A Simple Analysis for *Divorce* Model

Jingyue Wu

Department of Computing and Information Science, Cornell University
Advisor: Yang Ning, Department of Statistics and Data Science

2019, December 10

1 Executive Summary

The goal of this analysis is to create models that could accurately predict the divorce based on all the available features, and also find other significant items and features by interpreting the final results. The dataset we are given contains 84 (49%) divorced women/men and 86 (51%) women/men in happy marriage, each with 54 responses which includes detailed information about their marriage. In our research, we would mainly apply the correlation-based feature selection models on the whole dataset (`divorce.csv`).

2 Introduction and Problem Definition

The goal of this project is to apply different machine learning methods we've learned in class and develop different models to predict and shrink `divorce` model. This project consists of two parts: predict the divorce and choose the most significant features. For each part, we divide the dataset into training data set and test data set with the percentage of 70% and 30%, and apply the classification methods to predict the results.

1. In order to predict the `divorce` based on all features, we mainly apply classification methods and build the following models: Logistic Regression; Bayes' theorem-based methods including Linear Discriminant Analysis (LDA) and Naive Bayes; non-parametric method: K-nearest Neighborhoods (KNN); shrinkage methods including Lasso and Ridge Regression; tree-based method including Decision Tree, Bagging Tree and Random Forest; Support Vector Method including Support Vector Classifier and Support Vector Machine; Artificial Neural Network.

2. To identify significant features, we mainly apply Lasso Regression, Tree Pruning, Subset Selection and testify the efficiencies of each model.

3 Classification and Testing with All Features

3.1 Logistic Regression Method

We try to use multiple logistic regression to predict the binary response based on 54 quantitative predictors, and generalize the model as follows:

$$log(\frac{p(X)}{1-p(X)}) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p \tag{1}$$

where $X = (X_1, X_2, \ldots, X_{54})$ are 54 predictors, $p(X)$ is the probability of divorce and $1 - p(X)$ is the probability of a happy marriage. According to R, we get the test error rate is 7.69%.

However, due to the fact that the data are well-separated and maximum likelihood estimates are not unique, fitted probabilities numerically 0 or 1 occurred, we cannot conclude that the multiple logistic regression model as a good fit for our data.

3.2 Bayes' Theorem-based Method

While logistic regression directly generate a model for `divorce`, Bayes' theorem indicates another way of modeling by using conditional distribution.

Linear Discriminant Analysis In LDA, it models the distribution of predictors X separately in each response class (Y), and then apply Bayes' theorem into estimates for $Pr(Y = k|X = x)$:

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)} \tag{2}$$

LDA assumes each predictor is normally distributed, so each class has the same variance. In this method, library `MASS` is used, with test error rate of 1.92%.

Naive Bayes Naive Bayes has been one of the simplest but also most powerful algorithms for classification modeling, it is a form of discriminant analysis that assumes features are independent in each class. It is especially useful when p is large, hence may perform well in our situation. For this method, we install packages `caret` and using both `caret` and `e1071` library, then plug into `R` with `NBclassifier = naiveBayes(Class~ ., data=divorce.train)` to build a model with test MSE of 0%.

3.3 Non-parametric Method

KNN The KNN classifier first identifies the K points in the training data that are closest to X_0, represented by N_0. It then estimates the conditional probability for class j as the fraction of points in N0 whose response values equal *j*:

$$Pr(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \tag{3}$$

The choice of *K* has a drastic effect on the KNN classifier obtained. As *K* grows, the model becomes less flexible and produces a decision boundary that is close to linear. This represents the bias variance trade-off. Hence, we choose cross-validation to determine the value of *K*.

By dividing the data into 5 folds, we can obtain test error rate of 1.92%, so this result suggests this model is a good fit for our data.

3.4 Shrinkage Method

Lasso Regression & Ridge Regression With too many predictors, fitting the full model without penalization will result in large prediction intervals, and least square regression estimator may not uniquely exist. As an alternative, we can fit a model containing all *p* predictors by shrinking some of the coefficient estimates towards zero. The first approach is ridge regression, with β^R minimize :

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2 = RSS + \lambda \sum_{j=1}^{p}\beta_j^2 \tag{4}$$

$\lambda \sum_{j=1}^{p} \beta_j^2$ is a shrinkage penalty. In the `R` code, we use ridge regression to contain all the predictors and get final test error rate 1.92% which is same as the result of KNN approach.

The Lasso coefficients, $\hat{\beta}_\lambda^L$ minimizes the quantity:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j| = RSS + \lambda \sum_{j=1}^{p}|\beta_j| \tag{5}$$

As lasso could force some coefficients exactly equal to zero, it performs variable selection. In our case, the Lasso method obtains the same test error rate, 1.92%, as the ridge regression in `divorce` data.

3.5 Tree-based Method

Decision Tree & Tree Pruning Decision Tree is another powerful tool to manage our data. We predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. The classification error rate for one region is the fraction of the training observations in that region that do not belong to the most common class $E = 1 - \max_k(\hat{p}_{mk})$ by checking the efficiency of this model, we got a test error rate of 3.85%.

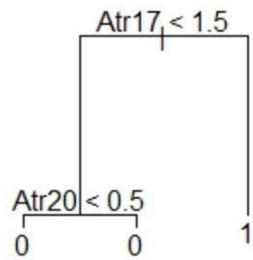

Figure 1: Part of Decision Tree

Tree Pruning operates as a special case of Tree Decision Methods, while there might be chances that tree could overfit the data. In Tree Pruning, it reduces the chances of overfitting the training dataset by building a larger tree and then prune it to select a subtree which leads to the lowest test error rate. Each value of there corresponds a subtree $T \subset T_0$ such that $\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$ is as small as possible. By building the Tree Pruning model in `R`, we also got a test error rate of 3.85%, which is exactly the same as Tree Decision.

Bagging To further reduce variance of the decision tree method discussed above, bagging method is applied to the training dataset. We consider a bagging tree with 500 decision trees in total, each split of the tree holds all 54 predictors, then apply majority vote to assign a final value for `Class`.

$$\hat{f}_{bag}(X) = MajorityVote\{\hat{f}^{*B}(X)\} \tag{6}$$

As prediction for `Class` is chosen as the most commonly occurring class among all 500 predictions, model variance is greatly reduced, indicating that we now have a more stable model. Test error rate for bagging method is 1.90%.

Random Forest To further look into the questionnaire, we could safely assume that the answer to some questions might be crucial for a happy marriage. For example, answers to question 7: “We are like two strangers who share the same environment at home rather than family” might strongly decide whether a couple would divorce or not, as such questions, in a sense, could tell if spouses are still in love.

Such attributes could be seen as strong predictors in the model, which means all predictors at each split of the tree are not necessary to consider. To improve accuracy of our prediction, random forest model is applied to decorrelate the bagging tree, with a subset of 7 predictors to consider at each split. Test error rate of random forest method for divorce mode is 1.90%.

3.6 SVM

Support Vector Classifier and Support Vector Machines are both intended for the binary classification setting in which there are two classes. They are both in the interest of greater robustness to individual observations, and better classication of most of the training observations. Both of these two methods can be represented as:

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i) \tag{7}$$

Where S is the collection of indices of these support points and K is kernel.

When we wish to get a linear boundary, we use linear kernel and the resulting classifier is known as Support Vector Classifier. When we assume the boundary is non-linear, we use a polynomial kernel of degree d, or a radial kernel with parameter λ, the resulting classifier is called Support Vector Machine.

Applying these two methods to the divorce data sets, we also use a cost argument, which allows us to specify the cost of a violation to the margin. When the cost argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. We then use cross-validation to choose the cost argument and parameter d or λ.

In our case, the estimate test error rates of Support Vector Classifier and Support vector Machine with radial kernel are both 1.92%, while the estimate test error rate of SVM with polynomial kernel is 0.00% (there is randomness in the result).

3.7 ANN

As some attributes might be stronger, artificial neural network is considered to improve the model. For better interpretation and lower computation assumption, feed-forward neural network is chosen in our analysis. In our case, answers to each attribute are collected as signals in input layer, with weight and activation function. This improves our model in a way that relationship between attributes inputs and their outputs are non-linear, which is correspondent with the real marriage situation; also, neurons in hidden layer serves as a filter for inputs, giving us a more efficient model.

To testify the improvement for different neural network models, four models have been considered in total: models with hidden layers and neurons of (1,1), (1,2), (2,1), (2,2). Graphic for neural network model with 1 hidden layer and 1 neuron is shown below:

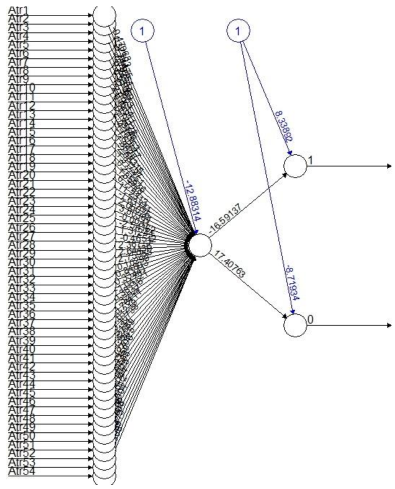

Figure 2: Neural Network with 1-hidden layer, 1-neuron

Test error rates for the models are 1.92%, 1.92%, 0%, 1.92% separately, which implies nueral network with 2-hidden layers, 1-neuron might be the best. As neural network model generates an output for `Class` as a probability value between 0 and 1, cross-entropy could be used to evaluate the models. The lowest cross-entropy errors among four models is 0.024 by model with one hidden layer and two neurons (the four errors are closed to each other, though).

4 Feature Selection

4.1 Stepwise Selection

We tried to find the important features in logistic regression by stepwise selection. We start with full model. In each step, we consider whether the ith variable belongs to the current variable set, if it exists, then we calculate AIC of the model after deleting this variable, if it does not exist, we calculate AIC of the model after adding this variable into current model. Each step we would get P different AIC for P different models, we chose the model that results in the smallest AIC and add or delete the relevant variable. We continue these steps until getting a model that no matter what variable we delete or add, we cannot get a smaller AIC. The variables in the final model are the variables that we consider important.

For the divorce data set, we use the training data set to perform stepwise selection, the variables remain in the final model are `Atr3`, `Atr26`, `Atr40`, which we think are important to logistic regression. We then build a logistic regression model based on these features, resulting in a 0 estimate test error rate. In this case, the feature selection may be effective.

4.2 Lasso Regression

As mentioned before, Lasso could force some coefficients exactly equal to zero, it performs variable selection. In our case, we apply Lasso to the training data set, the variables that has non-zero coefficients are `Atr3`, `Atr6`, `Atr15`, `Atr16`, `Atr17`, `Atr18`, `Atr19`, `Atr20`, `Atr26`, `Atr30`, `Atr33`, `Atr39`, `Atr40`, `Atr49`. We then build a logistic regression model based on these features selected by Lasso using the training data set, and test it with test data set. The estimate test error rate we get is 5.77%, which is smaller than 7.69%, the test error rate we get by applying logistic regression based on all features.

4.3 Tree Methods

From the result of tree pruning, possible important features are `Atr17` and `Atr20`. From the importance measure of random forest, `Atr26`, `Atr19`, `Atr17`, `Atr40` and `Atr20` may be important.

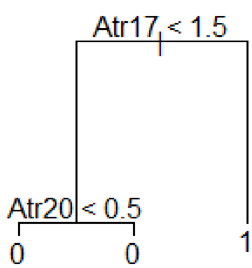

Figure 3: Tree Pruning Selection

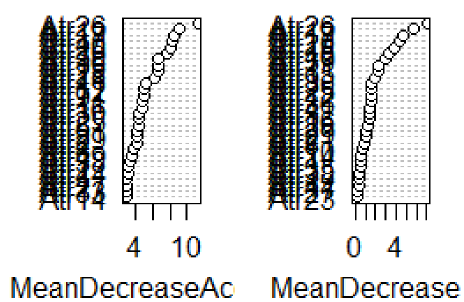

Figure 4: Random Forest Selection

5 Conclusion

In the analysis of `divorce` model, 12 machine learning methods are applied for variable prediction, with an average test error of 2.63%. To better compare test error rates between different models, a lollipop chart is attached below.

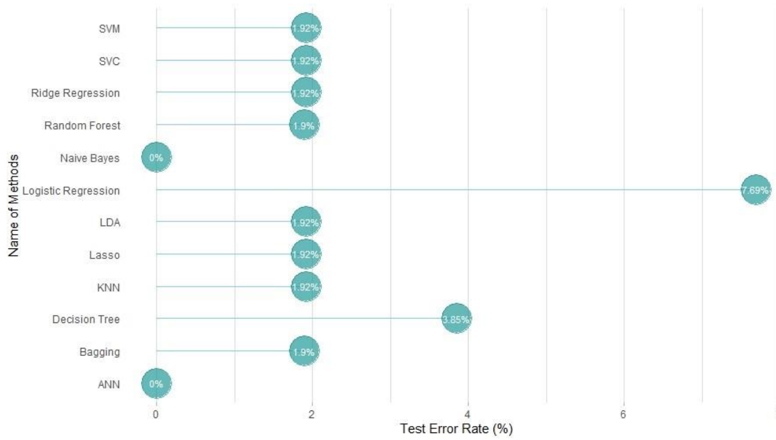

Figure 5: Test Error Rate for All Methods

As could be seen in the picture, both naive Bayes and neural network method have test error rate of 0. This might indicate they are the best two methods for model prediction; however, it can also be explained by the randomness of `set.seed()`, which gives us a well-separated testing dataset that perfectly corresponds with the classifier generated from the training set.

Also, as a result that `Class` values might be well separated as well, logistic regression method tends to have the largest test error rate among all methods.

Apart from these, test error rate for other methods are similar, which indicates that each of them could be seen as a strong and efficient method for building model `divorce` and predicting variable `Class`.

3 feature selection methods are applied to shrinkage the model into a more efficient one, each with a different subset of predictors while in general, some predictors occur more than once among all methods. For example, `3` (also the question: ”When we need it, we can take our discussions with my spouse from the beginning and correct it”). This might imply that there exists multicollinearity among attributes. In our case, it tells that marriage situation is not determined by separated problems: when some problems occur, others appear as well, leading to a deterioration in marriage.

Based on the results above, we could safely conclude that questions in the questionnaire would greatly determine whether a happy marriage would exist, while some questions have stronger effect compared to others.

6 Appendix: R

```

#################################################
#1.predictthedivorcebasedonallthefeatures
#################################################

#readdata
divorce =read.csv('D:/divorce.csv')
#dividedataintotrainingsetandtesttest
#attach(divorce)
set.seed(2)
train =sample(1:nrow(divorce),0.7 *nrow(divorce))
Class =as.factor(Class)
divorce =cbind(divorce[,-ncol(divorce)],Class)
divorce.train = divorce[train,]
divorce.test = divorce[-train,]


################################################
#Classificationmethod
################################################

#logisticregressiontopredict:
glm.fit =glm(Class~.,data=divorce.train,
              family=binomial,maxit=100)
contrasts(Class)
glm.pred =rep(0,nrow(divorce.test))
glm.probs =predict(glm.fit, newdata=divorce.test,
                    type='response')
glm.pred[glm.probs > 0.5] = 1
mean(glm.pred!=Class[-train])
#fittedprobabilitiesnumerically0or1occurred

#LDA
library(MASS)
lda.fit = lda(Class~.,data=divorce.train)
lda.pred =predict(lda.fit, divorce.test)
lda.class= lda.pred$class
mean(lda.class!=Class[-train])
table(lda.class,Class[-train])
```

```

#naivebayes
library(caret)
NBclassifier = naiveBayes(Class˜.,
                          data=divorce.train)
NBpred =predict(NBclassifier,
                 newdata=divorce.test,
                 type='class')
mean(NBpred!=Class[-train])

#KNN
library(class)
#usecross-validationtochoosek
set.seed(2)
fold =sample(1:5,length(train),replace=T)
cv.errors=matrix(NA,5,floor(length(fold)/5),
dimnames=list(NULL,paste(1:floor(length(fold)/5))))
for(i in 1:5){
  for(j in 1:floor(length(fold)/5)){
    knn.pred=knn(divorce.train[fold!=i,-ncol(divorce)]
            ,divorce.train[fold==i,-ncol(divorce)],
             Class[train][fold!=i],k=j)
cv.errors[i,j]=mean(knn.pred!=Class[train][fold==i])
  }
}
mean.cv.errors=apply(cv.errors ,2,mean)
knn.k =which.min(cv.errors)
knn.pred=knn(divorce.train[,-ncol(divorce)],
divorce.test[,-ncol(divorce)],Class[train],k=knn.k)
knn.error.test =mean(knn.pred!=Class[-train])
knn.error.test


#############################################
#ModelSelectionmethod
#############################################
#Lasso
library(glmnet)
divorce =read.csv('D:/divorce.csv')
x=as.matrix(divorce)
y=x[,ncol(x)]
```

```
x=as.matrix(divorce)[,-ncol(x)]
cv.out=cv.glmnet(x[train,],y[train],alpha=1,
                 family='binomial')
plot(cv.out)
bestlam=cv.out$lambda.min
lasso.mod=glmnet(x[train ,],y[train],alpha=1,
         lambda=bestlam,family='binomial')
lasso.probs=predict(lasso.mod,s=bestlam ,
                    newx=x[-train,])
lasso.pred =rep(0,nrow(divorce.test))
lasso.pred[lasso.probs > 0.5] = 1
mean(lasso.pred!=Class[-train])

#ridge
cv.out=cv.glmnet(x[train,],y[train],alpha=0,
                 family='binomial')
plot(cv.out)
bestlam=cv.out$lambda.min
lasso.mod=glmnet(x[train ,],y[train],alpha=0,
                 lambda=bestlam)
lasso.probs=predict(lasso.mod,s=bestlam ,
                    newx=x[-train,])
lasso.pred =rep(0,nrow(divorce.test))
lasso.pred[lasso.probs > 0.5] = 1
mean(lasso.pred!=Class[-train])


##########################################
#treemethods
##########################################
#Decisiontree
library(tree)
tree.divorce = tree(Class~.,divorce.train)
#summary(tree.divorce)
plot(tree.divorce)
text(tree.divorce,pretty=0)
tree.pred =predict(tree.divorce,
                    divorce.test, type='class')
mean(tree.pred!=Class[-train])

#Treepruning
```

```r
set.seed(2)
cv.divorce=cv.tree(tree.divorce,FUN=prune.misclass)
prune.divorce = prune.misclass(tree.divorce,
  best=cv.divorce$size[which.min(cv.divorce$dev)])
tree.pred =predict(prune.divorce,
            divorce.test, type='class')
mean(tree.pred!=Class[-train])
plot(prune.divorce)
text(prune.divorce,pretty=0)
#didnotcutnodes

#baggingandrandomforests;
library(randomForest)
set.seed(2)
bag.divorce<-randomForest(Class~.,
    data= divorce.train, mtry =ncol(divorce)-1,
    importance = TRUE)
#bag.divorce
pred.bagging<-predict(bag.divorce,
    newdata = divorce.test, type ="class")
table(pred.bagging, divorce.test$Class)
ER.bagging<-mean(pred.bagging!=divorce.test$Class)
cat("Testerrorrateforbaggingmethodis",
    ER.bagging*100,"%.")

set.seed(2)
rf.divorce=randomForest(Class~.,
      data=divorce.train,
      mtry=floor(sqrt(ncol(divorce)-1)),
      importance=TRUE)
yhat.rf=predict(rf.divorce,
      newdata=divorce.test,type='class')
mean(yhat.rf!=Class[-train])
importance(rf.divorce)
#toseewhichpredictorsaremoreimportant.
varImpPlot(rf.divorce)

#########################################
#SVM
#########################################
#Supportvectorclassifier
```

```r
library(e1071)
set.seed(2)
tune.out=tune(svm,Class~.,data=divorce.train,
kernel="linear",
ranges=list(cost=c(0.0001,0.001, 0.01, 0.1, 1,5,10,100)))
#summary(tune.out)
bestmod=tune.out$best.model
#summary(bestmod)
ypred=predict(bestmod,divorce.test)
mean(ypred!=Class[-train])

#Supportvectormachine
set.seed(2)
tune.out=tune(svm, Class~.,
data=divorce.train, kernel="radial",
ranges=list(cost=c(0.1,1,10,100,1000),
            gamma=c(0.01,0.1,0.5,1,2,3,4)))
#summary(tune.out)
ypred=predict(tune.out$best.model,divorce.test)
mean(ypred!=Class[-train])

set.seed(2)
tune.out=tune(svm, Class~.,data=divorce.train,
kernel="polynomial",
ranges=list(cost=c(0.1,1,10,100,1000),
            degree=c(2,3,4,5)))
#summary(tune.out)
ypred=predict(tune.out$best.model,divorce.test)
mean(ypred!=Class[-train])


###########################################
#neuralnetwork
##########################################
#1-hiddenlayer,1-neuron:
library(neuralnet)
library(tibble)
library(ggplot2)
set.seed(2)
model.nn1<-neuralnet(Class~.,
data= divorce.train, linear.output = FALSE,
```

```
err.fct ="ce", likelihood = TRUE)
#"linear.output&err.fct"forclassification;
#"likelihood"toseeAIC&BIC;
plot(model.nn1,rep="best")

nn1.CE<-model.nn1$result.matrix[1, 1]
nn1.AIC<-model.nn1$result.matrix[4, 1]
nn1.BIC<-model.nn1$result.matrix[5, 1]
#seetrainingerror&AIC&BIC.

paste("ForNN_model1,Cross-EntropyErroris",
round(nn1.CE, 3),",AICis",round(nn1.AIC, 3),
",BICis",round(nn1.BIC, 3),".")

pred.nn1<-compute(model.nn1, divorce.test)
#predictusingmodel.nn1.
pred.result<-data.frame(ActualValue = divorce.test$Class,
       prediction = pred.nn1$net.result[, 2])
#seethepredictionresultsinadataframe.
nn1.prediction<-round(pred.nn1$net.result[, 2], 0)
table(Actual = divorce.test$Class, Prediction = nn1.
   prediction)
#seeactualvaluesandpredictionresultsinatable.
ER.nn1<-mean(divorce.test$Class!=nn1.prediction)
#testerrorrate.
cat("Testerrorrateforneuralnetworkmethod
    with1-hiddeenlayeris",
    ER.nn1*100,"%.")

#comparewith1-hiddenlayer,2-neurons;
#2-hiddenlayers,1-neuron;2-hiddenlayers,2-neurons;
#1-layer,2-neurons:
set.seed(2)
model.nn2<-neuralnet(Class~.,data= divorce.train,
linear.output = FALSE, err.fct ="ce",
likelihood = TRUE, hidden =c(1,2))
nn2.CE<-model.nn2$result.matrix[1, 1]
nn2.AIC<-model.nn2$result.matrix[4, 1]
nn2.BIC<-model.nn2$result.matrix[5, 1]
paste("ForNN_model2,Cross-EntropyErroris",
      round(nn2.CE, 3),",AICis",round(nn2.AIC, 3),
```

```
      ",BICis",round(nn2.BIC, 3),".")

pred.nn2<-compute(model.nn2, divorce.test)
nn2.prediction<-round(pred.nn2$net.result[, 2], 0)
ER.nn2<-mean(divorce.test$Class!=nn2.prediction)
cat("Testerrorrateforneuralnetworkmethod
with1-hiddeenlayer,2-neuronsis", ER.nn2    *100,"%.")

#2-layers,1-neuron:
set.seed(2)
model.nn3<-neuralnet(Class~.,data= divorce.train,
  linear.output = FALSE, err.fct ="ce",
  likelihood = TRUE, hidden =c(2,1))
nn3.CE<-model.nn3$result.matrix[1, 1]
nn3.AIC<-model.nn3$result.matrix[4, 1]
nn3.BIC<-model.nn3$result.matrix[5, 1]
paste("ForNN_model3,Cross-EntropyErroris",
      round(nn3.CE, 3),",AICis",round(nn3.AIC, 3),
      ",BICis",round(nn3.BIC, 3),".")

pred.nn3<-compute(model.nn3, divorce.test)
nn3.prediction<-round(pred.nn3$net.result[, 2], 0)
ER.nn3<-mean(divorce.test$Class!=nn3.prediction)
cat("Testerrorrateforneuralnetworkmethod
with2-hiddeenlayers,1-neuronis", ER.nn3    *100,"%.")

#2-layers,2-neurons:
set.seed(2)
model.nn4<-neuralnet(Class~.,
data= divorce.train, linear.output = FALSE,
err.fct ="ce", likelihood = TRUE, hidden =c(2,2))
nn4.CE<-model.nn4$result.matrix[1, 1]
nn4.AIC<-model.nn4$result.matrix[4, 1]
nn4.BIC<-model.nn4$result.matrix[5, 1]
paste("ForNN_model4,Cross-EntropyErroris",
round(nn4.CE, 3),",AICis",round(nn4.AIC, 3),
",BICis",round(nn4.BIC, 3),".")

pred.nn4<-compute(model.nn4, divorce.test)
nn4.prediction<-round(pred.nn4$net.result[, 2], 0)
ER.nn4<-mean(divorce.test$Class!=nn4.prediction)
```

```
cat("Testerrorrateforneuralnetworkmethod
with2-hiddeenlayers,2-neuronsis", ER.nn4    *100,"%.")

#seeinalollipopplot:
library(ggplot2)

data<-data.frame(
  x=c("LogisticRegression","LDA","NaiveBayes",
     "KNN","Lasso","RidgeRegression",
     "DecisionTree","Bagging","RandomForest",
     "SVC","SVM","ANN"),
  y=c(7.69, 1.92, 0, 1.92, 1.92, 1.92,
      3.85, 1.92, 1.92, 1.92, 1.92, 1.92)
)

ggplot(data, aes(x=x, y=y, label = paste0(y,"%"))) +
  geom_segment( aes(x=x, xend=x, y=0, yend=y),
                color="skyblue") +
  geom_point( color="darkcyan", size=12, alpha=0.6) +
  theme_light() +
  coord_flip() +
  geom_text(color ="white", size = 3)+
  theme(
    panel.grid.major.y = element_blank(),
    panel.border = element_blank(),
    axis.ticks.y = element_blank()
  ) +
  xlab("NameofMethods") +
  ylab("TestErrorRate(%)")


########################################
#Choosingimportantfeatures
########################################

########################################
#stepwiseselection
########################################
mink=0
stepselect =function(data){
  np =dim(data)
```

```
    n = np[1]
    p = np[2]-1
    xn =names(data)[1:p]
    #data=cbind(1,data)
    glm.fit =glm(Class~.,data=data,
                  family=binomial)
    hhb =glm.fit$coefficients
    #loglik=logLik(glm.fit)
    #AIC=-2  *loglik+2  *(p+1)
    aic = AIC(glm.fit)
    A = 1:p
    MAIC = aic
    mAIC = aic
    flag =rep(FALSE,p)
  #markwhetherthevariableisdeleted,false=deleted
    repeat{
      if(length(A) < p ){
        B = (1:p)[flag]
      }else{
        B = NULL
      }
      AB =c(A,B)
      AICm =rep(0,p)
  #recordAICafteraddingorremovingthekthvariable
      ff = 1
      for(k in AB){
        if(!flag[k]){
          tA = A[-ff]
          formula=paste('Class~1',
                  paste(paste('Atr',tA,sep='')
                  ,collapse ='+'),sep='+')
          glm.fitk =glm(formula,
                  data=data,family=binomial)
          #loglikk=logLik(glm.fitk)
          #AICk=-2  *loglikk+2  *(length(A)-1)
          AICk = AIC(glm.fitk)
          AICm[k] = AICk
          ff = ff+1
          if( AICk < mAIC ){
  #ifAICisbecomingsmaller,recordcurrentk,
            #variableset,AIC
```

```
            mtA = tA
            mAIC = AICk

          }
        }else{
          tA<-c(A,k)
          formula=paste('Class~1',
                  paste(paste('Atr',tA,sep=''),
                  collapse ='+'),sep='+')
          glm.fitk =glm(formula,
                  data=data,family=binomial)
          #loglikk=logLik(glm.fitk)
          #AICk=-2  *loglikk+2  *(length(A)-1)
          AICk = AIC(glm.fitk)
          AICm[k] = AICk
          if( AICk < mAIC ){
            mink = k
            mtA = tA
            mAIC = AICk
          }
        }
      }
      if(mAIC >= MAIC)break
    #ifAICnotdecreaseafteraddingorremovingvariable
      if(mAIC<MAIC ){
        flag[mink] =!flag[mink]
        A = mtA
        MAIC = mAIC#smallestAIC
      }
    }
    formulaglm =paste('Class~1'
                  ,paste(paste('Atr',A,sep='')
                 ,collapse ='+'),sep='+')
    glm.fit =glm(formulaglm,data=data,
                 family=binomial)
    b =glm.fit$coefficients
    bm =rep(0,(p+1))
    bm[c(1,A+1)]<-b
    hhb = bm
    glmopt =paste('Atr',A,sep='')
    return(glmopt)
```

```
}

coefi2=stepselect(divorce.train)
#logisticregressiontopredict:
glm.fit =glm(Class~.,
              data=divorce.train[,c(coefi2,'Class')],
              family=binomial,maxit=100)
contrasts(Class)
glm.pred =rep(0,nrow(divorce.test))
glm.probs =predict(glm.fit,
          newdata=divorce.test[,c(coefi2,'Class')],
          type='response')
glm.pred[glm.probs > 0.5] = 1
mean(glm.pred!=Class[-train])


##########################################
#Lasso
##########################################
divorce =read.csv('D:/divorce.csv')
x=as.matrix(divorce.train)
y=x[,ncol(x)]
x=as.matrix(divorce.train[,-ncol(x)])
cv.out=cv.glmnet(x,y,alpha=1,family='binomial')
bestlam=cv.out$lambda.min
lasso.mod=glmnet(x,y,alpha=1,lambda=bestlam)
lasso.coef=coef(lasso.mod)[,1]
coefi =names(lasso.coef[lasso.coef!=0])[-1]
#testwith
#logisticregressiontopredict:
glm.fit =glm(Class~.,
    data=divorce.train[,c(coefi,'Class')],
    family=binomial,maxit=100)
contrasts(Class)
glm.pred =rep(0,nrow(divorce.test))
glm.probs =predict(glm.fit,
    newdata=divorce.test[,c(coefi,'Class')],
    type='response')
glm.pred[glm.probs > 0.5] = 1
mean(glm.pred!=Class[-train])
```