AI Capstone - Project 1 Report

110550089 周冠辰

1. Task

Text classification is a common natural language processing (NLP) task where the goal is to assign pre-defined categories or labels to the given texts. I collected Chinese news headlines and their corresponding categories from the ETtoday News website to use as a dataset for text classification. There are three main reasons why I chose this dataset. Firstly, due to its complex syntactic structure, Chinese is relatively more difficult to process in NLP tasks compared to English. Moreover, Taiwanese news headlines often employ exaggerated and sensational methods to attract people's attention, which I think it could make the classification task more challenging. Furthermore, the ETtoday News website has a simple structure and clear categorization, making web scraping relatively straightforward.

2. Dataset Description

I use Python with the BeautifulSoup library for web crawling. Each entry in my dataset contains an ID, the headline as a string, and the corresponding class as a string. I collected 50,000 news headlines from the entire year of 2023, spanning five categories in which I am interested: Politics (政治), Sports (體育), Finance (財經), Games (遊戲), and Entertainment (影劇). The distribution and the total number of entries for each class are shown in Table 1.

Class \ Number	Counts	Percentage
Politics	13336	26.67%
Sports	10114	20.23%
Finance	10634	21.27%
Games	1855	3.71%
Entertainment	14061	28.12%
Total	50000	100 %

Table 1. The numbers of data of each class in dataset.

3. Algorithm

I choose training Multinomial Naïve Bayes classifiers and fine-tuning pretrain BERT model as two of my supervised learning methods, and use K-means clustering as my unsupervised learning method.

3.1 Multinomial Naive Bayes

Multinomial Naïve Bayes is a supervised learning method based on Bayes' Theorem, suitable for classification tasks. The classification process involves three main steps: preprocessing the text string into tokens, converting the tokens into vectors, and training the Bayes Classifier with these vectors before evaluating its performance.

I use the Jieba library to tokenize the texts, TfidfVectorizer in scikit-learn for feature extraction, and MultinomialNB in scikit-learn as the model architecture. The designed experiment comprises the following aspects:

Number of Training Data

The number of training data points can affect the performance of Bayes classifiers. I have conducted experiments with various amounts of training data to evaluate the classifiers' performance.

Dimension of Features

Converting text into vectors serves to extract features from the text; altering the dimensionality of these features can impact the model's performance.

Class Weighting

As showing in Table 1, my dataset is not a balanced dataset, the Sports class is much less than other classes, I had added sample weights to the model and evaluate the change of performance of classifiers.

Data Augmentation

I utilize random deletion, a common data augmentation technique that entails randomly removing words from sentences. This approach encourages the model to concentrate on fewer data points and better understand the context.

Evaluation Metrics

I employ K-fold cross-validation which split the dataset into five subsets and do five iterations, in each iteration, one of the subsets is use for evaluation and other for training. I use Precision, Recall, F1-score, and Accuracy to evaluate the model.

3.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a popular architecture based on the encoder of the transformer, which is widely used in various types of NLP tasks, including text classification. In this task, I attempt to fine-tune a BERT model that is pre-trained on a Chinese dataset provided by Google on HuggingFace. I use the Transformers library provided by HuggingFace with PyTorch to train the model. The designed experiment comprises the following aspects:

Number of Training Data

BERT is quite different from traditional Bayes classifiers, so I'm also curious about how the number of training data affects the performance of BERT.

Class Weighting

Due to the unbalanced distribution of classes in the dataset, I add class weighting to the cross-entropy loss function while training the model and expect better performance.

Data Augmentation

I also try random deletion when fine-tuning the BERT model, expecting that this data augmentation technique can enhance the model's performance.

Evaluation Metrics

As mentioned in section 3.1, I employ K-fold cross-validation and use Precision, Recall, F1-score, and Accuracy to evaluate the model.

3.3 K-mean Clustering

K-means clustering is an unsupervised learning algorithm that can partition a dataset into K non-overlapping subsets. This algorithm aims to minimize the distance within each cluster, thus making the items in each cluster more similar. I use Jieba to tokenize the text, Word2Vec to extract the features and turn the tokens into vectors, and KMeansClusterer in NLTK as the backbone of K-means clustering. The designed experiment comprises the following aspects:

Number of Data

Although K-means clustering creates clusters based on the distribution of data features, the number of data points can affect the Word2Vec model and thus alter the performance of the results.

Dimension of Features

Word2Vec is quite different from TfidfVectorizer, which I use for Bayes classifiers. I'm also curious about how changes in the dimensionality of vectors affect the performance of K-means.

Evaluation Metrics

I use the Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) to evaluate K-means clustering. ARI ranges from -1 to 1, indicating a match between the clusters and the ground truth. NMI ranges from 0 to 1, measuring the amount of information obtained about one cluster through another.

4. Experimental Results & Analysis

4.1 Multinomial Naive Bayes

Number of Training Data

I trained the model in different amounts of data, with fixing all the other conditions. The results are shown in table below.

Num \ Metrics	Accuracy	Precision	Recall	F1-score
1000	0.758	0.650	0.622	0.618
5000	0.868	0.742	0.721	0.711
10000	0.893	0.919	0.747	0.744
50000	0.925	0.936	0.851	0.878

It is apparent that as the number of training data increases, the classifiers' performance improves. The reason is that more training data allows the classifiers to learn more important features, thereby enhancing the overall performance.

Dimension of Features

The dimension of the features is also highly related to the model's performance. I trained the model with different settings of feature dimensions.

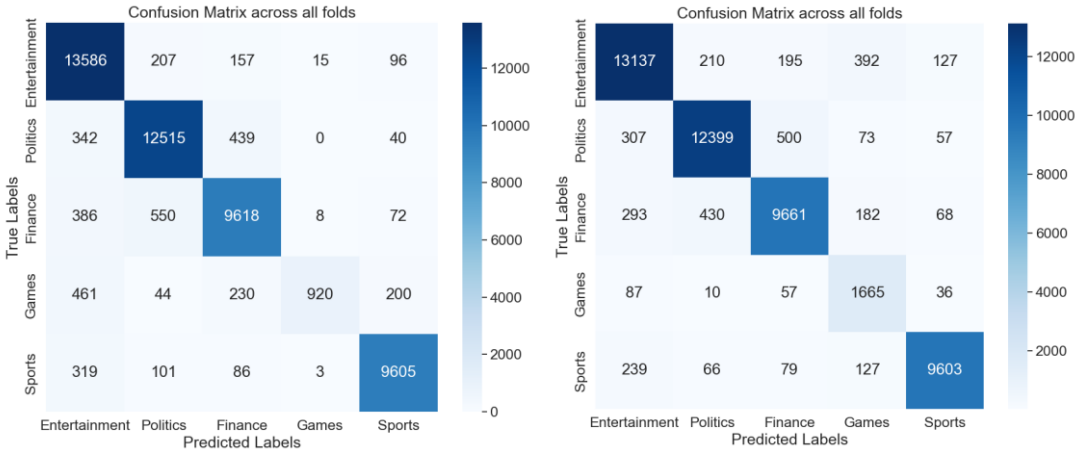
Dim \ Metrics	Accuracy	Precision	Recall	F1-score
100	0.597	0.616	0.490	0.501
500	0.783	0.791	0.673	0.690
1000	0.841	0.851	0.750	0.774
5000	0.914	0.923	0.843	0.868
10000	0.925	0.936	0.851	0.878
50000	0.918	0.935	0.788	0.801

According to the table, Bayes classifiers perform poorly when the feature dimension is small, but the classifier's performance reaches a peak when the feature dimension is increased to 10,000, and adding more dimensions doesn't further enhance the score. I think this is because higher dimensions of features contain more crucial information important to the classifier. However, as the dimension becomes too large, the vector might contain some useless information that could mislead the classifier.

Class Weighting

Adding sample weights to the classifier makes the classes with fewer instances more significant during training, which means that features crucial for the minority class are more likely to be used. The experimental results are shown in the table below.

Type \ Metrics	Accuracy	Precision	Recall	F1-score
Normal (left)	0.925	0.936	0.851	0.878
Sample Weight (right)	0.929	0.891	0.924	0.904



The accuracy has slightly enhanced, while the precision scores have gone down and both recall and F1-score have increased. I think this is because adding sample weighting to Bayes classifiers can improve their ability to classify the 'Sports' class, which was originally prone to misclassification into other classes.

Data Augmentation

Performing random deletion on the text is expected to allow the classifier to extract some unusual features, thus enhancing the model's performance.

Deletion Rate \ Metrics	Accuracy	Precision	Recall	F1-score
No Data Augmentation	0.925	0.936	0.851	0.878
0.1	0.925	0.936	0.849	0.876
0.2	0.924	0.936	0.846	0.873

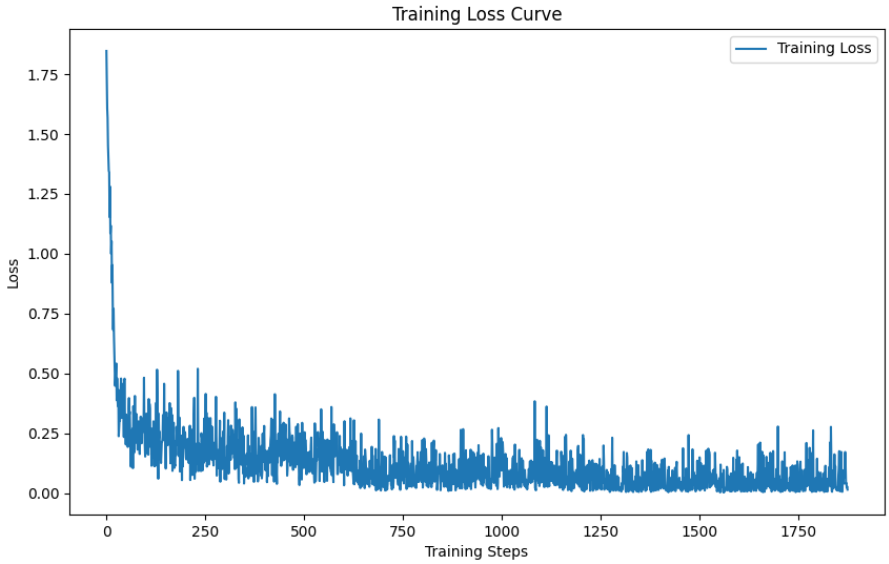
However, as shown in the experimental results, random deletion decreases the performance of Bayes classifiers. I suspect this is because the news headlines are too short and sometimes lack significant features. Random deletion can weaken these important features, thus lowering the overall score.

4.2 BERT

Training Curve

I first trained BERT with all 50,000 data points using the AdamW optimizer, which is commonly used for transformer architectures. The hyperparameters are as follows:

Batch size = 64, Learning rate = 0.00005, Epochs = 3.



As the graph shown above, the model converges significantly after 3 epochs, and the subsequent experiment is trained with the same hyperparameters.

Number of Training Data

I trained the model using different amounts of data, keeping all other conditions constant. The results are shown in the table below.

Num \ Metrics	Accuracy	Precision	Recall	F1-score
1000	0.912	0.896	0.858	0.871
5000	0.937	0.922	0.921	0.920
10000	0.947	0.940	0.925	0.931
50000	0.966	0.954	0.959	0.956

Similar to the experiment done with Bayes classifiers, training BERT with more data makes the model more powerful. A slight difference is that tuning the BERT model with a small amount of data can still achieve good performance. I think this is because BERT has already been well pre-trained and can easily capture the important features embedded in the texts.

Class Weighting

Adding class weighting to the loss function of BERT means that misclassification of the minority class will incur a larger loss, potentially leading the model to pay more attention to this minority class. The experimental results are as follows:

Type \ Metrics	Accuracy	Precision	Recall	F1-score
Normal	0.966	0.954	0.959	0.956
Class weight	0.964	0.947	0.959	0.952

The model did not show improvement after adding class weighting to the loss function. I suspect the reason is that the BERT model is already adept at capturing most of the important features of the minority class without class weighting.

Data Augmentation

Similar to what I mentioned in the Bayes classifier section, adding random deletion to the training dataset is expected to give the model a chance to catch some critical features that tend to be ignored. The experimental results are shown in the table below.

Deletion Rate \ Metrics	Accuracy	Precision	Recall	F1-score
No Data Augmentation	0.966	0.954	0.959	0.956
0.1	0.965	0.955	0.954	0.954
0.2	0.962	0.952	0.950	0.950

According to the results, adding data augmentation does not improve the model's performance. I believe the reason is similar to what I mentioned in the Bayes classifier section: the news headlines are too short and sometimes lack significant features. Additionally, BERT is proficient enough at capturing most of the crucial features, which means there is no need for data augmentation.

4.3 K-means Clustering

Number of Data

I do K-means clustering with different numbers of data. The experimental results are as follows:

Num \ Metrics	Adjusted Rand Index	Normalized Mutual Information
1000	0.011	0.017
5000	0.068	0.076
10000	0.116	0.144
50000	0.363	0.363

As the amount of data increases, K-means becomes better at separating the data. I think this is because having more data enhances Word2Vec's ability to extract distinct features between different classes.

Dimension of Features

The dimension of the features is also highly related to the model's performance. I set different dimensions for the features that Word2Vec outputs, and the experimental results are as follows:

Dim \ Metrics	Adjusted Rand Index	Normalized Mutual Information
5	0.323	0.332
10	0.363	0.363
50	0.352	0.356
100	0.330	0.325

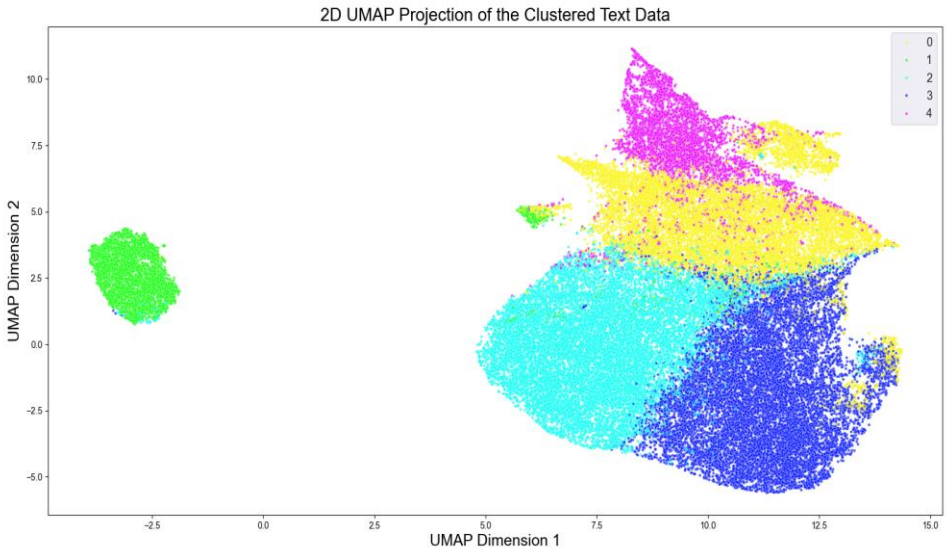
As the dimension of the vectors increases, the performance of K-means clustering doesn't change significantly, indicating that the clustering is determined based on a few crucial features. I also recorded the class distribution of each cluster with 50,000 data points and a feature dimension of 10 to analyze how the data is grouped within each cluster.

Cluster \ Class	Politics	Sports	Finance	Games	Entertainment
Cluster 0	1441	6250	3443	254	366
Cluster 1	288	178	4376	30	74
Cluster 2	180	153	157	1312	2327
Cluster 3	10170	2117	1132	68	597
Cluster 4	1257	1416	1526	191	10697

According to the number of instances for each class in each cluster, cluster 0 can be identified as Sports, cluster 1 as Finance, cluster 2 as Sports, cluster 3 as Politics, and cluster 4 as Entertainment. I then calculated the accuracy and other metrics as follows:

Accuracy	Precision	Recall	F1-score
0.659	0.607	0.655	0.594

Though K-means clustering does not understand the relationship between features and classes, it can still separate the data among the classes based on the features. However, the score is quite low compared to other supervised learning methods. This makes sense because K-means learns without ground truth.



I used UMAP library to map the vectors to a 2D space and plotted the data on the graph to visualize the distribution and clustering of the data points. The graph is shown above.

5. Discussion

After conducting quite a lot of experiments, most of the results were as I expected. However, I did not anticipate that adding random deletion would not enhance the model; I've used similar tricks like random cropping in image classification, which really enhanced the model. Also, the K-means clustering scores really amazed me because I originally thought this method could not effectively separate each class, as it didn't learn the relationship between features and classes.

Throughout this project, I learned that many factors affect performance, from the dataset aspect, sometimes adding class weighting techniques helps the model to classify better. Additionally, the number of data provided and the feature dimension are crucial aspects of each method.

If I had more time for further experiments, I might try other data augmentation methods and see if some can enhance the performance of the news headlines classification tasks. Also, I would like to try other non-deep learning supervised methods, such as Support Vector Machine (SVM), which is very commonly used in classification tasks.

After completing this project, I have a better understanding of how to design and evaluate experiments in machine learning tasks, which has made me more curious about how other machine learning domains, such as computer vision, operate in comparison to what I have done in this NLP task.

6. References

[1] Jeiba-tw, https://github.com/APCLab/jieba-tw

[2] scikit-learn, https://scikit-learn.org/stable/

[3] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2018.

[4] NLTK, https://www.nltk.org/howto/gensim.html

[5] bert-base-chinese, https://huggingface.co/google-bert/bert-base-chinese

[6] Transformers, https://huggingface.co/docs/transformers/index

[7] Beautiful Soup, https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[8] ETtoday News, https://www.ettoday.net/?from=logo

Appendix

Code of Web Crawler

I use Python for web crawling, and the code consists of two main parts. First, we iteratively access the website to collect all the news headlines from a specified day and store all the necessary information. Second, we dump all this data into a JSON file, completing the dataset.

1. Import the library which is needed.

```
import requests
import json
from tqdm import tqdm
from bs4 import BeautifulSoup
from datetime import datetime
```

2. Iteratively Access the web pages and store the information.

```
# Function to check if a date is the same as a specific date
def is_on_specific_date(date_str, specific_date_str):
    # Parse the date from the string
    date = datetime.strptime(date_str, "%Y/%m/%d %H:%M")
    # Parse the specific date to compare
    specific_date = datetime.strptime(specific_date_str, "%Y/%m/%d")
    # Check if the date from the string is the same as the specific date
    return date.date() == specific_date.date()
"""
政治、體育、財經、遊戲、影劇
  1    10   17    24    9
"""
# List of news types
typelist = [1,10,17,24,9]

# Lists to store the data
titles = []
classtypes = []

# Accessing all news headlines and urls from 2023/1/1 to 2023/12/31
print("Accessing all news headlines and urls from 2023/1/1 to 2023/12/31...")
for year in tqdm(range(2023,2024)):
```

```
    for month in tqdm(range(1,13)):
        for day in tqdm(range(1,32)):
            for type in typelist:
                u = 'https://www.ettoday.net/news/news-list-'+str(year) \
                    +'-'+str(month)+'-'+str(day)+'-'+str(type)+'.htm'
                try:
                    res = requests.get(u)
                    soup = BeautifulSoup(res.content, "lxml")
                    soup = soup.find("div", class_="part_list_2")
                    for a in soup.find_all("h3"):
                        #print(a.span.string)
                        if(is_on_specific_date(a.span.string, str(year)+'/'
                        +str(month)+'/'+str(day)) and a.a.string != None):
                            titles.append(a.a.string)
                            classtypes.append(a.em.string)
                except:
                    continue
```

3. Finally, Dump the data into json file.

```
# Dump the data into a json file
js_dataset = []
with open('dataset.json', 'w', encoding='UTF-8') as dataset:
    print("arrange the data and dump into json file")
    for index, (title, classtype) in tqdm(enumerate(zip(titles, classtypes))):
        if index >= 50000: # Limit the number of data samples
            break
        js_dataset.append({"id" : index, "title" : title, "class" : classtype})
    print("dump to train.json")
    for data in js_dataset:
        dataset.write(json.dumps(data , ensure_ascii=False) + "\n")
print("Finish")
```

4. The data in the json file be like:
   {"id": 0, "title": "中央不普發現金　陳致中喊話「體察基層感受」：這是利大於弊", "class": "政治"}

Code of Multinomial Naive Bayes

I use python to write the code. To train a Bayes classifier, we first need to fine-tune the feature extractor using the dataset. Then, we convert the text in the dataset into vectors and use these vectors to train the classifier.

1. Import the library

```
import json
import jieba
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import KFold
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.utils.class_weight import compute_sample_weight
import numpy as np
import random
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

2. Set up the configuration and load data from dataset.

```
# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)

# Configuration
reduce_stopwords = True
deletion_rate = 0.1
num_data = 50000
feature_dim = 10000
K = 5
do_weighted_sampling = False
do_random_deletion = False

# Load data
with open('dataset.json', 'r', encoding='utf-8') as file:
    data = [json.loads(line) for line in file]
data = random.sample(data, num_data)  # Randomly sampling data points
```

```
with open('stopwords.txt', 'r', encoding='utf-8') as file:
    stopwords = {line.strip() for line in file}
```

3. Preprocess the data, and define the random deletion function which will be used later.

```
# Preprocessing
titles = [d['title'] for d in data]
classes = [d['class'] for d in data]
if reduce_stopwords:
    titles = [" ".join([word for word in jieba.cut(title)
                        if word not in stopwords]) for title in titles]
else:
    titles = [" ".join(jieba.cut(title)) for title in titles]

# Function for random deletion
def random_deletion(texts, deletion_rate=0.2):
    result = []
    for text in texts:
        words = text.split()
        words = [word for word in words if word not in stopwords]
        num_to_delete = max(1, int(len(words) * deletion_rate))
        for _ in range(num_to_delete):
            if words:
                words.pop(random.randint(0, len(words) - 1))
        result.append(' '.join(words))
    return result

# Mapping classes to indices
class_list = sorted(set(classes))
class_to_index = {cls: index for index, cls in enumerate(class_list)}
indices = [class_to_index[cls] for cls in classes]
# Vectorization outside the loop to get vocabulary
vectorizer = TfidfVectorizer(max_features=feature_dim,token_pattern=r'\b\w+\b')
vectorizer.fit(titles)  # Fit to obtain the feature set but don't transform here
```

4. Do K-Fold cross-validation which separate the dataset into 5 parts. Iterate 5 times.

```
# K-Fold cross-validation
kf = KFold(n_splits=K, shuffle=True, random_state=42)
```

```
metrics = {'precision': [], 'recall': [], 'f1': [], 'accuracy': []}


# Initialize lists to store all true labels and predictions across folds
all_true_labels = []
all_pred_labels = []

# Split data and perform cross-validation
for train_index, test_index in kf.split(titles):
    X_train_raw = [titles[i] for i in train_index]
    y_train = [indices[i] for i in train_index]
    X_test_raw = [titles[i] for i in test_index]
    y_test = [indices[i] for i in test_index]

    # Apply data augmentation on the training data only
    if do_random_deletion:
        X_train_augmented = random_deletion(X_train_raw,
deletion_rate=deletion_rate)
    else:
        X_train_augmented = X_train_raw

    # Transform text data to feature vectors
    X_train = vectorizer.transform(X_train_augmented)
    X_test = vectorizer.transform(X_test_raw)

    # Weighted sampling if needed
    if do_weighted_sampling:
        sample_weights = compute_sample_weight('balanced', y_train)
        model = MultinomialNB().fit(X_train, y_train, sample_weight=sample_weights)
    else:
        model = MultinomialNB().fit(X_train, y_train)
    # Evaluation
    y_pred = model.predict(X_test)
    metrics['precision'].append(precision_score(y_test, y_pred, average='macro'))
    metrics['recall'].append(recall_score(y_test, y_pred, average='macro'))
    metrics['f1'].append(f1_score(y_test, y_pred, average='macro'))
    metrics['accuracy'].append(accuracy_score(y_test, y_pred))

    # Assuming y_test is a list of true labels for the current fold
```

```
    all_true_labels.extend(y_test)
    # Extend the overall prediction list with the predictions for the current fold
    all_pred_labels.extend(y_pred)
```

5. Calculate the scores of metrics and print them out. Also plot the graph if needed.

```
# Calculate and print average scores after cross-validation
print('Cross-Validation Metrics:')
for metric in metrics:
    average_score = np.mean(metrics[metric])
    print(f'{metric.capitalize()}: {average_score:.3f}')

# After the cross-validation loop, compute the confusion matrix for all folds
cm = confusion_matrix(all_true_labels, all_pred_labels,
                      labels=range(len(class_list)))
class_mapping = {
    '影劇': 'Entertainment',
    '政治': 'Politics',
    '財經': 'Finance',
    '遊戲': 'Games',
    '體育': 'Sports'
}
class_list = ['影劇', '政治', '財經', '遊戲', '體育']
english_class_list = [class_mapping[cls] for cls in class_list]
# Plotting the confusion matrix
sns.set(font_scale=1.5)  # Increase the font scale factor to make fonts larger
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=english_class_list, yticklabels=english_class_list)
plt.title('Confusion Matrix across all folds')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Code of Fine-tuning BERT

1. Import the library

```
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import json
import jieba
import numpy as np
from tqdm import tqdm
import random
import torch.nn.functional as F
from sklearn.utils.class_weight import compute_class_weight
import matplotlib.pyplot as plt
```

2. Set up the configuration

```
# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)

# Configuration
do_data_augmentation = False  # Set to True to perform data augmentation
deletion_rate = 0.2  # Deletion rate for random deletion
do_sample_weighting = True  # Set to True to perform sample weighting
num_data = 5000  # Number of data samples to use

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

3. Load data from dataset. Define random deletion function which will be used later. Preprocess the data with Dataset class.

```
# Load data
with open('dataset.json', 'r', encoding='utf-8') as file:
    data = [json.loads(line) for line in file.readlines()]
```

```
data = random.sample(data, num_data)  # Randomly sampling data points

# Function for random deletion
def random_deletion(texts, deletion_rate=0.2):
    augmented_texts = []
    for text in texts:
        words = list(jieba.cut(text))
        num_to_delete = max(1, int(len(words) * deletion_rate))
        for _ in range(num_to_delete):
            if len(words) > 1:  # Prevent empty text
                words.pop(random.randint(0, len(words) - 1))
        augmented_texts.append("".join(words))
    return augmented_texts

# Prepare data
titles = [item['title'] for item in data]
class_list = sorted(set(item['class'] for item in data))
class_to_index = {cls: index for index, cls in enumerate(class_list)}
labels = [class_to_index[item['class']] for item in data]

# Assuming class_list and labels are already defined
unique_classes = np.unique(np.array(labels))

class_weights = compute_class_weight(class_weight='balanced',
classes=unique_classes, y=labels)
print(f"Class weights: {class_weights}")
class_weights_tensor = torch.tensor(class_weights, dtype=torch.float).to(device)
# Tokenization
tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')

# Dataset class
class NewsDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
```

```
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long)
        return item

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

4. Train the BERT model with K-fold validation. Then calculate the scores of metrics and print them out.

```
# K-Fold cross-validation setup
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_results = []

# K-Fold Training and Evaluation
for fold, (train_ids, val_ids) in enumerate(kf.split(titles)):
    print(f"Folding {fold+1}/{kf.get_n_splits()}")

    # Data augmentation should be applied here
    train_titles = [titles[i] for i in train_ids]
    if do_data_augmentation:
        train_titles = random_deletion(train_titles, deletion_rate)

    # Tokenizing inside the loop to reflect updated training data
    train_encodings = tokenizer(train_titles, padding=True, truncation=True,
                                max_length=128, return_tensors='pt')
    val_encodings = tokenizer([titles[i] for i in val_ids], padding=True,
                        truncation=True, max_length=128, return_tensors='pt')

    train_labels = [labels[i] for i in train_ids]
    val_labels = [labels[i] for i in val_ids]

    # Create PyTorch datasets and dataloaders
    train_dataset = NewsDataset(train_encodings, train_labels)
    val_dataset = NewsDataset(val_encodings, val_labels)

    train_dataloader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    val_dataloader = DataLoader(val_dataset, batch_size=64)

    # Load pre-trained model for each fold to avoid information leak
```

```
    model = BertForSequenceClassification.from_pretrained('bert-base-chinese',
                                                          num_labels=len(class_list))
    model.to(device)

    optimizer = AdamW(model.parameters(), lr=2e-5)

    # Training loop
    model.train()
    for epoch in range(3):  # Adjust epochs as necessary
        for batch in tqdm(train_dataloader, desc=f"Training Epoch {epoch+1}, Fold
                                                                  {fold+1}"):
            batch = {k: v.to(device) for k, v in batch.items()}
            optimizer.zero_grad()
            outputs = model(**batch)
            if do_sample_weighting:
                logits = outputs.logits
                loss = F.cross_entropy(logits, batch['labels'],
                    weight=class_weights_tensor)  # Apply class weights here
            else:
                loss = outputs.loss
            loss.backward()
            optimizer.step()
    # Evaluation loop
    model.eval()
    val_preds, val_labels_arr = [], []
    for batch in tqdm(val_dataloader, desc=f"Evaluating Fold {fold+1}"):
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)
            logits = outputs.logits
            preds = torch.argmax(logits, dim=-1).cpu().numpy()
            val_preds.extend(preds)
            val_labels_arr.extend(batch['labels'].cpu().numpy())
    # Calculate metrics
    accuracy = accuracy_score(val_labels_arr, val_preds)
    precision, recall, f1, _ = precision_recall_fscore_support(val_labels_arr,
                                                       val_preds, average='macro')
    fold_results.append((accuracy, precision, recall, f1))
```

```
# Compute and print average metrics over all folds
average_results = np.mean(fold_results, axis=0)
print(f"\nAverage Accuracy: {average_results[0]:.3f}")
print(f"Average Precision: {average_results[1]:.3f}")
print(f"Average Recall: {average_results[2]:.3f}")
print(f"Average F1-score: {average_results[3]:.3f}")
```

Code of K-means Clustering

1. Import the library

```
import json
import jieba
from gensim.models import Word2Vec
from nltk.cluster import KMeansClusterer
import nltk
import numpy as np
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
from umap import UMAP
import random
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Set up the configuration.

```
# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)

# Configuration
vector_size = 10  # Dimensionality of the word vectors
num_data = 50000  # Number of data samples to use
```

3. Load the data from dataset

```
# Load and preprocess the dataset
with open('dataset.json', 'r', encoding='utf-8') as file:
    data = [json.loads(line) for line in file]
data = random.sample(data, num_data)  # Randomly sampling data points

texts = [item['title'] for item in data]
true_labels = [item['class'] for item in data]  # Extract actual class labels for
evaluation
tokenized_texts = [list(jieba.cut(text)) for text in texts]
```

4. Tune the Word2Vec model and then use the model to extract the feature from the texts

```
# Train a Word2Vec model
model = Word2Vec(sentences=tokenized_texts, vector_size=vector_size, window=5,
min_count=1, workers=8)

# Vectorize the text
def vectorize(text, model):
    vector = np.zeros(model.vector_size)
    count = 0
    for word in text:
        if word in model.wv:
            vector += model.wv[word]
            count += 1
    return vector / count if count > 0 else vector

vectors = np.array([vectorize(text, model) for text in tokenized_texts])
```

5. Do K-mean Clustering

```
# Clustering with NLTK KMeansClusterer
NUM_CLUSTERS = 5
kclusterer = KMeansClusterer(NUM_CLUSTERS,
     distance=nltk.cluster.util.cosine_distance, repeats=1)
assigned_clusters = kclusterer.cluster(vectors, assign_clusters=True)
```

6. Evaluate the clustering

```
# Evaluate clustering
ari = adjusted_rand_score(true_labels, assigned_clusters)
nmi = normalized_mutual_info_score(true_labels, assigned_clusters)
print(f'Adjusted Rand Index: {ari:.3f}')
print(f'Normalized Mutual Information: {nmi:.3f}')

# Output the clustering result
for i in range(NUM_CLUSTERS):
    print(f"\nCluster {i}:")
    a, b, c, d, e = 0, 0, 0, 0, 0
    for j, label in enumerate(assigned_clusters):
        if label == i:
```

```
            if(true_labels[j] == "政治"):
                a += 1
            elif(true_labels[j] == "體育"):
                b += 1
            elif(true_labels[j] == "財經"):
                c += 1
            elif(true_labels[j] == "遊戲"):
                d += 1
            elif(true_labels[j] == "影劇"):
                e += 1
    print(f"政治: {a}, 體育: {b}, 財經: {c}, 遊戲: {d}, 影劇: {e}")
```

7. Use UMAP library to project the data features to 2-D space, then plot the distribution graph.

```
# UMAP for dimensionality reduction to visualize the clusters
umap_model = UMAP(n_neighbors=15, min_dist=0.1, n_components=2, random_state=42)
reduced_vectors = umap_model.fit_transform(vectors)

# Plotting the reduced vectors colored by assigned clusters
plt.figure(figsize=(12, 10))
plt.title("2D UMAP Projection of the Clustered Text Data", fontsize=20)
# sns font
sns.set(font_scale=1.2)
sns.scatterplot(
    x=reduced_vectors[:, 0], y=reduced_vectors[:, 1],
    hue=assigned_clusters, palette=sns.color_palette("hsv", NUM_CLUSTERS),
    legend="full", alpha=0.7,
    s=10,
)
plt.xlabel('UMAP Dimension 1', fontsize=18)
plt.ylabel('UMAP Dimension 2', fontsize=18)
plt.show()
```