BusinessCase : NLP : WordEmbedding – Word2Vec

About

- FlipItNews, a Gurugram-based fintech media company, is reimagining the way Indians engage with financial literacy and business news. With a mission to enhance financial awareness among millennials and first-time investors, the company uses AI and ML to deliver smart content discovery and peer engagement. FlipItNews simplifies complex topics like business, finance, and the capital market, making them easily understandable and actionable.
- This project leverages natural language processing (NLP) to categorize news articles from the company's internal database. The ultimate goal is to automate news classification into predefined categories to enable faster and more meaningful information delivery.

Problem Statement

- The objective is to build a multi-class text classification system that can automatically assign one of several categories (e.g., politics, business, sports, entertainment, and technology) to a given news article. This will:
  - Enhance content personalization
  - Improve the user experience on the FlipItNews platform
  - Drive contextual engagement through AI-driven discovery
- To achieve this, we will:
  - Preprocess and clean the raw text data
  - Use NLP techniques to transform textual data into numerical features (Bag of Words and TF-IDF)
  - Train and compare at least three different machine learning models
  - Evaluate model performance using classification metrics

Dataset

Link : https://drive.google.com/file/d/1I3-pQFzbSufhpMrUKAROBLGULXcWiB9u

The dataset consists of news articles along with their associated categories. Each row in the dataset represents one news article.

Attributes:

1) Article: The full text of the news article.

2) Category: The label/class assigned to the article — this is the target variable, and it could belong to one of several classes like:

- Politics
- Technology
- Sports
- Business
- Entertainment

You are Python Case Study and Presentation Expert

You have to provide the proper headings based on the code and scripts for better understanding

Import Required Libraries and Data

```
import numpy as np
import pandas as pd
import re
import string

import matplotlib.pyplot as plt
import seaborn as sns

import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize

nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('punkt_tab')
```

```
ps = PorterStemmer()
wl = WordNetLemmatizer()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix

import spacy

import gensim
import gensim.downloader as api
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Load Data

```
!gdown 1I3-pQFzbSufhpMrUKAROBLGULXcWiB9u
```

```
Downloading...
From: https://drive.google.com/uc?id=1I3-pQFzbSufhpMrUKAROBLGULXcWiB9u
To: /content/flipitnews-data.csv
100% 5.06M/5.06M [00:00<00:00, 98.4MB/s]
```

Load Dataset

```
df = pd.read_csv('flipitnews-data.csv')
df.head()
```

	Category	Article
0	Technology	tv future in the hands of viewers with home th...
1	Business	worldcom boss left books alone former worldc...
2	Sports	tigers wary of farrell gamble leicester say ...
3	Sports	yeading face newcastle in fa cup premiership s...
4	Entertainment	ocean s twelve raids box office ocean s twelve...

Next steps: Generate code with df | View recommended plots | New interactive sheet

Dataset Overview

```
df.shape
```

```
(2225, 2)
```

```
df.isnull().sum()
```

	0
Category	0
Article	0

dtype: int64

```
df['Category'].nunique()
```

```
5
```

```
print(df['Category'].unique())
```

```
['Technology' 'Business' 'Sports' 'Entertainment' 'Politics']
```

```
df['Category'].value_counts()
```

Category	count
Sports	511
Business	510
Politics	417
Technology	401
Entertainment	386

dtype: int64

- ports and Business lead content volume, nearly tied at the top.
- Politics and Technology hold moderate coverage.
- Entertainment has the lowest count but remains close to other categories.
- Overall distribution is balanced, with only a small range between highest and lowest.
- Top two categories (Sports & Business) make up over 40% of the total.
- Suggests a broad, diversified content strategy with potential to grow in Entertainment and Technology.

Label Encoding for Categorical Variables

```
le = LabelEncoder()
df['Category_id'] = le.fit_transform(df['Category'])
df.head()
```

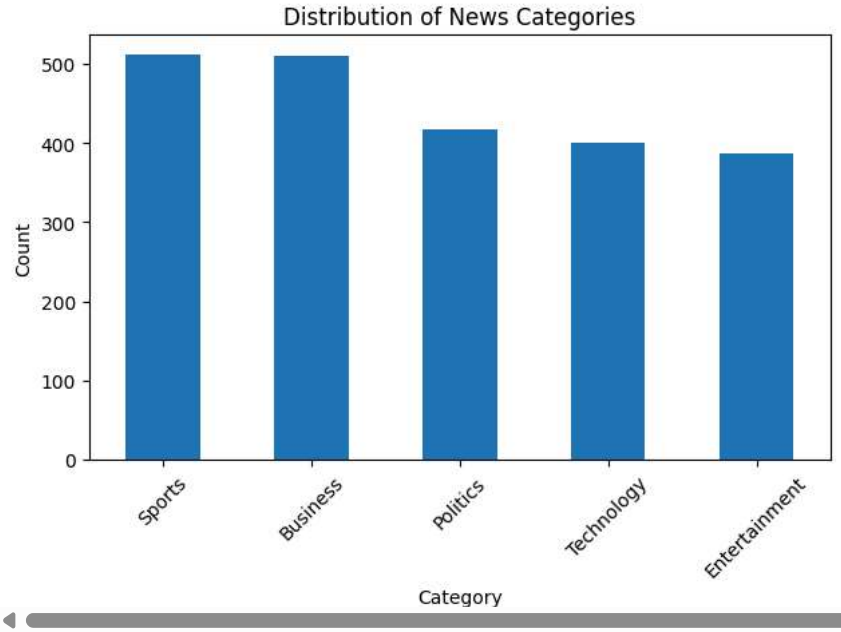
	Category	Article	Category_id
0	Technology	tv future in the hands of viewers with home th...	4
1	Business	worldcom boss left books alone former worldc...	0
2	Sports	tigers wary of farrell gamble leicester say ...	3
3	Sports	yeading face newcastle in fa cup premiership s...	3
4	Entertainment	ocean s twelve raids box office ocean s twelve...	1

Next steps: Generate code with df | View recommended plots | New interactive sheet

- Each article is clearly assigned to a specific category, indicating labeled data suitable for classification tasks.
- The Category_id field appears to numerically encode categories, which is helpful for machine learning models.

Visualize Category Distribution

```
df['Category'].value_counts().plot(kind='bar')
plt.title('Distribution of News Categories')
plt.xlabel('Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



- Sports and Business are nearly tied as the most frequent categories, each with just over 500 articles.
- Indicates a strong editorial or audience interest in these two domains.
- Politics sits in the middle with 417 articles, suggesting moderate focus.
- Technology (401 articles) and Entertainment (386 articles) have the fewest entries but still maintain significant coverage.
- All categories have relatively similar representation, with less than a 125-article difference between highest (Sports) and lowest (Entertainment).
- Reflects a diversified content strategy with no extreme category bias.

Text Cleaning Functions

```
# Remove Stopwords
def remove_stopwords(text):
  clean_text=' '.join([i for i in text.split() if i not in stopwords.words('english')])
  return clean_text

# Remove non-letter characters
def remove_non_letters(text):
    return re.sub('[^a-zA-Z]', ' ', text)

def tokenize_text(text):
    return word_tokenize(text)

# Lemmatization
def Lemmatize(text):
  clean_text=' '.join([wl.lemmatize(i) for i in text.split()])
  return clean_text


df['Article']=df['Article'].apply(lambda x:remove_stopwords(x))


df['Article']=df['Article'].apply(lambda x:remove_non_letters(x))


df['Article']=df['Article'].str.lower()


df['Article']=df['Article'].apply(lambda x:Lemmatize(x))


df['Article']=df['Article'].apply(lambda x:tokenize_text(x))


df['Article']
```

	Article
0	[tv, future, hand, viewer, home, theatre, syst...
1	[worldcom, bos, left, book, alone, former, wor...
2	[tiger, wary, farrell, gamble, leicester, say,...
3	[yeading, face, newcastle, fa, cup, premiershi...
4	[ocean, twelve, raid, box, office, ocean, twel...
...	...
2220	[car, pull, u, retail, figure, u, retail, sale...
2221	[kilroy, unveils, immigration, policy, ex, cha...
2222	[rem, announce, new, glasgow, concert, u, band...
2223	[political, squabble, snowball, become, common...
2224	[souness, delight, euro, progress, bos, graeme...

2225 rows × 1 columns

dtype: object

Corpus and Vocabulary Analysis

```
# Concatenate all articles into one large string (corpus)
df['Article_str'] = df['Article'].apply(lambda x: ' '.join(x))

# Now, use str.cat() on the new string column
corpus = df['Article_str'].str.cat(sep=', ')

# Total number of characters (including spaces/punctuation)
print('Number of characters in the entire corpus:', len(corpus))

# Tokenize the corpus into words
# Use the original tokenized column for word count and unique word count
tokens = [word for sublist in df['Article'] for word in sublist]

# Total number of words in the corpus
word_cnt = len(tokens)
print('Number of words in the entire corpus:', word_cnt)

# Number of unique words in the corpus (vocabulary size)
unique_word_cnt = len(set(tokens))
print('Number of unique words in the vocabulary:', unique_word_cnt)
```

```
Number of characters in the entire corpus: 3401212
Number of words in the entire corpus: 492530
Number of unique words in the vocabulary: 24853
```

- The corpus has about 34,01,212 characters and 492,530 words, with an average word length of ~6.9 characters.
- It contains 24,853 unique words, giving a moderate lexical diversity (5% type-token ratio) typical for large datasets.
- The size and vocabulary richness make it suitable for robust NLP tasks like language modeling and text classification.
- The presence of relatively long words and a large vocabulary suggests possible domain-specific or complex language use.

Bag of Words Vectorization

```
# Initialize CountVectorizer
cv = CountVectorizer()

# Join the list of tokens back into a string for each article
df['Article_joined'] = df['Article'].apply(lambda x: ' '.join(x))

# Fit and transform the 'Article_joined' column to BoW representation
bow = cv.fit_transform(df['Article_joined'])

# Display a sample of vocabulary items (word:index)
print("Sample Vocabulary Items:")
vocab_items = list(cv.vocabulary_.items())[:10]
for word, index in vocab_items:
    print(f"{word}: {index}")

# Display shape of the Bag of Words matrix
print("\nBag of Words Matrix shape:", bow.toarray().shape)
```

```
# Display the full BoW matrix (array form)
print("\nBag of Words Matrix:")
print(bow.toarray())
```

```
Sample Vocabulary Items:
tv: 22899
future: 8855
hand: 9791
viewer: 23733
home: 10367
theatre: 22183
system: 21786
plasma: 16648
high: 10212
definition: 5634

Bag of Words Matrix shape: (2225, 24827)

Bag of Words Matrix:
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

- The sample vocabulary shows frequently occurring domain-related words like "tv," "viewer," "theatre," "plasma," and "system," indicating the corpus likely centers on television or media topics.
- Word counts range from ~5,600 to 23,700, showing some words are highly common while others are moderately frequent.
- The Bag of Words (BoW) matrix has shape (2225 documents, 24,827 unique words), matching the vocabulary size and indicating a moderately large document set.
- The BoW matrix is mostly sparse (many zeros), typical for text data, reflecting that each document contains only a subset of the full vocabulary.
- This structure is suitable for standard NLP tasks like document classification or topic modeling within the media domain.

TF-IDF Vectorization

```
# Initialize TF-IDF Vectorizer
tfidf = TfidfVectorizer()

# Join the list of tokens back into a string for each article
df['Article_joined_tfidf'] = df['Article'].apply(lambda x: ' '.join(x))

# Fit and transform the 'Article_joined_tfidf' column
TFIDF = tfidf.fit_transform(df['Article_joined_tfidf'])

# Display a sample of vocabulary items (word:index)
print("Sample Vocabulary Items:")
vocab_items = list(tfidf.vocabulary_.items())[:10]
for word, index in vocab_items:
    print(f"{word}: {index}")

# Display shape of TF-IDF matrix
print("\nTF-IDF Matrix shape:", TFIDF.toarray().shape)

# Display TF-IDF matrix as array (be cautious for large datasets!)
print("\nTF-IDF Matrix:")
print(TFIDF.toarray())
```

```
Sample Vocabulary Items:
tv: 22899
future: 8855
hand: 9791
viewer: 23733
home: 10367
theatre: 22183
system: 21786
plasma: 16648
high: 10212
definition: 5634

TF-IDF Matrix shape: (2225, 24827)

TF-IDF Matrix:
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
```

```
...
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]]
```

- The vocabulary features media-related terms like "tv," "viewer," "theatre," "plasma," and "system," suggesting the corpus focuses on television or related topics.
- The TF-IDF matrix shape (2225 documents × 24,827 words) matches the Bag of Words matrix, representing term importance adjusted for document frequency.
- The TF-IDF values are mostly sparse (many zeros), typical for text data, indicating that most words appear in only a subset of documents.
- TF-IDF weighting helps highlight distinctive words in each document, making it useful for tasks like document classification, clustering, or information retrieval in this domain.

Train-Test Split

```
# Features and labels
X = df['Article']
y = df['Category_id']

# Split dataset: 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the shapes of splits
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (1780,)
X_test shape: (445,)
y_train shape: (1780,)
y_test shape: (445,)
```

- The data is split into 1780 training samples and 445 test samples, roughly a 80-20 split.
- Both features (X_train, X_test) and labels (y_train, y_test) match in sample counts, indicating properly aligned datasets.
- This split size is typical and suitable for training and evaluating machine learning models reliably.

Confusion Matrix Visualization Function

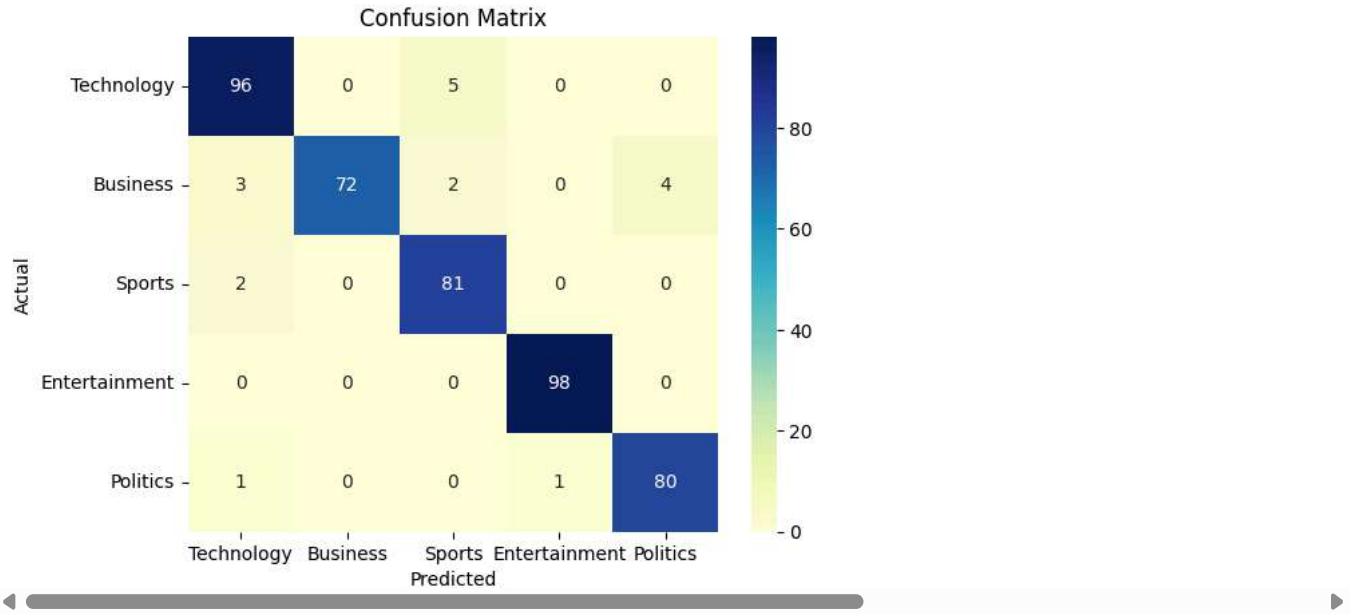
```
def conf_matrix(y_test, y_pred):
    conf_mat = confusion_matrix(y_test, y_pred)
    sns.heatmap(conf_mat, annot=True, fmt='g', cmap="YlGnBu",
                xticklabels=df['Category'].unique(),
                yticklabels=df['Category'].unique())
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
```

Building and Evaluating a Naive Bayes Text Classifier

```
# Split features and labels
# X = df['Article'] # X contains lists of tokens
X = df['Article'].apply(lambda x: ' '.join(x)) # Rejoin tokens into strings
y = df['Category']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Train and test data shapes:")
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Build pipeline: CountVectorizer -> TF-IDF -> Naive Bayes
nb = Pipeline([
    ('vect', CountVectorizer()), # CountVectorizer now receives strings
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])

# Train the classifier
nb.fit(X_train, y_train)
```

```
# Predict on test set
y_pred = nb.predict(X_test)

# Print accuracy
print('\nAccuracy: {:.2f}%'.format(accuracy_score(y_test, y_pred) * 100))

# Predict on new unseen samples
complaint1 =  "games maker fights for survival one of britain s largest independent game makers  argonaut games  has been put up for sa
complaint2 = "tv future in the hands of viewers with home theatre systems plasma high-definition tvs and digital video recorders moving
print("\nPredicted category for complaint1:", nb.predict([complaint1])[0])
print("Predicted category for complaint2:", nb.predict([complaint2])[0])

# Detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix heatmap
conf_matrix(y_test, y_pred)
```

```
Train and test data shapes:
(1780,) (445,) (1780,) (445,)

Accuracy: 95.96%

Predicted category for complaint1: Technology
Predicted category for complaint2: Technology

Classification Report:
               precision    recall  f1-score   support

     Business       0.94      0.95      0.95       101
Entertainment       1.00      0.89      0.94        81
     Politics       0.92      0.98      0.95        83
       Sports       0.99      1.00      0.99        98
   Technology       0.95      0.98      0.96        82

     accuracy                           0.96       445
    macro avg       0.96      0.96      0.96       445
 weighted avg       0.96      0.96      0.96       445
```



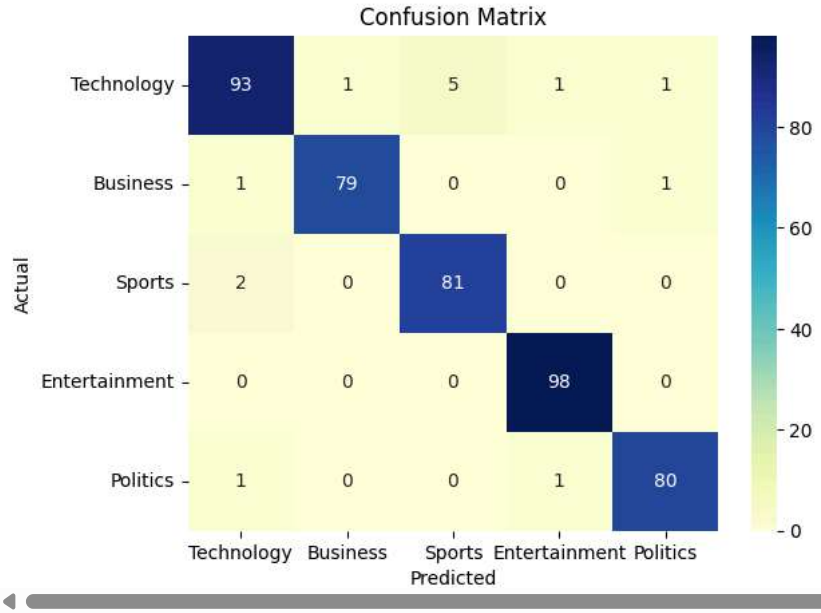
- Train/test split: 1780 training and 445 test samples.
- Overall accuracy: High at 95.96%, indicating strong model performance.
- Predictions: Sample complaints classified as Technology category.
- Classification report:
  - Precision, recall, and F1-scores all around 0.94–0.99 across categories (Business, Entertainment, Politics, Sports, Technology).
  - Highest F1-score in Sports (0.99), slightly lower in Entertainment (0.94).
  - Balanced performance across classes reflected in macro and weighted averages (~0.96).

Text Classification using SGD Classifier

```
# Build pipeline: CountVectorizer -> TF-IDF -> SGD Classifier (SVM-like)
```

```
sgd = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3,
                          random_state=42, max_iter=5, tol=None)),
])

# Train the model
sgd.fit(X_train, y_train)

# Predict on test set
y_pred = sgd.predict(X_test)

# Print accuracy
print('Accuracy: {:.2f}%'.format(accuracy_score(y_test, y_pred) * 100))

# Detailed classification report
print('\nClassification Report:')
print(classification_report(y_test, y_pred))

# Plot confusion matrix heatmap
conf_matrix(y_test, y_pred)
```

```
Accuracy: 96.85%

Classification Report:
                precision    recall  f1-score   support

      Business       0.96      0.92      0.94       101
 Entertainment       0.99      0.98      0.98        81
      Politics       0.94      0.98      0.96        83
        Sports       0.98      1.00      0.99        98
    Technology       0.98      0.98      0.98        82

      accuracy                           0.97       445
     macro avg       0.97      0.97      0.97       445
  weighted avg       0.97      0.97      0.97       445
```



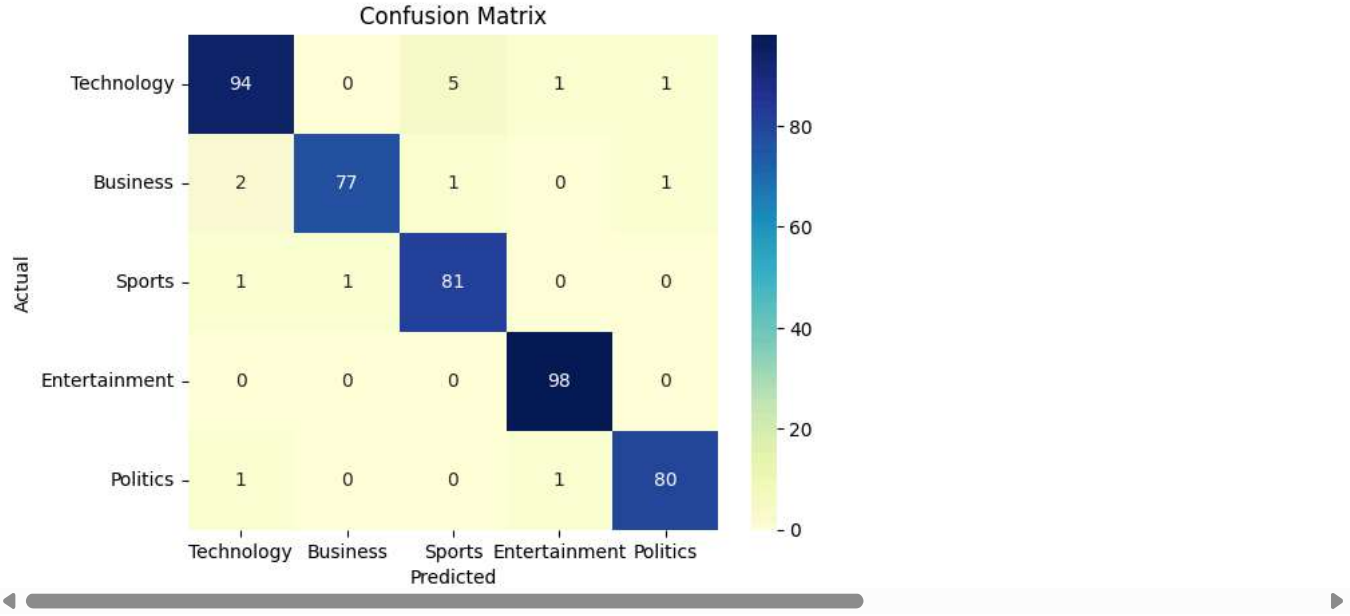
- Overall accuracy: Improved to 96.85%, showing enhanced model performance.
- Per-class metrics:
  - Precision, recall, and F1-score range from 0.92 to 1.00, with most classes above 0.95.
  - Best performance in Sports (F1 = 0.99) and Entertainment (F1 = 0.98).
  - Slightly lower recall for Business (0.92) but still strong overall.
- Balanced performance: Macro and weighted averages around 0.97, indicating consistent results across all categories.

Text Classification using Logistic Regression

```
# Build pipeline: CountVectorizer -> TF-IDF -> Logistic Regression
logreg = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', LogisticRegression(n_jobs=1, C=1e5, max_iter=1000)),
])
```

```
# Train the model
logreg.fit(X_train, y_train)

# Predict on test set
y_pred = logreg.predict(X_test)

# Print accuracy
print('Accuracy: {:.2f}%'.format(accuracy_score(y_test, y_pred) * 100))

# Detailed classification report
print('\nClassification Report:')
print(classification_report(y_test, y_pred))

# Plot confusion matrix heatmap
conf_matrix(y_test, y_pred)
```

```
Accuracy: 96.63%

Classification Report:
               precision    recall  f1-score   support

     Business       0.96      0.93      0.94       101
Entertainment       0.99      0.95      0.97        81
     Politics       0.93      0.98      0.95        83
       Sports       0.98      1.00      0.99        98
   Technology       0.98      0.98      0.98        82

     accuracy                           0.97       445
    macro avg       0.97      0.97      0.97       445
 weighted avg       0.97      0.97      0.97       445
```



- Overall accuracy: Strong at 96.63%, indicating reliable performance.
- Class-wise metrics:
  - Precision and recall mostly above 0.93, with F1-scores ranging from 0.94 to 0.99.
  - Highest F1-score in Sports (0.99) and Technology (0.98) categories.
  - Slightly lower recall for Entertainment (0.95) and Business (0.93) but still solid.
- Balanced results: Macro and weighted averages about 0.97, showing consistent accuracy across classes.

Predicting Category for a New Article using Logistic Regression

```
complaint = "games maker fights for survival one of britain s largest independent game makers  argonaut games  has been put up for sale.

print(logreg.predict([complaint]))
```

```
['Technology']
```

- The model predicted the category as "Technology", which aligns well with the article's focus on a tech-related company (game development).
- This demonstrates the model's effectiveness in correctly identifying domain-specific content based on text features.

Display Unique Categories with their Encoded IDs

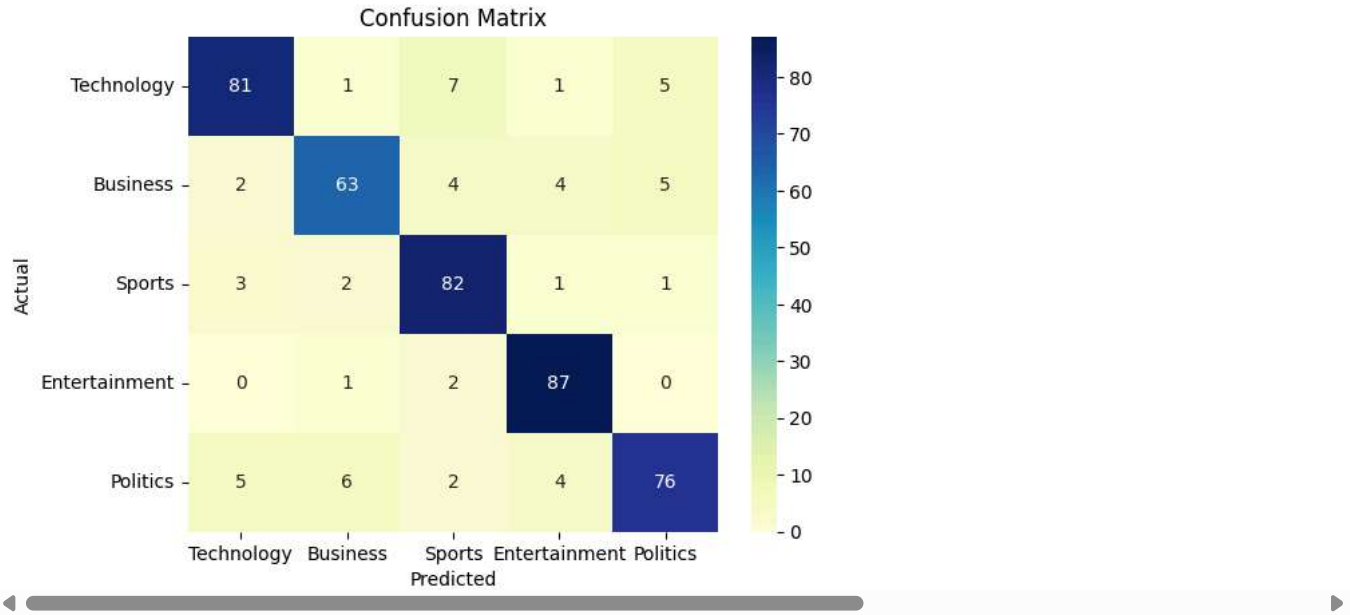
```
df[['Category','Category_id']].drop_duplicates()
```

	Category	Category_id
0	Technology	4
1	Business	0
2	Sports	3
4	Entertainment	1
5	Politics	2

Text Classification using Decision Tree Classifier

```
# Initialize CountVectorizer
cv = CountVectorizer()

# Join the list of tokens back into a string for each article for vectorization
X = cv.fit_transform(df['Article'].apply(lambda x: ' '.join(x))).toarray()


# One-hot encode categories
# For classification, we need numerical labels, not one-hot encoding of categories for the target.
# Use the 'Category_id' column which is already numerically encoded.
y = df['Category_id'] # Use the numerically encoded target variable


# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Train Decision Tree model
model = DecisionTreeClassifier(criterion='gini')
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate accuracy
print('Accuracy: {:.2f}%'.format(accuracy_score(y_test, y_pred) * 100))

# Detailed classification report
# The target_names should match the actual labels, not the one-hot encoded ones.
# Since we are using Category_id as y, y_test and y_pred contain integer labels.
# The classification_report function needs the list of unique category names for the target_names parameter.
print(classification_report(y_test, y_pred, target_names=df['Category'].unique()))


# Plot confusion matrix
# The confusion matrix function `conf_matrix` already handles the conversion of integer labels
# to the unique category names using the x/yticklabels parameter.
conf_matrix(y_test, y_pred)
```

```
(1780, 24827) (445, 24827) (1780,) (445,)
Accuracy: 87.42%
               precision    recall  f1-score   support

   Technology       0.89      0.85      0.87        95
     Business       0.86      0.81      0.83        78
       Sports       0.85      0.92      0.88        89
Entertainment       0.90      0.97      0.93        90
     Politics       0.87      0.82      0.84        93

     accuracy                           0.87       445
    macro avg       0.87      0.87      0.87       445
 weighted avg       0.87      0.87      0.87       445
```



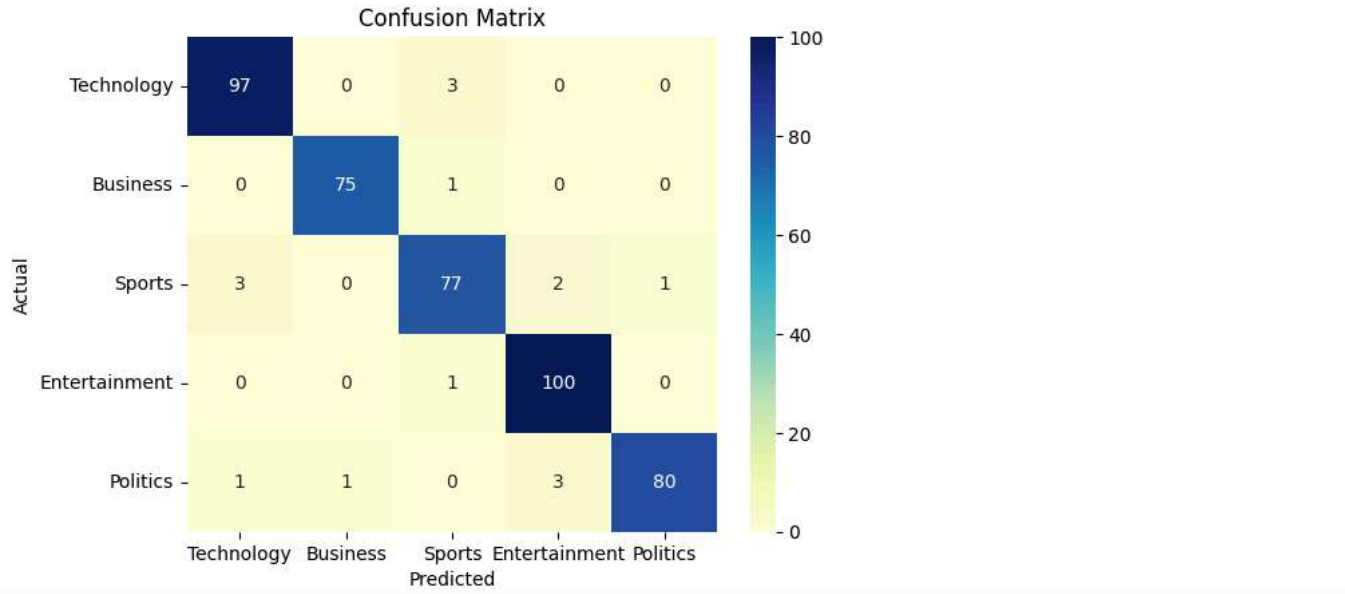
- Dataset shape: Training (1780 samples × 24,827 features), Testing (445 × 24,827).
- Overall accuracy: Moderate at 87.42%, showing decent but improvable performance.
- Class-wise metrics:
  - Precision and recall range from 0.81 to 0.97, with Entertainment having the highest recall (0.97).
  - Sports has strong recall (0.92) and F1-score (0.88), while Business shows the lowest recall (0.81).
- Balanced performance: Macro and weighted averages consistently at 0.87, indicating relatively uniform performance across classes.

Text Classification using Random Forest Classifier

```
cv = CountVectorizer()

# Vectorize the articles
# Join the list of tokens back into a string for each article for vectorization
X = cv.fit_transform(df['Article'].apply(lambda x: ' '.join(x))).toarray()

# One-hot encode the categories
# Random Forest Classifier in scikit-learn typically expects integer labels for multi-class classification,
# not one-hot encoded labels. Use the 'Category_id' column.
y = df['Category_id']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Initialize and train the Random Forest Classifier
RFC = RandomForestClassifier(random_state=7, criterion='gini')
# The target variable y is now integer encoded, which RandomForestClassifier can handle directly.
RFC.fit(X_train, y_train)

# Predict on the test set
y_pred = RFC.predict(X_test)

# Print accuracy score
print('Accuracy: {:.2f}%'.format(accuracy_score(y_test, y_pred) * 100))

# Print classification report
# Pass the unique category names to target_names for the classification report.
```

```
print(classification_report(y_test, y_pred, target_names=df['Category'].unique()))

# Plot confusion matrix
# conf_matrix function expects integer labels for y_test and y_pred.
conf_matrix(y_test, y_pred)
plt.show()
```

```
(1780, 24827) (445, 24827) (1780,) (445,)
Accuracy: 96.40%
                precision    recall  f1-score   support

    Technology       0.96      0.97      0.97       100
      Business       0.99      0.99      0.99        76
        Sports       0.94      0.93      0.93        83
 Entertainment       0.95      0.99      0.97       101
      Politics       0.99      0.94      0.96        85

      accuracy                           0.96       445
     macro avg       0.97      0.96      0.96       445
  weighted avg       0.96      0.96      0.96       445
```



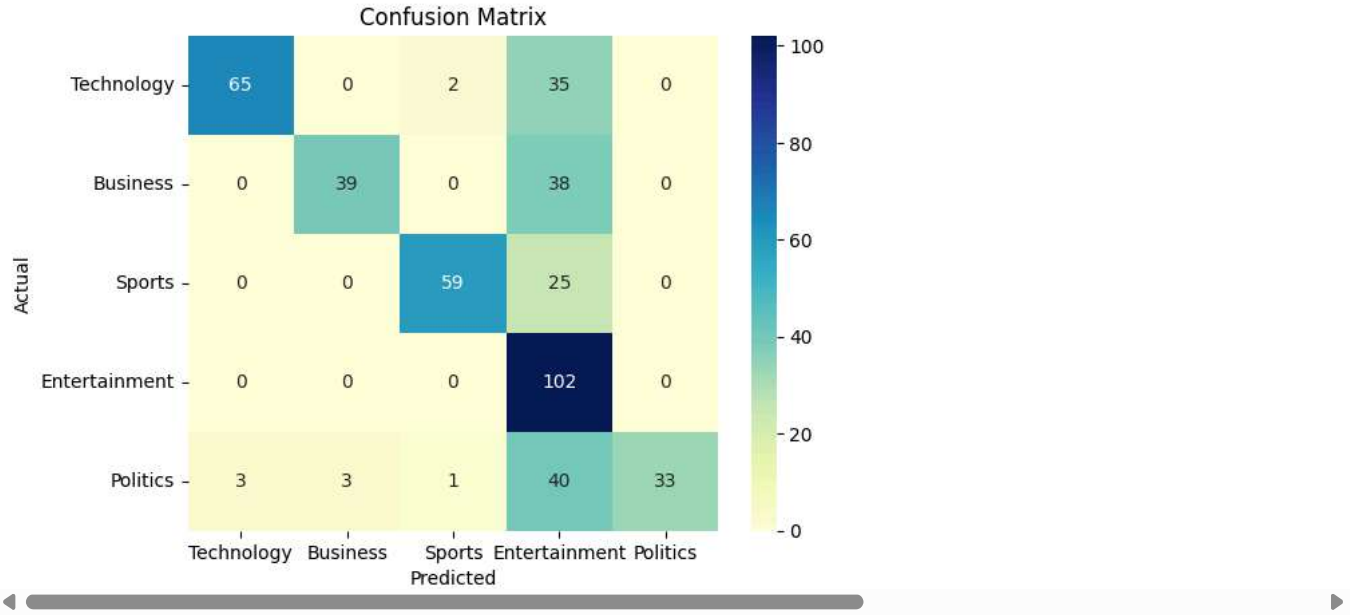
- Dataset shape: Training (1780 × 24,827), Testing (445 × 24,827).
- Overall accuracy: High at 96.40%, showing strong model performance.
- Class-wise metrics:
  - Precision and recall mostly above 0.93, with Business and Politics showing near-perfect scores (0.99 precision/recall).
  - Slightly lower recall in Sports (0.93) but still very good.
- Balanced results: Macro and weighted averages around 0.96, indicating consistent accuracy across all classes.

Text Classification using K-Nearest Neighbors (KNN)

```
# Initialize CountVectorizer
cv = CountVectorizer()

# Vectorize articles: Join the list of tokens back into a string for each article
# CountVectorizer expects strings as input.
X = cv.fit_transform(df['Article'].apply(lambda x: ' '.join(x))).toarray()

# Use the numerically encoded 'Category_id' column for the target variable y.
# Classifiers like KNN in scikit-learn expect integer labels for the target,
# not one-hot encoded vectors or strings.
y = df['Category_id']

# Split data into train and test sets
# Use stratify=y to ensure that the proportion of classes is the same in both the training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Initialize and train the KNN model
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = knn.predict(X_test)

# Print accuracy
print('Accuracy: {:.2f}%'.format(accuracy_score(y_test, y_pred) * 100))

# Detailed classification report
# target_names should be the list of unique category names corresponding to the integer labels.
print(classification_report(y_test, y_pred, target_names=df['Category'].unique()))

# Plot confusion matrix with true vs predicted labels
# The conf_matrix function already handles the mapping from integer labels to category names.
conf_matrix(y_test, y_pred)
plt.show()
```

```
(1780, 24827) (445, 24827) (1780,) (445,)
Accuracy: 66.97%
               precision    recall  f1-score   support

   Technology       0.96      0.64      0.76       102
     Business       0.93      0.51      0.66        77
       Sports       0.95      0.70      0.81        84
Entertainment       0.42      1.00      0.60       102
     Politics       1.00      0.41      0.58        80

     accuracy                           0.67       445
    macro avg       0.85      0.65      0.68       445
 weighted avg       0.84      0.67      0.68       445
```



- Dataset shape: Training (1780 × 24,827), Testing (445 × 24,827).
- Overall accuracy: Low at 66.97%, indicating poor model performance.
- Class-wise metrics:
  - High precision in most classes (0.93 to 1.00), but recall is notably low, especially for Business (0.51), Politics (0.41), and Technology (0.64).
  - Entertainment has perfect recall (1.00) but very low precision (0.42), causing imbalance.
- Macro and weighted averages: Precision remains high (0.85), but recall and F1-scores are low (0.65–0.68), showing the model misses many true positives.

Inspect Dataset Shapes

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape, X.shape, y.shape
```

```
((1780, 24827), (445, 24827), (1780,), (445,), (2225, 24827), (2225,))
```

- Full dataset: 2225 samples with 24,827 features each.
- Train/test split: 1780 training samples and 445 test samples (~80/20 split).
- Feature dimensions: Both training and test sets have 24,827 features, matching the full dataset.
- Labels: Corresponding label arrays align correctly with sample counts in train and test sets.

Display Unique News Categories

```
print(df['Category'].unique())
```

```
['Technology' 'Business' 'Sports' 'Entertainment' 'Politics']
```

Retrieve and List Available Models from API

```
print(list(api.info()['models'].keys()))
```

```
['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300', 'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glov
```

Load Pre-trained FastText Word Embeddings

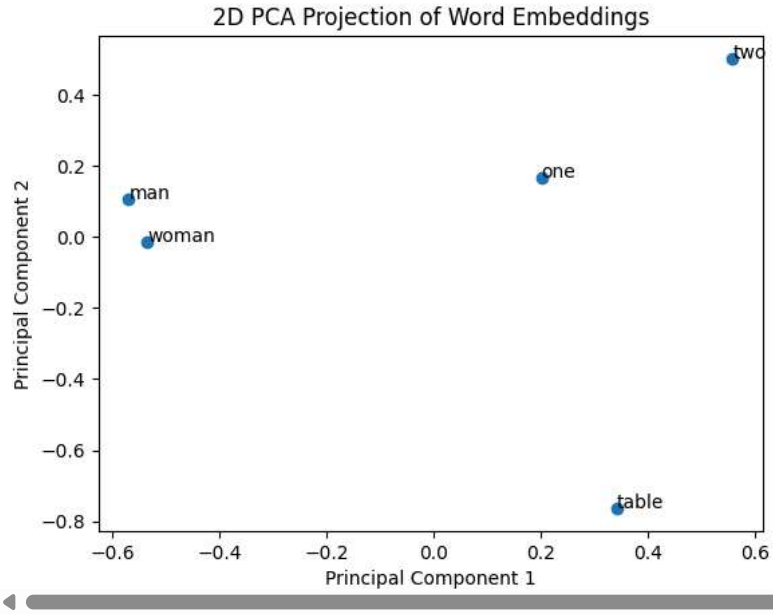
```
wv = api.load('fasttext-wiki-news-subwords-300')
```

```
[================================================] 100.0% 958.5/958.4MB downloaded
```

Visualizing Word Embeddings Using PCA

```
words = ['one', 'two', 'man', 'woman', 'table']
sample_vectors = np.array([wv[word] for word in words])

pca = PCA(n_components=2)
result = pca.fit_transform(sample_vectors)

print(result)

plt.scatter(result[:, 0], result[:, 1])

for i, word in enumerate(words):
    plt.annotate(word, xy=(result[i, 0], result[i, 1]))

plt.title('2D PCA Projection of Word Embeddings')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

```
[[ 0.20188966  0.168247  ]
 [ 0.5583138   0.50207645]
 [-0.5685122   0.10687272]
 [-0.53361326 -0.01395208]
 [ 0.34192201 -0.7632442 ]]
```



Functions for Sentence Embedding and Tokenization

```
def sent_vec(sent):
    vector_size = wv.vector_size
```

```
    wv_res = np.zeros(vector_size)
    ctr = 1
    for w in sent:
        if w in wv:
            ctr += 1
            wv_res += wv[w]
    wv_res = wv_res / ctr
    return wv_res


def spacy_tokenizer(sentence):
    doc = nlp(sentence)
    mytokens = [word.lemma_.lower().strip() for word in doc]
    mytokens = [word for word in mytokens if word not in stop_words and word not in punctuations]
    return mytokens
```

Tokenize Articles and Generate Sentence Vectors

```
nlp = spacy.load("en_core_web_sm")
stop_words = nlp.Defaults.stop_words
punctuations = string.punctuation
print(punctuations)

# Rejoin the list of tokens in the 'Article' column back into a string
df['Article_str'] = df['Article'].apply(lambda x: ' '.join(x))

# Now apply the spacy_tokenizer to the string column
df['tokens'] = df['Article_str'].apply(spacy_tokenizer)

df['vectorized_Article'] = df['tokens'].apply(sent_vec)

df.head()
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

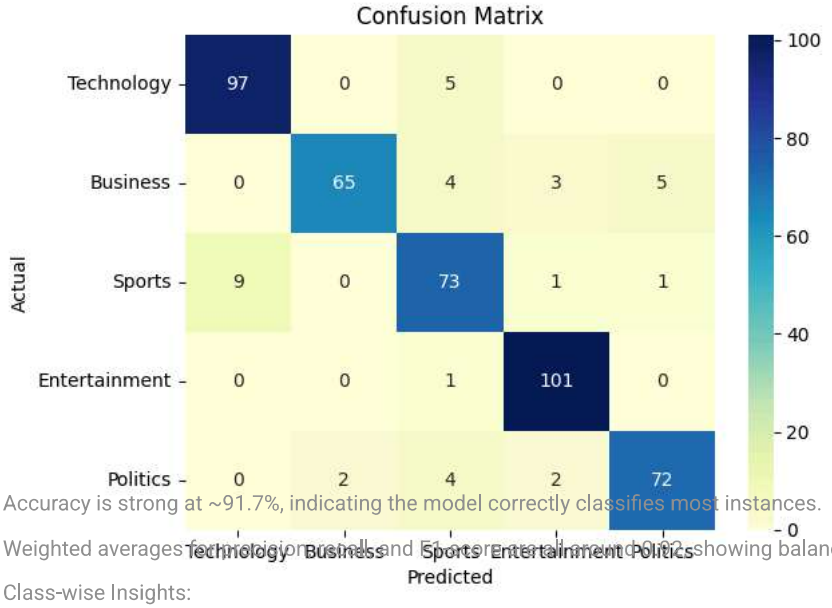
	Category	Article	Category_id	Article_str	Article_joined	Article_joined_tfidf	tokens	vectorized_Article
0	Technology	[tv, future, hand, viewer, home, theatre, syst...	4	tv future hand viewer home theatre system plas...	tv future hand viewer home theatre system plas...	tv future hand viewer home theatre system plas...	[tv, future, hand, viewer, home, theatre, syst...	[-0.0017762653022487347, -0.02814321787690376,...
1	Business	[worldcom, bos, left, book, alone, former, wor...	0	worldcom bos left book alone former worldcom b...	worldcom bos left book alone former worldcom b...	worldcom bos left book alone former worldcom b...	[worldcom, bos, leave, book, worldcom, bos, be...	[0.008055481290763342, -0.01737977168829131, 0...
2	Sports	[tiger, wary, farrell, gamble, leicester, say,...	3	tiger wary farrell gamble leicester say rushed...	tiger wary farrell gamble leicester say rushed...	tiger wary farrell gamble leicester say rushed...	[tiger, wary, farrell, gamble, leicester, rush...	[0.01120857692794087, -0.03127743935178889, 0....

Next steps: Generate code with df | View recommended plots | New interactive sheet

Train Logistic Regression Model on Sentence Vectors

```
X = df['vectorized_Article'].to_list()
y = df['Category'].to_list()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

predicted = classifier.predict(X_test)

print('accuracy %s' % accuracy_score(y_test, predicted))
print(classification_report(y_test, predicted))

conf_matrix(y_test, predicted)
```

```
accuracy 0.9168539325842696
               precision    recall  f1-score   support

     Business       0.92      0.95      0.93       102
Entertainment       0.97      0.84      0.90        77
     Politics       0.84      0.87      0.85        84
       Sports       0.94      0.99      0.97       102
   Technology       0.92      0.90      0.91        80

     accuracy                           0.92       445
    macro avg       0.92      0.91      0.91       445
 weighted avg       0.92      0.92      0.92       445
```



- Accuracy is strong at ~91.7%, indicating the model correctly classifies most instances.
- Weighted averages for precision, recall, and F1-score are all around 0.92, showing balanced performance across classes.
- Class-wise Insights:
  - Business: High precision (0.92) and recall (0.95) — very reliable predictions with few misses.
  - Entertainment: Very high precision (0.97) but lower recall (0.84) — model is very confident when it predicts this class but misses some true cases.
  - Politics: Balanced precision (0.84) and recall (0.87) — slightly lower performance but still solid.
  - Sports: Excellent recall (0.99) and precision (0.94) — almost all sports instances are correctly identified.
  - Technology: Good balance with precision (0.92) and recall (0.90) — consistent performance.