Matrixmethoden in der Datenanalyse - Project Alice

Ahmad Hizaji, David Krell, Fabian Wetzel, Joshua Enobore, Nureldien Gebril

06. Mai 2024

Aufgabenteil a)

Wir haben uns dazu entschlossen eindeutig identifizierende Wörter einer jeden Datei als Schlüsselwort zu nehmen. Eindeutig identifizierende Wörter sind also Wörter die in der einen Datei vorkommen, aber in der anderen Datei nicht. Wir haben uns dafür entschieden, da auf diesem Weg die Spalten unserer Term-Dokument-Matrix möglichst unterschiedlich sind. Wenn wir zum Beispiel Wörter nehmen die in beiden Dateien möglichst oft vorkommen, dann sind die Spalten sehr ähnlich. Das führt dann im Endeffekt, dazu, dass man sich bei so ca. 50% richtiger Klassifizierungen bewegt. Wir haben in Teil c) mit verschiedenen Mengen von Schlüsselwörtern getestet und dort hat sich dann auch herausgestellt, dass Wörter die sehr oft in beiden Dokumenten vorkommen nicht geeignet sind als Schlüsselwörter. Außerdem hat sich herausgestellt, dass auch Wörter die sehr kurz sind, wie z.B. *"a", "i", "in", ...* nicht als Schlüsselwörter eignen. Das liegt daran, dass wir jedes Vorkommen dieser Wörter zählen und dadurch, dass insbesondere die Vokale sehr häufige Buchstaben sind, wird die Term-Dokument-Matrix mit solchen kurzen Wörtern zu sehr verfälscht.

Mithilfe des folgenden Python Skripts haben wird dann alle Wörter berechnet die einzigartig für die jeweilige Datei sind. Davon haben wir dann die jeweils 10 am meisten vorkommenden gewählt und kommen somit auf eine Menge von insgesamt 20 Schlüsselwörtern.

```
from collections import Counter

def calculate_keywords(file1, file2):
    file1 = open(file1, "r", encoding="UTF-8")
    file2 = open(file2, "r", encoding="UTF-8")

    words1 = file1.read().lower().split()
    words2 = file2.read().lower().split()

    counts1 = Counter(words1)
    counts2 = Counter(words2)

    unique1 = set(words1) - set(words2)
    unique2 = set(words2) - set(words1)

    # We only take the counts for the unique words
    counts_unique1 = []
    counts_unique2 = []

    for word, count in counts1.items():
        if word in unique1:
            counts_unique1.append((word, count))

    for word, count in counts2.items():
        if word in unique2:
            counts_unique2.append((word, count))

    # We are interested in the most common unique words

    most_common_unique1 =
        sorted(counts_unique1, key=lambda x: x[1], reverse=True)
    most_common_unique2 =
        sorted(counts_unique2, key=lambda x: x[1], reverse=True)

    # We are only interested in the words
    unique_words1 = []
    unique_words2 = []

    # We remove the special characters from the words
    special_chars = ["'", '"', '.', ',', '  ', "  "]

    for word, count in most_common_unique1:
        if all(char not in word for char in special_chars):
            unique_words1.append(word)

    for word, count in most_common_unique2:
        if all(char not in word for char in special_chars):
            unique_words2.append(word)

    """ Our set of keywords is now the 10 most common unique
        words from alice.txt and the 10 most common unique words
        from looking-glass.txt"""

    terms = unique_words1[:10] + unique_words2[:10]
    return terms
```

Diese Methode produziert die folgenden Schlüsselwörter.

1. *mock*
2. *turtle*
3. *march*
4. *rabbit*
5. *hatter*
6. *dormouse*
7. *gryphon*
8. *mouse*
9. *hare*
10. *duchess*
11. *humpty*
12. *knight*
13. *dumpty*
14. *looking-glass*
15. *tweedledum*
16. *unicorn*
17. *lion*
18. *gnat*
19. *sheep*
20. *kitten*

Nun haben wir mit dem folgendem Python Code die Term-Dokument-Matrix berechnet:

```
def calculate_term_doc_matrix(terms, files):
    """Calculates the term document matrix for the
       given files and terms."""
    result = []

    for file in files:
        with open(file, 'r', encoding='utf-8') as f:
            file_content = f.read()
            file_content_lower = file_content.lower()
            keyword_counts =
                [file_content_lower.count(term.lower()) for term in terms]
            result.append(keyword_counts)

    return result
```

Hier müssen wir noch die Matrix transponieren und bekommen dann die folgende Term-Dokument Matrix A.

$$A = \begin{pmatrix} 56 & 0 \\ 61 & 0 \\ 35 & 0 \\ 52 & 0 \\ 57 & 1 \\ 40 & 0 \\ 55 & 0 \\ 84 & 2 \\ 32 & 0 \\ 42 & 0 \\ 0 & 56 \\ 0 & 61 \\ 0 & 56 \\ 1 & 23 \\ 0 & 37 \\ 0 & 25 \\ 0 & 21 \\ 0 & 18 \\ 1 & 23 \\ 0 & 26 \end{pmatrix}$$

Aufgabenteil b)

In diesem Teil berechnen wir wie unterschiedlich der Term-Dokument-Vektor eines Strings von den Spalten der Matrix ist. Dazu benutzen wir die folgende Formel aus Eldens Buch:

$$\cos\theta = \frac{x^T y}{||x||_2 \cdot ||y||_2}$$

Zuerst haben wir dafür eine Python Methode geschrieben die den Term-Dokument-Vektor eines Strings berechnet.

```
def calculate_term_doc_vector(terms, string):
    """Calculates the term document vector for a given string and terms."""
    results = []
    input_lower = string.lower()

    for term in terms:
        keyword_count = input_lower.count(term.lower())
        results.append(keyword_count)

    return results
```

Nun haben wir eine Methode geschrieben die eine Liste zurückgibt in der die beiden Winkel enthalten sind.

```
import numpy as np

def calculate_angles(term_doc_matrix, term_doc_vector):
    """Calculates cosine of the angle of the columns of
       the matrix and the term document vector."""
    angles = []

    for i in range(len(term_doc_matrix)):
        # If the term-document-vector is the zero vector
        # then it has angle 0.
        if all(element == 0 for element in term_doc_vector):
            angles.append(0)
            continue

        dot_product = np.dot(term_doc_vector, term_doc_matrix[i])
        norm1 = np.linalg.norm(term_doc_vector, 2)
        norm2 = np.linalg.norm(term_doc_matrix[i], 2)
        angle = dot_product / (norm1 * norm2)
        angles.append(angle)

    return angles
```

Anschließend haben wir eine Methode geschrieben die den Winkel auswählt, der näher an Eins ist. Zurückgegeben wird ein Index der anzeigt zu welcher Datei der String wahrscheinlich gehört.

```
import random

def choose_index(angles):
    """Returns the index of the angle which is closest to 1."""
    closest_index = None
    min_difference = float('inf')

    for i, value in enumerate(angles):
        difference = abs(value - 1)
        if difference < min_difference:
            min_difference = difference
            closest_index = i

    all_equal = all(value == angles[0] for value in angles)

    if all_equal:
        closest_index = random.randint(0, len(angles) - 1)

    return closest_index
```

Insgesamt ergibt sich folgender Algorithmus:

```
def classify(string, terms, matrix):
    term_document_vector = calculate_term_doc_vector(string, terms)
    angles = start_classifying(matrix, term_document_vector)
    file_index = choose_index(angles)
    return file_index
```

Aufgabenteil c)

Nun sollen die Ergebnisse des Algorithmus getestet werden. Dazu habe ich als erstes eine Methode geschrieben, welche die jeweiligen Dateien in x Substrings der Länge n zerlegt. Dabei ist $n \in \{20, 40, 60, \ldots, 400\}$.

```
def calculate_substrings(n, filepath):
    """Calculates substring of length n from the given filepath."""
    with open(filepath, 'r', encoding='utf-8') as file:
        content = file.read().lower()
        substrings = []

        for i in range(len(content) - n + 1):
            substrings.append(content[i:i+n])
        return substrings
```

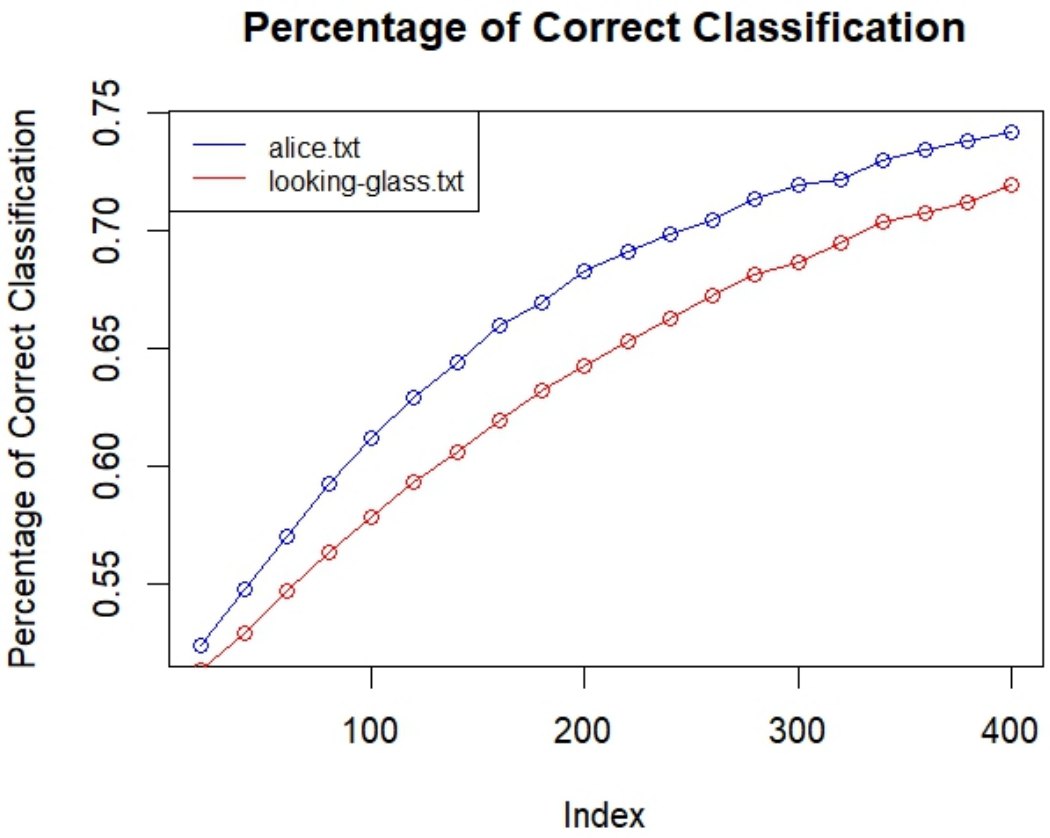
Nun werden diese Substrings genommen und für jeden dieser Substrings wird der Term-Document-Vector berechnet. Dann wird wie in Teil b) mithilfe der calculate_angles Methode der Winkel zwischen dem Term-Document-Vektor der Datei alice.txt und dem Term-Document-Vektor des Substrings berechnet. Das gleiche passiert nochmal für den Term-Document-Vektor der Datei looking-glass.txt. Anschließend wird der Winkel ausgewählt welcher am nächsten an der Eins ist. Wenn der richtige Winkel ausgewählt wurde. Wird ein correct_count erhöht, welcher am Ende widerspiegelt wie viele Substrings korrekt klassifiziert wurden. All dies wird in der test_substrings Methode erledigt. All dies wird in einer Schleife eingebettet so, dass wir für unterschiedliche n testen können.

```
def classifying_rounds(terms, matrix, files):
    """Classifies substrings of different lengths and returns two
       lists which contain the classification percentages of
       each file for different lengths."""
    full_results = []
    file_indicator = 0

    for file in files:
        results = []
        for i in range(20, 401, 20):
            correct_count = 0
            substrings = calculate_substrings(i, file)
            for substring in substrings:
                file_index = classify(substring, terms, matrix)
                if file_index == file_indicator:
                    correct_count += 1
            results.append(correct_count / len(substrings))
        file_indicator += 1
        full_results.append(results)

    return full_results
```

Für die oben genannte Menge an Schlüsselwörtern erhalten wir die unten dargestellten Ergebnisse.

Alice.txt	Looking-Glass.txt
0.5232313734166713	0.5130800240157818
0.547603241777999	0.5130800240157818
0.5699554608830364	0.546517470861471
0.5919403480618654	0.5633771741661866
0.6117660080801373	0.5779669743665183
0.6289956549414663	0.5929036450351263
0.6435313330380869	0.6056000392382868
0.6597181850898319	0.6190537507817961
0.6693684589401406	0.6317809966516625
0.6827389028881566	0.6418863388246262
0.6910103024260552	0.6529817329751083
0.6981760518516467	0.6621474056169712
0.7043811291484293	0.6722233811493011
0.7134485911395562	0.6808782199968092
0.7193180558634449	0.6863378051624465
0.7211775628412271	0.6944870677485484
0.7296621209601419	0.7032644592587136
0.7341848473927869	0.7070712031727712
0.7375720860596651	0.71197180504218191
0.7417505476527189	0.7193913267749898

Mithilfe der Programmiersprache R, habe ich dann diese Daten als Graphen dargestellt.



Wir haben auch mit weitern Mengen an Schlüsselwörtern getestet. Zum Beispiel eine Menge die zum größten Teil Wörter beinhaltet die sehr oft in beiden Texten vorkommen.

1. *turtle*
2. *hatter*
3. *knight*
4. *humpty*
5. *the*
6. *and*
7. *to*
8. *a*
9. *it*
10. *i*
11. *she*
12. *you*
13. *of*
14. *said*
15. *alice*
16. *in*
17. *was*
18. *queen*
19. *he*
20. *red*

Diese Menge an Schlüsselwörtern generiert folgende Term-Document-Matrix A':

$$A' = \begin{pmatrix} 61 & 0 \\ 57 & 1 \\ 0 & 61 \\ 0 & 56 \\ 2309 & 2356 \\ 958 & 1072 \\ 1014 & 1077 \\ 8846 & 9673 \\ 1333 & 1441 \\ 7558 & 8446 \\ 584 & 635 \\ 498 & 702 \\ 620 & 590 \\ 461 & 473 \\ 399 & 468 \\ 2044 & 2234 \\ 375 & 377 \\ 76 & 202 \\ 3789 & 4014 \\ 80 & 150 \end{pmatrix}$$

Wenn wir nun diese Matrix testen kriegen wir folgende Ergebnisse:

Alice.txt	**Looking-Glass.txt**
0.49282316499806406	0.5363177434967469
0.49668768843526123	0.5486379177236131
0.5104708420936152	0.5469096613680096
0.5133289986996099	0.5466947770981699
0.5193701920416183	0.5455910656712393
0.5214139429330529	0.5455353530399814
0.525617249778565	0.5456635562149768
0.5259806771309138	0.5467790000367904
0.5314040090813444	0.5506727337397127
0.5322667737379891	0.5529421144952958
0.5324720283593664	0.5535436065412879
0.5334459740187796	0.5524950001840423
0.5317953903263338	0.5503785601217283
0.5333998282215388	0.5500042953744952
0.5334044557747728	0.5510230383071693
0.5340742177756839	0.5502313933933196
0.5346263650978436	0.5518888193191166
0.5357746791228897	0.5547438085532213
0.5372213596539869	0.5567951567546326
0.5365194243407371	0.5573609422399076

Wir haben nun einmal die Ähnlichkeit zwischen den beiden Spalten der Matrix A' berechnet und das Gleiche haben wird auch für die Matrix A getan.

$$\cos\theta = \frac{a'.1^T a'.2}{||a'.1||_2 \cdot ||a'.2||_2} = 0.9996274835898984$$

$$\cos\theta = \frac{a.1^T a.2}{||a.1||_2 \cdot ||a.2||_2} = 0.013333739688666198$$

Wir sehen, dass die Spalten der Matrix A' sehr nah beieinander liegen. Das macht die Klassifikation sehr schwierig und deswegen kommen wir zu schlechten Ergebnissen bei Wörtern die sehr oft in beiden Dokumenten vorkommen.