Titanic: Survived or not

July 19, 2017

```r
library(randomForest)
library(e1071)
library(party)
library(gbm)
```

1. Data preparation

Wrapping up with our EDA, we got the training data and testing data,

```r
colnames(train)
```

```
##  [1] "PassengerId" "Survived"    "Pclass"      "Name"        "Sex"
##  [6] "SibSp"       "Parch"       "Ticket"      "Fare"        "Cabin"
## [11] "Embarked"    "Family"      "num_family"  "first_name"  "title"
## [16] "last_name"   "Age"         "adult"       "old"         "re_ticket"
## [21] "FamilyId"
```

To improve the performance of the model, we split the training data into two parts: `train_train` and `train_validation`, where we will training model on the `train_train` and test the performance on `train_validation`. This is always the basic idea in applying machine learning algorithms on a data set. Here we also wrap up it with cross-validation (5-folds), means we split the data into 5 parts and set 1 part as validation set.

```r
set.seed(123)
label <- rep(c(1, 2, 3, 4, 5), length.out = nrow(train))
label <- sample(label, nrow(train), replace = FALSE)
```

More preparation

```r
all_familyId <- base::union(train$FamilyId, train$FamilyId)
train %<>%
  mutate(Pclass = factor(Pclass),
         Family = factor(Family),
         FamilyId = factor(FamilyId, levels = all_familyId))
test %<>%
  mutate(Pclass = factor(Pclass),
         Family = factor(Family),
         FamilyId = factor(FamilyId, levels = all_familyId))
```

2. Logistics Regression

The first model I want to try is logistics regression. Here is the problem I met: Some categorical variables has too many levels, for example: *title*, *re_ticket*, *FamilyId*. It's hard to include the sample with all levels available, therefore when apply `predict()` function on the testing set, it will return error says:

```
Error in model.frame.default(Terms, newdata, na.action = na.action, xlev = object$xlevels) :
  factor re_ticket has new levels SCA5, AQ5, LP4, STONOQ6, SC5, AQ6
```

Since logistics regression will pass all predictors into the model, therefore when the test data has new 'level' it will return error.

For variable like *title*, we can merge some levels by common attributes or refer to the survival rate, however for variable like *FamilyId*, it's hard to deal with.

The only way to make logistics regression works here is to exclude these variables. But the accuracy is not very good.

```
# build logistics regression
log_reg_model1 <- train[label != 2, ] %>%
  glm(Survived ~ Pclass + Sex + Fare + Embarked +
                  Family + num_family + Age + adult,
      data = ., family = binomial)
# training error
log_reg_model1_pred_train <- predict(log_reg_model1, newdata = train[label != 2, ])
log_reg_model1_pred_train_lab <- ifelse(log_reg_model1_pred_train < 0, 0, 1)
table(train[label != 2, 'Survived'], log_reg_model1_pred_train_lab)
```

```
##    log_reg_model1_pred_train_lab
##       0   1
##   0 385  55
##   1  84 189
```

```
# validation error
log_reg_model_pred_valid <- predict(log_reg_model1, newdata = train[label == 2, ])
log_reg_model_pred_valid_lab <- ifelse(log_reg_model_pred_valid < 0, 0, 1)
log_reg_model1_pred_valid <- predict(log_reg_model1, newdata = train[label == 2, ])
log_reg_model1_pred_valid_lab <- ifelse(log_reg_model1_pred_valid < 0, 0, 1)
table(train[label == 2, 'Survived'], log_reg_model1_pred_valid_lab)
```

```
##    log_reg_model1_pred_valid_lab
##      0  1
##   0 90 19
##   1 17 52
```

3. Random Forest

Random forest is one of the ensemble methods, which could reach a very good result.

```
# build random forest
rf_model1 <- train[label != 5, ] %>%
  randomForest::randomForest(Survived ~ Pclass + Sex + Fare + Embarked + Family +
                  num_family + title + Age + adult + re_ticket,
               data = ., ntree = 1500, mtry = 4, importance = TRUE)
# training error
rf_model1_pred_train <- predict(rf_model1, newdata = train[label != 5, ])
rf_model1_pred_train_lab <- ifelse(rf_model1_pred_train < .5, 0, 1)
table(train[label != 5, 'Survived'], rf_model1_pred_train_lab)
```

```
##    rf_model1_pred_train_lab
##       0   1
##   0 433   9
##   1  26 245
```

```
# validation error
rf_model1_pred_valid <- predict(rf_model1, newdata = train[label == 5, ])
rf_model1_pred_valid_lab <- ifelse(rf_model1_pred_valid < .5, 0, 1)
table(train[label == 5, 'Survived'], rf_model1_pred_valid_lab)
```

```
##    rf_model1_pred_valid_lab
##      0  1
##   0 99  8
##   1 18 53
```

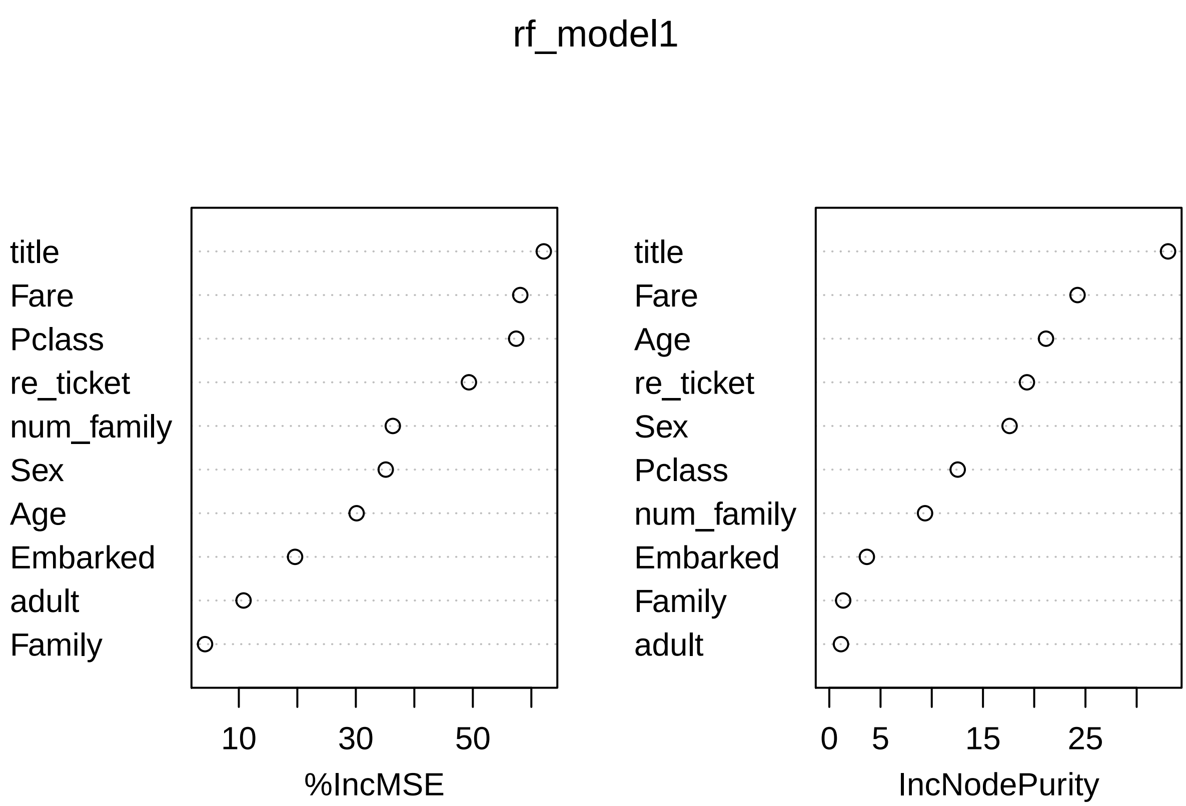
A little bit better than the logistics regression, still a lot of work to do. First let's check the variable importance,

```r
importance(rf_model1)
```

```
##              %IncMSE IncNodePurity
## Pclass     57.406744     12.533541
## Sex        35.113649     17.596412
## Fare       58.107877     24.227180
## Embarked   19.612180      3.665731
## Family      4.225874      1.355119
## num_family 36.319628      9.343868
## title      62.126256     33.062220
## Age        30.131687     21.150635
## adult      10.809139      1.138485
## re_ticket  49.331516     19.288672
```

Here we get two measure of variable importance, `%IncMSE` is based upon the mean decrease of accuracy in prediction on the out of bag samples when the variable is excluded; `IncNodePurity` measures the total decrease in node impurity that results from splits over that variable.

```r
varImpPlot(rf_model1)
```



```r
# build random forest
rf_model2 <- train[label != 5, ] %>%
  randomForest::randomForest(Survived ~ Pclass + Sex + Fare + Embarked +
                 num_family + title + Age + re_ticket,
               data = ., ntree = 1500, mtry = 4, importance = TRUE)
# training error
rf_model2_pred_train <- predict(rf_model2, newdata = train[label != 5, ])
rf_model2_pred_train_lab <- ifelse(rf_model2_pred_train < .5, 0, 1)
table(train[label != 5, 'Survived'], rf_model2_pred_train_lab)
```

```
##    rf_model2_pred_train_lab
##       0   1
##   0 434   8
##   1  18 253
```

```r
# validation error
rf_model2_pred_valid <- predict(rf_model2, newdata = train[label == 5, ])
rf_model2_pred_valid_lab <- ifelse(rf_model2_pred_valid < .5, 0, 1)
table(train[label == 5, 'Survived'], rf_model2_pred_valid_lab)
```

```
##    rf_model2_pred_valid_lab
##      0  1
##   0 98  9
##   1 18 53
```

According to the variable importance, we remove two variables *Family* and *adult*.

Then we try to tunning the parameter of random forest.

```r
# from e1071 pkg
rf_model_tune <- train[label != 5, ] %>%
  tune.randomForest(Survived ~ Pclass + Sex + Fare + Embarked +
                      num_family + title + Age + re_ticket,
                    data = .,
                    ntree = c(800, 1000, 1200),
                    nodesize = seq(3, 21, by = 2),
                    mtry = c(3, 4, 5))
rf_model_tune$best.parameters
```

```
##   nodesize mtry ntree
## 8       17    3   800
```

Let's plug in the optimal parameters,

```r
# build random forest
rf_model3 <- train[label != 5, ] %>%
  randomForest(Survived ~ Pclass + Sex + Fare + Embarked +
                 num_family + title + Age + re_ticket +
                 Family + adult,
               data = ., ntree = 800, mtry = 3, nodesize = 13, importance = TRUE)
# training error
rf_model3_pred_train <- predict(rf_model3, newdata = train[label != 5, ])
rf_model3_pred_train_lab <- ifelse(rf_model3_pred_train < .5, 0, 1)
table(train[label != 5, 'Survived'], rf_model3_pred_train_lab)
```

```
##    rf_model3_pred_train_lab
##       0   1
##   0 425  17
```

```
##   1  42 229
```

```
# validation error
rf_model3_pred_valid <- predict(rf_model3, newdata = train[label == 5, ])
rf_model3_pred_valid_lab <- ifelse(rf_model3_pred_valid < .5, 0, 1)
table(train[label == 5, 'Survived'], rf_model3_pred_valid_lab)
```

```
##    rf_model3_pred_valid_lab
##      0  1
##   0 98  9
##   1 18 53
```

```
varImpPlot(rf_model3)
```

We also meet a problem when applying random forest: `randomForest()` in `randomFores` package cannot deal with categorical variables with more than 53 categories. Therefore *FamilyId* must be numeric if we want to use it under this condition.

Here we introduce an extend version of random forest algorithm, which is available in `party` package.

We gonna use `cforest`, a.k.a. conditional inference trees, as the basic tree of random forest.

```
# build extend random forest
crf_model1 <- train[label != 5, ] %>%
  cforest(Survived ~ Pclass + Sex + Age + Fare +
            Embarked + Family + num_family + title +
            re_ticket + adult + FamilyId, data = . ,
          controls = cforest_unbiased(ntree = 1500, mtry = 3))
# training error
crf_model1_pred_train <- predict(crf_model1, newdata = train[label != 5, ])
crf_model1_pred_train_lab <- ifelse(crf_model1_pred_train < .5, 0, 1)
table(train[label != 5, 'Survived'], crf_model1_pred_train_lab)
```

```
##    crf_model1_pred_train_lab
##       0   1
##   0 424  18
##   1  76 195
```

```
# testing error
crf_model1_pred_valid <- predict(crf_model1, newdata = train[label == 5, ])
crf_model1_pred_valid_lab <- ifelse(crf_model1_pred_valid < .5, 0, 1)
table(train[label == 5, 'Survived'], crf_model1_pred_valid_lab)
```

```
##    crf_model1_pred_valid_lab
##      0  1
##   0 98  9
##   1 17 54
```

3. Boosting

Next step we gonna try a powerful family of algorithms: boosting.

```
# build boosting
bst_model1 <- train[label != 5, ] %>%
  gbm(Survived ~ Pclass + Sex + Fare + Embarked +
        Family + num_family + title + Age + re_ticket +
        FamilyId + adult, data = .,
      distribution = 'bernoulli', n.trees = 2500, interaction.depth = 5)
```

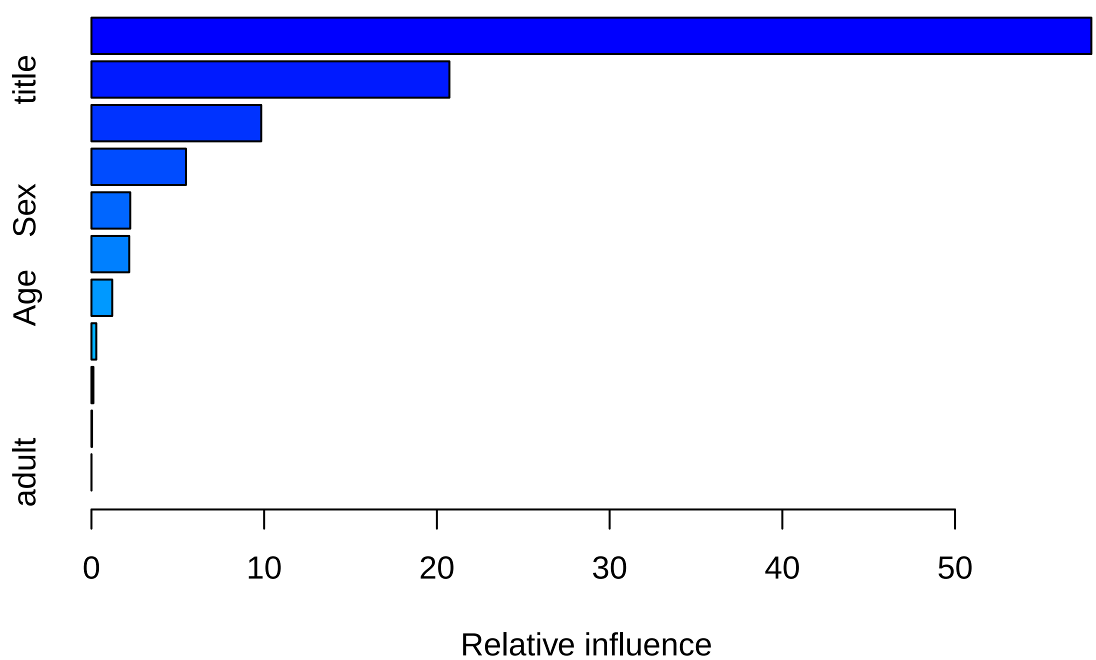
```r
# training error
bst_model1_pred_train <- predict(bst_model1, newdata = train[label != 5, ], n.tree = 2500)
bst_model1_pred_train_lab <- ifelse(bst_model1_pred_train < 0, 0, 1)
table(train[label != 5, 'Survived'], bst_model1_pred_train_lab)
```

```
##    bst_model1_pred_train_lab
##       0   1
##   0 417  25
##   1  52 219
```

```r
# tesing error
bst_model1_pred_valid <- predict(bst_model1, newdata = train[label == 5, ], n.tree = 2500)
bst_model1_pred_valid_lab <- ifelse(bst_model1_pred_valid < 0, 0, 1)
table(train[label == 5, 'Survived'], bst_model1_pred_valid_lab)
```

```
##    bst_model1_pred_valid_lab
##      0  1
##   0 82 25
##   1 11 60
```

```r
# summary - variable importance
summary(bst_model1)
```




```
##                   var      rel.inf
## FamilyId     FamilyId 57.890500194
## title           title 20.724211065
## re_ticket   re_ticket  9.831919392
## Fare             Fare  5.470770077
```

```
## Sex                 Sex  2.248606315
## Pclass           Pclass  2.189245444
## Age                 Age  1.205440529
## Embarked     Embarked  0.282028834
## num_family num_family  0.114760824
## Family           Family  0.037243638
## adult             adult  0.005273688
```