Lab 5 – Text Classification

Due Date: Feb 09, 05:00PM

Overview

In this lab, we will build a machine learning pipeline to classify news articles based on their relevance to the US Economy. We use a dataset of approximately 8,000 articles tagged as "relevant" or "not relevant".

Objectives

- Perform data exploration and cleaning on text datasets.
- Implement custom text preprocessing functions.
- Convert raw text into numerical features using the Bag of Words model.
- Train and evaluate different text classifier.

Lab Instructions

Step 0:

1. Create a jupyter notebook file called lab04.ipynb
2. For each step below, create a code cell in your notebook to write your code inside.
3. Create the first cell and install the followings

```
# Required libraries
# pip install numpy pandas scikit-learn matplotlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import string
import itertools

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import _stop_words
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, roc_auc_score
```

Step 1: Data Loading and Exploration

1. Reading Data

```
# Load the dataset
```

```
our_data = pd.read_csv("dataset/Full-Economic-News-DFE-839861.csv",
encoding="ISO-8859-1")

# Display initial shape and class distribution
print(f"Dataset Shape: {our_data.shape}")
print(our_data["relevance"].value_counts() / our_data.shape[0])
```

2. Data Cleaning: We remove "not sure" labels and map the target to numerical values: yes (Relevant) to 1 and no (Not Relevant) to 0.

```
# Filter and map labels
our_data = our_data[our_data.relevance != "not sure"]
our_data['relevance'] = our_data.relevance.map({'yes': 1, 'no': 0})

# Keep only the necessary columns
our_data = our_data[["text", "relevance"]]
print(f"Cleaned Shape: {our_data.shape}")
```

Step 2: Text Preprocessing

Raw text often contains noise like HTML tags, punctuation, and numbers. We define a clean function to handle these.

1. Remove unnecessary tokens

```
stopwords = _stop_words.ENGLISH_STOP_WORDS

def clean(doc):
    # Remove HTML line breaks
    doc = doc.replace("</br>", " ")
    # Remove punctuation and numbers
    doc = "".join([char for char in doc if char not in
string.punctuation and not char.isdigit()])
    # Remove stopwords and tokenize
    doc = " ".join([token for token in doc.split() if token not in
stopwords])
    return doc
```

Step 3: Modeling Pipeline

The modeling process follows these steps:

1. **Split**: Divide data into training (75%) and test sets (25%).
2. **Vectorize**: Convert text to feature vectors using CountVectorizer.
3. **Train**: Fit a Naive Bayes model.
4. **Evaluate**: Test performance on unseen data.

1. Train-Test Split:

```
X = our_data.text
y = our_data.relevance

X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=1)
```

2. Vectorization and Training: We use the custom clean function within the CountVectorizer to transform our text.

```
# Instantiate and fit Vectorizer
vect = CountVectorizer(preprocessor=clean)
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)

# Train Multinomial Naive Bayes
nb = MultinomialNB()
%time nb.fit(X_train_dtm, y_train)

# Make predictions
y_pred_class = nb.predict(X_test_dtm)
```

Step 4: Evaluation

To understand model performance, we look at accuracy, the ROC-AUC score, and the confusion matrix.

1. Performance Metrics

```
print("Accuracy: ", accuracy_score(y_test, y_pred_class))
print("ROC_AUC_Score: ", roc_auc_score(y_test, y_pred_class))
```

2. Visualization: Confusion Matrix

```
def plot_confusion_matrix(cm, classes, title='Confusion matrix',
cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
```

```
                horizontalalignment="center",
                color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Plotting
conf_mat = confusion_matrix(y_test, y_pred_class)
plot_confusion_matrix(conf_mat, classes=['Not Relevant', 'Relevant'])
plt.show()
```

Step 5: Logistic Regression Implementation

Logistic Regression is effective for text because it handles sparse, high-dimensional feature vectors (like our Bag of Words) well.

1. Train and Evaluate Logistic Regression Model

```
# Step 4: Train Logistic Regression
logreg = LogisticRegression()
%time logreg.fit(X_train_dtm, y_train)

# Predict
y_pred_logreg = logreg.predict(X_test_dtm)

# Evaluate
print("LogReg Accuracy: ", accuracy_score(y_test, y_pred_logreg))
print("LogReg ROC_AUC: ", roc_auc_score(y_test, y_pred_logreg))
```

Step 6: Support Vector Machine (SVM) Implementation

We use LinearSVC, which is specifically optimized for large datasets. SVMs attempt to find the hyperplane that maximizes the margin between classes.

1. Train and Evaluate SVM Model

```
# Step 5: Train Linear SVM
vect = CountVectorizer(preprocessor=clean, max_features=1000)
# Optional: restrict features for speed
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)

svm = LinearSVC()
%time svm.fit(X_train_dtm, y_train)

# Predict
y_pred_svm = svm.predict(X_test_dtm)

# Evaluate
```

```
print("SVM Accuracy: ", accuracy_score(y_test, y_pred_svm))
print("SVM ROC_AUC: ", roc_auc_score(y_test, y_pred_svm))
```

Deliverables

Students must submit one single lab05.ipynb.