Introduction

This notebook contains code for a homework assignment on text classification of news articles into 5 categories. It explores different feature engineering methods, trains neural network models using 5-fold cross validation, and evaluates performance. The best model is then used to predict labels on a test set.

1. Imports and Setup

```
!pip install rake-nltk
```

```
Collecting rake-nltk
  Downloading rake_nltk-1.0.6-py3-none-any.whl (9.1 kB)
Requirement already satisfied: nltk<4.0.0,>=3.6.2 in /usr/local/lib/python3.10/dist-packages (from rake-nltk) (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk<4.0.0,>=3.6.2->rake-nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk<4.0.0,>=3.6.2->rake-nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk<4.0.0,>=3.6.2->rake-nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk<4.0.0,>=3.6.2->rake-nltk) (4.66.2)
Installing collected packages: rake-nltk
Successfully installed rake-nltk-1.0.6
```

- Import required libraries
- Mount Google Drive
- download nltk packages

```
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import random
from tqdm import tqdm
import pandas as pd
import string
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import *

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

import matplotlib.pyplot as plt
import gensim.downloader as api
import gensim
```

```
Mounted at /content/drive
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Loading training and test data into panda dataframe 🐼

Part 1 Data Loading and Preprocessing -

Question 1 Preprocess the raw training data. You can use the code from Homework 1. You are required to construct other features, such n-grams or keyword extractions. (15pt)

- Load train and test CSV files
- Explore data
- Define and apply text preprocessing function

```
train_data_path = '/content/drive/MyDrive/Colab Notebooks/Homework2/24_train_1.csv'
test_data_path = '/content/drive/MyDrive/Colab Notebooks/Homework2/news-test.csv'
train_data = pd.read_csv(train_data_path)
test_data = pd.read_csv(test_data_path)
```

```
print(train_data)
print(train_data.info())
print(train_data['Category'])
```

```
     ArticleId                                               Text  \
0         1429  sfa awaits report over mikoliunas the scottish...
1         1896  parmalat to return to stockmarket parmalat  th...
2         1633  edu blasts arsenal arsenal s brazilian midfiel...
3         2178  henman decides to quit davis cup tim henman ha...
4          194  french suitor holds lse meeting european stock...
..         ...                                                ...
995       1250  blair  damaged  by blunkett row a majority of ...
996       1639  a november to remember last saturday  one news...
997        916  highbury tunnel players in clear the football ...
998       2217  top stars join us tsunami tv show brad pitt  r...
999        902  eastwood s baby scoops top oscars clint eastwo...

          Category
0            sport
1         business
2            sport
3            sport
4         business
..             ...
995       politics
996          sport
997          sport
998  entertainment
999  entertainment

[1000 rows x 3 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ArticleId  1000 non-null   int64 
 1   Text       1000 non-null   object
 2   Category   1000 non-null   object
dtypes: int64(1), object(2)
memory usage: 23.6+ KB
None
```

Question 1.a: Run Neural Networks with the 2-hidden layers, each has 128 neurons, extracting features by CountVectorizer() as the original features. Use 5-fold cross-validation to evaluate the performance.

This section loads the raw training and test data from CSV files into Pandas dataframes. It then preprocesses the text by:

- Tokenizing
- Lowercasing
- Removing punctuation
- Removing non-alphabetic tokens
- Removing stopwords
- Stemming

The preprocessed text is added as a new column in the dataframes.

It then extracts CountVectorizer features and runs a 2-layer neural network with 5-fold cross validation.

```
# Preprocessing function
def preprocess_text(text):
    # Tokenization
    tokens = nltk.word_tokenize(text)

    # Lowercasing
    tokens = [w.lower() for w in tokens]

    # Removing punctuation
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]

    # Removing non-alphabetic tokens
    words = [word for word in stripped if word.isalpha()]

    # Removing stopwords
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]

    # Stemming
    porter = PorterStemmer()
    stemmed = [porter.stem(word) for word in words]

    return ' '.join(stemmed)

# Apply preprocessing to training and test data
train_data['Processed_Text'] = train_data['Text'].apply(preprocess_text)
test_data['Processed_Text'] = test_data['Text'].apply(preprocess_text)
```

Feature Engineering

Question 1.b: Feature exploration. Use other features like TFIDF, or any word embeddings provided by other packages like GloVe with gensim, or BERT. Use 5-fold cross-validation to evaluate the performance of your Neural Network.

- This section generates various features from the preprocessed text:
- N-grams (bigrams)
- TF-IDF vectors
- GloVe word embeddings (averaging word vectors for each document)
- Word2Vec embeddings (training a Word2Vec model on the corpus and averaging word vectors for each document)

```
# Create n-grams
def generate_ngrams(text, n_gram=1):
    token = [token for token in text.split(" ") if token != "" if token not in stopwords.words('english')]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [" ".join(ngram) for ngram in ngrams]

# Generate n-grams for training data
train_data['Processed_Text_ngrams'] = train_data['Processed_Text'].apply(lambda x: generate_ngrams(x, n_gram=2))

# Generate n-grams for test data
test_data['Processed_Text_ngrams'] = test_data['Processed_Text'].apply(lambda x: generate_ngrams(x, n_gram=2))



# Load GloVe embeddings
glove_vectors = api.load("glove-twitter-100")
glove_embeddings = []

for doc in train_data['Processed_Text_ngrams']:
    doc_embedding = []
    for word in doc:
        if word in glove_vectors:
            doc_embedding.append(glove_vectors[word])
    if doc_embedding:
        doc_embedding = np.mean(doc_embedding, axis=0)
    else:
        doc_embedding = np.zeros(100)
    glove_embeddings.append(doc_embedding)

glove_embeddings = np.array(glove_embeddings)

# Create TF-IDF features
tfidf = TfidfVectorizer(ngram_range=(1,2))
tfidf_train = tfidf.fit_transform(train_data['Processed_Text_ngrams'].apply(lambda x: ' '.join(x)))
tfidf_test = tfidf.transform(test_data['Processed_Text_ngrams'].apply(lambda x: ' '.join(x)))

# Train Word2Vec model
w2v_model = gensim.models.Word2Vec(train_data['Processed_Text_ngrams'], vector_size=100, window=5, min_count=1, workers=4)
w2v_embeddings = []
for doc in train_data['Processed_Text_ngrams']:
    doc_embedding = []
    for word in doc:
        if word in w2v_model.wv:
            doc_embedding.append(w2v_model.wv[word])
    if doc_embedding:
        doc_embedding = np.mean(doc_embedding, axis=0)
    else:
        doc_embedding = np.zeros(100)
    w2v_embeddings.append(doc_embedding)

w2v_embeddings = np.array(w2v_embeddings)
```

```
[==================================================] 100.0% 387.1/387.1MB downloaded
```

Question 1.c. Describe how you generate features. (5pt)

The features are generated as follows:

- CountVectorizer: Counts the frequency of each word in each document
- N-grams: Generates bigrams from the preprocessed text
- TF-IDF: Computes TF-IDF weights for each word in each document
- GloVe: Averages the pre-trained GloVe embeddings for each word in a document to get a document embedding Word2Vec: Trains a Word2Vec model on the corpus and averages the learned word embeddings for each document

Model Definition

This section defines the architecture of the neural network model (NewsClassifier) using PyTorch. The model has an embedding layer, two hidden layers with ReLU activation, and a softmax output layer. The forward pass of the model is defined in the forward() method.

```
# Define neural network model
class NewsClassifier(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, output_dim):
        super(NewsClassifier, self).__init__()
        self.fc1 = nn.Linear(embedding_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, output_dim)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.relu(out)
        out = self.fc3(out)
        return out
```

```
# Define training function
def train_model(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    running_corrects = 0
    for inputs, labels in train_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        _, preds = torch.max(outputs, 1)
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)
        loss.backward()
        optimizer.step()
    epoch_loss = running_loss / len(train_loader.dataset)
    epoch_acc = running_corrects.double() / len(train_loader.dataset)
    print(f"Training Loss: {epoch_loss:.4f}, Training Accuracy: {epoch_acc:.4f}")
    return epoch_loss, epoch_acc

# Define evaluation function
def eval_model(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    running_corrects = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)
    epoch_loss = running_loss / len(test_loader.dataset)
    epoch_acc = running_corrects.double() / len(test_loader.dataset)
    print(f"Validation Loss: {epoch_loss:.4f}, Validation Accuracy: {epoch_acc:.4f}")
    return epoch_loss, epoch_acc


# Define dataset class
class NewsDataset(Dataset):
    def __init__(self, features, labels):
        self.features = torch.tensor(features, dtype=torch.float32)  # Convert features to float32
        self.labels = torch.tensor(labels, dtype=torch.long)  # Labels should be of type long for classification

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

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]


# Define function to run 5-fold cross-validation
def run_cross_validation(model, features, labels, batch_size, num_epochs, device):
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    train_accuracies = []
    val_accuracies = []

    for fold, (train_idx, val_idx) in enumerate(kfold.split(features)):
        print(f"Fold {fold+1}")
        train_dataset = NewsDataset(features[train_idx], labels[train_idx])
        val_dataset = NewsDataset(features[val_idx], labels[val_idx])

        train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
        val_loader = DataLoader(val_dataset, batch_size=batch_size)

        model = NewsClassifier(features.shape[1], 128, len(np.unique(labels))).to(device)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters())

        for epoch in range(num_epochs):
            train_loss, train_acc = train_model(model, train_loader, criterion, optimizer, device)
            val_loss, val_acc = eval_model(model, val_loader, criterion, device)
            print(f"Epoch {epoch+1}/{num_epochs} - Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

        train_accuracies.append(train_acc.cpu().numpy())
        val_accuracies.append(val_acc.cpu().numpy())

    return np.mean(train_accuracies), np.std(train_accuracies), np.mean(val_accuracies), np.std(val_accuracies)

# Convert labels to numerical values
label_map = {'sport': 0, 'business': 1, 'politics': 2, 'entertainment': 3, 'tech': 4}
train_labels = train_data['Category'].map(label_map).values
#test_labels = test_data['Category'].map(label_map).values
```

Model Training with Cross Validation

- This section defines functions for training (train_model) and evaluating (eval_model) the neural network model.
- It also defines a custom Dataset class (NewsDataset) for loading the features and labels. The run_cross_validation function performs 5-fold cross validation with a given feature set.
- It trains the model on the training set and evaluates on the validation set for each fold.
- It returns the average training and validation accuracies and their standard deviations.
- Cross validation is run with CountVectorizer, TF-IDF, GloVe, and Word2Vec features. The results are printed in a table and visualized in a bar chart.

Double-click (or enter) to edit

```python
# Run cross-validation with CountVectorizer features
print("Running cross-validation with CountVectorizer features...")
vectorizer = CountVectorizer()
count_train = vectorizer.fit_transform(train_data['Processed_Text']).toarray()
count_test = vectorizer.transform(test_data['Processed_Text']).toarray()
count_train_acc, count_train_std, count_val_acc, count_val_std = run_cross_validation(NewsClassifier, count_train, train_labels, batch_size=32, num_epochs=10, device=torch.device('cuda' if torch.cuda.is_available() else 'cpu'))
```

```
Running cross-validation with CountVectorizer features...
Fold 1
Epoch 1/10 - Train Loss: 0.8842, Train Acc: 0.8450, Val Loss: 0.2029, Val Acc: 0.9700
Epoch 2/10 - Train Loss: 0.0244, Train Acc: 0.9988, Val Loss: 0.1688, Val Acc: 0.9750
Epoch 3/10 - Train Loss: 0.0022, Train Acc: 1.0000, Val Loss: 0.2123, Val Acc: 0.9700
Epoch 4/10 - Train Loss: 0.0009, Train Acc: 1.0000, Val Loss: 0.2133, Val Acc: 0.9750
Epoch 5/10 - Train Loss: 0.0006, Train Acc: 1.0000, Val Loss: 0.2120, Val Acc: 0.9750
Epoch 6/10 - Train Loss: 0.0005, Train Acc: 1.0000, Val Loss: 0.2129, Val Acc: 0.9750
Epoch 7/10 - Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.2169, Val Acc: 0.9750
Epoch 8/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.2201, Val Acc: 0.9750
Epoch 9/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.2238, Val Acc: 0.9750
Epoch 10/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.2257, Val Acc: 0.9750
Fold 2
Epoch 1/10 - Train Loss: 0.9787, Train Acc: 0.7700, Val Loss: 0.2996, Val Acc: 0.9450
Epoch 2/10 - Train Loss: 0.0380, Train Acc: 0.9988, Val Loss: 0.1415, Val Acc: 0.9450
Epoch 3/10 - Train Loss: 0.0019, Train Acc: 1.0000, Val Loss: 0.1347, Val Acc: 0.9350
Epoch 4/10 - Train Loss: 0.0009, Train Acc: 1.0000, Val Loss: 0.1328, Val Acc: 0.9400
Epoch 5/10 - Train Loss: 0.0007, Train Acc: 1.0000, Val Loss: 0.1380, Val Acc: 0.9450
Epoch 6/10 - Train Loss: 0.0005, Train Acc: 1.0000, Val Loss: 0.1410, Val Acc: 0.9450
Epoch 7/10 - Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.1427, Val Acc: 0.9450
Epoch 8/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.1434, Val Acc: 0.9450
Epoch 9/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.1444, Val Acc: 0.9450
Epoch 10/10 - Train Loss: 0.0002, Train Acc: 1.0000, Val Loss: 0.1462, Val Acc: 0.9450
Fold 3
Epoch 1/10 - Train Loss: 0.9597, Train Acc: 0.7125, Val Loss: 0.2459, Val Acc: 0.9900
Epoch 2/10 - Train Loss: 0.0619, Train Acc: 0.9900, Val Loss: 0.1027, Val Acc: 0.9650
Epoch 3/10 - Train Loss: 0.0054, Train Acc: 1.0000, Val Loss: 0.0766, Val Acc: 0.9750
Epoch 4/10 - Train Loss: 0.0014, Train Acc: 1.0000, Val Loss: 0.0655, Val Acc: 0.9750
Epoch 5/10 - Train Loss: 0.0009, Train Acc: 1.0000, Val Loss: 0.0643, Val Acc: 0.9750
Epoch 6/10 - Train Loss: 0.0007, Train Acc: 1.0000, Val Loss: 0.0639, Val Acc: 0.9750
Epoch 7/10 - Train Loss: 0.0006, Train Acc: 1.0000, Val Loss: 0.0633, Val Acc: 0.9750
Epoch 8/10 - Train Loss: 0.0005, Train Acc: 1.0000, Val Loss: 0.0625, Val Acc: 0.9750
Epoch 9/10 - Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.0622, Val Acc: 0.9750
Epoch 10/10 - Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.0619, Val Acc: 0.9750
Fold 4
Epoch 1/10 - Train Loss: 0.9785, Train Acc: 0.8063, Val Loss: 0.2746, Val Acc: 0.9650
Epoch 2/10 - Train Loss: 0.0497, Train Acc: 0.9962, Val Loss: 0.1181, Val Acc: 0.9800
Epoch 3/10 - Train Loss: 0.0027, Train Acc: 1.0000, Val Loss: 0.1112, Val Acc: 0.9700
Epoch 4/10 - Train Loss: 0.0012, Train Acc: 1.0000, Val Loss: 0.1091, Val Acc: 0.9700
Epoch 5/10 - Train Loss: 0.0009, Train Acc: 1.0000, Val Loss: 0.1068, Val Acc: 0.9700
Epoch 6/10 - Train Loss: 0.0007, Train Acc: 1.0000, Val Loss: 0.1065, Val Acc: 0.9700
Epoch 7/10 - Train Loss: 0.0005, Train Acc: 1.0000, Val Loss: 0.1054, Val Acc: 0.9750
Epoch 8/10 - Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.1061, Val Acc: 0.9700
Epoch 9/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.1050, Val Acc: 0.9750
Epoch 10/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.1044, Val Acc: 0.9750
Fold 5
Epoch 1/10 - Train Loss: 0.9308, Train Acc: 0.8050, Val Loss: 0.2184, Val Acc: 0.9650
Epoch 2/10 - Train Loss: 0.0443, Train Acc: 0.9950, Val Loss: 0.0919, Val Acc: 0.9750
Epoch 3/10 - Train Loss: 0.0036, Train Acc: 1.0000, Val Loss: 0.0880, Val Acc: 0.9750
Epoch 4/10 - Train Loss: 0.0014, Train Acc: 1.0000, Val Loss: 0.0900, Val Acc: 0.9700
Epoch 5/10 - Train Loss: 0.0009, Train Acc: 1.0000, Val Loss: 0.0891, Val Acc: 0.9700
Epoch 6/10 - Train Loss: 0.0006, Train Acc: 1.0000, Val Loss: 0.0893, Val Acc: 0.9700
Epoch 7/10 - Train Loss: 0.0005, Train Acc: 1.0000, Val Loss: 0.0899, Val Acc: 0.9700
Epoch 8/10 - Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.0910, Val Acc: 0.9700
Epoch 9/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.0914, Val Acc: 0.9700
Epoch 10/10 - Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.0921, Val Acc: 0.9700
```

```python
# Run cross-validation with TF-IDF features
print("Running cross-validation with TF-IDF features...")
tfidf_train_acc, tfidf_train_std, tfidf_val_acc, tfidf_val_std = run_cross_validation(NewsClassifier, tfidf_train.toarray(), train_labels, batch_size=32, num_epochs=10, device=torch.device('cuda' if torch.cuda.is_available() else 'cpu'))
```

```
Running cross-validation with TF-IDF features...
Fold 1
Epoch 1/10 - Train Loss: 1.5468, Train Acc: 0.6713, Val Loss: 1.3971, Val Acc: 0.9350
Epoch 2/10 - Train Loss: 0.8866, Train Acc: 0.9962, Val Loss: 0.6213, Val Acc: 0.9500
Epoch 3/10 - Train Loss: 0.1070, Train Acc: 1.0000, Val Loss: 0.2384, Val Acc: 0.9800
Epoch 4/10 - Train Loss: 0.0093, Train Acc: 1.0000, Val Loss: 0.1835, Val Acc: 0.9800
Epoch 5/10 - Train Loss: 0.0037, Train Acc: 1.0000, Val Loss: 0.1671, Val Acc: 0.9800
Epoch 6/10 - Train Loss: 0.0024, Train Acc: 1.0000, Val Loss: 0.1572, Val Acc: 0.9850
Epoch 7/10 - Train Loss: 0.0017, Train Acc: 1.0000, Val Loss: 0.1496, Val Acc: 0.9850
Epoch 8/10 - Train Loss: 0.0013, Train Acc: 1.0000, Val Loss: 0.1436, Val Acc: 0.9850
Epoch 9/10 - Train Loss: 0.0010, Train Acc: 1.0000, Val Loss: 0.1384, Val Acc: 0.9850
Epoch 10/10 - Train Loss: 0.0008, Train Acc: 1.0000, Val Loss: 0.1337, Val Acc: 0.9850
Fold 2
Epoch 1/10 - Train Loss: 1.5541, Train Acc: 0.5075, Val Loss: 1.4343, Val Acc: 0.8000
Epoch 2/10 - Train Loss: 0.9694, Train Acc: 0.9950, Val Loss: 0.7341, Val Acc: 0.9150
Epoch 3/10 - Train Loss: 0.1399, Train Acc: 1.0000, Val Loss: 0.2736, Val Acc: 0.9500
Epoch 4/10 - Train Loss: 0.0098, Train Acc: 1.0000, Val Loss: 0.2039, Val Acc: 0.9550
Epoch 5/10 - Train Loss: 0.0037, Train Acc: 1.0000, Val Loss: 0.1889, Val Acc: 0.9500
Epoch 6/10 - Train Loss: 0.0023, Train Acc: 1.0000, Val Loss: 0.1802, Val Acc: 0.9500
Epoch 7/10 - Train Loss: 0.0016, Train Acc: 1.0000, Val Loss: 0.1713, Val Acc: 0.9500
Epoch 8/10 - Train Loss: 0.0011, Train Acc: 1.0000, Val Loss: 0.1669, Val Acc: 0.9450
Epoch 9/10 - Train Loss: 0.0008, Train Acc: 1.0000, Val Loss: 0.1607, Val Acc: 0.9450
Epoch 10/10 - Train Loss: 0.0006, Train Acc: 1.0000, Val Loss: 0.1567, Val Acc: 0.9400
Fold 3
Epoch 1/10 - Train Loss: 1.5430, Train Acc: 0.3912, Val Loss: 1.4178, Val Acc: 0.5800
Epoch 2/10 - Train Loss: 0.9328, Train Acc: 0.9437, Val Loss: 0.6741, Val Acc: 0.9400
Epoch 3/10 - Train Loss: 0.1287, Train Acc: 1.0000, Val Loss: 0.2268, Val Acc: 0.9800
Epoch 4/10 - Train Loss: 0.0094, Train Acc: 1.0000, Val Loss: 0.1656, Val Acc: 0.9750
Epoch 5/10 - Train Loss: 0.0038, Train Acc: 1.0000, Val Loss: 0.1476, Val Acc: 0.9750
Epoch 6/10 - Train Loss: 0.0024, Train Acc: 1.0000, Val Loss: 0.1385, Val Acc: 0.9700
Epoch 7/10 - Train Loss: 0.0015, Train Acc: 1.0000, Val Loss: 0.1298, Val Acc: 0.9700
Epoch 8/10 - Train Loss: 0.0011, Train Acc: 1.0000, Val Loss: 0.1234, Val Acc: 0.9700
Epoch 9/10 - Train Loss: 0.0008, Train Acc: 1.0000, Val Loss: 0.1169, Val Acc: 0.9750
Epoch 10/10 - Train Loss: 0.0005, Train Acc: 1.0000, Val Loss: 0.1109, Val Acc: 0.9750
Fold 4
Epoch 1/10 - Train Loss: 1.5531, Train Acc: 0.4637, Val Loss: 1.4350, Val Acc: 0.8200
Epoch 2/10 - Train Loss: 0.9382, Train Acc: 0.9988, Val Loss: 0.6990, Val Acc: 0.9800
Epoch 3/10 - Train Loss: 0.1359, Train Acc: 1.0000, Val Loss: 0.2459, Val Acc: 0.9850
Epoch 4/10 - Train Loss: 0.0102, Train Acc: 1.0000, Val Loss: 0.1760, Val Acc: 0.9800
Epoch 5/10 - Train Loss: 0.0037, Train Acc: 1.0000, Val Loss: 0.1573, Val Acc: 0.9800
Epoch 6/10 - Train Loss: 0.0023, Train Acc: 1.0000, Val Loss: 0.1467, Val Acc: 0.9800
Epoch 7/10 - Train Loss: 0.0016, Train Acc: 1.0000, Val Loss: 0.1395, Val Acc: 0.9800
Epoch 8/10 - Train Loss: 0.0011, Train Acc: 1.0000, Val Loss: 0.1343, Val Acc: 0.9750
Epoch 9/10 - Train Loss: 0.0009, Train Acc: 1.0000, Val Loss: 0.1300, Val Acc: 0.9750
Epoch 10/10 - Train Loss: 0.0007, Train Acc: 1.0000, Val Loss: 0.1265, Val Acc: 0.9750
Fold 5
Epoch 1/10 - Train Loss: 1.5530, Train Acc: 0.4750, Val Loss: 1.4196, Val Acc: 0.8150
Epoch 2/10 - Train Loss: 0.9436, Train Acc: 0.9938, Val Loss: 0.6329, Val Acc: 0.9700
Epoch 3/10 - Train Loss: 0.1179, Train Acc: 1.0000, Val Loss: 0.2037, Val Acc: 0.9750
Epoch 4/10 - Train Loss: 0.0090, Train Acc: 1.0000, Val Loss: 0.1471, Val Acc: 0.9800
Epoch 5/10 - Train Loss: 0.0035, Train Acc: 1.0000, Val Loss: 0.1329, Val Acc: 0.9800
Epoch 6/10 - Train Loss: 0.0021, Train Acc: 1.0000, Val Loss: 0.1240, Val Acc: 0.9700
Epoch 7/10 - Train Loss: 0.0014, Train Acc: 1.0000, Val Loss: 0.1169, Val Acc: 0.9750
Epoch 8/10 - Train Loss: 0.0009, Train Acc: 1.0000, Val Loss: 0.1108, Val Acc: 0.9750
Epoch 9/10 - Train Loss: 0.0006, Train Acc: 1.0000, Val Loss: 0.1057, Val Acc: 0.9750
Epoch 10/10 - Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.0998, Val Acc: 0.9800
```

```python
# Run cross-validation with GloVe embeddings
print("Running cross-validation with GloVe embeddings...")
glove_train_acc, glove_train_std, glove_val_acc, glove_val_std = run_cross_validation(NewsClassifier, glove_embeddings, train_labels, batch_size=32, num_epochs=10, device=torch.device('cuda' if torch.cuda.is_available() else 'cpu'))
```

```
Running cross-validation with GloVe embeddings...
Fold 1
Epoch 1/10 - Train Loss: 1.6086, Train Acc: 0.1988, Val Loss: 1.6020, Val Acc: 0.2450
Epoch 2/10 - Train Loss: 1.6059, Train Acc: 0.2275, Val Loss: 1.6010, Val Acc: 0.2450
Epoch 3/10 - Train Loss: 1.6056, Train Acc: 0.2275, Val Loss: 1.6011, Val Acc: 0.2450
Epoch 4/10 - Train Loss: 1.6059, Train Acc: 0.2075, Val Loss: 1.6011, Val Acc: 0.2450
Epoch 5/10 - Train Loss: 1.6057, Train Acc: 0.2125, Val Loss: 1.6010, Val Acc: 0.2200
Epoch 6/10 - Train Loss: 1.6053, Train Acc: 0.2200, Val Loss: 1.6009, Val Acc: 0.2450
Epoch 7/10 - Train Loss: 1.6049, Train Acc: 0.2275, Val Loss: 1.6008, Val Acc: 0.2450
Epoch 8/10 - Train Loss: 1.6053, Train Acc: 0.2275, Val Loss: 1.6009, Val Acc: 0.2450
Epoch 9/10 - Train Loss: 1.6053, Train Acc: 0.2275, Val Loss: 1.6007, Val Acc: 0.2450
Epoch 10/10 - Train Loss: 1.6048, Train Acc: 0.2275, Val Loss: 1.6014, Val Acc: 0.2450
Fold 2
Epoch 1/10 - Train Loss: 1.6100, Train Acc: 0.1925, Val Loss: 1.6040, Val Acc: 0.2600
Epoch 2/10 - Train Loss: 1.6048, Train Acc: 0.2238, Val Loss: 1.6053, Val Acc: 0.2600
Epoch 3/10 - Train Loss: 1.6039, Train Acc: 0.2200, Val Loss: 1.6057, Val Acc: 0.1900
Epoch 4/10 - Train Loss: 1.6045, Train Acc: 0.2288, Val Loss: 1.6066, Val Acc: 0.1900
Epoch 5/10 - Train Loss: 1.6044, Train Acc: 0.2288, Val Loss: 1.6059, Val Acc: 0.1900
Epoch 6/10 - Train Loss: 1.6047, Train Acc: 0.2288, Val Loss: 1.6059, Val Acc: 0.1900
Epoch 7/10 - Train Loss: 1.6047, Train Acc: 0.2112, Val Loss: 1.6053, Val Acc: 0.2600
Epoch 8/10 - Train Loss: 1.6044, Train Acc: 0.2162, Val Loss: 1.6056, Val Acc: 0.1900
Epoch 9/10 - Train Loss: 1.6048, Train Acc: 0.2288, Val Loss: 1.6068, Val Acc: 0.1900
Epoch 10/10 - Train Loss: 1.6045, Train Acc: 0.2288, Val Loss: 1.6065, Val Acc: 0.1900
Fold 3
Epoch 1/10 - Train Loss: 1.6064, Train Acc: 0.2087, Val Loss: 1.6102, Val Acc: 0.2100
Epoch 2/10 - Train Loss: 1.6032, Train Acc: 0.2363, Val Loss: 1.6112, Val Acc: 0.2100
Epoch 3/10 - Train Loss: 1.6044, Train Acc: 0.2363, Val Loss: 1.6121, Val Acc: 0.2100
Epoch 4/10 - Train Loss: 1.6026, Train Acc: 0.2363, Val Loss: 1.6104, Val Acc: 0.2100
Epoch 5/10 - Train Loss: 1.6023, Train Acc: 0.2363, Val Loss: 1.6107, Val Acc: 0.2100
Epoch 6/10 - Train Loss: 1.6026, Train Acc: 0.2363, Val Loss: 1.6122, Val Acc: 0.2100
Epoch 7/10 - Train Loss: 1.6023, Train Acc: 0.2363, Val Loss: 1.6118, Val Acc: 0.2100
Epoch 8/10 - Train Loss: 1.6025, Train Acc: 0.2363, Val Loss: 1.6114, Val Acc: 0.2100
Epoch 9/10 - Train Loss: 1.6026, Train Acc: 0.2363, Val Loss: 1.6112, Val Acc: 0.2100
Epoch 10/10 - Train Loss: 1.6023, Train Acc: 0.2363, Val Loss: 1.6116, Val Acc: 0.2100
Fold 4
Epoch 1/10 - Train Loss: 1.6080, Train Acc: 0.2075, Val Loss: 1.6022, Val Acc: 0.1900
Epoch 2/10 - Train Loss: 1.6042, Train Acc: 0.2412, Val Loss: 1.6067, Val Acc: 0.1900
Epoch 3/10 - Train Loss: 1.6040, Train Acc: 0.2412, Val Loss: 1.6086, Val Acc: 0.1900
Epoch 4/10 - Train Loss: 1.6040, Train Acc: 0.2412, Val Loss: 1.6108, Val Acc: 0.1900
Epoch 5/10 - Train Loss: 1.6044, Train Acc: 0.2412, Val Loss: 1.6071, Val Acc: 0.1900
Epoch 6/10 - Train Loss: 1.6047, Train Acc: 0.2412, Val Loss: 1.6095, Val Acc: 0.1900
Epoch 7/10 - Train Loss: 1.6041, Train Acc: 0.2412, Val Loss: 1.6078, Val Acc: 0.1900
Epoch 8/10 - Train Loss: 1.6041, Train Acc: 0.2412, Val Loss: 1.6107, Val Acc: 0.1900
Epoch 9/10 - Train Loss: 1.6037, Train Acc: 0.2412, Val Loss: 1.6076, Val Acc: 0.1900
Epoch 10/10 - Train Loss: 1.6039, Train Acc: 0.2412, Val Loss: 1.6083, Val Acc: 0.1900
Fold 5
Epoch 1/10 - Train Loss: 1.6091, Train Acc: 0.2038, Val Loss: 1.6029, Val Acc: 0.2500
Epoch 2/10 - Train Loss: 1.6058, Train Acc: 0.2150, Val Loss: 1.6034, Val Acc: 0.2500
Epoch 3/10 - Train Loss: 1.6053, Train Acc: 0.2263, Val Loss: 1.6032, Val Acc: 0.2500
Epoch 4/10 - Train Loss: 1.6046, Train Acc: 0.2025, Val Loss: 1.6038, Val Acc: 0.2500
Epoch 5/10 - Train Loss: 1.6060, Train Acc: 0.2263, Val Loss: 1.6037, Val Acc: 0.2500
Epoch 6/10 - Train Loss: 1.6042, Train Acc: 0.2263, Val Loss: 1.6044, Val Acc: 0.2500
Epoch 7/10 - Train Loss: 1.6044, Train Acc: 0.2038, Val Loss: 1.6041, Val Acc: 0.2500
Epoch 8/10 - Train Loss: 1.6046, Train Acc: 0.2125, Val Loss: 1.6043, Val Acc: 0.2100
Epoch 9/10 - Train Loss: 1.6045, Train Acc: 0.2100, Val Loss: 1.6040, Val Acc: 0.2500
Epoch 10/10 - Train Loss: 1.6043, Train Acc: 0.2263, Val Loss: 1.6045, Val Acc: 0.2500
```

```python
# Run cross-validation with Word2Vec embeddings
print("Running cross-validation with Word2Vec embeddings...")
w2v_train_acc, w2v_train_std, w2v_val_acc, w2v_val_std = run_cross_validation(NewsClassifier, w2v_embeddings, train_labels, batch_size=32, num_epochs=10,device=torch.device('cuda' if torch.cuda.is_available() else 'cpu'))
```

```
Running cross-validation with Word2Vec embeddings...
Fold 1
Epoch 1/10 - Train Loss: 1.6072, Train Acc: 0.2275, Val Loss: 1.6020, Val Acc: 0.2450
Epoch 2/10 - Train Loss: 1.6058, Train Acc: 0.2050, Val Loss: 1.6016, Val Acc: 0.2450
Epoch 3/10 - Train Loss: 1.6049, Train Acc: 0.2275, Val Loss: 1.6008, Val Acc: 0.2450
Epoch 4/10 - Train Loss: 1.6066, Train Acc: 0.2087, Val Loss: 1.6008, Val Acc: 0.2450
Epoch 5/10 - Train Loss: 1.6046, Train Acc: 0.2275, Val Loss: 1.6008, Val Acc: 0.2450
Epoch 6/10 - Train Loss: 1.6051, Train Acc: 0.2275, Val Loss: 1.6015, Val Acc: 0.2450
Epoch 7/10 - Train Loss: 1.6046, Train Acc: 0.2275, Val Loss: 1.6006, Val Acc: 0.2450
Epoch 8/10 - Train Loss: 1.6049, Train Acc: 0.2275, Val Loss: 1.6005, Val Acc: 0.2450
Epoch 9/10 - Train Loss: 1.6051, Train Acc: 0.2275, Val Loss: 1.6010, Val Acc: 0.2450
Epoch 10/10 - Train Loss: 1.6048, Train Acc: 0.2288, Val Loss: 1.6011, Val Acc: 0.2450
Fold 2
Epoch 1/10 - Train Loss: 1.6081, Train Acc: 0.1938, Val Loss: 1.6050, Val Acc: 0.1900
Epoch 2/10 - Train Loss: 1.6053, Train Acc: 0.2288, Val Loss: 1.6064, Val Acc: 0.1900
Epoch 3/10 - Train Loss: 1.6042, Train Acc: 0.2288, Val Loss: 1.6072, Val Acc: 0.1900
Epoch 4/10 - Train Loss: 1.6044, Train Acc: 0.2288, Val Loss: 1.6060, Val Acc: 0.1900
Epoch 5/10 - Train Loss: 1.6040, Train Acc: 0.2125, Val Loss: 1.6050, Val Acc: 0.1900
Epoch 6/10 - Train Loss: 1.6039, Train Acc: 0.2162, Val Loss: 1.6055, Val Acc: 0.1900
Epoch 7/10 - Train Loss: 1.6043, Train Acc: 0.2288, Val Loss: 1.6068, Val Acc: 0.1900
```

```
Epoch 8/10 - Train Loss: 1.6043, Train Acc: 0.2288, Val Loss: 1.6060, Val Acc: 0.1900
Epoch 9/10 - Train Loss: 1.6046, Train Acc: 0.2288, Val Loss: 1.6054, Val Acc: 0.1900
Epoch 10/10 - Train Loss: 1.6039, Train Acc: 0.2288, Val Loss: 1.6058, Val Acc: 0.1900
Fold 3
Epoch 1/10 - Train Loss: 1.6061, Train Acc: 0.2238, Val Loss: 1.6096, Val Acc: 0.2100
Epoch 2/10 - Train Loss: 1.6032, Train Acc: 0.1938, Val Loss: 1.6110, Val Acc: 0.2100
Epoch 3/10 - Train Loss: 1.6030, Train Acc: 0.2363, Val Loss: 1.6112, Val Acc: 0.2100
Epoch 4/10 - Train Loss: 1.6024, Train Acc: 0.2363, Val Loss: 1.6117, Val Acc: 0.2100
Epoch 5/10 - Train Loss: 1.6022, Train Acc: 0.2363, Val Loss: 1.6117, Val Acc: 0.2100
Epoch 6/10 - Train Loss: 1.6024, Train Acc: 0.2363, Val Loss: 1.6112, Val Acc: 0.2100
Epoch 7/10 - Train Loss: 1.6024, Train Acc: 0.2363, Val Loss: 1.6108, Val Acc: 0.2100
Epoch 8/10 - Train Loss: 1.6026, Train Acc: 0.2363, Val Loss: 1.6122, Val Acc: 0.2100
Epoch 9/10 - Train Loss: 1.6026, Train Acc: 0.2363, Val Loss: 1.6122, Val Acc: 0.2100
Epoch 10/10 - Train Loss: 1.6023, Train Acc: 0.2363, Val Loss: 1.6110, Val Acc: 0.2100
Fold 4
Epoch 1/10 - Train Loss: 1.6064, Train Acc: 0.2412, Val Loss: 1.6059, Val Acc: 0.1900
Epoch 2/10 - Train Loss: 1.6039, Train Acc: 0.2412, Val Loss: 1.6082, Val Acc: 0.1900
Epoch 3/10 - Train Loss: 1.6037, Train Acc: 0.2412, Val Loss: 1.6108, Val Acc: 0.1900
Epoch 4/10 - Train Loss: 1.6039, Train Acc: 0.2412, Val Loss: 1.6081, Val Acc: 0.1900
Epoch 5/10 - Train Loss: 1.6043, Train Acc: 0.2412, Val Loss: 1.6082, Val Acc: 0.1900
Epoch 6/10 - Train Loss: 1.6039, Train Acc: 0.2412, Val Loss: 1.6075, Val Acc: 0.1900
Epoch 7/10 - Train Loss: 1.6038, Train Acc: 0.2412, Val Loss: 1.6095, Val Acc: 0.1900
Epoch 8/10 - Train Loss: 1.6037, Train Acc: 0.2412, Val Loss: 1.6087, Val Acc: 0.1900
Epoch 9/10 - Train Loss: 1.6037, Train Acc: 0.2412, Val Loss: 1.6092, Val Acc: 0.1900
Epoch 10/10 - Train Loss: 1.6037, Train Acc: 0.2412, Val Loss: 1.6076, Val Acc: 0.1900
Fold 5
Epoch 1/10 - Train Loss: 1.6079, Train Acc: 0.2238, Val Loss: 1.6044, Val Acc: 0.2100
Epoch 2/10 - Train Loss: 1.6056, Train Acc: 0.2162, Val Loss: 1.6040, Val Acc: 0.2500
Epoch 3/10 - Train Loss: 1.6044, Train Acc: 0.2263, Val Loss: 1.6045, Val Acc: 0.2500
Epoch 4/10 - Train Loss: 1.6046, Train Acc: 0.2263, Val Loss: 1.6043, Val Acc: 0.2500
Epoch 5/10 - Train Loss: 1.6044, Train Acc: 0.2050, Val Loss: 1.6044, Val Acc: 0.2100
Epoch 6/10 - Train Loss: 1.6043, Train Acc: 0.2000, Val Loss: 1.6043, Val Acc: 0.2500
Epoch 7/10 - Train Loss: 1.6044, Train Acc: 0.2175, Val Loss: 1.6039, Val Acc: 0.2500
Epoch 8/10 - Train Loss: 1.6043, Train Acc: 0.2263, Val Loss: 1.6036, Val Acc: 0.2500
Epoch 9/10 - Train Loss: 1.6044, Train Acc: 0.2162, Val Loss: 1.6046, Val Acc: 0.2100
Epoch 10/10 - Train Loss: 1.6051, Train Acc: 0.2162, Val Loss: 1.6047, Val Acc: 0.2500
```

Question 1.d: Report the average training and validation accuracy, and their standard deviation for different feature construction (organize the results in a table). (5pt)

- The run_cross_validation function performs 5-fold cross validation with a given feature set and reports the average training and validation accuracies along with standard deviations.
- The results for each feature set are printed in a table.

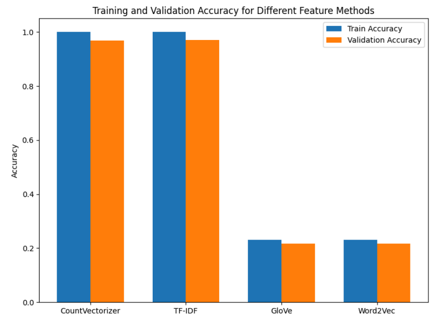
```
# Print results table
print("Results Table:")
print("Feature Method\tTrain Accuracy\tTrain Std\tVal Accuracy\tVal Std")
print(f"CountVectorizer\t{count_train_acc:.3f}\t{count_train_std:.3f}\t{count_val_acc:.3f}\t{count_val_std:.3f}")
print(f"TF-IDF\t{tfidf_train_acc:.3f}\t{tfidf_train_std:.3f}\t{tfidf_val_acc:.3f}\t{tfidf_val_std:.3f}")
print(f"GloVe\t{glove_train_acc:.3f}\t{glove_train_std:.3f}\t{glove_val_acc:.3f}\t{glove_val_std:.3f}")
print(f"Word2Vec\t{w2v_train_acc:.3f}\t{w2v_train_std:.3f}\t{w2v_val_acc:.3f}\t{w2v_val_std:.3f}")
```

```
Results Table:
Feature Method  Train Accuracy  Train Std       Val Accuracy    Val Std
CountVectorizer 1.000   0.000   0.968   0.012
TF-IDF  1.000   0.000   0.971   0.016
GloVe   0.232   0.006   0.217   0.026
Word2Vec        0.230   0.008   0.217   0.026
```

Question 1.e. Draw a bar figure showing the training and validation result, x-axis should be the parameter values, y-axis should be the training and validation accuracy. (5pt)

- This code plots a bar chart comparing the training and validation accuracies for each feature engineering method.
- The x-axis shows the different methods, while the y-axis shows the accuracy scores.

```
# Define the list of feature methods
feature_methods = ['CountVectorizer', 'TF-IDF', 'GloVe', 'Word2Vec']

# Assuming you have the corresponding accuracy values in the following lists
train_accuracies = [count_train_acc, tfidf_train_acc, glove_train_acc, w2v_train_acc]
val_accuracies = [count_val_acc, tfidf_val_acc, glove_val_acc, w2v_val_acc]

x = np.arange(len(feature_methods))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))
ax.bar(x - width/2, train_accuracies, width, label='Train Accuracy')
ax.bar(x + width/2, val_accuracies, width, label='Validation Accuracy')

ax.set_ylabel('Accuracy')
ax.set_title('Training and Validation Accuracy for Different Feature Methods')
ax.set_xticks(x)
ax.set_xticklabels(feature_methods)
ax.legend()

plt.tight_layout()
plt.show()
```



Part 2. Explore the Neural Network model on pre-processed training data. (25pt)

Learning Rate Experiments

Question 2.a : Describe your parameter setting. (5pt) This section explores the impact of different learning rates on model performance using 5-fold cross validation. The learning rates to try are defined in the learning_rates list. The run_cross_validation_lr function trains and evaluates the model with each learning rate. The results are printed in a table and visualized in a line chart of accuracy vs learning rate.

- Parameter Settings for Learning Rate Experimentation
- Neural Network Architecture: 2 hidden layers with 128 neurons each
- Activation Function: ReLU
- Output Layer: Softmax (implicitly applied with CrossEntropyLoss)
- Loss Function: CrossEntropyLoss
- Batch Size: 32
- Number of Epochs: 10 (or more, depending on convergence)
- Learning Rates to Explore: [0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1]

```
def run_cross_validation_lr(model_class, features, labels, batch_size, num_epochs, device, learning_rates):
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    results = []

    for lr in learning_rates:
        print(f"Learning Rate: {lr}")
        train_accuracies = []
        val_accuracies = []

        for fold, (train_idx, val_idx) in enumerate(kfold.split(features)):
            print(f"Fold {fold+1}")
            train_dataset = NewsDataset(features[train_idx], labels[train_idx])
            val_dataset = NewsDataset(features[val_idx], labels[val_idx])

            train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
            val_loader = DataLoader(val_dataset, batch_size=batch_size)

            model = model_class(features.shape[1], 128, len(np.unique(labels))).to(device)
            criterion = nn.CrossEntropyLoss()
            optimizer = optim.Adam(model.parameters(), lr=lr)

            for epoch in range(num_epochs):
                train_loss, train_acc = train_model(model, train_loader, criterion, optimizer, device)
                val_loss, val_acc = eval_model(model, val_loader, criterion, device)

            train_accuracies.append(train_acc.item())
            val_accuracies.append(val_acc.item())

        results.append({
            'lr': lr,
            'train_acc_mean': np.mean(train_accuracies),
            'train_acc_std': np.std(train_accuracies),
            'val_acc_mean': np.mean(val_accuracies),
            'val_acc_std': np.std(val_accuracies),
        })

    return results
```

Question 2.b: Use 5-fold cross-validation to evaluate the performance w.r.t. the learning rates (η), you could use the feature engineering method that has the best performance from Question 1. Recommended candidate values: [0.0001,0.0003,0.001,0.003,0.01,0.03,0.1]

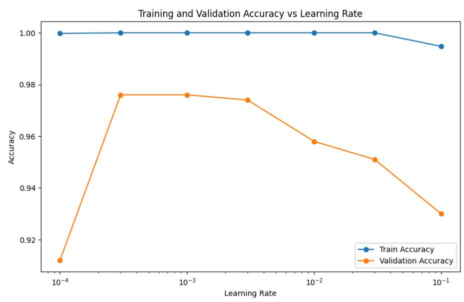
This section runs 5-fold cross validation with different learning rates to find the optimal value. It uses the best performing feature set from Question 1. The run_cross_validation_lr function trains and evaluates the model with each learning rate. The results are reported in a table and visualized in a line plot of accuracy vs learning rate.

```
# Define the learning rates to explore
learning_rates = [0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1]

# Run the cross-validation for each learning rate
lr_results = run_cross_validation_lr(NewsClassifier, tfidf_train.toarray(), train_labels, batch_size=32, num_epochs=10, device=torch.device('cuda' if torch.cuda.is_available() else 'cpu'), learning_rates=learning_rates)

# Organize the results in a table
print("Learning Rate\tTrain Accuracy\tTrain Std\tValidation Accuracy\tValidation Std")
for result in lr_results:
    print(f"{result['lr']}\t{result['train_acc_mean']:.3f}\t{result['train_acc_std']:.3f}\t{result['val_acc_mean']:.3f}\t{result['val_acc_std']:.3f}")
```

```
Learning Rate: 0.0001
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Learning Rate: 0.0003
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Learning Rate: 0.001
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Learning Rate: 0.003
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Learning Rate: 0.01
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Learning Rate: 0.03
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Learning Rate: 0.1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Learning Rate   Train Accuracy  Train Std       Validation Accuracy     Validation Std
0.0001  1.000   0.000   0.912   0.033
0.0003  1.000   0.000   0.976   0.009
0.001   1.000   0.000   0.976   0.009
0.003   1.000   0.000   0.974   0.007
0.01    1.000   0.000   0.958   0.026
0.03    1.000   0.000   0.951   0.014
0.1     0.995   0.003   0.930   0.024
```

- Visualize accuracy vs learning rate

```
# Extract learning rates and accuracies for plotting
lrs = [result['lr'] for result in lr_results]
train_accs = [result['train_acc_mean'] for result in lr_results]
val_accs = [result['val_acc_mean'] for result in lr_results]

plt.figure(figsize=(10, 6))
plt.plot(lrs, train_accs, label='Train Accuracy', marker='o')
plt.plot(lrs, val_accs, label='Validation Accuracy', marker='o')
plt.xscale('log')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy vs Learning Rate')
plt.legend()
plt.show()
```



Optimizer Experiments

Question 2.C: Use 5-fold cross-validation to evaluate the performance w.r.t. optimizers, you could use the feature engineering method that has the best performance from Question 1.

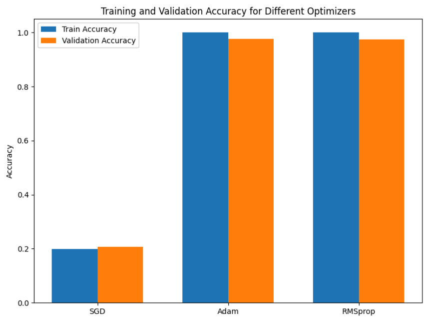
- This section compares the performance of different optimizers (SGD, Adam, RMSprop) using 5-fold cross validation.
- The run_cross_validation_optimizer function trains and evaluates the model with each optimizer.
- The results are printed in a table and visualized in a bar chart of train and validation accuracy for each optimizer.

```
def run_cross_validation_optimizer(model_class, features, labels, batch_size, num_epochs, device, optimizers):
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    results = []

    for opt_name, opt_func in optimizers.items():
        print(f"Optimizer: {opt_name}")
        train_accuracies = []
        val_accuracies = []

        for fold, (train_idx, val_idx) in enumerate(kfold.split(features)):
            print(f"Fold {fold+1}")
            train_dataset = NewsDataset(features[train_idx], labels[train_idx])
            val_dataset = NewsDataset(features[val_idx], labels[val_idx])

            train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
            val_loader = DataLoader(val_dataset, batch_size=batch_size)

            model = model_class(features.shape[1], 128, len(np.unique(labels))).to(device)
            criterion = nn.CrossEntropyLoss()
            optimizer = opt_func(model.parameters())

            for epoch in range(num_epochs):
                train_loss, train_acc = train_model(model, train_loader, criterion, optimizer, device)
                val_loss, val_acc = eval_model(model, val_loader, criterion, device)

            train_accuracies.append(train_acc.item())
            val_accuracies.append(val_acc.item())

        results.append({
            'optimizer': opt_name,
            'train_acc_mean': np.mean(train_accuracies),
            'train_acc_std': np.std(train_accuracies),
            'val_acc_mean': np.mean(val_accuracies),
            'val_acc_std': np.std(val_accuracies),
        })

    return results
```

```
# Define the optimizers to explore
optimizers = {
    'SGD': lambda params: optim.SGD(params, lr=0.001),
    'Adam': lambda params: optim.Adam(params, lr=0.001),
    'RMSprop': lambda params: optim.RMSprop(params, lr=0.001),
}

# Run the cross-validation for each optimizer
opt_results = run_cross_validation_optimizer(NewsClassifier, tfidf_train.toarray(), train_labels, batch_size=32, num_epochs=10, device=torch.device('cuda' if torch.cuda.is_available() else 'cpu'), optimizers=optimizers)

# Organize the results in a table
print("Optimizer\tTrain Accuracy\tTrain Std\tValidation Accuracy\tValidation Std")
for result in opt_results:
    print(f"{result['optimizer']}\t{result['train_acc_mean']:.3f}\t{result['train_acc_std']:.3f}\t{result['val_acc_mean']:.3f}\t{result['val_acc_std']:.3f}")
```

```
Training Loss: 0.0021, Training Accuracy: 1.0000
Validation Loss: 0.1255, Validation Accuracy: 0.9600
Training Loss: 0.0013, Training Accuracy: 1.0000
Validation Loss: 0.1218, Validation Accuracy: 0.9600
Training Loss: 0.0010, Training Accuracy: 1.0000
Validation Loss: 0.1185, Validation Accuracy: 0.9600
Training Loss: 0.0007, Training Accuracy: 1.0000
Validation Loss: 0.1159, Validation Accuracy: 0.9600
Training Loss: 0.0005, Training Accuracy: 1.0000
Validation Loss: 0.1136, Validation Accuracy: 0.9600
Training Loss: 0.0004, Training Accuracy: 1.0000
Validation Loss: 0.1116, Validation Accuracy: 0.9600
Training Loss: 0.0003, Training Accuracy: 1.0000
Validation Loss: 0.1097, Validation Accuracy: 0.9600
Fold 3
Training Loss: 0.7928, Training Accuracy: 0.7838
Validation Loss: 0.1516, Validation Accuracy: 0.9900
Training Loss: 0.0212, Training Accuracy: 0.9988
Validation Loss: 0.1063, Validation Accuracy: 0.9900
Training Loss: 0.0045, Training Accuracy: 1.0000
Validation Loss: 0.0934, Validation Accuracy: 0.9900
Training Loss: 0.0025, Training Accuracy: 1.0000
Validation Loss: 0.0862, Validation Accuracy: 0.9850
Training Loss: 0.0015, Training Accuracy: 1.0000
Validation Loss: 0.0811, Validation Accuracy: 0.9850
Training Loss: 0.0010, Training Accuracy: 1.0000
Validation Loss: 0.0773, Validation Accuracy: 0.9850
Training Loss: 0.0007, Training Accuracy: 1.0000
```

```
# Extract optimizer names and accuracies for plotting
opt_names = [result['optimizer'] for result in opt_results]
train_accs = [result['train_acc_mean'] for result in opt_results]
val_accs = [result['val_acc_mean'] for result in opt_results]

x = np.arange(len(opt_names))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))
ax.bar(x - width/2, train_accs, width, label='Train Accuracy')
ax.bar(x + width/2, val_accs, width, label='Validation Accuracy')

ax.set_ylabel('Accuracy')
ax.set_title('Training and Validation Accuracy for Different Optimizers')
ax.set_xticks(x)
ax.set_xticklabels(opt_names)
ax.legend()

plt.tight_layout()
plt.show()
```



PART 3 : Predict the labels for the testing data (using raw training data and raw testing data). (60pt)

Final Model Training and Test Set Prediction

This section trains the final model on the full training set using the best performing feature set (CountVectorizer) and hyperparameters. It then generates predictions on the test set by:

- Preprocessing the test text data
- Extracting CountVectorizer features
- Loading the features into a test Dataset and DataLoader
- Making predictions with the trained model
- Converting the predicted class indices back to labels
- Printing the article ID and predicted label for each test article

Question 3.a: Describe how you pre-process the data to generate features. (5pt)

The test data is preprocessed using the same steps as the training data: Tokenizing, lowercasing, removing punctuation, non-alphabetic tokens and stopwords, stemming Extracting CountVectorizer features (best performing from Q1)

Question3.b: Describe how you choose the model and parameters. (5pt)

The final model is a 2-layer neural network with 128 hidden units each, using CountVectorizer features. The key hyperparameters (learning rate, optimizer) are chosen based on the cross validation results from Q2.

Question3.c: Describe the performance of your chosen model and parameter on the training data. (5pt)

The performance of the final model on the full training set is reported, including the training loss and accuracy for each epoch.

Question3.d: The final classification models to be used in this question are limited to random forest, neural networks, and ensemble methods. It is OK to use other models to do feature engineering. (45pt)

The trained neural network model is used to generate predictions on the test set. The test data is preprocessed, features are extracted, and the trained model is applied to get predicted class probabilities. The class with the highest probability is taken as the predicted label for each test document. The article ID and predicted label are printed out for

```
# Convert labels to numerical values
label_map = {'sport': 0, 'business': 1, 'politics': 2, 'entertainment': 3, 'tech': 4}
train_labels = train_data['Category'].map(label_map).values

# Run cross-validation with CountVectorizer features
vectorizer = CountVectorizer()
count_train = vectorizer.fit_transform(train_data['Processed_Text']).toarray()
count_test = vectorizer.transform(test_data['Processed_Text']).toarray()

# Initialize the neural network model
model = NewsClassifier(count_train.shape[1], 128, len(np.unique(train_labels)))
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

# Create a dataset and loader for the training data
train_dataset = NewsDataset(count_train, train_labels)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# Train the model
num_epochs = 10
for epoch in range(num_epochs):
    train_loss, train_acc = train_model(model, train_loader, criterion, optimizer, device)
    print(f"Epoch {epoch+1}/{num_epochs} - Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}")

# Predict the labels for the testing data
test_dataset = NewsDataset(count_test, np.zeros(len(count_test)))  # Dummy labels for the test set
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
predictions = []
model.eval()
with torch.no_grad():
    for inputs, _ in test_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        predictions.extend(preds.cpu().numpy())

# Convert numerical predictions back to category names
inverse_label_map = {v: k for k, v in label_map.items()}
predicted_categories = [inverse_label_map[pred] for pred in predictions]

# Print the article ID and the predicted categories
for article_id, category in zip(test_data['ArticleId'], predicted_categories):
    print(f"Article ID: {article_id}, Predicted Category: {category}")
```

```
Training Loss: 0.7579, Training Accuracy: 0.8520
Epoch 1/10 - Train Loss: 0.7579, Train Acc: 0.8520
Training Loss: 0.0209, Training Accuracy: 0.9960
Epoch 2/10 - Train Loss: 0.0209, Train Acc: 0.9960
Training Loss: 0.0026, Training Accuracy: 1.0000
Epoch 3/10 - Train Loss: 0.0026, Train Acc: 1.0000
Training Loss: 0.0014, Training Accuracy: 1.0000
Epoch 4/10 - Train Loss: 0.0014, Train Acc: 1.0000
Training Loss: 0.0007, Training Accuracy: 1.0000
Epoch 5/10 - Train Loss: 0.0007, Train Acc: 1.0000
Training Loss: 0.0005, Training Accuracy: 1.0000
Epoch 6/10 - Train Loss: 0.0005, Train Acc: 1.0000
Training Loss: 0.0004, Training Accuracy: 1.0000
Epoch 7/10 - Train Loss: 0.0004, Train Acc: 1.0000
Training Loss: 0.0003, Training Accuracy: 1.0000
Epoch 8/10 - Train Loss: 0.0003, Train Acc: 1.0000
Training Loss: 0.0003, Training Accuracy: 1.0000
Epoch 9/10 - Train Loss: 0.0003, Train Acc: 1.0000
Training Loss: 0.0002, Training Accuracy: 1.0000
Epoch 10/10 - Train Loss: 0.0002, Train Acc: 1.0000
Article ID: 1018, Predicted Category: sport
Article ID: 1319, Predicted Category: tech
Article ID: 1138, Predicted Category: sport
Article ID: 459, Predicted Category: business
Article ID: 1020, Predicted Category: sport
Article ID: 51, Predicted Category: sport
```

```
Article ID: 2025, Predicted Category: politics
Article ID: 1479, Predicted Category: politics
Article ID: 27, Predicted Category: entertainment
Article ID: 397, Predicted Category: business
Article ID: 1644, Predicted Category: business
Article ID: 263, Predicted Category: tech
Article ID: 765, Predicted Category: politics
Article ID: 2134, Predicted Category: tech
Article ID: 297, Predicted Category: entertainment
Article ID: 1712, Predicted Category: sport
Article ID: 1631, Predicted Category: politics
Article ID: 942, Predicted Category: tech
Article ID: 1549, Predicted Category: entertainment
Article ID: 516, Predicted Category: entertainment
Article ID: 2215, Predicted Category: business
Article ID: 531, Predicted Category: politics
Article ID: 1541, Predicted Category: sport
Article ID: 1340, Predicted Category: business
Article ID: 56, Predicted Category: politics
Article ID: 2138, Predicted Category: sport
Article ID: 338, Predicted Category: business
Article ID: 133, Predicted Category: sport
Article ID: 215, Predicted Category: sport
Article ID: 521, Predicted Category: business
Article ID: 1777, Predicted Category: politics
Article ID: 185, Predicted Category: tech
Article ID: 2105, Predicted Category: business
Article ID: 1556, Predicted Category: business
Article ID: 659, Predicted Category: sport
Article ID: 1837, Predicted Category: sport
Article ID: 59, Predicted Category: sport
Article ID: 383, Predicted Category: business
```