STAT 621 Lecture Notes
Classification Examples

We've already seen several methods for classifying responses. Next we will work through an example or two to compare and contrast these procedures.

Example: Orange Juice Preference

The `OJ` data set in the `ISLR` package contains records of 1070 customer purchases of either Citrus Hill (`CH`) or Minute Maid (`MM`) orange juice. Seventeen other characteristics of the customer and product were also recorded. We'll reduce these a bit for this example and focus on predicting the brand purchased `Purchase` as a function of the follwing features.

`WeekofPurchase`: week of purchase

`StoreID`: store ID

`PriceCH`: price of Citrus Hill

`PriceMM`: price of Minute Maid

`DiscCH`: amount of discount for Citrus Hill

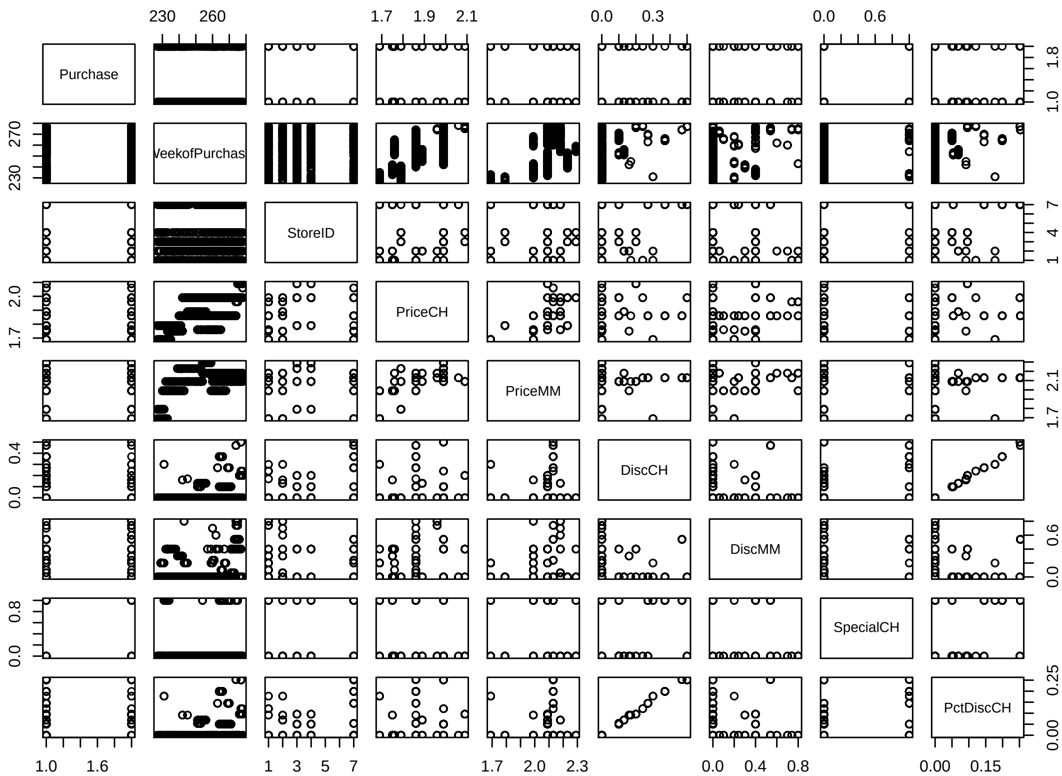
`DiscMM`: amount of discount for Minute Maid

`SpecialCH`: indicator for special on Citrus Hill (1=yes, 0=no)

`PctDiscCH`: percentage of discount for Citrus Hill

Here's a bit of the data and a pairwise scatterplot.

```
# ====================================================
library(MASS)
library(ISLR)
library(e1071)
library(caret)
library(klaR)

oj=OJ[,c(1:8,16)]   # reduce the no. X's for this example
head(oj)
pairs(oj)

 Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH PctDiscCH
1       CH            237       1    1.75    1.99   0.00    0.0         0  0.000000
2       CH            239       1    1.75    1.99   0.00    0.3         0  0.000000
3       CH            245       1    1.86    2.09   0.17    0.0         0  0.091398
4       MM            227       1    1.69    1.69   0.00    0.0         0  0.000000
5       CH            228       7    1.69    1.69   0.00    0.0         0  0.000000
6       CH            230       7    1.69    1.99   0.00    0.0         0  0.000000
```



We will consider predicting `Purchase` using the classification methods we've discussed recently. I'll splint the data into a training set and test set. We will use about 75% of the data for training, and predict the response on the remaining 25%.

```
train = sample(1:nrow(oj), 800)  # 800 = 75% of data for train
oj.train = oj[train, ]
oj.test = oj[-train, ]
```

Logistic Regression: First recall how this works. What is the model, and how do we use it to predict the response?

The logistic model is fit below. I use the `predict.glm` function to report the estimated probability of purchasing Minute Maid, $P(Y = 1)$, on the testing data. An estimated probability of greater than 0.5 is predicted as a Minute Maid purchase, otherwise it is predicted as a Citrus Hill purchase. The `confusionMatrix` function in the `caret` package is handy for summarizing the performance of the classifier.

```
> logistic1=glm(Purchase~., data=oj.train, family=binomial)
> summary(logistic1)

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      4.78071    1.59829   2.991 0.002779 **
WeekofPurchase  -0.02873    0.00825  -3.483 0.000496 ***
StoreID         -0.20337    0.03824  -5.318 1.05e-07 ***
PriceCH          3.53055    1.19794   2.947 0.003207 **
PriceMM         -1.88928    0.75344  -2.508 0.012157 *
DiscCH           9.05908   17.19986   0.527 0.598405
DiscMM           2.21239    0.41658   5.311 1.09e-07 ***
SpecialCH       -0.23796    0.29758  -0.800 0.423902
PctDiscCH      -19.94989   32.57272  -0.612 0.540225
---
    Null deviance: 1076.00  on 799  degrees of freedom
Residual deviance:  960.45  on 791  degrees of freedom
AIC: 978.45


> a=predict.glm(logistic1, oj.test, type="response")
> logistic.pred=factor(as.numeric(a>0.5))
> confusionMatrix(data = logistic.pred, reference = oj.test$Purchase)


Confusion Matrix and Statistics

          Reference
Prediction  CH  MM
        CH 142  60
        MM  30  38

               Accuracy : 0.6667
                 95% CI : (0.607, 0.7226)
    No Information Rate : 0.637
    P-Value [Acc > NIR] : 0.171400

                  Kappa : 0.2284

 Mcnemar's Test P-Value : 0.002237         # test of table symmetry

            Sensitivity : 0.8256          # prob predict CH given CH
            Specificity : 0.3878         #  prob predict MM given MM
         Pos Pred Value : 0.7030
         Neg Pred Value : 0.5588
             Prevalence : 0.6370
         Detection Rate : 0.5259
   Detection Prevalence : 0.7481
      Balanced Accuracy : 0.6067        # ave sens and spec

       'Positive' Class : CH
```

Naive Bayes Model: Let's see how well this one does to predict `Purchase`. First a little review:

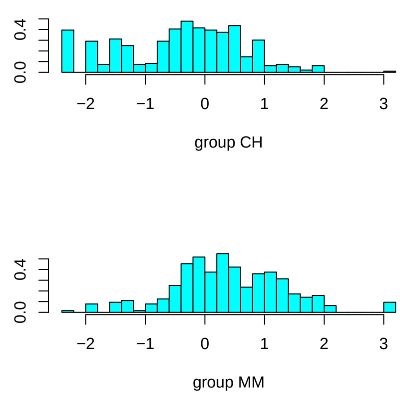
```
> nb1=NaiveBayes(Purchase~., data=oj.train, usekernel=T)
> nb.pred=predict(nb1, oj.test)
> confusionMatrix(data=nb.pred$class, reference=oj.test$Purchase)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  CH  MM
        CH 162  83
        MM  10  15

               Accuracy : 0.6556
                 95% CI : (0.5956, 0.7121)
    No Information Rate : 0.637
    P-Value [Acc > NIR] : 0.2859

                  Kappa : 0.113

 Mcnemar's Test P-Value : 8.264e-14

            Sensitivity : 0.9419
            Specificity : 0.1531
         Pos Pred Value : 0.6612
         Neg Pred Value : 0.6000
             Prevalence : 0.6370
         Detection Rate : 0.6000
   Detection Prevalence : 0.9074
      Balanced Accuracy : 0.5475

       'Positive' Class : CH
```

Linear Discrimimant Analysis: Recall how this works.

Here is the fit and some summaries. Since the response is binary, there's only one discriminant function (Best I can figure, R uses a scaling or normalizing to express the discriminator in terms of $p-1$ functions).

```
> lda1=lda(Purchase~., data=oj.train)
> lda1

Prior probabilities of groups:
     CH      MM
0.60125 0.39875

Group means:
   WeekofPurchase  StoreID  PriceCH  PriceMM     DiscCH     DiscMM  SpecialCH  PctDiscCH
CH       256.2599 4.413721 1.873306 2.100353 0.06688150 0.09692308 0.17671518 0.03523688
MM       251.9530 3.238245 1.864796 2.056301 0.02714734 0.17203762 0.09090909 0.01408230

Coefficients of linear discriminants:
                        LD1
WeekofPurchase  -0.03415863
StoreID         -0.25148057
PriceCH          4.18384055
PriceMM         -2.55493185
DiscCH           9.65383732
DiscMM           2.75079142
SpecialCH       -0.21736277
PctDiscCH      -20.49016136

> plot(lda1)
> lda.pred=predict(object=lda1, newdata=oj.test)
> confusionMatrix(lda.pred$class, oj.test$Purchase)
```



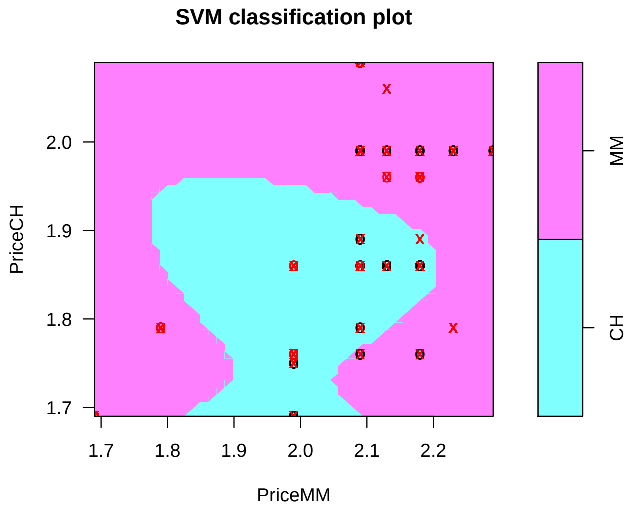

```
Confusion Matrix and Statistics
          Reference
Prediction  CH  MM
        CH 147  63
        MM  25  35

               Accuracy : 0.6741                Sensitivity : 0.8547
                 95% CI : (0.6146, 0.7296)      Specificity : 0.3571
                  Kappa : 0.2311                Balanced Accuracy : 0.6059
 Mcnemar's Test P-Value : 8.006e-05
```

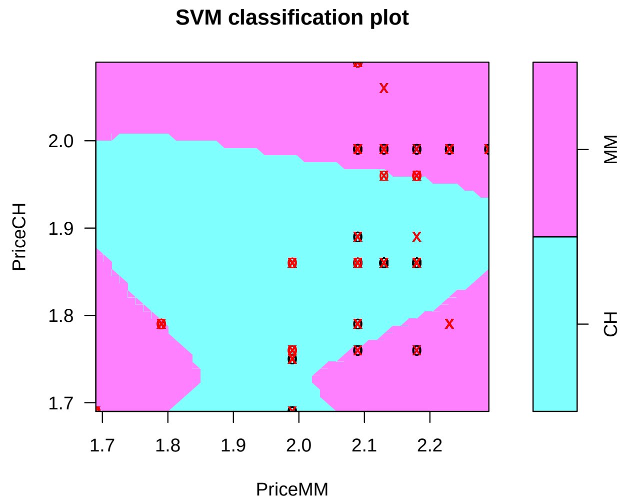
Support Vector Machine: Finally we'll try making predictions using the SVM. This should be pretty fresh so I'll skip the review. Below I fit the SVM using the radial kernel. The parameters for the model, `gamma` and `cost` are estimated by a grid search for the combination with the lowest crossvalidated prediction error. The `tune` function in the `e1070` package makes this easy.

```
> tune.out=tune(svm, Purchase~., data=oj.train, kernel="radial", ranges=list(cost=seq(20,70,10), gamma=seq(.1,.5,.1)))
> summary(tune.out)

Parameter tuning of svm:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   60   0.2

- best performance: 0.29875

- Detailed performance results:
   cost gamma   error dispersion
1    50   0.1 0.31625 0.03537988
2    60   0.1 0.31250 0.03486083
3    70   0.1 0.30750 0.04216370
4    80   0.1 0.30750 0.04377975
5    90   0.1 0.30875 0.04332131
6    50   0.2 0.30000 0.04409586
7    60   0.2 0.29875 0.04348132    <-----
8    70   0.2 0.30125 0.04505013
9    80   0.2 0.30000 0.04370037
10   90   0.2 0.29875 0.04466309
11   50   0.3 0.30250 0.04993051
... ETC

> svm1=svm(Purchase~., data=oj.train, kernel="radial", gamma=.2, cost=50)
> summary(svm1)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  50

Number of Support Vectors:  526
 ( 259 267 )

Number of Classes:  2

Levels:
 CH MM
```

With $p > 2$, the `plot` function will show projections of the decision boundary projected onto the plane spanned by any two features. This can be done for specific values of the other features, basically showing a slice of the projection.

```
> plot(svm1,oj.train, PriceCH~PriceMM, slice=list(WeekofPurchase=250, StoreID=3))
> plot(svm1,oj.train, PriceCH~PriceMM, slice=list(WeekofPurchase=250, StoreID=5))
```





Now let's see how it does at classifying OJ purchases.

```
> svm.pred=predict(svm1,oj.test)
> confusionMatrix(svm.pred, oj.test$Purchase)

Confusion Matrix and Statistics

          Reference
Prediction  CH  MM
        CH 138  41
        MM  34  57

               Accuracy : 0.7222
                 95% CI : (0.6647, 0.7748)
                  Kappa : 0.3899
 Mcnemar's Test P-Value : 0.488422

            Sensitivity : 0.8023
            Specificity : 0.5816
                    ...
      Balanced Accuracy : 0.6920
```

Summarize the results.

All the methods were fairly similar in terms of prediction accuracy. But remember these results were based on a single split of the data into a training set and a test set. Maybe a better way to select a procedure would be to repeat this and choose the procedure that has the lowest error rate. In other words, let's crossvalidate!

```
K=5
cv.error = matrix(0,nrow=4,ncol=K)
set.seed(135)
folds = sample(1:K,nrow(oj),replace=T)  # will give more or less same n each fold

for(i in 1:K)
  {
  CV.train = oj[folds != i,]
  CV.test = oj[folds == i,]
  # logistic
  logistic.fit = glm(Purchase ~., data = CV.train, family = binomial)
  logistic.probs = predict(logistic.fit, CV.test, type = "response")
  logistic.pred = factor(as.numeric(logistic.probs>0.5))
  cv.error[1,i]=mean(logistic.pred != as.numeric(CV.test$Purchase)-1)
  # naive bayes
  nb.fit=NaiveBayes(Purchase~., data=CV.train, usekernel=T)
  nb.pred=predict(nb.fit, CV.test)
  cv.error[2,i]=mean(nb.pred$class != CV.test$Purchase)
  # LDA
  lda.fit=lda(Purchase~., data=CV.train)
  lda.pred=predict(lda.fit, CV.test)
  cv.error[3,i]=mean(lda.pred$class != CV.test$Purchase)
  # SVM -- use the parameter values found earlier
  svm.fit=svm(Purchase~., data=CV.train, kernel="radial", gamma=.2, cost=60)
  svm.pred=predict(svm.fit,CV.test)
  cv.error[4,i]=mean(svm.pred != CV.test$Purchase)
}
```

```
> row.names(cv.error)=c("logistic","NBayes","LDA","SVM")
> apply(cv.error, 1, mean)

 logistic    NBayes       LDA       SVM
0.3532973 0.3757455 0.3479311 0.3003532
```

A pretty clear choice.