Individual Assignment Report on Machine Learning Algorithms

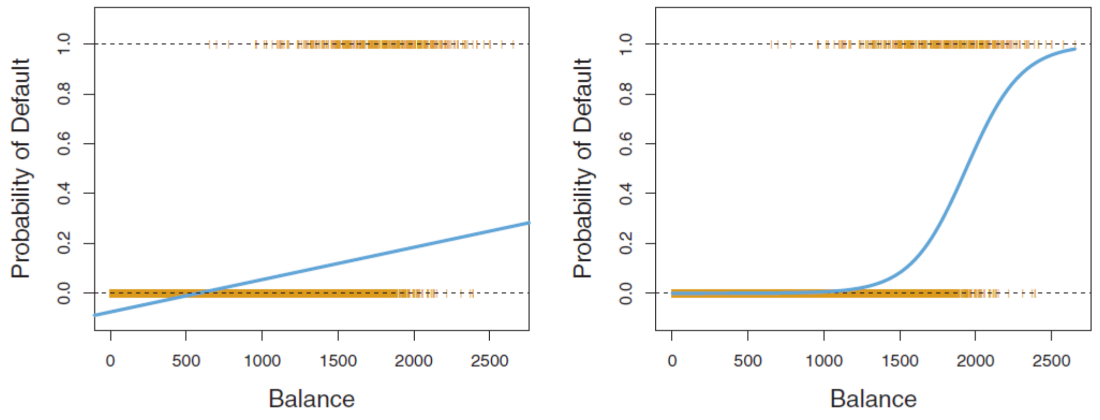
Chenxin Xie

Part I: Explanation on Five Machine Learning Algorithms

1. Logistic Regression

1.1 Description

As we know, linear regression is used to predict a numerical variable by fitting a line into linear shape observations. But when it comes to a classification setting, we are going to predict a binary variable, which means the observations would be either 0 or 1, and trying to fit a straight line among those observations would not be appropriate anymore. Hence, we will use logistic regression to fit a s-shape curve instead.



Logistic regression return a probability of the labeled variable, for example churn or not churn. All probabilities will range between 0 and 1.

Logistic regression can be applied to binary classification or multi-class classification. For example, it can applied to predict churn or not churn, can also used in health care to diagnose it is a stroke, drug overdose or epileptic seizure.

1.2 Objective function: Simple logistic regression

The function here is calculating the probability of X, using this function could avoid p(X) <0 or p(X)>1. And the low balances will never below 0. The logistic regression function will always produce a s-shape curve as we wanted.

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Then we can transform the function to below. The left side of the equation is used to calculate odds, which will range from 0 to infinite.

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}$$

By taking the log to both side of the equation, we will get a log-odds on the left side of the equation and the right side is the linear regression equation.

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

β_0 and β_1 are unknown and must be estimated based on the available training data. The general method is to use maximum likelihood to fit a logistic regression model. And to maximum the likelihood would be translate to maximum the likelihood function:

$$L(\beta_0,\beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1-p(x_i))$$

1.3 Fitting process

In R, we can use glm() function to fit generalized linear models including logistic regression. In the glm() function, we define the variable we are going to predict and predictors that we want to use specially, and then we need to pass in an argument: family=bionomial to specify this is for logistic regression.

Then we can use summary() function to the importance of each predictor to this fitted model.

To make a prediction with the fitted model, we can use predict() function, by specifying type=response, R will output the probability of prediction. We can use compute auc to see the model's performance.

1.4 Pros and Cons

Logistic regression is easy to fit and it's effective. Logistic regression can be applied to any classification problems. Additionally, there is no need to scale the input features.

But logistic regression might have poor performance with irrelevant and highly correlated features and it can be easily outperformed by other algorithms.

2. Random Forest

2.1 Description

Random forest is to keep randomly splitting predictors and fitting into decision tree and get the best prediction. It is named random forest for predictors of each decision tree are randomly splatted so all the trees in this forest are not highly related, which solve the problem in bagging trees. Different from bagging, which is using bootstrap randomly taking observations with all predictors to fit in the model, random forest is randomly taking predictors with all observations. By splitting predictors, random forest does not have any preference on the importance of the predict, which means it is totally random, no weights would be considered when subsetting predictors. The number of predictors that random forest takes for each split would be a key to improve the model. Normally, we will choose the number approximately equal to the square root of the total number of predictors. But, of course we can always use other numbers depends on the situation.

2.2 Fitting process

In R, to apply random forest we can use package called “randomForest”. Then, we use randomForest() function to fit a random forest model. We define the target variable and predictors, number of predictor for each split: mtry=n, n should be no more than total number of predictors. (if you specify n as the total number of predictors, it becomes a bagging model). Then we predict on the test data with the fitted model.

If we are going to perform a hyper parameter tuning on random forest, there are some parameters that we can use to improve the model:

ntree = number of trees in the forest

mtry = max number of features considered for splitting a node

nodesize = number of nodes in the forest

we can always use getParamSet() function in mlr, to check the available parameters and types of the parameters.

2.3 Pros and Cons

Compared with bagging, bagging tress are highly correlated. Random forest fix this problem by subsetting a certain number of random predictors to fit the model, which means random forest can decorrelate trees. Hence it reduce a model's variance. In random forest, each tree taking different subset of predictors and the final outcome takes all the trees in account, so random forest will have no problem of overfitting.

Random forest doesn’t take importance of predictors into account when splitting predictors, which make some strong predictors doesn’t work. And if we change the seed or some parameter, the outcome might change very time.

3. XGBoost (Extreme Gradient Boosting)

3.1 Description

XGBoost uses gradient boosting framework at, and it’s known as an optimized distributed gradient boosting. Gradient boosting is an extension of boosting. In boosting, trees are grown using the information from a previously grown tree one after the other. Trees then slowly learns from the information they captured. After the subsequent iterations, each of the classifiers has a misclassification error associated with them, at last, all these weak classifiers create a strong classifier combing each weak classifier with their weights being considered. Then the strong classifier can make a better prediction.

XGBoost can apply to both regression and classification problems:

In classification: based on what boosting does, a tree is grown one after other and attempts to reduce misclassification rate in subsequent iterations. Here, the next tree is built by giving a higher weight to misclassified points by the previous tree.

In regression: it uses regularization and gradient descent to generalize the model, the subsequent models are built on residuals generated by previous iterations

3.2 Fitting process

We can use xgb.cv() to perform XGBoost. First, we define parameters, including booster, objective, eta (learning rate), gamma (regularization parameter), max_depth (depth of the tree). For classification, we use booster = gbtree; In regression, we use booster = gbtree and booster = gblinear;

Second, we fit the model with the defined parameters, and specify the nrounds (maximum number of iterations), nfold (number of fold for cross-validation).

Third, we make prediction on test data with the fitted model, and compute the auc.

We can also use hypermeter tuning to improve the model with the parameters mentioned above.

3.3 Pros and Cons

XGBoost is known as a good execution speed and better model performance, meanwhile has less issue of overfitting. (and it is highly recommended for winning the Kaggle competitions) Additionally, to perform XGBoost, there is no need for scaling, normalizing data, can also handle missing values well. So there is no need for feature engineering.

But it will be difficult to interpret or visualize a XGBoost model.

4. Support Vector Machines (SVM)

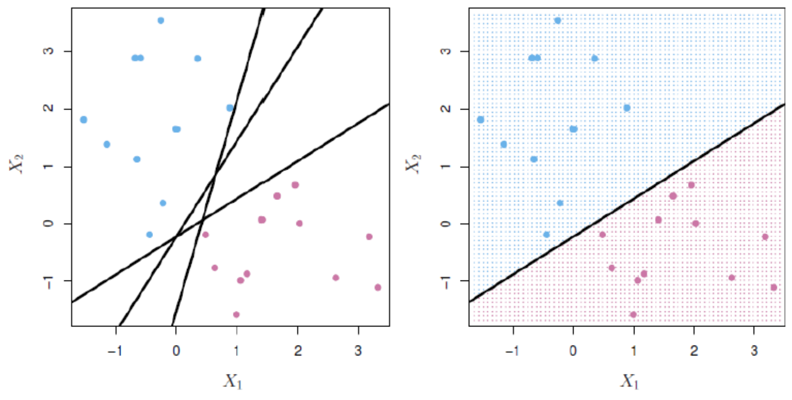
4.1 Description

In a 2 classs classification problem, sometimes there are several lines that can divide the data set into two classes. And sometime you can divide them in a two dimension way, then you can fit a hyperplane that will be able divide the dataset. SVM is used to find a best hyperplane that divides a dataset into two classes correctly.

To find the best hyperplane, it is crucial to define a regularization parameter: C (budget). It refers to the margins between the point closest to the hyperplane and the hyperplane. The larger the margin is, the higher tolerance of misclassification the model is, and the smaller the margin is, the lower tolerance of misclassification the model is.

4.2 Objective function: hyperplane

$$f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$



If fx_i>0, the xipoint is on one side of the hyperplane;

if fx_i<0, the point is on another side of the hyperplane;

If fx_i=0, the point is on the hyperplane.

4.3 Fitting process

In R we can use svm() function to perform SVM. In the svm() function, we need to define Kernel.

For linear problem, we specify kernel = “linear”, and then specify cost(cost of a violation to the margin). If you specify a small cost, the margin is large and there will be more support vectors violating the margin.

For non-linear problem, to fit a SVM with a polynomial kernel we use kernel = “polynomial”, then we specify degree (degree for polynomial). To fit a SVM with radial kernel we use kernel = “radial”, then we specify gamma.

4.4 Pros and Cons

SVM could be a good use high dimensional space problem.

But fitting a SVM with kernel could be tricky, because kernel is sensitive to the parameters you specify.

5. K Nearest Neighbors (KNN)

5.1 Description

KNN algorithm assume that similar objects are tend to close to each other, or a set of similar traits bring the objects together. By calculating the similarity or distance, data points will group with their nearest data point, which is call the “nearest neighbors”.

Since the number of nearest neighbors is defined by us, it is important to understand the impact of picking the right number of nearest neighbors. If we set K=1, means every data point itself is it’s nearest neighbor. By this time, the model is very sensitive when a new data point comes in. If we increase K, the model becomes more stable, but when we increase K up to a point that it includes some data points that actually not that similar, we will see the increase of error. So it's crucial to find the proper K that can make good prediction, at the same time being stable.

5.2 Fitting process

In R, we can use knn() function in class library to train a KNN model. Different from other model's two-step approach, knn() function uses a single command to fit the model and make predictions.

To perform a KNN model using knn() function, we need to specify four arguments: train.X which contains training data with all predictors, test.X contains data we wish to make prediction on, train.Direction contains the labeled variable of the training data, and the number of nearest neighbors K to be used in the model.

5.3 Pros and Cons

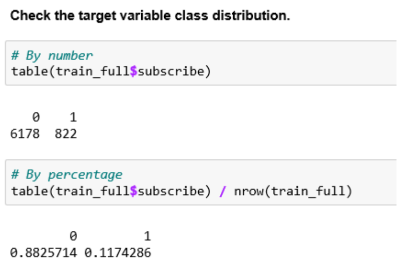
Firstly, KNN is easy to understand and implement. KNN can be used in both classification and regression. It can adjust itself when new data points come in.

But KNN doesn't work very well with the data set has a large number of predictors. And it is sensitive to outliners.

Part II: Setup the benchmark experiment to compare the 5 selected machine learning algorithms.

1. Setup libraries and install packages;

2. Read in data and check the target variable class distribution.



Check the target variable class distribution.

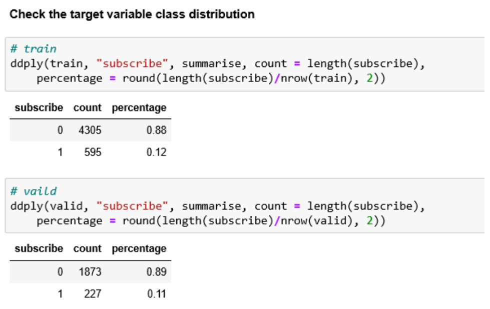
```
# By number
table(train_full$subscribe)
```

```
   0    1
6178  822
```

```
# By percentage
table(train_full$subscribe) / nrow(train_full)
```

```
        0         1
0.8825714 0.1174286
```

3. Split and resample train_full data into train and valid. 70% for train, 30% for vaild.



Check the target variable class distribution

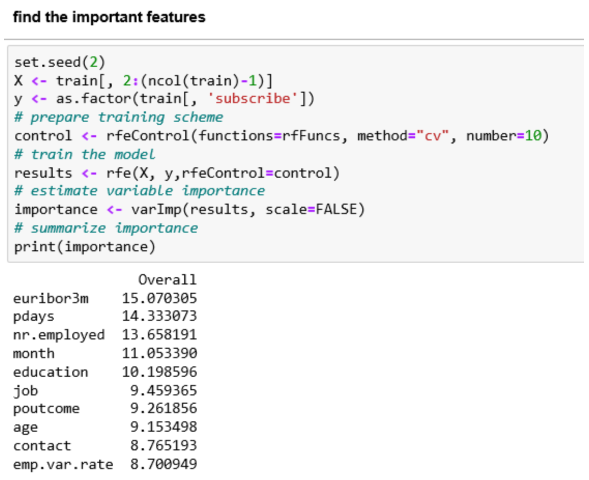
```
# train
ddply(train, "subscribe", summarise, count = length(subscribe),
    percentage = round(length(subscribe)/nrow(train), 2))
```

subscribe	count	percentage
0	4305	0.88
1	595	0.12

```
# vaild
ddply(valid, "subscribe", summarise, count = length(subscribe),
    percentage = round(length(subscribe)/nrow(valid), 2))
```

subscribe	count	percentage
0	1873	0.89
1	227	0.11

4. Use random forest with 10-fold CV to find the important features



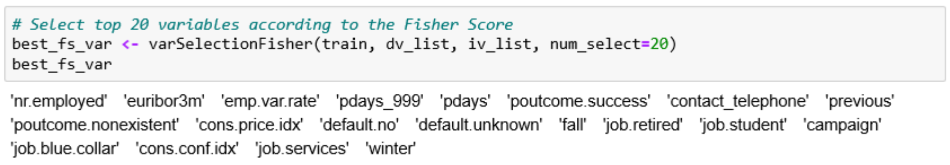
find the important features

```
set.seed(2)
X <- train[, 2:(ncol(train)-1)]
y <- as.factor(train[, 'subscribe'])
# prepare training scheme
control <- rfeControl(functions=rfFuncs, method="cv", number=10)
# train the model
results <- rfe(X, y,rfeControl=control)
# estimate variable importance
importance <- varImp(results, scale=FALSE)
# summarize importance
print(importance)
```

```
               Overall
euribor3m    15.070305
pdays        14.333073
nr.employed  13.658191
month        11.053390
education    10.198596
job           9.459365
poutcome      9.261856
age           9.153498
contact       8.765193
emp.var.rate  8.700949
```

5. Create some useful variables on the important features and make dummies for the categorical features (do the same for valid dataset and test dataset).

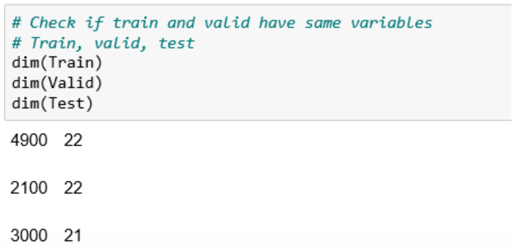
6. Feature selection using fisher score, and keep the top 20 features as the predictors. See the selected features below:



```
# Select top 20 variables according to the Fisher Score
best_fs_var <- varSelectionFisher(train, dv_list, iv_list, num_select=20)
best_fs_var
```

```
'nr.employed'  'euribor3m'  'emp.var.rate'  'pdays_999'  'pdays'  'poutcome.success'  'contact_telephone'  'previous'
'poutcome.nonexistent'  'cons.price.idx'  'default.no'  'default.unknown'  'fall'  'job.retired'  'job.student'  'campaign'
'job.blue.collar'  'cons.conf.idx'  'job.services'  'winter'
```

7. Subset the selected features on valid dataset and test dataset. Then we have the train, valid and test dataset ready for training and predicting.



```
# Check if train and valid have same variables
# Train, valid, test
dim(Train)
dim(Valid)
dim(Test)
```

```
4900  22

2100  22

3000  21
```

8. Logistic regression:

we specify “classfi.logreg” for logistic regression, predict type as “prob” to return probablities in makeLearner() function;

We specify target variable as “subscribe” in makeClassifTask() function;

We specify using “CV” (cross validation) , iters=10 at makeResampleDesc() function to resample with 10-Fold CV method.

After we ran the model with tuned parameter, we make prediction on valid dataset, and check its performance with AUC. It showed that the AUC is about 0.78, which is not bad but not satisfying.

9. Random forest:

we specify “classfi.randomForest”, predict type as “prob” to return probablities in makeLearner() function;

We specify target variable as “subscribe” in makeClassifTask() function;

We specify using “CV” (cross validation) , iters=10 at makeResampleDesc() function to resample with 10-Fold CV method;

We set parameter tuning with ntree between 50 and 500, mtry between 3 and 10, nodesize between 10 and 50;

We define a random tune control with maximum 100 iters with makeTuneControlRandom() function;

It showed we should use ntree=350, mtry=5 and nodesize=41; Then we retrain the model with this parameter set.

After we make prediction on valid dataset, and check its performance with AUC. It showed that the AUC is about 0.75, which is a bit lower than logistic regression.

10. XGBoost (Extreme Gradient Boosting):

we specify “classfi.xgboost”, predict type as “prob” to return probablities in makeLearner() function;

We specify target variable as “subscribe” in makeClassifTask() function;

We specify using “CV” (cross validation) , iters=10 at makeResampleDesc() function to resample with 10-Fold CV method;

We set parameter tuning with nrounds, max_depth, eta and lambda;

We define a MBO tune control with maximum 50 iters with makeTuneControlMBO() function;

It showed we should use nrounds=754, max_depth=4, eta=0.098 and lambda= 399.19; Then we retrain the model with this parameter set.

After we make prediction on valid dataset, and check its performance with AUC. It showed that the AUC is about 0.796, which is an improvement of random forest and logistic regression.

11. Support Vector Machines (SVM):

we specify “classfi.svm”, predict type as “prob” to return probablities in makeLearner() function;

We specify target variable as “subscribe” in makeClassifTask() function;

We specify using “Holdout” with split=2/3 at makeResampleDesc() function to resample with holdout method;

We set parameter tuning with degree, cost and gamma;

We define a random tune control with makeTuneControlRandom() function;

It showed we should use degree=1, cost=3.94 and gamma= 0.44; Then we retrain the model with this parameter set.

After we make prediction on valid dataset, and check its performance with AUC. It showed that the AUC is about 0.64, which is not an ideal performance.

12. K Nearest Neighbors (KNN):

We specify “classfi.kknn”, predict type as “prob” to return probablities in makeLearner() function. We use classif.kknn instead of classif.knn is because for classif.knn does not have predict type as probability;

We specify target variable as “subscribe” in makeClassifTask() function;

We specify using “CV” (cross validation), iters=10 at makeResampleDesc() function to resample with 10-Fold CV method;

We set parameter tuning with k and distance;

We define a random tune control of 100 maximum iters with makeTuneControlRandom () function;

It showed we should use k=14 and distance= 1.47; Then we retrain the model with this parameter set.

After we make prediction on valid dataset, and check its performance with AUC. It showed that the AUC is about 0.73, which is not as good as XGBoost, logistic regression or random forest, but it is better than SVM in this case.