Data Science Project Final Report

Miaomiao Wang
STUDENT NO.A1832767

April 25, 2023

Report submitted for **Master of Data Science (Research/Industry Project B)** at the School of Mathematical Sciences, University of Adelaide

Project Area: **Data science**
Project Supervisor: **Jono Tuke**

Abstract

The purpose of this project is to perform sentiment analysis on text data, mainly by setting a workflow to apply different models to predict ratings based on movie reviews, and compare different models to better improve the accuracy. The IMDB data set we used is from Kaggle[1]. This report consists of 5 sections:

1. Introduction: introduce the goals of the report and describe the data set.
2. Background: Describe the background need to know about the project.
3. Methods: give an outline of how to perform the analysis.
4. Results: discusses the findings including modeling performance.
5. Conclusion: provide a summary of the report.

[1]https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

1 Introduction

The key question of this project is to predict the ratings of movies based on the reviews. This project had two goals, one is to set the workflow and apply it to different models, which are a rule-based and pre-trained model Valence Aware Dictionary and sentiment Reasoner (VADER), a machine learning model: Naive Bayes Classifier, and a deep learning model: Long Short-Term Memory (LSTM). The second goal is to compare the performance of these three models and improve the accuracy and reliability of sentiment analysis results.

The IMDB data-set we use including reviews and ratings for 50,000 movies. In the data-set, each row is a subject, and each column is a variable. So, the subjects are different movies, each row represents a movie and we refer to review and sentiment as variables. We have two columns. The first is review with is a textual variable with has the review, and sentiment which is the rating of the movie and so is a binary categorical variables with two levels: positive and negative.

2 Background

In the background section, we will cover some background knowledge that you need to know to complete the project. It includes:

- sentiment analysis
- the basic principle of VADER
- the basic principle of Naive Bayes Classifier
- the basic principle of LSTM

2.1 Sentiment Analysis

Sentiment analysis is an important aspect of natural language processing, it aims to extract attitudes, evaluations, opinions, and emotions from text. This could be in the form of a binary rating (positive or negative) or in the form of a numerical rating from 1 to 5. In this project, we used sentiment analysis to determine whether the comments were positive or negative.

2.2 Valence Aware Dictionary and sentiment Reasoner (VADER)

VADER (Valence Aware Dictionary and sentiment Reasoner) is a vocabulary database and rules-based sentiment analysis tool that can be used to identify social media emotions[5]. The core of VADER is to use comprehensive, high-quality vocabulary (about 7500 features) to determine emotional polarity and generate emotion scores. A range from -4 to +4 indicates a range from extremely negative to extremely positive emotions.

The important thing is that VADER generates a compound score to summarize the emotional intensity of the input text through adding each feature in the dictionary, and the compound score is between -1 (extremely negative) and 1(extremely positive). Also, unlike other proposed emotional dictionaries, VADER considers emotions in the online context, such as commonly used abbreviations such as "WTF", and slang such as "giggly".

2.3 Naive Bayes Classifier

Naive Bayes classifier is a simple probabilistic classifier based on Bayes' theorem in machine learning that assumes strong independence between features. Therefore, before exploring Naive Bayes, we need to understand Bayes' theorem.

In statistics, Bayes' theorem describes the probability of an event, based on prior knowledge of the conditions that may be associated with that event. The "probability" here also considers the factors associated with the random event, so Bayes' theorem describes the way in which posterior probabilities are obtained. To express Bayes' theorem in mathematical terms:

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

P(Y=k|X=x) = conditional probability of Y=k given X=x,
P(X=x|Y=k)= conditional probability of X=x given Y=k,
P(Y=k) = probability of event Y=k,
P(X=x)=probability of event X=x.

Naive Bayes assumes that each feature X_i is conditionally independent of every other feature X_j for j not equal to i, give the category k. And according to the conditional probability distribution, we can get the equation:

$$\begin{aligned} P(X=x|Y=c_k) &= P(X^{(1)}=x^{(1)},\cdots,X^{(n)}=X^{(n)}|Y=c_k) \\ &= \prod P(X^{(j)}=x^{(j)}|Y=c_k) \end{aligned} \tag{2}$$

From this, we can deduce the mathematical formula of naive Bayes classification is:

$$P(Y=k|X=x)=\frac{\prod P(X^{(j)}=x^{(j)}|Y=k)P(Y=k)}{P(X=x)} \tag{3}$$

In this project, we have to figure it out separately P(Y=1|X=x) and P(Y=0|X=x), if P(Y=1|X=x) larger than P(Y=0|X=x), we can identify the review is positive and if P(Y=1|X=x) less than P(Y=0|X=x), the review is negative.

2.4 Long Short-Term Memory (LSTM)

Long Short Term Memory , we also called "LSTM", is an artifical neural network, and it is a type of Recurrent Neural Network (RNN) used to process temporal data[6].

2.4.1 RNN

The figure 1 shows the appearance of an unrolled recurrent neural network, we can find that the concept of hidden state h is introduced into

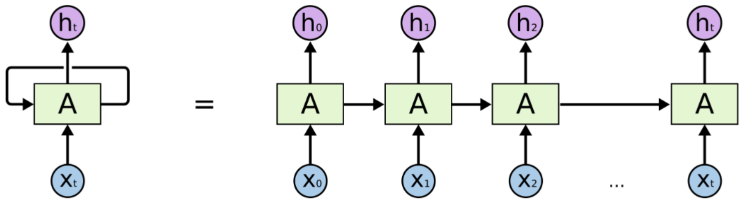


Figure 1: An unrolled recurrent neural network[6]

the recurrent neural network. The hidden state h can extract features from the data of sequence shape and then convert it into output. And RNN like multiple copies of the same network, each copy passes its contents to the next, and the parameters of each step are shared, which is an important feature of RNN.

Although RNN has advantages in processing sequence data, it also has some problems, such as gradient disappearance and gradient explosion. Because RNN would be influenced by short memory, which means if one sequence data is long enough, it will be hard to pass information from an earlier time step to a later time, because the gradient is the weight used to update the neural network (new weight = old weight - learning rate * gradient). In another word, layers that get small gradient updates will be stop learning, which are usually the earlier layers. Since these layers don't learn, the RNN forgets what it saw before in longer sequences, so the RNN only has short-term memory. Therefore, the LSTM was introduced to solve these gradient problems.

2.4.2 LSTM

LSTMS and RNN are not particularly structurally different, but in a standard RNN, the repeating module would have a very simple structure, such as a single tanh layer, however as the figure 2 shows, the LSTM has four neural layers interacting in a specific way. the four neural layers are three Sigmoid and one tanh layers.

As shown in the figure 2, the Sigmoid activation function represented by σ is similar to the tanh function, the only different thing is that sigmoid compressions the values between 0 and 1 instead of -1 and 1. Because this setting helps updating or forgetting information, so it normalizes the output of each neuron.

The figure 3 shows that the LSTM consists of a memory cell and three gates: the input gate, forget gate, and output gate. This structure of can be used to remove or add information to the cell state. A door is a way

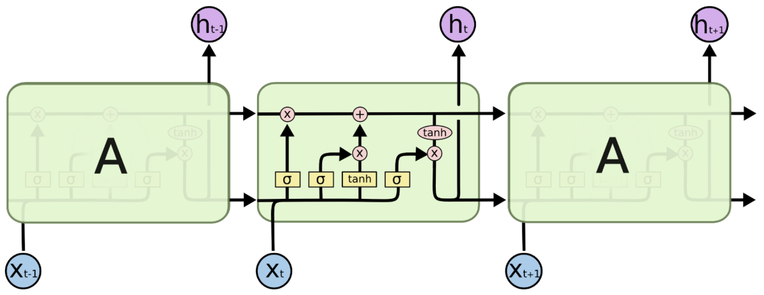


Figure 2: The repeating module in an LSTM contains four interacting layers.[6]

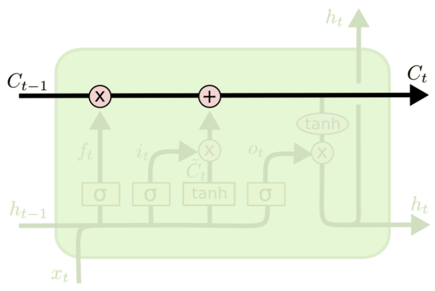


Figure 3: The cell state of LSTM[6]

of letting information through selectively. They contain a sigmoid neural network layer and a nonlinear operation of pointwise multiplication. Therefore, this operate allows the network to learn what data to forget and what data to save.

The forget gate layer is used to decide what information need to throw away from the cell state. Then in the input gate layer, it can decide what new information need be saved in the cell state. The forget gate layer is used to decide what information need to throw away from the cell state. Then in the input gate layer, it can decide what new information need be saved in the cell state. The last output gate will decide what information will to be output.

3 Methods

The language we used to complete the project is Python. Besides the common packages such as numpy[3], matplotlib [4]and pandas[7], we also used nltk[2], sklearn[8] and tensorflow[1] packages to analyse the data.

In this dataset, we counted the number of positive and negative ratings and found that both are 25,000, and then we removed the duplicate reviews. In this dataset, we counted the number of positive and negative ratings and found that both were 25,000, and then we removed the duplicate reviews. Then, we use three different methods including different data pre-processing to compare the predicted results. One is VADER: a rule-based pre-trained model, another is a machine learning algorithm: Naive Bayes Classifier and finally a deep learning algorithm: LSTM. Comparing the performance of different models by confusion matrix, F1 score and accuracy.

3.1 VADER

To better understand VADER, we have used VADER to analyse sentiment separately for data without text pre-processing and for data with text pre-processing and compared the results. To implement VADER simply use the SentimentIntensityAnalyzer() class provided by the NLTK library. This class returns a dictionary with four attributes, including an attribute called "compound" , which is the composite score of the sentiment analysis.

We first make a copy of the dataset for data pre-processing, and for dataset without data pre-processing we use VADER directly.

For data pre-processing, we used stopwords.words('english') method to remove some common stop words, and then created a clean() function to remove expressions, URL tags, punctuation, and to make all words lowercase.

Finally, the VADER predictions under the different treatments were compared with the indicator F1 and accuracy.

3.2 Naive Bayes Classifier

Naive Bayes classifier is a relatively simple classifier in machine Naive Bayes classifier is a relatively simple classifier in machine learning. It focuses on determining the classification to which an unknown data belongs by the probability of the known data. Machine learning for NLP requires pre-processing to extract features from the original text.

In the data pre-processing process, the predicted sentiment column (positive and negative) is converted into a numerical representation, with

"1" being positive and "0" being negative. Then, as with VADER, the data is removed from the stop words, emoticons, URL tags, punctuation etc. The difference is that here we used the WordNetLemmatizer() method to lemmatize a word based on its context and its usage within the sentence.

After data pre-processing, we need to build a new dictionary, which means putting all the words in the text into a vocabulary set. Then use 80 percent of the samples in the data set for the training set and 20 percent for the test set and set random seed to make sure the result is same each time.

Next, to implement the naive Bayesian algorithm, the two methods prior_probabilities() and conditional-probabilities() were created. The Prior_probabilities() method can be used to calculate the prior probabilities for each category. The Conditional_probabilities() method can be used to calculate the conditional probability under a given category, i.e. the conditional probability of each word under each category.

To train the model, I use training data with these two methods to obtain the prior probabilities for each category and the conditional probabilities for each word. In order to test the data in the test set, another method, classify() was created with the aim of making predictions for new reviews. If the probability that the words is 1 (positive) given the new review is greater than the probability that the words is 0 given the new review, then it can be identified to be a positive review. Finally, to evaluate the performance of the model, the same F1 and accuracy metrics were used to be compared with other methods.

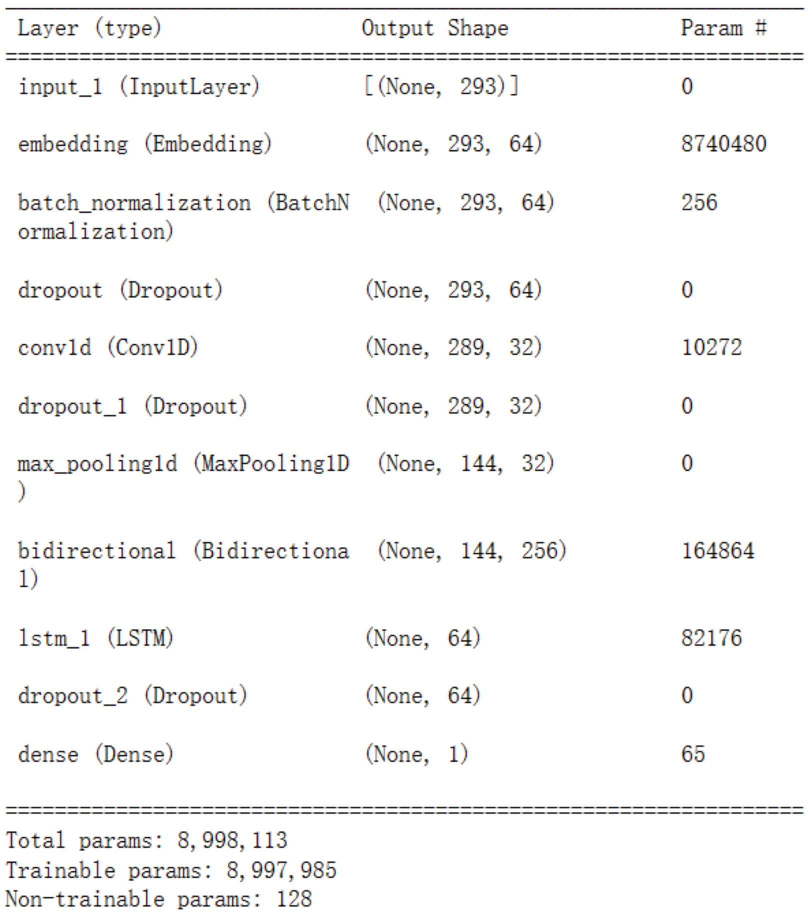
3.3 LSTM

The pre-processing of LSTM have some similar operations to the pre-processing of Naive Bayes Classifier, but LSTM did not use WordNetLemmatizer() to lemmatize words. And we need to split the training data for validation purposes. Furthermore, before implementing LSTM, we need to tokenize the review into words, and make texts to sequence, which we used sequence.pad_sequences() function. Then we used word embedding, which is a technique that converts each text into a sequence of numbers, where each number maps to a word in a vocabulary. Word embedding puts all words into a multidimensional vector space, so their similarity can be measured by distance.

In data modelling, we created a function called lstm_model to build the LSTM model by using Keras and set different layers and parameters, and then fit the model, and prints a summary of a neural network model as shown in figure 4. The inputs of the function are the predictors and outcome variables from training and validation dataset, the vocabulary length of the dataset, embedding dims, maximum length of sequences, and epochs. The following layers are included in the model:

- Embedding layer: defined parameters "V+1" represents the input dimension, "D" represents output dimension, and input-length equals to maxlen(the maximum length of sequences).
- Batch_ normalization layer.
- Two Dropout layer: the first one's parameter is set 0.3, and applied after embedding layer to drop 30 percent of the node outputs. The second one's parameter is 0.5, and it is applied after LSTM layer. After the first drop, a Conv1D layer with 32 filters and convolution kernel size of 5 is added, and ReLU is used as the activation function.
- Conv1D layer: with 32 filters and convolution kernel size of 5, and ReLU is used as the activation function.
- MaxPooling1D layer: set the size of pooling window is 2, in order to reduce the complexity of output and prevent data overfitting.
- Bidirectional layer: which have 128 hidden units, and we set return_sequences=True, it means that the LSTM layer should return the output sequences for each time step.
- LSTM layer: with 64 hidden units.
- Fully connected (Dense) layer : with the number of output neurons is 1, and set activation function is 'sigmoid'.

Then, we use fit() function to train model, with epochs is 5 and batch_size is 32. Finally, generating predictions for the test dataset and use same metrics (F1 score and accuracy) to compare the performance with other methods.

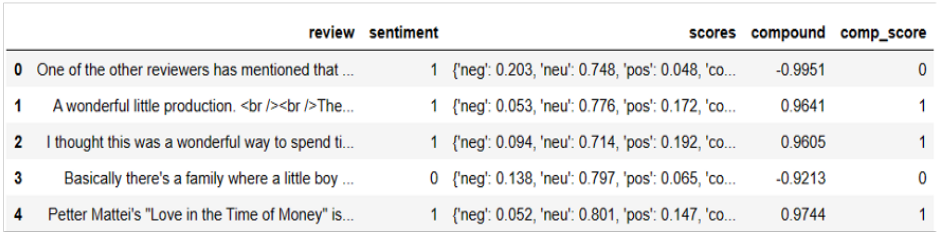


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 293)]             0

 embedding (Embedding)       (None, 293, 64)           8740480

 batch_normalization (BatchN  (None, 293, 64)          256
 ormalization)

 dropout (Dropout)           (None, 293, 64)           0

 conv1d (Conv1D)             (None, 289, 32)           10272

 dropout_1 (Dropout)         (None, 289, 32)           0

 max_pooling1d (MaxPooling1D  (None, 144, 32)          0
 )

 bidirectional (Bidirectiona  (None, 144, 256)         164864
 l)

 lstm_1 (LSTM)               (None, 64)                82176

 dropout_2 (Dropout)         (None, 64)                0

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 8,998,113
Trainable params: 8,997,985
Non-trainable params: 128
```

Figure 4: A summary of a neural network model

4 Results

Table 1: The dataset formed after VADER generates the compound score



	review	sentiment	scores	compound	comp_score
0	One of the other reviewers has mentioned that ...	1	{'neg': 0.203, 'neu': 0.748, 'pos': 0.048, 'co...	-0.9951	0
1	A wonderful little production. <br /><br />The...	1	{'neg': 0.053, 'neu': 0.776, 'pos': 0.172, 'co...	0.9641	1
2	I thought this was a wonderful way to spend ti...	1	{'neg': 0.094, 'neu': 0.714, 'pos': 0.192, 'co...	0.9605	1
3	Basically there's a family where a little boy ...	0	{'neg': 0.138, 'neu': 0.797, 'pos': 0.065, 'co...	-0.9213	0
4	Petter Mattei's "Love in the Time of Money" is...	1	{'neg': 0.052, 'neu': 0.801, 'pos': 0.147, 'co...	0.9744	1

Table 1 gives the dataset containing the "compound" column and the "comp -score" column returned after using Vader for sentiment analysis.

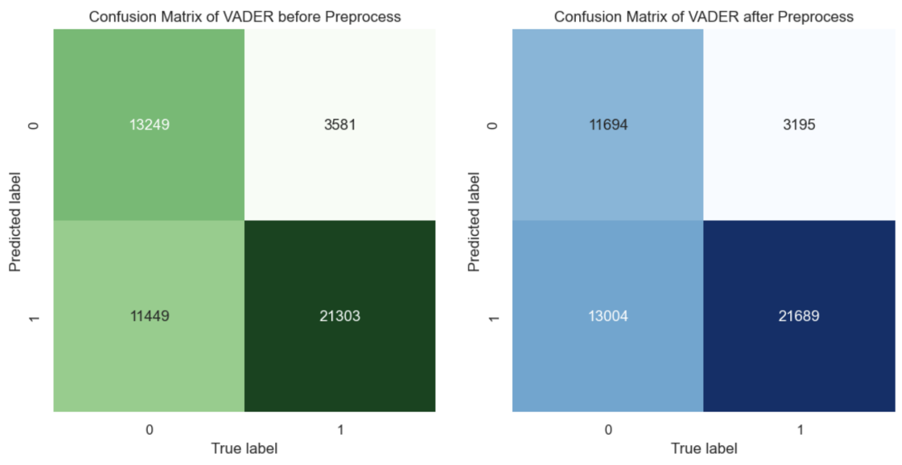


Figure 5: Confusion matrix generated after using Vader

Figure 5 gives a comparison of the confusion matrix plots between using the VADER model directly for sentiment analysis and using the VADER model after pre-processing the data, which clearly shows that using VADER without data pre-processing works better.

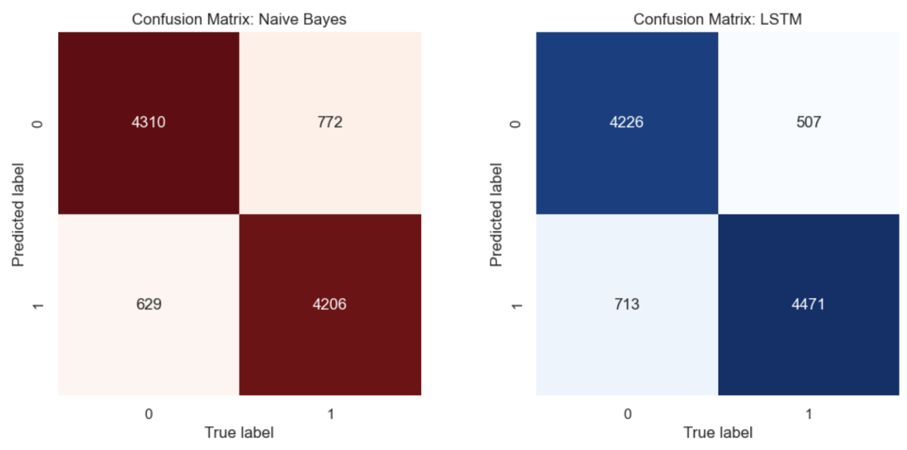
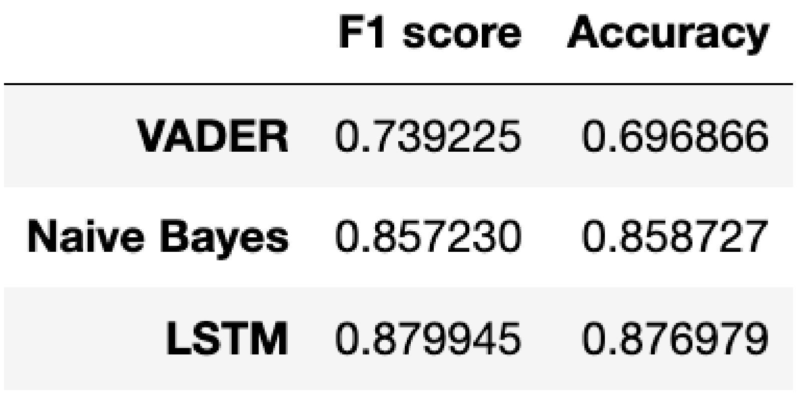


Figure 6: Confusion matrix: Naive Bayes versus LSTM

Figure 6 gives a comparison of the confusion matrix plots after using the Naive Bayes model and the LSTM model respectively for sentiment analysis. We can clearly see that the LSTM model gives better results.

Table 2: Model score results



	F1 score	Accuracy
VADER	0.739225	0.696866
Naive Bayes	0.857230	0.858727
LSTM	0.879945	0.876979

Table 2 gives a comparison of the performance of the different models, and it is clear that LSTM has the best prediction results, followed by Naive Bayes, and the worst is VADER.

Figure 7: Plot of the top 10 words in negative and positive ratings

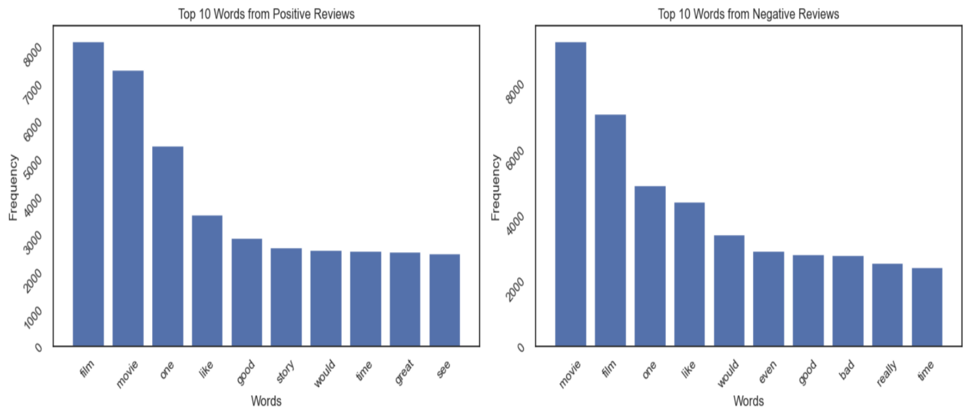


Figure 7 gives the top 10 most frequent words predicted by the LSTM model for the different ratings (positive and negative). We see that the terms good, great, story have positive effects on rating, while the terms bad, even have negative effects on rating.

5 Conclusion

We can see from the comparison of the results of different models that LSTM is the model that predicts the best results, with its accuracy is around 0.88. Naive bayes classifier is the second best, but the difference is not too much, while Vader works worst and its accuracy is only around 0.7. This is very reasonable as machine learning allows for processing data in a more sophisticated way, whereas Vader is a method for text sentiment analysis based on lexical dictionaries and grammatical rules that may be limited by outdated lexicons. In addition, compared to pre-trained models, Naive Bayes classifier is a machine learning model and LSTM is a deep learning where they can feed pairs of text and tokens (sentiment) into the algorithm to create models that can make predictions on new text, this custom model allows for better control of the output, and LSTM is based on the contextual information of words in the text for its analysis.

The models comparison reveals that the different models have advantages and disadvantages. VADER, although ineffective, does not require training. Although the LSTM works best on our dataset, is the most expensive to train and manipulate as a deep learning model. Naive Bayes classifiers, while slightly less effective than LSTM, usually provide faster training and prediction.

We also compared the impact of text pre-processing on the prediction results using VADER. According to Figure 6, it is interesting to note that after using text pre-processing, VADER's ability to correctly predict 1 (positive) increases, but its ability to correctly predict 0 (negative) decreases. This is probably because text pre-processing uses the removal of symbols, expressions, etc., but VADER's sentiment information takes into account punctuation, which results in a change in VADER's compound score after use.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Edward Loper Bird, Steven and Ewan Klein. Natural language processing with python. *O'Reilly Media Inc.*, 2009.

[3] Millman K.J. van der Walt S.J Harris, C.R. Array programming with numpy. *Nature*, (585):357–362, 2020.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[5] C.J. Hutto and E.E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the International AAAI Conference on Web and Social Media*, 8(1):216–225, 2014.

[6] Christopher Olah. Understanding lstm networkst. 2015.

[7] The pandas development team. pandas-dev/pandas: Pandas. February 2020.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.