DM-2024 Lab2 HW

code by 111060026 daisyliu0225
name: 劉孟瑋
Student ID: 111060026
github ID: daisyliu0225
Kaggle name: daisyliu0225

Adapted Versions

Code 1: BOW + Deep learning

The code in the repo is: version2_BOW

Versions

Version	Max Features	Train Data Sample	epoch	batch size	Result
Original	500	0.3	25	32	0.36322
Version 1	500	0.3	25	40	0.36322
Version 2	500	0.3	40	32	0.36322
Version 3	1000	0.3	40	512	0.37987
Version 4	2000	0.3	40	512	0.37959
Version 5	1800	0.3	40	512	0.38144
Version 6	1800	1	40	512	0.41604

Code implementation:
Here, I will give some of the code that I think important. If other addition reference is needed, please refer to the code in the repo.

1. Data preprocessing

```
# The code first takes in the json data through a for loop
data = []
with open('./dm-2024-isa-5810-lab-2-homework/tweets_DM.json', 'r') as f:
    for line in f:
        try:
            data.append(json.loads(line))  # Safeguard against malformed JSON
        except json.JSONDecodeError as e:
            print(f"Error decoding JSON: {e}")

f.close()
```

```
# The code next takes in the emotion csv and identification csv
emotion_list = pd.read_csv('./dm-2024-isa-5810-lab-2-homework/emotion.csv')
data_identification = pd.read_csv('./dm-2024-isa-5810-lab-2-
homework/data_identification.csv')
```

```
# Create dataframe and merge the tweet, hashtags, text into dataframe
df = pd.DataFrame(data)

# Extract '_source' and validate structure
if '_source' not in df.columns:
    raise KeyError("'_source' column not found in the data")

_source = df['_source'].apply(lambda x: x['tweet'])
df = pd.DataFrame({
    'tweet_id': _source.apply(lambda x: x['tweet_id']),
    'hashtags': _source.apply(lambda x: x['hashtags']),
    'text': _source.apply(lambda x: x['text']),
})

# Ensure tweet_id is of a consistent type
df['tweet_id'] = df['tweet_id'].astype(str)

# Validate and prepare `data_identification`
data_identification['tweet_id'] = data_identification['tweet_id'].astype(str)
df = df.merge(data_identification, on='tweet_id', how='left')

train_data = df[df['identification'] == 'train']
test_data = df[df['identification'] == 'test']
```

```
# Drop duplicates in the data
train_data.drop_duplicates(subset=['text'], keep=False, inplace=True)
```

```
# Split the training data to training data and testing data, because
# we cannot mix the true testing data with training data.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train_data, y_train_data,
test_size=0.2, random_state=42)
```

2. Deep learning process The deep learning process uses BOW process to mine the data. Then utilize deep learning to create a model.

```
# Create BOW analyzer
# build analyzers (bag-of-words)
BOW_500 = CountVectorizer(max_features=1800, tokenizer=nltk.word_tokenize,
                          token_pattern=None)

# apply analyzer to training data
BOW_500.fit(train_data['text'])
```

```
# Transfer the data that is able to input into the model
# standardize name (X, y)
X_trainv2 = BOW_500.transform(X_train['text'])
y_trainv2 = y_train['emotion']

X_testv2 = BOW_500.transform(X_test['text'])
y_testv2 = y_test['emotion']

# remember to transform the true testing data as well
ans_datav2 = BOW_500.transform(ans_data['text'])
```

```
# Deal with categorical label(y)
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
label_encoder.fit(y_trainv2)
print('check label: ', label_encoder.classes_)
print('\n## Before convert')
print('y_train[0:4]:\n', y_trainv2[0:4])
print('\ny_train.shape: ', y_trainv2.shape)
print('y_test.shape: ', y_testv2.shape)

def label_encode(le, labels):
    enc = le.transform(labels)
    return keras.utils.to_categorical(enc)

def label_decode(le, one_hot_label):
    dec = np.argmax(one_hot_label, axis=1)
    return le.inverse_transform(dec)

y_trainv2 = label_encode(label_encoder, y_trainv2)
y_testv2 = label_encode(label_encoder, y_testv2)
```

```
# build the model
from keras.models import Model
from keras.layers import Input, Dense
from keras.layers import ReLU, Softmax

# input layer
model_input = Input(shape=(input_shape, ))  # 500
X = model_input

# 1st hidden layer
X_W1 = Dense(units=64)(X)  # 64
H1 = ReLU()(X_W1)

# 2nd hidden layer
H1_W2 = Dense(units=64)(H1)  # 64
H2 = ReLU()(H1_W2)

# output layer
H2_W3 = Dense(units=output_shape)(H2)  # 4
H3 = Softmax()(H2_W3)

model_output = H3

# create model
model = Model(inputs=[model_input], outputs=[model_output])

# loss function & optimizer
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
from keras.callbacks import CSVLogger

# training setting
epochs = 40
batch_size = 512

# training!
history = model.fit(X_trainv2, y_trainv2,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data = (X_testv2, y_testv2))
print('training finish')
```

```
# predict the data using the trained model
pred_X_test = model.predict(X_testv2, batch_size=128)
pred_X_test[:5]
```

3. Evaluate the data

```
# To make things easier, we evaluate using accuracy because we don't need to know
the recall and precision
# Use the testing data we split during the process of preprocessing
from sklearn.metrics import accuracy_score
#Accuracy
print('testing accuracy:
{}'.format(round(accuracy_score(label_decode(label_encoder, y_testv2),
pred_X_test), 2)))
```

4. Output the result

```
# When the accuracy looks fine, we create the submission file
## predict the true test data
pred_result = model.predict(ans_datav2, batch_size=64)
pred_result[:5]

## create a submission file according to it
submission = pd.DataFrame({
    'id': test_data['tweet_id'],
    'emotion': pred_result,
})
submission.to_csv('./submission.csv', index=False)
```

A few conclusions:

1. epoch and batch size does not significant affect the result
2. Train data sample can affect the result(I did not test it on kaggle), but I had a small "accuracy test label" that helps me test what is going on in my code.

```
from sklearn.metrics import accuracy_score
#Accuracy
print('testing accuracy:
{}'.format(round(accuracy_score(label_decode(label_encoder, y_testv2),
pred_X_test), 2)))
```

3. The number of max features has a limit and will decrease if exceed the limit. The limit in this data is around 1800.

note: some of the .head() code in the code is used to check the shape of the input or it will output error messages.

Discarded Versions

Code 1: Decision Tree

The code in the repo is: version1_decision_tree
Discarded reason: Training the decision tree is too long and the accuracy is too low. Only around 0.3.

Code implementation:

1. Code preprocessing
   The preprocessing process is the same as BOW+Deep learning
2. Decision Tree implementation

```
# The code still uses BOW to do text mining
# build analyzers (bag-of-words)
BOW_500 = CountVectorizer(max_features=500, tokenizer=nltk.word_tokenize,
                          token_pattern=None)

# apply analyzer to training data
BOW_500.fit(train_data['text'])

train_data_BOW_features_500 = BOW_500.transform(train_data['text'])
```

```
# for a classificaiton problem, you need to provide both training & testing data
X_train = BOW_500.transform(train_data['text'])
y_train = train_data['emotion']

X_test = BOW_500.transform(train_data['text'])
y_test = train_data['emotion']
```

```
## build DecisionTree model
DT_model = DecisionTreeClassifier(random_state=1)

## training!
DT_model = DT_model.fit(X_train, y_train)
```

```
## predict!
y_train_pred = DT_model.predict(X_train)
y_test_pred = DT_model.predict(X_test)
```

3. Evaluate the data
   The same as the way of evaluating data in BOW+Deep learning

4. Output the data
   The same way as outputting data in BOW+Deep learning

Code 2: BERT

The code in the repo is: version3_bert
It takes too long to predict. According to GPT, it may take up to days if using a CPU and hours using GPU.

1. Data preprocessing
   The preprocessing process is the same as BOW+Deep learning

2. BERT implementation

```
# preprocessing in BERT
def preprocess_data(df):
    # Convert text to lowercase (or apply other preprocessing steps as needed)
    df['text'] = df['text'].str.lower()
    return df
```

```
# tokenize text data for BERT input
# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the text data for BERT input
def encode_data(df, tokenizer, max_length=128):
    return tokenizer(list(df['text']), padding=True, truncation=True,
max_length=max_length, return_tensors='pt')
```

```
# create labels
from sklearn.preprocessing import LabelEncoder
# Step 1: Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Step 2: Fit the label encoder on the training emotions and transform both train
and test labels
y_train['encoded'] = label_encoder.fit_transform(y_train['emotion'])
y_test['encoded'] = label_encoder.transform(y_test['emotion'])

# Convert the labels into tensors
train_labels = torch.tensor(y_train['encoded'].values)
test_labels = torch.tensor(y_test['encoded'].values)
```

```
# Create DataLoader for training and testing
train_dataset = TensorDataset(X_train_encodings.input_ids,
X_train_encodings.attention_mask, train_labels)
test_dataset = TensorDataset(X_test_encodings.input_ids,
X_test_encodings.attention_mask, test_labels)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16)
```

```
# load the pretrained BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=len(y_train['emotion'].unique()))

# Set up the optimizer
optimizer = AdamW(model.parameters(), lr=1e-5)
```

```
# training
# Function to train the model
def train_model(model, train_loader, optimizer, device):
    model.train()
    total_loss = 0
    for batch in train_loader:
        input_ids, attention_mask, labels = [item.to(device) for item in batch]
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
```

```
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        return total_loss / len(train_loader)

    # Function to evaluate the model
    def evaluate_model(model, test_loader, device):
        model.eval()
        all_preds = []
        all_labels = []
        with torch.no_grad():
            for batch in test_loader:
                input_ids, attention_mask, labels = [item.to(device) for item in
    batch]
                outputs = model(input_ids, attention_mask=attention_mask)
                logits = outputs.logits
                preds = torch.argmax(logits, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())
        return all_labels, all_preds
```

3. Evaluate the data
   The same as the way of evaluating data in BOW+Deep learning

4. Output the data
   The same way as outputting data in BOW+Deep learning

Code 3: n_grams + Deep learning

The code in the repo is: version4_n_grams
Discarded reason: Training n-grams is too long and the accuracy is too low. It is about 0.30501.

1. Data preprocessing
   The preprocessing process is the same as BOW+Deep learning

2. N-gram implementation

```
# Create a CountVectorizer with bigrams (2-grams)
vectorizer = CountVectorizer(ngram_range=(2, 2))  # (2, 2) means bigrams
vectorizer.fit(train_data['text'])

# Do the transformation to input the deep learning
X_training = vectorizer.transform(X_train_data['text'])
```

```
import keras
from sklearn.preprocessing import LabelEncoder
```

```
label_encoder = LabelEncoder()
label_encoder.fit(y_train)
print('check label: ', label_encoder.classes_)
print('\n## Before convert')
print('y_train[0:4]:\n', y_train[0:4])
print('\ny_train.shape: ', y_train.shape)
print('y_test.shape: ', y_test.shape)

def label_encode(le, labels):
    enc = le.transform(labels)
    return keras.utils.to_categorical(enc)

def label_decode(le, one_hot_label):
    dec = np.argmax(one_hot_label, axis=1)
    return le.inverse_transform(dec)

y_train = label_encode(label_encoder, y_train)
y_test = label_encode(label_encoder, y_test)

print('\n\n## After convert')
print('y_train[0:4]:\n', y_train[0:4])
print('\ny_train.shape: ', y_train.shape)
print('y_test.shape: ', y_test.shape)
```

```
# build the model
from keras.models import Model
from keras.layers import Input, Dense
from keras.layers import ReLU, Softmax

# input layer
model_input = Input(shape=(input_shape, ))  # 500
X = model_input

# 1st hidden layer
X_W1 = Dense(units=64)(X)  # 64
H1 = ReLU()(X_W1)

# 2nd hidden layer
H1_W2 = Dense(units=64)(H1)  # 64
H2 = ReLU()(H1_W2)

# output layer
H2_W3 = Dense(units=output_shape)(H2)  # 4
H3 = Softmax()(H2_W3)

model_output = H3

# create model
model = Model(inputs=[model_input], outputs=[model_output])
```

```
# loss function & optimizer
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# show model construction
model.summary()
```

```
# train the model
from keras.callbacks import CSVLogger

# csv_logger = CSVLogger('/content/drive/My Drive/NTHU/DM-Lab2-
HW/logs/training_log.csv')

# training setting
epochs = 3
batch_size = 256

# training!
history = model.fit(X_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data = (X_test, y_test))
print('training finish')
```

```
# predict the data
pred_X_test = model.predict(X_test, batch_size=128)
pred_X_test[:5]
```

3. Evaluate the data
   The same as the way of evaluating data in BOW+Deep learning

4. Output the data
   The same way as outputting data in BOW+Deep learning

Kaggle Results

Public:

76	OurStoRi/NTHU112062649		0.41968	3	13d
77	**daisyliu0225**		0.41604	12	2d
	Your Best Entry! Your most recent submission scored 0.41604, which is the same as your previous score. Keep trying!				
78	657166226		0.41481	4	2d

Private:

74	▲ 2	OurStoRi/NTHU112062649		0.40424	3	13d
75	▲ 2	**daisyliu0225**		0.40130	12	2d
76	▲ 2	657166226		0.40104	4	2d