Bioinformatics III

Third Assignment

Alexander Flohr (2549738)
Andrea Kupitz (2550260)

May 4, 2018

Exercise 3.1: Naive Bayes classifier

(a) Because the features are independent:

$$P(S) = \prod_i P(S_i) \tag{1}$$

Following the Bayes theorem:

$$\frac{P(S|C) * P(C)}{P(S)} = P(C|S) \tag{2}$$

Further

$$log(\frac{P(C|S)}{P(\bar{C}|S)}) = log(\prod_i \frac{\frac{P(S_i|C)*P(C)}{P(S_i)}}{\frac{P(S_i|\bar{C})*P(\bar{C})}{P(S_i)}}) = \sum_i log(\frac{P(S_i|C) * P(C)}{P(S_i|\bar{C}) * P(\bar{C})}) \tag{3}$$

The actual classification is done by evaluating the sum over the log-likelihood ratios. If the sum is smaller zero, it means most of the log-likelihoods were smaller zero because the probability of $\bar{C}$ was greater than the one for C. Therefore, the state sequence evaluates to "no interaction". Vice versa, if the sum over the likelihoods was greater than zero, we classify 1, which means there is an interaction between the proteins.

(b) The logarithm of the ratio enables to see directly if the probability is higher that the observed states belong to interacting or non-interacting proteins. Moreover the ratio of the likelihoods normalizes the probabilities because states occurring more often will have the same ratio than less frequent ones.
This classification may perform bad on real-world data if a feature classifies all interactions to the same class because we cannot handle probabilities being zero. Moreover classification is bad if the dataset is small because the classification may be biased to one class.

(c) Listing 1 shows source code.

Listing 1: Example Listing of source code

```
import csv
import math
import sys


class Model:
    """Builds a model. This basically means to determine all necessary priors and likelihoods
     - P(C), P(Si|C)..."""

    def __init__(self):
        """Inits empty list to store classification value and matrix for corresponding feature
        states"""
```

```
        self.classifier = []
        self.featureMatrix = []

    def readTSV(self,file):
        """Read tsv-file in and store values"""
        with open(file) as f:
            reader = csv.reader(f, delimiter='\t')
            for row in reader:
                self.classifier.append(row[0])
                features = []
                for i in range(1, 101):
                    features.append(row[i])
                self.featureMatrix.append(features)

    def classify(self):
        """
        Compute P(C)
        :return: number of occurrences of class=1
        """
        sumc = self.classifier.count("1")
        probc = float(sumc)/(len(self.classifier)-1)
        print("P(C) = " + str(probc))
        print("P(not C) = " + str(1-probc))
        return sumc

    def conditionalProb(self, sumc):
        """
        Compute P(Si|C) for every feature i and state, compute log-likelihood ratio for each S
        :param sumc: number of occurrences of class=1
        :return: matrix of log-likelihood ratio for each feature and state
        """
        probc = float(sumc) / (len(self.classifier) - 1)
        countnotc = len(self.classifier)-sumc
        probFeature = [0]*100
        probnotc = [0]*100
        for i in range(100):
            probFeature[i] = [0]*4
            probnotc[i] = [0]*4
        #probnotc = probFeature.copy()
        #count successfull classifications for each feature and state
        for rownumber, rowvalue in enumerate(self.featureMatrix):
            for feature, state in enumerate(rowvalue):
                if self.classifier[rownumber] == "1":
                    probFeature[feature][int(state)] += 1
                else:
                    probnotc[feature][int(state)] += 1
        #compute log-likelihood ratio
        for i in range(len(probFeature)):
            for j in range(len(probFeature[0])):
                if probnotc[i][j] != 0 and probFeature[i][j] != 0:
                    probFeature[i][j] = probFeature[i][j] / float(sumc) / (probnotc[i][j] / flo
                    probFeature[i][j] = math.log(probFeature[i][j])
        return probFeature, probc

    def reportBestK(self, k, probFeature):
        """
        Report the ten Si (feature number, variant and log-ratio) with the highest absolute
        log-likelihood ratios
        :param k: 10 here
        :param probFeature: matrix of log-likelihood ratio for each feature and state
        """
        bestK = []
        print("enumeration of best features & log ratio & feature number & state variant\\")
        for ki in range(k):
            max = 0
            r = -1
            c = -1
```

```
            for row, i in enumerate(probFeature):
                for column, j in enumerate(i):
                    if math.fabs(j) > math.fabs(max):
                        max = j
                        r = row
                        c = column
                        probFeature[row][column] = sys.float_info.min
            bestK.append([max, r, c])
            print(str(ki+1) + " & " + str(max) + " & " + str(r) + " & " + str(c) + "\\")

    def prediction(self, probFeature, featureMatrix, probc):
        """
        Predicts the ability to interact for the protein pairs
        :param probFeature: matrix of log-likelihood ratio for each feature and state that was
        learned from trainings test set
        :param featureMatrix: Matrix with values of each feature from test file
        :param probc: probability to classify a protein pair to 1
        :return: list with classified interactions for each protein pair
        """
        classification = []
        for rowvalue in featureMatrix:
            probC = 0
            for feature, state in enumerate(rowvalue):
                probC += probFeature[feature][int(state)]
            probC += math.log(float(probc)/(1-probc))
            print(probC)
            c = 1 if probC > 0 else 0
            classification.append(c)
        print(classification)
        return classification

    def accuracy(self, classification):
        """
        Computes the accuracy of the classifier
        :param classification: list with classified interactions for each protein pair
        :return: accuracy of classification
        """
        accuracy = 0.0
        for i, c in enumerate(classification):
            if str(c) == self.classifier[i]:
                accuracy += 1
        accuracy = accuracy / (len(classification)-1)
        print(accuracy)


m = Model()
m.readTSV("training1.tsv")
probc = m.classify()
probFeature, probc = m.conditionalProb(probc)
m.reportBestK(10, probFeature)
mtest = Model()
mtest.readTSV("test1.tsv")
classification = m.prediction(probFeature, mtest.featureMatrix, probc)
m.accuracy(classification)
```

$P(C) = 0.5125628140703518$
$P(\bar{C}) = 0.48743718592964824$
Table 3 shows best 10 absolute features.
The 10 features in table 3 seem to be the most promising features to recognize protein interactions correctly because their log-likelihood is very high. There is no specific region of features which seems to be very helpful. The best ten features are rather widely spread. Further, no state variant seems to indicate an interaction.

(d) Listing 1 shows source code.
The classification is equally distributed between both classes. But the accuracy is rather

enumeration of best features	log ratio	feature number	state variant
1	-1.896088525279219	33	0
2	1.2025011337144798	25	2
3	1.0175787952204678	33	2
4	0.7074238669166282	33	1
5	0.642885345779057	40	2
6	-0.6380484996830071	57	2
7	0.6003257313602611	81	0
8	-0.5271859068711975	95	2
9	-0.4885167657120435	88	0
10	-0.4885167657120435	89	1

Table 1: best 10 features

low.
For "test1.tsv" the classification is: [1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1]
Whereby each position in the list corresponds to the protein pair in the test file at the same position. The exact value of the accuracy is 0.5757575757575758.

(e) P(C) = 0.7839195979899497
P($\bar{C}$) = 0.2160804020100503
Table 5 shows best 10 features.

enumeration of best features	log ratio	feature number	state variant
1	-1.9818030721159199	33	0
2	1.2762934659055623	83	0
3	0.9886113934537812	99	0
4	0.9085686857802447	98	3
5	0.8514102719402962	99	3
6	-0.7290401036205519	87	1
7	0.7166776779701394	74	0
8	0.7037742731342315	90	1
9	0.6774569648168581	91	1
10	0.6155815610987706	63	0

Table 2: best 10 features

For "test2.tsv" the classification is: [0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0]
The accuracy is 0.37373737373737376.
The classification might be worse for the second test set because in the second trainings set, interacting proteins occur more often than non-interacting ones which may lead to proteins rather been interpreted as interacting.

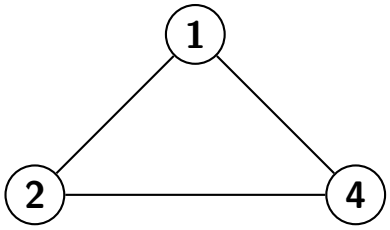

Figure 1: example of network with maximal clustering coefficient

Exercise 3.2: Network communities

(a) The maximal finite value that the edge-clustering coefficient can take is 2 because it is maximal if the numerator is maximal and the denominator is minimal. The denominator cannot get smaller than 0 because every pair of nodes will have at least one edge between them. Because division by zero is undefined, we take the next value 1, which means that the nodes both have at least one more edge than the one to each other. If both nodes have at least two edges, there is maximal one triangle between them if they are connected to the same node. Therefore, the numerator is 2. If there were more triangles between the nodes, the denominator would also increase and the ratio is smaller. Thus, 2 is the maximal value the coefficient can take. Figure 1 shows an example of a network with maximal clustering coefficient between all nodes.

(b) (1) Listing 2 shows source code.

Listing 2: Example Listing of source code

```
from os.path import exists
from AbstractNetwork import AbstractNetwork
from AbstractNetwork import AbstractNetwork
from Node import Node

class DecompositionNetwork(AbstractNetwork):
    def __createNetwork__(self, filename, dummy):
        """
        Create a network from a file
        """
        self.edgematrix = []
        self.nodeIDs = dict()
        self.count = 0  # count names
        self.num_edges = 0
        names = []
        if exists(filename):
            # "with" closes the file again after reading
            with open(filename) as openfile:
                for line in openfile:
                    # get entries of a line as list
                    content = line[0:(len(line)-1)].split(" ")
                    # and store them
                    if len(content) == 2:
                        n1 = self.getNode(content[0])
                        n2 = self.getNode(content[1])
                        n1.addLinkTo(n2)
                        n2.addLinkTo(n1)
                        self.num_edges += 1
                        # store names and positions for later access
                        if content[0] not in names:
                            names.append(content[0])
                            self.nodeIDs[self.count] = content[0]
```

```
                            self.count += 1
                        if content[1] not in names:
                            names.append(content[1])
                            self.nodeIDs[self.count] = content[1]
                            self.count += 1
            else:
                print(filename, "does not exist")

    def getC(self, node1, node2):
        if self.getNode(node1).hasLinkTo(self.getNode(node2)):
            triangles = 0
            # find triangles
            for i in self.getNode(node1).nodelist:
                for j in i.nodelist:
                    if j.id == node2:
                        triangles += 1
                        break
            # get minimal degree
            min_degree = self.getNode(node1).degree()
            if min_degree > self.getNode(node2).degree():
                min_degree = self.getNode(node2).degree()
            # return C + avoid errors
            if min_degree > 1:
                return float(triangles+1)/float(min_degree-1)
            else:
                return 0.0
        else:
            return 3.0

    def updateEdgeMatrix(self):
        """
        compute scores clustering coefficients for each edge
        """
        self.edgematrix = []
        for i in range(0,self.size()):
            self.edgematrix.append([3.0]*self.size())
        for i in range(0,self.size()):
            for j in range(0,self.size()):
                name1 = self.nodeIDs[i]
                name2 = self.nodeIDs[j]
                self.edgematrix[i][j] = self.getC(name1, name2)


    def deleteOneEdge(self):
        """
        Delete the edge with the smalest clustering coefficient
        """
        min_x = ""
        min_y = ""
        min_v = 3
        # serch in materix
        for i in range(0,self.size()):
            for j in range(i,self.size()):
                if self.edgematrix[i][j] < min_v:
                    min_v = self.edgematrix[i][j]
                    min_x = self.nodeIDs[i]
                    min_y = self.nodeIDs[j]
        # remove edge
        self.getNode(min_x).removeLinkTo(self.getNode(min_y))
        self.getNode(min_y).removeLinkTo(self.getNode(min_x))
        self.num_edges -= 1
        print("Remove edge with clustering coefficient", min_v, ":\t", min_x, "-", min_y)

    def decomposite(self):
        """
        Controle edge decomposition until each edge is removed
        """
```

```
        while self.num_edges > 0:
100         self.updateEdgeMatrix()
            self.deleteOneEdge()
```

Resulting detetion order:

```
Remove edge with clustering coefficient 0.0 :  Baelish - Varys
Remove edge with clustering coefficient 0.0 :  Tyrion - Shae
Remove edge with clustering coefficient 0.5 :  Hound - Joffrey
Remove edge with clustering coefficient 0.5 :  Jon - Eddard
Remove edge with clustering coefficient 0.5 :  Eddard - Robert
Remove edge with clustering coefficient 0.5 :  Sansa - Tyrion
Remove edge with clustering coefficient 1.0 :  Hound - Mountain
Remove edge with clustering coefficient 0.0 :  Hound - Arya
Remove edge with clustering coefficient 0.0 :  Mountain - Cersei
Remove edge with clustering coefficient 1.0 :  Joffrey - Jaime
Remove edge with clustering coefficient 1.3̄ :  Catelyn - Sansa
Remove edge with clustering coefficient 1.0 :  Arya - Catelyn
Remove edge with clustering coefficient 1.0 :  Eddard - Catelyn
Remove edge with clustering coefficient 0.0 :  Catelyn - Baelish
Remove edge with clustering coefficient 0.0 :  Sansa - Baelish
Remove edge with clustering coefficient 2.0 :  Joffrey - Cersei
Remove edge with clustering coefficient 0.0 :  Joffrey - Robert
Remove edge with clustering coefficient 0.0 :  Cersei - Robert
Remove edge with clustering coefficient 2.0 :  Arya - Eddard
Remove edge with clustering coefficient 0.0 :  Arya - Sansa
Remove edge with clustering coefficient 0.0 :  Eddard - Sansa
Remove edge with clustering coefficient 2.0 :  Cersei - Jaime
Remove edge with clustering coefficient 0.0 :  Cersei - Tyrion
Remove edge with clustering coefficient 0.0 :  Jaime - Tyrion
Remove edge with clustering coefficient 2.0 :  Jeor - Samwell
Remove edge with clustering coefficient 0.0 :  Jeor - Jon
Remove edge with clustering coefficient 0.0 :  Samwell - Jon
```

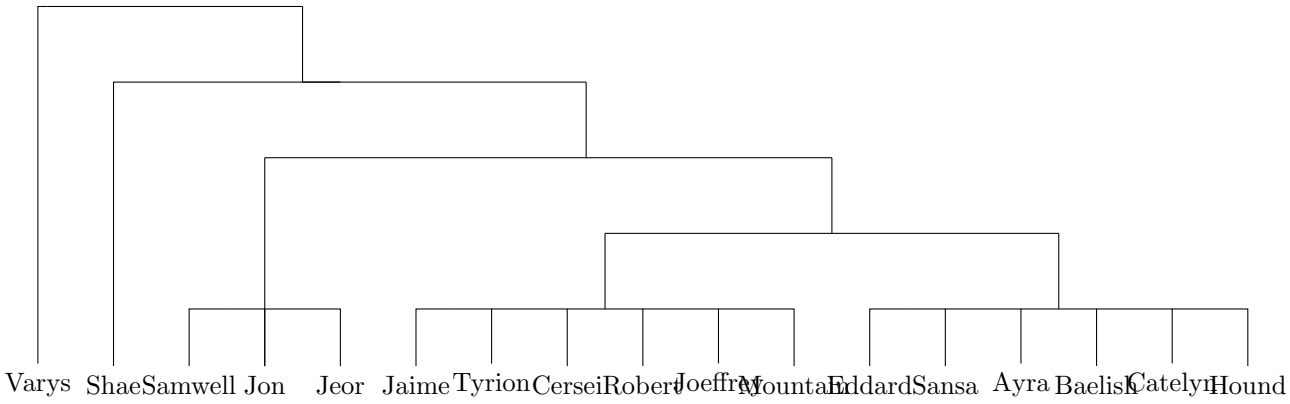

Figure 2: dendrogram of communities

(2) Figure 2b shows the dendrogram of the network.

(c) Figure 3 shows the visually determined two disjoint communities. For both communities the strong criterion applies. For all nodes besides Ayra, Eddard and Sansa from community one and the Hound, Robert and Tyrion from community two applies that they are only connected to nodes within the same community. Whereas the six outliers have one edge to the outside but at least two links to the inside, which means for every member of the subgraph V holds, it has more links to the inside of the community (k^{in}) than to the outside (k^{out}):

$$k_i^{in}(V) > k_i^{out}(V) \forall i \in V \tag{4}$$

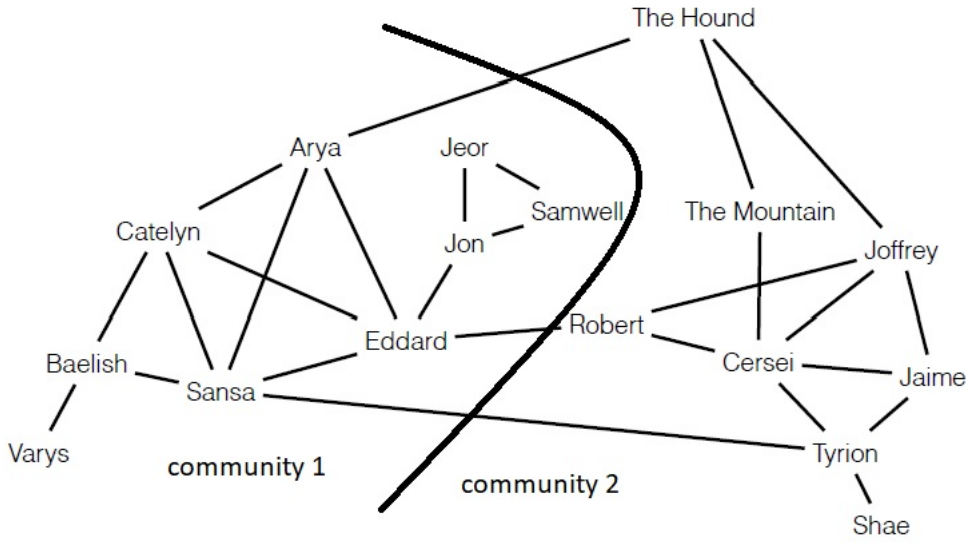

Figure 1: The network in "GoT.txt" visualized.

Figure 3: Visualization of two disjoint communities