Abstract

Sentiment analysis is one of the many widely used Natural Language Processing (NLP) applications today. With the explosion of social media over the past decade, researchers now have access to more data than ever before to combine NLP techniques with deep learning methods to build sophisticated machine learning models. This project begins with application of various NLP techniques—including text preprocessing and vectorization—to the well-known *Sentiment140* Twitter data set. Constructing a robust machine learning model is dependent on this NLP given the large amount of textual data. While I experiment with both traditional machine learning algorithms and deep learning methods to build classification models, my primary focus is exploring optimal neural network structures to generate a model that could accurately predict whether any given tweet has positive or negative sentiment. In implementing this machine learning project, I also demonstrate best practices and highlight challenges in applying NLP and deep learning with regard to sentiment analysis efforts.

Keywords: sentiment analysis, Twitter, natural language processing, deep learning, neural networks, binary classification

Using Deep Learning to Predict Twitter Sentiment

Twitter is a popular social media platform where entities from all demographics and levels—governments, businesses, country leaders, celebrities, adults, and children—can create, post, update, and read short text messages called *tweets*. Depending on the user and content, these tweets could have significant influence on government policies, traditional media agenda, and popular sentiment, particularly as government leaders, journalists, and society increasingly view Twitter as a timely and valuable source of on-the-ground information concerning a range of topics and events (Valenzuela, Puente, & Flores, 2017).

Sentiment analysis can be used to gauge the polarity of a user's tweet. With 326 million active users sending more than 500 million tweets per day (as of 2019),[1] the ability to quickly predict a tweet's sentiment has become an important field of study for organizations in both public and private sectors. In this paper, I explore techniques in Natural Language Processing (NLP) and machine learning—traditional and deep learning—to construct several binary classification models capable of automatically classifying tweets as positive or negative. More specifically, I use a crossed experimental design to evaluate four distinct neural network structures—dense neural networks (DNN), recurrent neural networks (RNN), long short-term memory networks (LSTM), and convolutional neural networks (CNN)—to determine optimal approaches in implementing this classification task.

Literature Review

Various methods and related works on Twitter sentiment classification have been conducted in the past. For example, Shah (2017) suggested using a hybrid method that combines machine learning and a lexical approach. This process entails using machine learning to identify the sentiment patterns of

[1] https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

tweets, then using a lexical approach to preprocess tweets to calculate sentiment score, which machine learning algorithms leverage to make better predictions. Han, Guo, & Schütze (2013) proposed a method that combines a Support Vector Machine (SVM) classifier and various features—such as bag-of-words (e.g., one word, two words) and part-of-speech tags—along with votes derived from character n-gram language models to achieve optimal results. Finally, Zhang & Wallace (2015) argued that deep learning methods, notably Convolutional Neural Networks (CNN), could achieve remarkably strong performance, akin to SVMs and logistic regression, on NLP applications, including sentence classification and sentiment analysis. I consider various methods discussed above to develop and implement an NLP-machine learning pipeline that identifies the optimal combination of word embeddings, classifiers, and deep learning methods to accurately predict the polarity of tweets.

Methods

The corpus for this project is Stanford University's *Sentiment140*, a CSV file containing 1.6 million English-language tweets. The tweets are annotated using six attributes: (1) polarity of the tweet (0 = negative, 2 = neutral, 4 = positive); (2) ID; (3) date; (4) query; (5) username; (6) text.[2]

For interpretability, the paper is divided into three broad sections: (1) text preprocessing; (2) text vectorization; (3) model training. The last section (model training) entails two sub-sections that provide a detailed overview of traditional classifiers and neural networks used.

Text Preprocessing

Upon initial exploratory data analysis (EDA), I determined that the text and tweet polarity values were the only essential features to successfully train a sentiment classification model. After the remaining features from the dataframe, further EDA revealed that the text required a significant amount

[2] The corpus is available for download here: http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip

of clean-up and processing. Given the massive number of documents in the corpus (one tweet = one document), regular expressions were essential in handling special characters. The prevalence of @mentions—a common practice in social media where users publicly acknowledge or promote another user—made it difficult to ensure all the documents were consistent and properly formatted.

To process the data in a consistent manner, I deleted combination of @, letters, and numbers (e.g., @[A-Za-z0-9]). In addition to @mentions, URLs and hashtags were dropped from the data set. While these symbols and characters could be useful for another business requirement, they would only act as noise during the model-training process. The cleaned and processed text was saved as a new CSV file that I used for all follow-on tasks, from word embeddings to model training.

Text Vectorization

Based on further EDA, using term frequency alone posed some challenges in accurately determining words associated with sentiment. For example, word clouds generated for both positive and negative clusters featured "love" and "lol," words that typically are representative of positive sentiment (see figures 1 and 2 in appendix). To resolve this issue, I split the data set into train, validate, and test sets and evaluated three text vectorization methods (CountVectorizer, TfidfVectorizer, and Doc2vec) using logistic regression, the go-to method for binary classification problems (Brownlee, 2019).

In the initial evaluation of the CountVectorizer approach, not removing stop words in the model resulted in nearly a three percent improvement in performance. Accordingly, I included stop words for all the following model evaluations. The CountVectorizer and TfidfVectorizer methods were each evaluated using three different n-gram word sequence models: unigram, bigram, and trigram. Significantly, trigram models achieved the highest performance, and TF-IDF vectorization performed better than CountVectorizer. For Doc2vec, I used the Distributed Bag of Word (DBOW) and Distributed

Memory Mean (DMM) methods with 100 vector dimensions. A trigram-TfidfVectorizer combination resulted in the highest accuracy (83.13%) among all three approaches (see figure 3).

Model Training

I used both traditional machine learning classifiers and neural network methods to train, test, and evaluate 18 distinct models. Limited time and resource was devoted to evaluating traditional classifiers since the focus of this project is on constructing optimal neural network models.

Traditional Classifiers

Having determined that a logistic regression model with the trigram-TfidfVectorizer and 80,000 features resulted in the highest performance, I used the same n-gram-vectorization parameters to run an ensemble voting classifier function to generate the accuracy scores for the following four alternative classifiers: Linear Support Vector Classification (SVC), multinomial Naive Bayes (MultinomialNB), Ridge Classifier, and Nearest Centroid. Other popular classifiers, such as K-Nearest Neighbors or Decision Trees/Random Forest, were not included in the ensemble due to time limitations and computational cost. Of the five models, the logistic regression classifier had the strongest performance, with a validation and test accuracy score of 83.13% and 81.91%, respectively (see figure 4)

Neural Network Models

Deep learning models for this project were trained, tested, and evaluated using Keras's Sequential model API. Each of the four neural network categories (DNN, RNN, LSTM, CNN) consisted of at least three rounds of experiments using the following architectures: 1) a base model without any dropouts or other regularization techniques; 2) adding dropouts in between each layer; 3) using trainable (i.e., setting the *trainable* parameter to *True*) pre-trained word embeddings to fine tune the Embedding layer during the model training. The pre-trained word embeddings used for this project is GloVe's

Twitter 200-dimension vectors.[3] When using the GloVe word vectors in embedding layer for neural networks, a 100,000 vocabulary size is used for the input dimension.

The hyperparameter settings for the model compilations and fit methods remained generally the same for consistency purposes. The model compilation consisted of an *RMSprop* optimizer, *binary crossentropy* for the loss function, and an *accuracy* score for metrics. *RMSprop* was used instead of alternative optimizers, such as Stochastic Gradient Descent, because of how it restricts the oscillations in the vertical direction; thus, increasing the learning rate and enabling the algorithm to take larger steps in the horizontal direction for faster convergence (Gandhi, 2018). The *binary crossentropy* loss function was used since there are only two predictable classes.

Using Keras callbacks, a timer and early stopping function (after three iterations of unimproved accuracy) was incorporated into the training process. Epochs and batch size were set to 10 and 64, respectively.[4] For consistency, all dropout layers were set as 20%. See figure 5 in the appendix for detailed code on the hyperparameter settings. For the activation functions, *ReLU* was used for all inner layers and *Sigmoid* was used for the final output layer. The *Adam* optimizer was used since it seems to combine the strengths of other Keras optimizers and requires little hyperparameter-tuning.

DNN Models. One base DNN model was built using three layers (input, hidden, and output). Like the logistic regression classifier model, an input dimension size of 80,000 was used. The input and hidden layers for both models consist of 64 and 32 nodes, respectively. The output layer for the full dataset model has 1 node for the final binary prediction class. Of all the DNN models, the base model

---

[3] GloVe stands for Global Vectors for Word Representation. The word vectors based on Twitter is called glove.twitter.27B.zip (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors) and is available https://github.com/stanfordnlp/GloVe

[4] The batch size for the 1D CNN model training was raised to 256 from 64 after a noticeable improvement in performance with the higher batch size.

(i.e., no dropouts and no pre-trained word embeddings) yielded the highest test accuracy (81.09%). Notably, performance deteriorated by nearly 3 percent (78.77%) when the pre-trained GloVe word embeddings were used to train the model.

RNN Models. The base RNN structure consists of an embedding layer using the pre-trained GloVe word vectors. The input is fed through a 128-unit *SimpleRNN* layer—a fully-connected RNN where the output is fed back to input—a fully-connected *Dense* layer with *ReLU* activation to add more representational capacity to the network, and finally a *Dense* fully-connected output layer with one node. The base model yielded a test accuracy of 80.19%. Notably, adding dropout layers caused more overfitting and dropped the test accuracy to 77.45%. However, when adjusting the network architecture so the word vectors are continuously updated throughout the training process (i.e., trainable = true), the test accuracy increased to 81.14%, which was the highest recorded score for the RNN models.

LSTM Models. The base LSTM network structure consists of an embedding layer that feeds the input through a 128-unit LSTM layer using the Keras's default cuDNN implementation requirements (e.g., *tanh* activation and *sigmoid* recurrent activation), a fully-connected *Dense* layer with *ReLU* activation to add more representational capacity to the network, and a *Dense* fully-connected output layer with one node. The base LSTM model resulted in a test accuracy score of 82.76%, performing better than any of the DNN or RNN models. Similar to the RNN models, adding dropouts to the network structure resulted in overfitting and a relatively poor performance (76.89% test accuracy). Also similar to the RNN models, adjusting the network structure to continuously update the pre-trained word vectors through the training process shot the test accuracy score to 83.19%.

CNN Models. While CNN models were developed for image classification problems—where the model learns an internal representation of a two-dimensional input—in a process referred to as feature learning, the same process can be harnessed on one-dimensional (1D) sequences of data, such as

textual data. The benefit of using CNNs for sequence classification is that they can learn from the raw time series data directly, and in turn do not require domain expertise to manually engineer input features (Brownlee, 2018). Accordingly, 1D CNN architectures are used in this project.

In order to determine the best base model, I experimented with various network structures, alternating between 128 and 256 filters, adding and removing *batch normalization* and 1D *max pooling* layers, and increasing the convolutional layers from two to three. The structure consisting of 256 filters, two convolutional layers, and neither *batch normalization* nor *max pooling* layers performed the best with a test accuracy of 82.52% (see figure 6). Unlike the RNN and LSTM models, adding dropout layers actually helped with regularization and improved the model, albeit by only 0.3%. The architecture also includes a batch normalization and a 1D max pooling layer after the convolutional layer to reduce the number of parameters in the model. As expected, updating the architecture settings so the pre-trained word vectors are continuously updated during the model training process resulted in a test accuracy of 83.25%, the highest recorded score among all the models. In an attempt to further improve the model, I added dropout layers. The resulting test accuracy was 83.08%, indicating no significant improvement.

Results and Findings

CNN model 3 (a 1D CNN with continuously trained word embeddings and no dropouts, batch normalization, and max pooling layers) performed the best with a test accuracy score of 83.25%. This was only 0.06% higher than LSTM model 3 (an LSTM network with continuously trained word embeddings and no dropouts). However, when taking into consideration the training time spent building the models, the CNN is the superior method.

When building DNN models, regularization methods, such as dropouts, and pre-trained word embeddings do not seem effective in improving the performance. The same applies to RNN and LSTM models. However, between RNNs and LSTMs, LSTMs performed better, suggesting that LSTMs are

more superior when it comes to conducting sequence classification tasks. However, the higher performance of LSTMs comes at the expense of more training time; training one LSTM model in this project took up to three hours whereas training each RNN model consistently took 30 minutes.

In addition to the test accuracy scores, the AUC ROC curve was also used to evaluate the top-performing models from each deep neural network model category (i.e., DNN, RNN, LSTM, CNN). The AUC scores for the best CNN, LSTM, RNN, and DNN models are 0.912, 0.911, 0.895, and 0.888, respectively (see figure 7). This indicates that both CNN and LSTM models have at least a 91% chance of distinguishing between positive and negative tweets.

Conclusion

This paper evaluated a total of 18 models: 13 deep neural network models based on DNN, RNN, LSTM, and CNN architectures and 5 traditional machine learning models. See figures 8-10 for performance metrics summaries of all 13 deep neural network models. While CNN and LSTM models in this project yielded the best results, there is no one-size-fits-all approach to sentiment analysis, particularly with regard to Twitter data. Alternative text pre-processing method and vectorization techniques should be considered. With more experimentation, time, and computational resources, it would be possible to build a more optimal model. Notably, further experimentation using attention-based models would yield interesting results since such models can combine the strengths of both conventional and deep learning methods—effectively capture the sequential structure of the text (e.g., LSTM) while attaching higher weights to important words (e.g., TF-IDF and CountVectorizer) (Vaswani, et al., 2017).

References

Brownlee, J. (2018, September 21). How to Develop 1D Convolutional Neural Network Models for Human Activity Recognition [Blog Post]. Retrieved from https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/

Brownlee, J. (2019, August 12). Logistic Regression for Machine Learning [Blog Post]. Retrieved from https://machinelearningmastery.com/logistic-regression-for-machine-learning/

Gandhi, R. (2018, June 19). A Look at Gradient Descent and RMSprop Optimizers [Blog Post]. Retrieved from https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop optimizers-f77d483ef08b

Han, Q., Guo, J., Schütze, H., Codex: Combining an svm classifier and character n-gram language models for sentiment analysis on twitter text, in: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), Association for Computational Linguistics, 2013.

Shah, H. (2018). Twitter Sentiment Analysis. International Journal of Advanced Research in Computer Science and Software Engineering, 7(12), 15.

Valenzuela, S., Puente, S., & Flores, P.M., Comparing disaster news on twitter and television: an intermedia agenda setting perspective, J. Broadcast. Electron. Media 61 (4) (2017) 615–637.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin I., (2017). Attention is All You Need.

Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

Figure 1. Word cloud of all negative tweets.

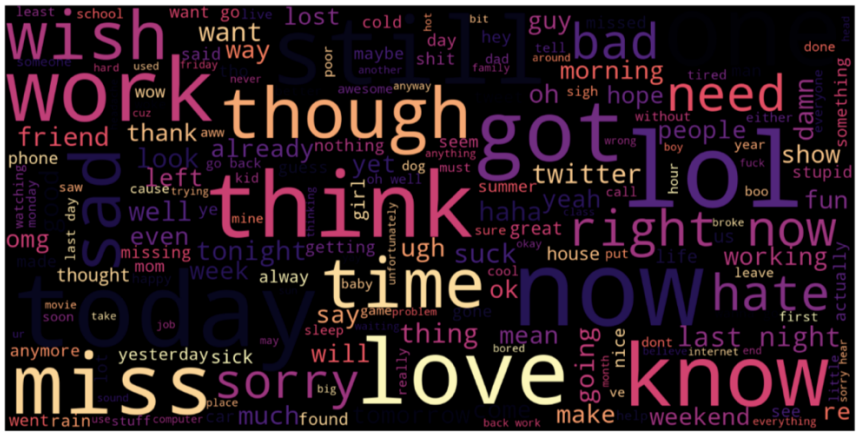

Figure 2. Word cloud of all positive tweets.

Figure 3. Summary table of vectorizer accuracy evaluation.

Method	n-gram	Features	Accuracy
CountVectorizer	unigram	80000	80.56%
CountVectorizer	bigram	100000	82.49%
CountVectorizer	trigram	70000	82.77%
TF-IDF	unigram	90000	80.78%
TF-IDF	bigram	90000	83.06%
TF-IDF	**trigram**	**80000**	**83.13%**
d2v-DBOW	unigram	100 (size)	73.57%
d2v-DMM	unigram	100 (size)	71.83%

Figure 4. Summary of validation accuracy for all traditional classifiers using a trigram TF-IDF vectorizer.

Classifier	Train Time (sec)	Val Accuracy	Test Accuracy
Logistic Regression	**223.58**	**83.13%**	**81.91%**
Linear SVC	229.46	82.82%	81.89%
MultinomialNB	166.51	80.76%	79.70%
Ridge Classifier	188.32	82.66%	81.62%
Nearest Centroid	169.22	72.43%	72.96%

Figure 5. Hyperparameter settings model compilation code for all models.

```
num_epochs = 10
batch_size = 64
patience = 3

optimizer='rmsprop'
loss='binary_crossentropy'
metrics='accuracy'

dropout = 0.2
RNN_units = 128
LSTM_units = 128
pool_size = 3
```

Figure 6. Network architecture for best-performing model (CNN-3).

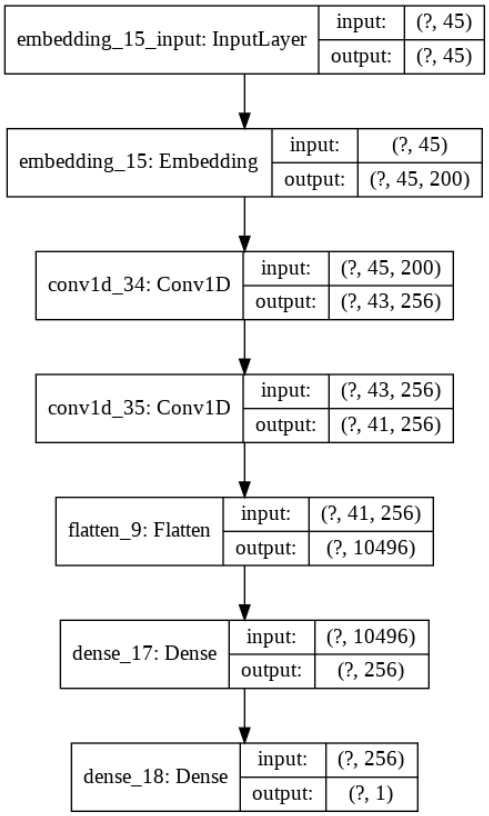

Figure 7. AUC-ROC curve of the best-performing models from each deep neural network model architecture**.**

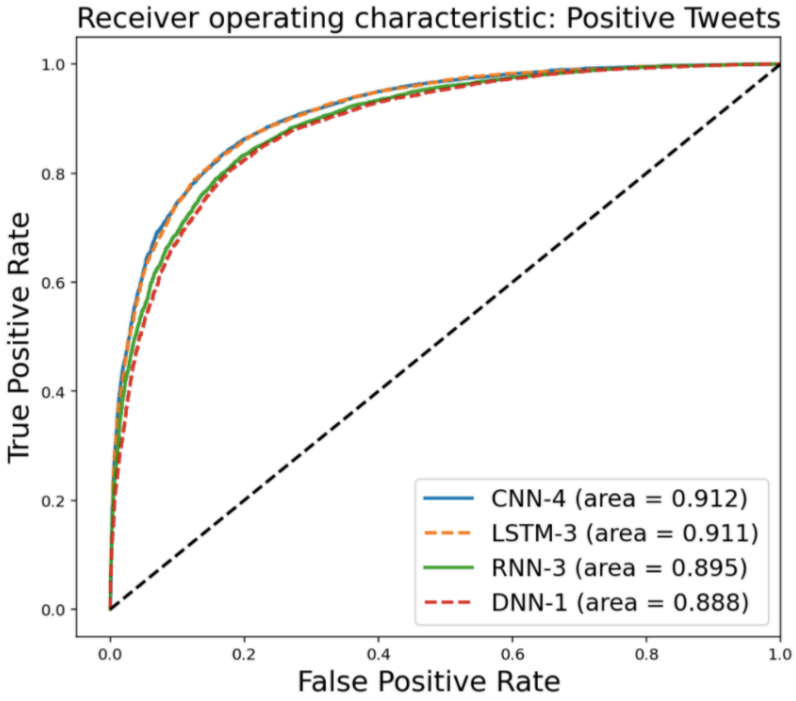

Figure 8. Summary table of model accuracy performance evaluation. The two highest-performing models (LSTM-3 and CNN-3) are in bold.

Model Name	Description	Train Loss	Train Acc	Val Loss	Val Acc	Test Acc	Train Time (Min)	Test Time (Sec)
DNN-1	Base DNN	0.3842	83.40%	0.4263	81.28%	81.09%	41.96	5.99
DNN-2	DNN w/ Dropout	0.4180	81.72%	0.4191	81.83%	81.02%	41.96	6.03
DNN-3	DNN w/ Trainable Word Embeddings	0.4281	80.89%	0.4424	79.55%	78.77%	17.17	0.85
RNN-1	Base Simple RNN	0.4391	80.01%	0.4691	78.21%	80.19%	30.58	1.98
RNN-2	RNN w/ Dropout	0.5537	71.78%	0.5115	75.13%	77.45%	30.58	2.33
RNN-3	RNN w/ Trainable Word Embeddings	0.4122	81.84%	0.4163	81.59%	81.14%	30.58	2.23
LSTM-1	Base LSTM	0.3517	84.67%	0.3676	84.22%	82.76%	181.24	8.23
LSTM-2	LSTM w/ Dropout	0.4199	81.12%	0.3917	82.55%	81.60%	143.63	10.18
LSTM-3	**LSTM w/ Trainable Word Embeddings**	**0.3581**	**84.37%**	**0.3659**	**84.22%**	**83.19%**	**192.26**	**10.21**
CNN-1	Base 1D CNN	0.3611	84.35%	0.3869	83.69%	82.52%	28.65	2.39
CNN-2	1D CNN w/ Dropout	0.3762	83.53%	0.3730	83.77%	82.83%	51.05	2.40
CNN-3	**CNN w/ Trainable Word Embeddings**	**0.3516**	**85.30%**	**0.3627**	**84.39%**	**83.25%**	**58.85**	**2.50**
CNN-4	CNN w/ Trainable Word Embeddings + Dropout	0.3599	84.91%	0.3693	83.92%	83.08%	114.79	2.38

Figure 9. Plot visualizing the train accuracy and corresponding fit times for each model.

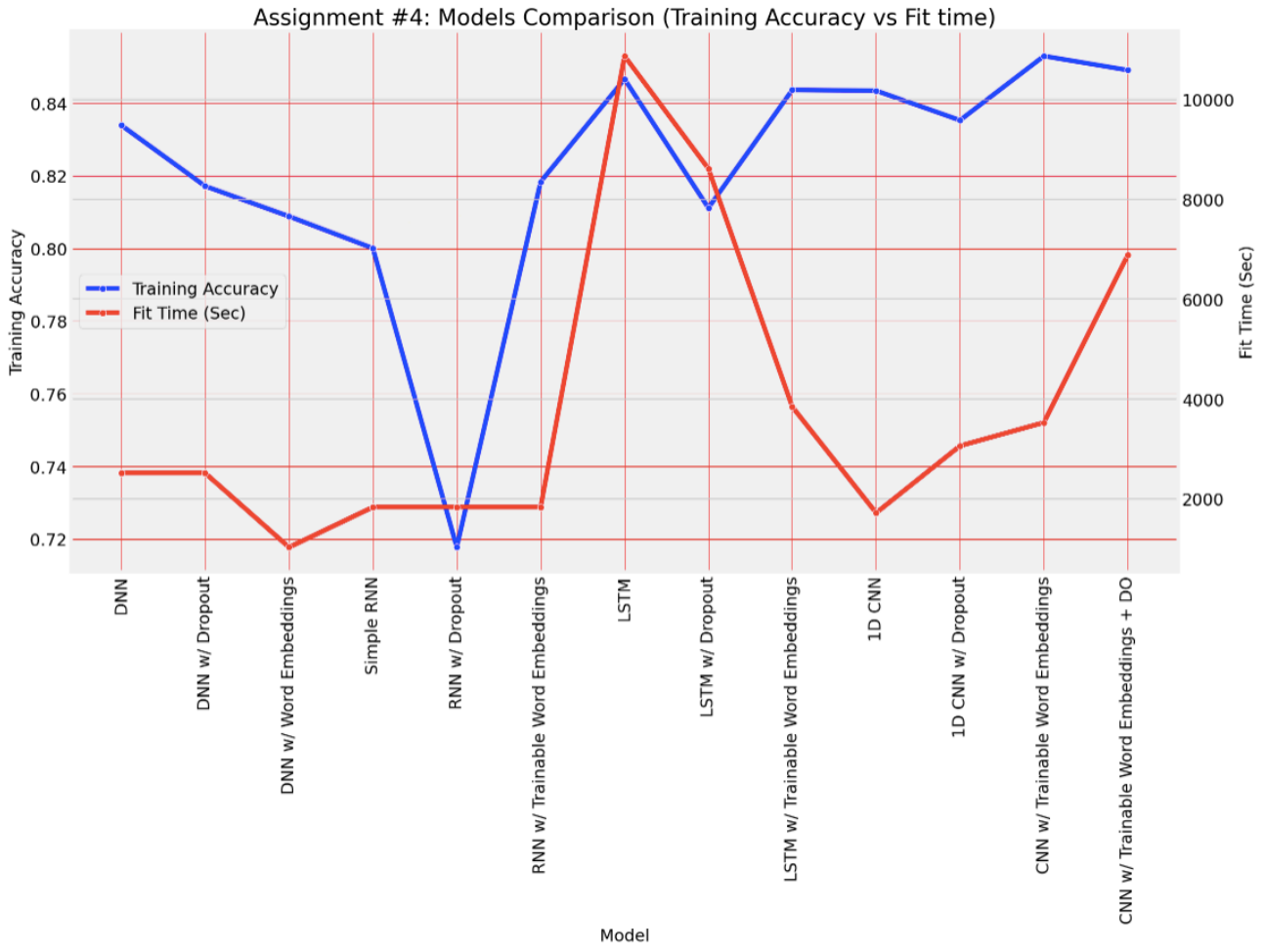

Figure 10. Plot visualizing the train/val/test accuracy for each model.