Optilearn.text.text_to_tokens

08 November 2024 19:47

text_to_tokens

The text_to_tokens function converts textual data into tokens based on the specified tokenization criteria. This function is highly configurable, allowing for tokenization based on words or individual characters, and provides additional options to include custom vocabulary, handle out-of-bound tokens, and adjust tokenization rules.

Parameters

1. **data** (list, tuple, set, or pandas.Series):
   - The input text data to be tokenized. Each element should be a string.
2. **apply_on** (str, optional):
   - Specifies the tokenization criteria.
   - Options include:
     - 'words' or 'word': Tokenizes the data by words.
     - 'alpha', 'alphabet', or 'alphabets': Tokenizes the data by individual characters.
   - Default is 'words'.
3. **add_vocabulary** (list, set, tuple, or dict, optional):
   - An additional vocabulary to be used for tokenization. If provided, this vocabulary is incorporated alongside the tokens from the input text.
   - Note: When an external vocabulary is used, it is recommended to set apply_on to 'words' or 'word'.
   - Default is None.
4. **oob_token** (str, optional):
   - Specifies an out-of-bound (OOB) token for words or characters not found in the vocabulary. If provided, this token will have a unique identifier higher than any existing token value.
   - Default is None.
5. **smart_recognition** (bool, optional):
   - If True, applies advanced rules for handling contractions, special cases, and common linguistic patterns.
   - Default is True.
6. **sp_character_tokens** (bool, optional):
   - If True, treats special characters (e.g., emojis, symbols) as individual tokens.
   - Default is False.
7. **token_case** (str, optional):
   - Specifies the case for tokens.
     - 'same': Maintains the original case of the text.
     - 'lower': Converts tokens to lowercase.
     - 'upper': Converts tokens to uppercase.
   - Default is 'same'.
8. **token_type** (str, optional):
   - Specifies the token type for output.
     - 'advance': Uses advanced encoding with unique numerical IDs for each token.
   - Default is 'advance'.

Returns

- A tuple containing:
  - **List of tokenized representations** of the input text data, where each word or character is converted into a unique token.
  - **Dictionary mapping each word or character to its corresponding token**, which can be used for quick reference or reusability.

Notes

- **Punctuation Handling**: For word-based tokenization, punctuation is separated from words and assigned as individual tokens.
- **Vocabulary Expansion**: When add_vocabulary is provided, the function incorporates any new words encountered in the input text to expand the vocabulary.
- **Out-of-Bound Token (OOB)**: If oob_token is specified, it provides a way to handle words or characters not found in the vocabulary, ensuring consistency and preventing errors.

Example

```
# Input text data
data = ['This is an example sentence.', 'Another sentence for testing.']

# Basic tokenization
tokens, token_dict = text_to_tokens(data)

# Output
print(tokens)
# Example Output: [[203451153151, 35, 2714, 273913, 3135], [6328519, 3135, 264, 5951157314

print(token_dict)
# Example Output: {'This': 203451153151, 'is': 35, 'an': 2714, 'example': 273913, 'sentenc
#                  'Another': 6328519, 'for': 264, 'testing': 5951157314539}
```

Benefits

- **Flexible Tokenization**: Allows tokenization based on words or individual characters, making it suitable for a range of text processing needs.
- **Custom Vocabulary Support**: Users can specify additional vocabulary, which helps maintain consistent tokenization across datasets.
- **Advanced Token Types**: Generates unique numerical IDs for each token, which can support advanced NLP tasks like embedding and model training.

This function is particularly useful for applications in Natural Language Processing (NLP) where consistent, customizable tokenization is required. Its options for case handling, punctuation separation, vocabulary expansion, and special character recognition make it a versatile tool for preprocessing textual data.