NLP Phishing Checker

Applied Machine Learning Final Project 2025

Wiktoria Domańska
Filip Niewczas

Introduction

Phishing - cyber attack to steal your information, usually from fake emails and messages.

Fake emails might vary in terms of emotions, sentiment and even grammar mistakes

Goal is to build a model that can distinguish between Phishing and Safe emails, just by analyzing the email's text

Dataset

17,539 emails total

Each email labeled as: Phishing Email or Safe Email

Real-world examples (text + HTML noise)

Imbalanced: ~37% phishing, ~63% safe

	Email Text	Email Type
15615	6th manchester phonology meeting - programme p...	Safe Email
6021	create a new credit file legally in 30 days ! ...	Phishing Email

Dataset Challnges

- Duplicate emails
- Noisy HTML content
- Class imbalance (solved by undersampling)
- Long text sequences ↣ truncation needed

```
–
List allows for
UNLIMITED DOWNLOADS!
Â Â Â
See this product's web page
CLICK HERE
Â
FAX MARKETING SYSTEM
– Fax broadcasting is the hot new way to market your product
or service!– People are 4 times more likely to read faxes than direct
mail.– Software turns your computer into a fax blaster with 4
million  leads on disk!
Â Â Â
See this product's web page
CLICK HERE
Â
Visit our web site or
call 618–288–6661 for more information.
Â
Â
Â
Â to be taken off of our list
click here
",Phishing Email
```

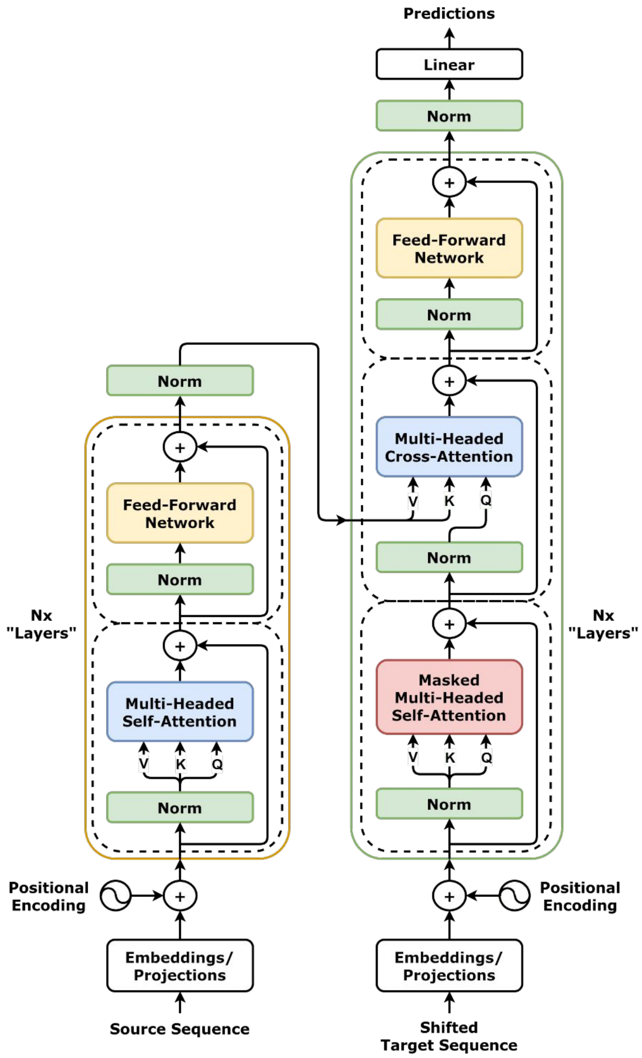
Model

distilbert-base-cased (pretrained) (light version of BERT)

Transformer-based architecture

Binary classification head

Hugging Face Trainer API



Fine tuning

🛠️ Training Setup

- 3 epochs, batch size = 6
- Gradient accumulation = 2
- Learning rate: 5e-5
- Weight decay: 0.02
- Warmup steps: 50

⚙️ Optimization

- AdamW optimizer
- Early evaluation per epoch
- Save best model checkpoint
- Mixed precision enabled (fp16)
- Logged with MLflow

Model Problems, Performance & Accuracy

📊 Final Accuracy

- 98.0% on validation set
- Loss: ~0.09
- Very strong generalization

🧪 Evaluation

- Balanced test set (50/50)
- Accuracy + manual testing
- Prompt-based stress testing

🚨 Model Weaknesses

- False positives: aggressive marketing emails
- False negatives: short, vague phishing

Model behaviour, testing and prompring

🧠 Model Usage

- Deployed as Hugging Face pipeline
- Input = free text,

  output = label + confidence
- Real-time prediction without retraining

🧪 Testing Strategy

- Manual prompting: short, long, tricky emails
- Creative test cases (e.g. "OMG click this!")
- Observed confidence scores

📌 Observations

- Very confident on obvious phishing
- More uncertain on neutral/ambiguous emails
- Reacts well to certain keywords (e.g. “click”, “urgent”, “password”)

```
sample_text = '''
Hey, what's up? Can we meet tommorow evening? I have something important to discuss with you. I will send you a link in a minute.
'''
```

```
'SAVE EMAIL', 'score': 0.8692324757575989},
'PHISHING EMAIL', 'score': 0.13076746463775635}]
```

```
sample_text = '''
Look at my kittens at this link: www.kittens.com
'''
```

```
'PHISHING EMAIL', 'score': 0.9818893671035767},
'SAVE EMAIL', 'score': 0.018110565841197968}]
```

Alternatives and Model limitations

🅰️ Alternatives Considered

- RoBERTa ⤐ stronger, but slower
- GPT-based models ⤐ needs more data & compute
- Classical ML (e.g. SVM) ⤐ worse on noisy text
- LLMs ⤐ overkill for binary classification

⚠️ Known Limitations

- Needs English input
- Handles plain text only
- Sensitive to text truncation
- Doesn't explain why it flagged phishing

Conclusion

🔍 What We Learned

- Transformers can detect phishing with minimal data
- Model performance depends more on prep than model size
- Prompt-based testing reveals strengths and flaws
- NLP = flexible, real-time solution for text classification

🌍 Real-World Relevance

- NLP already powers spam filters, smart replies, GPTs
- Email security is critical in finance, gov, personal use
- Our model could be deployed in browsers, inboxes, helpdesks
- Challenges: interpretability, multilinguality, adversarial attacks

Live Demo and Q&A

We can try some examples, feel free to suggest something!

The model will respond instantly with its prediction and confidence score.

Appendix

All participants contributed equally to the work.

Full TrainingArguments

batch size limited due to Colab RAM; warmup stabilizes early training.

```
#%pip install accelerate -U
training_args = TrainingArguments(
    output_dir="./phishing-email-detection", #"./phishing-email-detection"
    logging_dir='./logs',
    num_train_epochs=3, # 3
    per_device_train_batch_size=6, #16 - number of samples to process at once per batch
    per_device_eval_batch_size=6, #16
    gradient_accumulation_steps=2,  # Added this line to fix the MPS error
    logging_strategy='steps', # log every step
    logging_first_step=True,
    load_best_model_at_end=True, #trainer will load the best model found during training at the end of training
    logging_steps=1,
    eval_strategy='epoch',# when evaluate model - after each epoch
    warmup_steps=50, #50
    weight_decay=0.02, #0.02
    eval_steps=1,
    save_strategy='epoch',
    report_to="mlflow",  # log to mlflow
)


# Define the trainer:
# instantiate the trainer class and check for available devices
trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=balanced_dataset['train'],
    eval_dataset=balanced_dataset['test'],
    data_collator=data_collator # A function to batch together samples of data.
)
```

Prompt-based Testing Examples

Prompt-based evaluation was useful to identify model's reaction to tone, keywords, and structure.

Prompt	Prediction	Confidence
"Click here to claim your prize"	Phishing	98.6%
"Meeting at 3pm about budget"	Safe	98.3%
"Verify your account immediately"	Phishing	98.9%
"Your dog looks cute!"	Safe	98.6%
"Update your password now"	Phishing	99.1%

Tokenizer Settings and Preprocessing

Input truncated & padded to model max length

`title` used as text field

Tokenizer used fast Rust implementation (`use_fast=True`)

Dataset balanced manually before tokenization

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-cased", use_fast=True, low_cpu_mem_usage=False)
#tokenizer z BERTa , use fast - rust based , low_cpu_mem_usage - dont needed rn
```

```
def encode_examples(example):
    # Encode the text and return the encoding which includes 'input_ids'
    return tokenizer(example['title'], truncation=True, padding='max_length')


balanced_dataset = balanced_dataset.map(encode_examples, batched=True)
```

Number libraries and number of trainable parameters

pandas

numpy

torch

transformers

datasets

tqdm

accelerate

mlflow

nlp

```
# number of trainable parameters
print(model.num_parameters(only_trainable=True)/1e6)
```

```
65.783042
```

from transformers :
- AutoTokenizer,
- pipeline,
- Trainer,
- TrainingArguments,
- DataCollatorWithPadding,
- AutoModelForSequenceClassification