IDS703 Final Project Report[*]

Anna Dai, Satvik Kishore, Moritz Wilksch

December 12th, 2021

1 Introduction

In this project, we work on Tweet classification as a Natural Language Processing (NLP) problem, more specifically, as a document classification problem. Twitter is a microblogging service where users post publicly visible “Tweets”, which are essentially texts with less than 280 characters. These Tweets may also contain other media objects which are discarded for the purposes of this project. These Tweets most often serve as discussion pieces as part of larger conversations. Tweets may be relevant to any number of topics under discussion, and these “topics” are also often explicitly highlighted by the user using a “hashtag”, i.e. text with ‘#’ followed by the topic name, or a commonly used shorthand for it. For the purpose of our project, we treat these hashtags as “topics” for our document classification model, where each Tweet is an instance of a document.

2 Data

2.1 Collection

We manually select seven popular topics (hashtags) for classification: `crypto`, `tesla`, `formula1`, `championsleague`, `thanksgiving`, `holidays`, and `covid19`. These topics were intentionally selected so that some topics have some degree of overlap between them (e.g. `holidays` and `thanksgiving`), others are easier to differentiate (e.g. `crypto` vs `formula1`), and one is an independent topic that is often mentioned with the others (i.e. `covid19`). We leverage the python library TWINT (twintproject 2021) as well as the Twitter API directly (legally) to scrape approximately 10,000 Tweets, or documents, for each of these seven topics.

2.2 Pre-Processing

The raw scraped Tweets are quite messy as they are cluttered with media objects like links, images, mentions of other Twitter users (i.e. @usernames), other hashtags, etc., so we spend some time to carefully pre-process the data before proceeding with modeling. As a first step, we utilize regular expressions (regex) to remove any peripheral content such as URLs, mentions, hashtags, and cashtags. We then normalize our corpus by converting it to lowercase, stripping all punctuation marks, replacing all numbers with “<number>” with regex and removing stopwords (e.g. and, the, to) using the NLTK stopwords library. Lastly, we tokenize each document into separate words and encode these words using TorchNLP’s WhitespaceEncoder. We specifically choose to retain all emojis as tokens because we believe they have value in topic classification, however, multiple emojis strung together were treated as different tokens, resulting in an artificially inflated corpus. Thus, we leverage the emojis’ unicode character codings to break up them up into unique tokens.

Once cleaned, we randomly split the corpus of 70,000 documents to form our train, validation, and test data sets in a 60/20/20 ratio. We will train models on the train set, find optimal hyperparameters using the validation set, and report all performance metrics in the *Results* section on the test set, which is used only once to evaluate each model.

[*] `https://github.com/dai-anna/Duke-NLP-FinalProject`

3 Modeling

3.1 Generative Model

We use a Latent Dirichlet Allocation (LDA) model as a generative model to learn from the corpus we have collected. This type of model does not require any hyperparamter tuning, and thus is trained using a combination of the training and validation dataset. We used the LDA implementation from scikit-learn (Pedregosa et al. 2011). It is fit to the corpus to infer seven separate topics, as this is the number of actual topics in the training set. Subsequently, for each inferred topic, we manually inspect the top 50 words that are associated with it in order to assign it its topic label. This manual step is necessary as the order of topics is not preserved. In fact, the LDA model does not even guarantee to find the same topics that were collected in the original data set. This shortcoming will be discussed in the *Discussion* section. To use the LDA for document classification, we let the model infer the topic distribution of each document in the test set and use the $argmax(\cdot)$ function to assign each document the topic that is most prevalent according to the LDA. Finally, the LDA is used to generate synthetic data. For each topic, we sample 10,000 artificial documents, the length of which is sampled from the empirical distribution of tweet lengths each time. Similar to the actual data, the synthetic data is also split in a 60/20/20 ratio into a train, validation, and test set.

3.2 Discriminative Model

We develop a neural network model as a discriminative model in order to classify input documents into their respective categories. These documents can arise from the synthetic data generated by the LDA or the real data scraped from Twitter. The model architecture we employ is a bidirectional Gated Recurrent Unit (GRU) at its core. This model is able to accept inputs of varying lengths and outputs a vector of length seven for each input document. This output is the vector of predicted probabilities generated by the final Softmax layer for each of the seven topics. This model is implemented and trained using Keras on Tensorflow (Abadi et al. 2015). We use the training data to optimize model parameters using backpropagation while comparing the loss and accuracy on the validation model to optimize the hyperparameters. We apply cross entropy loss as the loss function to the final Softmax layer in the model to generate our final topic classifications.

3.2.1 Model Architecture

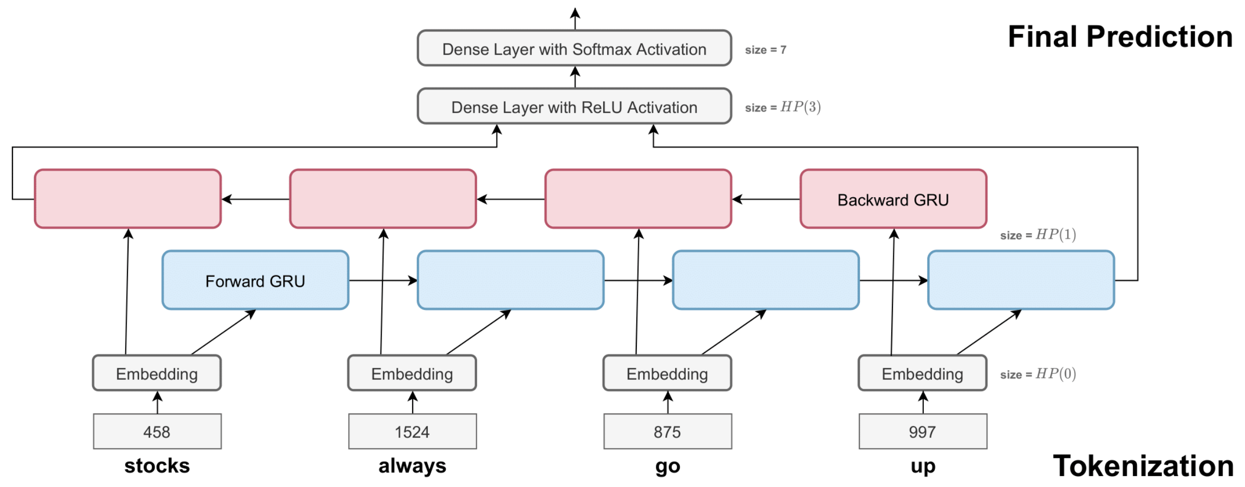


Figure 1: Neural Network Model Architecture

The model contains three essential subparts: the embedding layer, the bidirectional GRU layers, and the final densely connected layers. Each layer has one or more hyperparameters (denoted by $HP(i)$) that are subject to tuning. Documents are fed into the model in the form of tokens identified from the encoder. There are 16,003 unique tokens identified in the training data. The embedding layer is a lookup table that converts each token into a vector of length $HP(0)$. We transform every document to a length of 60 tokens by either padding with zeros or cropping it, so that the entire network is able to learn in batches of tensors. These embedded vectors are then received as input by the two GRU layers. GRU are a form of recurrent neural networks that use gates to better pass long term memory than simple recurrent neural nets (RNN), but have fewer parameters than long short-term memory (LSTM) neural nets. This allows for GRUs to be more expressive than simple RNNs and faster and easier to train than LSTMs. Research also suggest that performance of GRUs are comparable with LSTMs for some NLP models (Chung et al. 2014). GRU also do not differentiate between the output and hidden vectors and the size of these hidden vectors are $HP(1)$. The final hidden vectors of both GRUs (forward and backward) are then concatenated and fed into a dropout layer with dropout rate $HP(2)$ followed by a densely connected network of two layers separated by another dropout layer with the same rate of $HP(2)$. These two dense layers are of size $HP(3)$ and 7 with ReLU and Softmax activation functions respectively. We also implement ℓ_2 (Ridge) regularization by a factor of $HP(4)$ in the GRU, ReLU and Softmax dense layers to reduce overfitting. All training is done using the Adam Optimizer.

3.2.2 Training Process

In this project, we have trained three discriminative models that are compared and evaluated:

1. Neural network trained only on synthetic data generated by the LDA generative model
2. Neural network trained on synthetic data with continued training on real data
3. Neural network trained only on real data

We begin by tuning the models hyperparameters on the synthetic data set. The tuning is implemented using the optuna library (Akiba et al. 2019) and its implementation of Tree-structured Parzen Estimators, a sequential model-based optimization approach to intelligently exploring hyperparameter search spaces (Bergstra et al. 2011). Each experiment is run for 50 trials (1 trial = one entire training run) due to time constraints. Each model is trained for 20 epochs, but we employ an early stopping mechanisms to stop training after 3 epochs without validation accuracy increase as, empirically, our models converged quickly and performance plateaued. The batch size for our experiments is fixed at 64. The learning rate is also kept at $10^{-2.5}$ for all experiments. This value was determined using the learning rate range test proposed by Smith 2018. It works by fitting the model for a single epoch while changing the learning rate from small (10^{-6}) to large (1). Subsequently, we plot the loss over time, which hardly changes in the beginning of the epoch and diverges at the end of the epoch. The optimal learning rate can be visually identified close to the apex, i.e. the point where the loss decreases fastest.

Parameter	Description	Search Space
HP(0)	Embedding dim	$\{16k \vert k \in \{1,2,3,4,5\}\}$
HP(1)	GRU dim	$\{16k \vert k \in \{2,3,...,16\}\}$
HP(2)	Dropout rate	$[0, 0.5]$
HP(3)	Dense dim	$\{16k \vert k \in \{1,2,...,16\}\}$
HP(4)	ℓ_2 regularization	$[10\text{e}^{-9}, 0.5]$

Table 1: Hyperparameters and their search spaces.

After finding the optimal hyperparameter configuration (shown in Table 2 below), we refit the model using the optimal parameters and benchmark its performance on the synthetic and real data. Subsequently, we continue training the same model on the real data set. We hypothesize that pre-training on synthetic data might help the model converge faster on real-world data. After training for another 20 epochs, we benchmark the model again.

To gauge whether pre-training on synthetic data had an impact on model performance or training behavior, we conduct another experiment. We start a new hyperparameter search with the same model architecture and search space, but this time on the real data set. We use the optimal parameters to create a third neural network model that has never seen synthetic data and benchmark it on both, the synthetic and real test set. Table 2 shows the optimal parameter configurations found in each run.

Parameter	Description	Synthetic Data	Real Data
HP(0)	Embedding dim	32	64
HP(1)	GRU dim	64	48
HP(2)	Dropout rate	0.1	0.25
HP(3)	Dense dim	64	112
HP(4)	ℓ_2 regularization	$10e^{-9}$	$2e^{-8}$

Table 2: Optimal hyperparameter configuration for models on synthetic and real data

4 Results

4.1 LDA Model

The LDA model performs exceptionally well scoring precision and recall values of around 0.98 on synthetic data. This is to be expected, as the model itself generated the data. Therefore, there is only a small amount of randomness from the sampling procedure that prevents the model from achieving an accuracy of 100%. However, applying the same model to real-world data reveals that this performance does not translate well. The topics `thanksgiving` and `covid19` in particular seem to be hard to classify by the model as they have the lowest precision and recall values of all topics. The topic `formula1`, on the other hand, seems to be the easiest to classify, as around 55.8% of the models `formula1` predictions are correct (precision) and the model is able to identify more than half of the `formula1` related tweets (recall = 0.562). This model achieves an accuracy of $\approx 98\%$ on synthetic and $\approx 34\%$ on real data.

	precision	recall
thanksgiving	0.946	0.979
formula1	0.986	0.977
covid19	0.986	0.986
championsleague	0.986	0.969
crypto	0.986	0.987
tesla	0.986	0.976
holidays	0.967	0.969

	precision	recall
thanksgiving	0.150	0.107
formula1	0.558	0.562
covid19	0.034	0.022
championsleague	0.339	0.443
crypto	0.333	0.304
tesla	0.412	0.319
holidays	0.387	0.670

Tables 3 & 4: Benchmark results of LDA model on synthetic data (left) and real data (right)

To assess the LDA model qualitatively, we sample some words for three selected topics from the model. We see that the `formula1` topic is coherent, generating words that are mostly related to the sport (like gt, saudi, community, race, performance, and max). The same holds true for the `crypto` topic. The word "number" appears multiple times as this is the token that replaced digits in the preprocessing, which is especially prevalent in crypto-related tweets ("+420% return", "10x your investment"). Only the samples for `tesla` don't fully seem to match the topic. We saw that the topics `tesla` and `crypto` are closely related, but this incoherent set of words might just be a bad random draw.

Topic	Sampled
Formula1	gt saudi mooning community holders goals race performance max spooky
Crypto	number market number th token journals sold dollar rate number
Tesla	media zippy wanted run potential vill assets disgusted think house

Table 5: Samples from LDA model for three selected topics

4.2 Neural Networks

4.2.1 Training on Synthetic Data Only

The neural network trained on synthetic data is almost able to achieve the same performance metrics as the LDA model, although it performs slightly worse for almost every topic. The generalization to real data, however, performs slightly better than the LDA, although it is noticeable that `formula1` and `holidays` seem to be the hardest topics to classify. The model achieves an accuracy of $\approx 95\%$ on synthetic and $\approx 35\%$ on real data.

	precision	recall
thanksgiving	0.939	0.952
formula1	0.977	0.930
covid19	0.972	0.973
championsleague	0.972	0.957
crypto	0.937	0.931
tesla	0.946	0.980
holidays	0.930	0.949

	precision	recall
thanksgiving	0.548	0.359
formula1	0.023	0.032
covid19	0.191	0.416
championsleague	0.601	0.521
crypto	0.689	0.353
tesla	0.336	0.490
holidays	0.102	0.152

Tables 6 & 7: Benchmark results of neural net (trained on synthetic data only) on synthetic data (left) and real data (right)

4.2.2 Training on Synthetic Data and Real Data

Next, we assess the predictive performance of a model that has been trained on the synthetic data set first, and then continues training on the real data set. While its performance on the synthetic data after finishing the training is noticeably worse than the models discussed before, it outperforms them on the real data set achieving consistently good performance across all topics. The model achieves an accuracy of $\approx 51\%$ on synthetic and $\approx 83\%$ on real data.

	precision	recall
thanksgiving	0.503	0.515
formula1	0.024	0.031
covid19	0.734	0.467
championsleague	0.866	0.670
crypto	0.596	0.691
tesla	0.765	0.675
holidays	0.094	0.238

	precision	recall
thanksgiving	0.883	0.830
formula1	0.791	0.890
covid19	0.840	0.809
championsleague	0.861	0.783
crypto	0.815	0.773
tesla	0.777	0.812
holidays	0.880	0.955

Tables 8 & 9: Benchmark results of neural net (trained on synthetic data and real data) on synthetic data (left) and real data (right)

4.2.3 Training on Real Data Only

Lastly, we evaluate the performance of the neural network model that is trained only on real data. While this model reaches the lowest predictive performance overall compared to the previous models on synthetic data, it generally performs the best on real data, achieving a slightly higher precision and recall value than the continued training model for most topics. The high performance on predicting topics in the real data set is consistent across all topics. The model achieves an accuracy of $\approx 42\%$ on synthetic and $\approx 84\%$ on real data.

	precision	recall
thanksgiving	0.384	0.500
formula1	0.058	0.049
covid19	0.485	0.390
championsleague	0.737	0.760
crypto	0.533	0.599
tesla	0.664	0.594
holidays	0.084	0.101

	precision	recall
thanksgiving	0.884	0.856
formula1	0.841	0.821
covid19	0.864	0.788
championsleague	0.818	0.862
crypto	0.854	0.739
tesla	0.727	0.860
holidays	0.878	0.963

Tables 10 & 11: Benchmark results of neural net (trained on real data only) on synthetic data (left) and real data (right)

4.2.4 A Look at the Learned Word Embeddings

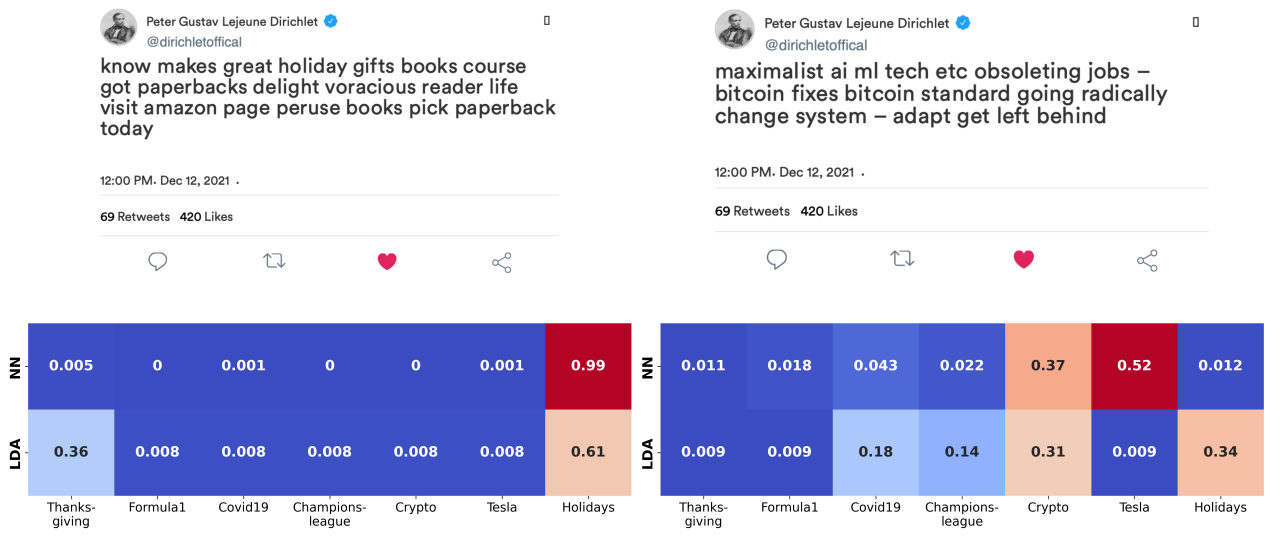
To verify that the neural network model has learned meaningful word embeddings, we feed it seven manually chosen words (one closely related to each topic, but never the name of the topic itself) and find the ten closest word in the embedding space using cosine similarity. We find that the word embeddings capture an impressive amount of meaning: The name of Formula 1 driver Lewis Hamilton is closest to other drivers' names as well as "formula" and "race". "Barcelona" maps to other sports clubs that plays in the Champions League. Similarly, we observe that "vaccine" is close to health- and covid-related words while "Christmas" is similar to "December" and "festive". For "Elon" and "BTC" the closest words are coherent and refer to Tesla and Cryptocurrencies. Only words associated with "turkey" are not clearly close to Thanksgiving, presumably as the topic `thanksgiving` has been hard to identify throughout the experiments. Nevertheless, the turkey emoji and "November" are related tokens. It is interesting to note that various relevant emojis have been identified as closely-related to each topic word, which confirms our initial hypothesis that emojis can be as meaningful as word tokens.

hamilton	barcelona	vaccine	christmas	turkey	elon	btc
max	bayernbara	immunity	december		musk	cryptos
ocon		vaccines	nicholas	nick	fsd	coins
lewis	zenit	deaths	festive	cotton	teslas	eth
vsc	barca	tests	visit	flex	ev	ssfeed
championship	bayern	measures	snowman	gratitude	supercharger	tether
abu	ucl	boris	pack	nov	binance	io
fia	liverpool	booster	wreath	rosemary	cointrade	opportunity
formula	milan	vaccinated	rainbow	chronicles	giga	dump
race	matchday	vaccination		skyrocket	autopilot	analyzing

Table 12: Top ten closest words in embedding space for seven examples.

4.3 Comparing Predictions

Finally, Figure 2 shows the predictions of both the LDA model and the neural network trained on real data only for two example Tweets. These examples demonstrate another difference between the prediction these models make. The LDA model returns a distribution over topics, where the assumption is that a document can (and often times will) contain multiple topics to some extent. Therefore, it tends to assign high probabilities to multiple topics, over which we take the $argmax(\cdot)$ to generate a single label. On the other hand, the prediction of the neural network is more clearly focused on one output, which is assumed to be the correct classification for a document. This behavior is encouraged by the Softmax function, which favors sparse activations. For the Tweet on the left, we see that the LDA and neural network agree that the most prevalent topic is `holidays`. On the right, however, the LDA detects multiple different topics, whereas the neural network is confident it is either `crypto` or `tesla` (two topics that are indeed quite similar), with `tesla` being the final prediction. Nonetheless, this example demonstrates that the LDA might have an advantage in detecting documents with multiple topics or documents that are hard to classify. Given that the task on hand is single-label classification, the $argmax(\cdot)$ function – while necessary to generate a label – destroys some valuable information.


Peter Gustav Lejeune Dirichlet
@dirichletoffical
know makes great holiday gifts books course got paperbacks delight voracious reader life visit amazon page peruse books pick paperback today
12:00 PM. Dec 12, 2021 .
69 Retweets 420 Likes

Peter Gustav Lejeune Dirichlet
@dirichletoffical
maximalist ai ml tech etc obsoleting jobs – bitcoin fixes bitcoin standard going radically change system – adapt get left behind
12:00 PM. Dec 12, 2021 .
69 Retweets 420 Likes

	Thanks-giving	Formula1	Covid19	Champions-league	Crypto	Tesla	Holidays
NN	0.005	0	0.001	0	0	0.001	0.99
LDA	0.36	0.008	0.008	0.008	0.008	0.008	0.61

	Thanks-giving	Formula1	Covid19	Champions-league	Crypto	Tesla	Holidays
NN	0.011	0.018	0.043	0.022	0.37	0.52	0.012
LDA	0.009	0.009	0.18	0.14	0.31	0.009	0.34

Figure 2: Predictions for two example Tweets

5 Discussion

Our results indicate that model performance is determined by a combination of the type and quality of data the model is trained on and the expressive power of the model. Our models that trained only on synthetic data performed well on predicting on synthetic test data but poorly on the real data, and vice versa. In addition, the more expressive neural networks performed better to varying extents than the LDA model in predicting topics in the real data.

5.1 Data Type and Quality

We observe a noticeable discrepancy between predictive performance of the models that were trained on different types of data (i.e. synthetic vs real). Specifically, the models trained on synthetic data perform better on synthetic data while performing poorly on real data. And similarly, the models trained on real data perform poorly on synthetic data. This can be partially explained by the LDA generative model being a bag-of-words model and, therefore, unable to preserve sentence structure and generate sequential data. In addition, the LDA model infers seven topics from our corpus, independently of the actual topic labels, and we had to manually match our predetermined labels to the inferred topics. The labelling process worked well for the more distinct topics (e.g. `formula1`, `tesla`), but turned out to not perfectly recreate the topics with more overlap with others (i.e. `covid19` with all other topics and `thanksgiving` with `holidays`). Therefore, the sampled synthetic data generated from our LDA model would be dissimilar from the real data, resulting the predictive performance on synthetic data to not generalize to real data and vice versa.

Here, we note that the two neural networks, one trained on synthetic followed by real data and the one trained only on real data exhibit similar performance on both the real holdout test data, and the synthetic holdout test data. This could be because continuing training on real data makes the model un-learn relationships in the artificial data set. This phenomenon is known as *Catastrophic Forgetting* and is an ongoing area of research (e.g. Kaushik et al. 2021).

5.2 Model Expressiveness

When comparing the LDA and the neural network models trained on synthetic data, we observe that both these models perform equally and extremely well on the holdout dataset. This was expected as the LDA classifier should be able to easily learn the data it generated itself, while the neural network has strong expressive power to imitate and learn similar parameters as the LDA classifier. These two models exhibit similar poor performance on real data, despite some differing performances across topics. This is also to be expected as the LDA outputs an entire topic distribution for each document. From this topic distribution, we only assign the most prevalent topic as a label, which is a rather simple approach. Similarly, the neural network overfits on the synthetic training data, which does not generalize well as discussed above. To truly differentiate the LDA and the neural network, we compare the predictive performance on real data between the LDA and the neural network model trained exclusively on real data. From this comparison, we observe that the neural network strongly outperforms the LDA (i.e. 84% accuracy vs 34% respectively). This is because real world data is the ideal scenario for neural networks to learn complex sentence structures, and it becomes evident that a simple bag-of-words approach is an inferior method that is unable to utilize essential components of the data, like the sentence structure.

While the neural network is clearly the better method when comparing only performance, we cannot say that it is the strictly better method. The LDA is much simpler when comparing time to train, both from a computational as well as a human practical outlook. The LDA has no hyperparameters that need to be tuned or carefully managed, allowing a greater volume of data to be fed into the model train step. The neural network also demands far more resources. The LDA model took approximately 1.5 minutes to run on a CPU, whereas the neural network took around 22 minutes to train on a CPU, and three minutes on a Nvidia Tesla T4 GPU. This is in addition to other working difficulties when working with complex models, like unexplained poor performances that are platform variant, additional step implementation to control for overfit, and subjective decisions made during the design of the architecture. LDA is also a better choice in scenarios where parameter interpretation is important. The parameters learned by LDA are easily interpretable. Parameters such as probability of a word belonging to a particular topic are intuitive in meaning. The neural network is much more of a black box, and the individual weights of a perceptron have no human-understandable interpretation.

6 Conclusion

In this project, we collected data from Twitter by scraping tweets for seven topics (hashtags). We used the data on these topics to build a generative model, benchmark its performance, and used it to generate synthetic data. Based on this synthetic data we trained several different neural networks. Our results indicate that the simple LDA model generates data that might not fully align with real world Tweets. While the LDA excels when used on synthetic data, it fails to perform on the real data set. The same holds true for the neural network that has been trained on synthetic data only. However, the neural networks we built in in this project perform well on real data *if they are trained on real data* achieving accuracies of around 84%.

For these neural networks we were able to show that the learned word embeddings managed to capture the expected word-topic relationships. Given the large number of unique words induced by slang and misspelled words that are common on Twitter, we still think training our own word embeddings over using a pre-trained set is adequate.

Future work on this problem could include experimenting with different generative models to produce synthetic data that more closely aligns with real data. In addition, we could also explore different regularization techniques: We see that the neural networks achieve training accuracies of over 95% on the training set, which does not fully reflect in their ability to generalize to the validation and test set.

References

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `https://www.tensorflow.org/`.

Akiba, Takuya et al. (2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *CoRR* abs/1907.10902. arXiv: `1907.10902`. URL: `http://arxiv.org/abs/1907.10902`.

Bergstra, James et al. (2011). "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems* 24.

Chung, Junyoung et al. (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555. arXiv: `1412.3555`. URL: `http://arxiv.org/abs/1412.3555`.

Kaushik, Prakhar et al. (2021). *Understanding Catastrophic Forgetting and Remembering in Continual Learning with Optimal Relevance Mapping*. arXiv: `2102.11343 [cs.LG]`.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Smith, Leslie N (2018). "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay". In: *arXiv preprint arXiv:1803.09820*.

twintproject (2021). *TWINT - Twitter Intelligence Tool*. URL: `https://github.com/twintproject/twint`.