BIEN 410 Individual Project
Thomas Lai 260864949

Representing the data

To develop my own approach, I began by defining the variables that I can use to describe by data set for a classification model. In literature, many models for protein secondary structure classification are based on evolutionary/alignment data, physical properties, and each residue's identity. Because alignment data would not be available, the characteristics that I considered to construct my variables were each residue's identity, their neighbors' identity, and their physical properties. Examples of physical properties include the overall charge or hydrophobicity of the sequence the residue is in, based on the residue type, or just in the region around the residue.

I decided to go with a simple, almost purely statistical representation by using categorical variables for the identity of each residue, the 5 residues before, and the 5 residues after. I hope that this approach can still capture some of the physical properties of the region around each residue. For example, if we find that the presence of lysine or isoleucine near a residue correlates with the residue being an alpha helix, we can suspect that it might have to do with the overall mass around said residue.

Thus, a categorical array ['V', 'A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y'] representing the identity of each amino acid x_0 and its neighbors $x_{-1}, x_1, x_{-2}, x_2, \ldots, x_5$ are appended into one python list of 20*11=220 features:

['V0', 'A0', 'R0', …, 'W0', 'Y0', 'V-1', 'A-1', 'R-1', 'N-1', …, 'W5', 'Y5']

Feature selection

Before I implemented a classification model, I conducted feature selection using LASSO regression, because I suspected that out of the 220 variables, many could have less to little correlation to our alpha helix classification. Briefly, LASSO (Least Absolute Shrinkage and Selection Operator) performs feature selection by minimizing the least-squares cost function

$$\frac{1}{2N_{training}} \sum_{i=1}^{N_{training}} \left(y_{real}^{(i)} - y_{pred}^{(i)}\right)^2 + \alpha \sum_{j=1}^{n} |a_j| \qquad (1)$$

by discarding features that are useless or redundant [1].

Using modified code from [1], I found that after a certain size of the data set, it becomes computationally unfeasible and very inconclusive (no features selected), so I controlled the data set size by performing LASSO on the residues of one sequence at a time ~100-200 data points. The relevant features of each sequence are different, but there was a general trend that the residues closer to the residue in question are more important than the residues further away.

I analyzed how often each feature was selected, compiled the data over the whole training set, and normalized it according to their importance given by the algorithm. The following trends were observed.

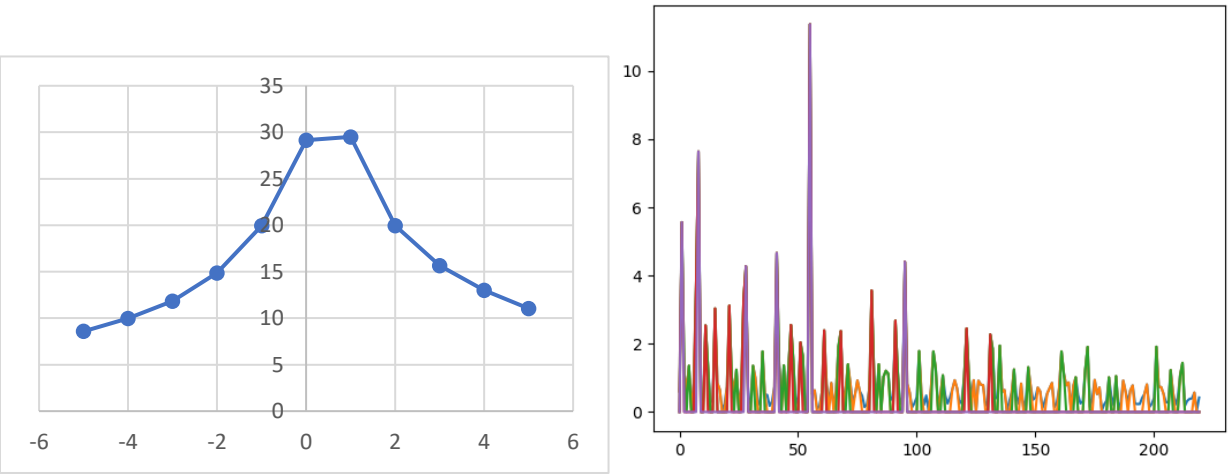

Figure 1. Feature selection results. Left: the amount of times a feature from each location is selected normalized to their importance. Right: the amount of times each feature is selected normalized to their importance. The x-axis is the feature index out of 220. Each colour represent features that are above a certain threshold. (The peak, interestingly, is whether or not the next residue in the sequence is a proline)

From this feature selection exercise, which results were shown in Figure 1, we can hypothesize that the identity of the x_0 residue as well as x_{-1}, x_1, and x_2 are most likely correlated with x_0 being an alpha helix or not. Interestingly, this is shifted to the right, and the residue one after is equally or more important than the actual residue. On the right, we can see how frequent/important each individual features pop up. Based on several arbitrary cutoffs, 6, 19, 30, 60, and 123 most important features can be isolated for training. This feature selection step was somehow validated later during the classification model.

Classification model

For the classification model, I chose logistic regression optimized with gradient descent. The motivation is that it fits nicely with the way I chose to represent the data. Further, compared to Naïve Bayes, it does not assume conditional independence between variables. While the feature selection step could remove redundant features, logistic regression is also theoretically more robust. However, LR has plenty of disadvantages. It takes a significantly longer amount of time to train, making hyperparameter-tuning very tedious, which was also one of the motivations for reducing the amount of input variables. In addition, I found it difficult to control when the data set is too large and variable. Since I also used gradient descent for training (as described in class), the combination is more easily influenced by large and variable data sets, causing it to converge into local minima (although feature selection made it more resistant in this regard).

Briefly, logistic regression minimizes the cost function

$$J(w_1, \dots, w_p, b) = \sum_{i=1}^{m} \log\left(1 + e^{-\{(2y^{(i)}-1)(\sum_{j=1}^{p} w_i x_j^{(i)} + b)\}}\right) \quad (2)$$

by tuning the parameters w_p and b of the equation

$$log\,\frac{p(y = 1|x \dots)}{p(y = 0|x \dots)} = wx \dots + b \quad (3)$$

that relates the input variables x to the probability of each of the binary classification results y=0 and y=1.

Further, in each iteration of gradient descent, the w and b estimates move closer to an extremum by moving in the direction of the first order partial derivatives.

Code

SStrain.py takes the input sequences data and outputs trained parameters for the linear classification model.

First, hyperparameters are defined, including the number of sequences that will be used for training. Next, selected features' indexes are defined, and if variable useSelectedFeatures = True, the code will run a modified version that only trains and output parameters for the specified features.

Next, an assortment of loops and list operations are used to read the input file create a matrix of size (# of residues x 221) which rows represent each individual residue and columns represent the 220 features and whether or not it is a helix.

The next function is the logistic regression. It is essentially a while loop that continuously updates w and b estimates using the equations derived in class until a stopping condition is met.

Finally, the code writes the final parameter values into the parameters.txt file with a parameter in each line.

SSPred.py closely follows the structure of the example code

The first addition is some hyperparameter definition and the same toggle for whether or not a reduced number of features should be considered.

The second major modification is where the prediction is made. The same organizational logic as the example code is used. To make a prediction for each sequence, the code iterates through each residue. In the loop, the residue goes through the following steps:

1. If the residue is part of the first or last 5 of the sequence, a '-' is assigned. This is essentially implementing a model that it is more likely (60 vs 40 percent) for that sequence to be not an alpha helix.
2. For other residues, it is then converted to the 220 element feature array in the same way as in the training code.
3. Next, if the toggle for reduced features is on, the non-selected features are deleted.

4. Then, the log odds of the residue being a helix is calculated using equation (3). If the result is greater than 0, an ‘H’ is appended to the prediction string. If it is less than 0, a ‘-‘ is assigned instead.

Choosing the final model

The model was trained for varying numbers of training set size, number or iterations before stopping, beta factor in gradient descent, and number of features included. Due to the number of time required for each training, the best model was selected by taking an informal DoE approach.

I found that, as I increase the size of the training set used past 1000 sequences, the accuracy of the classifier starts to decrease. It is counterintuitive at first, but it is most likely because the x variables that I defined are not good, universal, and linear predictors. I also found that the optimal number of iterations is 100. Provided that it does not get stuck in a local extremum, the results do not change much if this number is increased further. It can also be decreased to 50 but quality is not always guaranteed. The beta factor just needs to be low enough, so the algorithm is relatively more stable. Due to the summation nature of the algorithm, when training size increases, beta needs to be decreased accordingly.

Here is a list of my highest scoring models:

Relative rank	% achieved	# of features	# of sequenced used in training	# of iterations before stopping	beta	Training time (min)
1	69.26	220 (full)	1000	50	0.001	25
2	69.25	220 (full)	1000	100	0.001	50
3	68.98	154	1000	50	0.001	8.9
4	68.90	123	1000	50	0.001	6.9
5	68.14	30	4000	50	0.000005	10
6	68.05	30	1000	50	0.001	2

The top scoring model uses the full feature set, while the other ones are not far behind. However, when I train model 1 and 2 with larger data sets, the score starts to decrease. I think this is a sign that the full feature set is problematic and can have many interfering or uncorrelated variables. On the other hand, model 5 and 6 which are trained with only the 30 most important features is able to keep its consistency with large data sets. Interestingly, 30 is a good number of features to have in this case, while 60 and 80 are not, although 123 and 154 becomes better again. The 80-feature models (not shown) are trained with all the features in positions [-1,0,1, and 2], but they score lower than 30-feature models which features are selected using LASSO. The 154-feature models are the 80-feature models with additional features selected by LASSO.

Discussion

Logistic regression coupled with gradient descent is far from the optimal method for secondary structure prediction. Although preselecting the features before fitting it to the model improves

the consistency and overall accuracy of this method. I was lucky to have found models 1 and 2 that did well, but many of my other 220-feature (full set) models scored lower.

Other things that I would like to try is running Naïve Bayes with the LASSO-selected features, now that there is an effort to remove redundant features and the conditional independence assumption might hold. Stochastic logistic regression was another method that I would like to try, although I saved time by reducing the number of features used instead.

Further, there is no doubt an abundance of non-linearity in protein sequence data. While LR can be modified to account for non-linearities, it would drastically increase the complexity of computation further, and there are better suited models such as deep-learning models, as the top scoring published papers in literature most often include neural networks to capture all the nuances.

Reference

[1] G. Malato, “Feature selection in machine learning using Lasso regression,” Medium, 05-May-2021. [Online]. Available: https://towardsdatascience.com/feature-selection-in-machine-learning-using-lasso-regression-7809c7c2771a. [Accessed: 30-Nov-2021].