Data Mining and Machine Learning Problems 2

1. Why is it necessary to use regularization in linear regression when the number of predictor variables is greater than the number of observations in the training sample? Explain how regularization helps in this case. Are there other situations where regularization might help? What is the potential disadvantage of introducing regularization? Why is sparsity a reasonable assumption in the boosting context. Is it always? If not, why not?

i) When the number of predictor variables is greater than the number of observations in the training sample, the matrix X^TX is rank-deficient, then we cannot obtain a unique optimal solution $\hat{\beta}$ to minimize the mean squared error loss (which will be $\hat{\beta} = (X^TX)^{-1}X^TY$ if (X^TX) is of full rank in linear regression.) This will lead to infinitely many solutions for the estimation of β. Given one solution $\hat{\beta}$, the quantity $\hat{\beta}+\eta$ is also a solution for any $\eta \in null(X)$. This type of non-uniqueness makes interpretation meaningless. The fitted values from least squares regression are always unique, but in terms of actual predictions, at a new point $x_0 \in \mathbb{R}^n$, it will not get the case that $x_0^T\hat{\beta} = x_0^T\tilde{\beta}$ for two solutions $\hat{\beta}$ and $\tilde{\beta}$.

ii) By adding regularization, we are equivalently placing a restriction on the joint solution values:

$$\hat{\beta}(t) = \arg\min_{\beta} \hat{R}(\beta) \quad \text{s.t. } P(\beta) \leq t$$

where $R(\beta) = \mathbb{E}_{x,y}L(y, F(x,\beta))$, $F(x;\beta) = \beta_0 + \sum_{j=1}^{n} \beta_j x_j$, and L is some loss function. And $\hat{R}(\beta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, a_0 + \sum_{j=1}^{n} a_j x_{ij})$. Here $P(\beta)$ is a non-negative function of the parameters specifying the penalty/constraint. Setting $t = 0$ will make the solution values satisfy $P(\beta) = 0$, thereby producing the least variance. Setting $t > P(\hat{\beta})$ produces the unrestricted solution with maximal variance. Intermediate values $0 < t < P(\hat{\beta})$ provide degrees of restriction between these two extremes, thereby regulating the stability (variance) of the estimates $\hat{\beta}(t)$ with respect to different training samples.

iii) Regularization could also help when the sample size N is not large compared to the number of parameters $(n + 1)$. In this case, the least squares estimator is very poor because of the high variability triggered by the evaluation on different random samples drawn from the population distribution. The risk could be quite poor, i.e. $\sigma^2 n/N$ in sample error, which will be bad if n (the number of parameters) is an appreciable fraction of N(sample size).

iv) The potential disadvantage would be to increase bias for the estimator.

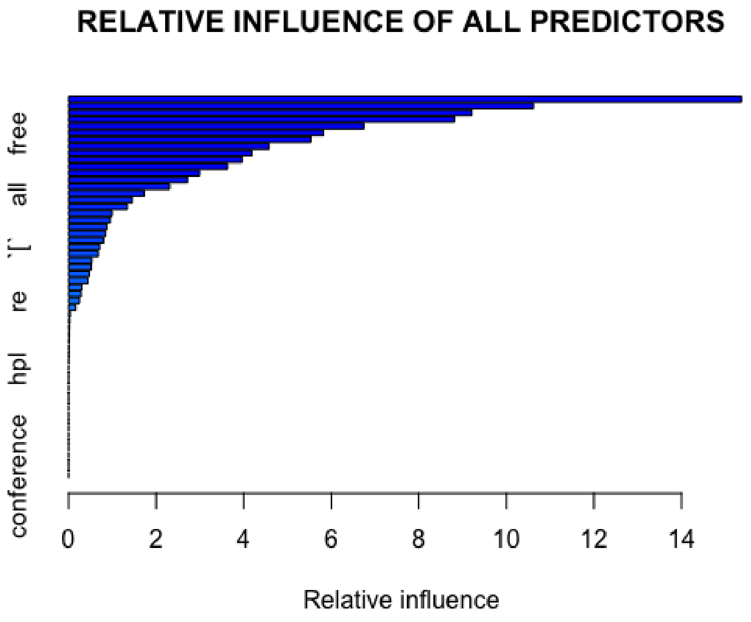
v) The structure model for "boosting" base learner is $F(x) = \sum_{j=1}^{J} a_j f(x; b_j)$, where $F(x; \{a_j\}_1^J) \in \mathcal{F}$, with $\mathcal{F}$ being all linear combinations of $f(x;b)$ in base learner function space. Usually J is very large, and J can be ∞. So we need regularized linear regression and sparsity is a reasonable assumption in the boosting context. Sparsity is desirable because it corresponds to performing variable selection (in the function space) in the constructed linear model, and provides a level of interpretability. (continue on next page)

It's not always reasonable. We know that $\hat{\beta} \simeq \beta^*$ then $S(\hat{\beta}(\lambda^*)) \simeq S(\beta^*)$, where S is the degree of sparsity. So if the original model is not sparse, but we assume sparsity, then the estimator would probably be very biased.

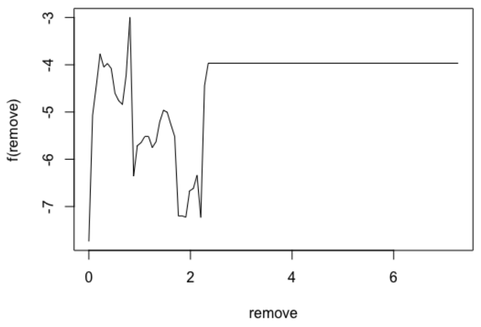
2. Binary classification: SpamEmail

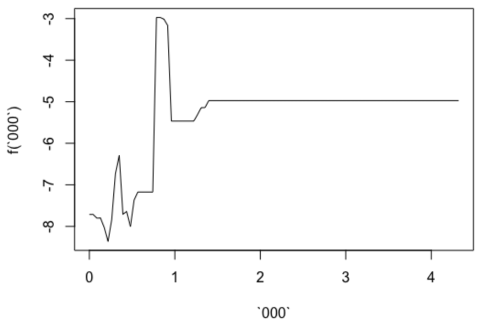
(a) The misclassification error on test is 2.411995%. (While the misclassification error on the training set is 0.8477339%). Of all the spam emails of the test set, 2.750809% was misclassified; and of all the non-spam emails, 2.183406% was misclassified.

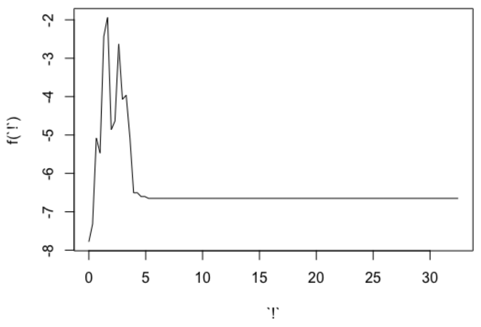
(b) (i) By adjusting the loss weights for non-spam email, the overall misclassification error of the final filter is 14.21121%, the percentage of good emails misclassified: 0.2183406%, the percentage of spam emails that were misclassified: 34.95146%.
(ii) The important variables are: remove, 000, !, *our*, \$, *free*, *CAPTOT*, *money*, *CAPAVE*, *CAPMAX*, 3*d*, *your*, *you*, *credit*, *all*, *business*, *receive*. (These are the variables with relative influence $\geq$ 1).

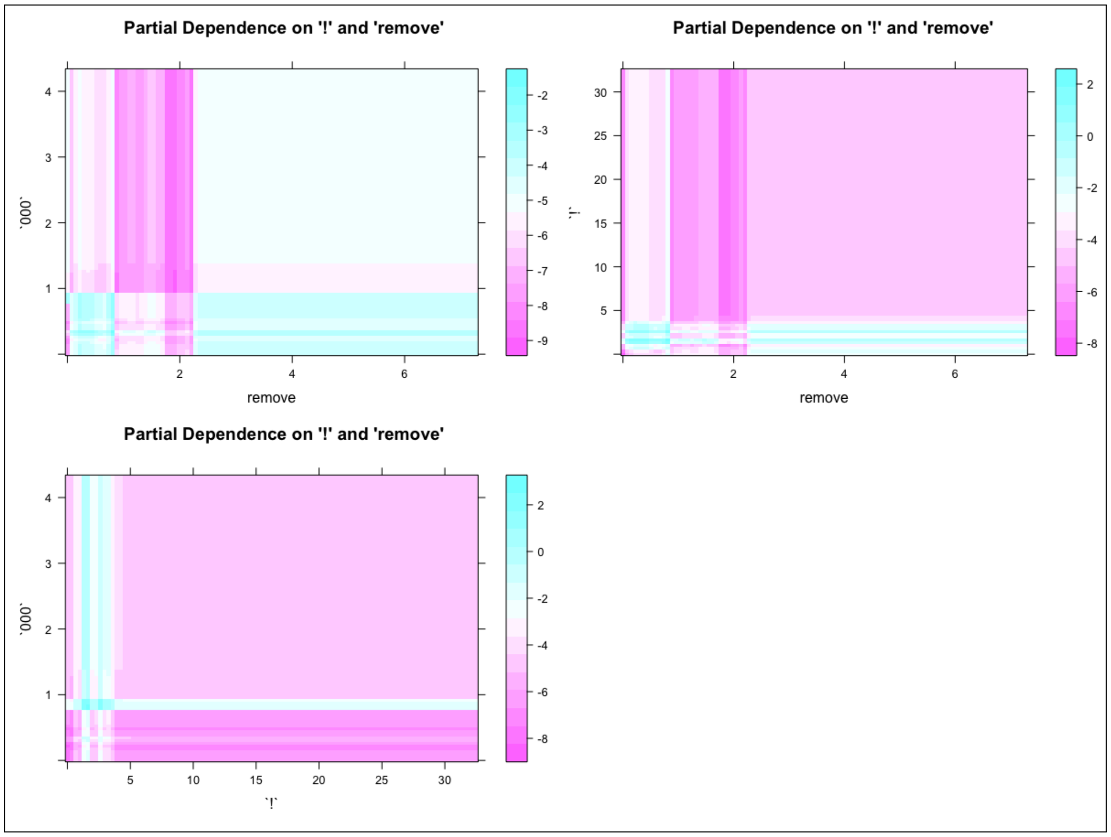


(iii) Now we look at the most important three attributes in this problem by drawing partial dependence plot on one or two variables (remove, 000, !).









Code:

```
rm(list=ls())
library(gbm)
spam_train<-read.table('spam_stats315B_train.csv',sep=',')
spam_test<-read.table('spam_stats315B_test.csv',sep=',')

rflabs<-c("make", "address", "all", "3d", "our", "over", "remove",
          "internet","order", "mail", "receive", "will",
          "people", "report", "addresses","free", "business",
          "email", "you", "credit", "your", "font","000","money",
          "hp", "hpl", "george", "650", "lab", "labs",
          "telnet", "857", "data", "415", "85", "technology", "1999",
          "parts","pm", "direct", "cs", "meeting", "original", "project",
          "re","edu", "table", "conference", ";", "(", "[", "!", "$", "#",
          "CAPAVE", "CAPMAX", "CAPTOT","type")
# Names for predictors and response

colnames(spam_train)<-rflabs
colnames(spam_test)<-rflabs
```

```r
#call the gbm function to train the Gradient Boosting Model
set.seed(130)
gbm0<-gbm(type~.,data=spam_train, train.fraction=0.8,
          interaction.depth=4, shrinkage=0.05,
          n.trees=2500, bag.fraction=0.5, cv.folds=5,
          distribution="bernoulli", verbose=T)

# optimal number of iterations
best.iter_test<-gbm.perf(gbm0, method="cv")
## 952

# prediction on the training set
gbm0.predict<-predict(gbm0,spam_train,type="response",n.trees=best.iter_tes

# set the threshold to the predicted value
gbm0.predict[gbm0.predict<0.5]=0
gbm0.predict[gbm0.predict>=0.5]=1

matrix.train<-table(gbm0.predict, spam_train$type)

# Training error: 0.008477339
(matrix.train[1,2]+matrix.train[2,1])/nrow(spam_train)

# non-spam error: 0.004326663
matrix.train[2,1]/(matrix.train[1,1]+matrix.train[2,1])

# spam error: 0.01477833
matrix.train[1,2]/(matrix.train[1,2]+matrix.train[2,2])

# prediction on the test set
gbm0.predict.test<-predict(gbm0, spam_test, type="response",n.trees=best.i

# still use the 0.5 threshold for prediction
gbm0.predict.test[gbm0.predict.test<0.5]=0
gbm0.predict.test[gbm0.predict.test>=0.5]=1
```

```
matrix.test<-table(gbm0.predict.test,spam_test$type)
# test_error: 0.02411995
(matrix.test[1,2]+matrix.test[2,1])/nrow(spam_test)
# non_spam test error: 0.02183406
matrix.test[2,1]/(matrix.test[2,1]+matrix.test[1,1])
# spam test error: 0.02750809
matrix.test[1,2]/(matrix.test[1,2]+matrix.test[2,2])

# adjust the loss weights
weights<-rep(1,nrow(spam_train))
weights[spam_train$type==0]=55
mod.gbm<-gbm(type~., data=spam_train, weights=weights,
             interaction.depth = 4,shrinkage=0.05, n.trees=2500,train.frac
             bag.fraction = 0.5, cv.folds=5, distribution='bernoulli', verl
best.iter_train<-gbm.perf(mod.gbm, method="cv")
gbm2.predict.test<-predict(mod.gbm, spam_test, type="response", n.trees=bes
gbm2.predict.test[gbm2.predict.test < 0.935]= 0
gbm2.predict.test[gbm2.predict.test >= 0.935]= 1
gbm2.matrix.test<-table(gbm2.predict.test, spam_test$type)

# overall test error: 0.1421121
(gbm2.matrix.test[1,2]+gbm2.matrix.test[2,1])/nrow(spam_test)
# test error of non-spam:0.002183406
gbm2.matrix.test[2,1]/(gbm2.matrix.test[1,1]+gbm2.matrix.test[2,1])
# test error of spam:0.3495146
gbm2.matrix.test[1,2]/(gbm2.matrix.test[1,2]+gbm2.matrix.test[2,2])

#import predictors:
summary(mod.gbm, n.tree = best.iter_train, main = "RELATIVE␣INFLUENCE␣OF␣AI

##                    var      rel.inf
## remove          remove 15.362687521
## `000`            `000` 10.617743228
## `!`                `!`  9.214814099
## our                our  8.818640005
## `$`                `$`  6.746712479
## free              free  5.822537804
## CAPTOT         CAPTOT  5.538820067
## money           money  4.580125554
## CAPAVE         CAPAVE  4.183621701
## CAPMAX         CAPMAX  3.963212518
## `3d`             `3d`  3.631351389
## your             your  2.992381081
## you               you  2.714547965
## credit         credit  2.312089483
## all               all  1.727584626
## business     business  1.452295347
## receive       receive  1.347498104
## --------------(etc.)---------------

#remove
plot(x=mod.gbm, i.var=7, n.tree=best.iter_train, main="Partial␣Dependence␣

# 000
plot(x=mod.gbm, i.var=23, n.tree=best.iter_train, main="Partial␣Dependence␣

# !
plot(x=mod.gbm, i.var=52, n.tree=best.iter_train, main="Partial␣Dependence␣
```