Data Mining Week 5

April 14, 2024

[6]:
```
# Installs
import pandas as pd
import zipfile
from textblob import TextBlob
from sklearn.metrics import accuracy_score
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

[8]:
```
# Load the movie review data
df = pd.read_csv('/Users/nickblackford/Desktop/Python/labeledTrainData.tsv',␣
↪delimiter='\t')
df.head()
```

[8]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w…
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi…
2	7759_3	0	The film starts with a manager (Nicholas Bell)…
3	3630_4	0	It must be assumed that those who praised this…
4	9495_8	1	Superbly trashy and wondrously unpretentious 8…

[12]:
```
# Part 2: Prepping Text for a Custom Model
import re
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
# Convert all text to lowercase letters
df['processed_review'] = df['review'].str.lower()
# Remove punctuation and special characters from the text
df['processed_review'] = df['processed_review'].apply(lambda x: re.
sub(r'[^a-z\s]', '', x))
# Remove stop words
stop_words = set(stopwords.words('english'))
df['processed_review'] = df['processed_review'].apply(lambda x: ' '.join([word␣
↪for word in x.split() if word not in stop_words]))
# Apply NLTK's PorterStemmer
stemmer = PorterStemmer()
```

```
df['processed_review'] = df['processed_review'].apply(lambda x: ' '.
    ↪join([stemmer.stem(word) for word in x.split()]))
```

[14]:
```
from sklearn.model_selection import train_test_split

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(df['processed_review'],␣
    ↪df['sentiment'], test_size=0.2, random_state=42)

# Show the size of each set
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[14]:
```
((20000,), (5000,), (20000,), (5000,))
```

[15]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the training data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
```

[16]:
```
# Transform the test data using the already-fitted vectorizer
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

Do not fit to the test data as fitting the vectorizer to the test data would incorporate knowledge about the test data into the model, which are supposed to be unknown attributes. This would cause overfitting to the test data and make the model less accurate when bringing in new test data. By only fitting the vectorizer to the training data, you are also maintaining cosistency by ensuring the same features and document frequency are being used on both the training and test sets.

[18]:
```
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
logistic_model = LogisticRegression(random_state=39, max_iter=1000)

# Fit the model on the training data
logistic_model.fit(X_train_tfidf, y_train)
```

[18]:
```
LogisticRegression(max_iter=1000, random_state=39)
```

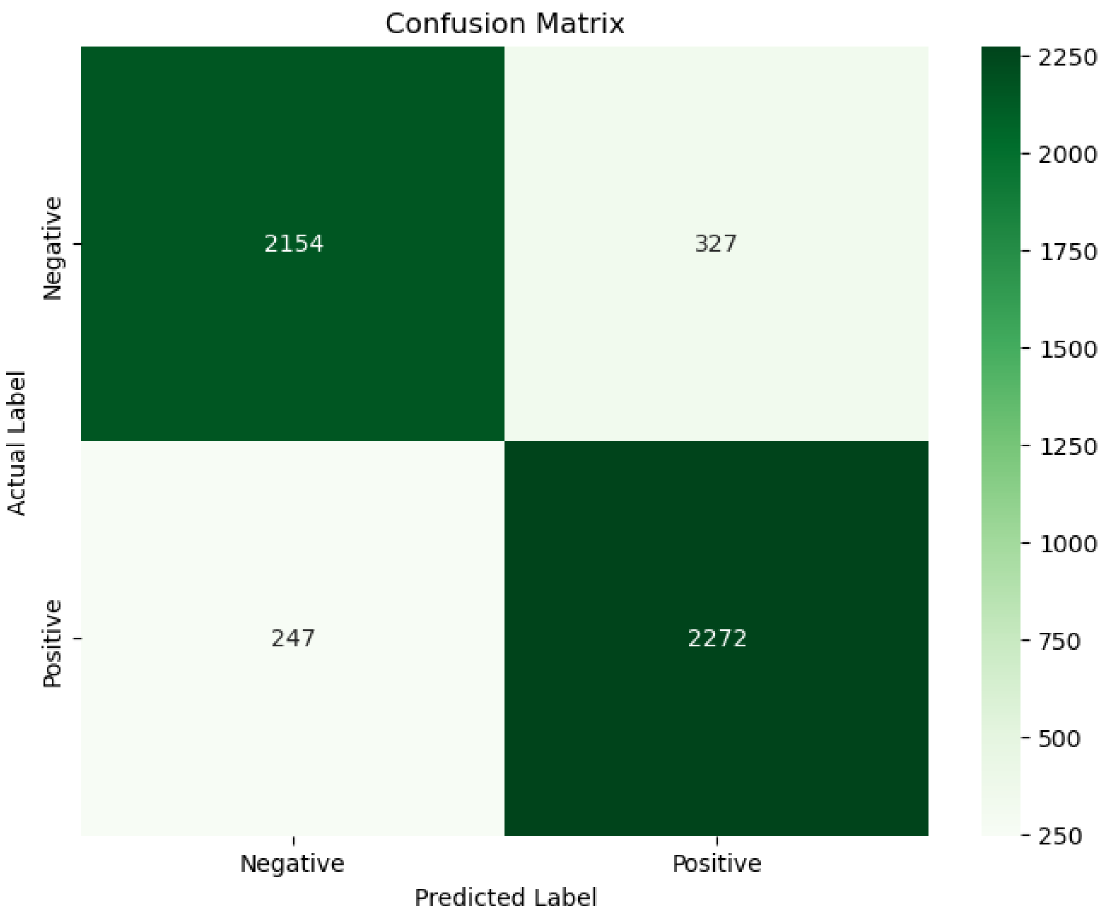
[19]:
```
from sklearn.metrics import accuracy_score

# Make predictions on the test data
y_pred = logistic_model.predict(X_test_tfidf)

# Calculate the accuracy of the model
```

```
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print("Accuracy of the logistic regression model:", accuracy)
```

Accuracy of the logistic regression model: 0.8852

[21]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix using seaborn for better visualization
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap='Greens', xticklabels=['Negative',␣
↪'Positive'], yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```

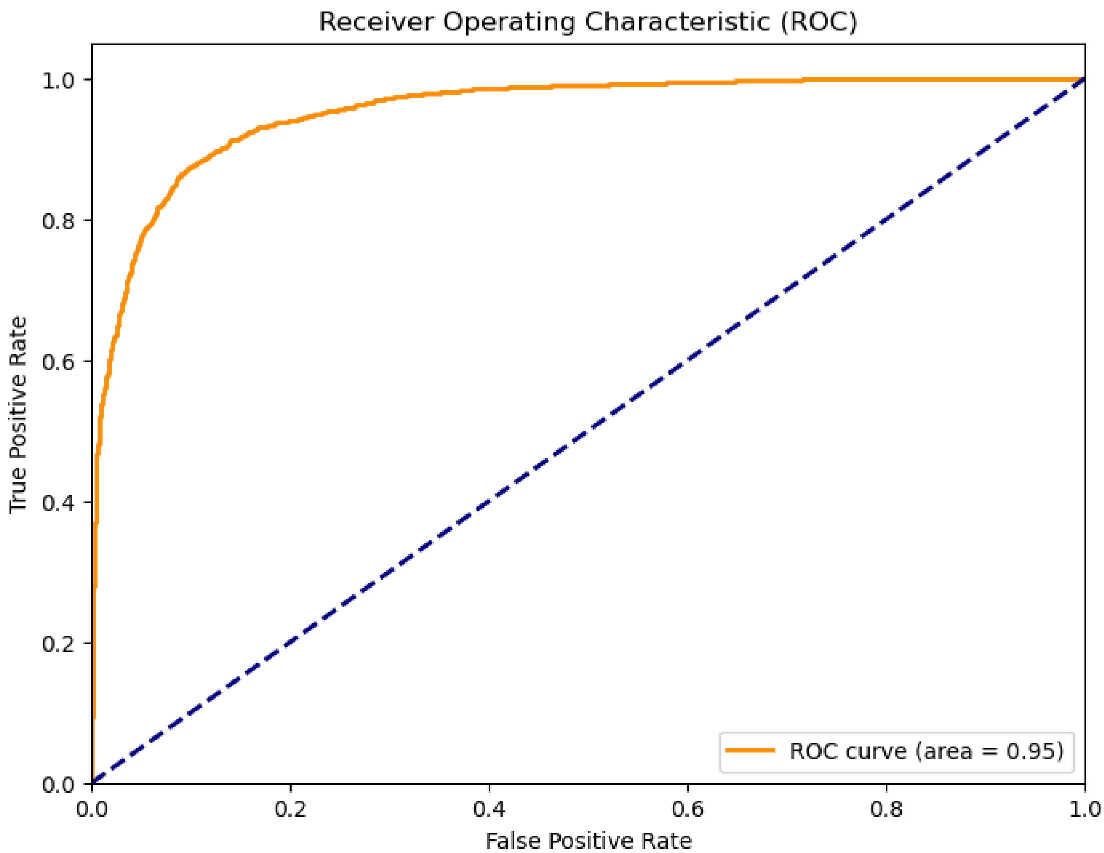


[22]:
```python
from sklearn.metrics import classification_report

# Generate the classification report
report = classification_report(y_test, y_pred, target_names=['Negative',␣
↪'Positive'])

# Print the classification report
print(report)
```

```
              precision    recall  f1-score   support

    Negative       0.90      0.87      0.88      2481
    Positive       0.87      0.90      0.89      2519

    accuracy                           0.89      5000
   macro avg       0.89      0.89      0.89      5000
weighted avg       0.89      0.89      0.89      5000
```

[23]:

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get the probability scores of the positive class
y_scores = logistic_model.predict_proba(X_test_tfidf)[:, 1]

# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

# Calculate the AUC (Area Under Curve)
roc_auc = auc(fpr, tpr)

# Plotting the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```



1 Naive Bayes Classifier

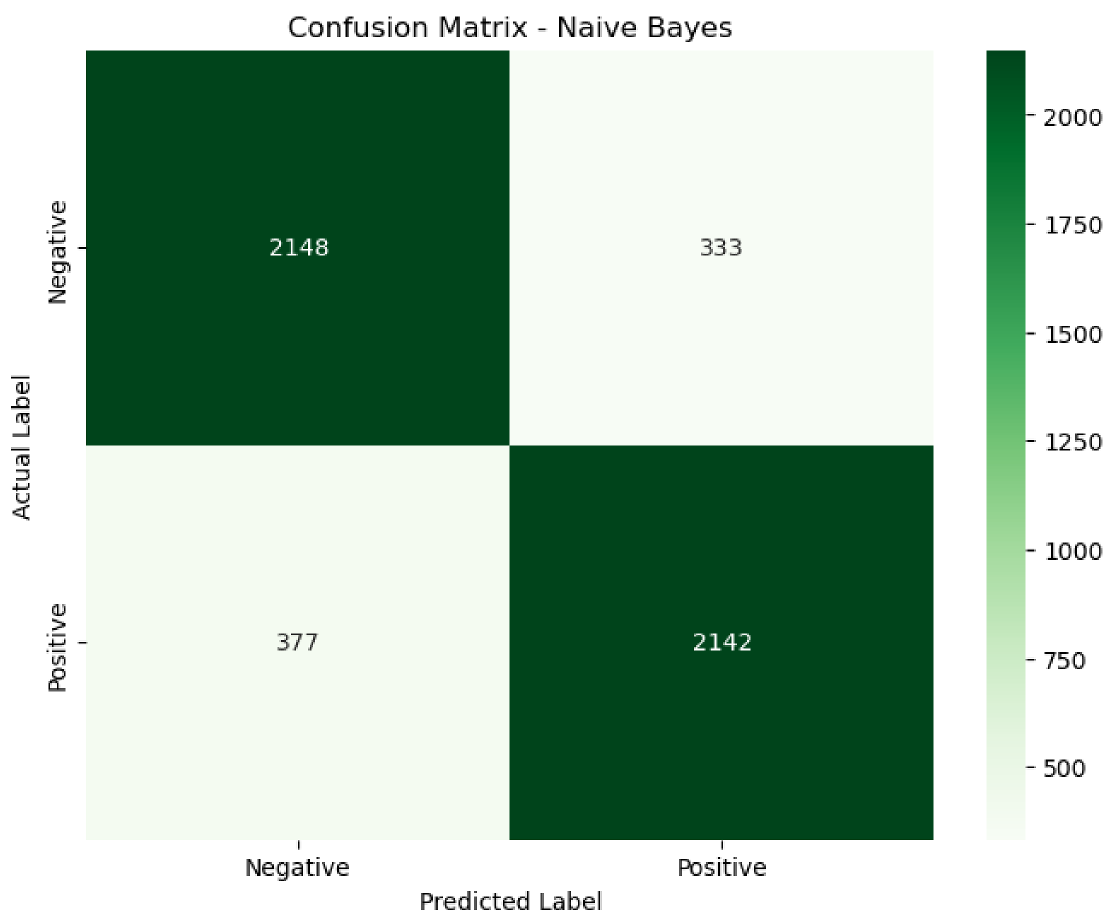
[24]:
```
from sklearn.naive_bayes import MultinomialNB

# Initialize the Naive Bayes model
nb_model = MultinomialNB()

# Fit the model on the training data
nb_model.fit(X_train_tfidf, y_train)
```

[24]: MultinomialNB()

[25]:
```
# Make predictions on the test data
y_pred_nb = nb_model.predict(X_test_tfidf)

# Calculate the accuracy of the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print("Accuracy of the Naive Bayes model:", accuracy_nb)
```

Accuracy of the Naive Bayes model: 0.858

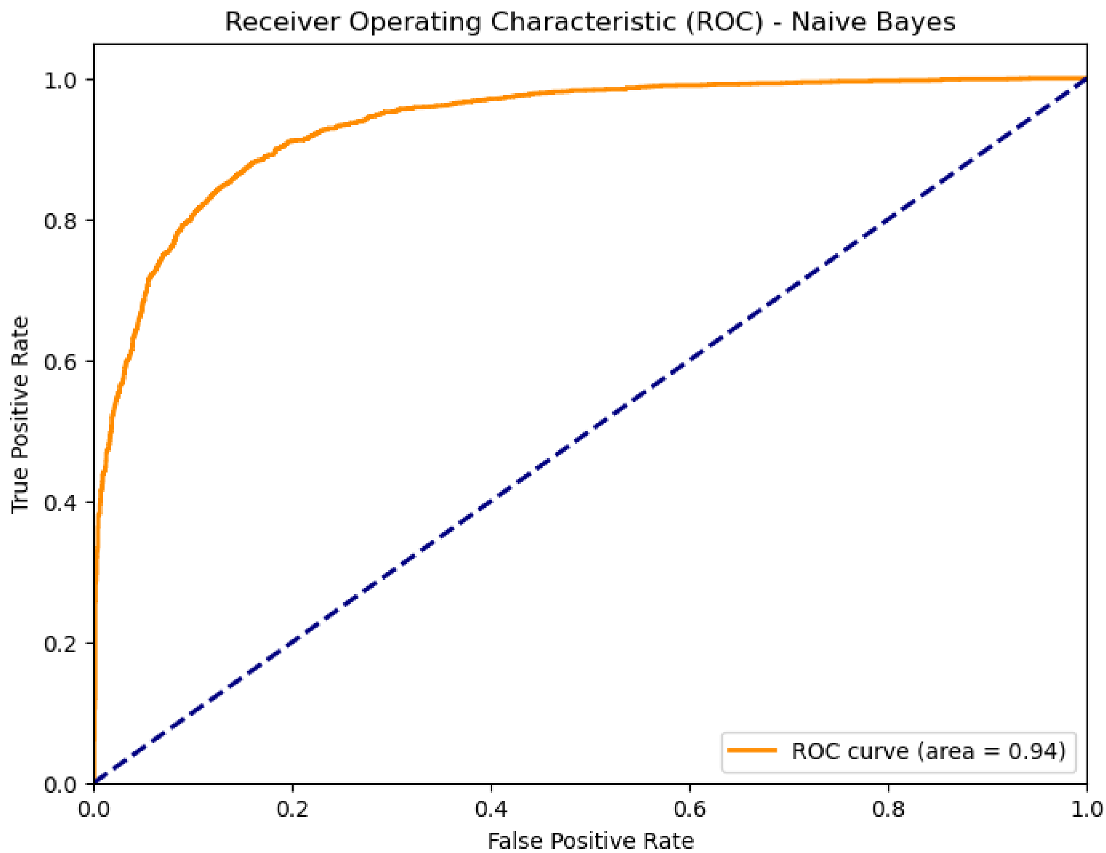
[27]:
```
# Generate the confusion matrix
cm_nb = confusion_matrix(y_test, y_pred_nb)

# Display the confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm_nb, annot=True, fmt="d", cmap='Greens', xticklabels=['Negative',␣
↪'Positive'], yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix - Naive Bayes')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```



[28]:
```
# Generate the classification report
report_nb = classification_report(y_test, y_pred_nb, target_names=['Negative',␣
↪'Positive'])
print(report_nb)
```

```
               precision    recall  f1-score   support

    Negative       0.85      0.87      0.86      2481
    Positive       0.87      0.85      0.86      2519

    accuracy                           0.86      5000
   macro avg       0.86      0.86      0.86      5000
weighted avg       0.86      0.86      0.86      5000
```

[29]:
```
# Get the probability scores of the positive class
y_scores_nb = nb_model.predict_proba(X_test_tfidf)[:, 1]

# Compute the ROC curve
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, y_scores_nb)

# Calculate the AUC (Area Under Curve)
roc_auc_nb = auc(fpr_nb, tpr_nb)

# Plotting the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, color='darkorange', lw=2, label='ROC curve (area = %0.
 ↪2f)' % roc_auc_nb)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Naive Bayes')
plt.legend(loc="lower right")
plt.show()
```


Receiver Operating Characteristic (ROC) - Naive Bayes
True Positive Rate
False Positive Rate
ROC curve (area = 0.94)
0.0
0.2
0.4
0.6
0.8
1.0