AI Capstone Project 1
Tang Poetry Style Classifier

110550095 王皓平

1 Introduction

The poetry of the Tang Dynasty possesses a unique style and is hailed as the pinnacle of classical Chinese poetry. Tang poets expressing their current emotions and thoughts with individualistic stylistic techniques, while obeying the rules of rhyme, tone, and word usage. Modern scholars classify several famous Tang poems into different poetic styles based on their descriptive techniques or themes.

The Four Great Styles of the Sheng Tang are well-known: the Romantic Styles, the Pastoral Landscape Styles, the Social Realism Styles, and the Frontier Fortress Styles. The project aims to train models that can distinguish the style of the poems which are not clearly classified yet.

2 Dataset

First, identify the poets representing the Four Great Styles of Poetry and their respective works in "The Complete Collection of Tang Poetry". Then, use web crawling tool such as Beautiful Soup to retrieve the works of these poets from an open-source database of Tang poetry. Save the lines of poetry as CSV files on a per-sentence basis, with two attributes for training and testing data: the text ba line of poetry and the label be the style of poeticd.

The table of the plain data after fetching

Style	Romantic	Pastoral Landscape	Social Realism	Frontier Fortress
Poet	李白	王維、孟浩然	杜甫	岑參、高適、王昌齡
Amount	6759	4526	9664	4761

Due to the uneven distribution of data, I've decided to augment the data for the Romantic style, the Pastoral Landscape style, and the Frontier Fortress style. My first attempt involved translating the poetry lines into different languages and then back into Chinese to achieve similar meanings. However, this method resulted in translated texts that were more like short passages instead of a line, which deviated significantly from the original data pattern, so I decided not to use it.

The second approach I tried involved changing the order of the sentences. Each poetry line is split by commas and exchanged their positions. For example, "床前明月光，疑似地上霜" will become "疑似地上霜，床前明月光". This method preserves the meaning of original poetic while adhering more closely to the rules and format of Tang poetry.

The table of the plain data after fetching

Style	Romantic	Pastoral Landscape	Social Realism	Frontier Fortress
Poet	李白	王維、孟浩然	杜甫	岑參、高適、王昌齡
Amount	8999	8989	9664	8997

3 Supervised learning

3.1 Naive Bayes Classifier

Naive Bayes classifier is a machine learning model based on probability theory, which assumes that each feature is independent of others for classification. It has low computational complexity and fast training speed, making it suitable for large-scale datasets. Additionally, Naive Bayes classifier is less sensitive to missing data, therefore I chose it to be the first train model for the project.

Bayes' Theorem:

$$P(y|x) = \frac{P(X|y_i) \cdot P(y_i)}{P(X)} \tag{1}$$

where:

- $P(y_i|X)$ is the probability of class y_i, given the feature X.
- $P(X|y_i)$ is the likelyhood, which representts the probability of observing the feature X given class y_i.
- $P(y_i)$ 和 $P(X)$ is the probabilities of y_i and X.

MAP, or Maximum a Posteriori estimation, is a principle in Bayesian statistics that aims to find the most probable hypothesis (or parameter values) given the observed data. In the context of classification, MAP estimation is used to determine the most likely class label given a set of features.

Mathematically, it can be represented as:

$$arg\ max_{y_i} P(y_i|X) \tag{2}$$

Scikit-learn provides implementations for three types of Naive Bayes classifiers:

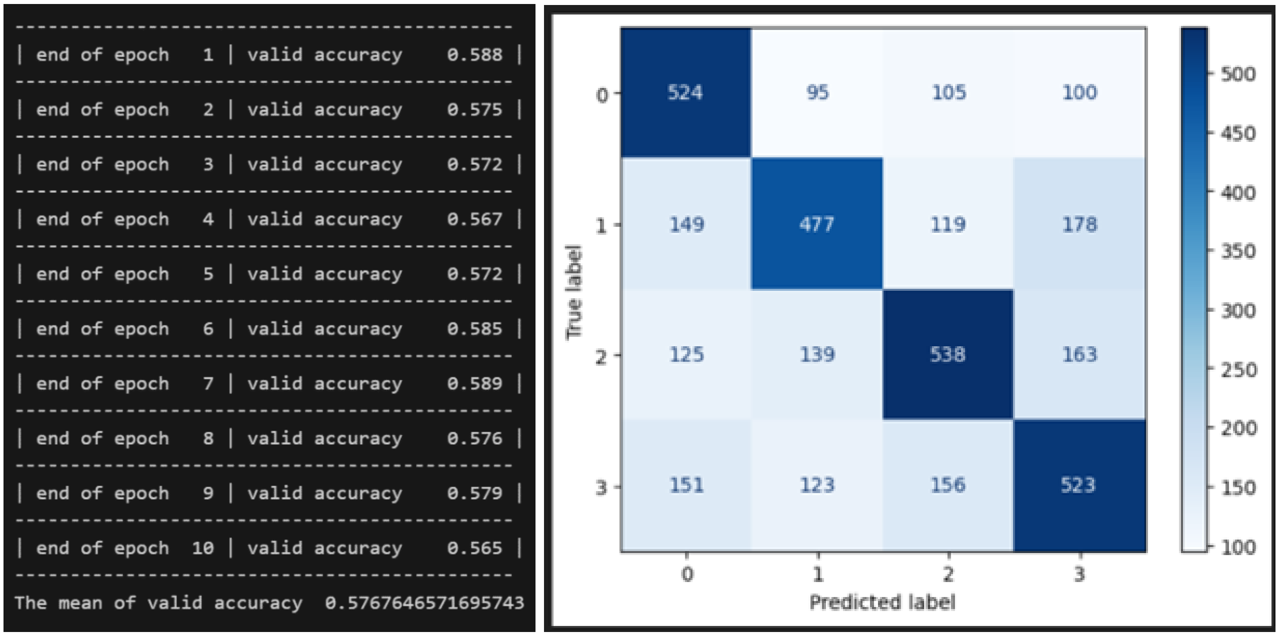
- Gaussian Naive Bayes (GaussianNB): This classifier assumes that the features are continuous-valued and follow a Gaussian distribution.
- Multinomial Naive Bayes (MultinomialNB): This classifier is suitable for features that represent counts or frequencies, such as word counts in text classification.
- Bernoulli Naive Bayes (BernoulliNB): This classifier is useful when features are binary-valued, such as presence or absence of a feature.

Since the project involves text classification and the statistical features are the frequencies of words, which follow a discrete distribution, I chose to use the **MultinomialNB** classifier.

First, segment each Chinese text data into words. Considering the differences between Tang poems and modern articles, which have more writing rules and semantic associations at the level of individual characters, I segmented the texts at the character level. I used the open-source tool Jieba for word segmentation. The segmented word corpus resulted in a total of 4874 Chinese characters.

Next, extract feature values based on the frequency of each word's occurrence with the CountVectorizer, text feature extraction tool from sklearn, to generate a sparse matrix.

The data was split into training and testing sets in a 9:1 ratio. I used cross-validation from sklearn to perform model evaluation. The cross-validation strategy employed k-fold cross-validation with 10 folds. The mean accuracy score obtained from cross-validation was 0.58 and the confiusion metrx is showed below.



(a) accuracy

(b) confusiosn metrix

Figure 1: result of MultinomialNB

3.2 Feedforward Neural Network (FNN)

Different poetic styles and poets in Tang poetry typically express their attitudes and emotions with specific ways. To analyze the emotions contained in the poetry lines, I chose to use deep learning to assist in the classification of poetic styles.

There are several types of neural networks in deep learning:

- Feedforward Neural Networks (FNN): The most basic neural network structure where data passes in one direction without feedback connections.
- Convolutional Neural Networks (CNN): Mainly used for processing data with grid structures, such as images.
- Recurrent Neural Networks (RNN): A type of neural network with recurrent connections that can handle variable-length sequential data and preserve temporal information in the data.

Since this project only classifies Tang poetry into four categories, I chose to implement a simpler FNN, which also allows for faster training.

Firstly, I established a tokenizer and vocabulary so that each character in the poetry lines could be represented by an integer. The data was split into training and testing sets in a 9:1 ratio. The data was then batched for training to reduce the variance of gradient estimates per batch and improve the stability and efficiency of optimization.

The neural network used for training consists of two layers: an Embedding Layer and a Fully Connected Layer. The Embedding Layer maps words in the text data to high-dimensional word embedding vectors, preserving semantic relationships between words and providing continuous value features for word representation. For example, words like " 明" and " 光" are related, so their feature vectors will be closer. The Fully Connected Layer maps the output features of the previous layer to category labels, achieving the final text classification.

Since the database is custom-defined, cross-validation was not performed using existing packages. 1/20 of the training dataset was used as a validation dataset, and ten epochs were conducted for cross-validation. The average accuracy obtained after cross-validation was 0.66.

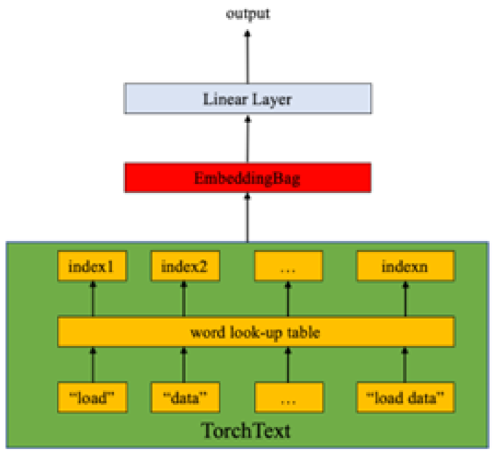
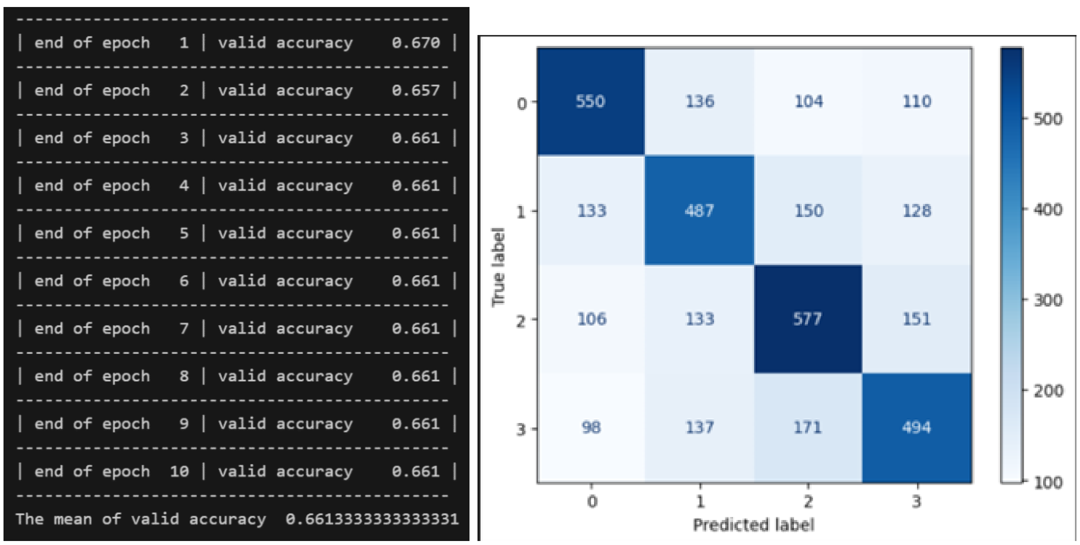


Figure 2: FNN model



(a) accuracy

(b) confusiosn metrix

Figure 3: result of FNN

4 Unsupervised learning

4.1 Clustering

Clustering is an unsupervised learning technique aimed at partitioning the objects in a dataset into different groups, such that objects within the same group are similar to each other, while objects in different groups are dissimilar. This similarity is typically defined based on distance or similarity measures in the feature space. The goal of clustering is to discover the underlying structure in the data and group similar objects together, thereby revealing patterns and organization within the data.

First, I used the Jieba tokenizer and the CountVectorizer text feature extraction tool from sklearn to extract high-dimensional feature vectors from the poetry lines. Then, I utilized the PCA tool from sklearn to reduce the data to two dimensions for easy visualization on a coordinate system. PCA can project high-dimensional data into a lower-dimensional space through linear transformation while retaining as much information as possible.

After visualizing the reduced-dimensional data, it was evident that the data roughly clustered into four categories. There were also some remaining data points that were close together due to similar features, leaving a small number of isolated data points should be considered as noise.

Data clustering is primarily considered using the following two methods:

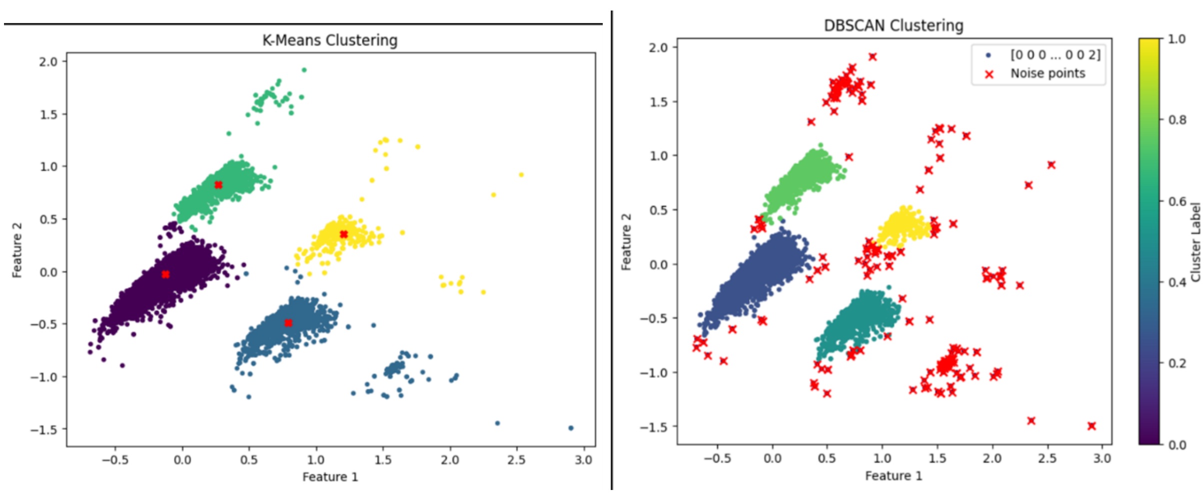
- K-Means Clustering: This method divides data points into K non-overlapping groups, with each group representing a cluster. Through iterative optimization to minimize the sum of squared distances between each data point and its cluster centroid, the clustering results are ultimately based on distance.
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise): This method is based on density, where data points are considered objects in high-density regions. It identifies clusters by defining a density threshold and is suitable for identifying clusters with irregular shapes. Additionally, it can detect noise in the data.

4.1.1 K-Means Clustering

The K-means clustering results align well with the expectations, with the data grouped into four clusters as intended. Although the clusters looks reasonable, some data that are far away from the center point, which is represent in red cross, should be noises and not be considered.

4.1.2 DBSCAN

Modify eps and min_sample parameters to achieve the best results. The four clusters look reasonable and the data representing in red cross are far away from the others, and are labeled as noises.



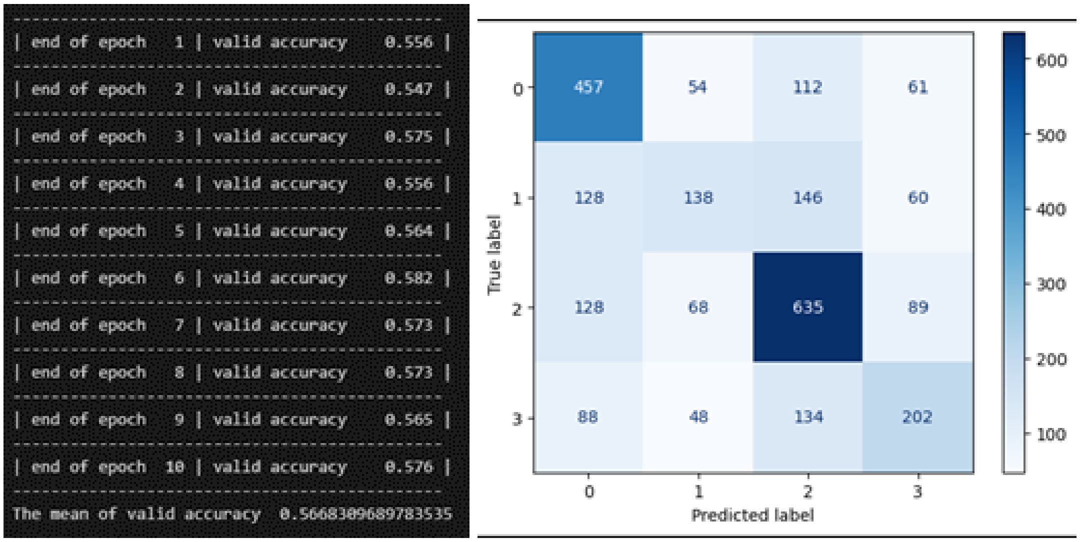
(a) K-means (b) DBSCAN

Figure 4: Visualization of the clusters

5 Experiment and Discussion

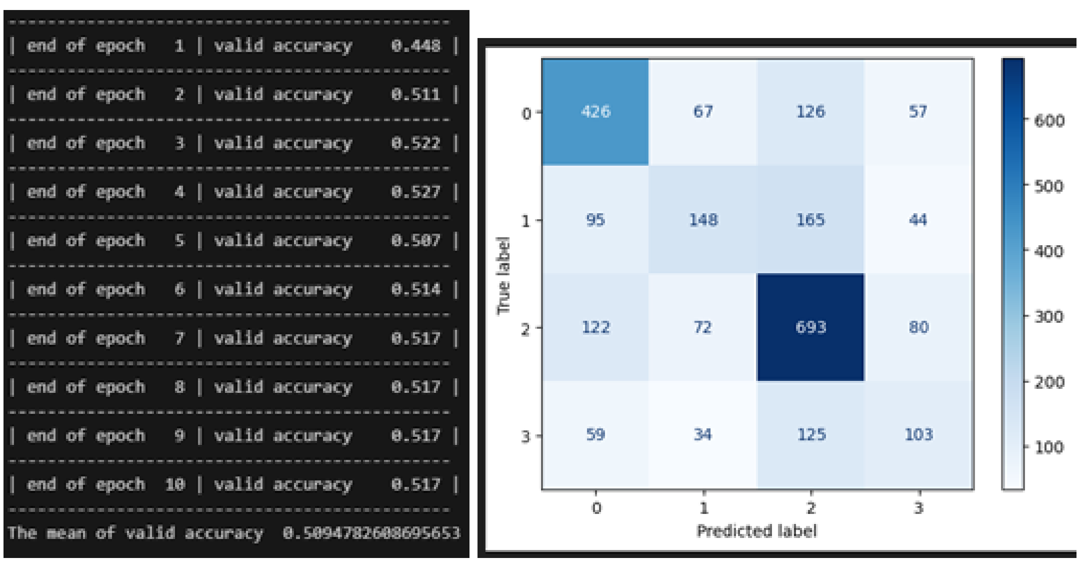
For the experiment, the supervised learning methods above are implemented on the data without balancing and augmentation.

The performance from both the algorithm have degraded, and the confusion metrics show that most of the poems are predicted to be in the Social Realism Styles. This is probably because of the data is unevenly distributed, with density concentrated in the Social Realism Styles.



(a) accuracy (b) confusiosn metrix

Figure 5: result of Naive Bayes without balancing and augmentation



(a) accuracy (b) confusiosn metrix

Figure 6: result of FNN without balancing and augmentation

Based on the results obtained in this project, the poetry style classifier performed quite well. It seems that the differences in style among different poets can indeed be discerned from the text itself, and the classification of poetry styles by modern scholars is reasonable. The effectiveness of the classifier may be attributed to the ample amount of data available, and the clear distinction of features among different poetry styles as evidenced by the distribution plots after dimensionality reduction.

If there were more time, I would like to experiment with different preprocessing techniques for ++the data, such as altering the tokenization standards and considering word-level features. Additionally, I would consider incorporating works from other poetry styles into the dataset and observe the results after retraining the model.

In terms of unsupervised learning, I am interested in trying Hierarchical Clustering to observe if there are hierarchical levels among poetry styles, such as some smaller poetry styles being subsumed under larger ones.

Currently, the data is organized at the level of individual lines of poetry, but not every line may contain features specific to a particular poetry style. Therefore, it would be worth exploring training models with data organized at the level of whole poems.

In addition to poetry style classification, the workflow for building this classifier could also be applied to distinguishing seasons or categorizing emotions in poetry. Perhaps in the future, when students encounter questions related to Tang poetry in exams, they could refer to the results obtained from training models using large databases of Tang poetry to identify the correct answers.

References

[1] 全唐詩- 中國哲學書電子化計劃 https://ctext.org/quantangshi/

[2] 蘑菇毒性判断中测试贝叶斯分类器中 GaussianNB, MultinomialNB,BernouliNB 的分类效果 https://blog.csdn.net/m0_71805359/article/details/134764010

[3] 自動商品分類器 https://kevinchung0921.medium.com/%E6%88%91%E7%9A%84app%E9%96%8B%E7%99%BC%E4%B9%8B%E8%B7%AF-%E8%87%AA%E5%8B%95%E5%95%86%E5%93%81%E5%88%86%E9%A1%9E%E5%99%A8-%E7%88%AC%E8%9F%B2%E8%88%87%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-391f0e89b9d4

[4] DBSCAN 分群法詳解 https://ctext.org/quantangshi/

[5] Text classification with the torchtext library https://pytorch.org/tutorials/beginner/text_sentiment_ngrams_tutorial.html

A Appendix

A.1 Data Collection

Code 1: web crowling

```python
# plain data collect
import requests as rq
from bs4 import BeautifulSoup
import re
import csv
base_url = "https://ctext.org/quantangshi/"
folder = "plain_data/"
file_path = ["浪漫派.csv", "山水田園派.csv", "社會寫實派.csv", "邊塞派.csv"]
books = [[[161, 185]], [[125, 128], [159, 160], [350, 353]], [[216, 234]]
         , [[198, 201], [211, 214], [140, 143]]]
for file_itr in range(4):
    with open (folder+file_path[file_itr], 'w+', newline='', encoding='utf-8-sig') as csvfile:
        cnt = 0
        csvwriter = csv.writer(csvfile)
        for j in range(len(books[file_itr])):
            for i in range(books[file_itr][j][0], books[file_itr][j][1]+1):
                rsp_web = rq.get(base_url+str(i))
                soup = BeautifulSoup(rsp_web.text, 'html.parser')
                content = soup.find('div', id = 'content3')
                ctexts = content.select('.ctext:not([class*=" "])')
                for ctext in ctexts:
                    if ctext.get_text().strip() == '':
                        continue
                    text = ctext.get_text().replace('\n','')
                    lines = re.split('[。？]', text)
                    for line in lines:
                        if line.strip()=='' or re.compile(r'[^\u4e00-\u9fff]').match(line):
                            continue
                        csvwriter.writerow([line])
                        cnt+=1
        print(folder+file_path[file_itr]+" done\ndata size: "+ str(cnt))
```

Code 2: data balance and augmentation

```python
#data balance and augmentation
import random
import csv
def data_augmentation(input_path, output_path):
    with open (input_path, 'r', newline='', encoding='utf-8-sig') as inputfile, \
         open(output_path, 'w', newline='', encoding='utf-8-sig') as outputfile:
        cnt = 0
        reader = csv.reader(inputfile)
        writer = csv.writer(outputfile)
        data = list(reader)
        rate = 9000/len(data)
        for row in data:
            writer.writerows([row])
            cnt+=1
        if(rate>1):
            sample = random.sample(data, int(len(data)*(rate-1)))
            for row in sample:
                segments = row[0].split("，")
                if(len(segments)>=2):
                    s = "{}，{}".format(segments[1],segments[0])
                    writer.writerow([s])
```

```
                    cnt+=1
        print(cnt)
folder1 = "plain_data/"
folder2 = "proccessed_data/"
files = ["浪漫派.csv", "山水田園派.csv", "社會寫實派.csv", "邊塞派.csv"]
for i in range(4):
    data_augmentation(folder1+files[i], folder2+files[i])
```

Code 3: split data into testing and traing set

```
# train/text data collecting
import os
import csv

folder_path = 'proccessed_data/'
train = open('./data/train.csv', 'w+', encoding='utf-8-sig')
test = open('./data/test.csv', 'w+', encoding='utf-8-sig')
all = open('./data/all.csv', 'w+', encoding='utf-8-sig')
files = os.listdir(folder_path)

# 四大分類
dict = {"浪漫派":1, "山水田園派":2, "社會寫實派":3, "邊塞派":4}

train_cnt = 0
test_cnt = 0

for file in files:
    cnt = 0
    file_path = os.path.join(folder_path, file)
    # print(file[:-4])
    with open(file_path, 'r', encoding = 'utf-8-sig') as f:
        csvreader = csv.reader(f)
        for row in csvreader:
            text = ' '.join(row[0])
            label = dict[file[:-4]]
            data = "{}, {}".format(label, text)
            all.write("{}, {}\n".format(label, row[0]))
            if(cnt%10==0):
                test.write(data+'\n')
                test_cnt+=1
            else:
                train.write(data+'\n')
                train_cnt+=1
            cnt+=1
print("train data cnt: {}\ntest data cnt: {}".format(train_cnt, test_cnt))
print("total data cnt: {}".format(train_cnt+test_cnt))# train/text data collecting
```

A.2 Naive Bayes Classifier

Code 4: build vocabulary and implement MultinomialNB

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import accuracy_score
import jieba
import pandas as pd

all_file = './data/all.csv'
all_data = pd.read_csv(all_file, header=None,names=['label', 'text'])

```

```python
def chinese_tokenizer(text):
    return [list(jieba.cut(text))]

# 初始化vectorizer，使用的是bag-of-word 最基礎的 CountVectorizer
vectorizer = CountVectorizer(analyzer='char',  tokenizer=chinese_tokenizer)

# 將 text 轉換成 bow 格式
corpus = all_data['text'].str.replace(' ', '').values
text = vectorizer.fit_transform(corpus)

# 查看词汇表
print("Vocabulary:", vectorizer.get_feature_names_out())
print("Vocabulary num:", len(vectorizer.get_feature_names_out()))

# 查看词频矩阵
print("Word counts matrix:")
print(text.toarray())

train_text, test_text, train_label, test_label= train_test_split(text, all_data['label'],
    test_size=0.1, random_state=42)

# 實例化(Instantiate) 這個 Naive Bayes Classifier
MNB_model = MultinomialNB()

# 把資料給它，讓他根據貝氏定理，去算那些機率。
MNB_model.fit(train_text, train_label)
```

Code 5: cross-validation

```python
kf = KFold(n_splits=10, shuffle=True, random_state=42)
mnb_scores = cross_val_score(MNB_model,train_text,train_label,cv=kf,scoring='accuracy')
cnt = 0
for s in mnb_scores:
    cnt+=1
    print("-" * 45)
    print(
        "| end of epoch {:3d} | valid accuracy {:8.3f} |".format(cnt, s)
    )
print("-" * 45)
print("The mean of valid accuracy ", mnb_scores.mean())
```

Code 6: confussion metrix

```python
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support,
    ConfusionMatrixDisplay
mnb_predict = MNB_model.predict(test_text)
print(confusion_matrix(test_label, mnb_predict))
evaluation = precision_recall_fscore_support(test_label, mnb_predict, average='macro')
mnb_acc = accuracy_score(test_label, mnb_predict)
print(mnb_acc)
display=ConfusionMatrixDisplay(confusion_matrix(test_label,mnb_predict)).plot(cmap='Blues')
```

A.3 FNN

Code 7: build vocabulary

```python
# vocab building
import os
import csv
from torchtext.data.utils import get_tokenizer
```

```
from torchtext.vocab import build_vocab_from_iterator

folder_path = 'proccessed_data/'
train = open('./data/train.csv', 'r', encoding='utf-8-sig')
test = open('./data/test.csv', 'r', encoding='utf-8-sig')
files = os.listdir(folder_path)

# 四大分類
dict = {"浪漫派":1, "山水田園派":2, "社會寫實派":3, "邊塞派":4}
# 建立 tokenizer
tokenizer = get_tokenizer("basic_english")

def yield_tokens(folder_path):
    for file in files:
        file_path = os.path.join(folder_path, file)
        # print(file[:-4])
        with open(file_path, 'r', encoding = 'utf-8-sig') as f:
            csvreader = csv.reader(f)
            for row in csvreader:
                for char in row[0]:
                    # vocab building
                    yield tokenizer(char)

vocab = build_vocab_from_iterator(yield_tokens(folder_path), specials=["<unk>"])
vocab.set_default_index(vocab["<unk>"])
text_pipeline = lambda x: vocab(tokenizer(x))
label_pipeline = lambda x: int(x) - 1
```

Code 8: define dataset

```
# dataset building
import pandas as pd
from torch.utils.data import Dataset

class CustomDataset(Dataset):
    def __init__(self, csv_file, transform=None):
        self.data = pd.read_csv(csv_file)
        self.transform = transform

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

    # data: {label, text}
    def __getitem__(self, idx):
        sample = self.data.iloc[idx]
        label = sample.iloc[0]
        text = sample.iloc[1]
        if self.transform:
            label = self.transform(label)
            text = self.transform(text)
        return label, text

def transform(sample):
    return sample

test_file = './data/test.csv'
train_file = './data/train.csv'

train_dataset = CustomDataset(train_file, transform=transform)
test_dataset = CustomDataset(test_file, transform=transform)
```

Code 9: define training model

```
import torch
from torch.utils.data import DataLoader
from torch import nn

# Generate data batch and iterato
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
def collate_batch(batch):
    label_list, text_list, offsets = [], [], [0]
    for _label, _text in batch:
        label_list.append(label_pipeline(_label))
        processed_text = torch.tensor(text_pipeline(_text), dtype=torch.int64)
        text_list.append(processed_text)
        offsets.append(processed_text.size(0))
    label_list = torch.tensor(label_list, dtype=torch.int64)
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    text_list = torch.cat(text_list)
    return label_list.to(device), text_list.to(device), offsets.to(device)

# model define
class TextClassificationModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded)

num_class = 4
vocab_size = len(vocab)
emsize = 64
model = TextClassificationModel(vocab_size, emsize, num_class).to(device)
```

Code 10: training proccess

```
import time

EPOCHS = 10
LR = 5  # learning rate
BATCH_SIZE = 64
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)

def train(dataloader, epoch):
    model.train()
    total_acc, total_count = 0, 0
    log_interval = 400
    start_time = time.time()
    for idx, (label, text, offsets) in enumerate(dataloader):
        optimizer.zero_grad()
        predicted_label = model(text, offsets)
        loss = criterion(predicted_label, label)
```

```
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
        optimizer.step()
        total_acc += (predicted_label.argmax(1) == label).sum().item()
        total_count += label.size(0)
        if idx % log_interval == 0 and idx > 0:
            total_acc, total_count = 0, 0
            start_time = time.time()


def evaluate(dataloader):
    model.eval()
    total_acc, total_count = 0, 0

    with torch.no_grad():
        for idx, (label, text, offsets) in enumerate(dataloader):
            predicted_label = model(text, offsets)
            loss = criterion(predicted_label, label)
            total_acc += (predicted_label.argmax(1) == label).sum().item()
            total_count += label.size(0)
    return total_acc / total_count
```

Code 11: cross-validation

```
from torch.utils.data.dataset import random_split
from torchtext.data.functional import to_map_style_dataset

scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.1)
total_accu = None
num_train = int(len(train_dataset) * 0.95)
split_train_, split_valid_ = random_split(
    train_dataset, [num_train, len(train_dataset) - num_train]
)

train_dataloader = DataLoader(
    split_train_, batch_size=BATCH_SIZE, shuffle=True, collate_fn=collate_batch
)
valid_dataloader = DataLoader(
    split_valid_, batch_size=BATCH_SIZE, shuffle=True, collate_fn=collate_batch
)
test_dataloader = DataLoader(
    test_dataset, batch_size=BATCH_SIZE, shuffle=True, collate_fn=collate_batch
)

total = 0
for epoch in range(1, EPOCHS + 1):
    epoch_start_time = time.time()
    train(train_dataloader, epoch)
    accu_val = evaluate(valid_dataloader)
    if total_accu is not None and total_accu > accu_val:
        scheduler.step()
    else:
        total_accu = accu_val
    total+=accu_val
    print("-" * 45)
    print(
        "| end of epoch {:3d} | valid accuracy {:8.3f} |".format(epoch, accu_val)
    )
print("-" * 45)
print("The mean of valid accuracy ", total/10)
```

Code 12: confussion metrix

```
import json
import io
import re
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

def confusion_matrix(dataloader):
    confusion_matrix = np.zeros((4, 4), dtype=np.int64)
    model.eval()
    with torch.no_grad():
        for idx, (label, text, offsets) in enumerate(dataloader):
            predicted_label = model(text, offsets)

            for i in range(label.shape[0]):
                confusion_matrix[label[i], predicted_label.argmax(1)[i]] +=1
    return confusion_matrix

print(confusion_matrix(test_dataloader))

display = ConfusionMatrixDisplay(confusion_matrix(test_dataloader)).plot(cmap='Blues')
```

A.4 Clustering

Code 13: build vocabulary

```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import jieba

all_file = './data/all.csv'
all_data = pd.read_csv(all_file, header=None,names=['label', 'text'])

def chinese_tokenizer(text):
    return [list(jieba.cut(text))]

# 初始化vectorizer, 使用的是bag-of-word 最基礎的 CountVectorizer
vectorizer = CountVectorizer(analyzer='char',  tokenizer=chinese_tokenizer)

# 將 text 轉換成 bow 格式
corpus = all_data['text'].str.replace(' ', '').values
text = vectorizer.fit_transform(corpus)
```

Code 14: dimensionality reduction

```
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import numpy as np

# 降至二維
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(text.toarray())

plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], marker='.')
```

Code 15: visualize the data distribution and implement k-menas algorithm

```python
from sklearn.cluster import KMeans
silhouette_avg = 0
while(silhouette_avg<0.65):
    kmeans = KMeans(n_clusters=4)
    kmeans.fit(reduced_data)
    centers = kmeans.cluster_centers_
    silhouette_avg = silhouette_score(reduced_data, kmeans.labels_)

plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=kmeans.labels_, cmap='viridis', marker
    ='.')
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-Means Clustering')
plt.show()
print("Silhouette Score:", silhouette_avg)
```

Code 16: visualize the data distribution and implement DBSCAN algorithm

```python
dbscan = DBSCAN(eps=0.1, min_samples=40)
labels = dbscan.fit_predict(reduced_data)

# 提取核心点、边界点和噪声点
core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True
noise_mask = dbscan.labels_ == -1

num_clusters = len(set(dbscan.labels_)) - (1 if -1 in dbscan.labels_ else 0)
print(num_clusters)

# 绘制聚类结果
plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=labels, cmap='viridis', marker='.',
    label=labels)
plt.scatter(reduced_data[noise_mask, 0], reduced_data[noise_mask, 1], c='red', marker='x',
    label='Noise points')
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.colorbar(label='Cluster Label')
plt.show()
silhouette_avg = silhouette_score(reduced_data, dbscan.labels_)
print("Silhouette Score:", silhouette_avg)
```