Jacek Radajewski
Student number: 43612772
BIOL3014 – Advanced Bioinformatics
Practical 3

Table of Contents

Exercise 1A

The following checks were performed (see code listing for details):

1. checked the total number of proteins against total number of secondary structures
2.

Code

```
'''
Created on 19/08/2014

@author: jacekrad
'''
from sequence import *
from prob import *
from __builtin__ import exit
from sys import stderr
from spred import slidewin

prot = readFastaFile('prot2.fa', Protein_Alphabet)
sstr = readFastaFile('sstr3.fa', DSSP3_Alphabet)

if len(prot) != len(sstr):
    stderr.write("prot2.fa contains different number of sequences than sstr3.fa\n")
    exit(1)

# index
i = 0
for protein in prot:
    secondary_structure = sstr[i]
    i += 1
    # make sure the name is the same for both protein
    # and its secondary structure entry
    if protein.name != secondary_structure.name:
        stderr.write(protein.name + " and " + secondary_structure.name \
                     + " are not the same where they should be\n")
    # check each amino acid and the corresponding secondary structure
    if len(protein.sequence) != len(secondary_structure.sequence):
        stderr.write(protein.name + " and it's secondary structure sequence "
                     + " differ in length: " + str(len(protein.sequence)) \
                     + " vs " + str(len(secondary_structure.sequence)) + "\n")
    # checking for invalid characters in the secondary structure as specified in
    # the question, but this is not strictly required as the check is already done
    # by readFastaFile (check against specified alphabet)
    elif 'C' in secondary_structure or 'H' in secondary_structure or 'H' in
secondary_structure:
        pass
    else:
        print "invalid character found in secondary structure of " +
secondary_structure.name


print "checked", i, "entries"

prot_trn = prot[0::2]  # even-numbered indices
prot_tst = prot[1::2]  # odd-numbered indices
sstr_trn = sstr[0::2]  # even-numbered indices
sstr_tst = sstr[1::2]  # odd-numbered indices
```

Exercise 1B

The length of the protein sequence is **221** residues and the length of the sliding window subsequences is **217** windows. The reason for this is that the number of windows will equal to the number of residues minus (windows size – 1).

This can be easily illustrated with a sequence of length 7 and windows of length 5.

There are 7 residues and 3 windows.

Number of windows = sequence length – (window length – 1)

= 7 – (5-1)

= 7 -4

= 3

Window	P	W	S	C	L	L	C
1							
2							
3							

Code

```
'''
Created on 20/08/2014

@author: jacekrad
'''
from sequence import *
from prob import *
from __builtin__ import exit
from sys import stderr
from spred import slidewin

prot = readFastaFile('prot2.fa', Protein_Alphabet)
sstr = readFastaFile('sstr3.fa', DSSP3_Alphabet)

prot_trn = prot[0::2]  # even-numbered indices
prot_tst = prot[1::2]  # odd-numbered indices
sstr_trn = sstr[0::2]  # even-numbered indices
sstr_tst = sstr[1::2]  # odd-numbered indices

W = 5
print "Sequence length  :", len(prot_trn[0])
print "Number of windows:", len(slidewin(prot_trn[0], W))
```

Output

```
Sequence length  : 221
Number of windows: 217
```

Exercise 1C

Each window from the amino acid sequence is matched against a single letter in the secondary structure. The secondary structure letter used is the one that is in the middle of the window. Example below demonstrates this by highlighting a windows in a protein sequence and the corresponding secondary structure letter:

```
IVGGYECKAYSQPHQVSLNSGYHFCGGSLVNENWVVSAAHCYKSRVAVRLGEHNIKVTEG
CECCEECCCCCCCCCEEEEECCCEEEEEEECCCCEEEECHHHCCCCCEEEECCCECCCCCC
```

Now, this means that beginning and the end of the secondary structure sequence will not have corresponding sequence. Below we can see that first window corresponds to third letter in secondary structure sequence. The first two and the last two letter letters in the secondary structure are an 'overhang' and will have to be bypassed in the processing.

```
IVGGYECKAYSQPHQVSLNSGYHFCGGSLVNENWVVSAAHCYKSRVAVRLGEHNIKVTEG
CECCEECCCCCCCCCEEEEECCCEEEEEEECCCCEEEECHHHCCCCCEEEECCCECCCCC
```

So the following code removes this overhang. W is the window size. Python splicing is used to strip the first and last letters.

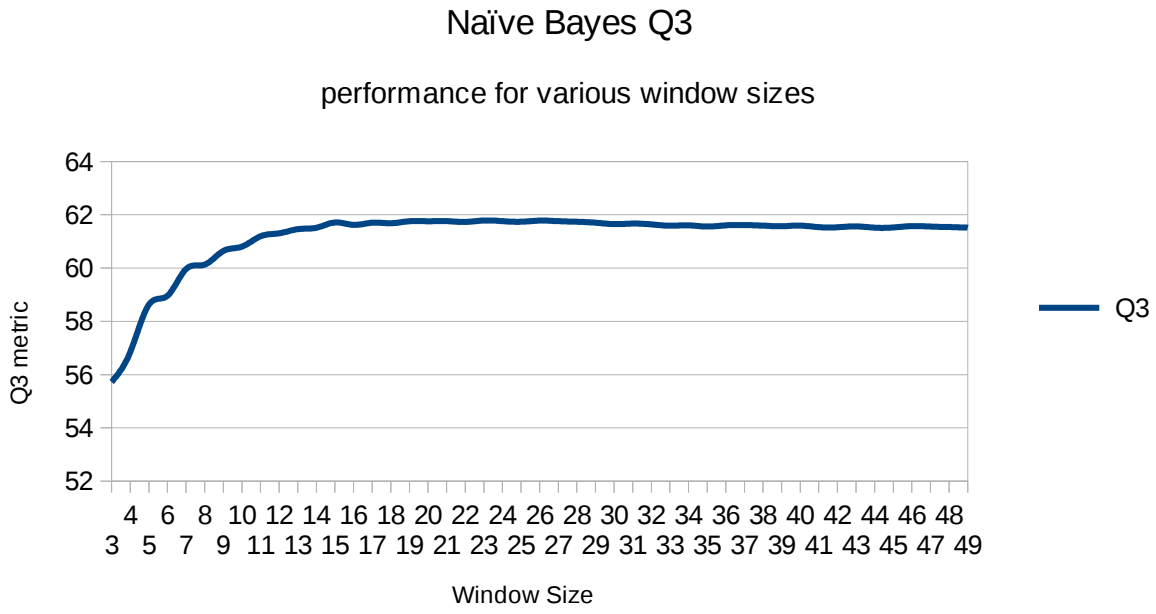
```
subtarg = sstr_tst[i][W / 2:-W / 2 + 1]
```

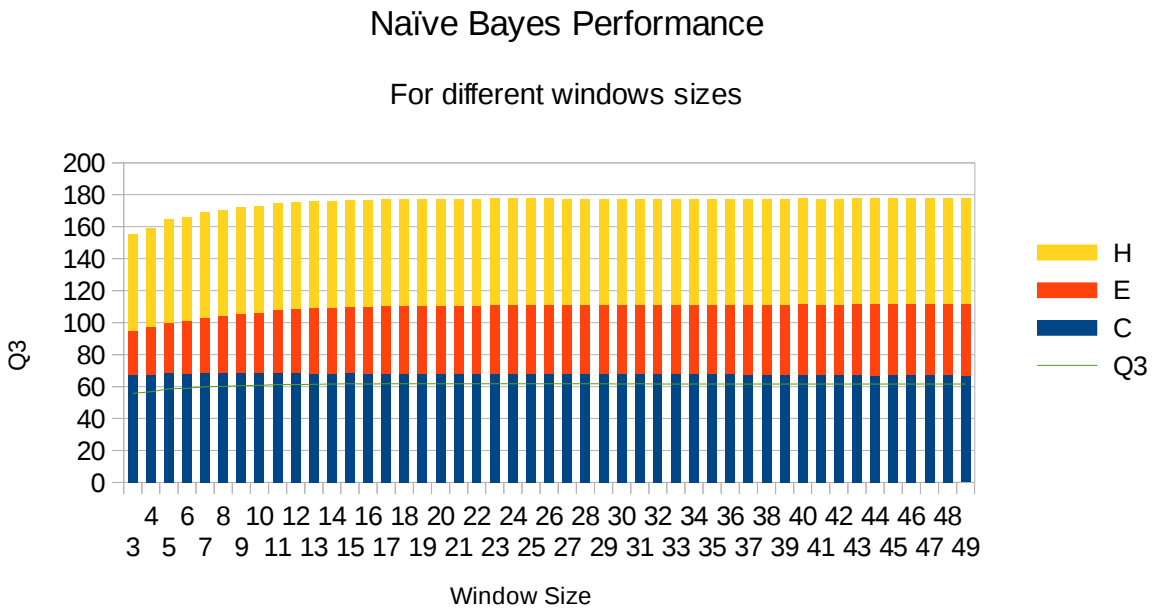
Confusion Matrix

```
[[ 85047.  37475.   6664.]
 [ 31983.  34775.   5166.]
 [ 62723.  45709.   6165.]]
```

		Predicted Class		
		Coil	**Beta Sheet**	**Alpha Helix**
Actual Class	**Coil**	**85047**	37475	6664
	Beta Sheet	31983	**34775**	5166
	Alpha Helix	62723	45709	**6165**

Exercise 1D


Naïve Bayes Q3
performance for various window sizes
Q3 metric
64
62
60
58
56
54
52
4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Window Size
Q3


Naïve Bayes Performance
For different windows sizes
Q3
200
180
160
140
120
100
80
60
40
20
0
4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Window Size
H
E
C
Q3

Exercise 1F

Exercise 1G

Exercise 1I