```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
import joblib
import pickle



data = pd.read_csv("https://raw.githubusercontent.com/amankharwal/Website-data/master/dataset.csv")
print(data.head())
```

```
                                                Text  language
0  klement gottwaldi surnukeha palsameeriti ning ...  Estonian
1  sebes joseph pereira thomas  på eng the jesuit...   Swedish
2  ถนนเจริญกรุง อักษรโรมัน thanon charoen krung เ...      Thai
3  விசாகப்பட்டினம் தமிழ்ச்சங்கத்தை இந்துப் பத்திர...     Tamil
4  de spons behoort tot het geslacht haliclona en...     Dutch
```

```
print(data.tail())
```

```
                                                    Text  language
21995  hors du terrain les années  et  sont des année...    French
21996  ใน พศ  หลักจากที่เสด็จประพาสแหลมมลายู ชวา อินเ...      Thai
21997  con motivo de la celebración del septuagésimoq...   Spanish
21998  年月，當時還只有歲的她在美國出道，以mai-k名義推出首張英文《baby i like》，由...   Chinese
21999   aprilie sonda spațială messenger a nasa și-a ...  Romanian
```

```
data.isnull().sum()
```

```
Text        0
language    0
dtype: int64
```

```
data["language"].value_counts()
```

```
Estonian      1000
Swedish       1000
English       1000
Russian       1000
Romanian      1000
Persian       1000
Pushto        1000
Spanish       1000
Hindi         1000
Korean        1000
Chinese       1000
French        1000
Portugese     1000
Indonesian    1000
Urdu          1000
Latin         1000
Turkish       1000
Japanese      1000
Dutch         1000
Tamil         1000
Thai          1000
Arabic        1000
Name: language, dtype: int64
```

```
x = np.array(data["Text"])
y = np.array(data["language"])

cv = CountVectorizer()
X = cv.fit_transform(x)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.001,
                                                    random_state=99)
```

Vectorizer

A vectorizer, in the context of natural language processing (NLP) and machine learning, is a tool or method used to convert text data into a numerical format that can be utilized by machine learning algorithms. The goal is to represent text data in a way that algorithms can understand and make predictions or perform various tasks.

One common type of vectorizer used for text data is the **CountVectorizer**. Its function is to convert a collection of text documents into a matrix of token counts. This matrix represents the frequency of each word (or token) in each document. The CountVectorizer works through the following steps:

Tokenization: It breaks down each document into individual words or tokens.

Vocabulary Building: It creates a vocabulary, which is a set of all unique words (or tokens) found in the entire collection of documents.

Counting: It counts the occurrences of each word in each document and constructs a matrix where each row corresponds to a document, each column corresponds to a word in the vocabulary, and the values represent the counts of each word in each document.

Other types of vectorizers include **TF-IDF Vectorizer**,** Word Embeddings** (e.g.,** Word2Vec**, **GloVe**), and more. The choice of vectorizer depends on the specific needs of your machine learning task and the characteristics of your text data.

```
from sklearn.feature_extraction.text import CountVectorizer


# Create a CountVectorizer
cv = CountVectorizer()

# Fit and transform the data
X = cv.fit_transform(x)

# Get feature names (words)
feature_names = cv.get_feature_names_out()

# Convert the sparse matrix to a dense array for better visibility
dense_array = X.toarray()

print("Feature Names:", feature_names)
print("Token Counts:\n", dense_array)
```

```
Feature Names: ['aa' 'aaa' 'aabdel' ... 'لاسونه' 'لاتراوسه' 'ｄｅａ論文']
Token Counts:
 [[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]]
```

Model Details

Multinomial Naive Bayes (NB) classifier, as implemented in scikit-learn, does not involve an iterative optimization process with epochs, as is typical in some other machine learning models like neural networks. Instead, it follows a closed-form solution based on probability theory.

Here's an overview of how the Multinomial Naive Bayes model is trained:

Training Process: **Data Preparation:**

Input Data: The training data consists of a set of documents (text) and their corresponding class labels.

Feature Extraction: The documents are typically represented as feature vectors, where each feature corresponds to a term (word) and the value represents the frequency of that term in the document. This step is often done using techniques like TF-IDF (Term Frequency-Inverse Document Frequency).

Parameter Estimation: Prior Probabilities: The prior probability of each class is calculated based on the distribution of class labels in the training data. This is the probability of encountering each class without considering any features.

Likelihood Estimation: For each term in the vocabulary, the likelihood of observing that term given a class is estimated. This is done by calculating the conditional probability of the term occurring in documents of each class.

Model Training:

The Multinomial Naive Bayes model is trained by storing these estimated probabilities.

Making Predictions: Once the model is trained, it can be used to make predictions on new, unseen data.

Data Preparation:

The new documents are transformed into the same feature space as the training data.

Prediction: For each document, the model calculates the probability of it belonging to each class using Bayes' theorem. The class with the highest probability is assigned as the predicted class for that document.

Key Assumption: The "naive" in Naive Bayes comes from the assumption that the features (terms in this case) are conditionally independent given the class label. Although this assumption is often violated in practice (terms in a document may be correlated), the model can still perform well, especially in text classification tasks.

In summary, the Multinomial Naive Bayes model is trained by estimating class priors and conditional probabilities of terms given a class. The training process does not involve iterative optimization with epochs, making it computationally efficient and particularly well-suited for text classification tasks with large vocabularies.

```
model = MultinomialNB()
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

```
1.0
```

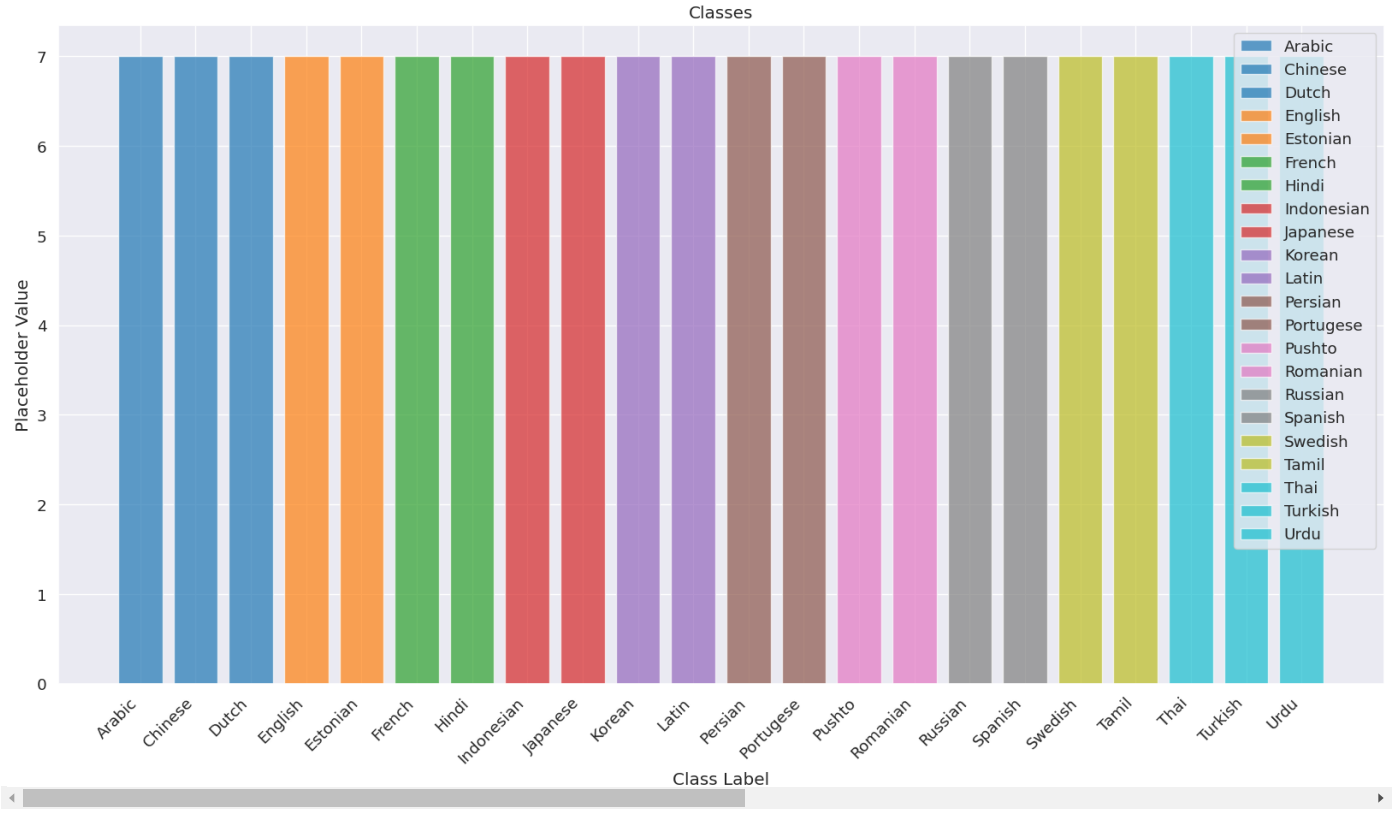
```
print("Classes:", model.classes_)
print("Class Log Priors:", model.class_log_prior_)
print("Class Prior Probabilities:", np.exp(model.class_log_prior_))
print("Feature Log Probabilities:", model.feature_log_prob_)
```

```
Classes: ['Arabic' 'Chinese' 'Dutch' 'English' 'Estonian' 'French' 'Hindi'
 'Indonesian' 'Japanese' 'Korean' 'Latin' 'Persian' 'Portugese' 'Pushto'
 'Romanian' 'Russian' 'Spanish' 'Swedish' 'Tamil' 'Thai' 'Turkish' 'Urdu']
Class Log Priors: [-3.09204396 -3.09004195 -3.09104245 -3.09104245 -3.09004195 -3.09204396
 -3.09004195 -3.09004195 -3.09004195 -3.09104245 -3.09004195 -3.09204396
 -3.09104245 -3.09304646 -3.09004195 -3.09304646 -3.09004195 -3.09204396
 -3.09004195 -3.09204396 -3.09104245 -3.09104245]
Class Prior Probabilities: [0.04540905 0.04550005 0.04545455 0.04545455 0.04550005 0.04540905
 0.04550005 0.04550005 0.04550005 0.04545455 0.04550005 0.04540905
 0.04545455 0.04536355 0.04550005 0.04536355 0.04550005 0.04540905
 0.04550005 0.04540905 0.04545455 0.04545455]
Feature Log Probabilities: [[-12.06189911 -12.75504629 -12.75504629 ... -12.75504629 -12.75504629
  -12.75504629]
 [-12.60973279 -12.60973279 -12.60973279 ... -12.60973279 -12.60973279
  -12.60973279]
 [-12.70959265 -12.01644547 -12.70959265 ... -12.70959265 -12.70959265
  -12.70959265]
 ...
 [-12.02724797 -12.72039515 -12.72039515 ... -12.72039515 -12.72039515
  -12.72039515]
 [-12.00603203 -12.69917921 -12.69917921 ... -12.69917921 -12.69917921
  -12.69917921]
 [-12.74961712 -12.74961712 -12.74961712 ... -12.74961712 -12.74961712
  -12.74961712]]
```

```
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have the classes (model.classes_)
# Replace model.classes_ with the actual classes obtained from your model

# Get the classes
classes = model.classes_
```

```
# Define a colormap
colors = plt.cm.get_cmap('tab10', len(classes))

# Create a bar chart with different colors for each class
plt.figure(figsize=(20, 10))
bars = plt.bar(range(len(classes)), [7] * len(classes), color=colors(range(len(classes))), alpha=0.7)
plt.xticks(range(len(classes)), labels=classes, rotation=45, ha='right')
plt.xlabel('Class Label')
plt.ylabel('Placeholder Value')  # This is a placeholder since we only want to show colors
plt.title('Classes')

# Add a legend with class labels
plt.legend(bars, classes, loc='upper right')

plt.show()
```

```
<ipython-input-45-50035e10141a>:11: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be re
  colors = plt.cm.get_cmap('tab10', len(classes))
```

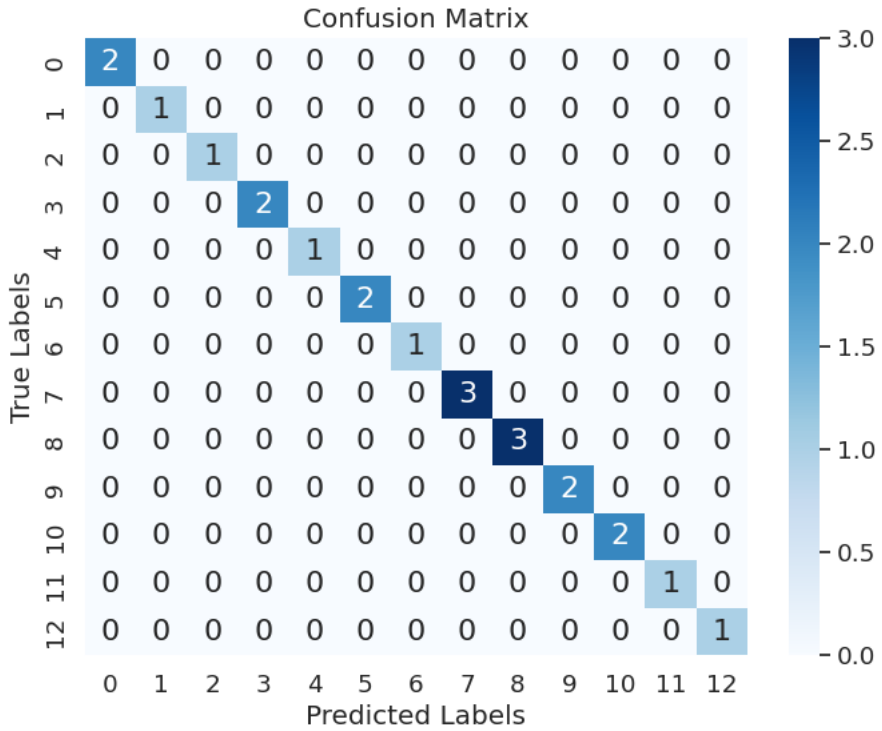



```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score

# Make predictions on the test set
y_pred = model.predict(X_test)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[2 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 2 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 2 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 3 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 3 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 2 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 2 0 0 0]
```

```
      [0 0 0 0 0 0 0 0 0 0 0 1 0]
      [0 0 0 0 0 0 0 0 0 0 0 0 1]]
```

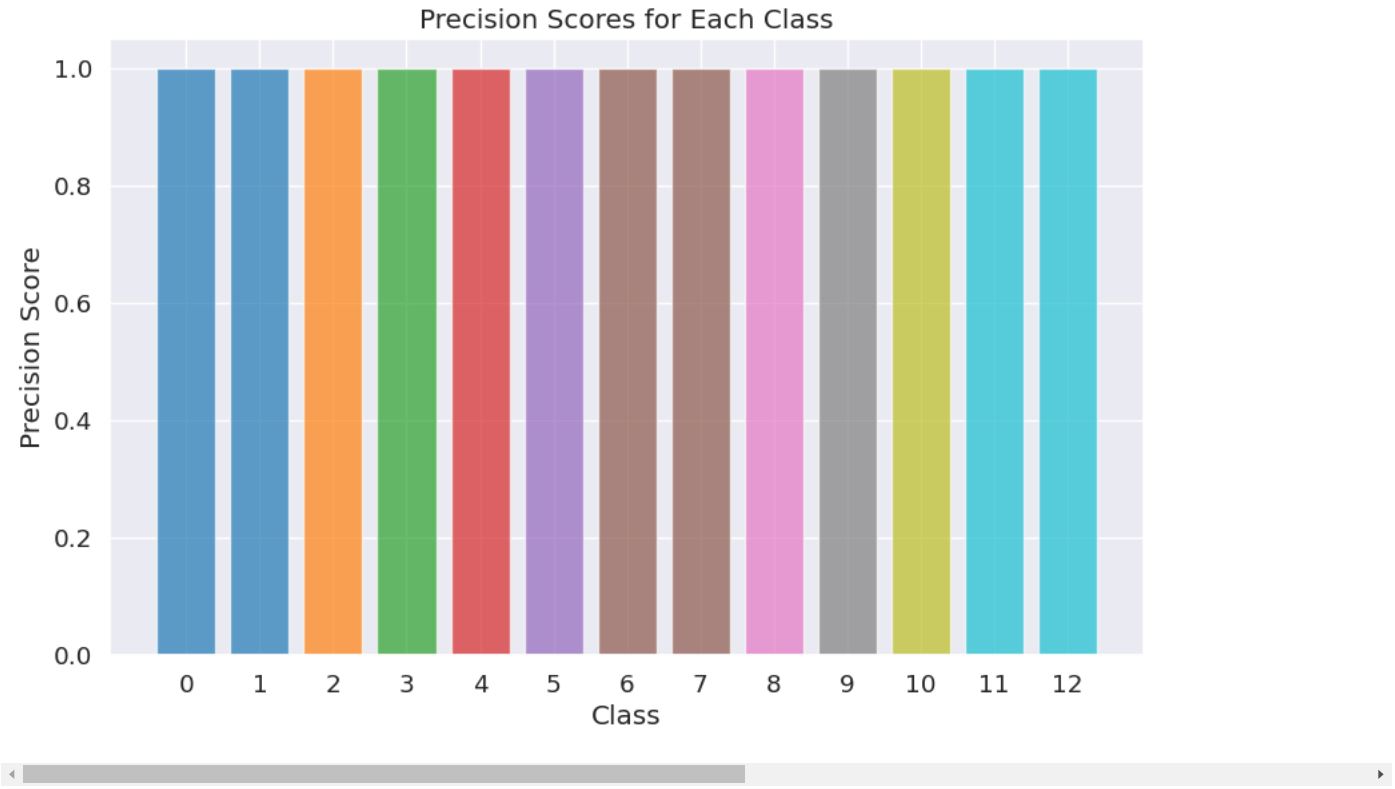
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

# Assuming you have the confusion matrix calculated
cm = confusion_matrix(y_test, y_pred)

# Plotting with matplotlib and seaborn
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)  # Adjust font size
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", annot_kws={"size": 16})

# Adding labels and title
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")

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




```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, confusion_matrix
import numpy as np

# Assuming you have the true labels (y_test) and predicted labels (y_pred)
# Replace y_test and y_pred with your actual test labels and predicted labels

# Calculate precision scores for each class
precision_scores = precision_score(y_test, y_pred, average=None)

# Get the number of classes
num_classes = len(np.unique(y_test))

# Define colors for each class
colors = plt.cm.get_cmap('tab10', num_classes)
```

```
# Create a bar chart with different colors for each class
plt.figure(figsize=(10, 6))
bars = plt.bar(range(num_classes), precision_scores, color=colors(range(num_classes)), alpha=0.7)
plt.xticks(range(num_classes), labels=range(num_classes))  # Assuming class labels are integers
plt.xlabel('Class')
plt.ylabel('Precision Score')
plt.title('Precision Scores for Each Class')

# Add legend with class labels
#plt.legend(bars, [f'Class {i}' for i in range(num_classes)], loc='upper right')

plt.show()
```

```
<ipython-input-33-ac4b121e64d5>:15: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be re
  colors = plt.cm.get_cmap('tab10', num_classes)
```




```
# Precision
precision = precision_score(y_test, y_pred, average='weighted')
print("Precision:", precision)
```

```
Precision: 1.0
```

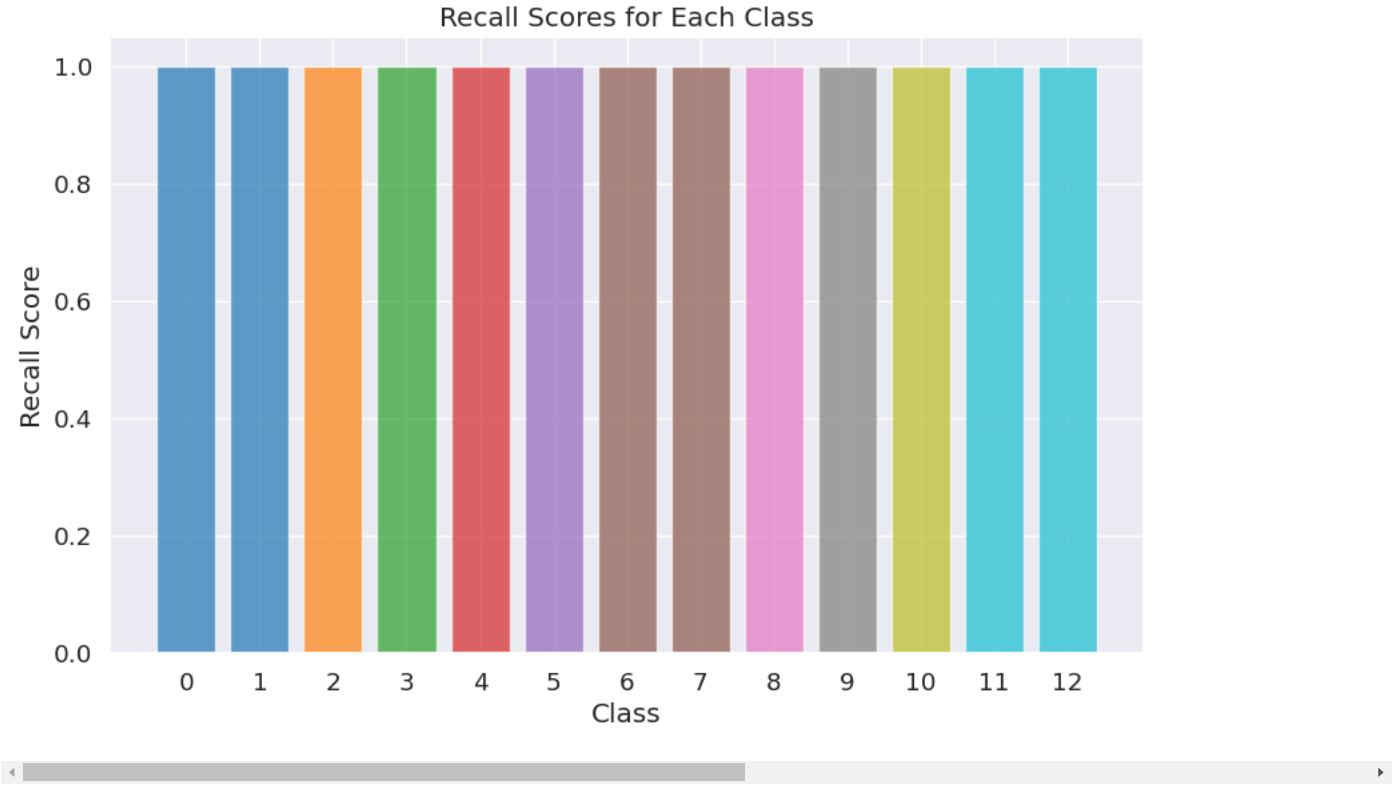
```
# Recall
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)
```

```
Recall: 1.0
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import recall_score
import numpy as np

# Assuming you have the true labels (y_test) and predicted labels (y_pred)
# Replace y_test and y_pred with your actual test labels and predicted labels

# Calculate recall scores for each class
recall_scores = recall_score(y_test, y_pred, average=None)
```

```
# Get the number of classes
num_classes = len(np.unique(y_test))

# Define colors for each class
colors = plt.cm.get_cmap('tab10', num_classes)

# Create a bar chart for recall scores
plt.figure(figsize=(10, 6))
bars_recall = plt.bar(range(num_classes), recall_scores, color=colors(range(num_classes)), alpha=0.7)
plt.xticks(range(num_classes), labels=range(num_classes))  # Assuming class labels are integers
plt.xlabel('Class')
plt.ylabel('Recall Score')
plt.title('Recall Scores for Each Class')
#plt.legend(bars_recall, [f'Class {i}' for i in range(num_classes)], loc='upper right')

plt.show()
```

```
<ipython-input-30-c1c0db9920e9>:15: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be re
  colors = plt.cm.get_cmap('tab10', num_classes)
```



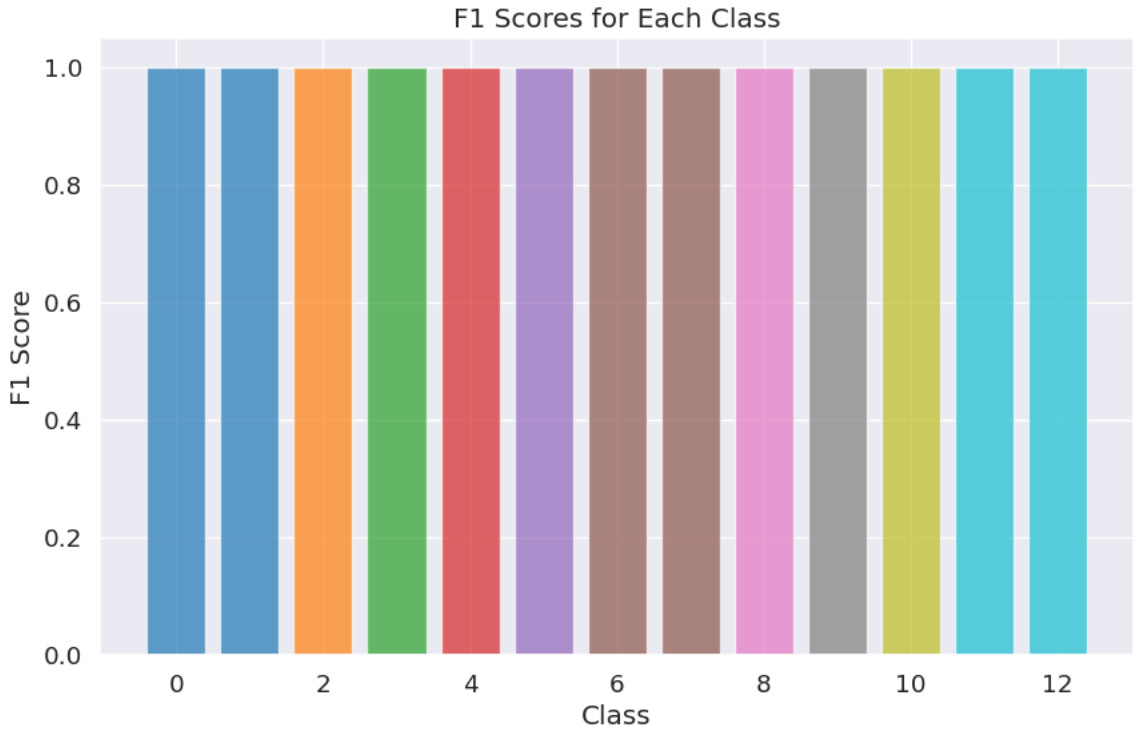

```
# F1 Score
f1 = f1_score(y_test, y_pred, average='weighted')
print("F1 Score:", f1)
```

```
F1 Score: 1.0
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, f1_score
import numpy as np

# Assuming you have the true labels (y_test) and predicted labels (y_pred)
# Replace y_test and y_pred with your actual test labels and predicted labels

# Calculate precision and F1 scores for each class
f1_scores = f1_score(y_test, y_pred, average=None)

# Define colors for each class
colors = plt.cm.get_cmap('tab10', num_classes)

# Create a bar chart with different colors for each class
plt.figure(figsize=(10, 6))
```

```
# Plot F1 Scores
bars_f1 = plt.bar(range(num_classes), f1_scores, color=colors(range(num_classes)), alpha=0.7)
plt.xlabel('Class')
plt.ylabel('F1 Score')
plt.set_title('F1 Scores for Each Class')
#ax2.legend(bars_f1, [f'Class {i}' for i in range(num_classes)], loc='upper right')

plt.show()
```

```
<ipython-input-28-e19a3e0a6580>:12: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be re
  colors = plt.cm.get_cmap('tab10', num_classes)
```




```
# Accessing feature log probabilities
feature_log_probs = model.feature_log_prob_

# Print or use these probabilities as needed
print("Feature Log Probabilities:", feature_log_probs)
```

```
Feature Log Probabilities: [[-12.06189911 -12.75504629 -12.75504629 ... -12.75504629 -12.75504629
  -12.75504629]
 [-12.60973279 -12.60973279 -12.60973279 ... -12.60973279 -12.60973279
  -12.60973279]
 [-12.70959265 -12.01644547 -12.70959265 ... -12.70959265 -12.70959265
  -12.70959265]
 ...
 [-12.02724797 -12.72039515 -12.72039515 ... -12.72039515 -12.72039515
  -12.72039515]
 [-12.00603203 -12.69917921 -12.69917921 ... -12.69917921 -12.69917921
  -12.69917921]
 [-12.74961712 -12.74961712 -12.74961712 ... -12.74961712 -12.74961712
  -12.74961712]]
```

```
joblib.dump(model, "Language_Detection_Model.pkl")
```

```
['Language_Detection_Model.pkl']
```

```
user = input("Enter a Text: ")
data = cv.transform([user]).toarray()
output = model.predict(data)
print(output)
```

```
Enter a Text: hello how are you
['English']
```

```
user = input("Enter a Text: ")
data = cv.transform([user]).toarray()
# Save the model to a file using pickle
with open("Language_Detection_Model.pkl", "wb") as file:
    pickle.dump(model, file)

output = model.predict(data)
print(output)
```

```
Enter a Text: how are you
['English']
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np

data = pd.read_csv("https://raw.githubusercontent.com/amankharwal/Website-data/master/dataset.csv")
# print(data.head())

x = np.array(data["Text"])
y = np.array(data["language"])

cv = CountVectorizer()
X = cv.fit_transform(x)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.001,
                                                    random_state=99)

model = MultinomialNB()
model.fit(X_train,y_train)
model.score(X_test,y_test)
# Number of experiments or iterations
num_experiments = 10

# Lists to store accuracy values
accuracy_values = []

# Run multiple experiments
for _ in range(num_experiments):
    # Create a model (example: Multinomial Naive Bayes)
    model = MultinomialNB()
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Store accuracy value
    accuracy_values.append(accuracy)

# Plot the accuracy values
```

```
plt.figure(figsize=(8, 6))
plt.plot(range(1, num_experiments + 1), accuracy_values, marker='o', linestyle='-')
plt.xlabel('Experiment')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Over Experiments')
plt.grid(True)
plt.show()
```