Stat 121B - Homework 4

Keyan Halperin

March 2, 2017

Problem 1

1.

```
library(e1071)
library(caret)

svm.rbf1 = svm(x = train1[ , 1:61], y = train1$Class, kernel = 'radial', gamma = 1, cost = 1)
```

2.

```
#Predict function
pred = function(model, data, y) {

  x = data[, names(data) != y]
  y = data[[y]]
  pred.y = predict(model, newdata = x)
  return(pred.y)

}

#Predictions on Test Data
test.pred.y = pred(svm.rbf1, test1, 'Class')
confusionMatrix(test.pred.y, test1$Class) #Accuracy = .723
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1   0   0   0   0
##          2   0   0   0   0
##          3  46  73 499  72
##          4   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.7232
##                  95% CI : (0.6882, 0.7563)
##     No Information Rate : 0.7232
##     P-Value [Acc > NIR] : 0.5195
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity           0.00000   0.0000   1.0000   0.0000
## Specificity           1.00000   1.0000   0.0000   1.0000
## Pos Pred Value            NaN      NaN   0.7232      NaN
## Neg Pred Value        0.93333   0.8942      NaN   0.8957
## Prevalence            0.06667   0.1058   0.7232   0.1043
## Detection Rate        0.00000   0.0000   0.7232   0.0000
## Detection Prevalence  0.00000   0.0000   1.0000   0.0000
## Balanced Accuracy     0.50000   0.5000   0.5000   0.5000
```

```
#Predictions on Train Data
train.pred.y = pred(svm.rbf1, train1, 'Class')
confusionMatrix(train.pred.y, train1$Class) #Accuracy = 1
```

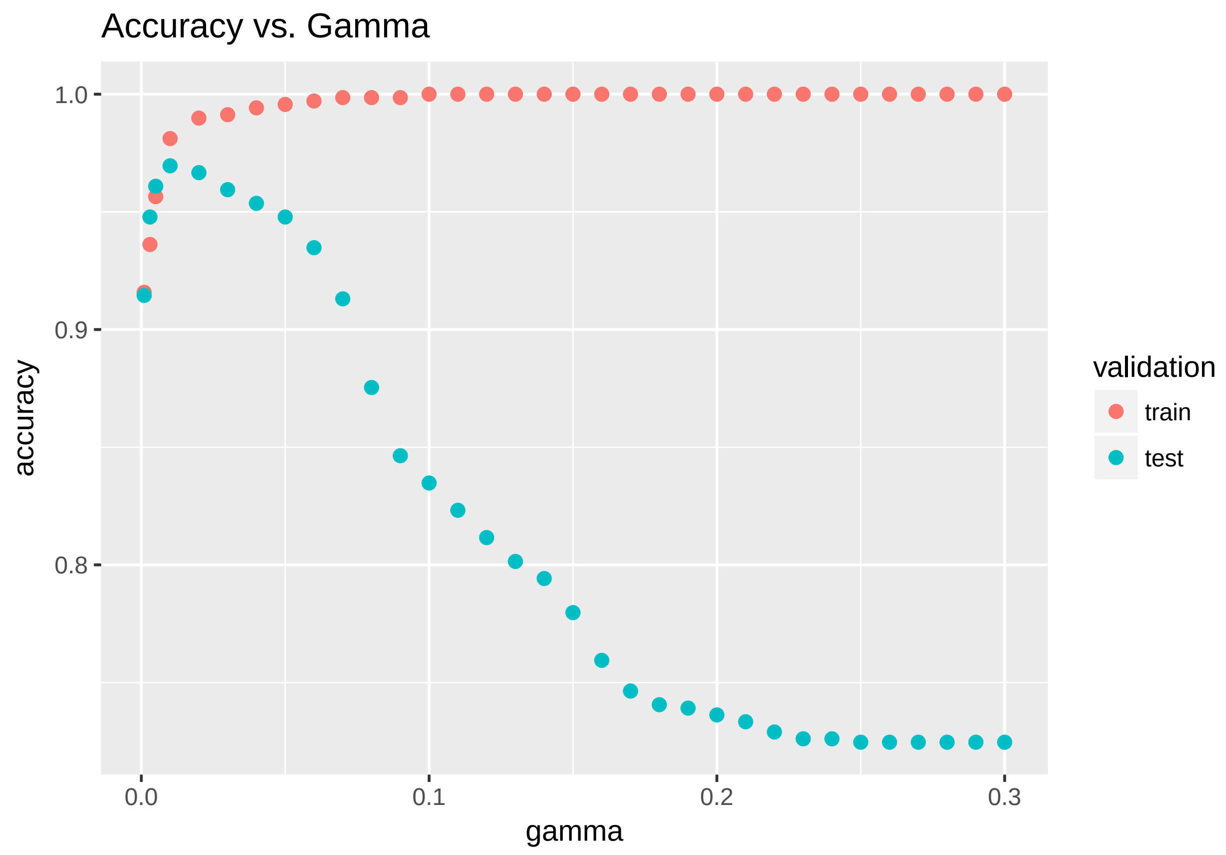
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1  44   0   0   0
##          2   0  72   0   0
##          3   0   0 492   0
##          4   0   0   0  81
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9947, 1)
##     No Information Rate : 0.7141
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity           1.00000   1.0000   1.0000   1.0000
## Specificity           1.00000   1.0000   1.0000   1.0000
## Pos Pred Value        1.00000   1.0000   1.0000   1.0000
## Neg Pred Value        1.00000   1.0000   1.0000   1.0000
## Prevalence            0.06386   0.1045   0.7141   0.1176
## Detection Rate        0.06386   0.1045   0.7141   0.1176
## Detection Prevalence  0.06386   0.1045   0.7141   0.1176
## Balanced Accuracy     1.00000   1.0000   1.0000   1.0000
```

The model achieves an accuracy of .723 on the test data, which seems pretty good, but as we can see from the confusion matrix, the model is simply classifying all observations in category 3, the largest category.

On the other hand, the accuracy on the train data is 1 since it correctly predicts all points. This seems to be a case of overfitting.

3.

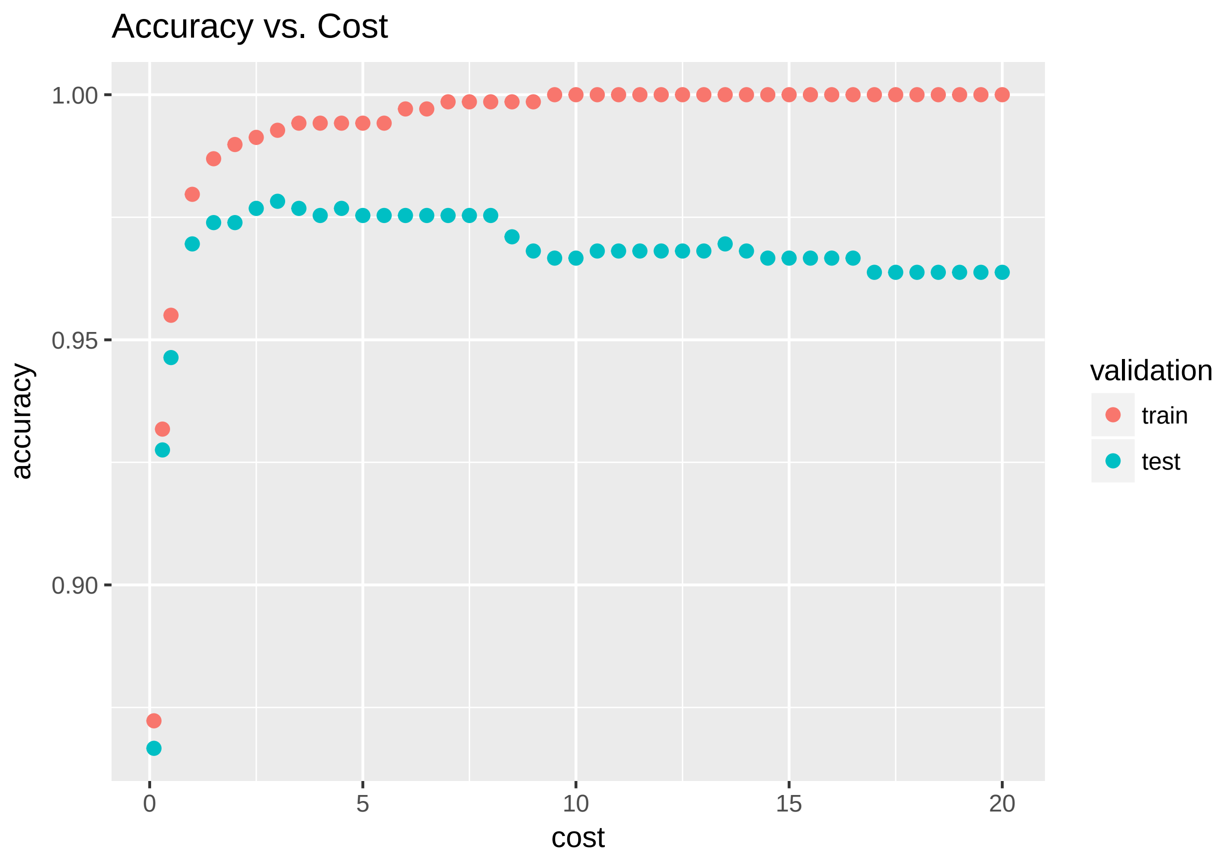
```
validation = tune.control(sampling = 'fix')

grid = list(gamma = c(.001, .003, .005, seq(.01, .3, by = .01)), cost = 1)

svm.train1 = tune(svm, train.x = train1[ , 1:61], train.y = train1$Class, kernel = 'radial',
                  ranges = grid, tunecontrol = validation, validation.x = train1[, 1:61], validation.y =

svm.test1 = tune(svm, train.x = train1[ , 1:61], train.y = train1$Class, kernel = 'radial',
                 ranges = grid, tunecontrol = validation, validation.x = test1[, 1:61], validation.y =

train.results1 = svm.train1$performances
train.table1 = data.frame(gamma = train.results1$gamma, accuracy = 1 - train.results1$error, validation

test.results1 = svm.test1$performances
test.table1 = data.frame(gamma = test.results1$gamma, accuracy = 1 - test.results1$error, validation =

data = rbind(train.table1, test.table1)

library(ggplot2)
ggplot(data, aes(x = gamma, y = accuracy)) + geom_point(aes(color = validation), size = 2) + ggtitle('A
```



Holding the cost parameter constant at 1, increasing gamma initially increases both the test and train accuracies, but the test accuracy quickly decreases after reaching .01, while the train accuracy continues to increase. This is because the value of gamma is an example of the bias-variance trade-off. As we increase gamma, bias decreases, variance increases, and if we go too far, we will overfit on the training data.

4.

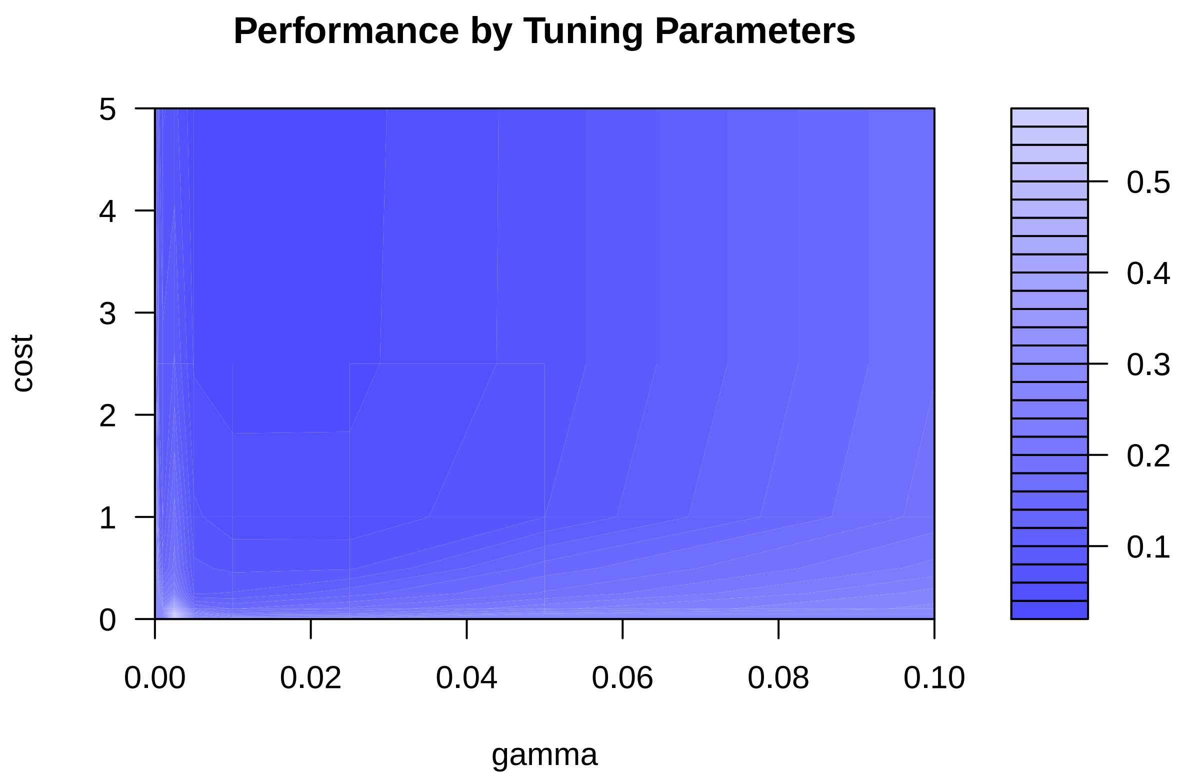
```
grid = list(gamma = .01, cost = c(0.1, 0.3, seq(0.5, 20, by = .5)))

svm.train2 = tune(svm, train.x = train1[ , 1:61], train.y = train1$Class, kernel = 'radial',
                  ranges = grid, tunecontrol = validation, validation.x = train1[, 1:61], validation.y =

svm.test2 = tune(svm, train.x = train1[ , 1:61], train.y = train1$Class, kernel = 'radial',
                 ranges = grid, tunecontrol = validation, validation.x = test1[, 1:61], validation.y = 

train.results2 = svm.train2$performances
train.table2 = data.frame(cost = train.results2$cost, accuracy = 1 - train.results2$error, validation =

test.results2 = svm.test2$performances
test.table2 = data.frame(cost = test.results2$cost, accuracy = 1 - test.results2$error, validation = 'te

data2 = rbind(train.table2, test.table2)
ggplot(data2, aes(x = cost, y = accuracy)) + geom_point(aes(color = validation), size = 2) + ggtitle('Ac
```



Holding gamma constant at .01, increasing cost initially increases both the test and train accuracies, but after about a cost of 3, the test accuracy starts to steadily decline. The value of cost tunes how much slack we are permitting the seperating hyperplane to have. Likewise, as we increase cost, bias decreases, variance increases, and if we go to far, we will overfit on the training data.

5/6.

```
options(scipen = 5)
cv = tune.control(sampling = 'cross', cross = 5) #5-fold cross validation

grid = list(gamma = c(.0001, .0025, .0005, .001, .0025, .005, .01, .025, .05, .1, .25, .5, 1, 10),
            cost = c(.0001, .001, .0025, .005, .01, .05, .1, .25, .5, 1, 2.5, 5, 10, 25, 50, 100))


svm.linear = tune(svm, train.x = train1[ , 1:61], train.y = train1$Class, kernel = 'linear',
                  ranges = grid, tunecontrol = cv)

svm.poly = tune(svm, train.x = train1[ , 1:61], train.y = train1$Class, kernel = 'poly', degree = 2,
                ranges = grid, tunecontrol = cv)

svm.radial = tune(svm, train.x = train1[ , 1:61], train.y = train1$Class, kernel = 'radial',
                  ranges = grid, tunecontrol = cv)


plot(svm.radial, xlim = c(0,.1), ylim = c(0, 5), main = 'Performance by Tuning Parameters')
```



```
#Cross-validated linear svm accuracy
1 - svm.linear$best.performance
```

```
## [1] 0.9738496
```

```
#Cross-validated poly svm accuracy
1 - svm.poly$best.performance
```

```
## [1] 0.9434254
```

```
#Cross-validated radial svm accuracy
1 - svm.radial$best.performance
```

```
## [1] 0.9738707
```

```
#Linear svm test accuracy
linear.test.pred.y = pred(svm.linear$best.model, test1, 'Class')
confusionMatrix(linear.test.pred.y, test1$Class)$overall[1]
```

```
##  Accuracy
## 0.9855072
```

```
confusionMatrix(linear.test.pred.y, test1$Class)$table
```

```
##           Reference
## Prediction   1   2   3   4
##          1  44   1   1   0
##          2   1  71   1   0
##          3   0   1 495   2
##          4   1   0   2  70
```

```
#Poly svm test accuracy
poly.test.pred.y = pred(svm.poly$best.model, test1, 'Class')
confusionMatrix(poly.test.pred.y, test1$Class)$overall[1]
```

```
##  Accuracy
## 0.9521739
```

```
confusionMatrix(poly.test.pred.y, test1$Class)$table
```

```
##           Reference
## Prediction   1   2   3   4
##          1  43   1   3   0
##          2   1  55   2   0
##          3   1  16 492   5
##          4   1   1   2  67
```

```
#Radial svm test accuracy
radial.test.pred.y = pred(svm.radial$best.model, test1, 'Class')
confusionMatrix(radial.test.pred.y, test1$Class)$overall[1]
```

```
##  Accuracy
## 0.9768116
```

```
confusionMatrix(radial.test.pred.y, test1$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1  43   0   1   0
##          2   0  69   3   0
##          3   2   4 494   4
##          4   1   0   1  68
##
## Overall Statistics
##
##                Accuracy : 0.9768
##                  95% CI : (0.9626, 0.9867)
##     No Information Rate : 0.7232
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.948
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity           0.93478   0.9452   0.9900  0.94444
## Specificity           0.99845   0.9951   0.9476  0.99676
## Pos Pred Value        0.97727   0.9583   0.9802  0.97143
## Neg Pred Value        0.99536   0.9935   0.9731  0.99355
## Prevalence            0.06667   0.1058   0.7232  0.10435
## Detection Rate        0.06232   0.1000   0.7159  0.09855
## Detection Prevalence  0.06377   0.1043   0.7304  0.10145
## Balanced Accuracy     0.96661   0.9702   0.9688  0.97060
```

Based on both the cross-validated accuracy scores and accuracy on the test data, all three models did very well. The linear svm and the radial one perfmored a little bit better than the polynomial svm, but there does not appear to be a significant difference between the linear and the radial svm. And as we can see from the confusion matrices, these models do an excellent job an classifying all 4 classes. Consequently, they perform substantially better than the naive classifier which predicts the most common class on all points.

```
## 'data.frame':    1934 obs. of  5 variables:
##  $ district         : int  101 101 101 101 101 101 101 101 101 101 ...
##  $ urban            : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ living.children  : int  4 1 4 2 4 4 4 1 3 2 ...
##  $ age_mean         : num  18.44 -5.56 8.44 -5.56 1.44 ...
##  $ contraceptive_use: int  0 0 0 0 0 0 1 0 1 1 ...
```

Problem 2

1.a.

```
logit1 = glm(contraceptive_use ~ living.children, family = binomial, data = data2)
summary(logit1)
```

```
##
## Call:
## glm(formula = contraceptive_use ~ living.children, family = binomial,
##     data = data2)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1109  -1.0245  -0.8631   1.2454   1.5285
##
## Coefficients:
##                 Estimate Std. Error z value     Pr(>|z|)
## (Intercept)     -1.00804    0.11413  -8.832      < 2e-16 ***
## living.children  0.21240    0.03816   5.565 0.0000000261 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2590.9  on 1933  degrees of freedom
## Residual deviance: 2559.4  on 1932  degrees of freedom
## AIC: 2563.4
##
## Number of Fisher Scoring iterations: 4
```

The intercept is the estimated log odds of using contraception for individuals with 0 living children, and the slope is the estimated increase in the log odds of using contraception for a one unit increase in the number of living children.

1.b.

```
logit2 = glm(contraceptive_use ~ -1 + living.children*district, family = binomial, data = data2)
summary(logit2)
```

```
##
## Call:
## glm(formula = contraceptive_use ~ -1 + living.children * district,
##     family = binomial, data = data2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7816  -0.9576  -0.6271   1.1045   2.2425
##
## Coefficients:
##                                Estimate  Std. Error z value Pr(>|z|)
## living.children                0.106367    0.180623   0.589 0.555935
## district101                   -1.375963    0.575780  -2.390 0.016860 *
## district102                   -2.061808    1.543107  -1.336 0.181504
## district103                   16.566068 3297.858778   0.005 0.995992
## district104                   -0.834315    1.090218  -0.765 0.444108
## district105                   -0.858955    0.808286  -1.063 0.287924
## district106                   -1.343021    0.699830  -1.919 0.054976 .
## district107                   -0.611737    1.164291  -0.525 0.599294
## district108                   -1.924097    0.907333  -2.121 0.033955 *
## district109                   -1.679155    1.543587  -1.088 0.276672
## district110                   -1.890979    2.049485  -0.923 0.356184
## district111                  -16.566068 1041.868363  -0.016 0.987314
## district112                    0.425899    1.105585   0.385 0.700071
## district113                   -1.919525    1.060581  -1.810 0.070314 .
## district114                    -0.071321    0.418992  -0.170 0.864837
## district115                    -2.022263    1.129848  -1.790 0.073477 .
## district116                    -0.704104    0.958943  -0.734 0.462796
## district117                    -5.155386    2.995887  -1.721 0.085283 .
## district118                    -0.928408    0.689907  -1.346 0.178399
## district119                    -1.661649    0.977120  -1.701 0.089026 .
## district120                    -0.405465    1.083165  -0.374 0.708156
## district121                     2.092436    1.274546   1.642 0.100650
## district122                    -0.262136    1.441725  -0.182 0.855723
## district123                    -3.545389    2.550484  -1.390 0.164503
## district124                   -27.908063 1432.305750  -0.019 0.984454
## district125                    -2.660953    0.765282  -3.477 0.000507 ***
## district126                    -1.284610    1.880886  -0.683 0.494618
## district127                    -1.524485    0.861839  -1.769 0.076915 .
## district128                    -2.791832    1.121659  -2.489 0.012810 *
## district129                    -2.408745    0.964867  -2.496 0.012544 *
## district130                    -2.150394    0.676772  -3.177 0.001486 **
## district131                    -0.713908    0.837167  -0.853 0.393789
## district132                    -1.982496    1.675511  -1.183 0.236723
## district133                    -5.714356    3.168314  -1.804 0.071295 .
## district134                     0.451211    1.005781   0.449 0.653708
## district135                    -0.545106    0.774690  -0.704 0.481655
## district136                    -0.645565    1.468301  -0.440 0.660177
## district137                     0.928963    1.664548   0.558 0.576785
## district138                   -28.960778 1866.415651  -0.016 0.987620
## district139                     0.758751    0.859748   0.883 0.377492
## district140                    -0.073860    0.689158  -0.107 0.914651
## district141                    -0.175395    1.067936  -0.164 0.869544
## district142                   -30.819019 1554.012801  -0.020 0.984177
## district143                    -0.844169    0.671885  -1.256 0.208964
## district144                    -2.261523    1.262530  -1.791 0.073251 .
## district145                    -1.942267    1.002943  -1.937 0.052798 .
## district146                    -0.349494    0.536976  -0.651 0.515140
## district147                    -2.248961    1.468026  -1.532 0.125532
## district148                    -0.528244    0.716095  -0.738 0.460714
## district149                   -16.566068 2012.907261  -0.008 0.993434
## district150                    -2.080824    1.181639  -1.761 0.078245 .
## district151                     0.121076    0.838702   0.144 0.885215
## district152                    -1.134439    0.618816  -1.833 0.066767 .
## district153                    -2.304685    1.304999  -1.766 0.077388 .
## district154                    42.251276 1445.847865   0.029 0.976687
## district155                     0.289113    0.647251   0.447 0.655107
## district156                    -3.106022    1.589848  -1.954 0.050741 .
## district157                    -0.617017    0.790570  -0.780 0.435114
## district158                    -2.687093    2.997012  -0.897 0.369937
## district159                    -2.744313    1.123020  -2.444 0.014538 *
## district160                    -0.077121    0.875173  -0.088 0.929781
## living.children:district102     0.392093    0.521937   0.751 0.452516
## living.children:district103    -0.106367 1131.156243   0.000 0.999925
## living.children:district104     0.168183    0.381775   0.441 0.659553
## living.children:district105    -0.008259    0.313587  -0.026 0.978988
## living.children:district106     0.058983    0.290729   0.203 0.839229
## living.children:district107    -0.243473    0.459042  -0.530 0.595840
## living.children:district108     0.435421    0.355267   1.226 0.220344
## living.children:district109    0.164346    0.493193   0.333 0.738961
## living.children:district110   -0.375871    0.887630  -0.423 0.671964
## living.children:district111   -0.106367  497.769879   0.000 0.999830
## living.children:district112   -0.445417    0.378086  -1.178 0.238764
## living.children:district113    0.475181    0.385352   1.233 0.217534
## living.children:district114    0.138985    0.239752   0.580 0.562115
## living.children:district115    0.486273    0.439985   1.105 0.269072
## living.children:district116    0.367154    0.486101   0.755 0.450067
## living.children:district117    1.136997    0.802939   1.416 0.156763
## living.children:district118   -0.001182    0.300739  -0.004 0.996863
## living.children:district119    0.347324    0.371919   0.934 0.350370
## living.children:district120   -0.106367    0.443958  -0.240 0.810650
## living.children:district121   -1.273285    0.604434  -2.107 0.035154 *
## living.children:district122   -0.514309    0.537937  -0.956 0.339033
## living.children:district123    0.693659    0.754397   0.919 0.357841
## living.children:district124    6.422709  358.076787   0.018 0.985689
## living.children:district125    0.772056    0.305459   2.528 0.011487 *
## living.children:district126    0.176831    0.641902   0.275 0.782947
## living.children:district127   -0.098572    0.344109  -0.286 0.774529
## living.children:district128    0.432921    0.371056   1.167 0.243321
## living.children:district129    0.481495    0.367588   1.310 0.190237
## living.children:district130    0.770817    0.314132   2.454 0.014136 *
## living.children:district131    0.085246    0.326443   0.261 0.793987
## living.children:district132    0.090782    0.508785   0.178 0.858387
## living.children:district133    1.562458    0.937602   1.666 0.095626 .
## living.children:district134   -0.038980    0.366794  -0.106 0.915366
## living.children:district135    0.088781    0.313969   0.283 0.777354
## living.children:district136   -0.092413    0.519602  -0.178 0.858838
## living.children:district137   -0.363183    0.546375  -0.665 0.506233
## living.children:district138    7.032461  466.604031   0.015 0.987975
## living.children:district139   -0.400564    0.345538  -1.159 0.246355
## living.children:district140   -0.133748    0.293400  -0.456 0.648495
## living.children:district141   -0.043900    0.397146  -0.111 0.911982
## living.children:district142   15.303143  777.006457   0.020 0.984287
## living.children:district143    0.321570    0.320762   1.003 0.316094
## living.children:district144    0.261137    0.443168   0.589 0.555693
## living.children:district145    0.360449    0.385630   0.935 0.349941
## living.children:district146    0.055854    0.255174   0.219 0.826738
## living.children:district147    0.665635    0.507882   1.311 0.189990
## living.children:district148    0.156591    0.327321   0.478 0.632365
## living.children:district149   -0.106367  923.585211   0.000 0.999908
## living.children:district150    0.625074    0.419316   1.491 0.136040
## living.children:district151   -0.208442    0.331495  -0.629 0.529483
## living.children:district152    0.238467    0.277801   0.858 0.390667
## living.children:district153    0.651119    0.476573   1.366 0.171859
## living.children:district154  -28.328351  850.753193  -0.033 0.973437
## living.children:district155   -0.096039    0.301219  -0.319 0.749852
## living.children:district156    0.430341    0.491255   0.876 0.381028
## living.children:district157    0.069875    0.338025   0.207 0.836232
## living.children:district158    0.063100    0.958576   0.066 0.947515
## living.children:district159    0.467274    0.408698   1.143 0.252905
## living.children:district160   -0.585386    0.377404  -1.551 0.120881
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2681.1  on 1934  degrees of freedom
## Residual deviance: 2285.1  on 1814  degrees of freedom
## AIC: 2525.1
##
## Number of Fisher Scoring iterations: 15
```

This model formula is accomplishing the task of fitting separate models per district since living.children*district incorporates the interactions between district and slope. As a result, for each district, there will be a different coefficient for living.children. And the -1 forces there to be a seperate intercept for each district without having a common intercept for the reference level.

1.c.

```r
library(MCMCpack)
```

```
## Warning: package 'MCMCpack' was built under R version 3.3.2

## Loading required package: coda

## Warning: package 'coda' was built under R version 3.3.2

## Loading required package: MASS

## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)

## ## Copyright (C) 2003-2017 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park

## ##
## ## Support provided by the U.S. National Science Foundation

## ## (Grants SES-0350646 and SES-0350613)
## ##
```

```r
mcmc.fit = MCMChlogit(contraceptive_use ~ -1 + living.children,
  random = ~ living.children,
  group = 'district',
  data = data2,
  burnin = 500,
  mcmc = 5000,
  thin = 1,
  verbose = 1,
  beta.start = NA,
  sigma2.start = NA,
  Vb.start = NA,
  mubeta = 0,
  Vbeta = 10000,
  r = 2,
  R = diag(c(1, .1)),
  nu = 0.001,
  delta = 0.001)
```

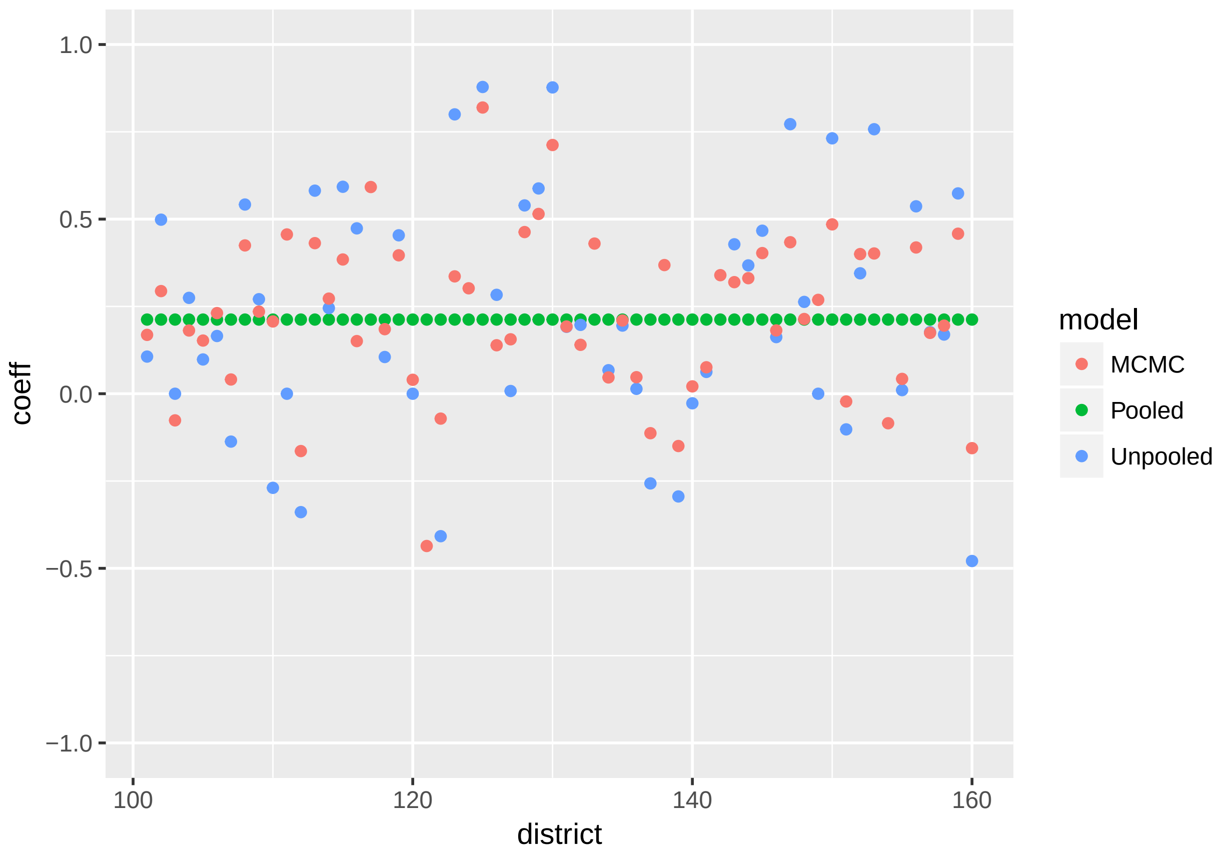
```
##
## Running the Gibbs sampler. It may be long, keep cool :)
```

```
##
## **********:10.0%, mean accept. rate=0.438
## **********:20.0%, mean accept. rate=0.559
## **********:30.0%, mean accept. rate=0.620
## **********:40.0%, mean accept. rate=0.671
## **********:50.0%, mean accept. rate=0.638
## **********:60.0%, mean accept. rate=0.507
## **********:70.0%, mean accept. rate=0.633
## **********:80.0%, mean accept. rate=0.723
## **********:90.0%, mean accept. rate=0.768
## **********:100.0%, mean accept. rate=0.749
```

What makes this model different from the previous two models is that it simultaneously incorporates both the information about each district (like the unpooled model) and the information about the entire dataset (like the pooled model). Consequently, it serves as somewhat of a compromise between the two.

2.a.

```r
options(scipen = 5)
n.districts = length(levels(data2$district))
logit1.coeff = rep(logit1$coefficients[[2]], n.districts)

logit2.coeff = logit2$coefficients[c(1, 62:length(logit2$coefficients))]

for (i in 2:length(logit2.coeff)){

  logit2.coeff[i] = logit2.coeff[1] + logit2.coeff[i]

}
logit2.coeff = unname(logit2.coeff)


mcmc.coeff = summary(mcmc.fit$mcmc)$statistics[,1]
mcmc.coeff = mcmc.coeff[62:121]


mcmc.coeff = unname(mcmc.coeff)

districts = as.numeric(as.character(levels(data2$district)))
coeff.data = data.frame(district = districts, coeff = c(logit1.coeff, logit2.coeff, mcmc.coeff),
                        model = rep(c('Pooled', 'Unpooled', 'MCMC'), each = n.districts))

ggplot(coeff.data, aes(x = district, y = coeff)) + geom_point(aes(color = model)) + ylim(-1, 1)
```



Some of the coefficients are noticeably larger in magnitude than is displayed on the graph. However, I have restricted the vertical axis from -1 to 1 as to not obscure the general pattern of the bulk of points.

2.b.

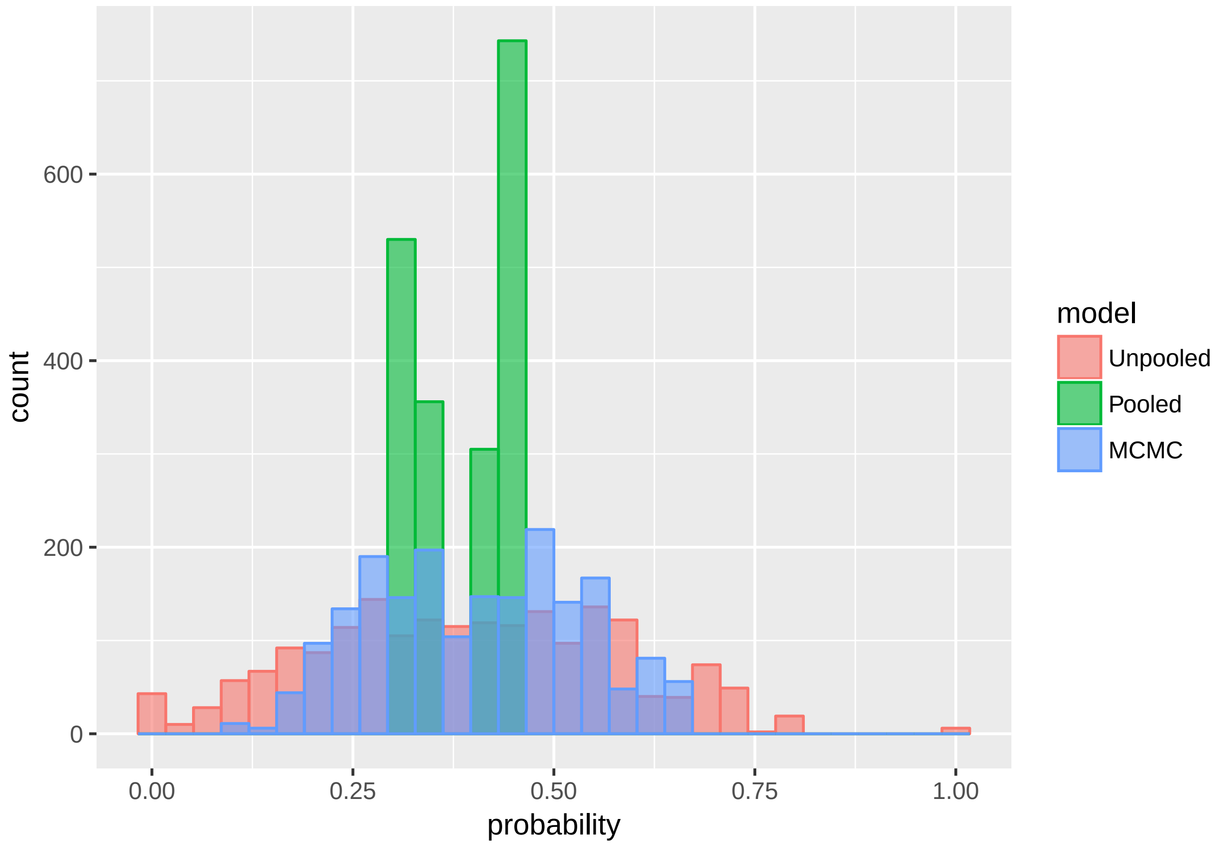
As we can see in the graph above, the pooled coefficients are the same for each district. This is because we did not include district in our model and so we only had one coefficient for living.children. The unpooled model on the other hand fit a completely separate model for each district, and so it gave us a different coefficient for each district.

The problem with the pooled model is that it completely ignores all information about the districts, and the problem with the unpooled model is that it ignores all of the information about all of the other districts. However, The Bayesian Hierarchical Logistic Model incorporates both the district data and the non-district data simultaneously. As a result it serves as a compromise between the pooled and unpooled methods. This can be seen on the graph above by the fact that Bayesian coefficients are less extreme than the unpooled coefficients. Essentially, the coefficients from the unpooled model are being pulled closer to the pooled model coefficients, and how much they are pulled depends on the number of observations within each district. The fewer observations a district has, the more it will be pulled towards the pooled estimate. This is because the fewer observations there are, the weaker the signal is, and thus the non-district data will be more influential on those coefficients.

3.a.

```r
pred = function(model, data, y) {

  x = data[, names(data) != y]
  y = data[[y]]
  pred.y = predict(model, newdata = x, type = 'response')
  return(pred.y)

}

logit1.pred = pred(logit1, data2, 'contraceptive_use')
logit2.pred = pred(logit2, data2, 'contraceptive_use')
mcmc.pred = mcmc.fit$theta.pred

n = nrow(data2)
pred.data = data.frame(probability = c(logit1.pred, logit2.pred, mcmc.pred),
                       model = rep(c('Pooled', 'Unpooled', 'MCMC'), each = n))

pred.data$model = factor(pred.data$model, rev(levels(pred.data$model)))

ggplot(pred.data, aes(x = probability, fill = model)) +
  geom_histogram(aes(color = model), alpha = .6, position = 'identity')
```

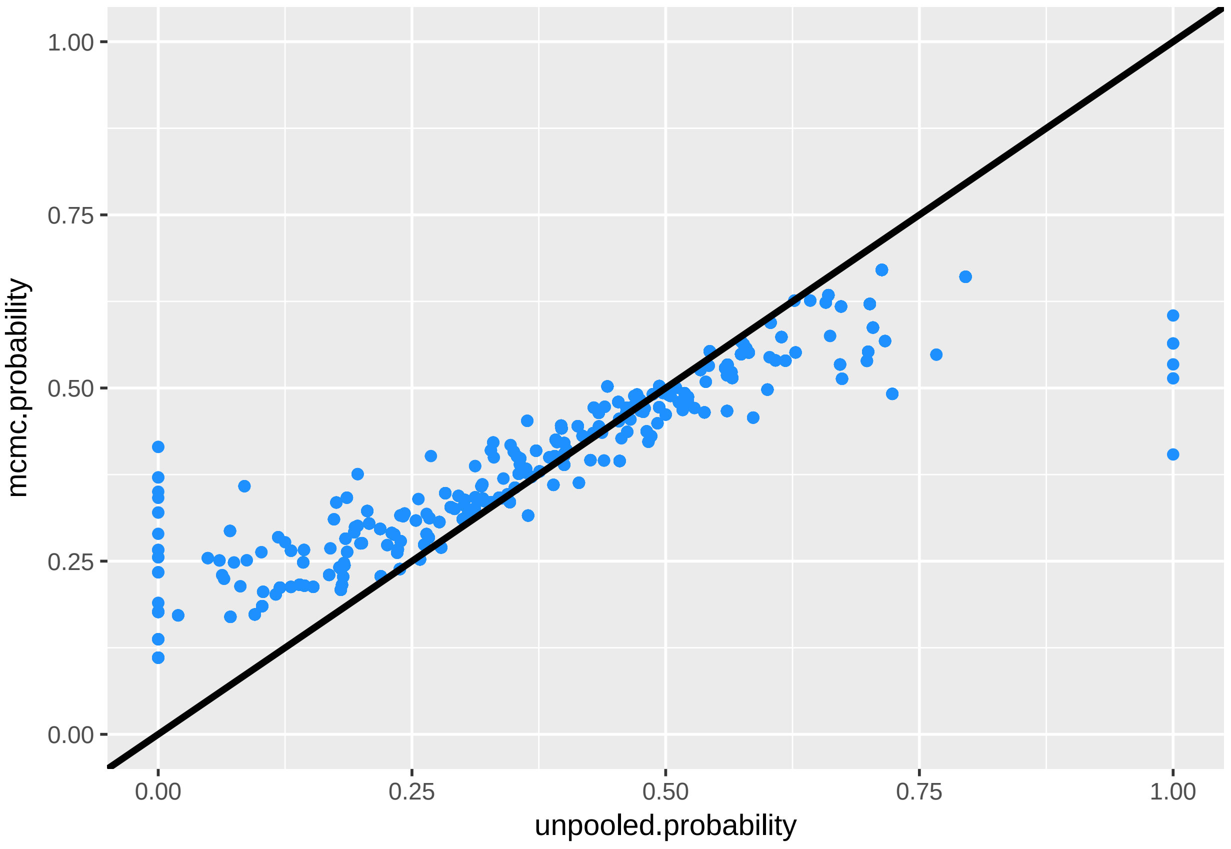
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Unsurprisingly, the unpooled probabilities vary the most, the pooled probabilities vary the least, and the bayesian probabilities are somewhat of a compromise between the two for analagous reasons discussed in the previous question.

3.b.

```r
plot.data = data.frame(unpooled.probability = logit2.pred, mcmc.probability = mcmc.pred)
ggplot(plot.data, aes(unpooled.probability, mcmc.probability)) + geom_point(col = 'dodgerblue') +
       geom_abline(slope = 1, intercept = 0, size = 1.2) + ylim(0, 1)
```



As we can see from the plot above, the bayesian model is more moderate in its predictions that the unpooled model. The bayesian model has lower probabilities for observations with high unpooled probabilities, and it has higher probabilities for observations with low unpooled probabilities. This can most clearly be seen at the extremes. For the observations with an unpooled probability of 1, the MCMC probabilities are around .5, and for the observations with an unpooled probability of 0, the MCMC probabilities are around .25. This further illustrates the benefits of using a bayesian logistic regression model as discussed previously.

Problem 3.

keyan_halperin@g.harvard.edu

Account ID: 692745491595