Report HW1 CSCI544

Name : Sahil Mondal

USC-ID : 5092826451

Python Version : 3.9.17

1. Dataset Preparation

For this part my code is preparing a dataset by loading an Amazon product review dataset, keeping only the 'star_rating' and 'review_body' columns by reading a tab-delimited file and skipping to read lines which have formatting issues. It then creates a new 'sentiment_class' column, assigning the value 1 to reviews with ratings 1, 2, or 3, and 2 to reviews with ratings 4 or 5. To balance the classes, I downsample both classes to 50,000 samples each using resample(). Finally, the dataset is split into a training set (80% of the data) and a testing set (20% of the data).

2. Data Cleaning

For data cleaning I drop rows of data which have nan values anywhere. Additionally, I convert the text_reviews to type 'str' to avoid potential data type issues while parsing the data using beautiful soup. I create a cleaning function where I convert the reviews to lower case first, remove HTML tags(Beautiful soup), remove URLs and non-alphabetic characters(regular expressions), perform contractions(contractions), remove extra spaces(using join()) and finally apply this function to the text_review column of the dataframe. Also I calculate the average review lengths before and after cleaning(character wise).

3. Preprocessing

For preprocessing, I download the list of stopwords and the wordnetlemmatizer from Natural Language Toolkit(nltk) to create variables of those classes. The stopwords are removed and the text_reviews are lemmatized element-wise to each row in the column which has the cleaned reviews from earlier using a lambda function which serves the purpose of efficient text preprocessing. I calculate the average review lengths before and after preprocessing(character wise).

4. Feature Extraction

For feature extraction TF-IDF and BagofWords have been used with the help of TfidfVectorizer() and CountVectorizer() functions respectively. I have a used a maximum of 4000 features for both of these feature extraction methods to ensure consistency of number of features and efficiency of computation while calculating the TFIDF sparse matrix, and focusing on the most relevant terms in the text corpus.

5. Perceptron

For the perceptron model the hyperparameters used for training are the eta0(learning rate), max_iter to limit the maximum iterations, a random seed for consistent accuracy results and early stopping to stop training if the test error curve doesn't improve further. The model is then fit on the BOW features and TFIDF features and predicted on the the test set features to calculate precision, recall and F1 scores.

6. SVM

For SVM I used LinearSVC() to find a linear decision boundary between classes to train the model for faster results. The random state is fixed, max iterations are fixed. I also used the 'dual' hyperparameter to automatically select a choice between the primal and dual problem optimizations. The rest of the part is similar as done above in the Perceptron.

7. Logistic Regression

For this part I use the base LogisticRegression() model with hyperparameters of random seed and max_iterations. The rest is similar to the Perceptron model.

8. Naive Bayes

For this part I used the MultinomialNB() class from sklearn. The rest is similar to the Perceptron model.

Code in IPYNB format

```
In [ ]: '''Installing required packages: (Please uncomment the below lines to install the packages)
        !pip install nltk
        !pip install bs4
        !pip install contractions
        '''
```

```
In [ ]: import pandas as pd
        import numpy as np
        import nltk
        from sklearn.utils import resample
        from sklearn.model_selection import train_test_split
        import re
        # ! pip install bs4
        from bs4 import BeautifulSoup
        # ! pip install contractions
        import contractions
        from nltk.corpus import stopwords
        nltk.download('stopwords')
        from nltk.stem import WordNetLemmatizer
        nltk.download('wordnet')
        nltk.download('punkt')  # For tokenization
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.linear_model import Perceptron
        from sklearn.svm import LinearSVC
        from sklearn.linear_model import LogisticRegression
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import precision_score, recall_score, f1_score
        import warnings

        # Suppress all warnings
        warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\sahil\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\sahil\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sahil\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Read Data

```
In [ ]: # Load the dataset
        data = pd.read_csv('data.tsv', sep='\t', on_bad_lines='skip', index_col=False)

        # Make an independent copy of the DataFrame
        df = data.copy()
```

Keep Reviews and Ratings

```
In [ ]:  # Keep only 'Reviews' and 'Ratings' columns
        df = df[['star_rating', 'review_body']]
```

We form two classes and select 50000 reviews randomly from each class.

In [ ]:
```
# Create class labels: 1 for ratings 1, 2, 3; 2 for ratings 4, 5
df['sentiment_class'] = df['star_rating'].apply(lambda x: 1 if x in [1, 2, 3] else 2)

# Downsample to 50,000 reviews per class
class1_data = resample(df[df['sentiment_class'] == 1], n_samples=50000, random_state=42)
class2_data = resample(df[df['sentiment_class'] == 2], n_samples=50000, random_state=42)
compressed_data = pd.concat([class1_data, class2_data])

# Split dataset into training (80%) and testing (20%)
train_df, test_df = train_test_split(compressed_data, test_size=0.2, random_state=42)
```

Data Cleaning

Pre-processing

In [ ]:
```
train_df.dropna(inplace=True)
test_df.dropna(inplace=True)

train_df['review_body'] = train_df['review_body'].astype(str)
test_df['review_body'] = test_df['review_body'].astype(str)


def clean_and_preprocess_text(input_text):
    # Convert text to lowercase
    cleaned_text = input_text.lower()

    # Remove HTML tags
    cleaned_text = BeautifulSoup(cleaned_text, 'html.parser').get_text()

    # Remove URLs
    cleaned_text = re.sub(r'http\S+|www\S+|https\S+', '', cleaned_text, flags=re.MULTILINE)

    # Remove non-alphabetical characters (keep only letters and spaces)
    cleaned_text = re.sub(r'[^a-zA-Z\s]', '', cleaned_text)

    # Perform contractions (e.g., won't → will not)
    cleaned_text = contractions.fix(cleaned_text)

    # Remove extra spaces
    cleaned_text = ' '.join(cleaned_text.split())

    return cleaned_text

train_df['cleaned_review'] = train_df['review_body'].apply(clean_and_preprocess_text)
test_df['cleaned_review'] = test_df['review_body'].apply(clean_and_preprocess_text)
```

In [ ]:
```
# Function to calculate average review length in terms of character length
def calculate_average_review_length(reviews_list):
    # Calculate the length of each review in the list
    review_lengths = [len(review) for review in reviews_list]

    # Calculate the average review length
    if len(review_lengths) > 0:
        average_length = sum(review_lengths) / len(review_lengths)
    else:
        average_length = 0  # Handle the case when the list is empty

    return average_length

# Calculate average length before preprocessing
avg_length_before_clean = calculate_average_review_length(train_df['review_body'])
avg_length_before_test_clean = calculate_average_review_length(test_df['review_body'])

# Calculate average length after preprocessing
avg_length_after_clean = calculate_average_review_length(train_df['cleaned_review'])
avg_length_after_test_clean = calculate_average_review_length(test_df['cleaned_review'])

# Print the results
print(f"{avg_length_before_clean},{avg_length_after_clean}")
```

```
321.35705535276765,304.9250712535627
```

remove the stop words

In [ ]:
```
stop_words = set(stopwords.words('english'))

# Apply preprocessing to the 'Cleaned_Reviews' column
train_df['preprocessed_reviews'] = train_df['cleaned_review'].apply(lambda review: ' '.join([word for word in review.split() if word.lower() not in stop_words]))
test_df['preprocessed_reviews'] = test_df['cleaned_review'].apply(lambda review: ' '.join([word for word in review.split() if word.lower() not in stop_words]))
```

perform lemmatization

In [ ]:
```
lemmatizer = WordNetLemmatizer()

# Apply preprocessing to the 'Cleaned_Reviews' column
train_df['preprocessed_reviews'] = train_df['preprocessed_reviews'].apply(lambda review: ' '.join([lemmatizer.lemmatize(word) for word in review.split()]))
test_df['preprocessed_reviews'] = test_df['preprocessed_reviews'].apply(lambda review: ' '.join([lemmatizer.lemmatize(word) for word in review.split()]))
```

preprocessing reviews further

In [ ]:
```
# Function to calculate average review length in terms of character length
def average_review_length(reviews):
    lengths = [len(review) for review in reviews]
    return sum(lengths) / len(lengths)

# Calculate average length before preprocessing
avg_length_before = average_review_length(train_df['cleaned_review'])
avg_length_before_test = average_review_length(test_df['cleaned_review'])

# Calculate average length after preprocessing
avg_length_after = average_review_length(train_df['preprocessed_reviews'])
avg_length_after_test = average_review_length(test_df['preprocessed_reviews'])

# Print the results
print(f"{avg_length_before},{avg_length_after}")
```

```
304.9250712535627,189.73141157057853
```

TF-IDF and BoW Feature Extraction

In [ ]:
```
#TF-IDF Vectorizer
# Create a TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=4000)  # You can adjust max_features as needed

# Fit and transform the training data
X_train_tfidf = tfidf_vectorizer.fit_transform(train_df['preprocessed_reviews'])

# Transform the test data
X_test_tfidf = tfidf_vectorizer.transform(test_df['preprocessed_reviews'])


# BoW Vectorizer
# Create a BoW vectorizer
bow_vectorizer = CountVectorizer(max_features=4000)  # You can adjust max_features as needed

# Fit and transform the training data
X_train_bow = bow_vectorizer.fit_transform(train_df['preprocessed_reviews'])

# Transform the test data
X_test_bow = bow_vectorizer.transform(test_df['preprocessed_reviews'])
```

Perceptron Using Both Features

In [ ]:
```
# Create a Perceptron model
perceptron = Perceptron(eta0=0.001, random_state=42, max_iter=30000, early_stopping=True)

# Train the Perceptron model using BoW features
perceptron.fit(X_train_bow, train_df['sentiment_class'])

# Predict on the training data
test_preds_bow = perceptron.predict(X_test_bow)

# Calculate precision, recall, and f1-score for BoW
precision_bow = precision_score(test_df['sentiment_class'], test_preds_bow)
recall_bow = recall_score(test_df['sentiment_class'], test_preds_bow)
f1_score_bow = f1_score(test_df['sentiment_class'], test_preds_bow)

# Print results for BoW
print(precision_bow, recall_bow, f1_score_bow)

# Train the Perceptron model using TF-IDF features
perceptron.fit(X_train_tfidf, train_df['sentiment_class'])

# Predict on the training data
test_preds_tfidf = perceptron.predict(X_test_tfidf)

# Calculate precision, recall, and f1-score for TF-IDF
precision_tfidf = precision_score(test_df['sentiment_class'], test_preds_tfidf)
recall_tfidf = recall_score(test_df['sentiment_class'], test_preds_tfidf)
f1_score_tfidf = f1_score(test_df['sentiment_class'], test_preds_tfidf)

# Print results for TF-IDF
print(precision_tfidf, recall_tfidf, f1_score_tfidf)
```

```
0.7945716880234084 0.7847533632286996 0.7896320064173268
0.7998933901918976 0.7476831091180867 0.7729075457120782
```

SVM Using Both Features

In [ ]:
```
# Create an SVM model
svm_classifier = LinearSVC(random_state=42, max_iter=20000, dual='auto')

# Train the SVM model using BoW features
svm_classifier.fit(X_train_bow, train_df['sentiment_class'])

# Predict on the training data
test_preds_bow_svm = svm_classifier.predict(X_test_bow)

# Calculate precision, recall, and f1-score for BoW with SVM
precision_bow_svm = precision_score(test_df['sentiment_class'], test_preds_bow_svm)
recall_bow_svm = recall_score(test_df['sentiment_class'], test_preds_bow_svm)
f1_score_bow_svm = f1_score(test_df['sentiment_class'], test_preds_bow_svm)

# Print results for BoW with SVM
print(precision_bow_svm, recall_bow_svm, f1_score_bow_svm)

# Train the SVM model using TF-IDF features
svm_classifier.fit(X_train_tfidf, train_df['sentiment_class'])

# Predict on the training data
test_preds_tfidf_svm = svm_classifier.predict(X_test_tfidf)

# Calculate precision, recall, and f1-score for TF-IDF with SVM
precision_tfidf_svm = precision_score(test_df['sentiment_class'], test_preds_tfidf_svm)
recall_tfidf_svm = recall_score(test_df['sentiment_class'], test_preds_tfidf_svm)
f1_score_tfidf_svm = f1_score(test_df['sentiment_class'], test_preds_tfidf_svm)

# Print results for TF-IDF with SVM
print(precision_tfidf_svm, recall_tfidf_svm, f1_score_tfidf_svm)
```

```
0.8471170033670034 0.8022919780767315 0.8240953989456984
0.8351397036889728 0.8369706028898855 0.8360541509058331
```

Logistic Regression Using Both Features

In [ ]:
```
# Create a Logistic Regression model
logistic_regression = LogisticRegression(random_state=42, max_iter=20000)

# Train the Logistic Regression model using BoW features
logistic_regression.fit(X_train_bow, train_df['sentiment_class'])

# Predict on the training data
test_preds_bow_lr = logistic_regression.predict(X_test_bow)

# Calculate precision, recall, and f1-score for BoW with Logistic Regression
precision_bow_lr = precision_score(test_df['sentiment_class'], test_preds_bow_lr)
recall_bow_lr = recall_score(test_df['sentiment_class'], test_preds_bow_lr)
f1_score_bow_lr = f1_score(test_df['sentiment_class'], test_preds_bow_lr)

# Print results for BoW with Logistic Regression
print(precision_bow_lr, recall_bow_lr, f1_score_bow_lr)

# Train the Logistic Regression model using TF-IDF features
logistic_regression.fit(X_train_tfidf, train_df['sentiment_class'])

# Predict on the training data
test_preds_tfidf_lr = logistic_regression.predict(X_test_tfidf)

# Calculate precision, recall, and f1-score for TF-IDF with Logistic Regression
precision_tfidf_lr = precision_score(test_df['sentiment_class'], test_preds_tfidf_lr)
```

```
recall_tfidf_lr = recall_score(test_df['sentiment_class'], test_preds_tfidf_lr)
f1_score_tfidf_lr = f1_score(test_df['sentiment_class'], test_preds_tfidf_lr)

# Print results for TF-IDF with Logistic Regression
print(precision_tfidf_lr, recall_tfidf_lr, f1_score_tfidf_lr)
```

```
0.847568578553616 0.8128550074738415 0.8298489241568747
0.8362492628268134 0.8478325859491779 0.8420010886238806
```

Naive Bayes Using Both Features

In [ ]:

```
# Create a Multinomial Naive Bayes model
naive_bayes = MultinomialNB()

# Train the Naive Bayes model using BoW features
naive_bayes.fit(X_train_bow, train_df['sentiment_class'])

# Predict on the training data
test_preds_bow_nb = naive_bayes.predict(X_test_bow)

# Calculate precision, recall, and f1-score for BoW with Naive Bayes
precision_bow_nb = precision_score(test_df['sentiment_class'], test_preds_bow_nb)
recall_bow_nb = recall_score(test_df['sentiment_class'], test_preds_bow_nb)
f1_score_bow_nb = f1_score(test_df['sentiment_class'], test_preds_bow_nb)

# Print results for BoW with Naive Bayes
print(precision_bow_nb, recall_bow_nb, f1_score_bow_nb)

# Train the Naive Bayes model using TF-IDF features
naive_bayes.fit(X_train_tfidf, train_df['sentiment_class'])

# Predict on the training data
test_preds_tfidf_nb = naive_bayes.predict(X_test_tfidf)

# Calculate precision, recall, and f1-score for TF-IDF with Naive Bayes
precision_tfidf_nb = precision_score(test_df['sentiment_class'], test_preds_tfidf_nb)
recall_tfidf_nb = recall_score(test_df['sentiment_class'], test_preds_tfidf_nb)
f1_score_tfidf_nb = f1_score(test_df['sentiment_class'], test_preds_tfidf_nb)

# Print results for TF-IDF with Naive Bayes
print(precision_tfidf_nb, recall_tfidf_nb, f1_score_tfidf_nb)
```

```
0.8405172413793104 0.7384155455904334 0.7861651901755875
0.8199210446401458 0.8071748878923767 0.8134980415787888
```