Sentiment Analysis for Imbalanced Hotel Reviews Dataset Using Machine Learning Techniques

Yuye Sheng
University of Waterloo
Waterloo, ON, Canada
y39sheng@uwaterloo.ca

***Abstract*—Text classification has been one of the extensively used tasks in Natural Language Processing (NLP). In this paper, we conducted a document-level sentiment classification based on hotel reviews data obtained from Booking. Since the dataset is highly imbalanced, we first applied randomly under-sampling on the training data. Then Naïve Bayes, Support Vector Machine (SVM), Convolutional Neural Network (CNN) and Gated Recurrent Unit (GRU) were developed and evaluated on the full dataset. The data was preprocessed using TF-IDF and pre-trained word embedding model respectively for non-deep and deep learning methods. We assessed the performance of these models and found that GRU achieved the best performance with classification accuracy 88.54% and in general deep learning methods outperformed non-deep learning methods.**

***Keywords*—Natural Language Processing, Binary Text Classification, Naïve Bayes, SVM, CNN, GRU**

I. Introduction

Sentiment analysis is the field of study that analyzes people's opinions, sentiments, evaluations, attitudes and emotions towards entities such as products, services and so on, and it mainly focuses on opinions which express or imply positive or negative sentiments [1]. With the opinions gathered from a large amount of reviews or feedback, the organizations are able to improve their products or services in order to grow their business. Individual customers also benefit from other people's opinions since they could evaluate products based on those reviews and then make an informed decision, and this would increase the chance of having a positive customer experience.

With the explosive growth of e-commerce, people have access to many review websites for decision making. For example, they could buy products on Amazon, look for restaurants on Yelp and reserve accommodations on Booking. However, finding and monitoring those opinion sites and distilling the information contained in them remains a formidable task because of the proliferation of diverse sites. The average human reader will have difficulty identifying relevant sites and extracting and summarizing the opinions in them [1]. Therefore automated sentiment analysis systems are needed.

For travellers, sites like Booking have been reliable to book hotels which they prefer to stay in. In this paper, sentiment analysis on hotel reviews collected from Booking [2] is conducted using non-deep learning techniques (Naïve Bayes and SVM) and deep learning techniques (CNN and GRU). More specifically, these techniques classify the review texts into two categories, positive and negative. This paper demonstrates every step of building four sentiment classifiers and compares the results among these classifiers.

The remainder of this paper is organized as follows. "Related Work" section provides an overview of previous work related to the above techniques in text classification. "Methodology" section provides a detailed description of the dataset, data preprocessing, feature extraction procedure, and machine learning techniques. "Results" section presents details on handling imbalanced data and the results for each model along with its specific architecture. Results comparison and potential considerations are provided in "Discussion" section. Finally, conclusions and future work are contained in "Conclusion" section.

II. Related work

Pang et al. [3] proposed sentiment classification using Naïve Bayes, Maximum Entropy and Support Vector Machine on unigrams and bigrams of movie reviews data in 2002. They reported that SVM paired with unigrams performed best with 82.9% accuracy while Naïve Bayes achieved 81% accuracy on this binary classification problem.

Isa et al. [4] utilized the Bayes formula to vectorize a document according to a probability distribution based on keywords reflecting the probable categories that the document may belong to. Using this way to represent the document, the text classification algorithms based on the vector space model such as SVM, can then be used to classify the documents on a multi-dimensional level, thus improving on the results obtained using only the highest probability to classify the document, such as Naïve Bayes Classifier.

Kim [5] investigated using convolutional neural networks (CNN) for sentence-level text classification, which built on top of pre-trained word vectors. A simple CNN with one layer of convolution and little hyperparameter tuning achieved competitive performance.

Guan et al. [6] presented a novel deep learning framework named Weakly-supervised Deep Embedding (WDE) and employed a weakly-supervised CNN to classify customer reviews obtained from Amazon, which outperformed baseline methods with 87% accuracy. In WDE, a high level embedding space is learned first which captured the overall distribution of sentences in terms of available review ratings and then a classification layer is added on the top of the embedding layer with supervised fine-tuning using labeled sentences.

Zhang and Wallace [7] conducted a sensitivity analysis of one-layer CNN to explore the effect of architecture components on model performance. They summarized their empirical findings and provided practical guidance for tuning hyperparameters, including the filter *region size*, regulatization parameters and so on.

Biswas et al. [8] showed the effectiveness of GRU by comparing with traditional bag of features models which disregard the order of the words. They fine-tuned the GRU-net through extensive cross-validation testing and verified that GRUs are faster in convergence and have lesser computational overhead than LSTM.

Zulqarnain et al. [9] addressed the issue of efficiently analyzing the document-level text classification approach based on GRU. They proposed unified structure to investigate the effects of word embedding and GRU on two benchmark datasets (Google snippets and TREC) and found that GRU is a suitable model for sequential data of text classification with accuracy 84.8% and 95.2% for two datasets, especially when there are more amounts of learning data because it can effectively capture long sequence data.

III. Methodology

A. Data Preprocessing

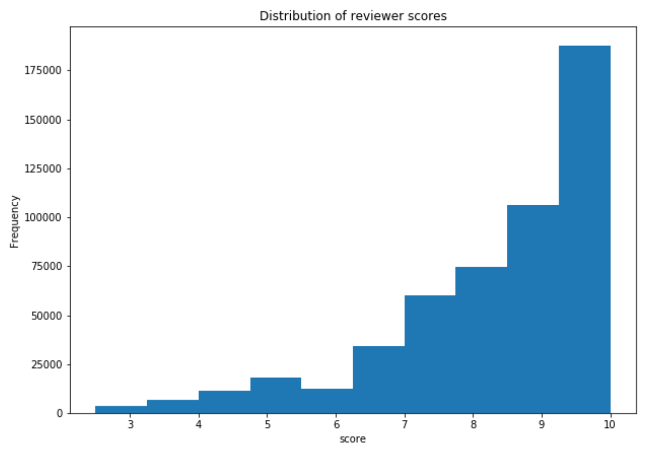

Fig. 1. Distribution of reviewer scores

The dataset contains 515,738 customer reviews and scoring of 1,493 luxury hotels across Europe [2]. There are 17 features but only review texts and scores are needed in this project. The customers gave positive and negative opinions separately in one record of reviews, so we combined those positive text and negative text together as the raw text data for each observation. The next step is to create labels, which indicates whether in general the customer is satisfied with the hotel services or not. The overall ratings provided by reviewers range from 2.5 to 10, and the distribution of those ratings is shown in Fig. 1. To simplify the problem, we assume that a review is positive if the overall score is over 6. Note that 89.77% reviews are recognized as positive and only 10.23% are assumed to be negative. In other words, our data is imbalanced and would bias the model towards positive class, thus metrics like confusion matrix, precision and recall and methods like under-sampling should be applied in order to avoid the effect of imbalance.

Fig. 2 depicts the key words in all of the reviews. Many words are related to the hotel like room, floor, window, staff and attitude towards experience such as ok and great. Note that the word cloud only gives us a glimpse of the most common words appearing in the reviews, not the most important words in classification.

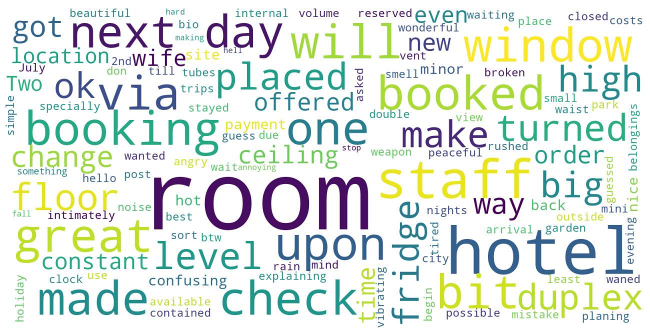

Fig. 2. Word cloud

B. Feature Extraction

Now we had all the review texts and their corresponding labels. Feature extraction was performed in the next stage. We tokenized each review text, removed the stopwords and converted textual reviews into feature vectors according to words in each review.

A simple method of feature extraction is bag-of-words. In the bag-of-words model, the frequency of each word in a document is used as a feature. The result was shown in a sparse matrix where each column represents a word (i.e., a feature) and each row represents a review (i.e., a document). The values in each row are the values of the corresponding feature vector which calculated by counting the number of occurrence of every word in that review.

However, we used TF-IDF (Term Frequency - Inverse Document Frequency) weighting scheme to extract feature importance in this project. It converted bags of words into fixed-length feature vectors (TF) and scaled those features (IDF). The most frequent words are not necessarily relevant words, for example stopwords like “the” and “a” might occur very often but they are meaningless. TF-IDF solves this problem and is intended to determine how relevant a given word is in a specific document by taking into account the document frequency. The TF-IDF score of any word in any document can be represented as per the following equation [10]:

$$TFIDF(word, doc) = TF(word, doc) * IDF(word)$$

where

$$TF(word, doc) = \frac{Frequencyofword \in thedoc}{No.ofwords \in thedoc}$$

$$IDF(word) = log_e(1 + \frac{No.ofdocs}{No.ofdocswithword})$$

In TF-IDF model, the documents are also represented as vectors but the values of the matrix, instead of the raw counts, are the TF-IDF scores for each of the words.

Spark is used for tokenization, TF-IDF transformation, and non-deep leaning models in the following sections.

C. Naïve Bayes

Naïve Bayes refers to the construction of a Bayesian probabilistic model that assigns a posterior class probability to an instance: $P(Y = y|X = \mathbf{x})$ where y represents the classes and $\mathbf{x} = (x_1, x_2, ..., x_n)$ represents n features [11]. We only have positive and negative instances in the dataset, so $y \in \{0, 1\}$. The Naïve Bayes classifier uses these probabilities to assign an instance to a class.

In order to compute $P(Y|\mathbf{X})$, Bayes' theorem is applied:

$$P(Y|\mathbf{X}) = \frac{P(\mathbf{X}|Y)P(Y)}{P(\mathbf{X})} \tag{1}$$

$P(Y)$, $P(\mathbf{X}|Y)$ and $P(\mathbf{X})$ can be estimated from the training data. Let us assume that the feature x_i are independent from each other. This is a strong assumption, which is clearly violated in most practical applications and is therefore *naive*—hence the name [11]. With this assumption, the numerator in (1), which is actually the joint probability of $\mathbf{X}$ and Y, can be rewritten as:

$$\begin{aligned} &P(\mathbf{X}|Y = y)P(Y = y) \\ &= P(x_1|x_2, x_3, ..., x_n, y)P(x_2|x_3, x_4..., x_n, y)...P(x_n|y)P(y) \\ &= P(x_1|y)P(x_2|y)...P(x_n|y)P(y) \\ &= \prod_{i=1}^{n} P(x_i|y)P(y) \end{aligned}$$

The denominator $P(\mathbf{X})$ acts as a scaling factor and ensures that the posterior probability $P(Y|\mathbf{X})$ is properly scaled (i.e., a number between 0 and 1) [11]. Last but not least, the predicted class $\hat{y}$ can be constructed as

$$\hat{y} = argmax_y \prod_{i=1}^{n} P(x_i|y)P(y) \tag{2}$$

This is also known as the *maximum a posteriori* (MAP) rule. A (simple) Naïve Bayes classifier implements (2).

Note that if we are interested in the posterior probability $P(Y|\mathbf{X})$, (1) can be rewritten as

$$P(Y|\mathbf{X}) = \frac{\prod_{i=1}^{n} P(x_i|y)P(y)}{\prod_{i=1}^{n} P(x_i|y)P(y) + \prod_{i=1}^{n} P(x_i|y^c)P(y^c)} \tag{3}$$

so that it is able to compute $P(Y|\mathbf{X})$.

Naïve Bayes works well on numeric and textual data, and is easy to implement and compute comparing with other algorithms, however the independence assumption is violated by real-world data and performs very poorly when features are highly correlated and does not consider frequency of word occurrences [12].

D. Support Vector Machine

Support Vector Machine (SVM) is one of the typical discriminative classifiers. It needs both positive and negative training data which are uncommon for other classification methods. SVM uses those data to seek for the decision surface that best separates the positive from the negative data in the high dimensional space, so called the hyperplane. The document representatives which are closest to the decision surface are called the support vector [12], as shown in Fig. 3 [13, fig. 1].

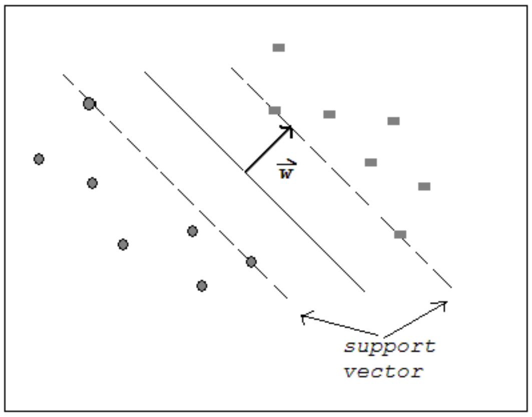

Fig. 3. Example of SVM hyperplane pattern

The linear hyperplane can be represented by $\mathbf{wx} + b = 0$ where $\mathbf{w}$ is a vector normal to the hyperplane and $\mathbf{x} = (x_1, x_2, ..., x_n)$ represents n features. SVM tries to find the optimal hyperplane satisfying

$$y_i(\mathbf{wx}_i + b) > 0$$

for all $(\mathbf{x}_i, y_i)$ in the dataset. Note that $y_i \in \{-1, 1\}$ but not $\{0,\ 1\}$ in this case. After finding the optimal $\mathbf{w}$ and b using training set, the predicted class $\hat{y}$ for a new $\mathbf{x}$ is computed as:

$$\hat{y} = sgn(\mathbf{wx} + b)$$

where $sgn()$ is the sign function. The main disadvantage of SVM is the relatively complex training and categorizing algorithms and also the high time and memory consumptions during training and prediction stage [12].

E. Word Embedding for deep learning

Word embedding is a set of feature learning techniques that map words into numeric vectors so that machine can understand it. We utilized it as feature extraction for deep learning methods. In Keras, review texts are tokenized by assigning each word an integer and converting the texts into sequences of integers. The length of the sequence is the number of words that review contains, which means sequences have different length so padding is needed to enforce all the sequences are of a fixed length.

In feature extraction for non-deep learning methods, each document is represented as a vector, but here, each word is represented as a vector and each document is a matrix with each row being a vector of the corresponding word. The advantage of word embedding is that it takes into account the relationship between words. Words that have similar meanings would have similar vectors, which indicates that they are closer to each other in the vector space. Therefore, the distance between vectors can represent the relationship between words.

Researchers can either use an embedding layer in the neural network to train the specific word embedding or create a word embedding with respect to the corpus using a pre-trained word embedding model. The pre-trained models usually have trained enormous amount of words into high-dimensional vectors and we only need to extract vectors of those words appearing in our own dataset.

Since our training data is sufficiently large, we used a pre-trained word embedding model GloVe to save computation cost and reduce training time. The word vectors have dimensionality of 100 and were trained on the nonzero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus [14]. Words not present in the set of pre-trained words were initialized as zero vectors.

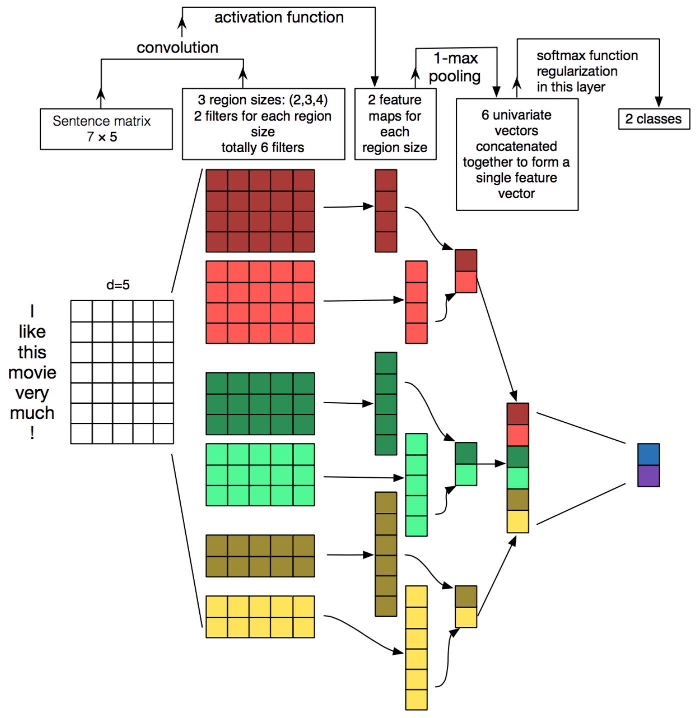
activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Fig. 4. Illustration of a CNN architecture for sentence classification

F. Convolutional Neural Network

Convolutional neural network (CNN) is a deep learning algorithm which performs image processing tasks successfully. The most important layers in CNN include convolutional layer, pooling layer and fully-connected layer. Fig. 4 illustrates an architecture for sentence classification [7, fig. 1].

In word embedding, a sentence was encoded into a matrix with shape $s \times d$ where s is the length of the sentence and d is the dimensionality of the word vectors. Then the sentence matrix can be treated as an "image" and the convolutional layer would apply a set of filters on it. Because rows represent discrete symbols (namely, words), it is reasonable to use filters with widths equal to the dimensionality of the word vectors (i.e., d). Thus we can simply vary the "height" of the filter, i.e., the number of adjacent rows considered jointly and the height is referred to the *region size* of the filter. After applying an activation function, the output dimensionality of the feature map generated by each filter will vary as a function of the sentence length and the filter region size. Therefore, a pooling layer is added to each feature map to induce a fixed-length vector. In Fig. 4, a 1-max pooling, which extracts a scalar from each feature map, is applied. Finally, we need a fully-connected layer to concatenate the outputs generated from each filter map into a vector, which is then fed through a softmax function to generate the final classification [7].

G. Bidirectional GRU

Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) are two types of Recurrent Neural Networks (RNN). RNN can interpret sequential information by taking the input and reusing the output of previous nodes or later nodes in the sequence to generate the final output. In other words, RNNs consider the order of words which is important to the meaning of sentences and can keep track of long-term dependencies.

LSTM was proposed in 1997 [15] and GRU is quite recent development proposed by Cho et al. [16] in 2014. We chose GRU rather than LSTM because GRUs are much simpler in structure than LSTM and it performs better in terms of accuracy and error rates [9]. Fig. 5 shows how GRU works for text classification [9, fig. 3].

GRU has gating units that control the flow of information inside the unit. It calculates two gates called update and reset gates. The update gate is calculated from the current input and the hidden state of previous time step and this gate handles how much of portions of new memory and old memory should be combine in the final memory. Similarly the reset gate is computed but it manages the balance among previous memory and the new input information in the new memory [9].

Sutskever et al. [17] claimed that the performance of a RNN can be improved by reversing the training sequence in some cases. Thus we attempted to model the sequences in both directions by implementing a bidirectional structure directly within the GRU architecture.

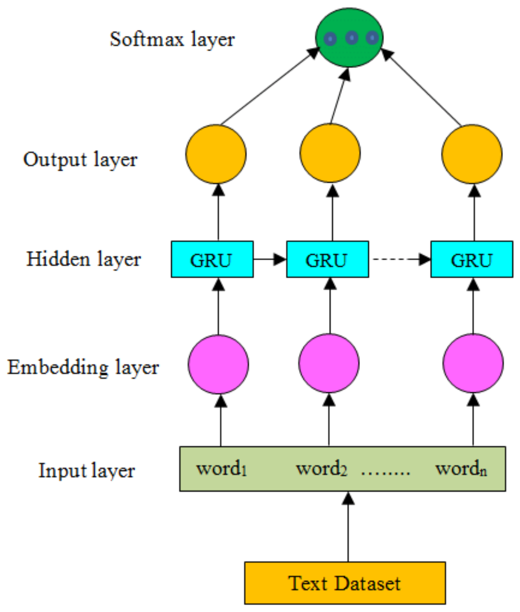

Fig. 5. The proposed GRU-Embedding base architecture for text classification

TABLE I
NAÏVE BAYES CONFUSION MATRIX FOR TRAINING SET AND TEST SET

		Predicted	
		Negative	Positive
True	Negative	34762	7452
	Positive	68126	302286

		Predicted	
		Negative	Positive
True	Negative	8157	2389
	Positive	17463	75103

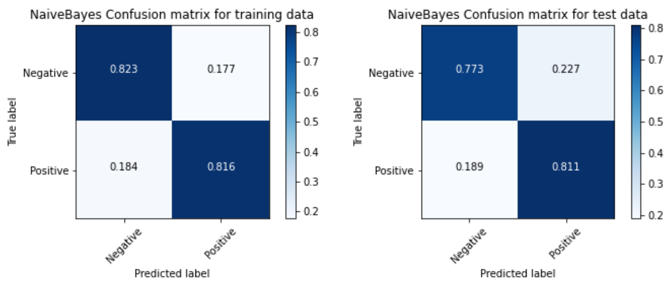

Fig. 6. Normalized confusion matrix for Naïve Bayes

IV. RESULTS

For all four methods, we split 80% of the dataset as the raw training data and the rest 20% as test data. As mentioned in III-A, 89.77% reviews are recognized as positive. We had attempted to train the models using the imbalanced training set, but these models predicted positive class very well (i.e., about 99% positive reviews were classified correctly) but negative class very bad (i.e., about 30% negative reviews were classified correctly). To avoid these bad models, we randomly selected 15% positive reviews and combined it with all negative reviews in the raw training data as our final training set. Now the training set was no longer imbalanced. Note that the test set was not touched during the under-sampling procedure and hence still imbalanced.

The balanced training set was used to train models but confusion matrices were constructed on the raw imbalanced training data and test data. That is, we trained models based on approximately 97,000 reviews with roughly half positive and half negative, and created the training confusion matrices on about 412,000 reviews of raw training data, which including 97,000 balanced training data, and the test confusion matrices on about 103,000 reviews of test set.

A. Naïve Bayes

The Naïve Bayes model was built on the pyspark.ml package with parameter *smoothing* of 1.0 and type of multinomial. Table I reports the confusion matrix for raw training data and test data seperately, and Fig. 6 shows the normalized confusion matrix.

The model predicted both classes well, while it classified positive class slightly better than negative class.

B. Support Vector Machine

We implemented a linear L2-regularization SVM with regularization parameter 0.1 and set max number of iterations to be 10.

As seen in Fig. 7, about 87.7% of positive reviews classified correctly while 73.8% of negative reviews predicted correctly. Compared to Naïve Bayes, SVM reported higher accuracy predicting positive class at the cost of lower accuracy predicting negative class.

TABLE II
SVM CONFUSION MATRIX FOR TRAINING SET AND TEST SET

		Predicted	
		Negative	Positive
True	Negative	34349	7865
	Positive	43315	327097

		Predicted	
		Negative	Positive
True	Negative	7779	2767
	Positive	11422	81144

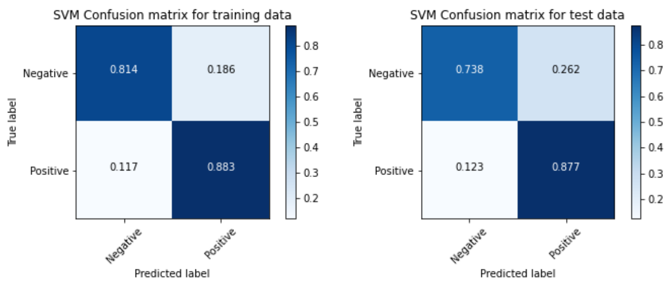

Fig. 7. Normalized confusion matrix for SVM

C. Convolutional Neural Network

Our CNN model was built on Keras. It is a simple one-layer CNN. After adding the word embedding layer, a 1D convolutional layer with 256 filters and kernel size (i.e., *region size*) of 5 for each filter is added. The activation function is Rectified Linear Unit (ReLU), which is commonly used. Then we add a global max pooling layer followed by a fully-connected layer with 32 hidden units and dropout rate of 0.5 to prevent overfitting. At last, the sigmoid function is applied to generate the binary classification. With respect to model fitting, we used binary cross entropy as loss function and Adam as optimizer. The best results were got with batch size of 128 and epochs of 9.

As elaborated in Fig. 8, CNN outperformed SVM on both classes with test accuracy 88.9% for positive reviews and 78.3% for negative reviews.

TABLE III
CNN CONFUSION MATRIX FOR TRAINING SET AND TEST SET

		Predicted	
		Negative	Positive
True	Negative	35340	6863
	Positive	38665	331722

		Predicted	
		Negative	Positive
True	Negative	8264	2293
	Positive	10298	82293

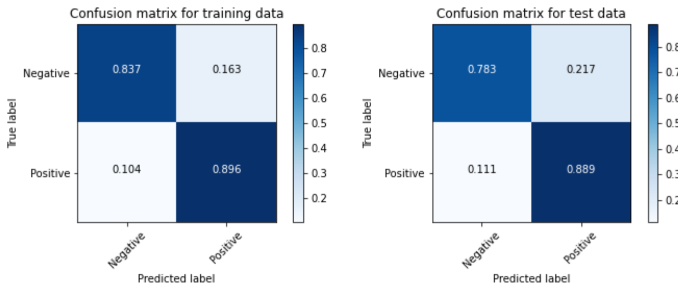

Fig. 8. Normalized confusion matrix for CNN

D. Bidirectional GRU

The GRU model consists of the word embedding layer which is same as CNN, a bidirectional GRU layer with 128 units per direction and drop out rate of 0.5, followed by a sigmoid layer. Parameters for model fitting are identical to CNN model. We obtained the best results with batch size of 128 and epochs of 10.

GRU accomplished 89.6% test accuracy for positive classes and 79.3% for negative classes as shown in Fig. 9, which were both better than that in CNN though those values were close to each other.

TABLE V
GRU CONFUSION MATRIX FOR TRAINING SET AND TEST SET

		Predicted	
		Negative	Positive
True	Negative	34104	8099
	Positive	37805	332582

		Predicted	
		Negative	Positive
True	Negative	8368	2189
	Positive	9632	82959

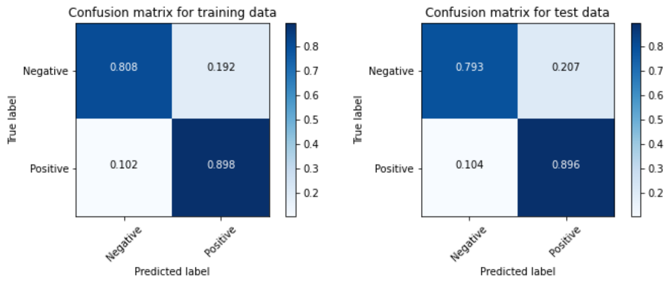

Fig. 9. Normalized confusion matrix for GRU

V. DISCUSSION

Given the confusion matrix, Precision, Recall and F1 Score are calculated as follows:

$$Precision = \frac{True\ Positive}{Total\ Predicted\ Positive}$$

$$Recall = \frac{True\ Positive}{Total\ Actual\ Positive}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

These scores along with accuracy for above methods have been given in Table IV.

In general, GRU outperformed all other models since it achieved the highest scores of all metrics for test set. CNN also performed well on the dataset and obtained very similar results as GRU. Moreover, Naïve Bayes had lowest scores of

TABLE IV
PERFORMANCE OF FOUR METHODS

Methods	**Precision**		**Recall**		**F1 Score**		**Accuracy**	
	train	**test**	**train**	**test**	**train**	**test**	**train**	**test**
Naïve Bayes	0.9759	0.9692	0.8113	0.8636	0.8889	0.8833	0.8168	0.8075
SVM	0.9765	0.9670	0.8831	0.8766	0.9274	0.9196	0.8760	0.8624
CNN	**0.9797**	0.9729	0.8956	0.8888	**0.9357**	0.9289	**0.8896**	0.8779
Bi-GRU	0.9762	**0.9743**	**0.8979**	**0.8960**	0.9354	**0.9335**	0.8887	**0.8854**

all metrics, and deep learning models seemed to have better performance than non-deep learning models.

The deep learning models were expected to work better, but the results indicated that they got similar results as SVM. One possible reason is, in word embedding, only 52.64% of the tokens in our corpus are covered by the pre-trained word embedding model. This is probably because the raw review texts have so many misspelled words. We did not train our own word embedding because of the time and computation cost, but the neural networks might perform better if we do so. On the other hand, since misspelled words may contain important information, it is better to take them into account instead of just ignoring them, for example attempting spelling corrections in data preprocessing stage.

In terms of dealing with imbalanced data, we operated randomly under-sampling on this dataset. Although it worked in this case, under-sampling has two major disadvantages. One is that some potentially important information can be discarded since only 15% of the positive data were selected in order to balance the training set. Another one is that the sample might be biased, which means it might not be able to represent the population. Other techniques such as gradient boosting would be considered to solve this problem.

VI. Conclusion

In this paper, we conducted sentiment analysis using four machine learning techniques on the hotel reviews dataset [2]. We conclude that GRU achieved superior performance compared to the other three models and deep learning techniques generally ourperformed non-deep learning techniques. As for future work, we will explore more proper ways to handle imablanced dataset and attempt other machine learning techniques to evaluate the effectiveness of different methods.

References

[1] B. Liu, "Sentiment Analysis and Opinion Mining," Synthesis Lectures on Human Language Technologies, 2012.

[2] J. Liu, "515K Hotel Reviews Data in Europe," August 2017. [Dataset]. Available at: https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe/version/1. [Accessed: February 2, 2020].

[3] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," in Proc. of Conf. on Empirical Methods in Natural Language Processing, pp. 79-86, 2002.

[4] D. Isa, L. H. Lee, V. P. Kallimani, R. RajKumar, "Text Documents Preprocessing with the Bayes Formula for Classification using the Support vector machine," IEEE, Traction of Knowledge and Data Engineering, Vol. 20, No. 9, pp. 1264-1272, 2008.

[5] Y. Kim, "Convolutional Neural Networks for Sentence Classification," Sep 2014, arXiv:1408.5882v2 [cs.CL].

[6] Z. Guan, L. Chen, W. Zhao, Y. Zheng, S. Tan, D. Cai, "Weakly-supervised deep learning for customer review sentiment classification," in Proc. IJCAI, 2016, pp. 3719–3725.

[7] Y. Zhang, B. Wallace, "A sensitivity analysis of (and practitioners' Guide to) convolutional neural networks for sentence classification," October 2015, arXiv:1510.03820 [cs.CL].

[8] S. Biswas, E. Chadda, F. Ahmad, "Sentiment Analysis with Gated Recurrent Units," Advances in Computer Science and Information Technology (ACSIT) Vol. 2, No. 11, pp. 59-63, 2015.

[9] M. Zulqarnain, R. Ghazali, M. G. Ghouse, and M. F. Mushtaq, "Efficient Processing of GRU Based on Word Embedding for Text Classification," JOIV : International Journal on Informatics Visualization, Vol. 3, No. 4, pp. 377-383, November 2019.

[10] B. Das and S. Chakraborty, "An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation," June 2018, arXiv:1806.06407 [cs.CL].

[11] D. Berrar, "Bayes' Theorem and Naive Bayes Classifier," 2018, doi: 10.1016/B978-0-12-809633-8.20473-1.

[12] B. Baharudin, L. H. Lee, and K. Khan, "A Review of Machine Learning Algorithms for Text-Documents Classification," J. Adv. Inf. Technol., Vol. 1, No. 1, pp. 4-20, 2010.

[13] A. Basu, C. Watters, and M. Shepherd, "Support Vector Machines for Text Categorization," in 36th Annual Hawaii International Conference on System Sciences, pp. 7, 2003, doi:10.1109/HICSS.2003.1174243.

[14] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," 2014.

[15] S. Hochreiter and J. Schmidhuber, "Long Short Term Memory," Neural Computation, Vol. 9, No. 8, pp. 1-32, 1997.

[16] K. Cho, B. v. Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014, arXiv:1409.1259 [cs.CL].

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014, arXiv:1409.3215 [cs.CL].