In [1]:
```
import nltk

nltk.download('popular', quiet=True)
import demoji
from nltk import word_tokenize
from nltk.corpus import stopwords
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)

In [2]:
```
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
```

In [3]:
```
corpus = collect_data()
text = ""
for data in corpus:
    text += " " + data
```

In [9]:
```
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("”","\"")
```

In [10]:
```
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
```

In [11]:
```
processed_text = data_preprocessing(text)
with open('week8_1.txt', 'w') as file:
    file.write(f'{processed_text}')
```

2. For the binary classification problem you came up last week, set up a MLP to solve it. (50 points)

In [12]:
```
df = pd.read_csv("../nhs/conditions_departments.csv", header=None)
df.columns = ["index", "condition", "department"]
department_dict = {
    row['condition']: row.drop('condition').to_dict()
    for index, row in df.iterrows()
}
```

In [13]:
```
"""
Get X as text data and Y as 0 if general medicine else 1
"""
X = []
y =[]
for file_name in list(corpus.keys()):
    idx = file_name.strip(" NHS.txt")
    text = data_preprocessing(corpus[file_name])
    X.append(text)
    y.append(0 if department_dict[idx]["department"] == "General Medicine" else 1)
```

3. Compute TF-IDF vectors on the text data. (10 points)

In [14]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
max_tokens = 1000

vectorizer = TfidfVectorizer(min_df = .1,
                             tokenizer = nltk.word_tokenize,
                             max_features = max_tokens)
X = vectorizer.fit_transform(X)
```

```
/Users/shireesh/opt/anaconda3/envs/COMP293/lib/python3.10/site-packages/sklearn/feature_extraction/text.py:525: UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

In [15]:
```
pd.DataFrame({"data":X,"general": y})
```

Out[15]:

	data	general
0	(0, 234)\t0.0323333686450092\n (0, 27)\t0.0...	0
1	(0, 153)\t0.0472045561666712\n (0, 306)\t0...	1
2	(0, 70)\t0.06541022666026458\n (0, 228)\t0....	0
3	(0, 90)\t0.1164002171842305\n (0, 341)\t0.0...	1
4	(0, 189)\t0.03823823168179656\n (0, 243)\t0...	1
...	...	...
973	(0, 38)\t0.04855655296617292\n (0, 143)\t0....	0
974	(0, 296)\t0.1094379591417735\n (0, 161)\t0....	0
975	(0, 170)\t0.046948426395212565\n (0, 311)\t...	0
976	(0, 38)\t0.10690698901167914\n (0, 143)\t0....	0
977	(0, 161)\t0.029068557808877748\n (0, 20)\t0...	0

978 rows × 2 columns

In [16]:
```
from sklearn.model_selection import train_test_split

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [23]:
```
X_train.shape, X_test.shape
```

Out[23]:
```
((782, 376), (196, 376))
```

In [89]:
```
from torch import optim
import torch
import torch.nn as nn
import torch.nn.functional as F
import pytorch_lightning as pl

class BinaryClassifier(pl.LightningModule):
    def __init__(self, in_channels):
        super(BinaryClassifier, self).__init__()
```

```
        self.model = nn.Sequential(
            nn.Linear(in_channels, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        # training_step defines the train loop.
        # it is independent of forward
        x, y = batch
        # print(x.shape)
        y_hat = self.model(x)
        y = y.unsqueeze(1)
        loss = F.binary_cross_entropy(y_hat, y)
        # Logging to TensorBoard (if installed) by default
        self.log("train_loss", loss)
        if batch_idx == 0:
            print(f"loss {batch_idx}: {loss}")
        return loss

    def configure_optimizers(self):
        optimizer = optim.Adam(self.parameters(), lr=2e-4)
        return optimizer
```

In [90]:
```
from torch.utils.data import DataLoader
from torch.utils.data import TensorDataset, random_split

dataset = TensorDataset(torch.tensor(X.toarray(), dtype=torch.float32),
                        torch.tensor(y, dtype=torch.float32))
# Define the size of your train and test data
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size

# Split your dataset
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

In [91]:
```
def calculate_accuracy(model, data_loader):
    model.eval()  # Set the model to evaluation mode
    correct = 0
    total = 0

    with torch.no_grad():  # Inference without gradient calculation
        for inputs, labels in data_loader:
            outputs = model(inputs)

            # Apply threshold to get binary predictions
            predicted = outputs.round()  # Using 0.5 as the threshold

            total += labels.size(0)
            correct += (predicted == labels.unsqueeze(1)).sum().item()

    accuracy = 100 * correct / total
    return accuracy
```

In [92]:
```
model = BinaryClassifier(X_train.shape[1])
```

In [93]:
```
early_stopping = pl.callbacks.EarlyStopping(monitor='train_loss', patience=5, min_delta=1e-6)
trainer = pl.Trainer(max_epochs=30, callbacks=[early_stopping])
trainer.fit(model=model, train_dataloaders=train_loader)
```

```
GPU available: False, used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs

  | Name  | Type       | Params
-------------------------------------
0 | model | Sequential | 24.2 K
-------------------------------------
24.2 K    Trainable params
0         Non-trainable params
24.2 K    Total params
0.097     Total estimated model params size (MB)
/Users/shireesh/opt/anaconda3/envs/COMP293/lib/python3.10/site-packages/pytorch_lightning/trainer/connectors/data_connector.py:441: The 'train_dataloader' does not have many workers which may be a bottleneck. Consider increasing the value of the `num_workers` argument` to `num_workers=11` in the `DataLoader` to improve performance.
/Users/shireesh/opt/anaconda3/envs/COMP293/lib/python3.10/site-packages/pytorch_lightning/loops/fit_loop.py:293: The number of training batches (13) is smaller than the logging interval Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you want to see logs for the training epoch.
```

```
Training: |          | 0/? [00:00<?, ?it/s]

loss 0: 0.7152660489082336
loss 0: 0.6991397142410278
loss 0: 0.6816490292549133
loss 0: 0.6543018221855164
loss 0: 0.6283830404281616
loss 0: 0.6144261956214905
loss 0: 0.5959171652793884
```

```
loss 0: 0.5644806027412415
loss 0: 0.5279189944267273
loss 0: 0.5515310764312744
loss 0: 0.4709937274456024
loss 0: 0.4650184214115143
loss 0: 0.46780768036842346
loss 0: 0.4079306721687317
loss 0: 0.4107171893119812
loss 0: 0.4138753414154053
loss 0: 0.3341768682003021
```

In [94]:
```
# Calculate accuracy on the test set
accuracy = calculate_accuracy(model, test_loader)
print(f'Accuracy of the model on the test set: {accuracy:.2f}%')
```

```
Accuracy of the model on the test set: 85.71%
```

3. Try to improve performance by modifying hyperparameters. (30 points)

In [114]:
```
from torch import optim
import torch
import torch.nn as nn
import torch.nn.functional as F
import pytorch_lightning as pl

class BinaryClassifier2(pl.LightningModule):
    def __init__(self, in_channels):
        super(BinaryClassifier2, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(in_channels, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 64),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        # training_step defines the train loop.
        # it is independent of forward
        x, y = batch
        # print(x.shape)
        y_hat = self.model(x)
        y = y.unsqueeze(1)
        loss = F.binary_cross_entropy(y_hat, y)
        # Logging to TensorBoard (if installed) by default
        self.log("train_loss", loss)
        if batch_idx == 0:
            print(f"loss {batch_idx}: {loss}")
        return loss

    def configure_optimizers(self):
        optimizer = optim.Adam(self.parameters(), lr=2e-4)
        return optimizer
```

In [121]:
```
model = BinaryClassifier2(X_train.shape[1])
trainer = pl.Trainer(max_epochs=50)
trainer.fit(model=model, train_dataloaders=train_loader)
```

```
GPU available: False, used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs

  | Name  | Type       | Params
-------------------------------------
0 | model | Sequential | 340 K
-------------------------------------
340 K     Trainable params
0         Non-trainable params
340 K     Total params
1.363     Total estimated model params size (MB)
/Users/shireesh/opt/anaconda3/envs/COMP293/lib/python3.10/site-packages/pytorch_lightning/trainer/connectors/data_connector.py:441: The 'train_dataloader' does not have many workers which may be a bottleneck. Consider increasing the value of the `num_workers` argument` to `num_workers=11` in the `DataLoader` to improve performance.
/Users/shireesh/opt/anaconda3/envs/COMP293/lib/python3.10/site-packages/pytorch_lightning/loops/fit_loop.py:293: The number of training batches (13) is smaller than the logging interval Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you want to see logs for the training epoch.
```

```
Training: |          | 0/? [00:00<?, ?it/s]
```

```
loss 0: 0.6831235289573669
loss 0: 0.6509532928466797
loss 0: 0.6108098030090332
loss 0: 0.5325679779052734
loss 0: 0.39180970191955566
```

```
loss 0: 0.45824435353279114
loss 0: 0.30083316564559937
loss 0: 0.3549799919128418
loss 0: 0.29201117157936096
loss 0: 0.2440807819366455
loss 0: 0.2251686453819275
loss 0: 0.302427738904953
loss 0: 0.17919957637786865
loss 0: 0.24869504570960999
loss 0: 0.4423391819000244
loss 0: 0.2803780734539032
loss 0: 0.15079404413700104
loss 0: 0.21919460594654083
loss 0: 0.27004358172416687
loss 0: 0.25015029311180115
loss 0: 0.2397792935371399
loss 0: 0.21643076837062836
loss 0: 0.22078466415405273
loss 0: 0.2155144214630127
loss 0: 0.22596460580825806
loss 0: 0.20870733261108398
loss 0: 0.17167425155639648
loss 0: 0.15788763761520386
loss 0: 0.14203757047653198
loss 0: 0.18673855066299438
loss 0: 0.14141425490379333
loss 0: 0.12673766911029816
loss 0: 0.22290022671222687
loss 0: 0.14004197716712952
loss 0: 0.22457602620124817
loss 0: 0.09881763160228729
loss 0: 0.11778198927640915
loss 0: 0.06151909381151199
loss 0: 0.0712595134973526
loss 0: 0.07483220100402832
loss 0: 0.05176405608654022
loss 0: 0.050762757658958435
loss 0: 0.032045240223407 75
loss 0: 0.05772785842418671
loss 0: 0.02596135064959526
loss 0: 0.027589453384280205
loss 0: 0.008000734262168407
loss 0: 0.03569462522864342
loss 0: 0.014005517587065697
loss 0: 0.013406251557171345
```

```
`Trainer.fit` stopped: `max_epochs=50` reached.
```

In [123]:
```
accuracy = calculate_accuracy(model, test_loader)
print(f'Accuracy of the model on the test set: {accuracy:.2f}%')
```

```
Accuracy of the model on the test set: 90.31%
```

4. Summarize what you have learned and discovered from Task 1-3 as well as the tasks you completed last week.

The results from MLP, SVM looks promising compared to SVM and MultinomialNB model

method	accuracy
mlp	90.31%
svm	90.3%
MultinomialNB	88%

I would rather choose MLP over SVM cause it can generalise even more data points.