BBC News Classification

I classify BBC news articles using both unsupervised (NMF) and supervised (Logistic Regression) models. I begin with data exploration, feature extraction using TF-IDF, modeling, and then performance evaluation.

1. Import Required Libraries

In [6]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.linear_model import LogisticRegression
import itertools
import warnings
warnings.filterwarnings('ignore')
```

2. Load Dataset

In [7]:

```
train_df = pd.read_csv('/kaggle/input/bbc-new-article/BBC News Train.csv')
test_df = pd.read_csv('/kaggle/input/bbc-new-article/BBC News Test.csv')
test_labels_df = pd.read_csv('/kaggle/input/bbc-new-article/BBC News Sample Solution.csv
```

3. Inspect and Clean Data

In [8]:

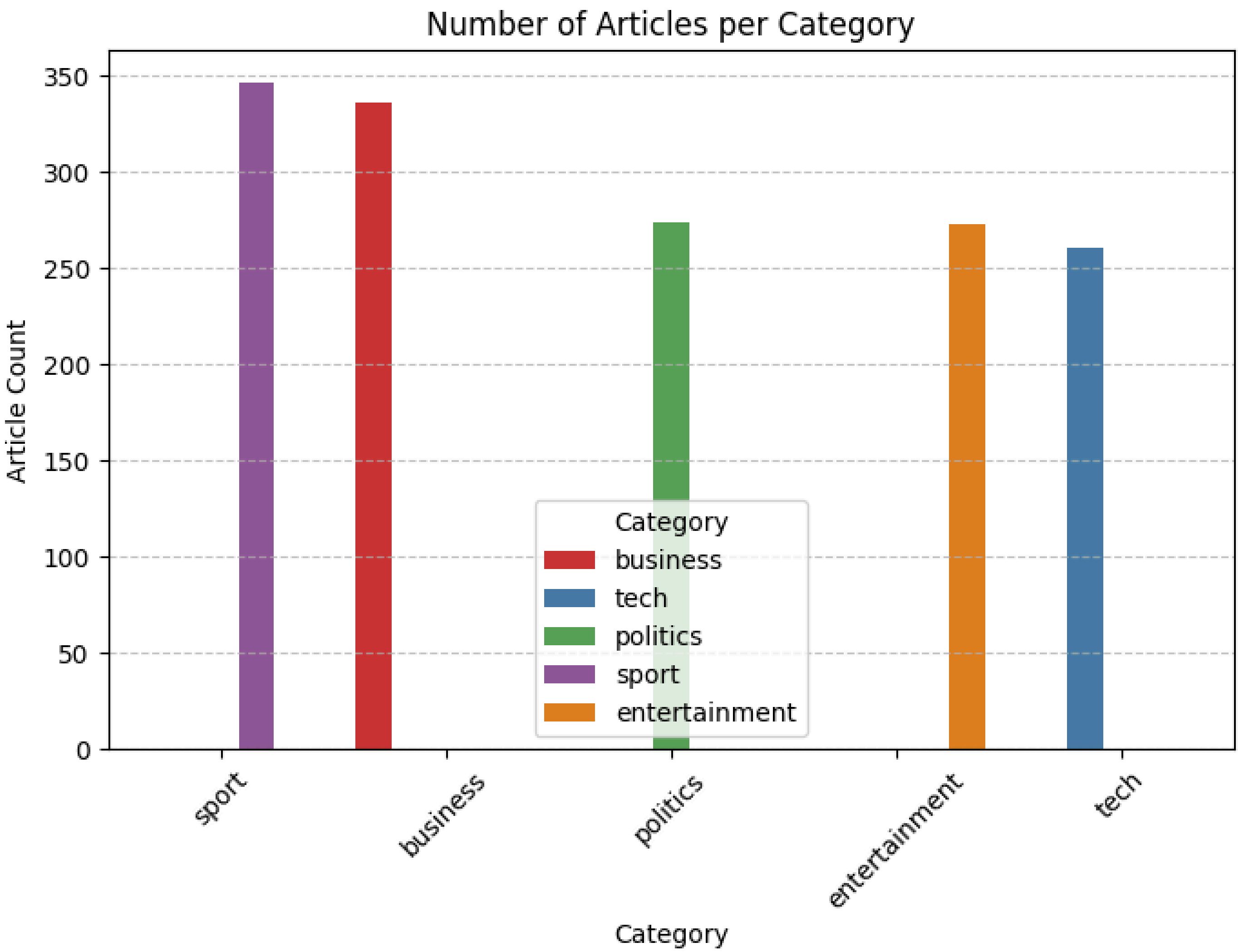
```
print("Training Data Preview:")
print(train_df.head())
print("\nTest Data Preview:")
print(test_df.head())
print("\nTest Labels Preview:")
print(test_labels_df.head())

print("\nShapes:")
print("Train:", train_df.shape)
print("Test :", test_df.shape)
print("Labels:", test_labels_df.shape)

print("\nMatching IDs in Test and Labels:", all(test_df['ArticleId'] == test_labels_df['

print("\nMissing values in train:")
print(train_df.isnull().sum())
print("\nMissing values in test:")
print(test_df.isnull().sum())
print("\nMissing values in labels:")
print(test_labels_df.isnull().sum())

print("\nEmpty strings in train text:", (train_df['Text'].str.strip() == '').sum())
print("Empty strings in test text:", (test_df['Text'].str.strip() == '').sum())
```

```
print("\nDuplicates in train:", train_df.duplicated().sum())
print("Duplicates in test:", test_df.duplicated().sum())

print("\nUnique training articles:", train_df['ArticleId'].nunique())
```

```
Training Data Preview:
   ArticleId                                               Text  Category
0       1833  worldcom ex-boss launches defence lawyers defe...  business
1        154  german business confidence slides german busin...  business
2       1101  bbc poll indicates economic gloom citizens in ...  business
3       1976  lifestyle  governs mobile choice  faster  bett...      tech
4        917  enron bosses in $168m payout eighteen former e...  business

Test Data Preview:
   ArticleId                                               Text
0       1018  qpr keeper day heads for preston queens park r...
1       1319  software watching while you work software that...
2       1138  d arcy injury adds to ireland woe gordon d arc...
3        459  india s reliance family feud heats up the ongo...
4       1020  boro suffer morrison injury blow middlesbrough...

Test Labels Preview:
   ArticleId       Category
0       1018          sport
1       1319           tech
2       1138       business
3        459  entertainment
4       1020       politics

Shapes:
Train: (1490, 3)
Test : (735, 2)
Labels: (735, 2)

Matching IDs in Test and Labels: True

Missing values in train:
ArticleId    0
Text         0
Category     0
dtype: int64

Missing values in test:
ArticleId    0
Text         0
dtype: int64

Missing values in labels:
ArticleId    0
Category     0
dtype: int64

Empty strings in train text: 0
Empty strings in test text: 0

Duplicates in train: 0
Duplicates in test: 0

Unique training articles: 1490
```

4. Visualize Category Distribution

In [9]:
```
plt.figure(figsize=(8, 5))
sns.countplot(data=train_df, x='Category', palette='Set1', hue='Category', order=train_d
plt.title('Number of Articles per Category')
plt.ylabel('Article Count')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



5. TF-IDF Vectorization and Top Keywords

In [10]:
```
vectorizer = TfidfVectorizer(stop_words='english')
vectors = vectorizer.fit_transform(train_df['Text'])
feature_names = vectorizer.get_feature_names_out()
dense = vectors.todense()
denselist = dense.tolist()
tfidf_matrix = pd.DataFrame(denselist, columns=feature_names)
train_df_tfidf = pd.concat([train_df.reset_index(drop=True), tfidf_matrix], axis=1)

avg_tfidf = tfidf_matrix.groupby(train_df['Category']).mean()
top_keywords = {}
for category, row in avg_tfidf.iterrows():
    top_keywords[category] = row.nlargest(10).index.tolist()
```

```
for cat, words in top_keywords.items():
    print(f"{cat}: {', '.join(words)}")
```

```
business: said, growth, economy, bank, market, year, firm, mr, oil, sales
entertainment: film, best, music, said, awards, band, actor, award, album, star
politics: mr, labour, said, election, blair, party, government, brown, minister, howard
sport: england, game, win, said, cup, chelsea, team, match, players, season
tech: people, mobile, said, software, users, technology, microsoft, net, phone, digital
```

6. Train and Evaluate NMF

In [11]:

```
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = vectorizer.fit_transform(train_df['Text'])

nmf = NMF(n_components=5, solver='mu', beta_loss='kullback-leibler', random_state=42)
W = nmf.fit_transform(X_train_tfidf)
cluster_labels = W.argmax(axis=1)
```

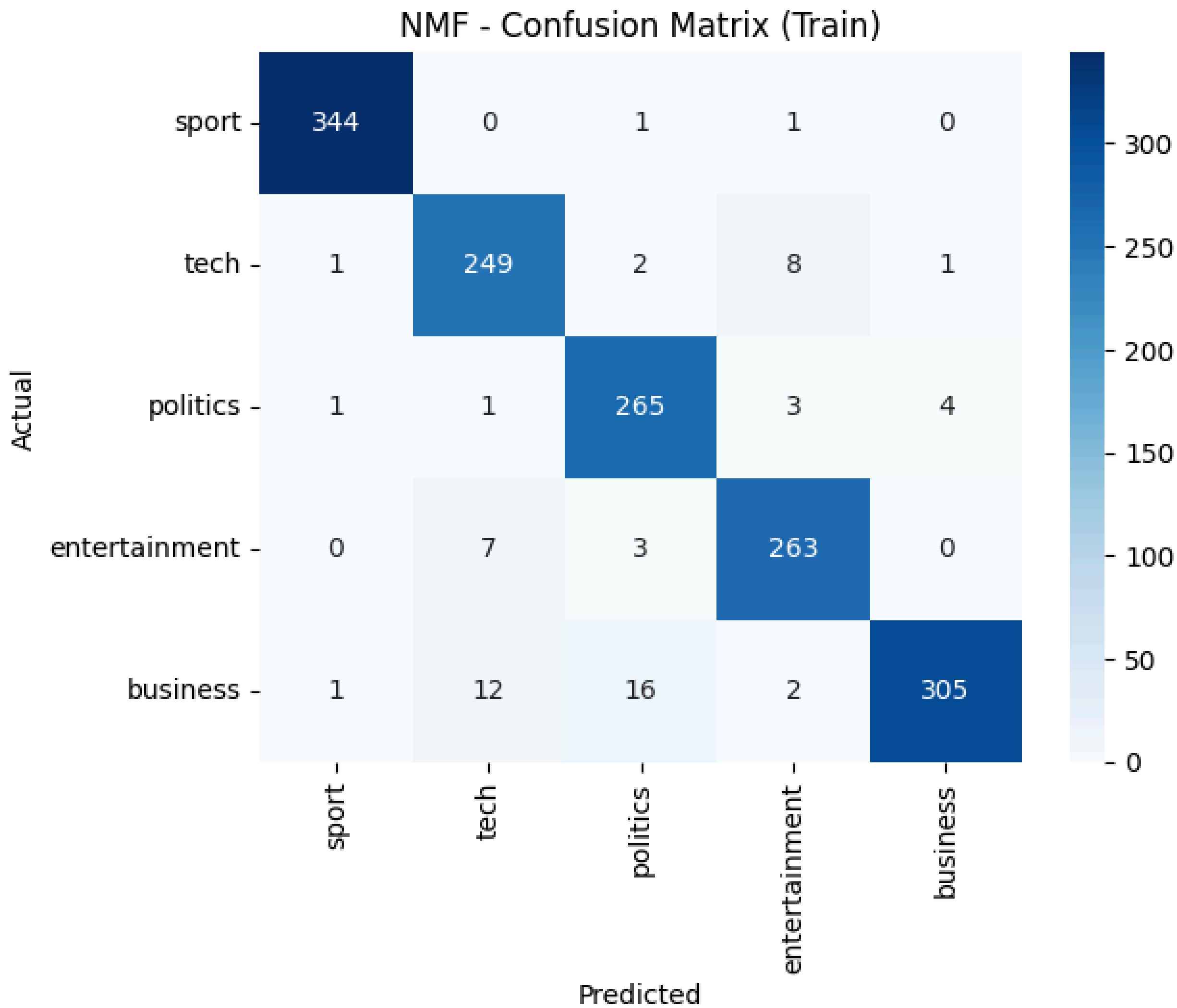
In [12]:

```
def label_permute_compare(y_true_df, y_pred, n=5):
    true_labels = y_true_df.values.flatten()
    label_to_int = {label: i for i, label in enumerate(set(true_labels))}
    true_numeric = np.array([label_to_int[label] for label in true_labels], dtype=int)
    best_perm, best_acc = None, 0.0
    for perm in itertools.permutations(range(n)):
        permuted_pred = [perm[label] if label < n else -1 for label in y_pred]
        acc = np.mean(true_numeric == permuted_pred)
        if acc > best_acc:
            best_perm, best_acc = perm, acc
    return best_perm, best_acc, label_to_int
```

In [13]:

```
perm, acc, label_to_int = label_permute_compare(train_df[['Category']], cluster_labels)
print(f"Best Label Permutation: {perm}")
print(f"NMF Clustering Accuracy (Train): {acc:.4f}")

true_labels = [label_to_int[label] for label in train_df['Category']]
predicted_labels = [perm[label] for label in cluster_labels]
labels_sorted = sorted(label_to_int, key=label_to_int.get)

cm = confusion_matrix(true_labels, predicted_labels, labels=range(len(labels_sorted)))
sns.heatmap(pd.DataFrame(cm, index=labels_sorted, columns=labels_sorted), annot=True, fm
plt.title('NMF - Confusion Matrix (Train)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("\nClassification Report (Train):")
print(classification_report(true_labels, predicted_labels, target_names=labels_sorted))
```

```
Best Label Permutation: (0, 2, 1, 3, 4)
NMF Clustering Accuracy (Train): 0.9570
```



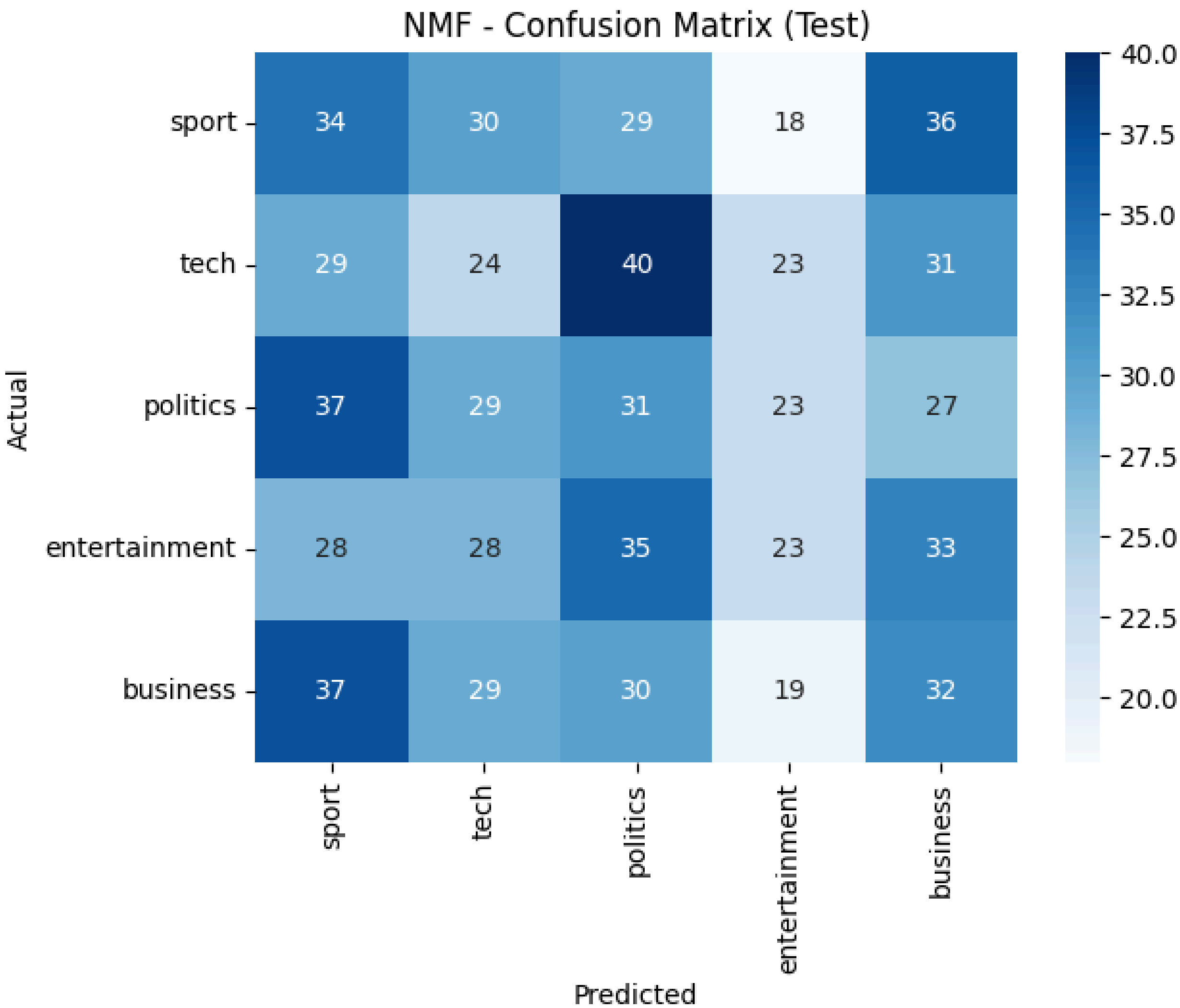

```
Classification Report (Train):
               precision    recall  f1-score   support

        sport       0.99      0.99      0.99       346
         tech       0.93      0.95      0.94       261
     politics       0.92      0.97      0.94       274
entertainment       0.95      0.96      0.96       273
     business       0.98      0.91      0.94       336

     accuracy                           0.96      1490
    macro avg       0.95      0.96      0.96      1490
 weighted avg       0.96      0.96      0.96      1490
```

7. Evaluate NMF on Test Set

In [14]:

```
X_test_tfidf = vectorizer.transform(test_df['Text'])
W_test = nmf.transform(X_test_tfidf)
cluster_labels_test = W_test.argmax(axis=1)

true_labels_test = [label_to_int[label] for label in test_labels_df['Category']]
predicted_labels_test = [perm[label] for label in cluster_labels_test]

cm_test = confusion_matrix(true_labels_test, predicted_labels_test, labels=range(len(lab
sns.heatmap(pd.DataFrame(cm_test, index=labels_sorted, columns=labels_sorted), annot=Tru
plt.title('NMF - Confusion Matrix (Test)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
plt.show()

print("\nClassification Report (Test):")
print(classification_report(true_labels_test, predicted_labels_test, target_names=labels
```




```
Classification Report (Test):
               precision    recall  f1-score   support

        sport       0.21      0.23      0.22       147
         tech       0.17      0.16      0.17       147
     politics       0.19      0.21      0.20       147
entertainment       0.22      0.16      0.18       147
     business       0.20      0.22      0.21       147

     accuracy                           0.20       735
    macro avg       0.20      0.20      0.19       735
 weighted avg       0.20      0.20      0.19       735
```

8. Train Logistic Regression

In [15]:
```
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train = vectorizer.fit_transform(train_df['Text'])
X_test = vectorizer.transform(test_df['Text'])
y_train = train_df['Category']
y_test = test_labels_df['Category']
```

```
clf = LogisticRegression(max_iter=1000, random_state=42)
clf.fit(X_train, y_train)
```

Out[15]:

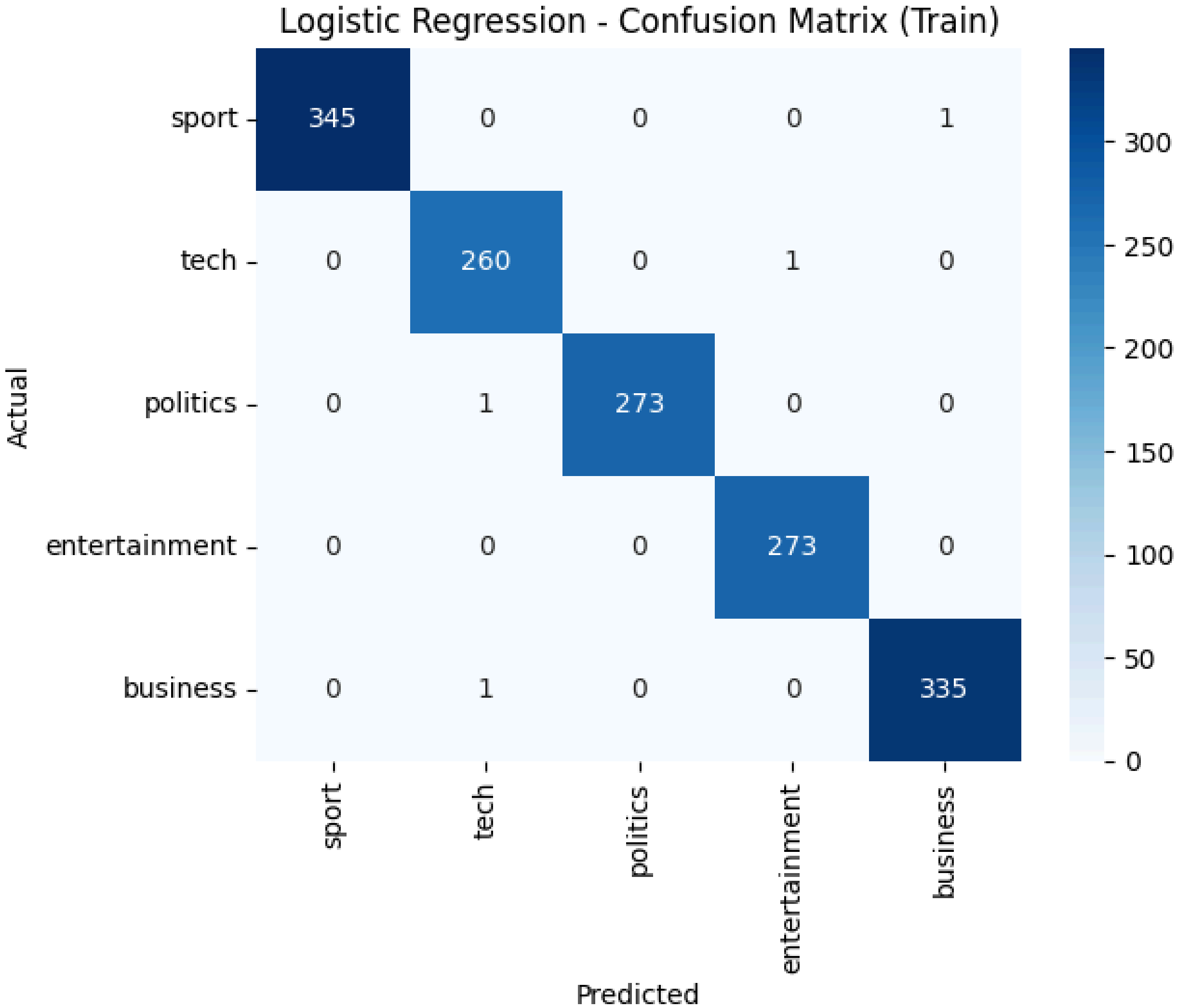
```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

9. Evaluate Logistic Regression

In [16]:

```
y_train_pred = clf.predict(X_train)
print(f"Logistic Regression Accuracy (Train): {accuracy_score(y_train, y_train_pred):.4f

cm_train = confusion_matrix(y_train, y_train_pred, labels=labels_sorted)
sns.heatmap(pd.DataFrame(cm_train, index=labels_sorted, columns=labels_sorted), annot=Tr
plt.title('Logistic Regression - Confusion Matrix (Train)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("\nClassification Report (Train):")
print(classification_report(y_train, y_train_pred, target_names=labels_sorted))

y_test_pred = clf.predict(X_test)
print(f"Logistic Regression Accuracy (Test): {accuracy_score(y_test, y_test_pred):.4f}")

cm_test = confusion_matrix(y_test, y_test_pred, labels=labels_sorted)
sns.heatmap(pd.DataFrame(cm_test, index=labels_sorted, columns=labels_sorted), annot=Tru
plt.title('Logistic Regression - Confusion Matrix (Test)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("\nClassification Report (Test):")
print(classification_report(y_test, y_test_pred, target_names=labels_sorted))
```

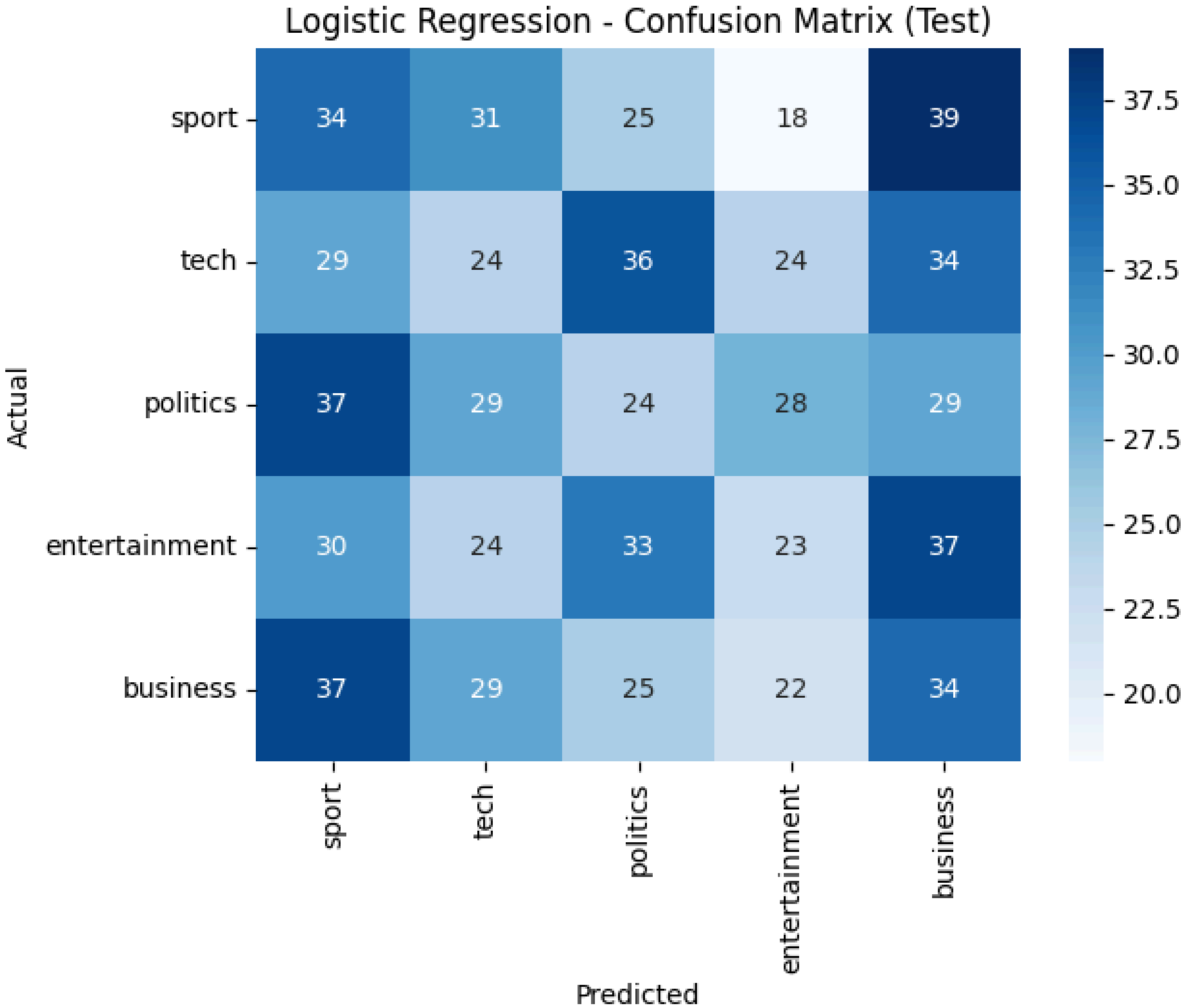
```
Logistic Regression Accuracy (Train): 0.9973
```



Classification Report (Train):

	precision	recall	f1-score	support
sport	1.00	1.00	1.00	336
tech	1.00	1.00	1.00	273
politics	1.00	1.00	1.00	274
entertainment	1.00	1.00	1.00	346
business	0.99	1.00	0.99	261
accuracy			1.00	1490
macro avg	1.00	1.00	1.00	1490
weighted avg	1.00	1.00	1.00	1490

Logistic Regression Accuracy (Test): 0.1891



Classification Report (Test):

	precision	recall	f1-score	support
sport	0.20	0.23	0.21	147
tech	0.20	0.16	0.18	147
politics	0.17	0.16	0.17	147
entertainment	0.20	0.23	0.22	147
business	0.18	0.16	0.17	147
accuracy			0.19	735
macro avg	0.19	0.19	0.19	735
weighted avg	0.19	0.19	0.19	735

Conclusion To begin with, using TF-IDF for text representation proves to be a sound choice. It provides a balance between capturing word relevance within individual documents and across the corpus. This makes it particularly effective for distinguishing between categories in tasks like news classification. When it comes to the NMF model, its performance on the training set was impressive—indicating that it can uncover meaningful latent topics that align closely with actual labels. However, its significantly lower accuracy on the test set suggests it lacks robustness when applied to new data. Since NMF is unsupervised, this drop isn’t surprising; it wasn’t given labels to guide its learning. Instead, it learned structure specific to the training data, which may not generalize well. In contrast, the logistic regression model benefited from labeled data and achieved near-perfect scores on the training set. But this came at a cost: the model overfit the training examples and performed just as poorly as NMF on the test set. This shows that even supervised

Citation https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
https://www.practicaldatascience.org/notebooks/class_3/week_3/33_cleaning_datatypes.html
https://patrickwalls.github.io/mathematicalpython/linear-algebra/eigenvalues-eigenvectors/
https://medium.com/analytics-vidhya/bbc-news-text-classification-a1b2a61af903