“Prediction by Machine Learning”

Practical BIOL3014: 5 marks (part 1)
Workshop BINF7000: 10 marks (parts 1-3)

Below are a number of practical exercises that aim to guide you through issues related to PREDICTION by MACHINE LEARNING. The report is submitted for marking via the course web site. Include code that you have written yourself. Marks and comments will be made available in Blackboard (via Grade Centre and Grade Mark, respectively).

In Part 1, all students work through an example of how a Naïve Bayes classifier is used to predict secondary structure of proteins and make observations on the way. In Part 2, postgraduate students (BINF7000) use a neural network to perform a similar task. In Part 3, postgrads look closer at the prediction in view of a full-blown 3D structure.

The target workload for Part 1 corresponds roughly to two hours of practical work that should ideally be completed within the next scheduled computer lab session (supported by a tutor or lecturer), and (max) two hours of additional work that should be completed during unscheduled time. Part 2 can be done in two hours but can take longer depending on your programming experience. Parts 3 and 4 are expected to take an hour each.

Code and data are available from the subversion repository:
http://bioinf.scmb.uq.edu.au/pubsvn/binfpy/src (code)
http://bioinf.scmb.uq.edu.au/pubsvn/binfpy/BIOL3014 (files for BIOL3014)
http://bioinf.scmb.uq.edu.au/pubsvn/binfpy/BINF7000 (files for BINF7000)

Relevant modules to this prac are

`symbol.py`	(see `DSSP3_Alphabet`)
`prob.py`	(see `NaiveBayes`)
`sequence.py`	(for loading sequence data)
`ml.py`	(see `NN`)
`spred.py`	(examples and NN wrapper for sequence models)

You will find snippets of example code below to get you started. However, you should always try to understand what the code does, including what is happening in other modules. If you do not, make sure you ask…

EXERCISES

For all examples to work make sure you import all modules into your current namespace, e.g.

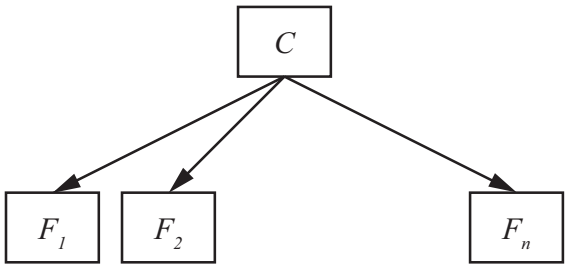
```
>>> from sequence import *
>>> from prob import *
...
```

Part 1 (5 marks)

A naïve Bayes classifier determines a probability of the form

$$P(C \mid F_1, F_2, \ldots, F_n)$$

where $F_1, F_2, \ldots, F_n$ are "features" and C is the class. Naïve Bayes classifiers can be depicted in the following way.



You will construct a naïve Bayes classifier for secondary structure prediction (where your class is secondary structure as defined by 3-class DSSP and your features are the amino acids in a window surrounding the residue for which the secondary structure applies).

You will require data to train it and test it to gauge what the accuracy is for truly novel proteins.

You are requested to work with secondary structure data that has been prepared for you. The data originate from PDB (Protein Data Bank; http://www.pdb.org) via PDBFINDER2 (http://swift.cmbi.ru.nl/gv/pdbfinder/). We randomly selected about 20% of all proteins, and then reduced this set to only include sequences with less than 20% sequence identity (according to `blastclust`; http://blast.ncbi.nlm.nih.gov).

a. Inspect data sets. The data directory contains a protein set (`prot2.fa`) and the corresponding secondary structure set (`sstr3.fa`). Start Python and load the files after importing the necessary modules.

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

Write a small piece of code to verify that the two files contain entries for the same ~2500 proteins (count them, check that the names are the same and in the same order in the two files), that each amino acid residue in the first correspond to a 3-class secondary structure element in the second file (H: Helix, E: Strand, C: Coil), *i.e.*

check that the sequences of amino acids and of secondary structure letters have the same size and that the latter only use the letters H, E and C. Report the code you wrote for checking these elements in your report.

Divide each data set in half, corresponding to a train and a test set respectively, e.g. divide them using odd and even number indices:

```
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
```

b. We have prepared a module `spred` with a method `slidewin` to generate multiple sub-sequences corresponding to a "moving-window" of a specified width (`W`). Verify that it generates appropriate `W`=5 sub-sequences for `prot_trn[0]`. What is the length of `prot_trn[0]`? What is the number of sub-sequences? Why are these numbers different?

```
W = 5
slidewin(prot_trn[0], W)
['IVGGY', 'VGGYE', 'GGYEC', 'GYECK', 'YECKA', ...
```

c. Create a probabilistic classifier of secondary structure. For example

```
nb = NaiveBayes([Protein_Alphabet for _ in range(W)], DSSP3_Alphabet)
```

- Train a Naïve Bayes classifier using the training data above. In the example below, explain how the target sequence is constructed to match the input window. (Takes ~5 seconds on my laptop.)

```
for i in range(len(prot_trn)):           # for each sequence
subseqs = slidewin(prot_trn[i], W)       # construct sub-seqs
subtarg = sstr_trn[i][W/2:-W/2+1]        # secondary structure elem
     for j in range(len(subseqs)):
          nb.observe(subseqs[j], subtarg[j])    # let NB count
```

- Evaluate how well your classifier is doing on the test data. Look up "confusion matrix" in Wikipedia. You should construct one from your test set, e.g. (Note: testing takes some time also; ~10 seconds on my laptop.) Print your matrix, with labels for the three classes. Report which class (of the three) is more difficult to predict.

```
cm = numpy.zeros((len(DSSP3_Alphabet), len(DSSP3_Alphabet)))
for i in range(len(prot_tst)):
    subseqs = slidewin(prot_tst[i], W)
    subtarg = sstr_tst[i][W/2:-W/2+1]
    for j in range(len(subseqs)):
        out = nb[subseqs[j]]
        c_targ = DSSP3_Alphabet.index(subtarg[j])
        c_pred = out.prob().index(max(out.prob()))
        cm[c_targ, c_pred] += 1
```

d. From the confusion matrix many metrics can be computed, e.g. Q_3

$$Q_3 = \frac{\text{number of residues correctly predicted}}{\text{total number of residues}}$$

```
VLHQASGNSVILFGSDVTVPGATNAEQAR  amino acid sequence 29 residues
HHHHHCCCCEEEECCCEEECCCCCHHHHH  actual secondary structure
CHHHCCCCEEEECCCCCEEECCCHHHHHH  prediction 1: Q3 = 22/29 = 76%:
HHHHHCCCCHHHHCCCHHHCCCCCHHHHH  prediction 2: Q3 = 22/29 = 76%:
```

Investigate what window size that works best. Report this size. To say what is better, either implement Q_3 or use the generic implementation you find in module `ml`.

```
Qk, Q = Qk(cm, DSSP3_Alphabet)
```

e. Inspect the 5-window classifier to determine the effect Glutamic acid (E), Proline (P) and Glycine (G) have on 3-class secondary structure at different positions in a window. Have a look at the implementation of the classifier (`prob.py`). Note that you can retrieve the relevant probabilities using:

```
print nb.condprobs['H'][2]  # P(aa|ss=H,col=2)
< A=0.11 C=0.01 D=0.05 E=0.09 F=0.04 G=0.04 H=0.02 I=0.06 K=...
nb.condprobs['H'][2]['E']   # P(aa=E|ss=H,col=2)
0.09044209730393894
col = 0
for aa in nb.condprobs['H']:  # P(aa|ss=H,col)
    print "E @ col %d is %5.3f" % (col, aa['E'])
    col += 1
E @ col 0 is 0.081
E @ col 1 is 0.087
...
```

f. Compare these probabilities to the background, *i.e.* frequencies of amino acids in the test and training sequences (see our old friend `Distrib` in `prob.py`). Report the propensities of these three amino acids in the three classes compared to their random distribution.

g. According to your `W = 5` classifier, what is the probability of seeing a glutamic acid at the centre of the window in a Helix, *i.e.*

$P(aa=\text{'E'} \mid ss=\text{'H'}, col=2)$?

h. What is the probability of strand, if there is a Glycine in the first column of a 5-residue window, *i.e.* (you will need to access the "classprob" distribution, see implementation of the class "NaiveBayes")

$P(ss=\text{'E'} \mid aa=\text{'G'}, col=0)$?

i. Now use the classifier with the window size that had the best accuracy. Which is the most probable secondary structure if there is a Proline in the middle column, *i.e.*

$P(ss \mid aa=\text{'P'}, col=\text{W}/2)$?

(Beware: the latter two questions, *h.* and *i.*, require some extra calculation. But you can answer this without presenting positions in a window because the Naïve Bayes classifier treats positions independently given the class.)

Part 2 (3 marks; programming exercises; BINF7000 only)

j. Create a neural network classifier of secondary structure. The main neural network class `NN` is in `ml.py`. However, a neural network requires numeric input and output—not symbols like those in sequences… Indeed, `NN` operates only on `numpy` arrays. To make it easier for you, there is a wrapper class `SeqNN` in `spred.py` to manage sequences, transforming them into numeric "one-hot" vectors (concatenated for "windows" of symbols, stacked for separate examples—converted into `numpy` arrays that are processed by `NN`.)

```
W = 15
nHid = 30
nn = SeqNN(W, Protein_Alphabet, DSSP3_Alphabet, nHid)
```

- Train a neural network using the training data above. The following call takes about a minute on my laptop. Describe the steps that it involves. What does `eta` mean? What does `niter` mean?

```
nn.observeAll(prot_trn, sstr_trn, eta = 0.01, niter = 1)
```

- Cheat and load an already trained NN. It takes a long time to train a big neural network with a large data set. Weights can be saved (after training) and loaded (before testing or more training). The file `sstr3.nn` contains the weights that resulted from 40 iterations through the training data.

```
nn.nn = readNNFile('sstr3.nn') # nn is NN instance in our SeqNN
```

- Evaluate how well it is working. This is simple to do with the wrapper class but you should investigate what is happening under the hood. What steps are involved? What is the neural network's Q_3? Which class is easiest and which is most difficult to predict? (The following call takes half a minute on my machine.)

```
cm = nn.testAll(prot_tst, sstr_tst)    # get a confusion matrix
```

- Literature search: What is a PSI-BLAST profile? Why are such profiles used in secondary structure prediction?

- Literature search: When we created the data sets, we removed all sequences that shared more than 20% sequence similarity. This is sometimes referred to as "sequence redundancy reduction". Why is this a good idea?

Part 3 (2 marks; web and scripting exercises; BINF7000 only)

k. Select any PDB entry—avoiding those that are in the data set above. For example, retrieve "3ZXQ" (PDB database) which was added after I constructed the FASTA files you have been using. Download the atomic description file (known as the "PDB file"). Use DSSP (http://www.cmbi.ru.nl/dssp.html) to find its secondary structure.

How well does your Naïve Bayes and neural network predictors perform on this

particular protein in terms of its secondary structure? Visualize your protein in 3D using PDB's built-in tools or using stand-alone tools such as PyMol, chimera, or others (include a screenshot or a rendered image generated by these software in your report). Manually/informally project predictions to 3D: Identify and discuss what structures that are easy/difficult to identify.