Naïve Semantic Text Similarity Model

Zachary Parent

UPC

November 30, 2024

Outline

Introduction

- Semantic Text Similarity (STS) is crucial for many NLP tasks
- Challenge: Which features best capture semantic similarity?
- Our approach: Unbiased feature analysis using Random Forests

Methodology

- Approach
- Feature extraction
- Feature selection
- Model training
- Model evaluation

Approach

- Naïve approach which requires no knowledge of the corpus
- Use categorized steps to process sentences in every permutation
  - 521 permutations
  - e.g. sentence_to_doc, chunk_NEs, remove_stopwords, lemmatize_tokens, get_characters, get_2grams
- Apply 4 similarity metrics to each permutation
- Used Random Forest's feature importance capabilities
- Let the data guide feature selection

Feature Extraction

```
def generate_valid_permutations(
    functions: List[Callable] = all_functions,
) -> List[Tuple[Callable, ...]]:
    valid_permutations = []
    for n in range(1, len(functions) + 1):
        for perm in itertools.permutations(functions, n):
            if _is_valid_permutation(perm):
                valid_permutations.append(perm)
    valid_permutations = (
        [tuple([sentence_to_doc]) + perm for perm in valid_permutations])
    valid_permutations = (
        [new_perm for perm in valid_permutations for new_perm in add_final_step(perm)])
    return valid_permutations
```

Top Features

- Jaccard similarity dominates (7 of top 10)
- Common pipeline steps: lemmatization, stopwords, n-grams
- Top feature accounts for 20% importance

Feature Pipeline	Importance
score_jaccard_165	0.197
score_cosine_257	0.089
score_cosine_165	0.069
score_jaccard_258	0.033
score_cosine_258	0.022

Figure: Top 5 Features by Importance

Conclusions

- Simple features can be highly effective
- Pipeline complexity isn't always better
- Character-level analysis with n-grams shows promise