

ALG5I - Übung 04

Alen Kocaj

09. November 2017

1 Position Specific Scoring

Implementieren Sie in einer Programmiersprache Ihrer Wahl ein Framework für positionsspezifisches Scoring

- Programmiersprache: Python
- Arbeitsaufwand: 5h
- Git Repository: <https://github.com/rathalos64/algo5>

Diese Aufgabe wurde in Python umgesetzt. Das git repository für die Aufgabe findet sich unter github.com/rathalos64/algo5.

Der geschätzte Arbeitsaufwand beträgt 12h.

Die Struktur der Abgabe besteht aus den folgenden Files.

```
/root
├── main.py
├── pss.py
└── alphabet.py
```

Kurz zur Erklärung der einzelnen Files.

- **main.py** präsentiert das Hauptprogramm, welches die Sequenzen einliest, die diversen Matrizen - PFM (Position Frequency Matrix), PPM (Position Probability Matrix) und PWM (Position Weight Matrix) - berechnet und basierend auf den Targetsequenzen für jede einzelne Sequenz den PS Score berechnet.
- **pss.py** definiert die einzelnen Berechnungsschritte des PSS Algorithmus und stellt passende Methoden bereit.
- **alphabet.py** stellt statische Definitionen für die diverse Sequenzalphabete bereit. Aktuell ist nur das Alphabet für Nukleotide definiert.

Spicy jalapeno bacon ipsum dolor amet brisket burgdoggen turducken ground round turkey landjaeger salami chicken tenderloin bacon. Ground round alcatra pork belly kevin, beef chicken spare ribs salami short ribs shankle beef ribs. Tail landjaeger alcatra doner tenderloin, jowl meatball jerky shankle brisket andouille beef cupim spare ribs. Jerky kevin shank flank doner kielbasa boudin alcatra hamburger cow pastrami. Filet mignon beef capicola picanha short loin ribeye meatball corned beef shankle chuck chicken buffalo.

Landjaeger ribeye fatback, short ribs pork belly short loin doner. Venison shankle swine pork chop tri-tip. Landjaeger kielbasa ball tip t-bone, shoulder jowl tongue hamburger sausage. Sirloin t-bone cow bacon burgdoggen biltong ribeye filet mignon. Tongue swine cow jerky, venison ground round buffalo chuck bacon turducken leberkas ribeye meatball ham hock strip steak. Ham chicken pig shoulder andouille.

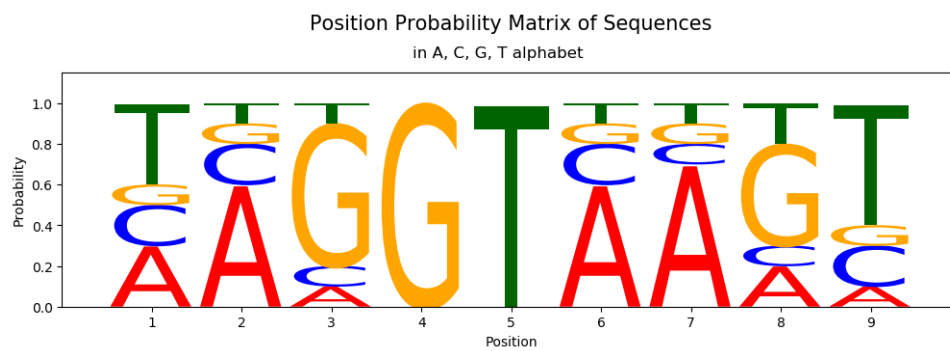


Abbildung 1: Die Position Probability Matrix (PPM) als Sequenzlogo geplottet. Man sieht den Anteil an C,G,T,A basierend auf der Größe des Buchstaben.

```

=====
Position Specific Scoring (PSS)
    by Alen Kocaj

[i] reading source alignments out of 'testing/source'
[i] reading target sequences out of 'testing/target'
=====
[PPM] Position Probability Matrix
      0      1      2      3      4      5      6      7      8
A  0.3  0.6  0.1  1.000000e-10  1.000000e-10  0.6  0.7  0.2  0.1
C  0.2  0.2  0.1  1.000000e-10  1.000000e-10  0.2  0.1  0.1  0.2
G  0.1  0.1  0.7  1.000000e+00  1.000000e-10  0.1  0.1  0.5  0.1
T  0.4  0.1  0.1  1.000000e-10  1.000000e+00  0.1  0.1  0.2  0.6

[i] plot sequence logo of PPM
[i] figure saved at 'pss_ppm.png'
=====
[PPM] Probability Weight Matrix
      0      1      2      3      4      5      6  \
A -1.203973 -0.510826 -2.302585 -23.025851 -23.025851 -0.510826 -0.356675
C -1.609438 -1.609438 -2.302585 -23.025851 -23.025851 -1.609438 -2.302585
G -2.302585 -2.302585 -0.356675  0.000000 -23.025851 -2.302585 -2.302585
T -0.916291 -2.302585 -2.302585 -23.025851  0.000000 -2.302585 -2.302585

      7      8
A -1.609438 -2.302585
C -2.302585 -1.609438
G -0.693147 -2.302585
T -1.609438 -0.510826
=====
[i] scoring targets

# [0th target] #####
> Sequence: GAGGTAAAC
> Log Score by PSS: -7.25646205327

# [1th target] #####
> Sequence: TCCGTAAGT
> Log Score by PSS: -6.89978710933

# [2th target] #####
> Sequence: CAGGTTGGA
> Log Score by PSS: -10.0778409397

# [3th target] #####
> Sequence: ACAGTCAGT
> Log Score by PSS: -8.28608147045

# [4th target] #####
> Sequence: TAGGTCATT
> Log Score by PSS: -5.87016769215

# [5th target] #####
> Sequence: TAGGTACTG

```

```
> Log Score by PSS: -8.50922502177

# [6th target] #####
> Sequence: ATGGTAACT
> Log Score by PSS: -7.54414412572

# [7th target] #####
> Sequence: CAGGTATAC
> Log Score by PSS: -8.50922502177

# [8th target] #####
> Sequence: TGTGTGAGT
> Log Score by PSS: -9.38469375912

# [9th target] #####
> Sequence: AAGGTAAGT
> Log Score by PSS: -4.14294674406

=====
[i] thank you and goodnight
=
```

Listing 1: Der Konsolenoutput des PSS Programms. Als Zielsequenzen zum Berechnen eines Scores wurde dasselbe File verwendet wie zum Einlesen des alignierten Sequenzen.

```
1 #!/usr/bin/env python
2 #
3 # Position Specific Scoring (PSS)
4 #   by Alen Kocaj
5
6 from functools import reduce
7
8 import matplotlib as mpl
9 from matplotlib import pyplot as plt
10 from matplotlib.patches import PathPatch
11
12 from pss import PSS
13 from alphabet import Alphabet
14
15 def main():
16     # source of aligned sequences being used to
17     # build scoring matrix
18     source_scoring_file = "testing/source"
19
20     # list of target sequences which should be scored
21     target_scoring_file = "testing/target"
22
23     # path where sequence logo of PPM should be saved
24     path = "pss_ppm.png"
25
26     # use nucleotide alphabet
27     alphabet = Alphabet.nucleotide()
28     # the random model used
29     weights = {k: v["weight"] for k, v in alphabet.items()}
30
31     # pseudocount for preventing zero probabilities
32     pseudocount = 0.0000000001
33
34     print("
35         ===== "
36     )
37     print("Position Specific Scoring (PSS)")
38     print("\tbody Alen Kocaj")
39     print()
40
41     # read source alignments
42     print(f"[i] reading source alignments out of '{source_scoring_file}'")
43     sources = []
44     with open(source_scoring_file) as sourcefm:
45         sources = sourcefm.read().split("\n")
46
47     expected_length = len(sources[0]) * len(sources)
48     observed_length = reduce(lambda acc, curr: acc + len(curr), sources, 0)
49
50     # validate sources
51     avg_length = int(expected_length / len(sources))
52     if observed_length != expected_length:
53         print(f"[w] not all sequences are of same length. Expected length: {
54             avg_length}")
```

```

52         print("The overhanging onegram will not be considered.")
53
54     # read target sources
55     print(f"[i] reading target sequences out of '{target_scoring_file}'")
56     targets = []
57     with open(target_scoring_file) as targetfm:
58         targets = targetfm.read().split("\n")
59
60     # build matrices
61     pss = PSS(sources, alphabet.keys(), weights, avg_length, pseudocount)
62     pfm = pss.build_frequency_matrix()
63     ppm = pss.build_probability_matrix(pfm)
64
65     print("
===== "
        )
66     print("[PPM] Position Probability Matrix")
67     print(ppm)
68     print()
69     print("[i] plot sequence logo of PPM")
70     plot_ppm_sequence_logo_pd(alphabet, ppm, path, 1.15)
71     print(f"[i] figure saved at '{path}'")
72
73     # build Postion Weight Matrix (PWM)
74     weight_matrix = pss.build_weight_matrix(ppm)
75     print("
===== "
        )
76     print("[PPM] Probability Weight Matrix")
77     print(weight_matrix)
78
79     # score targets
80     print("
===== "
        )
81     print("[i] scoring targets\n")
82     for i, target in enumerate(targets):
83         print(f"# [{i}th target] #####")
84         print(f"> Sequence: {target}")
85         print(f"> Log Score by PSS: " + str(pss.score(target, weight_matrix)
            ))
86         print()
87     print("
===== "
        )
88
89     print("[i] thank you and goodnight")
90
91 # plot_ppm_sequence_logo_pd plots a Position Probability Matrix (PPM)
92 # as a Sequence Logo (https://en.wikipedia.org/wiki/Sequence_logo).
93 # The ppm is given in a Pandas DataFrame format.
94 # The ppm was built based on a given alphabet.
95 # Globscale determines the size of each letter within the sequence logo.
96 #

```

```

97 # Kudos to https://github.com/saketkc
98 # with https://github.com/saketkc/motif-logos-matplotlib for initial code.
99 def plot_ppm_sequence_logo_pd(alphabet, ppm, path, custom_y=-1, globscale
    =1.35):
100     fig, ax = plt.subplots(figsize=(10,3))
101
102     x = 1
103     maxy = 0
104     row_indices = list(ppm.index)
105     for column in ppm:
106
107         y = 0
108         for row in range(0, len(ppm)):
109             score = ppm[column][row]
110             base = row_indices[row]
111
112             letter_at(alphabet, base, globscale, x, y, score, ax)
113             y += score
114             x += 1
115             maxy = max(maxy, y)
116
117     plt.xticks(range(1, x))
118     plt.xlim((0, x))
119
120     if custom_y != -1:
121         maxy = custom_y
122     plt.ylim((0, maxy))
123     plt.tight_layout()
124
125     plt.xlabel("Position")
126     plt.ylabel("Probability")
127     plt.title(f"Position Probability Matrix of Sequences", y=1.15, fontsize
        =15)
128     plt.suptitle("in " + str.join(", ", row_indices) + " alphabet", y=1.03)
129     plt.savefig(f"{path}", bbox_inches="tight")
130
131 # letter_at returns the plot element of a given letter from an alphabet
132 # scaled by globscale and transformed around its x and y axis within the
    plot
133 def letter_at(alphabet, letter, globscale, x, y, yscale=1, ax=None):
134     text = alphabet[letter]["text"]
135
136     t = mpl.transforms.Affine2D().scale(1*globscale, yscale*globscale) + \
137         mpl.transforms.Affine2D().translate(x,y) + ax.transData
138     p = PathPatch(text, lw=0, fc=alphabet[letter]["color"], transform=t)
139     if ax != None:
140         ax.add_artist(p)
141     return p
142
143 if __name__ == "__main__":
144     main()

```

Listing 2: main.py

```

1  #!/usr/bin/env python
2
3  import numpy as np
4  import pandas as pd
5
6  # PSS describes methods for calculating the position specific scoring
7  class PSS():
8      def __init__(self, sources, alphabet, weights, avg_sequence_length,
9                  pseudocount):
10         self.sources = sources
11         self.alphabet = alphabet
12         self.weights = weights
13         self.avg_sequence_length = avg_sequence_length
14         self.pseudocount = pseudocount
15
16         # build_frequency_matrix computes a position frequency matrix (PFM)
17         # out of the given sources. The sequences are based on the given
18         # alphabet.
19         def build_frequency_matrix(self):
20             frequency_matrix = {}
21             for onegram in self.alphabet:
22                 frequency_matrix[onegram] = np.zeros(self.avg_sequence_length)
23
24             for source in self.sources:
25                 for i in range(0, self.avg_sequence_length):
26                     onegram = source[i]
27                     # exclude invalid characters
28                     if onegram not in frequency_matrix.keys():
29                         continue
30
31                     frequency_matrix[onegram][i] += 1
32
33             return pd.DataFrame(list(frequency_matrix.values()), dtype=int,
34                                index=self.alphabet)
35
36         # build_probability_matrix computes a position probability matrix (PPM)
37         # based on the given frequency matrix. Zero values are removed
38         # by adding a given pseudocount prevent zero-frequency problems.
39         def build_probability_matrix(self, pfm):
40             probability_matrix = round(pfm / len(self.sources), 2)
41             return probability_matrix.clip(self.pseudocount)
42
43         # build_weight_matrix constructs the position weight matrix (PWM)
44         def build_weight_matrix(self, ppm):
45             return np.log(ppm)
46
47         # score computes a score for a given target sequence, given
48         # the position weight matrix (PWM)
49         def score(self, target, pwm):
50             score = 0
51             for i, onegram in enumerate(target):
52                 score += pwm[i][onegram]
53             return score

```

Listing 3: pss.py


```
1 #!/usr/bin/env python
2
3 import collections
4 from matplotlib.text import TextPath
5
6 # Alphabet provides information about sequence alphabets
7 # like nucleotides, proteins
8 class Alphabet:
9     # nucleotide returns the alphabet for nucleotides (C, G, T, A)
10    @staticmethod
11    def nucleotide():
12        return collections.OrderedDict(sorted(dict({
13            "T": {
14                "text": TextPath((-0.305, 0), "T", size=1),
15                "color": "darkgreen",
16                "weight": 0.25
17            },
18            "G": {
19                "text": TextPath((-0.384, 0), "G", size=1),
20                "color": "orange",
21                "weight": 0.25
22            },
23            "A": {
24                "text": TextPath((-0.35, 0), "A", size=1),
25                "color": "red",
26                "weight": 0.25
27            },
28            "C": {
29                "text": TextPath((-0.366, 0), "C", size=1),
30                "color": "blue",
31                "weight": 0.25
32            }
33        })).items()))
34
35    # nucleotide returns the alphabet for proteins (amino acids)
36    @staticmethod
37    def protein():
38        # to be implemented
39        return {}
```

Listing 4: alphabet.py