Multinomial Naive Bayes Text Classification Model

GKS4008
20221009 GKS EunhongKim
https://github.com/eunhong0925/gks-4008/tree/main

Abstract

This report delves into the comprehensive development and implementation of a 'Multinomial Naive Bayes Text Classification Model.' Specifically tailored for categorizing news text data across 20 distinct categories, the model is rooted in the robust mathematical framework of Bayesian theory. Bayesian theory, which calculates the probability of an event based on prior knowledge, forms the foundation for the subsequent application of the Multinomial Naive Bayes algorithm. This algorithm is strategically employed to ensure efficient and accurate text classification, highlighting the synergy between Bayesian principles and machine learning techniques. The operational process unfolds through essential phases, encompassing prediction and learning. This involves creating a dictionary to assess category frequencies, computing the log probability mass function (pmf), and ultimately predicting categories. The strategic incorporation of Laplace smoothing ensures algorithmic stability. Following this step, the model achieves an accuracy of approximately 76%. The report utilizes visualization tools, such as confusion matrices and histograms, for a detailed interpretation of the classification process. The discussion highlights the inherent simplicity and efficiency of Naive Bayes classifiers, particularly in the nuanced domain of text data categorization. Key advantages, such as speed, interpretability, and minimal tunable parameters, further underscore the efficacy of these classifiers in this context.

01. Thoery

Machine learning is an algorithm that enhances judgment based on learning patterns and inferences, reducing prediction error rates. As part of machine learning, a multinomial naive Bayes classifier was constructed for text classification tasks. Text classification is a natural language processing (NLP) task that involves assigning predefined categories or labels to text documents. For this implementation, the Multinomial Naive Bayes classifier is constructed, which is a probabilistic algorithm based on Bayes' theorem, which assumes that the features are conditionally independent. One research reveals that, when classifying user opinions in English, the Multinomial Naive Bayes classifier consistently outperforms the Bernoulli Naive Bayes classifier in terms of accuracy, and this performance gap widens as the volume of data increases. (Kalcheva 2023: 31) Therefore, a Multinomial model for text classification was constructed, and the Laplace smoothing technique was employed to stabilize the solution.

(1) Bayes' Theorem

Bayes' theorem calculates the posterior probability of an event. The objective is to identify the probability class to which object x belongs by selecting the class with the highest probability p(y = c | x) from all available classes.

$$P(A|B) = \frac{P(B\,|A)P(A)}{P(B)}$$

P(A): Prior Probability. The probability of event A.
P(B|A): Likelihood Probability. The probability of event B occurring given the assumption that cause A has occurred.
P(A|B): Posterior Probability. The probability of cause A occurring given the assumption that event B has occurred.

(2) Navie Bayes model

Naive Bayes models are a set of extremely fast and simple classification algorithms, often suitable for very high-dimensional datasets. This algorithm is based on Bayesian theory and calculates the posterior probability under the assumption that events are mutually exclusive and independent. Because they are so fast and have so few tunable parameters, they end up being very useful as a quick-and-dirty baseline for a classification problem. This text classification systems also lies the Naive Bayes algorithm, a probabilistic model rooted in Bayes' theorem. The "naive" in Naive Bayes arises from the assumption of conditional independence among features, in this case, the words in a document. Despite this simplification, Naive Bayes has proven remarkably effective for text classification tasks, demonstrating its resilience and efficiency in the face of large and diverse datasets.

(3) Laplace Smoothing

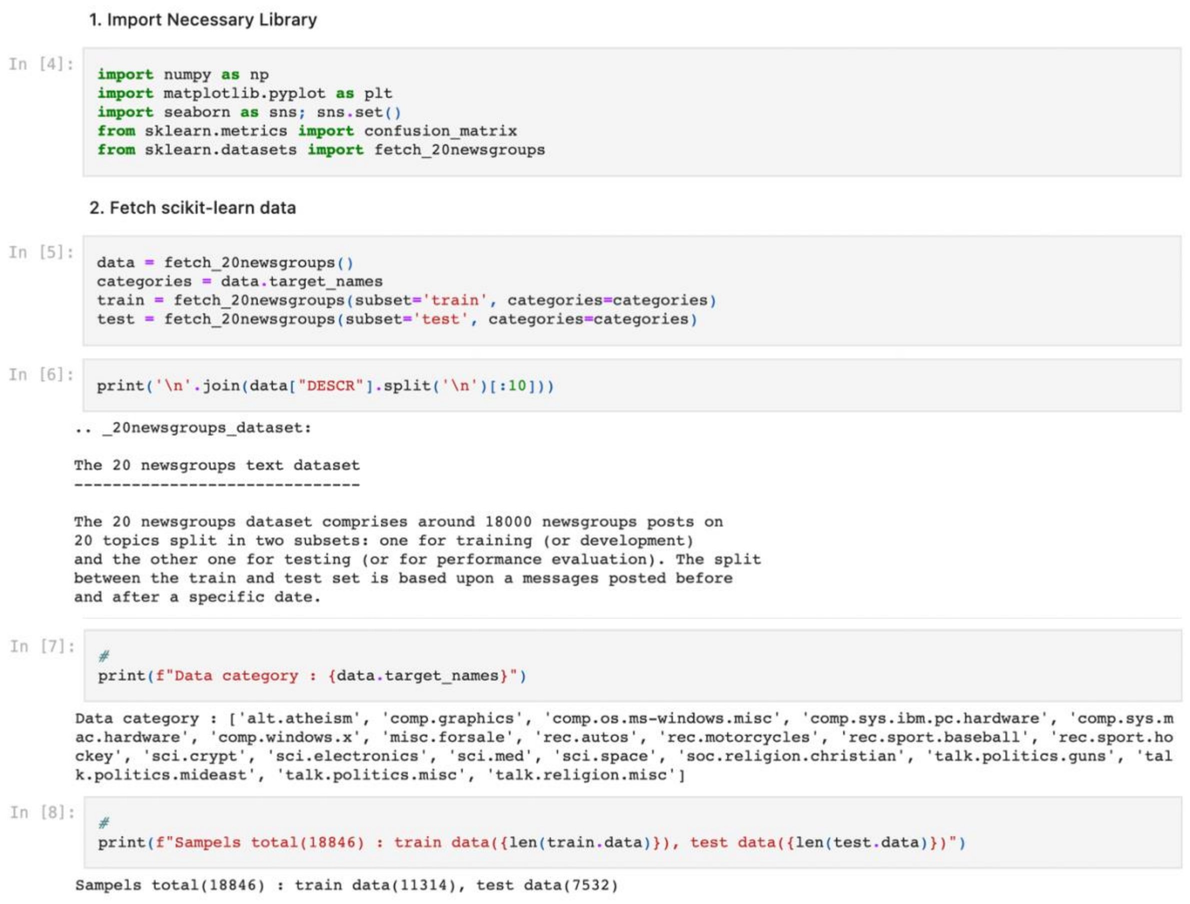
Laplace smoothing is a technique applied to the MultinomialNB. Its primary purpose is to ensure stable algorithm operation by addressing the issue of probability calculations becoming 0. This is achieved by adding 1 to the frequency of word occurrences to prevent cases where the frequency becomes 0.

(4) Application

Specifically, for this task, the data consists of news categorized into 20 classes provided by scikit-learn. As this task involves text data classification, a Naive Bayes model based on multinomial distribution has been developed. Prior to the prediction task, all text data was loaded into a python set-based vocabulary, creating a set of words that serve as the basis for classification without duplicates. Subsequently, this vocabulary was transformed into an easy-to-handle python list format for further processing. During the prediction process, the test data is used to ultimately generate 20 probability mass functions (PMF), and during the learning process, the classification task is carried out using all the available data to return the predicted category and the true category. The model's accuracy is then determined by comparing these two outcomes.To enhance the accuracy and stability of this process, Laplace smoothing technique is employed. Additionally, the PMF is designed to return log probabilities, utilizing the logarithm function from the NumPy library. Furthermore, the code is structured in a dictionary format to ensure that the data is organized in a visually appealing manner.

02. Implementation

(1) Data Fetching



1. Import Necessary Library

```
In [4]:
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
from sklearn.datasets import fetch_20newsgroups
```

2. Fetch scikit-learn data

```
In [5]:
data = fetch_20newsgroups()
categories = data.target_names
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)
```

```
In [6]:
print('\n'.join(data["DESCR"].split('\n')[:10]))
```

```
.. _20newsgroups_dataset:

The 20 newsgroups text dataset
------------------------------

The 20 newsgroups dataset comprises around 18000 newsgroups posts on
20 topics split in two subsets: one for training (or development)
and the other one for testing (or for performance evaluation). The split
between the train and test set is based upon a messages posted before
and after a specific date.
```

```
In [7]:
#
print(f"Data category : {data.target_names}")
```

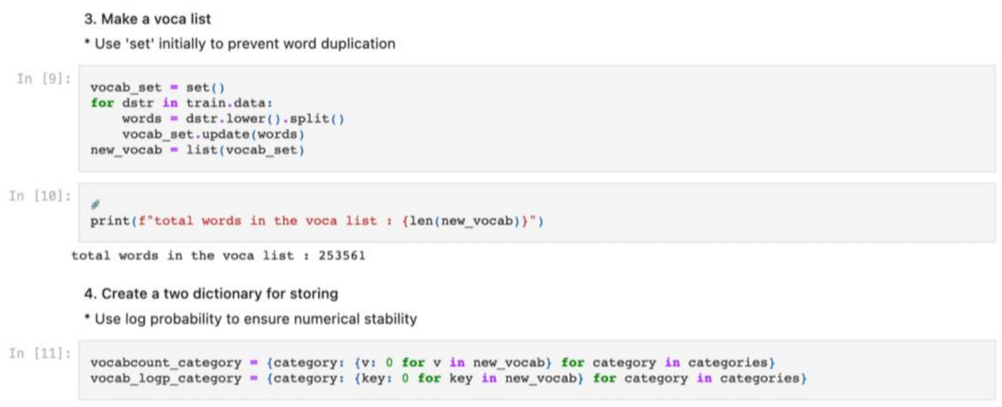
```
Data category : ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.m
ac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.ho
ckey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'tal
k.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
```

```
In [8]:
#
print(f"Sampels total(18846) : train data({len(train.data)}), test data({len(test.data)})")
```

```
Sampels total(18846) : train data(11314), test data(7532)
```

The provided code utilizes the `fetch_20newsgroups` function from the scikit-learn library to load the 20 newsgroups dataset. With a total of 20 classes, 18,846 samples, and a dimensionality of 1, the dataset serves as a valuable resource for text classification tasks. The training set encompasses 11,314 samples, and the testing set is composed of 7,532 samples. This entire process serves as a validation step for text-based analysis and modeling.

(2) Vocabulary Creation:


3. Make a voca list

* Use 'set' initially to prevent word duplication

```
In [9]:
vocab_set = set()
for dstr in train.data:
    words = dstr.lower().split()
    vocab_set.update(words)
new_vocab = list(vocab_set)
```

```
In [10]:
#
print(f"total words in the voca list : {len(new_vocab)}")
```

```
total words in the voca list : 253561
```

4. Create a two dictionary for storing

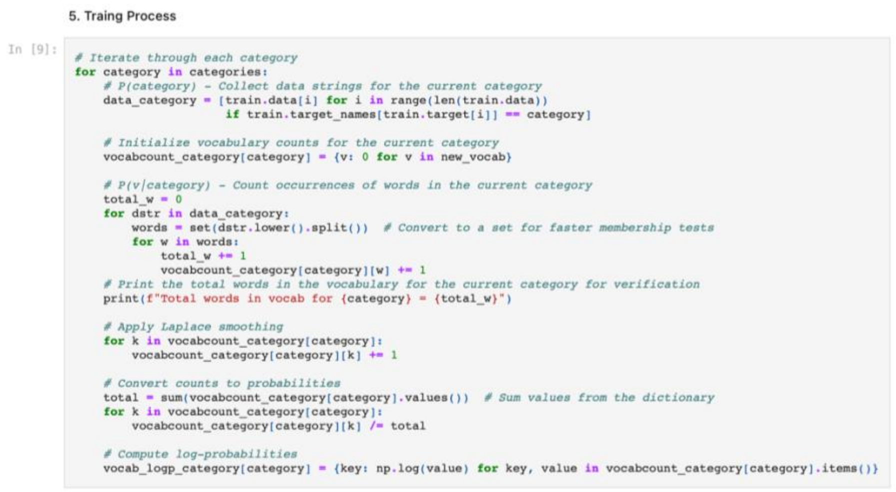
* Use log probability to ensure numerical stability

```
In [11]:
vocabcount_category = {category: {v: 0 for v in new_vocab} for category in categories}
vocab_logp_category = {category: {key: 0 for key in new_vocab} for category in categories}
```

In the provided code, a vocabulary list is created to capture unique words from the training data while preventing duplication. Initially, an empty set named *‘vocab_set’* is used to collect unique words. For each document (`dstr`) in the training data, the text is converted to lowercase and split into words. These words are then added to the ‘*vocab_set*’. After processing all documents, the set is converted back to a list named *‘new_vocab’*. The total number of unique words in the vocabulary list is then printed.

Following this, two dictionaries are initialized to manage information pertaining to word counts and log probabilities. The 'vocabcount_category' dictionary is created to track the occurrences of each word in every category, while the 'vocab_logp_category' dictionary is similarly initialized, with each category containing a nested dictionary for words and their corresponding log probabilities. The utilization of log probabilities ensures numerical stability, especially when dealing with small probabilities. These two resulting dictionaries offer a structured representation of word counts and log probabilities for each category in the dataset.

(3) Model Training:


5. Traing Process

```
In [9]:
# Iterate through each category
for category in categories:
    # P(category) - Collect data strings for the current category
    data_category = [train.data[i] for i in range(len(train.data))
                     if train.target_names[train.target[i]] == category]

    # Initialize vocabulary counts for the current category
    vocabcount_category[category] = {v: 0 for v in new_vocab}

    # P(v|category) - Count occurrences of words in the current category
    total_w = 0
    for dstr in data_category:
        words = set(dstr.lower().split())  # Convert to a set for faster membership tests
        for w in words:
            total_w += 1
            vocabcount_category[category][w] += 1
    # Print the total words in the vocabulary for the current category for verification
    print(f"Total words in vocab for {category} = {total_w}")

    # Apply Laplace smoothing
    for k in vocabcount_category[category]:
        vocabcount_category[category][k] += 1

    # Convert counts to probabilities
    total = sum(vocabcount_category[category].values())  # Sum values from the dictionary
    for k in vocabcount_category[category]:
        vocabcount_category[category][k] /= total

    # Compute log-probabilities
    vocab_logp_category[category] = {key: np.log(value) for key, value in vocabcount_category[category].items()}
```

The provided code snippet corresponds to the training process of a text classification model using a Naive Bayes approach. Here is a description of the key components:

1. Collect Data for Current Category: For the current category, data strings are collected from the training set that belongs to that category.

2. Initialize Vocabulary Counts: Vocabulary counts for the current category are initialized as a dictionary, with each unique word in the vocabulary (`new_vocab`) having an initial count

of zero.

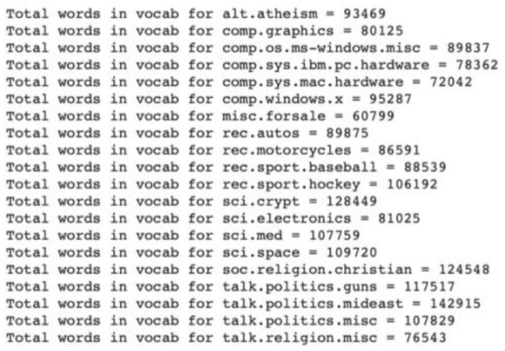
3. Count Occurrences of Words: The code then iterates through each document in the data for the current category, counting the occurrences of each word. The words are converted to lowercase.

4. Laplace Smoothing: Laplace smoothing is applied to handle cases where certain words may not appear in the training data for a specific category. This involves incrementing the count of each word by one.

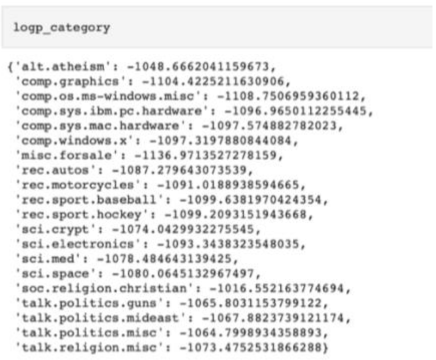
5. Convert Counts to Probabilities: The counts are then converted to probabilities by dividing each count by the total number of words in the vocabulary for the current category.

6. Compute Log-Probabilities: Log-probabilities are computed for each word in the vocabulary for the current category. This step is crucial for numerical stability, especially when dealing with small probabilities.

The overall process is repeated for each category, resulting in dictionaries (`vocabcount_category` and `vocab_logp_category`) that store word counts and log-probabilities for each category in the dataset. These dictionaries serve as the foundation for making predictions during the subsequent prediction process.



```
Total words in vocab for alt.atheism = 93469
Total words in vocab for comp.graphics = 80125
Total words in vocab for comp.os.ms-windows.misc = 89837
Total words in vocab for comp.sys.ibm.pc.hardware = 78362
Total words in vocab for comp.sys.mac.hardware = 72042
Total words in vocab for comp.windows.x = 95287
Total words in vocab for misc.forsale = 60799
Total words in vocab for rec.autos = 89875
Total words in vocab for rec.motorcycles = 86591
Total words in vocab for rec.sport.baseball = 88539
Total words in vocab for rec.sport.hockey = 106192
Total words in vocab for sci.crypt = 128449
Total words in vocab for sci.electronics = 81025
Total words in vocab for sci.med = 107759
Total words in vocab for sci.space = 109720
Total words in vocab for soc.religion.christian = 124548
Total words in vocab for talk.politics.guns = 117517
Total words in vocab for talk.politics.mideast = 142915
Total words in vocab for talk.politics.misc = 107829
Total words in vocab for talk.religion.misc = 76543
```

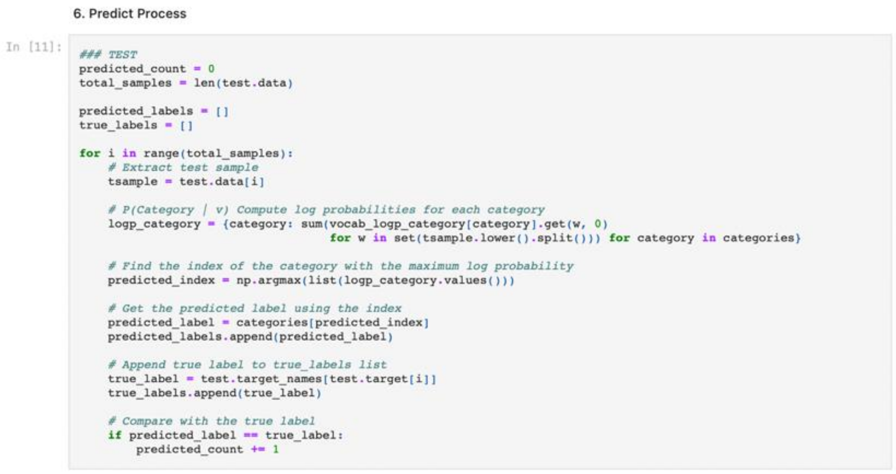


```
logp_category
```

```
{'alt.atheism': -1048.6662041159673,
 'comp.graphics': -1104.4225211630906,
 'comp.os.ms-windows.misc': -1108.7506959360112,
 'comp.sys.ibm.pc.hardware': -1096.9650112255445,
 'comp.sys.mac.hardware': -1097.574882782023,
 'comp.windows.x': -1097.3197880844084,
 'misc.forsale': -1136.9713527278159,
 'rec.autos': -1087.279643073539,
 'rec.motorcycles': -1091.0188938594665,
 'rec.sport.baseball': -1099.6381970424354,
 'rec.sport.hockey': -1099.209315194366 8,
 'sci.crypt': -1074.0429932275545,
 'sci.electronics': -1093.3438323548035,
 'sci.med': -1078.484643139425,
 'sci.space': -1080.0645132967497,
 'soc.religion.christian': -1016.552163774694,
 'talk.politics.guns': -1065.8031153799122,
 'talk.politics.mideast': -1067.8823739121174,
 'talk.politics.misc': -1064.7998934358893,
 'talk.religion.misc': -1073.4752531866288}
```

* Verifying the occurrence of each word and log probability in the process

(4) Model Testing:



6. Predict Process

```
In [11]:
### TEST
predicted_count = 0
total_samples = len(test.data)

predicted_labels = []
true_labels = []

for i in range(total_samples):
    # Extract test sample
    tsample = test.data[i]

    # P(Category | v) Compute log probabilities for each category
    logp_category = {category: sum(vocab_logp_category[category].get(w, 0)
                                   for w in set(tsample.lower().split())) for category in categories}

    # Find the index of the category with the maximum log probability
    predicted_index = np.argmax(list(logp_category.values()))

    # Get the predicted label using the index
    predicted_label = categories[predicted_index]
    predicted_labels.append(predicted_label)

    # Append true label to true_labels list
    true_label = test.target_names[test.target[i]]
    true_labels.append(true_label)

    # Compare with the true label
    if predicted_label == true_label:
        predicted_count += 1
```

The provided code snippet corresponds to the prediction process. It begins with the initialization of variables, including `predicted_count` to keep track of correctly predicted samples, and lists (`predicted_labels` and `true_labels`) to store predicted and true labels, respectively.

1. Extract Test Sample : The text of the current test sample (`tsample`) is extracted.

2. Compute Log Probabilities: Log probabilities for each category are computed based on the words present in the test sample. This is done by summing up the log probabilities of individual words in the sample using the precomputed `vocab_logp_category` dictionary.

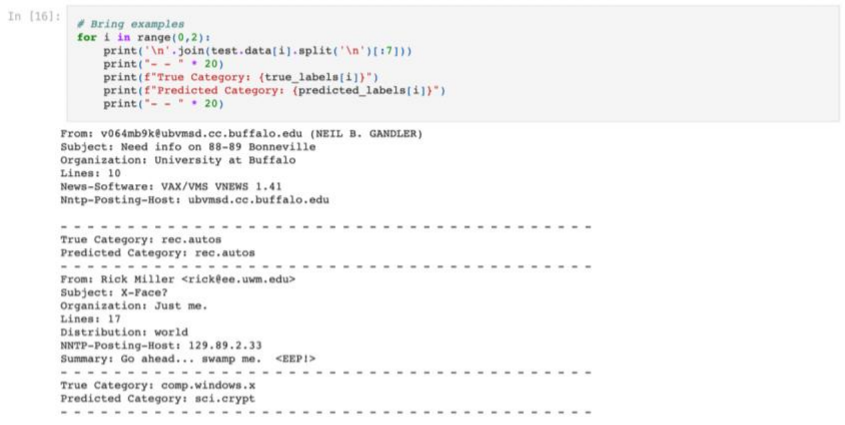
3. Determine Predicted Label: The index of the category with the maximum log probability is identified using NumPy's `argmax` function.

4. Get Predicted Label: The predicted label is obtained by mapping the index to the corresponding category from the `categories` list.

5. Record True Label: The true label for the current test sample is extracted from the `test.target_names` list and appended to the `true_labels` list.

6. Comparison and Accuracy Count: The predicted label is compared with the true label. If they match, the `predicted_count` variable is incremented, indicating a correct prediction.

The entire process iterates through all test samples, accumulating predictions and true labels. The accuracy of the model is calculated by comparing predicted labels with true labels, and the result is stored in the `predicted_count` variable. This section of the code is crucial for evaluating the model's performance on unseen data and assessing its accuracy in predicting the categories of test samples.



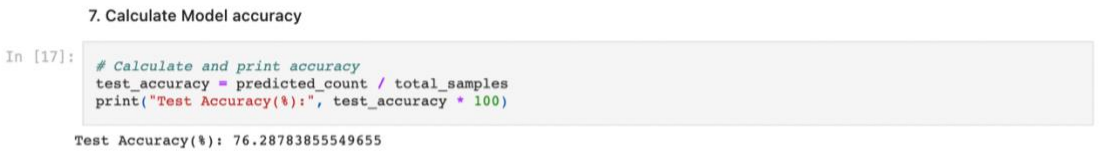
```
In [16]: # Bring examples
         for i in range(0,2):
             print('\n'.join(test.data[i].split('\n')[:7]))
             print("- - " * 20)
             print(f"True Category: {true_labels[i]}")
             print(f"Predicted Category: {predicted_labels[i]}")
             print("- - " * 20)
```

```
From: v064mb9k@ubvmsd.cc.buffalo.edu (NEIL B. GANDLER)
Subject: Need info on 88-89 Bonneville
Organization: University at Buffalo
Lines: 10
News-Software: VAX/VMS VNEWS 1.41
Nntp-Posting-Host: ubvmsd.cc.buffalo.edu

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
True Category: rec.autos
Predicted Category: rec.autos
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
From: Rick Miller <rick@ee.uwm.edu>
Subject: X-Face?
Organization: Just me.
Lines: 17
Distribution: world
NNTP-Posting-Host: 129.89.2.33
Summary: Go ahead... swamp me.  <EEP!>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
True Category: comp.windows.x
Predicted Category: sci.crypt
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The code snippet goes through the first two samples in the test data, presenting a short excerpt from each message and indicating both the true and predicted categories. The output offers a succinct comparison between the model's predictions and the actual categories for the selected samples. The first example illustrates a match between the true and predicted category, which is "rec.autos." However, the second example demonstrates a discrepancy, where the true category is "comp.windows.x," but the model predicted either "comp.windows.x" or "sci.crypt." This process exemplifies a quick assessment of the model's performance on individual instances.

(5) Calculate Model Accuracy:
The accuracy is a measure of the proportion of correctly predicted labels among all test samples. The measured model accuracy was about 76.28%.



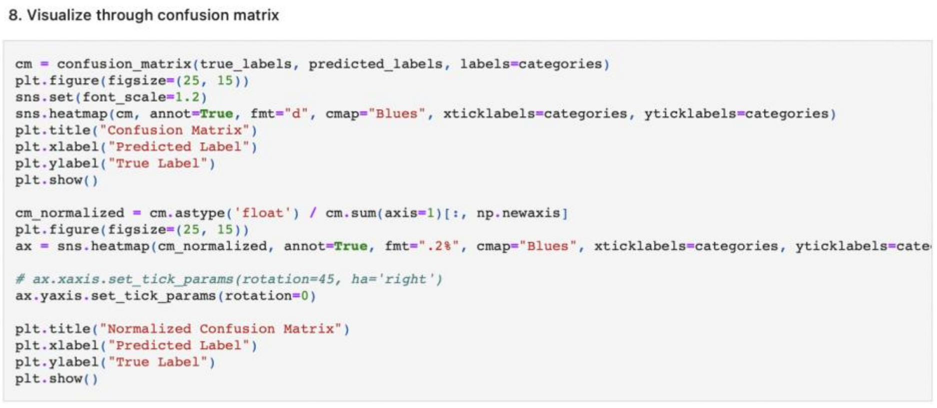
7. Calculate Model accuracy

```
In [17]:  # Calculate and print accuracy
          test_accuracy = predicted_count / total_samples
          print("Test Accuracy(%):", test_accuracy * 100)

Test Accuracy(%): 76.28783855549655
```

03. Experiments and Evaluation

(1) Confusion Matrix Visualization: Two confusion matrix was generated to provide a detailed breakdown of correct and incorrect predictions across different categories. The confusion matrix was visualized as a heatmap using seaborn. The second matrix provides the accuracy ratios for the conducted process, aiding in an intuitive understanding.



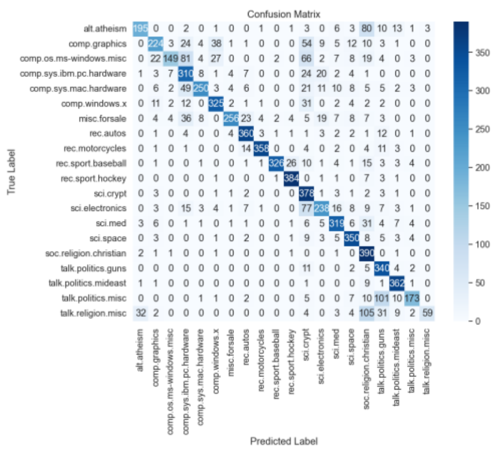
8. Visualize through confusion matrix

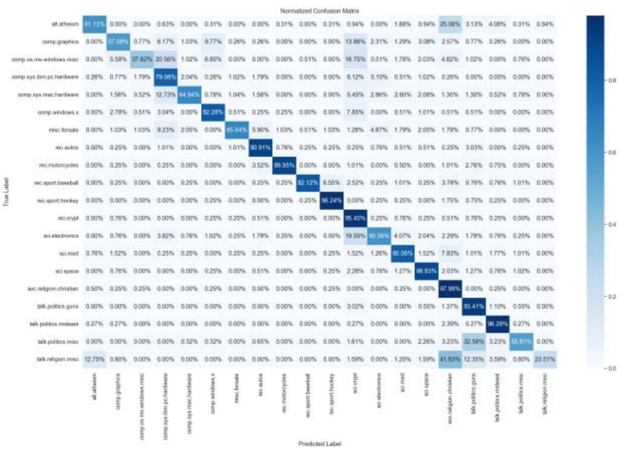
```
cm = confusion_matrix(true_labels, predicted_labels, labels=categories)
plt.figure(figsize=(25, 15))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=categories, yticklabels=categories)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
plt.figure(figsize=(25, 15))
ax = sns.heatmap(cm_normalized, annot=True, fmt=".2%", cmap="Blues", xticklabels=categories, yticklabels=cate

# ax.xaxis.set_tick_params(rotation=45, ha='right')
ax.yaxis.set_tick_params(rotation=0)

plt.title("Normalized Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

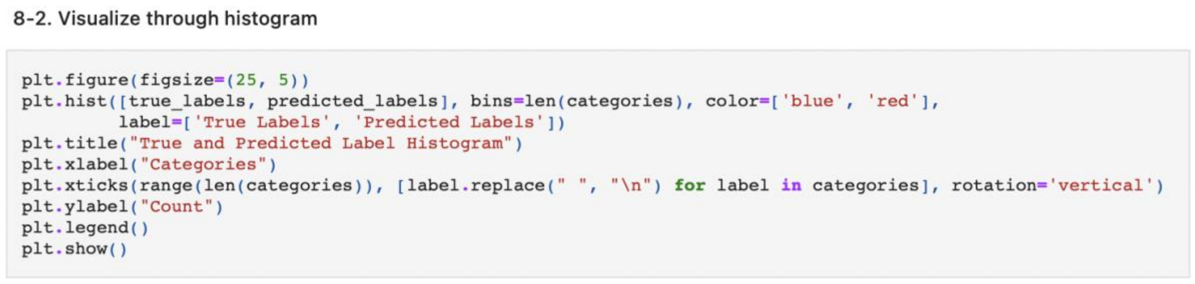




These plots aid in assessing how well the model is classifying instances across different categories. The first matrix illustrates the raw counts of true positive predictions, false positives, false negatives, and true negatives for each category. The second matrix, normalized version, presents the confusion matrix with values expressed as percentages. This allows for a more intuitive understanding of the model's performance on a per-category basis, as it accounts for varying class sizes.

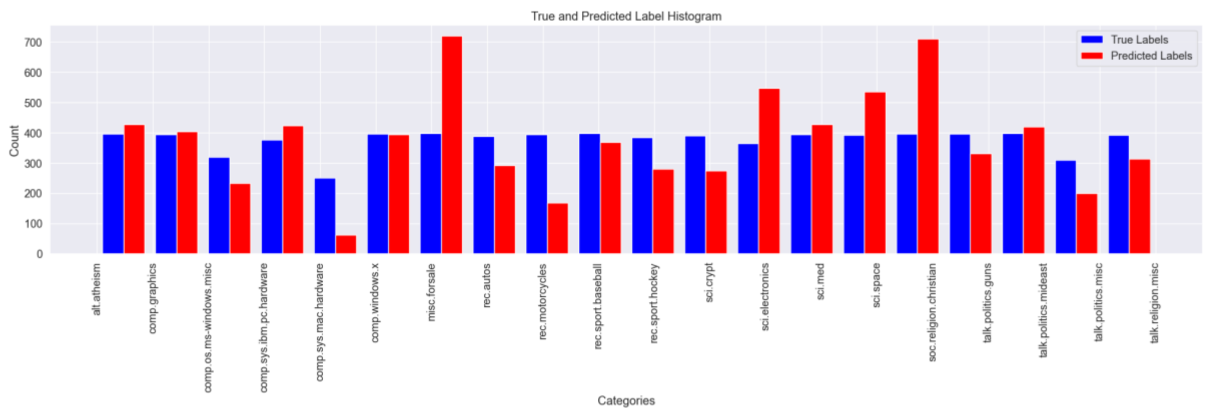
Specifically, the category with the highest accuracy is soc.religion.christian, showing 97.99%, followed by talk.politics.mideast with 96.28%. The two categories with the lowest accuracy are talk.religion.misc at 23.51% and comp.os.ms-windows.misc at 37.82%. While most categories exhibit accuracy rates of at least 50%, these two categories show comparatively lower figures. For talk.religion.misc, around 40% of predictions were classified as soc.religion.christian, and for comp.os.ms-windows.misc, approximately 20% were predicted as comp.sys.ibm.pc.hardware. This suggests that the initially assumed independence among the 20 news categories may not hold true, or errors may have occurred in the manual categorization process, indicating a potential need for further investigation.

(2) Histogram Visualization



8-2. Visualize through histogram

```
plt.figure(figsize=(25, 5))
plt.hist([true_labels, predicted_labels], bins=len(categories), color=['blue', 'red'],
         label=['True Labels', 'Predicted Labels'])
plt.title("True and Predicted Label Histogram")
plt.xlabel("Categories")
plt.xticks(range(len(categories)), [label.replace(" ", "\n") for label in categories], rotation='vertical')
plt.ylabel("Count")
plt.legend()
plt.show()
```



This visualization is valuable for assessing how well the model's predictions align with the actual labels across different categories. It provides a quick overview of the distribution of predictions and actual labels, highlighting potential patterns or discrepancies in the model's performance.

A notable observation is that soc.religion.christian, which exhibited the highest accuracy, showed the greatest disparity between true and predicted labels. While a higher number of predicted labels may contribute to a seemingly higher accuracy score, it is not necessarily indicative of more accurate predictions. For instance, misc.forsale, despite having more predicted labels, does not demonstrate a correspondingly high accuracy rate at 65%, emphasizing the importance of not generalizing based solely on the number of predictions. Moreover, the discovery highlights that achieving the highest accuracy does not necessarily mean the model performed the most accurately. Additionally, the matrix reveals two categories with low values, showing a similar pattern where true labels are slightly higher than predicted labels, but the difference is not substantial.

Interpreting the results for 20 categories proves challenging due to the difficulty in deriving general correlations from the histogram alone. Incorporating supplementary analytical tools, such as confusion matrices, is recommended to augment comprehension regarding the model's efficacy. Through this analysis, it can be concluded that evaluating the model from various perspectives during the interpretation process allows for a more comprehensive understanding of its performance.

04. Discussion

The implemented Multinomial Naive Bayes Text Classification Model, based on Bayes' theorem and developed using scikit-learn on the 20 newsgroups dataset, represents a notable advancement in text categorization. Demonstrating scalability and generalization across 20 diverse categories, the model showcases its adaptability for various text classification tasks. Achieving an overall accuracy of 76% through meticulous training and Laplace smoothing, the model's evaluation includes detailed confusion metrics and a histogram. Moreover, this model, initially designed for the scikit-learn data package, exhibits potential for adaptation to other text datasets through encapsulation into a Python class.

In the broader discussion on Naive Bayes classifiers for text data categorization, their simplicity and efficiency are underscored despite their stringent assumptions. These classifiers, with advantages such as speed, straightforward probabilistic predictions, interpretability, and minimal tunable parameters, prove effective in scenarios with well-separated categories or high-dimensional data. Their competitive performance, especially in text categorization tasks, is highlighted. The report concludes by suggesting future developments for the classification process, such as incorporating advanced feature engineering techniques, exploring sophisticated algorithms, and leveraging deep learning approaches to further enhance model performance.

References

Ji, Keungyeup and Kwon, Youngmi, "Implementation of Malicious Mail Filtering System through Refinement of MultinomialNB Technique", The Journal of Korean Institute of Information Technology, Vol. 21, No. 7, pp. 63-69, 2023.

Kelly, Anthony & Johnson, Marc. (2021). “Investigating the Statistical Assumptions of Naïve Bayes Classifiers”. 1-6.

M. R. d. H. Maia, A. Plastino and A. A. Freitas, "An Ensemble of Naive Bayes Classifiers for Uncertain Categorical Data," 2021 IEEE International Conference on Data Mining (ICDM), Auckland, New Zealand, 2021, pp. 1222-1227.

N. Kalcheva, G. Marinova and M. Todorova, "Comparative Analysis of the Bernoulli and Multinomial Naive Bayes Classifiers for Text Classification in Machine Learning," 2023 International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 2023, pp. 28-31.

Jake VanderPlas, (2016, November). Python Data Science Handbook https://jakevdp.github.io/PythonDataScienceHandbook/, (Accessed: 2023, December 22)