**Project 5**

**Building a Support Vector Machine for Secondary Structure Prediction**

**Introduction**

The purpose of this project is to build and train a support vector machine that can predict protein secondary structure, based on sequence and then build way neural network predictor to measure the accuracy of the predicted data.

**Issues**

Following issues associated with raw data needed to be addressed before implementing the algorithm

* Our input file consist of sequence ID, sequence and predicted secondary structure. So we need to map our sequence ID with sequence and secondary structure.
* Amino acid into numerical form ranging from 1 to 20
* We need to consider that all the characters in our sequence from N to C termini are considered while sliding through th e sequence for this purpose we will prepend and append “X” characters in our original sequence. X will be translated as 21
* Lastly we need to transform our input train data into the following format

+1 11:1 24:1 41:1 79:1 84:1 101:1 121:1 (etc)

<label> <descriptor: value> <descriptor: value> (etc)

Where label represent that either the descriptor in focused is mapped to any of our assigned secondary structure . For example if I am working with coiled(C or T) structure so any time I encounter an amino acid mapped to C or T the label will be 1 for for that position and zero for all the other structures. The descriptor is actually representing the amino acid in focus and values are assumed zero for any descriptor not specified, and the code keeps track of the maximum descriptor index used in any example.

* Collecting the training data margins and validation data margins for each binary classifiers by running libsvm.
* Combination of all binary classifier data(training and validation) into three way data input for neural network i.e 3 columns for labels and 3 columns for margines

**Overview of an Algorithm**

In this project to we are trying to predict secondary structures such as helix, beta and loops. We can build three binary classifier for each one. We decided to follow “Support Vector Machine” approach for this purpose. The reason behind choosing SVM over other learning networks is that they have much more rigorous learning theory then others and also because of the their relative simple construction

SVM is actually a set of methods which tries to classify data into different categories. For a given training data, it will try to find line that is as far away as possible from each of the classes. This line is called the maximum- margin hyperplane.

The dividing line has been chosen so that the parallel lines that touch the items from each class are as far from it as possible. The points near the line are called *support vectors.* The algorithm that finds the support vectors and uses them to find the dividing line is the support-vector machine.

**Implementation of the Algorithm**

**Mapping the sequence with secondary structure**

1. First task is to separate the sequence and the secondary structure from our input data in such a way that a sequence D maps to respective sequence and secondary structure
2. Input file categorizes the secondary structure in following three categories

H= Helix , E = Beta sheet, C,T = coil or Turns.

1. This will return a dictionary in following format

<ID> : [seq, structure]

LECTRONTRANSFER(IRON-SULFURPROTEIN)15-OC': ['MKKYTCTVCGYIYDPEDGDPDDGVNPGTDFKDIPDDWVCPLCGVGKDEFEEVEE', 'CCCEEETTTCCEECTTTCCHHHTCCTTCCHHHCCTTCCCTTTCCCHHHEEECCC']

**Sliding Window**

In order to proceed with this project we will use sliding window approach over the sequence. A window of fixed length (11 residues) which is centered on one residue whose secondary structure we are going to predict. In order to make sure every character from N to C termini is considered we prepend and append some 5 random character such as “X”.

**Indicator Vector Approach and labelling**

In order to convert our input data into the desired format form for *libsvm* we used indicator vector approach. It involves following steps

1. Translation of Amino acid into base number from 1 to 20 and extra character ‘X’ to 21. which is achieved by building a dictionary ‘aaToIndex ’.
2. Once our window length is fixed then it moves over the sequence 11 times and then the position of the residue in center of each window is represented by adding product of window number and 21
3. Since there are three type of secondary structure H/~H, B/~B and C/~C or T/~T. So we will be producing three types of training and validation files one for each type. So I labeled all those position which have H, B and C or T against that particular residue as 1 and rest of them as -1 in their respective file. Which will form our positive and negative examples of our training data.
4. The final format is as follows

1 21:1 42:1 63:1 75:1 90:1 122:1 127:1 148:1 169:1 201:1 224:1

-1 12:1 27:1 59:1 64:1 85:1 106:1 138:1 161:1 174:1 191:1 217:1

**Assigning Training / Validation set**

After getting the vector transformation of our positive and negative examples. It is now ready to be assigned as Training and Validation set. Then we assign them to Training set and validation set with random number threshold 0.80 i.e less than 0.80 will be assigned to training set where as more then will be assigned to the validation set.

**Implementation of the Algorithm**

1. Once we have all the training data and Validation data for each of the binary classifier now it is ready to train svm model and predict margins
2. I separately trained the svm model with each of the binary classifier training data with following parameters

* s = 1 nu sv(multi class Classifier)
* t = 2(kernel type radial basis function)
* nu with threshold 0.25

1. Then I predicted the margins for our training data by using libsvm predicted tool
2. I repeat the same steps as stated in step 3 for predicting the validation set(all three binary classifiers) margins based upon the model created in step 2
3. Once we have all type margins data then we need to combine all the training data labels and predicted margins in one file
4. I performed this by following a unix command paste as follows:

*paste H.training.labels E.training.labels C.training.labels H.nu.margines E.nu.margines C.nu.margines | column -s $'\t' -t > nnTraining.txt*

The final sample output comprised of first their columns of labels of training data and last three columns are their respected predicted margins

-1.0 -1.0 +1.0 -3.562983 -3.958345 4.462895

-1.0 -1.0 +1.0 -1.723684 -2.986452 2.420078

-1.0 -1.0 +1.0 -0.894724 -1.288176 1.677657

1. I repeat the same procedure as in step 6 for combining all the labels and margins related to validation data as ‘nnvalidation.txt’.
2. Now , I trained my neural network(neurolab) with my training data with 3 columns of margin as my input and 3 columns of labels as my output and 10 hidden layer.
3. Once I trained the data I validated the performance of the neural network with my validation set.
4. I converted all the predicted result of my ValidationIN data into rows of 1s(the most positive ) and -1(the negative one).
5. Then I compare the resulted predicted array from step 10 to the ValidationOUT array(original labels) and calculate accuracy.
6. I repeat the step 8 to 11 for different topology of neural network i.e 10,18,3.

**Training with**

Once the training data set was assigned, it is now ready to be trained. I use libsvm module for this purpose. I trained the training data with libsvm train tool with parameters s = 0 which represent multi classifiers classifiers and Kernal type(t = 2) radial basis function. Then I run the libsvm predict tool to predict the labels. I evaluate the performance of the model by running the same tool as

**Performance Measure**

In the end it is time to check the performance of our model. For this purpose I perform various performance measure analysis on validation set, which was developed independently from the training set. I calculate sensitivity, selectivity and accuracy of the validation set writing a script named as performance.py

**Input File**

We obtain our training data from the following website.

[*http://antheprotpbil.ibcp.fr/Rost.html*](http://antheprotpbil.ibcp.fr/Rost.html)

**Sample input**

>1ACX-1ANTIBACTERIALPROTEIN17-DE

APAFSVSPASGASDGQSVSVSVAAAGETYYIAQCAPVGGQDACNPATATSFTTDASGAASFSTVRKSYAGQTPSGTPVGSVDCATDACNLGAGNSGLNLGHVALTF\* CCEEEEECCCCCCCCCEEEEEEECCCCEEEEEEECEETTEECCCTTTCCEEECCCCCCCEEEECCCEEEEECTTCCEEEE EETTTCCCEEEEECCCCCCCCCCCCC\*

**Output**

The result by running performance.py is as follows

sensitivity = 1.000000

accuracy = 0.867784

specificity = 0.780671

**Conclusion**

The results of these tests shows that overall performance on the basis of validation data set is biased. The sensitivity is 100% which mean this model is good in predicting true positives but not so good in predicting true negatives.