NEW CATEGORY CLASSIFICATION

Using Machine Learning

Presented To

Arooj Fatima

Presented By

Thejas Sunil - 2118074

Agenda

1. Load Dataset
2. Analyze Dataset
3. Pre-processing Data
4. Application Of NLP Techniques
5. Choosing Model
6. Training the Model
7. Evaluation
8. Testing the Model

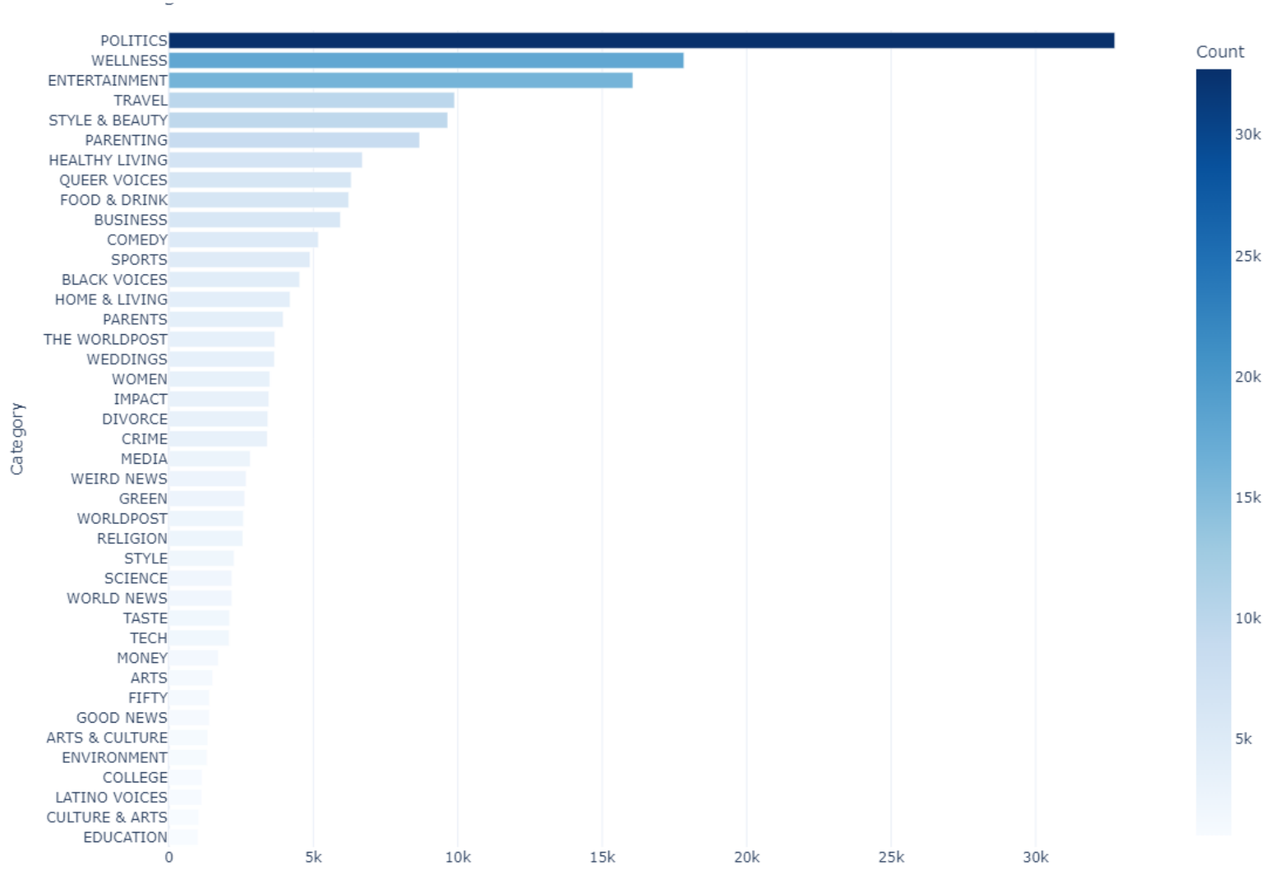
Data Analysis

Highly Imbalance

Empty values

Duplicate data

Empty spaces



Data Analysis

32,739

POLITICS

Category with the highest number of articles

1,004

EDUCATION

Category with the lowest number of articles

Pre–Processing

Highly Imbalance

Shortlisting Categories -

```
#get the category with number of news more than 3000
category_list = count_df.loc[count_df["headline"] >= 3000].index.to_list()
category_list
```

```
['POLITICS',
 'WELLNESS',
 'ENTERTAINMENT',
 'TRAVEL',
 'STYLE & BEAUTY',
 'PARENTING',
 'HEALTHY LIVING',
 'QUEER VOICES',
 'FOOD & DRINK',
 'BUSINESS',
 'COMEDY',
 'PARENTS',
 'SPORTS',
 'HOME & LIVING',
 'BLACK VOICES']
```

Pre–Processing

Highly Imbalance

Downsampling Data -

```
#creating an empty list
my_list = []
#function to concatenate headline and short description, limiting the data with only 500 data from each selected category.
for cat in category_list:
    my_list.append(df.loc[df["category"] == cat,['headline','category','short_description']].head(500))
df = pd.concat(my_list, ignore_index=True)
df["News"] = df["headline"] +' '+ df["short_description"]
df
```

```
category
POLITICS          500
WELLNESS          500
ENTERTAINMENT     500
TRAVEL            500
STYLE & BEAUTY    500
PARENTING         500
HEALTHY LIVING    500
QUEER VOICES      500
FOOD & DRINK      500
BUSINESS          500
COMEDY            500
PARENTS           500
SPORTS            500
HOME & LIVING     500
BLACK VOICES      500
```

Pre–Processing

Empty values

Dropping Null Values -

```
df.isna().sum()
```

```
Unnamed: 0             0
category               0
headline               6
authors            36620
link                   0
short_description  19712
date                   0
dtype: int64
```

```
#Function to remove empty row
def removeEmptyLine(df,col_name):
    df.replace("", pd.NA, inplace = True)
    df.dropna(subset=[col_name], inplace=True)
    return df
```

```
df = removeEmptyLine(df,"headline")
df = removeEmptyLine(df,"short_description")
df = removeEmptyLine(df,"authors")
```

```
#ensuring that empty lines were removed successfully
print((df["headline"] == "").sum())
print((df["short_description"].isna().sum()))
print((df["authors"] == "").sum())
```

```
0
0
0
```

Pre–Processing

Duplicate data

Duplicate data -

‘PARENTING’ - ‘PARENTS’
‘HOME & LIVING’ - ‘HEALTHY LIVING’

```
# droping these categories as the dataframe already have similar categories.
df = df[df['category'] != 'PARENTS']
df = df[df['category'] != 'HOME & LIVING']
```

‘POLITICS’
‘WELLNESS’
‘ENTERTAINMENT’
‘TRAVEL’
‘STYLE & BEAUTY’
‘PARENTING’
‘HEALTHY LIVING’
‘QUEER VOICES’
‘FOOD & DRINK’
‘BUSINESS’
‘COMEDY’
‘PARENTS’
‘SPORTS’
‘HOME & LIVING’
‘BLACK VOICE’

Pre–Processing

Feature Exploration

Dropping irrelevant feature and concatenation of relevant feature -

```
#creating an empty list
my_list = []
#function to concatenate headline and short description, limiting the data with only 500 data from each selected category.
for cat in category_list:
    my_list.append(df.loc[df["category"] == cat,['headline','category','short_description']].head(500))
df = pd.concat(my_list, ignore_index=True)
df["News"] = df["headline"] +' '+ df["short_description"]
df
```

	headline	category	short_description	News
0	Trump's Crackdown On Immigrant Parents Puts Mo...	POLITICS	Last month a Health and Human Services officia...	Trump's Crackdown On Immigrant Parents Puts Mo...
1	'Trump's Son Should Be Concerned': FBI Obtaine...	POLITICS	The wiretaps feature conversations between Ale...	'Trump's Son Should Be Concerned': FBI Obtaine...
2	Edward Snowden: There's No One Trump Loves Mor...	POLITICS	But don't count on Robert Mueller to nail him,...	Edward Snowden: There's No One Trump Loves Mor...
3	Booyah: Obama Photographer Hilariously Trolls ...	POLITICS	Just a peeping minute.	Booyah: Obama Photographer Hilariously Trolls ...
4	Ireland Votes To Repeal Abortion Amendment In ...	POLITICS	Irish women will no longer have to travel to t...	Ireland Votes To Repeal Abortion Amendment In ...
...	...	...	...	...
7495	Natasha Rothwell On 'Insecure,' Representation...	BLACK VOICES	The comedian discusses how she uses her passio...	Natasha Rothwell On 'Insecure,' Representation...
7496	Students Punished For Wearing Confederate Flag...	BLACK VOICES	They wore them in response to black students p...	Students Punished For Wearing Confederate Flag...
7497	There Is No Debate: The Tyranny Of The Mobiliz...	BLACK VOICES	Recent outrage expressed toward Ta-Nehisi Coat...	There Is No Debate: The Tyranny Of The Mobiliz...
7498	Air Force Academy Says Offensive Graffiti Was ...	BLACK VOICES	"You can never overemphasize the need for a cu...	Air Force Academy Says Offensive Graffiti Was ...
7499	Here's Why Music Education Is Essential For Un...	BLACK VOICES	Sway Calloway and VH1 Save The Music Foundatio...	Here's Why Music Education Is Essential For Un...

7500 rows × 4 columns

Pre–Processing

Highly Imbalance

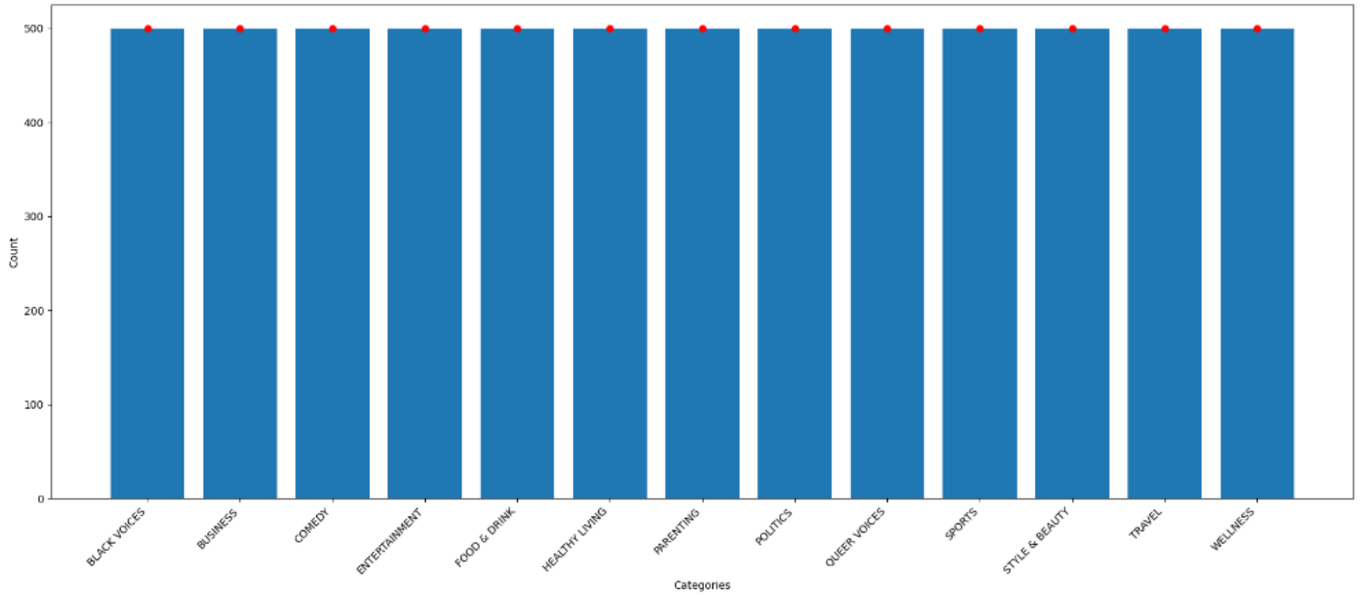
Empty values

Duplicate data

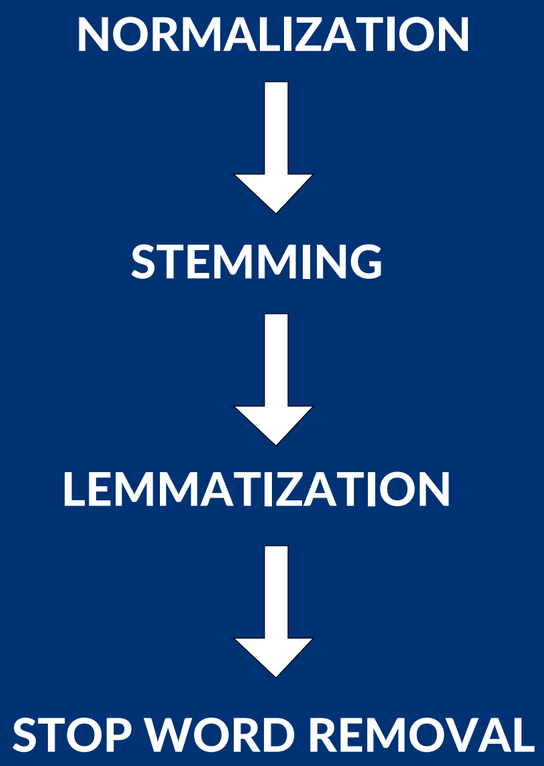
Feature Exploration

Result-

```
count_df = pd.DataFrame(df.groupby(["category"]).count()["News"].sort_values(ascending=False))

plt.figure(figsize=(18,8))
plt.xlabel("Categories")
plt.ylabel("Count")
plt.bar(count_df.index,count_df["News"])
plt.plot(count_df.index,count_df["News"],'ro')
plt.xticks(rotation=45, ha='right')  # Rotating x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent overlapping labels
```



Natural Language Processing



Natural Language Processing

Normalization

Stemming

Lemmatization

Stop word removal

Normalization-

```
# function to remove punctuations, tags and white spaces.
def pre_process_text(text):
 processed_text = text.lower()
 processed_text = re.sub(f"[{re.escape(punctuation)}]", "", processed_text)
 processed_text = " ".join(processed_text.split())
 return processed_text

df["News"] = df["News"].apply(lambda x:pre_process_text(x))
df
```

Text to lower case

Removes punctuation

	category	News
0	POLITICS	trumps crackdown on immigrant parents puts mor...
1	POLITICS	trumps son should be concerned fbi obtained wi...
2	POLITICS	edward snowden theres no one trump loves more ...
3	POLITICS	booyah obama photographer hilariously trolls t...
4	POLITICS	ireland votes to repeal abortion amendment in ...
...	...	...
7495	BLACK VOICES	natasha rothwell on insecure representation an...
7496	BLACK VOICES	students punished for wearing confederate flag...
7497	BLACK VOICES	there is no debate the tyranny of the mobilize...
7498	BLACK VOICES	air force academy says offensive graffiti was ...
7499	BLACK VOICES	heres why music education is essential for und...

5500 rows × 2 columns

Natural Language Processing

Normalization

Stemming

Lemmatization

Stop word removal

Stemming -

```
nltk.download('punkt')
```

```
# function for stemming

def stemming(text):
    tokens = word_tokenize(text)
    ps = PorterStemmer()
    required_words = [ps.stem(x) for x in tokens]
    sentence_with_stemmed_words = ' '.join(required_words)
    return sentence_with_stemmed_words

df["News"] = df["News"].apply(lambda x: stemming(x))
df
```

reducing the word to their base

eg. driving to driv

	category	News
0	POLITICS	trump crackdown on immigr parent put more kid ...
1	POLITICS	trump son should be concern fbi obtain wiretap...
2	POLITICS	edward snowden there no one trump love more th...
3	POLITICS	booyah obama photograph hilari troll trump spi...
4	POLITICS	ireland vote to repeal abort amend in landslid...
...	...	...
7495	BLACK VOICES	natasha rothwel on insecur represent and her s...
7496	BLACK VOICES	student punish for wear confeder flag tell bla...
7497	BLACK VOICES	there is no debat the tyranni of the mobil uni...
7498	BLACK VOICES	air forc academi say offens graffiti wa fake b...
7499	BLACK VOICES	here whi music educ is essenti for underserv s...

6500 rows × 2 columns

Natural Language Processing

Normalization

Stemming

Lemmatization

Stop word removal

Lemmatization -

```
nltk.download('wordnet')
```

```
#function for lemmatization
def lemmatization(text):
    wordnet_lemmatizer = WordNetLemmatizer()
    tokens = text.split()
    required_words = [wordnet_lemmatizer.lemmatize(x, 'v') for x in tokens]
    sentence_with_lemmatized_word = " ".join(required_words)
    return sentence_with_lemmatized_word

df['News'] = df['News'].apply(lambda x: lemmatization(x))
df
```

reducing the words to their root and removing all the irrelevant words

	category	News
0	POLITICS	trump crackdown on immigr parent put more kid ...
1	POLITICS	trump son should be concern fbi obtain wiretap...
2	POLITICS	edward snowden there no one trump love more th...
3	POLITICS	booyah obama photograph hilari troll trump spi...
4	POLITICS	ireland vote to repeal abort amend in landslid...
...	...	...
7495	BLACK VOICES	natasha rothwel on insecur represent and her s...
7496	BLACK VOICES	student punish for wear confeder flag tell bla...
7497	BLACK VOICES	there be no debat the tyranni of the mobil uni...
7498	BLACK VOICES	air forc academi say offens graffiti wa fake b...
7499	BLACK VOICES	here whi music educ be essenti for underserv s...

5500 rows × 2 columns

Natural Language Processing

Normalization

Stemming

Lemmatization

Stop word removal

Stop Word-

```
nltk.download('stopwords')
```

```
def remove_stopwords(text):
    tokens = word_tokenize(text)
    required_words = [x for x in tokens if x not in stopwords.words('english')]
    sentence_with_no_stopwords = ' '.join(required_words)
    return sentence_with_no_stopwords

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

Removing common words like the, an, is, this etc.

	category	News
0	POLITICS	trump crackdown immigr parent put kid alreadi ...
1	POLITICS	trump son concern fbi obtain wiretap putin all...
2	POLITICS	edward snowden one trump love vladimir putin d...
3	POLITICS	booyah obama photograph hilari troll trump spi...
4	POLITICS	ireland vote repeal abort amend landslid refer...
...	...	...
7495	BLACK VOICES	natasha rothwel insecur represent space comedi...
7496	BLACK VOICES	student punish wear confeder flag tell black k...
7497	BLACK VOICES	debat tyranni mobil uninform recent outrag exp...
7498	BLACK VOICES	air forc academi say offens graffiti wa fake r...
7499	BLACK VOICES	whi music educ essenti underserv school sway c...

6500 rows × 2 columns

Natural Language Processing

Normalization

Stemming

Lemmatization

Stop word removal

Stop Word-

```
nltk.download('stopwords')
```

```
def remove_stopwords(text):
    tokens = word_tokenize(text)
    required_words = [x for x in tokens if x not in stopwords.words('english')]
    sentence_with_no_stopwords = ' '.join(required_words)
    return sentence_with_no_stopwords

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

Removing common words like the, an, is, this etc.

	category	News
0	POLITICS	trump crackdown immigr parent put kid alreadi ...
1	POLITICS	trump son concern fbi obtain wiretap putin all...
2	POLITICS	edward snowden one trump love vladimir putin d...
3	POLITICS	booyah obama photograph hilari troll trump spi...
4	POLITICS	ireland vote repeal abort amend landslid refer...
...	...	...
7495	BLACK VOICES	natasha rothwel insecur represent space comedi...
7496	BLACK VOICES	student punish wear confeder flag tell black k...
7497	BLACK VOICES	debat tyranni mobil uninform recent outrag exp...
7498	BLACK VOICES	air forc academi say offens graffiti wa fake r...
7499	BLACK VOICES	whi music educ essenti underserv school sway c...

6500 rows × 2 columns

Using Vectorization Techniques

Vectorization techniques are essential tools in the field of data science and machine learning. By converting text data into numerical representations, vectorization allows algorithms to process and analyze the information effectively. There are various methods for vectorizing text, such as Bag of Words, TF-IDF, and Word Embeddings like Word2Vec and GloVe. Each technique has its strengths and weaknesses, making it crucial to choose the most suitable approach based on the specific requirements of the project. Experimenting with different vectorization methods and fine-tuning the parameters can significantly impact the performance and accuracy of the models. Continuous practice and exploration of vectorization techniques are key to mastering the art of text analysis and achieving optimal results in data-driven tasks.

- **HASHING**
- **TF-IDF**
- **BAG OF WORDS**

Vectorization

Hashing Vectorization

Hashing Vectorization is a technique commonly used in natural language processing and machine learning. It involves converting text data into numerical vectors using hashing functions. This method helps in reducing the dimensionality of the data, making it easier to process and analyze large amounts of text. Hashing Vectorization assigns a unique numerical value to each word in the text, creating a compact representation of the original data. This approach is efficient and effective for tasks such as text classification, sentiment analysis, and document clustering. By leveraging Hashing Vectorization, researchers and data scientists can efficiently work with text data to derive valuable insights and patterns.

```
#Library
from sklearn.feature_extraction.text import HashingVectorizer

hv= HashingVectorizer()#vectorization
label= LabelEncoder()#encoder

x= hv.fit_transform(dataset["News"])#vectorizing text
y= label.fit_transform(dataset["category"])#encoding labels

#test_train_split 80% training and 20% testing
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.20,random_state=0)
x_train=np.abs(x_train)
x_test=np.abs(x_test)
```

Vectorization

TF-IDF

Term Frequency-Inversion Document Frequency (TF-IDF) is a popular technique used in information retrieval and text mining to determine the importance of a term within a document or a collection of documents. The TF-IDF value increases proportionally to the number of times a word appears in a document but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words are more common than others. This method is valuable in identifying the relevance of a term in a specific context and is widely used in search engines, text analysis, and machine learning algorithms. By incorporating both term frequency and inverse document frequency, TF-IDF provides a more nuanced understanding of the significance of words, allowing for more accurate analysis and interpretation of textual data.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf= TfidfVectorizer()
label= LabelEncoder()

x= tfidf.fit_transform(dataset["News"])#vectorizing text
y= label.fit_transform(dataset["category"])#encoding label

#test_train_split 80% training and 20% testing
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.20,random_state=0)
```

Vectorization

Bag Of Words

Bag of words is a Natural Language Processing technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.
A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

```
from sklearn.feature_extraction.text import CountVectorizer

cv= CountVectorizer()
label= LabelEncoder()

x= cv.fit_transform(dataset["News"])
y= label.fit_transform(dataset["category"])

#test_train_split 80:20
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.20,random_state=0)
```

Cross Validation

k – Fold Method
Using Random Forest Classifer

```python
from sklearn.model_selection import cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier

# RandomForestClassifier
rf_classifier = RandomForestClassifier()

#  k-Fold Cross-Validation object (with k=5)
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Cross-Validation
cv_scores = cross_val_score(rf_classifier, x, y, cv=kf, scoring='accuracy')

# Print
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", np.mean(cv_scores))
print("Standard Deviation of CV Accuracy:", np.std(cv_scores))
```

- Hash Vectorizer

```
Cross-Validation Scores: [0.67846154 0.69230769 0.70615385 0.68307692 0.66230769]
Mean CV Accuracy: 0.6844615384615385
Standard Deviation of CV Accuracy: 0.014565908158208775
```

- TF–IDF

```
Cross-Validation Scores: [0.65230769 0.63076923 0.67       0.64692308 0.65076923]
Mean CV Accuracy: 0.6501538461538462
Standard Deviation of CV Accuracy: 0.012530672427259856
```

- Bag Of Words

```
Cross-Validation Scores: [0.64769231 0.63538462 0.66230769 0.64       0.64076923]
Mean CV Accuracy: 0.6452307692307693
Standard Deviation of CV Accuracy: 0.009400994533816704
```

Classification Models

Classification is a supervised machine learning method where the model tries to predict the correct label of a given input data. In classification, the model is fully trained using the training data, and then it is evaluated on test data before being used to perform prediction on new unseen data.

- RANDOM FOREST
- MULTINOMIAL NB
- LOGISTIC REGRESSION
- SVM
- KNN

Classification Model

```
from sklearn.ensemble import RandomForestClassifier

rfc= RandomForestClassifier()

#train
rfc.fit(x_train,y_train)

#prediction
pred= rfc.predict(x_test)

#evaluate
print("Classification Report :",classification_report(y_test,pred,zero_division=1,target_names=dataset['category'].unique()))
print ("Accuracy : ", accuracy_score(y_test, pred)*100)
print("Precision score :",precision_score(y_test,pred,average="weighted")*100)
print("Recall score :",recall_score(y_test,pred,average="weighted")*100)

cmx = confusion_matrix(y_test, pred)

plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Label")
plt.show()
```

RANDOM FOREST

Random forest is a commonly-used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

Classification Model

```
from sklearn.naive_bayes import MultinomialNB

mnb= MultinomialNB()
mnb.fit(x_train,y_train)

pred= mnb.predict(x_test)

print("Classification Report :",classification_report(y_test,pred,zero_division=1,target_names=dataset['category'].unique()))
print ("Accuracy : ", accuracy_score(y_test, pred)*100)
print("Precision score :",precision_score(y_test,pred,average="weighted")*100)
print("Recall score :",recall_score(y_test,pred,average="weighted")*100)

cmx = confusion_matrix(y_test, pred)

plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
```

MULTINOMIAL NB

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of features values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

Classification Model

```python
from sklearn.linear_model import LogisticRegression

# Logistic Regression Model
lr = LogisticRegression()
lr.fit(x_train, y_train)  # Assuming x_train is a sparse matrix

# Predictions
pred = lr.predict(x_test)

# Classification Report
print("Classification Report:")
print(classification_report(y_test, pred, zero_division=1, target_names=dataset['category'].unique()))

# Accuracy, Precision, Recall
accuracy = accuracy_score(y_test, pred) * 100
precision = precision_score(y_test, pred, average="weighted") * 100
recall = recall_score(y_test, pred, average="weighted") * 100
print("Accuracy:", accuracy)
print("Precision score:", precision)
print("Recall score:", recall)

# Confusion Matrix
cmx = confusion_matrix(y_test, pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

LOGISTIC REGRESSION

A multinomial logistic regression (or multinomial regression for short) is used when the outcome variable being predicted is nominal and has more than two categories that do not have a given rank or order. This model can be used with any number of independent variables that are categorical or continuous.

Classification Model

```python
from sklearn.svm import SVC

# Create SVM model
svm = SVC(kernel='linear')  # You can choose different kernels such as 'linear', 'poly', 'rbf', etc.
# Train the model
svm.fit(x_train, y_train)  # Assuming x_train is your feature matrix and y_train is your target vector
# predictions on the test
pred = svm.predict(x_test)
# Evaluate the model
accuracy = accuracy_score(y_test, pred) * 100
precision = precision_score(y_test, pred, average="weighted") * 100
recall = recall_score(y_test, pred, average="weighted") * 100
print("Accuracy:", accuracy)
print("Precision score:", precision)
print("Recall score:", recall)
# Print classification report
print("Classification Report:")
print(classification_report(y_test, pred, zero_division=1, target_names=dataset['category'].unique()))

# Generate confusion matrix
cmx = confusion_matrix(y_test, pred)
# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space. SVMs were developed in the 1990s by Vladimir N.

Classification Model

```
from sklearn.neighbors import KNeighborsClassifier

# KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can choose the number of neighbors (k) as per your choice

# Training the model
knn.fit(x_train, y_train)
# predictions on the test
pred = knn.predict(x_test)
# Evaluate the model
accuracy = accuracy_score(y_test, pred) * 100
precision = precision_score(y_test, pred, average="weighted") * 100
recall = recall_score(y_test, pred, average="weighted") * 100
print("Accuracy:", accuracy)
print("Precision score:", precision)
print("Recall score:", recall)
# Print classification report
print("Classification Report:")
print(classification_report(y_test, pred, zero_division=1, target_names=dataset['category'].unique()))
# Generating confusion matrix
cm = confusion_matrix(y_test, pred)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

KNN

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It is one of the popular and simplest classification and regression classifiers used in machine learning today.

Evaluation

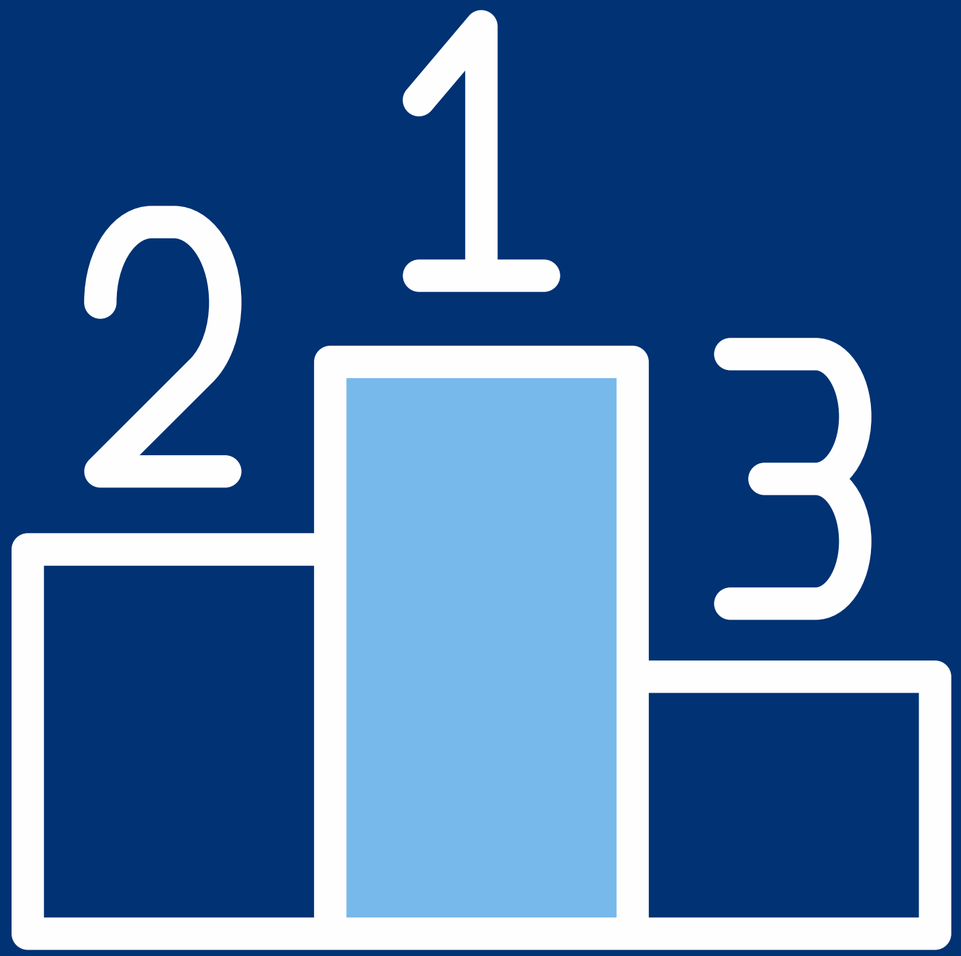
MODEL ACCURACY %

	HASHING VECTORIZATION	TF-IDF	BAG OF WORDS
RANDOM FOREST CLASSIFICATION	67.54%	64.54%	63.38%
MULTINOMIAL NAÏVE BAYES	63.23%	67.15%	66.38%
LOGISTIC REGRESSION	64.38%	69.38%	66.62%
SUPPORT VECTOR MACHINES	64.31%	68.46%	64.30%
K-NEAREST NEIGHBOR	52.08%	56.77%	21.61%

Evaluation

Best Vectorizer -

1. TF –IDF
2. Hashing
3. Bag of Words



Prediction

```python
# function to Preprocess the input news headline
def preprocess_input(input_headline):
    input_features = cv.transform([input_headline])
    return input_features

# function for prediction
def predict_category(input_features):
    prediction = rfc.predict(input_features)
    predicted_category = label.inverse_transform(prediction)[0]
    return predicted_category

# get input for performing perdiction of the category
input_headline = input("Enter the news headline: ")

# preprocess the input
input_features = preprocess_input(input_headline)

# make a prediction
predicted_category = predict_category(input_features)

# displays the prediction
print("Predicted category:", predicted_category)
```

```
Enter the news headline:  Voting for the trump was a bad decision says voters
Predicted category: POLITICS
```

```
Enter the news headline:  new burger king restaurant opening soon
Predicted category: FOOD & DRINK
```

```
Enter the news headline:  Children are now more relying on their parents
Predicted category: PARENTING
```

```
Enter the news headline:  How far would the activist go to protect their rights
Predicted category: HEALTHY LIVING
```

```
Enter the news headline:  Apple announces the best update ever
Predicted category: FOOD & DRINK
```

Further Experiment

- Increase data
- Reduce categories
- GridSearchCV Tuning

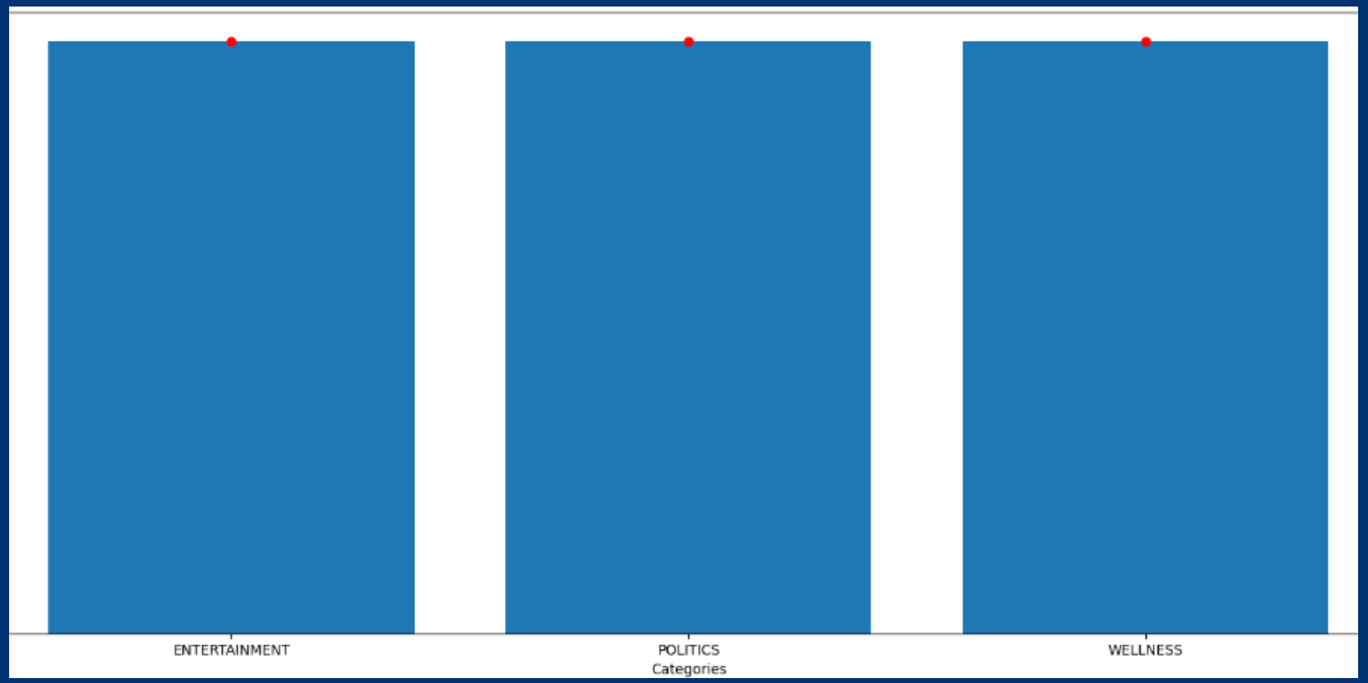
News Categories

- POLITICS
- WELLNESS
- ENTERTAINMENT

‘2000’

```
#creating an empty list
my_list = []
#function to concatenate headline and short description, limiting the data with only 500 data from each selected category.
for cat in category_list:
    my_list.append(df.loc[df["category"] == cat,['headline','category','short_description']].head(2000))
df = pd.concat(my_list, ignore_index=True)
df["News"] = df["headline"] +' '+ df["short_description"]
df
```

```
count_df = pd.DataFrame(df.groupby(["category"]).count()["News"].sort_values(ascending=False))

plt.figure(figsize=(18,8))
plt.xlabel("Categories")
plt.ylabel("Count")
plt.bar(count_df.index,count_df["News"])
plt.plot(count_df.index,count_df["News"],'ro')
```



Grid Search CV

Using TF-IDF
Logistic Regression

- TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf= TfidfVectorizer(max_features=5000, ngram_range=(1, 2)) #max feature only c
#ngram 1&2 make sure the vectrizer considers both unigram and bi gram individual
label= LabelEncoder()

x= tfidf.fit_transform(dataset["News"])#vectorizing text
y= label.fit_transform(dataset["category"])#encoding label

#test_train_split 80% training and 20% testing
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.25,random_state=0
```

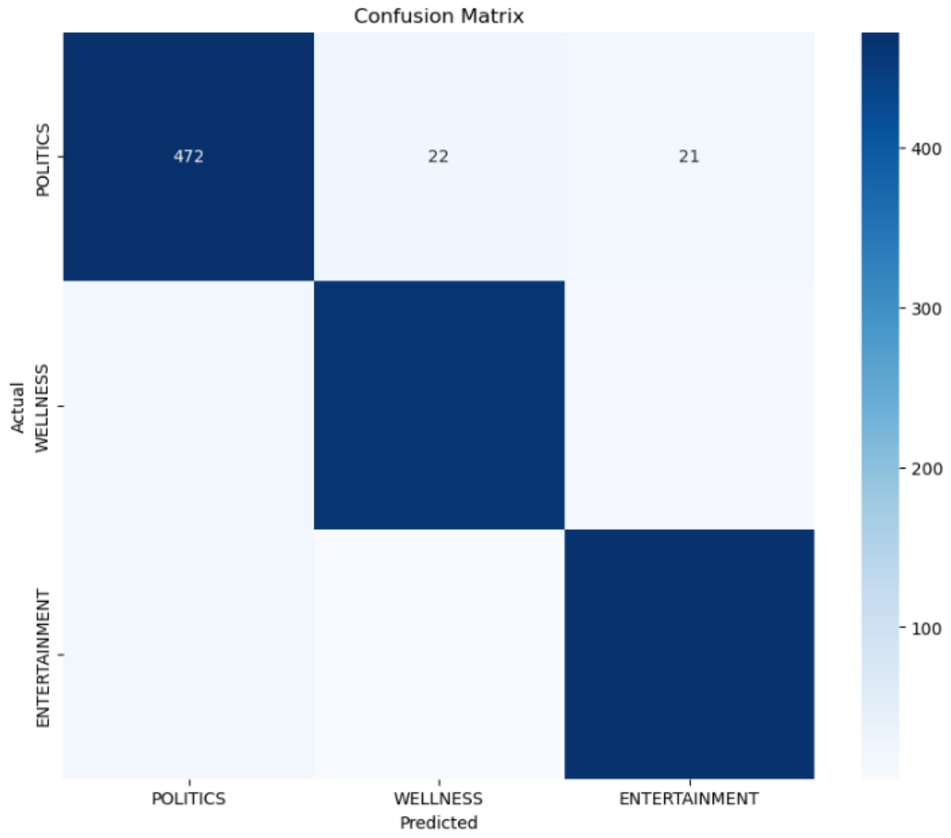
- Logistic Regression

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the parameter grid
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear']
}

# Instantiate Logistic Regression classifier
lr = LogisticRegression()

# Instantiate GridSearchCV
grid_search = GridSearchCV(lr, param_grid, cv=5, scoring='accuracy')

# Perform Grid Search
grid_search.fit(x_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
Best Parameters: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Best Score: 0.93
```

Classification Model

LOGISTIC REGRESSION

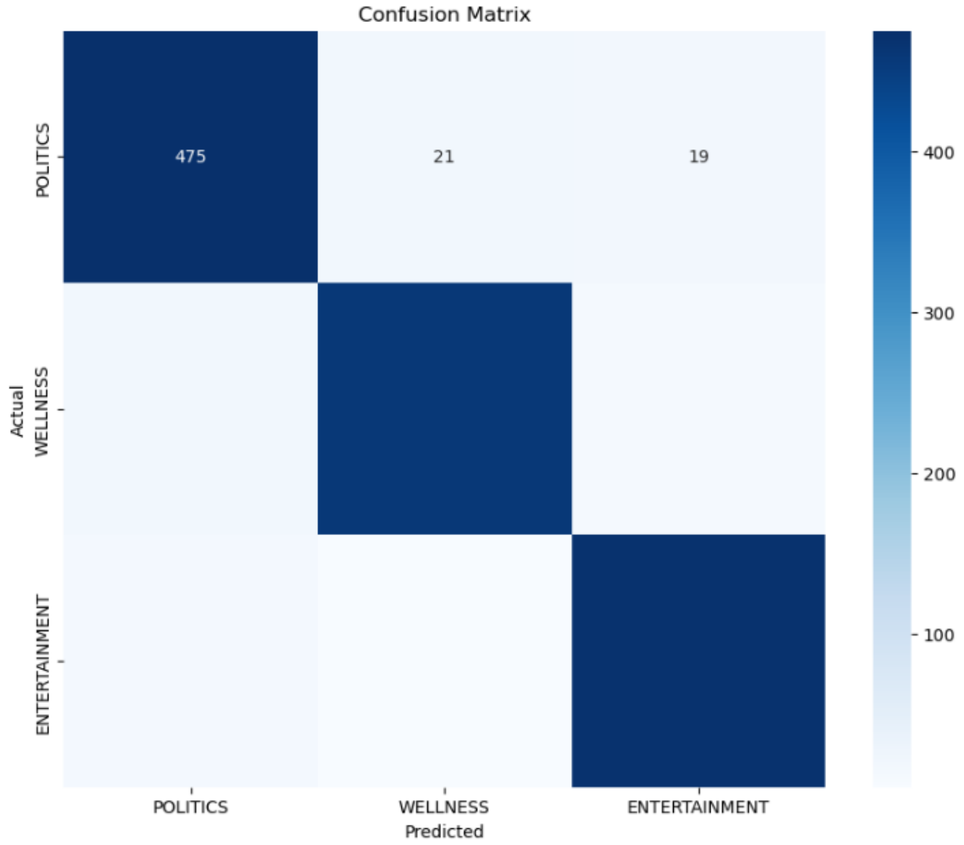



```python
from sklearn.linear_model import LogisticRegression

# Logistic Regression Model
lr = LogisticRegression(multi_class='multinomial', solver='lbfgs') # solver newton-cg for big datasets #optimization technique
lr.fit(x_train, y_train)  # Assuming x_train is a sparse matrix

# Predictions
pred = lr.predict(x_test)

# Classification Report
print("Classification Report:")
print(classification_report(y_test, pred, zero_division=1, target_names=dataset['category'].unique()))

# Accuracy, Precision, Recall
accuracy = accuracy_score(y_test, pred) * 100
precision = precision_score(y_test, pred, average="weighted") * 100
recall = recall_score(y_test, pred, average="weighted") * 100
print("Accuracy:", accuracy)
print("Precision score:", precision)
print("Recall score:", recall)

# Confusion Matrix
cmx = confusion_matrix(y_test, pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

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

     POLITICS       0.92      0.92      0.92       515
     WELLNESS       0.94      0.94      0.94       494
ENTERTAINMENT       0.94      0.95      0.94       491

     accuracy                           0.93      1500
    macro avg       0.93      0.93      0.93      1500
 weighted avg       0.93      0.93      0.93      1500

Accuracy: 93.4
Precision score: 93.40017182883147
Recall score: 93.4
```

Classification Model

SUPPORT VECTOR MACHINE

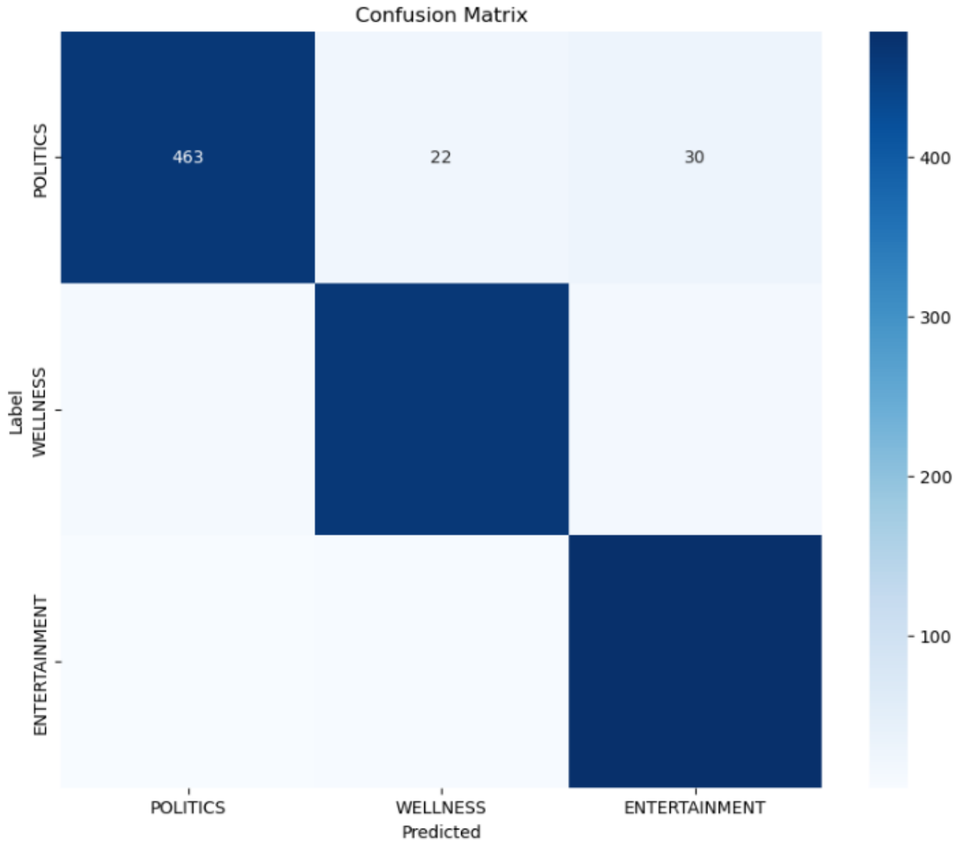



```python
from sklearn.svm import SVC

# Create SVM model
svm = SVC(kernel='rbf')  #can choose different kernels such as 'linear', 'poly', 'rbf', etc.

# Train the model
svm.fit(x_train, y_train)

# predictions on the test
pred = svm.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, pred) * 100
precision = precision_score(y_test, pred, average="weighted") * 100
recall = recall_score(y_test, pred, average="weighted") * 100
print("Accuracy:", accuracy)
print("Precision score:", precision)
print("Recall score:", recall)
# Print classification report
print("Classification Report:")
print(classification_report(y_test, pred, zero_division=1, target_names=dataset['category'].unique()))

# Generate confusion matrix
cmx = confusion_matrix(y_test, pred)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Accuracy: 93.73333333333333
Precision score: 93.73690903684115
Recall score: 93.73333333333333
Classification Report:
              precision    recall  f1-score   support

     POLITICS       0.93      0.92      0.93       515
     WELLNESS       0.95      0.93      0.94       494
ENTERTAINMENT       0.94      0.96      0.95       491

     accuracy                           0.94      1500
    macro avg       0.94      0.94      0.94      1500
 weighted avg       0.94      0.94      0.94      1500
```

Classification Model

MULTINOMIAL NB




```python
from sklearn.naive_bayes import MultinomialNB

mnb= MultinomialNB()
mnb.fit(x_train,y_train)

pred= mnb.predict(x_test)

print("Classification Report :",classification_report(y_test,pred,zero_division=1,target_names=dataset['category'].unique()))
print ("Accuracy : ", accuracy_score(y_test, pred)*100)
print("Precision score :",precision_score(y_test,pred,average="weighted")*100)
print("Recall score :",recall_score(y_test,pred,average="weighted")*100)

cmx = confusion_matrix(y_test, pred)

plt.figure(figsize=(10, 8))
sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", xticklabels=dataset['category'].unique(), yticklabels=dataset['category'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Label")
plt.show()
```

```
Classification Report :                    precision    recall

      POLITICS       0.96      0.90      0.93       515
      WELLNESS       0.94      0.94      0.94       494
 ENTERTAINMENT       0.91      0.98      0.94       491

      accuracy                           0.94      1500
     macro avg       0.94      0.94      0.94      1500
  weighted avg       0.94      0.94      0.94      1500

Accuracy :  93.73333333333333
Precision score : 93.84129294760781
Recall score : 93.73333333333333
```

Evaluation & Analysis

- Multinomial Naive Bayes & SVM 97.3%
- Exposing the model with more data gives better Accuracy
- Lemmatization is better than stemming
- Grid Search CV Helps find the optimal parameters

Thank you!