Snowball Algorithm

A Deep Dive into Efficient Stemming

Luca Bazzetto
Michele Zazzaretti

Aim of Snowball

- Provide a canonical implementation of the Porter algorithm
- Facilitate creation of stemmers for languages other than English
- Create a way to define stemmers that could be automatically translated into various programming languages

How Snowball Works

- **Rule-Based Suffix Stripping:**
  - Define language-specific rules (e.g., remove "-ing" after vowels).
  - Apply recursively in multiple steps.
- **Example:**
  - "fishing" → Remove "-ing" → "fish"
  - "fished" → Remove "-ed" → "fish"

Porter Algorithm in Snowball

The Snowball implementation of the Porter algorithm:

1. Defines consonants and vowels (consonant = letter other than A, E, I, O, U, and Y preceded by consonant)
2. Calculates "measure" (m) of a word part as the number of VC sequences
3. Applies rules in sequential steps to remove suffixes
4. Rules are based on the longest matching pattern and conditions like minimum stem length

Example structure: `(condition) S1 → S2` where S1 is replaced by S2 when condition is met.

Real-World Applications

- Search Engines
- Sentiment Analysis
- Document Clustering

Python Implementation

```
import nltk

from nltk.stem import SnowballStemmer

stemmer = SnowballStemmer("english")

words = ["running", "runs", "runner"]

stems = [stemmer.stem(word) for word in words]

print(stems)

# Output: ['run', 'run', 'runner']
```

Advantages & Limitations

Pros:

- Multi-language support (Spanish, French, etc.)
- Lightweight and fast

Cons:

- Still heuristic-based (not perfect)
- Requires linguistic expertise to create new rules

Conclusion

- Snowball balances speed and accuracy for text normalization.
- Integration with AI for adaptive rule-learning.

Resources

The Porter stemming algorithm - Snowball
The English (Porter2) stemming algorithm - Snowball
Snowball: A language for stemming algorithms