Regression) - Average Precision: {0:0.2f}'
    .format(average_precision))
plt.savefig(
    "./results/rq3/prec-recall_logreg.png",
    dpi=800,
    bbox_inches='tight')
```

8.4 RQ3 Source Code: Random Forest Classifier

```
#!/usr/bin/env python
# coding: utf-8

# # RQ3
# > can we predict the relationship between the frequency
# > of tokens of a review and the polarity?
# - independent: token frequency
# - dependent: polarity (positive / negative)

# In[2]:


import string

# import numpy as np
import matplotlib.pyplot as plt
# natural language processing
import nltk
import pandas as pd
import seaborn as sns
from funcsigs import signature
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk import pos_tag
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
# data processing
from sklearn.metrics import (auc, average_precision_score,
                             precision_recall_curve, roc_curve)
from sklearn.model_selection import train_test_split

nltk.data.path.append('/usr/share/nltk_data/')


# machine learning imports
```

```
# matplotlib things
plt.style.use('seaborn-v0_8-poster')


# In[3]:


# import the data
df = pd.read_csv(
    './data/combined_sentiments.csv',
    header=0,
    sep=',',
    on_bad_lines='skip')

# lemmatise


def get_wordnet_pos(pos_tag):
    if pos_tag.startswith('J'):
        return wordnet.ADJ
    elif pos_tag.startswith('V'):
        return wordnet.VERB
    elif pos_tag.startswith('N'):
        return wordnet.NOUN
    elif pos_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

# check whether there is a digit or not


def check_digits(text):
    return any(i.isdigit() for i in text)

# tokenise
```

```
def clean_review(review):
    review = str(review)
    review = review.lower()  # turn into lowercase
    review = [word.strip(string.punctuation)
              for word in review.split(' ')]  # remove punctuation
    # remove digits
    review = [word for word in review if not check_digits(word)]

    # remove stop words
    stop = stopwords.words('english')
    review = [token for token in review if token not in stop]
    # remove empty tokens
    review = [token for token in review if len(token) > 0]

    # tag each token with its part of speech (pos)
    pos_tags = pos_tag(review)
    review = [
        WordNetLemmatizer().lemmatize(
            tag[0], get_wordnet_pos(
                tag[1])) for tag in pos_tags]

    # remove words with only one letter
    review = [token for token in review if len(token) > 1]
    review = ' '.join(review)
    return review


# generate a cleaned, tokenised and lemmatised version of the reviews
df['reviews.clean'] = df['reviews.text'].apply(
    lambda x: clean_review(x))


# In[4]:


# extract vector representations for each review.
documents = [
    TaggedDocument(
        doc, [i]) for i, doc in enumerate(
```

```
            df["reviews.clean"].apply(
                lambda x: x.split(' '))]

# train a doc2vec model
model = Doc2Vec(
    documents,
    vector_size=5,
    window=2,
    min_count=1,
    workers=4)

# transform each document into vec data
df_vec = df['reviews.clean'].apply(
    lambda x: model.infer_vector(
        x.split(' '))).apply(
            pd.Series)
df_vec.columns = ['vec_' + str(x) for x in df_vec.columns]
df = pd.concat([df, df_vec], axis=1)


# In[5]:


# add the term frequency - inverse document frequency values for every
# word
tfidf = TfidfVectorizer(min_df=10)
tfidf_result = tfidf.fit_transform(df['reviews.clean']).toarray()
tfidf_df = pd.DataFrame(
    tfidf_result,
    columns=tfidf.get_feature_names_out())
tfidf_df.columns = ['word_' + str(x) for x in tfidf_df.columns]
tfidf_df.index = df.index
df = pd.concat([df, tfidf_df], axis=1)


# In[6]:


# distribution of sentiments
```

```
# for polar in [-1, 1]: # positive or negative (don't consider
# neutrals)
plt.title('Distribution of Reviews by Sentiment Score')
plt.xlabel('Sentiment Score')
plt.ylabel('Number of Reviews')

subset_neg = df[df['sent.polarity'] == -1]
sns.histplot(subset_neg['sent.net'], label='negative', kde=True)

subset_pos = df[df['sent.polarity'] == 1]
sns.histplot(subset_pos['sent.net'], label='positive', kde=True)
plt.savefig(
    "./results/rq3/distribution.png",
    dpi=1200,
    bbox_inches='tight')
plt.clf()


# In[7]:


# is_bad: True if polarity == -1 else False
df['review.is_bad'] = df['sent.polarity'].apply(lambda x: x == -1)

# feature selection
label = 'review.is_bad'
ignore_cols = [
    label,
    "sent.polarity",
    "sent.pos",
    "sent.neg",
    "sent.net",
    "index",
    "reviews.rating",
    "reviews.clean",
    "reviews.title",
    "reviews.text"]
features = [col for col in df.columns if col not in ignore_cols]
```

```
# split the data into train and test
x_train, x_test, y_train, y_test = train_test_split(
    df[features], df[label], test_size=0.2, random_state=42)


# In[8]:


# train a random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x_train, y_train)

# show feature importance
feature_importances_df = pd.DataFrame(
    {
        'feature': features,
        'importance': rf.feature_importances_}).sort_values(
            'importance',
    ascending=False)
# feature_importances_df.head(20)


# In[9]:


y_pred = [pred[1] for pred in rf.predict_proba(x_test)]
# false +ve rate, true +ve rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred, pos_label=1)

'''
receiver operating characteristic:
the higher the curve above the diagonal baseline, the better the preds
'''
roc_auc = auc(fpr, tpr)

# plot the roc curve
plt.figure(1, figsize=(15, 10))
lw = 2
plt.plot(fpr, tpr, color='darkorange',
```

```
         lw=lw, label='ROC Curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], lw=lw, linestyle='-')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (Random Forest)')
plt.legend(loc="lower right")
plt.show()
plt.savefig("./results/rq3/roc_rf.png", dpi=1200, bbox_inches='tight')


# In[10]:


# area-under-curve precision-recall
average_precision = average_precision_score(y_test, y_pred)

precision, recall, _ = precision_recall_curve(y_test, y_pred)
plt.clf()

step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})

plt.figure(1, figsize=(15, 10))
plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(
    recall,
    precision,
    alpha=0.2,
    color='b',
    **step_kwargs)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
```

```
plt.title(
    '2-Class Precision-Recall Curve (Random Forest) - Average Precision: {0:0.2f}'
    .format(average_precision))
plt.savefig(
    "./results/rq3/prec-recall_rf.png",
    dpi=1200,
    bbox_inches='tight')
```
