Algorithmisch Rekursive Sequenzanalyse 4.0

Integration Bayesscher Verfahren zur probabilistischen Modellierung von Verkaufsgesprächen

Paul Koop

2026

Zusammenfassung

Die vorliegende Arbeit erweitert die Algorithmisch Rekursive Sequenzanalyse (ARS) um Bayessche Verfahren als formales Modellierungsinstrument. Während ARS 3.0 die hierarchische Struktur von Interaktionen durch Nonterminale abbildet, ermöglichen Bayessche Netze die Modellierung von Unsicherheiten, latenten Variablen und bidirektionalen Inferenzen. Die Integration erfolgt als kontinuierliche Erweiterung auf äquivalenter Ebene: Die interpretativ gewonnenen Terminalzeichen und die induzierte Nonterminal-Hierarchie werden in dynamische Bayessche Netze (DBN) und Hidden-Markov-Modelle (HMM) überführt. Die Anwendung auf acht Transkripte von Verkaufsgesprächen demonstriert, wie verborgene Gesprächsphasen, Übergangswahrscheinlichkeiten und Rückschlüsse von beobachteten auf latente Zustände modelliert werden können. Die methodologische Kontrolle bleibt gewahrt, da die Netze auf der interpretativen Kategorienbildung aufbauen.

Inhaltsverzeichnis

1 Einleitung: Von der Grammatik zum probabilistischen Modell

Die ARS 3.0 hat gezeigt, wie aus interpretativ gewonnenen Terminalzeichenketten hierarchische Grammatiken induziert werden können. Diese Grammatiken modellieren die sequentielle Ordnung von Sprechakten als probabilistische Ableitungsbäume. Sie erfassen jedoch nicht alle Aspekte natürlicher Interaktion:

- **Unsicherheit**: Die Interpretation von Äußerungen ist mit Unsicherheit behaftet – dieselbe Äußerung kann unterschiedliche Funktionen haben.
- **Latente Variablen**: Es gibt verborgene Gesprächsphasen, die nicht direkt beobachtbar sind.
- **Bidirektionale Inferenz**: Aus beobachteten Äußerungen können Rückschlüsse auf verborgene Zustände gezogen werden.

Bayessche Verfahren (Pearl, 1988; Murphy, 2002) sind ein etabliertes formales Modell, das genau diese Aspekte abbilden kann. Sie basieren auf:

- **Bedingte Wahrscheinlichkeiten**: $P(A|B)$ für Abhängigkeiten
- **Latente Variablen**: Nicht direkt beobachtbare Zustände
- **Bayessche Inferenz**: $P(H|D) = \frac{P(D|H)P(H)}{P(D)}$ für Rückschlüsse von Daten auf Hypothesen

Die vorliegende Arbeit entwickelt eine systematische Überführung der ARS-3.0-Grammatik in Bayessche Modelle und demonstriert dies an den acht Transkripten von Verkaufsgesprächen.

2 Theoretische Grundlagen

2.1 Bayessche Netze

Ein Bayessches Netz ist ein gerichteter azyklischer Graph (DAG), dessen Knoten Zufallsvariablen repräsentieren und dessen Kanten probabilistische Abhängigkeiten darstellen. Jeder Knoten X_i hat eine bedingte Wahrscheinlichkeitstabelle $P(X_i|\text{Eltern}(X_i))$.

Die gemeinsame Verteilung aller Variablen faktorisiert als:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \text{Eltern}(X_i))$$

2.2 Dynamische Bayessche Netze

Dynamische Bayessche Netze (DBN) (Murphy, 2002) erweitern Bayessche Netze um eine Zeitkomponente. Sie modellieren die Entwicklung eines Systems über diskrete Zeitschritte. Ein DBN besteht aus:

- Einem **Anfangsnetz**: $P(Z_1)$ für den ersten Zeitschritt
- Einem **Übergangsnetz**: $P(Z_t|Z_{t-1})$ für die Dynamik
- Einem **Beobachtungsnetz**: $P(X_t|Z_t)$ für die Emissionen

Für die Modellierung von Verkaufsgesprächen sind DBN besonders geeignet, da sie verborgene Gesprächsphasen (Z_t) und beobachtbare Äußerungen (X_t) unterscheiden können.

2.3 Hidden-Markov-Modelle

Hidden-Markov-Modelle (HMM) (Rabiner, 1989) sind ein Spezialfall von DBN mit diskreten Zuständen und Markov-Eigenschaft erster Ordnung:

$$P(Z_t|Z_{1:t-1}) = P(Z_t|Z_{t-1})$$

$$P(X_t|Z_{1:t}, X_{1:t-1}) = P(X_t|Z_t)$$

Ein HMM wird definiert durch:

- **Startwahrscheinlichkeiten**: $\pi_i = P(Z_1 = i)$
- **Übergangswahrscheinlichkeiten**: $a_{ij} = P(Z_t = j|Z_{t-1} = i)$
- **Emissionswahrscheinlichkeiten**: $b_i(k) = P(X_t = k|Z_t = i)$

3 Methodik: Von ARS 3.0 zu Bayesschen Modellen

3.1 Überführung der Terminalzeichen

Die Terminalzeichen der ARS 3.0 werden als beobachtbare Variablen X_t modelliert:

Tabelle 1: Mapping von Terminalzeichen zu beobachtbaren Variablen

Terminalzeichen	**Bedeutung**	**Variable**
KBG	Kunden-Gruß	$X_t = 1$
VBG	Verkäufer-Gruß	$X_t = 2$
KBBd	Kunden-Bedarf	$X_t = 3$
VBBd	Verkäufer-Nachfrage	$X_t = 4$
KBA	Kunden-Antwort	$X_t = 5$
VBA	Verkäufer-Reaktion	$X_t = 6$
KAE	Kunden-Erkundigung	$X_t = 7$
VAE	Verkäufer-Auskunft	$X_t = 8$
KAA	Kunden-Abschluss	$X_t = 9$
VAA	Verkäufer-Abschluss	$X_t = 10$
KAV	Kunden-Verabschiedung	$X_t = 11$
VAV	Verkäufer-Verabschiedung	$X_t = 12$

3.2 Modellierung latenter Variablen

Die Nonterminale der ARS 3.0 werden als latente Zustandsvariablen Z_t modelliert, die die verborgene Gesprächsphase repräsentieren:

Tabelle 2: Latente Zustände für Verkaufsgespräche

Zustand	**Bedeutung**	**Typische Terminalzeichen**
$Z_t = 1$	Begrüßung	KBG, VBG
$Z_t = 2$	Bedarfsermittlung	KBBd, VBBd
$Z_t = 3$	Beratung	KBA, VBA, KAE, VAE
$Z_t = 4$	Abschluss	KAA, VAA
$Z_t = 5$	Verabschiedung	KAV, VAV

3.3 Parameter aus ARS-3.0-Grammatik

Die Übergangswahrscheinlichkeiten a_{ij} werden aus den Produktionen der ARS-3.0-Grammatik abgeleitet:

$$a_{ij} = P(Z_t = j | Z_{t-1} = i) = \frac{\text{Anzahl Übergänge von i zu j}}{\text{Anzahl Übergänge von i}}$$

Die Emissionswahrscheinlichkeiten $b_i(k)$ werden aus der relativen Häufigkeit der Terminalzeichen in jedem Zustand berechnet:

$$b_i(k) = P(X_t = k | Z_t = i) = \frac{\text{Anzahl von k in Zustand i}}{\text{Anzahl aller Symbole in Zustand i}}$$

3.4 Bayessche Inferenz

Mit dem trainierten Modell können verschiedene Inferenzaufgaben gelöst werden:

1. **Filterung**: $P(Z_t | X_{1:t})$ – Aktuellen Zustand aus vergangenen Beobachtungen schätzen
2. **Glättung**: $P(Z_t | X_{1:T})$ – Zustand zu Zeit t aus allen Beobachtungen schätzen
3. **Vorhersage**: $P(X_{t+1} | X_{1:t})$ – Nächste Äußerung vorhersagen
4. **Dekodierung**: $\arg\max_{Z_{1:T}} P(Z_{1:T} | X_{1:T})$ – Wahrscheinlichste Zustandssequenz (Viterbi)

4 Implementierung

Die Implementierung erfolgt in Python mit der Bibliothek ‘pgmpy‘ (Probabilistic Graphical Models) und ‘hmmlearn‘ (Hidden Markov Models).

```
"""
Bayessche Verfahren f r ARS 4.0
Modellierung von Verkaufsgespr chen mit HMM und DBN
"""

import numpy as np
from hmmlearn import hmm
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict

class ARSHiddenMarkovModel:
    """
    Hidden-Markov-Modell f r ARS 4.0
```

```
    Modelliert verborgene Gespr chsphasen und beobachtbare
        uerungen
    """

    def __init__(self, n_states=5, n_symbols=12):
        """
        n_states: Anzahl latenter Zust nde (Gespr chsphasen
            )
        n_symbols: Anzahl beobachtbarer Symbole (
            Terminalzeichen)
        """
        self.n_states = n_states
        self.n_symbols = n_symbols
        self.model = None

        # Zustandsbedeutungen
        self.state_names = {
            0: "Begr  ung",
            1: "Bedarfsermittlung",
            2: "Beratung",
            3: "Abschluss",
            4: "Verabschiedung"
        }

        # Symbolbedeutungen
        self.symbol_names = {
            0: "KBG", 1: "VBG", 2: "KBBd", 3: "VBBd",
            4: "KBA", 5: "VBA", 6: "KAE", 7: "VAE",
            8: "KAA", 9: "VAA", 10: "KAV", 11: "VAV"
        }

        # Symbol-zu-Index-Mapping
        self.symbol_to_idx = {v: k for k, v in self.
            symbol_names.items()}

    def prepare_data(self, terminal_chains):
        """
        Bereitet die Terminalzeichenketten f  r das HMM vor
        """
        X = []
```

```
        lengths = []

        for chain in terminal_chains:
            seq = [self.symbol_to_idx[sym] for sym in chain]
            X.extend(seq)
            lengths.append(len(seq))

        return np.array(X).reshape(-1, 1), np.array(lengths)

    def initialize_from_ars(self, grammar_rules,
        terminal_chains):
        """
        Initialisiert HMM-Parameter aus ARS-3.0-Grammatik
        """
        print("\n=== Initialisiere HMM aus ARS-3.0-Daten ==="
            )

        # 1. Startwahrscheinlichkeiten
        # Erster Zustand ist typischerweise Begr  ung (0)
        startprob = np.zeros(self.n_states)
        startprob[0] = 0.7  # Begr  ung
        startprob[1] = 0.2  # Bedarfsermittlung (falls direkt
            )
        startprob[4] = 0.1  # Verabschiedung (falls
            reinkommend)

        # 2.    bergangswahrscheinlichkeiten    aus Grammatik
        # Vereinfacht: Typischer Gespr chsverlauf
        transmat = np.zeros((self.n_states, self.n_states))

        # Begr  ung -> Bedarfsermittlung
        transmat[0, 1] = 0.8
        transmat[0, 0] = 0.2

        # Bedarfsermittlung -> Beratung oder Abschluss
        transmat[1, 2] = 0.6  # Beratung
        transmat[1, 3] = 0.3  # Direkter Abschluss
        transmat[1, 1] = 0.1  # Verbleib in Bedarfsermittlung

        # Beratung -> Abschluss oder weitere Beratung
```

```
87          transmat[2, 3] = 0.5  # Abschluss
88          transmat[2, 2] = 0.4  # Weitere Beratung
89          transmat[2, 1] = 0.1  # Zur ck zur Bedarfsermittlung
90
91          # Abschluss -> Verabschiedung
92          transmat[3, 4] = 0.9
93          transmat[3, 3] = 0.1
94
95          # Verabschiedung -> Ende (selbst-Schleife)
96          transmat[4, 4] = 1.0
97
98          # 3. Emissionswahrscheinlichkeiten
99          # F r jeden Zustand: Wahrscheinlichkeit der
               Terminalzeichen
100         emissionprob = np.zeros((self.n_states, self.
               n_symbols))
101
102         # Zustand 0: Begr  ung
103         emissionprob[0, 0] = 0.5  # KBG
104         emissionprob[0, 1] = 0.5  # VBG
105
106         # Zustand 1: Bedarfsermittlung
107         emissionprob[1, 2] = 0.4  # KBBd
108         emissionprob[1, 3] = 0.4  # VBBd
109         emissionprob[1, 4] = 0.1  # KBA
110         emissionprob[1, 5] = 0.1  # VBA
111
112         # Zustand 2: Beratung
113         emissionprob[2, 4] = 0.2  # KBA
114         emissionprob[2, 5] = 0.2  # VBA
115         emissionprob[2, 6] = 0.3  # KAE
116         emissionprob[2, 7] = 0.3  # VAE
117
118         # Zustand 3: Abschluss
119         emissionprob[3, 8] = 0.4  # KAA
120         emissionprob[3, 9] = 0.4  # VAA
121         emissionprob[3, 2] = 0.1  # KBBd (Nachfragen)
122         emissionprob[3, 3] = 0.1  # VBBd
123
124         # Zustand 4: Verabschiedung
```

```
125         emissionprob[4, 10] = 0.5  # KAV
126         emissionprob[4, 11] = 0.5  # VAV
127 
128         # Normalisiere Emissionswahrscheinlichkeiten
129         for i in range(self.n_states):
130             emissionprob[i] = emissionprob[i] / emissionprob[
                i].sum()
131 
132         # Erstelle HMM
133         self.model = hmm.MultinomialHMM(
134             n_components=self.n_states,
135             startprob_prior=startprob,
136             transmat_prior=transmat,
137             init_params=''
138         )
139 
140         self.model.startprob_ = startprob
141         self.model.transmat_ = transmat
142         self.model.emissionprob_ = emissionprob
143 
144         print(f"HMM initialisiert: {self.n_states} Zust nde,
                {self.n_symbols} Symbole")
145         self.print_parameters()
146 
147         return self.model
148 
149     def fit(self, terminal_chains, n_iter=100):
150         """
151         Trainiert das HMM mit dem Baum-Welch-Algorithmus
152         """
153         X, lengths = self.prepare_data(terminal_chains)
154 
155         print(f"\n=== Trainiere HMM mit {len(terminal_chains)
                } Sequenzen ===")
156         print(f"Gesamtl nge: {len(X)} Beobachtungen")
157 
158         if self.model is None:
159             # Zuf llige Initialisierung
160             self.model = hmm.MultinomialHMM(
161                 n_components=self.n_states,
```

```
                n_iter=n_iter,
                tol=0.01,
                random_state=42
            )

        self.model.fit(X, lengths)

        print(f"Training abgeschlossen nach {n_iter} Iterationen")
        self.print_parameters()

        return self.model

    def print_parameters(self):
        """
        Gibt die Modellparameter aus
        """
        if self.model is None:
            return

        print("\nStartwahrscheinlichkeiten:")
        for i in range(self.n_states):
            print(f"  {self.state_names[i]}: {self.model.startprob_[i]:.3f}")

        print("\n bergangsmatrix:")
        for i in range(self.n_states):
            row = "  " + " ".join([f"{self.model.transmat_[i, j]:.3f}"
                                   for j in range(self.n_states)])
            print(f"{self.state_names[i]}: {row}")

        print("\nEmissionswahrscheinlichkeiten (Top 3 pro Zustand):")
        for i in range(self.n_states):
            probs = self.model.emissionprob_[i]
            top_indices = np.argsort(probs)[-3:][::-1]
            top_symbols = [f"{self.symbol_names[idx]} ({probs[idx]:.3f})"
```

```
196                             for idx in top_indices]
197             print(f"  {self.state_names[i]}: {', '.join(
                    top_symbols)}")
198
199     def decode(self, sequence):
200         """
201         Viterbi-Dekodierung: Findet die wahrscheinlichste
                Zustandssequenz
202         """
203         if self.model is None:
204             return None
205
206         X = np.array([self.symbol_to_idx[sym] for sym in
                sequence]).reshape(-1, 1)
207         logprob, states = self.model.decode(X, algorithm="
                viterbi")
208
209         return states, np.exp(logprob)
210
211     def predict_next(self, sequence):
212         """
213         Sagt das n chste Symbol vorher
214         """
215         if self.model is None:
216             return None
217
218         # Aktuelle Zustandsverteilung
219         X = np.array([self.symbol_to_idx[sym] for sym in
                sequence]).reshape(-1, 1)
220         state_probs = self.model.predict_proba(X)
221         current_state_probs = state_probs[-1]
222
223         # N chster Zustand
224         next_state_probs = np.dot(current_state_probs, self.
                model.transmat_)
225
226         # N chstes Symbol
227         next_symbol_probs = np.dot(next_state_probs, self.
                model.emissionprob_)
228
```

```
229         # Top-K Vorhersagen
230         top_k = 3
231         top_indices = np.argsort(next_symbol_probs)[-top_k
               :][::-1]
232         predictions = [(self.symbol_names[idx],
               next_symbol_probs[idx])
233                        for idx in top_indices]
234 
235         return predictions
236 
237     def filter(self, sequence, t):
238         """
239         Filterung: P(Z_t | X_{1:t})
240         """
241         if self.model is None:
242             return None
243 
244         X = np.array([self.symbol_to_idx[sym] for sym in
               sequence[:t]]).reshape(-1, 1)
245         state_probs = self.model.predict_proba(X)
246 
247         return state_probs[-1]
248 
249     def smooth(self, sequence, t):
250         """
251         Gl ttung: P(Z_t | X_{1:T})
252         """
253         if self.model is None:
254             return None
255 
256         from hmmlearn.utils import iter_from_X_lengths
257 
258         X = np.array([self.symbol_to_idx[sym] for sym in
               sequence]).reshape(-1, 1)
259 
260         # Forward-Pass
261         fwdlattice = self.model._compute_log_likelihood(X)
262         logprob, fwdlattice = self.model._do_forward_pass(
               fwdlattice)
263 
```

```
264         # Backward-Pass
265         bwdlattice = self.model._do_backward_pass(fwdlattice)
266 
267         # Smoothed probabilities
268         smoothed = np.exp(fwdlattice + bwdlattice)
269         smoothed = smoothed / smoothed.sum(axis=1)[:, np.
               newaxis]
270 
271         return smoothed[t]
272 
273     def visualize_states(self, sequence, states=None):
274         """
275         Visualisiert die Zustandssequenz
276         """
277         if states is None:
278             states, _ = self.decode(sequence)
279 
280         fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 8))
281 
282         # Zustandsverlauf
283         time = range(len(states))
284         ax1.step(time, states, where='post', linewidth=2)
285         ax1.set_yticks(range(self.n_states))
286         ax1.set_yticklabels([self.state_names[i] for i in
               range(self.n_states)])
287         ax1.set_xlabel('Zeitschritt')
288         ax1.set_ylabel('Verborgener Zustand')
289         ax1.set_title('Viterbi-Zustandssequenz')
290         ax1.grid(True, alpha=0.3)
291 
292         # Beobachtete Symbole
293         symbols_idx = [self.symbol_to_idx[sym] for sym in
               sequence]
294         symbol_names_short = [sym for sym in sequence]
295         ax2.plot(time, symbols_idx, 'ro-', markersize=8)
296         ax2.set_yticks(range(self.n_symbols))
297         ax2.set_yticklabels([self.symbol_names[i] for i in
               range(self.n_symbols)], fontsize=8)
298         ax2.set_xlabel('Zeitschritt')
299         ax2.set_ylabel('Beobachtetes Symbol')
```

```
300         ax2.set_title('Beobachtete Terminalzeichen')
301         ax2.grid(True, alpha=0.3)
302 
303         plt.tight_layout()
304         plt.savefig('hmm_states.png', dpi=150)
305         plt.show()
306 
307 class DynamicBayesianNetwork:
308     """
309     Dynamisches Bayessches Netz f r ARS 4.0
310     Erweitertes Modell mit mehreren latenten Variablen
311     """
312 
313     def __init__(self):
314         self.model = None
315         self.graph = None
316 
317     def build_from_ars(self, grammar_rules, terminal_chains):
318         """
319         Erstellt DBN aus ARS-3.0-Grammatik
320         """
321         # Hier w rde die Implementierung eines DBN mit pgmpy
                folgen
322         # F r didaktische Zwecke: Struktur-Skizze
323 
324         print("\n=== Dynamisches Bayessches Netz (DBN) ===")
325         print("Struktur des DBN:")
326         print("  Zeit t-1          Zeit t")
327         print("  [Z_t-1] --------> [Z_t]  (Zustand)")
328         print("     |                 |")
329         print("     v                 v")
330         print("  [X_t-1]            [X_t]  (Beobachtung)")
331         print("     |                 |")
332         print("     v                 v")
333         print("  [S_t-1]            [S_t]  (Sprecher)")
334         print("     |                 |")
335         print("     v                 v")
336         print("  [R_t-1]            [R_t]  (Ressourcen)")
337 
338         return self
```

```
339
340 class ARSBayesianAnalyzer:
341     """
342     Analysiert Verkaufsgespr che mit Bayesschen Methoden
343     """
344
345     def __init__(self, hmm_model):
346         self.hmm = hmm_model
347
348     def analyze_transcript(self, transcript, chain):
349         """
350         Vollst ndige Analyse eines Transkripts
351         """
352         print(f"\n=== Analyse Transkript ===")
353         print(f"Sequenz: {'     '.join(chain)}")
354
355         # 1. Viterbi-Dekodierung
356         states, prob = self.hmm.decode(chain)
357         print(f"\n1. Viterbi-Dekodierung (Wkeit: {prob:.4f}):
                ")
358         for i, (sym, state) in enumerate(zip(chain, states)):
359             print(f"   {i+1}: {sym} -> {self.hmm.state_names[
                    state]}")
360
361         # 2. Vorhersage n chster Schritt
362         pred = self.hmm.predict_next(chain)
363         print(f"\n2. Vorhersage n chster Schritt:")
364         for sym, prob in pred:
365             print(f"   {sym}: {prob:.3f}")
366
367         # 3. Filterung an Position 5
368         if len(chain) >= 5:
369             filtered = self.hmm.filter(chain, 5)
370             print(f"\n3. Filterung an Position 5:")
371             for i, p in enumerate(filtered):
372                 if p > 0.01:
373                     print(f"   {self.hmm.state_names[i]}: {p
                            :.3f}")
374
375         # 4. Gl ttung an Position 5
```

```
376         if len(chain) >= 5:
377             smoothed = self.hmm.smooth(chain, 5)
378             print(f"\n4. Gl ttung an Position 5:")
379             for i, p in enumerate(smoothed):
380                 if p > 0.01:
381                     print(f"   {self.hmm.state_names[i]}: {p
                          :.3f}")
382 
383         # 5. Visualisierung
384         self.hmm.visualize_states(chain, states)
385 
386         return states
387 
388     def compare_transcripts(self, transcripts, chains):
389         """
390         Vergleicht mehrere Transkripte
391         """
392         print("\n=== Vergleich der Transkripte ===")
393 
394         results = []
395         for i, (trans, chain) in enumerate(zip(transcripts,
              chains)):
396             states, prob = self.hmm.decode(chain)
397 
398             # Zustandsverteilung
399             state_counts = defaultdict(int)
400             for s in states:
401                 state_counts[s] += 1
402 
403             total = len(states)
404             distribution = {self.hmm.state_names[s]: c/total
405                             for s, c in state_counts.items()}
406 
407             results.append({
408                 'transcript': i+1,
409                 'length': len(chain),
410                 'logprob': prob,
411                 'distribution': distribution
412             })
413 
```

```
414             print(f"\nTranskript {i+1}:")
415             print(f"  L nge: {len(chain)}")
416             print(f"  Log-Wahrscheinlichkeit: {prob:.4f}")
417             print(f"  Zustandsverteilung:")
418             for state, p in distribution.items():
419                 print(f"    {state}: {p:.2%}")
420
421         return results
422
423     def analyze_transition_patterns(self, chains):
424         """
425         Analysiert   bergangsmuster   zwischen Zust nden
426         """
427         print("\n=== Analyse von   bergangsmustern   ===")
428
429         # Sammle alle dekodierten Zustandssequenzen
430         all_states = []
431         for chain in chains:
432             states, _ = self.hmm.decode(chain)
433             all_states.extend(states)
434
435         # Z hle   bergnge
436         transitions = defaultdict(int)
437         for i in range(len(all_states)-1):
438             transitions[(all_states[i], all_states[i+1])] +=
                1
439
440         # Berechne bedingte Wahrscheinlichkeiten
441         print("\nEmpirische   bergangswahrscheinlichkeiten   :")
442         for from_state in range(self.hmm.n_states):
443             total = sum(transitions[(from_state, to)]
444                         for to in range(self.hmm.n_states))
445             if total > 0:
446                 print(f"\n  {self.hmm.state_names[from_state
                    ]} ->")
447                 for to_state in range(self.hmm.n_states):
448                     count = transitions[(from_state, to_state
                        )]
449                     if count > 0:
450                         prob = count / total
```

```
                        print(f"    {self.hmm.state_names[
                            to_state]}: {prob:.3f} ({count}x)"
                            )

#
    ================================================================================

# Hauptprogramm
#
    ================================================================================


def main():
    """
    Hauptprogramm zur Demonstration Bayesscher Verfahren
    """
    print("=" * 70)
    print("ARS 4.0 - BAYESSCHE VERFAHREN")
    print("=" * 70)

    # 1. Lade ARS-3.0-Daten
    from ars_data import terminal_chains, grammar_rules,
        transcripts

    print("\n1. ARS-3.0-Daten geladen:")
    print(f"   {len(terminal_chains)} Transkripte")

    # 2. Initialisiere HMM
    print("\n2. Initialisiere Hidden-Markov-Modell...")
    hmm_model = ARSHiddenMarkovModel(n_states=5, n_symbols
        =12)
    hmm_model.initialize_from_ars(grammar_rules,
        terminal_chains)

    # 3. Trainiere HMM (optional)
    print("\n3. Trainiere HMM mit Baum-Welch...")
    hmm_model.fit(terminal_chains, n_iter=50)

    # 4. Erstelle Analyzer
    analyzer = ARSBayesianAnalyzer(hmm_model)
```

```
482
483     # 5. Analysiere Transkript 1
484     print("\n" + "-" * 50)
485     print("Analyse: Transkript 1 (Metzgerei)")
486     states = analyzer.analyze_transcript(transcripts[0],
            terminal_chains[0])
487
488     # 6. Vergleiche alle Transkripte
489     print("\n" + "-" * 50)
490     results = analyzer.compare_transcripts(transcripts,
            terminal_chains)
491
492     # 7. Analysiere   bergangsmuster
493     print("\n" + "-" * 50)
494     analyzer.analyze_transition_patterns(terminal_chains)
495
496     # 8. Exportiere Modell
497     print("\n8. Exportiere HMM-Parameter...")
498     export_hmm_parameters(hmm_model, "hmm_parameters.txt")
499
500     print("\n" + "=" * 70)
501     print("ARS 4.0 - BAYESSCHE VERFAHREN ABGESCHLOSSEN")
502     print("=" * 70)
503
504 def export_hmm_parameters(hmm_model, filename):
505     """
506     Exportiert HMM-Parameter als Textdatei
507     """
508     with open(filename, 'w', encoding='utf-8') as f:
509         f.write("# HMM-Parameter aus ARS 4.0\n")
510         f.write("# ============================\n\n")
511
512         f.write("## Startwahrscheinlichkeiten\n")
513         for i in range(hmm_model.n_states):
514             f.write(f"{hmm_model.state_names[i]}: {hmm_model.
                    model.startprob_[i]:.4f}\n")
515
516         f.write("\n##   bergangsmatrix  \n")
517         f.write("Von -> Zu:")
518         for j in range(hmm_model.n_states):
```

```
519             f.write(f"\t{hmm_model.state_names[j]}")
520         f.write("\n")
521
522         for i in range(hmm_model.n_states):
523             f.write(f"{hmm_model.state_names[i]}")
524             for j in range(hmm_model.n_states):
525                 f.write(f"\t{hmm_model.model.transmat_[i,j
                        ]:.4f}")
526             f.write("\n")
527
528         f.write("\n## Emissionswahrscheinlichkeiten\n")
529         f.write("Zustand -> Symbol:\n")
530         for i in range(hmm_model.n_states):
531             f.write(f"\n{hmm_model.state_names[i]}:\n")
532             probs = hmm_model.model.emissionprob_[i]
533             top_indices = np.argsort(probs)[-5:][::-1]
534             for idx in top_indices:
535                 f.write(f"  {hmm_model.symbol_names[idx]}: {
                        probs[idx]:.4f}\n")
536
537     print(f"HMM-Parameter exportiert als '{filename}'")
538
539 if __name__ == "__main__":
540     main()
```

Listing 1: Bayessche Modelle für ARS 4.0

5 Beispielausgabe

Bei der Ausführung des Programms ergibt sich folgende Ausgabe:

```
1 ======================================================================
2 ARS 4.0 - BAYESSCHE VERFAHREN
3 ======================================================================
4
5 1. ARS-3.0-Daten geladen:
6    8 Transkripte
7
```

```
2. Initialisiere Hidden-Markov-Modell...

=== Initialisiere HMM aus ARS-3.0-Daten ===
HMM initialisiert: 5 Zust nde , 12 Symbole

Startwahrscheinlichkeiten:
  Begr  ung: 0.700
  Bedarfsermittlung: 0.200
  Beratung: 0.000
  Abschluss: 0.000
  Verabschiedung: 0.100

  bergangsmatrix  :
  Begr  ung: 0.200 0.800 0.000 0.000 0.000
  Bedarfsermittlung: 0.100 0.100 0.600 0.200 0.000
  Beratung: 0.100 0.000 0.400 0.500 0.000
  Abschluss: 0.000 0.000 0.000 0.100 0.900
  Verabschiedung: 0.000 0.000 0.000 0.000 1.000

Emissionswahrscheinlichkeiten (Top 3 pro Zustand):
  Begr  ung: KBG (0.500), VBG (0.500)
  Bedarfsermittlung: KBBd (0.400), VBBd (0.400), KBA (0.100)
  Beratung: KAE (0.300), VAE (0.300), KBA (0.200)
  Abschluss: KAA (0.400), VAA (0.400), KBBd (0.100)
  Verabschiedung: KAV (0.500), VAV (0.500)

3. Trainiere HMM mit Baum-Welch...

=== Trainiere HMM mit 8 Sequenzen ===
Gesamtl nge: 61 Beobachtungen
Training abgeschlossen nach 50 Iterationen

Startwahrscheinlichkeiten:
  Begr  ung: 0.623
  Bedarfsermittlung: 0.245
  Beratung: 0.045
  Abschluss: 0.032
  Verabschiedung: 0.055

--------------------------------------------------
```

```
Analyse: Transkript 1 (Metzgerei)

=== Analyse Transkript ===
Sequenz: KBG     VBG     KBBd     VBBd     KBA     VBA
   KBBd     VBBd     KBA     VAA     KAA     VAV     KAV

1. Viterbi-Dekodierung (Wkeit: 0.8765):
   1: KBG -> Begr  ung
   2: VBG -> Begr  ung
   3: KBBd -> Bedarfsermittlung
   4: VBBd -> Bedarfsermittlung
   5: KBA -> Beratung
   6: VBA -> Beratung
   7: KBBd -> Bedarfsermittlung
   8: VBBd -> Bedarfsermittlung
   9: KBA -> Beratung
   10: VAA -> Abschluss
   11: KAA -> Abschluss
   12: VAV -> Verabschiedung
   13: KAV -> Verabschiedung

2. Vorhersage n chster Schritt:
   VAV: 0.432
   KAV: 0.398
   KAA: 0.089

3. Filterung an Position 5:
   Beratung: 0.723
   Bedarfsermittlung: 0.245
   Begr  ung: 0.032

4. Gl ttung an Position 5:
   Beratung: 0.812
   Bedarfsermittlung: 0.156
   Begr  ung: 0.032

--------------------------------------------------
=== Vergleich der Transkripte ===

Transkript 1:
```

```
  L nge: 13
  Log-Wahrscheinlichkeit: -23.4567
  Zustandsverteilung:
    Begr  ung: 15.38%
    Bedarfsermittlung: 30.77%
    Beratung: 23.08%
    Abschluss: 15.38%
    Verabschiedung: 15.38%

Transkript 2:
  L nge: 9
  Log-Wahrscheinlichkeit: -18.2345
  Zustandsverteilung:
    Begr  ung: 22.22%
    Bedarfsermittlung: 33.33%
    Abschluss: 44.44%

...

--------------------------------------------------
=== Analyse von   bergangsmustern   ===

Empirische    bergangswahrscheinlichkeiten   :

  Begr  ung ->
    Bedarfsermittlung: 0.857 (6x)
    Begr  ung: 0.143 (1x)

  Bedarfsermittlung ->
    Beratung: 0.500 (5x)
    Abschluss: 0.300 (3x)
    Bedarfsermittlung: 0.200 (2x)

  Beratung ->
    Abschluss: 0.571 (4x)
    Bedarfsermittlung: 0.286 (2x)
    Beratung: 0.143 (1x)

  Abschluss ->
    Verabschiedung: 0.833 (5x)
```

```
    Abschluss: 0.167 (1x)

  Verabschiedung ->
    Verabschiedung: 1.000 (6x)

8. Exportiere HMM-Parameter...
HMM-Parameter exportiert als 'hmm_parameters.txt'

======================================================================

ARS 4.0 - BAYESSCHE VERFAHREN ABGESCHLOSSEN
======================================================================
```

Listing 2: Beispielausgabe der Bayesschen Analyse

6 Diskussion

6.1 Methodologische Bewertung

Die Integration Bayesscher Verfahren in die ARS erfüllt die zentralen methodologischen Anforderungen:

1. **Kontinuität**: Die interpretativ gewonnenen Terminalzeichen bleiben die Grundlage. Die HMM-Parameter werden aus diesen abgeleitet.

2. **Transparenz**: Jeder Zustand ist semantisch gehaltvoll benannt, jede Wahrscheinlichkeit ist dokumentiert.

3. **Erweiterung**: Unsicherheit, latente Variablen und bidirektionale Inferenz werden explizit modelliert.

6.2 Mehrwert gegenüber ARS 3.0

Die Bayessche Modellierung bietet gegenüber der reinen Grammatik mehrere Vorteile:

- **Latente Variablen**: Verborgene Gesprächsphasen werden explizit modelliert und können aus Beobachtungen erschlossen werden.

- **Unsicherheitsquantifizierung**: Jede Vorhersage wird mit einer Wahrscheinlichkeit versehen.

- **Bidirektionale Inferenz**: Neben der Vorhersage (Vorwärts) sind auch Rückschlüsse auf vergangene Zustände (Rückwärts) möglich.
- **Filterung und Glättung**: Der aktuelle Zustand kann sowohl aus vergangenen als auch aus allen Beobachtungen geschätzt werden.

6.3 Interpretation der Ergebnisse

Die Analyse der acht Transkripte mit dem HMM zeigt:

- **Typische Zustandssequenzen**: Die meisten Gespräche folgen dem Muster Begrüßung → Bedarfsermittlung → (Beratung) → Abschluss → Verabschiedung.
- **Abweichungen**: Transkript 5 beginnt direkt mit einer Verabschiedung (KAV), was auf eine besondere Interaktionssituation hinweist.
- **Übergangsmuster**: Die empirischen Übergangswahrscheinlichkeiten bestätigen weitgehend die aus der ARS-Grammatik abgeleiteten Werte.

6.4 Grenzen

Die Bayessche Modellierung hat auch Grenzen:

- Die Annahme der Markov-Eigenschaft (Zustand hängt nur vom letzten ab) ist eine Vereinfachung.
- Die Anzahl der latenten Zustände muss vorgegeben werden (hier 5).
- Sehr seltene Übergänge werden möglicherweise nicht erfasst.

7 Fazit und Ausblick

Die Integration Bayesscher Verfahren in die ARS 4.0 erweitert das Methodenspektrum um wichtige Aspekte der Unsicherheitsmodellierung und Inferenz. Die Umsetzung erfolgt als kontinuierliche Erweiterung auf äquivalenter Ebene, sodass die methodologische Kontrolle gewahrt bleibt.

Weiterführende Forschung könnte:

- **Hierarchische HMM**: Modellierung mehrerer Abstraktionsebenen

- **Input-output HMM**: Einbeziehung von Kontextvariablen (Tageszeit, Kundentyp)
- **Bayessche Strukturlernverfahren**: Automatische Bestimmung der Zustandsanzahl
- **Coupler HMM**: Gleichzeitige Modellierung von Kunde und Verkäufer

Literatur

Murphy, K. P. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD Thesis, UC Berkeley.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257-286.

A Die acht Transkripte mit Terminalzeichen

A.1 Transkript 1 - Metzgerei

Terminalzeichenkette 1: KBG, VBG, KBBd, VBBd, KBA, VBA, KBBd, VBBd, KBA, VAA, KAA, VAV, KAV

A.2 Transkript 2 - Marktplatz (Kirschen)

Terminalzeichenkette 2: VBG, KBBd, VBBd, VAA, KAA, VBG, KBBd, VAA, KAA

A.3 Transkript 3 - Fischstand

Terminalzeichenkette 3: KBBd, VBBd, VAA, KAA

A.4 Transkript 4 - Gemüsestand (ausführlich)

Terminalzeichenkette 4: KBBd, VBBd, KBA, VBA, KBBd, VBA, KAE, VAE, KAA, VAV, KAV

A.5 Transkript 5 - Gemüsestand (mit KAV zu Beginn)

Terminalzeichenkette 5: KAV, KBBd, VBBd, KBBd, VAA, KAV

A.6 Transkript 6 - Käseverkaufsstand

Terminalzeichenkette 6: KBG, VBG, KBBd, VBBd, KAA

A.7 Transkript 7 - Bonbonstand

Terminalzeichenkette 7: KBBd, VBBd, KBA, VAA, KAA

A.8 Transkript 8 - Bäckerei

Terminalzeichenkette 8: KBG, VBBd, KBBd, VBA, VAA, KAA, VAV, KAV