Documentation

Introduction

In this documentation I will show a solution to the problem of sentiment analysis. More specifically determining whether a short passage of text is negative or positive in nature. I will present 3 techniques which attempt to solve this problem. The first uses word embeddings. The second uses a rule-based system and the third uses a gradient boosting model. The three techniques show drastically different results proving their ability to solve the problem.

	target	ids	date	flag	user	tweet
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
...	...	...	...	...	...	...
1599995	4	2193601966	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	AmandaMarie1028	Just woke up. Having no school is the best fee...
1599996	4	2193601969	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	TheWDBoards	TheWDB.com - Very cool to hear old Walt interv...
1599997	4	2193601991	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	bpbabe	Are you ready for your MoJo Makeover? Ask me f...
1599998	4	2193602064	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	tinydiamondz	Happy 38th Birthday to my boo of alll time!!! ...
1599999	4	2193602129	Tue Jun 16 08:40:50 PDT 2009	NO_QUERY	RyanTrevMorris	happy #charitytuesday @theNSPCC @SparksCharity...

Figure 1. Overview of the Twitter sentiment dataset

The dataset I will be using to solve this problem is the sentiment 140 dataset from kaggle [3]. This consists of 1.6 million labelled tweets. The tweets are labelled either 4 (positive) or 0 (negative). The neutral label is not present in the dataset even though the kaggle page says neutral is a present label. Figure 1 shows the dataframe when importing the dataset. As you can see there are some columns that are unneeded and will be removed during implementation. The tweets can be seen having problematic characters in them such as '@' and links. These will also be removed in implementation. The longest tweet is 374 characters long while the shortest is 6. This shows the dataset consists of short form passages which are appropriate to solve the problem statement.

Literature Review and Technologies Used

Word Embeddings

Word embeddings are a type of tokenisation to convert words into a language that a machine can understand. Word embeddings can be created in a variety of ways. One of these is word2vec [8] where words are converted into vectors where similar words like man and women are near each other in the vector space. This is useful to bring meaning to words so that the machine can understand the link between them. However, different words can mean different things depending on what surrounds them in a sentence. Word2Vec fails to represent these contextual differences. A method that attempts to solve this uses

transformers to produce word embeddings. These models use self attention layers to assess the context of a word and change its position in vector space based on that. First the sentence is split into words. These words are then converted into numbers using a dictionary. Position embeddings are added onto these words to represent their location within the sentence. These newly formed numbers will then go through the transformer encoder layers where they will be adapted so that each number (representing one word) will contain the information and context of all the words around it. These context dense ‘embeddings’ are then outputted to use in many problems. In this case sentiment analysis.

As mentioned in the section above word embeddings can be generated by transformers. However, transformers are memory intensive models and can be slow to use. For this reason I chose to use two distilled versions of the transformer BERT [9]. I used DistillBERT [5] and all-MiniLM-L12-v2 [7] [6]. These models are smaller than the original BERT model and are trained to achieve similar performance to the BERT model using knowledge distillation. DistillBERT holds 97% of BERT's performance. MINILM outperforms DistillBERT. These models are smaller and take up less memory allowing for it to fit on a single commercial GPU. Although this sounds fast these smaller models are still slow compared to the other techniques in this documentation.

I will be using these distilled models to create a new dataset consisting of labels and tweets that have been translated into word embeddings using these distilled models. After this I will ‘fine tune’ the models by training a small classifying network to classify the word embedding as positive or negative using supervised learning.

VADER

VADER (Valence Aware Dictionary for sEntiment Reasoning) is a rule-based system that can detect sentiment within small passages of text [1]. It works by assessing the sentiment of each word in a sentence and tallying up the positive and negative words leading to an overall score of positive or negative. It pre-processes words heavily to try to reach their base word and fits that word into a positive or negative basket. This technique is extremely quick, however, as it is rule based these rules might not generalise well to all passages and context is not taken into account when producing results.

XGBoost

XGBoost is a gradient boosting technique [10] that leverages a collective of bad performers to create a better performing system. First many classification models are trained such as KNN, SVM and Logistic regression models. These models perform badly on their own but are assembled together using a decision tree. This decision tree uses the outputs of each model to get a final output being the classification of the input text.

Implementation

Dataset Pre-Processing

I took steps to pre-process the data as the text included words and text that would not be beneficial to pass through the models. I first removed the html tags from the text using regex. Then I removed any usernames as twitter allows you to '@username' someone. This adds some context but these usernames are not in the models vocabulary and hold almost no meaning to the model hence the removal of them. Then I removed URLs and translated slang words to their full form. These slang words such as 'WBU' were translated using a dictionary found online. For example, 'WBU' translates to 'what about you'. I was not able to capture all the abbreviations as there are too many to keep track of. Next I removed punctuation and leading white spaces that might be present. Finally, I lemmatized the text. This is the process of changing the forms of one word such as 'walked' or 'walks' to a base form 'walk'. This helps reduce the different amounts of words but removes some information that could be useful. This preprocessing was able to clean most of the data, however, tweets suffer from mis-spelling. Many tweets contain mis-spelt words leading to the lemmentasize process not converting the words. I tried using a spell correction library. It worked for some words but not all and was extremely slow so I chose not to use it.

Transformer

My first solution involved creating a new dataset consisting of transformer made embeddings to use as input into a classifier model instead of just text. I pre-processed the tweets using the methods outlined above. Then I processed around 100,000 tweets through the two transformer models I outlined in my literature review. I chose 100,000 tweets as processing all 1.6 million would take too long to process. I had to process these tweets in batches as the embeddings took up large amounts of space in memory. DistillBERT needed to be processed in batches of 256 and MiniLM in batches of 1024. These numbers were based on the largest number of embeddings I could fit in memory at one time. I also processed a further 10,000 through both models to use as a testing/validation set.

Now the embeddings were created I trained the classifier model. I had to train two different models for each transformer model as they had different sized output layers. Results of training are outlined in the evaluation section. The two models had 4 layers to start with. The input layer differed based on the input model. 384 for MiniLM and 768 for DistillBERT. The input layer is followed by a dense linear layer with the size of 128 then another linear layer with a size of 64 then 32 then finally an output layer of 1 with a sigmoid activation function. I used a leaky_relu as activation for all hidden layers and binary cross entropy as a loss function.

VADER

The implementation for VADER was simple. As it is a rule-based system requiring no training I pre-processed the data then fed it to the model sentence by sentence taking the compound output of the model as the prediction. This compound output is a representation of how negative or positive it thinks the input is. A negative number being a negative sentiment and a positive number being a positive sentiment.

XGBoost

I was not able to train the XGBoost model using the transformer embeddings as they take up alot of memory. I chose to use the tf-idfvectorizer which converts words to a vector version of themselves. I trained the model on the same tweets used for the transformer technique. I used the xgboost library to implement the classifier.

Evaluation and Hyperparameter Tuning

Transformer Fine-Tuning

I experimented with different hyper-parameters for the fine tuning of the classifier model. I limited the variables I chose to change as there are an infinite number of variables you can change to an infinite number of values so limiting what I changed allowed me to evaluate the model in a reasonable way and in a reasonable amount of time.

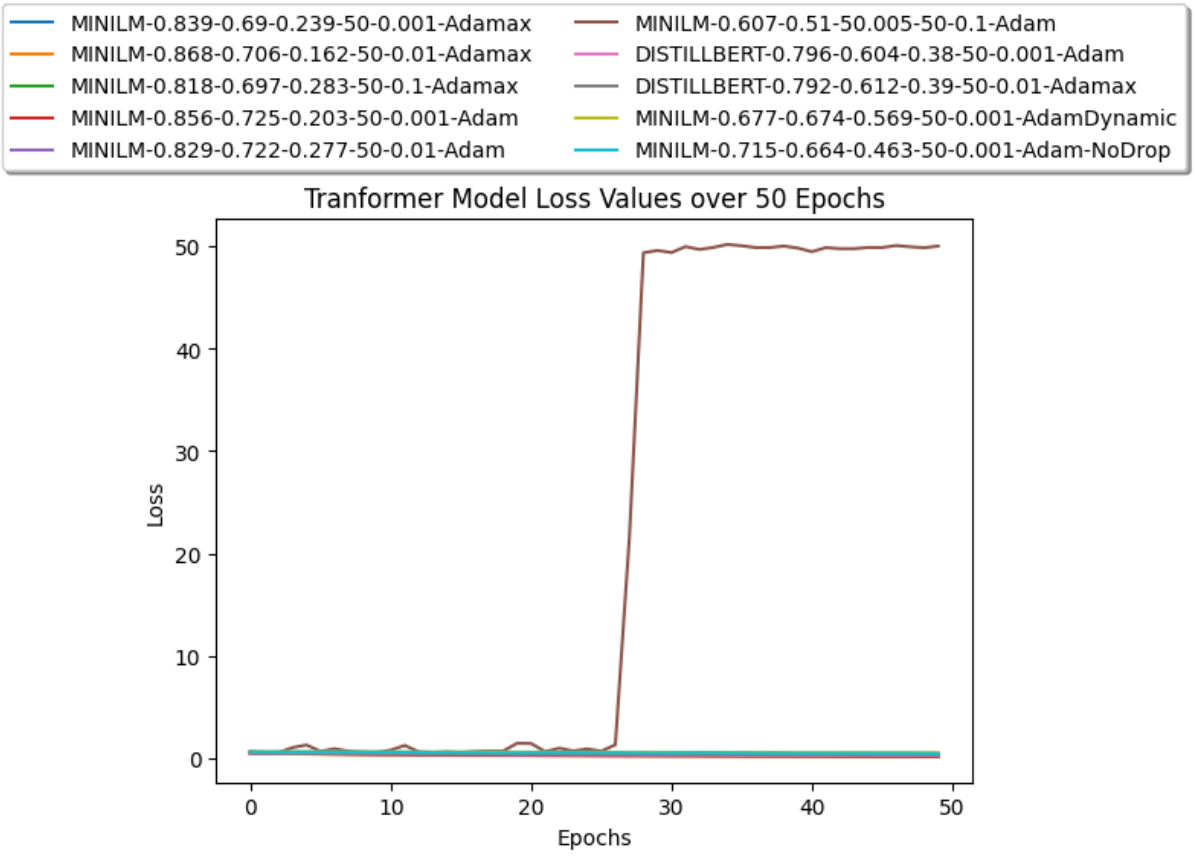

Figure 2. Initial testing of hyper parameters over 50 epochs

Figure 2 shows the first experiments I tried. This involved changing the optimizer used, learning rate and removing dropout from the model. The model name is formatted as so: EMBEDDING MODEL USED - FINAL TRAINING ACCURACY - FINAL VALIDATION ACCURACY - FINAL LOSS VALUE - TOTAL EPOCHS - LEARNING RATE - OPTIMIZER.

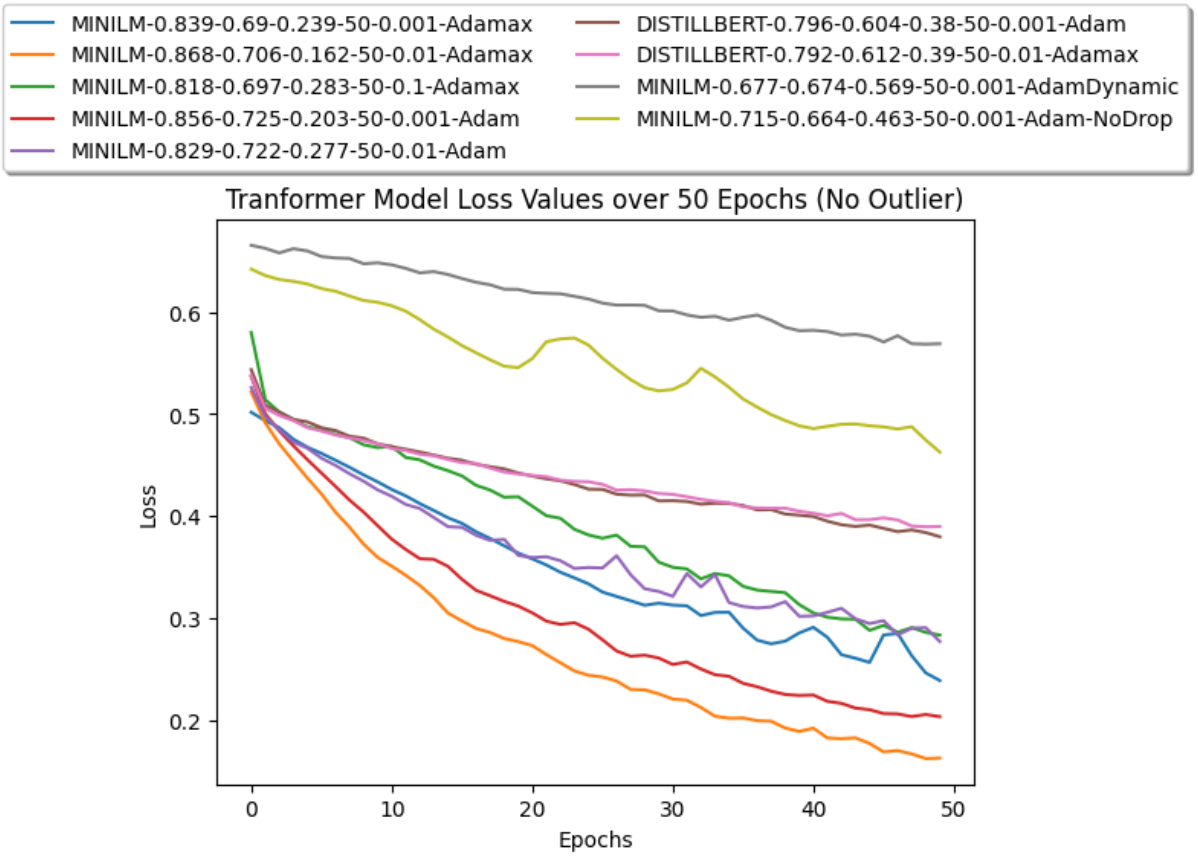

Figure 3. Shows loss over 50 epochs for models

This graph is hard to read as one of the models spiked in loss and did not recover. For this reason I removed this model from the graph to gain a better insight into the data. Figure 3 shows the graph with no outlier model. The ‘adam dynamic model’ produced the worst loss line. This model had a dynamic learning rate dividing by 10 at a set amount of epochs. The next worst is the ‘Adam-NoDrop’ model. This model has its dropout removed. The next worst were the two ‘DISTILLBERT’ models proving it performs worse than the MiniLM models. The rest of the lines represent models with different constant learning rates and optimizers. The ‘0.01-Adamax’ model performed the best with the ‘0.001-Adam’ model having the second best loss curve. I chose to take these two models for further analysis.

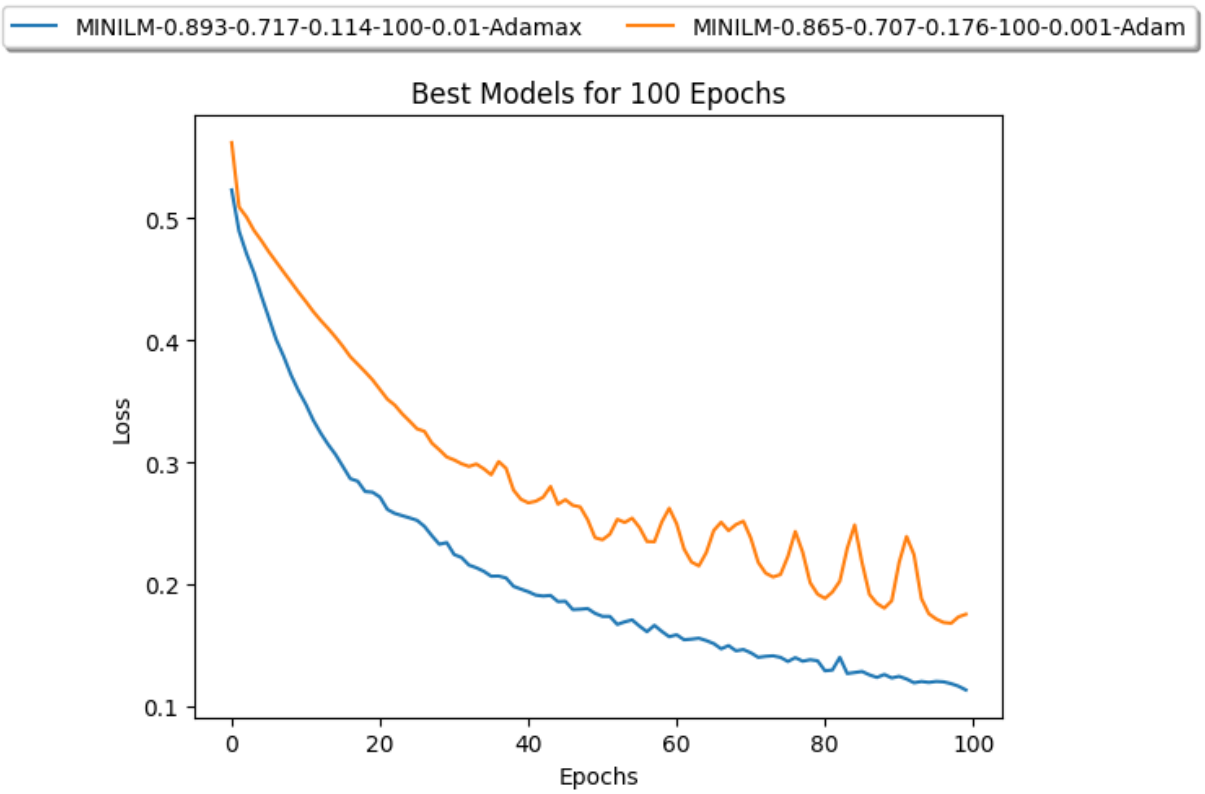

Figure 4. Shows best performers over 100 epochs

Figure 4 shows the two best models running for 100 epochs instead of 50. They show steady decreases in loss towards convergence which is promising. The ‘adam’ model shows a more dynamic line while the ‘adamax’ model’s line is smoother. This could suggest the loss landscape is hard to traverse for the ‘adam’ model. This could also be a result of the start position. As seen at epoch 0 the two models start in different positions in the loss landscape which may lead to drastically different journeys to convergence leading to the ‘adam’ model having a more dynamic journey to convergence.

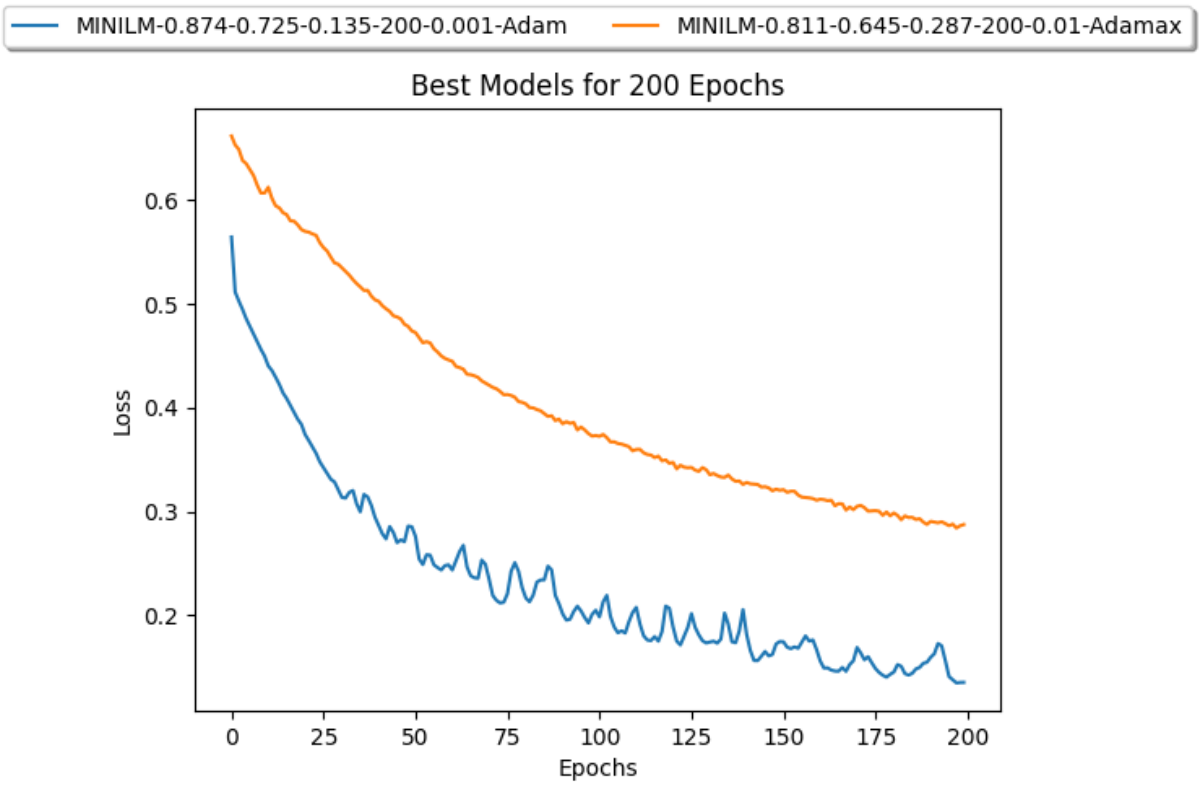

Figure 5. Shows best performers over 200 epochs

The success of the previous experiment led me to run the best models for 200 epochs to see whether they would reach convergence. This was successful for the ‘adam’ model but not for the ‘adamax’ model. I think this was due to the starting value of each model. It could have also been due to the ability of the models to traverse this loss space. From figure 5 it can be concluded that the loss space is quite dynamic. This leads to an optimizer with a better fit to this kind of landscape performing better.

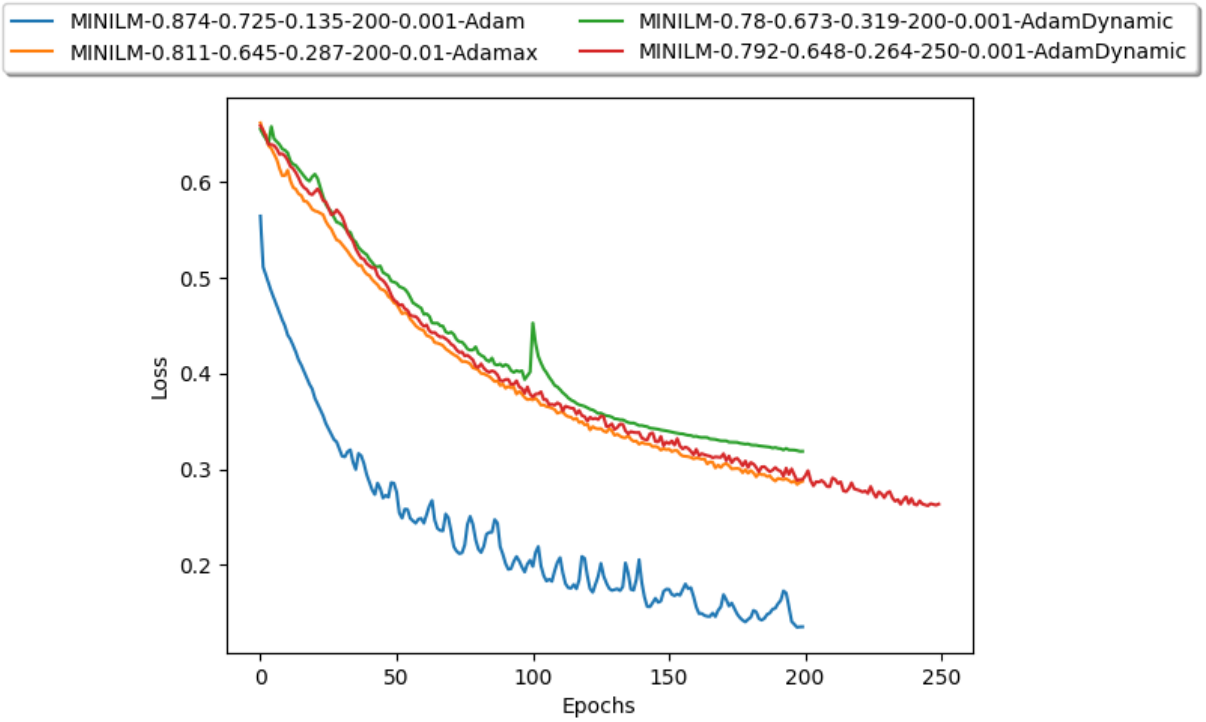

Figure 6. Dynamic learning rate compared to bst performers

I noticed that the best ‘adam’ model bounced around alot when converging especially when getting lower in loss value. For this reason I attempted to add a dynamic learning rate to try to slow the steps taken by the optimizer when reaching a lower loss. This was to hopefully reduce the bouncing of the line. As seen in figure 6 this did work for that purpose but ended in the model not dynamic models not being able to reach a loss anywhere near as low as the original model. I think it was due to the starting value of the model.

Overall I set different values for a limited amount of hyper-parameters which could be a limitation of my testing. However, I think this shows a good process of fine-tuning to get a better model with the final model having significantly better validation performance than the starting model (60% for the worst model and 72% for the best).

VADER

VADER is a rule based system with no training so the only fine-tuning available is to add more pre-processing to text before being run through the model. However, I found the performance good enough with the pre-processing I used for the other models. VADER was able to get 100% accuracy of the test and training data. This is a testament to how well rule-based systems can perform. It was also faster than the classifier model outlined above as it did not have to train.

XGBoost

I tested different parameters on the XGBoost model such as changing the n_estimators as well as changing the max_depth the tree could get to. However, the model's performance stayed at around 49% showing it was worse than guessing at this task. I am not sure why this model did not perform as it has been shown to be a good classifier. I did not have enough time to fully look into and solve this problem so the xgboost model ended up being the worst performing model.

Ethics and Real Life Applications

This solution can be applied to real life situations to benefit customer satisfaction. Using this system reviews and customer feedback can be sorted into positive and negative without having to ask the customer for their star rating or satisfaction. This means a large amount of reviews can be processed and analysed to find out what you need to improve about your product or service to make your customer happy.

This can also be used for moderation of social media sites. Although this system is quite primitive only allowing a binary classification you could imagine another system that takes in the negatively classified data and moderates it to make sure it is still appropriate. Moderation is expensive for companies and a system like this can save the company money.

Leaving an automated system to decide whether information is positive or negative is potentially unethical. Training data is biassed and therefore the system will hold a bias towards certain things which could enable it to target certain phases or types of speech as more negative than others or more positive than others even if it objectively isn't. This could be harmful when applying the system to real life situations if the system is able to take meaningful action towards the parties producing the data.

Conclusion

In conclusion I think this documentation shows 3 potential approaches to solving the problem with 2 promising solutions reached. The documentation demonstrates the effectiveness of word embeddings and rule-based systems for the problem of binary sentiment analysis and highlights the weakness of gradient boosted systems.

References

[1] Hutto, C., & Gilbert, E. (2014). 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. https://ojs.aaai.org/index.php/ICWSM/article/view/14550/14399

[2] Kumar, S., Roy, P., Dogra, D., & Kim, B.-G. (2023). *A Comprehensive Review on Sentiment Analysis: Tasks, Approaches and Applications*. https://arxiv.org/pdf/2311.11250.pdf

[3] Μαριος Μιχαηλιδης KazAnova. (2017). *Sentiment140 dataset with 1.6 million tweets*. Www.kaggle.com. https://www.kaggle.com/datasets/kazanova/sentiment140/data

[4] Reimers, N. (n.d.). *Pretrained Models — Sentence-Transformers documentation*. Www.sbert.net. Retrieved November 28, 2023, from https://www.sbert.net/docs/pretrained_models.html

[5] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. https://arxiv.org/pdf/1910.01108.pdf

[6] *sentence-transformers/all-MiniLM-L12-v2 · Hugging Face*. (2023, June 8). Huggingface.co. https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2

[7] Wang, W., Li, F., Hangbo, D., Yang, B., & Zhou, M. (2020). *MINILM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers*. https://arxiv.org/pdf/2002.10957.pdf

[8] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. https://arxiv.org/pdf/1301.3781.pdf

[9] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. https://arxiv.org/pdf/1810.04805.pdf

[10] Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. https://arxiv.org/pdf/1603.02754.pdf