Homework 4 - SVMs & Return of the Bayes

Harvard CS109B, Spring 2017

Danqing Wang

Mar 8 2017

Problem 1: Celestial Object Classification

SVMs are computationally intensive, much more so than other methods we've used in the course. Expect run times for your analyses to be much larger than before. Several SVM packages are available, we recommend using the `e1071` library, though you're free to use whatever package you feel comfortable with – we'll provide extra hints for the `svm` function from this package.

In this problem, the task is to classify a celestial object into one of 4 categories using photometric measurements recorded about the object. The training and testing datasets are provided in the `dataset_1_train.txt` and `dataset_1_test.txt` respectively. Overall, there are a total of 1,379 celestial objects described by 61 attributes. The last column contains the object category we wish to predict, `Class`.

We'll be working with Support Vector Machines, trying out different kernels, tuning, and other fun things. *Hint*: Use the `kernel`, `degree`, `cost`, `gamma` arguments of the `svm` function appropriately.

First, ensure that the that `Class` is a factor (quantitative values). These should be object categories and not integer values – use `as.factor` if needed.

Answer

```
train <- read.csv('./datasets/dataset_1_train.txt')
test <- read.csv('./datasets/dataset_1_test.txt')

str(train)
```

```
## 'data.frame':    689 obs. of  62 variables:
##  $ Amplitude                    : num  0.649 0.648 0.131 0.954 0.608 ...
##  $ AndersonDarling              : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Autocor_length               : int  1 1 1 1 1 16 4 1 1 1 ...
##  $ Beyond1Std                   : num  0.216 0.225 0.208 0.254 0.447 ...
##  $ CAR_mean                     : num  -1.53e-04 -2.15e+02 -1.31e-01 -4.22e+02 -7.90e+01 ...
##  $ CAR_sigma                    : num  -0.000518 -1.390233 0.004147 -4.48052 1.846285 ...
##  $ CAR_tau                      : num  2.55e+04 1.79e-02 4.47e+01 7.84e-03 5.83e-02 ...
##  $ Color                        : num  -0.2858 -0.1147 0.4386 0.0844 0.0524 ...
##  $ Con                          : num  0 0.000876 0 0.000918 0 ...
##  $ Eta_color                    : num  5863 8080 2546 3144 541 ...
##  $ Eta_e                        : num  1019 2024 416 3131 738 ...
##  $ FluxPercentileRatioMid20     : num  0.0977 0.102 0.1011 0.1158 0.1726 ...
##  $ FluxPercentileRatioMid35     : num  0.18 0.191 0.186 0.215 0.308 ...
##  $ FluxPercentileRatioMid50     : num  0.283 0.304 0.293 0.319 0.437 ...
##  $ FluxPercentileRatioMid65     : num  0.421 0.446 0.431 0.476 0.589 ...
##  $ FluxPercentileRatioMid80     : num  0.65 0.69 0.67 0.721 0.798 ...
##  $ Freq1_harmonics_amplitude_0  : num  0.0657 0.0605 0.0145 0.1008 0.3545 ...
##  $ Freq1_harmonics_amplitude_1  : num  0.02472 0.01308 0.00363 0.01474 0.14089 ...
##  $ Freq1_harmonics_amplitude_2  : num  0.0162 0.00463 0.00543 0.02285 0.09777 ...
##  $ Freq1_harmonics_amplitude_3  : num  0.0139 0.0139 0.0032 0.0276 0.0537 ...
##  $ Freq1_harmonics_rel_phase_1  : num  -1.905 -1.511 -1.569 2.32 0.965 ...
##  $ Freq1_harmonics_rel_phase_2    : num  -0.419 -1.029 -1.544 1.005 2.188 ...
##  $ Freq1_harmonics_rel_phase_3    : num  -1.211 -0.956 -0.636 -0.373 0.497 ...
##  $ Freq2_harmonics_amplitude_0    : num  0.0573 0.0605 0.0125 0.1027 0.0499 ...
##  $ Freq2_harmonics_amplitude_1    : num  0.02554 0.01277 0.00787 0.02689 0.02038 ...
##  $ Freq2_harmonics_amplitude_2    : num  0.02365 0.02085 0.00144 0.02901 0.00437 ...
##  $ Freq2_harmonics_amplitude_3    : num  0.01562 0.0182 0.00267 0.01098 0.01222 ...
##  $ Freq2_harmonics_rel_phase_1    : num  1.316 -2.219 -1.728 0.41 -0.329 ...
##  $ Freq2_harmonics_rel_phase_2    : num  1.4971 -1.9697 0.0435 -0.8164 0.2143 ...
##  $ Freq2_harmonics_rel_phase_3    : num  0.249 0.252 0.708 0.313 -1.304 ...
##  $ Freq3_harmonics_amplitude_0    : num  0.0586 0.059 0.0123 0.0967 0.0467 ...
##  $ Freq3_harmonics_amplitude_1    : num  0.00579 0.00919 0.00658 0.03424 0.02181 ...
##  $ Freq3_harmonics_amplitude_2    : num  0.02062 0.01483 0.00322 0.01072 0.03798 ...
##  $ Freq3_harmonics_amplitude_3    : num  0.02198 0.00415 0.00139 0.04559 0.02451 ...
##  $ Freq3_harmonics_rel_phase_1    : num  -1.317 -1.147 2.616 -1.39 -0.162 ...
##  $ Freq3_harmonics_rel_phase_2    : num  -0.124 1.592 1.841 -0.249 0.122 ...
##  $ Freq3_harmonics_rel_phase_3    : num  -1.734 0.816 2.017 -1.659 -0.136 ...
##  $ LinearTrend                    : num  -2.30e-05 2.11e-05 -7.33e-06 3.60e-05 3.19e-06 ...
##  $ MaxSlope                       : num  53.74 117.16 8.82 201.73 39.49 ...
##  $ Mean                           : num  -3.9 -3.85 -5.84 -3.31 -4.6 ...
##  $ Meanvariance                   : num  -0.0791 -0.0782 -0.0105 -0.1402 -0.0716 ...
##  $ MedianAbsDev                   : num  0.134 0.143 0.027 0.232 0.193 ...
##  $ MedianBRP                      : num  0.807 0.78 0.775 0.717 0.547 ...
##  $ PairSlopeTrend                 : num  -0.0333 -0.0333 0.0333 -0.1667 0.0333 ...
##  $ PercentAmplitude               : num  -0.473 -0.444 -0.051 -0.739 -0.327 ...
##  $ PercentDifferenceFluxPercentile: num  -0.2485 -0.2448 -0.0322 -0.4465 -0.2358 ...
##  $ PeriodLS                       : num  0.0632 0.0579 0.9969 0.2044 0.5194 ...
##  $ Period_fit                     : num  6.08e-01 5.76e-01 2.95e-02 3.04e-01 1.39e-77 ...
##  $ Psi_CS                         : num  0.0732 0.0511 0.0512 0.0622 0.1997 ...
##  $ Psi_eta                        : num  1.98 2.007 1.884 2.013 0.568 ...
##  $ Q31                            : num  0.276 0.286 0.055 0.466 0.462 ...
##  $ Q31_color                      : num  0.4 0.383 0.062 0.524 0.195 0.19 0.017 0.725 0.594 0.595 ..
##  $ Rcs                            : num  0.0413 0.0501 0.0653 0.0505 0.0437 ...
##  $ Skew                           : num  -0.359 -0.5093 0.0241 -0.6302 -0.7182 ...
##  $ SlottedA_length                : num  0.116 0.116 0.128 0.126 0.594 ...
##  $ SmallKurtosis                  : num  4.6313 3.7874 4.0553 2.7477 -0.0612 ...
##  $ Std                            : num  0.3083 0.3012 0.0613 0.4636 0.3297 ...
##  $ StetsonJ                       : num  0.166 0.18 0.112 0.312 3.311 ...
##  $ StetsonK                       : num  0.783 0.708 0.74 0.555 0.789 ...
##  $ StetsonK_AC                    : num  0.638 0.577 0.623 0.664 0.797 ...
##  $ StetsonL                       : num  0.125 0.1021 0.085 0.0806 2.076 ...
##  $ Class                          : int  3 3 3 3 4 1 3 3 3 3 ...
```

```r
summary(train$Class)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   3.000   2.885   3.000   4.000
```

```r
train$Class <- as.factor(train$Class)
test$Class <- as.factor(test$Class)

str(train$Class)
```

```
##  Factor w/ 4 levels "1","2","3","4": 3 3 3 3 4 1 3 3 3 3 ...
```

```
summary(train$Class)
```

```
##   1   2   3   4
##  44  72 492  81
```

1. Fit an RBF kernel to the training set with parameters `gamma` and `cost` both set to 1. Use the model to predict on the test set.

Answer

```
library(e1071)
# training model
m1 <- svm(Class ~ .,
          data = train,
          kernel = "radial",
          gamma = 1,
          cost = 1)
m1
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, kernel = "radial", gamma = 1,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  689
```

```
# predict on test set
pred.m1.test <- predict(m1, newdata = test)
pred.m1.train <- predict(m1, newdata = train)
```

2. Look at the confusion matricies for both the training and testing predictions from the above model. What do you notice about the predictions from this model? *Hint*: The `confusionMatrix` function in the `caret` package is quite useful.

Answer

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
# confusion matrix for prediction on train set
cat('Train set: ')
```

```
## Train set:
```

```
confusionMatrix(pred.m1.train, train$Class)
```

```
## 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
```

```r
# confusion matrix for prediction on test set
cat('Test set: ')
```

```
## Test set:
```

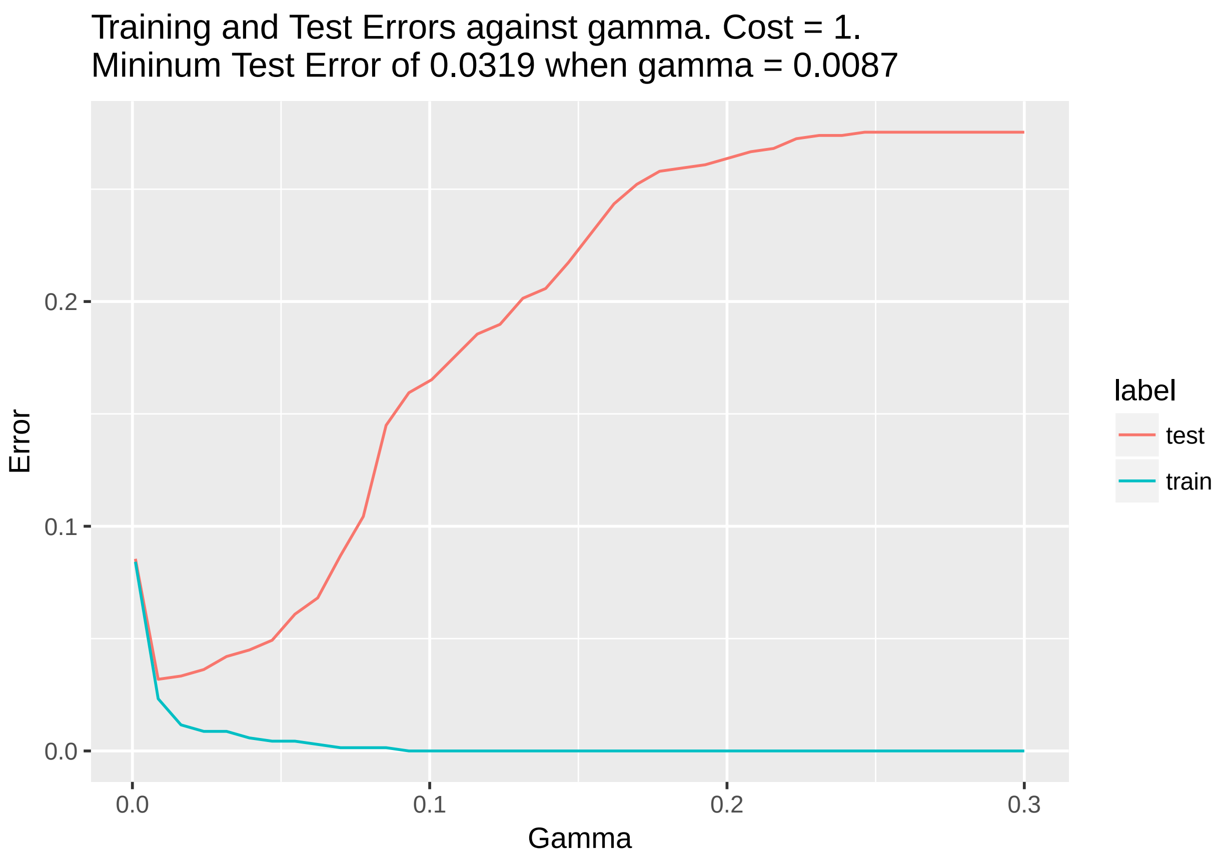
```r
confusionMatrix(pred.m1.test, test$Class)
```

```
## 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
```

The overall accuracy of prediction on the training set is 1. The overall accuracy of prediction on the test set is 0.72. However, the test set is predicting everything as class 3. Without tuning for gamma and cost, the model is labeling everything as class 3 since it is the majority. In order to obtain a more accuracte model, we should tune gamma and cost as we will do in the following.

3. For the RBF kernel, make a figure showing the effect of the kernel parameter γ on the training and test errors? Consider some values of `gamma` between 0.001 and 0.3. Explain what you are seeing.

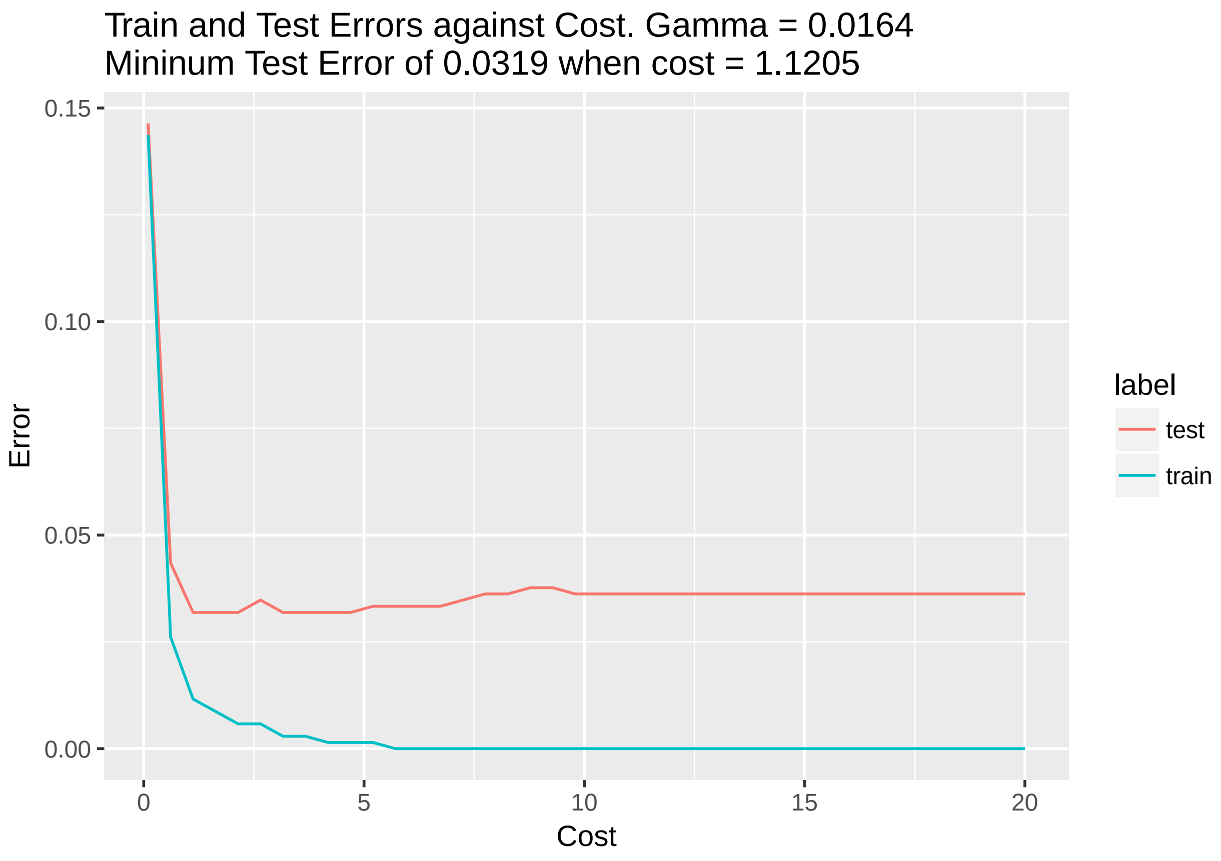
```r
# initialization of error lists
error.train <- list()
error.test <- list()

# gamma values to vary
gamma <- seq(0.001, 0.3, length.out = 40)

# train error and test errors
for(i in 1:length(gamma)){
mod <- svm(Class ~ .,
          data = train,
          kernel = "radial",
          gamma = gamma[i],
          cost = 1)

pred.train <- predict(mod, newdata = train)
pred.test <- predict(mod, newdata = test)

error.train[[i]] <- 1-mean(pred.train == train$Class)
error.test[[i]] <- 1-mean(pred.test == test$Class)
}

# index of error.test with lowest gamma
ind <- which(as.numeric(error.test) == min(as.numeric(error.test)))

errors <- data.frame(gamma = gamma,
           error = c(as.numeric(error.train),as.numeric(error.test)),
           label = c(rep('train', 40), rep('test', 40)))


library(ggplot2)
title <- sprintf('Training and Test Errors against gamma. Cost = 1.
Mininum Test Error of %.4f when gamma = %.4f',
                 as.numeric(error.test)[ind],
                 gamma[ind])
ggplot(errors, aes(x = gamma, y = error, color = label))+
  geom_line()+
  labs(x = 'Gamma', y = 'Error', title = title)
```



The training error decreases as we increase the value of gamma. The test error decreases to a minimum before increasing again as we increase the value of gamma. We can think of gamma as a parameter for the radial function that sits on top of each training data points that provides information when we try to classify a new point. The larger the value of gamma, the narrower and more peaked is the radial function, this means that it has a smaller radius of influence for nearby points. A large gamma with narrow peaks will fit well on the training data set but provides little information on areas that are far away from the peaks or in between the peaks. Hence, when using the model to fit on the test dataset, the error increases for we have little information on most of the areas. On the other hand, when gamma is small, the radial functions are broad and have a greater radius of influence on the nearby area. However, each of these functions sitting on top of different training data points may interfere with one another, for a point that is sitting in between two peaks, both have a moderate effect on the point, this cause the model to break down at some point and the error will increase as gamma becomes smaller and smaller.

4. For the RBF kernel, make a figure showing the effect of the `cost` parameter on the training and test errors? Consider some values of `cost` in the range of 0.1 to 20. Explain what you are seeing.

```
# initialization of error lists
error.train <- list()
error.test <- list()

# gamma values to vary
cost <- seq(0.1, 20, length.out = 40)

# train error and test errors
for(i in 1:length(cost)){
mod <- svm(Class ~ .,
           data = train,
           kernel = "radial",
          cost = cost[i])

pred.train <- predict(mod, newdata = train)
pred.test <- predict(mod, newdata = test)

error.train[[i]] <- 1-mean(pred.train == train$Class)
error.test[[i]] <- 1-mean(pred.test == test$Class)
}

# index of error.test with lowest cost
ind <- which(as.numeric(error.test) == min(as.numeric(error.test)))

errors <- data.frame(cost = cost,
           error = c(as.numeric(error.train),as.numeric(error.test)),
           label = c(rep('train', 40), rep('test', 40)))


library(ggplot2)
title <- sprintf('Train and Test Errors against Cost. Gamma = %.4f
Mininum Test Error of %.4f when cost = %.4f',
                 mod$gamma,
                 as.numeric(error.test)[ind],
                 cost[ind])
ggplot(errors, aes(x = cost, y = error, color = label))+
  geom_line()+
  labs(x = 'Cost', y = 'Error', title = title)
```



The train error decreases with increased value of Cost. The test error decreases and then increases with increased value of cost. A larger cost parameter gives emphasis on the accuracy of the fit and creates a softer margin. However, when cost is too large, the model will overfit to the training data.

5. Now the fun part: fit SVM models with the linear, polynomial (degree 2) and RBF kernels to the training set, and report the misclassification error on the test set for each model. Do not forget to tune all relevant parameters using 5-fold cross-validation on the training set (tuning may take a while!). *Hint*: Use the `tune` function from the `e1071` library. You can plot the error surface using `plot` on the output of a `tune` function.

Linear

```
## Linear svm
# linear model does not need to tune gamma
linear.tune <- tune(svm,
                Class ~ .,
                data = train,
                kernel = "linear",
                ranges = list(cost = seq(0.01, 1, by = 0.01)),
                tunecontrol = tune.control(cross = 5))

linear.tune
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 5-fold cross validation
##
## - best parameters:
##  cost
##  0.08
##
## - best performance: 0.02899609
```

```
# linear svm evaluation
str(linear.tune$performances)
```

```
## 'data.frame':    100 obs. of  3 variables:
##  $ cost      : num  0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 ...
##  $ error     : num  0.061 0.0392 0.0363 0.0334 0.0305 ...
##  $ dispersion: num  0.01521 0.00824 0.00513 0.00651 0.00789 ...
```

```
# best model
cat('Best Model:\n')
```

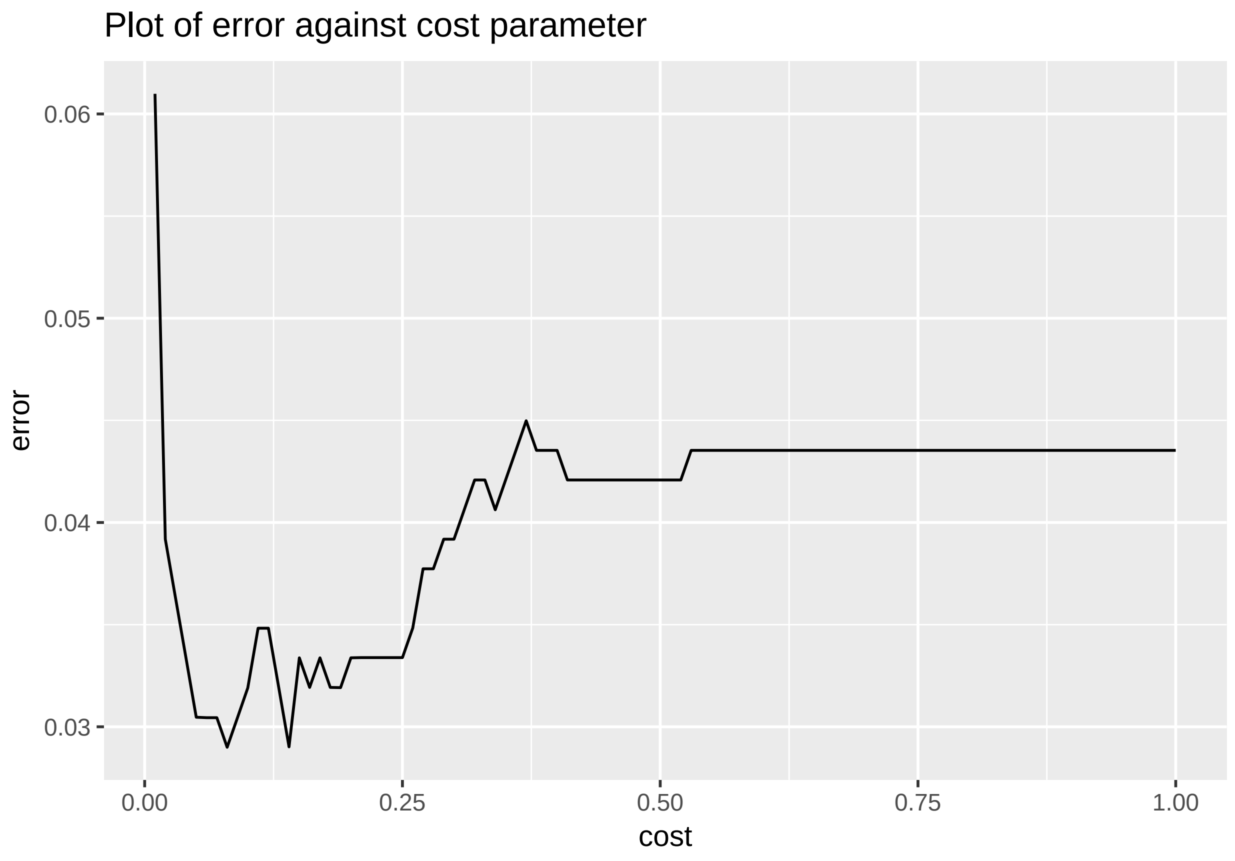
```
## Best Model:
```

```
linear.tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = train, ranges = list(cost = seq(0.01,
##     1, by = 0.01)), tunecontrol = tune.control(cross = 5), kernel = "linear")
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.08
##       gamma:  0.01639344
##
## Number of Support Vectors:  124
```

```
# visualization
ggplot(linear.tune$performances, mapping = aes(x = cost, y = error)) +
  geom_line()+
  ggtitle("Plot of error against cost parameter")
```



```
# linear svm prediction
pred.linear.test <- predict(linear.tune$best.model, newdata = test)
confusionMatrix(pred.linear.test, test$Class)
```

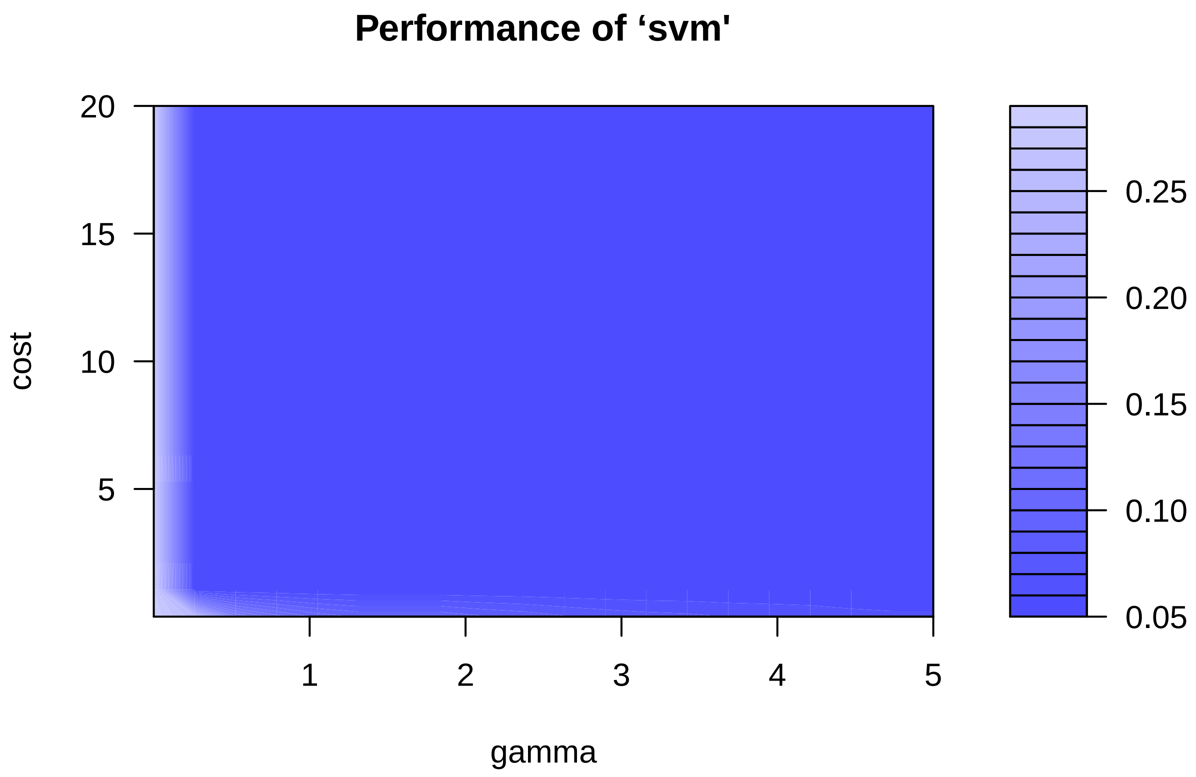
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1  43   1   1   0
##          2   1  70   1   0
##          3   1   2 495   2
##          4   1   0   2  70
##
## Overall Statistics
##
##                Accuracy : 0.9826
##                    95% CI : (0.9698, 0.991)
##       No Information Rate : 0.7232
##       P-Value [Acc > NIR] : < 2.2e-16
##
##                     Kappa : 0.9613
##    Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                       Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.93478   0.9589   0.9920   0.9722
## Specificity            0.99689   0.9968   0.9738   0.9951
## Pos Pred Value         0.95556   0.9722   0.9900   0.9589
## Neg Pred Value         0.99535   0.9951   0.9789   0.9968
## Prevalence             0.06667   0.1058   0.7232   0.1043
## Detection Rate         0.06232   0.1014   0.7174   0.1014
## Detection Prevalence   0.06522   0.1043   0.7246   0.1058
## Balanced Accuracy      0.96584   0.9778   0.9829   0.9837
```

Polynmial with degree 2

```
## Polynomial (degree 2) svm
poly.tune <- tune(svm,
                Class ~ .,
                data = train,
                kernel = "polynomial",
                degree = 2,
                ranges = list(gamma = seq(0.0001, 5, length.out = 20),
                              cost = seq(0.0001, 20, length.out = 20)),
                tunecontrol = tune.control(cross = 5))

poly.tune
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 5-fold cross validation
##
## - best parameters:
##      gamma     cost
##  0.2632526 1.052726
##
## - best performance: 0.05081985
```

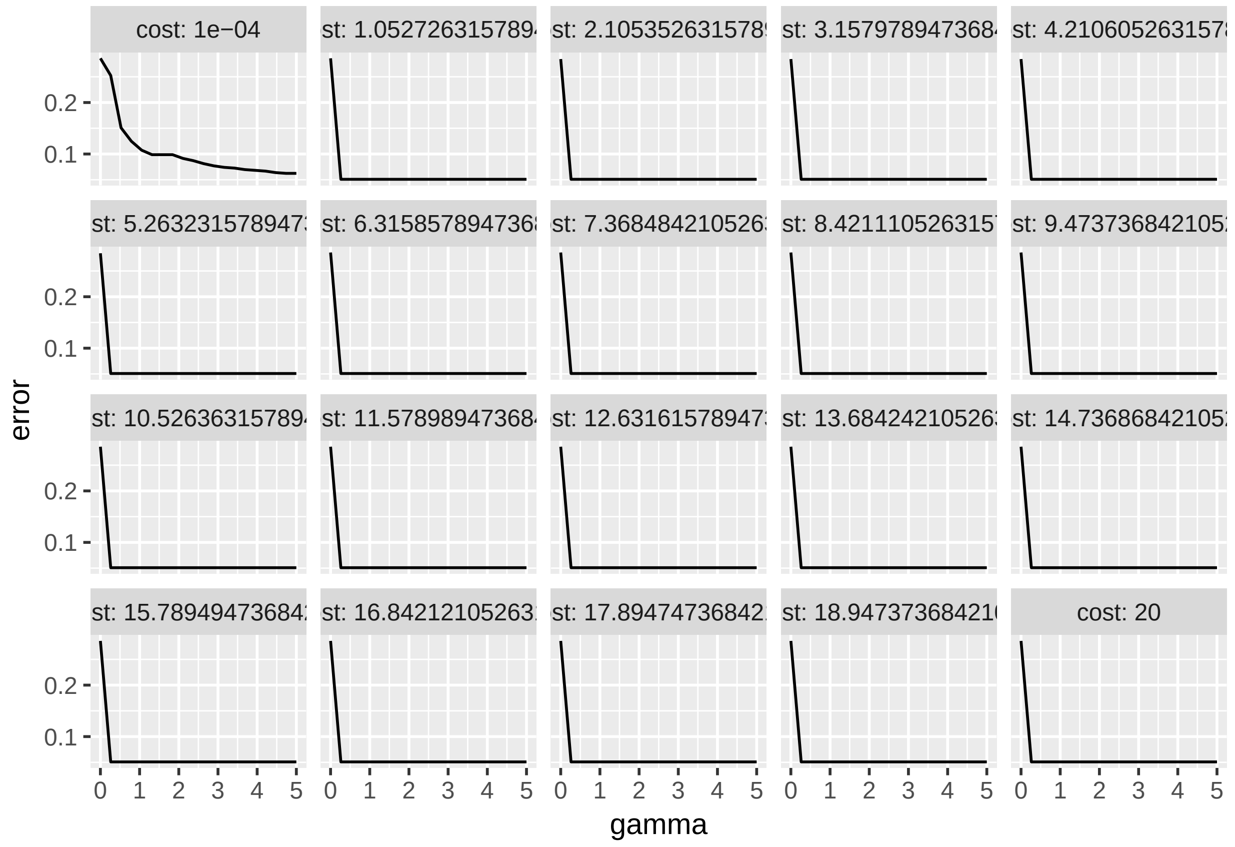
```
# polynomial svm evaluation
plot(poly.tune)
```



```
str(poly.tune$performances)
```

```
## 'data.frame':	400 obs. of  4 variables:
##  $ gamma     : num  0.0001 0.2633 0.5264 0.7896 1.0527 ...
##  $ cost      : num  1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 ...
##  $ error     : num  0.286 0.253 0.151 0.125 0.107 ...
##  $ dispersion: num  0.0238 0.0455 0.0221 0.0337 0.0283 ...
```

```
ggplot(poly.tune$performances,
       mapping = aes(x = gamma, y = error)) +
  geom_line() +
  facet_wrap(~cost, labeller = label_both)
```



```
# best model
cat('Best Model:\n')
```

```
## Best Model:
```

```
poly.tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = train, ranges = list(gamma = seq(1e-04,
##     5, length.out = 20), cost = seq(1e-04, 20, length.out = 20)),
##     tunecontrol = tune.control(cross = 5), kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1.052726
##      degree:  2
##       gamma:  0.2632526
##      coef.0:  0
##
## Number of Support Vectors:  239
```

```
# poly svm prediction
pred.poly.test <- predict(poly.tune$best.model, newdata = test)
confusionMatrix(pred.poly.test, test$Class)
```

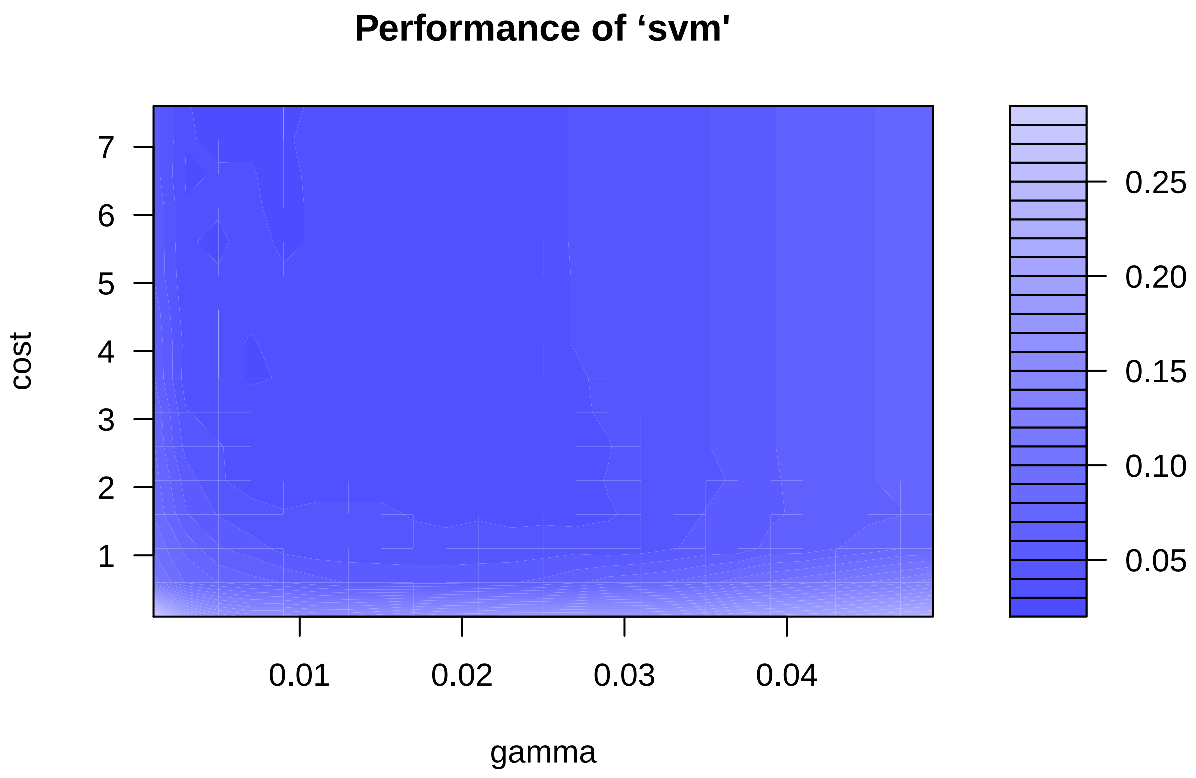
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1  41   1   3   0
##          2   1  54   4   0
##          3   3  16 490   6
##          4   1   2   2  66
##
## Overall Statistics
##
##                Accuracy : 0.9435
##                  95% CI : (0.9235, 0.9595)
##     No Information Rate : 0.7232
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8704
##  Mcnemar's Test P-Value : 0.05765
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity           0.89130  0.73973   0.9820  0.91667
## Specificity           0.99379  0.99190   0.8691  0.99191
## Pos Pred Value        0.91111  0.91525   0.9515  0.92958
## Neg Pred Value        0.99225  0.96989   0.9486  0.99031
## Prevalence            0.06667  0.10580   0.7232  0.10435
## Detection Rate        0.05942  0.07826   0.7101  0.09565
## Detection Prevalence  0.06522  0.08551   0.7464  0.10290
## Balanced Accuracy     0.94255  0.86581   0.9255  0.95429
```

RBF

```r
## RBF svm
rbf.tune <- tune(svm,
                Class ~ .,
                data = train,
                kernel = "radial",
                ranges = list(gamma = seq(0.001, 0.05, by = 0.002),
                              cost = seq(0.1, 8, by = 0.5)),
                tunecontrol = tune.control(cross = 5))
rbf.tune
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 5-fold cross validation
##
## - best parameters:
##  gamma cost
##  0.005  7.6
##
## - best performance: 0.02758913
```

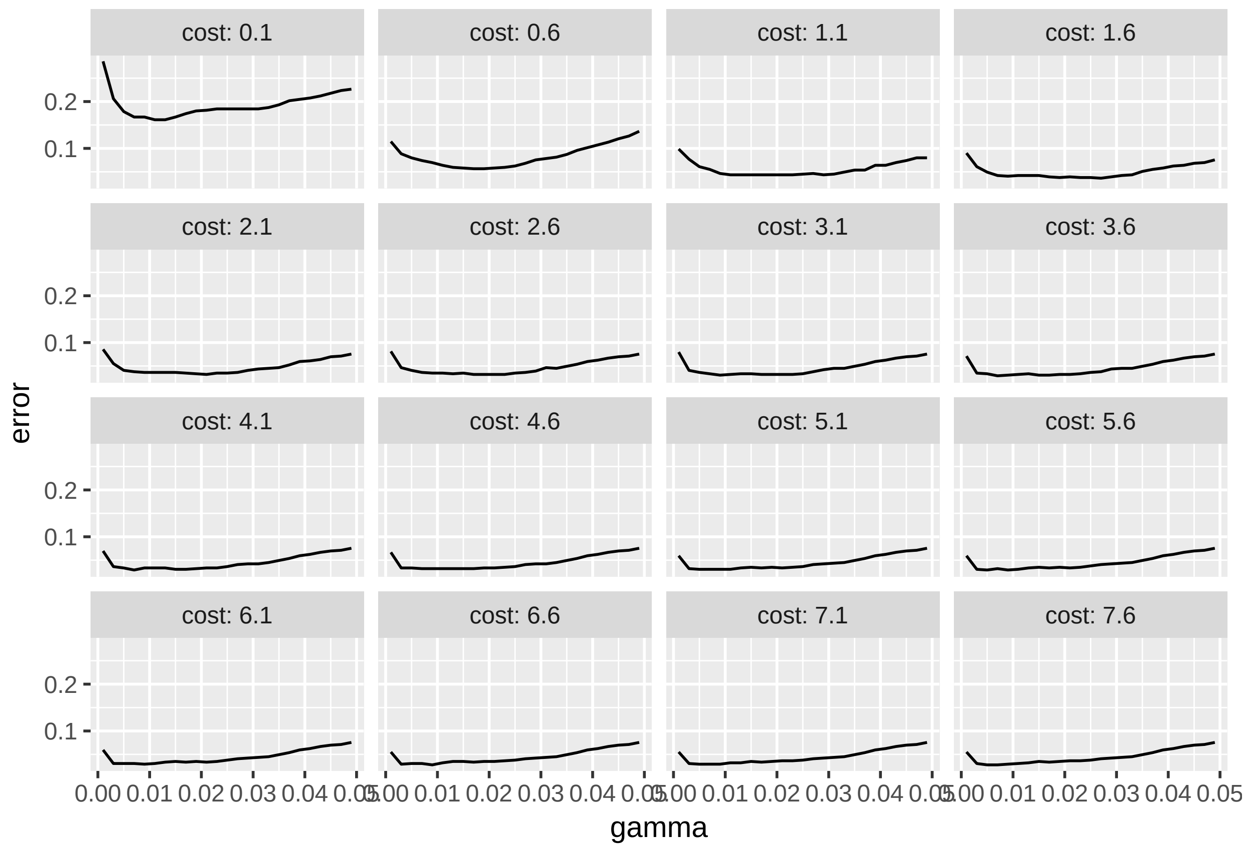
```r
# rbf svm evaluation
plot(rbf.tune)
```



```r
str(rbf.tune$performances)
```

```
## 'data.frame':    400 obs. of  4 variables:
##  $ gamma     : num  0.001 0.003 0.005 0.007 0.009 0.011 0.013 0.015 0.017 0.019 ...
##  $ cost      : num  0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
##  $ error     : num  0.286 0.206 0.179 0.167 0.167 ...
##  $ dispersion: num  0.051 0.0468 0.0379 0.0359 0.0344 ...
```

```r
ggplot(rbf.tune$performances,
       mapping = aes(x = gamma, y = error)) +
  geom_line() +
  facet_wrap(~cost, labeller = label_both)
```



```r
# best model
cat('Best Model:\n')
```

```
## Best Model:
```

```r
rbf.tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = train, ranges = list(gamma = seq(0.001,
##     0.05, by = 0.002), cost = seq(0.1, 8, by = 0.5)), tunecontrol = tune.control(cross = 5),
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  7.6
##       gamma:  0.005
##
## Number of Support Vectors:  156
```

```r
# rbf svm prediction
pred.rbf.test <- predict(rbf.tune$best.model, newdata = test)
confusionMatrix(pred.rbf.test, test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1  42   0   1   0
##          2   0  70   3   0
##          3   3   3 493   4
##          4   1   0   2  68
##
## Overall Statistics
##
##                Accuracy : 0.9754
##                  95% CI : (0.9608, 0.9856)
##     No Information Rate : 0.7232
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9448
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity           0.91304   0.9589   0.9880  0.94444
## Specificity           0.99845   0.9951   0.9476  0.99515
## Pos Pred Value        0.97674   0.9589   0.9801  0.95775
## Neg Pred Value        0.99382   0.9951   0.9679  0.99354
## Prevalence            0.06667   0.1058   0.7232  0.10435
## Detection Rate        0.06087   0.1014   0.7145  0.09855
## Detection Prevalence  0.06232   0.1058   0.7290  0.10290
## Balanced Accuracy     0.95575   0.9770   0.9678  0.96980
```

6. What is the best model in terms of testing accuracy? How does your final model compare with a naive classifier that predicts the most common class (3) on all points?

Hint: This is a moderate-sized dataset, but keep in mind that computation will always be a limiting factor when tuning machine learning algorithms. For timing reference, attempting 40 combinations of `cost` and `gamma` using an RBF kernel on the training dataset took about 15 minutes to tune on a recent Macbook. The other kernels were much faster, e.g. linear should be done in only a few minutes.

Answer

The best model in terms of test accuracy is the linear model with an overall accuracy of 98.55%, while the polynomial with degree 2 model has an overall accuracy of 94.35%, and the rbf model has an overall accuracy of 96.67%.

The fact that the linear model performs well means the four classes are well separated from one another. This is also reflected in our rbf model when we are tuning for gamma and cost, as we increase cost, the error rate drops very fast, indicating that with only a small compensation for accuracy, the model is able to separate the classes well.

Although the linear model has a lower overall accuracy compared to the naive classifier which classified everything as class 3, the linear model is better as it is able to predict each class with more than 95% accuracy.

Problem 2: Return of the Bayesian Hierarchical Model

We're going to continue working with the dataset introduced in Homework 3 about contraceptive usage by 1934 Bangladeshi women. The data are in `dataset_2.txt` which is now a merge of the training and test data that appeared in Homework 2.

In order to focus on the benefits of Hierarchical Modeling we're going to consider a model with only one covariate (and intercept term).

1. Fit the following three models

   (a) Pooled Model: a single logistic regression for `contraceptive_use` as a function of `living.children`. Do not include `district` information. You should use the `glm` function to fit this model. Interpret the estimated model.

Answer

```
df <- read.csv('./datasets/dataset_2.txt')
str(df)
```

```
## '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 ...
```

```
summary(df)
```

```
##     district         urban        living.children    age_mean
##  Min.   :101.0   Min.   :0.0000   Min.   :1.000   Min.   :-13.560000
##  1st Qu.:114.0   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.: -7.559900
##  Median :129.0   Median :0.0000   Median :3.000   Median : -1.559900
##  Mean   :129.3   Mean   :0.2906   Mean   :2.652   Mean   :  0.002198
##  3rd Qu.:145.0   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:  6.440000
##  Max.   :160.0   Max.   :1.0000   Max.   :4.000   Max.   : 19.440000
##  contraceptive_use
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3925
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

```
mod.pooled <- glm(contraceptive_use ~ living.children,
                  data = df,
                  family = binomial(link = "logit"))

summary(mod.pooled)
```

```
##
## Call:
## glm(formula = contraceptive_use ~ living.children, family = binomial(link = "logit"),
##     data = df)
##
## 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 2.61e-08 ***
## ---
## 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
```

```
# coef
coef <- mod.pooled$coefficients
coef.pooled <- as.numeric(coef["living.children"])
```

The pooled model disregard the differences between districts and fit an overall model to all data points. We are assuming a single errors distribution for all districts. The coefficient for living.children in this model is the same regardless of which district a woman comes from.

(b) Unpooled Model: a model that instead fits a separate logistic regression for each `district`. Use the `glm` function to this model. *Hint* The separate logistic regression models can be fit using one application of `glm` by having the model formula be `contraceptive_use ~ -1 + living.children * as.factor(district)`. Explain why this model formula is accomplishing the task of fitting separate models per district. Examine the summary output of the fitted model.

Answer

```
mod.unpooled <- glm(contraceptive_use ~ -1 + living.children * as.factor(district),
                    data = df,
                    family = binomial(link = "logit"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(mod.unpooled)
```

```
##
## Call:
## glm(formula = contraceptive_use ~ -1 + living.children * as.factor(district),
##     family = binomial(link = "logit"), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7816  -0.9576  -0.6271   1.1045   2.2425
##
## Coefficients:
##                                           Estimate Std. Error z value
## living.children                          1.064e-01  1.806e-01   0.589
## as.factor(district)101                  -1.376e+00  5.758e-01  -2.390
## as.factor(district)102                  -2.062e+00  1.543e+00  -1.336
## as.factor(district)103                   1.657e+01  3.298e+03   0.005
## as.factor(district)104                  -8.343e-01  1.090e+00  -0.765
## as.factor(district)105                  -8.590e-01  8.083e-01  -1.063
## as.factor(district)106                  -1.343e+00  6.998e-01  -1.919
## as.factor(district)107                  -6.117e-01  1.164e+00  -0.525
## as.factor(district)108                  -1.924e+00  9.073e-01  -2.121
## as.factor(district)109                  -1.679e+00  1.544e+00  -1.088
## as.factor(district)110                  -1.891e+00  2.049e+00  -0.923
## as.factor(district)111                  -1.657e+01  1.042e+03  -0.016
## as.factor(district)112                    4.259e-01  1.106e+00   0.385
## as.factor(district)113                   -1.920e+00  1.061e+00  -1.810
## as.factor(district)114                   -7.132e-02  4.190e-01  -0.170
## as.factor(district)115                   -2.022e+00  1.130e+00  -1.790
## as.factor(district)116                   -7.041e-01  9.589e-01  -0.734
## as.factor(district)117                   -5.155e+00  2.996e+00  -1.721
## as.factor(district)118                   -9.284e-01  6.899e-01  -1.346
## as.factor(district)119                   -1.662e+00  9.771e-01  -1.701
## as.factor(district)120                   -4.055e-01  1.083e+00  -0.374
## as.factor(district)121                    2.092e+00  1.275e+00   1.642
## as.factor(district)122                   -2.621e-01  1.442e+00  -0.182
## as.factor(district)123                   -3.545e+00  2.550e+00  -1.390
## as.factor(district)124                   -2.791e+01  1.432e+03  -0.019
## as.factor(district)125                   -2.661e+00  7.653e-01  -3.477
## as.factor(district)126                   -1.285e+00  1.881e+00  -0.683
## as.factor(district)127                   -1.524e+00  8.618e-01  -1.769
## as.factor(district)128                   -2.792e+00  1.122e+00  -2.489
## as.factor(district)129                   -2.409e+00  9.649e-01  -2.496
## as.factor(district)130                   -2.150e+00  6.768e-01  -3.177
## as.factor(district)131                   -7.139e-01  8.372e-01  -0.853
## as.factor(district)132                   -1.982e+00  1.676e+00  -1.183
## as.factor(district)133                   -5.714e+00  3.168e+00  -1.804
## as.factor(district)134                    4.512e-01  1.006e+00   0.449
## as.factor(district)135                   -5.451e-01  7.747e-01  -0.704
## as.factor(district)136                   -6.456e-01  1.468e+00  -0.440
## as.factor(district)137                    9.290e-01  1.665e+00   0.558
## as.factor(district)138                   -2.896e+01  1.866e+03  -0.016
## as.factor(district)139                    7.588e-01  8.597e-01   0.883
## as.factor(district)140                   -7.386e-02  6.892e-01  -0.107
## as.factor(district)141                   -1.754e-01  1.068e+00  -0.164
## as.factor(district)142                   -3.082e+01  1.554e+03  -0.020
## as.factor(district)143                   -8.442e-01  6.719e-01  -1.256
## as.factor(district)144                   -2.262e+00  1.263e+00  -1.791
## as.factor(district)145                   -1.942e+00  1.003e+00  -1.937
## as.factor(district)146                   -3.495e-01  5.370e-01  -0.651
## as.factor(district)147                   -2.249e+00  1.468e+00  -1.532
## as.factor(district)148                   -5.282e-01  7.161e-01  -0.738
## as.factor(district)149                   -1.657e+01  2.013e+03  -0.008
## as.factor(district)150                   -2.081e+00  1.182e+00  -1.761
## as.factor(district)151                    1.211e-01  8.387e-01   0.144
## as.factor(district)152                   -1.134e+00  6.188e-01  -1.833
## as.factor(district)153                   -2.305e+00  1.305e+00  -1.766
## as.factor(district)154                    4.225e+01  1.446e+03   0.029
## as.factor(district)155                    2.891e-01  6.473e-01   0.447
## as.factor(district)156                   -3.106e+00  1.590e+00  -1.954
## as.factor(district)157                   -6.170e-01  7.906e-01  -0.780
## as.factor(district)158                   -2.687e+00  2.997e+00  -0.897
## as.factor(district)159                   -2.744e+00  1.123e+00  -2.444
## as.factor(district)160                   -7.712e-02  8.752e-01  -0.088
## living.children:as.factor(district)102  3.921e-01  5.219e-01   0.751
## living.children:as.factor(district)103 -1.064e-01  1.131e+03   0.000
## living.children:as.factor(district)104  1.682e-01  3.818e-01   0.441
## living.children:as.factor(district)105 -8.259e-03  3.136e-01  -0.026
## living.children:as.factor(district)106  5.898e-02  2.907e-01   0.203
## living.children:as.factor(district)107 -2.435e-01  4.590e-01  -0.530
## living.children:as.factor(district)108  4.354e-01  3.553e-01   1.226
## living.children:as.factor(district)109  1.643e-01  4.932e-01   0.333
## living.children:as.factor(district)110 -3.759e-01  8.876e-01  -0.423
## living.children:as.factor(district)111 -1.064e-01  4.978e+02   0.000
## living.children:as.factor(district)112 -4.454e-01  3.781e-01  -1.178
## living.children:as.factor(district)113  4.752e-01  3.854e-01   1.233
## living.children:as.factor(district)114  1.390e-01  2.398e-01   0.580
## living.children:as.factor(district)115  4.863e-01  4.400e-01   1.105
## living.children:as.factor(district)116  3.672e-01  4.861e-01   0.755
## living.children:as.factor(district)117  1.137e+00  8.029e-01   1.416
## living.children:as.factor(district)118 -1.182e-03  3.007e-01  -0.004
## living.children:as.factor(district)119  3.473e-01  3.719e-01   0.934
## living.children:as.factor(district)120 -1.064e-01  4.440e-01  -0.240
## living.children:as.factor(district)121 -1.273e+00  6.044e-01  -2.107
## living.children:as.factor(district)122 -5.143e-01  5.379e-01  -0.956
## living.children:as.factor(district)123  6.937e-01  7.544e-01   0.919
## living.children:as.factor(district)124  6.423e+00  3.581e+02   0.018
## living.children:as.factor(district)125  7.721e-01  3.055e-01   2.528
## living.children:as.factor(district)126  1.768e-01  6.419e-01   0.275
## living.children:as.factor(district)127 -9.857e-02  3.441e-01  -0.286
## living.children:as.factor(district)128  4.329e-01  3.711e-01   1.167
## living.children:as.factor(district)129  4.815e-01  3.676e-01   1.310
## living.children:as.factor(district)130  7.708e-01  3.141e-01   2.454
## living.children:as.factor(district)131  8.525e-02  3.264e-01   0.261
## living.children:as.factor(district)132  9.078e-02  5.088e-01   0.178
## living.children:as.factor(district)133  1.562e+00  9.376e-01   1.666
## living.children:as.factor(district)134 -3.898e-02  3.668e-01  -0.106
## living.children:as.factor(district)135  8.878e-02  3.140e-01   0.283
## living.children:as.factor(district)136 -9.241e-02  5.196e-01  -0.178
## living.children:as.factor(district)137 -3.632e-01  5.464e-01  -0.665
## living.children:as.factor(district)138  7.032e+00  4.666e+02   0.015
## living.children:as.factor(district)139 -4.006e-01  3.455e-01  -1.159
## living.children:as.factor(district)140 -1.337e-01  2.934e-01  -0.456
## living.children:as.factor(district)141 -4.390e-02  3.971e-01  -0.111
## living.children:as.factor(district)142  1.530e+01  7.770e+02   0.020
## living.children:as.factor(district)143  3.216e-01  3.208e-01   1.003
## living.children:as.factor(district)144  2.611e-01  4.432e-01   0.589
## living.children:as.factor(district)145  3.604e-01  3.856e-01   0.935
## living.children:as.factor(district)146  5.585e-02  2.552e-01   0.219
## living.children:as.factor(district)147  6.656e-01  5.079e-01   1.311
## living.children:as.factor(district)148  1.566e-01  3.273e-01   0.478
## living.children:as.factor(district)149 -1.064e-01  9.236e+02   0.000
## living.children:as.factor(district)150  6.251e-01  4.193e-01   1.491
## living.children:as.factor(district)151 -2.084e-01  3.315e-01  -0.629
## living.children:as.factor(district)152  2.385e-01  2.778e-01   0.858
## living.children:as.factor(district)153  6.511e-01  4.766e-01   1.366
## living.children:as.factor(district)154 -2.833e+01  8.508e+02  -0.033
## living.children:as.factor(district)155 -9.604e-02  3.012e-01  -0.319
## living.children:as.factor(district)156  4.303e-01  4.913e-01   0.876
## living.children:as.factor(district)157  6.988e-02  3.380e-01   0.207
## living.children:as.factor(district)158  6.310e-02  9.586e-01   0.066
## living.children:as.factor(district)159  4.673e-01  4.087e-01   1.143
## living.children:as.factor(district)160 -5.854e-01  3.774e-01  -1.551
##                                          Pr(>|z|)
## living.children                          0.555935
## as.factor(district)101                   0.016860 *
## as.factor(district)102                   0.181504
## as.factor(district)103                   0.995992
## as.factor(district)104                   0.444108
## as.factor(district)105                   0.287924
## as.factor(district)106                   0.054976 .
## as.factor(district)107                   0.599294
## as.factor(district)108                   0.033955 *
## as.factor(district)109                   0.276672
## as.factor(district)110                   0.356184
## as.factor(district)111                   0.987314
## as.factor(district)112                   0.700071
## as.factor(district)113                   0.070314 .
## as.factor(district)114                   0.864837
## as.factor(district)115                   0.073477 .
## as.factor(district)116                   0.462796
## as.factor(district)117                   0.085283 .
## as.factor(district)118                   0.178399
## as.factor(district)119                   0.089026 .
## as.factor(district)120                   0.708156
## as.factor(district)121                   0.100650
## as.factor(district)122                   0.855723
## as.factor(district)123                   0.164503
## as.factor(district)124                   0.984454
## as.factor(district)125                   0.000507 ***
## as.factor(district)126                   0.494618
## as.factor(district)127                   0.076915 .
## as.factor(district)128                   0.012810 *
## as.factor(district)129                   0.012544 *
## as.factor(district)130                   0.001486 **
## as.factor(district)131                   0.393789
## as.factor(district)132                   0.236723
## as.factor(district)133                   0.071295 .
## as.factor(district)134                   0.653708
## as.factor(district)135                   0.481655
## as.factor(district)136                   0.660177
## as.factor(district)137                   0.576785
## as.factor(district)138                   0.987620
## as.factor(district)139                   0.377492
## as.factor(district)140                   0.914651
## as.factor(district)141                   0.869544
## as.factor(district)142                   0.984177
## as.factor(district)143                   0.208964
## as.factor(district)144                   0.073251 .
## as.factor(district)145                   0.052798 .
## as.factor(district)146                   0.515140
## as.factor(district)147                   0.125532
## as.factor(district)148                   0.460714
## as.factor(district)149                   0.993434
## as.factor(district)150                   0.078245 .
## as.factor(district)151                   0.885215
## as.factor(district)152                   0.066767 .
## as.factor(district)153                  0.077388 .
## as.factor(district)154                  0.976687
## as.factor(district)155                  0.655107
## as.factor(district)156                  0.050741 .
## as.factor(district)157                  0.435114
## as.factor(district)158                  0.369937
## as.factor(district)159                  0.014538 *
## as.factor(district)160                  0.929781
## living.children:as.factor(district)102 0.452516
## living.children:as.factor(district)103 0.999925
## living.children:as.factor(district)104 0.659553
## living.children:as.factor(district)105 0.978988
## living.children:as.factor(district)106 0.839229
## living.children:as.factor(district)107 0.595840
## living.children:as.factor(district)108 0.220344
## living.children:as.factor(district)109 0.738961
## living.children:as.factor(district)110 0.671964
## living.children:as.factor(district)111 0.999830
## living.children:as.factor(district)112 0.238764
## living.children:as.factor(district)113 0.217534
## living.children:as.factor(district)114 0.562115
## living.children:as.factor(district)115 0.269072
## living.children:as.factor(district)116 0.450067
## living.children:as.factor(district)117 0.156763
## living.children:as.factor(district)118 0.996863
## living.children:as.factor(district)119 0.350370
## living.children:as.factor(district)120 0.810650
## living.children:as.factor(district)121 0.035154 *
## living.children:as.factor(district)122 0.339033
## living.children:as.factor(district)123 0.357841
## living.children:as.factor(district)124 0.985689
## living.children:as.factor(district)125 0.011487 *
## living.children:as.factor(district)126 0.782947
## living.children:as.factor(district)127 0.774529
## living.children:as.factor(district)128 0.243321
## living.children:as.factor(district)129 0.190237
## living.children:as.factor(district)130 0.014136 *
## living.children:as.factor(district)131 0.793987
## living.children:as.factor(district)132 0.858387
## living.children:as.factor(district)133 0.095626 .
## living.children:as.factor(district)134 0.915366
## living.children:as.factor(district)135 0.777354
## living.children:as.factor(district)136 0.858838
## living.children:as.factor(district)137 0.506233
## living.children:as.factor(district)138 0.987975
## living.children:as.factor(district)139 0.246355
## living.children:as.factor(district)140 0.648495
## living.children:as.factor(district)141 0.911982
## living.children:as.factor(district)142 0.984287
## living.children:as.factor(district)143 0.316094
## living.children:as.factor(district)144 0.555693
## living.children:as.factor(district)145 0.349941
## living.children:as.factor(district)146 0.826738
## living.children:as.factor(district)147 0.189990
## living.children:as.factor(district)148 0.632365
## living.children:as.factor(district)149 0.999908
## living.children:as.factor(district)150 0.136040
## living.children:as.factor(district)151 0.529483
## living.children:as.factor(district)152 0.390667
## living.children:as.factor(district)153 0.171859
## living.children:as.factor(district)154 0.973437
## living.children:as.factor(district)155 0.749852
## living.children:as.factor(district)156 0.381028
## living.children:as.factor(district)157 0.836232
## living.children:as.factor(district)158 0.947515
## living.children:as.factor(district)159 0.252905
## living.children:as.factor(district)160 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
```

```
coef <- mod.unpooled$coefficients
coefnames.unpooled <- names(coef)[seq(61,120)]
coef.unpooled <- as.numeric(coef[seq(62,120)])
```

This model formula is accomplishing the task of fitting separate models per distric by including interaction terms of living.children with each district. The coefficient for living.children:district(i) where i is the district number represents the model specific to the ith district.

In the unpooled model, we are considering each district independently and assuming that the errors in each district vary independently of the other districts. This is to say, the errors in each district has its own distribution and are not drawn from a common error distribution. By doing so, we notice that in this model the values of coefficients fitted for living.children in each district varies a lot.

(c) Bayesian Hierarchical Logistic Model: a Bayesian hierarchical logistic regression model with `district` as the grouping variable. Use the `MCMChlogit` function in the `MCMCpack` library using arguments similar to the reaction time model in the lecture notes. Make sure that both coefficients of the linear predictor are assumed to vary by `district` in the model specification. Describe briefly in words how the results of this model are different from the pooled and unpooled models of parts (a) and (b).

Answer

```
library(MCMCpack)
```

```
## Loading required package: coda
```

```
## 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)
## ##
```

```
mod.hierarchical <- MCMChlogit(fixed = contraceptive_use ~ living.children,
                               random = ~ living.children,
                               group = "district",
                               data = df,
                               burnin = 5000, mcmc = 10000, # burn-in and total iterations
                               thin = 1, verbose = 1,
                               beta.start = NA, sigma2.start = NA, Vb.start = NA,
                               mubeta = c(-1.00804, 0.21240), # prior mean of beta_0 and beta_1
                               Vbeta = 10000,
                               r = 3, R = diag(c(1, 0.1)), nu = 0.001, delta = 0.001,
                               FixOD = 1)
```

```
##
## Running the Gibbs sampler. It may be long, keep cool :)
##
## **********:10.0%, mean accept. rate=0.445
## **********:20.0%, mean accept. rate=0.446
## **********:30.0%, mean accept. rate=0.443
## **********:40.0%, mean accept. rate=0.444
## **********:50.0%, mean accept. rate=0.445
## **********:60.0%, mean accept. rate=0.447
## **********:70.0%, mean accept. rate=0.445
## **********:80.0%, mean accept. rate=0.446
## **********:90.0%, mean accept. rate=0.445
## **********:100.0%, mean accept. rate=0.448
```

```
# list of coefficients
coef <- apply(mod.hierarchical$mcmc, 2, mean)
coefnames.hierarchical <- names(coef)[seq(63,122)]
coef.hierarchical <- as.numeric(coef[seq(63,122)])

# prediction accuracy on training set
pred <- round(mod.hierarchical$theta.pred)
mean(pred == df$contraceptive_use)
```

```
## [1] 0.6571872
```

The overall accuracy for prediction on the training set for this model is 65.7%.

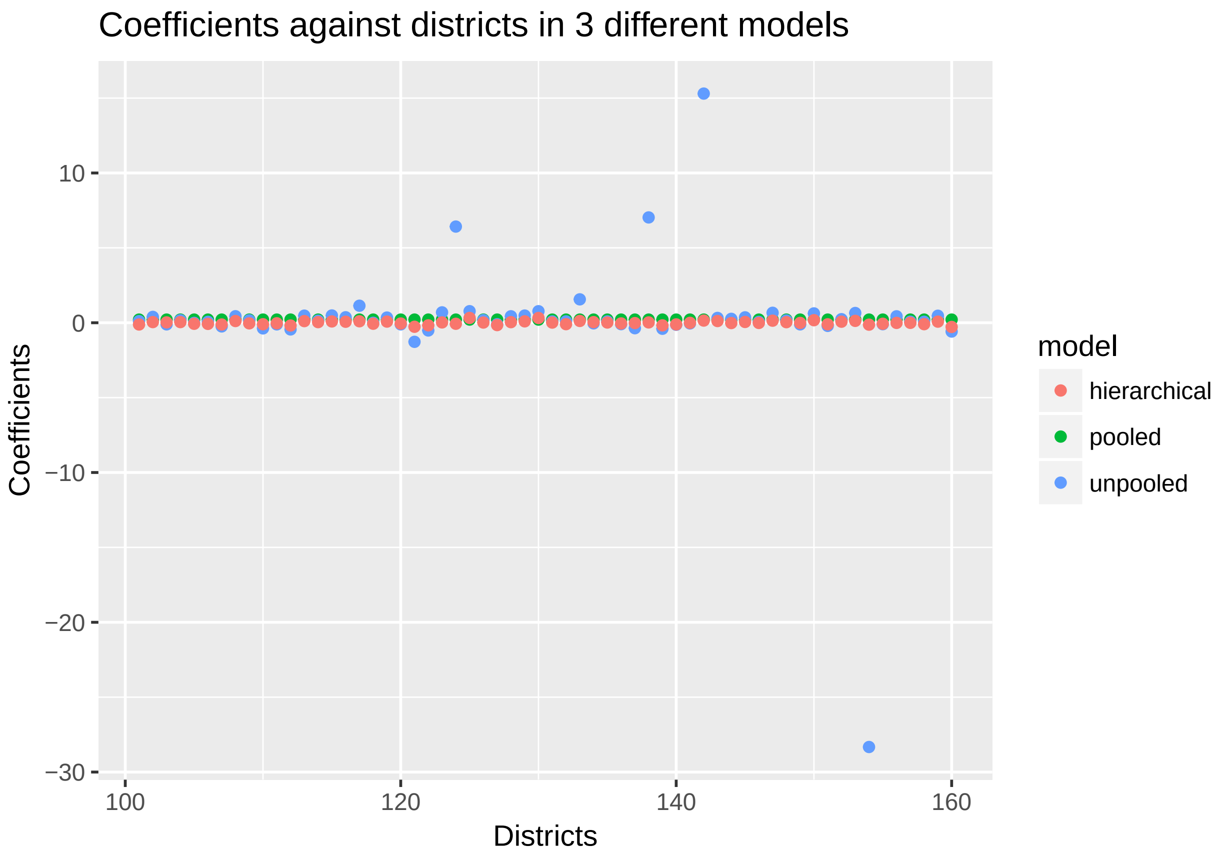
The Bayesian hierarchical model assumes that all districts behave in a similar way, but each district has some random noise, and each district's noise is a random draw from a common distribution of noise. We can think of the distribution as a rubber band around the coefficients for each district, they can vary, but the random effects distribution keeps them from being too far apart. We notice that the coefficients vary less in the Bayesian hierarchical model compared to the unpooled model. This is because the unpooled model assumes that each district has an independent error distribution, and we are fitting models to each district independently, which results in a higher variability in the coefficients. The coefficients in the pooled model are the same regardless of district as the model does not differentiate between different districts.

2. In class we discussed that one of the benefits of using Bayesian hierarchical models is that it naturally shares information across the groupings. In this case, information is shared across districts. This is generally known as shrinkage. To explore the degree of shrinkage, we are going to compare coefficients across models and districts based on your results from part 1 above.

   (a) Create a single figure that shows the estimated coefficient to `living.children` as a function of district in each of the three models above. The horizontal axis should be the districts, and the vertical axis should be the estimated coefficient value (generally three estimated coefficients at each district corresponding to the three models). Make sure that the points plotted for each model are distinct (different colors and/or plotting characters), and that you create a legend identifying the model-specific points on the figure. You may want to consider adjusting the vertical axis if some estimated coefficients are so large (positively or negatively) that they obscure the general pattern of the bulk of points. Be sure to explain your decision.

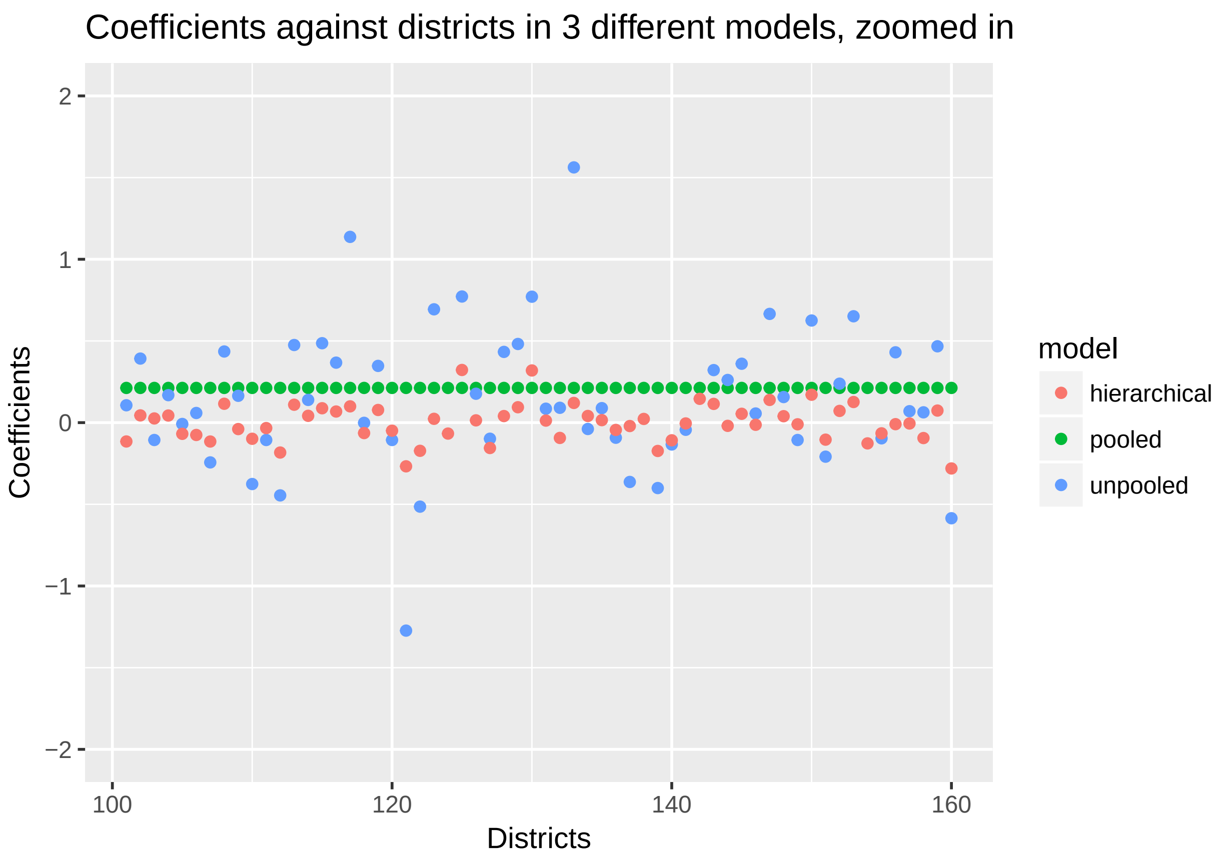
Answer

```r
df.coef <- data.frame(districts = c(seq(101, 160), seq(101, 160), seq(101, 160)),
          coef = c(rep(coef.pooled, 60), 1.064e-01, coef.unpooled, coef.hierarchical),
          model = c(rep('pooled', 60), rep('unpooled', 60), rep('hierarchical', 60)))

library(ggplot2)
ggplot(df.coef, aes(x = districts, y = coef, color = model))+
  geom_point()+
  labs(x = 'Districts', y = 'Coefficients',
       title = 'Coefficients against districts in 3 different models')
```



```r
ggplot(df.coef, aes(x = districts, y = coef, color = model))+
  geom_point()+
  ylim(-2, 2)+
  labs(x = 'Districts', y = 'Coefficients',
       title = 'Coefficients against districts in 3 different models, zoomed in')
```

```
## Warning: Removed 4 rows containing missing values (geom_point).
```

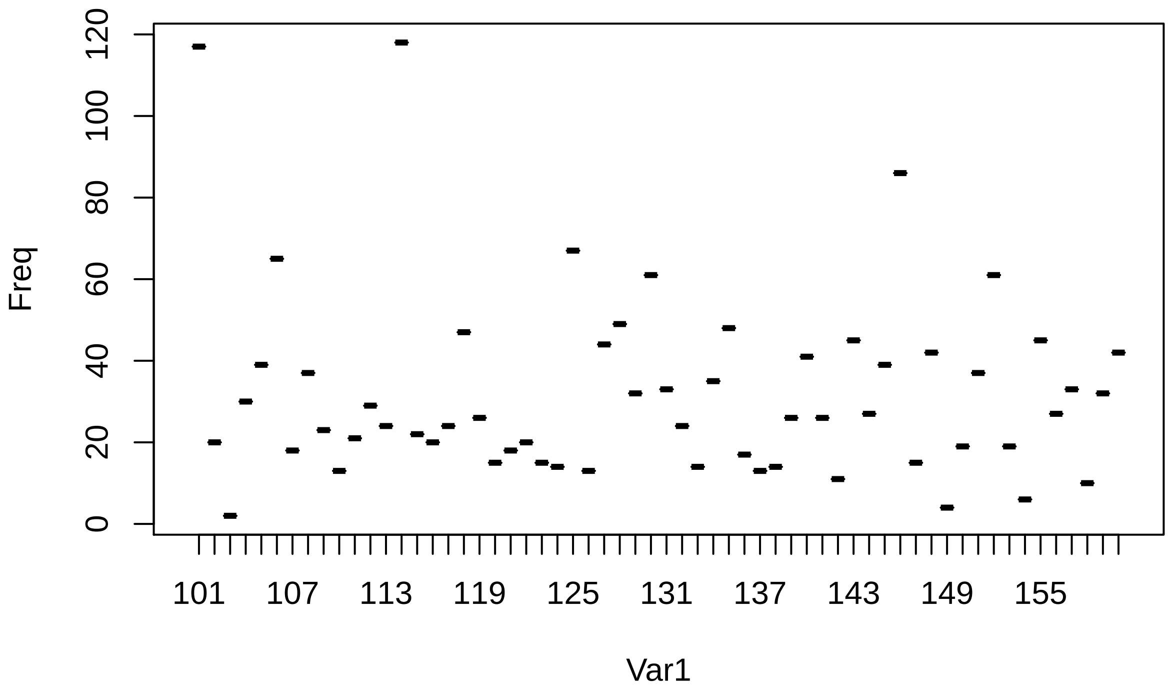


Since there is a large variabioity in the coefficients of the unpooled model but very small variability in the coefficients of the hierarchical model, which are centered around 0, we zoom in to the range of (-2, 2) for the coefficients axis to take a better look at the variation in hierarchical model coefficients.

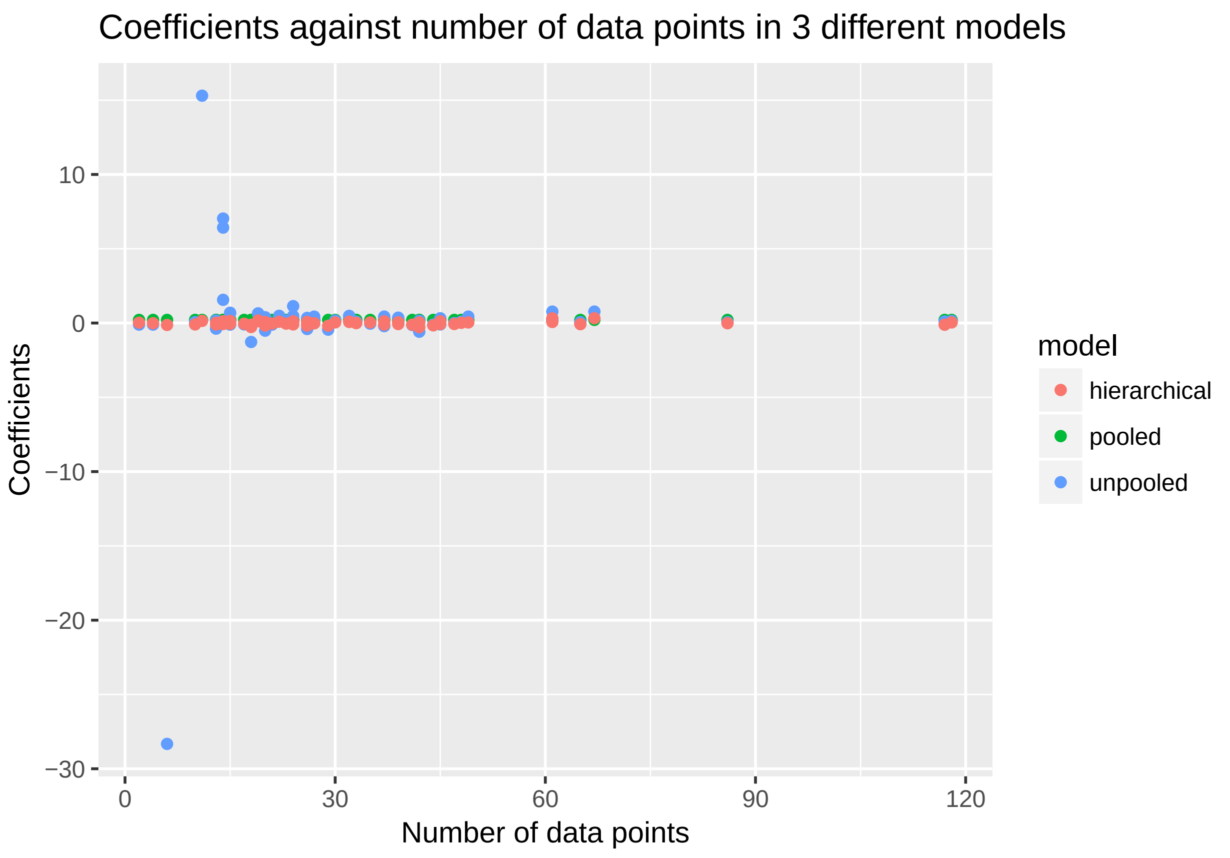
(b) Write a short summary (300 words or less) that interprets the graph from part (a). Pay particular attention to the relationship between the coefficients within each district, and how or whether the number of observations within each district plays a role in the relationship. You may speculate on the reasons for what you are seeing.

Answer

```
# plot of number of data points availabe in each district
district.counts <- data.frame(table(df$district))
plot(district.counts)
```



```r
# dataframe containing the coefficient values from
# the 3 models and the number of datapoints by district
df.coef_counts <- data.frame(districts = c(seq(101, 160), seq(101, 160), seq(101, 160)),
            coef = c(rep(coef.pooled, 60), 1.064e-01, coef.unpooled, coef.hierarchical),
            counts = c(district.counts[[2]], district.counts[[2]],district.counts[[2]]),
            model = c(rep('pooled', 60), rep('unpooled', 60), rep('hierarchical', 60)))

library(ggplot2)
ggplot(df.coef_counts, aes(x = counts, y = coef, color = model))+
  geom_point()+
  labs(x = 'Number of data points', y = 'Coefficients',
       title = 'Coefficients against number of data points in 3 different models')
```



Answer

From the plot in part a, we see that there is a large variability in the coefficients of the unpooled model, very small variability in the coefficients of the hierarchical model, and no variability in the coefficients of the pooled model.

The hierarchical model is usually used when data are strutured in groups, such as in districts in this case. It assumes that all districts behave in a similar way, but each district may vary by having some random noise, and each district's noise is a random draw from the same distribution of noises. We can think of the distribution as a rubber band around the coefficients, beta_i, for each district, the betas can vary, but the random effects distribution keeps them from being too far apart. This results in the coefficients having a very small variability. The unpooled model, on the other hand, assumes that every district is independent, and we are effectively fitting an independent model to each district. The coefficients do not follow any distribution and thus the variability is large in the coefficients for different districts. In the pooled model, we are ignoring the differences between districts and fitting one model for all the data points. Thus, the coefficient is one and the same for all districts.

In terms of the number of data points available, we notice from the figure above that for districts with fewer data points, the variability of coefficients in the unpooled model is very high. However, there is little variability in the coefficients of the hierarchical model even when only a few data points are present for some districts. This is because the random effects distribution of the hierarchical model is shrinking the coefficients to a common population mean. This effect is particularly noticeable when the district has a small number of data points.

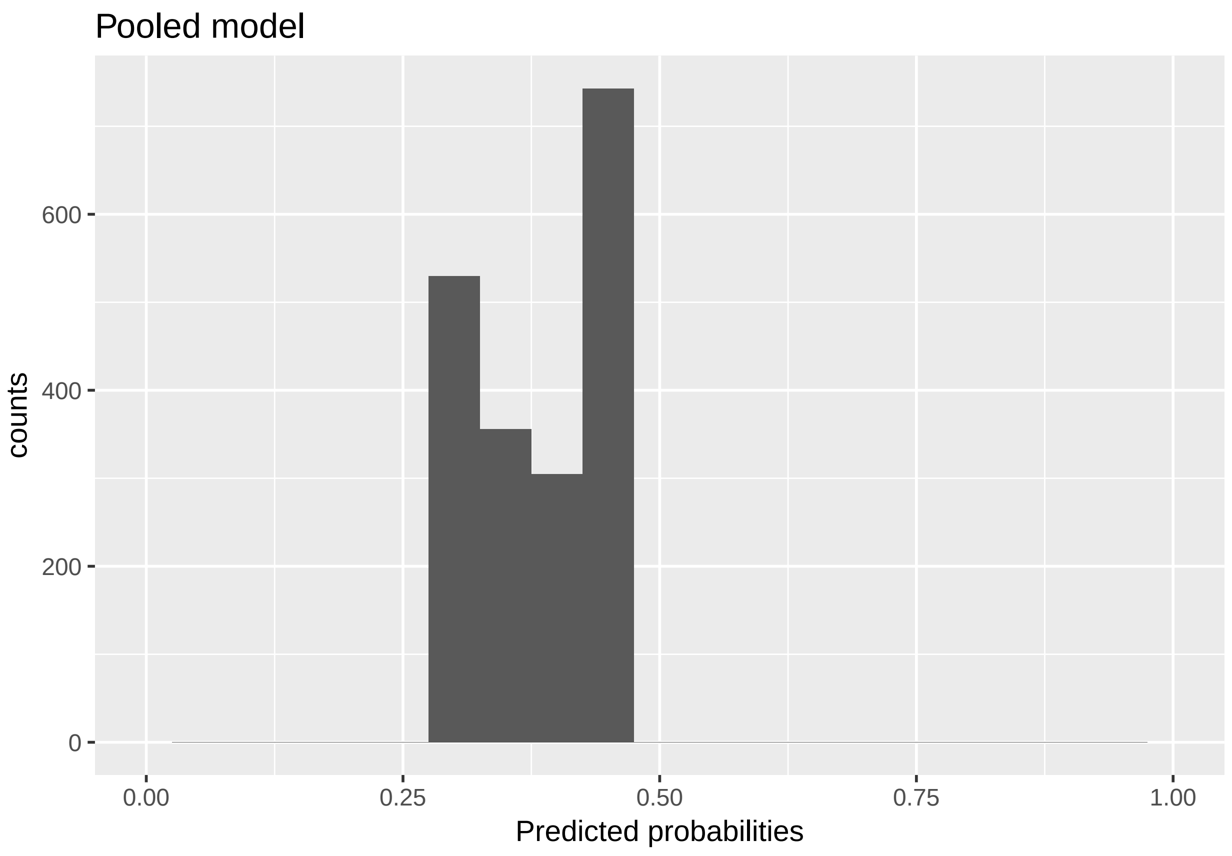
To summarize, the best model is the hierarchical model, which is able to take into account the behavior of contraceptive use in the entire population, at the same time recognizing the differences in behavior within each district. The unpooled model is the worst as it has too much variability in the coefficients, and does not take into consideration the common population behavior across districts.

3. Another benefit of shrinkage is how it affects probability estimates (recall the lucky, drunk friend from lecture whose classical estimate for the probability of guessing correctly was 100%). Extract the estimated probabilities from each model applied to the training data. That is, for the pooled and unpooled analyses, use the `predict` function applied to the fitted object, using the argument `type="response"`. For the hierarchical model, the `$theta.pred` component of the fitted model contains the estimated probabilities.

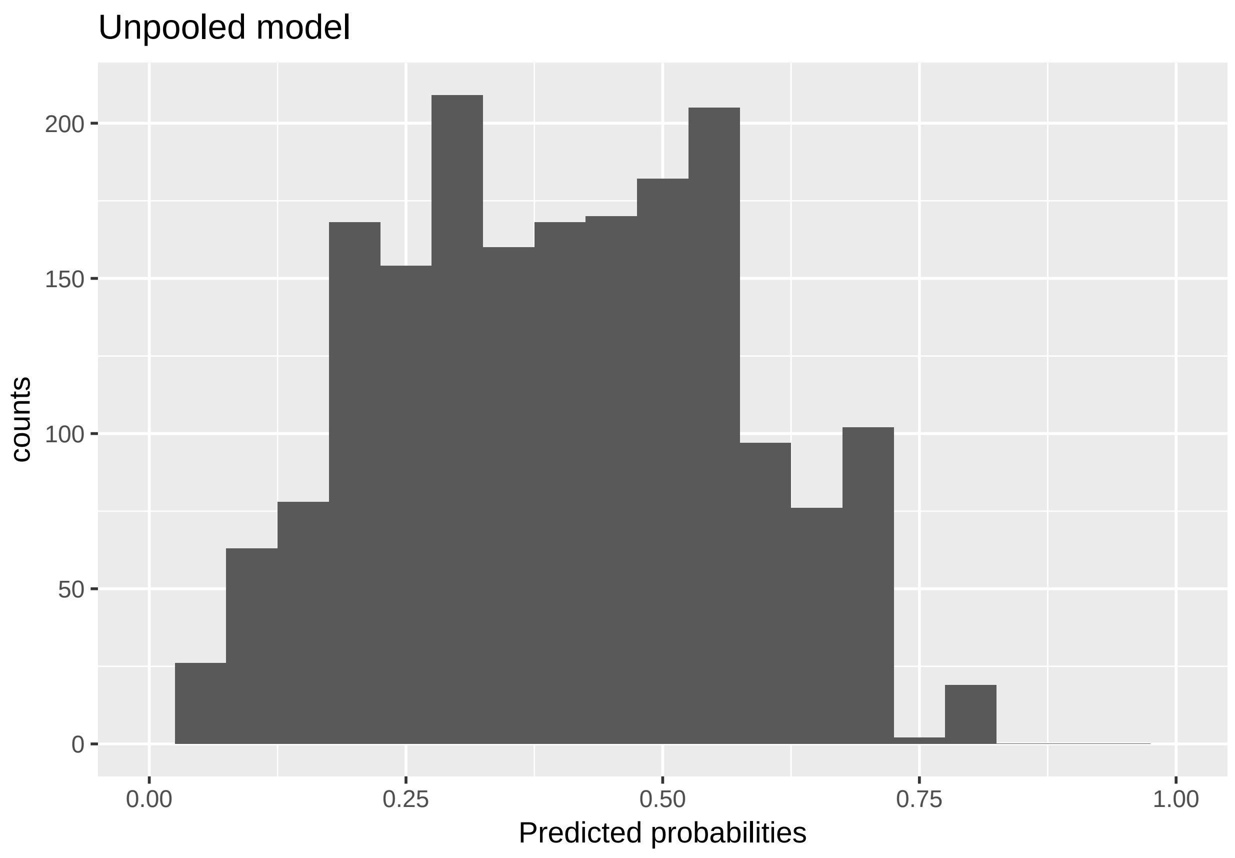
   (a) Plot histograms of the vectors of probability estimates for each model separately. Make sure you standardize the horizontal axis so that the scales are the same. How does the distribution of estimated probabilities compare across the three models?

Answer

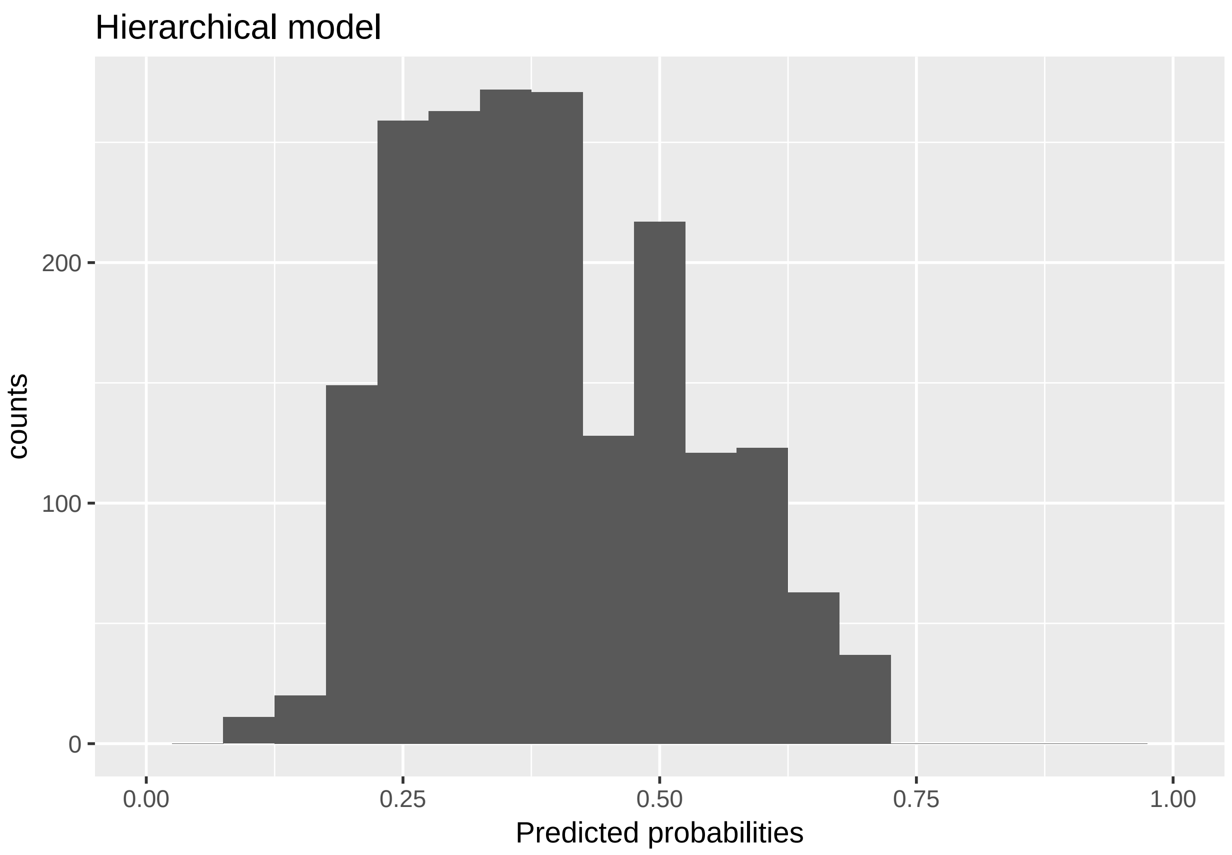
```r
pred.pooled <- data.frame(prob = as.numeric(predict(mod.pooled, type = "response")))
pred.unpooled <- data.frame(prob = as.numeric(predict(mod.unpooled, type = "response")))
pred.hierarchical <- data.frame(prob = mod.hierarchical$theta.pred)

ggplot(pred.pooled, aes(x = prob))+
  geom_histogram(binwidth = 0.05)+
  xlim(0, 1)+
  labs(x = 'Predicted probabilities', y = 'counts',
       title = 'Pooled model')
```



```r
ggplot(pred.unpooled, aes(x = prob))+
  geom_histogram(binwidth = 0.05)+
  xlim(0, 1)+
  labs(x = 'Predicted probabilities', y = 'counts',
       title = 'Unpooled model')
```



```
ggplot(pred.hierarchical, aes(x = prob))+
  geom_histogram(binwidth = 0.05)+
  xlim(0, 1)+
  labs(x = 'Predicted probabilities', y = 'counts',
       title = 'Hierarchical model')
```

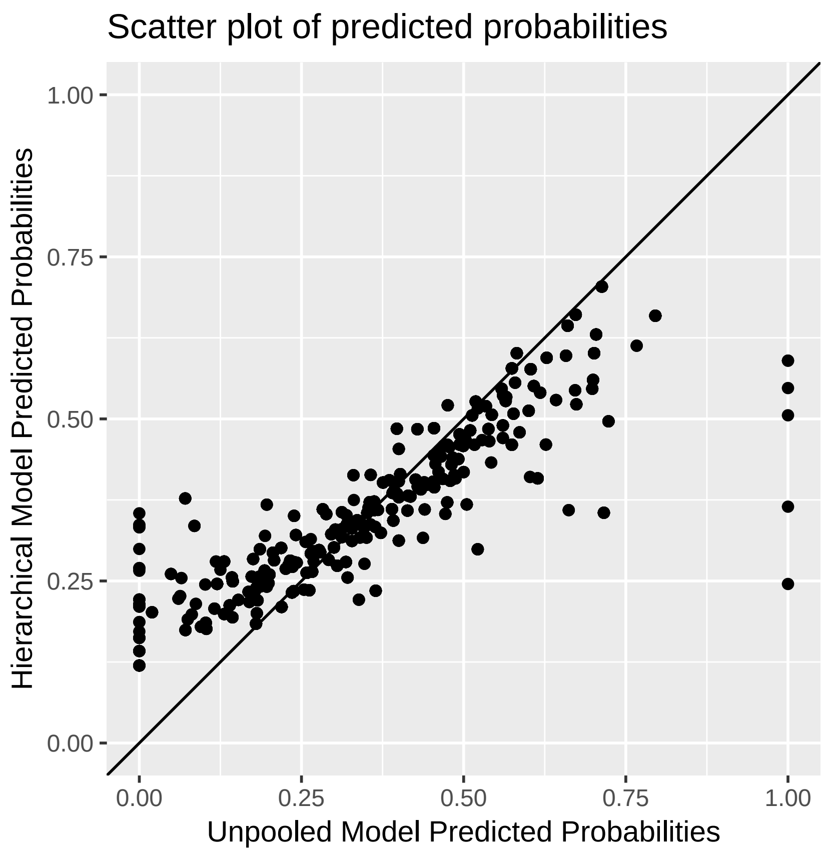


The distribution of probabilities for the pooled model is very narrow. The distributions of prediction probabilities for the unpooled model and hierarchical model are much wider in comparison. However, we notice that the unpooled model has a wider distribution of probabilities with extreme values, but the hierarchical model has a shinked distribution of probabilities, with less extreme values. This is a result of the aforementioned shrinkage effect of the hierarchical model as it shrinks coefficients towards a common population mean.

(b) Create a scatter plot comparing predicted values from Unpooled and Hierarchical Models, making sure that the scale of the horizontal and vertical axes are the same, and that the plotting region is square rather than rectangular. Include on the plot the line $y = x$ (why do you think this is a useful line to superimpose?). Briefly interpret the relationship between the probability estimates for these two models. Are there particular features of the plot that highlight the intended benefits of using a hierarchical model over the unpooled analysis? Briefly explain.

Answer

```r
pred.prob3 <- data.frame(
  unpooled <- as.numeric(predict(mod.unpooled, type = "response"),
  hierarchical <- mod.hierarchical$theta.pred))

ggplot(pred.prob3, aes(x = unpooled, y = hierarchical))+
  geom_point()+
  xlim(0, 1)+
  ylim(0, 1)+
  coord_fixed(ratio=1)+
  geom_abline(intercept = 0, slope = 1)+
  labs(x="Unpooled Model Predicted Probabilities",
       y="Hierarchical Model Predicted Probabilities",
      title = "Scatter plot of predicted probabilities ")
```



This is a scatter plot of predicted probabilities by the unpooled and the hierarchical models. Points that lie on the y = x line refers to those points predicted to have the same probability by both the unpooled and the hierarchical model. We notice that the unpooled model has a wider spread of probabilities compared to the hierarchical model. This is another way to visualize the large variability of the unpooled model, which resulted in large variability of coefficients, and hence predicted probabilities. The smaller spread of predicted probabilities of the hierarchical model is a way to visualize the shinkage to mean inherent to the model. The shrinkage to mean is indeed one intended benefits of the hierarchical model.

Problem 3: AWS Preparation

In prepartion for the upcoming Spark and Deep Learning modules, submit your AWS account information. This should have been created in Homework 0. We need specifically:

- The email address associated with your AWS account
- The email address associated with your Harvard ID, if different from above
- Your AWS ID. This should be a 10 digit number. (Instructions)

We need this information to enable GPU capable compute instances.

- danqing@seas.harvard.edu
- danqing@seas.harvard.edu
- 801484355261