Bilenkin550Week5_Exercise_5.2

April 12, 2025

1. Get the stemmed data using the same process you did in Week 3.

[27]:
```
import pandas as pd
import re
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
import nltk
import logging

# Suppressing NLTK download messages
logging.getLogger('nltk.data').setLevel(logging.ERROR)

# Downloading stopwords
nltk.download('stopwords', quiet=True)

# Loading dataset from your path
df = pd.read_csv(r"C:\Users\maxim\OneDrive\Desktop\BU\DSC 550\labeledTrainData.
 ↪tsv", sep='\t')

# Basic text cleaning and stemming
stemmer = PorterStemmer()
stop_words = set(stopwords.words("english"))

def preprocess_and_stem(text):
    # Removing HTML tags
    text = re.sub(r"<.*?>", " ", text)
    # Keeping only letters
    text = re.sub(r"[^a-zA-Z]", " ", text)
    # Converting to lowercase and split
    words = text.lower().split()
    # Removing stopwords and stem
    words = [stemmer.stem(w) for w in words if w not in stop_words]
    return " ".join(words)

# Applying preprocessing
df["clean_review"] = df["review"].apply(preprocess_and_stem)

# Previewing the result by displaying the first five rows
```

```
df[["review", "clean_review", "sentiment"]].head()
```

[27]:

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

                                        clean_review  sentiment
0  stuff go moment mj start listen music watch od…          1
1  classic war world timothi hine entertain film …          1
2  film start manag nichola bell give welcom inve…          0
3  must assum prais film greatest film opera ever…          0
4  superbl trashi wondrous unpretenti exploit hoo…          1
```

2. Split this into a training and test set.

[28]:

```
from sklearn.model_selection import train_test_split

# Splitting the clean text and labels
X = df["clean_review"]
y = df["sentiment"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

3. Fit and apply the tf-idf vectorization to the training set.

[29]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initializing TF-IDF with a limit on max features
vectorizer = TfidfVectorizer(max_features=5000)

# Fitting on training data only
X_train_tfidf = vectorizer.fit_transform(X_train)

# Transforming test data using the same fitted vectorizer
X_test_tfidf = vectorizer.transform(X_test)
```

4. Apply but DO NOT FIT the tf-idf vectorization to the test set (Why?).

[30]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initializing TF-IDF vectorizer
vectorizer = TfidfVectorizer(max_features=5000)

# Fitting and transforming only the training data
```

```
X_train_tfidf = vectorizer.fit_transform(X_train)

# Transforming the test data (DO NOT FIT)
X_test_tfidf = vectorizer.transform(X_test)
```

The reason we do not fit the TF-IDF vectorizer on the test set is that doing so would create data leakage. This means we would allow the test data to influence the model before evaluation, which can lead to overly optimistic performance results and violates proper machine learning practices. Instead, we should fit the vectorizer only on the training data to learn the vocabulary and weighting. Then, we apply and transform this learned vocabulary on the test data. This ensures the model is evaluated on truly unseen data, providing a more realistic estimate of its performance.

5. Train a logistic regression using the training data.

[31]:
```
from sklearn.linear_model import LogisticRegression

# Creating a logistic regression model
model = LogisticRegression()

# Training the model using the training data
model.fit(X_train_tfidf, y_train)
```

[31]:
```
LogisticRegression()
```

6. Find the model accuracy on test set.

[32]:
```
from sklearn.metrics import accuracy_score

# Predicting the test set results
y_pred = model.predict(X_test_tfidf)

# Calculating accuracy
accuracy = accuracy_score(y_test, y_pred)

# Printing the accuracy
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.8842
```

As we can see from the above result, the accuracy of the model on the test set is 0.8842, or 88.42%. This indicates that the model is performing well, correctly predicting the sentiment of the reviews in most cases.

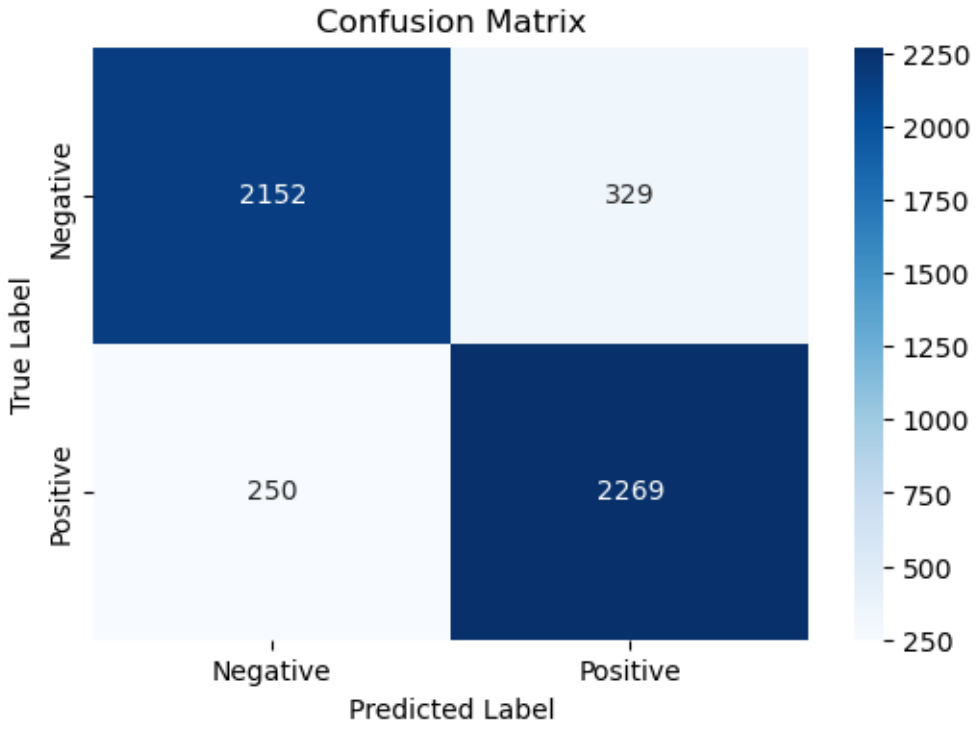
7. Create a confusion matrix for the test set predictions.

[33]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Creating confusion matrix
```

```
cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Negative",␣
 ↪"Positive"], yticklabels=["Negative", "Positive"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



As we can see from the confusion matrix, the model correctly classified 2,152 negative reviews and 2,269 positive reviews. However, there were some inaccuracies, with 250 false positives (negative reviews incorrectly classified as positive) and 329 false negatives (positive reviews incorrectly classified as negative). Despite these misclassifications, the model still demonstrates a high level of accuracy in classification.

8. Get the precision, recall, and F1-score for the test set predictions.

[34]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculating precision, recall, and F1-score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred)

# Displaying all results
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Precision: 0.8734
Recall: 0.9008
F1-Score: 0.8868
```

The above results confirm that the precision of the model is 0.8734 or 87.34%, which means the model correctly predicted positive reviews with a high level of accuracy. The recall is 0.9008 or 90.08%, indicating that the model was able to identify most of the actual positive reviews. The F1-score of 0.8868 or 88.68% balances both precision and recall, confirming that the model performs well in identifying true positives while minimizing false positives and false negatives.

9. Create a ROC curve for the test set.

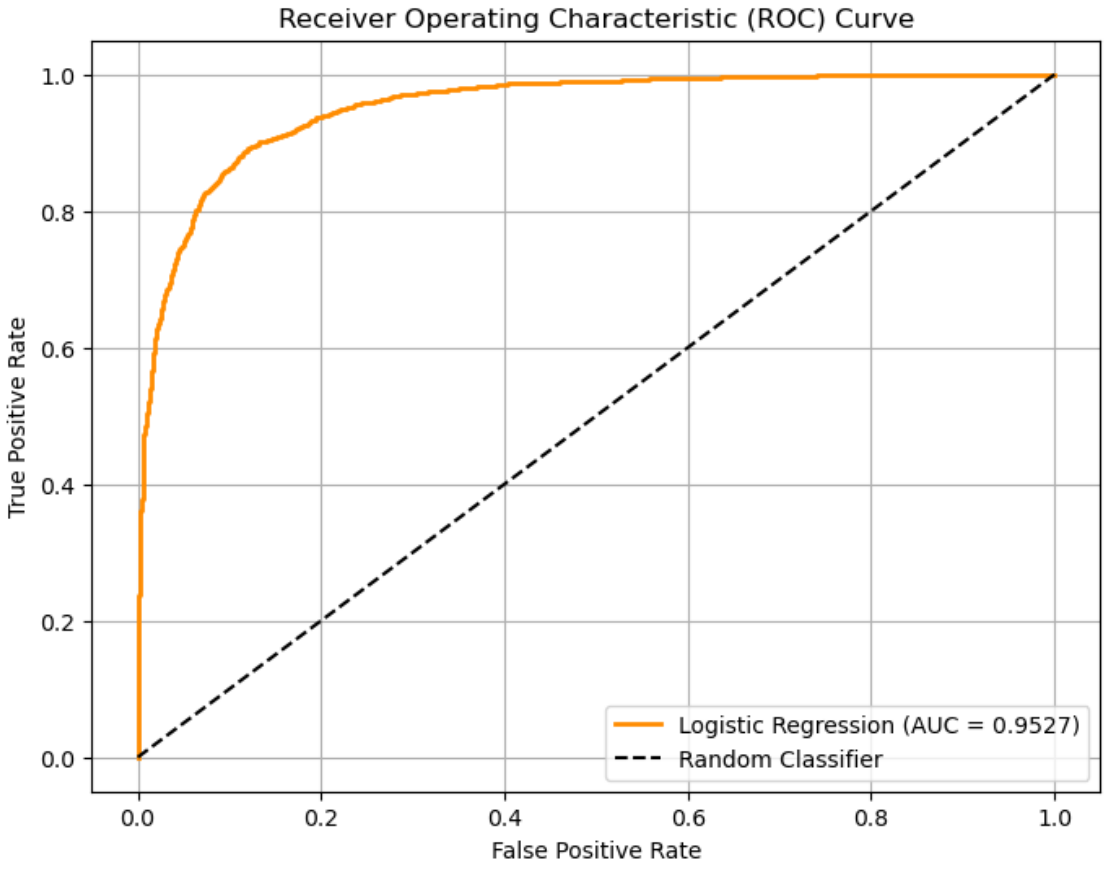
[35]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Getting predicted probabilities for the positive class
y_probs = model.predict_proba(X_test_tfidf)[:, 1]

# Calculating the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

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

# Plotting the ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.4f})',␣
 ↪color='darkorange', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



From the above graph, we can see that the Logistic Regression model (yellow line) performs very well, with an Area Under the Curve (AUC) of 0.9527 or 95.27%. This indicates excellent performance in distinguishing between positive and negative reviews. The ROC curve starts steeply, indicating that the model correctly classifies a high proportion of true positives with a low false positive rate. In contrast, the diagonal line represents a random classifier with 50/50 chance, which the Logistic Regression model clearly outperforms. The curve's steep and high trajectory confirms the model's strong discriminative power.

10. Pick another classification model you learned about this week and repeat steps (5) – (9).

Train a Random Forest Classifier model (Step 5)

[36]:
```
from sklearn.ensemble import RandomForestClassifier

# Initializing the model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Training the model
rf_model.fit(X_train_tfidf, y_train)
```

[36]:
```
RandomForestClassifier(random_state=42)
```

Model Accuracy (Step 6)

[37]:
```
# Predicting on test set
rf_predictions = rf_model.predict(X_test_tfidf)

# Calculating accuracy
from sklearn.metrics import accuracy_score

rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Accuracy: {rf_accuracy:.4f}")
```

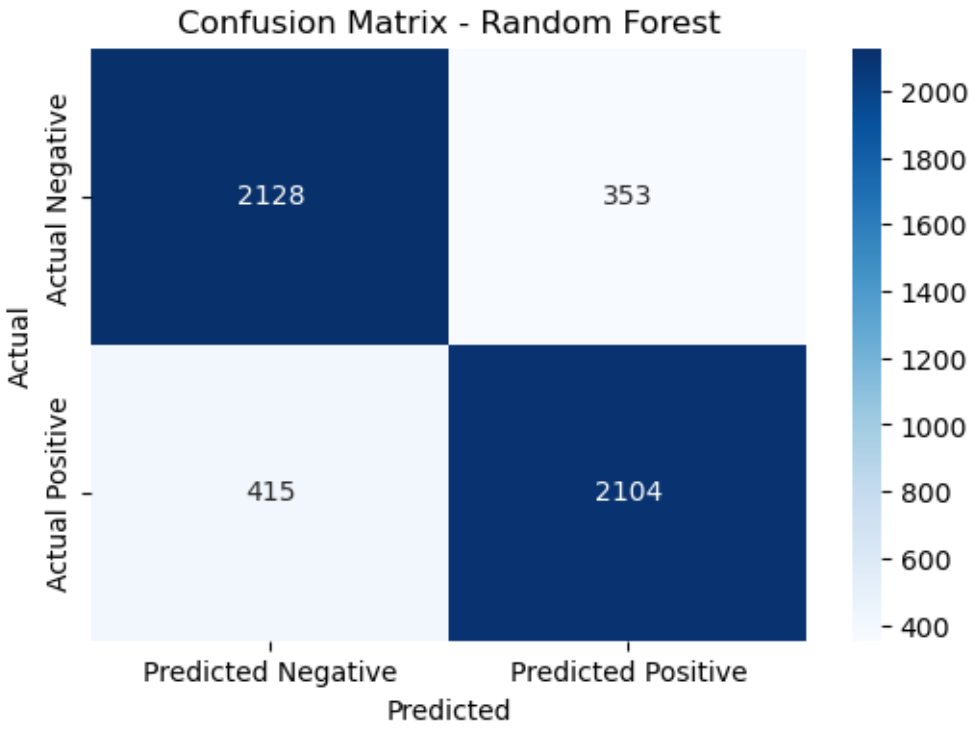
```
Accuracy: 0.8464
```

Careating a confusion matrix (Step 7)

[38]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Generate the confusion matrix
rf_cm = confusion_matrix(y_test, rf_predictions)

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(rf_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title('Confusion Matrix - Random Forest')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Getting the precision, recall, and F1-score (Step 8)

[39]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_f1 = f1_score(y_test, rf_predictions)

print(f"Precision: {rf_precision:.4f}")
print(f"Recall: {rf_recall:.4f}")
print(f"F1-Score: {rf_f1:.4f}")
```

```
Precision: 0.8563
Recall: 0.8353
F1-Score: 0.8457
```

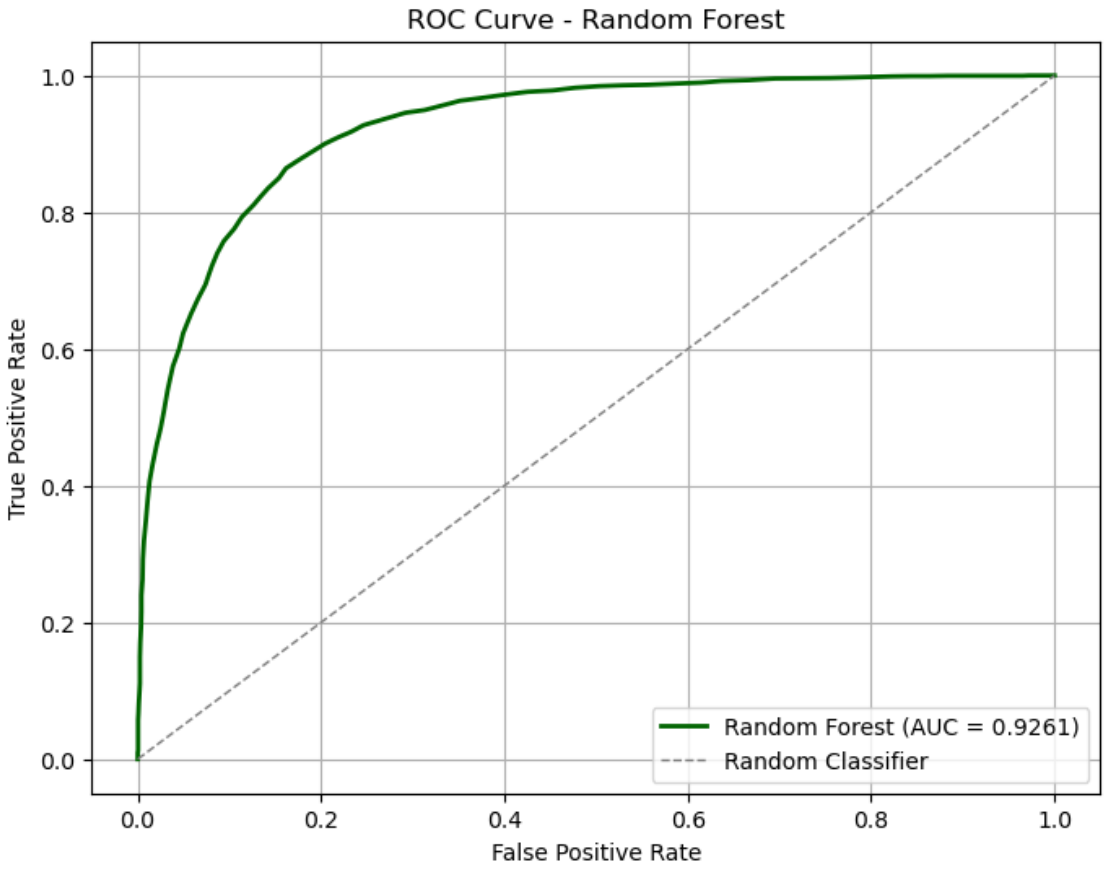
Creating a ROC cure (Step 9)

[40]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Getting predicted probabilities
rf_probs = rf_model.predict_proba(X_test_tfidf)[:, 1]
```

```
# Computing ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, rf_probs)
roc_auc = auc(fpr, tpr)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkgreen', lw=2, label=f'Random Forest (AUC =␣
 ↪{roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--', label='Random␣
 ↪Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend()
plt.grid(True)
plt.show()
```



As we can see from the Random Forest classifier model, the precision is 0.8563 (85.63%) and the recall is 0.8353 (83.53%). The F1-score, which balances precision and recall, is 0.8457 (84.57%). These results are slightly lower than those from the Logistic Regression model. However, the

Random Forest model still performs well, correctly predicting positive reviews with high accuracy and effectively identifying most actual positive cases.