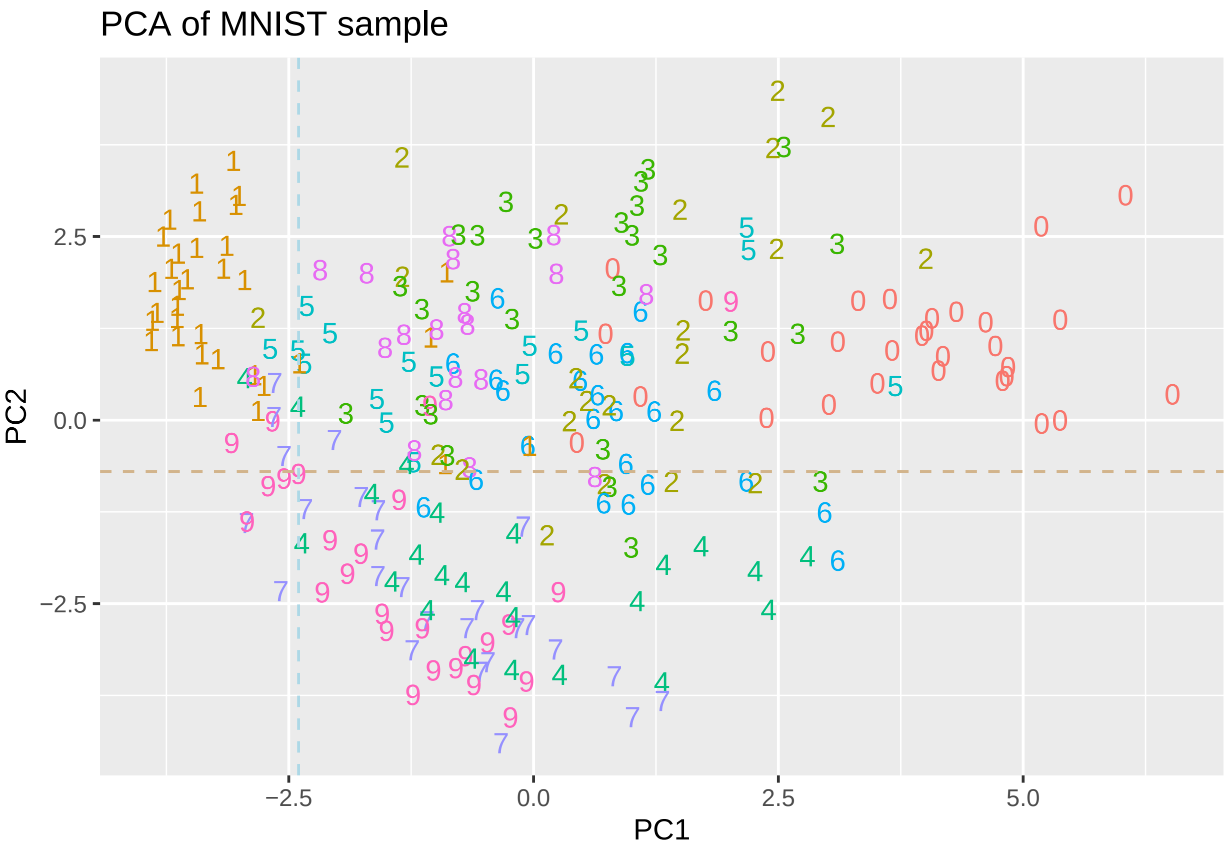
Experiment with PCA on the MNIST dataset

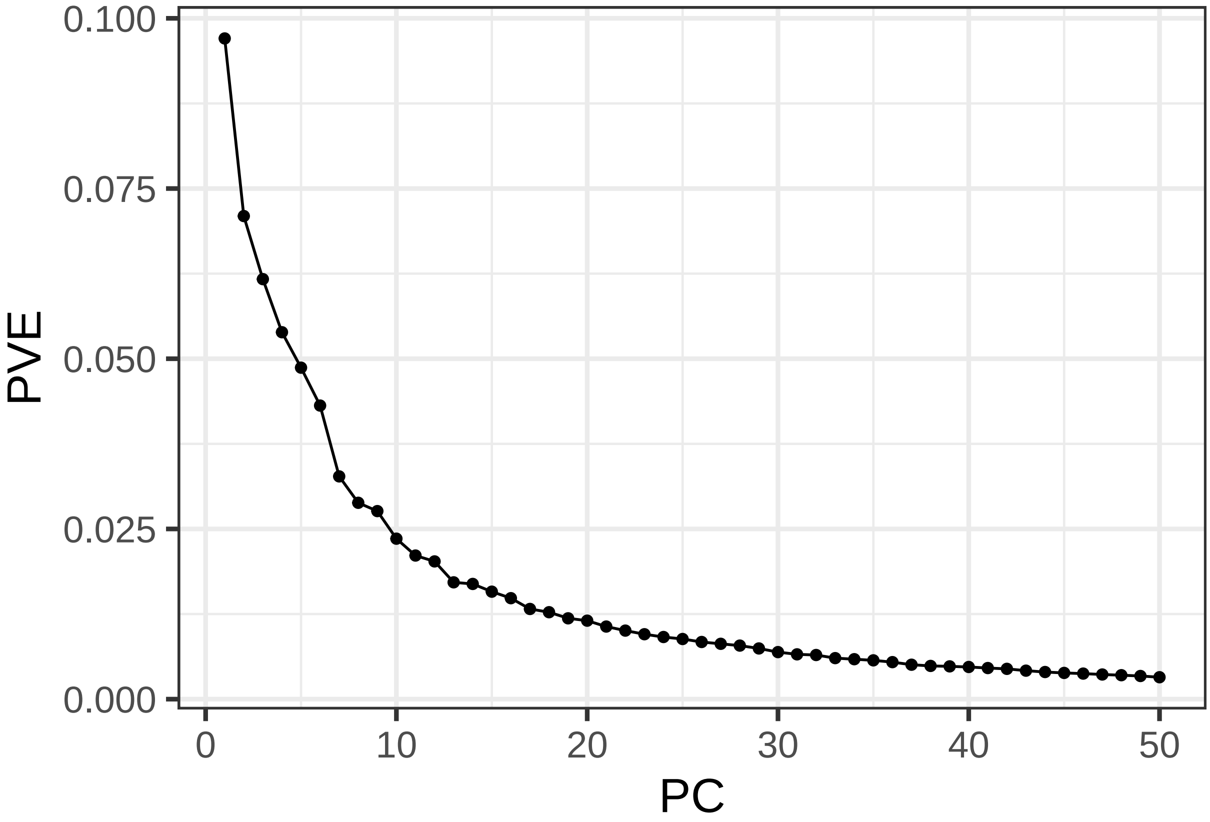
```r
# Fit PCA on the training dataset
pca <- prcomp(train_df[, -1])

# Fit PCA on the test dataset
test_pca <- predict(pca, newdata = test_df)
```

```r
# Store the first two coordinates and the label in a data frame
pca_plot <- data.frame(PC1 = pca$x[, "PC1"], PC2 = pca$x[, "PC2"],
                       label = as.factor(train_df$label))

# Plot the first two principal components using the true labels as color
g <- ggplot(pca_plot[1:250,], aes(x = PC1, y = PC2, color = label)) +
    ggtitle("PCA of MNIST sample") +
    geom_text(aes(label = label)) +
    theme(legend.position = "none")
```

```r
g + geom_vline(xintercept = -2.4, col = "lightblue", linetype="dashed") +
  geom_hline(yintercept = -.7, col = "tan", linetype="dashed")
```



```r
d <- data.frame(PC = 1:784,
                PVE = pca$sdev^2 / sum(pca$sdev^2))

ggplot(d[1:50,], aes(x = PC, y = PVE)) +
  geom_line() +
  geom_point() +
  theme_bw(base_size = 18)
```



We only need 20 PCs to capture 90% of the variance in our dataset.

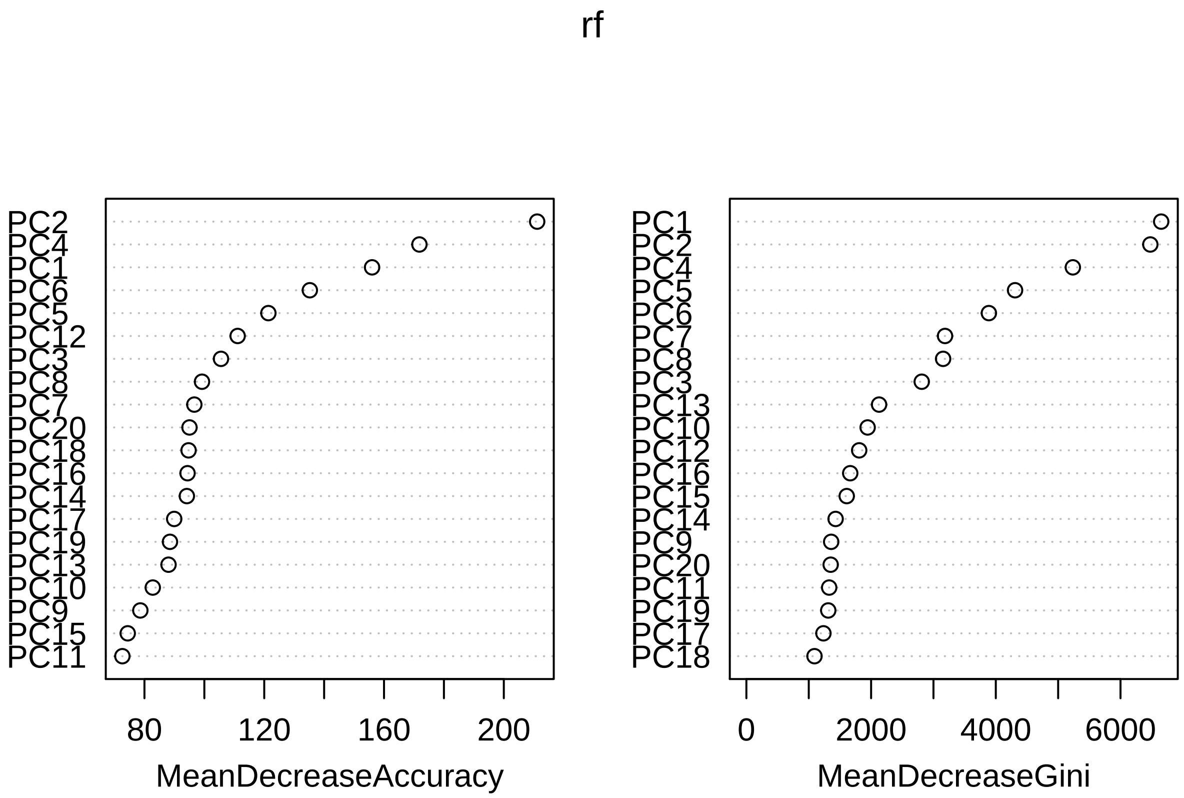
```
set.seed(1)

# select the first 20 PCs for the training dataset
pca.tr <- data.frame(label = train_df[, 1], pca$x[, 1:20])
pca.tr$label <- as.factor(pca.tr$label)

# select the first 20 PCs for the test dataset
pca.tst <- test_pca[, 1:20]
pca.tst <- data.frame(label = test_df$label, pca.tst)

pca.tst$label <- as.factor(pca.tst$label)
```

Random Forest

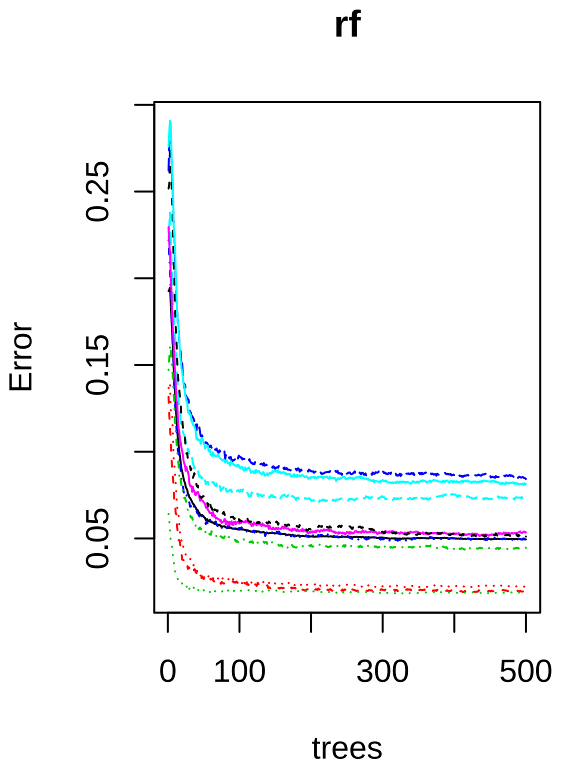
```
set.seed(1)

rf <- randomForest(pca.tr[, -1], pca.tr$label, ntree=500, importance = TRUE)
rf
```

```
##
## Call:
##  randomForest(x = pca.tr[, -1], y = pca.tr$label, ntree = 500,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 4.95%
## Confusion matrix:
##      0    1    2    3    4    5    6    7    8    9 class.error
## 0 5807    1   15    5   11    8   47    2   20    7  0.01958467
## 1    0 6616   40   21    9   11    9   11   19    6  0.01868882
## 2   36   10 5663   56   30    7   26   50   72    8  0.04951326
## 3    8    3   74 5678    3  100   20   58  137   50  0.07388680
## 4    8   23   23    4 5529    1   40   19   19  176  0.05357754
## 5   23    3   20   83   23 5144   47   11   39   28  0.05109758
## 6   30    6   11    2   14   53 5787    0   13    2  0.02213586
## 7    3   23   67    9   37   10    0 5986   16  114  0.04453312
## 8   12   38   53  170   27   93   28   15 5358   57  0.08425910
## 9   18   15   15   88  154   29    9  103   53 5465  0.08135821
```

```
par(mfrow = c(1,2))
varImpPlot(rf)
```



```
plot(rf)
```



```
pred.rf <- predict(rf, pca.tst, type = "class")
(conf.rf <- table(pred.rf, pca.tst$label))
```

```
##
## pred.rf    0    1    2    3    4    5    6    7    8    9
##       0  962    0   10    1    1    3    9    1    6    4
##       1    0 1123    1    0    2    1    4    4    0    7
##       2    4    2  976    8    3    5    1   19    7    1
##       3    0    2   11  954    0   16    0    2   20   11
##       4    0    0    5    0  921    6    4    5    8   28
##       5    4    1    2   14    4  845    5    1   19    8
##       6    8    4    4    0    7    6  934    1    5    2
##       7    1    0   10   10    3    1    0  969    5   11
##       8    1    2   13   20    5    6    1    2  896   10
##       9    0    1    0    3   36    3    0   24    8  927
```

```
(sum(conf.rf) - sum(diag(conf.rf))) /
  sum(conf.rf)
```

```
## [1] 0.0493
```

The misclassification rate is 4.89%. The pair that is most difficult to predict are 4 and 9.

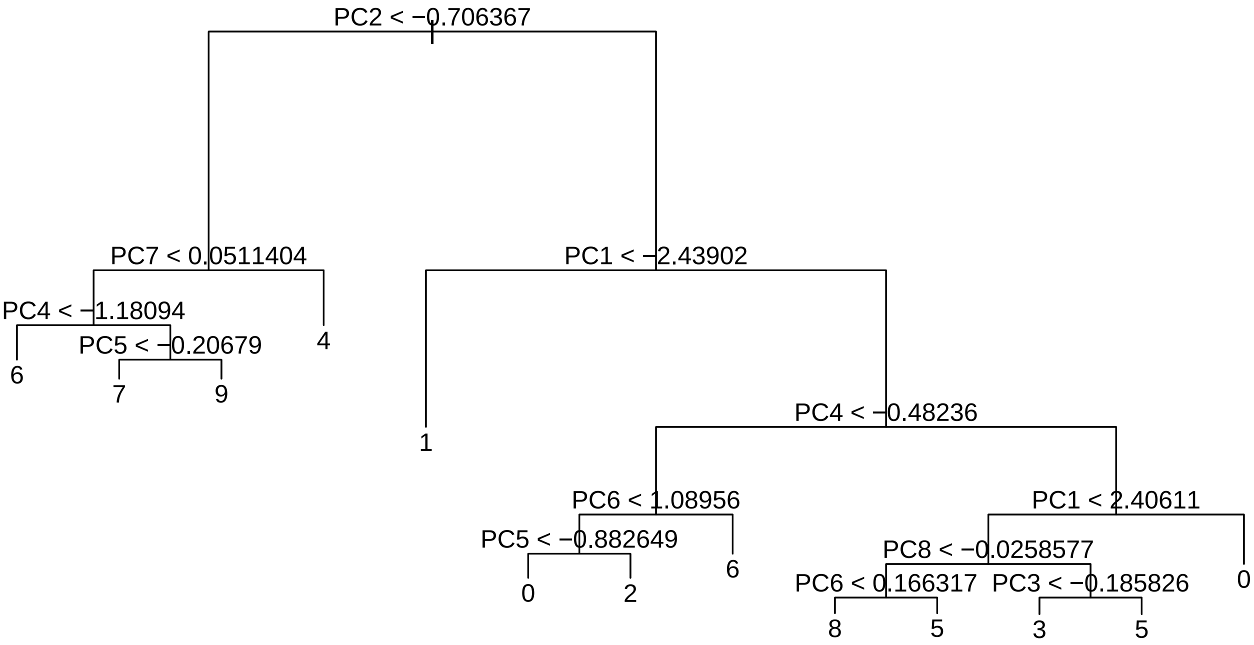
Classification Tree

```
t <- tree(label ~., data = pca.tr, split = "deviance")
summary(t)
```

```
##
## Classification tree:
```

```
## tree(formula = label ~ ., data = pca.tr, split = "deviance")
## Variables actually used in tree construction:
## [1] "PC2" "PC7" "PC4" "PC5" "PC1" "PC6" "PC8" "PC3"
## Number of terminal nodes:  13
## Residual mean deviance:  2.338 = 140200 / 59990
## Misclassification error rate: 0.3579 = 21474 / 60000
```

```
plot(t)
text(t, pretty = 0)
```



```
pred.tree <- predict(t, newdata = pca.tst, type = "class")
(conf.tree <- table(pred.tree, test_df$label))
```

```
##
## pred.tree    0    1    2    3    4    5    6    7    8    9
##         0  747    0   66   99    1  163   39    3   78   10
##         1    0 1055   11   11   26   13   10   53    8   30
##         2   26   24  705   26   10   42   82   20   85    4
##         3   10    4   27  601    2   52    1    0   49    5
##         4   17    0   37    9  820  112   35  101   37  572
##         5   77   41   42  164   15  388   83   18  169   18
##         6   53   11   57   37   32   33  706   13   21   30
##         7   21    0    4    5    7   18    0  592    4   38
##         8   14    0   75   47    6   67    1   30  452   11
##         9   15    0    8   11   63    4    1  198   71  291
```

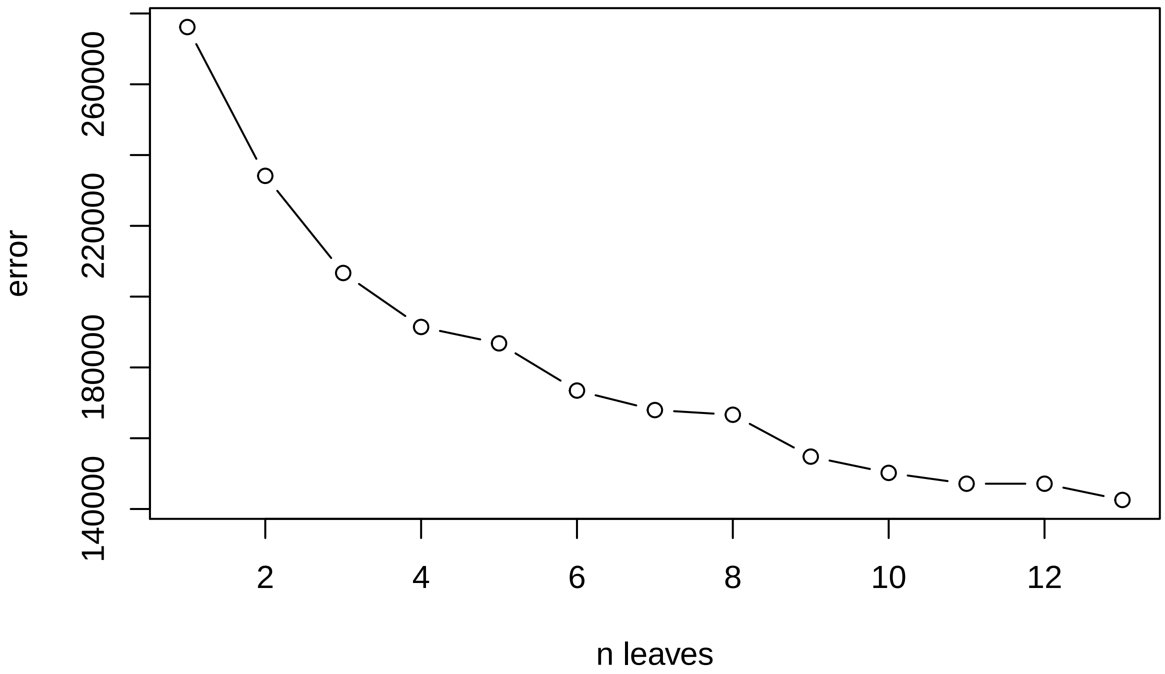
```
(sum(conf.tree) - sum(diag(conf.tree))) /
  sum(conf.tree)
```

```
## [1] 0.3643
```

4-9 is still the most difficult pair to predict, followed closely by 5-0, 7-9, 5-3, 5-8.

Pruning tree

```
t.cv <- cv.tree(t)
plot(t.cv$size, t.cv$dev, type = "b", xlab = "n leaves", ylab = "error")
```



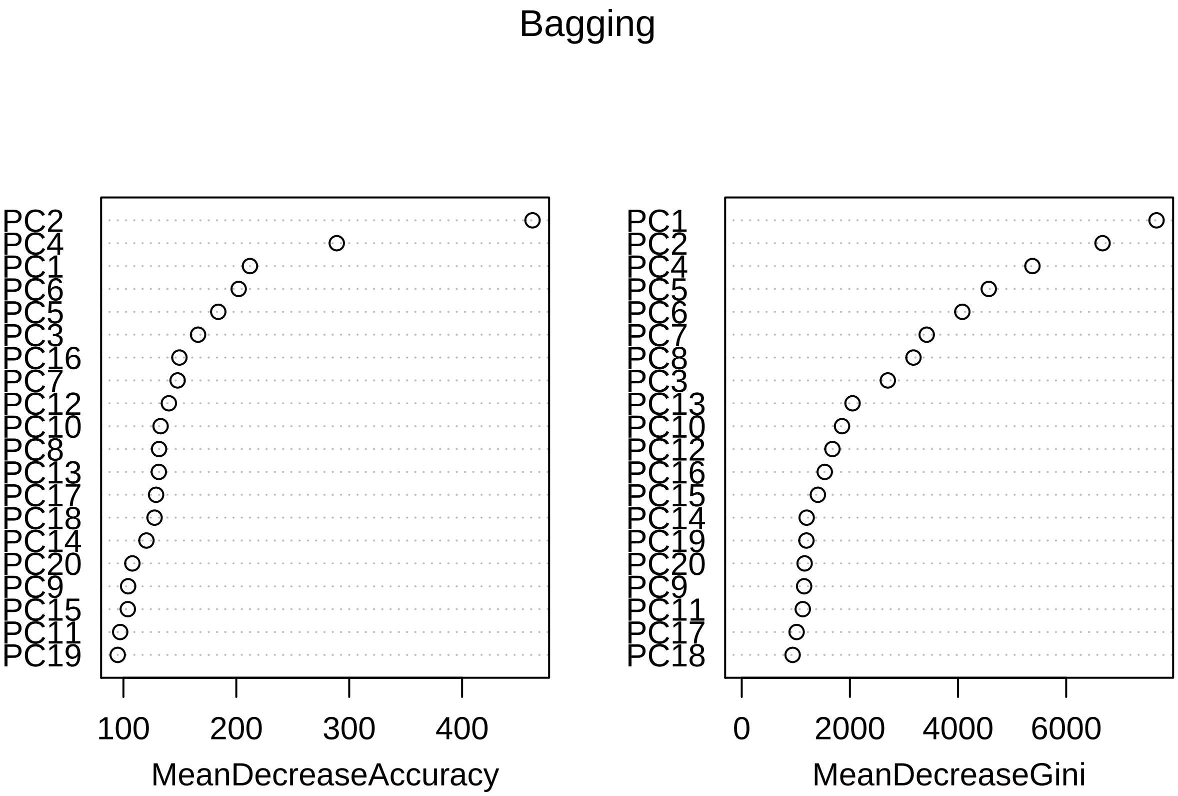
Not a good case for pruning (best n = 13 was already chosen).

Bagging

```
set.seed(1)

p <- ncol(pca.tr)-1
rf.bag <- randomForest(label ~., data = pca.tr,
                mtry = p/3, importance = TRUE)
rf.bag
```

```
##
## Call:
##  randomForest(formula = label ~ ., data = pca.tr, mtry = p/3,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 5.27%
## Confusion matrix:
##      0    1    2    3    4    5    6    7    8    9 class.error
## 0 5796    0   12    8   11   15   47    6   17   11  0.02144184
## 1    0 6611   38   18    7   14   14    9   21   10  0.01943044
## 2   39   10 5637   54   44    9   28   52   72   13  0.05387714
## 3   11    6   83 5656    4  109   19   60  136   47  0.07747513
## 4    7   21   32    6 5495    2   40   28   24  187  0.05939747
## 5   27    5   16   88   30 5107   45   12   48   43 0.05792289
## 6   27    6   17    2   16   54 5783    0   10    3 0.02281176
## 7    6   23   65   10   41   12    0 5970   19  119 0.04708699
## 8   13   41   52  164   28  110   26   20 5338   59 0.08767732
## 9   20   14   14   91  148   34   11  115   58 5444 0.08488822
```

```
varImpPlot(rf.bag, main="Bagging")
```



```
pred.bag <- predict(rf.bag, newdata = pca.tst, type = "class")
(conf.bag <- table(pred.bag, pca.tst$label))
```

```
##
## pred.bag    0    1    2    3    4    5    6    7    8    9
##        0  960    0    9    1    0    3    9    1    7    4
##        1    0 1122    1    0    1    2    3    4    0    8
##        2    4    1  981    9    6    4    2   19   10    2
##        3    0    2   11  951    0   13    0    2   21    9
##        4    0    0    5    0  916    8    4    6    9   31
##        5    4    2    2   16    3  842    4    2   20    7
##        6    9    5    2    1    9    6  934    0    5    1
##        7    2    0    7    9    4    1    0  968    5   12
##        8    1    3   13   20    7    8    2    2  888   11
##        9    0    0    1    3   36    5    0   24    9  924
```

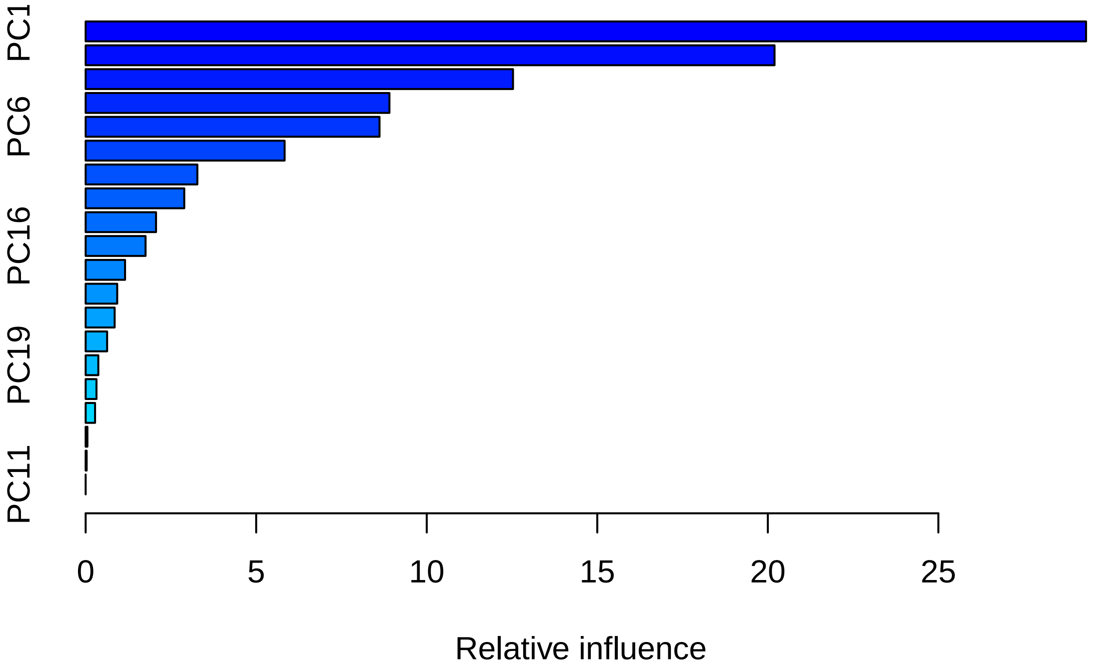
```
(sum(conf.bag) - sum(diag(conf.bag))) /
  sum(conf.bag)
```

```
## [1] 0.0514
```

The misclassification rate is 5.15%.

Boosting tree

```
set.seed(1)

boost.mnist <- gbm(label~., data = pca.tr,
                   distribution = "multinomial",
                   n.trees = 50, interaction.depth = 1,
                   shrinkage = 0.1)
summary(boost.mnist)
```



```
##       var     rel.inf
## PC1   PC1 29.32983526
## PC2   PC2 20.19729124
## PC4   PC4 12.52937052
## PC5   PC5  8.90577175
## PC6   PC6  8.61493591
## PC7   PC7  5.83248543
## PC8   PC8  3.27468586
## PC3   PC3  2.89073995
## PC13 PC13  2.06341543
## PC16 PC16  1.75585198
## PC9   PC9  1.15496116
## PC20 PC20  0.92517800
## PC12 PC12  0.85059045
## PC15 PC15  0.62954685
## PC19 PC19  0.37121670
## PC14 PC14  0.31766673
## PC10 PC10  0.27587466
## PC17 PC17  0.05185516
## PC18 PC18  0.02872696
## PC11 PC11  0.00000000
```

```r
pred.boost <- predict(boost.mnist, newdata = pca.tst, n.trees = 50)
pred.boost <- apply(pred.boost, 1, which.max) -1

(conf.boost <- table(pred.boost, pca.tst$label))
```

```
##
## pred.boost    0    1    2    3    4    5    6    7    8    9
##          0  862    0   27   14    2   37   21    7   32   12
##          1    0 1067    3    7   21    6    4   41    3   28
##          2   17    7  781   17   16   31   53   30   25   11
##          3   11    6   41  805    1  119    7    2   72   13
##          4    2    0   15    9  721   44   10   10   10  158
##          5   42    8   14   59   14  571   50   11   44   18
##          6   24   20   49   37   45   39  786    2    6   11
##          7   10    3   20   12   21   25    9  825   14   46
##          8    3   24   72   41   22   14   10   40  745   17
##          9    9    0   10    9  119    6    8   60   23  695
```

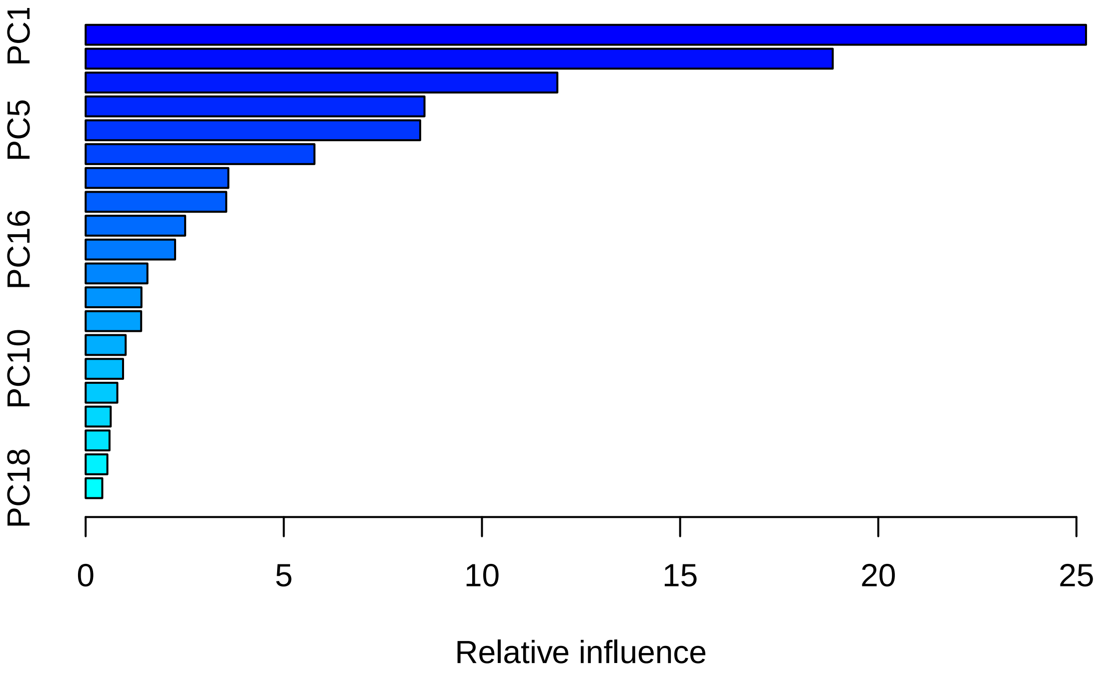
```r
(sum(conf.boost) - sum(diag(conf.boost))) /
  sum(conf.boost)
```

```
## [1] 0.2142
```

Using 5-fold CV

PCA Dataset

```r
boost.pca.cv <- gbm(label~., data = pca.tr,
                distribution = "multinomial",
                n.trees = 500,
                interaction.depth = 1,
                shrinkage = 0.1,
                cv.folds = 5
                )
summary(boost.pca.cv)
```



```
##       var    rel.inf
## PC1   PC1 25.2419322
## PC2   PC2 18.8506391
## PC4   PC4 11.9026076
## PC6   PC6  8.5498507
## PC5   PC5  8.4408193
## PC7   PC7  5.7728162
## PC3   PC3  3.6008571
## PC8   PC8  3.5477115
## PC13 PC13  2.5109315
## PC16 PC16  2.2569362
## PC12 PC12  1.5587557
## PC20 PC20  1.4083821
## PC9   PC9  1.4007824
## PC15 PC15  1.0100330
## PC10 PC10  0.9439162
## PC19 PC19  0.7995447
## PC17 PC17  0.6318926
## PC14 PC14  0.6027056
## PC11 PC11  0.5486264
## PC18 PC18  0.4202600
```

```r
print(boost.pca.cv)
```

```
## gbm(formula = label ~ ., distribution = "multinomial", data = pca.tr,
##     n.trees = 500, interaction.depth = 1, shrinkage = 0.1, cv.folds = 5)
## A gradient boosted model with multinomial loss function.
## 500 iterations were performed.
## The best cross-validation iteration was 413.
## There were 20 predictors of which 20 had non-zero influence.
```

The best number of trees chosen by the boosted model using 5-fold CV on the PCA dataset is 411.

```
pred.boost.cv <- predict(boost.pca.cv, newdata = pca.tst, n.trees = 500)
pred.boost.cv <- apply(pred.boost.cv, 1, which.max) -1

(conf.boost <- table(pred.boost.cv, pca.tst$label))
```

```
##
## pred.boost.cv    0    1    2    3    4    5    6    7    8    9
##             0  943    0   12    6    2   10    9    4    9    8
##             1    0 1113    3    1    6    4    4   13    1    7
##             2    8    5  896   17    6   10    3   37    9    7
##             3    2    3   19  886    1   46    1    3   33   15
##             4    0    1   12    3  882   15   16    6   10   60
##             5    8    2    9   38    4  745   20    2   26   13
##             6   11    3   25    3   11   22  902    0   11    3
##             7    1    2   17   13    2   10    0  920    8   23
##             8    6    6   30   32   14   20    3    6  850   13
##             9    1    0    9   11   54   10    0   37   17  860
```

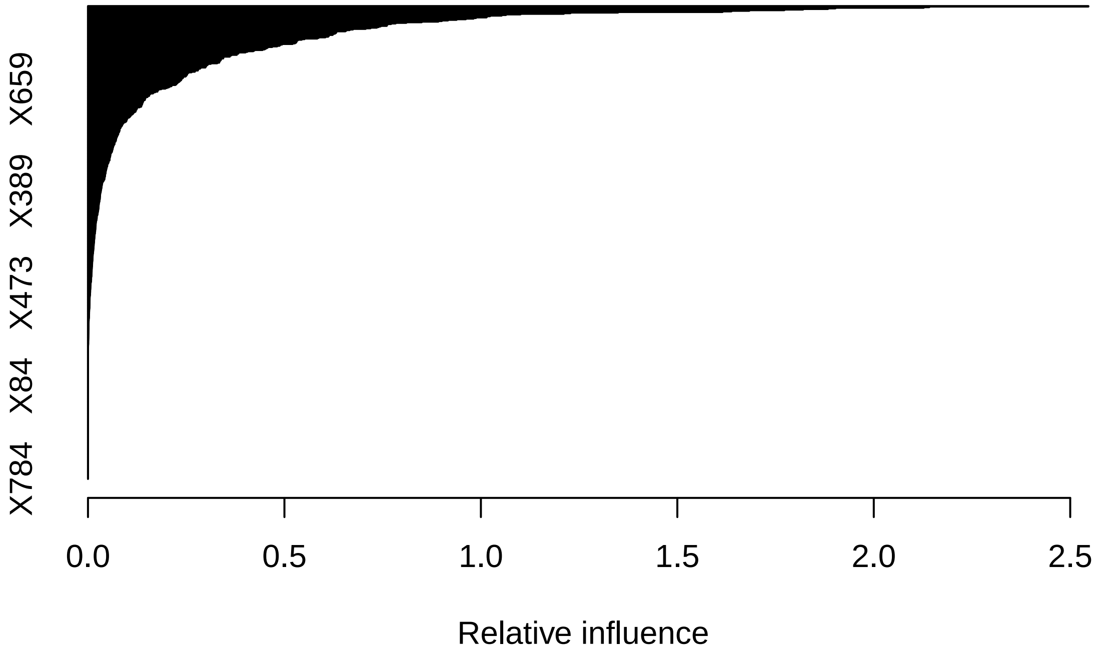
```
(sum(conf.boost) - sum(diag(conf.boost))) /
  sum(conf.boost)
```

```
## [1] 0.1003
```

The misclassification rate is 9.81. 4-9 is still the most difficult pair to predict.

Original Dataset

```
boost.og.cv <- gbm(label~., data = train_df,
                distribution = "multinomial",
                n.trees = 500,
                interaction.depth = 1,
                shrinkage = 0.1,
                cv.folds = 5)
summary(boost.og.cv)
```



```
##       var      rel.inf
## X351 X351 2.5462195500
## X212 X212 2.1413055303
## X359 X359 2.1271324193
## X436 X436 1.9022945603
## X387 X387 1.8833568053
## X101 X101 1.8199774890
## X408 X408 1.7723903032
## X438 X438 1.6830568348
## X406 X406 1.6376548168
## X515 X515 1.6150122829
## X221 X221 1.3490162999
## X324 X324 1.2278729872
## X102 X102 1.2110645172
## X103 X103 1.1007288568
## X584 X584 1.0638216664
## X409 X409 1.0528091151
## X713 X713 1.0232646391
## X347 X347 1.0154425547
## X407 X407 1.0143286990
## X490 X490 0.9874882352
## X743 X743 0.9812381751
## X375 X375 0.9615574067
## X740 X740 0.9381115000
## X658 X658 0.9161566260
## X711 X711 0.9006404310
## X125 X125 0.8915762829
## X429 X429 0.8493568557
## X718 X718 0.8101597781
## X70   X70 0.7828783938
## X410 X410 0.7713370658
## X712 X712 0.7624657721
## X516 X516 0.7606544107
## X435 X435 0.7604432822
## X243 X243 0.7445896590
## X71   X71 0.7397610636
## X742 X742 0.7351344810
## X379 X379 0.7188744265
## X156 X156 0.7066873849
## X491 X491 0.6742048185
## X744 X744 0.6660141750
## X544 X544 0.6574411347
## X72   X72 0.6551371067
## X599 X599 0.6324701837
## X211 X211 0.6318856758
## X327 X327 0.6307862514
## X236 X236 0.6283033734
## X488 X488 0.6239195316
## X383 X383 0.6225974530
## X381 X381 0.6129332069
## X463 X463 0.6127933782
## X512 X512 0.6111551661
## X348 X348 0.6041376830
## X192 X192 0.5868006510
## X104 X104 0.5836730854
## X401 X401 0.5510722552
## X151 X151 0.5437574349
## X157 X157 0.5340833043
## X405 X405 0.5318952013
## X350 X350 0.5305440279
## X541 X541 0.5304576524
## X434 X434 0.5292355934
## X278 X278 0.5265053990
## X271 X271 0.5207560090
## X714 X714 0.4939556366
## X657 X657 0.4890541276
## X377 X377 0.4875239834
## X291 X291 0.4813662017
## X250 X250 0.4688392735
## X318 X318 0.4575462812
## X433 X433 0.4552935090
## X177 X177 0.4521001899
## X349 X349 0.4494385473
## X571 X571 0.4442720765
## X378 X378 0.4233693561
## X522 X522 0.4206022506
## X328 X328 0.4056476463
## X719 X719 0.4008940589
## X178 X178 0.3830110808
## X518 X518 0.3825774523
## X403 X403 0.3788710169
## X489 X489 0.3787705775
## X360 X360 0.3646052208
## X539 X539 0.3612972640
## X570 X570 0.3609052955
## X583 X583 0.3450153952
## X517 X517 0.3443075164
## X264 X264 0.3441915927
## X428 X428 0.3393101607
## X598 X598 0.3379119582
## X466 X466 0.3378652401
## X376 X376 0.3361110305
## X430 X430 0.3350466002
## X745 X745 0.3340418705
## X487 X487 0.3339480602
## X152 X152 0.3288064263
## X105 X105 0.3123401221
## X127 X127 0.3054142968
## X465 X465 0.3039435168
## X432 X432 0.3009823986
## X557 X557 0.3008952899
## X244 X244 0.2992021139
## X126 X126 0.2987472409
## X184 X184 0.2864676911
## X373 X373 0.2851239538
## X431 X431 0.2805863449
## X240 X240 0.2797078165
## X193 X193 0.2796460877
## X721 X721 0.2723615615
## X568 X568 0.2719343194
## X738 X738 0.2634017062
## X710 X710 0.2556179891
## X67   X67 0.2530605024
## X277 X277 0.2527016756
## X259 X259 0.2521740608
## X543 X543 0.2504766987
## X382 X382 0.2499140209
## X484 X484 0.2481309801
## X437 X437 0.2425179529
## X208 X208 0.2420238580
## X709 X709 0.2406548749
## X179 X179 0.2384046126
## X319 X319 0.2371818706
## X550 X550 0.2370948734
## X249 X249 0.2351147751
## X323 X323 0.2332193397
## X269 X269 0.2317283939
## X722 X722 0.2290078002
## X502 X502 0.2275863185
## X69   X69 0.2249348443
## X623 X623 0.2248213421
## X415 X415 0.2223711696
## X346 X346 0.2133078589
## X124 X124 0.2122434827
## X464 X464 0.2079387316
## X285 X285 0.2052457194
## X68   X68 0.2010655041
## X402 X402 0.1969242636
## X659 X659 0.1861672493
## X456 X456 0.1798150683
## X358 X358 0.1781276675
## X404 X404 0.1769112438
## X519 X519 0.1766658280
## X320 X320 0.1709643779
## X106 X106 0.1667394891
## X564 X564 0.1652868923
## X321 X321 0.1585433682
## X611 X611 0.1564333268
## X190 X190 0.1558696657
## X345 X345 0.1556722837
## X268 X268 0.1521653145
## X716 X716 0.1512125192
## X708 X708 0.1465058420
## X573 X573 0.1462204920
## X355 X355 0.1449679183
## X306 X306 0.1449063823
## X715 X715 0.1445013592
## X163 X163 0.1411829864
## X100 X100 0.1404067935
## X183 X183 0.1401996137
## X660 X660 0.1388620648
## X582 X582 0.1382594821
## X530 X530 0.1381851207
## X235 X235 0.1377770983
## X297 X297 0.1367876826
## X299 X299 0.1360858812
## X529 X529 0.1356141180
## X191 X191 0.1352712482
## X653 X653 0.1313246170
## X593 X593 0.1252527003
## X442 X442 0.1244494061
## X440 X440 0.1240097883
## X260 X260 0.1232502324
## X334 X334 0.1214551011
## X213 X213 0.1213786055
## X298 X298 0.1212373294
## X107 X107 0.1191884154
## X128 X128 0.1155643440
## X737 X737 0.1155162087
## X542 X542 0.1140921341
## X551 X551 0.1123743512
## X746 X746 0.1092806092
## X427 X427 0.1089607665
## X494 X494 0.1073272352
## X272 X272 0.1063644922
## X380 X380 0.1045722182
## X292 X292 0.1008557095
## X462 X462 0.0999236237
## X185 X185 0.0992288767
## X258 X258 0.0980481035
## X118 X118 0.0979967713
## X153 X153 0.0974603178
## X98   X98 0.0963405988
## X628 X628 0.0934940607
## X538 X538 0.0898579831
## X600 X600 0.0896221287
## X276 X276 0.0877857344
## X275 X275 0.0872478049
## X486 X486 0.0868309510
## X566 X566 0.0857038750
## X565 X565 0.0848741758
## X540 X540 0.0832636382
## X513 X513 0.0828934659
## X460 X460 0.0819451622
## X66   X66 0.0804902709
## X344 X344 0.0804693044
## X109 X109 0.0804245205
## X652 X652 0.0802385070
## X475 X475 0.0792725371
## X467 X467 0.0792654730
## X329 X329 0.0772170911
## X610 X610 0.0771477582
## X99   X99 0.0769199883
## X386 X386 0.0767834132
## X96   X96 0.0754968083
## X720 X720 0.0750833958
## X470 X470 0.0739649875
## X206 X206 0.0737501885
## X388 X388 0.0725871875
## X536 X536 0.0723266471
## X209 X209 0.0717123991
## X207 X207 0.0713577295
## X474 X474 0.0711976081
## X214 X214 0.0708222176
## X232 X232 0.0704312142
## X317 X317 0.0697892904
## X629 X629 0.0677120909
## X574 X574 0.0674344887
## X150 X150 0.0674256619
## X175 X175 0.0674172238
## X356 X356 0.0667727888
## X483 X483 0.0652975452
## X129 X129 0.0651333315
## X77   X77 0.0639323379
## X210 X210 0.0635994036
## X662 X662 0.0635169177
## X622 X622 0.0629579338
## X569 X569 0.0625534089
## X108 X108 0.0623836511
## X205 X205 0.0617736288
## X625 X625 0.0611467940
## X303 X303 0.0608396231
## X305 X305 0.0607487727
## X274 X274 0.0597010596
## X576 X576 0.0587515211
## X302 X302 0.0573925515
## X353 X353 0.0570915368
## X372 X372 0.0568082932
## X508 X508 0.0566835217
## X545 X545 0.0566053302
## X290 X290 0.0562608350
## X439 X439 0.0556965985
## X362 X362 0.0553233483
## X330 X330 0.0550064358
## X174 X174 0.0549849852
## X458 X458 0.0548554247
## X158 X158 0.0547847321
## X747 X747 0.0529678356
## X352 X352 0.0526993804
## X65   X65 0.0523193676
## X332 X332 0.0519624552
## X263 X263 0.0510907738
## X417 X417 0.0501329398
## X204 X204 0.0489530414
## X202 X202 0.0489039238
## X181 X181 0.0488883508
## X696 X696 0.0488480033
## X706 X706 0.0480598781
## X457 X457 0.0472262825
## X239 X239 0.0469819854
## X579 X579 0.0468099579
## X284 X284 0.0464613842
## X655 X655 0.0458121938
## X391 X391 0.0457589951
## X270 X270 0.0450619760
## X419 X419 0.0449740766
## X215 X215 0.0448670074
## X159 X159 0.0447398665
## X374 X374 0.0443791771
## X603 X603 0.0438200347
## X368 X368 0.0434506008
## X304 X304 0.0431715279
## X241 X241 0.0431597899
## X189 X189 0.0430746259
## X572 X572 0.0424252637
## X537 X537 0.0420115114
## X242 X242 0.0416827318
## X635 X635 0.0416068922
## X400 X400 0.0414786787
## X524 X524 0.0401199340
## X248 X248 0.0393342160
## X496 X496 0.0388064946
## X295 X295 0.0368555225
## X492 X492 0.0363701429
## X639 X639 0.0362498454
## X390 X390 0.0354594835
## X201 X201 0.0354295276
## X354 X354 0.0353658628
## X609 X609 0.0347699219
## X707 X707 0.0347428220
## X122 X122 0.0344554151
## X340 X340 0.0341515903
## X717 X717 0.0340676066
## X333 X333 0.0337725137
## X164 X164 0.0333861618
## X155 X155 0.0333265904
## X649 X649 0.0326138911
## X389 X389 0.0325329196
## X219 X219 0.0323749971
## X468 X468 0.0322753480
## X511 X511 0.0315037724
## X370 X370 0.0310720054
## X267 X267 0.0310268803
## X630 X630 0.0308638370
## X301 X301 0.0306167910
## X459 X459 0.0305337887
## X679 X679 0.0304908544
## X694 X694 0.0302652625
## X723 X723 0.0301074509
## X176 X176 0.0300779085
## X581 X581 0.0299680980
## X371 X371 0.0299106328
## X601 X601 0.0290459289
## X94   X94 0.0290336930
## X661 X661 0.0288788044
## X95   X95 0.0288526436
## X296 X296 0.0278832339
## X585 X585 0.0277940600
## X361 X361 0.0275625890
## X447 X447 0.0274983461
## X343 X343 0.0272378305
## X312 X312 0.0270658936
## X606 X606 0.0270522486
## X510 X510 0.0269759915
## X552 X552 0.0268927171
## X229 X229 0.0265436495
## X416 X416 0.0265297517
## X705 X705 0.0261106404
## X697 X697 0.0260889358
## X597 X597 0.0258940560
## X656 X656 0.0254842157
## X683 X683 0.0251775083
## X509 X509 0.0250294791
## X396 X396 0.0246025670
## X322 X322 0.0245368827
## X748 X748 0.0241150464
## X363 X363 0.0240329036
## X461 X461 0.0236179409
## X369 X369 0.0223847389
## X677 X677 0.0223066313
## X154 X154 0.0222701258
## X680 X680 0.0221821824
## X203 X203 0.0221307653
## X724 X724 0.0220789352
## X136 X136 0.0214001871
## X161 X161 0.0213595262
## X595 X595 0.0211607272
## X148 X148 0.0201688743
## X580 X580 0.0201559215
## X681 X681 0.0200220654
## X307 X307 0.0197926835
## X326 X326 0.0196336075
## X678 X678 0.0194783082
## X695 X695 0.0194136429
## X331 X331 0.0193917409
## X634 X634 0.0192736346
## X750 X750 0.0191625103
## X546 X546 0.0188416047
## X339 X339 0.0188091791
## X97   X97 0.0187517069
## X341 X341 0.0186669757
## X640 X640 0.0185320198
## X480 X480 0.0183163685
## X446 X446 0.0182191417
## X602 X602 0.0182014811
## X233 X233 0.0175404326
## X485 X485 0.0175127106
## X663 X663 0.0174971399
## X578 X578 0.0174295418
## X633 X633 0.0167685763
## X689 X689 0.0167189672
## X693 X693 0.0166054718
## X93   X93 0.0163173407
## X123 X123 0.0162611837
## X452 X452 0.0161450232
## X495 X495 0.0161435933
## X650 X650 0.0159713420
## X135 X135 0.0158917654
## X289 X289 0.0155455822
## X325 X325 0.0155166505
## X418 X418 0.0154655680
## X247 X247 0.0154498665
## X149 X149 0.0152460504
## X666 X666 0.0149190278
## X451 X451 0.0147246702
## X638 X638 0.0146904584
## X608 X608 0.0145087756
## X222 X222 0.0144963481
## X300 X300 0.0144202894
## X64   X64 0.0140612673
## X612 X612 0.0140116037
## X234 X234 0.0139678245
## X357 X357 0.0138167484
## X399 X399 0.0136973859
## X137 X137 0.0134956376
## X245 X245 0.0132944610
## X76   X76 0.0132908280
## X521 X521 0.0131267454
## X424 X424 0.0128362172
## X607 X607 0.0126648345
## X384 X384 0.0126446510
## X664 X664 0.0123678227
## X554 X554 0.0120520436
## X684 X684 0.0119056630
## X257 X257 0.0118503318
## X567 X567 0.0117169002
## X686 X686 0.0116441026
## X553 X553 0.0114321861
## X134 X134 0.0113956897
## X262 X262 0.0112361507
## X121 X121 0.0111801302
## X632 X632 0.0111780999
## X676 X676 0.0111500154
## X455 X455 0.0109512902
## X454 X454 0.0107890334
## X751 X751 0.0106353976
## X220 X220 0.0105399714
## X110 X110 0.0105052779
## X397 X397 0.0104852349
## X605 X605 0.0104195333
## X469 X469 0.0103649759
## X520 X520 0.0101929121
## X627 X627 0.0101268314
## X482 X482 0.0097576019
## X481 X481 0.0096656416
## X665 X665 0.0096223439
## X172 X172 0.0095589224
## X237 X237 0.0093496503
## X523 X523 0.0093153033
## X293 X293 0.0093031790
## X266 X266 0.0092673864
## X555 X555 0.0092213668
## X273 X273 0.0090740860
## X316 X316 0.0090228143
## X501 X501 0.0088528969
## X186 X186 0.0087856182
## X493 X493 0.0086686634
## X230 X230 0.0086488280
## X92   X92 0.0086226659
## X335 X335 0.0085791555
## X596 X596 0.0080364973
## X256 X256 0.0080013835
## X238 X238 0.0079674649
## X685 X685 0.0079235075
## X182 X182 0.0079165305
## X636 X636 0.0078639669
## X188 X188 0.0077316482
## X411 X411 0.0076738536
## X217 X217 0.0072098772
## X670 X670 0.0069848405
## X288 X288 0.0066174069
## X527 X527 0.0065845431
## X133 X133 0.0065292561
## X166 X166 0.0064612181
## X313 X313 0.0063114960
## X734 X734 0.0060641394
## X558 X558 0.0060259313
## X692 X692 0.0060129039
## X146 X146 0.0059992194
## X395 X395 0.0058617565
## X525 X525 0.0058197985
## X160 X160 0.0055787935
## X261 X261 0.0054811271
## X691 X691 0.0054749834
## X669 X669 0.0054678581
## X120 X120 0.0054529680
## X473 X473 0.0052639623
## X385 X385 0.0051721829
## X604 X604 0.0051394226
## X180 X180 0.0051062761
## X631 X631 0.0048570675
## X132 X132 0.0045610579
## X514 X514 0.0043211425
## X549 X549 0.0043143437
## X162 X162 0.0041740596
## X613 X613 0.0041219777
## X441 X441 0.0040924499
## X777 X777 0.0040901822
## X687 X687 0.0040487660
## X218 X218 0.0040022452
## X741 X741 0.0039938436
## X667 X667 0.0039806452
## X90   X90 0.0039331319
## X426 X426 0.0039122699
## X548 X548 0.0037636447
## X78   X78 0.0037475880
## X749 X749 0.0037170013
## X412 X412 0.0036909230
## X526 X526 0.0035779758
## X79   X79 0.0035608903
## X145 X145 0.0035237367
## X398 X398 0.0034558748
## X577 X577 0.0034211047
## X231 X231 0.0031683380
## X45   X45 0.0030566608
## X425 X425 0.0029663348
## X413 X413 0.0029080569
## X165 X165 0.0028865699
## X187 X187 0.0028151959
## X556 X556 0.0028051491
## X637 X637 0.0027665749
## X528 X528 0.0027435107
## X294 X294 0.0026646368
## X75   X75 0.0026408227
## X287 X287 0.0024930415
## X688 X688 0.0024537302
## X138 X138 0.0023594214
## X228 X228 0.0023569697
## X265 X265 0.0022273322
## X559 X559 0.0018727858
## X668 X668 0.0018221829
## X246 X246 0.0017583212
## X592 X592 0.0017401551
## X471 X471 0.0015910939
## X37   X37 0.0015433579
## X131 X131 0.0015157159
## X624 X624 0.0014761683
## X147 X147 0.0014586122
## X586 X586 0.0013821258
## X311 X311 0.0013795572
## X547 X547 0.0013707967
## X642 X642 0.0013675945
## X117 X117 0.0013338335
## X423 X423 0.0013323037
## X614 X614 0.0013035413
## X194 X194 0.0012783459
## X453 X453 0.0012780714
## X775 X775 0.0011315180
## X648 X648 0.0010617526
## X621 X621 0.0010495336
## X443 X443 0.0010139982
## X500 X500 0.0010091144
## X778 X778 0.0010011584
## X342 X342 0.0009407367
## X733 X733 0.0009202255
## X119 X119 0.0009165057
## X173 X173 0.0008768709
## X216 X216 0.0008728034
## X251 X251 0.0008366514
## X619 X619 0.0008211205
## X654 X654 0.0007902223
## X130 X130 0.0007819905
## X472 X472 0.0007506361
## X445 X445 0.0006883035
## X776 X776 0.0006754497
## X91   X91 0.0006216872
## X74   X74 0.0006078406
## X139 X139 0.0005545893
## X200 X200 0.0005470334
## X42   X42 0.0005160387
## X615 X615 0.0005142636
## X38   X38 0.0004485316
## X575 X575 0.0004072754
## X651 X651 0.0002996371
## X626 X626 0.0002137341
## X444 X444 0.0002093869
## X1     X1 0.0000000000
## X2     X2 0.0000000000
## X3     X3 0.0000000000
## X4     X4 0.0000000000
## X5     X5 0.0000000000
## X6     X6 0.0000000000
## X7     X7 0.0000000000
## X8     X8 0.0000000000
## X9     X9 0.0000000000
## X10   X10 0.0000000000
## X11   X11 0.0000000000
## X12   X12 0.0000000000
## X13   X13 0.0000000000
## X14   X14 0.0000000000
## X15   X15 0.0000000000
## X16   X16 0.0000000000
## X17   X17 0.0000000000
## X18   X18 0.0000000000
## X19   X19 0.0000000000
## X20   X20 0.0000000000
## X21   X21 0.0000000000
## X22   X22 0.0000000000
## X23   X23 0.0000000000
## X24   X24 0.0000000000
## X25   X25 0.0000000000
## X26   X26 0.0000000000
## X27   X27 0.0000000000
## X28   X28 0.0000000000
## X29   X29 0.0000000000
## X30   X30 0.0000000000
## X31   X31 0.0000000000
## X32   X32 0.0000000000
## X33   X33 0.0000000000
## X34   X34 0.0000000000
## X35   X35 0.0000000000
## X36   X36 0.0000000000
## X39   X39 0.0000000000
## X40   X40 0.0000000000
## X41   X41 0.0000000000
## X43   X43 0.0000000000
## X44   X44 0.0000000000
## X46   X46 0.0000000000
## X47   X47 0.0000000000
## X48   X48 0.0000000000
## X49   X49 0.0000000000
## X50   X50 0.0000000000
## X51   X51 0.0000000000
## X52   X52 0.0000000000
## X53   X53 0.0000000000
## X54   X54 0.0000000000
## X55   X55 0.0000000000
## X56   X56 0.0000000000
## X57   X57 0.0000000000
## X58   X58 0.0000000000
## X59   X59 0.0000000000
## X60   X60 0.0000000000
## X61   X61 0.0000000000
## X62   X62 0.0000000000
## X63   X63 0.0000000000
## X73   X73 0.0000000000
## X80   X80 0.0000000000
## X81   X81 0.0000000000
## X82   X82 0.0000000000
## X83   X83 0.0000000000
## X84   X84 0.0000000000
## X85   X85 0.0000000000
## X86   X86 0.0000000000
## X87   X87 0.0000000000
## X88   X88 0.0000000000
## X89   X89 0.0000000000
## X111 X111 0.0000000000
## X112 X112 0.0000000000
## X113 X113 0.0000000000
## X114 X114 0.0000000000
## X115 X115 0.0000000000
## X116 X116 0.0000000000
## X140 X140 0.0000000000
## X141 X141 0.0000000000
## X142 X142 0.0000000000
## X143 X143 0.0000000000
## X144 X144 0.0000000000
## X167 X167 0.0000000000
## X168 X168 0.0000000000
## X169 X169 0.0000000000
## X170 X170 0.0000000000
## X171 X171 0.0000000000
## X195 X195 0.0000000000
## X196 X196 0.0000000000
## X197 X197 0.0000000000
## X198 X198 0.0000000000
## X199 X199 0.0000000000
## X223 X223 0.0000000000
## X224 X224 0.0000000000
## X225 X225 0.0000000000
## X226 X226 0.0000000000
## X227 X227 0.0000000000
## X252 X252 0.0000000000
## X253 X253 0.0000000000
## X254 X254 0.0000000000
## X255 X255 0.0000000000
## X279 X279 0.0000000000
## X280 X280 0.0000000000
## X281 X281 0.0000000000
## X282 X282 0.0000000000
## X283 X283 0.0000000000
## X286 X286 0.0000000000
## X308 X308 0.0000000000
## X309 X309 0.0000000000
## X310 X310 0.0000000000
## X314 X314 0.0000000000
## X315 X315 0.0000000000
## X336 X336 0.0000000000
## X337 X337 0.0000000000
## X338 X338 0.0000000000
## X364 X364 0.0000000000
## X365 X365 0.0000000000
## X366 X366 0.0000000000
## X367 X367 0.0000000000
## X392 X392 0.0000000000
## X393 X393 0.0000000000
## X394 X394 0.0000000000
## X414 X414 0.0000000000
## X420 X420 0.0000000000
## X421 X421 0.0000000000
## X422 X422 0.0000000000
## X448 X448 0.0000000000
## X449 X449 0.0000000000
## X450 X450 0.0000000000
## X476 X476 0.0000000000
## X477 X477 0.0000000000
## X478 X478 0.0000000000
## X479 X479 0.0000000000
## X497 X497 0.0000000000
## X498 X498 0.0000000000
## X499 X499 0.0000000000
## X503 X503 0.0000000000
## X504 X504 0.0000000000
## X505 X505 0.0000000000
## X506 X506 0.0000000000
## X507 X507 0.0000000000
## X531 X531 0.0000000000
## X532 X532 0.0000000000
## X533 X533 0.0000000000
## X534 X534 0.0000000000
## X535 X535 0.0000000000
## X560 X560 0.0000000000
## X561 X561 0.0000000000
## X562 X562 0.0000000000
## X563 X563 0.0000000000
## X587 X587 0.0000000000
## X588 X588 0.0000000000
## X589 X589 0.0000000000
## X590 X590 0.0000000000
## X591 X591 0.0000000000
## X594 X594 0.0000000000
## X616 X616 0.0000000000
## X617 X617 0.0000000000
## X618 X618 0.0000000000
## X620 X620 0.0000000000
## X641 X641 0.0000000000
## X643 X643 0.0000000000
## X644 X644 0.0000000000
## X645 X645 0.0000000000
## X646 X646 0.0000000000
## X647 X647 0.0000000000
## X671 X671 0.0000000000
## X672 X672 0.0000000000
## X673 X673 0.0000000000
## X674 X674 0.0000000000
## X675 X675 0.0000000000
## X682 X682 0.0000000000
## X690 X690 0.0000000000
## X698 X698 0.0000000000
## X699 X699 0.0000000000
## X700 X700 0.0000000000
## X701 X701 0.0000000000
## X702 X702 0.0000000000
## X703 X703 0.0000000000
## X704 X704 0.0000000000
## X725 X725 0.0000000000
## X726 X726 0.0000000000
## X727 X727 0.0000000000
## X728 X728 0.0000000000
## X729 X729 0.0000000000
## X730 X730 0.0000000000
## X731 X731 0.0000000000
## X732 X732 0.0000000000
## X735 X735 0.0000000000
## X736 X736 0.0000000000
## X739 X739 0.0000000000
## X752 X752 0.0000000000
## X753 X753 0.0000000000
## X754 X754 0.0000000000
## X755 X755 0.0000000000
## X756 X756 0.0000000000
## X757 X757 0.0000000000
## X758 X758 0.0000000000
## X759 X759 0.0000000000
## X760 X760 0.0000000000
## X761 X761 0.0000000000
## X762 X762 0.0000000000
## X763 X763 0.0000000000
## X764 X764 0.0000000000
## X765 X765 0.0000000000
## X766 X766 0.0000000000
## X767 X767 0.0000000000
## X768 X768 0.0000000000
## X769 X769 0.0000000000
## X770 X770 0.0000000000
## X771 X771 0.0000000000
## X772 X772 0.0000000000
## X773 X773 0.0000000000
## X774 X774 0.0000000000
## X779 X779 0.0000000000
## X780 X780 0.0000000000
## X781 X781 0.0000000000
## X782 X782 0.0000000000
## X783 X783 0.0000000000
## X784 X784 0.0000000000
```

```
print(boost.og.cv)
```

```
## gbm(formula = label ~ ., distribution = "multinomial", data = train_df,
##     n.trees = 500, interaction.depth = 1, shrinkage = 0.1, cv.folds = 5)
## A gradient boosted model with multinomial loss function.
## 500 iterations were performed.
## The best cross-validation iteration was 500.
## There were 784 predictors of which 565 had non-zero influence.
```

The best number of trees chosen by the boosted model using 5-fold CV on the PCA dataset is 454.

```
pred.boost.og.cv <- predict(boost.og.cv, newdata = test_df, n.trees=500)
pred.boost.og.cv <- apply(pred.boost.og.cv, 1, which.max) -1

(conf.boost <- table(pred.boost.og.cv, test_df$label))
```

```
##
## pred.boost.og.cv    0    1    2    3    4    5    6    7    8    9
##                0  957    0   11    4    1    9   12    2   10   10
##                1    0 1116    5    1    2    0    4    9    8    9
##                2    3    3  906   20    5    2    6   24    8    2
##                3    1    2   20  904    2   37    1    9   22   15
##                4    0    0   16    2  905   12   12   11   11   29
##                5    6    3    0   23    3  759   24    1   17    6
##                6    8    4   21    4   13   22  888    1   13    2
##                7    1    0   16   14    2   11    4  938   11   19
##                 8    3    7   32   27    9   31    7    1  857   15
##                 9    1    0    5   11   40    9    0   32   17  902
```

```
(sum(conf.boost) - sum(diag(conf.boost))) /
  sum(conf.boost)
```

```
## [1] 0.0868
```

The misclassification rate is 8.38. 4-9 is still the most difficult pair to predict.

Logistic Regression

```
ProbabilityOfEachValue <- data.frame(predict(prob.zero, test.zero),
                                     predict(prob.one, test.one),
                                     predict(prob.two, test.two),
                                     predict(prob.three, test.three),
                                     predict(prob.four, test.four),
                                     predict(prob.five, test.five),
                                     predict(prob.six, test.six),
                                     predict(prob.seven, test.seven),
                                     predict(prob.eight, test.eight),
                                     predict(prob.nine, test.nine))


# Find the index with the highest probability predicted by the models for each class and store it in a

Label <- rep(NA, nrow(ProbabilityOfEachValue))
for (i in seq(nrow(ProbabilityOfEachValue)))
{
  Label[i] <- which.max(ProbabilityOfEachValue[i,]) - 1
}
(conf.log <- table(Label, pca.tst$label))
```

```
##
## Label    0    1    2    3    4    5    6    7    8    9
##     0  948    0   14    4    3   19   18    4   16    9
##     1    0 1100   16    1    3    4    4   10   17   10
##     2    4    3  844   22    9    7   12   39   16   16
##     3    3    2   29  872    1   68    2    4   52   16
##     4    1    0   13    1  868   21   15   15   10   76
##     5   10    3    5   43   13  679   26    3   39   29
##     6    8    4   30    4   14   25  878    1   16    0
##     7    1    1   19   19    2   15    1  914    6   38
##     8    5   22   45   30   12   33    2    5  781   14
##     9    0    0   17   14   57   21    0   33   21  801
```

```
(sum(conf.log) - sum(diag(conf.log))) / sum(conf.log)
```

```
## [1] 0.1315
```

The misclassification rate is 13.15%.

KNN

```
knn.pred <- knn(pca.tr[,-1], pca.tst[, -1], pca.tr[,1], k =5) # use CV the best k is 5
table(knn.pred, pca.tst[,1])
```

```
## 
## knn.pred    0    1    2    3    4    5    6    7    8    9
##        0  971    0    5    0    0    4    3    1    1    1
##        1    1 1129    0    2    0    2    4   16    1    4
##        2    1    2 1002    6    0    3    1   10    3    2
##        3    1    1    0  967    0    7    0    0   12    6
##        4    0    0    0    0  946    0    2    1    3   11
##        5    1    0    2   11    0  859    1    0   12    3
##        6    4    1    3    0    3    7  945    1    4    1
##        7    1    0   12    8    1    1    1  985    3    8
##        8    0    1    8   15    1    3    1    0  930    9
##        9    0    1    0    1   31    6    0   14    5  964
```

```
knn.MCR <- mean(knn.pred != pca.tst[,1])
knn.MCR
```

```
## [1] 0.0302
```

Clearly, with KNN method, the misclassifcation rate is 3.02%. 4-9 pair is the hardest one to predict.

SVM

```
pca.svm <- svm(label~., data = pca.tr, method="C-classification", kernal="radial", gamma= 0.1, cost=10)
```

```
svm.pred <- predict(pca.svm, pca.tst)
table(svm.pred, pca.tst[,1])
```

```
## 
## svm.pred    0    1    2    3    4    5    6    7    8    9
##        0  973    0    4    0    0    2    5    1    1    1
##        1    0 1127    2    0    0    0    3    3    0    3
##        2    1    2 1008    5    2    0    1   11    4    0
##        3    0    1    5  989    0   13    0    0    5    4
##        4    0    0    0    0  960    2    4    4    1   13
##        5    2    0    0    4    0  865    4    0    2    5
##        6    1    1    1    0    3    2  936    0    0    1
##        7    1    1    9    4    0    1    1  998    2    7
##        8    2    2    3    5    2    3    4    1  955    3
##        9    0    1    0    3   15    4    0   10    4  972
```

```
svm.MCR <- mean(svm.pred != pca.tst[,1])
svm.MCR
```

```
## [1] 0.0217
```

Clearly, with SVM method, the misclassifcation rate is 2.17%. 4-9 pair is the hardest one to predict.