Assignment 3 solutions

Question 1: SVM on the OJ data

(a) [2 points] *Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.*

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.1.1
```

```
set.seed(101)
train=sample(nrow(OJ),800)
OJ.train = OJ[train,]
OJ.test = OJ[-train,]
```

(b) [3 points] *Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.*

A support vector classifier corresponds to `svm` with `kernel=linear`.

[1 point of 3 above for intelligent treatment of these variables] Note that in the code below I discover that `Store7` has the same information as `STORE` and `StoreID`. The variable "Store7" is an indicator for one of the stores. There appear to be 5 stores, labelled 1, 2, 3, 4, 7 in `StoreID` and 0, 1, 2, 3, 4 in `STORE`. We'll use `STORE` as a factor in the model.

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.1.1
```

```
table(OJ$STORE,as.factor(OJ$STORE))
```

```
##
##        0   1   2   3   4
##    0 356   0   0   0   0
##    1   0 157   0   0   0
##    2   0   0 222   0   0
##    3   0   0   0 196   0
##    4   0   0   0   0 139
```

```
OJ$STORE = as.factor(OJ$STORE)
OJ$Store7 = NULL
OJ$StoreID = NULL
OJ.train = OJ[train,]  # redo the train test split for the modified data...
OJ.test = OJ[-train,]

svm1 = svm(Purchase~.,data=OJ.train,kernel='linear',cost=0.01)
summary(svm1)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05263
##
## Number of Support Vectors:  437
##
##  ( 218 219 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

We see that the model selects 437 out of 800 observations as support points. The summary doesn't tell us much else that is useful, other than that we are indeed predicting 2 classes.

(c) [2 points] *What are the training and test error rates?*

The code below indicates that in training, we are getting about 83-84% right (16-17% misclassified), and in the test set we get similar results.

```r
source("http://www.mathstat.dal.ca/~aarms2014/StatLearn/R/A2funs.R")
svm1.train.pred = predict(svm1,newdata=OJ.train)
class.table(obs=OJ.train$Purchase,pred=svm1.train.pred)
```

```
##     pred
## obs    CH   MM
##   CH 89.5 10.5
##   MM 25.2 74.8
## overall: 83.8
```

```r
svm1.test.pred = predict(svm1,newdata=OJ.test)
class.table(obs=OJ.test$Purchase,pred=svm1.test.pred)
```

```
##     pred
## obs    CH   MM
##   CH 88.6 11.4
##   MM 24.0 76.0
## overall: 83.7
```

(d) [2 points] *Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.*

I considered values on a semi log scale (i.e. increasing orders of magnitude, with multiples of 1, 2, 5 in an order of magnitude.). Note that you have to specify `kernel="linear"` The summary below suggests that a wide range of values of `cost` gives essentially the same performance. I re-ran the same code several times, corresponding to different random divisions of the data into folds. Quite different values of `cost` were selected, although the cross-validated misclassification rate was usually close to 17% in all cases.

```
svm1.tune = tune(svm,Purchase~.,data=OJ.train,
                 ranges=list(cost=c(.01,.02,.05,.1,.2,.5,1,2,5,10)),kernel='linear')
summary(svm1.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.05
##
## - best performance: 0.17
##
## - Detailed performance results:
##     cost  error dispersion
## 1   0.01 0.1750    0.03909
## 2   0.02 0.1713    0.03775
## 3   0.05 0.1700    0.03782
## 4   0.10 0.1713    0.03682
## 5   0.20 0.1713    0.03336
## 6   0.50 0.1725    0.03670
## 7   1.00 0.1738    0.03701
## 8   2.00 0.1713    0.03538
## 9   5.00 0.1725    0.03623
## 10 10.00 0.1725    0.04158
```

(e) [2 points] *Compute the training and test error rates using this new value for cost.*

The R code below selects the best model and predicts for the training and test sets using that model. In the case I ran, the accuracy of the "best"" model on the test set is actually slightly lower than for the svm that used `cost=0.01` in (b) and (c). I think that this is likely due to random variation in the train/test split.

```
svm1.best.train.pred = predict(svm1.tune$best.model,newdata=OJ.train)
class.table(obs=OJ.train$Purchase,pred=svm1.best.train.pred)
```

```
##     pred
## obs    CH   MM
##   CH 89.5 10.5
##   MM 24.3 75.7
## overall: 84.1
```

```
svm1.best.test.pred = predict(svm1.tune$best.model,newdata=OJ.test)
class.table(obs=OJ.test$Purchase,pred=svm1.best.test.pred)
```

```
##      pred
## obs     CH   MM
##    CH 89.2 10.8
##    MM 21.2 78.8
## overall: 85.2
```

(f) [2 points] *Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma. NOTE: I requested that you tune both gamma and cost.*

```
svm2.tune = tune(svm , Purchase~. , data=OJ.train , ranges=list(
  cost=c(.01,.02,.05,.1,.2,.5,1,2,5,10),gamma=c(.001,.002,.005,.01,.02,.05,.1,.2,.5,1,2,5,10)),
  kernel='radial')
summary(svm2.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     5 0.005
##
## - best performance: 0.1625
##
## - Detailed performance results:
##      cost gamma  error dispersion
## 1    0.01 1e-03 0.3912    0.06720
## 2    0.02 1e-03 0.3912    0.06720
## 3    0.05 1e-03 0.3912    0.06720
## 4    0.10 1e-03 0.3912    0.06720
## 5    0.20 1e-03 0.3912    0.06720
## 6    0.50 1e-03 0.3387    0.07872
## 7    1.00 1e-03 0.1925    0.03689
## 8    2.00 1e-03 0.1750    0.03584
## 9    5.00 1e-03 0.1750    0.03819
## 10  10.00 1e-03 0.1688    0.03920
## 11   0.01 2e-03 0.3912    0.06720
## 12   0.02 2e-03 0.3912    0.06720
## 13   0.05 2e-03 0.3912    0.06720
## 14   0.10 2e-03 0.3912    0.06720
## 15   0.20 2e-03 0.3850    0.08203
## 16   0.50 2e-03 0.1925    0.03594
## 17   1.00 2e-03 0.1762    0.03459
## 18   2.00 2e-03 0.1738    0.03654
## 19   5.00 2e-03 0.1687    0.03964
## 20  10.00 2e-03 0.1638    0.03884
## 21   0.01 5e-03 0.3912    0.06720
## 22   0.02 5e-03 0.3912    0.06720
## 23   0.05 5e-03 0.3912    0.06720
## 24   0.10 5e-03 0.3575    0.08482
## 25   0.20 5e-03 0.1925    0.04495
## 26   0.50 5e-03 0.1725    0.04670
## 27   1.00 5e-03 0.1725    0.04116
## 28   2.00 5e-03 0.1637    0.03929
## 29   5.00 5e-03 0.1625    0.03819
## 30  10.00 5e-03 0.1638    0.03839
## 31   0.01 1e-02 0.3912    0.06720
## 32   0.02 1e-02 0.3912    0.06720
## 33   0.05 1e-02 0.3837    0.07841
## 34   0.10 1e-02 0.2038    0.05745
## 35   0.20 1e-02 0.1700    0.04175
## 36   0.50 1e-02 0.1762    0.04144
## 37   1.00 1e-02 0.1650    0.03810
## 38   2.00 1e-02 0.1663    0.04169
## 39   5.00 1e-02 0.1663    0.04127
## 40  10.00 1e-02 0.1688    0.04259
## 41   0.01 2e-02 0.3912    0.06720
## 42   0.02 2e-02 0.3912    0.06720
## 43   0.05 2e-02 0.2487    0.06108
## 44   0.10 2e-02 0.1775    0.04402
## 45   0.20 2e-02 0.1688    0.04459
## 46   0.50 2e-02 0.1700    0.04378
## 47   1.00 2e-02 0.1675    0.04091
## 48   2.00 2e-02 0.1663    0.04251
## 49   5.00 2e-02 0.1750    0.04249
## 50  10.00 2e-02 0.1763    0.03839
## 51   0.01 5e-02 0.3912    0.06720
## 52   0.02 5e-02 0.3912    0.06720
## 53   0.05 5e-02 0.2038    0.04210
## 54   0.10 5e-02 0.1725    0.04743
## 55   0.20 5e-02 0.1700    0.04758
## 56   0.50 5e-02 0.1713    0.04412
## 57   1.00 5e-02 0.1713    0.04642
## 58   2.00 5e-02 0.1800    0.04378
## 59   5.00 5e-02 0.1750    0.04640
## 60  10.00 5e-02 0.1800    0.04338
## 61   0.01 1e-01 0.3912    0.06720
## 62   0.02 1e-01 0.3912    0.06720
## 63   0.05 1e-01 0.2275    0.05164
## 64   0.10 1e-01 0.1787    0.04967
## 65   0.20 1e-01 0.1800    0.03873
## 66   0.50 1e-01 0.1738    0.04185
## 67   1.00 1e-01 0.1787    0.04753
## 68   2.00 1e-01 0.1825    0.04610
## 69   5.00 1e-01 0.1800    0.04005
## 70  10.00 1e-01 0.1837    0.03336
## 71   0.01 2e-01 0.3912    0.06720
## 72   0.02 2e-01 0.3912    0.06720
## 73   0.05 2e-01 0.3125    0.06428
## 74   0.10 2e-01 0.2025    0.05130
## 75   0.20 2e-01 0.1825    0.04417
## 76   0.50 2e-01 0.1825    0.04091
## 77   1.00 2e-01 0.1775    0.04241
## 78   2.00 2e-01 0.1863    0.03408
## 79   5.00 2e-01 0.1950    0.04005
## 80  10.00 2e-01 0.2050    0.03918
## 81    0.01 5e-01 0.3912    0.06720
## 82    0.02 5e-01 0.3912    0.06720
## 83    0.05 5e-01 0.3912    0.06720
## 84    0.10 5e-01 0.2925    0.04902
## 85    0.20 5e-01 0.2150    0.04200
## 86    0.50 5e-01 0.2037    0.03489
## 87    1.00 5e-01 0.1988    0.03654
## 88    2.00 5e-01 0.1988    0.03884
## 89    5.00 5e-01 0.2112    0.04619
## 90   10.00 5e-01 0.2188    0.04832
## 91    0.01 1e+00 0.3912    0.06720
## 92    0.02 1e+00 0.3912    0.06720
## 93    0.05 1e+00 0.3912    0.06720
## 94    0.10 1e+00 0.3588    0.06694
## 95    0.20 1e+00 0.2437    0.05441
## 96    0.50 1e+00 0.2188    0.04536
## 97    1.00 1e+00 0.2112    0.04308
## 98    2.00 1e+00 0.2100    0.04780
## 99    5.00 1e+00 0.2162    0.04679
## 100  10.00 1e+00 0.2213    0.04825
## 101   0.01 2e+00 0.3912    0.06720
## 102   0.02 2e+00 0.3912    0.06720
## 103   0.05 2e+00 0.3912    0.06720
## 104   0.10 2e+00 0.3837    0.06797
## 105   0.20 2e+00 0.2963    0.05775
## 106   0.50 2e+00 0.2350    0.05676
## 107   1.00 2e+00 0.2225    0.05458
## 108   2.00 2e+00 0.2137    0.04388
## 109   5.00 2e+00 0.2250    0.04526
## 110  10.00 2e+00 0.2425    0.05144
## 111   0.01 5e+00 0.3912    0.06720
## 112   0.02 5e+00 0.3912    0.06720
## 113   0.05 5e+00 0.3912    0.06720
## 114   0.10 5e+00 0.3912    0.06720
## 115   0.20 5e+00 0.3312    0.05629
## 116   0.50 5e+00 0.2612    0.06413
## 117   1.00 5e+00 0.2400    0.05707
## 118   2.00 5e+00 0.2463    0.06040
## 119   5.00 5e+00 0.2475    0.05798
## 120  10.00 5e+00 0.2450    0.05780
## 121   0.01 1e+01 0.3912    0.06720
## 122   0.02 1e+01 0.3912    0.06720
## 123   0.05 1e+01 0.3912    0.06720
## 124   0.10 1e+01 0.3912    0.06720
## 125   0.20 1e+01 0.3463    0.06182
## 126   0.50 1e+01 0.2888    0.06933
## 127   1.00 1e+01 0.2475    0.05827
## 128   2.00 1e+01 0.2538    0.06320
## 129   5.00 1e+01 0.2537    0.06720
## 130  10.00 1e+01 0.2475    0.06450
```

The results are not much better than the linear kernel. We find that the best parameters are `cost =` 0.1625 and `gamma=` NA, and the corresponding cross-validated misclassification rate is 0.1625. So in this case, it seems that the linear svm is just as good.

(g) *QUESTION DELETED*

(h) [2 points] *Overall, which approach seems to give the best results on this data*

It appears that the linear kernel with any cost between 0.05 and 5 is as good as any other model. This has the advantage of being simpler.

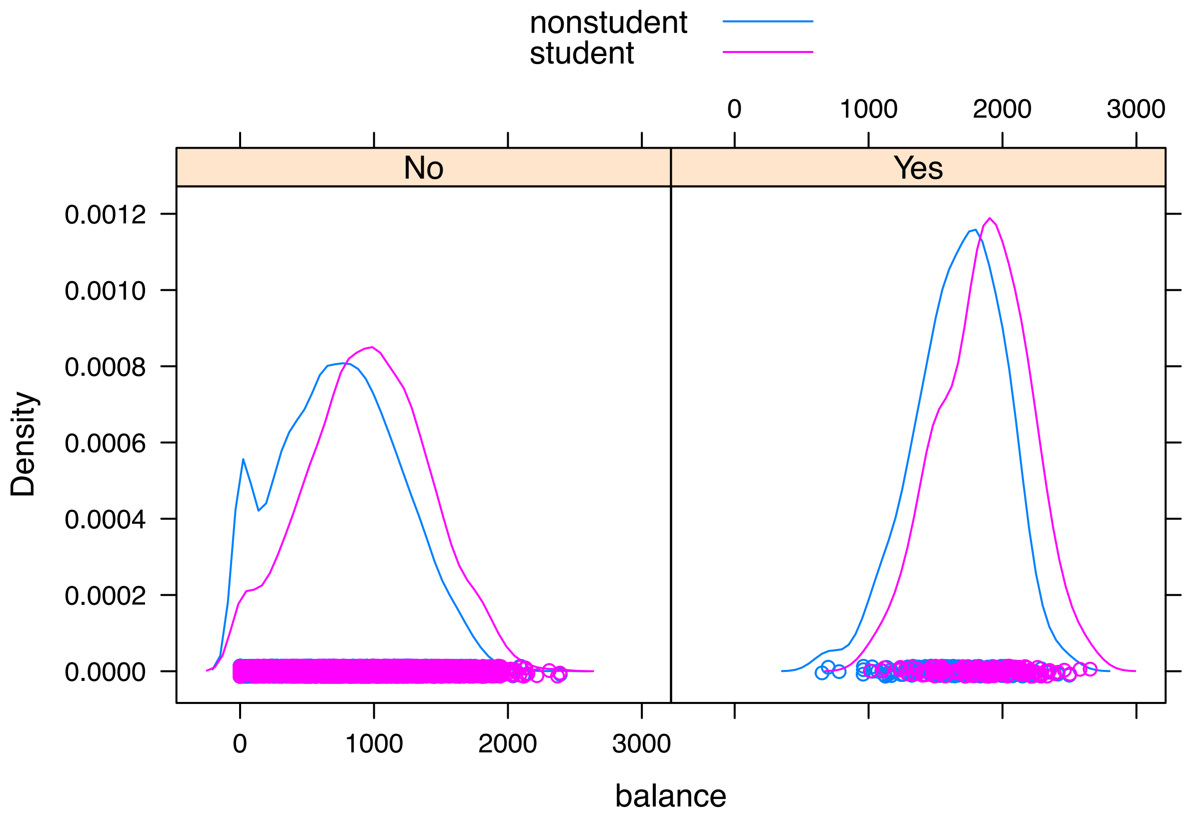
QUESTION 2

[4 points]

Using lattice plots (or any other plot you wish), develop one or more plots of the "Default" data discussed in Ch 4 that conveys the confounding between "balance" and "student" in predicting "default". That is, among people with equal balances, students are less likely to default, but students have higher balances. You don't need to fit any models to generate this graphic. You may find it helpful to adjust the levels of the factor "student" to be "student" and "nonstudent" since currently both student and default are yes/no. For example:

```
levels(Default$student)=c('nonstudent','student')
```

```r
library(ISLR)
levels(Default$student)=c("nonstudent","student")
library(lattice)
densityplot(~balance|default,group=student,data=Default,auto.key=TRUE)
```



There may be other plots, but this one is not bad. The left panel is "non-defaulters" and the right is "defaulters". The densities are the kernel density estimates of `balance`. Within each panel there are separate density estimates for students and nonstudents.

The defaulters (right panel) clearly have much higher balances than nondefaulters (left panel). Within each panel, the students (pink) have higher balances than nonstudents (blue). This suggests that students carry higher balances on their credit cards. Since the biggest effect on defaulting seems to be the balance, it is reasonable to expect that among the students, their larger balances are what is associated with the increased risk of defaulting.