LLM VS Human Generated Text: Report

Andrew Cohn, Noah Sher, Benjamin Seckeler, Bryan Bielawa

December 11, 2024

Single Perceptron with Word2Vec Model Analysis (Noah)

I used only a single perceptron to implement a binary classification model to predict whether given texts were AI (bot) generated or original texts. The actual training does not start until after I scaled the data to ensure uniformity, I converted the words into embeddings using Google's pre-trained Word2Vec model and averaged them out for each text. Random weights and bias are initlized for the model, with the single perceptron being trained over 20 epochs by a stochastic gradient descent, which updates parameters after each iteration rather than once on the entire dataset. I then provide an evaluation with sklearn's classification report.

A single perceptron model is good for binary classifications. One reason for this is because the model learns a linear decision boundary between two classes, and in this case, I only needed to put texts under the two classes: "bot" or "not bot." I used pre-trained Word2Vec embeddings because it has the advantage of already learning from previous data to classify texts with more accuracy.

The classification report at the end showed the precision (96%), recall (96%), F1-score (96%), and the accuracy (96%)of the entire model. Overall my model had a 96% accuracy. I think the reason for such high scores is due to the simplicity of the model, mainly with using just a single perceptron. On top of that, using an already trained Word2Vec embedding model helped with accuracy because of its prior knowledge with embedding other data. I also think the training data used to train the model helped with finding patterns in the data that help classify text as AI generated or not much easier.

Perceptron Bag of Words Analysis (Ben)

The file **src/models/P_BOW/P_BOW.ipynb** implements a bag of words based perceptron classifier for identifying LLM generated text. This model breaks each entry in the dataset into a vector of word counts which it then uses to train a binary classifier. Data is loaded in by the BotFinderDataset class. The data is then used to train a single perceptron linear classifier via the PerceptronModel class. This class implements a single linear perceptron per class. Training is performed using an Adam optimizer. Once training is completed, the model is evaluated on the testing dataset and its performance is recorded. All data generated from this model is stored in **src/models/P_BOW/P_BOW.csv**.

Throughout the notebook, all configurations and performance metrics are saved to the disk by using an instance of the ModelResults class (implemented in **src/utils/recorder_util.py**).

The performance of this model was quite poor. A confusion matrix and evaluation statistics from evaluation of the best iteration of the model on testing data is shown below:

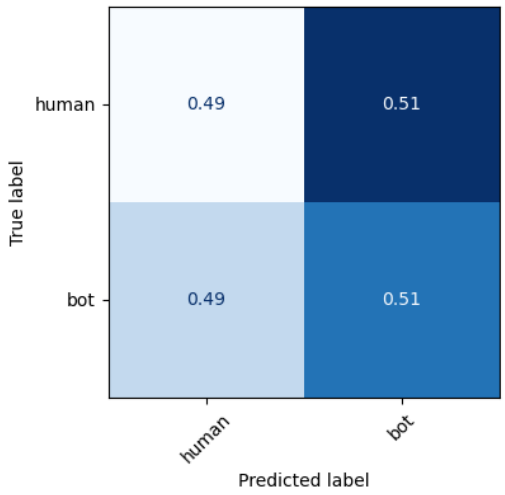

Figure 1: Confusion matrix for perception bag of words

Class	Precision	Recall	F1-Score	Support
Human	0.479	0.488	0.484	3435
Bot	0.515	0.506	0.510	3435
Accuracy	0.497 (6632 instances)			
Macro Avg	0.497	0.497	0.497	6632
Weighted Avg	0.498	0.497	0.497	6870

Table 1: Classification report for bag of words perceptron

The model's poor performance is most attributable to the poor separability of the BoW features generated from the text. This tells us that LLM generated text generally will use words of simulary frequency as human generated text — a trend that will continue in future BoW models. This is something we would expect as LLM are trained to predict next words on human generated text, implying that the corpus level word frequencies in LLM generated text would be very similar to human generated text.

To investigate the frequencies of certain tokens across our training dataset, we extracted the top 10 highest weighted tokens used by the model presented above. These are the results:

Top 10 words that for classification as human:

student_name	0.06168070808053017
teacher_name	0.05392399802803993
although	0.050340887159109116
etc	0.050271764397621155
very	0.04901818186044693
method	0.04817750304937363
everyday	0.047019198536872864
text	0.04650139436125755
boxing	0.04615069180727005
china	0.045328784734010696

Top 10 words that for classification as bot:

re	0.05849381163716316
hey	0.05404694005846977
cool	0.05289079621434212
ensure	0.05162741616368294
writing	0.05103255435824394
conclusion	0.0510084442794323
super	0.049321796745061874
remember	0.04905489459633827
plus	0.04869118332862854
firstly	0.04866937920451164

The occurrence of placeholder words in our training data such as "student_name" and "teacher_name" likey distorts this model's performance. This was an issue found later in the project. Once these tokens were removed from the training/test data performance improved. This will be explored in more detail in a later section. The prefix 're–' and the word 'hey' and 'cool' were also more common in the LLM generated texts. The higher frequency of 'cool' and 'hey' suggests that when these models were generating text they may not have understood that their essay response was to be written more formally, so they included more vernacular language which was different from the human generated writing. These frequencies also hint at a broader issue with our dataset — its very specific domain of student essays. We see this in the use of the word 'conclusion' — a word that might not appear very often in other forms of text generated by an LLM (e.g. social media posts or short stories).

Feed Forward Neural Network with Bag Of Words (Ben)

The file **src/models/FNN_BOW/FNN_BOW.ipynb** implements a bag of words based feedforward neural network classifier for identifying LLM generated text. This model breaks each entry in the dataset into a vector of word counts which it then uses to train a binary classifier. Data is loaded in by the BotFinderDataset class. The data is then used to train a feedforward linear classifier implemented in the BinaryFFNModel class. This class supports an arbitrary number of dense layers, uses ReLU activation, and integrates a dropout hyperparameter.

Training is performed using an Adam optimizer. Once training is completed, the model is evaluated on the testing dataset and its performance is recorded. Throughout the notebook, all configuration and performance metrics are saved to the disk by using an instance of the ModelResults class (implemented in **src/utils/recorder_util.py**). All data generated from this model is stored in **src/models/FNN_BOW/FNN_BOW.csv**. This model performed quite poorly. A confusion matrix and evaluation statistics from evaluation on testing data is shown below:

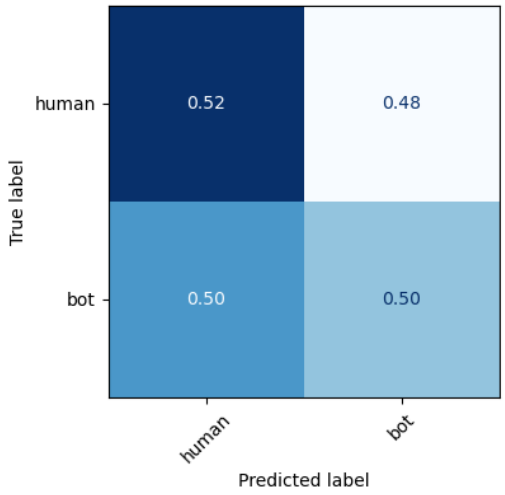

Figure 2: Confusion matrix for FFNN bag of words

Class	Precision	Recall	F1-Score	Support
Human	0.507	0.517	0.512	3435
Bot	0.507	0.497	0.502	3435
Accuracy		0.507 (6870 instances)		
Macro Avg	0.507	0.507	0.507	6870
Weighted Avg	0.507	0.507	0.507	6870

Table 2: Classification report for bag of words FFNN

The model's poor performance is again attributable to the poor separability of the BoW features generated from the text. The confusion matrix looks more as expected with the model being roughly equally unable to classify all

texts. This improvement is a result of the model's increased number of nodes, letting it better fit the data.

Model Performance Recording Library (Ben)

Early in this project, we decided to build a unified method of storing information about our models and their performance. Our solution to this is implemented in src/utils/recorder_util.py. This file includes the ModelResults class. This class is used to maintain a record on the disk of hyperparameters and model performance. This class is responsible for holding information about the model's training / eval time, its performance, and its hyperparameters. This data is then written out to the disk. An example of data output by this class is shown below:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	model	timestamp	seed	accuracy	f1_macro	f1_micro	precision_macro	precision_micro	recall_macro	recall_micro	duration_training	duration_te	performance_full	hyperparameters_full	author	notes
2	feed_forward_bow	1732144988	1234	0.5	0.333333	0.333333	0.25	0.25	0.5	0.5	488.399454	0	{'human': {'precision	{'seed': 1234, 'dropout':	ben_secekeler	

Figure 3: Example CSV report

Hyperparameters are formatted as follows:

```
{‘HYPERPARAMETER NAME’ : VALUE, …}
```

For example:

```
{'seed': 1234,
'dropout': 0.3,
'layers': [68854, 100, 50],
'learning_rate': 1e-06,
'weight_decay': 0.0001,
'batch_size': 500,
'shuffle': True,
'n_epochs': 5}
```

Performance metrics are formatted as output by PyTorch. As shown below:

```
{ ‘CLASS 0’: { ‘METRIC 0’ : VALUE,  ‘METRIC 1’ : VALUE, … }, …, {GLOBAL METRIC : VALUE, …}}
```

For example:

```
{'human': {'precision': 0.5, 'recall': 1.0, 'f1-score': 0.6666666666666666,
'support': 3435.0}, 'bot': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0,
 'support': 3435.0}, 'accuracy': 0.5, 'macro avg':
{'precision': 0.25, 'recall': 0.5, 'f1-score': 0.3333333333333333, 'support': 6870.0},
 'weighted avg': {'precision': 0.25, 'recall': 0.5,
'f1-score': 0.3333333333333333, 'support': 6870.0}}
```

FFNN Word2Vec Report (Bryan)

This model trains a binary classifier to detect the presence of AI written text. The file **src/models/FFNN_W2V/FFNN_W2V.ipynb** implements a Word2Vec based feedforward neural network classifier for identifying LLM generated text. This model

uses pre-trained Word2Vec model embeddings. The model uses 2 layers in its architecture and loads the data set into the custom MyDataset class. We ran our model through 8 epochs. With the training being finished, the final f1 score was 0.885. More epoch during training likely would have raised this value further, but issues with overfitting likely would have gotten worse.

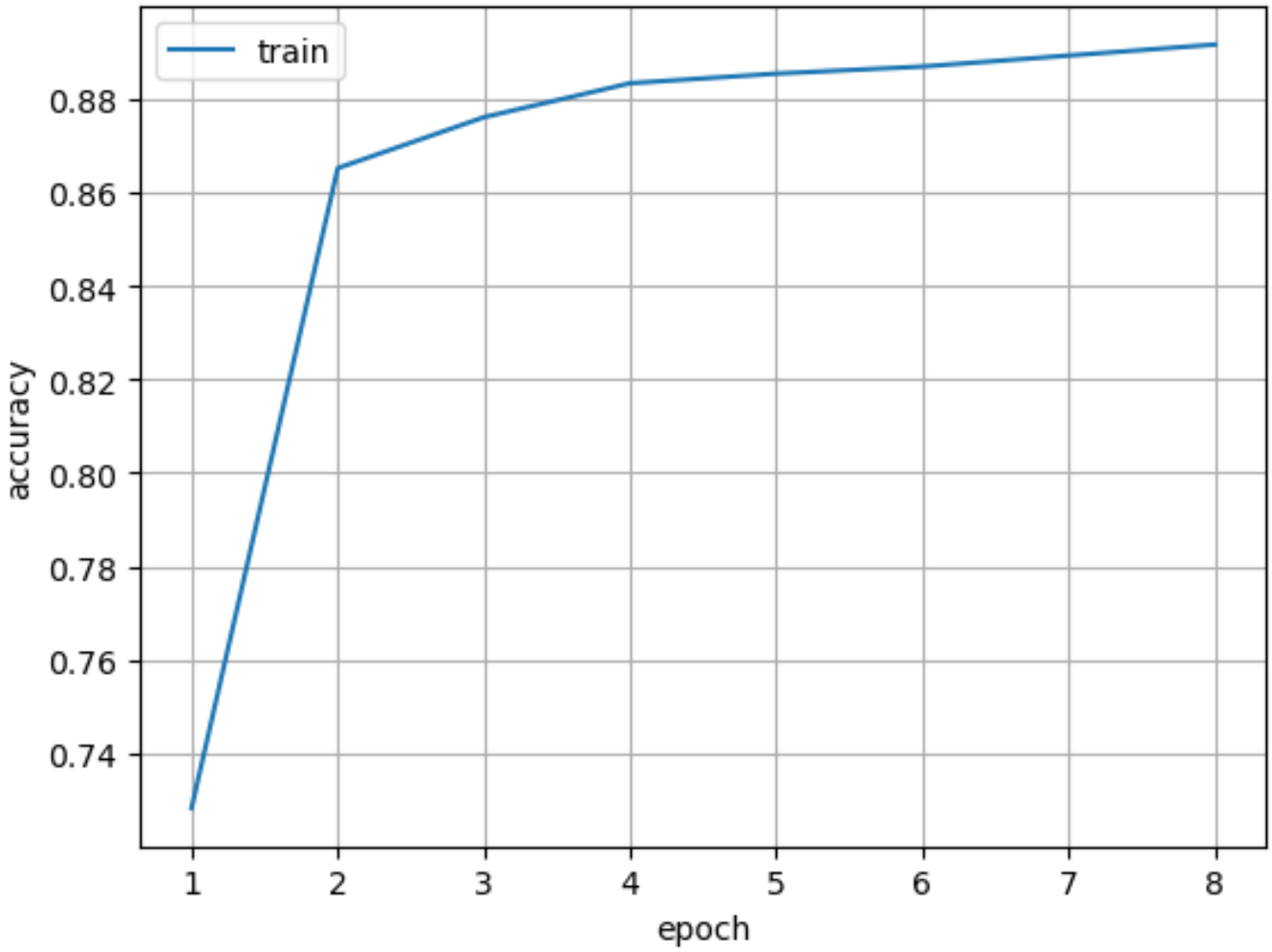

Figure 4: Training accuracy graph

Why this worked so well

The main thing may be that the architecture of the model worked well for this application when combined with word2vec. However, we used 100 and 50 neurons in our hidden layers which may have allowed for identification of complex patterns. Additionally 8 epochs seems like the sweet spot for this model in particular as more would give less and less returns for the time spent running this. Below is the classification report for this model:

Class	Precision	Recall	F1-Score	Support
Human	0.80	0.99	0.88	3435
Bot	0.98	0.75	0.85	3435
Accuracy		0.87 (6870 instances)		
Macro Avg	0.89	0.87	0.87	6870
Weighted Avg	0.89	0.87	0.87	6870

Table 3: Classification report for FFNN with W2V embeddings

As with all the other models we also ran into an issue with our training and testing data containing tokens such as “student_name” or “teacher_name” in most of the human written text. Upon correcting this issue we found that the model results did not change at all. As a conclusion from this adjustment we can conclude that outside of the overfitting issue this model was largely accurate in its classifications.

LSTM with Bag of Words Analysis (Andrew)

Overview

We had pretty low expectations for this model. The strength of the LSTM lies in it’s ability to process sequential data. However, bag of words is only concerned with the count of words in the input data- the relative ordering of the words is not considered at all. As such, our feature extraction and our model architecture are working against each other. These factors made this model our worst performing model by far.

Technical Details

The file **src/models/LSTM_BOW/LSTM_BOW.ipynb** contains this model. First, we preprocess the text to make it nicer, and then, we use a count vectorizer to extract the counts of the words as our features. Then, we pass this into an LSTM (implemented in the LSTMModel class) with a sequence length of 1. We train the LSTM, then have it classify from our test dataset. This gave the following results:

Class	Precision	Recall	F1-Score	Support
Human	0.44	0.21	0.28	3435
Bot	0.48	0.74	0.58	3435
Accuracy		0.47 (6870 instances)		
Macro Avg	0.46	0.47	0.43	6870
Weighted Avg	0.46	0.47	0.43	6870

These are not particularly good- worse than even the bag of words perceptron. The reason the simpler model performed better in this case is because the simpler model was not directly at odds with the feature extraction mechanism.

LSTM with Word2Vec Embeddings Analysis (Andrew)

Overview

We had very high hopes for this model. Firstly, we were beginning to approach a transformer, which is the current industry standard for ai detection. In addition, since Word2Vec encoded data maintains the relative order of the words, we could actually leverage the strengths of the LSTM, and allow it to consider longer sequences.

Technical Details

The file **src/models/LSTM_W2V/LSTM_W2V.ipynb** contains this model. First, the we load the precomputed GLOVE embeddings, and then encode our essays using these embeddings. After that, we pass it to an almost idential LSTM as the other model (except this one has a longer sequence length), and train. After training, we get the following results:

Class	Precision	Recall	F1-Score	Support
Human	0.94	0.98	0.96	3435
Bot	0.98	0.93	0.95	3435
Accuracy		0.96 (6870 instances)		
Macro Avg	0.96	0.96	0.96	6870
Weighted Avg	0.96	0.96	0.96	6870

Table 4: Classification report for the LSTM model with W2V embeddings

This was our most performant model, with the exception of the super perceptron. However, much like the super perceptron, it is highly unlikely the exceptional performance on this data set will translate nicely to real world data- our data set is very nicely curated- which is a dual edged sword, as it means statistical models will perform nicely on our data set, but won't reflect real trends.

A note about overfitting (Andrew)

Upon discovering that the bag of words models were hyper focusing on the tokens ”student_name” and ”teacher_name” (which, unfortunately, was after our presentation), we adjusted the preprocessing to remove these tokens, to avoid leaving obvious markers and retrained our models. The new largest weights for the FFNN with bag of words is as follows:

Top 10 words that for classification as human:

although	0.0542895682156086
very	0.05100160464644432
im	0.047695696353912354
everyday	0.047035399824380875
many	0.0467977300286293
itself	0.04654332995414734
etc	0.04620114713907242
almost	0.046155497431755066
emotion	0.046127576380968094
baseball	0.04549897089600563

Top 10 words that for classification as bots:

cool	0.05308391526341438
re	0.05249093472957611
hey	0.05190479755401611
conclusion	0.050896529108285904
urge	0.050181593745946884
writing	0.04995037615299225
commitments	0.04974362626671791
super	0.049633245915174484
yeah	0.048913486301898956
essay	0.04829477518796921

Across the board, we saw slight increases in the perforamce of the bag of words models, and the word2vec models were largely unaffected by the change. This suggests that it is likely that the bag of words models were relying on those tokens to detect human text, and removing them handicapped those models, while the networks which use word2vec were fitting to those particular tokens significantly less.

Conclusions

We found that as our models became more complex, they performed better on our dataset, with the exception of the perception models, which quickly succumbed to overfitting. In the future, a transformer model may perform better, as this is the industry standard. In addition, acquiring more data, via online comments, emails, etc, would allow us to build a more robust model. In addition, we need to be wary of data poisoning- or misclassified elements of our dataset which impede performance.