Problem Statement

Business Context

In the dynamic landscape of the media and news industry, the ability to swiftly categorize and curate content has become a strategic imperative. The vast volume of information demands efficient systems to organize and present content to the audience.

The media industry, being the pulse of information dissemination, grapples with the continuous influx of news articles spanning diverse topics. Ensuring that the right articles reach the right audience promptly is not just a logistical necessity but a critical component in retaining and engaging audiences in an age of information overload.

Common Industry Challenges: Amidst the ceaseless flow of news, organizations encounter challenges such as:

- Information Overload: The sheer volume of news articles makes manual categorization impractical.
- Timeliness: Delays in categorizing news articles can result in outdated or misplaced content.

Problem Definition

E-news Express, a news aggregation startup, faces the challenge of categorizing the news articles collected. With news articles covering sports, busie=ness, politics, and more, the need for an advanced and automated system to categorize them has become increasingly evident. The manual efforts required for categorizing such a diverse range of news articles are substantial, and human errors in the categorization of news articles can lead to reputational damage for the startup. There is also the factor of delays and potential inaccuracies. To streamline and optimize this process, the organization recognizes the imperative of adopting cutting-edge technologies, particularly machine learning, to automate and enhance the categorization of content.

As a data scientist on the E-news Express data team, the task is to analyze the text in news articles and build a model for categorizing them. The goal is to optimize the categorization process, ensuring timely and personalized delivery.

Data Dictionary

- **Article**: The main body of the news article
- **Category**: The category the article belongs to

Please read the instructions carefully before starting the project.

This is a commented Python Notebook file in which all the instructions and tasks to be performed are mentioned.

- Blanks '___' are provided in the notebook that needs to be filled with an appropriate code to get the correct result. With every '___' blank, there is a comment that briefly describes what needs to be filled in the blank space.
- Identify the task to be performed correctly, and only then proceed to write the required code.
- Please run the codes in a sequential manner from the beginning to avoid any unnecessary errors.
- Add the results/observations (wherever mentioned) derived from the analysis in the presentation and submit the same. Any mathematical or computational details which are a graded part of the project can be included in the Appendix section of the presentation.

Note:

1. Please make sure to use Google Colab for this project.
2. Please set the Colab runtime to **T4 GPU** before starting the project.

Installing and Importing Necessary Libraries and Dependencies

In [ ]:

```
# installing the libraries for transformers
!pip install -U -q sentence-transformers transformers bitsandbytes accelerate sentencepiece
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 156.5/156.5 kB 4.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 105.0/105.0 MB 9.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 280.0/280.0 kB 33.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 73.3 MB/s eta 0:00:00
```

In [ ]:

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.25
.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11
.4)
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option('max_colwidth', None)

import torch
from sentence_transformers import SentenceTransformer
from transformers import T5Tokenizer, T5ForConditionalGeneration, pipeline
# To build a Random Forest model
from sklearn.ensemble import RandomForestClassifier

import pickle
# to split the data
from sklearn.model_selection import train_test_split
# to compute performance metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,make_scorer,recall_score,prec
ision_score,f1_score
from sklearn.model_selection import GridSearchCV


# to ignore unnecessary warnings
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
import warnings
warnings.filterwarnings("ignore")
import scipy.stats as stats

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_scor
e, f1_score, roc_auc_score
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import BaggingClassifier,RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassi
fier, StackingClassifier
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
```

Loading the Dataset

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
# Complete the code to read the CSV file
data = pd.read_csv("/content/drive/MyDrive/article_data.csv")
```

Data Overview

In [ ]:

```
# Write the code to check the first 5 rows of the data
data.head(5)
```

Out[ ]:

	Article	Category
0	Sudan Govt rejects call to separate religion, state Sudanese rebel leaders #39; demand that Islam be kept out of government in the war-torn region of Darfur, has been rejected by government negotiators.	0
1	Hassan: #39;Abhorrent act #39; says Blair Western political leaders have united to condemn the kidnappers of charity worker Margaret Hassan after a video surfaced apparently showing a militant firing a pistol into the head of a blindfolded woman wearing an orange jumpsuit.	0
2	Sharon Says Gaza Evacuation Set for 2005 (AP) AP - Israel's evacuation of the Gaza Strip will begin next summer and will take about 12 weeks, Prime Minister Ariel Sharon said Wednesday, reversing an earlier decision to speed up the pullout.	0
3	Prince Charles chastised for quot;old fashioned quot; views A minister has launched a scathing attack on heir to the throne Prince Charles, accusing him of being quot;very old fashioned quot; and out of touch in his views on teaching in schools.	0
4	U.S. Says N.Korea Blast Probably Not Nuclear SEOUL (Reuters) - A huge explosion rocked North Korea last week but U.S. and South Korean officials said on Sunday it was unlikely to have been a nuclear weapons test despite the appearance of a "peculiar cloud" over the area.	0

In [ ]:

```
# Write the code to check the shape of the data
data.shape
```

Out[ ]:

```
(4000, 2)
```

In [ ]:

```
## Complete the code to check the value counts in Category column
data["Category"].value_counts()
```

Out[ ]:

```
0    1000
1    1000
2    1000
3    1000
Name: Category, dtype: int64
```

In [ ]:

```
#Check data type and not null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Article   4000 non-null   object
 1   Category  4000 non-null   int64
dtypes: int64(1), object(1)
memory usage: 62.6+ KB
```

In [ ]:

```
data.isna().sum()
```

Out[ ]:

```
Article     0
Category    0
dtype: int64
```

In [ ]:

```
cols = data.select_dtypes(['object'])
cols.columns
```

Out[ ]:

```
Index(['Article'], dtype='object')
```

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Article   4000 non-null   object
 1   Category  4000 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 62.6+ KB
```

In [ ]:

```
data.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Category	4000.0	1.5	1.118174	0.0	0.75	1.5	2.25	3.0

Exploratory Data Analysis (EDA)

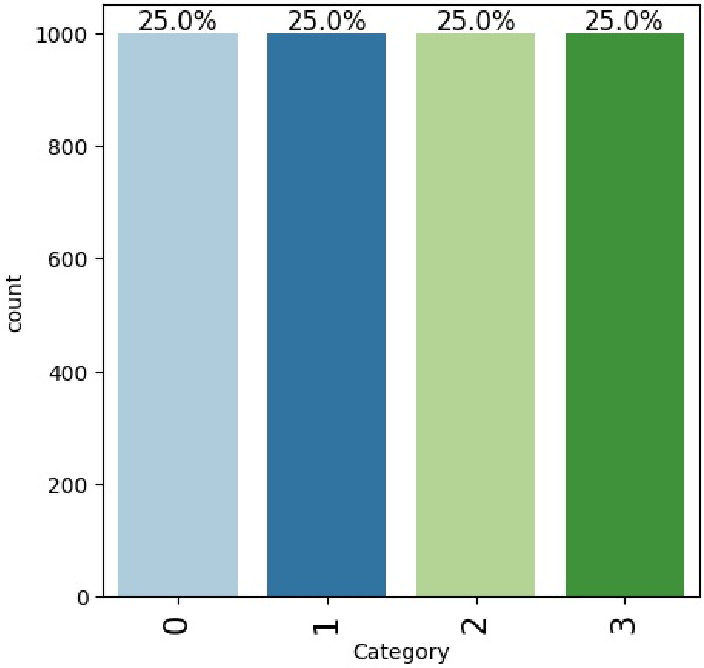
In [ ]:

```
# function to create labeled barplots

def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

Distribution of category

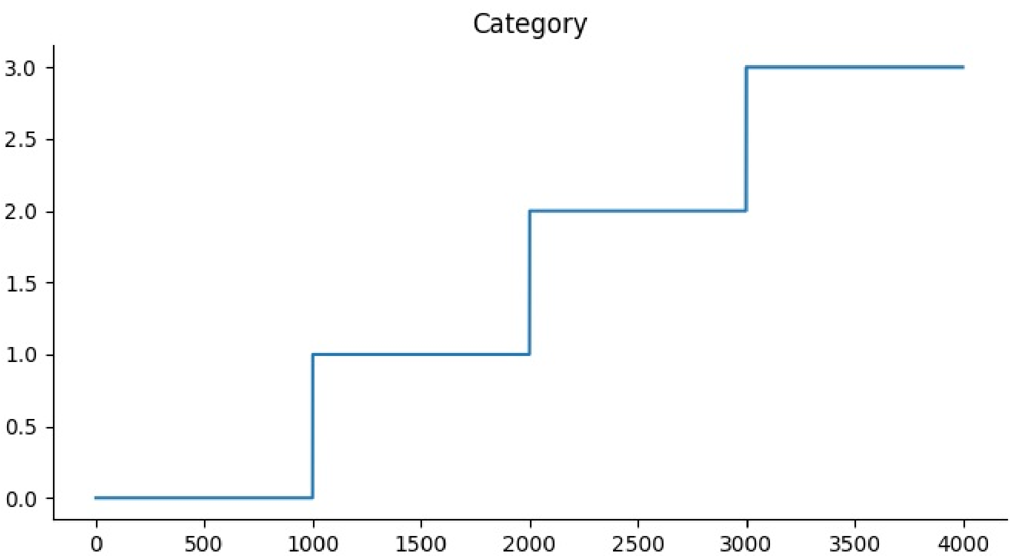
In [ ]:

```
labeled_barplot(data, "Category", perc=True)   ## Complete the code to get the barplot of Category variable
```



In [ ]:

```
data['Category'].plot(kind='line', figsize=(8, 4), title='Category')
plt.gca().spines[['top', 'right']].set_visible(False)
```



Model Building - Sentence Transformer + ML

Defining the SentenceTransformer Model

In [ ]:

```
## Defining the model.
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
```

In [ ]:

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(cm.shape[0], cm.shape[1])

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Encoding the data

In [ ]:

```
# setting the compute device
device = "cuda" if torch.cuda.is_available() else "cpu"

## Encoding the dataset.
embedding_matrix = model.encode(data["Article"],show_progress_bar=True,device=device)
```

Train-Test Split

In [ ]:

```
data["Category"] = data["Category"].apply(lambda x: 0 if x == "Denied" else 1)
```

In [ ]:

```
# Split the data
X = embedding_matrix
y = data["Category"]
```

In [ ]:

```
# Initial split into training (80%) and testing (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.20, random_state=42)

# Further split the temporary set into validation (10%) and test (10%) sets
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.50, random_state=42)
```

In [ ]:

```
print("Shape of the set of input variables for training:", X_train.shape)    # Complete the code to get the shape
of training input data
print("Shape of the set of input variables for validation:", X_valid.shape)    # Complete the code to get the sha
pe of validation input data
print("Shape of the set of input variables for testing:", X_test.shape)     # Complete the code to get the shape
of testing input data
```

```
Shape of the set of input variables for training: (3200, 384)
Shape of the set of input variables for validation: (400, 384)
Shape of the set of input variables for testing: (400, 384)
```

In [ ]:

```
print("Shape of the set of output variables for training:", y_train.shape)    # Complete the code to get the shap
e of training output data
print("Shape of the set of output variables for validation:", y_valid.shape)    # Complete the code to get the sh
ape of validation output data
print("Shape of the set of output variables for testing:", y_test.shape)     # Complete the code to get the shape
of testing output data
```

```
Shape of the set of output variables for training: (3200,)
Shape of the set of output variables for validation: (400,)
Shape of the set of output variables for testing: (400,)
```

In [ ]:

```
X_train.shape, X_test.shape, y_train.shape
```

Out[ ]:

```
((3200, 384), (400, 384), (3200,))
```

In [ ]:

```
y.value_counts(1)
```

Out[ ]:

```
0    0.25
1    0.25
2    0.25
3    0.25
Name: Category, dtype: float64
```

In [ ]:

```
y_train.value_counts(1)
```

Out[ ]:

```
3    0.25
0    0.25
2    0.25
1    0.25
Name: Category, dtype: float64
```

In [ ]:

```
y_test.value_counts(1)
```

Out[ ]:

```
0    0.2800
1    0.2525
3    0.2425
2    0.2250
Name: Category, dtype: float64
```

Random Forest Model (base)

In [ ]:

```
# defining a function to compute different metrics to check performance of a classification model built using skl
earn

def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred,average="weighted")  # to compute Recall
    precision = precision_score(target, pred,average="weighted")  # to compute Precision
    f1 = f1_score(target, pred,average="weighted")  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    return df_perf
```

In [ ]:

```
## Building the model
rf = RandomForestClassifier(random_state = 42)

## Compete the code to fit the model on X_train and y_train
rf.fit(X_train, y_train)
```

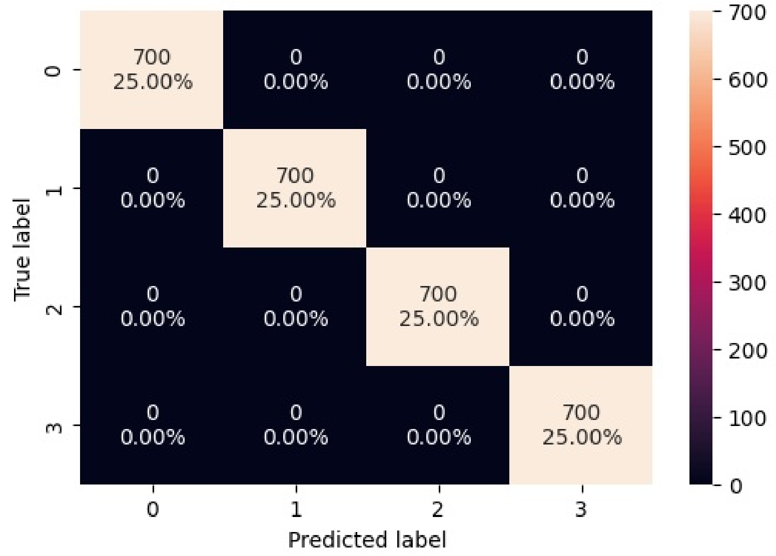
Out[ ]:

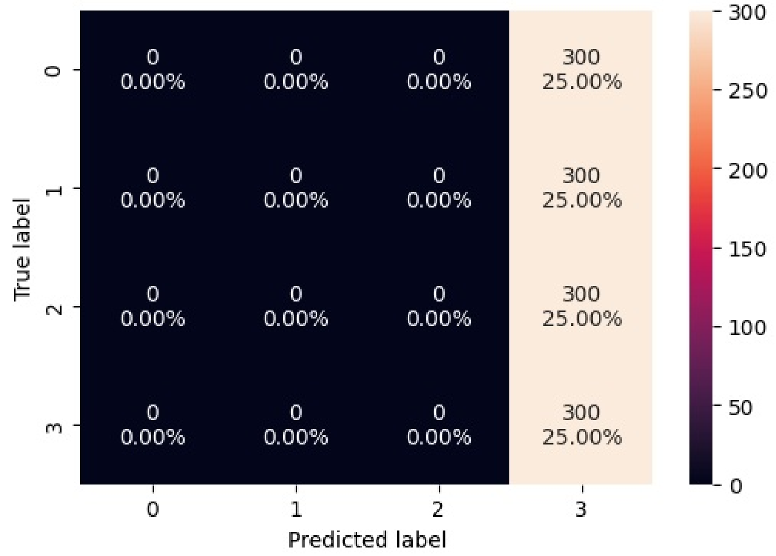
```
▾         RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Confusion Matrix

In [ ]:

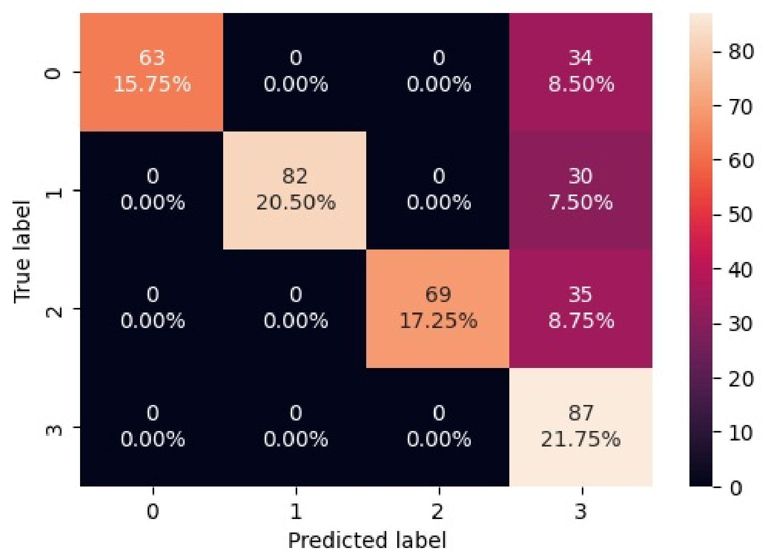
```
## To get the confusion matrix on X_train and y_train
confusion_matrix_sklearn(rf, X_train, y_train)
confusion_matrix_sklearn(rf, X_test, y_test)
```





In [ ]:

```
## Write the code to get the confusion matrix for X_valid and y_valid
confusion_matrix_sklearn(rf, X_valid, y_valid)
```



In [ ]:

```
# Predicting on train data
y_pred_train = rf.predict(X_train)

# Predicting on validation data
y_pred_valid = rf.predict(X_valid)
```

Classification report

In [ ]:

```
## Classification report for train data
print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

           0       1.00      0.70      0.82       791
           1       1.00      0.70      0.82       787
           2       1.00      0.71      0.83       806
           3       0.53      1.00      0.70       816

    accuracy                           0.78      3200
   macro avg       0.88      0.78      0.79      3200
weighted avg       0.88      0.78      0.79      3200
```

In [ ]:

```
## Write the code to get the classification report for validation data
print(classification_report(y_valid, y_pred_valid))
```

```
              precision    recall  f1-score   support

           0       1.00      0.65      0.79        97
           1       1.00      0.73      0.85       112
           2       1.00      0.66      0.80       104
           3       0.47      1.00      0.64        87

    accuracy                           0.75       400
   macro avg       0.87      0.76      0.77       400
weighted avg       0.88      0.75      0.77       400
```

In [ ]:

```
## Storing the metrics
rf_train_perf = model_performance_classification_sklearn(
    rf, X_train, y_train
)
```

In [ ]:

```
## Storing the metrics
rf_valid_perf = model_performance_classification_sklearn(
    rf, X_valid, y_valid
)
```

Random Forest (with class_weights)

In [ ]:

```
## Building the model
rf_balanced = RandomForestClassifier(class_weight="balanced", random_state=42)

## Complete the code to fit the model on X_train and y_train
rf_balanced.fit(X_train, y_train)
```

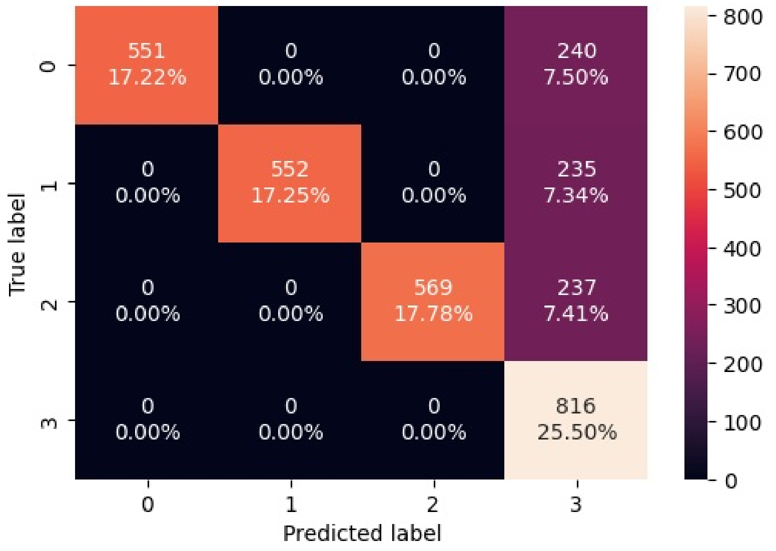
Out[ ]:

```
▾ RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=42)
```

Confusion Matrix

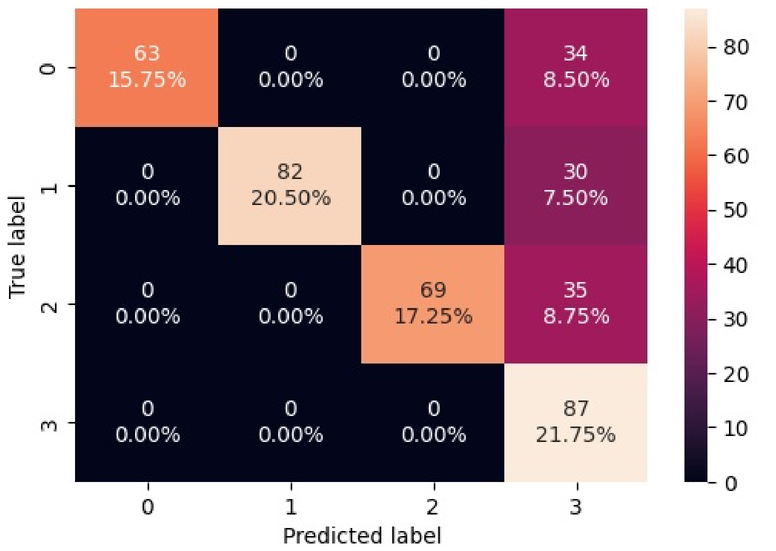
In [ ]:

```
## To get the confusion matrix on X_train and y_train
confusion_matrix_sklearn(rf_balanced, X_train, y_train)
```



In [ ]:

```
## Write the code to get the confusion matrix for X_valid and y_valid
confusion_matrix_sklearn(rf, X_valid, y_valid)
```



In [ ]:

```
## Predicting on train data
y_pred_train = rf_balanced.predict(X_train)

## Complete the code to predict the model on X_valid
y_pred_valid = rf_balanced.predict(X_valid)
```

Classification report

In [ ]:

```
## Classification report for train data
print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       700
           1       1.00      1.00      1.00       700
           2       1.00      1.00      1.00       700
           3       1.00      1.00      1.00       700

    accuracy                           1.00      2800
   macro avg       1.00      1.00      1.00      2800
weighted avg       1.00      1.00      1.00      2800
```

In [ ]:

```
## Write the code to get the classification report for validation data
print(classification_report(y_valid, y_pred_valid))
```

```
              precision    recall  f1-score   support

           0       1.00      0.65      0.79        97
           1       1.00      0.73      0.85       112
           2       1.00      0.66      0.80       104
           3       0.47      1.00      0.64        87

    accuracy                           0.75       400
   macro avg       0.87      0.76      0.77       400
weighted avg       0.88      0.75      0.77       400
```

In [ ]:

```
## Storing the metrics
rf_bal_train_perf = model_performance_classification_sklearn(
    rf_balanced, X_train, y_train
)
```

In [ ]:

```
## Complete the code to store the metrics of validation data
rf_bal_valid_perf = model_performance_classification_sklearn(
    rf_balanced, X_valid, y_valid
)
```

Random Forest (with hyperparamter tuning)

In [ ]:

```
## Building the model
rf_tuned = RandomForestClassifier(class_weight="balanced", random_state=42)

## Defining the hyperparameter grid for tuning
parameters = {
    "max_depth": list(np.arange(4, 10, 3)),
    "max_features": ["sqrt", 0.5, 0.7],
    "min_samples_split": [5, 6],
    "n_estimators": np.arange(30, 110, 15),
}

## Defining the type of scoring used to compare parameter combinations
## We need to specify the mechanism of averaging as we have more than 2 target classes
scorer = make_scorer(recall_score, average='weighted')
## Running the grid search
grid_obj = GridSearchCV(rf_tuned, parameters, scoring=scorer, cv=3, n_jobs=-1)

## Complete the code to fit the model on X_train and y_train
grid_obj = grid_obj.fit(X_train, y_train)
```

In [ ]:

```
print(grid_obj)
```

```
GridSearchCV(cv=3,
             estimator=RandomForestClassifier(class_weight='balanced',
                                              random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [4, 7],
                         'max_features': ['sqrt', 0.5, 0.7],
                         'min_samples_split': [5, 6],
                         'n_estimators': array([ 30,  45,  60,  75,  90, 105])},
             scoring=make_scorer(recall_score, average=weighted))
```

In [ ]:

```
## Create a new model with the best combination of parameters
rf_tuned = grid_obj.best_estimator_

## Fit the new model to X_train and y_train
rf_tuned.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=4,
                       min_samples_split=5, n_estimators=30, random_state=42)
```

In [ ]:

```
## Creating a new model with the best combination of parameters
rf_tuned = grid_obj.best_estimator_

## Complte the code to fit the new model to X_train and y_train
rf_tuned.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=4,
                       min_samples_split=5, n_estimators=30, random_state=42)
```

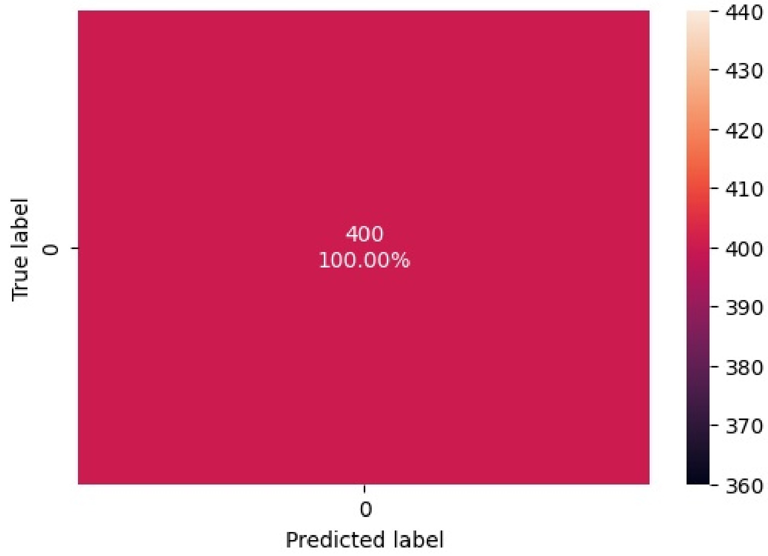
Confusion Matrix

In [ ]:

```
## Write the code to get the classification report for train data
confusion_matrix_sklearn(rf_tuned, X_train, y_train)
```

In [ ]:

```
## Write the code to get the classification report for validation data
confusion_matrix_sklearn(rf_tuned, X_valid, y_valid)
```



In [ ]:

```
## Complete the code to predict the model on train data
y_pred_train = rf_tuned.predict(X_train)

## Complete the code to predict the model on validation data
y_pred_valid = rf_tuned.predict(X_valid)
```

Classification report

In [ ]:

```
## Write the code to get the classification report for train data
print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      3200

    accuracy                           1.00      3200
   macro avg       1.00      1.00      1.00      3200
weighted avg       1.00      1.00      1.00      3200
```

In [ ]:

```
## Write the code to get the classification report for validation data
print(classification_report(y_valid, y_pred_valid))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       400

    accuracy                           1.00       400
   macro avg       1.00      1.00      1.00       400
weighted avg       1.00      1.00      1.00       400
```

In [ ]:

```
## Complete the code to store the metrics of train data
rf_tuned_train_perf = model_performance_classification_sklearn(
    rf_tuned, X_train, y_train
)
```

In [ ]:

```
## Complete the code to store the metrics of validation data
rf_tuned_valid_perf = model_performance_classification_sklearn(
    rf_tuned, X_valid, y_valid
)
```

Model Building - Transformer

Target Mapping

In [ ]:

```
class_map = {0:"World",1:"Sports",2:"Business",3:"Sci/Tech"}
```

In [ ]:

```
class_map
```

Out[ ]:

```
{0: 'World', 1: 'Sports', 2: 'Business', 3: 'Sci/Tech'}
```

In [ ]:

```
reverse_class_map = {}
for key,value in class_map.items():
    reverse_class_map[value]=key

reverse_class_map
```

Out[ ]:

```
{'World': 0, 'Sports': 1, 'Business': 2, 'Sci/Tech': 3}
```

Defining the Tokenizer

In [ ]:

```
## Initializing a T5 tokenizer using the pre-trained model
tokenizer = T5Tokenizer.from_pretrained("google/flan-t5-large")
```

```
You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5T
okenizer'>. This is expected, and simply means that the `legacy` (previous) behavior will be used so
nothing changes for you. If you want to use the new behaviour, set `legacy=False`. This should only
be set if you understand what it means, and thoroughly read the reason why this was added as explain
ed in https://github.com/huggingface/transformers/pull/24565
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-
tuned or trained.
```

Defining the Model

In [ ]:

```
## Initializing a T5 model for conditional generation using the pre-trained model "google/flan-t5-large"
model = T5ForConditionalGeneration.from_pretrained("google/flan-t5-large", load_in_8bit=True, device_map="auto")
```

```
The `load_in_4bit` and `load_in_8bit` arguments are deprecated and will be removed in the future ver
sions. Please, pass a `BitsAndBytesConfig` object in `quantization_config` argument instead.
```

Functions for making predictions

In [ ]:

```
## Defining a function to compute different metrics.

def model_performance_classification(pred, target):
    """
    Function to compute different metrics to check classification model performance

    pred : prediction of the target variable.
    target: dependent variable
    """


    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred,average="weighted")  # to compute Recall
    precision = precision_score(target, pred,average="weighted")  # to compute Precision
    f1 = f1_score(target, pred,average="weighted")  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    return df_perf
```

In [ ]:

```
## Creating a function to plot the confusion matrix
def plot_confusion_matrix(actual, predicted):
    cm = confusion_matrix(actual, predicted)

    plt.figure(figsize = (5, 4))
    label_list = ['World','Sports','Business','Sci/Tech']
    sns.heatmap(cm, annot = True,  fmt = '.0f', xticklabels = label_list, yticklabels = label_list)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

In [ ]:

```
## Defining a function to generate, process, and return a response
def generate_response(prompt):
    input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to("cuda")    ### using the tokenizer to create
tokens in tensor format from an input
    outputs = model.generate(input_ids, max_length=16, do_sample=True, temperature=0.001)    ### generating the m
odel output in tensor format
    return tokenizer.decode(outputs[0])[6:-4]    ### using the tokenizer to decode the model output, and then ret
urn it
```

In [ ]:

```
## Checking a customer review and it's sentiment
print('Article:\t', data.iloc[4]["Article"])
print('Actual Category:\t', class_map[y[4]])
```

```
Article:         U.S. Says N.Korea Blast Probably Not Nuclear  SEOUL (Reuters) - A huge explosion ro
cked North Korea last  week but U.S. and South Korean officials said on Sunday it was  unlikely to h
ave been a nuclear weapons test despite the  appearance of a "peculiar cloud" over the area.
Actual Category:         Sports
```

Base Prompt for Prediction

In [ ]:

```
## Defining a prompt which tells the model what to do
sys_prompt = """
    <Write the instruction for the task here>
"""

## Predicting the category using the model by incorporating the system prompt and the provided review text

pred_sent = generate_response(
    """
        {}
        news article: '{}'
    """.format(sys_prompt, X[4])
)

print(pred_sent)
```

Token indices sequence length is longer than the specified maximum sequence length for this model (3329 > 512). Running this sequence through the model will result in indexing errors

he task is to find the most relevant information for the following news i

In [ ]:

```
## Defining a function to generate a sentiment prediction
def predict_category(news_article):
    pred = generate_response(
        """
            {}
            news article: '{}'
        """.format(sys_prompt,news_article)
    )

    if "Sports" in pred:
       pred="Sports"
    elif "Business" in pred:
       pred="Business"
    elif "World" in pred:
       pred="World"
    else:
      pred="Sci/Tech"

    return reverse_class_map[pred]
```

In [ ]:

```
## Selecting and assigning specific columns
X_train = data.iloc[y_train.index]["Article"]
X_valid = data.iloc[y_test.index]["Article"]
X_test = data.loc[y_valid.index]["Article"]
```

In [ ]:

```
## Applying predict_category function on the train and validation data
from sklearn.preprocessing import LabelEncoder
reverse_class_map = LabelEncoder().inverse_transform(y_train)
y_pred_train_flan = X_train.apply(predict_category)
y_pred_valid_flan = X_valid.apply(predict_category)
```

In [ ]:

```
## Plotting the confusion matrix
plot_confusion_matrix(y_train, y_pred_train_flan)
```

In [ ]:

```
## Complete the code to get the confusion matrix for validation data
plot_confusion_matrix(y_valid, y_pred_valid_flan)
```

In [ ]:

```
## Getting the classification report for train data
print(classification_report(y_train, y_pred_train_flan))
```

In [ ]:

```
## Complete the code to get the classification report for validation data
print(classification_report(y_valid, y_pred_valid_flan))
```

In [ ]:

```
## Storing the metrics
flan_train_base = model_performance_classification(y_pred_train_flan,y_train)
flan_valid_base = model_performance_classification(y_pred_valid_flan,y_valid)
```

Improved Prompt for Prediction

In [ ]:

```
# defining a prompt which tells the model what to do
sys_prompt = """
    <Write the instruction for the task here>
    <This prompt will be an improved version of the previous prompt to improve model performance>
"""

# predicting the sentiment using the model by incorporating the system prompt and the provided review text

pred_sent = generate_response(
    """
        {}
        news article: '{}'
    """.format(sys_prompt, X[4])
)

print(pred_sent)
```

```
<unk> The following are the new prompts for the news article:
```

In [ ]:

```
## Applying predict_category function on the train and validation data
y_pred_train_flan_imp = X_train.apply(predict_category)
y_pred_valid_flan_imp = X_valid.apply(predict_category)
```

In [ ]:

```
## Plotting the confusion matrix for train data
plot_confusion_matrix(y_train, y_pred_train_flan_imp)
```

In [ ]:

```
## Complete the codet to get the confusion matrix for validation data
plot_confusion_matrix(y_valid, y_pred_valid_flan_imp)
```

In [ ]:

```
## Getting the classification report for train data
print(classification_report(y_train, y_pred_train_flan_imp))
```

In [ ]:

```
## Complete the code to get the classification report for validation data
print(classification_report(y_valid, y_pred_valid_flan_imp))
```

In [ ]:

```
## Storing the metrics
flan_train_imp = model_performance_classification(y_pred_train_flan_imp,y_train)
flan_valid_imp = model_performance_classification(y_pred_valid_flan_imp,y_valid)
```

Model Performance Comparison and Final Model Selection

In [ ]:

```
## Training performance comparison

models_train_comp_df = pd.concat(
    [
        rf_train_perf.T,
        rf_bal_train_perf.T,
        rf_tuned_train_perf.T,
        flan_train_base.T,
        flan_train_imp.T
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Random Forest(base)",
    "Random Forest with class_weights",
    "Random Forest(tuned)",
    "Flan (base prompt)",
    "Flan (improvised prompt)"
]
print("Training performance comparison:")
models_train_comp_df
```

In [ ]:

```
## Validation set performance comparison
models_valid_comp_df = pd.concat(
    [
        rf_valid_perf.T,
        rf_bal_valid_perf.T,
        rf_tuned_valid_perf.T,
        flan_valid_base.T,
        flan_valid_imp.T
    ],
    axis=1,
)
models_valid_comp_df.columns = [
    "Random Forest(base)",
    "Random Forest with class_weights",
    "Random Forest(tuned)",
    "Flan (base prompt)",
    "Flan (improvised prompt)"
]
print("Validation set performance comparison:")
models_valid_comp_df
```

Pick the best model from the above table and apply on test data

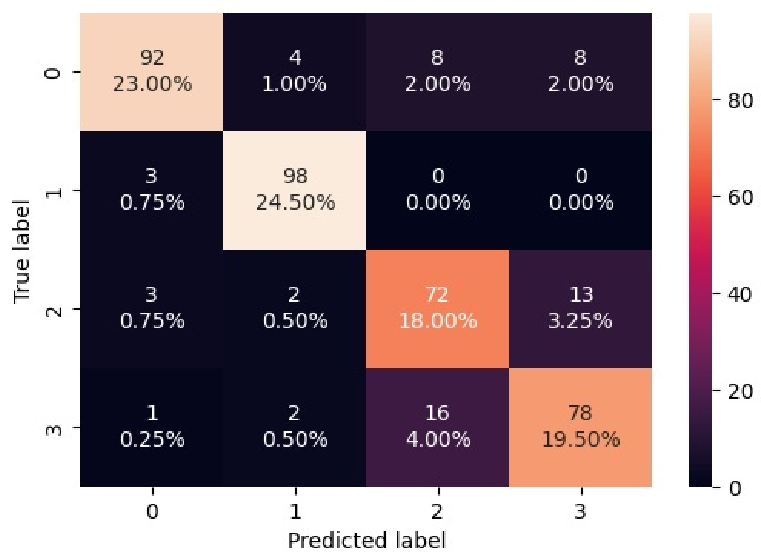
In [ ]:

```
## Assigns test rows based on index
X_test = embedding_matrix[y_test.index]
```

In [ ]:

```
print(confusion_matrix_sklearn(rf_balanced, X_test, y_test))
```

None



In [ ]:

```
# Predicting on test data
y_pred_test = rf_balanced.predict(X_test)
```

In [ ]:

```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.93      0.82      0.87       112
           1       0.92      0.97      0.95       101
           2       0.75      0.80      0.77        90
           3       0.79      0.80      0.80        97

    accuracy                           0.85       400
   macro avg       0.85      0.85      0.85       400
weighted avg       0.85      0.85      0.85       400
```

Actionable Insights and Recommendations

- The Random Forest is able to give generalized prediction on training & testing datasets and is able to explain maximum information where accuracy of 85% on test dataset & F1 score of 95% on test dataset.
- The precision & recall are likewise both high which are 92% & 97% respectively. The confusion matrix is able to identify a higher percentage of articles. The model is still helpful, as only a small subset of articles will need further reevaluation.
- The leftover articles can be categorized by reevaluating them. The above model is still efficient in categorizing.