

Vilnius Gediminas technical university

Faculty of fundamental sciences

Department of Information Technology

Arailym Issayeva

ITfuc-21

**code EXPLANATION**

Title of the research paper: Automated sentiment evaluation

of study quality feedback survey open questions answers

Supervisor in the university: Prof. Simona Ramanauskaitė

Vilnius, 2023

1. Research Methodology
   1. Dataset

In this chapter, we will explain how we approach sentiment analysis on the dataset "exchange\_students\_feedback.csv." The dataset includes 224 written feedback entries from exchange students. Our main goal is to analyze the sentiments expressed in these feedbacks using two different methods: a lexicon-based approach with VADER and an alternative machine learning approach with the Naive Bayes algorithm. Before we perform sentiment analysis, we need to preprocess the raw text data, so it becomes structured and suitable for analysis.

To conduct sentiment analysis on the exchange students' feedback dataset, we will use the Python programming language along with popular libraries such as NLTK (Natural Language Toolkit) and scikit-learn. The NLTK library will help us with text preprocessing tasks, including tokenization and stopword removal. Additionally, we will leverage the VADER lexicon from the NLTK library to perform the lexicon-based sentiment analysis. For the machine learning approach, we will utilize the Naive Bayes algorithm implemented in the scikit-learn library.

We'll perform the analysis using Jupyter Lab, an interactive environment for executing code, visualizing data, and documenting our methodology and results.

* 1. Data Pre-processing

To get started, we break down each feedback sentence into individual words, also known as tokens. This step helps convert the unstructured text data into a well-organized form that we can analyze effectively using computational methods (Nasim et al. 2017).

We'll begin by walking through the text processing steps using a simple example with just one text. This will give us a clear understanding of text pre-processing before we move on to work with the complete CSV data. During this process, we will also handle common text cleaning tasks, such as converting all text to lowercase and removing any special characters or punctuation marks.

* Tokenization

Let's start by considering an example from the 10th row of the dataset, which contains a lengthy text consisting of three sentences

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Tokenization serves as the first step in the text processing pipeline. Using the `sent\_tokenize` function, it breaks down the lengthy text into separate sentences and organizes them into an array. This enables us to analyze the sentiments expressed in each sentence independently. Now, we have three separate sentences, which you can see below:

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Furthermore, after dividing the text into sentences, we move on to the tokenization of individual words using the word\_tokenize function. This important step allows us to extract the necessary information that will help us in the next step to filter out unnecessary details and noise from the text. The results of this process are displayed below:

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

In addition, it is crucial to make a clear distinction between sentence tokens and word tokens, as demonstrated in the arrays provided below. Word tokenization is the process of breaking down each word into separate elements, and this step becomes especially important when we proceed with removing stop words.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Once we have performed lower casing and removed punctuation, we effectively eliminate any extraneous information from the text, ensuring a cleaner and more focused representation of the content.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

* Stopwords Removal

Stopwords are words that occur frequently in a language and generally do not carry significant meaning. In natural language processing (NLP) tasks, it is common to remove stopwords to reduce noise and highlight the most relevant words in a text.

The following code makes use of NLTK's stopwords corpus, which contains a collection of common English stopwords. It creates a set called sw to store these stopwords.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт, типография

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

The provided code below filters out stop words from a list of word tokens and then displays the resulting filtered word tokens for each sentence.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

* Stemming

The code provided below makes use of NLTK's PorterStemmer to perform stemming, which reduces words to their base form by removing prefixes and suffixes. This process allows for simplification and grouping of similar words. The resulting `stemmed\_word\_tokens` list contains the reduced root forms of the words. By comparing the word tokens before and after stemming, you can observe the differences and how words are transformed to their basic forms.

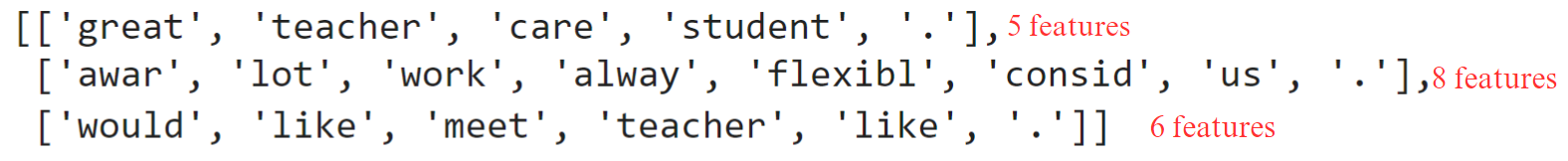
Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

* Converting words into vector (Bag-of-Words)

Bag of Words (BoW) is a fundamental technique in NLP that helps computers understand and process text. Since computers cannot comprehend words directly, BoW converts text into numbers for analysis.

The BoW process starts with tokenization, where the text is split into individual words. Each word becomes a feature, creating a feature set denoted as "X."



To address the challenge of different text lengths, as evident from the arrays shown above, the Bag of Words (BoW) technique utilizes a fixed-size dictionary. This dictionary assigns unique numbers to each word in the corpus. The BoW representation is achieved by counting the occurrences of words from the dictionary in a given text. This process results in a fixed-size array, with each element indicating the frequency of a specific word in the text. Words not present in the text have corresponding zero values. For a visual representation, please refer to the figure provided above.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Now that we have covered the theoretical aspects, let me elaborate on our practical implementation of converting text into numerical vectors using the Bag of Words (BoW) approach.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Автоматически созданное описание

With the text processing completed, it's time to explore the opportunities for sentiment analysis. We have two approaches at our disposal: a lexicon-based method called VADER and a machine learning-based method known as Naive Bayes. Let's proceed to the next paragraph to delve into these options further.

* 1. Used Methods
* Lexicon-based approach

The given code utilizes the VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon from NLTK (Natural Language Toolkit) for sentiment analysis. VADER is particularly useful in scenarios where we deal with short text and lack a pre-trained dataset.

Here's an overview of how the code operates:

The code imports the `Sentiment Intensity Analyzer` class from NLTK's VADER module and initializes an instance of it, to analyze the sentiment of the given text.

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

The code iterates through a list of sentences, and for each sentence, it applies the `analyzer.polarity\_scores(sentence)` method. This method returns a dictionary containing sentiment scores, including 'compound' score, which represents the overall sentiment of the sentence. The 'compound' score ranges from -1 to 1, where -1 indicates highly negative sentiment, 1 indicates highly positive sentiment, and values close to 0 indicate a more neutral sentiment.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

The sentiment label is assigned using a threshold approach. If the 'compound' score is greater than 0.05, the sentence is classified as positive. If the 'compound' score is lower than -0.05, the sentence is classified as negative. Sentences with 'compound' scores between -0.05 and 0.05 are classified as neutral.

The reason for choosing a threshold of 0.05 is to strike a balance between precision and recall in sentiment analysis. Using lower thresholds like 0.01 may result in classifying more sentences as positive or negative, which would increase recall (the ability to correctly identify positive or negative sentences). However, this could also lead to some false positives or false negatives, where sentences are misclassified. The upcoming chapter we will discuss which is the most efficient threshold for sentiment analysis. By testing various thresholds with manually labeled data, the goal is to determine the optimal threshold that achieves the desired balance between precision and recall. This iterative testing process helps ensure the sentiment analysis model provides accurate and reliable results while minimizing misclassifications.

Finally, the code prints each sentence along with its sentiment scores and sentiment label to provide a clear overview of the sentiment analysis results.

Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание

Now that we understand text preprocessing and sentiment analysis using VADER, let's move on to the Naive Bayes approach.

* Machine learning approach

The Machine Learning approach in sentiment analysis requires a pre-existing dataset for training, which allows for more precise and accurate predictions when provided with abundant data. This method is advantageous over using VADER lexicon when dealing with the ever-evolving nature of internet and social media communication, as VADER may struggle to keep up with new expressions and trends. However, the Machine Learning approach has a limitation in that it requires a larger amount of labeled data to achieve higher levels of accuracy in its predictions.

Now, let's begin the explanation of our code:

Importing libraries: `pandas` for data handling, `CountVectorizer` for feature extraction, `MultinomialNB` for the Naive Bayes classifier, and `accuracy\_score` for evaluating model performance.

We start by defining an example, which will serve as our test input for the sentiment analysis. Our training data is represented by the 'labeled\_feedback.csv' file. Using the Naive Bayes algorithm, we train the model on the training data. Once the model is trained, we feed the test data (example) into the model, and the model predicts its sentiment label. This process follows the logic of machine learning, as discussed in the 2nd chapter.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Afterwards, text preprocessing is carried out on both the testing and training data, followed by the conversion of words into numerical vectors.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

We initialize the Naive Bayes Classifier and train it using a labeled dataset. The classifier learns patterns and associations between features and sentiment labels during training, allowing it to make accurate predictions on new, unseen data. We then transform the 'example' into a numerical vector.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Once the ‘example’ text is transformed into numerical feature vectors, we deploy the trained classifier to predict the sentiment. The classifier examines the learned patterns from the training data and employs them to assign a sentiment label to the example text.///

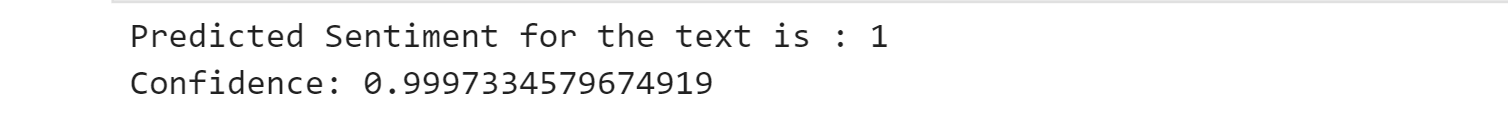
Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

In addition to predicting the sentiment, the classifier also provides a measure of confidence associated with its prediction. A higher confidence score implies a more certain prediction, while a lower score suggests some uncertainty.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание



-

1. Results and Discussion

As stated earlier, we described each step using a single text row for clarity. Now, we will proceed to work with our CSV file/dataset. Our initial focus will be on using Vader sentiment analysis, where we will experiment with various thresholds to find the most optimal one.

Starting the analysis with a threshold set to 0, we achieved an accuracy and recall of 44.19%. The fact that both metrics are the same indicates that the model is making correct predictions. It's a positive sign as the model is not biased towards one category over the other. Additionally, the precision obtained was 51.92%, which tells us how well the model is capturing relevant positive instances among its predictions.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

When we set the threshold to 0.01, we observed a consistent accuracy of 61.60%. Surprisingly, the accuracy remained the same even when the threshold was increased to 0.02.

In both cases, the accuracy and recall were identical. Moreover, the precision obtained at these thresholds was 73.74%. Precision indicates how many of the predicted positive instances were relevant, and in our case, it's showing a relatively good level of accuracy in identifying true positive sentiments.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

At thresholds ranging from 0.03 to 0.05, we achieved an accuracy of 61.16% with identical accuracy and recall. Moreover, the precision attained at these thresholds was 73.5%. In another study focused on movie reviews, using a threshold of 0.05 with the Vader lexicon, the researchers obtained an accuracy of 77%, a precision of 78.46%, and a recall of 85% (Bonta et al., 2019).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Considering the results, we can conclude that thresholds 0.01 and 0.02 yield higher accuracy, precision and recall in Vader lexicon. However, it's worth noting that the achieved accuracy of 61.6% is still relatively low, indicating potential limitations in accurately predicting sentiments using this approach.

Additionally, I would like to mention MonkeyLearn, an alternative sentiment analysis tool, which achieved an accuracy of 63.83% and precision of 65.26%. Comparing this result with the accuracy obtained using Vader, we can observe a slightly higher performance with MonkeyLearn.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Let's shift our focus to Naive Bayes, which achieved an impressive accuracy of 72.27%. This high accuracy demonstrates Naive Bayes' robust performance in sentiment analysis, surpassing lexicon-based methods like Vader and MonkeyLearn.After splitting the data, 80% means 180 rows of text was allocated for training, and the remaining 44 rows was used for testing. As a result, the model's accuracy stands at 72.27%, and precision is equal to 73.87%.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

In the context of Naive Bayes, Singh et al. (2017) conducted a study using a small training dataset and observed that it led to more accurate predictions, achieving an accuracy of 85.127%. Additionally, Pang et al. (2002) found that using Naive Bayes for sentiment analysis on movie reviews resulted in an accuracy of 78.7%.