ISYE 6501 Intro Analytics Modeling - HW1

Question 2.1: Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

A classification model will help to **build target lists for email marketing campaigns**. A successful email marketing campaign needs a target list full of qualified customer that is interested in what the company has to offer, otherwise no matter how good the newsletter is, it will be marked as unread or unsubscribed along with a barrage of other automated email newsletters.

To build this classification model, I think consumer's registration personal information, membership history, email activity, and order history are all good predictors.

Predictors are listed below:

1. **Age** (Numeric): we can get this feature from members' date of birth. Purchase behaviors and product types vary greatly by age, receptivity to email is also varied by age a lot.
2. **Gender** (Binary F/M)**:** The same as age, male and female have a big difference in receptivity to marketing emails.
3. **Length of subscription** (Numeric): How many months has this member subscribed. I think the longer one subscribed means the higher his loyalty is. Those people might be more willing to know about new information.
4. **Whether open any marketing email within the last 3 months** (Binary Y/N): This is from email activity history, if one person didn't open any email from the last 3 months' campaign, I assume this person can be removed from the target lists.
5. **Purchased amount within the last 3 months** (Numeric): The purchased amount within the last 3 months can indicate how activity one consumer is. I assume those people will have a higher receptivity.

Question 2.2

Loading and examining data

The first step is to read in the dataset and do some simple data summary. From the code below, I get to know this dataset contains 654 **data points**, 10 **predictors** (6 numeric and 4 binary), and one binary (1/0) **response**. The **event rate** (percentage of R1 equal to 1) is 45.3%. From the summary of each variable, I found the **data range** of 6 numeric variables varies a lot: the maximum A15 value is in the ten of hundreds while others are less than one hundred, so **data scaling** will be necessary on modeling.

1. Using the SVM in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set.

In this step, I tried 12 C values (cost) in different magnitude (10^{-4} to 10^{7}) to find the best classifier *(Fig.1).* The first figure shows how the error changes across different C value. Within the C values from **10^{-2} to 10^{2},** those models have the **smallest error: 0.136**. which means among 654 data points those models can predict 565 correctly.

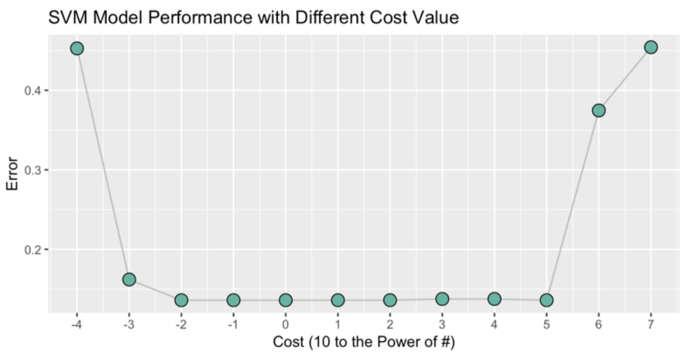
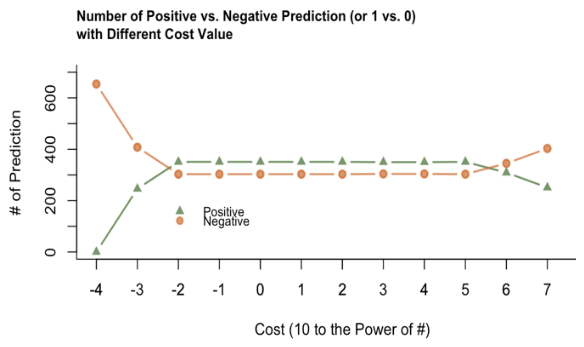


Fig.1

To figure out why too big or too small cost value has a big error. I plot how many data points are predicted as positive (or 1) and negative (or 0) at different C values *(Fig.2)*. From the figure, when C=10^{-4} the model predicts all most everything as 0, so It has a very high Specificity (When it's actually 0, how often does it predict 0). When C is too big (>10^5), the model starts to predict more 0, but both Specificity and sensitivity start to decrease.



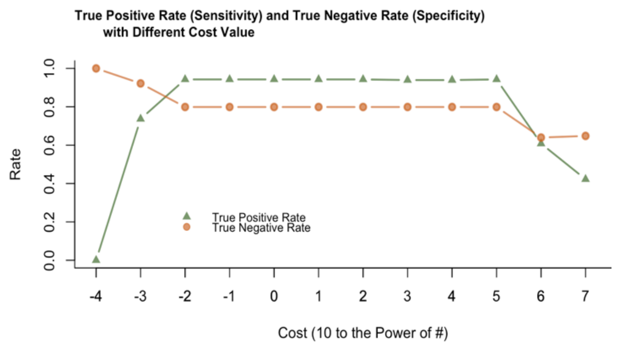


Fig.2

Basically, classifiers with c value 10^{-2}, 10^{-1}, 1, 10, 10^2 gave us the same prediction. So, I will just list the equation for the classifier with C value=1.

```
Support Vector Machine object of class "ksvm
        SV type: C-svc (classification)
        parameter: cost C = 1
        Linear (vanilla) kernel function.
        Number of Support Vectors: 190
        Objective Function Value: -179.385
       Training error: 0.136086
```
Equation: $y_j^{predicted} = -1.102664e-03*A1_j + (-8.980539e-04)*A2_j + (-1.607456e-03)*A3_j + 2.904170e-03*A8_j + 1.0047363*A9_j + (-2.985211e-03)*A10_j + (-2.035179e-04)*A11_j + (-5.504803e-04)*A12_j + (-1.251919e-03)*A14_j+1.064405e-01*A15_j + (-8.148382e-02)$

Confusion Matrix:

		Actual Response		
		1 *(n=296)*	0 *(n=358)*	
Linear SVM Classifier	*1 (n=351)*	279 *True positives*	72 False positives	Positive Predictive Value (PPV) 79.5%
	0 (n=303)	17 *False negatives*	286 *True negatives*	Negative Predictive Value (NPV) 94.4%
		Sensitivity 94.3%	*Specificity* 80.0%	

2. Try other (nonlinear) kernels as well; we're not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

I tried even other kernel functions, they are *Rbfdot*: Radial Basis kernel "Gaussian", *polydot*: Polynomial kernel, *tanhdot*: Hyperbolic tangent kernel, *laplacedot*: Laplacian kernel, *besseldot*: Bessel kernel, *anovadot*, ANOVA RBF kernel and, *splinedot*: Spline kernel.

Under the same parameter, there are three functions perform better than Linear kernel function. They are splinedot, rbdot, and besseldot. **Splinedot** has the **smallest error: 0.034** *(Fig.3)*.

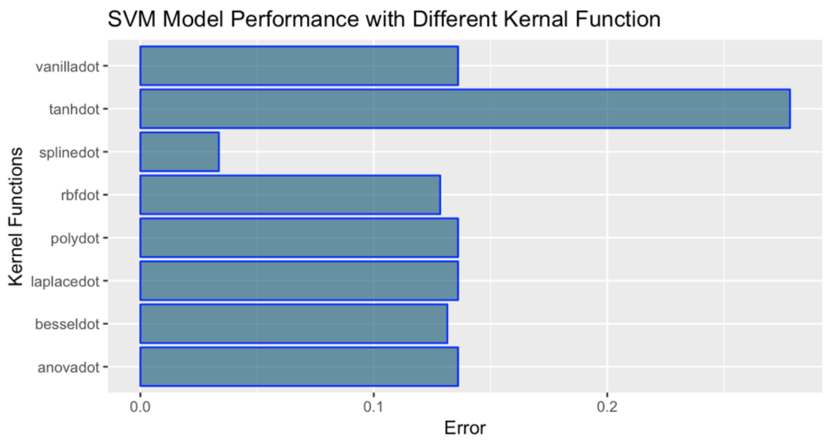


Fig.3

Because its non-linear, it has a larger number of support vectors than linear function, so it can do better at classifying. Both sensitivity and specificity are higher than the linear model. Here is the detail of Splinedot model:

```
Support Vector Machine object of class "ksvm"
    SV type: C-svc (classification);
    parameter: cost C = 1;
    Spline kernel function;
    Number of Support Vectors: 216;
    Objective Function Value: -65.4605;
    Training error: 0.033639
    Confusion Matrix:
```

		Actual Response		
		1 *(n=296)*	0 *(n=358)*	
Nonlinear SVM Classifier	*1* *(n=296)*	285 *True positives*	11 False positives	Positive Predictive Value (PPV) 96.3%
	0 *(n=358)*	11 *False negatives*	347 *True negatives*	Negative Predictive Value (NPV) 96.9%
		Sensitivity 96.3%	*Specificity* 96.9%	

3. Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set.

To build KNN models, I randomly selected **70%** of data points as the training set and the rest 30% as the testing set. To find out the best K value, I tried k value from 1 to 15. When **K=13**, I get the **smallest error: 0.127** *(Fig.4).* which is smaller than linear SVM. Below is the detail of the KNN model:

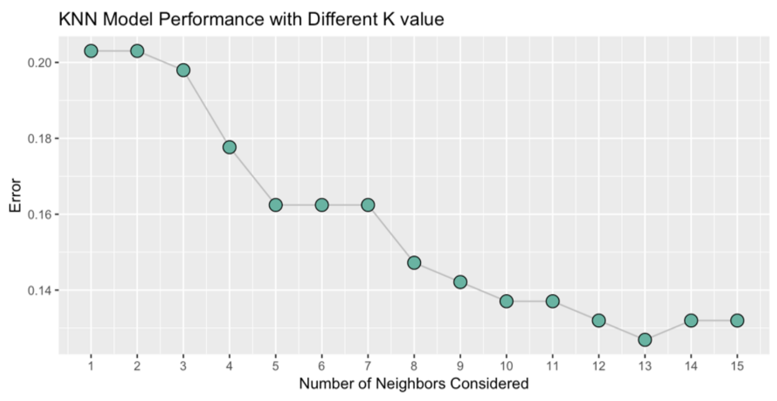


Fig.4

```
parameter: k = 13;
       distance = 2;
       kernel = "triangular"
       scale = TRUE
```

```
Confusion Matrix (Only Test set):
```

		Actual Response		
		1 *(n=93)*	0 *(n=104)*	
KNN Classifier	*1 (n=94)*	81 *True positives*	13 False positives	Positive Predictive Value (PPV) 86.2%
	0 (n=103)	12 *False negatives*	91 *True negatives*	Negative Predictive Value (NPV) 88.3%
		Sensitivity 87.1%	*Specificity* 87.5%	

The Code

```r
# ISYE 6501 Intro Analytics Modeling - HW1 Q2
# IP credit_card_data-headers.txt

# Loading and examining data
df<-read.delim("data 2.2/credit_card_data-headers.txt", header = TRUE, sep = "\t")

head(df,2)          ## Display Head Lines
attributes(df)      ## Show attributes, number of row, and number of col
ncol(df)             ## Number of Col
nrow(df)            ## Number of Row

for (i in attributes(df)$names ){
  print(paste0('Name:',i,'   Class:',class(df[[i]]),'   nlevels:',length(unique(df[[i]]))))
  print(summary(df[[i]]))
  }             ## Show the type, number of unique value, and summary of each variable

length(df$R1[df$R1=="1"])/nrow(df)                ## Event Rate

# 1.Using the SVM in the R package kernlab, find a good classifier for this data. Show the equ
ation of your classifier, and how well it classifies the data points in the full data set. (Do
n’t worry about test/validation data yet; we’ll cover that topic soon.)

library(kernlab)

cost<-c(10^(-4:7))                                ## Create a table to record models results
summary_table <- data.frame(row.names=paste0("C",cost),c= c(-4:7))

for (c in cost){                                 ## Create classifier with different C value
  svm <- ksvm(  as.matrix(df[,1:10]),
                as.factor(df[,11]),
                type="C-svc",
                kernel="vanilladot",
                C=c,
                scaled=TRUE   )

  for (x in 1:10){
      summary_table[paste0("C",c),paste0("a",x)]= colSums(svm@xmatrix[[1]] * svm@coef[[1]])[x]
    }                                              ## calculate a1...am

  summary_table[paste0("C",c),"a0"]= svm@b                            ## Record a0
  summary_table[paste0("C",c),"error"]=as.numeric(svm@error)          ## Record error

  fitted(model)
  summary_table[paste0("C",c),"tn"] = table(df$R1, fitted(svm))[1] ## True negative
  summary_table[paste0("C",c),"fn"] = table(df$R1, fitted(svm))[2] ## False positive
  summary_table[paste0("C",c),"fp"] = table(df$R1, fitted(svm))[3] ## False negative
  summary_table[paste0("C",c),"tp"] = table(df$R1, fitted(svm))[4] ## True positive
}

summary_table$P=summary_table$tp+summary_table$fp
summary_table$N=summary_table$tn+summary_table$fn
summary_table$TPR=summary_table$tp/(summary_table$tp+summary_table$fn)  ## True positive rate
summary_table$TNR=summary_table$tn/(summary_table$tn+summary_table$fp)  ## Ture negative rate

# Visualization
library(ggplot2)
library(dplyr)

p1<-ggplot( summary_table, aes(x=c, y=error)) +                 ## Fig 1. Cost vs. Error
  geom_line( color="grey") +
  geom_point(shape=21, color="black", fill="#69b3a2", size=4) +
  labs(title ="SVM Model Performance with Different Cost Value")+
  xlab("Cost (10 to the Power of #)") + ylab("Error")+
  scale_x_continuous(breaks = c(-4:7))

p2<-plot(summary_table$P~summary_table$c , type="b" , bty="l" , ## Fig 2. too large/small Cost
         xlab="Cost (10 to the Power of #)" ,
         ylab="# of Prediction" ,
         col=rgb(0.2,0.4,0.1,0.7) ,
         lwd=2 , pch=17, xlim=c(-4,7), ylim=c(0,700))+
    lines(summary_table$N~summary_table$c,
         col=rgb(0.8,0.4,0.1,0.7) ,
         lwd=2 , pch=19 , type="b" ) +
    axis(side=1, at=seq(-4, 7, by=1), labels =c(-4:7)) +
    title("Number of Positive vs. Negative Prediction (or 1 vs. 0)
           with Different Cost Value",adj =0,cex.main=0.9) +
    legend("bottomleft", legend = c("Positive", "Negative"),
       col = c(rgb(0.2,0.4,0.1,0.7),  rgb(0.8,0.4,0.1,0.7)), pch = c(17,19), bty = "n",
       pt.cex = 1, cex = 0.8, text.col = "black", horiz = F , inset = c(0.15, 0.15))

# 2. You are welcome, but not required, to try other (nonlinear) kernels as well; we’re not co
vering them in this course, but they can sometimes be useful and might provide better predicti
ons than vanilladot.

kernel<-c("rbfdot","polydot","vanilladot","tanhdot","laplacedot",
          "besseldot","anovadot","splinedot")
                                                        ## Create classifier with different kernel function

summary_table2 <- data.frame(row.names=kernel)
summary_table2$model=kernel

for (k in kernel){
  svm <- ksvm(  as.matrix(df[,1:10]),
                as.factor(df[,11]),
                type="C-svc",
                kernel=k,
                C=1,
                scaled=TRUE  )

  for (x in 1:10){                                              ## calculate a1...am
    summary_table2[k,paste0("a",x)]= colSums(svm@xmatrix[[1]] * svm@coef[[1]])[x]
  }

  summary_table2[k,"a0"]= svm@b
  summary_table2[k,"error"]=as.numeric(svm@error)

  summary_table2[k,"tn"] = table(df$R1, fitted(svm))[1]
  summary_table2[k,"fn"] = table(df$R1, fitted(svm))[2]
  summary_table2[k,"fp"] = table(df$R1, fitted(svm))[3]
  summary_table2[k,"tp"] = table(df$R1, fitted(svm))[4]
}

summary_table2$P=summary_table2$tp+summary_table2$fp
summary_table2$N=summary_table2$tn+summary_table2$fn
summary_table2$TPR=summary_table2$tp/(summary_table2$tp+summary_table2$fn)
summary_table2$TNR=summary_table2$tn/(summary_table2$tn+summary_table2$fp)

p3<-ggplot(summary_table2, aes(x=model, y=error)) +              ## Fig 3. Functions vs. Error
    geom_bar(stat = "identity",color="blue", fill=rgb(0.1,0.4,0.5,0.7)) +
    coord_flip()+xlab("Kernel Functions") +ylab("Error") +
    labs(title ="SVM Model Performance with Different Kernal Function")

# 3.Using the k-nearest-neighbors classification function kknn contained in the R kknn packag
e, suggest a good value of k, and show how well it classifies that data points in the full dat
a set. Don’t forget to scale the data (scale=TRUE in kknn).

set.seed(101)               # Set Seed so that same sample can be reproduced in future
                            # Selecting 70% of data as sample from total 'n' rows of the data
sample <- sample.int(n = nrow(df), size = floor(.7*nrow(df)), replace = F)
train <- df[sample,c(1:11) ]
test  <- df[-sample,c(1:11) ]

library(kknn)
summary_table3 <- data.frame(row.names= c(1:15),k=c(1:15))
for (x in c(1:15)){
      KNN<-kknn(R1~., train, test, distance =2,scale=TRUE, k=13,
                kernel = "triangular")
summary_table3[x,"tn"] = table(test$R1, round(KNN$fit))[1]
summary_table3[x,"fn"] = table(test$R1, round(KNN$fit))[2]
summary_table3[x,"fp"] = table(test$R1, round(KNN$fit))[3]
summary_table3[x,"tp"] = table(test$R1, round(KNN$fit))[4]
summary_table3[x,"error"]=(summary_table3[x,"fn"] +summary_table3[x,"fp"])/nrow(test)
}

p4<-ggplot( summary_table3, aes(x=k, y=error)) +                ## Fig 4. K values vs. Error
geom_line( color="grey") +
geom_point(shape=21, color="black", fill="#69b3a2", size=4) +
labs(title ="KNN Model Performance with Different K value")+ xlab("Number of Neighbors Conside
red") +ylab("Error")+
scale_x_continuous(breaks = c(1:15))
```