# 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

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 Berechnungssteps 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.

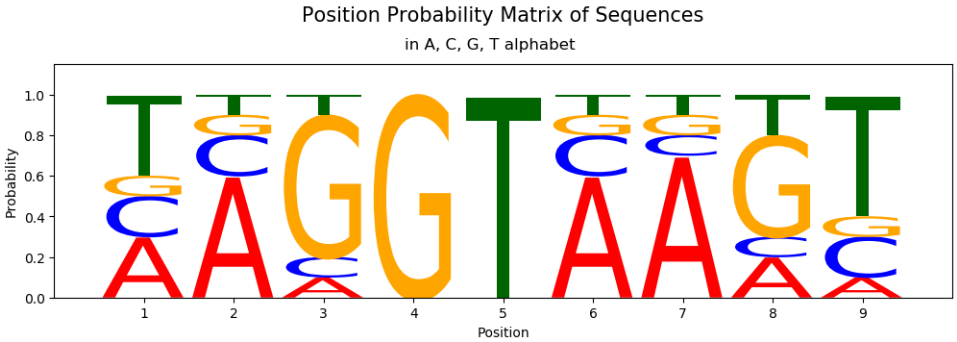


Abbildung 1: Die Position Probability Matrix (PPM) als Sequenzlogo geplottet. Man sieht den Anteil an C,G,T,A basierend auf der Größe des Buchstaben.

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.

```
=================================================================
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.

```python
#!/usr/bin/env python
#
# Position Specific Scoring (PSS)
#   by Alen Kocaj

from functools import reduce

import matplotlib as mpl
from matplotlib import pyplot as plt
from matplotlib.patches import PathPatch

from pss import PSS
from alphabet import Alphabet

def main():
    # source of aligned sequences being used to
    # build scoring matrix
    source_scoring_file = "testing/source"

    # list of target sequences which should be scored
    target_scoring_file = "testing/target"

    # path where sequence logo of PPM should be saved
    path = "pss_ppm.png"

    # use nucleotide alphabet
    alphabet = Alphabet.nucleotide()
    # the random model used
    weights = {k: v["weight"] for k, v in alphabet.items()}

    # pseudocount for preventing zero probabilities
    pseudocount = 0.0000000001

    print("====================================================================")
    print("Position Specific Scoring (PSS)")
    print("\tby Alen Kocaj")
    print()

    # read source alignments
    print(f"[i] reading source alignments out of '{source_scoring_file}'")
    sources = []
    with open(source_scoring_file) as sourcefm:
        sources = sourcefm.read().split("\n")

    expected_length = len(sources[0]) * len(sources)
    observed_length = reduce(lambda acc, curr: acc + len(curr), sources, 0)

    # validate sources
    avg_length = int(expected_length / len(sources))
    if observed_length != expected_length:
        print(f"[w] not all sequences are of same length. Expected length: {avg_length}")
```

```python
        print("The overhanging onegram will not be considered.")

    # read target sources
    print(f"[i] reading target sequences out of '{target_scoring_file}'")
    targets = []
    with open(target_scoring_file) as targetfm:
        targets = targetfm.read().split("\n")

    # build matrices
    pss = PSS(sources, alphabet.keys(), weights, avg_length, pseudocount)
    pfm = pss.build_frequency_matrix()
    ppm = pss.build_probability_matrix(pfm)

    print("====================================================================")
    print("[PPM] Position Probability Matrix")
    print(ppm)
    print()
    print("[i] plot sequence logo of PPM")
    plot_ppm_sequence_logo_pd(alphabet, ppm, path, 1.15)
    print(f"[i] figure saved at '{path}'")

    # build Postion Weight Matrix (PWM)
    weight_matrix = pss.build_weight_matrix(ppm)
    print("====================================================================")
    print("[PPM] Probability Weight Matrix")
    print(weight_matrix)

    # score targets
    print("====================================================================")
    print("[i] scoring targets\n")
    for i, target in enumerate(targets):
        print(f"# [{i}th target] #####")
        print(f"> Sequence: {target}")
        print(f"> Log Score by PSS: " + str(pss.score(target, weight_matrix)))
        print()
    print("====================================================================")

    print("[i] thank you and goodnight")

# plot_ppm_sequence_logo_pd plots a Position Probability Matrix (PPM)
# as a Sequence Logo (https://en.wikipedia.org/wiki/Sequence_logo).
# The ppm is given in a Pandas DataFrame format.
# The ppm was built based on a given alphabet.
# Globscale determines the size of each letter within the sequence logo.
#
```

```python
# Kudos to https://github.com/saketkc
# with https://github.com/saketkc/motif-logos-matplotlib for initial code.
def plot_ppm_sequence_logo_pd(alphabet, ppm, path, custom_y=-1, globscale=1.35):
    fig, ax = plt.subplots(figsize=(10,3))

    x = 1
    maxy = 0
    row_indices = list(ppm.index)
    for column in ppm:

        y = 0
        for row in range(0, len(ppm)):
            score = ppm[column][row]
            base = row_indices[row]

            letter_at(alphabet, base, globscale, x, y, score, ax)
            y += score
        x += 1
        maxy = max(maxy, y)

    plt.xticks(range(1, x))
    plt.xlim((0, x))

    if custom_y != -1:
        maxy = custom_y
    plt.ylim((0, maxy))
    plt.tight_layout()

    plt.xlabel("Position")
    plt.ylabel("Probability")
    plt.title(f"Position Probability Matrix of Sequences", y=1.15, fontsize=15)
    plt.suptitle("in " + str.join(", ", row_indices) + " alphabet", y=1.03)
    plt.savefig(f"{path}", bbox_inches="tight")

# letter_at returns the plot element of a given letter from an alphabet
# scaled by globscale and transformed around its x and y axis within the plot
def letter_at(alphabet, letter, globscale, x, y, yscale=1, ax=None):
    text = alphabet[letter]["text"]

    t = mpl.transforms.Affine2D().scale(1*globscale, yscale*globscale) + \
        mpl.transforms.Affine2D().translate(x,y) + ax.transData
    p = PathPatch(text, lw=0, fc=alphabet[letter]["color"], transform=t)
    if ax != None:
        ax.add_artist(p)
    return p

if __name__ == "__main__":
    main()
```

Listing 2: main.py

```python
#!/usr/bin/env python

import numpy as np
import pandas as pd

# PSS describes methods for calculating the position specific scoring
class PSS():
    def __init__(self, sources, alphabet, weights, avg_sequence_length,
        pseudocount):
        self.sources = sources
        self.alphabet = alphabet
        self.weights = weights
        self.avg_sequence_length = avg_sequence_length
        self.pseudocount = pseudocount

    # build_frequency_matrix computes a position frequence matrix (PFM)
    # out of the given sources. The sequences are based on the given
    # alphabet.
    def build_frequency_matrix(self):
        frequency_matrix = {}
        for onegram in self.alphabet:
            frequency_matrix[onegram] = np.zeros(self.avg_sequence_length)

        for source in self.sources:
            for i in range(0, self.avg_sequence_length):
                onegram = source[i]
                # exclude invalid characters
                if onegram not in frequency_matrix.keys():
                    continue

                frequency_matrix[onegram][i] += 1

        return pd.DataFrame(list(frequency_matrix.values()), dtype=int,
                index=self.alphabet)

    # build_probability_matrix computes a position probability matrix (PPM)
    # based on the given frequency matrix. Zero values are removed
    # by adding a given pseudocount prevent zero-frequency problems.
    def build_probability_matrix(self, pfm):
        probability_matrix = round(pfm / len(self.sources), 2)
        return probability_matrix.clip(self.pseudocount)

    # build_weight_matrix constructs the position weight matrix (PWM)
    def build_weight_matrix(self, ppm):
        return np.log(ppm)

    # score computes a score for a given target sequence, given
    # the position weight matrix (PWM)
    def score(self, target, pwm):
        score = 0
        for i, onegram in enumerate(target):
            score += pwm[i][onegram]
        return score
```

Listing 3: pss.py

```python
#!/usr/bin/env python

import collections
from matplotlib.text import TextPath

# Alphabet provides information about sequence alphabets
# like nucleotides, proteins
class Alphabet:
    # nucleotide returns the alphabet for nucleotides (C, G, T, A)
    @staticmethod
    def nucleotide():
        return collections.OrderedDict(sorted(dict({
            "T": {
                "text": TextPath((-0.305, 0), "T", size=1),
                "color": "darkgreen",
                "weight": 0.25
            },
            "G": {
                "text": TextPath((-0.384, 0), "G", size=1),
                "color": "orange",
                "weight": 0.25
            },
            "A": {
                "text": TextPath((-0.35, 0), "A", size=1),
                "color": "red",
                "weight": 0.25
            },
            "C": {
                "text": TextPath((-0.366, 0), "C", size=1),
                "color": "blue",
                "weight": 0.25
            }
        }).items()))

        # nucleotide returns the alphabet for proteins (amino acids)
        @staticmethod
        def protein():
            # to be implemented
            return {}
```

Listing 4: alphabet.py