Naive Bayes Classifier for Sentiment Analysis on Movie Reviews

Harry Denell

December 2024

1 Background

Sentiment analysis, often referred to as opinion mining is a special area within natural language processing (NLP) that aims to automatically detect and classify emotions or sentiments conveyed in text [1, 2]. The primary goal is to uncover information from text, which can offer insights into emotion or intent by its author [1, 2]. In recent years sentiment analysis has gained much intereest due to its ability to support tasks like analyzing public opinions, and providing valuable business intelligence [2]. Its widespread adoption by academics, businesses, and institutions is driven by its practical uses, including analyzing review feedback, tracking social media sentiment, and interpreting various forms of textual communication [2]. By leveraging sentiment analysis, organizations can make more informed choices, enhance customer satisfaction, and develop strategies aligned with consumer behavior [2].

However, despite its advantages sentiment analysis present several challenges, including dealing with informal writing, irony, and other language specific details, all of which make sentiment classification more complex [2]. An important aspect of sentiment analysis involves feature extraction, where the most useful information is retrieved from textual data with the goal of improving the performance of classification models [2]

1.1 Naive Bayes

Naive Bayes (NB) is a rather straightforward machine learning algorithm, which applies Bayes' theorem for classification tasks [3]. It operates under the assumption of conditional independence among features given the class label [3], which despite often being violated in real-world scenarios, allows the model to handle high dimensional data [2]. The assumption simplifies the computation of the posterior probability $P(C_k \mid X)$, as described by:

$$P(C_k \mid X) \propto P(C_k) \cdot P(X \mid C_k)$$

where $P(C_k)$ is the prior probability of class C_k and $P(X \mid C_k)$ is the likelihood of the feature X given class C_k (also known as the conditional probability). Despite its simplicity, NB often delivers high performance in tasks such as sentiment analysis and text classification, particularly when the training dataset is limited [2].

However, the performance may lack when the assumption of feature independence is

significantly violated by the data [2]. Furthermore, the performance of Naive Bayes can be hindered by issues such as class imbalance and the zero-frequency problem, where features absent from the training data lead to zero-probabilities. These challenges can be addressed using smoothing techniques, such as Laplace Smoothing.

1.1.1 Laplace Smoothing

Mathematically, the conditional probability is calculated as:

$$P(w_i \mid C_k) = \frac{\text{Count}(w_i, C_k)}{\text{Total words in } C_k}$$

where:

- $\text{Count}(w_i, C_k)$ is the number of occurrences of the word w_i in class C_k.
- Total words in C_k is the total number of word occurrences in class C_k.

This formulation assumes that all probabilities can be reliably estimated from the training data. However, in practice, some words may not appear in the training set for a specific class. When $\text{Count}(w_i, C_k) = 0$, the resulting probability $P(w_i \mid C_k)$ becomes zero, leading to the entire probability estimate for a review being zero due to the Naive Bayes assumption of independence. To address the issue of zero probabilities, we can apply Laplace smoothing [4], which ensures that every word has a small, nonzero probability, even if it was not observed in the training data for a specific class. The adjusted formula is:

$$P(w_i \mid C_k) = \frac{\text{Count}(w_i, C_k) + 1}{\text{Total words in } C_k + V}$$

where:

- V is the vocabulary size (the total number of unique words across all classes).
- The addition of $+1$ to the numerator is the smoothing factor, ensuring that every word has a baseline count of at least 1.
- The denominator is adjusted by V, accounting for the additional $+1$ added to each word count.

2 Objective

The objective of this project is to design and implement a Naive Bayes classifier from scratch to perform sentiment analysis on a large movie review dataset. The classifier will be built using Spark, with the goal of leveraging distributed computing capabilities to process and analyze the dataset efficiently. This implementation aims to achieve the following:

1. **Understand Naive Bayes Mechanics**: By avoiding pre built libraries, the project seeks to deepen the understanding of fundamental principles underlying Naive Bayes, including prior and conditional probabilities, and applying Laplace smoothing.
2. **Efficient Processing**: Ensure the training and evaluation processes are handled efficiently, leveraging Sparks distributed computation.

3. **Evaluation on Real-World Data**: Test the classifier's effectiveness using the IMDb Large Movie Review Dataset, assessing its accuracy, precision, recall, and F1-score.

3 Methodology

3.1 Dataset Aquisition and Splitting

For this project the IMDb Large Movie Review Dataset[1] was utilized, containing 50,000 movie reviews evenly divided into two classes: positive and negative sentiment. The original dataset was in CSV format. It was split into training and testing subsets. We used a 80/20 split between training and testing data, which is a common practice for training machine learning models. The dataset contained two columns, 'review', which is the actual review, and 'sentiment', which is the label, either 'negative' or 'positive'. The dataset was split to preserve class balance, maintaining class proportions in each subset. An ID column was added to aid model evaluation by linking predictions to specific reviews.

3.2 Data Preprocessing

Preprocessing the raw movie reviews was essential to clean and standardize the data, enabling the Naive Bayes model to focus on meaningful patterns. Spark's built-in tools were used for efficiency and to explore their capabilities, as the project focused on manually implementing the Naive Bayes model. Two distinct methods were implemented for this: one for the training phase, where the pipeline was fitted and saved, and another for the testing phase, which applied the same preprocessing using the saved model. This approach ensured consistent preprocessing between training and testing.

The following describes the full preprocessing:

- **Convert Sentiment Labels to Numerical Values:** Sentiment labels ("positive" and "negative") were converted into numerical values (1.0 and 0.0) to facilitate machine learning tasks.
- **Remove HTML Tags:** HTML markup within the reviews was stripped to ensure the model only relies on linguistic content.
- **Convert Text to Lowercase:** All text was converted to lowercase to treat words with different cases (e.g., "Good" and "good") as identical.
- **Tokenization:** Using RegexTokenizer, the text was split into tokens (words) based on non-alphanumeric characters, transforming continuous text strings into lists of terms.
- **Remove Stopwords:** Common words like "the", "and", and "is," which add little semantic meaning to a review, were removed using StopWordsRemover.
- **Feature Vector Representation:** Cleaned tokens were transformed into numerical feature vectors suitable for machine learning. CountVectorizer was used to represent each review as a sparse vector of token counts.

[1] `https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`

- **Pipeline Integration:** All preprocessing steps above were put into a pipeline, that was trained and saved. This returned id, cleaned features (from each review) and the corresponding label.

3.3 Model Training

The Naive Bayes classifier was implemented from scratch, with the following steps made to train the model:

- **Loading and Preprocessing Training Data:** The training data was loaded from a CSV file and processed through the preprocessing pipeline described earlier.
- **Computation of Statistics:** To optimize performance, the idea was to gather all necessary statistics with a single pass through the data. A single flatMap transformation emitted key-value pairs for label counts, total words per class, and word counts per class. Using reduceByKey, all emitted key-value pairs were aggregated. This step resulted in counts for labels, total words per class, and word frequencies in one pass.
- **Calculating Priors:** Priors were computed by dividing the count of documents in each class by the total number of documents. This provided the prior probability $P(\text{class})$.
- **Broadcasting for Efficient Computation:** The total word counts per class were collected into a map and broadcasted to all workers. Broadcasting this map provided access during the computation of conditional probabilities.
- **Calculating Conditional Probabilities with Laplace Smoothing:** Conditional probabilities $P(\text{word}|\text{class})$ were estimated by dividing the count of each word in a class by the total number of words in that class, adjusted with Laplace smoothing.
- **Saving Model Parameters:** The computed priors, word count per class, and conditional probabilities were saved to text files.

3.4 Model Testing

Once the model was trained, it was evaluated on the test data. The following steps were carried out to ensure consistency with the training process:

- **Loading and Preprocessing Test Data:** The raw test data was read the same way as the training data.The previously saved preprocessing pipeline from the training phase was applied to the test data. This ensured that all transformations were identical between training and testing phases.
- **Loading Model Parameters:** The priors, conditionals, and total words per class computed during training were loaded from external text files. Priors, conditionals and and total word counts were broadcasted. Broadcasting ensured all nodes had access to these parameters. This approach enabled parallel prediction of test data.
- **Computing Predictions:** For each instance in the test data, the model used the feature vector alongside the precomputed priors and conditional probabilities to calculate log-scores for each class label. The log-score for a given class was computed

by summing the logarithm of the prior probability of the class with the sum of the logarithms of the conditional probabilities of the observed features. If a feature was missing from the training data for a given class, Laplace smoothing was applied. This ensured all features contributed to the log-score, preventing undefined probabilities. To address potential numerical instability during computation, the highest log-score across all classes was subtracted from each log-score. This operation preserves relative probabilities. The adjusted log-scores were then transformed into probabilities using the softmax function, ensuring the resulting probabilities for all classes summed to one. The class associated with the highest probability was selected as the predicted label.

- **Generating Probability Distributions:** As described above, in addition to determining the predicted label, the model generated a probability distribution over the possible class labels for each instance. By including these probabilities in the output, a more detailed interpretation of the models predictions was enabled, which was used later in the evaluation step.
- **Saving Predictions:** The predictions were saved to text file in a tab separated format. Each row contained the ID, the actual label, the predicted label, and the probability distribution for the labels.

3.5 Model Evaluation

The model's performance was evaluated using common classification metrics to assess its ability to predict sentiment:

- **Accuracy:** The proportion of correctly classified instances relative to the total number of instances.
- **Precision & Recall:** Evaluate trade-off between false positives and false negatives, providing insights into the model's behavior under imbalanced outcomes.
- **F1-Score:** Represents the harmonic mean of precision and recall, offering a balanced measure for classification performance.

A confusion matrix was also generated to visualize the four different types of predictions. The evaluation process was implemented using a Python script that parsed the model's predictions and calculated these calculated metrics.

3.6 Output testing

To verify the correctness of the implementation, a controlled dataset was used to evaluate both the Traning phase and Testing phase components. The dataset was carefully chosen to ensure predictability in vocabulary and class distributions, allowing for manual computation and validation of the results. For simplicity, the number of features for the CountVectorizer (vocab) was set to 3, which was equal to all the words to be used in the dataset.

4 Results

4.1 Preprocessing

The original reviews contained unstructured text with HTML tags, inconsistent capitalization, and numerous stopwords. Table 1 shows an example review before preprocessing and the result after tokenization.

Table 1: Preprocessing Transformation Example

Stage	Review
Original Review	"If you like original gut wrenching laughter you will like this movie. If you are young or old then you will love this movie, hell even my mom liked it. Great Camp!!!", positive
After Tokenization	["if", "you", "like", "original", "gut", "wrenching", "laughter", "you", "will", "like", "this", "movie", "if", "you", "are", "young", "or", "old", "then", "you", "will", "love", "this", "movie", "hell", "even", "my", "mom", "liked", "it", "great", "camp"], 1.0

4.2 Model Training

The model training step computed and saved priors, conditionals, and word counts per class to text files. Table 2 displays the prior probabilities for each class label, showing an equal distribution between the two classes (which is coherent with what we knew of the data).

Table 2: Prior Probabilities of Each Class

Class Label	Prior Probability
0	0.5
1	0.5

Table 3 presents conditional probabilities for specific feature indices within classes, indicating the likelihood of a word appearing in a class.

Table 3: Sample Conditional Probabilities

Class Label	Feature Index	P(word | Class)
1	3933	5.25×10^{-5}
1	1002	2.04×10^{-4}
1	4771	3.64×10^{-5}
0	3348	5.41×10^{-5}
0	3079	5.15×10^{-5}

Table 4 summarizes the total number of words associated with each class. We recognize nearly identical word counts across both classes ($1,974,506.0$ for class 1 and $1,973,759.0$ for class 0).

Table 4: Total Word Counts per Class

Class Label	Total Word Count
1	1974506
0	1973759

4.3 Model Testing

After training the Naive Bayes classifier, the model was evaluated on the test dataset to assess it. Predictions were generated for each test instance.The prediction outputs were in a tab-separated format. Table 5 shows a subset of the predictions.

Table 5: Sample Predictions from the Naive Bayes Classifier

ID	Actual Label	Predicted Label	P(Class 0.0)	P(Class 1.0)
25096	1.0	1.0	7.69×10^{-10}	1.00
2048	0.0	0.0	1.00	6.53×10^{-10}
42616	1.0	1.0	0.1177	0.8823
11841	0.0	1.0	0.0116	0.9884

We conclude that the model showed rather high confidence in most of its predictions. For instance, ID 25096, a positive review, was correctly predicted with a probability of nearly 1.0 for class positive and almost 0 for the negative class. Some reviews, such as ID 42616, had lower confidence scores. ID 11841 was incorrectly predicted as positive with a high probability of 0.9884.

4.4 Model Evaluation

The performance of the classifier was evaluated using metrics such as accuracy, precision, recall, and F1-score. Table 6 summarizes the evaluation results. The classifier correctly identified 85.1% of the test instances. The F1-score reflects a well-rounded performance, though some misclassifications suggest potential for improvement in feature representation or handling ambiguous reviews.

Table 6: Performance Metrics of the Naive Bayes Classifier

Metric	Value
Accuracy	85.1%
Precision	85.3%
Recall	84.8%
F1-Score	85.1%

The confusion matrix in table 7 provides a breakdown of correct and incorrect predictions. This shows a balance between false positives (731) and false negatives (759), with the large majority of the predictions being correct.

4.5 Further evaluation

The evaluation script converted predictions to a CSV file, simplifying further analysis. The ID column helped tracing predictions back to specific reviews in the test dataset for further insights.

Table 7: Confusion Matrix

	Predicted Negative	Predicted Positive
Actual Negative	4269	731
Actual Positive	759	4241

4.5.1 Correct Prediction

The following entry in the prediction data represents a review that the model correctly classified as positive (although with some uncertainty). The corresponding CSV record is shown in table 8.

Table 8: CSV Record for Correctly Classified Review

ID	Actual Label	Predicted Label	Probabilities
10042	1.0	1.0	[0.3845, 0.6155]

The full review with ID 10042 is found below (formatted without HTML):

> "I wasn't expecting much from this tale of a kid whose term paper is stolen and turned into a movie script.. who then he travels to hollywood to get even.. but.. hey.. it's still a fun film. Frankie Muniz of ""Malcom in the Middle"" fame stars as the kid and is fairly good and Amanda Bynes as his friend is also very good. Yea the film does work as an advertisement for Universal Studios theme park and is really kinda a silly kids film.. but I enjoyed it anyways. GRADE: B-"

The model correctly predicted the positive sentiment of this review with a probability of 61.55% for the positive class. Despite some neutral or critical elements in the review, such as "kinda a silly kids film," the overall sentiment is positive, as evidenced by phrases like "fun film" and "I enjoyed it anyways.".

4.5.2 False Negative Prediction

We selected a CSV entry where the model misclassified a positive review as negative, shown in Table 9. The model assigned a 59.4% probability to the negative class and 40.6% to the positive class, reflecting moderate confidence in the incorrect prediction.

Table 9: Formatted Review Prediction Example

ID	Actual Label	Predicted Label	Probabilities
41612	1.0	0.0	[0.594, 0.406]

The full review with ID 41612 is found below (formatted without HTML):

> "I admit creating great expectations before watching because some friends mentioned it (and they are not pervs!) as a must see. And it is a must see! Just don't expect to see something outbreaking.
>
> The Freudian psychoanalyzes are interesting in many parts of the film, but there's just too much perversion and it doesn't stick in the end.

> Some of the good things are the analyzes of Kieslowski's Blue, most of David Lynch's, some of Hitchcock's and perhaps a couple more I missed (I just remembered...Dogville), and I usually don't miss things unless they are too obvious or loose in the air.
>
> Other than being repetitive, which makes it too long, the documentary is enjoyable in the sense of noticing some perversions fed by our unconscious, hence the commercial success of most thrillers studied and used as basis for this theory.
>
> I really enjoyed the energetic tone of the narration and the effort of Mr. Zizek to revive Freud's theory, which has been numb for too long, specially in north America. Again, it's way over the top and I believe not to be a completely waste of time for I do believe most humans have a dark appreciation for death and blood."

The misclassification suggests the model weighed negative phrases like "too much perversion," "doesn't stick in the end," and "being repetitive" more heavily than positive ones like "a must see" and "I really enjoyed the energetic tone," leading to the incorrect prediction.

4.5.3 False Positive Prediction

Another prediction data includes a review misclassified as positive, despite its actual negative label, as shown in Table 10.

Table 10: CSV Record for Misclassified Review

ID	Actual Label	Predicted Label	Probabilities
39792	0.0	1.0	[0.0101, 0.9899]

The full review with ID 39792 is found below (formatted without HTML):

> "The story of the bride fair is an amusing and engaging one, and it is to the filmmaker's credit that he sets out to portray rural Minnesotans with the same respect ordinarily reserved for Coast-dwellers. It is weird, though, to find an independent movie, the brainchild of a single person, that is as unambitious and cliché-ridden as a committee-brewed Hollywood potboiler.
>
> The portrait of rural people is intended to be affectionate, I think, but these characters don't ring true to me–I have had quite a few meals in small-town diners, but never overheard a debate on the merits of different nineteenth-century English novelists. One might suggest that writer/director Semans has no more experience with rural culture than the Coen brothers, and considerably less satiric verve."

The model assigned a high 98.99% probability to the positive label, likely influenced by phrases like "amusing and engaging" and "affectionate." However, the review's overall sentiment is negative, as shown by phrases like "unambitious and cliché-ridden" and "doesn't ring true.", suggesting the model overemphasized isolated positive terms over the broader sentiment context.

4.6 Testing

4.6.1 Training Phase

The training (test) data consisted of the following records:

```
id,review,sentiment
1,"cat cat dog",negative
2,"cat bird bird",positive
3,"cat dog bird",negative
```

The output from the training step was checked against precomputed expected values, in order to check the functionality of the code. A quick check of word counts per class confirmed that the output matched the expected values; P(Negative) = 0.6667 and P(Positive) = 0.3333.

For the word counts in each class, the results matched the expected values; Class Negative (0): cat: 3, dog: 2, bird: 1 and Class Positive (1): cat: 1, dog: 0, bird: 2.

The vocabulary generated by the CountVectorizer included the top three most frequent words: cat, bird, and dog. The words were assigned indices based on their term frequency [5], resulting in: cat: Index 0, bird: Index 1 and dog: Index 2. Using Laplace smoothing, the conditional probabilities $P(\text{word} \mid \text{class})$ were manually calculated as follows:

Class Negative (0): $P(\text{cat} \mid 0) = 0.444$, $P(\text{bird} \mid 0) = 0.222$, $P(\text{dog} \mid 0) = 0.333$

Class Positive (1): $P(\text{cat} \mid 1) = 0.333$, $P(\text{bird} \mid 1) = 0.5$

The expected output for the conditionals file was therefore:

```
0   0   0.444
0   1   0.222
0   2   0.333
1   0   0.333
1   1   0.500
```

This matched almost exactly (except for precision) with the code output for the conditionals, indicating that our implementation of the training phase for the naive bayes seems to be working.

4.6.2 Testing Phase

To evaluate the model, we tested unseen word-class combinations and single-word reviews to ensure the code handled these cases. The test data included the following records:

```
id,review,sentiment
4,"cat dog",negative
5,"cat",positive
```

We utilized the previously computed priors and conditionals from the training phase to predict the sentiment of each test review. Using these probabilities, we calculated (using non-log probabilities), for the test record:

```
4, "cat dog"
```

$$
\begin{aligned}
P(0 \mid \text{"cat dog"}) &\propto P(0) \times P(\text{cat} \mid 0) \times P(\text{dog} \mid 0) \\
&\approx 0.0988
\end{aligned}
$$

$$
\begin{aligned}
P(1 \mid \text{"cat dog"}) &\propto P(1) \times P(\text{cat} \mid 1) \times P(\text{dog} \mid 1) \\
&\approx 0.0185
\end{aligned}
$$

Normalizing, we get the following:

$$
\begin{aligned}
\text{Sum} &= 0.0988 + 0.0185 = 0.1173 \\
P(0 \mid \text{"cat dog"}) &= \frac{0.0988}{0.1173} \approx 0.842 \\
P(1 \mid \text{"cat dog"}) &= \frac{0.0185}{0.1173} \approx 0.158
\end{aligned}
$$

So the prediction is negative (0) with probability 0.842.

For the test record

```
id=5, "cat"
```

$$
\begin{aligned}
P(0 \mid \text{"cat"}) &\propto P(0) \times P(\text{cat} \mid 0) \\
&= 0.6667 \times 0.4444 \\
&\approx 0.2963
\end{aligned}
$$

$$
\begin{aligned}
P(1 \mid \text{"cat"}) &\propto P(1) \times P(\text{cat} \mid 1) \\
&= 0.3333 \times 0.3333 \\
&\approx 0.1111
\end{aligned}
$$

Normalizing this we get the following:

$$
\begin{aligned}
\text{Sum} &= 0.2963 + 0.1111 = 0.4074 \\
P(0 \mid \text{"cat"}) &= \frac{0.2963}{0.4074} \approx 0.727 \\
P(1 \mid \text{"cat"}) &= \frac{0.1111}{0.4074} \approx 0.273
\end{aligned}
$$

So the prediction is negative (0) with probability 0.727.

Following the format used in the training phase's conditionals file, the predictions are:

```
4 0.0 0.0 0.842 0.158
5 1.0 0.0 0.727 0.273
```

We conclude that this matched almost exactly with the output (only difference due to precision), making us even more confident that the naive bayes seems to be working as it should.

5 Discussion

The classifier delivered good performance on the test dataset, achieving reasonably high and balanced accuracy, recall, and precision. Manual verification of priors, conditionals, word counts, and test outputs seemed to confirm the correctness of our implementation. However, there is likely room for optimization and improvement in certain steps.

5.1 Implementation Evaluation and Challenges

Implementing the Naive Bayes classifier from scratch involved several considerations. One of the initial tasks was calculating the total number of documents. An accumulator was initially considered for this purpose, but Spark's lazy evaluation model required triggering an additional action to obtain the document count from this, which seemed to introduce some computational overhead, so it was decided not to use it here. Optimizing the statistical computations necessary for the training phase was another area were much attention was spent. Efforts were made to gather all necessary data in a single pass to enhance efficiency. It is possible that both of these challenges could have been handled more gracefully.

The prediction phase presented its own set of complexities, particularly in distributing the necessary information across the cluster to perform predictions in a distributed manner. Implementing the softmax function to convert log-scores into probabilities required careful handling to maintain numerical stability and ensure accurate probability normalization across distributed nodes. Due to our results it seems like it is working.

5.2 Possible Improvements

Several future improvements would be possible to explore.

Incorporating N-grams: Expanding the feature set to include n-grams could capture additional contextual patterns in the text, enabling the model to better understand relationships between consecutive words. This should be straight-forward after the second assignment of the course.

Lemmatization: Reducing words to their base form ("liked" $\rightarrow$ "like") could likely help the model generalize better by treating different forms of a word as equivalent.

TF-IDF: While the current implementation uses raw term frequencies, applying TF-IDF (Term Frequency–Inverse Document Frequency) could better highlight informative terms by weighting down common words.

Parameter Tuning: Experimenting with the vocabulary size (vocab) could possibly lead to some performance gains.

Benchmark: It would be both beneficial and interesting to compare the performance of this Naive Bayes implementation against Spark's MLlib Naive Bayes model as a benchmark to evaluate accuracy, efficiency, and possibly scalability if testing on new datasets.

6 Conclusion

In conclusion, the objective of implementing a Naive Bayes classifier for sentiment analysis on the IMDb dataset was achieved. Distributed computing was utilized to enable efficient processing, and key principles for Naive Bayes such as prior and conditional probabilities, Laplace smoothing, were applied. Overall, this project provided a valuable opportunity to apply course concepts in a practical setting, deepening both technical and theoretical knowledge.

References

[1] K. L. Tan, C. P. Lee, and K. M. Lim, "A survey of sentiment analysis: Approaches, datasets, and future research," *Applied Sciences*, vol. 13, no. 7, p. 4550, 2023.

[2] M. Wankhade, A. C. S. Rao, and C. Kulkarni, "A survey on sentiment analysis methods, applications, and challenges," *Artificial Intelligence Review*, vol. 55, no. 7, pp. 5731–5780, 2022.

[3] G. I. Webb, E. Keogh, and R. Miikkulainen, "Naïve bayes." *Encyclopedia of machine learning*, vol. 15, no. 1, pp. 713–714, 2010.

[4] University of Washington, "Section 6: Naive Bayes," [Online]. Available: https://courses.cs.washington.edu/courses/cse446/20wi/Section7/naive-bayes.pdf (accessed on: 2024-12-08).

[5] Apache Spark, "Extracting, transforming and selecting features.," [Online]. Available: https://spark.apache.org/docs/latest/ml-features.html#countvectorizer (accessed on: 2024-12-08).