4NL3 Assignment 4

Guneev Arora - 400477579

April 1, 2025

Contents

1 Dataset

1.1 The Task

To begin with, I chose the SMS Spam Detection task which using the ucirvine/sms_spam dataset by Tiago Almeida and Jos Hidalgo.
I chose this dataset as it seemed very interesting to me and wanted to play around with it. But, another reason I chose this was because of the size being pretty small and easy to work with.
This dataset has one collection composed by 5,574 English, real and non-encoded messages, tagged according being legitimate (ham) or spam with a file size of 359 kB.
With the goal obviously being to find out whether SMS message is a spam message or not a spam message.

1.2 Data collection

In terms of data collection, I got my information from the UC Irvine Machine Learning Repository. Based on what I read here is how they collected the data.(Copied and pasted from the website to keep accuracy)

This corpus has been collected from free or free for research sources at the Internet:

- A collection of 425 SMS spam messages was manually extracted from the Grumbletext website. This is a UK forum where cell phone users publicly report SMS spam messages, most of them without including the actual spam message received. Identifying the text of spam messages in these claims was a very challenging and time-consuming task, involving careful scanning of hundreds of web pages. The Grumbletext website is: http://www.grumbletext.co.uk/.
- A subset of 3,375 SMS randomly chosen ham messages from the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans, mostly students attending the University. These messages were collected from volunteers who were informed that their contributions would be made publicly available. The NUS SMS Corpus is available at: http://www.comp.nus.edu.sg/~rpnlpir/downloads/corpora/smsCorpus/.
- A list of 450 SMS ham messages collected from Caroline Tagg's PhD Thesis, available at http://etheses.bham.ac.uk/253/1/Tagg09PhD.pdf.
- The SMS Spam Corpus v.0.1 Big, which includes 1,002 SMS ham messages and 322 spam messages. It is publicly available at: http://www.esp.uem.es/jmgomez/smsspamcorpus/. This corpus has been used in several academic research studies.

1.3 Input, Output & Metric Information

1.3.1 Data information

To begin with the Input, as mentioned before the dataset has a total of 5,574 text/SMS messages from the websites mentioned above with labels 0 and 1 where 0 is ham and 1 is spam for training

and just the text/SMS when testing.

For the output, as you would expect, the same two labels of 0 and 1 are used for the output as well for classification where the model would take the message as input and return the label as output when testing.

1.3.2 Metric information

For this I used the basic forms of metrics which are accuracy, precision, recall and f1 score.

This was done using the evaluate library as it was easy to use and worked well on this task. This was used in a compute_metrics function that was passed into the trainer that I will talk about later.

1.4 Data split

As this dataset was not manually split, I split it myself into an 80% training and 20% testing split.

Split	Number of Samples	Percentage
Training	4,459	80%
Testing	1,115	20%
Total	5,574	100%

Table 1: Data split

2 Fine-tuned models

2.1 Model information

To begin with, for my 2 models, I decided to use models Bert and roBERTa from the Hugging Face model hub linked in the assignment document.
I decided to use the BERT model as they were used in the tutorial and was easier to work with due to the tutorial.
I used roBERTa as it was based on BERT but still different enough to make me curious as to how each of these would perform against each other.

Both of these are MLM models and are models that are meant to be fine-tuned on a downstream task.

2.2 Fine-tuning steps

For the fine-tuning of both of these models, I used Trainer from Transformers along with TrainingArguments and the data split and compute metrics mentioned in the previous section.

For the arguments, I used 3 epochs with a batch size of 8, learning rate of 2e-5 and decay of 0.01. I used these as they were given in the tutorial and gave me the best results after playing around with the hyperparameters.

2.3 Model size

Model	BERT (bert-base-uncased)	roBERTa (roberta-base)
Number of parameters	110M	125M
Number of layers	12	12
Vocabulary size	30522	50265

Table 2: Model size table

2.4 Pretraining compute requirements

Before I begin, we can tell that roBERTa is just a better version of BERT as it has the same dataset of BERT with a few more datasets and some other modifications that will be shown bellow.

2.4.1 BERT

Training data:

- BookCorpus: A dataset consisting of 11,038 unpublished books.
- English Wikipedia: Excluding lists, tables, and headers.

Preprocessing steps:

- Texts are lowercased and tokenized using WordPiece and a vocabulary size of 30,000.
- Takes input in form: $[CLS]$ Sentence A $[SEP]$ Sentence B $[SEP]$
- With probability 0.5, sentence A and sentence B correspond to two consecutive sentences in the original corpus, and in the other cases, it's another random sentence in the corpus.

Pretraining steps:

- Trained on 4 cloud TPUs in Pod configuration (16 TPU chips total)
- One million steps with a batch size of 256
- Sequence length was limited to 128 tokens for 90% of the steps and 512 for the remaining 10%
- Adam optimizer with a learning rate of 1e-4, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and decay = 1e-6
- Learning rate warmup for 10,000 steps followed by linear decay

2.4.2 roBERTa

Training data:

- BookCorpus: A dataset consisting of 11,038 unpublished books.
- English Wikipedia: Excluding lists, tables, and headers.
- CC-News: A dataset containing 63 million English news articles crawled between September 2016 and February 2019.
- OpenWebText: An open-source recreation of the WebText dataset used to train GPT-2.
- Stories: A dataset containing a subset of CommonCrawl data filtered to match the story-like style of Winograd schemas.

Preprocessing steps:

- Texts are tokenized using a byte version of Byte-Pair Encoding (BPE).
- Vocabulary size of 50,000.
- 512 contiguous tokens in the input.
- $< s >$ and $< /s >$ used to mark document start and end respectively.

Pretraining steps:

- Trained on 1024 V100 GPUs
- Five hundred thousand steps with a batch size of 8,000
- Sequence length of 512
- Adam optimizer with a learning rate of 6e-4, $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon =$ 1e-6 and decay = 0.01
- Learning rate warmup for 24,000 steps followed by linear decay

2.4.3 Masking

The masking strategy for both models were the same and done this way:

- 15% of the tokens are masked.
- In 80% of the cases, the masked tokens are replaced by `[MASK]`.
- In 10% of the cases, the masked tokens are replaced by a random token (different from the one they replace).
- In the remaining 10% of cases, the masked tokens are left as is.

The difference is that roBERTas masking is done dynamically when pretraining while BERT uses a static masking strategy.

3 Zero-shot classification

3.1 Models

3.1.1 Models used

For this part of the assignment, I used models Deepseek-1.5B and TinyLlama-1.1B as linked in the Hugging Face page linked on the assignment.
I chose these as my laptop was able to run models at around 1-2B parameters and were still decently big models that can give me different results. Both of these are text generation models as requested for the zero-shot classification task.

3.1.2 Model size

Model	Deepseek-R1-1.5B	TinyLlama-1.1B
Number of parameters	1.78B	1.1B
Vocabulary size	128k	32k
Dataset size	14.8T	950B

Table 3: Model size table

3.1.3 Pretraining

Deepseek-R1-1.5B

- Used 2048 H800 GPU which took 3.7 days to pretrain
- Can be done in 2.788M H800 GPU hours
- Optimized the pre-training by enhancing math and programming ratio
- Does not incorporate cross-sample attention masking
- Uses Fill-in-Middle (FIM) rate of 0.1
- Number of Transformer layers is 61
- Hidden dimension is 7168
- uses the AdamW optimizer

If interested, you can find more information in this paper: https://arxiv.org/pdf/2412.19437

TinyLlama-1.1B

- Used 16 A100-40G GPUs for 90 days
- Uses mixture of 70% SlimPajama and 30% Starcodercode for 3 epochs
- Pre-norm and RMSNorm used for normalization
- SwiGLU activation function used

- Grouped-query Attention used for attention
- Uses AdamW optimizer

If interested, you can find more information in this paper: https://arxiv.org/pdf/2401.02385

3.2 Prompts

Now, for the prompting part of this assignment, I tried to use many different kinds of prompts and also minor changes to other prompts whenever I thought it needed some.

Here are some of the types and examples:

- No prompt: Just gave it the text with label names.
- Small prompt: Gave it basic instructions on what labels mean in less than 15 words.
- Long prompt: Gave a prompt with over 30 words on what labels are as well as instructions.
- Basic prompt: Just told it what to do.
- Updated basic prompt: Updated the small prompt to be more specific and clear.

Based on those prompts, I had many attempts of using them, all leading to different results.
The longer prompt seemed to give me the lowest accuracy most the time as I think the model was confused if all the other text was a part of the spam or not.
The small prompt seemed to help but still didn't give me what I wanted as it had a bias to one side of the labels.
No prompt seemed to do as good as the small prompt but made random errors.
Finally, the basic prompt seemed to do a lot better at understanding the classification task and was giving me better results so modifying it at the end to give me the best results I got throughout.

Prompt examples:

- "Label the message after the colon as 'spam' or 'not spam': " + text (What I ended up using)
- "Label whether the following message is spam or not spam: " + text
- "Is this spam?: " + text
- "Label the SMS message: " + text
- "Classify a text message, between 'spam' and 'not spam'. Spam messages usually promote something bad or have fake links, numbers or emails. Here is the text message: " + text + "(newline) Is this spam or not spam?"

Thus I ended up using the first example as it gave me the most accurate results.

4 Baseline

Now for the baseline, I used a TF-IDF model along with Logistic Regression for the classifier using TfidfVectorizer from the sk-learn library.
For this, we used the vectorizer to get our training and testing data in the right format. We then run the LogisticRegression function from the sk-learn library to create the model with the default 100 max iterations as after some testing it did not seem to make much of a difference.
Finally, we train our model with the training messages and labels and can then start testing it by testing it with our testing dataset.
Ofcourse, with that we are able to compare our results and able to calculate accuracy and other metrics using classification_report from the sk-learn library as mentioned before.

I also tested how the results are compared to the Majority and Random baseline classifiers to compare by using the max function for the majority and random function for the random classifier.

I was shocked by how well the Majority classifier did, but it made sense with the data we have access to as most of the examples were not spam.

5 Results

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
BERT (Fine-tuned)	0.99193	0.99286	0.94558	0.96864
roBERTa (Fine-tuned)	0.98924	0.95302	0.96598	0.95946
BERT (not Fine-tuned)	0.38027	0.15482	0.82993	0.26096
roBERTa (not Fine-tuned)	0.86816	0.0	0.0	0.0
Deepseek-R1-1.5B (Zero-shot)	0.65919	0.24045	0.71812	0.36027
TinyLlama-1.1B (Zero-shot)	0.43229	0.13222	0.58389	0.21561
TF-IDF + Logistic Regression (BoW)	0.97578	0.98387	0.82993	0.90037
Random Baseline	0.49955	0.13106	0.49659	0.20739
Majority Baseline	0.86816	0.00	0.00	0.00

Table 4: Final results

So, as you can see, our original fine-tuned models did the best by a lot compared to everything other than the Logistic Regression Model.
We can also tell that bias based results worked well with 85% accuracy due to the dataset being very unbalanced.

Based on everything I did, the not fine-tuned models ended up taking longer than the models to run and didn't even have good results, this makes me doubt in the ability of zero-shot classification. Logistic Regression having a 97% accuracy and 98% precision was surprising as I thought it would be a lot lower.
Logistic Regression was also the fastest to run and had really good results and in my opinion the best because of amount of resources used, time and accuracy, precision but slightly worse Recall and F1-score, though if you have resources available fine-tuning a model is easily your best bet.

6 Reflection

6.1 What I learned

This assignment was a lot of fun! I got to learn a lot of fine-tuning models, on how to be able to fine tune them, as well as get better results by doing this. I learned how powerful it can be to fine-tune with a new dataset as it increases dataset accuracy by a lot and is not too complicated to do again later.

Furthermore, I also learned how to Zero-shot classification and how prompt engineering can make such a big difference for all of these massive models. Overall it was really cool to get some hands on work using big models as well as smaller ones to check their difference.

6.2 What was unexpected

I was super surprised on how much of a difference 3 epochs can make to a dataset and how much it can learn from that. As well as, the amount a small dataset can affect such a big dataset with minor updates and finetuning.

I was also caught off guard by how much prompt engineering can affect the results of the classification as well as how accurate Logistic Regression can be off using the TF-IDF bag of words model.

6.3 Challenges & how I overcame them

The part I struggled the most with was easily the zero-shot classification as I kept getting false or one-sided results when testing, taking me a long time to figure out the perfect prompt to get decent results.

I was able to overcome this, by spending a lot of time just trying different prompts on small subsets of the dataset and checking the result and accuracy.

7 AI Tool Usage

For AI tools, I used ChatGPT, I did around 37 queries which each are 4.26g = 157.62 grams of CO2 total with all of them being queries on some coding questions but not directly copying code and some other questions on certain topics.