```
import pandas as pd
import nltk
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
```

```
import os
```

```
os.listdir('./drive/MyDrive/datasets/')
```

```
['test.csv',
 'clean_train_nlp.csv',
 '.ipynb_checkpoints',
 'train.csv',
 'clean_train_3.csv',
 'lemmatized_data.csv',
 'mpr_test.csv',
 'mpr_train.csv',
 'NLP MPR',
 'lemmatized_data.gsheet',
 'clean_train_3.gsheet',
 'preprocessed_train.csv']
```

```
nltk.download('maxent_treebank_pos_tagger')
nltk.download('punkt')
nltk.download('words')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package maxent_treebank_pos_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_treebank_pos_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
df = pd.read_csv('./drive/MyDrive/datasets/lemmatized_data.csv')
```

```
df.head(5)
```

| | Unnamed: 0.1 | Unnamed: 0 | id | movie_name | synopsis | genre | filtered_synopsis | le |
|---|---|---|---|---|---|---|---|---|
| | | | | | a young scriptwriter | | young scriptwriter | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 44978 | Super Me | starts bringing valuable ... | fantasy | starts bringing valuable ob... | y br |
| **1** | 1 | 1 | 50185 | Entity Project | a director and her friends renting a haunted h... | horror | director friends renting haunted house capture... | ha |
| | | | | Behavioral | this is an | | | |

```
df = df[:5000]
```

# Tokenizing

```
df['tokenized'] = df['lemmatized_synopsis'].apply(nltk.word_tokenize)
```

```
<ipython-input-33-b3364e91ab33>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
  df['tokenized'] = df['lemmatized_synopsis'].apply(nltk.word_tokenize)
```

```
df['tokenized'][:5]
```

```
0    [young, scriptwriter, start, bringing, valuabl...
1    [director, friend, renting, haunted, house, ca...
2    [educational, video, family, family, therapist...
3    [scientist, working, austrian, alp, discover, ...
4    [buy, day, four, men, widely, apart, life, nig...
Name: tokenized, dtype: object
```

```
df['pos'] = df['tokenized'].apply(nltk.pos_tag)
```

```
<ipython-input-35-a67f6250f7fb>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
  df['pos'] = df['tokenized'].apply(nltk.pos_tag)
```

```
print(df['pos'][:5])
```

```
0    [(young, JJ), (scriptwriter, JJR), (start, NN)...
1    [(director, NN), (friend, NN), (renting, VBG),...
2    [(educational, JJ), (video, NN), (family, NN),...
3    [(scientist, NN), (working, VBG), (austrian, J...
4    [(buy, VB), (day, NN), (four, CD), (men, NNS),...
Name: pos, dtype: object
```

```
def wordnet_analysis(tokens):
    synsets = [wordnet.synsets(token) for token in tokens]
    return synsets
```

```
df['synsets'] = df['tokenized'].apply(wordnet_analysis)
```

```
<ipython-input-38-9d62062fc29d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
  df['synsets'] = df['tokenized'].apply(wordnet_analysis)
```

```
df['synsets'][:5]
```

```
0    [[Synset('young.n.01'), Synset('young.n.02'), ...
1    [[Synset('director.n.01'), Synset('director.n....
2    [[Synset('educational.a.01'), Synset('educatio...
3    [[Synset('scientist.n.01')], [Synset('working....
4    [[Synset('bargain.n.02'), Synset('buy.v.01'), ...
Name: synsets, dtype: object
```

```
!pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.34.1-py3-none-any.whl (7.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 43.8 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.0/302.0 kB 28.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
Collecting tokenizers<0.15,>=0.14 (from transformers)
  Downloading tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x8
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 90.5 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x8
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 71.6 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-
 equirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python
```

```
     Collecting  huggingface-hub<1.0,>=0.16.4  (from  transformers)
       Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 295.0/295.0 kB 26.6 MB/s eta 0:00:00
     Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.
     Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pack
     Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dis
     Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dis
     Installing collected packages: safetensors, huggingface-hub, tokenizers, transform
     Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0 tokenizers-0.14.1
```

```
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from transformers import BertTokenizer, BertModel
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification, AdamW
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import LabelEncoder

train_data = df


label_encoder = LabelEncoder()

train_data['genre'] = train_data['genre'].apply(lambda genres: ', '.join(genres))

y_train_encoded = label_encoder.fit_transform(train_data['genre'])

model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
embedding_model = BertModel.from_pretrained(model_name)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
embedding_model.to(device)

max_length = 20
concatenated_text = train_data['synopsis']
encoded_inputs = tokenizer(list(concatenated_text), padding='max_length', truncation=Tr

train_dataset = TensorDataset(torch.tensor(encoded_inputs['input_ids']), torch.tensor(en
train_loader = DataLoader(train_dataset, batch_size=12, shuffle=True)

class CustomClassifier(nn.Module):
    def___init__(self, embedding_model, num_classes):
        super(CustomClassifier, self)._init__()
        self.embedding_model = embedding_model
        self.fc = nn.Linear(embedding_model.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        embeddings = self.embedding_model(input_ids, attention_mask=attention_mask).last
        logits = self.fc(embeddings)
        return logits
```

```
num_classes = len(label_encoder.classes_)
model = CustomClassifier(embedding_model, num_classes)
model.to(device)

optimizer = optim.AdamW(model.parameters(), lr=1e-5)
loss_fn = nn.CrossEntropyLoss()

model.train()
for epoch in range(4):
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch + 1}/5", leave=False)
    total_correct = 0
    total_samples = 0

    for batch in progress_bar:
        optimizer.zero_grad()
        input_ids, attention_mask, labels = [item.to(device) for item in batch]
        logits = model(input_ids, attention_mask)
        loss = loss_fn(logits, labels)
        loss.backward()
        optimizer.step()

        #  accuracy
        _, predicted = torch.max(logits, 1)
        total_correct += (predicted == labels).sum().item()
        total_samples += labels.size(0)
        accuracy = total_correct / total_samples

        progress_bar.set_postfix({"loss": loss.item(), "accuracy": accuracy})

    # Accuracy * epoche
    print(f'Epoch {epoch + 1} - Accuracy: {accuracy:.4f}')



# Eval_model
model.eval()
total_correct = 0
total_samples = 0
with torch.no_grad():
    progress_bar = tqdm(train_loader, desc="Evaluating", leave=False)
    for batch in progress_bar:
        input_ids, attention_mask, labels = [item.to(device) for item in batch]
        logits = model(input_ids, attention_mask)
        _, predicted = torch.max(logits, 1)
        total_correct += (predicted == labels).sum().item()
        total_samples += labels.size(0)
        progress_bar.set_postfix({"accuracy": total_correct / total_samples})

accuracy = total_correct / total_samples
print(f'Final Accuracy: {accuracy:.4f}')

     Epoch 1 - Accuracy: 0.2342
     Epoch 2 - Accuracy: 0.3982
```

```
Epoch 3 - Accuracy: 0.5048
Epoch 4 - Accuracy: 0.6072
```

Final

```
import os
```

```
os.listdir()
```

```
['test.csv',
 'clean_train_nlp.csv',
 '.ipynb_checkpoints',
 'train.csv',
 'clean_train_3.csv',
 'lemmatized_data.csv',
 'mpr_test.csv',
 'mpr_train.csv',
 'NLP MPR',
 'lemmatized_data.gsheet',
 'clean_train_3.gsheet',
 'preprocessed_train.csv',
 'exp_nine.pth',
 'exp_nine.pkl',
 'exp_nine(1).pkl']
```

```
torch.save(model.state_dict(), 'exp_nine(2).pkl')
```

```
class CustomClassifier(nn.Module):
    def___init__(self, embedding_model, num_classes):
        super(CustomClassifier, self)._init__()
        self.embedding_model = embedding_model
        self.fc = nn.Linear(embedding_model.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        embeddings = self.embedding_model(input_ids, attention_mask=attention_mask).last
        logits = self.fc(embeddings)
        return logits
```

```
num_classes = len(label_encoder.classes_)
model = CustomClassifier(embedding_model, num_classes)
```

```
os.listdir()
```

```
['test.csv',
 'clean_train_nlp.csv',
 '.ipynb_checkpoints',
 'train.csv',
 'clean_train_3.csv',
 'lemmatized_data.csv',
 'mpr_test.csv',
 'mpr_train.csv',
 'NLP MPR',
 'lemmatized_data.gsheet',
 'clean_train_3.gsheet',
```

```
 'preprocessed_train.csv',
 'exp_nine.pth',
 'exp_nine.pkl',
 'exp_nine(1).pkl',
 'exp_nine(2).pkl']
```

```
model.load_state_dict(torch.load('exp_nine(2).pkl'))
```

```
<All keys matched successfully>
```

```
model.eval()
```

```
CustomClassifier(
  (embedding_model): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (fc): Linear(in_features=768, out_features=10, bias=True)
)
```

```
def preprocess_input(input_text, tokenizer, max_length, label_encoder, device):
    input_data = tokenizer(input_text, padding=True, truncation=True, max_length=max_len

    input_ids = torch.tensor(input_data['input_ids'], dtype=torch.long).unsqueeze(0).to
    attention_mask = torch.tensor(input_data['attention_mask'], dtype=torch.long) unsqu

    return input_ids, attention_mask

def predict_genre(input_text, model, tokenizer, max_length, label_encoder, device):
    input_ids, attention_mask = preprocess_input(input_text, tokenizer, max_length, labe

    with torch.no_grad():
        logits = model(input_ids, attention_mask)

    _, predicted = torch.max(logits, 1)

    return "".join(label_encoder.classes_[predicted.item()].split()).replace(",","")


input_text = df['synopsis'][0]
print(predict_genre(input_text, model, tokenizer, max_length, label_encoder, device))
```

```
 fantasy
```

```
"".join(df['genre'][0].split()).replace(",","")
```

```
 'fantasy'
```