MHI3000 Big Data Analytics for Health Care Final Project Report

Zhongtian Fred Qiu (1002274530)

<http://gallery.azureml.net/Details/81ddb2ab137046d4925584b5095ec7aa>

# 1. Data pre-processing

The data given to use is quite huge and along with many problems I need to consider.

1) Many missing data, e.g. AVG\_Income, we need to determine by which way to process those empty spots. 2) There might be duplicate data, should I remove the duplicate or not? 3) Some variable only contains one value. E.g. Test\_F. 4) With large number of variables, shall we need to do dimensionality reduction for less computational cost?

Thus, I’ve done following pre-processing things before getting started:

1. Try different ways of replacing missing value and it turns out that replacing missing with 0 is the best way for my validation set AUC. However, I think for variable AVG\_Income, Distance, we should use replace empty with mean or median instead of 0 is much more making sense.
2. I found there is no difference if removing duplicates or not because luckily we don’t have duplicates in our dataset. Even if there are duplicates, I suppose real life patients obtain all those so it’s applicable to use anyway.
3. The only variable I need to remove is Test\_F. You can see my remove reason in

4.4 of my report. As for some variables that only contains very few options (missing value proportion > 0.5) of values or only few spot contains value, I found they are actually incredibly significant for the whole model. E.g. Nausia\_Score. As for Test\_F, it’s reasonable to eliminate this column.

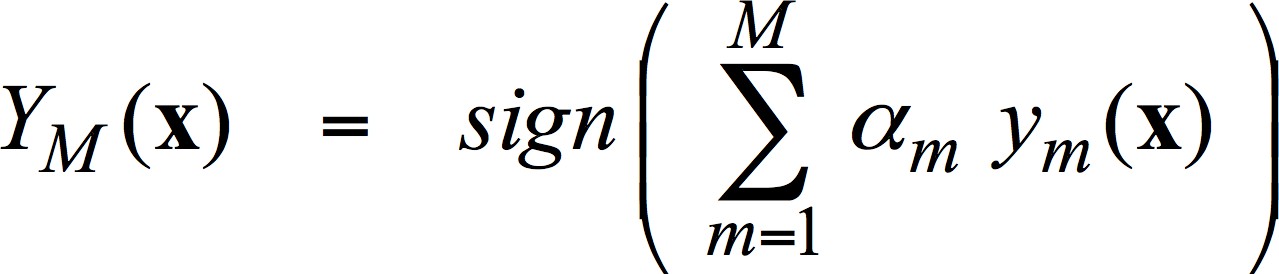
1. Essentially, PCA is a process of retaining most important information by project original data to a lower dimensionality. Also, PCA isn’t a robust to outliers. Thus, PCA will probably lower the final AUC and I didn’t use it.

# Model Selection

Among all those classifiers, I experimented with them all and sum up a table to illustrate why I choose ***two-class boosted decision tree*** over the rest choices.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Accuracy** | **AUC** | **Parameters** |
| **Two-class Average Perception** | 0.974 | 0.979 | 0.1/10 |
| **Two-class Bayes Machine** | 0.913 | 0.904 | 30 |
| **Two-class Decision forest** | 0.969 | 0.967 | 8/32/128/1 |
| **Two-class Decision Jungle** | 0.969 | 0.957 | 8/32/128/2048 |
| **Two-class Neural Network** | takes forever |  | 100 hidden units & 100  iters |
| **Two-class Locally-Deep SVM** | 0.973 | 0.963 | 3/0.1/0.01/0.01/1 |
| **Two-class SVM** | 0.974 | 0.976 | 1/0.001 |
| **Two-class boosted decision tree** | 0.975 | 0.985 | 20/10/0.2/100 |
| 0.976 | 0.985 | 20/10/0.1/200 |
| **0.977** | **0.987** | **20/10/0.01/500~700** |
| **Multi-class Methods** | Unapplicable |  |  |

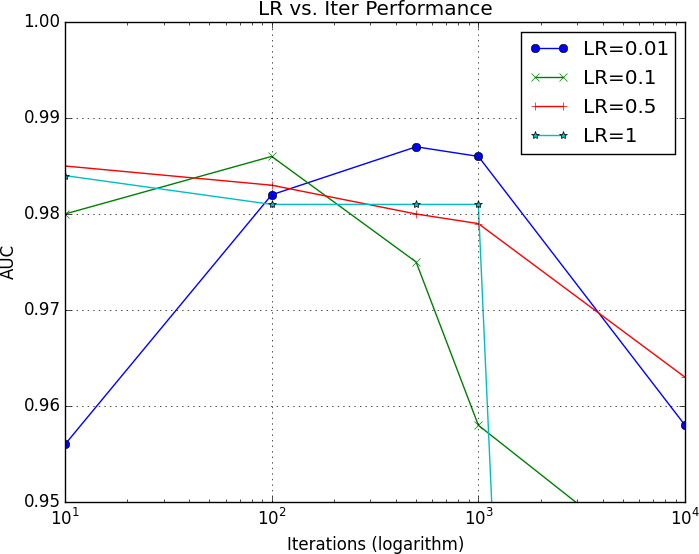
Two-class boosted decision tree is a kind of boosting ensemble methods that consist of weak learns(i.e. decision tree). Every classifier trains one after another and every classifier depends on the previous one. First train the base classifier on all the training data with equal importance weights on each case. Then re-weight the training data to emphasize the hard cases and train a second model. Finally, use a weighted union of all the models for the test data.



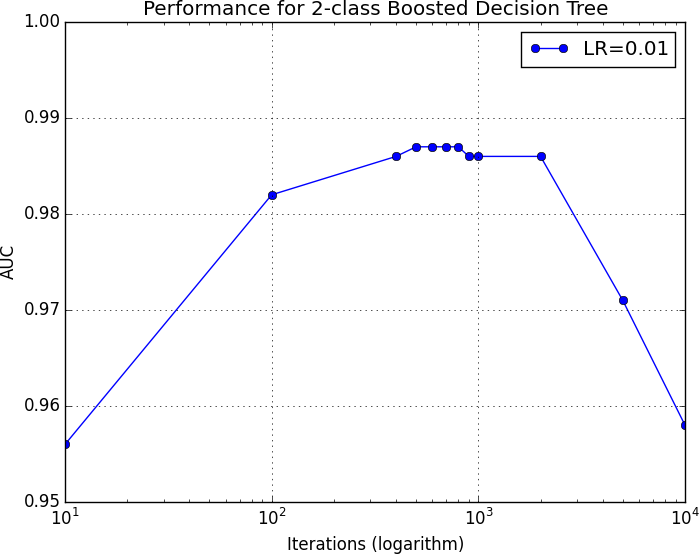
# Parameter setting:

Maximum number of leaves per tree: 20 Minimum number of samples per leaf node: 10 Learning rate: 0.01

Number of trees constructed: 500 ~ 700



You can see from above that LR = 0.01 is the best one among all.



For more accurate observation, I plotted this graph and it turns out that Iter = 500~800 are the peak before overfitting. That’s what I’m looking for the best parameters.

# Effectiveness and Prediction

Such model has been working well on our training dataset that accuracy achieved 97.7% and AUC eventually reached 98.7%, which is obviously higher than any other models. Thus, I used entire training data to train the model for prediction. I predicted the score and score probability for evaluation dataset in the attachment.

# Problem Unresolved / Interesting Discovery

1. **The meaning of some features**

Many columns like “admission data” is not clear about what this column to do. Thus, it’s hard for me to decide whether should I keep this column or what kind of method should I use for missing data.

# Recall vs. Precision

In my experiment, my Recall and Precision are 0.553 and 0.656 respectively, which are relatively balanced. Since those data come from ED, the meaning of Recall is supposed to be how many emergency out there correctly come to the ED while the meaning of Precision is how many of my ED patients are truly need ED. In the meantime, these two elements are kind of tradeoff that we can’t have them both. In this case, if the hospital has enough capacity for patients, I would be concern about Recall because this means we don’t care receive more patients but only focus on true emergency. However, if the resource for this hospital is very intense, then I will suggest consider more Precision rather than Recall for we need to be accurate about the ED patients we selected.

# Overfitting

I tried several combinations of my parameters. In this case, since we have a lot of features therefore it’s reasonable to find a big tree with many leaves. The larger maximum number of tree leaves, the more possibility that the tree might be divided into more detail pieces. As for Minimum number of samples per leaf node, this parameter cares about the generalization of a certain model. If this number is too small, there might be some noise in one node as a way of classification.

Besides, an appropriate learning rate combining with proper iterations could gradually push the model to the best without overfitting. If the learning rate is too big, it’s probable that the curve will never converge and bounce around instead. Thus, I prefer small learning rate with more iterations so that we can push to the

best fitting model gradually. Let’s take LR=0.01 as an example, when I set iterations to (500,700), the AUC of the model will be 0.987 but when the iteration comes to 1000, the AUC dropped to 0.986. And for iterations equals to 10000, the AUC dropped to 0.958. I believe the charts shown below could explain my basic idea of looking for the best model settings.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iter  LR | 10 | 100 | 500 | 1000 | 10000 |
| 0.01 | 0.956 | 0.982 | **0.987** | 0.986 | 0.958 |
| 0.1 | 0.98 | **0.986** | 0.975 | 0.958 | 0.972 |
| 0.5 | **0.985** | 0.983 | 0.98 | 0.979 | 0.963 |
| 1 | **0.984** | 0.981 | 0.981 | 0.981 | 0.501 |

According to the chart, it’s not hard to tell that LR=0.01 and iteration=500 is the best option.

# Filter Based Features

In Azure, we can use Filter Based Feature Selection to tell which columns are the useless ones and exclude some columns then. In order to make the overall and reasonable decision, I tried all the methods provided and made a chart like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fisher Score** | GP Code | Test\_A | **Test\_F** | Test\_G |
| 0 | 0 | **0** | 0 |
| **Chi-Square** | Hospital\_Admisisons | Test\_E | Gender | **Test\_F** |
| 170.371268 | 19.626987 | 0.185827 | **0** |
| **Mutual Information** | Hospital\_Admisisons | Test\_E | Gender | **Test\_F** |
| 0.000886 | 0.000138 | 0.000001 | **0** |
| **Spearman**  **Correlation** | GP Code | Test\_A | **Test\_F** | Test\_G |
| 0 | 0 | **0** | 0 |
| **Kendall Correlation** | GP Code | Test\_A | **Test\_F** | Test\_G |
| 0 | 0 | **0** | 0 |
| **Pearson** | GP Code | Test\_A | **Test\_F** | Test\_G |
| 0 | 0 | **0** | 0 |