Department of Computer Engineering
University of Peradeniya
CO544: Machine Learning and Data Mining

Lab 05: Text Classification and Performance Analysis

July 08, 2025

Objectives

The aim of the lab is to provide students hands-on experience in text classification using Python.

At the end of the lab, students should be able to:

- Preprocess a given text dataset.
- Develop classification models using different machine learning algorithms and compare performance of the models using multiple performance measures.

1. Introduction

Text classification is the process of classifying text strings or documents into different categories, depending upon the content of the document. Detecting user sentiment from a tweet, classifying an email as spam or ham, automatic tagging of customer queries, or classifying news articles into different categories like Politics, Stock Market, Sports, etc. are some of the real world applications of text classification.

We can complete this task with the use of Natural Language Processing (NLP) and classification algorithms. NLP enables computers to understand and interpret human languages.

In this lab, you will perform a simple text classification task: classifying movie reviews as either positive or negative based on their content. You will use the `movie_reviews` dataset, which consists of two categories: `pos` for positive reviews and `neg` for negative reviews. Each document in the dataset is a movie review written in plain text, and the goal is to train a classifier that can automatically predict the sentiment of a given review as positive or negative. This dataset is commonly used as a benchmark for sentiment analysis tasks.

2. Text Classification

(a) Importing required modules

```
import re   # Regular expressions
from sklearn.datasets import load_files  # For loading dataset folders
import nltk  # Natural Language Toolkit

from nltk.corpus import stopwords  # Stop words
from nltk.stem import WordNetLemmatizer  # Lemmatization

# Download required NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')
```

(b) Loading data

Before you run your code, make sure you have the dataset properly organized. First, download the `movie_reviews.zip` file provided for this lab. Extract its contents. Inside the extracted folder, you will find a subfolder named `movie_reviews`.

Important: Move the entire `movie_reviews` folder so that it is in the same directory as your Python source code file. Your folder structure should look like this:

```
project_folder/
  your_script.py
  movie_reviews/
     pos/
     neg/
```

In this structure:

- The `movie_reviews` folder contains two subfolders: `pos` and `neg`.
- Each subfolder (`pos` and `neg`) represents a separate class label.

When you run `load_files("movie_reviews")`, scikit-learn will automatically treat each subfolder name as a separate category.

- The variable `X` will contain the raw text data for each file as bytes.
- The variable `y` will contain the target labels as integers. For example, if you have two subfolders named `neg` and `pos`, they will be labeled numerically in alphabetical order: `0` for `neg` and `1` for `pos`.

Later steps will decode the text, preprocess it, and use these labels for training and evaluation.

```
# Instantiate lemmatizer (needed for later)
lemmatizer = WordNetLemmatizer()

# movie_data = load_files(r"txt_sentoken")
movie_data = load_files(r"movie_reviews")
X , y = movie_data.data, movie_data.target
```

You can use the following to check the summary of the dataset.

```
# Show basic summary information
print(f"Number of documents: {len(X)}")
print(f"Number of labels: {len(y)}")
print(f"Target names (classes): {movie_data.target_names}")

# Show a sample file (before decoding)
print("\nFirst document (raw bytes):")
print(X[0][:500])  # show first 500 bytes

# Decode and print a preview
print("\nFirst document (decoded):")
print(X[0].decode('utf-8')[:500])  # show first 500 characters

# Check label of first document
print(f"\nLabel of first document: {y[0]}")
```

(c) Data preprocessing

`X = movie_data.data` is a list. Here we preprocess data in each row using a loop and later combine results into one list.

```
documents = []
for i in range(len(X)):
    # 1. Decode from bytes to string
    document = X[i].decode('utf-8')

    # # you can add a small check as follows.
    # print(X[0])               # Before decoding (bytes)
    # print(X[0].decode('utf-8'))  # After decoding


    # 2. Apply your regex substitutions
    document = re.sub(r'\W', ' ', document)  # remove special characters
    document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)  # single chars at beginning
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)  # single chars in middle
    document = re.sub(r'\d+', '', document)  # remove numbers
    document = re.sub(r'\s+', ' ', document, flags=re.I)  # multiple spaces to one

    # 3. Lowercase
    document = document.lower()

    # 4. Tokenize
    document = document.split()

    # 5. Lemmatize
    document = [lemmatizer.lemmatize(word) for word in document]

    # 6. Rejoin tokens if needed (optional)
    document = ' '.join(document)

    # 7. Append to new list
    documents.append(document)
```

TASK 1: Find data preprocessing steps other than mentioned above.

(d) Convert text into numbers

There are different approaches to convert text into the corresponding numerical form. The Bag of Words Model and the Word Embedding Model are two of the most commonly used approaches. Here I have used the Bag of Words model.

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords

vectorizer = CountVectorizer(
    max_features=1500,
    min_df=7,
    max_df=0.8,
    stop_words=stopwords.words('english')
)
```

```
X_vectors = vectorizer.fit_transform(documents).toarray()

# To check the shape and vocabulary:
print(X_vectors.shape)  # (number_of_documents, number_of_features)
print(vectorizer.get_feature_names_out())  # List of feature words
```

TASK 2: Discuss advantages and disadvantages of the Bag of Words model.

(e) Text Classification

Now classification algorithms can be applied.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_vectors, y, test_size=0.2,
    random_state=0)

# Logistic Regression model
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

predictions = log_reg.predict(X_test)
```

(f) Evaluating Model Performance

To evaluate a classifier thoroughly, use multiple metrics such as confusion matrix, precision, recall, and F1-score.

Below is an example of how to compute these measures using scikit-learn:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))

print("\nClassification Report:")
print(classification_report(y_test, predictions))

print("\nAccuracy:")
print(accuracy_score(y_test, predictions))
```

TASK 3: Train a Random Forest model, a Support Vector Machine model and a Naive Bayesian classifier. Compare the accuracies and other performance measures (precision, recall, F1-score, confusion matrix) of all four models including the Logistic Regression model. What is the best model? Justify your answer based on these measures.

Note: It is important to evaluate classification models using multiple performance measures, such as precision, recall, F1-score, and confusion matrix, as accuracy alone may not provide enough insight, especially for imbalanced datasets.

Submission

Submit:

- Python file(s) and/or Python Notebook(s) used to train all four models in Task 3.
- A single PDF containing answers to the questions in all three tasks.

Place all files in a folder, compress it, and name it `eyyxxxlab5`, where `yyxxx` is your registration number.