R Notebook

Code

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Hide

```
library(devtools)
```

```
Loading required package: usethis
```

Hide

```
devtools::install_github("jlmelville/snedata", force=TRUE)
```

```
Downloading GitHub repo jlmelville/snedata@HEAD
```

```
   checking for file '/private/var/folders/qm/h305gl4j10z4njsb8xmc6vtm0000gn/T/RtmpFh
CAUY/remotesbf354c147785/jlmelville-snedata-02e6e7f/DESCRIPTION' ...

✓  checking for file '/private/var/folders/qm/h305gl4j10z4njsb8xmc6vtm0000gn/T/RtmpFh
CAUY/remotesbf354c147785/jlmelville-snedata-02e6e7f/DESCRIPTION'


─  preparing 'snedata':


   checking DESCRIPTION meta-information ...

✓  checking DESCRIPTION meta-information


─  checking for LF line-endings in source and make files and shell scripts


─  checking for empty or unneeded directories


─  building 'snedata_0.0.0.9000.tar.gz'
```

```
* installing *source* package 'snedata' ...
** using staged installation
** R
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded from temporary location
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (snedata)
```

Hide

```
library(snedata)
dat<- snedata::download_fashion_mnist()
dim(dat)
```

```
[1] 70000   786
```

Hide

```
# For the initial experiments, pick
# 1000 points as training data
set.seed(1)
train<-sample(70000,1000)

# training data
x.train<-dat[train,1:784]
y.train<-dat[train,785]
table(y.train)
```

```
y.train
  0   1   2   3   4   5   6   7   8   9
111  88 106 109  97 117 100  85  91  96
```

Hide

```
# types of objects to classify here
types<-dat[train,786]
table(types)
```

```
types
T-shirt/top     Trouser    Pullover       Dress        Coat      Sandal       Shirt
        111          88         106         109          97         117         100
    Sneaker         Bag  Ankle boot
         85          91          96
```

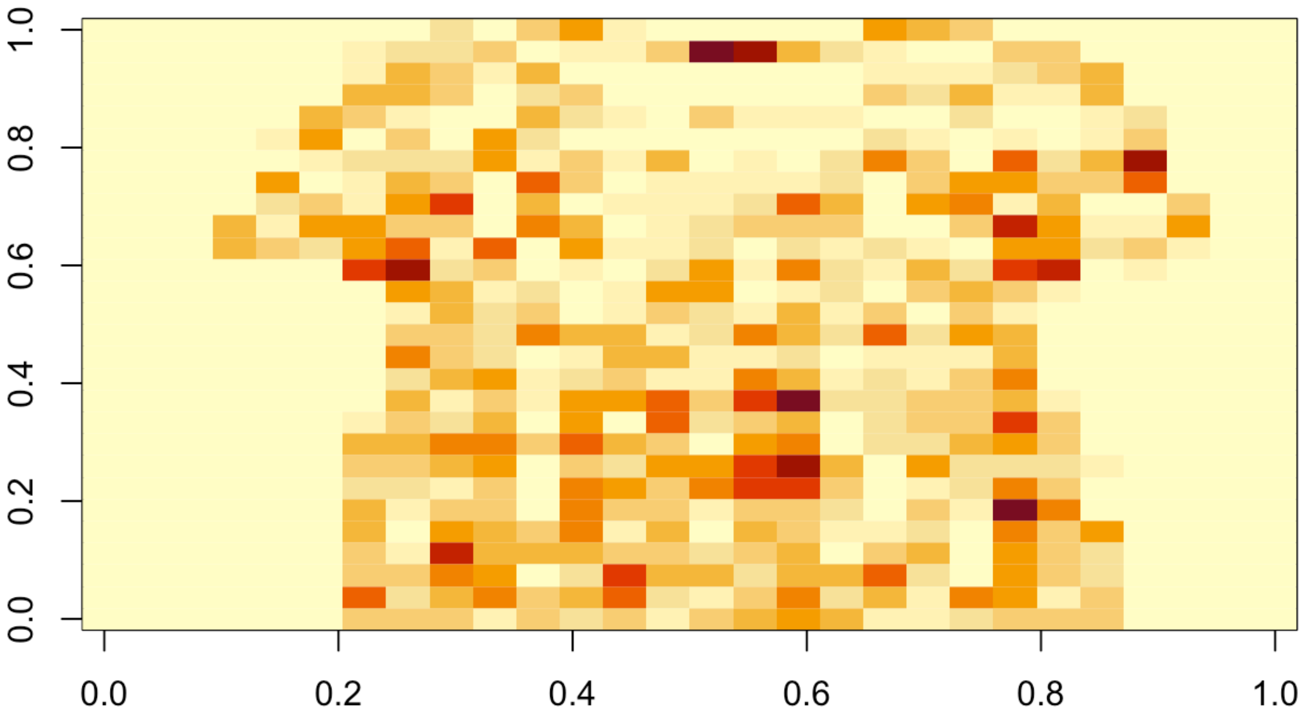
Hide

```
# look at one of the clothing articles
types[1]
```

```
[1] Shirt
Levels: T-shirt/top Trouser Pullover Dress Coat Sandal Shirt Sneaker Bag Ankle boot
```

Hide

```
x<-matrix(as.numeric(x.train[1,]),nrow=28)
image(x[,28:1])
```



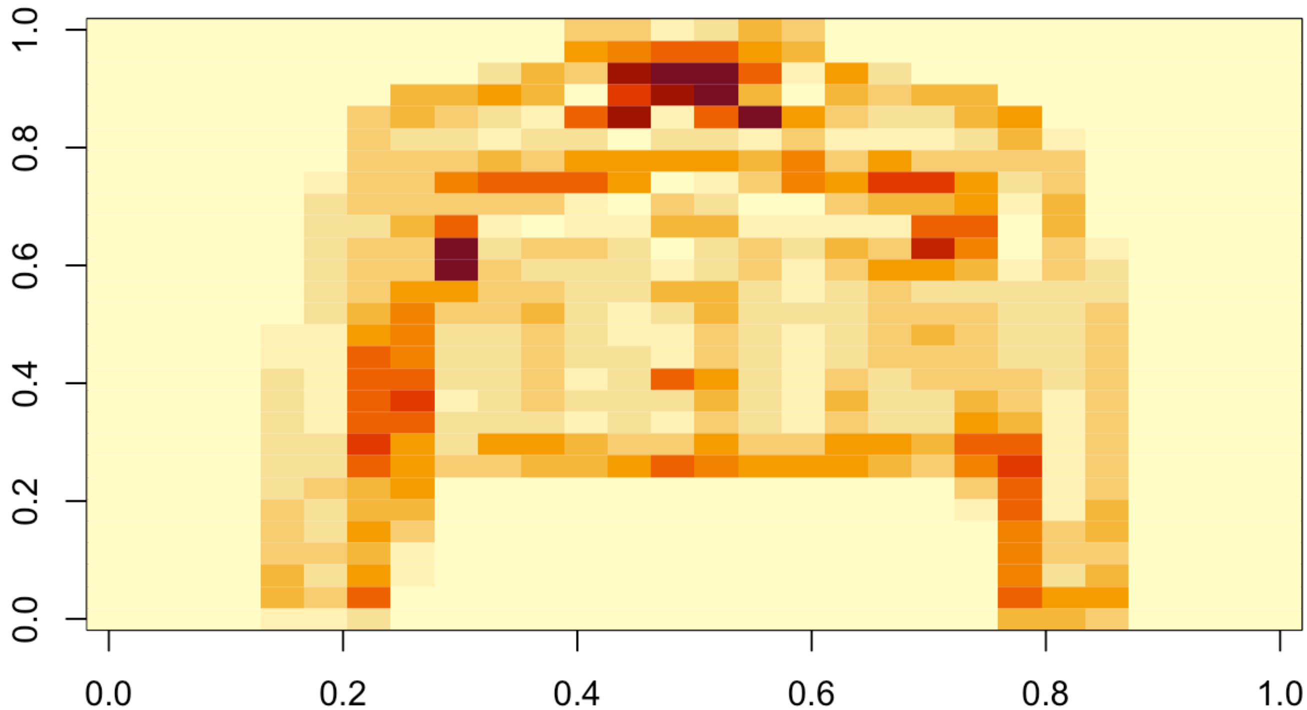
Hide

```
# look at another clothing article
types[2]
```

```
[1] Coat
Levels: T-shirt/top Trouser Pullover Dress Coat Sandal Shirt Sneaker Bag Ankle boot
```

Hide

```
x<-matrix(as.numeric(x.train[2,]),nrow=28)
image(x[,28:1])
```



Hide

```
# testing data
ind<-1:70000
test<-sample(ind[-train],1000)
x.test<-dat[test,1:784]
y.test<-dat[test,785]
table(y.test)
```

```
y.test
  0   1   2   3   4   5   6   7   8   9 
108  89 108  92 105  97 107  89  98 107 
```

Hide

```
set.seed(2)
# first apply PCA to the training data to find the transformation
# into a lower dimensional space
pca.train<-prcomp(x.train,center=TRUE,scale.=FALSE)

# how many dimensions do we need to explain
# 90% of the total variation in the data?
var.explained<-cumsum(pca.train$sdev^2)/sum(pca.train$sdev^2)
pca.dim<-min(which(var.explained > 0.90))
pca.dim
```

```
[1] 71
```

Hide

```
# lower dimensional data
x.pca.train<-pca.train$x[,1:pca.dim]
dim(x.pca.train)
```

```
[1] 1000   71
```

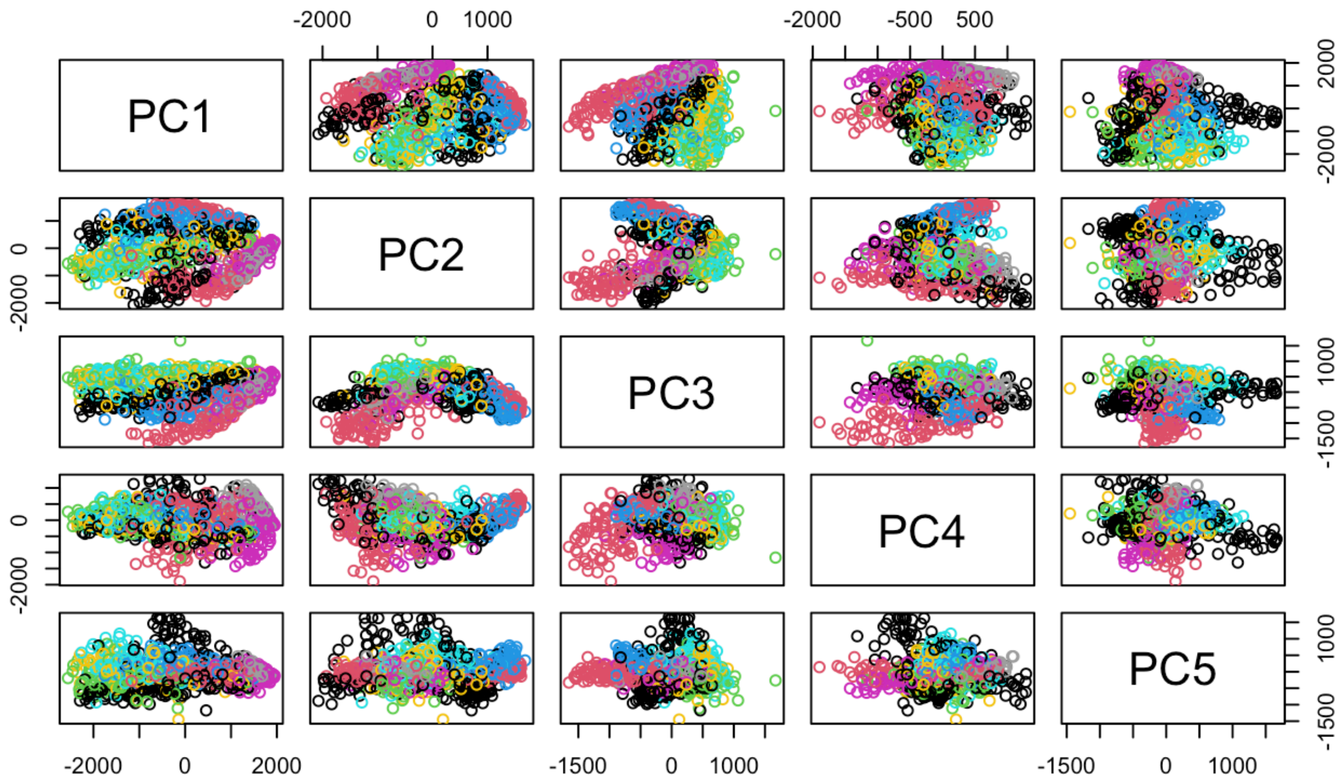
Hide

```
# from SVD
proj.train<-pca.train$rotation[,1:pca.dim]
x.centered<-scale(x.train,center=TRUE,scale=FALSE)
table(x.pca.train == x.centered%*%proj.train)
```

```

 TRUE
71000
```

Hide

```
# look at pairs of PC's, with data colored
# by class label
pairs(x.pca.train[,1:5],col=y.train)
```



Hide

```
# apply the SAME centering and projection to the test data
cc<-colMeans(x.train)
x.centered<-sweep(x.test,2,cc)
x.pca.test<-as.matrix(x.centered)%*% (pca.train$rotation)
x.pca.test<-x.pca.test[,1:pca.dim]
dim(x.pca.test)
```

```
[1] 1000   71
```

Hide

```
# What dimension did we embed into?
```

Hide

```
# knn on the 784-dimensional data
library(caret)
```

```
Loading required package: ggplot2
Loading required package: lattice
```

Hide

```
set.seed(3)
dat.train=data.frame(x=x.train, y=as.factor(y.train))
```

Hide

```


# Training K-NN using repeated 5-fold CV
# possible hyperparameter choices are K=1,3,5,7,9,11
rt <- proc.time()
knn_5CV <- train(y~ .,
                    # running time
                    data=dat.train,
                    method = "knn",
                    tuneGrid = expand.grid(k=c(1,3,5,7,9,11)),
                    trControl=trainControl(method='repeatedcv',
                                          number=5, repeats=10))
proc.time()-rt
```

```
   user  system elapsed 
 67.978   2.510  71.102 
```

Hide

```
knn_5CV
```

```
k-Nearest Neighbors

1000 samples
 784 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 10 times)
Summary of sample sizes: 799, 800, 801, 801, 799, 799, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.7594225  0.7326635
   3  0.7633731  0.7369928
   5  0.7706775  0.7451102
   7  0.7667798  0.7407695
   9  0.7585900  0.7316841
  11  0.7526022  0.7250147

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
```

Hide

```
# Repeat the above with repeated 10-fold CV. How does the running
# time compare?  Why is this reasonable?
rt <- proc.time()
knn_10CV <- train(y~ .,
                    # running time
                    data=dat.train,
                    method = "knn",
                    tuneGrid = expand.grid(k=c(1,3,5,7,9,11)),
                    trControl=trainControl(method='repeatedcv',
                                           number=10, repeats=10))
proc.time()-rt
```

```
   user  system elapsed
 72.235   4.118  76.351
```

Hide

```
knn_10CV
```

```
k-Nearest Neighbors

1000 samples
 784 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 900, 902, 900, 899, 900, 898, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.7627316  0.7363034
   3  0.7671068  0.7411017
   5  0.7807331  0.7562399
   7  0.7713882  0.7458663
   9  0.7658712  0.7397445
  11  0.7614539  0.7348448

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
```

Hide

```
# knn on the reduced dimension data obtained via PCA
dat.pca.train=data.frame(x=x.pca.train, y=as.factor(y.train))
```

Hide

```
# Training K-NN using repeated 5-fold CV
# possible hyperparameter choices are K=1,3,5,7,9,11
rt <- proc.time()
knn_5CV_pca <- train(y~ .,
                        data=dat.pca.train,
                        method = "knn",
                        tuneGrid = expand.grid(k=c(1,3,5,7,9,11)),
                        trControl=trainControl(method='repeatedcv',
                                               number=5, repeats=10))
# running time
proc.time()-rt
```

```
   user  system elapsed
  5.665   0.225   5.895
```

Hide

```
knn_5CV_pca$results
```

	k	Accuracy	Kappa	AccuracySD	KappaSD
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1	0.7695579	0.7438369	0.02532837	0.02811739
2	3	0.7672627	0.7412522	0.02477286	0.02753377
3	5	0.7767911	0.7518056	0.02500623	0.02776440
4	7	0.7778600	0.7530019	0.02507683	0.02782353
5	9	0.7741634	0.7488833	0.02489273	0.02764224
6	11	0.7679723	0.7420122	0.02289752	0.02542127

6 rows

Hide

```
knn_5CV_pca
```

```
k-Nearest Neighbors

1000 samples
  71 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 10 times)
Summary of sample sizes: 800, 800, 801, 799, 800, 801, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.7695579  0.7438369
   3  0.7672627  0.7412522
   5  0.7767911  0.7518056
   7  0.7778600  0.7530019
   9  0.7741634  0.7488833
  11  0.7679723  0.7420122

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.
```

Hide

```
# Run repeated CV knn on the PCA'd data with 10-fold CV.
# How does the running time compare?  Why is this reasonable?
rt <- proc.time()
knn_10CV_pca <- train(y~ .,
                        data=dat.pca.train,
                        method = "knn",
                        tuneGrid = expand.grid(k=c(1,3,5,7,9,11)),
                        trControl=trainControl(method='repeatedcv',
                                               number=10, repeats=10))
# running time
proc.time()-rt
```

```
   user  system elapsed
  7.494   0.371   7.872
```

Hide

```
knn_10CV_pca$results
```

	k	Accuracy	Kappa	AccuracySD	KappaSD
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1	0.7780081	0.7531824	0.03526113	0.03919539
2	3	0.7666552	0.7405758	0.03653428	0.04059481
3	5	0.7808731	0.7563153	0.03473852	0.03862623
4	7	0.7810990	0.7565747	0.03834285	0.04263473
5	9	0.7818831	0.7574184	0.03990770	0.04439294
6	11	0.7716565	0.7460550	0.04062545	0.04517790

6 rows

Hide

```
knn_10CV_pca
```

```
k-Nearest Neighbors

1000 samples
  71 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 900, 900, 901, 898, 900, 900, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.7780081  0.7531824
   3  0.7666552  0.7405758
   5  0.7808731  0.7563153
   7  0.7810990  0.7565747
   9  0.7818831  0.7574184
  11  0.7716565  0.7460550

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

Hide

```
# MUCH faster on PCA'd data, what about accuracy?
# unprojected data
colnames(x.test)<-colnames(dat.train)[1:784]
y.pred<-predict(knn_5CV, newdata=x.test)
```

Hide

```
# where are the errors occuring?
table(y.pred,y.test)
```

```
      y.test
y.pred   0   1   2   3   4   5   6   7   8   9
     0  97   2   2  11   2   2  34   0   3   0
     1   0  78   0   1   0   0   0   0   0   0
     2   1   2  64   0  10   0  14   0   1   0
     3   4   5   2  75   4   0   6   0   2   0
     4   0   0  27   5  68   0   8   0   1   0
     5   0   0   0   0   0  75   0   2   1   0
     6   5   2  13   0  20   0  43   0  11   1
     7   0   0   0   0   0  12   0  81   4   6
     8   1   0   0   0   1   3   2   0  74   0
     9   0   0   0   0   0   5   0   6   1 100
```

Hide

```
# remember, the labels are given by
table(dat[,785],dat[,786])
```

```

    T-shirt/top Trouser Pullover Dress Coat Sandal Shirt Sneaker  Bag Ankle boot
  0        7000       0        0     0    0      0     0       0    0          0
  1           0    7000        0     0    0      0     0       0    0          0
  2           0       0     7000     0    0      0     0       0    0          0
  3           0       0        0  7000    0      0     0       0    0          0
  4           0       0        0     0 7000      0     0       0    0          0
  5           0       0        0     0    0   7000     0       0    0          0
  6           0       0        0     0    0      0  7000       0    0          0
  7           0       0        0     0    0      0     0    7000    0          0
  8           0       0        0     0    0      0     0       0 7000          0
  9           0       0        0     0    0      0     0       0    0       7000
```

Hide

```
# accuracy
sum(y.pred==y.test)/1000
```

```
[1] 0.77
```

Hide

```
# projected data
colnames(x.pca.test)<-colnames(dat.pca.train)[1:pca.dim]
y.pred.pca<-predict(knn_5CV_pca, newdata=x.pca.test)
```

Hide

```
# where are the errors occuring?
table(y.pred.pca,y.test)
```

```
          y.test
y.pred.pca  0  1  2  3  4  5  6  7  8  9
         0 80  1  5 10  3  0 33  0  2  0
         1  0 98  0  0  0  0  0  0  0  0
         2  1  2 62  1 12  0  7  0  1  0
         3  4  4  1 74  6  0  6  0  0  0
         4  0  1 18  4 64  0 10  0  1  0
         5  0  0  0  0  0 84  0  8  3  3
         6  4  0 12  1 11  1 42  0  1  0
         7  0  0  0  0  0  8  0 97  1  7
         8  0  0  1  0  0  0  2  0 96  0
         9  0  0  0  0  0  6  0 12  1 88
```

Hide

```
# accuracy
sum(y.pred.pca==y.test)/1000
```

```
[1] 0.785
```

Hide

```
# Was there a significant loss in accuracy here?
# Compute the accuracy estimated by 10-fold CV as well.  How does
# it compare to the estimated 5-fold CV accuracy?
colnames(x.test)<-colnames(dat.train)[1:784]
y.pred<-predict(knn_10CV, newdata=x.test)
table(y.pred,y.test)
```

```
      y.test
y.pred   0   1   2   3   4   5   6   7   8   9
     0  80   1   7  11   2   1  29   0   2   0
     1   0  99   0   2   0   0   0   0   0   0
     2   1   2  52   0  12   1   9   0   0   1
     3   5   4   1  71   7   0   9   0   0   0
     4   0   0  21   5  60   0   9   0   2   0
     5   0   0   0   0   0  77   0   2   0   1
     6   3   0  18   1  15   0  42   0   1   0
     7   0   0   0   0   0  11   0 103   3   9
     8   0   0   0   0   0   0   2   0  97   0
     9   0   0   0   0   0   9   0  12   1  87
```

Hide

```
sum(y.pred==y.test)/1000
```

```
[1] 0.768
```

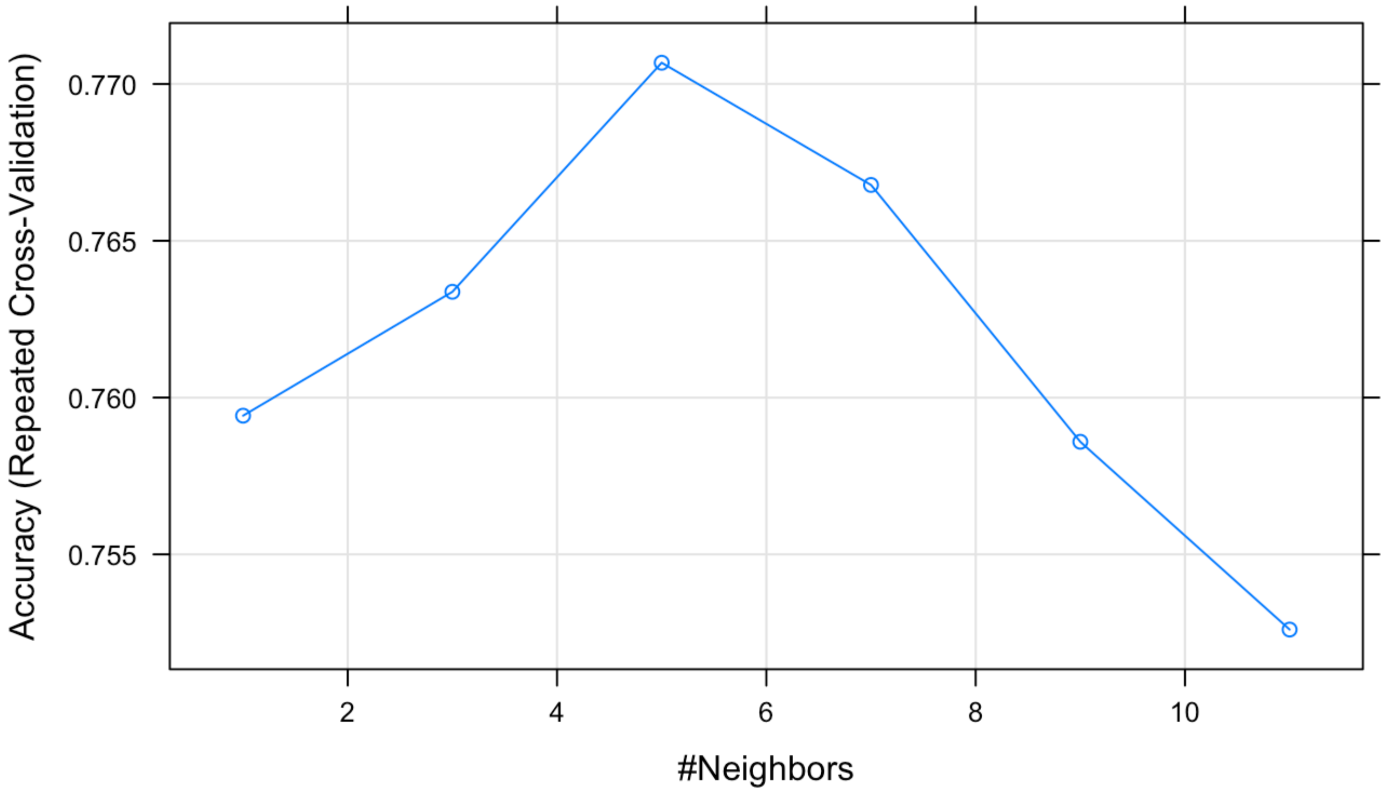
Hide

```
#knn_10CV_pca
colnames(x.pca.test)<-colnames(dat.pca.train)[1:pca.dim]
y.pred.pca<-predict(knn_10CV_pca, newdata=x.pca.test)
sum(y.pred.pca==y.test)/1000
```

```
[1] 0.783
```

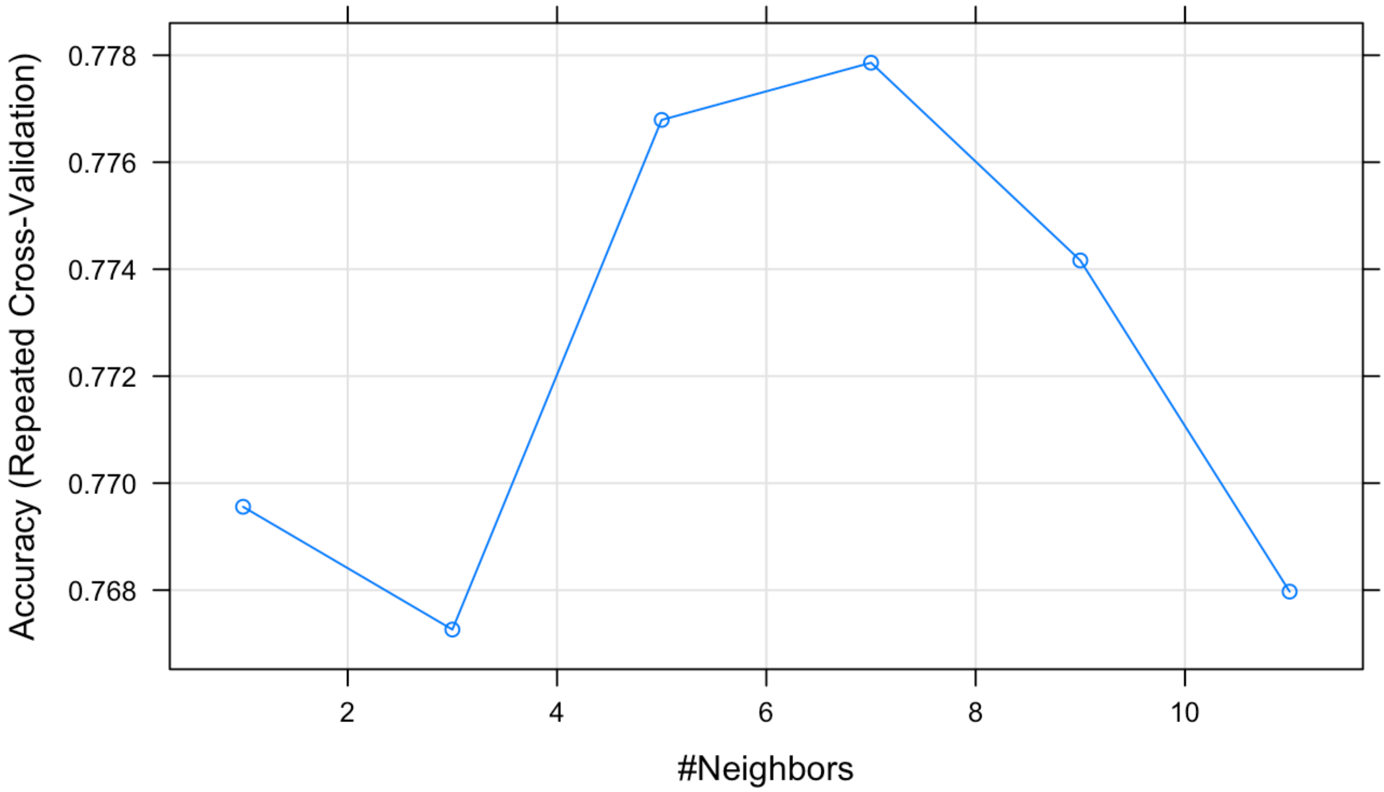
Hide

```
#  We saw how to use the CV to choose our model parameter k above.
#  Often, a good figure can do wonders for explaining this!
# plot the predicted test accuracy  here
plot(knn_5CV)
```



Hide

```
plot(knn_5CV_pca)
```

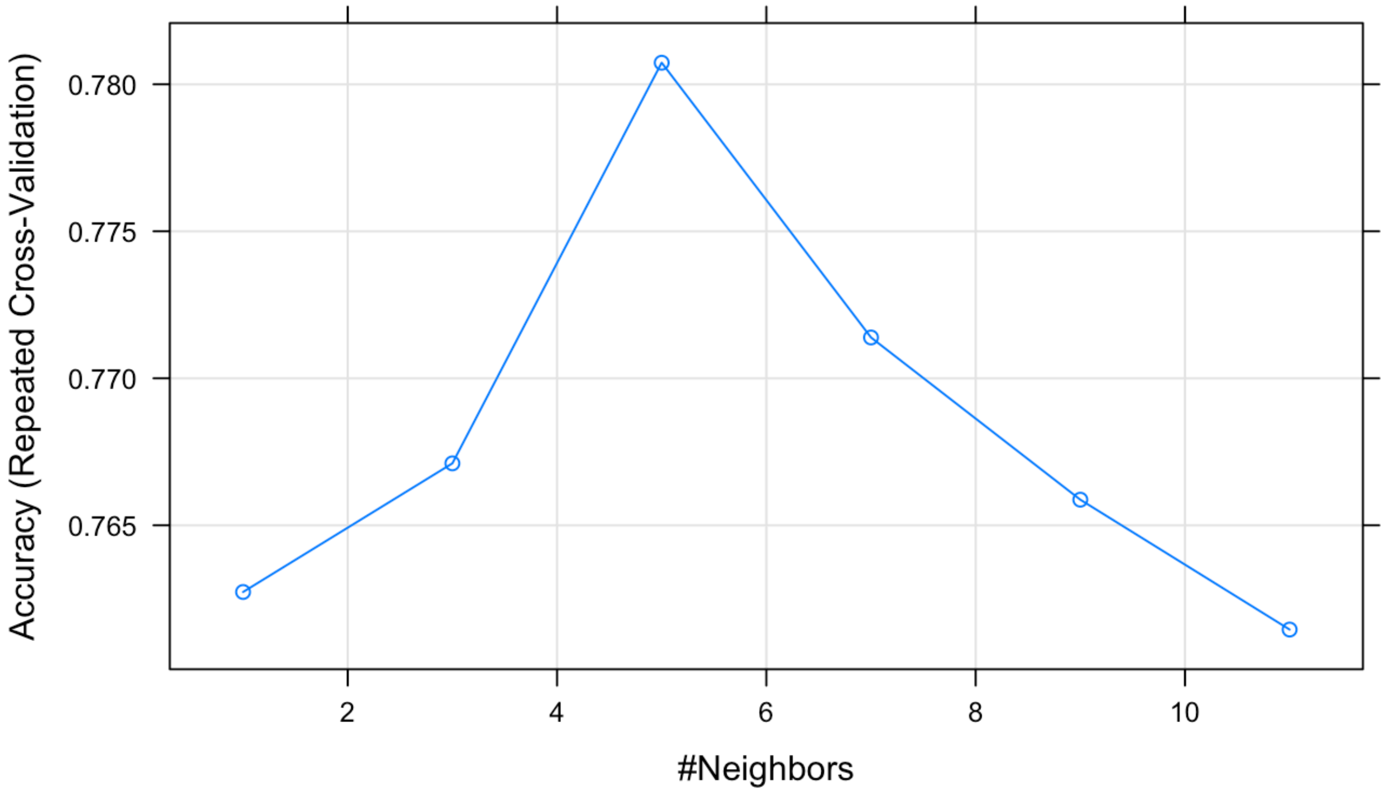


Hide

```
# Q: What are these predicted accuracies estimating?
# Q: More generally, what is cross validation estimating?
```

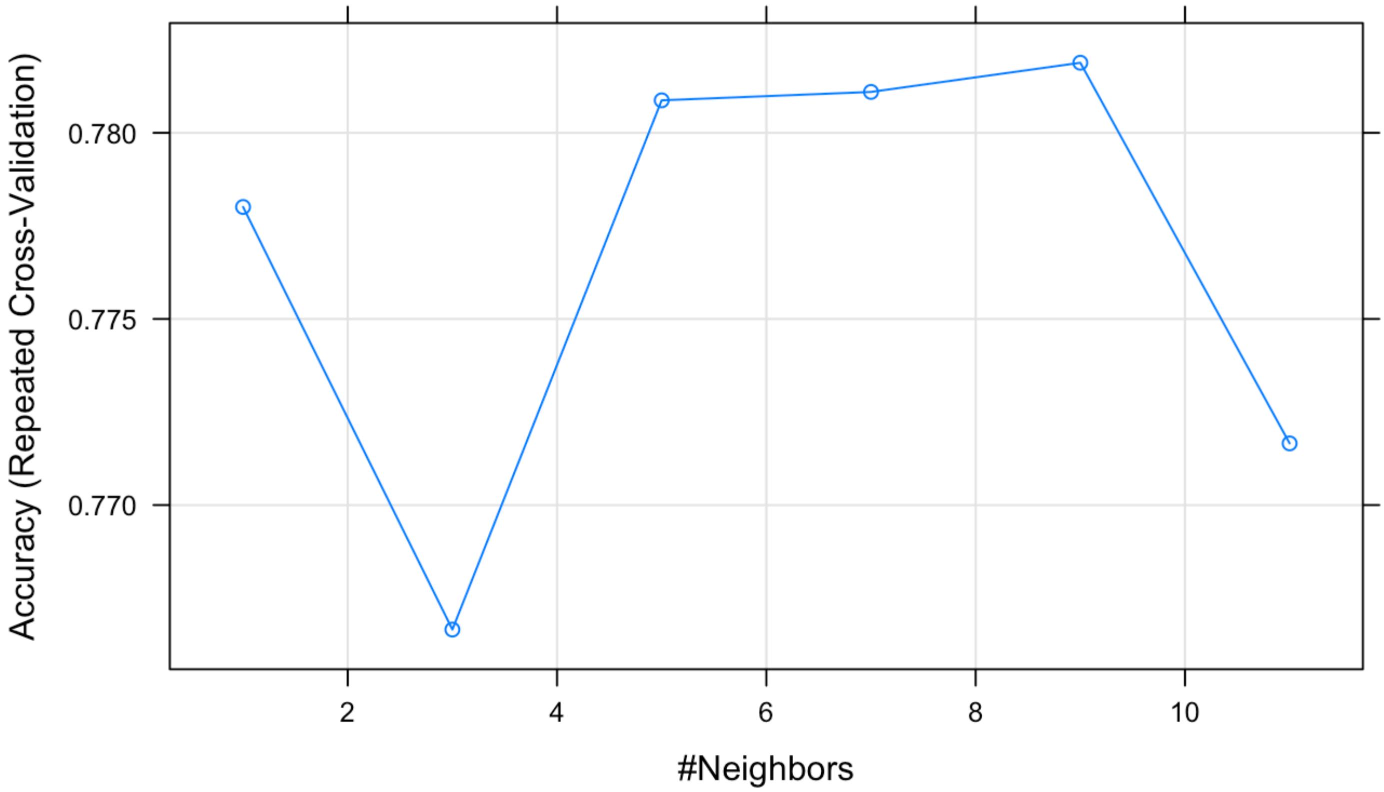
Hide

```
# Produce plotted accuracies for the 10-fold CV as well.  How
# does the estimated optimal k differ across the 4 scenarios (5-fold
# and 10-fold on the full data and the PCA'd data).
plot(knn_10CV)
```



Hide

```
plot(knn_10CV_pca)
```



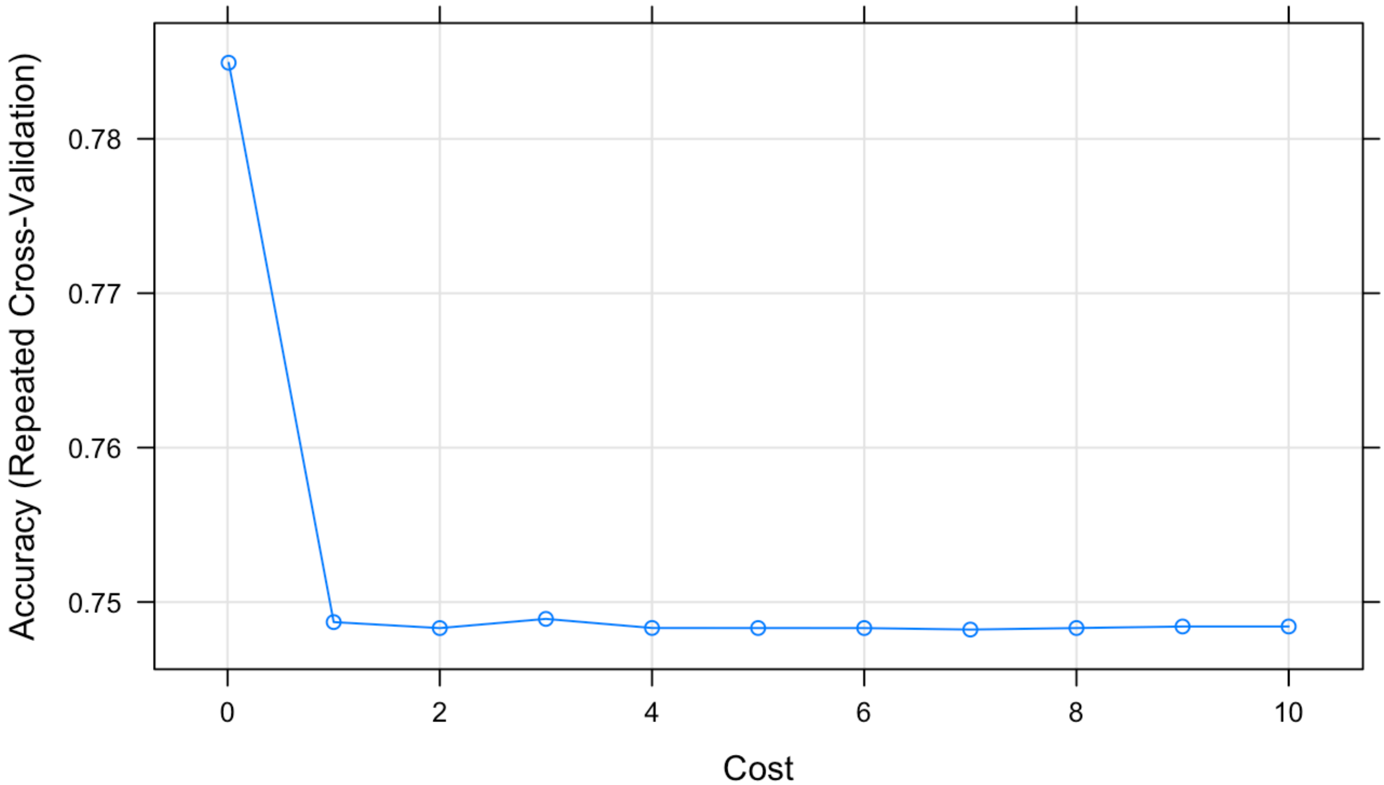
Hide

```
set.seed(4)
# First we use linear SVM
# Costs (C) considered
cgrid<-0:10
# 0 cost causes trouble sometimes... why?
cgrid[1]<-0.01
svm_linear <- train(
    y~., data = dat.pca.train, method = "svmLinear",
    trControl = trainControl(method='repeatedcv',
                              number=5, repeats=10),
    tuneGrid = expand.grid(C = cgrid)
)
svm_linear
```

```
Support Vector Machines with Linear Kernel

1000 samples
  71 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 10 times)
Summary of sample sizes: 799, 799, 801, 800, 801, 801, ...
Resampling results across tuning parameters:

  C      Accuracy   Kappa
   0.01  0.7849279  0.7605244
   1.00  0.7486993  0.7203188
   2.00  0.7483105  0.7199010
   3.00  0.7489066  0.7205741
   4.00  0.7483185  0.7199223
   5.00  0.7483140  0.7199184
   6.00  0.7483130  0.7199156
   7.00  0.7482160  0.7198066
   8.00  0.7483150  0.7199167
   9.00  0.7484140  0.7200269
  10.00  0.7484140  0.7200269

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.01.
```

Hide

```
plot(svm_linear)
```



Hide

```
# How well does it do?
y.pred <- predict(svm_linear, newdata=x.pca.test)
sum(y.pred==y.test)/1000
```

```
[1] 0.796
```

Hide

```
# Next consider radial kernel SVM.
# Be patient, this could take a while...
# Q: How many models need to be fit for 5-fold CV # with 10 repeats here?
grid <- expand.grid(sigma = c(0.01, 0.1, 0.5,1,2),
                              C = cgrid[1:5])
grid
```

sigma <dbl>	C <dbl>
0.01	0.01

0.10	0.01
0.50	0.01
1.00	0.01
2.00	0.01
0.01	1.00
0.10	1.00
0.50	1.00
1.00	1.00
2.00	1.00

1-10 of 25 rows Previous **1** 2 3 Next

Hide

```
NA
```

Hide

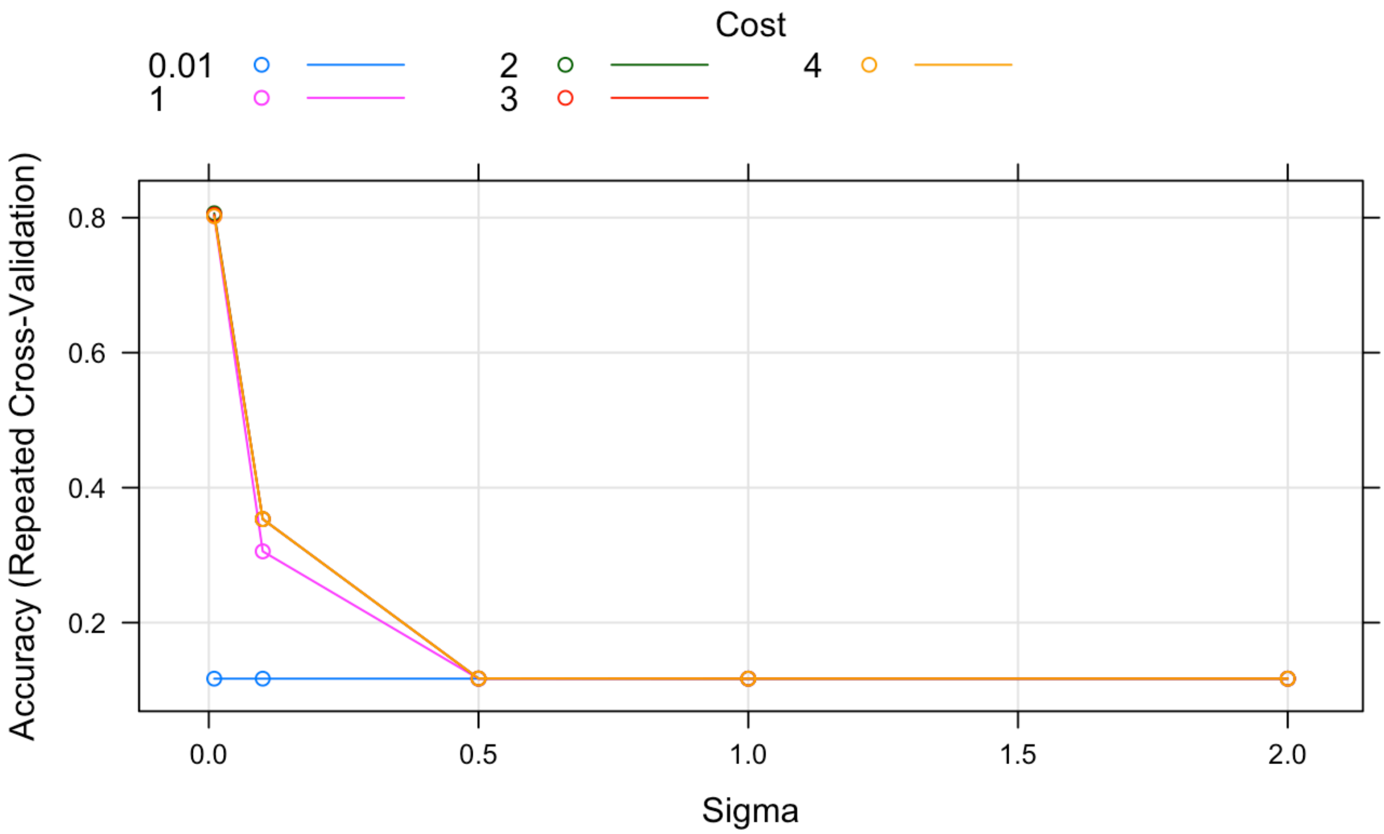
```
# In practice, we would grid finer than this. Ideally,
# something like
# grid <- expand.grid(sigma = seq(0.01, 2, length = 20),
#                     C = seq(0.01, 10, length = 100))
# but then each CV would take a long time...

# To run the radial basis SVM (which you should name
# svm_radial), you use the same syntax as the command
# to run the linear SVM with two key changes
# method = "svmRadial"
# tuneGrid = grid

svm_radial <- train(
    y~., data = dat.pca.train, method = "svmRadial",
    trControl = trainControl(method='repeatedcv',
                             number=5, repeats=10),
    tuneGrid = grid
)
# You can find the estimated best tuning parameters via
svm_radial$bestTune
```

sigma	C
<dbl>	<dbl>

3	0.01	2

1 row

```
plot(svm_radial)
```




```
svm_radial
```

```
Support Vector Machines with Radial Basis Function Kernel

1000 samples
  71 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 10 times)
Summary of sample sizes: 800, 800, 799, 801, 800, 799, ...
Resampling results across tuning parameters:

  sigma  C     Accuracy   Kappa
  0.01   0.01  0.1169996  0.0000000000
  0.01   1.00  0.8023094  0.7798867235
  0.01   2.00  0.8065170  0.7846142521
  0.01   3.00  0.8041184  0.7819536718
  0.01   4.00  0.8020154  0.7796152719
  0.10   0.01  0.1169996  0.0000000000
  0.10   1.00  0.3056154  0.2165812029
  0.10   2.00  0.3534097  0.2713454615
  0.10   3.00  0.3534097  0.2713454615
  0.10   4.00  0.3534097  0.2713454615
  0.50   0.01  0.1169996  0.0000000000
  0.50   1.00  0.1169996  0.0000000000
  0.50   2.00  0.1171996  0.0002346158
  0.50   3.00  0.1171996  0.0002346158
  0.50   4.00  0.1171996  0.0002346158
  1.00   0.01  0.1169996  0.0000000000
  1.00   1.00  0.1169996  0.0000000000
  1.00   2.00  0.1169996  0.0000000000
  1.00   3.00  0.1169996  0.0000000000
  1.00   4.00  0.1169996  0.0000000000
  2.00   0.01  0.1169996  0.0000000000
  2.00   1.00  0.1169996  0.0000000000
  2.00   2.00  0.1169996  0.0000000000
  2.00   3.00  0.1169996  0.0000000000
  2.00   4.00  0.1169996  0.0000000000

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.01 and C = 2.
```

Hide

```
# How well does it do?
y.pred <- predict(svm_radial, newdata=x.pca.test)
sum(y.pred==y.test)/1000
```

```
[1] 0.828
```

Hide

```
library(MASS)
# To run lda here, your data is
# y=y and
# X = dat.pca.train
# For data (X,y), your command would then look like
# lda.fit = lda(y~., data= X)
y <-
X <- dat.pca.train

lda.fit = lda(y~., data= X)
# Run lda (using the above hints) and predict the
# labels on the test set via

lda.pred=predict(lda.fit, data.frame(x.pca.test))
lda.class=lda.pred$class

# Where do the errors occur?
table(lda.class,y.test)
```

```
         y.test
lda.class  0  1  2  3  4  5  6  7  8  9
        0 76  0  1  3  0  0 19  0  2  0
        1  0 95  0  0  0  0  0  0  0  0
        2  2  1 65  1  9  0  9  0  0  0
        3  6  8  1 80  7  0  8  0  1  0
        4  0  1 18  1 63  0  8  0  1  0
        5  0  0  0  0  0 87  0 17  4  6
        6  5  1 13  4 16  0 55  0  3  1
        7  0  0  0  0  0  5  0 91  1  5
        8  0  0  1  1  1  2  1  0 94  1
        9  0  0  0  0  0  5  0  9  0 85
```

Hide

```
# Performance?
sum(lda.class==y.test)/1000
```

```
[1] 0.791
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.