Токенизация в NLP: Разбираем текст на части

Токенизация – это процесс разбиения текста на более мелкие единицы, называемые токенами. Эти токены могут быть словами, частями слов или даже отдельными символами. Правильный выбор метода токенизации критически важен для производительности модели, так как он напрямую влияет на:

- **Размер словаря (vocabulary size):** Слишком большой словарь увеличивает вычислительные затраты и риск переобучения. Слишком маленький словарь может привести к потере важной информации.
- **Обработку редких слов (out-of-vocabulary, OOV):** Модели должны уметь справляться со словами, которые не встречались во время обучения.
- **Языковую специфику:** Некоторые языки (например, китайский) требуют более сложной токенизации, чем другие.

Основные методы токенизации:

1. **Токенизация на основе пробелов (Whitespace Tokenization):**
    - **Принцип:** Простейший метод, разделяет текст на токены по пробелам.
    - **Пример:** `"Hello world!"` -> `["Hello", "world!"]`
    - **Плюсы:** Очень быстро и просто.
    - **Минусы:**
        - Плохо обрабатывает пунктуацию (знаки препинания привязываются к словам).
        - Не подходит для языков без пробелов (китайский, японский).
        - Не решает проблему редких слов.
2. **Токенизация на основе слов (Word-based Tokenization):**
    - **Принцип:** Использует предопределенный словарь слов. Если слово есть в словаре, оно становится токеном.
    - **Пример:** (Используя словарь `{"hello", "world", "!"}`) `"Hello world!"` -> `["hello", "world", "!"]`
    - **Плюсы:** Проста для понимания.
    - **Минусы:**
        - Огромный размер словаря (особенно для больших корпусов).
        - Неизбежная проблема OOV (редкие слова). Невозможно включить все слова в словарь.
        - Не обрабатывает словообразование (разные формы одного и того же слова рассматриваются как разные токены).
3. **Токенизация на основе подслов (Subword Tokenization): Решение проблемы OOV**
    - **Основная идея:** Разбивать слова на более мелкие, значимые части (морфемы, части слов). Это позволяет модели понимать и генерировать новые слова, используя известные подслова.
    - **Преимущества:**
        - Уменьшает размер словаря по сравнению с word-based tokenization.

        - Эффективно обрабатывает редкие слова (OOV), разбивая их на известные подслова.
        - Может улавливать морфологические закономерности в языке.
    - **Основные алгоритмы:** BPE, WordPiece, Unigram, SentencePiece

Рассмотрим Subword Tokenization алгоритмы подробнее:

4. **Byte Pair Encoding (BPE)**

    - **Принцип:**
        1. Начинаем с словаря, содержащего все отдельные символы (буквы) в корпусе.
        2. Итеративно:
            - Считаем частоту каждой пары токенов, идущих подряд в корпусе.
            - Объединяем наиболее частую пару токенов в новый токен и добавляем его в словарь.
        3. Повторяем шаг 2 до достижения желаемого размера словаря.
    - **Пример:**
        - Исходный текст: `"low low low lower newer newer"`
        - Начальный словарь: `{"l", "o", "w", "e", "r", "n"}`
        - Счетчик пар: `("l", "o"): 3, ("o", "w"): 3, ("l", "o"): 1, ("o", "w"): 1, ("e", "r"): 2, ("n", "e"): 2, ("w", "e"): 2`
        - Объединяем наиболее частую пару `"lo"` -> добавляем `"lo"` в словарь.
        - Счетчик пар: `("lo", "w"): 3, ("lo", "w"): 1, ("e", "r"): 2, ("n", "e"): 2, ("w", "e"): 2`
        - Объединяем наиболее частую пару `"low"` -> добавляем `"low"` в словарь.
        - И так далее...
    - **Токенизация:** При токенизации, алгоритм пытается разбить слово на самые длинные возможные токены из словаря. Например, если в словаре есть `"lower"` и `"low"`, то `"lower"` будет токенизировано как `["lower"]`, а не `["low", "er"]`.
    - **Плюсы:**
        - Прост в реализации.
        - Эффективно уменьшает размер словаря.
        - Хорошо обрабатывает редкие слова.
    - **Минусы:**
        - Может создавать токены, которые не имеют лингвистического смысла.
        - Детерминированный: результат зависит от порядка обработки данных.
        - Жадный алгоритм: может принимать неоптимальные решения на ранних этапах, которые потом сложно исправить.

5. **WordPiece**

    - **Принцип:**

        1. Начинаем с словаря, содержащего все отдельные символы.
        2. Итеративно:
            - Обучаем языковую модель на текущем словаре (например, униграммную модель).

        - Для каждой пары токенов, идущих подряд, вычисляем "оценку" того, насколько вероятно объединение этой пары улучшит языковую модель. Эта оценка обычно связана с вероятностью встретить эту пару вместе. (Упрощенно говоря, WordPiece выбирает пару токенов, объединение которых приведет к наименьшей потере правдоподобия данных).
        - Объединяем пару, которая имеет наибольшую оценку.
    3. Повторяем шаг 2 до достижения желаемого размера словаря.

- **Ключевое отличие от BPE:** WordPiece использует языковую модель для выбора пары для объединения, а не просто частоту встречаемости.

- **Пример:** Представьте, что объединение "un" и "able" улучшит предсказательную способность модели больше, чем объединение "low" и "er", даже если "lower" встречается чаще в тексте.

- **Плюсы:**

    - Более лингвистически обоснованный, чем BPE (хотя и не всегда).
    - Лучше обобщает на новые данные.

- **Минусы:**

    - Сложнее в реализации, чем BPE (требуется обучение языковой модели).
    - Все еще может создавать токены без очевидного лингвистического смысла.

6. **Unigram Language Model Tokenizer**

- **Принцип:**
    1. Начинаем с большого словаря, содержащего все возможные токены (слова, части слов, символы).
    2. Обучаем униграммную языковую модель на этом словаре. Униграммная модель присваивает каждому токену вероятность его появления в тексте.
    3. Итеративно:
        - Для каждого токена в словаре вычисляем, насколько увеличится правдоподобие данных (likelihood) при *удалении* этого токена из словаря. Токены, удаление которых сильно ухудшает правдоподобие, считаются важными.
        - Удаляем из словаря определенный процент токенов, удаление которых наименее сильно ухудшает правдоподобие.
    4. Повторяем шаги 2 и 3 до достижения желаемого размера словаря.
- **Токенизация:** Использует алгоритм Viterbi для нахождения наиболее вероятной последовательности токенов для данного слова. То есть, разбивает слово на токены таким образом, чтобы максимизировать произведение вероятностей этих токенов (согласно униграммной модели).
- **Плюсы:**
    - Позволяет получить несколько возможных токенизаций одного и того же слова (это может быть полезно для обучения моделей).
    - Относительно прост в реализации.
- **Минусы:**

        - Может быть вычислительно дорогим, особенно для больших словарей.
        - Требует хорошей настройки параметров (например, процент удаляемых токенов).

7. **SentencePiece**

    - **Принцип:** Не является самостоятельным алгоритмом токенизации, а скорее *библиотекой и фреймворком*, который реализует BPE и Unigram, но с важными дополнениями.
    - **Ключевые особенности:**
        - **Рассматривает пробелы как обычные символы:** Это позволяет токенизировать текст без предварительной токенизации на основе пробелов. Пробелы кодируются специальным символом (например, _). Это особенно важно для языков, где пробелы не используются для разделения слов.
        - **Реализует BPE и Unigram:** Вы можете использовать SentencePiece для обучения и применения моделей BPE и Unigram.
        - **Кросс-платформенность:** Поддерживает различные языки программирования и платформы.
    - **Плюсы:**
        - Простота использования.
        - Поддержка различных алгоритмов токенизации.
        - Удобная обработка пробелов.
    - **Минусы:** Не является принципиально новым алгоритмом, а скорее удобной оберткой для существующих.

Подводные камни и общие соображения:

- **Размер словаря:** Важно найти баланс между размером словаря и качеством представления текста. Слишком маленький словарь приведет к потере информации, а слишком большой - к переобучению.
- **Языковая специфика:** Не существует универсального алгоритма токенизации, который идеально подходит для всех языков. Некоторые языки (например, китайский, японский, корейский) требуют специальных подходов. Для них обычно используются морфемные анализаторы или токенизаторы, обученные на больших корпусах.
- **Предобработка текста:** Качество токенизации сильно зависит от предобработки текста. Важно правильно обработать пунктуацию, регистр, специальные символы и т.д.
- **Выбор алгоритма:** Выбор алгоритма зависит от конкретной задачи и данных. BPE - хороший выбор для начала, WordPiece - для более сложных задач, Unigram - для задач, требующих нескольких возможных токенизаций. SentencePiece - отличный инструмент, чтобы упростить работу с BPE и Unigram.
- **Эксперименты:** Важно экспериментировать с разными алгоритмами и параметрами токенизации, чтобы найти оптимальный вариант для вашей задачи.

Примеры кода (Python):

```
from tokenizers import ByteLevelBPETokenizer, BertWordPieceTokenizer,
UnigramBPETokenizer
from sentencepiece import SentencePieceTrainer, SentencePieceProcessor
```

```
# BPE
tokenizer_bpe = ByteLevelBPETokenizer()
tokenizer_bpe.train_from_iterator(["This is a sample text."],
vocab_size=50, min_frequency=1)
tokens_bpe = tokenizer_bpe.encode("This is a sample text.").tokens
print(f"BPE Tokens: {tokens_bpe}")

# WordPiece
tokenizer_wp = BertWordPieceTokenizer(lowercase=False)
tokenizer_wp.train_from_iterator(["This is a sample text."], vocab_size=50,
min_frequency=1)
tokens_wp = tokenizer_wp.encode("This is a sample text.").tokens
print(f"WordPiece Tokens: {tokens_wp}")

# Unigram
tokenizer_unigram = UnigramBPETokenizer()
tokenizer_unigram.train_from_iterator(["This is a sample text."],
vocab_size=50, min_frequency=1)
tokens_unigram = tokenizer_unigram.encode("This is a sample text.").tokens
print(f"Unigram Tokens: {tokens_unigram}")

# SentencePiece (Unigram)
SentencePieceTrainer.train('--input=<input_file> --model_prefix=spm --
vocab_size=50 --model_type=unigram') # Create <input_file> first
sp = SentencePieceProcessor(model_file='spm.model')
tokens_sp = sp.encode_as_pieces("This is a sample text.")
print(f"SentencePiece Tokens: {tokens_sp}")
```

Заключение:

Токенизация – это важный шаг в обработке естественного языка. Понимание различных алгоритмов токенизации и их особенностей позволит вам строить более эффективные и robust NLP-модели. Не бойтесь экспериментировать и адаптировать подходы к вашим конкретным задачам!