Importing the neccesary Libraries

[ ] ↳ *1 cell hidden*

Import Dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files bbc-news-data.csv

- **bbc-news-data.csv**(text/csv) - 5080260 bytes, last modified: 7/31/2020 - 100% done

```
Saving bbc-news-data.csv to bbc-news-data.csv
```

```
import pandas as pd                    #Imports the pandas library as pd
import numpy as np                     #Imports the numpy library for numerical computing

data_file="bbc-news-data.csv"

data = pd.read_csv(data_file,sep='\t')     ## Load the CSV file into a DataFrame
```

```
data.head(10)
```

	category	filename	title	content
0	business	001.txt	Ad sales boost Time Warner profit	Quarterly profits at US media giant TimeWarne...
1	business	002.txt	Dollar gains on Greenspan speech	The dollar has hit its highest level against ...
2	business	003.txt	Yukos unit buyer faces loan claim	The owners of embattled Russian oil giant Yuk...
3	business	004.txt	High fuel prices hit BA's profits	British Airways has blamed high fuel prices f...
4	business	005.txt	Pernod takeover talk lifts Domecq	Shares in UK drinks and food firm Allied Dome...
5	business	006.txt	Japan narrowly escapes recession	Japan's economy teetered on the brink of a te...
6	business	007.txt	Jobs growth still slow in the US	The US created fewer jobs than expected in

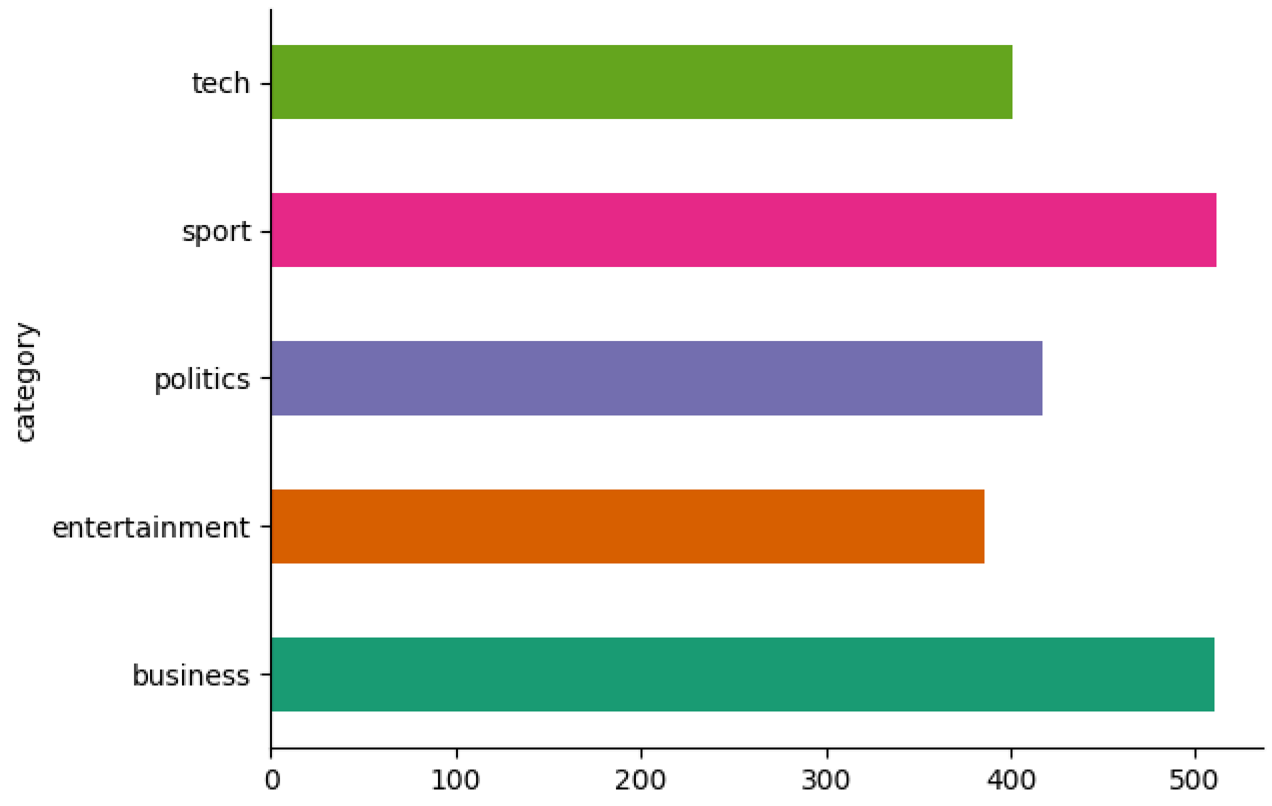
Next steps: Generate code with `data` View recommended plots

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2225 entries, 0 to 2224
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   category  2225 non-null   object
 1   filename  2225 non-null   object
 2   title     2225 non-null   object
 3   content   2225 non-null   object
dtypes: object(4)
memory usage: 69.7+ KB
```

category

```


# @title category

from matplotlib import pyplot as plt
import seaborn as sns
data.groupby('category').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Ydata Profiling

```
from ydata_profiling import ProfileReport
# Generate the data profiling report
ydatareport = ProfileReport(data)
#Display report
ydatareport
```

```
Summarize dataset: 100%                                   13/13 [00:02<00:00,  4.73it/s, Completed]
Generate report structure: 100%                                 1/1 [00:04<00:00,  4.97s/it]
Render HTML: 100%                                         1/1 [00:00<00:00,  2.17it/s]
```

Overview

Brought to you by YData (https://ydata.ai/?utm_source=opensource&utm_medium=ydataprofiling&utm_campaign=report)

Dataset statistics

Number of variables	4
Number of observations	2225
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	69.7 KiB
Average record size in memory	32.1 B

Variable types

Categorical	1
Text	3

Reproduction

Analysis started	2024-07-22 00:09:02.674365

Balancing Dataset

In an imbalanced dataset, models can become biased towards the majority class. This means that the model will predict the majority class more often simply because there are more examples of it, which can result in poor performance on the minority class.So we balanced the dataset by reducing the number of examples in the majority class to match the minority class.

Key Research Question: • How does balancing of dataset affect the performance?

We have used the random sampling and stratified sampling and found that stratified sampling has improved the performance of three classifiers.

```
# Balance the dataset
min_samples = data['category'].value_counts().min()
balanced_data = data.groupby('category').apply(lambda x: x.sample(min_samples)).reset_index(drop
```

Text cleaning/Stop Words Removal

Removing irrelevant characters, punctuation, and extra spaces reduces noise in the data, allowing the model to focus on meaningful information.Converting text to lowercase ensures uniformity, so the model treats "The" and "the" as the same word.Removing stop words (common words like "the", "is", "in") reduces the number of features, focusing the model on more informative words.Clean data can lead to faster and more efficient training because the model has fewer irrelevant features to consider. Reducing feature dimensionality (e.g., through stemming and stop word removal) can lead to better generalization and less overfitting. Lemmatization reduces words to their base or root form. For example, "running" and "ran" both become "run". This normalization helps the model to better understand the context and meaning of words, leading to more accurate predictions.By reducing different forms of a word to a single form, lemmatization reduces the number of unique words (features). This helps the model to generalize better and avoid overfitting.

Text Cleaning without stemming/stop word removal

```
#import re                            #Import module for regular expressions

# Text cleaning function
#def clean_text(text):
#   text = re.sub(r'\s+', ' ', text)  # to remove extra spaces, newlines, and tabs.
#   text = re.sub(r'\W', ' ', text)   # to replace all non-word characters with a space.
#  return text.strip().lower()       # to remove any leading and trailing whitespace from the te
```

Text Cleaning using stemming/stop word removal

```
# Text cleaning function
def clean_text(text):
    text = re.sub(r'\W', ' ', text)
    text = text.lower()
    text = re.sub(r'\s+', ' ', text)
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    #text = ' '.join([word for word in text.split() if word not in stop_words])
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if word not in stop_wor
    return text


# Clean the content
balanced_data['cleaned_content'] = balanced_data['content'].apply(clean_text)
```

TFIDF Vectorization

Max Features in Content(text) Column Unigram

```
# Extract the content column
content = balanced_data['content']

# Use CountVectorizer to count unique terms
vectorizer = CountVectorizer(stop_words='english')
X_counts = vectorizer.fit_transform(content)

# Get the number of unique terms
num_unique_terms = len(vectorizer.get_feature_names_out())

print(f'Number of unique terms (Unigrams): {num_unique_terms}')
```

```
Number of unique terms (Unigrams): 27474
```

Key research question:

• Is TF-idf effective for the features extraction from news articles?

News articles typically have a large vocabulary with many unique words. TF-IDF can handle this by giving more importance to unique and meaningful words and less to common words.News articles usually have sparse data, meaning most words appear only in a few documents. The total number of unique words are 27451 .So we experimented with choosing the optimal number of max features and find that it was giving the best performance at max feature=5000. The more number of feature will use more resources.

```
# Vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(balanced_data['cleaned_content'])
y = balanced_data['category']
```

```python
# Get feature names (terms) from the vectorizer
feature_names = tfidf_vectorizer.get_feature_names_out()

# Convert the TF-IDF matrix to a DataFrame for better readability
tdm_df = pd.DataFrame(X.toarray(), columns=feature_names)

# Display the Term-Document Matrix
print(tdm_df)
```

```
           000  000m   05        10       100  100m  10bn  10m  10th  \
0     0.000000   0.0  0.0  0.000000  0.000000   0.0   0.0  0.0   0.0   
1     0.000000   0.0  0.0  0.000000  0.000000   0.0   0.0  0.0   0.0   
2     0.000000   0.0  0.0  0.090486  0.000000   0.0   0.0  0.0   0.0   
3     0.152794   0.0  0.0  0.000000  0.000000   0.0   0.0  0.0   0.0   
4     0.000000   0.0  0.0  0.016331  0.066307   0.0   0.0  0.0   0.0   
...        ...   ...  ...       ...       ...   ...   ...  ...   ...   
1925  0.000000   0.0  0.0  0.063983  0.000000   0.0   0.0  0.0   0.0   
1926  0.000000   0.0  0.0  0.000000  0.000000   0.0   0.0  0.0   0.0   
1927  0.000000   0.0  0.0  0.000000  0.000000   0.0   0.0  0.0   0.0   
1928  0.000000   0.0  0.0  0.044904  0.000000   0.0   0.0  0.0   0.0   
1929  0.000000   0.0  0.0  0.000000  0.000000   0.0   0.0  0.0   0.0   

            11  ...  youngster  youth  yuan  yugansk  yuganskneftegas  yukos  \
0     0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
1     0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
2     0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
3     0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
4     0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
...        ...  ...        ...    ...   ...      ...              ...    ...   
1925  0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
1926  0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
1927  0.079399  ...        0.0    0.0   0.0      0.0              0.0    0.0   
1928  0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   
1929  0.000000  ...        0.0    0.0   0.0      0.0              0.0    0.0   

      zealand  zero  zombie      zone  
0         0.0   0.0     0.0  0.000000  
1         0.0   0.0     0.0  0.000000  
2         0.0   0.0     0.0  0.000000  
3         0.0   0.0     0.0  0.000000  
4         0.0   0.0     0.0  0.000000  
...       ...   ...     ...       ...  
1925      0.0   0.0     0.0  0.000000  
1926      0.0   0.0     0.0  0.000000  
1927      0.0   0.0     0.0  0.067167  
1928      0.0   0.0     0.0  0.000000  
1929      0.0   0.0     0.0  0.043205  

[1930 rows x 5000 columns]
```

Splitting data into Train-test subset

Stratified sampling is used so that each class (or stratum) is represented proportionally in both the training and test sets. This helps with imbalanced datasets because it maintains the same class distribution across splits, which helps in providing a more reliable estimate of model performance.

```
from sklearn.model_selection import StratifiedShuffleSplit, train_test_split
# Splitting the balanced data
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.25, random_state=42)
for train_index, test_index in sss.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

# Check the distribution of the categories in the test set
print("Category distribution in test set:")
print(y_test.value_counts())
```

```
Category distribution in test set:
category
politics         97
business         97
sport            97
tech             96
entertainment    96
Name: count, dtype: int64
```

X: The feature matrix (TF-IDF transformed text data). y: The labels (news categories). test_size=0.25: 25% of the data is reserved for testing, and 75% for training. random_state=42: A seed value for random number generation to ensure reproducibility. Using the same seed value will always produce the same train-test split.

Model Training and Evaluation

Below are the four fumctions for training and evaluation, confusion matrix, plotting confusion matrix and for cross-validation.

Cross-validation provides a more reliable estimate of a model's performance by averaging the results from multiple train-test splits. This reduces the likelihood of overestimating or underestimating the model's true performance. By using multiple folds, cross-validation reduces the variance associated with a single train-test split, leading to a more stable and reliable performance estimate. We used the 10 fold cross validation for all three algorithms. Ten-fold cross-validation is recommended as a good compromise between computational cost and accuracy estimation." (Kohavi, 1995)

```python
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score
import seaborn as sns

def train_and_evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=y_test.unique())
    cm = confusion_matrix(y_test, y_pred, labels=y_test.unique())
    return accuracy, report, cm, y_pred

# Calculate and print TP, FP, FN, TN
def calculate_metrics(cm):
    for i in range(len(cm)):
        tp = cm[i, i]
        fp = cm[:, i].sum() - tp
        fn = cm[i, :].sum() - tp
        tn = cm.sum() - (tp + fp + fn)
        print(f'Class {i}: TP: {tp}, FP: {fp}, TN: {tn}, FN: {fn}')

def plot_confusion_matrix(cm, classes):
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title('Confusion Matrix')
    plt.show()

# Cross-validation
def cross_validate_model(model, X, y, cv=10):
    scores = cross_val_score(model, X, y, cv=cv)
    print(f"Cross-Validation Scores: {scores}")
    print(f"Mean Accuracy: {scores.mean()}")
    print(f"Standard Deviation: {scores.std()}")
```

Naive Bayes

Naive bayes Model training: Naive Bayes classifiers, particularly the Multinomial Naive Bayes, are well-suited for handling sparse data, such as text data represented as TF-IDF or count vectors.

```python
nb_model = MultinomialNB()
nb_accuracy, nb_report, nb_cm, nb_y_pred = train_and_evaluate_model(
    nb_model, X_train, y_train, X_test, y_test)
```

Evaluation_nb

```python
# Print results
print("Naive Bayes Classifier:")
print("Accuracy:", nb_accuracy)
print("Classification Report:\n", nb_report)
```

```
Naive Bayes Classifier:
Accuracy: 0.9772256728778468
```

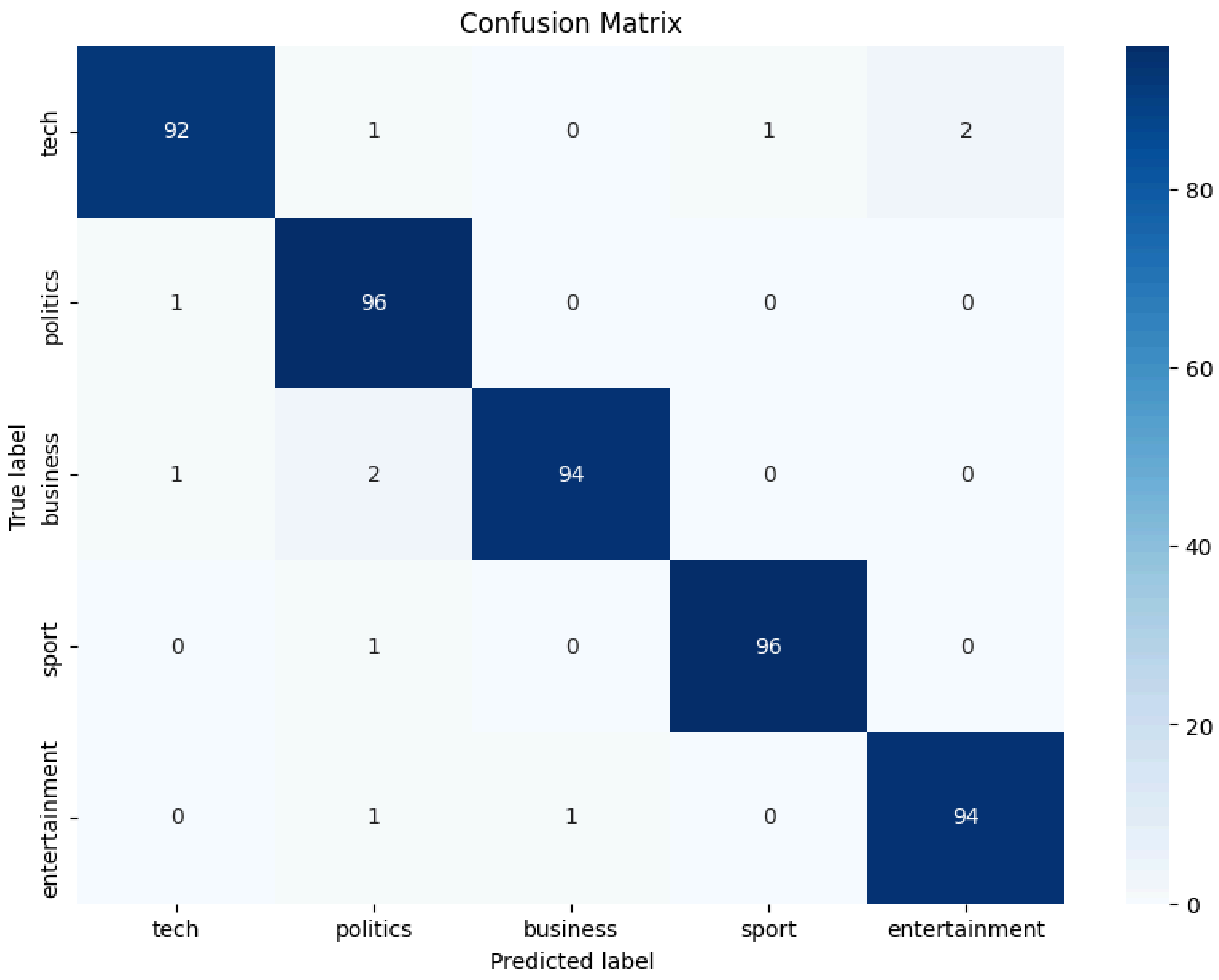
```
Classification Report:
                precision    recall  f1-score   support

         tech       0.99      0.97      0.98        97
     politics       0.98      0.98      0.98        96
     business       0.95      0.99      0.97        97
        sport       0.99      0.99      0.99        97
entertainment       0.98      0.96      0.97        96

     accuracy                           0.98       483
    macro avg       0.98      0.98      0.98       483
 weighted avg       0.98      0.98      0.98       483
```

```

print("\nMetrics per Class:")
calculate_metrics(nb_cm)
```

```
Metrics per Class:
Class 0: TP: 92, FP: 2, TN: 385, FN: 4
Class 1: TP: 96, FP: 5, TN: 381, FN: 1
Class 2: TP: 94, FP: 1, TN: 385, FN: 3
Class 3: TP: 96, FP: 1, TN: 385, FN: 1
Class 4: TP: 94, FP: 2, TN: 385, FN: 2
```

The above confusion matrix is for five news categories. Class 1 is tech and it shows that out of total 483 samples used for testing. It predicted 93 out of 97 tech texts correctly, and gave 381 True negatives.

```
# Plot the confusion matrix
plot_confusion_matrix(nb_cm, classes=y_test.unique())
```



Cross Validation_nb

We used the 10 fold cross validation for all three algorithms. Ten-fold cross-validation is recommended as a good compromise between computational cost and accuracy estimation." (Kohavi, 1995)

```
print("Naive Bayes Classifier (Cross-Validation):")
cross_validate_model(nb_model, X, y)
```

```
Naive Bayes Classifier (Cross-Validation):
Cross-Validation Scores: [0.97927461 0.98963731 0.99481865 0.97927461 0.96373057 0.96373057
 0.95854922 0.98445596 0.96891192 0.97409326]
Mean Accuracy: 0.9756476683937823
Standard Deviation: 0.011363581450498086
```

Decision Tree

Decision trees do not handle sparse data as efficiently as models like Naive Bayes or linear models. Sparse matrices often contain many zero entries, which may not provide useful information for splits. So we desparse the sparse matrix to a dense format for decision tree implementation and find that perfromance was improved.

```
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()
```

Decsion tree model training

```
dt_model = DecisionTreeClassifier(random_state=42)
dt_accuracy, dt_report, dt_cm, dt_y_pred = train_and_evaluate_model(
    dt_model, X_train, y_train, X_test, y_test)
```

Evaluation_dt

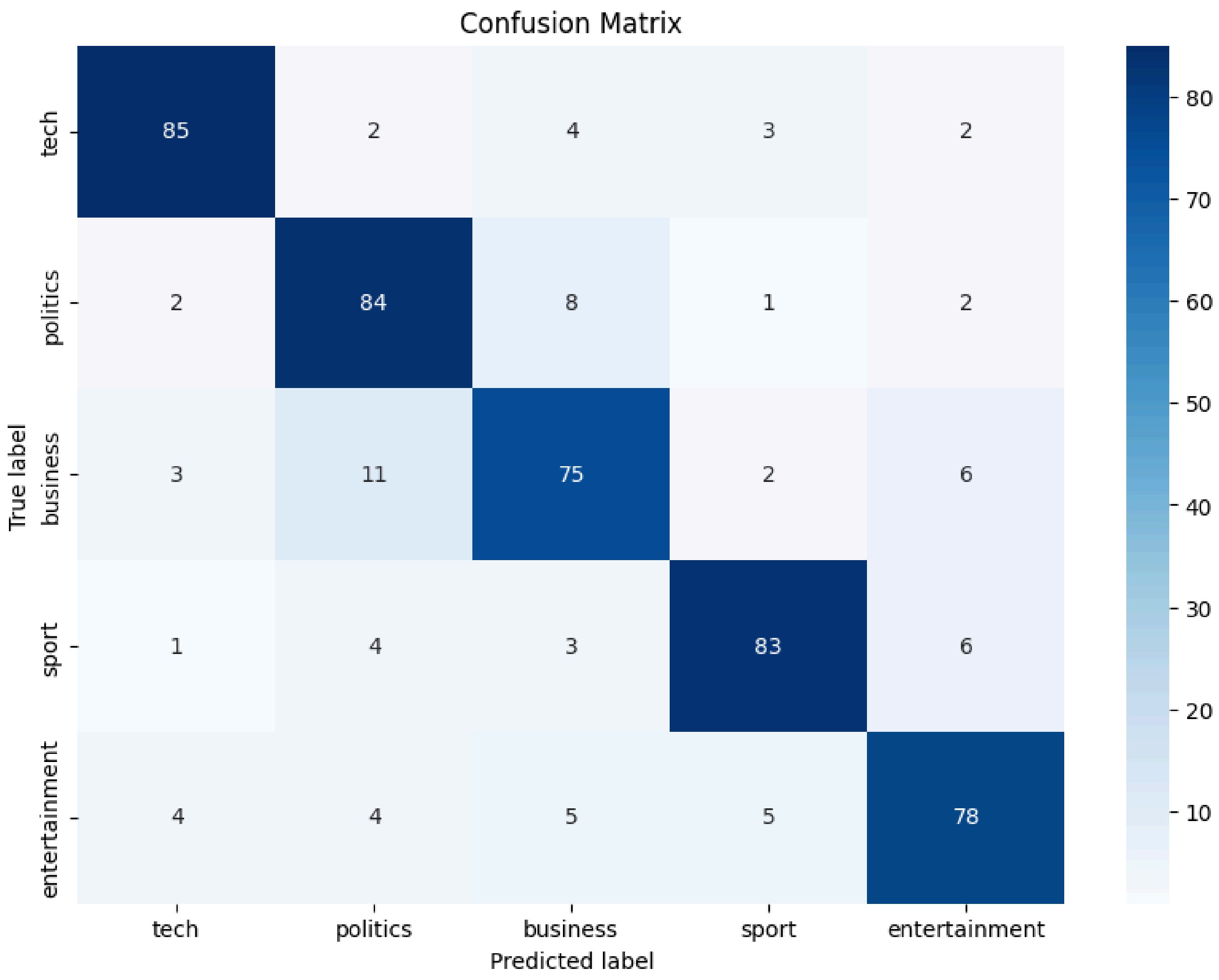
```
print("Decision Tree Classifier:")
print("Accuracy:", dt_accuracy)
print("Classification Report:\n", dt_report)
```

```
Decision Tree Classifier:
Accuracy: 0.8385093167701864
Classification Report:
                precision    recall  f1-score   support

         tech       0.79      0.77      0.78        97
     politics       0.83      0.81      0.82        96
     business       0.80      0.87      0.83        97
        sport       0.88      0.86      0.87        97
entertainment       0.89      0.89      0.89        96

     accuracy                           0.84       483
    macro avg       0.84      0.84      0.84       483
 weighted avg       0.84      0.84      0.84       483
```

```
print("\nMetrics per Class:")
calculate_metrics(dt_cm)
```

```

Metrics per Class:
Class 0: TP: 85, FP: 10, TN: 377, FN: 11
Class 1: TP: 84, FP: 21, TN: 365, FN: 13
Class 2: TP: 75, FP: 20, TN: 366, FN: 22
Class 3: TP: 83, FP: 11, TN: 375, FN: 14
Class 4: TP: 78, FP: 16, TN: 371, FN: 18
```

```
# Plot the confusion matrix
plot_confusion_matrix(dt_cm, classes=y_test.unique())
```



Cross Validation_dt

```
print("Decision Tree Classifier (Cross-Validation):")
cross_validate_model(dt_model, X, y)
```

```
Decision Tree Classifier (Cross-Validation):
Cross-Validation Scores: [0.88082902 0.83937824 0.88601036 0.84455959 0.83937824 0.82901554
 0.83937824 0.86010363 0.88601036 0.83419689]
Mean Accuracy: 0.8538860103626942
Standard Deviation: 0.021338093556449764
```

Random Forest

```
# Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_accuracy, rf_report, rf_cm, rf_y_pred = train_and_evaluate_model(
    rf_model, X_train, y_train, X_test, y_test)
```

Evaluation-rf

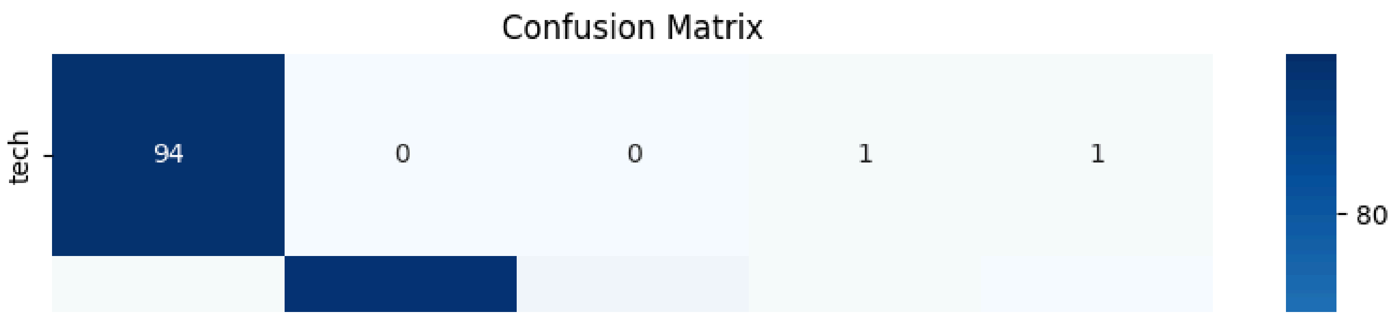
```
print("Random Forest Classifier:")
print("Accuracy:", rf_accuracy)
print("Classification Report:\n", rf_report)
```

```
Random Forest Classifier:
Accuracy: 0.9730848861283644
Classification Report:
                precision    recall  f1-score   support

         tech       0.96      0.97      0.96        97
     politics       0.99      0.98      0.98        96
     business       0.96      0.96      0.96        97
        sport       0.97      0.98      0.97        97
entertainment       0.99      0.98      0.98        96

     accuracy                           0.97       483
    macro avg       0.97      0.97      0.97       483
 weighted avg       0.97      0.97      0.97       483
```

```
print("\nMetrics per Class:")
calculate_metrics(rf_cm)
```

```

Metrics per Class:
Class 0: TP: 94, FP: 1, TN: 386, FN: 2
Class 1: TP: 93, FP: 4, TN: 382, FN: 4
Class 2: TP: 94, FP: 4, TN: 382, FN: 3
Class 3: TP: 95, FP: 3, TN: 383, FN: 2
Class 4: TP: 94, FP: 1, TN: 386, FN: 2
```

```
# Plot the confusion matrix
plot_confusion_matrix(rf_cm, classes=y_test.unique())
```



Cross Validation_rf

```
print("Random Forest Classifier (Cross-Validation):")
cross_validate_model(rf_model, X, y)

```

```
Random Forest Classifier (Cross-Validation):
Cross-Validation Scores: [0.97409326 0.97409326 0.97927461 0.96373057 0.96373057 0.97409326
 0.96373057 0.96891192 0.96373057 0.96891192]
Mean Accuracy: 0.9694300518134715
Standard Deviation: 0.005409485237777506
```

Comparison

We compared the performance of three classifiers and found that Naive bayes and Random forest outperfromed the decision tree in terms of accuracy,precision and recall. This answers one of our key research questions:

- **How efficient are Naïve Bayes, Decision Tree and Random Forest for the news article classification?**