DM 2024 Fall Lab 2 Homework

IEEM 110034103 陳詩凱

Part 1. Load data

```python
import json
import pandas as pd

# Step 1: Read the JSON file line by line and parse valid records
records = []
with open(r'C:\Users\skych\DM2024-Lab2-Homework\Data\tweets_DM.json', 'r', encoding='utf-8') as file:
    for line in file:
        line = line.strip()  # Remove leading/trailing whitespace
        if not line:  # Skip empty lines
            continue
        try:
            # Parse each line as JSON
            data = json.loads(line)
            tweet = data.get('_source', {}).get('tweet', {})
            tweet_id = tweet.get('tweet_id')
            text = tweet.get('text')

            # Remove or replace newline characters in text
            if text:
                text = text.replace('\n', ' ').replace('\r', ' ')

            # Append only valid records with both id and text
            if tweet_id and text:
                records.append({'id': tweet_id, 'text': text})
        except json.JSONDecodeError:
            # Skip lines that fail to parse
            continue

# Step 2: Create a DataFrame from the valid records
df = pd.DataFrame(records)

# Step 3: Save to a CSV file with proper quoting
df.to_csv('tweets_cleaned.csv', index=False, quoting=1)  # quoting=1 ensures all text fields are quoted

# Step 4: Display the number of records
print(f"Processed {len(records)} valid tweets.")
df
```
✓ 10.4s Python

Since it is a json file, we need to extract id and emotions and transform it into the dataset.

```python
import pandas as pd

# Step 1: Load the datasets
tweets_df = pd.read_csv('Data/tweets_cleaned.csv')  # Contains columns: id, text
identification_df = pd.read_csv('Data/data_identification.csv')  # Contains columns: tweet_id, identification
emotion_df = pd.read_csv('Data/emotion.csv')  # Contains columns: tweet_id, emotion

# Step 2: Merge the datasets on tweet_id/id
merged_df = tweets_df.merge(identification_df, left_on='id', right_on='tweet_id')
merged_df = merged_df.merge(emotion_df, left_on='id', right_on='tweet_id', how='left')  # Add emotion column

# Step 3: Split the data into train and test based on the 'identification' column
train_df = merged_df[merged_df['identification'] == 'train']
test_df = merged_df[merged_df['identification'] == 'test']

train_df = train_df.drop(columns=['tweet_id_x', 'tweet_id_y'])
test_df = test_df.drop(columns=['identification', 'tweet_id_x', 'tweet_id_y'])

# Step 4: Save the split datasets
train_df[['id', 'text', 'emotion']].to_csv('train.csv', index=False, header=True)  # Save with headers
test_df[['id', 'text', 'emotion']].to_csv('test.csv', index=False, header=True)    # Save with headers

# Step 5: Print information
print(f"Train data: {len(train_df)} entries saved to 'train.csv'")
print(f"Test data: {len(test_df)} entries saved to 'test.csv'")
test_df
```
✓ 9.8s Python

There is a file that indicates which train and test data. Extract all the train data as a CSV file and test data as another.

```python
print("Shape of Training df: ", train_df.shape)
print("Shape of Testing df: ", test_df.shape)
```
✓ 0.0s Python

```
Shape of Training df:  (1455563, 4)
Shape of Testing df:  (411972, 3)
```

We can tell that the training and test df shape are 1455563 and 411972.

Part 2. EDA

```python
# group to find distribution
train_df.groupby(['emotion']).count()['text']
```

✓ 0.1s Python

```
emotion
anger            39867
anticipation    248935
disgust         139101
fear             63999
joy             516017
sadness         193437
surprise         48729
trust           205478
Name: text, dtype: int64
```

Most of the classifications is joy, then anticipation, quite imbalanced.

```python
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

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

def preprocess_text(text):
    text = re.sub(r'<[^>]+>', '', text)  # Remove HTML tags
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = text.lower()  # Lowercase
    tokens = word_tokenize(text)  # Tokenize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)

train_df['text'] = train_df['text'].apply(preprocess_text)
test_df['text'] = test_df['text'].apply(preprocess_text)
```

Python

The step helps to filter the HTML tags and punctuations and tokenize the text data.

Part 3. Feature Engineering

```python
import nltk
nltk.download('punkt_tab')

# build analyzers (bag-of-words)
BOW_500 = CountVectorizer(max_features=1000, tokenizer=nltk.word_tokenize)

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

train_data_BOW_features_500 = BOW_500.transform(train_df['text'])

## check dimension
train_data_BOW_features_500.shape
```

Python

```python
train_data_BOW_features_500.toarray()
```

Python

```python
# observe some feature names
feature_names_500 = BOW_500.get_feature_names_out()
feature_names_500[100:110]
```

Python

```python
"😂" in feature_names_500
```

Python

Use nltk to Vectorize the train_df[‘text’] and fit them into array.

Part3. Modeling

```python
from sklearn.preprocessing import LabelEncoder

# Encode the labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(train_df['emotion'])
# y_test_encoded = label_encoder.transform(y_test)
```

Python

Firstly, use LabelEncoder to encode train_df[‘emotion’] as binary, we can actually distinguish different emotions.

```python
import keras
from keras.models import Sequential
from keras.layers import Dense, ReLU, Softmax

# Convert sparse matrices to dense
X_train = train_data_BOW_features_500.toarray()
X_test = test_data_BOW_features_500.toarray()

# One-hot encode labels
y_train = keras.utils.to_categorical(y_train_encoded)

print("Input shape:", X_train.shape)
print("Output shape:", y_train.shape)

# Build the neural network model
model = Sequential([
    Dense(units=64, input_shape=(X_train.shape[1],)),
    ReLU(),
    Dense(units=64),
    ReLU(),
    Dense(units=y_train.shape[1]),  # Number of classes
    Softmax()
])
```

```python
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
epochs = 25
batch_size = 256
history = model.fit(X_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size)

print("Training completed.")

# Predict emotions for the test dataset
y_pred_proba = model.predict(X_test)  # Predict probabilities for each class
y_pred = np.argmax(y_pred_proba, axis=1)  # Get class indices
y_pred_labels = label_encoder.inverse_transform(y_pred)  # Convert indices back to original labels

# Save the predictions to test_df
test_df['emotion'] = y_pred_labels

# Save the updated test_df to a CSV file
test_df[['id', 'emotion']].to_csv('test_with_predictions_1206.csv', index=False)
print("Predictions saved to 'test_with_predictions_1206.csv'.")
```

Python

Construct a DL model using Keras, and we can see that the training process is quite slow, and most importantly, the accuracy increases from 0.3 to 0.5.

Part 4. Train as decision tree

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assuming 'train_df' is your training DataFrame with 'text' and 'emotion' columns
X = train_df['text']
y = train_df['emotion']

# Split the data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Create TF-IDF features
vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)

# Train the decision tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_tfidf, y_train)
```

```python
# Make predictions on validation set
y_pred = dt_model.predict(X_val_tfidf)

# Calculate accuracy
accuracy = accuracy_score(y_val, y_pred)
print(f"Validation accuracy: {accuracy:.4f}")

# Prepare test data
X_test = test_df['text']
X_test_tfidf = vectorizer.transform(X_test)

# Make predictions on test set
test_predictions = dt_model.predict(X_test_tfidf)

# Create submission DataFrame
submission_df = pd.DataFrame({'id': test_df['id'], 'emotion': test_predictions})
submission_df.to_csv('decision_tree_submission_1203.csv', index=False)
```
Python

The process of constructing decision tree is extremely waste of time (about 30 mins+). I assume this is because of an imbalanced dataset, which makes the training process very unsuccessful. The accuracy is only 0.18.

Why do I not use BERT or other LLM model?

Actually, I have tried many times; however, the BERT consumes a lot of time, even to 138 hours. Namely, it takes about 6 days to model it. Another reason is that it consumes too much of my laptop resources, many times the terminal shows Resource Exhaust Error. That’s why I didn’t implement LLM model to analyze this data.