**第二部分：方法 - 描述程序设计和详细方法，并附上一些图表。**

In natural language processing tasks, data preprocessing is a crucial step. For the VADER algorithm, first, relevant libraries need to be imported, and an instance of the sentiment analyzer is created through SentimentIntensityAnalyzer(). Then, the data is traversed from the outer layer (language) to the inner layer (training set, etc.). During this process, the tokenized data (tokenized\_data) and the original un-tokenized data (original\_data) are extracted. For each tokenized sentence, the sentiment score is obtained by using the created analyzer through analyzer.polarity\_scores(sent), and these scores are added to the sentiment\_scores list. Finally, the result dictionaries of each dataset type are aggregated, with the language as the key, to form a dictionary containing the sentiment analysis results of all languages and each dataset type, thus completing the data preprocessing of the VADER algorithm.

For the Word2Vec algorithm, the data of different languages also needs to be traversed first, operating from the outer layer to the inner layer. During the training stage, the sentence data is used for model training, and after the training is completed, the trained Word2Vec model is obtained. When using it, based on the given word list "words", the corresponding word vectors of each word are obtained with the help of the model's word vector dictionary and returned in the form of a list. In addition, for the visualization effect, the word vectors of some words can be selected, the corresponding vectors are obtained in the trained Word2Vec model, and then through dimensionality reduction methods such as t-SNE, the high-dimensional vectors are mapped to a two-dimensional or three-dimensional space, and scatter plots and other visual charts are made to visually observe the relationships between word vectors and assist in understanding the data characteristics, further improving the effect of data preprocessing and the understanding of the data.

图表：

Vader：表格

描述已自动生成

Word2Vec：  
图表

中度可信度描述已自动生成

**第三部分：实施细节 - 总结程序实施的详细信息。**

**VADER:**

Overall Function:

This function uses the VADER model to perform sentiment analysis on the data loaded by the load\_sentiment\_data function and returns the analysis result in a specific dictionary structure.

Steps and Functions:

1. Initialize the Analyzer:
   * Code line: analyzer = SentimentIntensityAnalyzer()
   * Function: Creates an instance of the VADER sentiment analyzer for subsequent sentiment analysis of sentences and obtaining sentiment scores.
2. Traverse Language Data:
   * Code segment: for lang, lang\_data in data\_dict.items(): and related internal loops
   * Function:
     + The outer loop traverses each language (such as Indonesian, English, Javanese, etc.) in the loaded data dictionary.
     + For each language, the inner loop will further process the training, validation, and test data of that language.
3. Process Each Dataset Type:
   * Code segment: for set\_type in ['train', 'valid', 'test']: and related internal operations
   * Function:
     + For each language, process its training set, validation set, and test set data respectively.
     + First, extract the tokenized data (tokenized\_data) and the original un-tokenized data (original\_data).
4. Analyze Sentence Sentiment and Record Scores:
   * Code segment: for sent in tokenized\_data: and related internal operations
   * Function:
     + Traverse the tokenized sentence data.
     + For each sentence, use the previously created analyzer to obtain the sentiment score through analyzer.polarity\_scores(sent) and add the score to the sentiment\_scores list.
5. Construct the Return Result Dictionary:
   * Code segment: analyzed\_lang\_data[set\_type] = (sentiment\_scores, original\_data) and related operations
   * Function:
     + Combine the analyzed sentiment score list and the original data into a tuple as the value corresponding to the dataset type (training, validation, test).
     + Then aggregate the result dictionaries of each dataset type into a dictionary with the language as the key, and finally return this dictionary containing the sentiment analysis results of all languages and each dataset type.

**Word2Vec：**

**Overall Function**:

This code mainly realizes training the Word2Vec model based on machine translation data and can obtain the word vectors of specified words from the trained model.

**Steps and Functions**:

**train\_word2vec\_on\_mt\_data Function**:

1. **Data Preparation**:
   * Code segment: for lang, lang\_data in mt\_data\_dict.items(): and internal loops
   * Function: Traverse the machine translation data dictionary loaded by the load\_mt\_data function. For each language's training, validation, and test datasets, extract the tokenized data and further process it into a word list form, and merge all these word lists into all\_sentences to prepare data for training the Word2Vec model.
2. **Model Creation and Parameter Setting**:
   * Code line: model = Word2Vec(sentences=all\_sentences, vector\_size=100, window=5, min\_count=1, workers=4)
   * Function: Use gensim.models.Word2Vec to create an instance of the Word2Vec model and set relevant parameters. For example, vector\_size specifies the dimension of the word vector as 100, window sets the context window size as 5, min\_count indicates that a word must appear at least 1 time to be included in the training, and workers sets the number of threads used for training as 4.
3. **Model Training**:
   * Code line: model.train(all\_sentences, total\_examples=len(all\_sentences), epochs=10)
   * Function: Use all the prepared sentence data to train the model, specifying the total number of examples as the length of all\_sentences and the number of training epochs as 10, so that the model can learn the distributed representation of words.
4. **Return the Trained Model**:
   * Code line: return model
   * Function: Return the trained Word2Vec model for subsequent use.

**get\_word\_vectors Function**:

1. **Obtain Word Vectors**:
   * Code line: return [model.wv[word] for word in words]
   * Function: From the input trained Word2Vec model, according to the given word list words, obtain the corresponding word vector of each word through the model's word vector dictionary wv and return it in the form of a list. The elements in the list are numpy arrays representing word vectors.

**第四部分：实验 - 评估方法并分析结果，包括定性和定量分析，附上表格、图表和讨论。**

**VADER:**

**Qualitative Analysis**:

When the confusion matrix made is applied on the test set, it can be seen that on the English test set, the accuracy of the model is relatively high.

图表, 树状图

描述已自动生成 图表, 树状图

描述已自动生成 图表

描述已自动生成

**Quantitative Analysis**:

**Accuracy (Accuracy Rate)**: Measures the proportion of correctly predicted samples in the total samples, reflecting the overall prediction accuracy of the model

**Precision (Precision Rate)**: Among the samples predicted as positive cases, the proportion of true positive cases, reflecting the precision of the model's prediction of positive cases.

**Recall (Recall Rate)**: Among the samples that are actually positive cases, the proportion of samples correctly predicted as positive cases by the model, reflecting the model's ability to capture positive cases.

**F1 (F1 Score)**: Comprehensively considers the precision rate and recall rate, and is the harmonic mean of the two, used to comprehensively evaluate the balanced performance of the model in predicting positive and negative cases.

文本

描述已自动生成