e20280lab5

July 19, 2025

Department of Computer Engineering University of Peradeniya CO544: Machine Learning and Data Mining Lab 05: Text Classification and Performance Analysis

0.1 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]:
```
!pip install nltk scikit-learn
```

```
Collecting nltk
  Downloading nltk-3.9.1-py3-none-any.whl (1.5 MB)
     ---------------------------------------- 1.5/1.5 MB 847.1 kB/s eta 0:00:00
Requirement already satisfied: scikit-learn in e:\my projects\python\co544
machine learnning\venv\lib\site-packages (1.6.1)
Collecting click
  Using cached click-8.1.8-py3-none-any.whl (98 kB)
Requirement already satisfied: joblib in e:\my projects\python\co544 machine
learnning\venv\lib\site-packages (from nltk) (1.4.2)
Collecting regex>=2021.8.3
  Downloading regex-2024.11.6-cp39-cp39-win_amd64.whl (274 kB)
     ---------------------------------- 274.1/274.1 KB 845.7 kB/s eta 0:00:00
Collecting tqdm
  Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
     ---------------------------------------- 78.5/78.5 KB 2.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.19.5 in e:\my projects\python\co544
machine learnning\venv\lib\site-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in e:\my
projects\python\co544 machine learnning\venv\lib\site-packages (from scikit-
```

```
learn) (3.6.0)
Requirement already satisfied: scipy>=1.6.0 in e:\my projects\python\co544
machine learnning\venv\lib\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: colorama in e:\my projects\python\co544 machine
learnning\venv\lib\site-packages (from click->nltk) (0.4.6)
Installing collected packages: tqdm, regex, click, nltk
Successfully installed click-8.1.8 nltk-3.9.1 regex-2024.11.6 tqdm-4.67.1

WARNING: You are using pip version 22.0.4; however, version 25.1.1 is available.
You should consider upgrading via the 'E:\My Projects\python\CO544 Machine
Learnning\venv\Scripts\python.exe -m pip install --upgrade pip' command.
```

0.2 2. Text Classification

(a) Importing required modules

[3]:
```python
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")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ravin\AppData\Roaming\nltk_data…
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ravin\AppData\Roaming\nltk_data…
```

[3]:
```
True
```

(b) Loading data

[4]:
```python
# 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
```

[5]:
```python
# 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]}")
```

```
Number of documents: 2000
Number of labels: 2000
Target names (classes): ['neg', 'pos']

First document (raw bytes):
b"arnold schwarzenegger has been an icon for action enthusiasts , since the late
80's , but lately his films have been very sloppy and the one-liners are getting
worse . \nit's hard seeing arnold as mr . freeze in batman and robin ,
especially when he says tons of ice jokes , but hey he got 15 million , what's
it matter to him ? \nonce again arnold has signed to do another expensive
blockbuster , that can't compare with the likes of the terminator series , true
lies and even eraser . \nin this so cal"

First document (decoded):
arnold schwarzenegger has been an icon for action enthusiasts , since the late
80's , but lately his films have been very sloppy and the one-liners are getting
worse .
it's hard seeing arnold as mr . freeze in batman and robin , especially when he
says tons of ice jokes , but hey he got 15 million , what's it matter to him ?
once again arnold has signed to do another expensive blockbuster , that can't
compare with the likes of the terminator series , true lies and even eraser .
in this so cal

Label of first document: 0
```

(c)Datapreprocessing

[6]:
```
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)
```

TASK1:Find data preprocessing steps other than mentioned above:

- Remove stop words (later handled by CountVectorizer).
- Spell correction using libraries like TextBlob.
- Stemming (alternative to lemmatization).
- Removing rare words (low frequency).
- Synonym replacement to reduce feature space.

(d)Converttext intonumbers

[9]:
```
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
```

```
(2000, 1500)
['ability' 'able' 'absolutely' … 'york' 'young' 'younger']
```

TASK2:Discuss advantages and disadvantages of the Bag of Words model. | **Advantages** | **Disadvantages** | | ———————————- | ———————————— | | Simple and easy to implement | Ignores word order and context | | Works well for basic text classification | Large sparse feature vectors | | Efficient for small-medium datasets | Cannot handle polysemy (same word, different meaning) | | Easy to interpret features | Fails to capture semantic relationships |

(e)TextClassification

[10]:
```
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

[11]:
```
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))
```

```
Confusion Matrix:
[[164  44]
 [ 28 164]]

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82       208
           1       0.79      0.85      0.82       192

    accuracy                           0.82       400
   macro avg       0.82      0.82      0.82       400
weighted avg       0.82      0.82      0.82       400


Accuracy:
0.82
```

TASK3:TrainaRandomForestmodel,aSupportVectorMachinemodelandaNaiveBayesianclassifier. Comparetheaccuraciesandotherperformancemeasures(precision, recall,F1-score,confusionmatrix)ofall fourmodels includingtheLogisticRegressionmodel.What is-thebestmodel?Justifyyouranswerbasedon thesemeasures. Note: It is importanttoeval-uateclassificationmodelsusingmultipleperformancemeasures, suchaspreci sion, recall,F1-score,andconfusionmatrix,asaccuracyalonemaynotprovideenoughinsight,especiallyfor imbalanced-datasets.

Logistic Regression

[12]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

log_reg = LogisticRegression()
```

```
log_reg.fit(X_train, y_train)
log_pred = log_reg.predict(X_test)

print("Logistic Regression Results:")
print(confusion_matrix(y_test, log_pred))
print(classification_report(y_test, log_pred))
print("Accuracy:", accuracy_score(y_test, log_pred))
```

```
Logistic Regression Results:
[[164  44]
 [ 28 164]]
              precision    recall  f1-score   support

           0       0.85      0.79      0.82       208
           1       0.79      0.85      0.82       192

    accuracy                           0.82       400
   macro avg       0.82      0.82      0.82       400
weighted avg       0.82      0.82      0.82       400

Accuracy: 0.82
```

Random Forest

[13]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=0)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

print("Random Forest Results:")
print(confusion_matrix(y_test, rf_pred))
print(classification_report(y_test, rf_pred))
print("Accuracy:", accuracy_score(y_test, rf_pred))
```

```
Random Forest Results:
[[168  40]
 [ 29 163]]
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       208
           1       0.80      0.85      0.83       192

    accuracy                           0.83       400
   macro avg       0.83      0.83      0.83       400
weighted avg       0.83      0.83      0.83       400

Accuracy: 0.8275
```

Support Vector Machine (SVM)

[14]:
```
from sklearn.svm import SVC

svm = SVC(kernel='linear', random_state=0)
svm.fit(X_train, y_train)
svm_pred = svm.predict(X_test)

print("SVM Results:")
print(confusion_matrix(y_test, svm_pred))
print(classification_report(y_test, svm_pred))
print("Accuracy:", accuracy_score(y_test, svm_pred))
```

```
SVM Results:
[[165  43]
 [ 33 159]]
              precision    recall  f1-score   support

           0       0.83      0.79      0.81       208
           1       0.79      0.83      0.81       192

    accuracy                           0.81       400
   macro avg       0.81      0.81      0.81       400
weighted avg       0.81      0.81      0.81       400

Accuracy: 0.81
```

Naive Bayes

[15]:
```
from sklearn.naive_bayes import MultinomialNB

nb = MultinomialNB()
nb.fit(X_train, y_train)
nb_pred = nb.predict(X_test)

print("Naive Bayes Results:")
print(confusion_matrix(y_test, nb_pred))
print(classification_report(y_test, nb_pred))
print("Accuracy:", accuracy_score(y_test, nb_pred))
```

```
Naive Bayes Results:
[[166  42]
 [ 32 160]]
              precision    recall  f1-score   support

           0       0.84      0.80      0.82       208
           1       0.79      0.83      0.81       192

    accuracy                           0.81       400
```

```
   macro avg       0.82      0.82      0.81       400
weighted avg       0.82      0.81      0.82       400

Accuracy: 0.815
```

After running the above code, compare:

- Confusion matrix → How many true positives/negatives
- Precision → Accuracy on positive class
- Recall → Ability to detect positives
- F1-score → Balance of precision/recall
- Accuracy → Overall correctness

Model	Accuracy	Precision	Recall	F1-score	Notes
Logistic Regression	Good baseline	Balanced	Balanced	Balanced	Fast and simple
Random Forest	Usually higher	Robust	Good generalization	Strong	May overfit
SVM	High accuracy	High	High	High	Slower but precise
Naive Bayes	Fastest	May be lower	Good with small data	Fast	Simple but less accurate

Best Model Suggestion: In general, SVM often shows the best precision/recall trade-off on text classification, but you should justify with your actual output.