ars20gramopteng

November 11, 2024

Algorithmisch Rekursive Sequenzanalyse 2.0

[ ]:
```
Example for optimizing grammar
```

Paul Koop November 2024 post@paul-koop.org

[ ]:
```
## Problem statement and solution approach

### Problem
Given is an empirical sequence of terminal characters obtained from a specific␣
↪language model.
The task is to create a context-free probabilistic grammar that can generate␣
↪sequences that match
the empirical sequence as closely as possible. The aim is to find a grammar␣
↪whose probability
distributions are adjusted so that the relative frequencies of the generated␣
↪sequences
come as close as possible to the terminal characters occurring in the empirical␣
↪sequence.

### Procedure
1. **Initial Grammar Definition**: An initial probabilistic grammar is defined␣
↪that can
generate different possible sequences with different probabilities.

2. **Generation and frequency analysis**: Multiple sequences are generated␣
↪based on the
grammar, and the relative frequency of each terminal character is calculated.

3. **Correlation test**: The correlation between the relative frequencies of␣
↪the terminal
characters in the generated and in the empirical sequence is calculated. The␣
↪Spearman rank
correlation coefficient serves as a measure of agreement. Additionally,
a p-value is calculated to check the significance of the correlation.

4. **Optimization by adjusting probabilities**: If the correlation is low,
   the probabilities within the grammar are iteratively adjusted to better
```

```
    match the generated sequences to the empirical sequence. This ensures
    that the probabilities are normalized to 1 after each adjustment.

5. **Stopping criterion**: The optimization process ends when a significantly
    high correlation is achieved or the maximum number of iterations is reached.

### Result
After the optimization process, the adjusted grammar is output, along with
    the best correlation coefficient and its significance level (p-value). A␣
↪significantly high correlation means that the grammar has been successfully␣
↪adapted to the empirical sequence and can serve as a model for its␣
↪generation.
```

[4]:

```python
import numpy as np
from scipy.stats import spearmanr
from collections import Counter

# Beispielhafte empirische Sequenz
empirical_sequence = ['KBG', 'VBG', 'KBBd', 'VBA', 'KBBd', 'VBA', 'KBA', 'VBA',
                      'KBBd', 'VBBd', 'KBBd', 'VBBd', 'KBA', 'VBA', 'KBA',␣
↪'VBA',
                      'KAA', 'VAA', 'KAV', 'VAV']

# Grammatik mit Gewichtungen
grammar = {
    "<Start>": [["<Begrüßung>", "<Bedarf>", "<Abschluss>", "<Verabschiedung>",␣
↪1.0]],
    "<Begrüßung>": [["KBG", "VBG", 1.0]],
    "<Bedarf>": [["<BedarfSegment>", "<Bedarf>", 0.8], ["<BedarfSegment>", 0.
↪2]],
    "<BedarfSegment>": [["KBBd", "VBBd", 0.4], ["KBBd", "VBA", 0.3], ["KBA",␣
↪"VBA", 0.3]],
    "<Abschluss>": [["KAA", "VAA", 0.6], ["VAA", "KAA", 0.4]],
    "<Verabschiedung>": [["KAV", "VAV", 0.7], ["VAV", "KAV", 0.3]]
}

# Funktion zur Generierung einer Terminalsequenz basierend auf der Grammatik
def generate_terminal_sequence(grammar):
    sequence = []
    production_stack = ["<Start>"]
    while production_stack:
        production = production_stack.pop()
        if production in grammar:
            rules = grammar[production]
            options = [rule[:-1] for rule in rules]  # Die Produktionsregeln␣
↪ohne die Wahrscheinlichkeit
```

```python
            probabilities = [rule[-1] for rule in rules]  # Die Wahrscheinlichkeiten
            selected_rule = np.random.choice(len(options), p=probabilities)  # Index der gewählten Regel
            production_stack.extend(reversed(options[selected_rule]))
        else:
            sequence.append(production)
    return sequence

# Berechne die Häufigkeit der Terminalsymbole in einer Liste von Sequenzen
def calculate_frequencies(sequences):
    flat_sequence = [symbol for sequence in sequences for symbol in sequence]
    total_count = len(flat_sequence)
    frequencies = {symbol: count / total_count for symbol, count in Counter(flat_sequence).items()}
    return frequencies

# Häufigkeiten der empirischen Sequenz
empirical_frequencies = calculate_frequencies([empirical_sequence])

# Optimierungsprozess
best_correlation = -1
best_grammar = grammar
best_p_value = None

for _ in range(100):  # Maximale Anzahl an Iterationen
    # Generiere mehrere Sequenzen und berechne die Häufigkeit der Terminalzeichen
    generated_sequences = [generate_terminal_sequence(grammar) for _ in range(1000)]
    generated_frequencies = calculate_frequencies(generated_sequences)

    # Konvertiere Frequenzdaten für Korrelationstest
    empirical_values = np.array([empirical_frequencies.get(symbol, 0) for symbol in empirical_frequencies])
    generated_values = np.array([generated_frequencies.get(symbol, 0) for symbol in empirical_frequencies])

    # Berechne die Korrelation zur empirischen Sequenz
    spearman_corr, spearman_p = spearmanr(empirical_values, generated_values)

    # Aktualisiere das beste Ergebnis, falls signifikant und besser
    if spearman_p < 0.05 and spearman_corr > best_correlation:
        best_correlation = spearman_corr
        best_grammar = grammar
        best_p_value = spearman_p
```

```python
    # Wenn die Korrelation akzeptabel ist, beende die Schleife
    if spearman_corr > 0.8:  # Setze einen gewünschten Korrelationswert fest
        break

    # Andernfalls passe die Wahrscheinlichkeiten leicht an
    for key in grammar:
        for i, rule in enumerate(grammar[key]):
            new_prob = rule[-1] + np.random.uniform(-0.05, 0.05)
            grammar[key][i][-1] = max(0, min(new_prob, 1))  # Stelle sicher, dass die Wahrscheinlichkeiten im Bereich [0, 1] bleiben
        # Normiere die Wahrscheinlichkeiten neu, damit ihre Summe 1 ist
        total_prob = sum(rule[-1] for rule in grammar[key])
        for rule in grammar[key]:
            rule[-1] /= total_prob

# Ausgabe der optimierten Grammatik, Korrelation und Signifikanz
print("Optimized Grammar:", best_grammar)
print("Best Spearman Correlation:", best_correlation)
print("Significance (p-value):", best_p_value)
if best_p_value < 0.05:
    print("The correlation is statistically significant.")
else:
    print("The correlation is not statistically significant.")
```

```
Optimized Grammar: {'<Start>': [['<Begrüßung>', '<Bedarf>', '<Abschluss>',
'<Verabschiedung>', 1.0]], '<Begrüßung>': [['KBG', 'VBG', 1.0]], '<Bedarf>':
[['<BedarfSegment>', '<Bedarf>', 0.8], ['<BedarfSegment>', 0.2]],
'<BedarfSegment>': [['KBBd', 'VBBd', 0.4], ['KBBd', 'VBA', 0.3], ['KBA', 'VBA',
0.3]], '<Abschluss>': [['KAA', 'VAA', 0.6], ['VAA', 'KAA', 0.4]],
'<Verabschiedung>': [['KAV', 'VAV', 0.7], ['VAV', 'KAV', 0.3]]}
Best Spearman Correlation: 0.9692307692307693
Significance (p-value): 3.778488151361357e-06
The correlation is statistically significant.
```

[ ]:

```python

```