7500NLP-1 (Data Preprocessing)

Code:

```
import pandas as pd
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import re
import nltk
nltk.download('stopwords')
nltk.download('wordnet')

# Load dataset
df = pd.read_csv('C:/Users/siyua/OneDrive/Desktop/MBTI 500.csv')
print(df.info)
df=df.dropna()
print(df.info)

# Create 4 dimension
df['I/E'] = df['type'].apply(lambda x: 1 if x[0] == 'I' else 0)
df['S/N'] = df['type'].apply(lambda x: 1 if x[1] == 'S' else 0)
df['F/T'] = df['type'].apply(lambda x: 1 if x[2] == 'F' else 0)
df['P/J'] = df['type'].apply(lambda x: 1 if x[3] == 'P' else 0)

print(df)

# Lemmatization
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Define All MBTI types
mbti_types = ['INTJ', 'INTP', 'ENTJ', 'ENTP', 'INFJ', 'INFP', 'ENFJ', 'ENFP',
              'ISTJ', 'ISFJ', 'ESTJ', 'ESFJ', 'ISTP', 'ISFP', 'ESTP', 'ESFP']

def clean_text(text):
    text = re.sub(r'http\S+', '', text)  # Remove URL
    text = re.sub(r'[^A-Za-z\s]', '', text)  # Remove non-alphabetic characters
    text = re.sub(r'@\w+', '', text) # Remove users ID
    for mbti in mbti_types:
        text = re.sub(mbti, '', text, flags=re.IGNORECASE) # Remove MBTI types (both upper- and lower-case)
    text = re.sub(r'[\(\){}<>:;]', '', text) # Remove emoticons (?)
    text = text.lower()  # Convert to lowercase
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return " ".join(tokens)
```

```
df['clean_posts'] = df['posts'].apply(clean_text)
print(df.head())


# Define the file path
file_path = r'C:\Users\siyua\OneDrive\Desktop\cleaned_data.csv'


# Export as csv file
df.to_csv(file_path, index=False, encoding='utf-8-sig')
```

Output:

```
                                               posts  type  I/E  S/N  F/T  P/J                                         clean_posts
0  know intj tool use interaction people excuse a...  INTJ    1    0    0    0  know tool use interaction people excuse antiso...
1  rap music ehh opp yeah know valid well know fa...  INTJ    1    0    0    0  rap music ehh opp yeah know valid well know fa...
2  preferably p hd low except wew lad video p min...  INTJ    1    0    0    0  preferably p hd low except wew lad video p min...
3  drink like wish could drink red wine give head...  INTJ    1    0    0    0  drink like wish could drink red wine give head...
4  space program ah bad deal meing freelance max ...  INTJ    1    0    0    0  space program ah bad deal meing freelance max ...
PS C:\Users\siyua>
```

7500NLP-2 (Bert for I/E Classification)

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight


import time
import torch
from transformers import DistilBertForSequenceClassification
from transformers import DistilBertTokenizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, matthews_corrcoef,
roc_curve, auc


df = pd.read_csv('C:/Users/siyua/OneDrive/Desktop/cleaned_data.csv', encoding='ISO-8859-1')
print(df.head())


# See the distribution of dataset
# Create figure and axis
fig, axes = plt.subplots(2, 2, figsize=(10, 8))


dimensions = {
    "I/E": ["I", "E"],
    "S/N": ["S", "N"],
    "F/T": ["F", "T"],
    "P/J": ["P", "J"],
}
```

```
# Draw the plot
for ax, (col, labels) in zip(axes.flatten(), dimensions.items()):
    counts = df[col].value_counts()
    ax.bar(labels, [counts.get(1, 0), counts.get(0, 0)], color='skyblue')
    ax.set_title(f"{col} Proportion")
    ax.set_ylabel("Count")
    ax.set_xticks([0, 1])
    ax.set_xticklabels(labels)

plt.tight_layout()
plt.show()

# Split dataset into training set, validation set and test set
X = df['clean_posts']
y = df['I/E']

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42,
stratify=y_temp)

# X_train: 60%
# X_val: 20%
# X_test: 20%

print("Training set:", X_train.shape, y_train.shape)
print("Validation set:", X_val.shape, y_val.shape)
print("Test set:", X_test.shape, y_test.shape)

# Bert Model
# 1. Bert Tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

def tokenize_data(texts, tokenizer, max_len=256):
    return tokenizer(texts.tolist(), return_tensors='pt', max_length=max_len, padding='max_length',
truncation=True)

train_encodings = tokenize_data(X_train, tokenizer)
val_encodings = tokenize_data(X_val, tokenizer)
test_encodings = tokenize_data(X_test, tokenizer)

# 2. Create data loader
class MBTIDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
```

```python
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

train_dataset = MBTIDataset(train_encodings, y_train.tolist())
val_dataset = MBTIDataset(val_encodings, y_val.tolist())
test_dataset = MBTIDataset(test_encodings, y_test.tolist())

# 3. WeightedRandomSampler
class_counts = y_train.value_counts().to_dict()
weights = [1.0 / class_counts[label] for label in y_train]
sampler = torch.utils.data.WeightedRandomSampler(weights, len(weights))

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=16, sampler=sampler)
valid_loader = torch.utils.data.DataLoader(val_dataset, batch_size=16, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=16, shuffle=True)

# 4. Load and fine-tune pre-trained Bert models
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
model.to(DEVICE)
model.train()

optim = torch.optim.Adam(model.parameters(), lr=5e-5)

def compute_accuracy(model, data_loader, device):
    with torch.no_grad():
        correct_pred, num_examples = 0, 0
        for batch_idx, batch in enumerate(data_loader):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs['logits']
            predicted_labels = torch.argmax(logits, 1)
            num_examples += labels.size(0)
```

```
            correct_pred += (predicted_labels == labels).sum()

    return correct_pred.float() / num_examples * 100

start_time = time.time()
NUM_EPOCHS = 3

for epoch in range(NUM_EPOCHS):

    model.train()

    for batch_idx, batch in enumerate(train_loader):

        ### Prepare data
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        ### Forward pass
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss, logits = outputs['loss'], outputs['logits']

        ### Backward pass
        optim.zero_grad()
        loss.backward()
        optim.step()

        ### Logging
        if not batch_idx % 250:
            print(f'Epoch: {epoch+1:04d}/{NUM_EPOCHS:04d}'
                  f' | Batch'
                  f'{batch_idx:04d}/'
                  f'{len(train_loader):04d} | '
                  f'Loss: {loss:.4f}')

    model.eval()

    with torch.set_grad_enabled(False):
        print(f'Training accuracy: '
              f'{compute_accuracy(model, train_loader, DEVICE):.2f}%'
              f'\nValid accuracy: '
              f'{compute_accuracy(model, valid_loader, DEVICE):.2f}%')

    print(f'Time elapsed: {(time.time() - start_time)/60:.2f} min')
```

```
print(f'Total Training Time: {(time.time() - start_time)/60:.2f} min')

print(f'Test accuracy: {compute_accuracy(model, test_loader, DEVICE):.2f}%')
```

Output:

```
Epoch: 0001/0003 | Batch0000/3978 | Loss: 0.7001
Epoch: 0001/0003 | Batch0250/3978 | Loss: 0.6037
Epoch: 0001/0003 | Batch0500/3978 | Loss: 0.6420
Epoch: 0001/0003 | Batch0750/3978 | Loss: 0.6818
Epoch: 0001/0003 | Batch1000/3978 | Loss: 0.5652
Epoch: 0001/0003 | Batch1250/3978 | Loss: 0.5421
Epoch: 0001/0003 | Batch1500/3978 | Loss: 0.3817
Epoch: 0001/0003 | Batch1750/3978 | Loss: 0.5397
Epoch: 0001/0003 | Batch1750/3978 | Loss: 0.5397
Epoch: 0001/0003 | Batch2000/3978 | Loss: 0.5458
Epoch: 0001/0003 | Batch2250/3978 | Loss: 0.3016
Epoch: 0001/0003 | Batch2500/3978 | Loss: 0.4490
Epoch: 0001/0003 | Batch2500/3978 | Loss: 0.4490
Epoch: 0001/0003 | Batch2750/3978 | Loss: 0.5433
Epoch: 0001/0003 | Batch3000/3978 | Loss: 0.5510
Epoch: 0001/0003 | Batch3250/3978 | Loss: 0.5138
Epoch: 0001/0003 | Batch3500/3978 | Loss: 0.5551
Epoch: 0001/0003 | Batch3750/3978 | Loss: 0.4386
Training accuracy: 79.28%
Valid accuracy: 77.87%
Time elapsed: 21.52 min
```

```
Epoch: 0002/0003 | Batch0000/3978 | Loss: 0.2899
Epoch: 0002/0003 | Batch0250/3978 | Loss: 0.4220
Epoch: 0002/0003 | Batch0500/3978 | Loss: 0.3639
Epoch: 0002/0003 | Batch0750/3978 | Loss: 0.1947
Epoch: 0002/0003 | Batch1000/3978 | Loss: 0.1688
Epoch: 0002/0003 | Batch1250/3978 | Loss: 0.4286
Epoch: 0002/0003 | Batch1500/3978 | Loss: 0.3680
Epoch: 0002/0003 | Batch1750/3978 | Loss: 0.7440
Epoch: 0002/0003 | Batch2000/3978 | Loss: 0.2694
Epoch: 0002/0003 | Batch2250/3978 | Loss: 0.1311
Epoch: 0002/0003 | Batch2500/3978 | Loss: 0.1736
Epoch: 0002/0003 | Batch2750/3978 | Loss: 0.3979
Epoch: 0002/0003 | Batch3000/3978 | Loss: 0.7480
Epoch: 0002/0003 | Batch3250/3978 | Loss: 0.3038
Epoch: 0002/0003 | Batch3500/3978 | Loss: 0.4466
Epoch: 0002/0003 | Batch3750/3978 | Loss: 0.5858
Training accuracy: 87.54%
Valid accuracy: 78.35%
Time elapsed: 43.02 min
```

```
Epoch: 0003/0003 | Batch0000/3978 | Loss: 0.4178
Epoch: 0003/0003 | Batch0250/3978 | Loss: 0.1010
Epoch: 0003/0003 | Batch0500/3978 | Loss: 0.2756
Epoch: 0003/0003 | Batch0750/3978 | Loss: 0.8839
Epoch: 0003/0003 | Batch1000/3978 | Loss: 0.1316
Epoch: 0003/0003 | Batch1250/3978 | Loss: 0.2152
Epoch: 0003/0003 | Batch1500/3978 | Loss: 0.2493
Epoch: 0003/0003 | Batch1750/3978 | Loss: 0.3030
Epoch: 0003/0003 | Batch2000/3978 | Loss: 0.3166
Epoch: 0003/0003 | Batch2250/3978 | Loss: 0.2860
Epoch: 0003/0003 | Batch2500/3978 | Loss: 0.1867
Epoch: 0003/0003 | Batch2750/3978 | Loss: 0.3026
Epoch: 0003/0003 | Batch3000/3978 | Loss: 0.3563
Epoch: 0003/0003 | Batch3250/3978 | Loss: 0.3839
Epoch: 0003/0003 | Batch3500/3978 | Loss: 0.3519
Epoch: 0003/0003 | Batch3750/3978 | Loss: 0.2669
Training accuracy: 93.70%
Valid accuracy: 77.03%
Time elapsed: 64.54 min
Total Training Time: 64.54 min
Test accuracy: 77.11%
```

7500NLP-3 (Bert for S/N Classification)

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight

import time
import torch
from transformers import DistilBertForSequenceClassification
from transformers import DistilBertTokenizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, matthews_corrcoef,
roc_curve, auc

df = pd.read_csv('C:/Users/siyua/OneDrive/Desktop/cleaned_data.csv', encoding='ISO-8859-1')
print(df.head())

# See the distribution of dataset
# Create figure and axis
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

dimensions = {
    "I/E": ["I", "E"],
    "S/N": ["S", "N"],
    "F/T": ["F", "T"],
    "P/J": ["P", "J"],
}

# Draw the plot
for ax, (col, labels) in zip(axes.flatten(), dimensions.items()):
    counts = df[col].value_counts()
    ax.bar(labels, [counts.get(1, 0), counts.get(0, 0)], color='skyblue')
    ax.set_title(f"{col} Proportion")
    ax.set_ylabel("Count")
    ax.set_xticks([0, 1])
    ax.set_xticklabels(labels)

plt.tight_layout()
plt.show()

# Split dataset into training set, validation set and test set
X = df['clean_posts']
```

```python
y = df['S/N']

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42,
stratify=y_temp)

# X_train: 60%
# X_val: 20%
# X_test: 20%

print("Training set:", X_train.shape, y_train.shape)
print("Validation set:", X_val.shape, y_val.shape)
print("Test set:", X_test.shape, y_test.shape)

# Bert Model
# 1. Bert Tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

def tokenize_data(texts, tokenizer, max_len=256):
    return tokenizer(texts.tolist(), return_tensors='pt', max_length=max_len, padding='max_length',
truncation=True)

train_encodings = tokenize_data(X_train, tokenizer)
val_encodings = tokenize_data(X_val, tokenizer)
test_encodings = tokenize_data(X_test, tokenizer)

# 2. Create data loader
class MBTIDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

train_dataset = MBTIDataset(train_encodings, y_train.tolist())
val_dataset = MBTIDataset(val_encodings, y_val.tolist())
test_dataset = MBTIDataset(test_encodings, y_test.tolist())
```

```python
# 3. WeightedRandomSampler
class_counts = y_train.value_counts().to_dict()
weights = [1.0 / class_counts[label] for label in y_train]
sampler = torch.utils.data.WeightedRandomSampler(weights, len(weights))

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=16, sampler=sampler)
valid_loader = torch.utils.data.DataLoader(val_dataset, batch_size=16, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=16, shuffle=True)

# 4. Load and fine-tune pre-trained Bert models
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
model.to(DEVICE)
model.train()

optim = torch.optim.Adam(model.parameters(), lr=5e-5)

def compute_accuracy(model, data_loader, device):
    with torch.no_grad():
        correct_pred, num_examples = 0, 0
        for batch_idx, batch in enumerate(data_loader):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs['logits']
            predicted_labels = torch.argmax(logits, 1)
            num_examples += labels.size(0)
            correct_pred += (predicted_labels == labels).sum()

    return correct_pred.float() / num_examples * 100

start_time = time.time()
NUM_EPOCHS = 3

for epoch in range(NUM_EPOCHS):

    model.train()

    for batch_idx, batch in enumerate(train_loader):

        ### Prepare data
```

```python
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        ### Forward pass
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss, logits = outputs['loss'], outputs['logits']

        ### Backward pass
        optim.zero_grad()
        loss.backward()
        optim.step()

        ### Logging
        if not batch_idx % 250:
            print(f'Epoch: {epoch+1:04d}/{NUM_EPOCHS:04d}'
                  f' | Batch'
                  f'{batch_idx:04d}/'
                  f'{len(train_loader):04d} | '
                  f'Loss: {loss:.4f}')

    model.eval()

    with torch.set_grad_enabled(False):
        print(f'Training accuracy: '
              f'{compute_accuracy(model, train_loader, DEVICE):.2f}%'
              f'\nValid accuracy: '
              f'{compute_accuracy(model, valid_loader, DEVICE):.2f}%')

    print(f'Time elapsed: {(time.time() - start_time)/60:.2f} min')

print(f'Total Training Time: {(time.time() - start_time)/60:.2f} min')
print(f'Test accuracy: {compute_accuracy(model, test_loader, DEVICE):.2f}%')
```

Output:

```
Epoch: 0001/0003 | Batch0000/3978 | Loss: 0.6825
Epoch: 0001/0003 | Batch0250/3978 | Loss: 0.5010
Epoch: 0001/0003 | Batch0500/3978 | Loss: 0.3982
Epoch: 0001/0003 | Batch0750/3978 | Loss: 0.5725
Epoch: 0001/0003 | Batch1000/3978 | Loss: 0.5212
Epoch: 0001/0003 | Batch1250/3978 | Loss: 0.2292
Epoch: 0001/0003 | Batch1500/3978 | Loss: 0.3699
Epoch: 0001/0003 | Batch1750/3978 | Loss: 0.3033
Epoch: 0001/0003 | Batch2000/3978 | Loss: 0.4907
Epoch: 0001/0003 | Batch2250/3978 | Loss: 0.4904
Epoch: 0001/0003 | Batch2500/3978 | Loss: 0.5781
Epoch: 0001/0003 | Batch2750/3978 | Loss: 0.3089
Epoch: 0001/0003 | Batch3000/3978 | Loss: 0.5004
Epoch: 0001/0003 | Batch3250/3978 | Loss: 0.5745
Epoch: 0001/0003 | Batch3500/3978 | Loss: 0.3185
Epoch: 0001/0003 | Batch3750/3978 | Loss: 0.2720
Training accuracy: 94.18%
Valid accuracy: 89.97%
Time elapsed: 21.62 min
```

```
Epoch: 0002/0003 | Batch0000/3978 | Loss: 0.2346
Epoch: 0002/0003 | Batch0250/3978 | Loss: 0.3588
Epoch: 0002/0003 | Batch0500/3978 | Loss: 0.3800
Epoch: 0002/0003 | Batch0750/3978 | Loss: 0.3329
Epoch: 0002/0003 | Batch1000/3978 | Loss: 0.0612
Epoch: 0002/0003 | Batch1250/3978 | Loss: 0.0962
Epoch: 0002/0003 | Batch1500/3978 | Loss: 0.0458
Epoch: 0002/0003 | Batch1750/3978 | Loss: 0.1701
Epoch: 0002/0003 | Batch2000/3978 | Loss: 0.0365
Epoch: 0002/0003 | Batch2250/3978 | Loss: 0.0840
Epoch: 0002/0003 | Batch2500/3978 | Loss: 0.2626
Epoch: 0002/0003 | Batch2750/3978 | Loss: 0.0334
Epoch: 0002/0003 | Batch3000/3978 | Loss: 0.0075
Epoch: 0002/0003 | Batch3250/3978 | Loss: 0.1849
Epoch: 0002/0003 | Batch3500/3978 | Loss: 0.0276
Epoch: 0002/0003 | Batch3750/3978 | Loss: 0.2849
Training accuracy: 98.04%
Valid accuracy: 90.51%
Time elapsed: 43.29 min
```

```
Epoch: 0003/0003 | Batch0000/3978 | Loss: 0.0086
Epoch: 0003/0003 | Batch0250/3978 | Loss: 0.0724
Epoch: 0003/0003 | Batch0500/3978 | Loss: 0.0040
Epoch: 0003/0003 | Batch0750/3978 | Loss: 0.1930
Epoch: 0003/0003 | Batch1000/3978 | Loss: 0.0699
Epoch: 0003/0003 | Batch1250/3978 | Loss: 0.1635
Epoch: 0003/0003 | Batch1500/3978 | Loss: 0.0047
Epoch: 0003/0003 | Batch1750/3978 | Loss: 0.0046
Epoch: 0003/0003 | Batch2000/3978 | Loss: 0.0043
Epoch: 0003/0003 | Batch2250/3978 | Loss: 0.0039
Epoch: 0003/0003 | Batch2500/3978 | Loss: 0.0215
Epoch: 0003/0003 | Batch2750/3978 | Loss: 0.0178
Epoch: 0003/0003 | Batch3000/3978 | Loss: 0.0273
Epoch: 0003/0003 | Batch3250/3978 | Loss: 0.1615
Epoch: 0003/0003 | Batch3500/3978 | Loss: 0.0074
Epoch: 0003/0003 | Batch3750/3978 | Loss: 0.0701
Training accuracy: 98.90%
Valid accuracy: 90.49%
Time elapsed: 64.97 min
Total Training Time: 64.97 min
Test accuracy: 90.96%
```

7500NLP-4 (Bert for F/T Classification)

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight

import time
import torch
from transformers import DistilBertForSequenceClassification
from transformers import DistilBertTokenizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, matthews_corrcoef,
roc_curve, auc

df = pd.read_csv('C:/Users/siyua/OneDrive/Desktop/cleaned_data.csv', encoding='ISO-8859-1')
print(df.head())

# See the distribution of dataset
# Create figure and axis
```

```python
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

dimensions = {
    "I/E": ["I", "E"],
    "S/N": ["S", "N"],
    "F/T": ["F", "T"],
    "P/J": ["P", "J"],
}

# Draw the plot
for ax, (col, labels) in zip(axes.flatten(), dimensions.items()):
    counts = df[col].value_counts()
    ax.bar(labels, [counts.get(1, 0), counts.get(0, 0)], color='skyblue')
    ax.set_title(f"{col} Proportion")
    ax.set_ylabel("Count")
    ax.set_xticks([0, 1])
    ax.set_xticklabels(labels)

plt.tight_layout()
plt.show()

# Split dataset into training set, validation set and test set
X = df['clean_posts']
y = df['F/T']

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42,
stratify=y_temp)

# X_train: 60%
# X_val: 20%
# X_test: 20%

print("Training set:", X_train.shape, y_train.shape)
print("Validation set:", X_val.shape, y_val.shape)
print("Test set:", X_test.shape, y_test.shape)

# Bert Model
# 1. Bert Tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

def tokenize_data(texts, tokenizer, max_len=256):
    return tokenizer(texts.tolist(), return_tensors='pt', max_length=max_len, padding='max_length',
truncation=True)
```

```python
train_encodings = tokenize_data(X_train, tokenizer)
val_encodings = tokenize_data(X_val, tokenizer)
test_encodings = tokenize_data(X_test, tokenizer)

# 2. Create data loader
class MBTIDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

train_dataset = MBTIDataset(train_encodings, y_train.tolist())
val_dataset = MBTIDataset(val_encodings, y_val.tolist())
test_dataset = MBTIDataset(test_encodings, y_test.tolist())

# 3. WeightedRandomSampler
class_counts = y_train.value_counts().to_dict()
weights = [1.0 / class_counts[label] for label in y_train]
sampler = torch.utils.data.WeightedRandomSampler(weights, len(weights))

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=16, sampler=sampler)
valid_loader = torch.utils.data.DataLoader(val_dataset, batch_size=16, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=16, shuffle=True)

# 4. Load and fine-tune pre-trained Bert models
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
model.to(DEVICE)
model.train()

optim = torch.optim.Adam(model.parameters(), lr=5e-5)

def compute_accuracy(model, data_loader, device):
    with torch.no_grad():
        correct_pred, num_examples = 0, 0
```

```
        for batch_idx, batch in enumerate(data_loader):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs['logits']
            predicted_labels = torch.argmax(logits, 1)
            num_examples += labels.size(0)
            correct_pred += (predicted_labels == labels).sum()

    return correct_pred.float() / num_examples * 100

start_time = time.time()
NUM_EPOCHS = 3

for epoch in range(NUM_EPOCHS):

    model.train()

    for batch_idx, batch in enumerate(train_loader):

        ### Prepare data
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        ### Forward pass
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss, logits = outputs['loss'], outputs['logits']

        ### Backward pass
        optim.zero_grad()
        loss.backward()
        optim.step()

        ### Logging
        if not batch_idx % 250:
            print(f'Epoch: {epoch+1:04d}/{NUM_EPOCHS:04d}'
                  f' | Batch'
                  f'{batch_idx:04d}/'
                  f'{len(train_loader):04d} | '
                  f'Loss: {loss:.4f}')
```

```python
    model.eval()


    with torch.set_grad_enabled(False):
        print(f'Training accuracy: '
              f'{compute_accuracy(model, train_loader, DEVICE):.2f}%'
              f'\nValid accuracy: '
              f'{compute_accuracy(model, valid_loader, DEVICE):.2f}%')


    print(f'Time elapsed: {(time.time() - start_time)/60:.2f} min')


print(f'Total Training Time: {(time.time() - start_time)/60:.2f} min')
print(f'Test accuracy: {compute_accuracy(model, test_loader, DEVICE):.2f}%')
```

Output:

```
Epoch: 0001/0003 | Batch0000/3978 | Loss: 0.6938
Epoch: 0001/0003 | Batch0250/3978 | Loss: 0.5164
Epoch: 0001/0003 | Batch0500/3978 | Loss: 0.7098
Epoch: 0001/0003 | Batch0750/3978 | Loss: 0.4506
Epoch: 0001/0003 | Batch1000/3978 | Loss: 1.0582
Epoch: 0001/0003 | Batch1250/3978 | Loss: 0.5330
Epoch: 0001/0003 | Batch1500/3978 | Loss: 0.7261
Epoch: 0001/0003 | Batch1750/3978 | Loss: 0.2675
Epoch: 0001/0003 | Batch2000/3978 | Loss: 0.3309
Epoch: 0001/0003 | Batch2250/3978 | Loss: 0.3628
Epoch: 0001/0003 | Batch2500/3978 | Loss: 0.2950
Epoch: 0001/0003 | Batch2750/3978 | Loss: 0.5004
Epoch: 0001/0003 | Batch3000/3978 | Loss: 0.2120
Epoch: 0001/0003 | Batch3250/3978 | Loss: 0.1940
Epoch: 0001/0003 | Batch3500/3978 | Loss: 0.3319
Epoch: 0001/0003 | Batch3750/3978 | Loss: 0.3656
Training accuracy: 85.79%
Valid accuracy: 78.24%
Time elapsed: 21.59 min
```

```
Epoch: 0002/0003 | Batch0000/3978 | Loss: 0.3400
Epoch: 0002/0003 | Batch0250/3978 | Loss: 0.2638
Epoch: 0002/0003 | Batch0500/3978 | Loss: 0.3804
Epoch: 0002/0003 | Batch0750/3978 | Loss: 0.3694
Epoch: 0002/0003 | Batch1000/3978 | Loss: 0.1864
Epoch: 0002/0003 | Batch1250/3978 | Loss: 0.3413
Epoch: 0002/0003 | Batch1500/3978 | Loss: 0.3218
Epoch: 0002/0003 | Batch1750/3978 | Loss: 0.2885
Epoch: 0002/0003 | Batch2000/3978 | Loss: 0.0831
Epoch: 0002/0003 | Batch2250/3978 | Loss: 0.2221
Epoch: 0002/0003 | Batch2500/3978 | Loss: 0.3736
Epoch: 0002/0003 | Batch2750/3978 | Loss: 0.2695
Epoch: 0002/0003 | Batch3000/3978 | Loss: 0.3383
Epoch: 0002/0003 | Batch3250/3978 | Loss: 0.1526
Epoch: 0002/0003 | Batch3500/3978 | Loss: 0.4593
Epoch: 0002/0003 | Batch3750/3978 | Loss: 0.2812
Training accuracy: 91.40%
Valid accuracy: 81.51%
Time elapsed: 43.33 min
```

```
Epoch: 0003/0003 | Batch0000/3978 | Loss: 0.3417
Epoch: 0003/0003 | Batch0250/3978 | Loss: 0.0668
Epoch: 0003/0003 | Batch0500/3978 | Loss: 0.1208
Epoch: 0003/0003 | Batch0750/3978 | Loss: 0.1609
Epoch: 0003/0003 | Batch1000/3978 | Loss: 0.1380
Epoch: 0003/0003 | Batch1250/3978 | Loss: 0.3669
Epoch: 0003/0003 | Batch1500/3978 | Loss: 0.7141
Epoch: 0003/0003 | Batch1750/3978 | Loss: 0.0700
Epoch: 0003/0003 | Batch2000/3978 | Loss: 0.4155
Epoch: 0003/0003 | Batch2250/3978 | Loss: 0.5027
Epoch: 0003/0003 | Batch2500/3978 | Loss: 0.0544
Epoch: 0003/0003 | Batch2750/3978 | Loss: 0.1144
Epoch: 0003/0003 | Batch3000/3978 | Loss: 0.1711
Epoch: 0003/0003 | Batch3250/3978 | Loss: 0.2044
Epoch: 0003/0003 | Batch3500/3978 | Loss: 0.3838
Epoch: 0003/0003 | Batch3750/3978 | Loss: 0.3467
Training accuracy: 94.00%
Valid accuracy: 81.60%
Time elapsed: 65.09 min
Total Training Time: 65.09 min
Test accuracy: 81.45%
```

7500NLP-5 (Bert for P/J Classification)

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight

import time
import torch
from transformers import DistilBertForSequenceClassification
from transformers import DistilBertTokenizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, matthews_corrcoef,
roc_curve, auc

df = pd.read_csv('C:/Users/siyua/OneDrive/Desktop/cleaned_data.csv', encoding='ISO-8859-1')
print(df.head())

# See the distribution of dataset
# Create figure and axis
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

dimensions = {
    "I/E": ["I", "E"],
    "S/N": ["S", "N"],
    "F/T": ["F", "T"],
    "P/J": ["P", "J"],
}
```

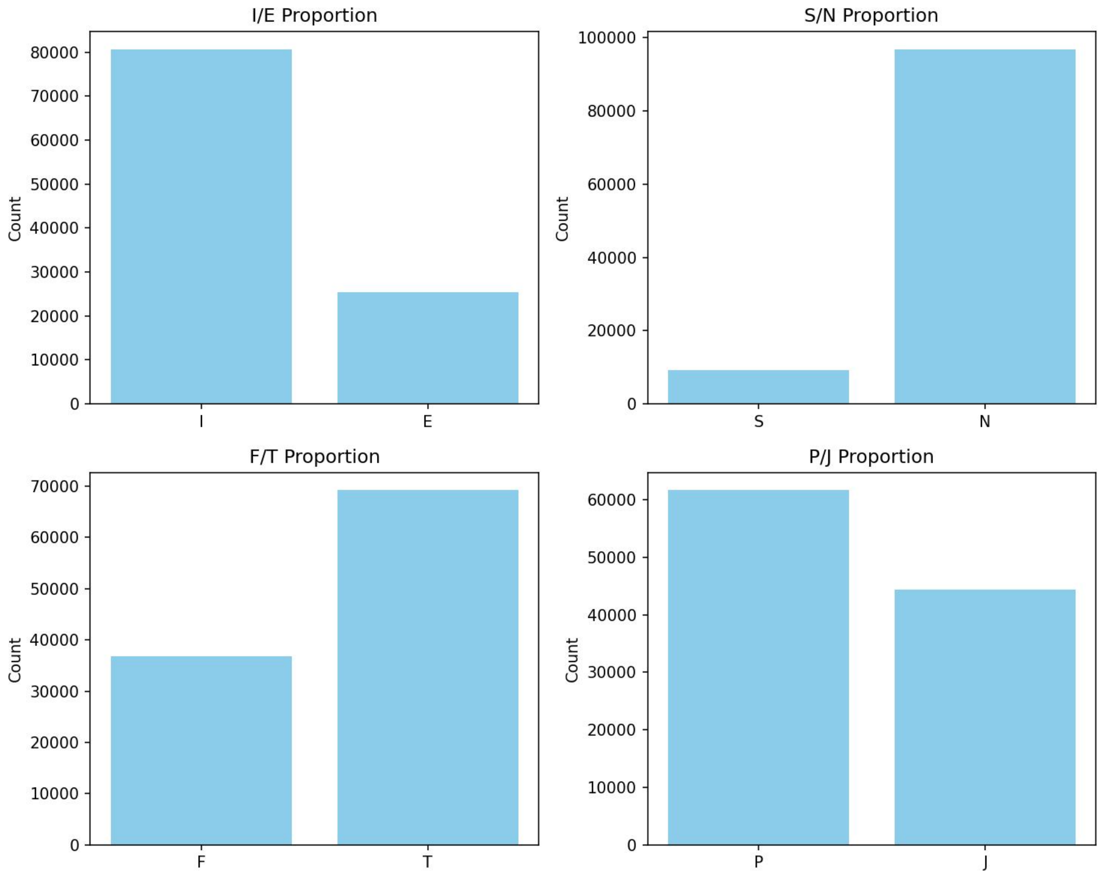
```python
# Draw the plot
for ax, (col, labels) in zip(axes.flatten(), dimensions.items()):
    counts = df[col].value_counts()
    ax.bar(labels, [counts.get(1, 0), counts.get(0, 0)], color='skyblue')
    ax.set_title(f"{col} Proportion")
    ax.set_ylabel("Count")
    ax.set_xticks([0, 1])
    ax.set_xticklabels(labels)

plt.tight_layout()
plt.show()

# Split dataset into training set, validation set and test set
X = df['clean_posts']
y = df['P/J']

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42,
stratify=y_temp)

# X_train: 60%
# X_val: 20%
# X_test: 20%

print("Training set:", X_train.shape, y_train.shape)
print("Validation set:", X_val.shape, y_val.shape)
print("Test set:", X_test.shape, y_test.shape)

# Bert Model
# 1. Bert Tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

def tokenize_data(texts, tokenizer, max_len=512):
    return tokenizer(texts.tolist(), return_tensors='pt', max_length=max_len, padding='max_length',
truncation=True)

train_encodings = tokenize_data(X_train, tokenizer)
val_encodings = tokenize_data(X_val, tokenizer)
test_encodings = tokenize_data(X_test, tokenizer)

# 2. Create data loader
class MBTIDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
```

```python
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

train_dataset = MBTIDataset(train_encodings, y_train.tolist())
val_dataset = MBTIDataset(val_encodings, y_val.tolist())
test_dataset = MBTIDataset(test_encodings, y_test.tolist())

# 3. WeightedRandomSampler
class_counts = y_train.value_counts().to_dict()
weights = [1.0 / class_counts[label] for label in y_train]
sampler = torch.utils.data.WeightedRandomSampler(weights, len(weights))

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=16, sampler=sampler)
valid_loader = torch.utils.data.DataLoader(val_dataset, batch_size=16, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=16, shuffle=True)

# 4. Load and fine-tune pre-trained Bert models
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
model.to(DEVICE)
model.train()

optim = torch.optim.Adam(model.parameters(), lr=1e-5)

def compute_accuracy(model, data_loader, device):
    with torch.no_grad():
        correct_pred, num_examples = 0, 0
        for batch_idx, batch in enumerate(data_loader):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs['logits']
            predicted_labels = torch.argmax(logits, 1)
            num_examples += labels.size(0)
```

```
        correct_pred += (predicted_labels == labels).sum()

    return correct_pred.float() / num_examples * 100

start_time = time.time()
NUM_EPOCHS = 3

for epoch in range(NUM_EPOCHS):

    model.train()

    for batch_idx, batch in enumerate(train_loader):

        ### Prepare data
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        ### Forward pass
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss, logits = outputs['loss'], outputs['logits']

        ### Backward pass
        optim.zero_grad()
        loss.backward()
        optim.step()

        ### Logging
        if not batch_idx % 250:
            print(f'Epoch: {epoch+1:04d}/{NUM_EPOCHS:04d}'
                  f' | Batch'
                  f'{batch_idx:04d}/'
                  f'{len(train_loader):04d} | '
                  f'Loss: {loss:.4f}')

    model.eval()

    with torch.set_grad_enabled(False):
        print(f'Training accuracy: '
              f'{compute_accuracy(model, train_loader, DEVICE):.2f}%'
              f'\nValid accuracy: '
              f'{compute_accuracy(model, valid_loader, DEVICE):.2f}%')

    print(f'Time elapsed: {(time.time() - start_time)/60:.2f} min')
```

```
print(f'Total Training Time: {(time.time() - start_time)/60:.2f} min')

print(f'Test accuracy: {compute_accuracy(model, test_loader, DEVICE):.2f}%')
```

Output:




```
Epoch: 0001/0003 | Batch0000/3978 | Loss: 0.6845
Epoch: 0001/0003 | Batch0250/3978 | Loss: 0.7255
Epoch: 0001/0003 | Batch0500/3978 | Loss: 0.5540
Epoch: 0001/0003 | Batch0750/3978 | Loss: 0.7222
Epoch: 0001/0003 | Batch1000/3978 | Loss: 0.5454
Epoch: 0001/0003 | Batch1250/3978 | Loss: 0.3730
Epoch: 0001/0003 | Batch1500/3978 | Loss: 0.6154
Epoch: 0001/0003 | Batch1750/3978 | Loss: 0.5853
Epoch: 0001/0003 | Batch2000/3978 | Loss: 0.3266
Epoch: 0001/0003 | Batch2250/3978 | Loss: 0.6249
Epoch: 0001/0003 | Batch2500/3978 | Loss: 0.3957
Epoch: 0001/0003 | Batch2750/3978 | Loss: 0.6442
Epoch: 0001/0003 | Batch3000/3978 | Loss: 0.5204
Epoch: 0001/0003 | Batch3250/3978 | Loss: 0.7706
Epoch: 0001/0003 | Batch3500/3978 | Loss: 0.2053
Epoch: 0001/0003 | Batch3750/3978 | Loss: 0.5987
Training accuracy: 76.86%
Valid accuracy: 74.69%
Time elapsed: 47.75 min
```

```
Epoch: 0002/0003 | Batch0000/3978 | Loss: 0.5171
Epoch: 0002/0003 | Batch0250/3978 | Loss: 0.4552
Epoch: 0002/0003 | Batch0500/3978 | Loss: 0.5733
Epoch: 0002/0003 | Batch0750/3978 | Loss: 0.4107
Epoch: 0002/0003 | Batch1000/3978 | Loss: 0.6254
Epoch: 0002/0003 | Batch1250/3978 | Loss: 0.4866
Epoch: 0002/0003 | Batch1500/3978 | Loss: 0.4892
Epoch: 0002/0003 | Batch1750/3978 | Loss: 0.5747
Epoch: 0002/0003 | Batch2000/3978 | Loss: 0.5595
Epoch: 0002/0003 | Batch2250/3978 | Loss: 0.3841
Epoch: 0002/0003 | Batch2500/3978 | Loss: 0.4201
Epoch: 0002/0003 | Batch2750/3978 | Loss: 0.5781
Epoch: 0002/0003 | Batch3000/3978 | Loss: 0.5069
Epoch: 0002/0003 | Batch3250/3978 | Loss: 0.3413
Epoch: 0002/0003 | Batch3500/3978 | Loss: 0.5731
Epoch: 0002/0003 | Batch3750/3978 | Loss: 0.4137
Training accuracy: 83.03%
Valid accuracy: 75.67%
Time elapsed: 95.57 min
```

```
Epoch: 0003/0003 | Batch0000/3978 | Loss: 0.3644
Epoch: 0003/0003 | Batch0250/3978 | Loss: 0.2467
Epoch: 0003/0003 | Batch0500/3978 | Loss: 0.3685
Epoch: 0003/0003 | Batch0750/3978 | Loss: 0.2589
Epoch: 0003/0003 | Batch1000/3978 | Loss: 0.4437
Epoch: 0003/0003 | Batch1250/3978 | Loss: 0.4245
Epoch: 0003/0003 | Batch1500/3978 | Loss: 0.3833
Epoch: 0003/0003 | Batch1750/3978 | Loss: 0.3120
Epoch: 0003/0003 | Batch2000/3978 | Loss: 0.5036
Epoch: 0003/0003 | Batch2250/3978 | Loss: 0.4839
Epoch: 0003/0003 | Batch2500/3978 | Loss: 0.6509
Epoch: 0003/0003 | Batch2750/3978 | Loss: 0.2016
Epoch: 0003/0003 | Batch3000/3978 | Loss: 0.3986
Epoch: 0003/0003 | Batch3250/3978 | Loss: 0.2104
Epoch: 0003/0003 | Batch3500/3978 | Loss: 0.3600
Epoch: 0003/0003 | Batch3750/3978 | Loss: 0.7193
Training accuracy: 86.73%
Valid accuracy: 75.45%
Time elapsed: 143.37 min
Total Training Time: 143.37 min
Test accuracy: 75.64%
```

7500NLP-EDA (Distribution of word counts in posts)

Code:

```python
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('C:/Users/siyua/OneDrive/Desktop/cleaned_data.csv', encoding='ISO-8859-1')
print(df.head())

# 1. EDA
# 1.1. Split the review into words
df['word_count'] = df['clean_posts'].str.split().str.len()

# 1.2. Draw the plot
plt.figure(figsize=(10, 6))
plt.hist(df['word_count'], bins=30, color='red', edgecolor='black')
```

```
plt.title('Distribution of Word Counts in Reviews')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
df = df.drop(['word_count'], axis=1)
```

Output: