TERM PROJECT REPORT

BIOM/SYSC5405 – Pattern Classification and Experiment Design

Classification of Methylated protein structures Using Radial Basis Function Neural Networks

Prepared for – Prof. Jila Hosseinkhan

Prepared By

Yujie Yao	101203790
Zixuan Liu	101235609
Ragunath Anbarasu	101166517

1. Abstract

This article is a report about the process of distinguishing protein lysine methylation. The function of a methylated protein will be changed and thus causing acute effect to its stability. For each site, we are provided with 29 descriptors. Based on the training dataset and calibration dataset, our group is required to develop a classifier to determine whether a site is methylated or not. The method our group chose is Radial Basis Function (RBF) Network. And we also tried Adaptive Boost (AdaBoost) as our meta learning method to observe its effect. After the classifier is developed, we used resample to predict Pr@Re50 on the given blind test. We used python and Weka (Waikato Environment for Knowledge Analysis) for our project.

2. Introduction

Our group chose Radial Basis Function (RBF) Network for the regular classifier and AdaBoost for meta learning algorithm. In [1], the authors also applied RBF Networks to predict protein interaction sites using sequence features, which achieves a great success. More importantly, they compared RBF Networks with neural networks and support vector machine to show its advantage. In our project we explored various data preprocessing and meta learning strategies and decided on the optimal solution for the given problem.

Radial basis function classification

Compared to other neural networks, RBF network has a faster learning speed. Figure 1 shows the structure of RBF network. In the hidden layer, we define a certain number of neurons, each with a radial basis function. During the training process, unsupervised algorithm is applied for finding center point and variance of each basis function and supervised algorithm is applied for finding the weights. Normally, a Gaussian function is used as the transfer function in computational units. The locations of Gaussian neurons in RBF network are commonly determined by clustering. The classification performance can be improved by performing class-specific clustering [2]. The training completion is controlled by the calculated error or number of training iterations.

Meta learning

AdaBoost is easy to implement and can be applied to any classifier. It aims to train different weak classifiers for the same training set, and then combine these weak classifiers to form a stronger final classifier.

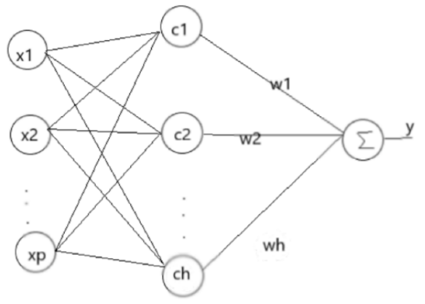


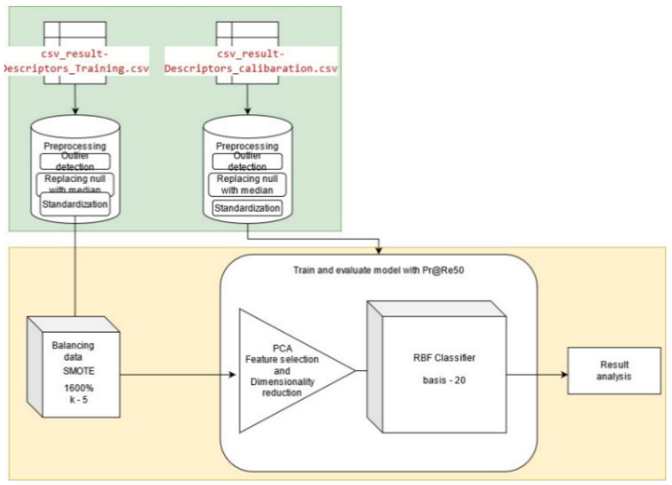
Figure 1: Structure of RBF Network

During its iterations, the classifier will increase the weight of samples that are misclassified and reduces the weight of samples that are correctly classified, making the classifier pay attention to those samples that are difficult to classify in subsequent iterations.

3. Methodology

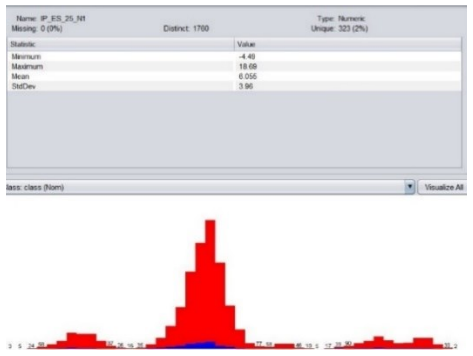
Experimental Design

As soon as we received the training and calibration data, we performed explorative data analysis (EDA) using WEKA and Python. After exploration we decided on the workflow as shown in Figure 2. The Figure 3 shows an overview of one of the features in the given training data set. One important criterion noticed in the dataset is the class imbalance. Figure 4 shows the number of P and N samples present in the given training data. It is clear that the data is highly imbalance.



csv_result-Descriptors_Training.csv
csv_result-Descriptors_calibaration.csv
Preprocessing
Outlier detection
Replacing null with median
Standardization
Preprocessing
Outlier detection
Replacing null with median
Standardization
Train and evaluate model with Pr@Re50
Balancing data SMOTE 1600% k - 5
PCA Feature selection and Dimensionality reduction
RBF Classifier basis - 20
Result analysis

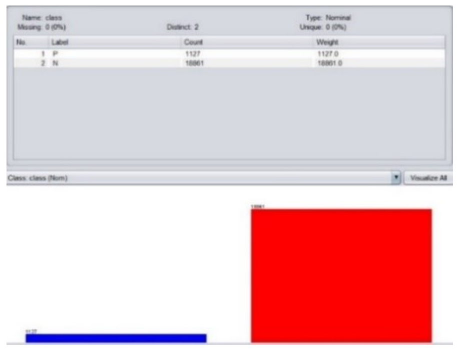
Figure 2: Workflow



Name: IP_ES_25_N1
Missing: 0 (0%)
Distinct: 1700
Type: Numeric
Unique: 323 (2%)

Statistic	Value
Minimum	-4.49
Maximum	18.69
Mean	6.056
StdDev	3.96

Figure 3: Histogram Sample



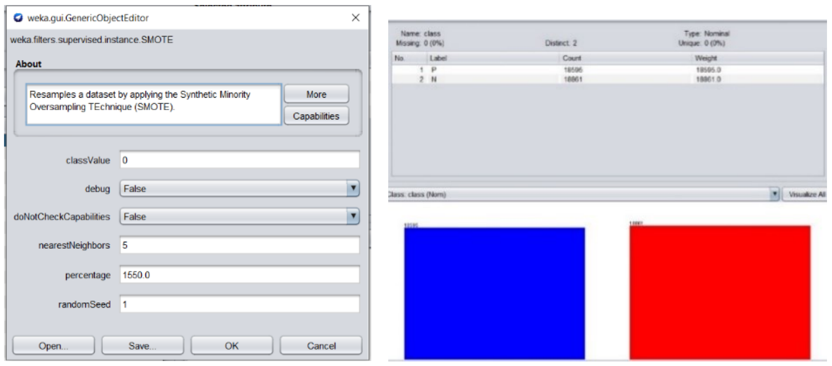
Name: class
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	P	1127	1127.0
2	N	18861	18861.0

Figure 4: Class imbalance

Preprocessing

After performing EDA on the data, we defined the preprocessing pipeline as per the requirements. We performed preprocessing in Google Colab using python. The green block in Figure 1 shows the preprocessing pipeline used on the data. We used Inter Quartile Range (1.5 IQR) for detection of outliers. For the final model we decided not to remove any samples instead replace the outliers and null values with the median value of the feature. After taking care of the outliers, we standardized the data using normalization technique to have the dataset scaled and shifted. We applied Synthetic Minority Oversampling TEchnique (SMOTE) available in WEKA to balance the data as shown in Figure 5. Histogram of the feature shown in Figure 3 after preprocessing and applying SMOTE is shown in Figure 6.



weka.gui.GenericObjectEditor

weka.filters.supervised.instance.SMOTE

About

Resamples a dataset by applying the Synthetic Minority Oversampling TEchnique (SMOTE).

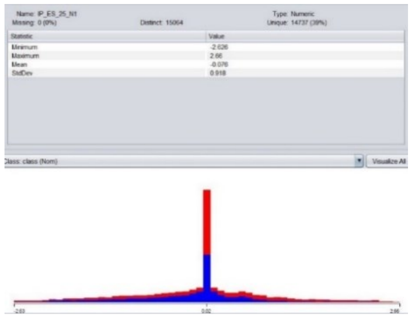
classValue	0
debug	False
doNotCheckCapabilities	False
nearestNeighbors	5
percentage	1550.0
randomSeed	1

Name: class
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	P	18596	18595.0
2	N	18861	18861.0

Class: class (Nom)

Figure 5: SMOTE parameters and results



Name: IP_ES_25_N1
Missing: 0 (0%)
Distinct: 15064
Type: Numeric
Unique: 14737 (39%)

Statistic	Value
Minimum	-2.626
Maximum	2.66
Mean	-0.076
StdDev	0.918

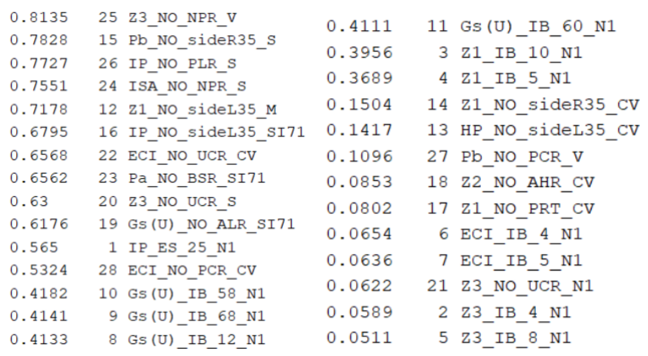
Class: class (Nom)

Figure 6: Data after standardization

4. Training

After we completed data preprocessing, we started to train our RBF classifier. We decided to use the training data for training and the calibration data for testing. Firstly, we tried to perform PCA as our feature selection method in 'preprocess' page of Weka. This can allow us to choose the number of the attributes according to each attribute's standard deviation. But we did not know how to apply the same PCA to our calibration dataset. So, we also tried Info Gain Attribute as another feature selection method. The calculated information gain of each attribute is shown in Figure 7.

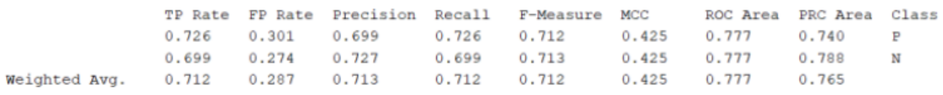
We then fixed the number of attributes to 21 and see how the number of functions and tolerance will affect the precision. We set the number of functions to 5, 6, 10, and the corresponding precision is 0.713, 0.715 and 0.745, as shown in Figure 8, 9, 10.



```
0.8135   25 Z3_NO_NPR_V            0.4111   11 Gs(U)_IB_60_N1
0.7828   15 Pb_NO_sideR35_S        0.3956    3 Z1_IB_10_N1
0.7727   26 IP_NO_PLR_S            0.3689    4 Z1_IB_5_N1
0.7551   24 ISA_NO_NPR_S           0.1504   14 Z1_NO_sideR35_CV
0.7178   12 Z1_NO_sideL35_M        0.1417   13 HP_NO_sideL35_CV
0.6795   16 IP_NO_sideL35_SI71     0.1096   27 Pb_NO_PCR_V
0.6568   22 ECI_NO_UCR_CV          0.0853   18 Z2_NO_AHR_CV
0.6562   23 Pa_NO_BSR_SI71         0.0802   17 Z1_NO_PRT_CV
0.63     20 Z3_NO_UCR_S            0.0654    6 ECI_IB_4_N1
0.6176   19 Gs(U)_NO_ALR_SI71      0.0636    7 ECI_IB_5_N1
0.565     1 IP_ES_25_N1            0.0622   21 Z3_NO_UCR_N1
0.5324   28 ECI_NO_PCR_CV          0.0589    2 Z3_IB_4_N1
0.4182   10 Gs(U)_IB_58_N1         0.0511    5 Z3_IB_8_N1
0.4141    9 Gs(U)_IB_68_N1
0.4133    8 Gs(U)_IB_12_N1
```

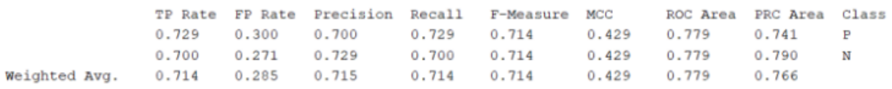
Figure 7: Calculated information gain

We then set the tolerance from e-4 to e-7, and the precision remained the same. It is obvious that the parameters we need to change is the number of functions. However, all the models we tried with Info Gain Attribute was not as effective as the model with PCA. Finally, we found another way to perform PCA and not worrying about applying same PCA on both training dataset and calibration dataset.



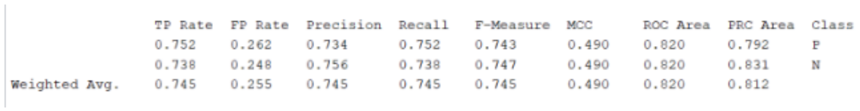
```
                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.726    0.301    0.699      0.726    0.712      0.425    0.777     0.740     P
                 0.699    0.274    0.727      0.699    0.713      0.425    0.777     0.788     N
Weighted Avg.    0.712    0.287    0.713      0.712    0.712      0.425    0.777     0.765
```

Figure 8: numFunctions = 5



```
                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.729    0.300    0.700      0.729    0.714      0.429    0.779     0.741     P
                 0.700    0.271    0.729      0.700    0.714      0.429    0.779     0.790     N
Weighted Avg.    0.714    0.285    0.715      0.714    0.714      0.429    0.779     0.766
```

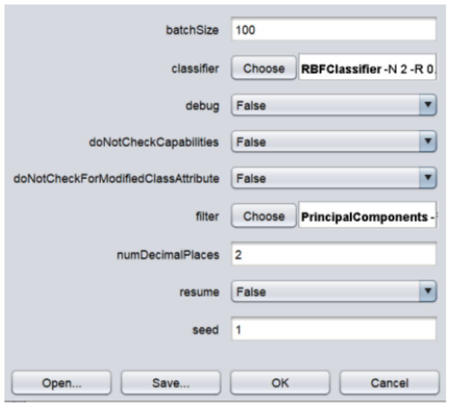
Figure 9: numFunctions = 6



```
                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.752    0.262    0.734      0.752    0.743      0.490    0.820     0.792     P
                 0.738    0.248    0.756      0.738    0.747      0.490    0.820     0.831     N
Weighted Avg.    0.745    0.255    0.745      0.745    0.745      0.490    0.820     0.812
```

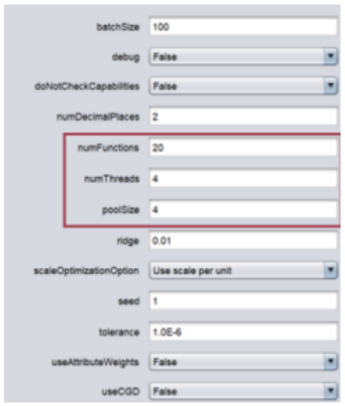
Figure 10: numFunctions = 10

In ‘Classify’ page of Weka, we selected FilteredClassifier in meta column. Then after clicking FilteredClassifier, we can choose RBF classifier as our regular classifier and PCA as our feature selection, as shown in Figure 11.



```
batchSize                          100
classifier                         Choose  RBFClassifier -N 2 -R 0.
debug                              False
doNotCheckCapabilities             False
doNotCheckForModifiedClassAttribute False
filter                             Choose  PrincipalComponents -
numDecimalPlaces                   2
resume                             False
seed                               1
```

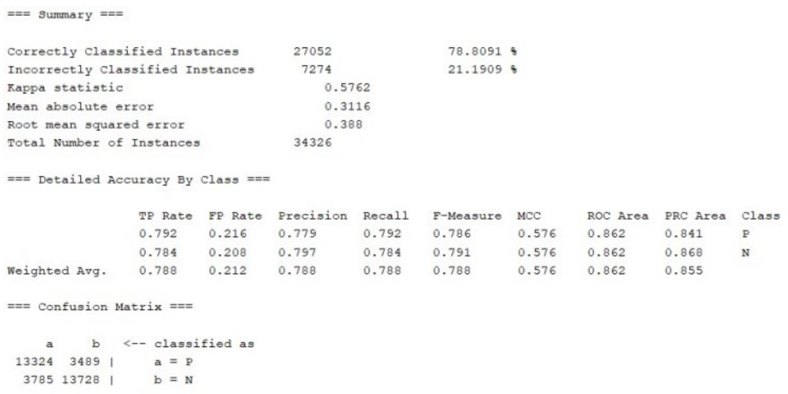
Figure 11: FilteredClassifier Settings



```
batchSize               100
debug                   False
doNotCheckCapabilities  False
numDecimalPlaces        2
numFunctions            20
numThreads              4
poolSize                4
ridge                   0.01
scaleOptimizationOption Use scale per unit
seed                    1
tolerance               1.0E-6
useAttributeWeights     False
useCGD                  False
```

Figure 12: RBF Classifier Settings

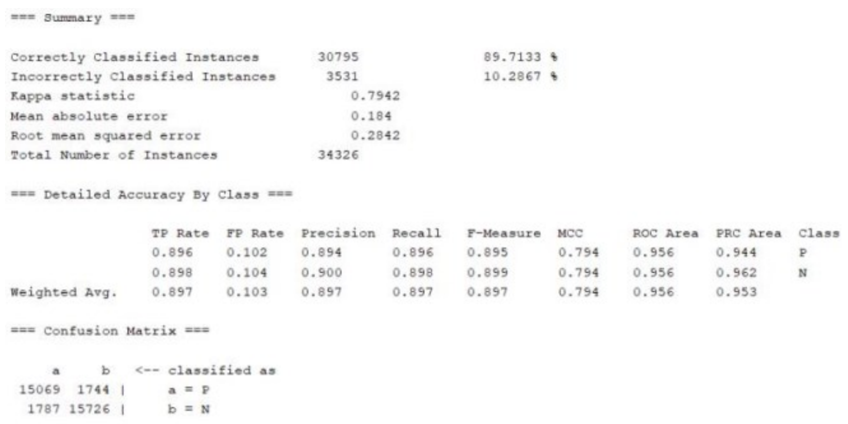
For the parameter of PCA, we only change the number of maximum attributes to 10 and others are all default values. During our whole training process, 10 remains the same and we only need to find the best parameters for RBF classifier. After performing PCA, we get 25 PCA attributes. For the RBF Classifier, the parameters are shown in Figure 12. The reason why we set numThreads and poolSize to 4 is that we can accelerate the building speed of the model. It took 125.65 seconds to build the classifier and the results are shown in Figure 13.



```
=== Summary ===

Correctly Classified Instances       27052               78.8091 %
Incorrectly Classified Instances      7274               21.1909 %
Kappa statistic                          0.5762
Mean absolute error                      0.3116
Root mean squared error                  0.388
Total Number of Instances            34326

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.792    0.216    0.779      0.792    0.786      0.576    0.862     0.841     P
                 0.784    0.208    0.797      0.784    0.791      0.576    0.862     0.868     N
Weighted Avg.    0.788    0.212    0.788      0.788    0.788      0.576    0.862     0.855

=== Confusion Matrix ===

     a     b   <-- classified as
 13324  3489 |     a = P
  3785 13728 |     b = N
```

Figure 13: PCA attributes = 10, numFunCtions =20

After the blind dataset was released, we found that we need to use dataset without removing outliers, so we adjusted the model a little bit. The number of functions was set to 50 and we found that by changing scale optimization option to per unit per attribute can improve the precision. The results are shown in Figure 14.



```
=== Summary ===

Correctly Classified Instances       30795               89.7133 %
Incorrectly Classified Instances      3531               10.2867 %
Kappa statistic                          0.7942
Mean absolute error                      0.184
Root mean squared error                  0.2842
Total Number of Instances            34326

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.896    0.102    0.894      0.896    0.895      0.794    0.956     0.944     P
                 0.898    0.104    0.900      0.898    0.899      0.794    0.956     0.962     N
Weighted Avg.    0.897    0.103    0.897      0.897    0.897      0.794    0.956     0.953

=== Confusion Matrix ===

     a     b   <-- classified as
 15069  1744 |     a = P
  1787 15726 |     b = N
```

Figure 14: Final Results

Then we applied AdaBoost to see if it will improve the precision. The parameter settings are all default values and the precision we got is 0.84. Applying AdaBoost gave a marginal improvement in the classifier performance on training data while taking four times more duration for training .

5. Testing

Two tests were applied to both the trained RBF network with and without AdaBoost. One is applying the models over the entire calibration dataset in order to test how well the classifiers perform on new dataset. Another test uses resampling on the calibration dataset, we use this method to collect enough samples of Pr@Re50 of the classifier in order to predict a distribution of our prediction on the classifier’s Pr@Re50.

5.1 Re-evaluate

We first standardized calibration set. In Weka, we select this standardized set as our supplied test set and let each model do re-evaluating on it. The result provides us sufficient information on the models' performance, including confusion matrix, the rate of correctly classified instances, Precision-Recall curve and ROC curve.

5.2 Resampling

In Weka, we did resampling on standardized calibration set. Each time, we set random seed to a different value, apply it, we would have a sub dataset whose size is half of the original one, but the ratio of two classes is unchanged. We let our models re-evaluate on the resampled set. Each time we are able to view the PR curve and find out the maximal precision at recall greater or equal to 50%. We repeat this process multiple times, and we get a sequence of Pr@Re50. Then, assume these samples reflect the distribution of our prediction on Pr@Re50, we computed the mean and standard variation of our predicted score1.

6. Result analysis

6.1 Re-evaluate

The confusion matrices are shown below:

P	N	←Classified as
67	214	P
768	3947	N

Table 1: RBF Classifier

P	N	←Classified as
70	211	P
874	3841	N

Table 2: AdaBoost RBFClassifier

From these two matrices we can see, the number of TP instances increases a little bit by using meta-learning. If we aim to focus more on how well the classifier can distinguish Positive samples, the AdaBoost model is better. We can see that the AdaBoost model also has much more numbers of FP instances. Additionally, in RBF model, the ratio of our predicted labels are closer to the ratio of true labels. These two facts indicate that AdaBoost model is more biased toward Positive. Thus, in general, if we wish the classifier to have less bias, the original RBF model would be better.

The following results show the Precision-Recall curves.

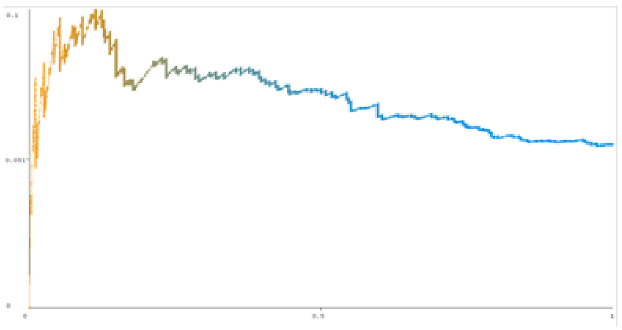


Figure 15: RBF Classifier

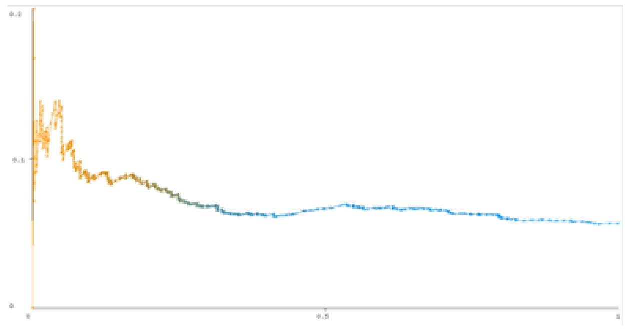


Figure 16: AdaBoost RBF Classifier

The area under PR curve of RBF model with respect to Positive is 0.073, and of AdaBoost model is 0.072. Therefore, from this perspective, the performances of both models don't have much difference.

In total, the rate of correctly classified instances of RBF model is 80.34%, and this rate of AdaBoost RBF model is 78.28%, so in general original RBF model has better performance. Also, even we wish to focus more on finding Positive instances as accurately as possible, the performance of AdaBoost model is just slightly better. Considering the huge amount of time training an AdaBoost model, which is ten times the time of training an RBF model, we believe that it is not worthy just to gain such little improvement. Hence in conclusion, we have decided to select the original RBF model as our final classifier.

6.2 Resampling

We did a twenty-time resampling and get 20 samples of the value of Pr@Re50 on the models. Using MATLAB, we computed that our prediction on Pr@Re50 of RBF model could have a normal distribution with mean=0.0629, std=0.0053, and as for AdaBoost model, our prediction has mean=0.0640, std=0.0035. This is consistent with the areas under PR curve of two models.

In conclusion, even we wish to focus more on finding Positive instances as accurately as possible, the performance of AdaBoost model is just slightly better. Considering the huge amount of time training an AdaBoost model, which is ten times the time of training an RBF model, we believe that it's not worthy just to gain such little improvement. Hence in conclusion, we have decided to select the original RBF model as our final classifier.

6.3 Comparing with true accuracy

From the file on CULearn, we have been acknowledged the true Pr@Re50 of our RBF model when applying on blind dataset, which is 0.0586, and the position of true value of Pr@Re50 in our predicted distribution is shown below, which suggests that our prediction is pretty accurate.

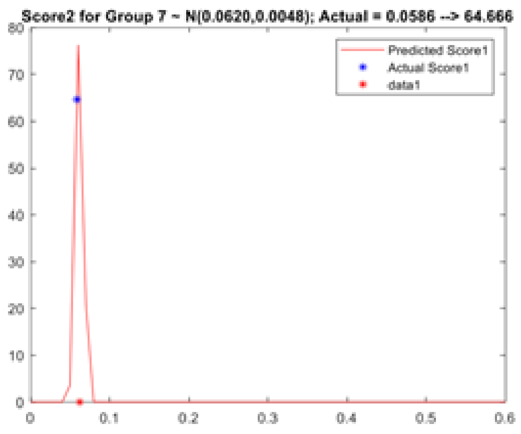


Figure 17: Comparison of predicted accuracy with true accuracy

6.3 Comparing with true accuracy

From the result of re-evaluate on the whole calibration set, we have gained information of the prediction probability of our RBF model on each instance, i.e. the probability that the model classify each instance to a label. We collected these probabilities, and for instances that have been wrongly classified, we replace its probability p with 1-p, i.e. the probability that these samples are classified to their true labels. Then all these probabilities form a sequence of prediction scores, and we compare these scores with the scores of a random guesser.

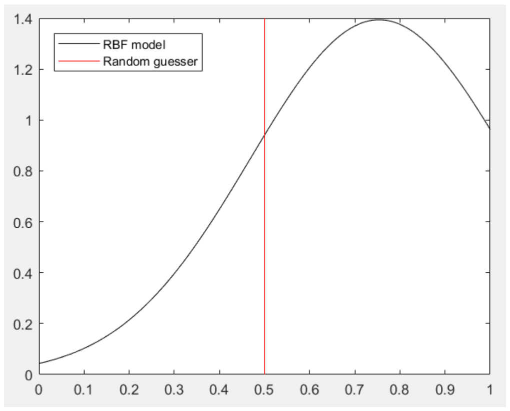


Figure 18: Score of RBF vs Score of Random Guesser

The distribution of prediction scores of RBF model has mean=0.7538 and std=0.28, and since the distribution is constrained in the interval [0,1], we have derived that our RBF model performs better than a random guesser with confidence coefficient approximately 0.81.

7. Conclusion

With such imbalance in data, we faced multiple challenges during classifier training. We mitigated the class imbalance and applied optimal parameters for training the classifier. Applying the classifier on the blind data set yielded results as expected. After looking at the results published as shown in Table 3, we realized we could have improved our classifiers accuracy by employing different feature selection methods. When performing PCA for feature selection, one shortcoming we faced is that we could not choose the number of the attributes we use to train the classifier. However, overall performance of the classifier gave our team the first place in the competition, we could still improve our accuracy.

#	Members	Predicted Score1	Actual Score1	Score2
7	RagunathA, ZixuanL, YuijeY	0.062 +/-0.005	0.058599	64.6659

Table 3: Published results for competition

Reference

[1] Y. Chen, J. Xu, B. Yang, Y. Zhao, and W. He, “A novel method for prediction of protein interaction sites based on integrated RBF neural networks,” Computers in biology and medicine, vol. 42, no. 4, pp. 402–407, 2012, doi: 10.1016/j.compbiomed.2011.12.007.

[2] J. Raitoharju, S. Kiranyaz, and M. Gabbouj, “Training Radial Basis Function Neural Networks for Classification via Class-Specific Clustering,” IEEE transactions on neural networks and learning systems, vol. 27, no. 12, pp. 2458–2471, 2016, doi: 10.1109/TNNLS.2015.2497286.