University of Southampton

COMP3212 Computational Biology

Secondary Protein Structure Prediction

Author:
Richard Cann Perez
ID: 28207696

Lecturer:
Professor Adam Prugel-Bennett

May 16, 2018

Contents

1 Dataset **2**

2 JPred4 (JNet) **2**

2.1 Testing 3

3 Implemented Algorithm **3**

3.1 Testing and Results 4

A Screenshots **7**

1 Dataset

The chosen dataset was taken from the RCSB Protein Data Bank [1]. The archive contains one text file with close to 250,000 proteins and their matching secondary sequence which were calculated from 'the experimental coordinates' from [2]. The output text file also has proteins in the FASTA format which makes it easier when testing the dataset with different algorithms. The following is a list of the possible codes that a secondary sequence structure has in the given file:

- H = alpha helix
- B = residue in isolated beta-bridge
- E = extended strand, participates in beta ladder
- G = 3-helix (3/10 helix)
- I = 5 helix (pi helix)
- T = hydrogen bonded turn
- S = bend

In order to generalize the classes that each amino acid is classified as, the secondary structures were changed to only having **H** representing all helixes (H, G, I), **E** representing Beta sheets (B, E) and **C** representing coil (T, S) as well as any 'space' or 'X' which indicate irregular structures and difference between residues.

2 JPred4 (JNet)

JPred4 is web server that allows people to input multiple sequences to get a prediction of their secondary protein structure through the use of the JNet algorithm. JNet uses two neural networks in order to make the predictions. The algorithm first feeds the sequence into PSI-BLAST in order to get multiple sequence alignment which is then fed to the first neural network. This first network uses a sliding window of 17 residues with nine hidden nodes and three output nodes. The second network gets the output from the first one, but uses window size of 19 keeping the hidden and output layers the same.

In order to assess the prediction, different profiles were created by feeding different alignment data into the neural networks. Where all profiles agreed on the a prediction the outcome was indicated as a 'jury agreement', while different predictions indicated 'no jury' [3]. Those decisions where not all profiles agreed on were then used to train a separate neural network and its outputs replaced the original 'no jury' decisions.

2.1 Testing

For the testing of JNet I used the JPred4 online server. From the dataset collected I extracted all the FASTA sequence headers as well as the sequences without their secondary structure to input to the server. They then emailed me a file with all the resulting predictions for each sequence. I was only able to submit 150 sequences in one file due to a limit imposed by the server. From the resulting files I parsed the results and compared them against the real results to determine the accuracy of their predictions. After testing the the given files the accuracy I computed was 73.9% which is close to the accuracy mentioned in [3] of 76.4%

3 Implemented Algorithm

For my own prediction algorithm I decided to implement a neural network with the help of Google's Tensorflow [4] package for Python. From the 250,000 sequences collected from the data I used 90% of the data to be used as the training data and of the remaining 10% of data I used 1,000 sequences for testing. Using 10% of testing gave a significant increase in computing time and therefore I had to pick only 1,000. I used random windowing in order to account for the algorithm memorizing a long section of the structure. This was done by picking a random index from the sequence and then picking $s/2$ amino acids to the left of the chosen index and $(s/2) - 1$ to the right of the index, where s indicates the window size.

As suggested during lectures, the representation of proteins was done through a $sequenceLengthX(22^{*}windowSize)$ matrix where each amino acid would have a True value if present on the given window and another matrix of dimensions $sequenceLengthX3$ which represented what class each amino acid belonged to. The use of only one hidden layer was taken from [5] where they experimented with the window size and number of hidden neurons as a modification to the PSIPRED algorithm. The algorithm then uses a softmax cross entropy loss function as opposed to the sigmoid function used in [5] because I aimed to have three output neurons instead of three neural networks outputting one neuron corresponding to each class like they proposed.

A stochastic gradient descent approach for optimization was used through the Adam optimizer. Compared to another common method, sum-of-functions (SFO), Adam tends to be faster and more effective since SFO's stochastic regularization methods tend to show over-fitting when tested in MLP's [6]. The training was done in batches of 100 for a given number of training steps, window size, and number of hidden neurons.

3.1 Testing and Results

The algorithm was tested using different number of training steps, window sizes and number of hidden neurons as proposed in [5].

Training Steps	Accuracy
15	59.3
20	**61.2**
25	61.0

Table 1: Training Steps with same window size

Window Size	Accuracy
13	60.4
15	**61.7**
17	60.8

Table 2: Window Size tests against Accuracy

Hidden Neurons	Accuracy
32	**60.3**
50	59.3
101	59.3

Table 3: Ideal number of neurons on hidden layer

The above tables point out the test for the different parameters used in order to find the highest accuracy. The window sizes and suggested hidden neuron numbers were also taken from [5]. The accuracy of my algorithm was 61.70% at its best which was below JNet. This is due to a couple factors including not using PSI-BLAST in my algorithm to format the matrix into the neural network. PSI-BLAST is able to match sequences using different databases and with the data provided there were some inputs ('X' and ' ') that were unknown.

A difference as to why my algorithm was also significantly lower than the 76.02% accuracy in [5] is that they implemented three neural networks with each corresponding to one class. Each network had its own optimal window size and hidden neurons number which could accommodate better to the prediction of each class by achieving general training properties in each network that are tailored to their corresponding class.

References

[1] Rcsb protein data bank. https://www.rcsb.org/pdb/, 2018.

[2] Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):25772637, 1983.

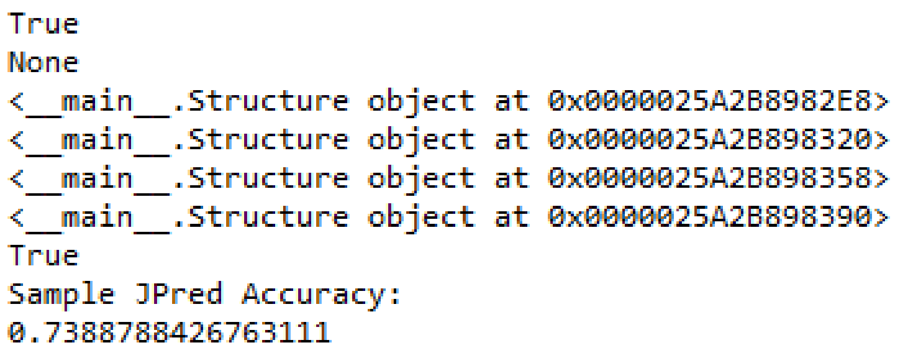
[3] James A. Cuff and Geoffrey Barton. Application of multiple sequence alignment profiles to improve protein secondary structure prediction. 40:502–511, 08 2000.

[4] Tensorflow. https://www.tensorflow.org/, 2018.

[5] V. Dzikovska, M. Oreskovic, S. Kalajdziski, K. Trivodaliev, and D. Davcev. Protein secondary structure prediction method based on neural networks. In *2008 2nd International Conference on Bioinformatics and Biomedical Engineering*, pages 176–179, May 2008.

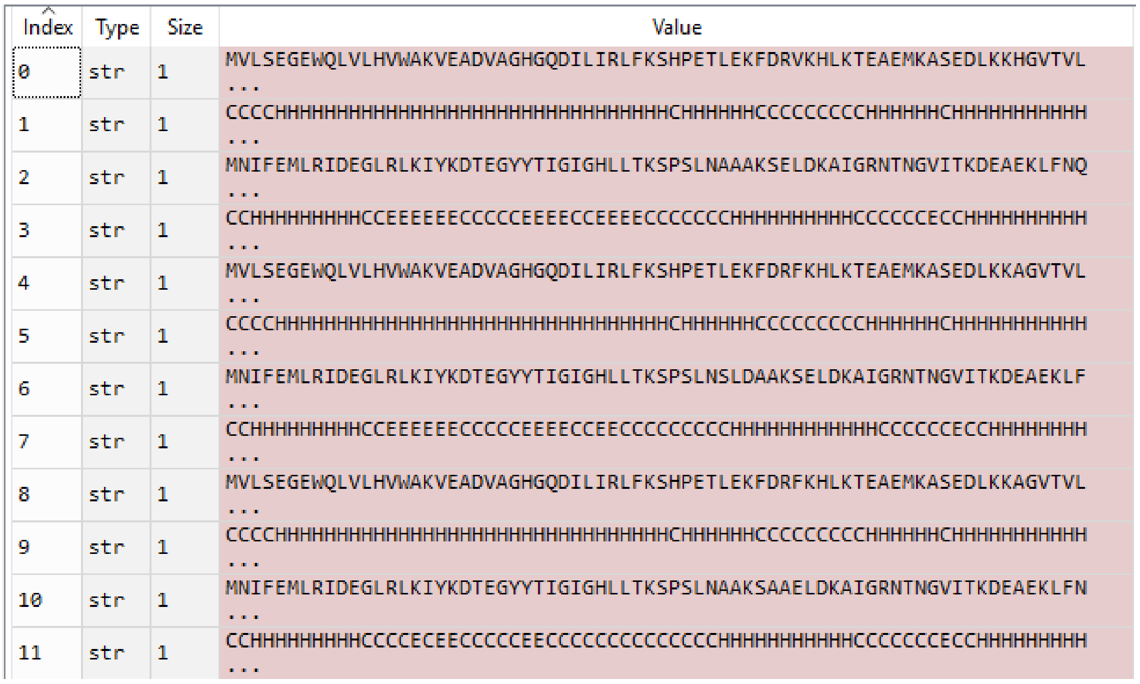
[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

A Screenshots



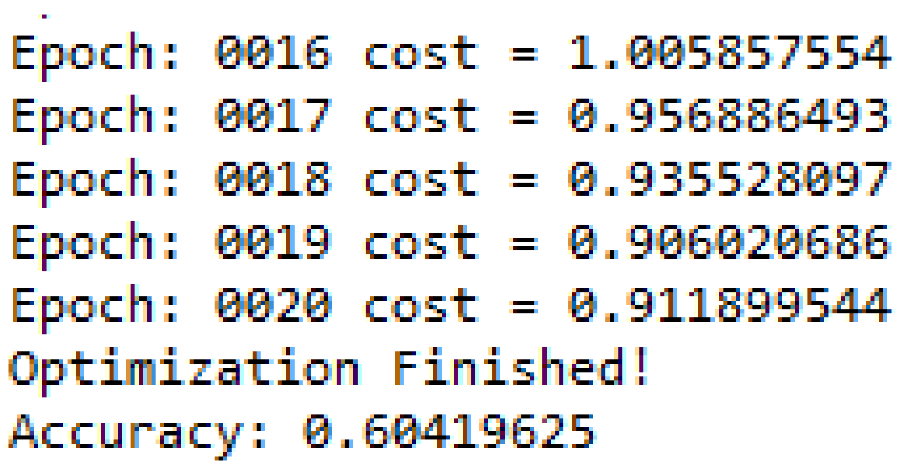
```
True
None
<__main__.Structure object at 0x0000025A2B8982E8>
<__main__.Structure object at 0x0000025A2B898320>
<__main__.Structure object at 0x0000025A2B898358>
<__main__.Structure object at 0x0000025A2B898390>
True
Sample JPred Accuracy:
0.7388788426763111
```

Figure 1: Sample output for JPred Accuracy



Index	Type	Size	Value
0	str	1	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRVKHLKTEAEMKASEDLKKHGVTVL ...
1	str	1	CCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCHHHHHHCCCCCCCCHHHHHHCHHHHHHHHHHH ...
2	str	1	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSELDKAIGRNTNGVITKDEAEKLFNQ ...
3	str	1	CCHHHHHHHHHCCEEEEEEECCCCCEEEECCEEEECCCCCCCHHHHHHHHHHCCCCCCECCHHHHHHHHHH ...
4	str	1	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKKAGVTVL ...
5	str	1	CCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCHHHHHHCCCCCCCCHHHHHHCHHHHHHHHHHH ...
6	str	1	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAKSELDKAIGRNTNGVITKDEAEKLF ...
7	str	1	CCHHHHHHHHHCCEEEEEEECCCCCEEEECCEECCCCCCCCCHHHHHHHHHHHHCCCCCCECCHHHHHHHH ...
8	str	1	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKKAGVTVL ...
9	str	1	CCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCHHHHHHCCCCCCCCHHHHHHCHHHHHHHHHHH ...
10	str	1	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKSAAELDKAIGRNTNGVITKDEAEKLFN ...
11	str	1	CCHHHHHHHHHCCCCECEECCCCCEECCCCCCCCCCCCCCHHHHHHHHHHHCCCCCCCECCHHHHHHHHH ...

Figure 2: Sample Sequences with corresponding secondary structure



```
Epoch: 0016 cost = 1.005857554
Epoch: 0017 cost = 0.956886493
Epoch: 0018 cost = 0.935528097
Epoch: 0019 cost = 0.906020686
Epoch: 0020 cost = 0.911899544
Optimization Finished!
Accuracy: 0.60419625
```

Figure 3: Sample Output of Neural Network