DAQ2_Pritz_UE04

April 18, 2023

1 Exercise 4: Unstrukturierten zu strukturierten Text

Aufwand: 5h

Implementieren Sie eines natürlichen Sprachalgorithmus, angewandt auf Webeinträge, Blogeinträge oder ähnliche-Postings (für die deutsche Sprache). Beispiele dafür wären (Instagram, Wikipedia, Blog, etc.) Folgende Funktionalitäten sollen dabei implementiert werden:

1) Als Eingabeparameter soll eine Datei mit >20 Webeinträgen (Instagram Kommentare, Wikipedia Absätze, Blog Einträge, usw.) verwendet werden. Verwenden Sie hierbei das gelernte Wissen aus der Vorlesung und crawlen Sie die Beiträge, achten Sie darauf, dass der in der Vorlesung verwendete Blog nicht verwendet werden darf. Die Texte sollten unbedingt Emojis enthalten (gegebenenfalls können Emojis selbständig ergänzt werden).
2) Textblöcke sollen vorverarbeitet werden, indem Emoticons, Umlaute und sonstige Sonderzeichen entsprechend ersetzt werden. Emoticons sollen mit einem Wort ersetzt werden (z.B.: :-), :) → happy, gluecklich, freudig).
3) Die Wörter eines Satzes sollen in ihren Wortstamm und semantische Bedeutung zurückgeführt werden. Sie können dafür gerne bereits ausimplementierte Stemming-Algorithmen verwenden. Führen Sie dazu eine ausführliche Literaturrecherche durch. (Hinweise: TreeTager1, Porter Stemming Algorithmus2). Speichern Sie die Ergebnisse in einer Textdatei ab.
4) Nachdem die unstrukturierten Texte entsprechend vorverarbeitet wurden, sollen Synonyme identifiziert werden unter Verwendung der Nomenklatur von Openthesaurus3 (Diese Datei wird Ihnen zur Verfügung gestellt – openthesaurus.txt).
5) Auszugeben ist eine csv Datei, welche folgende Spalten enthält: (1) Original-Satz, (2) vorverarbeiteter Satz (Emoticons, Umlaute, Sonderzeichen), (3) gestemmter Satz, (4) Referenzliste mit Synonymen.

1.1 Literaturrecherche

- TreeTagger: TreeTagger annotiert Text mit POS und Lemmatization und ist für viele Sprachen verfügbar, darunter Deutsch und Englisch. Die Methode basiert auf probabilistischen Modellen, genauer gesagt Markov Modelle und Decision Trees.
- Porter-Stemming Algorithmus: Ist regelbasiert und überprüft mittels Bedingungen, ob ein Wort einen gewissen Suffix enthält, der entfernt werden kann, sodass ein noch "valides" aber womöglicherweise nicht existierendes Wort entsteht. Die Regeln basieren allesamt auf der Unterteilung eines Wortes in Vokale und Konsonanten. Es gibt eine Erweiterung des Algorithmus namens "Snowball", die das Wort von rechts nach links durchläuft und den längsten möglichen Suffix sucht und entfernt, aber auch auf die vorangehenden Buchstaben achtet, und somit eventuell weniger lange Suffixe entfernt, um richtige Wörter zu erhalten

bzw. Überschneidungen zu vermeiden (z.B. “cared” würde zu “car” werden, was sich mit “Auto” überschneidet. Bei Snowball bleibt es “care”, was dem Zweck entspricht)

1.2 Implementierung

[1]:
```
import requests
from bs4 import BeautifulSoup
import emoji
import re
from nltk.stem.snowball import GermanStemmer
st = GermanStemmer()
```

[2]:
```
url = r'https://www.reddit.com/r/de/comments/12o4vat/gew%C3%BCrzketchupbrunnen/'

# define headers, so reddit actually returns something
headers = {'User-Agent': 'Mozilla/5.0 (Windows; U; Windows NT 6.1; zh-CN)␣
 ↪AppleWebKit/533+ (KHTML, like Gecko)'}
response = requests.get(url, headers=headers)

soup = BeautifulSoup(response.text, 'html.parser')
```

[3]:
```
def parse_thesaurus(filepath):
    synonym_list = []
    with open(filepath,'r', encoding='utf-8') as file:
        for line in file.readlines():
            if not line.startswith("#"):
                # preproccess each element by removing special chars and stuff␣
 ↪in parentheses
                synonyms = line.split(";")
                synonyms = [re.sub(r"\([^);]+\)", "", synonym) for synonym in␣
 ↪synonyms]
                synonyms = [re.sub("[\W_]", " ", synonym) for synonym in␣
 ↪synonyms]
                synonyms = [re.sub(r"\s+", " ", synonym) for synonym in␣
 ↪synonyms]
                synonyms = [re.sub("ä", "ae", synonym) for synonym in synonyms]
                synonyms = [re.sub("ü", "ue", synonym) for synonym in synonyms]
                synonyms = [re.sub("ö", "oe", synonym) for synonym in synonyms]
                synonyms = [synonym.strip().lower() for synonym in synonyms]
                synonym_list.append(synonyms)
    return synonym_list

# method to easily fetch the synonyms for a given word
def get_synonyms(word, synonym_list):
    res = []
    for synonyms in synonym_list:
        if word in synonyms:
```

```
            for synonym in synonyms:
                if synonym != word:
                    res.append(synonym)
    return res
```

[4]:
```
# Parse the thesaurus file.
thesaurus = parse_thesaurus("openthesaurus.txt")
```

[5]:
```
elements = soup.find_all("div", {"data-testid": "comment"})
print(len(elements))

with open("outfile.csv", "w", encoding="utf-8") as outfile:
    outfile.write("original;preprocessed;stemmed;references\n")
    for element in elements:
        p_res = element.find_all("p")
        for p in p_res:
            # fetch the content of <p> tag
            p = p.get_text()

            #Preprocessing. Removing special characters, umlauts, emojis and␣
 ↪multiple whitespaces.
            preprocessed = emoji.demojize(p.lower(), language="de")
            preprocessed = re.sub("[\W_]", " ", preprocessed)
            preprocessed = re.sub("\s+", " ", preprocessed)
            preprocessed = re.sub("ä", "ae", preprocessed)
            preprocessed = re.sub("ü", "ue", preprocessed)
            preprocessed = re.sub("ö", "oe", preprocessed)

            # stem it using spaCy
            stemmed = " ".join([st.stem(token) for token in preprocessed.
 ↪split(" ")])

            # finally, create a dictionary using the stemmed sentence and the␣
 ↪thesaurus
            references = dict(zip(stemmed.split(" "), [get_synonyms(word,␣
 ↪thesaurus) for word in stemmed.split(" ")]))
            outfile.write(f"{p};{preprocessed};{stemmed};{references}\n")
```

17

1.3 Notizen

Die obige Abfrage returniert leider nur 17 statt den geforderten 20+ Posts. Ich schätze, Reddit rendert gewisse Kommentare erst später durch JavaScript, weshalb sich die Menge hier so drastisch von 200+ auf 17 reduziert. Zeitbedingt habe ich es aber so belassen und hoffe, dass dies kein allzugroßes Problem ist. Zumindest ein Emoji hat es hineingeschafft, womit ich zumindest da Glück hatte :)