Dokumentation der Code-Entwicklung
Masterarbeit

Sven Burkhardt

11. April 2025

1 Übersicht

Dieses Dokument enthält eine Dokumentation der am 11. April 2025 erstellten oder aktualisierten Code-Komponenten für die Masterarbeit. Diese Module unterstützen den Datenverarbeitungsprozess für historische Dokumente und automatisieren die Extraktion und Anreicherung von Metadaten.

2 Module

2.1 Dokumenttyp-Erkennung (type_matcher.py)

Das Modul `type_matcher.py` ermöglicht die automatische Erkennung und Zuordnung von Dokumenttypen anhand von Dateinamen und/oder XML-Metadaten:

Dokumenttyp-Erkennung

```python
def get_document_type(filename: str, xml_path: Optional[str] = None, debug: bool = False) -> str:
    match = re.match(r"(\d{7})_Akte_.*?p(?:age)?(\d+)", filename, re.IGNORECASE)
    if not match:
        if debug:
            print(f"[DEBUG] Kein Match f r Dateiname: {filename}")
        return ""

    transkribus_id = match.group(1)
    page_number = match.group(2).zfill(3)

    row = type_df[
        (type_df["Transkribus-ID"] == transkribus_id) &
        (type_df["csv_page_number"] == page_number)
    ]

    if not row.empty:
        doc_type = row.iloc[0].get("Dokumententyp", "").strip()
        if debug:
            print(f"[DEBUG] Typ aus CSV: {doc_type} f r ID {transkribus_id}, Seite {page_number}")
        return doc_type

    # Fallback auf XML-Metadaten, falls keine  bereinstimmung   in der CSV gefunden wurde
    if xml_path:
        try:
            tree = ET.parse(xml_path)
            root = tree.getroot()
            for elem in root.iter():
                if elem.tag.endswith("CSVData"):
                    typ = elem.findtext("Dokumententyp", default="").strip()
                    if typ:
                        if debug:
                            print(f"[DEBUG] Fallback-Typ aus XML: {typ}")
                        return typ
        except Exception as e:
            if debug:
                print(f"[DEBUG] Fehler beim XML-Fallback: {e}")

    return ""
```

2.2 Ortserkennung (place_matcher.py)

Das Modul `place_matcher.py` implementiert einen Fuzzy-Matching-Algorithmus zur Erkennung und Standardisierung von Ortsangaben in historischen Doku-

menten:

Ortserkennung mit Fuzzy-Matching

```python
class PlaceMatcher:
    def __init__(self, csv_path, threshold=90):
        self.threshold = threshold
        try:
            self.places_df = pd.read_csv(csv_path, sep=";")
            self.known_name_map = self._build_known_place_map()
        except Exception as e:
            logging.error(f"Fehler beim Laden der Ortsdaten aus {csv_path}: {e}")
            self.places_df = pd.DataFrame()
            self.known_name_map = {}

    def match_place(self, input_place: str):
        """Fuzzy-Matching gegen alle bekannten Namen & Alternativnamen"""
        if not input_place or not input_place.strip():
            return None

        if not self.known_name_map:
            logging.warning("Keine bekannten Orte zum Abgleich verf gbar.")
            return None

        try:
            # Verschiedene Fuzzy-Matching-Methoden kombinieren f r bessere Ergebnisse
            match, score, _ = process.extractOne(
                input_place.strip(),
                list(self.known_name_map.keys()),
                scorer=fuzz.token_sort_ratio
            )

            # Verschiedene Vertrauensstufen zur ckgeben
            confidence = "low"
            if score >= self.threshold:
                confidence = "high"
            elif score >= 75:  # Mittlere Vertrauensstufe
                confidence = "medium"

            if score >= 75:  # Wir akzeptieren auch mittlere Matches
                return {
                    "matched_name": match,
                    "score": score,
                    "confidence": confidence,
                    "data": self.known_name_map[match]
                }
        except Exception as e:
            logging.error(f"Fehler beim Matching des Ortes '{input_place}': {e}")

        return None
```

2.3 Rollenerkennung (Assigned_Roles_Module.py)

Das Modul `Assigned_Roles_Module.py` erkennt Rollen und zugehörige Organisationen für identifizierte Personen:

Rollenerkennung für Personen

```python
# Ground Truth Mapping laut CSV (nur deutsche Rollenbezeichnungen)
ROLE_MAPPINGS_DE = {
    "ehrenpr sident": "Ehrenpr sident",
    "ehrenmitglied": "Ehrenmitglied",
    "vorstand": "Vorstand",
    "schriftf hrer": "Schriftf hrer",
    "kassierer": "Kassierer",
    "sachwalter": "Sachwalter, Notenwart",
    "notenwart": "Sachwalter, Notenwart",
    "zweiter vorstand": "ZweiterVorstand",
    "dirigent": "Dirigent",
    "chorleiter": "Chorleiter",
    "ehrenf hrer": "Ehrenf hrer",
}

# Regex zur Rollenerkennung (Textrollen)
POSSIBLE_ROLES = list(set(ROLE_MAPPINGS_DE.keys()) | {
    "vereinsf hrer", "leiter", "obmann", "pr sident"  # zus tzliche Rollen
})

ROLE_ORG_REGEX = re.compile(
    r"(?P<n>[A-Z      ][a-z      ]+(?:\s+[A-Z      ][a-z      ]+)?)\s*,?\s*(?P<role>" +
    "|".join(POSSIBLE_ROLES) + r")\s*(des|der|vom)?\s*(?P<organisation>[A-Z      ][\w\s\-]+)?",
    re.IGNORECASE | re.UNICODE
)

def assign_roles_to_known_persons(persons: List[Dict[str, str]], full_text: str) -> List[Dict[str, str]]:
    """
    Reiche Rollen und Organisationen f r bekannte Personen anhand des Kontexts im Transkripttext an.
    """
    for match in ROLE_ORG_REGEX.finditer(full_text):
        name = match.group("name")
        raw_role = match.group("role")
        organisation = match.group("organisation") or ""

        name_parts = name.strip().split(" ")
        if len(name_parts) >= 2:
            forename_candidate = " ".join(name_parts[:-1])
            familyname_candidate = name_parts[-1]

            for person in persons:
                if (person.get("familyname") == familyname_candidate and
                    forename_candidate in person.get("forename", "")):
                    person["role"] = raw_role  # Original aus Text
                    person["role_schema"] = map_role_to_schema_entry(raw_role)
                    person["associated_organisation"] = organisation.strip()
    return persons
```

2.4 LLM-Anreicherung (llm_enricher.py)

Das Modul `llm_enricher.py` implementiert eine API-basierte Anreicherung historischer Dokumente mit Hilfe von Large Language Models:

LLM-basierte Datenanreicherung

```python
def enrich_document_with_llm(json_data: dict, client: openai.OpenAI, model="gpt-4", temperature=0.0) -> Dict:
    prompt = f"""
    Temperatur: 0,4
    Du bekommst ein vollst ndiges JSON-Dokument aus einem historischen Transkriptionsworkflow.
    Deine Aufgabe ist es, folgende Felder **zu erg nzen oder zu korrigieren**:

    - ‘author‘     Wer hat den Text verfasst? Suche nach Gru formeln
    - ‘recipient‘     An wen ist der Text gerichtet? Analysiere das Adressfeld
    - ‘creation_date‘     Nutze Datumsangaben im Text
    - ‘creation_place‘     Oft steht der Ort vor dem Datum
    - ‘content_tags_in_german‘     Themen oder Gef hle im Text
    - ‘mentioned_persons‘, ‘mentioned_organizations‘, ‘mentioned_places‘     Dubletten entfernen

           Besondere Regeln:
    - **  Laufenburg   (Baden) Rhina"** oder  hnliche Kombinationen sind **in der Regel ein Ortsname**
    - **   Mnnerchor    Murg"** oder  hnliche  Begriffe sind **in der Regel eine Organisation**

    Wenn ein Feld **nicht eindeutig bestimmbar ist**, verwende ‘"[...]"‘.
    """

    response = client.chat.completions.create(
        model=model,
        temperature=temperature,
        messages=[{"role": "user", "content": prompt}]
    )

    output = response.choices[0].message.content
    input_tokens = response.usage.prompt_tokens
    output_tokens = response.usage.completion_tokens

    try:
        enriched_data = json.loads(output)
    except Exception as e:
        print("Fehler beim Parsen der LLM-Antwort:", e)
        enriched_data = json_data

    enriched_data["llm_metadata"] = {
        "input_tokens": input_tokens,
        "output_tokens": output_tokens,
        "cost_usd": round((input_tokens / 1000 * INPUT_COST_PER_1K) +
                          (output_tokens / 1000 * OUTPUT_COST_PER_1K), 4),
        "model": model
    }

    return enriched_data
```

3 Zusammenfassung

Die am 11. April 2025 implementierten oder aktualisierten Module bilden zusammen eine Pipeline zur Verarbeitung und Anreicherung historischer Dokumente aus dem Transkribus-System:

1. **Dokumenttyperkennung**: Automatische Erkennung des Dokumenttyps (Brief, Postkarte, etc.) auf Basis von Dateinamen und Metadaten
2. **Ortserkennung**: Fuzzy-Matching von Ortsnamen gegen eine Ground-Truth-Datenbasis zur Standardisierung
3. **Rollenerkennung**: Erkennung und Standardisierung von Personenrollen in Vereinen/Organisationen
4. **LLM-Anreicherung**: Nutzung von Large Language Models zur automatischen Anreicherung der JSON-Daten mit bisher unerkannten Metadaten

Die Module sind Teil eines größeren Systems zur Erfassung und semantischen Anreicherung historischer Dokumente und unterstützen die digitale Erschließung der Sammlung des Männerchors Murg.