Brian Tenneson

ADS 652

6/28/2024

Final Report

In this document I present a view of two case studies of language analyses of aggregate websites. The first is from rotten tomatoes. This data was provided for us by Professor Blanchard during the course in the file called critics.csv. To obtain a similar scrape from a different aggregate website, I found that the websites I tried had policies against scraping their data. However, I was able to find a miniature set of aggregate language data on a GitHub website which comprises the second case study. It also happens to be about movies so it might be reasonable to compare certain aspects to the rotten tomatoes analysis.

I will provide both sets of code in order followed by what each code's output is. Each code's output is a series of metrics including confusion matrix to accuracy.

To solidify into my mind, the confusion matrix is a 2x2 matrix consisting of false positives, false negatives, true positives, and true negatives. From Wikipedia [2]:

Predicted condition			
Total population = P + N	**Positive (PP)**	**Negative (PN)**	
Actual condition	**Positive (P)**	True positive **(TP)**	False negative **(FN)**
	Negative (N)	False positive **(FP)**	True negative **(TN)**

The four underlined links show what position the metrics go. (For instance, false positives go in the pink intersection of actual condition and predicted condition labeled FP.) The main diagonal counts true positive and true negatives while the other diagonal counts false negatives and false positives.

First Project (Rotten Tomatoes)

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score

from sklearn.naive_bayes import MultinomialNB

from sklearn.feature_extraction.text import CountVectorizer


# Load the data

df = pd.read_csv('critics.csv')


# Strip any leading or trailing whitespace from column names

df.columns = df.columns.str.strip()


# Check for the 'quote' column and proceed

if 'quote' in df.columns:

    df = df[~df['quote'].isnull()]

    df = df.reset_index(drop=True)

    df = df[df['fresh'].isin(['fresh', 'rotten'])]


    # Vectorize the text data

    vectorizer = CountVectorizer()

    vectorized = vectorizer.fit_transform(df['quote'])
    # Create a sparse DataFrame from the vectorized data

    X = pd.DataFrame.sparse.from_spmatrix(vectorized,
columns=vectorizer.get_feature_names_out())


    # Create the target variable

    y = df['fresh'] == 'fresh'


    # Split the data into training and test sets

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=100)


    # Initialize and fit the MultinomialNB model

    nb = MultinomialNB(alpha=0.01)

    nb.fit(X_train, y_train)


    # Make predictions

    predictions = nb.predict(X_test)

    actuals = y_test


    # Calculate the metrics

    conf_matrix = confusion_matrix(actuals, predictions)

    accuracy = accuracy_score(actuals, predictions)

    precision = precision_score(actuals, predictions)

    recall = recall_score(actuals, predictions)
    f1 = f1_score(actuals, predictions)

    roc_auc = roc_auc_score(actuals, predictions)


    # Print the results

    print(f"Confusion Matrix:\n{conf_matrix}")

    print(f"Accuracy: {accuracy:.4f}")

    print(f"Precision: {precision:.4f}")

    print(f"Recall: {recall:.4f}")

    print(f"F1 Score: {f1:.4f}")

    print(f"ROC-AUC Score: {roc_auc:.4f}")

else:

    print("The 'quote' column does not exist in the DataFrame")
```

Output:

```
Confusion Matrix:

[[ 989  557]

 [ 453 1885]]

Accuracy: 0.7400

Precision: 0.7719

Recall: 0.8062

F1 Score: 0.7887

ROC-AUC Score: 0.7230
```

Second Project (GitHub [1])
Here, to clean the data obtained from [1], notice that I created a score derived from the biehn scale rating of each movie. The bein scale ranges from 0 to 10 and I decided to call scores of at least 7 fresh and other scores rotten. This may be why, as you will see, the metrics turn out so terribly (e.g., accuracy). What I mean by that is that altering the freshness threshold may have led to better metrics in terms of things like the confusion matrix. Let me provide you with a snippet of the cleaned data in all_data-2.csv:

movie_name	Score derived from biehn_scale_rating
Zoolander 2	fresh
Dope	fresh
The Big Short	fresh
Deadpool	fresh
The Martian	fresh
Hardcore Henry	fresh
San Andreas	rotten
Terminator	fresh
Captain America: Civil War	fresh
All the Way	fresh
The Man from U.N.C.L.E.	fresh
Chef	fresh
Independence Day: Resurgence	rotten
Suicide Squad	rotten

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score

from sklearn.naive_bayes import MultinomialNB

from sklearn.feature_extraction.text import CountVectorizer


# Load the data

df = pd.read_csv("G:\\My Drive\\Summer 1 - 2024\\ADS 652\\all_data-
2.csv")


# Strip any leading or trailing whitespace from column names

df.columns = df.columns.str.strip()


# Check if there are at least two columns

if df.shape[1] >= 2:

    # Rename the first two columns for clarity

    df.columns = ['quote', 'fresh'] + list(df.columns[2:])


    df = df[~df['quote'].isnull()]

    df = df[df['fresh'].isin(['fresh', 'rotten'])]

    df = df.reset_index(drop=True)
    # Vectorize the text data

    vectorizer = CountVectorizer()

    X = vectorizer.fit_transform(df['quote'])


    # Create the target variable

    y = df['fresh'].apply(lambda x: 1 if x == 'fresh' else 0)


    # Split the data into training and test sets

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=100)


    # Initialize and fit the MultinomialNB model

    nb = MultinomialNB(alpha=0.01)

    nb.fit(X_train, y_train)


    # Make predictions

    predictions = nb.predict(X_test)

    actuals = y_test


    # Calculate the metrics

    conf_matrix = confusion_matrix(actuals, predictions)

    accuracy = accuracy_score(actuals, predictions)

    precision = precision_score(actuals, predictions)

    recall = recall_score(actuals, predictions)
    f1 = f1_score(actuals, predictions)
    roc_auc = roc_auc_score(actuals, predictions)

    # Print the results
    print(f"Confusion Matrix:\n{conf_matrix}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print(f"ROC-AUC Score: {roc_auc:.4f}")
else:
    print("The DataFrame does not have at least two columns")
```

output:

```
Confusion Matrix:
[[ 9  4]
 [14  8]]
Accuracy: 0.4857
Precision: 0.6667
Recall: 0.3636
F1 Score: 0.4706
ROC-AUC Score: 0.5280
```

Conclusions

When we look at the confusion matrices of rotten tomatoes and the GitHub scrape, there are some differences and some similarities. Recall that the main diagonal gives us true positives and true negatives while the other diagonal gives false positives and false negatives. The rotten tomatoes confusion matrix has a strong showing along its main diagonal compared to its other diagonal. Also, the accuracy and f1-score of the metrics for the rotten tomatoes data is better than the GitHub data by far.

In contrast, the confusion matrix of the GitHub project is balanced when comparing the main diagonal to another diagonal. Note, though, that the largest number in the confusion matrix is for a false positive. The accuracy and other scores are significantly less than for the corresponding numbers in the rotten tomatoes analysis.

I can speculate that two things may increase the overall reliability of the GitHub analysis: more movies and changing how I created the fresh/rotten choice. I chose to call a movie fresh if its score is 7 or higher; rotten otherwise. However, things may change dramatically if I use a different number like 5.

Links

[1] https://github.com/dkhundley/movie-ratings-model/blob/main/data/clean/train.csv

[2] https://en.wikipedia.org/wiki/Confusion_matrix