DHBW Lörrach
Duale Hochschule
Baden-Württemberg

Grundlagen KI und Maschinelles Lernen

Ugurtan Can Cetin, Patrick Furtwängler, Niclas Gugel,
Marvin Obert, Mikhail Sen

Prüfungsaufgabe 1

a) Programmieren Sie eine Python-Funktion für das Ranking von Art Stories im Word2vec Embedding, die einer Anfrage durch einen Besucher Ihrer Kunst-Website am ähnlichsten sind.
b) Beurteilen Sie den Einfluss der Erweiterung auf das Doc2Vec Embedding auf die Ergebnisqualität.
c) Programmieren Sie einen Empfehlungsdienst für Art Stories durch Clustering mit Top2Vec.

Prüfungsaufgabe 1 - a

```python
def rank_art_stories_python_function(user_query_string):
    user_query_words = text_to_word_list(user_query_string)
    user_query_vector = {}
    for word in allWordsRanking:
        user_query_vector[word] = user_query_words.count(word)
    dataframe_blueprint = {
        "Query" : [],
        "Title" : [],
        "Cosine Similarity": [],
        "Euclidian Distance": [],
        "Similarity Value": [],
    }

    dataframe = pd.DataFrame(dataframe_blueprint)
    alpha = 1e-10 # 1 * 10^-10 = 0.0000000001
    for vector in vectors:
        similarity = cosine_similarity([list(user_query_vector.values())], [list(vector.values())])[0][0]
        distance = euclidean_distances([list(user_query_vector.values())], [list(vector.values())])[0][0]
        dict = {"Query": user_query_string, "Title": titles[vectors.index(vector)], "Cosine Similarity": similarity,
                "Euclidian Distance": distance, "Similarity Value": similarity + 1 / (distance + alpha)}
        dataframe = dataframe.append(dict, ignore_index = True)
    return dataframe.sort_values(by="Similarity Value", ascending=False)
```

	Query	Title	Cosine Similarity	Euclidian Distance	Similarity Value
0	Paris	My Paris: Patrick Roger, master chocolatier	0.289430	13.564660	0.363151
1	Paris	Beyond the waves: Alex Ayed’s journey	0.159561	37.456642	0.186258
2	Paris	Painter Sean Landers unleashes his critters in Paris	0.043806	68.447060	0.058416
4	Paris	Hans Ulrich Obrist remembers Agnès Varda	0.019309	103.561576	0.028966
3	Paris	Artist-led initiatives are transforming Accra into a world-class cultural hub	0.000000	114.686529	0.008719

Prüfungsaufgabe 1 - b

```python
def doc2Vec_model(user_query_string):

    # Read the given documents into array. Here the bag of words from earlier won't be used, and we will make sure to use the convenience of provided methods.
    documents = []
    with open('./Aufgabe 1/art_stories_examples.csv', 'r', encoding="utf-8") as file:
        for line in file.readlines()[1:]:
            lineSecs = line.split(";")
            documents.append(clean_text(lineSecs[1]) + clean_text(lineSecs[2]) + clean_text(lineSecs[3]))


    # Tokenize and tag the documents
    tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()), tags=[str(i)]) for i, _d in enumerate(documents)]

    # Create instance of Doc2Vec and provide tagged data to train it
    model = Doc2Vec(vector_size=20, min_count=1, epochs=100)
    model.build_vocab(tagged_data)
    model.train(tagged_data, total_examples=model.corpus_count, epochs=model.epochs)

    # Create Vector for user query, with infer_vector method, because the words in the user query can not assure if they are in the vocabulary
    user_vector = model.infer_vector(word_tokenize(user_query_string.lower()))

    # Go through every document and generate Ranking. The ranking will be a list of tuples, where the first value is the document and the second value is the rank.
    ranks = []
    for i in range(len(documents)):
        doc_vector = model.docvecs[str(i)]
        rank = cosine_similarity([user_vector.tolist()], [doc_vector.tolist()])[0][0]
        ranks.append((documents[i], rank))

    # Sort by rank
    ranks.sort(key=lambda x: x[1])

    return ranks
```

Prüfungsaufgabe 1 - c

```python
def recommend_art_stories_python_function(identifier_of_visited_art_stories_list):
    """Based of the created clusters of the top2vec we search in
    which cluster the visited art stories where assigned and therefore
    recommend the top 3 similar documents (top 1 of each cluster)"""

    # Read the documents into a dataframe with pandas
    dataframe = pd.read_csv('./Aufgabe 1/art_stories_examples.csv', delimiter=';')

    # create the document for the Top2Vec clustering
    documents = dataframe['Text'].tolist()
    # in this case we multiply the amount of documents by 10. The library we use needs enough documents to train the model with
    # this changes the value in this case, but because we didnt have a big enough dataset, this is what we used.
    # when testing and having a big enough data set please comment the line
    documents = documents * 10

    # Here we give Top2Vec our documents, and it will create a model out of every document.
    # We use the universal-sentence-encoder, because those are pretrained and it will run faster.
    # there is another model 'doc2vec' but that would take longer to generate
    top2vec_model = Top2Vec(documents, embedding_model='universal-sentence-encoder', min_count=1, speed='fast-learn', workers=4)

    # after the model is trained, we will get the topic for each document.
    # this method wont only return the topic. it will return a list of items in this order:
    #  a list of numbers which represent the topic numbers of the document
    #  a list of scores, which represent the similarity of the document to the topics
    #  for each topic the top 50 words
    #  a list of word scores for each topic and document, which is represented by cosine similarity
    visited_vector_topics = top2vec_model.get_documents_topics(identifier_of_visited_art_stories_list)

    # get the list of topic indices
    visited_topic_indices = visited_vector_topics[0]

    recommendation = []
    # go through every topic
    for visited_topic_index in visited_topic_indices:
        # get the most similar document in the topic
        similar_documents_based_on_topic = top2vec_model.search_documents_by_topic(visited_topic_index, num_docs=1)
        # add the document, which is the most similar in the topic to the recommendations
        recommendation.append(similar_documents_based_on_topic[0])
    return recommendation
```

Prüfungsaufgabe 2

Empfehlung von Kunstwerken mit Matrix-Faktorisierung im Content- und Collaborative Filtering und Kundenprofilanreicherung

a) Übertragen Sie die Tracking Daten (Sample_Tracking_Data_for_Recommender_System_ABCD.csv) in das User-Item Interaction Matrix Template (User-Item_Interaction_-Matrix_Template_With_Randomized_Values.csv).
   - Tipp: Definieren Sie den zeitlichen Rahmen für die Verweildauer auf einem Kunstwerk so, dass diese als implizites Feedback interpretiert werden kann.
   - Tipp: Um bessere Ergebnisse zu erzielen, können z.B. weitere Tracking-Profile aus synthetischen Daten erzeugt werden.

b) Optimieren Sie relevante Parameter (z.B. von bm25_weight, AlternatingLeastSquares) in der Code-Vorlage so, dass beim Aufruf der Funktionen user_recommend() und artwork_recommend() sinnvolle Empfehlungen von Kunstwerken ausgesprochen werden. Begründen Sie jeweils Ihre Entscheidungen.

c) Implementieren Sie ein Python-Programm zur textbasierten Anreicherung der Kundenprofile (s. Spalte BESCHREIBUNG) in einem Customer Relationship Management System einschl. einfachem Flask Web-App User Interface.

Prüfungsaufgabe 3

Erstellung eines Wissensgraphen für Kunst (Aufgabe 1) und Abfrage mithilfe natürlicher Sprache(Aufgabe 2)

1. Optimieren Sie das Python-Programm Art_Knowledge_Graph.py auf die Verwendung mit dem Datensatz Art_Stories.csv (s. Moodle) für die Erzeugung einens sinnvollen Wissensgraphen.
2. Implementieren Sie eine Python-Funktion zur SPARQL-Abfrage des Wissensgraphen aus *Aufgabe 1* mithilfe natürlicher Sprache. Gehen Sie hierzu wie folgt vor:

   a) Analysieren Sie den generellen Aufbau des Programmcodes QAsparql.
      - Aktualisierter Code s. Moodle (Aus: https://github.com/Sylvia-Liang/QAsparql)
      - Aus: Querying knowledge graphs in natural language, Liang et. al, ETH Zürich, Springer Open, Journal of Big Data.
      - **Evtl. hilfreiche Installationshinweise:**
      - python -m spacy download en_core_web_lg
      - pip install fasttext-wheel
      - Glove-Download nach QAsparql-master\learning\treelstm\data\glove: https://github.com/stanfordnlp/GloVe?tab=readme-ov-file (z.B. Wikipedia 2014 + Gigaword 5)

   b) Implementieren Sie eine Python-Funktion zur Extraktion von RDF-Tripeln *(z.B. als .ttl-File, vgl. dbpedia 3Eng class.ttl)* aus Ihrem in *Aufgabe 1* erzeugten Wissensgraphen.

   c) Erstellen Sie für die 3 im Paper erwähnten Fragetypen List, Count und Boolean jeweils Beispiel-Trainingsdaten
   Hinweis: Die Beispiele können analog zu den Trainingsdaten in QAsparql-master/data/LC-QUAD/train-data.json aufgebaut sein.

   d) Implementieren Sie eine Python-Funktion, die eine Anfrage in natürlicher Sprache in die entsprechende SPARQL-Query basierend auf Ihren in c) generierten Beispiel-Trainingsdaten übersetzt:
      - **Funktionsname:** natural_language_to_sparql_query(natural_language_query_string)
      - **return:** sparql_query_for_knowledge_graph_string

Prüfungsaufgabe 3 - 1

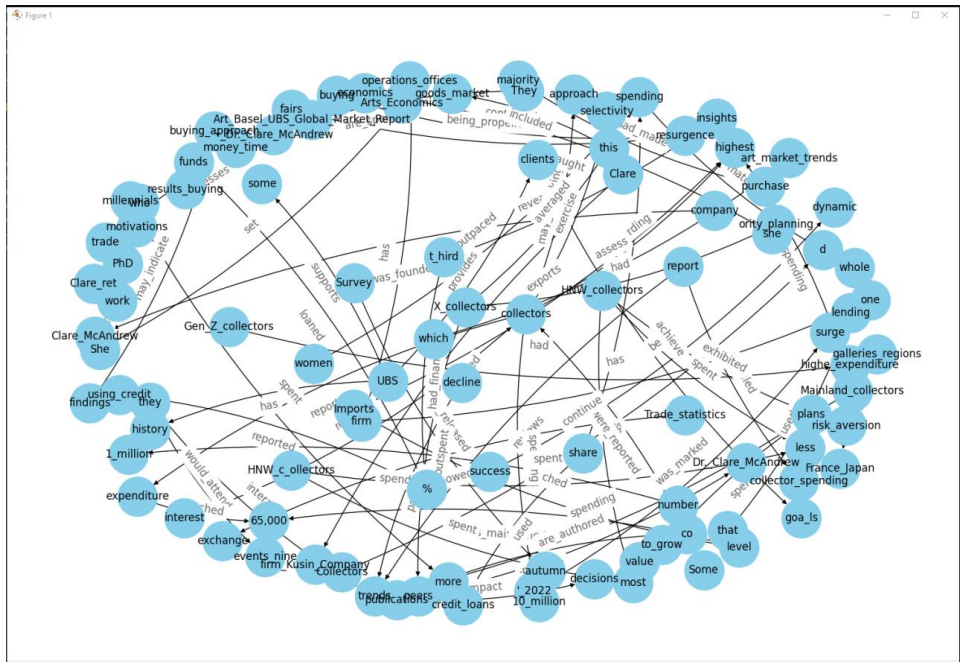
```python
def __init__(self, path, isPDF = False, topic="Art Basel Fair") -> None:
    dataSource = None
    text = ""
    if(isPDF):
        pdfFileObj = open(path, 'rb')
        pdfReader = PyPDF2.PdfReader(pdfFileObj)
        for page in pdfReader.pages:
            text += page.extract_text()
        pdfFileObj.close()
    else:
        textDataFrame = pd.read_csv(path, sep=",")
        textDataFrame['STORY_TEXT'] = textDataFrame['STORY_TEXT'].apply(self.clean_text_from_html_content)
        for id, line in textDataFrame.iterrows():
            text += line[1]
```

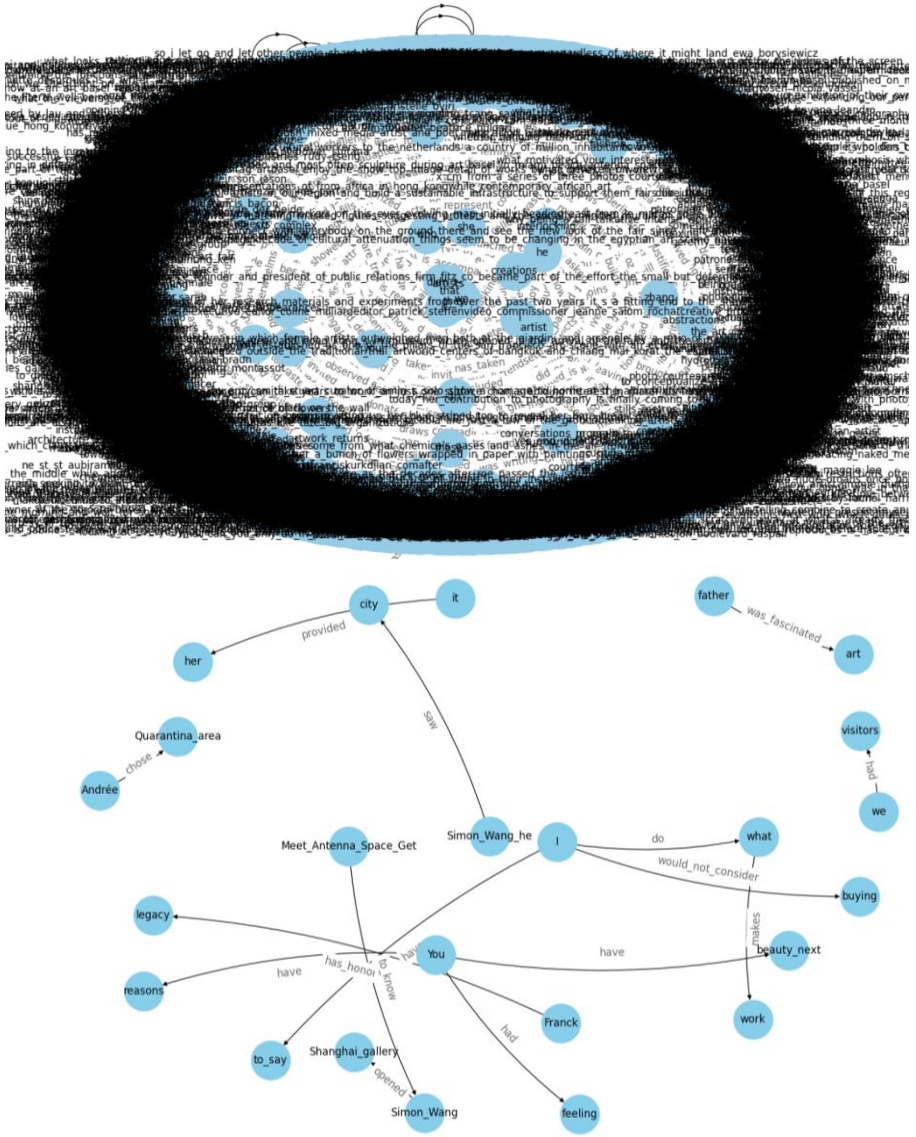
```python
def clean_text_from_html_content(self, string):
    text = re.sub('<.*?>', ' ', string)
    soup = BeautifulSoup(text, 'html.parser')
    text = soup.get_text()
    text = re.sub(r'\s+', ' ', text)
    re.sub(r"[^\w\s\.\!\?\(\)]", "", text)

    with open(self.filepath, "w", encoding="utf-8") as t_file:
        t_file.write(text)
        t_file.close()
    return text
```

DHBW Lörrach
Duale Hochschule
Baden-Württemberg

Prüfungsaufgabe 3 - 1





Prüfungsaufgabe 3 – 2 a

1. Frageanalyse: Fragen werden syntaktisch durchsucht und tokenized.
2. Fragetypklassifizierung: Es wird identifiziert ob es ein List, Boolean oder Count Frage ist.
3. Phrase-Mapping: Die finalen Queries werden aufgebaut. Mithilfe von verschiedene Phrase-Mapping-Systeme genutzt
4. Query Erzeugung: Zuvor wurden die Fragetypen identifiziert, nun werden die Variablen Tripel erzeugt.
5. Query Ranking: Die erzeugten Queries werden nun geranked mithilfe von tree-LSTM
6. Evaluierung: Mithilfe von den Datensätzen werden Ergebnisse bewertet

Prüfungsaufgabe 3 – 2 b

```python
# This method will take the generated graph and save it as a .ttl file
# For the conveniance of the devs (us :) ), a library called RDFLib is used
# the graph is generated using the networkx library
def generate_ttl_file(self, GRAPH, output_file="output.ttl"):
    graph = rdflib.Graph()

    for edge in GRAPH.edges(data=True):
        subj = rdflib.URIRef(edge[0])
        pred = rdflib.URIRef(edge[2]['relation'])
        obj = rdflib.URIRef(edge[1])
        graph.add((subj, pred, obj))
    graph.serialize(destination=output_file, format="turtle")
```

```
<Andrée> <chose> <Quarantina_area> .

<Franck> <has_honored> <legacy> .

<I> <do> <what> ;
    <have> <to_say> ;
    <would_not_consider> <buying> .

<Meet_Antenna_Space_Get> <to_know> <Simon_Wang> .

<Simon_Wang_he> <saw> <city> .

<You> <had> <feeling> ;
    <have> <beauty_next>,
        <reasons> .

<father> <was_fascinated> <art> .

<it> <provided> <her> .

<we> <had> <visitors> .

<Simon_Wang> <opened> <Shanghai_gallery> .

<what> <makes> <work> .
```

Prüfungsaufgabe 3 – 2 c
Boolean


```
{
"_id": "66",
"corrected_question": "Is the Mona Lisa housed in the Louvre Museum?",
"intermediary_question": "Is the <Louvre Museum> the <housing location> of the <Mona Lisa>?",
"sparql_query": "ASK WHERE { http://dbpedia.org/resource/Mona_Lisa http://dbpedia.org/property/housedIn http://dbpedia.org/resource/Louvre_Museum }",
"sparql_template_id": 153
},
{
"_id": "67",
"corrected_question": "Did Alexander Graham Bell invent the telephone?",
"intermediary_question": "Is <Alexander Graham Bell> the <inventor> of the <telephone>?",
"sparql_query": "ASK WHERE { http://dbpedia.org/resource/Alexander_Graham_Bell http://dbpedia.org/property/inventor http://dbpedia.org/resource/Telephone }",
"sparql_template_id": 154
},
{
"_id": "68",
"corrected_question": "Is Mount Everest the tallest mountain in the world?",
"intermediary_question": "Is <Mount Everest> the <tallest mountain> in the <world>?",
"sparql_query": "ASK WHERE { http://dbpedia.org/resource/Mount_Everest http://dbpedia.org/property/tallestMountainIn http://dbpedia.org/resource/World }",
"sparql_template_id": 155
},
```

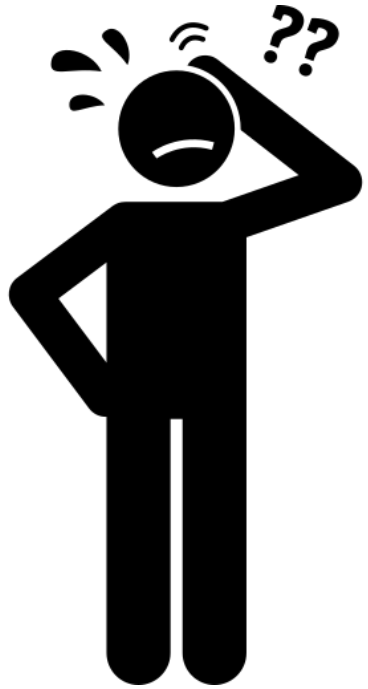
Prüfungsaufgabe 3 – 2 c
List


```
{
    "_id": "2701",
    "corrected_question": "Who directed the movie 'Inception'?",
    "intermediary_question": "Who directed the movie '<Inception>'?",
    "sparql_query": "SELECT DISTINCT ?director WHERE {<http://dbpedia.org/resource/Inception> <http://dbpedia.org/ontology/director> ?director}",
    "sparql_template_id": 315
},
{
    "_id": "2702",
    "corrected_question": "List the prime ministers of the United Kingdom.",
    "intermediary_question": "List the prime ministers of the <United Kingdom>.",
    "sparql_query": "SELECT DISTINCT ?primeMinister WHERE {<http://dbpedia.org/resource/United_Kingdom> <http://dbpedia.org/ontology/leaderName> ?primeMinister}",
    "sparql_template_id": 316
},
```

Prüfungsaufgabe 3 – 2 c
Count


```
{
    "_id": "230",
    "corrected_question": "How many films has Steven Spielberg directed?",
    "intermediary_question": "Count the number of <films> directed by <Steven Spielberg>.",
    "sparql_query": "SELECT DISTINCT COUNT(?film) WHERE { ?director <http://dbpedia.org/property/director> <http://dbpedia.org/resource/Steven_Spielberg> . ?film <http://dbpedia.org/property/directedBy> ?director . }",
    "sparql_template_id": 102
},
{
    "_id": "231",
    "corrected_question": "How many Grammy Awards has Beyoncé won?",
    "intermediary_question": "Count the number of <Grammy Awards> won by <Beyoncé>.",
    "sparql_query": "SELECT DISTINCT COUNT(?award) WHERE { <http://dbpedia.org/resource/Beyoncé> <http://dbpedia.org/property/grammyAwards> ?award . }",
    "sparql_template_id": 102
},
```

Prüfungsaufgabe 3 – 2 d



Prüfungsaufgabe 3 – 2 d

```python
def natural_language_to_sparql(natural_language_query_string):
    """
    Convert a natural language query to a SPARQL query.
    :param natural_language_query: The natural language query.
    :return: The SPARQL query.
    """

    """ PROBLEM:
        The problem is, that in the code, a package which is used tries to reference sklearn.ensemble.forest.
        But that is deprecated. So the code needs to be updated to use the new package. I can not update the code of the package tho.
        And besides that, the API calls are deprecated as well, because the APIs are not responding.
        So I won't be able to test out, which will work and which won't. But I still want to explain the general
        idea what we would go for to solve the task.
    """

    #IDEA

    # After Training the Model, the test Files are the ones, which allow us to test it.
    # There they will first load in the data which have been generated by training, and then they will read the data, that only has the questions in it.
    # For those questions, multiple answers will be generated. The generated queries (which are the answers) will be sorted.
    # The best answer will be the one, which has the highest score.
    # After picking the best answer, the model is rated in the test files according to precision, recall and f1 score.

    # What would we do to generate a query, for given natural language?
    # According to the task we have to do it according to the data created in Task 2 c)
    # We would use the data to train a model, which is able to generate queries for given natural language.

    # So we need to preprocess the dataset, and generate linked_answers.json.
    # After that we will generate the output file with the queries, which are generated according to extracted entitites and relations.
    # all of that is to train the model with the newly generated data.
    # after splitting the data into training, trial and test data, the dependency tree will be generated.
    # Part of that is the needed input and output files, which are used to train the model.
    # Then the phrase mappings are generated, which are mapping of entities and relationships.

    # Afterwords the queries can be generated, and tested.
    # In this case, the testing will of the queries afterwords and calculating recall, precision and f1 score is unnecessary.
    # Because the goal is only to generate the Sparql Query not test if the results match up with pre calculated ones, which we don't have in this case.

    # Do calculate a Sparql Query, after traingin and preparing the model, we need a linker, a parser, a question_type_classifier, and a knowledge base.
    # The Knowledge Base and Parser, can be extracted by the linked_test.json.
    # The question type classifier is a result, that has been put into output directory before.
    # The linker is a result of the generated phrase mappings.
    # In the given code, not only given Knowledge base is used, but also a online knowledge base. The problem is that these weren't able to be tested because the API calls are not working in this project.

    # from the linker we extract entities and ontologies. If there is a (or multiple) answer(s) in the question answer pair, we will iterate through the entities.
    # In those entities, we will generate the subsets, and same think for the ontologies. Then the Entities and Ontologies will be combined into one list of tuples.
    # Now the questions will be passed into the generate_query method, with the entity and ontology.
    # THere the question type is classified.
    # Afterwards the graph is being build of the knowledge base, and a minimal subgraph is being found with given entities and relations. Then out of that graph, a query builder is run over.
    # It will generate possible queries (or in the code referred to as valid walks, which are the paths in the graph).
    # the rank method is then used to rank those walks.
    # In the case of a good rank, the queries will be returned.
    # The where questions are filtered out, and the one with the highest calculated score/confidence is returned.



    return natural_language_query_string
```

Grundidee: Nutzen was gegeben ist

1. Modelle nutzen und Entitäten und Ontologies extrahieren
2. Question Type klassizifieren -> Keyword Analysis (Who, Where, How many, Is, …)
3. Queries aufbauen je nach Question Type

Daher dass der Code bei uns nicht lief, und auch auf einem Clean Environment nicht lief, konnten wir diese Aufgabe nicht wie erwartet bearbeiten.

Prüfungsaufgabe 4

Entwicklung eines Kunst-Chatbots

1. Erstellen Sie repräsentative Anfragen *(engl. Prompts)* an einen Kunst-Chatbot während einer Kunstmesse mit den entsprechenden User Stories in tabellarischer Form aus Sicht von
   - Kunst-Sammler (z.B. Privatier, Museum) und
   - Kunst-Aussteller (z.B. Gallerie, Künstler).

Prompt	As a <role>	I want <goal>	so that <benefit>	Acceptance criteria (Conditions of satisfaction)
...	...	...	...	...

2. Optimieren Sie den Kunst-Chatbot im Programmcode Chatbot_LangChain_RAG.py auf die beiden Metriken
   - Sensibleness und Specifity Average (SSA) und
   - Perplexität.
3. Identifizieren Sie zusätzliche Optimierungspotentiale des Kunst-Chatbots und beurteilen diese im Hinblick auf den Einsatz bei einer realen Kunstmesse.

Repräsentative Anfragen

Prompt	As a <role>	I want <goal>	so that <benefit>	Acceptance Criteria
Could you provide me information about Art from Africa	As an art collector (private collector)	I want to receive information about the material and technique of an artwork and of course the culturel aspect of it	So that I can somehow connect with the artwork and decide to buy recommended ones or not	The chatbot provides accurate information about the material and technique of the artwork and the cultural background.
Can you provide me some other Artworks related to France, more likely Paris	As an art collector (museum in france)	I want to be able to ask the chatbot for recommendations on similar artists or artworks so that I can find new art pieces to put into the museum	To expand my knowledge of various art styles and artists	The chatbot provides relevant recommendations based on the art collector's interests.
Hey I make art about mostly landscapes like Bob Ross, can you inspire me with some for new ideas	As an art maker (artist)	I want to get inspired by new art or art stories	So that I can draw something new or even out of my normal habitat	The chatbot should in this case provide logical recommandation about landscapes...

Sensibleness and Specifity Average (SSA)

- Erinnerung: Wie sinnvoll und spezifisch sind die Antworten?
- Rogue als verwendete Metrik (Recall-Oriented Understudy for Gisting Evaluation)
  - Extraktion von n-Grammen aus automatisch generierter und Referenz-Zusammenfassung
  - Zählen der Übereinstimmungen zwischen den n-Grammen
  - Berechnung von Recall und Precision
  - Berechnung des F-Measures als gewichteter Durchschnitt von Recall und Precision

Perplexity

- Erinnerung: Ein niedriger Perplexitätswert bedeutet, dass die Vorhersagegenauigkeit des nächsten Tokens hoch ist
- Verwendung des vorab trainierten gpt-2-Modells
- Laden des Tokenizers für das Modell
- Laden des Modells selbst
- Tokenisierung und Codierung der Antwort als "input IDs"
- Analyse der Antwort durch das Modell
- Extrahierung des Verlusts
- Berechnung der Perplexität aus dem Verlust (als Exponentialfunktion)

SSA und Perplexity Beispiel

Ask your question

Can you provide me some other Artworks related to France, more likely Paris

User

Can you provide me some other Artworks related to France, more likely Paris

SSA: 0.4989313879070265

Perplexity: 33.415706634521484

Optimierungspotentiale des Kunst-Chatbots

- Rogue, BLEU oder andere Metriken ersetzen durch selbsterstellte Metrik zur Berechnung von SSA
- 4 Antworttypen
  - Extraktiv
    - Antwort ist eine Teilzeichenfolge aus dem Kontext
  - Abstrakt
    - Antwort ist in freier Form geschrieben und nicht unbedingt im Kontext
  - Ja-nein
  - Multiple choice
- Selbsterstelle Metrik optimieren für die Antworttypen des Kunst-Chatbots

Prüfungsaufgabe 5

Aufgabenteil 1: Messung der Ähnlichkeit von Künstlern

a) Beschreiben Sie die im Programmcode *artists_similarity_cnn.py* beschriebene Vorgehensweise zur Messung der Ähnlichkeit von Künstlern.

b) Clustern Sie Künstler in sinnvolle Gruppen.

Tip: Verwenden Sie hierfür die Projektion der Künstler in das Koordinatensystem aus dem Diagramm *„Artists Projected on First Two PCs.“*

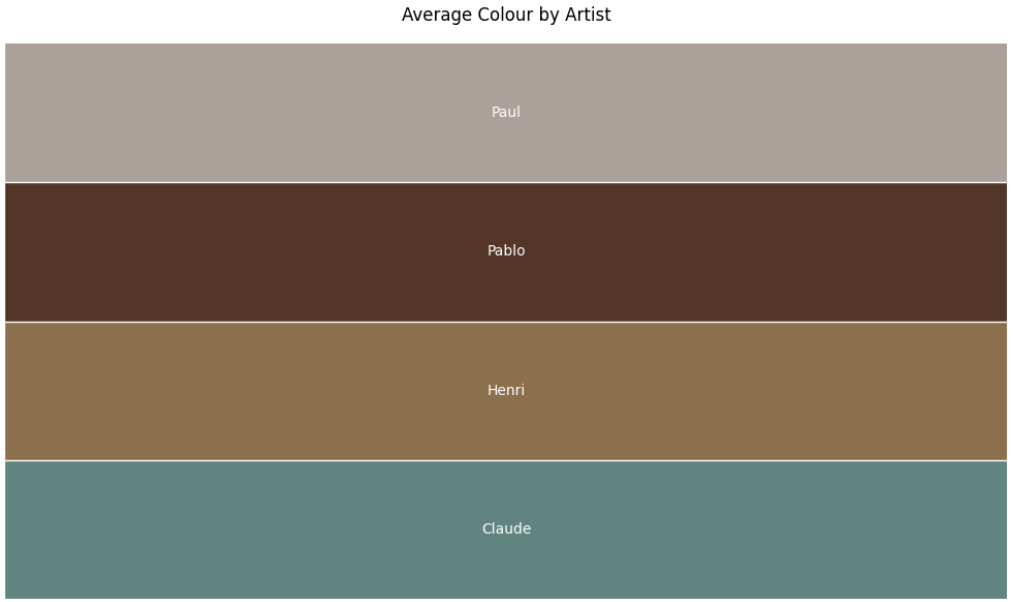
c) Beurteilen Sie die Aussagekraft der im Programmcode beschriebenen Ähnlichkeitsmessung von Künstlern.

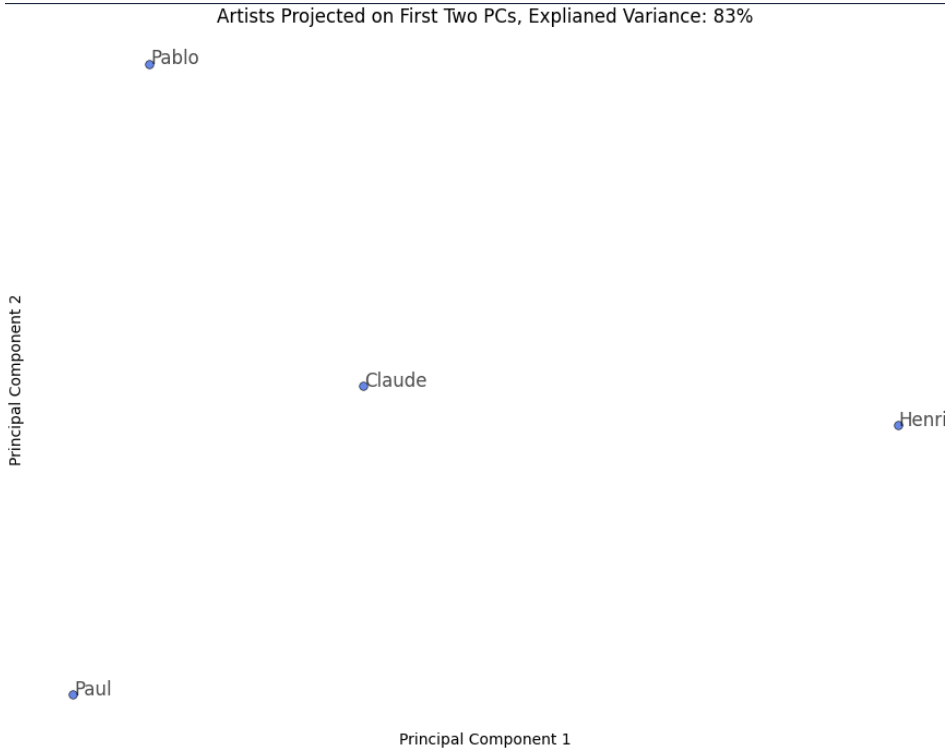
Aufgabenteil 2: Generierung von Bildbeschreibungen

- Passen Sie die Parameter im Programm *Image_Caption_Generator_Transformer.py* so an, dass sinnvolle textbasierte Bildbeschreibungen *(engl.: image captions)* für Kunstwerke erzeugt werden.

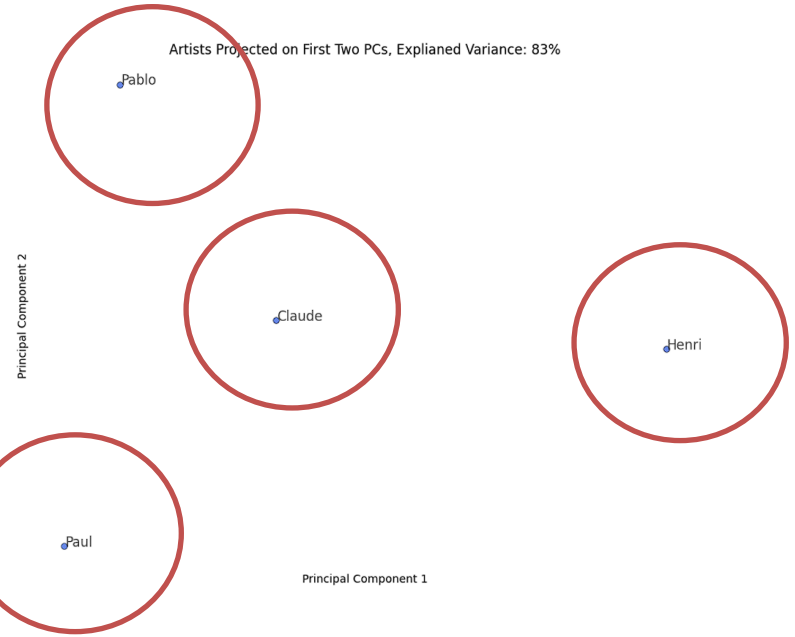
  Hinweis: Sinnvoll bedeutet insb., dass die Bildbeschreibungen potentiell für weitere Anwendungsfälle wie z.B. textbasiertes Clustern eingesetzt werden können.

Prüfungsaufgabe 5 - 1





Prüfungsaufgabe 5




```python
def dbscan_cluster(primary_components):
    dbscan = DBSCAN(eps=2, min_samples=1)
    clusters = dbscan.fit_predict(primary_components)
    return clusters
```

- Unbekannte Clusteranzahl
- Aufgrund ähnlicher Eigenschaften von Epochen schwer linear zu trennen

Prüfungsaufgabe 6

1. Trainieren Sie die Q-Tabelle für die in der Übersicht dargestellte Momentaufnahme mithilfe der Belohnungsmatrix (Reward_Matrix_Show_Snapshot.xlsx) und geben ein Beispiel für einen idealen Pfad an.
2. Programmieren Sie eine Python-Funktion, die für einen gegebenen Zustand innerhalb der drei Serviceanfrage-Phasen während des Messeverlaufs die optimale Strategie zurückgibt:
   - def optimal_strategy_function(current_timestep, current_phase_number, current_state_number, current_reward_matrix_dataframe)
   - current_timestep: Integer increment
   - current_phase_number in [1,2,3]
   - current_state_number in [0,...,30]
   - current_reward_matrix_dataframe als 31x31 Dataframe
   - return updated_timestep, updated_phase_number, next_state_number, updated_reward_matrix_dataframe
3. Erstellen Sie eine einfache Visualisierung der resultierenden Bewegungsabläufe des Roboters in der Maze-Darstellung für jeweils eine Beispiel-Sequenz in den drei Phasen.
4. Beurteilen Sie, inwiefern der Einsatz von Dataset Aggregation zur Effizienzsteigerung Ihres Roboters bei einer realen Kunstmesse beitragen kann.

Prüfungsaufgabe 6 - 1

```python
def read_xslx_into_two_dimensional_list(self, file_path):
    result = []
    workbook = openpyxl.load_workbook(file_path)
    sheet = workbook.active
    for row in sheet.iter_rows(values_only=True):
        result.append(list(row))
    result = [inner[2:] for inner in result[2:]]
    return result
```

```python
elif i == 7:
    possible_keys = dict[i].keys()
    if 12 in possible_keys:
        q_table[i]["L"] = dict[i][12]
        state_table[i]["L"] = 12
    if 13 in possible_keys:
        q_table[i]["R"] = dict[i][13]
        state_table[i]["R"] = 13
    if 0 in possible_keys:
        q_table[i]["U"] = dict[i][0]
        state_table[i]["U"] = 0
    if 24 in possible_keys:
        q_table[i]["D"] = dict[i][24]
        state_table[i]["D"] = 24
    continue
```

```python
def create_q_table_and_state_table(self, reward_matrix):
    dict = {}

    for i, inner_list in enumerate(reward_matrix):
        dict[i] = {}
        for j, value in enumerate(inner_list):
            if value != -1:
                dict[i][j] =  value
    q_table = {}
    state_table = {}

    for i in dict.keys():
        q_table[i] = {}
        state_table[i] = {}
        for j in dict[i].keys():
            if i == 12:···
            elif i == 20:···
            elif i == 7:···
            elif i == 11:···
            if j-1 == i:
                q_table[i]["R"] = dict[i][j]
                state_table[i]["R"] = j
            elif j+1 == i:
                q_table[i]["L"] = dict[i][j]
                state_table[i]["L"] = j
            elif j+1 < i:
                q_table[i]["U"] = dict[i][j]
                state_table[i]["U"] = j
            elif j-1 > i:
                q_table[i]["D"] = dict[i][j]
                state_table[i]["D"] = j

    return q_table, state_table
```

```
> 0: {'R': 0, 'D': 5}
> 1: {'L': 5, 'R': 0, 'D': 5}
> 2: {'L': 0, 'R': 0}
> 3: {'L': 0, 'R': 0, 'D': 0}
> 4: {'L': 0, 'R': 0}
> 5: {'L': 0, 'R': 5, 'D': 0}
> 6: {'L': 0, 'D': 5}
> 7: {'L': 1, 'R': 0, 'U': 5, 'D': 5}
> 8: {'U': 0, 'D': 0}
> 9: {'U': 0, 'D': 0}
> 10: {'U': 0, 'D': 0}
```

```
> 0: {'R': 1, 'D': 7}
> 1: {'L': 0, 'R': 2, 'D': 8}
> 2: {'L': 1, 'R': 3}
> 3: {'L': 2, 'R': 4, 'D': 9}
> 4: {'L': 3, 'R': 5}
> 5: {'L': 4, 'R': 6, 'D': 10}
> 6: {'L': 5, 'D': 11}
> 7: {'L': 12, 'R': 13, 'U': 0, 'D': 24}
> 8: {'U': 1, 'D': 14}
> 9: {'U': 3, 'D': 16}
> 10: {'U': 5, 'D': 18}
```

Ein Idealer Pfad: [12, 7, 0, 1, 8, 14, 21, 25, 24, 25, 21, 14, 15, 16, 22, 27, 28, 29, 30, 11, 6, 11, 20]

Prüfungsaufgabe 6 - 2

1. Es sind keine Serviceanfragen vorhanden. Hinweise:
   - Verwenden Sie die Strategie aus Aufgabe 1 (vgl. Exploitation Prinzip) in Verbindung mit einer zufälligen Komponente bei der Auswahl der nächsten Aktion (Exploration).
   - Die Dauer der Phase 1 beträgt 100 Zeitschritte (z.B. Minuten)

```
# Phase 1
if current_phase_number == 1:  # Walk until EXIT or time expires and than print out the explored (probably bad path)
    state_after_action = current_state_number

    if current_timestep == 100:
        return current_timestep, 2, 12, pd.DataFrame(current_reward_matrix_dataframe)

    current_timestep += 1
    possible_actions = ["L", "R", "U", "D"]
    random_action = self.create_random_action()
    while not self.is_action_valid(current_state_number, random_action):
        possible_actions.remove(random_action)
        random_action = self.create_random_action(possible_actions)
    state_after_action = self.state_table[current_state_number][random_action]

    if state_after_action == exit_state:
        return current_timestep, 2, 12, pd.DataFrame(current_reward_matrix_dataframe)
    return current_timestep, 1, state_after_action, pd.DataFrame(current_reward_matrix_dataframe)
```

Prüfungsaufgabe 6 - 2

2. Die zu Beginn vorhandenen Serviceanfragen (vgl. Reward_Matrix_Show_Snapshot.xlsx) werden sukzessive abgearbeitet, bis alles erledigt ist. Hinweise:
   - Nachdem eine Dienstleistung durch den Roboter erbracht wurde, wird die entsprechende Belohnung in der Belohnungsmatrix auf -1 zurückgesetzt und die Q-Tabelle aus Aufgabe 1 wird neu berechnet.
   - Die Dauer der Phase 2 hängt von der Anzahl Serviceanfragen ab.

```python
def get_next_best_action_with_depth(self, current_state_number, depth):
    """
        This method is conceptual, it didn't work how we wanted it to.
        It is supposed to look for the next best action with a given depth recursively
    """
    if depth == 0:
        return self.get_next_best_action(current_state_number)

    bestAction = None
    bestReqard = float('-inf')

    for action in ["L", "R", "D", "U"]:
        if not self.is_action_valid(current_state_number, action):
            continue
        reward = self.q_table[current_state_number][action]

        if depth > 0:
            next_state = self.state_table[current_state_number][action]
            next_action = self.get_next_best_action_with_depth(next_state, depth-1)
            while not self.is_action_valid(next_state, next_action):
                next_action = self.get_next_best_action_with_depth(next_state, depth-1)
            if next_action is not None:
                reward += 0.9 * self.q_table[next_state][next_action]
        if reward > bestReqard:
            bestReqard = reward
            bestAction = action
    return bestAction
```

```python
elif current_phase_number == 2:
    current_timestep += 1
    tasks = self.count_service_tasks(current_reward_matrix_dataframe)
    if len(tasks) == 0:
        return 0, 3, 12, pd.DataFrame(current_reward_matrix_dataframe)

    best_action, best_reward = self.get_next_best_action(current_state_number)
    while not self.is_action_valid(current_state_number, best_action):
        best_action, best_reward = self.get_next_best_action(current_state_number)

    new_state_number = self.state_table[current_state_number][best_action]
    current_reward_matrix_dataframe = self.remove_reward_from_matrix(current_state_number)
    self.q_table, self.state_table = self.create_q_table_and_state_table(current_reward_matrix_dataframe)

    return current_timestep, 2, new_state_number, pd.DataFrame(current_reward_matrix_dataframe)
```

Prüfungsaufgabe 6 - 2

3. Es sind immer Serviceanfragen vorhanden und neue treffen zufällig ein:
   - Vorgehen analog zu Phase 2 durch jeweiliges Aktualisieren der Belohnungsmatrix und Neuberechnen der Q-Tabelle.
   - Die Dauer der Phase 3 beträgt 10 Zeitschritte

```python
def create_random_service_task(self):
    """
    This is a method for phase 3. Creates a random "service task" and updates the q-table regarding the reward.
    :return: the updated q-table with new "services"
    """
    rand_number_state = randint(0, 30)
    rand_number_reward = random.choice([1, 5, 10])


    for row in range(len(self.reward_matrix)):
        if self.reward_matrix[row][rand_number_state] != -1:
            self.reward_matrix[row][rand_number_state] = rand_number_reward
    self.q_table, self.state_table = self.create_q_table_and_state_table(self.reward_matrix)
```

```python
# Phase 3
else:
    current_timestep += 1
    self.create_random_service_task()
    best_action, best_reward = self.get_next_best_action(current_state_number)
    while not self.is_action_valid(current_state_number, best_action):
        best_action, best_reward = self.get_next_best_action(current_state_number)
    new_state_number = self.state_table[current_state_number][best_action]
    current_reward_matrix_dataframe = self.remove_reward_from_matrix(current_state_number)
    self.q_table, self.state_table = self.create_q_table_and_state_table(current_reward_matrix_dataframe)
    if current_timestep > 33:
        return -1, 3, new_state_number, pd.DataFrame(current_reward_matrix_dataframe)
    return current_timestep, 3, new_state_number, pd.DataFrame(current_reward_matrix_dataframe)
```

Prüfungsaufgabe 6 - 3


```python
def redraw_everything(self, currentStateID, currentPathList, font, text, textRect):
    self.screen.fill((255, 255, 255))
    self.screen.blit(self.floor, (0, 0))
    self.screen.blit(self.robot, self.get_plot_position(currentPathList[currentStateID]))
    text = font.render("Moves: " + str(currentStateID) + " / " + str(len(currentPathList)-1), True, (255, 0, 0))
    self.screen.blit(text, textRect)

def plot(self, pathLists):
    self.floor = pygame.transform.scale(self.floor, (1250, 500))
    self.robot = pygame.transform.scale(self.robot, (50, 50))

    self.screen.blit(self.floor, (0, 0))
    self.screen.blit(self.robot, self.get_plot_position(12))


    stateID = 0

    for pathList in pathLists:
        font = pygame.font.Font(None, 36)
        text = font.render("Moves: " + str(stateID) + " / " + str(len(pathList)-1), True, (255, 0, 0))
        textRect = text.get_rect()
        self.screen.blit(text, textRect)
        print(pathList)
        while self.running:
            for i in pygame.event.get():
                if i.type == pygame.QUIT:
                    self.running = False
                elif i.type == pygame.KEYDOWN:
                    if i.key == pygame.K_ESCAPE:
                        self.running = False
                    elif i.key == pygame.K_RIGHT:
                        stateID = stateID + 1 if stateID < len(pathList)-1 else stateID
                    elif i.key == pygame.K_LEFT:
                        stateID = stateID - 1 if stateID > 0 else stateID
                self.redraw_everything(stateID, pathList, font, text, textRect)

            pygame.display.update()
```

Prüfungsaufgabe 6 - 4

4. Beurteilen Sie, inwiefern der Einsatz von Dataset Aggregation zur Effizienzsteigerung Ihres Roboters bei einer realen Kunstmesse beitragen kann.

- Wenigere Iterationen benötigt, um maximum zu erreichen
- Anpassungsfähig mithilfe von Experten Daten

- Wie stellt man sicher dass die Expertendaten auch gut sind?
- Die Effizienz des Roboters steigt zwar schneller, aber mehr Aufwand für Entwickler/“Experten“

Danke für die Aufmerksamekeit!