ars20gramopt

November 11, 2024

Algorithmisch Rekursive Sequenzanalyse 2.0

Beispeil für die Optimierung der Grammatik

Paul Koop November 2024 post@paul-koop.org

0.1 Problemstellung und Lösungsansatz

0.1.1 Problem

Gegeben ist eine empirische Sequenz von Terminalzeichen, die aus einem bestimmten Sprachmodell gewonnen wurde. Die Aufgabe besteht darin, eine kontextfreie probabilistische Grammatik zu erstellen, die Sequenzen generieren kann, die möglichst stark mit der empirischen Sequenz übereinstimmen. Ziel ist es, eine Grammatik zu finden, deren Wahrscheinlichkeitsverteilungen so angepasst sind, dass die generierten Sequenzen in ihren relativen Häufigkeiten den in der empirischen Sequenz vorkommenden Terminalzeichen möglichst nahekommen.

0.1.2 Vorgehensweise

1. **Initiale Grammatikdefinition**: Eine erste probabilistische Grammatik wird definiert, die verschiedene mögliche Sequenzen mit unterschiedlichen Wahrscheinlichkeiten generieren kann.
2. **Generierung und Frequenzanalyse**: Mehrere Sequenzen werden basierend auf der Grammatik erzeugt, und die relative Häufigkeit der einzelnen Terminalzeichen wird berechnet.
3. **Korrelationstest**: Die Korrelation zwischen den relativen Häufigkeiten der Terminalzeichen in der generierten und in der empirischen Sequenz wird berechnet. Der Spearman-Rangkorrelationskoeffizient dient dabei als Maß für die Übereinstimmung. Zusätzlich wird ein p-Wert berechnet, um die Signifikanz der Korrelation zu überprüfen.
4. **Optimierung durch Anpassung der Wahrscheinlichkeiten**: Falls die Korrelation niedrig ist, werden die Wahrscheinlichkeiten innerhalb der Grammatik iterativ angepasst, um die generierten Sequenzen besser an die empirische Sequenz anzupassen. Dabei wird sichergestellt, dass die Wahrscheinlichkeiten nach jeder Anpassung auf 1 normiert werden.
5. **Abbruchkriterium**: Der Optimierungsprozess endet, wenn eine signifikant hohe Korrelation erreicht wird oder die maximale Anzahl von Iterationen erreicht ist.

0.1.3 Ergebnis

Nach dem Optimierungsprozess wird die angepasste Grammatik ausgegeben, zusammen mit dem besten Korrelationskoeffizienten und seinem Signifikanzniveau (p-Wert). Eine signifikant hohe Ko-

rrelation bedeutet, dass die Grammatik erfolgreich an die empirische Sequenz angepasst wurde und als Modell für deren Generierung dienen kann.

[4]:
```
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
            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

    # 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.
```

[ ]: