BIOM 5405: Term Project

Classification of Protein Ubiquitination Site

April 17th 2017

<u>Authors:</u>
George Hanna (100887599)
Jinny Lee (101027966)

I. Introduction

The goal of this project is to develop a pattern classification system using the given training dataset in order to accurately predict ubiquitination on a blind test dataset. The given dataset is a collection of protein windows centered on lysine residues. 80% of the dataset is provided with its correctly classified classes: the positive windows correspond to the sites that are known to be ubiquitinated and the negative windows are assumed to be ubiquitinated. Among the dataset, only 30% of the data was labelled and each project team is allowed to request to label up to 1000 unlabelled sites. The 20% of the total dataset is withheld as a blind test set to be identified as ubiquitination. Initial features for each site has been selected by the application of ProtDCal and reduced to 435 features.

We, group 11, chose Hidden Markov models (HMM) at the beginning of the project but soon, realized that the method, was not suited for non-sequential data. We quickly changed to multilayer perceptron as a pattern classifier and genetic algorithm as a feature selector. Multilayer perceptron is a logistic regression classifier and networks with one or more layers between input, hidden and output [1]. A single output from multilayers of inputs is formed according to its input weights and forwards through some nonlinear activation function. The power of multilayer perceptron is its learning ability from a training set and how to relate to the best to the output that an operator wants to predict. Genetic algorithm is derived from the natural selection, the process of biological evolution. At each step of the algorithm, it randomly selects individuals from the present generation and produces the next generation. Over consecutive generations, the 'fittest' survivor among individuals proceeds to the next generation and the successful point in each competition provides a possible solution for solving a problem.

II. Method

1) Data Preprocessing

Several preprocessing steps needed to be taken prior to conducting feature selection and the training of the classifier. The networks trained through MATLAB's Neural Network Toolbox generally employ sigmoid transfer functions in the hidden layers. To avoid saturating these functions during the training phase, MATLAB automatically normalizes the inputs of the dataset.

Another pre-processing step involved the handling of missing values (denoted by -9999 in the dataset file). To ensure the proper classification of samples which include missing values amongst some of their features, the missing values were replaced by the column medians. Outlier detection was another important pre-processing step, which would ultimately enhance the classifier's robustness towards noise. To accomplish this, the mean absolute deviation of values within each feature column were calculated. Values that were more than 10 deviations away from the median were replaced by the median. This process was used as it is less error prone than standard deviations from the mean.

The final pre-processing step involved testing different techniques for mitigating class imbalance. Four methods were examined and compared iteratively during the training and testing of the classifiers. The first technique was the synthetic minority oversampling technique (SMOTE); it was used on 70% of the labelled data to equalize the number of samples in each class. Adaptive SMOTE is second technique which was used; it samples the minority class near the boundary between both classes. The two final techniques explored include penalizing the algorithm for misclassifying positive class samples and undersampling of the majority class. The four methods discussed above were to be evaluated during the testing of the classifier.

2) Feature Selection

For the feature selection method, we identified two methods to increase performance of our classifier (in this case, performance is the optimal precision/recall point). First, we selected CfsSubsetEval in Weka to perform feature selection. This evaluator considers the individual predictive ability of each feature and evaluates a possible subset of attributes [2]. The subset of features are correlated with the class. After a time-consuming and computationally heavy greedy search, we generated 60 selected features. Greedy algorithm provides an optimal choice at each stage, the chosen solution contributes to a solution and iterates. Based on the resulting features from greedy search, we implemented a genetic search feature selection method in Weka to boost the accuracy of the selected features. The genetic algorithm resulted in a similar number of features to the greedy search. However, the individual elements in genetic algorithm are evaluated from high performance 'offspring' from their parents, as a result, this leads to an improvement of the dataset and their features to the given goal [3].

3) Training and Testing the classifier

Optimizing Network Parameters

The unlabelled dataset was divided in a 70-30 split. A feedforward neural network was designed in MATLAB. The number of hidden layers and learning rate were iteratively optimized. A figure depicting the final network structure is shown in figure 1.

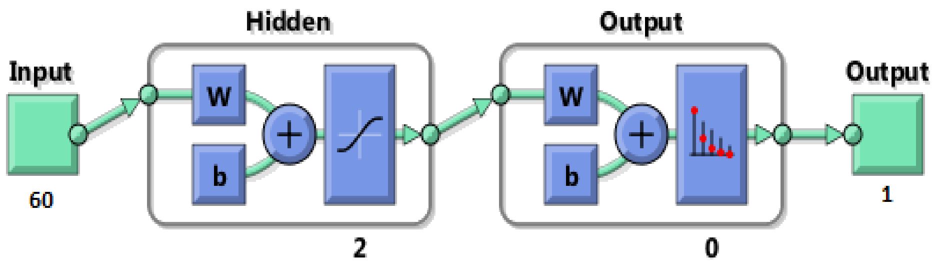


Figure 1: Feedforward Neural Network Structure

Steps were taken to ensure the classifier does not overfit the training set. The number of hidden layers was kept low (between 1 and 3), to avoid over-estimating the complexity of the target problem [4]. A number of training functions were then tested, and were assessed based on the ability of the network to generalize to the hold-out set. These include ‘trainlm’ which is based on Levenberg-Marquardt backpropagation, ‘trainscg’ which is scaled conjugate gradient backpropagation, and ‘trainbr’ which is a bayesian regularization backpropagation algorithm based on Levenberg-Marquardt optimization. ‘Trainbr’ seemed to perform the best, when used with its default training parameters.

Assessing Effectiveness of Class Imbalance Mitigation Techniques

The class imbalance mitigation techniques suggested in section 1 were examined here by cycling through the training and testing phases. When SMOTE and Adaptive SMOTE data were used to train the classifier, in both cases the neural network did not generalize well on the test data. Cost-sensitivity was then tested during the training process, by penalizing false negatives. This resulted in a dramatic increase in recall, and a sharp decrease in precision. Finally, undersampling the majority class was done and this resulted in a modest increase in sensitivity (see confusion matrix in figure 2 for results).

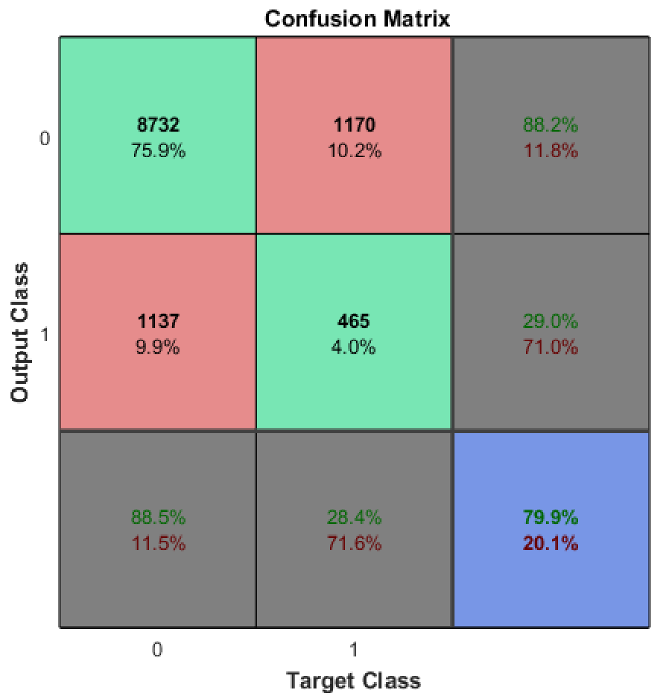
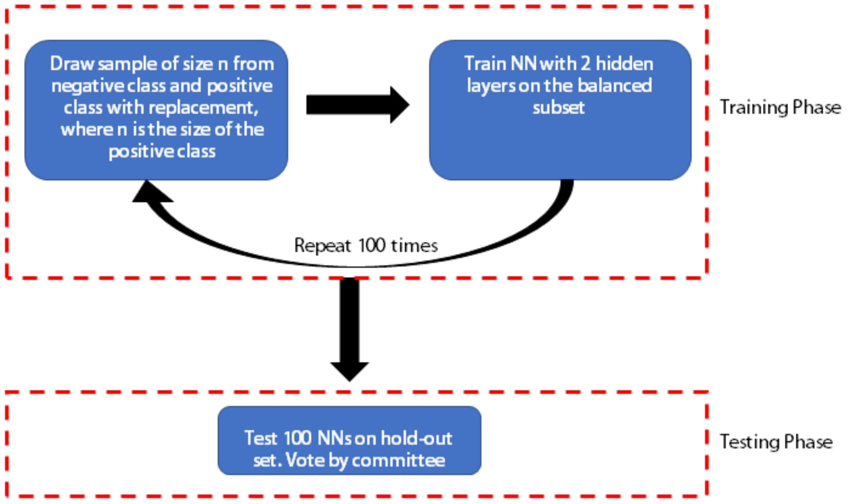


Figure 2: Confusion Matrix Following Training on an Undersampled Dataset

However, to fully benefit from the entirety of the training size, a more complex scheme was required. One idea was to divide the majority class into 6 subsets of the positive class' size (since the class imbalance was 6:1) and to train a network on each subset. This idea will be discussed further under section 4, as it involves the use of meta-learning.

4) Meta-learning Approach

Given the success of undersampling in increasing the performance of the classifier, a variation on bagging was explored to further improve the sensitivity and recall. The diagram below outlines the bagging strategy implemented:


Draw sample of size n from negative class and positive class with replacement, where n is the size of the positive class
Train NN with 2 hidden layers on the balanced subset
Training Phase
Repeat 100 times
Test 100 NNs on hold-out set. Vote by committee
Testing Phase

As the diagram suggests, at the testing phase, the networks are applied to the test set and voting by committee was performed to determine the final decision. The decision threshold of the vote was tweaked to find the optimum operating point along the P-R curve. The use of bagging improved the optimal operating point of the classifier slightly (see confusion matrix in figure 3).

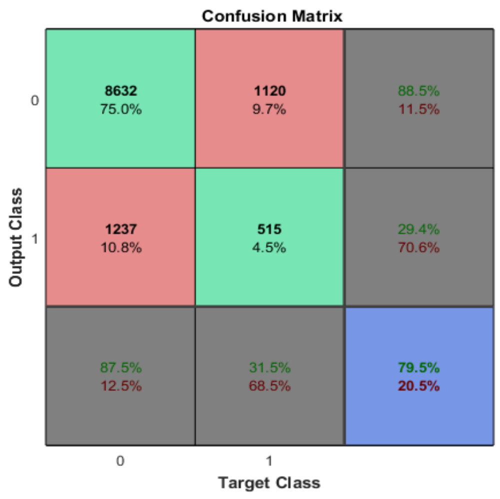


Figure 3: Confusion Matrix Following the Use of Meta-Learning

5) Active Learning Approach

The main goal of active learning is to optimize the performance of a classifier iteratively by accessing data that is otherwise too expensive to access in bulk. In a large dataset like the given data, manual labeling on unlabelled data is expensive hence, learning algorithms can actively query the operator for labels. The unlabelled data was first passed to our ensemble of neural networks, which in turn provided class predictions in the form of probabilities (ranged between 0 and 1). To optimize our classifier's discriminatory ability, we chose voting-based query selection as our active learning method. The points towards which our classifier was most uncertain were sent to the Oracle. Ideally, this would have been done in an iterative manner (i.e. 100 points at a time). However, due to time constraints, 1000 points were requested in bulk. The difference between the traditional way and learning with queries is the query patterns will be mapped to the most informative boundaries than relying on the highest prior probability. This allows the decision boundary more directly using non-parametric techniques. The confusion matrix below shows the difference between before (left) and after (right) using active learning.

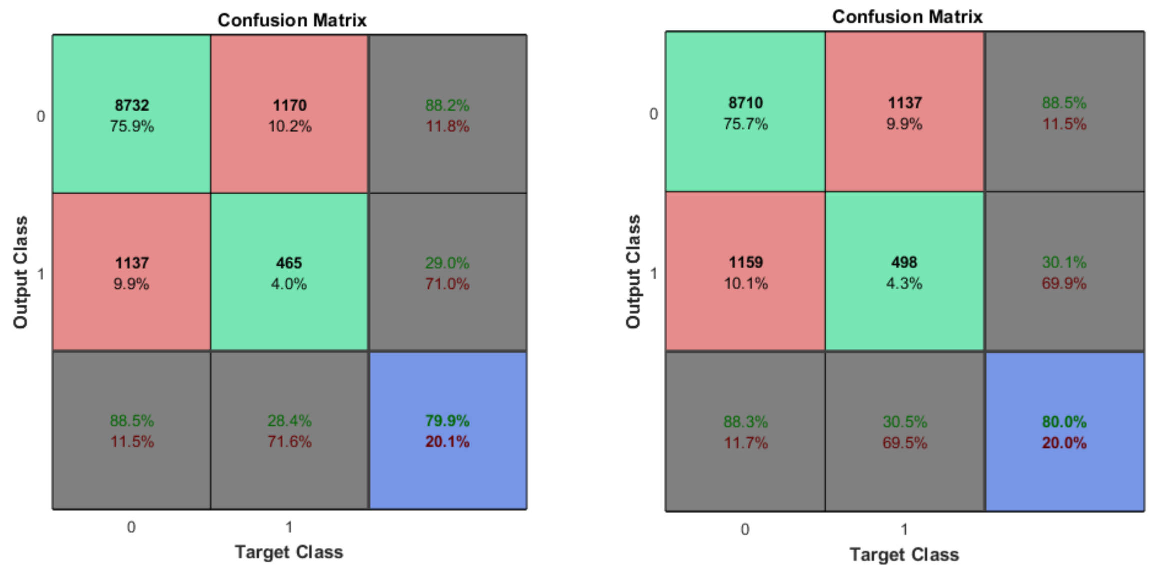


Figure 4: (Left) Results Before Using Active learning. (Right) Results After Using Active learning

III. Results

Using multilayer perceptron as a classifier, genetic algorithm as a feature selector, and voting-based query as an active learning method, the following precision-recall curve was generated.

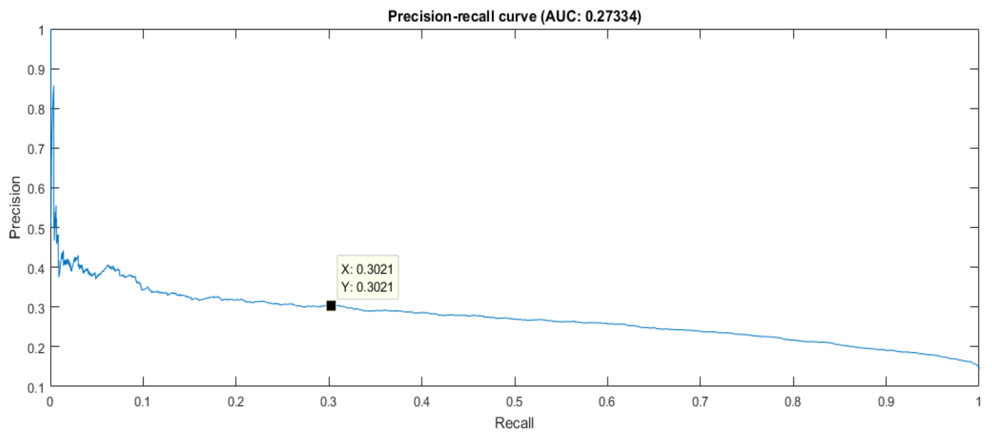


Figure 5: Precision Recall Curve for the Neural Network Ensemble

After fine-tuning our decision threshold, we were able to achieve precision and recall rate similarly: 30.0% and 30.6%. To estimate classifier performance on the blind test set, we used bootstrapping, to generate 200 class imbalanced samples from the hold-out test set. The classifier's precision at a recall rate of 30% was computed at each step. The histogram below shows the distribution of the results.

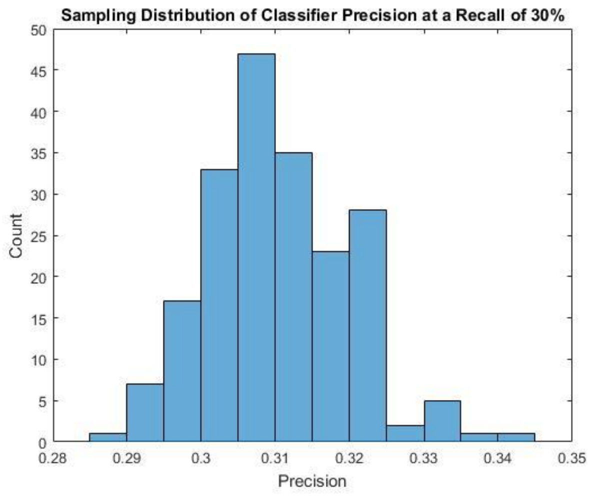


Figure 6: Sampling Distribution for the Classifier Precision at a Recall Rate of 30%

IV. Discussion and Conclusion

The objective of the project was to achieve the highest optimal balance between precision and recall in predicting protein ubiquitination sites. Several observations can be made, following the development process of the classifier. First, it was noted that preprocessing data to replace outliers and missing values was key prior to conducting the feature selection process. Extreme values and/or missing values forced the selection method to avoid the feature altogether. Additionally, neural networks are known to be sensitive to noise especially if they are overfit to training data containing already high levels of noise.

With regards to the handling of class imbalance, the training process needed to be done carefully to avoid overfitting. Given that most training cost functions rely on minimizing MSE, which in this case is not representative of classifier performance, the networks were given a small number of hidden layers and a short amount of time for training (through early stopping and regularization). Additionally, synthetic sample generation was found to be surprisingly ineffective. One suspects this is due to the high degree of overlap between the classes and the amount of interpolation done to replace missing data. The undersampling method adopted solved this issue and allowed for the easy incorporation of meta-learning. According to literature, a modification that would have further improved the voting system used would have been the use of genetic algorithms to assign weights to the votes of individual networks (i.e. networks with better P-R curves would have a stronger voting weight) [5]. Finally, it is worth noting the importance of randomization in assessing generalization of the classifier. Given that the population parameters were unknown, bootstrapping allowed for the construction of the confidence interval of our classifier.

REFERENCES

[1] R. David, “Learning representations by back-propagating errors”, Nature Vol. 323 9 October 1986

[2] M. A. Hall (1998). Correlation-based Feature Subset Selection for Machine Learning. Hamilton, New Zealand

[3] V. Haleh, “Feature selection method: Genetic algorithm vs. Greedy-like search”, National Science Foundation

[4] Y. Liu and J. Starzyk, "Optimizing number of hidden neurons in neural networks", in IASTED International Conference on Artificial Intelligence and Applications, 2007, pp. 121-126.

[5] Z. Zhou J. Wu Y. Jiang S. Chen "Genetic algorithm based selective neural network ensemble" *Proc. 17th Int. Joint Conf. Artificial Intelligence.* vol. 2 pp. 797-802 2001