Bioinformatics III

Third Assignment

Max Jakob (2549155)
Carolin Mayer (2552320)

May 3, 2018

Exercise 3.1: Naive Bayes classifier

(a) To derive a term that uses observable probabilities such as $P(S_{ij}|C)$ to calculate the log-likelihood ratio $log\frac{P(C|S)}{P(\bar{C}|S)}$, we use the Bayes' such that

$P(C|S) = \frac{P(C)P(S|C)}{P(S)}$

and $P(\bar{C}|S) = \frac{P(\bar{C})P(S|\bar{C})}{P(S)}$

Thus, we can write

$\frac{P(C|S)}{P(\bar{C}|S)} = \frac{P(C)P(S|C)}{P(S)} \cdot \frac{P(S)}{P(\bar{C})P(S|\bar{C})} = \frac{P(C)P(S|C)}{P(\bar{C})P(S|\bar{C})} = \frac{P(C)}{P(\bar{C})} \cdot \frac{P(S|C)}{P(S|\bar{C})}$

As we assume the features to be independent, we can write

$= \frac{P(C)}{P(\bar{C})} \cdot \sum_i \frac{P(S_i|C)}{P(S_i|\bar{C})}$, such that

$log\frac{P(C|S)}{P(\bar{C}|S)} = log\frac{P(C)}{P(\bar{C})} \cdot \sum_i log\frac{P(S_i|C)}{P(S_i|\bar{C})}$

We then classify to "proteins from a complex" if $log\frac{P(C|S)}{P(\bar{C}|S)} > 0$ and "no interaction between the proteins" otherwise

(b) Advantages:
This classifier is very simple, easy to implement and yields a fast runtime.
Disadvantages:
The classier may perform poorly on a real world dataset if the given dataset is not big enough. The result might yield a low variance but very likely show a too high bias, as the classifier will overfit according to the given data. Thus, the classifier might be to specific and show poorly accuracy on new datasets. (This disadvantage is more a general classification problem)
Another disadvantage of this classifier occur if the data does not show evenly distributed classification states, which might happen if we use real life data. Hence, the majority of the elements might belong to one specific class, and the other classes are poorly represented. Another big disadvantage is that we assume the features to be independent. Thus, tgis classifier will not yield good results if the features show a high correlation and are highly depending on each other.

(c) The script which was used to compute the results below is listed in Listing 1 The feature 33, 11 and 25 seem to have a high influence on the state of the proteins, as they appear multiple times in the table.

variant	feature number	log-ratio
0	33	1.88583202511203
1	11	1.4318111999443535
2	25	1.2127576338816688
2	33	1.0278352953876568
2	11	0.8509355508300279
1	25	0.8227646738633316
3	11	0.8021453866605959
1	33	0.7176803670838172
2	40	0.653141845946246
0	11	0.6277919995158181

Table 1: Top 10 features according to the absolute log-ratio value (test1-dataset):
$P(C) = 0.51$, $P(\bar{C}) = 0.49$

Listing 1: Source code of the script

```
import re
import math

class NaiveBayes:


    def __init__(self, filename_training):
        """read in the trainings dataset and store the sum of feature variants according to the
        self.proportion = [0] * 2
        for i in range(2):
            self.proportion[i] = [0] * 401

        self.file = filename_training
        file_training = open(filename_training, "r")

        for line in file_training:

            sample = re.split(r'\t+', line)
            interaction = int(sample[0])
            self.proportion[interaction][0] = (self.proportion[interaction][0]) + 1

            for index, feature in enumerate(sample):
                if index != 0:
                    i = (index-1) * 4 + (int(feature)+1)
                    self.proportion[interaction][i] = self.proportion[interaction][i] + 1

    def predict_given_features(self, sample):
        """ predicts the state of the proteins (complex/no complex) given a list of features"""
        total_interaction = float(self.proportion[1][0])
        total_no_interaction = float(self.proportion[0][0])
        total = total_interaction + total_no_interaction

        sum_p = 0.0
        for index, feature in enumerate(sample):
            if index != 0:

                i = (index-1) * 4 + (int(feature)+1)
                feature_interaction = float(self.proportion[1][i])
                feature_no_interaction = float(self.proportion[0][i])
                total = feature_interaction + feature_no_interaction
                if (total != 0) & (feature_no_interaction != 0) & (feature_interaction != 0):
                    sum_p = sum_p + math.log((feature_interaction / total_interaction) / (featu


        return sum_p + math.log((total_interaction/total) / (total_no_interaction/total))

```

```
    def test(self, filename_test):
        """read in test dataset, perform classification and check predicted classification"""
        file_test = open(filename_test, "r")

        result = []
        accuracy = 0.0
        total = 0
        for line in file_test:
            total = total + 1
            sample = re.split(r'\t+', line)
            classified = self.predict_given_features(sample)
            if ((classified > 0) & (int(sample[0])) == 1) | ((classified <= 0) & (int(sample[0]
                accuracy = accuracy + 1
                if classified > 0:
                    result.append([1, "true"])
                else:
                    result.append([0, "true"])

            else:
                if classified > 0:
                    result.append([1, "false"])
                else:
                    result.append([0, "false"])

        print(accuracy/total)
        return result

    def find_highest_log_ratio(self):
        """ return the 10 highest absolute log-ratio values and prints them with their respecti
        max_ration = [-1] * 10
        feature_var = [[0,0]] * 10
        total_interaction = float(self.proportion[1][0])
        total_no_interaction = float(self.proportion[0][0])
        counter_ = 0

        for index, feature in enumerate(self.proportion[0]):

            if index != 0:
                ratio_0 = math.log((float(self.proportion[1][index]) / total_interaction)/(floa
                ratio_0 = math.fabs(ratio_0)

                if ratio_0 > min(max_ration):
                    i = max_ration.index(min(max_ration))
                    max_ration[i] = ratio_0
                    feature_var[i] = [[counter_, ((index - counter_ - 1)/4)]]

                counter_ = counter_ + 1
                if counter_ > 3:
                    counter_ = 0

        print(feature_var + max_ration)
        return max_ration

if __name__ == "__main__":
        bayes = NaiveBayes("training1.tsv")
        NaiveBayes.find_highest_log_ratio(bayes)
        NaiveBayes.test(bayes, "test1.tsv")

        bayes = NaiveBayes("training2.tsv")
        NaiveBayes.find_highest_log_ratio(bayes)
        NaiveBayes.test(bayes, "test2.tsv")
```

(d) Accuracy: 0.73, which means that 73 % of the predicted classification were correct. This accuracy can demonstrate the capability of the trained classifier.
Output:

variant	feature number	log-ratio
0	33	1.9588135538912212
1	11	1.6574952073878797
0	83	1.2992829841302609
3	59	1.109239381242396
0	99	1.0116009116784799
2	25	0.9315582040049436
3	98	0.9315582040049436
0	4	0.9315582040049435
3	58	0.9090853481528849
2	11	0.8861767516263722

Table 2: Top 10 features according to the absolute log-ratio value (test2-dataset):
$P(C) = 0.78$, $P(\bar{C}) = 0.22$

[[0,′ true′], [1,′ true′], [1,′ true′], [1,′ true′], [1,′ true′], [1,′ true′], [1,′ true′], [0,′ true′], [0,′ true′],
[1,′ true′], [1,′ true′], [0,′ true′], [1,′ true′], [0,′ true′], [0,′ true′], [1,′ true′], [1,′ true′], [1,′ true′],
[0,′ false′], [1,′ false′], [0,′ true′], [1,′ true′], [0,′ true′], [1,′ false′], [0,′ true′], [0,′ true′], [1,′ true′],
[1,′ true′], [1,′ true′], [0,′ true′], [1,′ true′], [0,′ true′], [1,′ false′], [1,′ false′], [0,′ true′], [0,′ false′],
[1,′ true′], [1,′ false′], [1,′ true′], [0,′ true′], [1,′ false′], [0,′ true′], [0,′ true′], [0,′ false′], [1,′ true′],
[0,′ true′], [1,′ false′], [1,′ true′], [1,′ true′], [0,′ true′], [0,′ true′], [1,′ true′], [1,′ true′], [0,′ false′],
[0,′ false′], [1,′ false′], [0,′ true′], [0,′ true′], [0,′ true′], [0,′ true′], [1,′ true′], [0,′ true′], [1,′ true′],
[0,′ true′], [0,′ true′], [1,′ false′], [1,′ true′], [1,′ true′], [0,′ true′], [0,′ true′], [1,′ true′], [0,′ true′],
[0,′ false′], [0,′ false′], [1,′ false′], [1,′ true′], [0,′ true′], [1,′ true′], [1,′ true′], [1,′ false′], [1,′ false′],
[0,′ true′], [1,′ true′], [1,′ false′], [1,′ true′], [0,′ false′], [1,′ true′], [1,′ true′], [1,′ true′], [0,′ true′],
[0,′ true′], [0,′ false′], [0,′ false′], [0,′ false′], [1,′ true′], [1,′ true′], [0,′ false′], [1,′ false′], [1,′ false′],
[1,′ true′]]

(e) Accuracy: 0.32, which means that only 32 % of the predicted classifications were correct. This inferior performance is not surprising, as the training dataset shows a poorly representation of the state "no interactions between proteins" (with $P(\bar{C} = 0.22)$. Hence, the classification states are not evenly distributed, and consequentially lead to a poorly classification prediction.

Output:
[[1,′ false′], [0,′ true′], [1,′ false′], [1,′ true′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′],
[0,′ true′], [1,′ false′], [0,′ true′], [1,′ true′], [1,′ false′], [1,′ false′], [1,′ false′], [0,′ true′], [1,′ true′],
[1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [0,′ true′], [1,′ false′],
[1,′ false′], [1,′ false′], [1,′ true′], [0,′ true′], [1,′ true′], [1,′ false′], [0,′ true′], [1,′ false′], [1,′ false′],
[1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [0,′ true′], [1,′ false′], [0,′ true′], [0,′ true′],
[1,′ false′], [1,′ false′], [1,′ false′], [0,′ true′], [1,′ false′], [1,′ false′], [1,′ true′], [1,′ true′], [1,′ false′],
[1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [0,′ true′], [1,′ true′],
[1,′ false′], [1,′ false′], [0,′ true′], [1,′ false′], [1,′ true′], [1,′ false′], [1,′ false′], [1,′ true′], [1,′ false′],
[1,′ false′], [1,′ false′], [1,′ true′], [1,′ false′], [1,′ false′], [1,′ true′], [0,′ true′], [1,′ false′], [1,′ true′], [0,′ true′],
[1,′ false′], [0,′ true′], [1,′ true′], [1,′ true′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′],
[1,′ true′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′], [1,′ false′],
[1,′ false′]]

Exercise 3.2: Network communities

(a) We have given the two following formulas:

$$z_{i,j}^{(3)} = \text{number of triangles between i and j} \tag{1}$$

$$c_{i,j}^{(3)} = \frac{z_{i,j}^{(3)} + 1}{mindegree(i-1, j-1)} \tag{2}$$

As the number of existing triangles between two nodes is limited by the minimal degree of both nodes -1, we can assume the supremum of $z_{i,j}^{(3)}$ to be

$$sup\left(z_{i,j}^{(3)}\right) = mindegree(i,j) - 1 \tag{3}$$

We can use this supremum in (2) as follows:

$$c_{i,j}^{(3)} = \frac{mindegree(i,j) - 1 + 1}{mindegree(i-1, j-1)} \tag{4}$$

Assuming $x = mindegree(i,j)$, then it is easy to see that $mindegree(i-1, j-1) = mindegree(i,j) - 1 = x - 1$. This can be used in (4) as

$$c_{i,j}^{(3)} = \frac{x}{x-1} \tag{5}$$

This function is will monotonically decrease converge to 1 with a large degree of x. (x is only dependent of the minimum degree of i and j, thus we can see it as a monotonic function over the interval $[2, inf[$, as 1 results in a infinite number).
The maximum value hence is the border point of the interval ($mindegree(i,j) = 2$). This results in an maximal edge-cluster coefficient of $c_{2,2}^{(3)} = 2$.

(b) Below, the output of the desired edge-cluster algorithm is shown. To get this output, run the script listed in Listing 2. The handwritten dendogram can be found in Figure 1

Name A	Name B	edge-cluster coef.
Sansa	Tyrion	0.3333333333333333
Hound	Joffrey	0.5
Jon	Eddard	0.5
Eddard	Robert	0.5
Hound	Mountain	1.0
Arya	Catelyn	1.0
Eddard	Arya	1.0
Arya	Sansa	1.0
Catelyn	Baelish	1.0
Baelish	Sansa	1.0
Jaime	Joffrey	1.0
Cersei	Tyrion	1.0
Cersei	Jaime	1.0
Jeor	Samwell	2.0
Eddard	Catelyn	2.0
Robert	Joffrey	2.0
Tyrion	Shae	inf
Tyrion	Jaime	inf
Cersei	Joffrey	inf
Robert	Cersei	inf
Baelish	Varys	inf
Catelyn	Sansa	inf
Eddard	Sansa	inf
Jon	Samwell	inf
Jon	Jeor	inf
Mountain	Cersei	inf
Hound	Arya	inf

Listing 2: Source code of the script

```
import re
import math


class node:

    def __init__(self, name):
        self.name = name
        self.links = []

    def __eq__(self, other):
        return self.name == other.name

    def add(self, name):
        self.links.append(name)

    def remove(self, name):
        self.links.remove(name)

    def degree(self):
        return self.links.__len__()
# i = number of triangles
# j = min degree of both nodes
def c(i,j):
    if(j == 1):
        return math.inf
    else:
        return (i+1)/(j-1)

def numTri(A, B, net):
    n1 = net[A].links
    n2 = net[B].links
    return list(set(n1) & set(n2)).__len__()

file = open("GoT.txt", "r")
net = {}
links = []
for line in file:
    line = line.rstrip()
    temp = re.split("␣", line)
    if temp[0] not in net:
        n1 = node(temp[0])
    else:
        n1 = net[temp[0]]
    if temp[1] not in net:
        n2 = node(temp[1])
    else:
        n2 = net[temp[1]]
    n1.add(temp[1])
    n2.add(temp[0])
    net[n1.name] = n1
    net[n2.name] = n2
    links.append([temp[0], temp[1], 0])

while links.__len__() != 0:
    min_val = math.inf
    min_index = -1
    for i in range(0,links.__len__()):
        l = links[i]
        c1 = net[l[0]].degree()
        c2 = net[l[1]].degree()
        s = min(c1, c2)
        z = c(numTri(l[0], l[1], net), s)
        ltemp = [l[0], l[1], z]
        if z < min_val:
            min_val = z
```

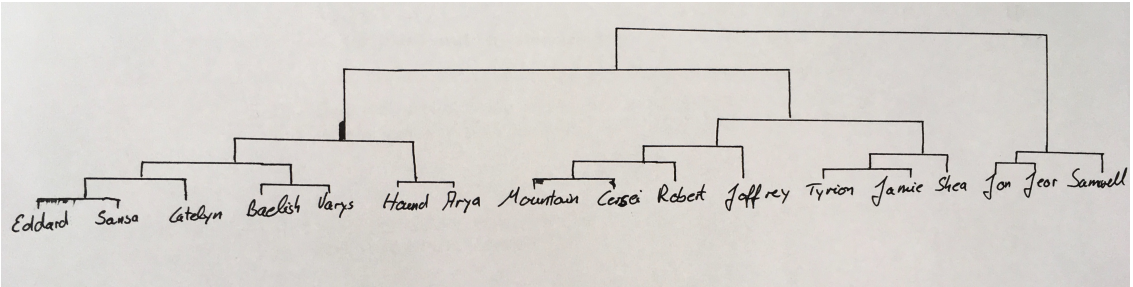

Figure 1

```
                min_index = i
            links[i] = ltemp

        print(links[min_index])
70      l = links[min_index]
        n1 = net[l[0]]
        n2 = net[l[1]]
        n1.remove(n2.name)
        n2.remove(n1.name)
75      net[n1.name] = n1
        net[n2.name] = n2
        del links[min_index]
```

(c)

- Community 1: Jon, Jeor, Samwell
  This community is a strong, as every node got more connections to nodes inside the community as to the outside.
- Community 2: Catelyn, Arya, Sansa, Eddard
  This community is a strong, as every node got more connections to nodes inside the community as to the outside.