Movie Genre Prediction

Bangguo Xu& Simei Yan& Liang Liu & Zitai Wu

1. Text preprocessing and word2vec word vector embedding

```
import pandas as pd
import string
import torchtext
from torchtext.data.utils import get_tokenizer

# 1. Read Data
file_path = '/content/drive/MyDrive/movie.csv'   # Replace with your file path
movie_data = pd.read_csv(file_path)

# 2. Remove punctuation and digits from comments
def remove_punctuation_and_digits(text):
    return text.translate(str.maketrans('', '', string.punctuation + string.digits))

movie_data['cleaned_comment'] = movie_data['comment'].apply(remove_punctuation_and_digits)

# 3. Tokenize using torchtext
tokenizer = get_tokenizer("basic_english")
movie_data['tokenized_comment'] = movie_data['cleaned_comment'].apply(tokenizer)
```

```
import gensim
model = gensim.models.Word2Vec(movie_data['tokenized_comment'], vector_size=300, window=5, min_count=10, workers
model.train(movie_data['tokenized_comment'], total_examples=model.corpus_count, epochs=model.epochs)
```

2.Get the feature vector of the comment

```python
import numpy as np

# Initialize feature vector list
feature_vectors = []

# Generate a feature vector for each comment
for tokens in movie_data['tokenized_comment']:
    vector_sum = np.zeros(model.vector_size)
    words_in_model = 0

    for token in tokens:
        if token in model.wv:
            vector_sum += model.wv[token]
            words_in_model += 1

    # Avoid division by zero
    if words_in_model > 0:
        average_vector = vector_sum / words_in_model
    else:
        average_vector = vector_sum

    feature_vectors.append(average_vector)

# Convert feature vector list to Numpy array for subsequent processing
feature_vectors = np.array(feature_vectors)

# Feature vector dimension should be consistent with Word2Vec model's vector_size
print("Dimension of feature vectors:", feature_vectors.shape)
```

```
Dimension of feature vectors: (54214, 300)
```

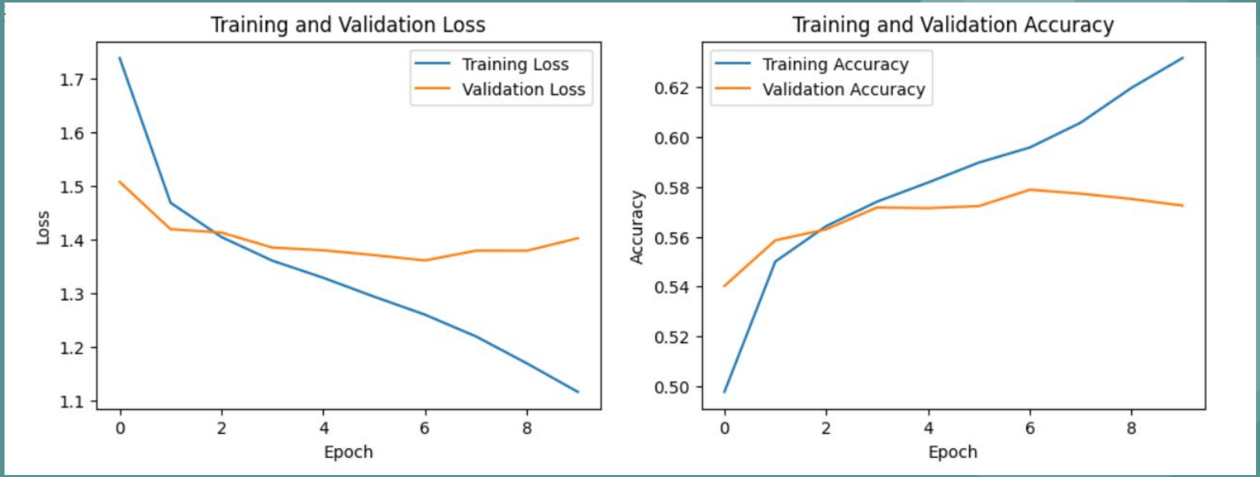
3. LSTM Model

```
TextLSTM(
  (lstm): LSTM(300, 500, num_layers=2, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (fc): Linear(in_features=1000, out_features=27, bias=True)
)
```

```
# Network parameters
input_size = 300 # Set according to Word2Vec word vector dimension
hidden_size = 500
output_size = len(set(encoded_labels)) # Number of categories
num_layers = 2
bidirectional=True
dropout_prob = 0.1

# Create model instance
model = TextLSTM(input_size, hidden_size, output_size, num_layers, bidirectional, dropout_prob)
```

4.LSTM Result



Possible reasons for low validation accuracy

1.Data imbalance: There are too many samples of some types, so the model may have a preference for these types.

2. Data cleaning and preprocessing may be insufficient, causing the model to learn noise.

3. Insufficient feature extraction:The current characterization may not be sufficient to capture the complex relationship between comment and film genres.Consider using more complex embeddings: such as pretrained BERT or GloVe embeddings.

5.SVM Result

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Create SVM classifier instance
svm_classifier = SVC(kernel='linear')

# Train model
svm_classifier.fit(X_train, y_train)

# Make predictions on the validation set
y_pred = svm_classifier.predict(X_valid)

# Evaluate the model
accuracy = accuracy_score(y_valid, y_pred)
report = classification_report(y_valid, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:")
print(report)
```

Accuracy: 0.5727197270128194

Classification Report:

	precision	recall	f1-score	support
0	0.41	0.25	0.32	263
1	0.63	0.23	0.34	112
2	0.30	0.09	0.13	139
3	0.62	0.08	0.14	104
4	0.00	0.00	0.00	61
5	0.48	0.55	0.51	1443
6	0.00	0.00	0.00	107
7	0.67	0.87	0.75	2659
8	0.53	0.77	0.63	2697
9	0.53	0.07	0.12	150
10	0.00	0.00	0.00	74
11	0.88	0.55	0.68	40
12	0.00	0.00	0.00	45
13	0.55	0.58	0.56	431
14	0.67	0.55	0.60	144
15	0.00	0.00	0.00	50
16	0.00	0.00	0.00	56
17	0.00	0.00	0.00	34
18	0.52	0.22	0.31	192
19	0.00	0.00	0.00	151
20	0.47	0.27	0.34	143
21	0.53	0.24	0.33	1045
22	0.61	0.25	0.35	93
23	0.59	0.16	0.25	81
24	0.49	0.06	0.11	309
25	0.00	0.00	0.00	20
26	0.81	0.76	0.79	200
accuracy			0.57	10843
macro avg	0.38	0.24	0.27	10843

6. Bert

Text Preprocessing:

Remove stopwords ,apply stemming using NLTK's PorterStemmer,apply lemmatization using NLTK's WordNetLemmatizer.Perform part-of-speech tagging and use it to improve lemmatization.

```
# Remove stop words
from nltk.corpus import stopwords
stopwords = stopwords.words('english')

title = [' '.join([w for w in t.split() if w not in stopwords]) for t in title]
comment = [' '.join([w for w in c.split() if w not in stopwords]) for c in comment]
```

```
# Stem extraction
from nltk.stem import PorterStemmer
ps = PorterStemmer()
title = [' '.join([ps.stem(w) for w in t.split()]) for t in title]
comment = [' '.join([ps.stem(w) for w in c.split()]) for c in comment]
# Lemmatization
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer()
title = [' '.join([wnl.lemmatize(w) for w in t.split()]) for t in title]
comment = [' '.join([wnl.lemmatize(w) for w in c.split()]) for c in comment]
```

6. Bert

Encoding:

```
class TextDataset(Dataset):
    def __init__(self, title, comment, genre, tokenizer, max_length):
        self.title = title
        self.comment = comment
        self.genre = genre
        self.tokenizer = tokenizer
        self.max_length = max_length

        # Use LabelEncoder to encode genre
        self.label_encoder = LabelEncoder()
        self.genre_encoded = self.label_encoder.fit_transform(self.genre)

    def __len__(self):
        return len(self.title)

    def __getitem__(self, idx):
        text = f"{self.title[idx]} {self.comment[idx]}"
        encoding = self.tokenizer(text, truncation=True, padding='max_length', max_length=self.max_length, return_tensors='pt')

        input_ids = encoding['input_ids'].squeeze()
        attention_mask = encoding['attention_mask'].squeeze()

        label = self.genre_encoded[idx]

        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'label': label
        }
```

6. Bert

Model:

```python
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
from tqdm import tqdm

# define the model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=len(set(train_dataset.genre)))
print(model)
```

The model is pre-trained on a large corpus of unlabelled text including the entire Wikipedia (that's 2,500 million words long) and Book Corpus.
The 'bert-base-uncased' version indicates that the model is a smaller version of BERT with 12 layers (transformer blocks), 768 hidden units (hi parameters. 'Uncased' means that the text has b hence, the model does not differentiate between

Result:

```
Epoch 1/5 - Training: 100%|██████████| 1186/1186 [1:59:16<00:00,  6.03s/it]
Epoch 1/5 - Testing: 100%|██████████| 509/509 [14:50<00:00,  1.75s/it]

Epoch 1/5 - Train Loss: 0.0522, Train Accuracy: 0.5231, Test Loss: 0.0421, Test Accuracy: 0.6037

Epoch 2/5 - Training: 100%|██████████| 1186/1186 [1:46:22<00:00,  5.38s/it]
Epoch 2/5 - Testing: 100%|██████████| 509/509 [13:08<00:00,  1.55s/it]

Epoch 2/5 - Train Loss: 0.0376, Train Accuracy: 0.6438, Test Loss: 0.0397, Test Accuracy: 0.6192

Epoch 3/5 - Training: 100%|██████████| 1186/1186 [1:40:28<00:00,  5.08s/it]
Epoch 3/5 - Testing: 100%|██████████| 509/509 [12:44<00:00,  1.50s/it]

Epoch 3/5 - Train Loss: 0.0295, Train Accuracy: 0.7175, Test Loss: 0.0402, Test Accuracy: 0.6165

Epoch 4/5 - Training: 100%|██████████| 1186/1186 [1:38:53<00:00,  5.00s/it]
Epoch 4/5 - Testing: 100%|██████████| 509/509 [12:44<00:00,  1.50s/it]

Epoch 4/5 - Train Loss: 0.0219, Train Accuracy: 0.7944, Test Loss: 0.0440, Test Accuracy: 0.6125

Epoch 5/5 - Training: 100%|██████████| 1186/1186 [1:38:33<00:00,  4.99s/it]
Epoch 5/5 - Testing: 100%|██████████| 509/509 [12:31<00:00,  1.48s/it]

Epoch 5/5 - Train Loss: 0.0152, Train Accuracy: 0.8607, Test Loss: 0.0485, Test Accuracy: 0.6017
Training finished.
```

Thanks !