

ASSESSMENT COVER SHEET

Assignment Details

Course: Applied Natural Language Processing_____

Semester/Academic Year: Sem 1 2023_____

Assignment title: Sentiment Analysis using Naïve Bayes____

Assessment Criteria

Assessment Criteria are included in the Assignment Descriptions that are published on each course's website.

Plagiarism and Collusion

Plagiarism: using another person's ideas, designs, words, or works without appropriate acknowledgment.

Collusion: another person assisting in the production of an assessment submission without the express requirement, consent or knowledge of the assessor.

Consequences of Plagiarism and Collusion


The penalties associated with plagiarism and collusion are designed to impose sanctions on offenders that reflect the seriousness of the University's commitment to academic integrity. Penalties may include the requirement to revise and resubmit assessment work, receiving a result of zero for the assessment work, failing the course, expulsion, and/or receiving a financial penalty.

declaration

I declare that all material in this assessment is my own work except where there is clear acknowledgment and reference to the work of others. I have read the University Policy Statement on Plagiarism, Collusion, and Related Forms of Cheating:

<http://www.adelaide.edu.au/policies/?230>

I give permission for my assessment work to be reproduced and submitted to academic staff for the purposes of assessment and to be copied, submitted, and retained in a form suitable for electronic checking of plagiarism.



Yash Kasundra (A1838670)

16/04/2023

SIGNATURE AND DATE

Introduction

Sentiment analysis has gained popularity in recent years as more people and businesses alike strive to understand the beliefs and attitudes of the audiences they serve. The analysis of movie reviews is a significant application for sentiment analysis, as it may help guide marketing and promotional efforts and offer insights into a movie's success or failure.

In this project, we will use the Naive Bayes technique to do sentiment analysis on a dataset of Internet Movie Database (IMDB) movie reviews. For text classification applications like sentiment analysis, the Naive Bayes machine learning method is frequently employed. It is straightforward but efficient. It functions by determining the likelihood that a given text belongs to a specific class (such as positive or negative) based on the frequency of specific words or phrases in that text.

Our objective is to use a subset of the IMDB dataset to train a Naive Bayes model with known sentiment labels (positive or negative), and then to use that model to categorise the sentiment of previously unobserved reviews. We will assess the F1 score of our model to determine its accuracy, and we will look into ways to enhance its performance through feature selection and hyperparameter tuning.

In addition to offering insights into the sentiment of moviegoers and the elements that influence their perceptions, this project's overall goal is to provide a practical introduction to sentiment analysis and Naive Bayes classification.

1. Exploratory Data Analysis:

Any data science project should include exploratory data analysis since it can help to understand the structure and features of the data and uncover any patterns or trends that might be pertinent to the investigation.

There are a number of EDA techniques that can help you interpret the data and get it ready for analysis for our IMDB sentiment research project.

- One popular method is to generate a word cloud, which gives the most frequently occurring words in the dataset a visual representation. This can be used to spot any outliers or anomalies as well as recurring themes or subjects in the evaluations.



Fig-1: Word Cloud for positive labelled reviews

- Analyzing the distribution of review ratings is another beneficial EDA method. To do this, a histogram of the critic reviews that displays the frequency of each review within the dataset can be created. By ensuring that the dataset is balanced between good and negative reviews, this can assist in identifying any biases or tendencies in the ratings.

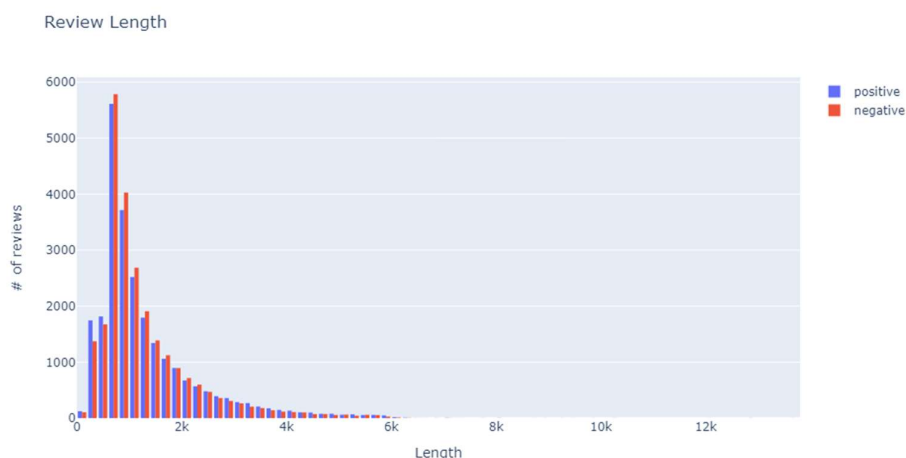


Fig-2: Histogram for checking distribution of positive and negative reviews

- Unigrams, bigrams, and trigrams are various n-gram models used in natural language processing to examine the frequency and distribution of words in a text corpus. You may find common language patterns and phrases in both positive and negative reviews

by using n-gram models.

Unigrams: A unigram model simply calculates the frequency of each word in a corpus of text without considering the words' order or context. In addition to giving you an understanding of the entire vocabulary used in the dataset, this might be helpful for identifying the most frequent words used in both positive and negative reviews.

Bigrams: A bigram model counts the frequency of a set of adjacent word pairs in a corpus of text. This can be helpful for spotting recurring words or phrases in the reviews, like "excellent movie" or "poor acting." The most frequent language patterns employed in each type of review can be found by comparing the frequency of bigrams in positive and negative evaluations individually.

Trigrams: A trigram model examines groups of three adjacent words in a corpus of text and calculates their frequency. This might be helpful for locating reviews that utilise more intricate terminology, such as "the acting was poor" or "the plot was interesting." You can learn more about the linguistic motifs and topics that are most prevalent in each type of review by examining the frequency of trigrams in the positive and negative reviews separately.

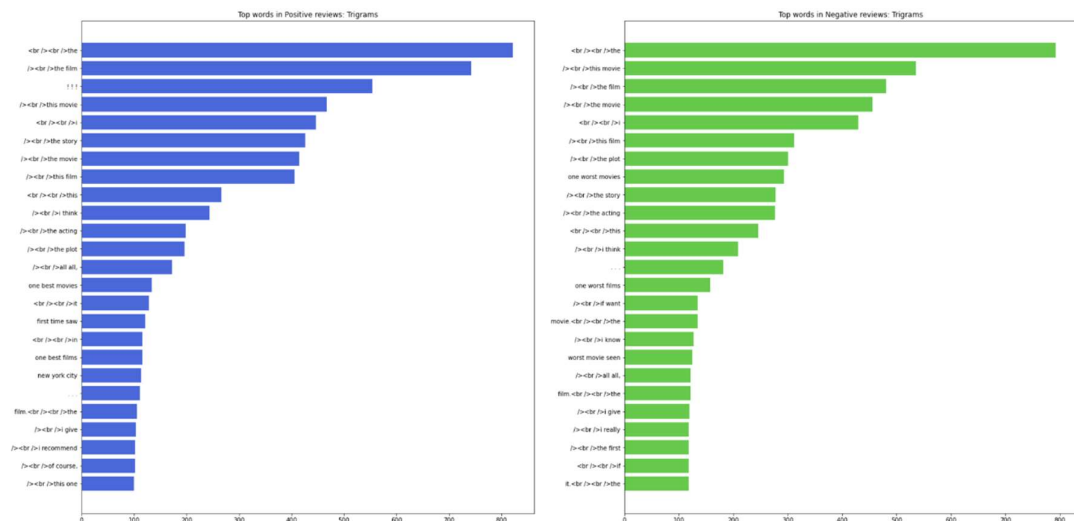


Fig-3: Trigram plot for positive and negative reviews

2.Preprocessing

Any natural language processing (NLP) task, including sentiment analysis, must first go through preprocessing. The purpose of preprocessing is to clean and change the raw text data into a format better suited for machine learning algorithms to analyze.

In the case of IMDB movie reviews, the preprocessing steps typically involve the following:

- **Lower_casing:** In order to avoid having duplicate vocabulary words, lowercase all the text. By lowering the amount of distinct terms, this procedure can also aid in simplifying the analysis[2].
- **Stop Word Removal:** Eliminating common words like "the", "a", "and" etc that aren't likely to convey much emotion or meaning. By doing so, you can make the vocabulary smaller and raise the precision of sentiment analysis[6,7].
- **Elimination of Special Characters and Punctuation:** By removing any special characters, punctuation, or numerals that are not important for sentiment analysis, you can decrease the feature space's dimensionality and improve the algorithm's performance[2].
- **Stemming or Lemmatization:** Reducing words to their root form, also known as stemming or lemmatization, aids in grouping together words with related meanings. As an illustration, the phrases "running," "runs," and "ran" might all be shortened to the word "run." Libraries like NLTK or SpaCy are frequently used for this stage[12].
- **Tokenization:** Breaking down the text into individual words or tokens, which serve as the fundamental units of analysis for NLP tasks, is known as tokenization. Often, a tokenizer library like NLTK or Spacy is used for this stage[5].

type	review	label	review_lower	review_transformed	review_lemma	review_stem	review_byte_pair
0	test	Once again Mr. Costner has dragged out a movie...	neg	once again mr. costner has dragged out a movie...	mr costner dragged movie far longer necessary ...	costner drag movie far longer necessary aside ...	mr costner drag movi far longer necessari asid...
1	test	This is an example of why the majority of acti...	neg	this is an example of why the majority of acti...	example majority action films generic boring r...	example majority action film generic bore real...	ex amp le _major ity _act ion _fil ms _gener ...
2	test	First of all I hate those moronic rappers, who...	neg	first of all i hate those moronic rappers, who...	first hate moronic rappers couldnt act gun pre...	first hate moron rapper couldnt act gun press ...	_first _h ate _m or on ic _ra p p ers _c ould ...
3	test	Not even the Beatles could write songs everyon...	neg	not even the beatles could write songs everyon...	even beatles could write songs everyone liked ...	even beatl could write song everyon like altho...	_ev en _be at les _c ould _writ e _song s _eve...
4	test	Brass pictures (movies is not a fitting word f...	neg	brass pictures (movies is not a fitting word f...	brass pictures movies fitting word really some...	brass picture movies fit word really somewhat ...	_br ass _p ict ures _mov ies _f it ting _wor d...

Table-1: Preprocessed review

3. Model

The Bayes theorem, which is a method of determining the likelihood of an event based on its prior information, is the foundation of the Naive Bayes algorithm, a classification algorithm. Because it makes the simplifying assumption that the features are conditionally independent of one another given the label name, the algorithm is referred to as "naive"[8].

For text classification applications like sentiment analysis, the probabilistic method Naive Bayes is frequently utilized. Calculating the likelihood of a certain label (such as "positive" or "negative") given some observable features is the fundamental notion behind Naive Bayes (e.g., the words in a text review)[9]. Bayes' rule can be used to convey the following:

$$P\left(\frac{Label}{Features}\right) = \frac{P\left(\frac{Features}{Label}\right) * P(label)}{P(Features)}$$

Equation-1: Bayes Theorem

P(label) is the prior probability of the label (i.e., the probability of the label in the absence of any evidence) and P(features) is the probability of the observed features. P(label | features) is the probability of the label given the observed features, and P(features | label) is the probability of the observed features given the label.

According to Naive Bayes, the characteristics (i.e., the text review's words) are conditionally independent given the label. In other words, the likelihood of a word not being in a review is unaffected by the presence of a word. This presumption enables us to assume that the probability of each word in the review given the label is independent of the other words in the review, which simplifies the computation of P(features | label). Hence, we can write P(features | label) as follows:

$$P\left(\frac{Features}{Label}\right) = P\left(\frac{word1}{Label}\right) * P\left(\frac{word2}{Label}\right) * ... * P\left(\frac{wordn}{Label}\right)$$

Equation-2: Simplified Naïve Bayes

where word1, word2, ..., wordn are the words in review and P(wordi | label) denotes the likelihood that the ith word will appear given the label.

With a training dataset of review labelled with their associated labels, we estimate the probability P(wordi | label). We count the instances of each word in the review from each label, and then, to prevent zero probability, we use Laplace smoothing:

$P(wordi | label) = (\text{count of wordi in reviews belonging to label} + 1) / (\text{total number of words in reviews belonging to label} + \text{size of vocabulary})$

where the training dataset's total number of distinct words is the vocabulary size.

We simply count the number of reviews in each label and divide by the total number of reviews to determine the prior probability for each label:

$P(label) = \text{count of reviews belonging to label} / \text{total number of reviews}$

These computations are carried out for a specific training dataset by the "fit" function in the code. The "predict" function then makes use of these probabilities to categorize additional reviews according to the observed words. Using the aforementioned algorithm, it determines the likelihood of each label given the observed words and then selects the label with the highest likelihood as the predicted label. Below Fig- 4,5 shows the implemented “fit” and “predict” methods for naïve bayes.

Each word's count in the input list of words is mapped to by the dictionary that the get_word_counts function produces.

A pandas dataframe called df_fit and a string called s are the inputs for the fit function. It determines the size of the vocabulary as well as the prior probability for each class and the number of words in each class. It gives back a tuple with the number of words in each class, their prior probabilities, the number of words in each class's vocabulary, and the number of messages in each class.

The output of the fit function is passed as input along with a list of strings called df_predict. For each message in df_predict, it calculates the log-probabilities of each class, adds the log-prior probability for each class, and then returns a list of predicted labels.

```
def get_word_counts(words):
    count_words = {}
    for word in words:
        count_words[word] = count_words.get(word, 0.0) + 1.0
    return count_words

def fit(df_fit, s):
    msg_num = {}
    prior_log_class = {}
    count_words = {}
    vocab = set()

    n = df_fit.shape[0]
    msg_num['pos'] = df_fit[df_fit['label']=='pos'].shape[0]
    msg_num['neg'] = df_fit[df_fit['label']=='neg'].shape[0]
    prior_log_class['pos'] = math.log(msg_num['pos'] / n)
    prior_log_class['neg'] = math.log(msg_num['neg'] / n)
    count_words['pos'] = {}
    count_words['neg'] = {}

    for x, y in zip(df_fit[s], df_fit['label']):
        counts = get_word_counts(nltk.word_tokenize(x))
        for word, count in counts.items():
            if word not in vocab:
                vocab.add(word)
            if word not in count_words[y]:
                count_words[y][word] = 0.0

            count_words[y][word] += count

    return count_words, prior_log_class, vocab, msg_num
```

Fig-4: Implemented “Fit” function for Naïve Bayes

```
def predict(df_predict, vocab, count_words, msg_num, prior_log_class):
    result = []
    for y in df_predict:
        counts = get_word_counts(nltk.word_tokenize(y))
        positive_score = 0
        negative_score = 0
        for word, _ in counts.items():
            if word not in vocab: continue

            # add Laplace smoothing
            positive_log_w = math.log((count_words['pos'].get(word, 0.0) + 1) / (msg_num['pos'] + len(vocab)))
            negative_log_w = math.log((count_words['neg'].get(word, 0.0) + 1) / (msg_num['neg'] + len(vocab)))

            positive_score += positive_log_w
            negative_score += negative_log_w

        positive_score += prior_log_class['pos']
        negative_score += prior_log_class['neg']

        if positive_score > negative_score:
            result.append('pos')
        else:
            result.append('neg')
    return result
```

Fig-5: Implemented “predict” function for Naïve Bayes

Evaluation Metric: We will be using F1 score as our evaluation metric. The F1 score, which combines precision and recall to produce a single, all-inclusive measure of model performance, is a frequently used evaluation metric in statistical analysis and machine learning. In binary classification issues, when there are only two classes, it is particularly helpful (e.g., positive and negative).

Recall is the percentage of real positive examples that the model properly detected whereas precision measures the proportion of positive forecasts that are accurate. The harmonic mean of recall and precision makes up the F1 score, which equally emphasises both metrics.

$$F1 \text{ score} = 2 * \frac{Precision * Recall}{Precision + Recall} \quad F1 \text{ Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Equation-3: Equation for calculating F1 score

An increase in the F1 score, which goes from 0 to 1, indicates greater model performance. When there is a class imbalance in the data or when both precision and recall matter, it is a helpful metric. Yet depending on the issue at hand, alternative evaluation metrics like accuracy or AUC (Area Under the Curve) may be more appropriate. It might not always be the ideal statistic to employ.

4. Analysis of various factors in the implementation

It's crucial to remember that your preprocessing techniques will probably have a favorable effect on how well the classifier performs. Preprocessing has probably lessened the noise in the data and made it simpler for the classifier to discern between various sentiments by eliminating stopwords, HTML, numerals, and punctuation.

Nonetheless, the results of various preprocessing methods can differ based on the particular dataset and issue at hand. For instance, while eliminating stopwords can be useful for tasks involving sentiment analysis, it might not be as useful for activities involving topic modeling. Consequently, it's crucial to test out various preprocessing methods and gauge how they affect the particular work at hand.

I found it intriguing that utilizing data with only preprocessing (without stemming or lemmatization) led to a somewhat higher F1 score than using data without any preprocessing but only lowercasing the review alone. This shows that even without stemming or lemmatization, some preprocessing may be advantageous for tasks involving sentiment analysis.

It's important to remember that the F1 score for naive Bayes trained on data with preprocessing and stemming was lower compared to preprocessing without stemming. This might be because stemming might cause some textual information to be lost, which could have a detrimental effect on how well the classifier performs.

Lemmatization, on the other hand, produced the greatest F1 score during preprocessing, indicating that it would be a helpful preprocessing method for sentiment analysis tasks.

The lowest F1 score was obtained with byte-pair encoding, which is important to note. Byte-pair encoding is a text normalisation method that includes joining the text's most frequent letter pairs. These findings imply that while it may be efficient for some natural language processing applications, sentiment analysis may not be one of them.

In conclusion, it's critical to test out various preprocessing methods and gauge how they affect the particular task at hand. In order to choose the preprocessing methods that will work best for your specific assignment, it's also crucial to take into account any potential trade-offs between them, such as the information loss caused by stemming.

Naive Bayes Model	F1 score
Lower case review without preprocessing	0.8336860957342884
With only preprocessing	0.8414361389052384
With preprocessing and stemming	0.7622397934484428
With preprocessing and lemmatization	0.8397785777881005
With preprocessing and byte pair encoding	0.6663815683968568

Table-2: Result comparison for different naïve bayes model trained on different data

References:

- [1] Jurafsky, D., & Martin, J. H. (2019). Speech and Language Processing (3rd ed.). Pearson. [Link: <https://web.stanford.edu/~jurafsky/slp3/>]
- [2] Zhang, H., & Wallace, B. C. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. arXiv preprint arXiv:1510.03820. [Link: <https://arxiv.org/abs/1510.03820>]
- [3] Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval, 2(1-2), 1-135. [Link: <https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf>]
- [4] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781. [Link: <https://arxiv.org/abs/1301.3781>]
- [5] Manning, C. D., & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. MIT Press. [Link: <https://nlp.stanford.edu/fsnlp/>]
- [6] Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media, Inc. [Link: <http://www.nltk.org/book/>]
- [7] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781. [Link: <https://arxiv.org/abs/1301.3781>]
- [8] Ranga, V. (2020) Naïve Bayes algorithm -implementation from scratch in python., Medium. Medium. [Link: <https://medium.com/%40rangavamsi5/na%C3%AFve-bayes-algorithm-implementation-from-scratch-in-python-7b2cc39268b9>]
- [9] Stanina, I. (2021) Implementing naive Bayes algorithm from scratch - python., Medium. Towards Data Science. [Link: <https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41>]
- [10] Byte-pair encoding tokenization - hugging face course (no date) Byte-Pair Encoding tokenization - Hugging Face Course. [Link: <https://huggingface.co/course/chapter6/5?fw=pt>]
- [11] Saumyab (2022) Stemming vs lemmatization in NLP: Must-know differences, Analytics Vidhya. [Link: <https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/#:~:text=Stemming%20is%20a%20process%20that,%27%20would%20return%20%27Car%27>]