Institute of Technology of Cambodia

Department Information Technology and Communication

Natural Language Processing

Mini Project2: Text Classification

Lecturer: Dona VALY

Group: 07

Team Members

SOR Sopheak	20191066
SUN Vysing	20191124
THUE Naratt	20191186
TY Sophearum	20191219
VEN Thon	20191250

2023 ~ 2024

Contents

I. INTRODUCTION

Text classification using Natural Language Processing (NLP) is project focuses on techniques to predict whether product reviews are positive or negative. The goal is to build a text classifier using the top 80% of each file as the training set and the remaining 20% as the test set. Multiple classification models will be implemented and the objective is to develop a robust text classifier that can distinguish between positive and negative reviews. The provided corpus consists of two text files, each containing a substantial number of reviews. The classification models will be trained on the majority of the data and evaluated on the remaining portion.

II. PROPOSED METHODOLOGY AND ALGORITHM

1. Technology

In this training model we are use Python programming language with google Co lab.

2. Machines Algorithms

We are employing various algorithms for testing purposes, including:

- Logistic Regression.
- Naïve bayes
- K-Nearest Neighbor
- Random forest

3. Conception

Initially, we commence the project by contemplating the methodology for executing the solution. We break down the implementation into six specific segments, as outlined below:

- Import the required libraries
- Import and read the text files
- Remove new line characters from a list of strings
- Convert a list of strings to lowercase
- Tokenize a list of strings using NLTK
- Tokenize and remove stop words
- Extract and combine feature extraction
- Train the dataset
- Define the five text classifier models

III. Implementation

For clarity, we will detail each part mentioned in the conceptual introduction.

Step1: Import the required libraries

Sets up a foundation for conducting machine learning tasks. It includes tools for mathematical computations, natural language processing, and machine learning models such as Logistic Regression, Naive Bayes, K-Nearest Neighbors, and Random Forest. The NLTK library is particularly useful for text processing tasks, such as tokenization and stop-word removal, which are common in natural language processing projects. The scikit-learn library is used for machine learning tasks, providing tools for data splitting, model evaluation, and various classification algorithms.

```
import math
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('stopwords')

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
```

Figure 1: Import Libraries

Step2: Import and read the text files

Both functions (readfile & readword) read the content of a file specified by its filename, remove unnecessary whitespaces from each line, and return a list of lines. The first function uses UTF-8 encoding, while the second uses ISO-8859-1 encoding.

```
# Read file and return line
def readfile(fn):
  with open(fn, 'r', encoding='utf-8') as f:
    lines = [line.strip() for line in f.readlines()]
  return lines


# Read word from file
def readword(fn):
  with open(fn, 'r', encoding='ISO-8859-1') as f:
    lines = [line.strip() for line in f.readlines()]
  return lines
```

Figure 2: Read the text file

Step3: Remove new line characters from a list of strings and convert a list of strings to lowercase

These functions "remove_newline_char" address common tasks in text processing removing newline characters for cleaner string representation and converting text by function "convert_to_lowercase" to lowercase for standardized analysis.

```
# Remove new line characters from a list of strings
def remove_newline_chars(lines):
    return [line[:-1] for line in lines]

# Convert a list of strings to lowercase
def convert_to_lowercase(lines):
    return [line.lower() for line in lines]
fn = r'/content/gdrive/MyDrive/project2_NLP/negative-words.txt'
line = readword(fn)
test =  convert_to_lowercase(line)
print(test)

['2-faced', '2-faces', 'abnormal', 'abolish', 'abominable', 'abominably', 'abominate', 'abomination', 'abort',
```

Figure 3: Remove new characters&

Step4: Tokenize a list of strings using NLTK

Tokenization is the process of breaking down text into individual units, or tokens. In this case, the tokens are likely words. This function is designed to take a list of strings, tokenize each string into a list of words using NLTK's word_tokenize, and return a list of tokenized sequences.

```
# Tokenize a list of strings using NLTK
def tokenize_reviews(lines):
    return [word_tokenize(line) for line in lines]
```

Figure 4: Tokenize List using NLTK

Step5: Tokenize and remove stop words

- For this function called preprocess_and_tokenize for tokenizing and removing stop words from a list of reviews. It uses the NLTK library for natural language processing, specifically importing functions for stop words and word tokenization. The function converts each review to lowercase, tokenizes it into words, and then removes common English stop words before storing the processed reviews in a list.

```
# Tokenize and remove stop words
def preprocess_and_tokenize(reviews):
    stop_words = set(stopwords.words('english'))
    tokenized_reviews = []
    for review in reviews:
        tokens = word_tokenize(review.lower())
        # Remove stop words
        tokens_without_stopwords = [word for word in tokens if word.lower() not in stop_words]
        tokenized_reviews.append(tokens_without_stopwords)
    return tokenized_reviews
```

Figure 5: Tokenize and remove stop words

- This algorithm is missing the definitions for the readfile and readword functions. To reads positive and negative reviews from files (positive-reviews.txt and negative-reviews.txt), as well as positive and negative words from other files (positive-words.txt and negative-words.txt). It utilizes placeholder functions (readfile and readword) to read the content from these files. Ensure that file paths are accurate and the files are formatted as expected.

```
# Read your positive and negative reviews
pos_reviews = readfile('positive-reviews.txt')
neg_reviews = readfile('negative-reviews.txt')

# Read positive and negative words
pos_words = readword('positive-words.txt')
neg_words = readword('negative-words.txt')
```

Figure 6: Read positive, negative word and reviews

- Cleans and preprocesses positive and negative reviews by removing newline characters and converting them to lowercase. Placeholder functions (remove_newline_chars and convert_to_lowercase) are provided for these operations, and you can customize them as needed.

```
# Clean and preprocess positive and negative reviews
pos_reviews = remove_newline_chars(pos_reviews)
neg_reviews = remove_newline_chars(neg_reviews)
pos_reviews = convert_to_lowercase(pos_reviews)
neg_reviews = convert_to_lowercase(neg_reviews)
```

Figure 7: Clean Process Positive and Negative Reviews

- After that It utilizes a placeholder function tokenize_reviews that employs NLTK's word_tokenize for each review. The tokenized reviews are stored in pos_tokenized and neg_tokenized lists. Adjust the tokenization function as needed for specific requirements.

```
# Tokenize positive and negative reviews
pos_tokenized = tokenize_reviews(pos_reviews)
neg_tokenized = tokenize_reviews(neg_reviews)
```

Figure 8: Tokenize Positive and Negative Reviews

- And removes stop words from positive and negative reviews using the preprocess_and_tokenize function. It then prints the first two tokenized reviews for both positive and negative categories. The code aims to showcase the processed reviews. Ensure that the preprocess_and_tokenize function is properly defined with appropriate imports and runs successfully.

```python
# Tokenize and remove stop words
pos_tokenized_no_stopwords = preprocess_and_tokenize(pos_reviews)
neg_tokenized_no_stopwords = preprocess_and_tokenize(neg_reviews)

# Print the first two tokenized reviews
print("Tokenized Positive Reviews:")
print(pos_tokenized[:2])
print("\nTokenized Negative Reviews:")
print(neg_tokenized[:2])
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

Figure 9: Remove stop words

Step6: Extract and combine feature extraction

- With this function extract_features for feature extraction from tokenized reviews. Features include counts of positive and negative words, presence of 'no', count of selected pronouns, presence of exclamation mark, review length, and log-transformed review lengt and then extracts features for both positive and negative reviews using this function and prints the features for the first two reviews in each category.For Ensure that tokenized reviews and word lists are correctly provided for feature extraction.

```
Positive Review Features:
[[33928, 2040, 0, 0, 1, 12.119691978691984]]

Negative Review Features:
[[5499, 16268, 0, 0, 1, 11.89407738849611]]
```

Figure 10: Extract and combine feature extraction

- The printed positive review features represent a review with 33,928 positive words, 2,040 negative words, no occurrences of 'no', no pronouns, one exclamation mark, and a log-transformed review length of approximately 12.12.
- The negative review features represent a review with 5,499 positive words, 16,268 negative words, no occurrences of 'no', no pronouns, one exclamation mark, and a log-transformed review length of approximately 11.89.

Step8: Train the dataset

- ❖ Logistic Regression.

The logistic regression model aims to predict whether a given review is positive or negative based on the extracted features. So, we are going to trains a logistic regression model on a dataset consisting of features and labels extracted from positive and negative reviews. The model is evaluated on a testing set, and the accuracy is printed.The model is fitted on the training set (X_train, y_train), and then evaluated on the testing set (X_test, y_test) fo find with accuracy of the logistic regression model on the testing set.

```
# Train a logistic regression model
model_logistic = LogisticRegression()
model_logistic.fit(X_train, y_train)

# Evaluate the model on the testing set
accuracy_logistic = model_logistic.score(X_test, y_test)
print(f"Logistic Accuracy: {accuracy_logistic:.2%}")
```

```
Logistic Accuracy: 81.51%
```

Figure11: Logistic Regression Medel Training and Accuracy

- ❖ Naïve bayes

For this algorithm we are creates a Multinomial Naive Bayes classifier (nb_classifier), trains it on the training data (X_train, y_train), makes predictions on the test set (X_test), and evaluates its accuracy. Additionally, the code prints a classification report containing precision, recall, and F1-score. The Multinomial Naive Bayes classifier achieved an accuracy of 81.47% on the test set. The classification report provides additional metrics:

- Precision: 77% for class 0, 87% for class 1
- Recall: 89% for class 0, 73% for class 1
- F1-score: 83% for class 0, 80% for class 1
- Support: 4607 instances for class 0, 4567 instances for class 1

```python
# Create a Multinomial Naive Bayes classifier
nb_classifier = MultinomialNB()

# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy_NB = accuracy_score(y_test, y_pred)
print(f"Naive Bayes Accuracy: {accuracy_NB:.2%}")

# Display additional evaluation metrics (optional)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Naive Bayes Accuracy: 81.47%

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.89      0.83      4607
           1       0.87      0.73      0.80      4567

    accuracy                           0.81      9174
   macro avg       0.82      0.81      0.81      9174
weighted avg       0.82      0.81      0.81      9174
```

Figure 11: Naive Bayes Model and accuracy

- K-Nearest Neighbor

KNN is a non-parametric classification algorithm that assigns labels based on the majority class among its k nearest neighbors in the feature space. For k-Nearest Neighbors (KNN)model classifier (knn_classifier) with k=3, trains it on the training data (X_train, y_train), makes predictions on the test data (X_test), and evaluates its accuracy. The classifier is evaluated on a test set, and its accuracy is printed.

```python
# Create a KNN classifier with k=3
knn_classifier = KNeighborsClassifier(n_neighbors=3)
# Train the classifier on the training data
knn_classifier.fit(X_train, y_train)
# Make predictions on the test data
y_pred = knn_classifier.predict(X_test)
# Evaluate the accuracy of the classifier
accuracy_knn = accuracy_score(y_test, y_pred)
print(f"KNN Accuracy: {accuracy_knn:.2%}")
```

```
KNN Accuracy: 74.25%
```

Figure 12: KNN Model and accuracy

❖ Random forest

The Random Forest classifier, with 100 estimators, achieved an accuracy of 81.92% on the test set. The confusion matrix provides a breakdown of its performance, showing a balance between true positives (3429) and true negatives (4086), with some false positives (521) and false negatives (1138). Overall, the classifier demonstrates effectiveness in predicting both positive and negative classes.

The confusion matrix provides additional insights into the classifier's performance:

- True Positives (TP): 3429
- True Negatives (TN): 4086
- False Positives (FP): 52
- False Negatives (FN): 1138

```
# Create a random forest classifier
model_random_forest = RandomForestClassifier(n_estimators=100)

# Train the classifier on the training data
model_random_forest.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model_random_forest.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy_rf = accuracy_score(y_test, y_pred)
print(f'Random Forest Accuracy: {accuracy_rf:.2%}')

# Display the confusion matrix
confusion_matrix(y_test, y_pred)
```

```
Random Forest Accuracy: 81.92%
array([[4086,  521],
       [1138, 3429]])
```

Figure 13: Random Forest Model and accuracy

Step9: Accuracy score comparison

```
import matplotlib.pyplot as plt # for data visualization
# Create a bar chart
plt.bar(range(4), [accuracy_logistic, accuracy_knn, accuracy_NB, accuracy_rf], tick_label=['accuracy_logistic', 'accuracy_knn', 'accuracy_nb', 'accuracy_rf'])
plt.xticks(rotation=90)
plt.xlabel('Models')
plt.ylabel('Percentage')
plt.title('Accuracy score comparison')
plt.show()
```

Figure 14: Plot Graph

We are uses Matplotlib to create a bar chart comparing the accuracy scores of four models: logistic regression, k-Nearest Neighbors (KNN), Naive Bayes (NB), and Random Forest (RF). The chart visually represents the performance of each model in terms of accuracy, with the x-axis indicating the models and the y-axis indicating the percentage accuracy.

IV. Conclusion

Throughout the process of cleaning data, splitting the dataset into training and testing set, training data through each model, we could find out the best model for our data with the highest accuracy compared to other models. For our dataset, random forest algorithms are the most suitable model for giving high rate of accuracy.

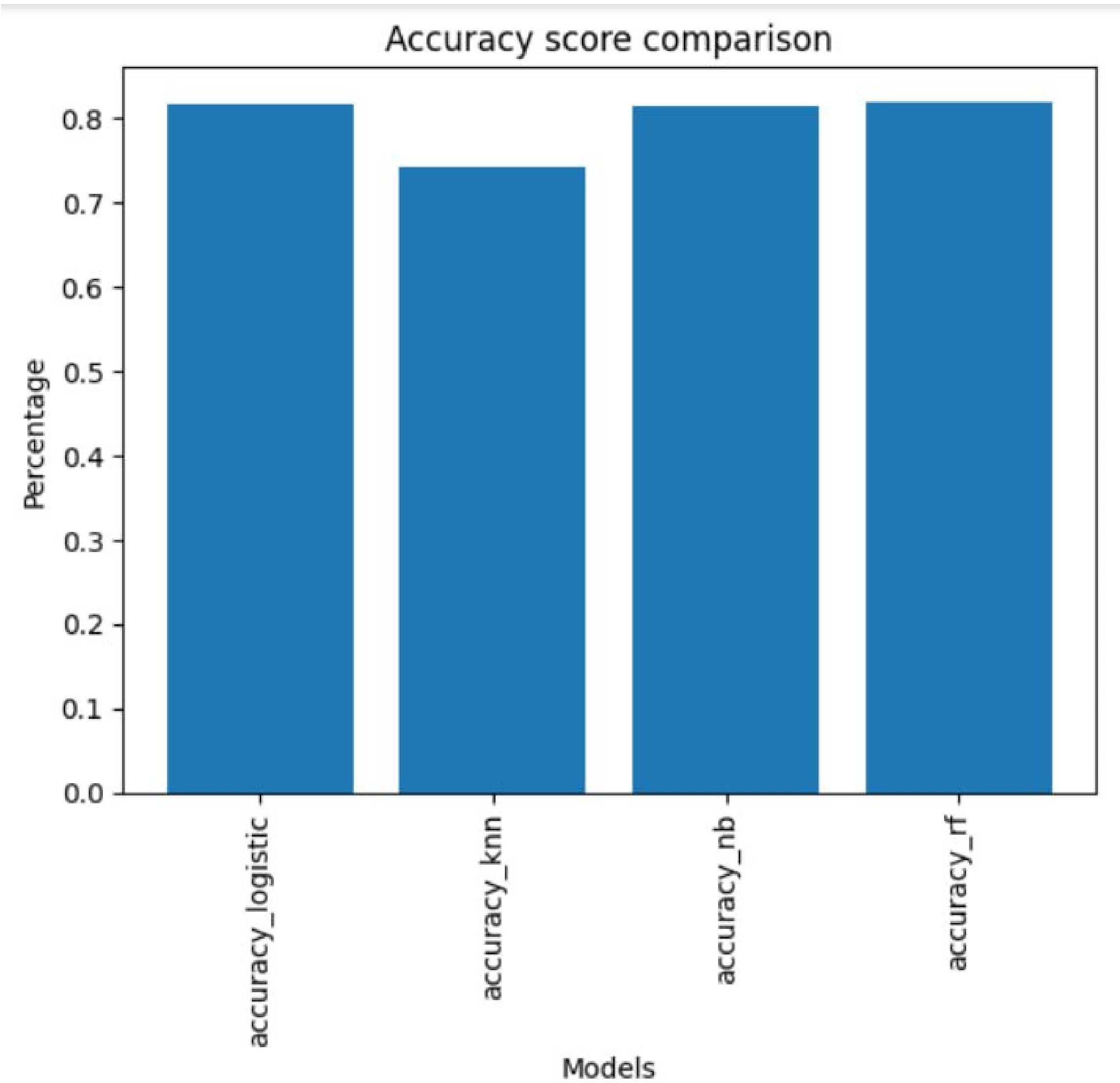

Figure 15: Plot Graph Compare Model