IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Nepali News Classification using Naive Bayes: A Data-Driven Approach

KHADKA PILOT[1*] AND POUDEL KSHITIZ[2*]

[1]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: pilot.076bct025@tcioe.edu.np)
[2]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: kshitizpoudel18@gmail.com)

Corresponding author: Khadka Pilot (e-mail: pilot.076bct025@tcioe.edu.np).

* Authors contributed equally

ABSTRACT The Nepali news landscape has witnessed an exponential growth in digital content, demanding mechanisms for news classification to help information retrieval and organization. This study explores the application of the Naive Bayes algorithm for the classification of Nepali news articles. The proposed approach leverages the simplicity and effectiveness of Naive Bayes, which assumes feature independence to achieve high classification accuracy while minimizing computational overhead. A comprehensive dataset of labeled Nepali news articles across diverse categories is utilized for model training and evaluation. The performance of the Naive Bayes classifier is assessed through various metrics, including precision and recall. The experimental results demonstrate promising outcomes, indicating the viability of the Naive Bayes approach for Nepali news classification tasks.

INDEX TERMS Naive Bayes, News classification

I. INTRODUCTION

OUR ancestors were motivated to find stable sources of food, water, and shelter to survive and protect their families. This deep need for certainty is true today as we work to address our basic needs and provide for ourselves and loved ones in a highly unpredictable world. The reality of our everyday existence is that things change every day, even when we wish that they would stay the same, and nothing is certain. Random events can and do happen when we're unprepared to deal with the fallout. [1]

This inherent uncertainty is not exclusive to our daily lives but can also be observed in the field of data mining. [2] Several reasons contribute to uncertainty in this domain:

- Inherent properties of the system being modeled: Take, for instance, quantum mechanics, where the behavior of subatomic particles is probabilistic.
- Incomplete observations: Weather forecasting serves as an example, where numerous variables interact, making it difficult to accurately include all variables.
- Incomplete modeling: While creating models, we often need to discard certain information. Such omissions can introduce uncertainty in our predictions, as seen in discretization of values.

Probability serves as a tool for quantifying uncertainty. One widely used technique that deals with uncertainty is the Naive Bayes classifier. This method relies on probabilities and operates under the assumption of conditional independence between features to make predictions.

II. METHODOLOGY

A. THEORY

Bayes classifiers, based on Bayes theorem, are statistical classifiers that predict class membership probabilities.

The Naive Bayesian classifier assumes that the effect of an attribute value on a given class is independent of the values of other attributes. This assumption is called conditional independence.

Let $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ be a data tuple. In Bayesian terms, $\mathbf{X}$ is considered as "evidence." Let H be the hypothesis that $\mathbf{X}$ belongs to a specified class C. For classification, we want to determine $P(H|\mathbf{X})$, which represents the probability that hypothesis H holds true given tuple $\mathbf{X}$. [3]

The terms used in Bayes' theorem are as follows:

$$\begin{aligned} P(H|\mathbf{X}) &\text{ is the posterior probability,} \\ P(H) &\text{ is the prior probability,} \\ P(\mathbf{X}|H) &\text{ is the likelihood,} \\ P(\mathbf{X}) &\text{ is the prior probability of } \mathbf{X}. \end{aligned}$$

Bayes' theorem provides a way of calculating the posterior probability $P(H|\mathbf{X})$ from $P(\mathbf{X}|H)$, $P(\mathbf{X})$, and $P(H)$, according to the following equation:

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H) \cdot P(H)}{P(\mathbf{X})} \tag{1}$$

The numerator is focused on, as the denominator is constant for all classes. $C_1, C_2, \ldots, C_m$. So, only $P(\mathbf{X}|C_i) \cdot P(C_i)$ needs to be maximized. However, given the large number of attributes, it would be computationally expensive to calculate $P(\mathbf{X}|C_i)$ directly. To reduce computation, the naive assumption is that of class conditional independence, i.e., there is no dependence relation among the attributes. This allows us to represent $P(\mathbf{X}|C_i)$ as the product of the individual attribute probabilities:

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \tag{2}$$

When we break text into words/features,multiplication of all likelihoods results in almost zero probabilities . to solve this we apply log which converts sum to product.Unlike multiplication logarithm is monotonously increasing function so the probablities go on increasing.

$$\sum_{i=1}^{n} \log(p(x_i|C_k)) \tag{3}$$

B. TRAINING NAIVE BAYESIAN CLASSIFIER

For the class prior $P(C)$, the percentage of documents in training set that belong to each class C is determined. [4] Let N_c be the number of documents in our training data with class C, and N_{doc} be the total number of documents. Then, the class prior is calculated as follows:

$$P(C) = \frac{N_c}{N_{\text{doc}}} \tag{4}$$

To find $P(X|C)$, assumption is made that a feature is just the existence of a word w_i in the document's bag of words. Therefore,$P(w_i|c)$ can be computed as the fraction of times the word w_i appears among all words in all documents of topic C. Let V represent the vocabulary, which consists of the union of all word types in all classes, not just the words in one class C. The likelihood term is then calculated as follows:

$$P(w_i|C) = \frac{\text{count}(w_i, C)}{\sum_{w \in V} \text{count}(w, C)} \tag{5}$$

1) Zero frequency problem

Naive bayes faces zero frequency problem where it assigns zero probability to a word in test data set if it wasn't available in training dataset.To solve this smoothing is required.

C. ALGORITHM

D. SYSTEM BLOCK DIAGRAM

Appendix

Algorithm 1 Train Naive Bayes Classifier

```
function TrainNaiveBayes(D, C)
    N_doc = number of documents in D
    N_c = number of documents from D in class C
    logprior[c] ← log (N_c / N_doc)
    V ← vocabulary of D
    bigdoc[c] ← append(d) for d ∈ D with class c
    for each word w in V do
        count(w, c) ← number of occurrences of w in bigdoc[c]
        loglikelihood[w, c] ← log( (count(w,c)+1) / Σ_{w'∈V}(count(w',c)+1) )
    end for
    return logprior, loglikelihood, V
end function
```

Algorithm 2 Test Naive Bayes Classifier

```
function TestNaiveBayes(testdoc, logprior, loglikelihood, C, V)
    sum ← {}
    for c ∈ C do
        sum[c] ← logprior[c]
        for i in 1 to length of testdoc do
            word ← testdoc[i]
            if word ∈ V then
                sum[c] ← sum[c]+loglikelihood[word, c]
            end if
        end for
    end for
    return arg max_{c∈C} sum[c]
end function
```

E. INSTRUMENTATION

The following libraries were instrumental in completion of this project :

- **Pandas**: It is a library for data manipulation and analysis.
  - `df.drop`: Remove unnecessary columns.
  - `df.apply`: Remove punctuation, stopword from text.
- **Matplotlib**: It is a popular data visualization library for Python.
  - `plt.pie`: To visualize distribution of classes.
  - `plt.hist`: To visualize frequency of text length.
- **Scikit-learn**: Scikit-learn features algorithms like regression, classification, etc.
  - `train_test_split`: To split the dataset into training and test sets in the ratio 8:2.
  - `confusion_matrix`: To plot Confusion matrix after prediction.
  - `CountVectorizer,TfidfVectorizer`: To vectorize nepali text.
  - `GaussianNB,MultinomialNB,BernoulliNB`: To implement Naive Bayes for classification.
- **WordCloud**: It was used to visualize text data where the size of the word represents its frequency.

- **NLTK**: NLTK is a leading platform for building Python programs to work with human language data.
  - word_tokenize: To tokenize nepali text

III. WORKING PRINCIPLE

A. DATASET COLLECTION

The Nepali News classification dataset was gathered from diverse online news platforms. With a total of 7,470 instances, this comprehensive dataset is made up of three essential features: 'heading','paragraph' and 'label.' The 'heading' feature represents the concise title of the news, providing a quick glimpse into its content. On the other hand, the 'paragraph' feature delves deeper into the news, presenting a detailed description or body of the article. Finally, the 'label' attribute categorizes the news into one of three types:

- Business
- Sports
- Entertainment

B. DATA CLEANING

In this step, the data was examined for missing values, unexpected characters, and other anomalies. Out of the 7,470 instances, none of the values had null values. The title column was dropped. The reason behind this decision is that the 'paragraph' column already provides a more comprehensive and descriptive representation of the news content. Therefore, retaining the 'title' column would be redundant and offer no additional valuable information.

C. DATA PREPROCESSING

1) Remove punctuation

Various punctuation marks such as commas, colons, vertical bars, and others were thoughtfully removed from the text data. This punctuation removal process ensures that each text is treated equally during subsequent analysis and processing. Figure(2) shows the original sentence and Figure (3) shows the sentence after removing punctuation.

2) Stop word removal

Next, we address the removal of stop words, which are commonly occurring words used in sentence construction. In our dataset, we have identified several stop words such as shown in the figure(4). These words are deemed redundant for sentiment analysis and are therefore eliminated during the preprocessing stage.

3) Tokenization

Tokenization involves division of raw text into smaller unit known as tokens. These tokens can be individual words or entire sentences, depending on the chosen tokenizer. In our dataset, we opted for word tokenization.Figure (1) displays the original text before tokenization. After applying word tokenizer, Figure (2) exhibits the text transformed into a sequence of individual word tokens.

4) Vectorization

Vectorization or word embedding is a fundamental process in natural language processing that involves converting textual data into numerical vectors. By representing documents as numerical vectors, meaningful analytics and machine learning algorithms can be applied.

Term Frequency-Inverse Document Frequency (TF-IDF) is a NLP technique used to quantify the importance of a term in a specific document within a larger collection (corpus) of documents. TF represents the number of times a term appears in a particular document, making it document-specific.

- Raw Frequency: $tf(t) =$ Number of times term t occurs in a document.
- Term Frequency (Normalized):

$$(tf(t) = \frac{\text{Times term } t \text{ occurs in a document}}{\text{Total number of terms in the document}}. \tag{6}$$

Inverse Document Frequency (IDF) is a measure of how common or rare a term is across the entire corpus of documents. It helps in distinguishing terms that are unique and significant within the corpus. The formula to calculate the IDF for a term t is:

$$idf(t) = 1 + \log_e\left(\frac{n}{\mathrm{df}(t)}\right) \tag{7}$$

where: n is the total number of documents in the corpus. $\mathrm{df}(t)$ is the number of documents in the corpus that contain the term t.

The TF-IDF value of a term in a document is the product of its TF and IDF. This combined value serves as a weight that indicates the relevance of the term in that specific document. A higher TF-IDF value implies that the term is more significant and relevant to that particular document compared to other terms.

D. CLASSIFICATION

The dataset is divided into two subsets in an 8:2 ratio, with 80% of the data used for training and the remaining 20% for testing. The training data is then utilized to train the Naive Bayes classifier. After the classifier has been trained, it is tested on the test set to evaluate its performance.

E. EVALUATION

Most real datasets aren't balanced. Even if the dataset is balanced, relying solely on accuracy may provide a misleading sense of model performance. In such cases, the confusion matrix becomes essential as it provides more comprehensive information.

Precision: Precision measures the proportion of cases that the model correctly flags as positive out of all the cases it predicted as positive. It helps us understand the accuracy of positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{8}$$

Recall: Recall (also known as sensitivity) measures the percentage of actual positive cases in the dataset that the model correctly identified as positive. It helps us assess how well the model captures positive instances.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{9}$$

F1-score: The F1 score is defined as the harmonic mean of precision (P) and recall (R) and is calculated using the following equation:

$$F1 = \frac{2 \times P \times R}{P + R} \tag{10}$$

By analyzing both precision and recall, we can gain a more nuanced understanding of the model's performance. High precision indicates that the model is making fewer false positive predictions, while high recall signifies that the model captures more positive cases correctly.

IV. RESULT AND ANALYSIS

A. DATASET ANALYSIS

The dataset consists of the following categories:

- Business: 2628 entries
- Sports: 2574 entries
- Entertainment: 2268 entries

Business: This category includes news related to business, economy, share market, and other economic activities.

Sports: This category covers various sports-related news, including competitions, tournaments, and other sporting events.

Entertainment: This category encompasses news related to movies and music, such as movie releases and music album launches.

B. RESULT ANALYSIS

Figure 7 shows distribution of training data in pie chart ,and shows that each class has roughly equal number of instances,so our dataset is not skewed. Figure 8 and 9 shows the distribution of text length in dataset before and after removing stopwords. Stopwords are words that do not have much impact on the meaning of text . we see that the number of words have been reduced by removing stopwords.

Figure 10 shows wordcloud of all the words in our dataset. It shows the most frequent words in our dataset.

Figure 11,12,13 shows wordcloud for each class label. We can see that each class is characterized by few repeated words. Thus it is helpful for our naive bayes model to identify what sorts of feature fall into what class. Even if the features are assumed to be independent upon each other,we can classify text into corresponding classes by looking at individual words present.

Figure 14,15 shows confusion matrix for classification. Both models have comparatively less accuracy and f1 score on entertainment class . the matrix also shows that the entertainment class is being confused for mostly sports class .

Table 1 shows the comparision of accuracy between different naive bayes and vectorization approaches.we can see that count vectorization performed significantly better than tf-idf

TABLE 1. Classification report with Multinomial NB classifier and counter vectorization

Class	Precision	Recall	F1-score	Support
sports	0.86	0.88	0.87	527
entertainment	0.79	0.80	0.80	428
business	0.90	0.87	0.88	539
Accuracy			0.85	1494
Macro avg	0.85	0.85	0.85	1494
Weighted avg	0.85	0.85	0.85	1494

TABLE 2. Classification report with Multinomial NB classifier and tfidf vectorization

Class	Precision	Recall	F1-score	Support
sports	0.67	0.80	0.73	532
entertainment	0.71	0.48	0.57	448
business	0.74	0.80	0.77	514
Accuracy			0.70	1494
Macro avg	0.71	0.69	0.69	1494
Weighted avg	0.71	0.70	0.70	1494

vectorization. This maybe due to the fact that the texts are very short and also frequency of certain words matter very much to know the class of news. Also count vectorization duce-provides straightforward and interpretable representation and doesnt introduce extra complexity which may confuse the model.We also observe that the Multinomial naive bayes is better since our representation consists of frequency of words. The gaussian naive bayes performs poor as our representation is sparse and also doesnt contain continious attributes,The bernoulli naive bayes lies in between as it is equivalent to using bag-of –words approach which is not as good as counting the frequencies.

V. DISCUSSION

A. GAUSSIAN VS BERNOULLI VS MULTINOMIAL NAIVE BAYES

TABLE 3. Accuracy Comparison Chart

	Gaussian NB	Multinomial NB	Bernoulli NB
Count-vectorizer	79	85	84
TF-IDF vectorizer	54	70	64

Gaussian Naive Bayes is more suitable for dataset having continuous attribute values and where the distribution is Gaussian.Example weight,height of person etc.We fit Gaussian distribution to each feature and belonging to a particular class in which parameters(mean and variance) are determined by Maximum likelihood estimation.Then given a test instance, we calculate posterior probability of belonging to a particular class from all features . The class with highest probability is then assigned.

$$P(\mathbf{X}|C_k) = \prod_{i=1}^{n} p_{k_i}(x_i)\,(1 - p_{k_i})^{1-x_i} \tag{11}$$

Bernoulli Naive Bayes is suitable for data binary feature variables.For example if bag-of-words approach is used then it is easier to use Bernoulli distribution to calculate probabilities.

$$p(\text{word}_i|\text{Class}_k) = \frac{\sum_{\text{training set}} \text{count}_{ik}}{\sum_{\text{training set}} \text{count}_k} \tag{12}$$

Unlike the bag of words approach we may want to count occurrence of each word/feature in text. **Multinomial Naive Bayes** is more suitable for dataset having frequencies of discrete data. Which makes it suitable for text classification with count vectorization.

B. SMOOTHING

As we are dealing with textual data, there is a great chance that we encounter unseen features in test instances. Not only that, the number of features is also not fixed. If new types of features occur, we can simply ignore them and see only those features encountered in training. However, this is an incorrect approach as it is equivalent to assigning a probability of 1. If not, then the probability becomes zero. To solve this, smoothing is required.

The Gaussian Naive Bayes doesn't require smoothing because it assumes all features are continuous. When it encounters a new feature value, it calculates the probability from a Gaussian distribution.

On the other hand, Categorical, Multinomial, and Bernoulli Naive Bayes require Laplace smoothing. Laplace smoothing calculates the probability as:

$$P_{\text{lap},k}(x|y) = \frac{c(x,y)+k}{c(y)+k|X|} \tag{13}$$

Where: x is the unseen word, y is the class label, $c(x,y)$ is the count of class 'y' containing word 'x', $c(y)$ is the total count of class 'y', k is the smoothing factor, $|X|$ represents the number of features/dimensions in the data.

So, Laplace smoothing basically adds a new instance for each class containing the unseen feature value to the training data and calculates the probability for the required class.

C. POSSIBLE SOURCES OF ERRORS

- **Ambiguous or mislabeled data**: If the training dataset contains incorrectly labeled news articles or articles with ambiguous category assignments, the classifier might learn from these errors and make incorrect predictions during testing.
- **Vocabulary mismatch**: If the words present in the testing data are not present in the training data, the classifier may not know how to handle them, leading to misclassifications.
- **Out-of-vocabulary words**: New and previously unseen words (out-of-vocabulary words) in the testing data can cause issues if the model has not encountered them during training. These words will be ignored, leading to potential loss of information.
- **Dependencies between words**: Naive Bayes assumes that features (words) are conditionally independent, but in reality, words in a sentence often have dependencies. Ignoring these dependencies may lead to errors and performance loss.
- **Uninformative features**: Some words might be present in multiple categories and not contribute much to distinguishing between classes. These uninformative features can confuse the classifier and lead to misclassifications.

VI. CONCLUSION

The Naive Bayes algorithm implementation on Nepali news classification demonstrated promising results. Through careful data preprocessing, including tokenization, stop word removal, and vectorization, the algorithm effectively transformed the textual data into a suitable format for analysis. The construction of a comprehensive vocabulary from the training dataset allowed for representation of the news articles as feature vectors.

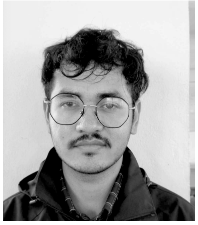
Despite the simplicity of the Naive Bayes approach and its assumption of conditional independence between features, the classifier performed reasonably well in distinguishing between different categories of Nepali news. The model showcased its ability to handle large-scale datasets, making it suitable for real-world news classification applications.

However, there were certain challenges encountered during the analysis. Ambiguous or mislabeled data in the training set posed a risk of model learning from errors, leading to inaccurate predictions during testing. Additionally, the presence of out-of-vocabulary words in the testing data raised concerns about potential information loss and misclassifications. Furthermore, the assumption of independence between words in the Naive Bayes algorithm might not hold true for Nepali news articles, which could have influenced the model's performance.

To improve the classifier's accuracy, future work could focus on refining the data preprocessing techniques, addressing the issue of vocabulary mismatch, and exploring advanced algorithms that can capture word dependencies more effectively. Also, regular updates to model with new data would ensure that the classifier remains relevant and accurate in an evolving news landscape.

REFERENCES

[1] Living with Uncertainty: How to Accept and Be More Comfortable with Unpredictability.// https://news.stanford.edu/report/2020/11/04/living-with-uncertainty/.
[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *"Deep Learning."* MIT Press.
[3] Han, J., Kamber, M., Pei, J. (2012). *Data Mining: Concepts and Techniques*, Third Edition. Morgan Kaufmann Publishers.
[4] Daniel Jurafsky & James H. Martin. *Speech and Language Processing*.
[5] M. Narasimha Murty, V. Susheela Devi. *Pattern Recognition: An Algorithmic Approach*.



PILOT KHADKA is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. During his time at Thapathali Campus, Pilot has actively engaged in various academic and extracurricular activities. He has participated in coding competitions, collaborated on software development projects, and demonstrated a keen interest in exploring new technologies.

KSHITIZ POUDEL is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. His relentless pursuit of knowledge and dedication to uplifting and empowering others make him an exceptional contributor and an invaluable asset to the academic community.

VII. APPENDIX

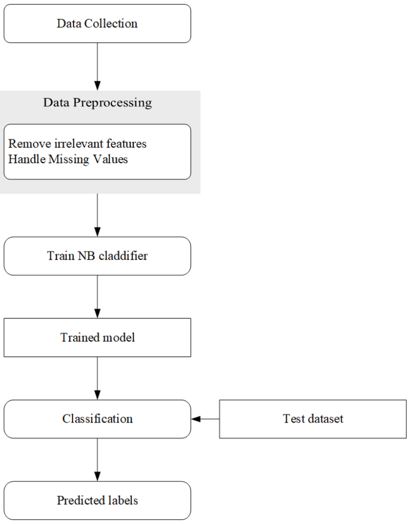


FIGURE 1. System Block Diagram

	headings	paras	label
0	कर्णालीका सुइना	उमेरले ७० कटेका पूर्णबहादुर विश्वकर्मा ७ वर्षी...	entertainment
1	साकुराजस्तो प्रेमकथा	जापान भन्नेबित्तिकै पैँयु साकुरा फूलको चर्चा ह...	entertainment
2	भद्रगोल नै ट्रेन्डिङमा	नेपाल टेलिभिजनबाट हरेक शुक्रबार साँझ प्रसारण ह...	entertainment
3	फेरिए लोकभाका चर्चामा सामाजिक गीत	दोहोरी भन्नासाथ सोचिन्छ, यो केटा–केटीबीचको जुह...	entertainment
4	भूकम्पले भत्केको सपना	च्यान्टे र मनमायाको स्थानीय परम्पराअनुसार विवा...	entertainment

FIGURE 2. Nepali Sentiment Dataset

उमेरले ७० कटेका पूर्णबहादुर विश्वकर्मा ७ वर्षीया बिरामी नातिनीको उपचार गर्न घरबाट हिँड्छन्। बाटोमा झमक्क साँझ पर्छ। दायाँबायाँछिटपुट घर छन् तर दलित भएकै कारण बास पाउँदैनन्। बल्लतल्ल गोठमा ओत लाग्न पाए पनि खानेकुरा दिइँदैन। राति बिरामी नातिनी'भोक लाग्यो' भन्दै मुख बाउँछिन्।

FIGURE 3. Original Sentence

उमेरले ७० कटेका पूर्णबहादुर विश्वकर्मा ७ वर्षीया बिरामी नातिनीको उपचार गर्न घरबाट हिँड्छन् बाटोमा झमक्क साँझपर्छ दायाँबायाँ छिटपुट घर छन् तर दलित भएकै कारण बास पाउँदैनन् बल्लतल्ल गोठमा ओत लाग्न पाए पनि खानेकुरादिइँदैन राति बिरामी नातिनी भोक लाग्यो भन्दै मुख बाउँछिन्

FIGURE 4. After removing punctuation

'छ','र', 'पनि', 'छन्', 'लागि', 'भएको', 'गरेको'

FIGURE 5. Sample stopwords

उमेरले ७० कटेका पूर्णबहादुर विश्वकर्मा ७ वर्षीया बिरामी नातिनीको उपचार गर्न घरबाट हिँड्छन् बाटोमा झमक्क साँझपर्छ दायाँबायाँ छिटपुट घर छन् तर दलित भएकै कारण बास पाउँदैनन् बल्लतल्ल गोठमा ओत लाग्न पाए पनि खानेकुरादिइँदैन राति बिरामी नातिनी भोक लाग्यो भन्दै मुख बाउँछिन्

FIGURE 6. After removing stopwords

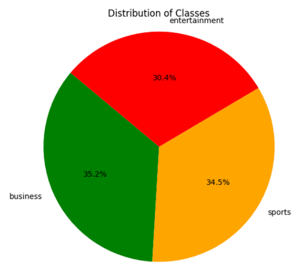


FIGURE 7. Distribution of classes

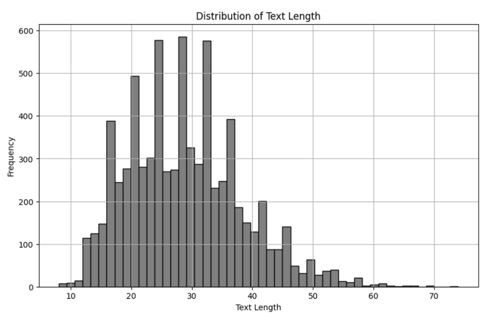


FIGURE 8. Text length histogram

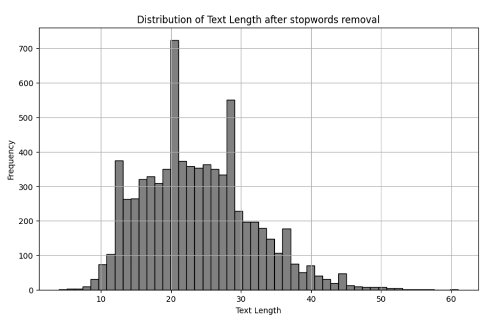


FIGURE 9. Text length histogram after stopwords removal

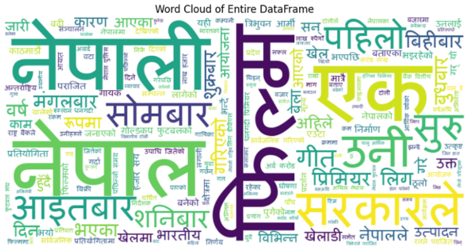


FIGURE 10. Wordcloud of dataset

FIGURE 11. Wordcloud for entertainment class

FIGURE 12. Wordcloud for business class

FIGURE 13. Wordcloud for Sports class

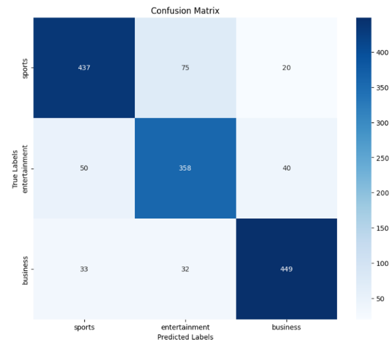


FIGURE 14. confusion matrix with Multinomial NB classifier and countvectorization

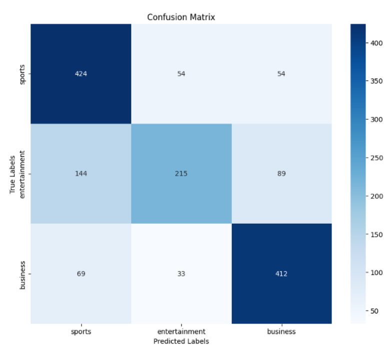


FIGURE 15. confusion matrix with Multinomial NB classifier and tfidf vectorization

VIII. CODE

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

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

dataset_path = '/content/drive/MyDrive/
    Colab_Notebooks/Nepali_News_Classification.csv
    '

# Load the dataset into a pandas DataFrame
df = pd.read_csv(dataset_path)

df.head(10)

#check for null values
df.isnull()

df.info()

"""##Visualizing the dataset"""

df.shape

class_counts = df['label'].value_counts()
print(class_counts)

#remove title column
df = df.drop(df.columns[0], axis=1)

df.head()

df.sample(frac=1).reset_index(drop=True)

class_counts = df['label'].value_counts()
print(class_counts)
# Create a pie chart
plt.figure(figsize=(6, 6))
plt.pie(class_counts, labels=class_counts.index,
    autopct='%1.1f%%', startangle=140, colors=['
    green', 'orange', 'red'])
plt.axis('equal')

# Add a title
plt.title('Distribution of Classes')

# Display the pie chart
plt.show()

df['paras'][0]

len(df['paras'][0])

#punctuation removal
import re
# Define a function to remove unnecessary
    punctuations from Nepali text
def remove_punctuation(text):
    # Define the regular expression pattern to
    remove punctuations
    punctuation_pattern = r'[   ?,:;\'",.()\n&
                    !-]'  # Add other punctuations
    as needed

    # Use regular expression to remove
    punctuations
    return re.sub(punctuation_pattern, '', text)

# Apply the function to remove punctuations from
    the 'Sentences' column
df['paras'] = df['paras'].apply(remove_punctuation
    )

print(df['paras'][0])

len(df['paras'][0])

#define a proper function to calculate words in a
    nepali text.
def get_length(text):
    # Split the text into words based on spaces
    words = text.split()

    # Return the number of words in the text
    return len(words)

get_length(df['paras'][0])

# Apply the function to calculate the text length
    for each instance
text_length = df['paras'].apply(get_length)

# Plot the text length distribution
plt.figure(figsize=(10, 6))
plt.hist(text_length, bins=50, color='gray',
    edgecolor='black')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.title('Distribution of Text Length')
plt.grid(True)
plt.show()

#stopwords removal

# Function to remove Nepali stopwords from a
    sentence
def remove_stopwords(sentence):
    words = sentence.split()
    filtered_sentence = [word for word in words if
     word.lower() not in nepali_stopwords]
    return ' '.join(filtered_sentence)

# Create a new DataFrame with processed 'Sentences
    ' (Nepali stopwords removed)
new_df = df.copy()
new_df['paras'] = new_df['paras'].apply(
    remove_stopwords)

new_df.head()

new_df['paras'][0]

# Apply the function to calculate the text length
    for each instance
text_length = new_df['paras'].apply(get_length)

# Plot the text length distribution
plt.figure(figsize=(10, 6))
plt.hist(text_length, bins=50, color='gray',
    edgecolor='black')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.title('Distribution of Text Length after
    stopwords removal')
plt.grid(True)
plt.show()

print(np.max(np.array(text_length)),',',np.min(np.
    array(text_length)))

"""##Wordcloud"""

font_path="/content/drive/MyDrive/Colab_Notebooks/
    Mangal400.TTF"

```

```
from wordcloud import WordCloud
# Combine all the text data into a single string
text_data = ' '.join(new_df['paras'])
# Create a WordCloud object with the custom font
wordcloud = WordCloud(width=800, height=400, background_color='white', font_path=font_path, regexp=r"[\u0900-\u097F]+").generate(text_data)
# Display the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Entire DataFrame')
plt.show()

#word cloud of preprocessed data
sentiments = new_df['label'].unique()

# Function to create and display a WordCloud for a specific sentiment category
def generate_wordcloud_for_sentiment(sentiment):
    # Filter the DataFrame for the current sentiment
    filtered_df = new_df[new_df['label'] == sentiment]

    # Combine all the text data for the current sentiment into a single string
    textual_data = ' '.join(filtered_df['paras'])

    # Create a WordCloud object with the custom font
    wordcloud = WordCloud(width=800, height=400, background_color='white', font_path=font_path, regexp=r"[\u0900-\u097F]+").generate(textual_data)

    # Display the WordCloud for the current sentiment
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f'Word Cloud for Class: {sentiment}')
    plt.show()

# Generate WordClouds for each sentiment category
for sentiment in sentiments:
    generate_wordcloud_for_sentiment(sentiment)

"""##Vectorizing the textual data through count-vectorizer"""

new_df=new_df.sample(frac=1).reset_index(drop=True)

new_df.head()

from sklearn.feature_extraction.text import CountVectorizer
# Create an instance of TfidfVectorizer
count_vectorizer= CountVectorizer(max_features=300)#most repeated 300 words

# Fit and transform the text data using the vectorizer
X1=count_vectorizer.fit_transform(new_df['paras']).toarray()

print(X1[0],X1[0].shape)#6238 unique words

# Convert the NumPy ndarray to a pandas DataFrame
X1_df = pd.DataFrame(X1)
X1_df.head()

y=new_df['label']

print(y)

from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1_df, y, test_size=0.2, random_state=42)

X_train1.shape

y_train1.shape

from sklearn.naive_bayes import GaussianNB, MultinomialNB,BernoulliNB
modelGNB=GaussianNB()
modelMNB=MultinomialNB()
modelBNB=BernoulliNB()

modelGNB.fit(X_train1,y_train1)
modelMNB.fit(X_train1,y_train1)
modelBNB.fit(X_train1,y_train1)

y_pred11=modelGNB.predict(X_test1)
y_pred12=modelMNB.predict(X_test1)
y_pred13=modelBNB.predict(X_test1)

y_pred11

from sklearn.metrics import confusion_matrix, classification_report
def get_report_and_cm(name):
  cm = confusion_matrix(y_test1, name)

  # Create a classification report
  class_names = ['sports', 'entertainment','business']
  cr = classification_report(y_test1, name, target_names=class_names)

  # Print the confusion matrix and classification report
  print("Confusion Matrix:")
  print(cm)

  print("\nClassification Report:")
  print(cr)

get_report_and_cm(y_pred11)
get_report_and_cm(y_pred12)
get_report_and_cm(y_pred13)

class_names = ['sports', 'entertainment','business']

import seaborn as sns
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_matrix(y_test1, y_pred12), annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

"""##Vectorizing textual data through TF_IDF"""

new_df.head()

from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import nltk


nltk.download('punkt')
nltk.download('stopwords')

all_text = new_df['paras'].tolist()
# Join all the text elements into a single string
combined_text = ' '.join(all_text)

# Tokenize the combined text into words
words = word_tokenize(combined_text)

#remove stopwords
stop_words = set(stopwords.words('nepali'))
words = [word for word in words if word not in stop_words]

# Create the vocabulary (set of unique words)
vocabulary = set(words)

print(vocabulary)

print(len(vocabulary))

def tokenize_text(text):
    words = word_tokenize(text)
    words = [word for word in words if word not in stop_words]
    return words

# Apply tokenization to the 'text' column in the DataFrame
new_df['tokenized_text'] = new_df['paras'].apply(tokenize_text)

new_df.head()

#finally apply tf-idf vectorization
# Convert tokenized_text back to strings from lists of words
new_df['tokenized_text'] = new_df['tokenized_text'].apply(lambda tokens: ' '.join(tokens))

# Initialize the TF-IDF vectorizer with your vocabulary
tfidf_vectorizer = TfidfVectorizer(vocabulary=vocabulary, max_features=500)
# Fit and transform the tokenized_text to get the TF-IDF vectors
tfidf_vectors = tfidf_vectorizer.fit_transform(new_df['tokenized_text'])

tfidf_df = pd.DataFrame(tfidf_vectors.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
# Concatenate the original DataFrame with the TF-IDF DataFrame
new_df = pd.concat([new_df, tfidf_df], axis=1)

print()

new_df.head()

np.max(np.array(tfidf_df.head(1)))

X_train2, X_test2, y_train2, y_test2 = train_test_split(tfidf_df, y, test_size=0.2, random_state=42)

modelGNB.fit(X_train2,y_train2)
modelMNB.fit(X_train2,y_train2)
modelBNB.fit(X_train2,y_train2)

y_pred21=modelGNB.predict(X_test2)
y_pred22=modelMNB.predict(X_test2)
y_pred23=modelBNB.predict(X_test2)

get_report_and_cm(y_pred21)
get_report_and_cm(y_pred22)
get_report_and_cm(y_pred23)

import seaborn as sns
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_matrix(y_test2, y_pred22), annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```