In [ ]:
```
!pip install transformers datasets accelerate evaluate -q
```

```
[?25l   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m0.0/84.1 kB[0m [31m?[0m eta [36m-:--:--[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m84.1/84.1 kB[0m [31m3.7 MB/s[0m eta [36m0:00:00[0m
[?25h
```

In [ ]:
```
from datasets import load_dataset
from transformers import DistilBertTokenizerFast,
DistilBertForSequenceClassification
from transformers import TrainingArguments, Trainer
import evaluate
import numpy as np
```

In [ ]:
```
dataset = load_dataset("sms_spam")
dataset
```

```
/usr/local/lib/python3.12/dist-
packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional
to access public models or datasets.
  warnings.warn(
```

```
README.md: 0.00B [00:00, ?B/s]
```

```
plain_text/train-00000-of-00001.parquet:   0%|          | 0.00/359k
[00:00<?, ?B/s]
```

```
Generating train split:   0%|          | 0/5574 [00:00<?, ?
examples/s]
```

```
DatasetDict({
    train: Dataset({
        features: ['sms', 'label'],
        num_rows: 5574
    })
})
```

In [ ]:
```
dataset = dataset["train"].train_test_split(test_size=0.2, seed=42)

train_dataset = dataset["train"]
test_dataset = dataset["test"]
```

In [ ]:
```
tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-
uncased")
model = DistilBertForSequenceClassification.from_pretrained("distilbert-
base-uncased")
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

```
Some weights of DistilBertForSequenceClassification were not
initialized from the model checkpoint at distilbert-base-uncased
and are newly initialized: ['classifier.bias',
'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be
able to use it for predictions and inference.
```

In [ ]:
```
def tokenize(batch):
    return tokenizer(batch["sms"], padding="max_length", truncation=True,
max_length=128)

train_dataset = train_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.map(tokenize, batched=True)

train_dataset = train_dataset.rename_column("label", "labels")
test_dataset = test_dataset.rename_column("label", "labels")

train_dataset.set_format(type="torch", columns=
["input_ids","attention_mask","labels"])
test_dataset.set_format(type="torch", columns=
["input_ids","attention_mask","labels"])
```

```
Map:   0%|          | 0/4459 [00:00<?, ? examples/s]

Map:   0%|          | 0/1115 [00:00<?, ? examples/s]
```

In [ ]:
```
accuracy = evaluate.load("accuracy")
f1 = evaluate.load("f1")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)

    return {
        "accuracy": accuracy.compute(predictions=preds, references=labels),
        "f1": f1.compute(predictions=preds, references=labels)
    }
```

```
Downloading builder script: 0.00B [00:00, ?B/s]


Downloading builder script: 0.00B [00:00, ?B/s]
```

In [ ]:
```
training_args = TrainingArguments(
    output_dir="./distilbert-spam-detector",
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    logging_steps=50,
    report_to="none",
    num_train_epochs=2,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    load_best_model_at_end=True
)
```

In [ ]:
```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics
)
```

In [ ]:
```
trainer.train()
```

[558/558 02:01, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.042500	0.039573	{'accuracy': 0.9901345291479821}	{'f1': 0.9627118644067797}
2	0.005000	0.055815	{'accuracy': 0.9901345291479821}	{'f1': 0.9619377162629758}

```
TrainOutput(global_step=558, training_loss=0.045546295742193856,
metrics={'train_runtime': 122.4142, 'train_samples_per_second':
72.851, 'train_steps_per_second': 4.558, 'total_flos':
295336065303552.0, 'train_loss': 0.045546295742193856, 'epoch': 2.0})
```

In [20]:
```
text = "WINNER!! As a valued network customer you have been selected to
received 900$ prize reward! To claim..."

inputs = tokenizer(text, return_tensors="pt").to(model.device)

output = model(**inputs)

prediction = output.logits.argmax().item()

print("Spam" if prediction == 1 else "Not Spam")
```

```
Spam
```

In [ ]:
```
text = "I'm gonna be home soon and i don't want to talk about this stuff
anymore tonight"

inputs = tokenizer(text, return_tensors="pt").to(model.device)

output = model(**inputs)

prediction = output.logits.argmax().item()

print("Spam" if prediction == 1 else "Not Spam")
```

```
Not Spam
```