CSE842 HW3

Javen W. Zamojcin
zamojci1@msu.edu

2024, November 22

1 Problem 1: Fine-tune a Pretrained Model

1.1 A

I used the HuggingFace Transformers library along with its PyTorch-based Trainer library for fine-tuning the model.

1.2 B

The pre-trained model I fine-tuned was `distilbert-case-uncased`.

1.3 C

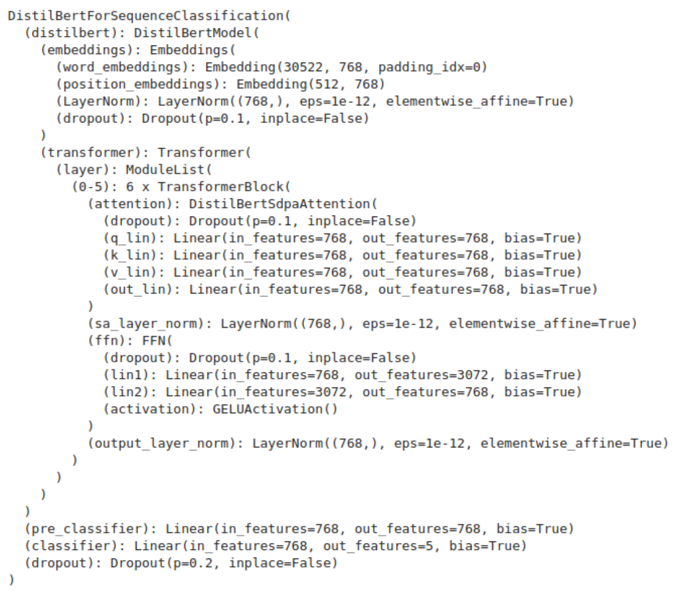
See Figure 1 for model summary.

1.4 D

Training Output:

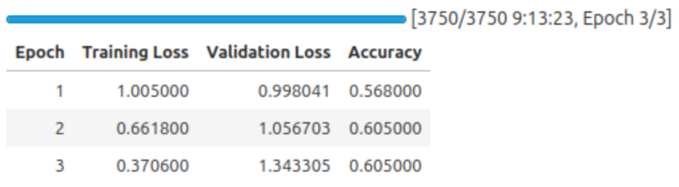
- `global_step=3750`
- `training_loss=0.66000`
- `train_runtime=33212.5936`
- `train_samples_per_second=0.903`
- `epoch=3`
- `total_flos=39742345728000000.0`

See Figure 2 for training history. After three training epochs, the model evaluated to an accuracy of 0.605000. I used the same data subset size and training parameters as the tutorial, with the only difference being I used DistilBERT instead of original BERT. I'm not sure why my model performance was significantly worse.



```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): DistilBertSdpaAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

Figure 1: Q1.3: Model Summary



[3750/3750 9:13:23, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	1.005000	0.998041	0.568000
2	0.661800	1.056703	0.605000
3	0.370600	1.343305	0.605000

Figure 2: Q1.4 Training Results

1.5 E

The primary thing I learned from this section was becoming familiarized with the Transformers library and getting a scope for the many different and useful tools it provides.

2 Problem 2: Outperform BERT

2.1 How To Run

The program for this part was implemented using `Python 3.10` in the included Jupyter notebook titled `HW3_Q2.ipynb`. All libraries used and their respective versions may be found in the included Conda environment `env.yml` file.

2.2 A

The NLP task I chose was Text Classification, specifically whether a given document was machine-generated or not. I thought this task would be interesting to investigate what patterns exist in LLM generated text and how easily they can be detected. Additionally, the general problem of detecting whether media was generated by AI is becoming increasingly relevant so I wanted to familiarize myself with the solutions.

2.3 B

The dataset I chose for this task is the M4GT-Benchmark. This dataset is appropriate as it includes around 152K samples of text generated by either human sources or LLM sources. Around 65K are human generated from wikihow, reddit, arxiv, wikipedia, and peerread. The remain samples were generated uniformly from chatGPT, gpt4, davinci, bloomz, dolly, and cohere. For the scope of this project, I'm only doing a binary classification of whether it was human or machine generated.

Additionally, a multitude of models have already experimented on this dataset, including both BERT-based and various statistical models, which allows me to compare my own results and set goal posts.

2.4 C

The model I used was a SVM provided by the `Sklearn` library. I implemented several SVM model iterations using different various features and hyperparameters.

2.5 D

In my first iteration, I used only Count/TFIDF vectorization for the linear SVM features. The count vectorizer used the parameters of 3000 max features, a minimum frequency of 2, a maximum frequency of the 70th percentile, and a n-gram range of $(1, 1)$.

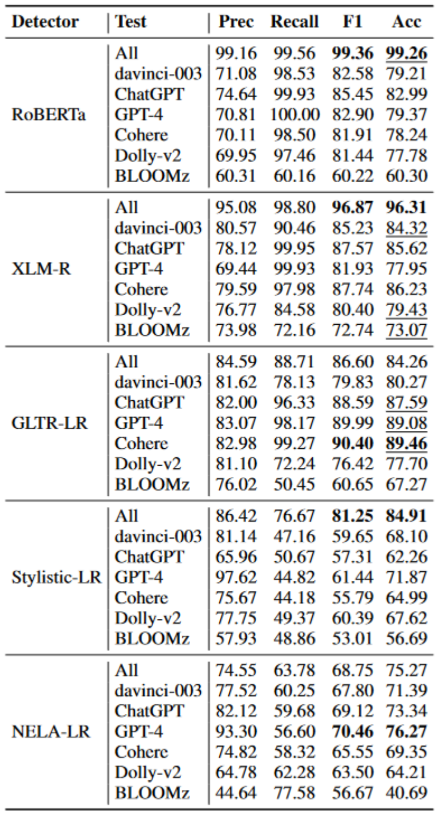
In my second iteration, I used only Word2Vec embeddings with PCA for the linear SVM features. The parameters used were a padded sequence length of 15, a vector size of 100, a window size of 5, a minimum frequency of 1, and a PCA sum of variance ratios of 0.99 (1288/1500 principal components). Additionally, due to resource limitations, I used a subset sample size of $10,000$.

In my third and final iteration, I used a combination of Count/TFIDF vectorization and Word2Vec embeddings for the RBF SVM features. I implemented this by independently calculating both the TFIDF and Word2Vec embeddings and then concatenating them together column-wise $(3000 + 1288$ features). The

Count/TFIDF vectorization parameters remained the same from the first iteration. The Word2Vec parameters remained the same from the second iteration except that $25,000$ samples were used instead.

2.6 E

See Figure 3 for the SOTA model performances on this benchmark.



Detector	Test	Prec	Recall	F1	Acc
RoBERTa	All	99.16	99.56	**99.36**	**99.26**
	davinci-003	71.08	98.53	82.58	79.21
	ChatGPT	74.64	99.93	85.45	82.99
	GPT-4	70.81	100.00	82.90	79.37
	Cohere	70.11	98.50	81.91	78.24
	Dolly-v2	69.95	97.46	81.44	77.78
	BLOOMz	60.31	60.16	60.22	60.30
XLM-R	All	95.08	98.80	**96.87**	**96.31**
	davinci-003	80.57	90.46	85.23	84.32
	ChatGPT	78.12	99.95	87.57	85.62
	GPT-4	69.44	99.93	81.93	77.95
	Cohere	79.59	97.98	87.74	86.23
	Dolly-v2	76.77	84.58	80.40	79.43
	BLOOMz	73.98	72.16	72.74	73.07
GLTR-LR	All	84.59	88.71	86.60	84.26
	davinci-003	81.62	78.13	79.83	80.27
	ChatGPT	82.00	96.33	88.59	87.59
	GPT-4	83.07	98.17	89.99	89.08
	Cohere	82.98	99.27	**90.40**	**89.46**
	Dolly-v2	81.10	72.24	76.42	77.70
	BLOOMz	76.02	50.45	60.65	67.27
Stylistic-LR	All	86.42	76.67	**81.25**	**84.91**
	davinci-003	81.14	47.16	59.65	68.10
	ChatGPT	65.96	50.67	57.31	62.26
	GPT-4	97.62	44.82	61.44	71.87
	Cohere	75.67	44.18	55.79	64.99
	Dolly-v2	77.75	49.37	60.39	67.62
	BLOOMz	57.93	48.86	53.01	56.69
NELA-LR	All	74.55	63.78	68.75	75.27
	davinci-003	77.52	60.25	67.80	71.39
	ChatGPT	82.12	59.68	69.12	73.34
	GPT-4	93.30	56.60	**70.46**	**76.27**
	Cohere	74.82	58.32	65.55	69.35
	Dolly-v2	64.78	62.28	63.50	64.21
	BLOOMz	44.64	77.58	56.67	40.69

Figure 3: Q2.6 SOTA Models

2.7 F

See Table 1 for experiment results. All three model experiments were evaluated and averaged using 3-folds cross-validation.

Iteration	Accuracy	Recall	Precision	F1
1	0.887768	0.904635	0.900156	0.902390
2	0.610867	0.659757	0.768751	0.710068
3	0.726229	0.986138	0.713071	0.827657

Table 1: Q2.7 Experiment Results

2.8 G

2.8.1

My first iteration (TFIDF features only) performed the best of the three, and even out-performed the SOTA non-NN models (see Figure 3) on this benchmark.

2.8.2

I think the first iteration model could be even further improved by increasing the max features from 3000. But again, computational resource limitations were problematic for me in this project.

I was surprised that the Word2Vec embeddings performed significantly worse. But it is likely that using data subsets for training and evaluation played a major role in this. I still suspect that given the entire dataset for training, the TFIDF + Word2Vec embeddings models could perform much better than just using TFIDF.

2.8.3

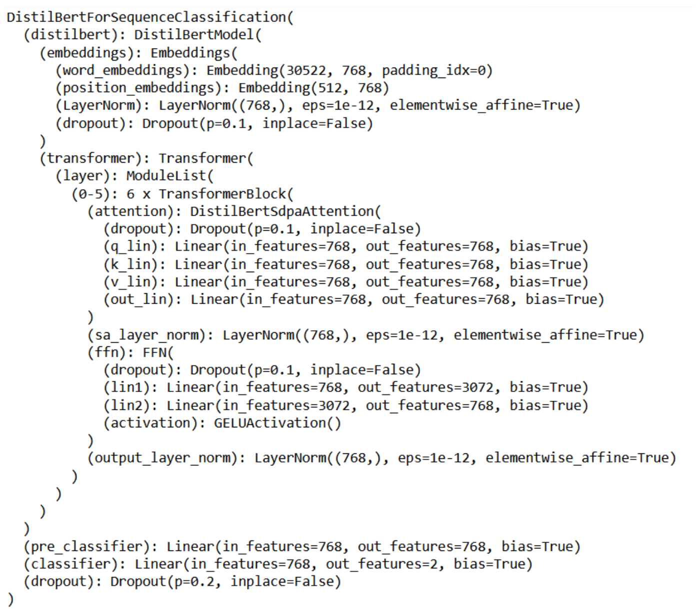
It is difficult to completely compare my approaches to the SOTA BERT model without first dedicating more computational resources to training. While the token frequency features seem to capture a majority of the text patterns, there are likely still nuanced text patterns that are more appropriately captured by the Transformer architectures.

3 Problem 3: Last 10 Points

I choose to fine-tune `DistilBERT` to solve the machine-generated text classification problem. The final evaluation accuracy was 0.941, significantly better than my non-NN models. See Table 2 for the training history, and Figure 4 for the model summary.

Epoch	Training Loss	Validation Loss	Accuracy
1	0.260500	0.423286	0.888000
2	0.099700	0.231532	0.950000
3	0.023800	0.352535	0.941000

Table 2: Q3: Training History



```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): DistilBertSdpaAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

Figure 4: Q3: Model Summary