Bayesian learning for classifying Netnews text articles

Zakaria Khitirishvili
Report Due Date: 6/21/2024

June 11, 2024

1 Problem Statement

In the digital age, the volume of text data generated daily is immense, making it challenging to organize and classify information efficiently. One common application is the classification of news articles into predefined categories. Accurate classification aids in better content organization, improved search functionality, and enhanced user experience. Traditional methods struggle with the high-dimensional and sparse nature of text data, requiring advanced machine learning techniques.

2 Solution

The solution involves using the Naive Bayes classifier, a probabilistic machine learning algorithm well-suited for text classification due to its simplicity and effectiveness. This project will classify 20,000 Netnews articles into 20 distinct categories using the sparse word count features model to represent the text data. The algorithm will be implemented in Python, leveraging libraries like scikit-learn for the Naive Bayes classifier and more. Traditional methods like manual classification or rule-based systems are not scalable for large datasets and cannot handle the variability and nuances in text data effectively. The Naive Bayes classifier, on the other hand, provides a robust approach to model the probabilistic relationships between words and document classes, offering a scalable and automated solution.

Input: Preprocessed text data represented as feature vectors.

Output: Predicted class labels for each document.

Target Variable: Newsgroup category labels for each document.

The dataset consists of 20,000 newsgroup messages, with 1,000 documents from each of the 20 newsgroups. This data will be preprocessed to convert text into numerical features using a bag-of-words model. The dataset is available for download from a provided link. Features: the sparse word count features model values representing the textual content.

3 Assumptions, Constraints and Implications

The Naive Bayes classifier makes several key assumptions that impact its performance. It assumes that features (words) are conditionally independent given the class label, simplifying computation but potentially overlooking contextual dependencies in real-world text. The model also assumes a fixed vocabulary derived from the training data, meaning any words in the test data not present in the training set will be ignored, possibly losing some information. Furthermore, the evaluation presumes that the class distribution in the dataset is balanced; if not, the model might favor majority classes, leading to misleading accuracy metrics. Lastly, the model assumes that the training data is representative of the overall distribution of text documents in each category; otherwise, it may fail to generalize well to new, unseen data, resulting in poor test performance.

Limited computational resources (CPU/GPU, memory) may restrict the size of the dataset or the complexity of the model, which calls for simpler models or smaller datasets that might not fully capture the problem's complexity. The quality of the input text data, including issues like noise, irrelevant information, and inconsistent formatting, can impact model performance, requiring extensive and time-consuming preprocessing such as removing stop words. The length and complexity of text documents can affect the performance of the TF-IDF vectorizer and the Naive Bayes classifier, with very short or very long documents possibly leading to overfitting or underfitting.

The model is likely to perform well on classes with distinct vocabularies, providing high accuracy, precision, recall, and F1-scores, but may struggle with overlapping classes or nuanced language, resulting in degraded performance. Its ability to generalize to new data depends on the representativeness of the training data and the validity of the independence assumption. While this approach is scalable to reasonably large datasets, it may struggle with very large datasets or high dimensions, requiring more scalable methods or some sort of dimensionality reduction techniques.

4 Solution implementation

The solution utilized the 20 Newsgroups dataset, with tasks consisting of approximately 20,000 documents categorized into 20 different newsgroups. This dataset was provided as a zip file for the course assignment and stored locally, with each category represented by a directory containing 1000 text files. For model training and evaluation, pairs of specific categories were selected with tri-class classification and class-vs-class classification, allowing the model to be trained and tested on distinct subsets of the data.

The text data underwent preprocessing to ensure high-quality input before feature extraction. This preprocessing included removing stop words (common, uninformative words like "the" and "is"), tokenizing the text (splitting it into individual words or tokens), and standardizing the text by handling punctuation and converting it to lowercase. The primary feature learning technique applied was Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, which transforms text documents into numerical vectors that reflect the importance of each word relative to the entire dataset. TF-IDF combines Term Frequency (TF), which measures how often a word appears in a document, with Inverse Document Frequency (IDF), which measures how unique or rare a word is across all documents, thus weighting frequent but less informative words lower.

The model chosen for classification was the Multinomial Naive Bayes classifier, well-suited for text classification due to its efficiency and effectiveness in handling high-dimensional data. This algorithm leverages the probabilistic relationships between words and class labels. To streamline the process, a pipeline was created using "make-pipeline" from scikit-learn, which sequentially combines the TF-IDF vectorizer and the Naive Bayes classifier, ensuring consistent application of text data transformation and model training. The dataset was split into training and test sets using an 80-20 split, with the training set used to fit the model and the test set used to evaluate its performance. The model was trained on 5 groups of (1 vs 1) and (3 vs 3) of categories, and its effectiveness was assessed using performance metrics such as accuracy, precision, recall, and F1-score.

5 Analysis of Performance

We used following performance metrics:

Precision: Precision measures the proportion of correctly predicted positive observations to the total predicted positives.

Recall: Recall measures the proportion of correctly predicted positive observations to all observations in the actual class.

F1-score: The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both concerns.

Accuracy: Accuracy is the proportion of correctly predicted observations to the total observations.

Macro Average (macro avg): Macro averaging calculates the metric independently for each class and then takes the average (hence treating all classes equally).

Weighted Average (weighted avg): Weighted averaging calculates the metric independently for each class and then takes the average weighted by the number of instances in each class.

In Table 1 below, precision, recall, F1-score, and acuracy are all 1.00, indicating perfect classification performance for these two categories. This suggests that the model has no difficulty distinguishing between 'alt.atheism' and 'comp.graphics', likely because these categories have distinct vocabularies.

	Precision	Recall	F1-score	Support
alt.atheism	1.00	1.00	1.00	199
comp.graphics	1.00	1.00	1.00	201
Accuracy			1.00	400
Macro avg	1.00	1.00	1.00	400
Weighted avg	1.00	1.00	1.00	400

Table 1: Results for categories: ('alt.atheism', 'comp.graphics')

We used confusion matrix to calculate the performance metrics show in tables. Figure 1 below shows the confusion matrix for table 1. As we can see model did very well to correctly distinguish the 2 group.

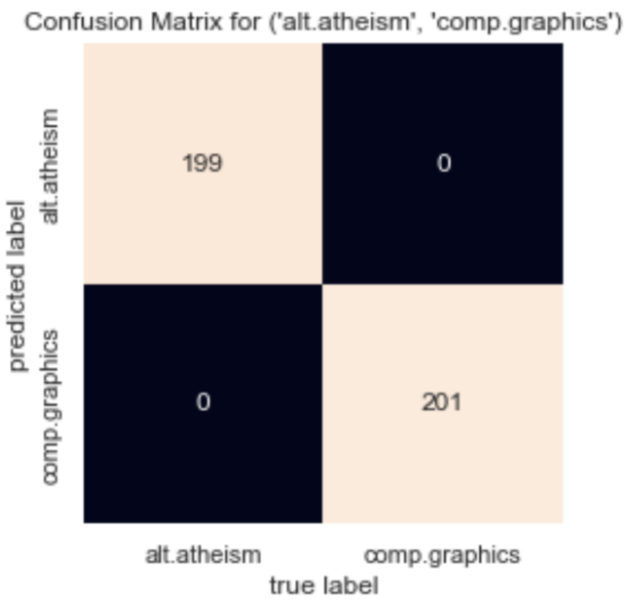

Figure 1: Confusion Matrix graph

In the Table 2 below, precision, recall, F1-score, and accuracy have almost perfect scores, with a slight drop in precision for 'alt.atheism' (0.99). This small dip indicates a very minor challenge in distinguishing between these two categories but overall excellent performance.

	Precision	Recall	F1-score	Support
alt.atheism	0.99	1.00	1.00	199
comp.sys.ibm.pc.hardware	1.00	1.00	1.00	201
Accuracy			1.00	400
Macro avg	1.00	1.00	1.00	400
Weighted avg	1.00	1.00	1.00	400

Table 2: Results for categories: ('alt.atheism', 'comp.sys.ibm.pc.hardware')

Confusion matrix for Table 2 is shown below in Figure 2. We observe that precision here wasn't 100% because for 1 example during testing, our model predicted 'alt.atheism' when it actually was 'comp.sys.ibm.pc.hardware'.

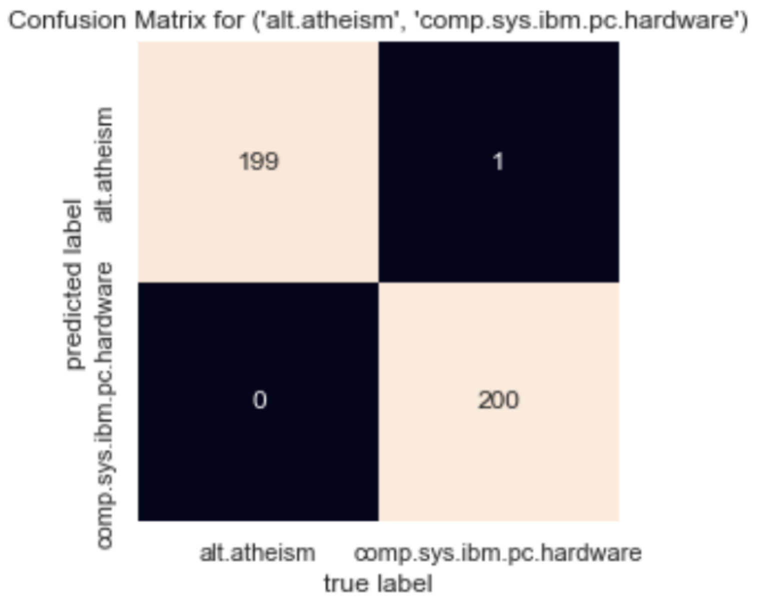

Figure 2: Confusion Matrix graph

In Table 3 below, alt.atheism achieves perfect scores across precision, recall, and F1-score, indicating the model has no difficulty classifying documents in this category. In comp.graphics, precision is high at 0.98, but recall is slightly lower at 0.93, resulting in an F1-score of 0.96. This suggests that while most predicted 'comp.graphics' documents are correct, some true 'comp.graphics' documents are missed. comp.sys.ibm.pc.hardware shows good performance with a precision of 0.94 and a high recall of 0.98, leading to an F1-score of 0.96. This indicates that most 'comp.sys.ibm.pc.hardware' documents are correctly identified, though there are a few false positives. The overall accuracy is 0.97, indicating that the model performs very well across these categories. Both macro and weighted averages for precision, recall, and F1-score are 0.97, demonstrating balanced performance across categories.

	Precision	Recall	F1-score	Support
alt.atheism	1.00	1.00	1.00	217
comp.graphics	0.98	0.93	0.96	197
comp.sys.ibm.pc.hardware	0.94	0.98	0.96	186
Accuracy			0.97	600
Macro avg	0.97	0.97	0.97	600
Weighted avg	0.97	0.97	0.97	600

Table 3: Results for categories: ('alt.atheism', 'comp.graphics', 'comp.sys.ibm.pc.hardware')

Similar to before, Figure 3 below shows confusion matrix for Table 3. We see that model had hardest time when it was looking at 'comp.graphics' but predicted it wrong as 'comp.sys.ibm.pc.hardware' 12 times.

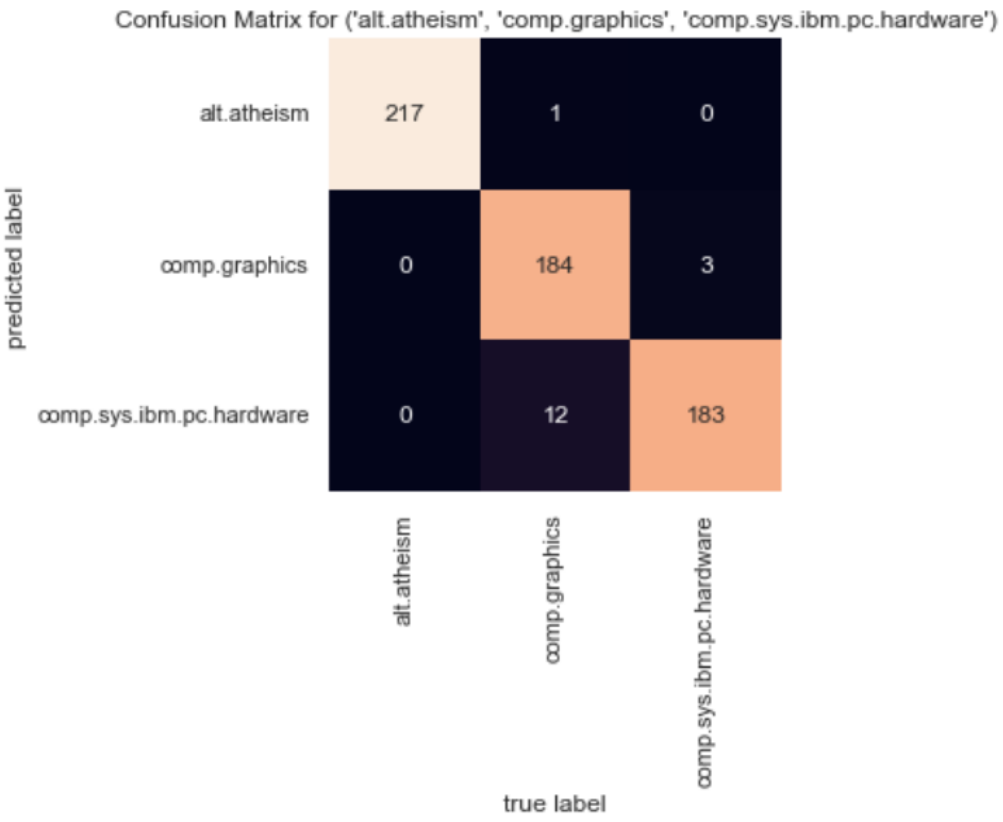

Figure 3: Confusion Matrix graph

In Table 4 below, alt.atheism: again, achieves near-perfect scores. comp.graphics maintains high precision at 0.99 but lower recall at 0.93, leading to an F1-score of 0.96. Similar to the previous set, true 'comp.graphics' documents are sometimes missed. comp.sys.mac.hardware shows strong performance with precision at 0.93 and a high recall of 0.98, resulting in an F1-score of 0.96, indicating effective classification with few false positives. The overall accuracy remains high at 0.97, showing the model's effectiveness. Both macro and weighted averages are 0.97, reflecting consistent performance across the categories.

	Precision	Recall	F1-score	Support
alt.atheism	0.99	1.00	0.99	217
comp.graphics	0.99	0.93	0.96	197
comp.sys.mac.hardware	0.93	0.98	0.96	186
Accuracy			0.97	600
Macro avg	0.97	0.97	0.97	600
Weighted avg	0.97	0.97	0.97	600

Table 4: Results for categories: ('alt.atheism', 'comp.graphics', 'comp.sys.mac.hardware')

Figure 4 below shows confusion matrix for Table 4. Where we observe that our model struggeled most with when we had 'comp.graphics' and predicted wrong as 'comp.sys.mac.hardware'

13 times.

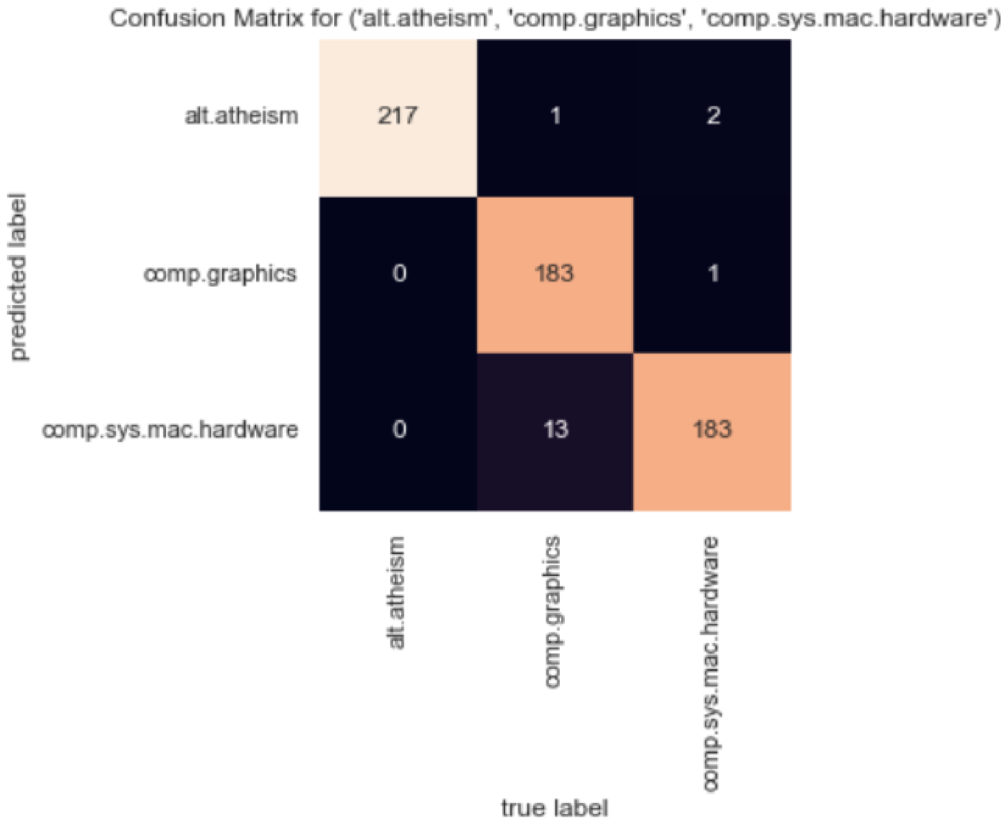

Figure 4: Confusion Matrix graph

Overall we see that when topics are similar for different groups, model gets lower score because it has harder time to distinguish between them. Conversly, if topic in categories (i.e words) are more different, then model gets high score since its easier to distinguish them. We observe this accross the board when running all possible pairs of (1 vs 1) and (3 vs 3).

6 Summary

The immense volume of daily text data poses a challenge for efficient classification, essential for better content organization and search functionality. This project addressed the problem using the Naive Bayes classifier to categorize 20,000 Netnews articles into 20 categories. The solution involved extensive preprocessing, including stop word removal, tokenization, and text standardization, followed by TF-IDF vectorization to transform text data into numerical features.

Key assumptions include feature independence and a fixed vocabulary, which simplify computation but may overlook contextual dependencies. Constraints such as limited computational resources, data quality issues, and the availability of labeled data impact performance, necessitating thorough preprocessing. Despite these challenges, the Naive Bayes classifier, combined with TF-IDF vectorization, offers a scalable and interpretable solution, performing well on distinct vocabularies but struggling with overlapping classes.

The implementation used the 20 Newsgroups dataset, with each category stored as text files. The model was evaluated on pairs and groups of categories using metrics like accuracy, precision, recall, and F1-score, demonstrating its effectiveness and scalability for text classification tasks.