Data Structures and Algorithms I

Homework Assignment 3

Wachmann Elias

31. Oktober 2021

Inhaltsverzeichnis

1 Algotithmus in Worten & Pseudo-Code

Aufgabenstellung ist es, aus einer Liste mit beliebig vielen Produkt-Reviews eine abwärtssortierte Liste mit der Häufigkeit von k-hintereinander-stehenden Worten `c` und eine zugehörige Liste dieser Worte `y` zu generieren.

Der Algorithmus in Worten

Dem Algorithmus `count_vectorizer` wird eine list of lists namens `texts` und die Anzahl der aufeinanderfolgenden Wörter `k` übergeben. Zuerst werden für jede Review in `texts` immer k-hintereinander-stehende Wörter in einem String konkateniert und in ein Python `Dict` gespeichert, hierbei wird zuerst versucht den String als Key zu verwenden, um so den count um 1 zu erhöhen. Schlägt dies fehl, so wird ein neuer Key mit count = 1 angelegt. Nun werden die Keys und Values des Dictionaries in die beiden Listen `y` bzw. `c` entpackt. Nun wird `c` mittels `merge_sort` absteigend sortiert, dabei wird die Liste der Worte `y` gleich sortiert, sodass weiterhin jede Stelle in `c` die Anzahl der dazugehörigen Phrase in `y` gibt.

Pseudo-Code

Algorithm 1 `merge_`

1: **function** MERGE__(words, weights, start, k, end)

Algorithm 2 `count_vectorizer`

```
1: function COUNT_VECTORIZER(texts, k=1)
2:     // Input: texts array of arrays with each containing the words from a review
3:     // k integer with the count of consecutive words in a phrase (defaults to 1)
4:     // Output: y array of unique phrases (with k consecutive words)
5:     // c array of occurrence count for corresponding index in y
6:     y ← [ ]
7:     c ← [ ]
8:     vals ← { }
9:     entries ← 0                                          ▷ empty Hashmap
10:    for text ← 0 to length of texts do
11:        counter ← 0
12:        len_ ← length of texts
13:        while counter ≤ (len_ - k) do
14:            for i ← 0 to k do
15:                word ← word + texts[text][counter+i] + " "
16:            word ← strip right space from word
17:            try:
18:                vals[word] ← vals[word] +1
19:            catch KeyNotFoundError:                  ▷ Create new Key in Hashmap
20:                vals[word] ← 1
21:            counter ← counter +1
22:        entries ← entries + counter
23:    y ← convert keys of vals to list
24:    c ← convert values of vals to list
25:    arrlen_ ← length of y
26:    if arrlen_ = entries then
27:        return y,c
28:    return_vals ← MERGE_SORT(y,c,0,arrlen_−1)
29:    return return_vals[0], return_vals[1]          ▷ return_vals is a (y,c) tuple
```

Abbildungsverzeichnis

Tabellenverzeichnis