統計學習初論（106-2）

作業六 (第一部分)

---第二部分將在下周二公布 ---

作業設計：盧信銘
國立台灣大學資管系

截止時間：2018 年 5 月 22 日上午 9 點

請至 RSAND 上批改，第一題範例命令為: sl_check_hw6q1 ./your_program。作業自己做。嚴禁抄襲。不接受紙本繳交，不接受遲交。請以英文或中文作答。

第一題

(50 points) The goal of sentence categorization is to assign a sentence to one of two or more predefined categories. The underlying problem is to detect sentences that mentioned about the effort of introducing new products or services in current fiscal year. We denoted this category as “S.PRODUCT.”

The hw5ds1.rdata file contains a data frame named ds1. The first column of ds1 is annotated S.PRODUCT. The feature values (unigrams) are in the remaining columns. Each row in ds1 represents a sentence. All feature values in columns 2 and beyond are term-frequency inverse document frequency (TF-IDF) values. A value of zero indicate the corresponding unigram did not appear in the sentence. A value larger than zero indicates that that particular unigram appeared in the sentence.

Also, there is another variable (list) named cvfold that specify how data should be partitioned for a ten-fold cross validation. The variable cvfold is a list of ten elements. Each element contains the row IDs in that particular fold.

Write a function named rf_carton to estimate the testing performance of the Random Forest classifier using a typical train-tune-test method (details will be given below). You should use the randomForest function from the randomForest library to learn your random forest models. This function should take the following input parameters (in this order):

1. dsall: The input dataset. The first column should be the outcome of the classification problem.
2. folds: The variable specifies how dsall should be partitioned for cross validation.

3. testfold: the fold that should be reserved for testing.
4. ypos: The label for a positive case in the first column. The default value is "pos."
5. chi_threshold: The minimal value of chi squared statistics for a feature that should be returned. The default value is 0.1.
6. grid_length: The length of grids that should be used in parameter tuning. The default value is 20.
7. grid_type: The type of grids. The value should be either "equal" or "loglinear." The default value is "loglinear." See below for additional information regarding this parameter.
8. rfntree: Number of trees in a forest. The default value is 500.
9. debuglevel: Whether to output debug information. The default value is 0. You should not output any debug information when the value is 0. Set this value to a positive integer when developing the function.

We refer to a fold by its index in folds. For example, fold 1 is the row index in folds[[1]]. The function should take all row index in folds[[testfold]] to form the testing set. All rows not in folds[[testfold]] should be the training set. To conduct parameter tuning, we need to further partition the training set into a sub-training set and a tuning set. The tuning set is folds[[(testfold-1)]] if testfold is larger than 1. It is the last fold if testfold is 1. For example, if testfold is 3, then tuning fold is 2. If testfold is 6, then tuning fold is 5. If testfold is 1, then tuning fold is 10. The sub-training set is the training set excluding the rows in tuning fold.

For random forest classifier, there are at least two important parameters that need to be tuned. The first parameter is number of trees in a forest. The second is the m parameter that determine number of features to be randomly selected as candidates before splitting a node. Following the randomForest function, we are going to call this parameter mtry. The parameter mtry will be the focus of this question, we are going to fix the number of tree to rfntree.

After constructing the training, sub-training, tuning, and testing datasets, we need to determine the values of mtry that are going to be tried on sub-training and tuning datasets. The number of values (i.e., grid length) is determined by the grid_length parameter. The minimal mtry is 2 and the maximal mtry is the number of input features. There are two approaches to determine the grids between the minimal and maximal mtry. The first approach is equal-space grids (grid_type="equal"). This approach simply lays out equally-spaced grids. This can be done in R via the seq() command. The following is an example of constructing 15 equally-spaced grids from 2 to 100:

```
> m_min=2
> m_max = 100
> grid_length=15
> grids = unique(round(seq(m_min, m_max, length=grid_length )))
> print(grids)
```

```
 [1]   2   9  16  23  30  37  44  51  58  65  72  79  86  93 100
```

The second approach is log-linear grids (grid_type=”loglinear”). This approach lays out equally-spaced grid in log space. The following command demonstrate how to do this in R to construct 15 log-linear grids from 2 to 100:

```
> m_min=2
> m_max = 100
> grid_length=15
> grids = unique(round(exp(seq(log(m_min), log(m_max),
length=grid_length))))
> print(grids)
 [1]   2   3   5   6   8  11  14  19  25  33  43  57  76 100
```

Note that the above example only returns 14 grid points because of duplicated values after rounding to the closest integers.

For each value *m1* in the grids, you should conduct feature selection via Chi-square statistics on the sub-training dataset, and drop all features with a Chi-square statistics smaller than 0.1. Use the filter_chisq() function for the task (provided at the end of this question, and also available in the homework package). Note that you do not need to include this function in your submission to RSAND. You can call filter_chisq() directly in your program submitted to RSAND.

Set chi_threshold of filter_chisq() to the value of chi_threshold in rf_carton(). The features with a Chi-square statistics larger than chi_threshold are included to train a random forest model with mtry set to *m1* and ntree set to rfntree. Compute and record the F1 score of this model on the tuning dataset. F1 score is defined as 2 * precision * recall / (precision + recall).

Note that you are strongly recommended to pass the tuning dataset to randomForest() using xtest and ytest parameters so that randomForest() can give you the prediction performance directly. You should not do this in a two-step procedure. That is, you should not first train a model on sub-training, and then use predict() to compute prediction on the tuning dataset and to compute its tuning performance. In theory, using xtest and ytest should give you exactly the same result as that using the two-step procedure. However, the result is usually different. I believe this is a small bug in randomForest, and you should consciously avoid the potential problem caused by this bug.

Another potential problem related to randomForest is the order of training and testing dataset. Since random forest is a stochastic algorithm, the order of training data may influence how the forest is constructed. To ensure that your result will be identical to that of RSAND auto-checking program, use the following code segment to construct your training, sub-train, tuning, and testing dataset:

```
ycol=1
#this is the training dataset
dstrain_all = dsall[-folds[[testfold]],]
#testing dataset
dstest = dsall[folds[[testfold]],]
```

```
if(testfold==1) {
    tunefold=10
} else {
    tunefold=testfold-1
}
#tuning dataset
dstune = dsall[folds[[tunefold]],]
#this is the sub-train dataset
dstrain = dsall[-c(folds[[tunefold]], folds[[testfold]]),]


dstrain_y = as.factor(dstrain[,ycol] == ypos)
dstune_y = as.factor(dstune[,ycol] == ypos)
```

After going through all mtry values in the grids, pick the mtry value with the largest F1 score. Ties are break by favoring smaller mtry value. Call this value best_mtry.

Conduct another run of feature selection using filter_chisq() on the training dataset using the same chi_threshold. Train a random forest model using the selected features. Set mtry to best_mtry, and ntree to rtftree. Test the performance of this model on the testing dataset, and record precision, recall, and F1 score.

The returned list should include the following components (in this order):

- mgrids: the grid value used to search for the best mtry.
- f1_all: a vector that contain the F1 score on tuning dataset for each of the values in mgrds.
- best_m: the best mtry.
- test: a list that contain the following three component (in this order): precision, recall, f1. The three component should store the precision, recall, and F1 score on the testing dataset.
- fselect: the variable returned by filter_chisq when conducting feature selection on the training dataset.

Below is the source code of filter_chisq() function.

```
filter_chisq = function(dstrain, ypos="pos", min_count=5,
chi_threshold = 1e-5) {
    nugrams = ncol(dstrain) #number of unigram+1
    chiall = rep(-1, nugrams) #the first column is always -1, and will
not be selected.
    yvec = as.numeric(dstrain[,1]==ypos)
    options(warn = -1) #silence the warning
    for(ii in 2:nugrams) {
        tmp1=cbind(yvec, as.numeric(dstrain[,ii]>0))
        tmp1a=table(tmp1[,1], tmp1[,2])

        if(nrow(tmp1a)<2 | ncol(tmp1a)<2) {
            #stop("tmp1a table dimension too small!")
            chiall[ii] = 0
        } else if(sum(tmp1[,2])<=min_count) {
            chiall[ii] = 0
```

```
                #cat("feature", ii, "count too low, skip\n")
            } else {
                tmp2=chisq.test(tmp1a, correct=FALSE)
                chiall[ii] = tmp2$statistic
            }
        }
        options(warn = 0) #turn the warnings back on
        o1 = order(chiall, decreasing=TRUE)

        tmpind1 = chiall[o1] > chi_threshold
        if(sum(tmpind1) ==0) {
            #cat("We have not features selected. The maximum value of
chisq test is ", max(chiall), "\n")
            return(list(colpos = NULL, colname=NULL, chistat=NULL))
        } else {
            o2=o1[tmpind1]
            retname = names(dstrain)[o2]
            return(list(colpos = o2, colname=retname, chistat=chiall[o2]))
        }
}
```

Sample input and output

```
> #Sample 1
> load('hw5ds1.rdata')
> set.seed(5555)
> rftest=rf_carton(ds1, cvfold, testfold=1, debuglevel=0)
Loading required package: randomForest
randomForest 4.6-12
Type rfNews() to see new features/changes/bug fixes.
> print(rftest$mgrids)
 [1]   2   3   4   6   7  10  13  16  21  28  36  47  62  80 104 136
177 230 300
> print(rftest$f1_all)
 [1] 0.4615385 0.6250000 0.7500000 0.7500000 0.7500000 0.7058824
0.6666667
 [8] 0.7058824 0.6666667 0.7058824 0.7058824 0.6666667 0.6666667
0.7058824
[15] 0.6666667 0.6666667 0.6666667 0.6666667 0.6666667
> print(rftest$best_m)
[1] 4
> print(rftest$test)
$precision
[1] 1

$recall
[1] 0.7

$f1
[1] 0.8235294

> print(head(rftest$fselect$colpos, n=15))
 [1] 4111 4061 3150 4032 2985 3510 3783 3891 3309 3416 3422 3937 1497
2999 3503
```

```
> print(tail(rftest$fselect$colpos, n=15))
 [1] 3961 3966 4117 3642 3688 3733 3827 3842 3849 3855 3873 3945 4006
4020 4125
> print(length(rftest$fselect$colpos))
[1] 347

> #Sample 2
> load('hw5ds1.rdata')
> set.seed(5556)
> rftest=rf_carton(ds1, cvfold, testfold=7, debuglevel=0)
> print(rftest$mgrids)
 [1]   2   3   4   6   8  10  13  17  22  28  37  48  63  82 107 139
181 236 308
> print(rftest$f1_all)
 [1] 0.3000000 0.5217391 0.5833333 0.5833333 0.6400000 0.6400000
0.6923077
 [8] 0.7407407 0.7407407 0.7857143 0.7857143 0.7857143 0.7857143
0.7857143
[15] 0.7857143 0.7857143 0.7857143 0.7857143 0.7857143
> print(rftest$best_m)
[1] 28
> print(rftest$test)
$precision
[1] 0.9230769

$recall
[1] 0.8571429

$f1
[1] 0.8888889

> print(head(rftest$fselect$colpos, n=15))
 [1] 4111 4061 3150 4032 2985 3806 3783 3510 2999 3309 3416 2953 3422
1497 3503
> print(tail(rftest$fselect$colpos, n=15))
 [1] 3688 3733 3814 3943 3945 3947 3971 4110 4046 4072 3898 4055 4125
3966 4126
> print(length(rftest$fselect$colpos))
[1] 346
```

Evaluation: All credits will be given based on the correctness of 10 testing cases. Correct output in a case is worth 5 points.