```
[ ] import nltk
    import pandas as pd
    nltk.download('popular', quiet=True)
    import demoji
    from wordcloud import WordCloud
    from nltk import word_tokenize
    from nltk.corpus import stopwords
    from nltk.util import bigrams
    from nltk import FreqDist
    import spacy
    import string
    nltk.download("wordnet", quiet=True)
    nltk.download("stopwords", quiet=True)
    from nltk.stem.wordnet import WordNetLemmatizer
    from nltk import pos_tag
    from nltk.corpus import wordnet
    from collections import defaultdict
    import pandas as pd
    nlp = spacy.load("en_core_web_sm")
```

1. Perform necessary data preprocessing, e.g. removing punctuation and stop words, stemming, lemmatizing. You may use the outputs from previous weekly assignments. (10 points)

```
[ ] import os
    import glob
    def collect_data():
        text_file_pattern = "*.txt"  # You can adjust the pattern to match your file extensions
        text_files = glob.glob(os.path.join("../nhs/content", text_file_pattern))
        data = {}
        for file_path in text_files:
            with open(file_path, 'r', encoding='utf-8') as file:
                file_name = os.path.basename(file_path)
                file_content = file.read()
                data[file_name] = file_content
        return data
```

```
[ ] corpus = collect_data()
    text = ""
    for data in corpus:
        text += " " + data
```

```
[ ] def remove_punctuation(text):
        # Create a translation table to remove punctuation
        translator = str.maketrans('', '', string.punctuation)
        # Use translate method to remove punctuation
        cleaned_text = text.translate(translator)
        return cleaned_text

    def remove_stop_words(text):
        nltk_stopwords = stopwords.words('english')
        spacy_stopwords = nlp.Defaults.stop_words

        stop_words = (*nltk_stopwords, *spacy_stopwords, "NHStxt")

        tokens = word_tokenize(text)
        tokens = [token for token in tokens if token not in stop_words]
        return " ".join(tokens)

    def apply_lemmitization(text):
        tag_map = defaultdict(lambda : wordnet.NOUN)
        tag_map['V'] = wordnet.VERB
        tag_map['A'] = wordnet.ADJ
        tag_map['R'] = wordnet.ADJ

        lemmitizer = WordNetLemmatizer()
        lemmitized_result = ""
        tokens = word_tokenize(text)
        for token, tag in pos_tag(tokens):
            lemma = lemmitizer.lemmatize(token, tag_map[tag[0]])
            lemmitized_result = lemmitized_result + " " + lemma
        return lemmitized_result

    def remove_emoji_and_smart_quotes(text):
        # replacing emojis with description
        text = demoji.replace_with_desc(text)
        #Removing smart quotes
        return text.replace("""", "\"").replace("""","\"")
```

```
[ ] def data_preprocessing(text):
        text = remove_emoji_and_smart_quotes(text)
        text = remove_punctuation(text)
        text = remove_stop_words(text)
        text = apply_lemmitization(text)
        return text

    def apply_data_preprocessing_to_corpus(corpus):
        new_corpus = {}
        for idx, key in enumerate(corpus.keys()):
            new_corpus[key] = data_preprocessing(corpus[key])
            print(f"idx: {idx}")
        return new_corpus
```

```
[ ] processed_text = data_preprocessing(text)
    with open('week11_1.txt', 'w') as file:
        file.write(f'{processed_text}')
```

```
[ ] data = []
    for filename in corpus.keys():
        data.append({
            "label": filename.removesuffix("  NHS.txt"),
            "data": data_preprocessing(corpus[filename])
        })
    df = pd.DataFrame(data)
```

```
[ ] data = []
    import textwrap
    indexes = {k: v for v, k in enumerate(sorted(corpus.keys()))}
    for filename in corpus.keys():
        text = data_preprocessing(corpus[filename])
        lines = textwrap.wrap(text, 100, break_long_words=False)
        for line in lines:
            data.append({
                "label": filename.removesuffix("  NHS.txt"),
                "data": line,
                "idx": indexes[filename]
            })
    df = pd.DataFrame(data)
```

```
[ ] from datasets import Dataset
    dataset = Dataset.from_pandas(df)
```

```
[ ] dataset.push_to_hub("shireesh-uop/nhs_classification")
```

```
[4] !pip -q install datasets
    !pip -q install transformers[torch]
    !pip -q install accelerate -U
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 521.2/521.2 kB 7.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 9.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 10.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 265.7/265.7 kB 9.0 MB/s eta 0:00:00
```

2. For the binary classification problem you came up previously, build your own model by combining BERT with a classifier. (30 points)

```
[91] from datasets import load_dataset
     dataset = load_dataset("shireesh-uop/nhs_classification")
```

```
[92] df=dataset["train"].to_pandas()
     df
```

```
[93] indexes = {}
     for i in df.label.unique():
       indexes[i] = df[df.label == i][:1]["idx"].item()
```

```
[94] from transformers import AutoTokenizer
     tokenizer = AutoTokenizer.from_pretrained('bert-base-cased')
```

```
[95] tokenizer("Attention is all you need")
```

```
     {'input_ids': [101, 1335, 5208, 2116, 1110, 1155, 1128, 1444, 102], 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```
[96] def tokenize_data(example):
         return tokenizer(example['data'], padding='max_length', truncation=True,
                          return_tensors="pt")

     def transform_labels(label):
         label = label['label']
         num = label["idx"]
         return {'labels': num}


     dataset = dataset.map(tokenize_data, batched=True)

     remove_columns = ['label']
     dataset = dataset.remove_columns(remove_columns)
```

```
[97] from transformers import TrainingArguments
     import accelerate
     # Batch size per GPU for training
     per_device_train_batch_size = 20

     # Batch size per GPU for evaluation
     per_device_eval_batch_size = 4

     training_args = TrainingArguments("test_trainer",
                                       num_train_epochs=30,
                                       hub_strategy="checkpoint",
                                       save_steps=2000,
                                       per_device_train_batch_size=per_device_train_batch_size)
     device_map = {"": 0}
```

```
[98] from transformers import AutoModelForSequenceClassification
     model = AutoModelForSequenceClassification.from_pretrained("shireesh-uop/nhs_classification", num_labels=978)
```

```
[99] from transformers import AutoModelForSequenceClassification, AutoConfig
     config = AutoConfig.from_pretrained("bert-base-cased", num_labels=978, device_map=device_map)
     model_untrained = AutoModelForSequenceClassification.from_config(config)
```

```
[100] dataset = dataset["train"]
```

```
[101] start = int(len(dataset)*.9)
      end = int(len(dataset) *0.1)
      dataset = dataset.rename_column('data', 'sentence')
      dataset = dataset.rename_column('idx', 'labels')
```

```
[102] train_dataset = dataset.shuffle(seed=10).select(range(start))
      eval_dataset = dataset.shuffle(seed=10).select(range(start, end+start))
```

```
[103] from transformers import Trainer
      import torch
      import numpy as np
      from datasets import load_metric

      metric = load_metric("accuracy")

      def compute_metrics(eval_pred):
          logits, labels = eval_pred
          predictions = np.argmax(logits, axis=-1)
          return metric.compute(predictions=predictions, references=labels)

      # move model to tpu
      trainer = Trainer(
          model=model, args=training_args,
          train_dataset=train_dataset, eval_dataset=eval_dataset,
          compute_metrics=compute_metrics
      )
```

```
[104] train_dataset
```

```
Dataset({
    features: ['sentence', 'labels', 'input_ids', 'token_type_ids', 'attention_mask'],
    num_rows: 24411
})
```

```
[105] for i in train_dataset:
          print(i)
          break
```

```
{'sentence': 'spot close Almost spot form round slightly oval blister 1 spot appear flatterThe blister pink shiny', 'labels': 764, 'input_ids': [101, 3205, 1601, 8774, 3205, 1532, 1668,
```

3. Train your own model by fine-tuning BERT. And save your model and use it to classify sentences (50 points)

```
[106] trainer.train()
```

[1360/36630 12:35 < 5:26:49, 1.80 it/s, Epoch 1.11/30]

Step	Training Loss
500	4.484700
1000	3.934500

[6050/36630 56:19 < 4:44:48, 1.79 it/s, Epoch 4.95/30]

Step	Training Loss
500	4.484700
1000	3.934500
1500	3.323300
2000	2.863400
2500	2.622800
3000	2.166100
3500	2.069300
4000	1.822800
4500	1.627200
5000	1.479000
5500	1.198300
6000	1.195600

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-106-3435b262f1ae> in <cell line: 1>()
----> 1 trainer.train()

                                 5 frames

/usr/local/lib/python3.10/dist-packages/torch/autograd/__init__.py in backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables, inputs)
    249     # some Python versions print out the first line of a multi-line function
    250     # calls in the traceback and some print out the last line
--> 251     Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
    252         tensors,
    253         grad_tensors_,

KeyboardInterrupt:
```

EXPLAIN ERROR

```
[108]
      trainer.save_model("./model")
      model.push_to_hub("shireesh-uop/nhs_classification")
```

model.safetensors: 100% 436M/436M [00:15<00:00, 26.5MB/s]

```
CommitInfo(commit_url='https://huggingface.co/shireesh-uop/nhs_classification/commit/1a8c0930c04222525402ff3fd6232a25efb60e0c', commit_message='Upload BertForSequenceClassification',
commit_description='', oid='1a8c0930c04222525402ff3fd6232a25efb60e0c', pr_url=None, pr_revision=None, pr_num=None)
```

```
[83] model = model.from_pretrained("./model")
```

```
[107] """
      before training
      {'eval_loss': 6.946279525756836,
       'eval_accuracy': 0.0003687315634218289,
       'eval_runtime': 28.4205,
       'eval_samples_per_second': 95.424,
       'eval_steps_per_second': 11.928}
       """

      trainer.evaluate()
```

[6050/36630 56:19 < 4:44:48, 1.79 it/s, Epoch 4.95/30]

Step	Training Loss	Validation Loss	Accuracy
500	4.484700		
1000	3.934500		

1500	3.323300		
2000	2.863400		
2500	2.622800		
3000	2.166100		
3500	2.069300		
4000	1.822800		
4500	1.627200		
5000	1.479000		
5500	1.198300		
6000	1.195600		
6049	1.195600	2.526456	0.533923

```
{'eval_loss': 2.526455879211426, 'eval_accuracy': 0.5339233038348082}
```

4. Summarize what you have learned and discovered from Task 1-3. (10 points)

- I see that thee model is converging towards higher accuracy but needs more training time.
- this would have been faster with SVM or ANN, but they will have a limit on how accurate they can be.

So, models need to be chosen based on the use case.