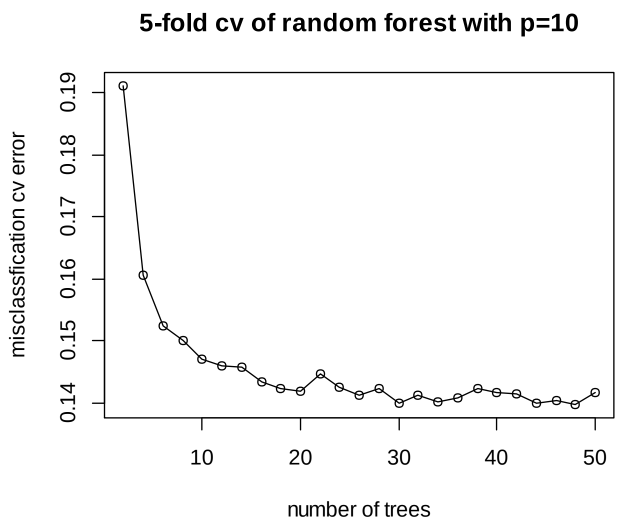
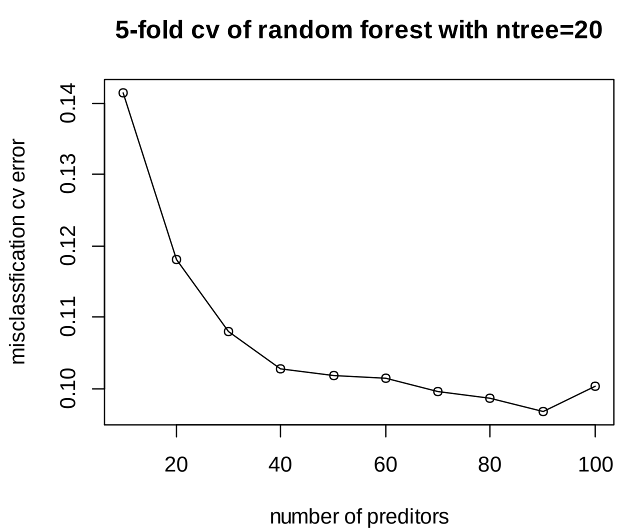
Part I. Classification on 20newsgroup Data

1. Fixing mtry=sqrt(100)=10 with different ntree (2 to 50, step = 2), the result is:



We can see that the misclassification cv error decreases as number of tree increase until ntree = 20, and the error does not vary much for ntree>20.

Fixing ntree = 20 with different mtry (10 to 100, step=10), the result is:



The best CV error was 0.0967375 with ntree = 20, mtry = 90.

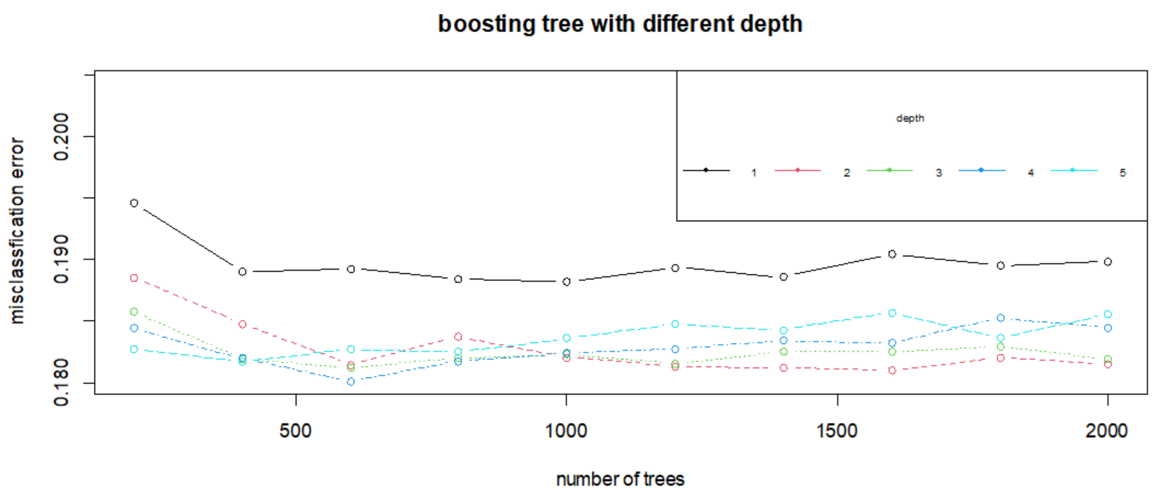
A random forest model is retrained with ntree=20 and mtry=90, the misclassification error is 0.09826376 and the corresponding confusion matrix is:

```
                  newsgroup
predicted_newsgroup    1    2    3    4
                  1 4367  205  271  158
                  2   90 3096  119  131
                  3   73   78 2111  100
                  4   75  140  156 5072
```

The higher the Gini index, the more important the variable is. The 10 most important keywords are:

Keywords	Gini Index
windows	720.1825
god	530.8854
government	443.3101
team	418.4419
car	373.9237
christian	371.5577
jews	262.3375
space	239.3162
baseball	234.6961
graphics	216.2313

2. Boosting trees were built with ntree=2000, interation.depth from 1 to 5 and the default shrinkage=0.1 using 5-fold cross validation. The misclassifaction cv error are plotted as below:



Best CV error was 0.18 with ntree=600 and interation.depth =4.

A boosting tree model is retrained with ntree=600, interation.depth =4 and shrinkage=0.1, the misclassification error is 0.1520749 and the corresponding confusion matrix is:

```
                 newsgroup
predicted_newsgroup    1    2    3    4
                  1 4220  254  476  216
                  2   63 2821   96  101
                  3  153  129 1781  194
                  4  169  315  304 4950
```

3. In general, boosting tree model outperforms random forest when it is well tuned. However, for this dataset, random forest is a better model than boosting tree. One reason is that the data contains much noise which caused boosting tree model to over fit.

4. Building a multi-class LDA classifier, the 5-fold CV error of misclassification is 0.202045. A multi-class LDA classifier is retrained with all data and the confusion matrix is:

predicted_newsgroup \ newsgroup	1	2	3	4
1	4102	341	620	268
2	45	2607	119	125
3	239	181	1530	299
4	219	390	388	4769

5. When building a multi-class QDA classifier with all variables, a rank deficiency error is given. The error was due to the covariance matrix is not invertible in some groups. It is observed that values of certain variables in certain group are all 0, which means all records of certain group do not contain such keywords. To fix this error, we should not include any keywords that are all 0 in particular group in the qda model. The following keywords are identified from the groups of each folds of the cross validation that should be removed:

```
> unique(word_not_use)
 [1] "bmw"      "hockey"   "jews"     "lunar"    "nhl"      "puck"     "vitamin"  "aids"
 [9] "bible"    "dos"      "israel"   "orbit"    "patients" "scsi"     "shuttle"  "honda"
[17] "jesus"    "mars"     "dealer"   "season"
```

Thus, building a multi-class QDA classifier with all but the above variables, the 5-fold CV error of misclassification is 0.2738134. A multi-class LDA classifier is retrained with all data and the confusion matrix is:

predicted_newsgroup \ newsgroup	1	2	3	4
1	4246	753	1075	738
2	133	2464	217	432
3	149	88	1058	198
4	77	214	307	4093

6. Both LDA and QDA did not perform better than random forest and boosting tree. LDA has large bias while some important variables (e.g. “jews”) could not be used in QDA due to its limitation which lowered the prediction accuracy.

Part II. Spectral Clustering on 20newsgroup Data

1. The mis-clustering error rate using top 4 left singular vectors from PCA and K-means with K=4 is $1 - \frac{4567+1056+336+2102}{16242} = 0.503694$.
2. The mis-clustering error rate using top 5 left singular vectors from PCA and K-means with K=4 is $1 - \frac{4573+1054+338+2247}{16242} = 0.494397$.

```
> doc.pc=prcomp(doc.table, scale=TRUE)
> set.seed(1)
> doc.km4=kmeans(doc.pc$x[,1:4],4,nstart=20)
> require(plyr)
> doc.km4$cluster=mapvalues(doc.km4$cluster,from=c(1,2,3,4),to=c(2,1,4,3))
> table(doc.km4$cluster,doc.group[,1])

       1    2    3    4
  1 4567 2418 1996 3341
  2    3 1056    5    7
  3   18    0  336   11
  4   17   45  320 2102
> doc.km5=kmeans(doc.pc$x[,1:5],4,nstart=20)
> doc.km5$cluster=mapvalues(doc.km5$cluster,from=c(1,2,3,4),to=c(4,3,2,1))
> table(doc.km5$cluster,doc.group[,1])

       1    2    3    4
  1 4573 2429 2138 3195
  2    3 1054    5    8
  3   18    0  338   11
  4   11   36  176 2247
```

3. The performances using PCA with the top 4 and 5 left singular vectors for clustering is much worse than the method used from part I. One reason is that the top 4 and top 5 only account for 13.7% and 16% of total variance of the data. Another reason is simply K-means is not suitable for clustering in this dataset which may be due to the way that K-means measuring the distance does not align with how the groups are separated.

```
> summary(doc.pc)
Importance of components:
                           PC1     PC2     PC3     PC4     PC5
Standard deviation     2.00581 1.92846 1.76839 1.68292 1.53043
Proportion of Variance 0.04023 0.03719 0.03127 0.02832 0.02342
Cumulative Proportion  0.04023 0.07742 0.10869 0.13702 0.16044
```

Part III. Classification on MNIST Data

1. Using 5-fold cross validation of linear kernel, the cost parameter has been tuned for 53.7 seconds in which the best cost is 0.1.

```
> mnist.train=read.csv("train_resized.csv",header=TRUE)
> mnist.test=read.csv("test_resized.csv",header=TRUE)
> mnist.train$label=as.factor(mnist.train$label)
> mnist.test$label=as.factor(mnist.test$label)
> mnist.train.i36=mnist.train[mnist.train[,1]==3 | mnist.train[,1]==6,]
> mnist.test.i36=mnist.test[mnist.test[,1]==3 | mnist.test[,1]==6,]
> library(e1071)
> set.seed(1)
> tc=tune.control(cross = 5)
> start_time=Sys.time()
> tune.out=tune(svm,label~.,data=mnist.train.i36,kernel="linear",tunecontrol=tc,ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,10
0)))
> end_time=Sys.time()
> end_time-start_time
Time difference of 53.71672 secs
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 5-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.003816051

- Detailed performance results:
   cost       error  dispersion
1 1e-03 0.005641777 0.002712912
2 1e-02 0.003816051 0.002594954
3 1e-01 0.005475665 0.003190432
4 1e+00 0.007964878 0.002660070
5 5e+00 0.008130853 0.002444353
6 1e+01 0.008130853 0.002444353
7 1e+02 0.008130853 0.002444353
```

The misclassification error on the test data is $\frac{6+11}{1251+6+11+1194} = 0.69\%$ and the confusion matrix is:

```
> bestmod=tune.out$best.model
> ypred=predict(bestmod ,mnist.test.i36)
> table(predict=ypred, truth=mnist.test.i36$label)
       truth
predict    0    1    2    3    4    5    6    7    8    9
      0    0    0    0    0    0    0    0    0    0    0
      1    0    0    0    0    0    0    0    0    0    0
      2    0    0    0    0    0    0    0    0    0    0
      3    0    0    0 1251    0    0    6    0    0    0
      4    0    0    0    0    0    0    0    0    0    0
      5    0    0    0    0    0    0    0    0    0    0
      6    0    0    0   11    0    0 1194    0    0    0
      7    0    0    0    0    0    0    0    0    0    0
      8    0    0    0    0    0    0    0    0    0    0
      9    0    0    0    0    0    0    0    0    0    0
```

2. Using 5-fold cross validation of radial kernel, the cost and gamma parameters have been tuned for 12 mins in which the best (cost, gamma) pair is (100, 0.0001).

```
> start_time=Sys.time()
> tune.out_1=tune(svm,label~.,data=mnist.train.i36,kernel="radial",tunecontrol=tc,ranges=list(cost=c(1,10,100,1000),gamma=c(0.
0001,0.001,0.01,0.1)))
> end_time=Sys.time()
> end_time-start_time
Time difference of 12.02154 mins
> summary(tune.out_1)

Parameter tuning of ‘svm’:

- sampling method: 5-fold cross validation

- best parameters:
 cost gamma
  100 1e-04

- best performance: 0.004314802

- Detailed performance results:
   cost gamma       error  dispersion
1     1 1e-04 0.008961417 0.001486086
2    10 1e-04 0.005144540 0.001799604
3   100 1e-04 0.004314802 0.002517410
4  1000 1e-04 0.006305815 0.002390097
5     1 1e-03 0.005310790 0.002391457
6    10 1e-03 0.004646890 0.003246419
7   100 1e-03 0.005144815 0.003776069
8  1000 1e-03 0.004978978 0.003936735
9     1 1e-02 0.009127943 0.004066597
10   10 1e-02 0.008463905 0.002899872
11  100 1e-02 0.008463905 0.002899872
12 1000 1e-02 0.008463905 0.002899872
13    1 1e-01 0.132591675 0.002932574
14   10 1e-01 0.123631221 0.001344517
15  100 1e-01 0.123631221 0.001344517
16 1000 1e-01 0.123631221 0.001344517
```

The misclassification error on the test data is $\frac{4+10}{1252+4+10+1196} = 0.57\%$ and the confusion matrix is:

```
> bestmod_r1=tune.out_1$best.model
> ypred=predict(bestmod_r1 ,mnist.test.i36)
> table(predict=ypred, truth=mnist.test.i36$label)
       truth
predict    0    1    2    3    4    5    6    7    8    9
      0    0    0    0    0    0    0    0    0    0    0
      1    0    0    0    0    0    0    0    0    0    0
      2    0    0    0    0    0    0    0    0    0    0
      3    0    0    0 1252    0    0    4    0    0    0
      4    0    0    0    0    0    0    0    0    0    0
      5    0    0    0    0    0    0    0    0    0    0
      6    0    0    0   10    0    0 1196    0    0    0
      7    0    0    0    0    0    0    0    0    0    0
      8    0    0    0    0    0    0    0    0    0    0
      9    0    0    0    0    0    0    0    0    0    0
```

3. Both models above perform equally well for this problem when the parameters are carefully tuned while it takes longer to tuned the model with radial kernel.

4. Using 5-fold cross validation of linear kernel, the cost parameter has been tuned for 9 mins in which the best cost is 0.1.

```
> mnist.train.i1258=mnist.train[mnist.train[,1]==1 | mnist.train[,1]==2 | mnist.tra
in[,1]==5 | mnist.train[,1]==8,]
> mnist.test.i1258=mnist.test[mnist.test[,1]==1 | mnist.test[,1]==2 | mnist.test[,
1]==5 | mnist.test[,1]==8,]
> start_time=Sys.time()
> tune.out.1258=tune(svm,label~.,data=mnist.train.i1258,kernel="linear",tunecontrol
=tc,ranges=list(cost=c(0.01,0.1,1,10,100)))

> end_time=Sys.time()
> end_time-start_time
Time difference of 8.997765 mins
> summary(tune.out.1258)

Parameter tuning of 'svm':

- sampling method: 5-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.03970427

- Detailed performance results:
   cost      error  dispersion
1 1e-02 0.04180286 0.004959515
2 1e-01 0.03970427 0.004464172
3 1e+00 0.04348173 0.003614521
4 1e+01 0.04902195 0.005449180
5 1e+02 0.05179178 0.004165644
```

The misclassification error on the test data is $1-\frac{1344+1140+1062+1041}{4806}=4.56\%$ and the confusion matrix is:

```
> bestmod_1258=tune.out.1258$best.model
> ypred=predict(bestmod_1258 ,mnist.test.i1258)
> table(predict=ypred, truth=mnist.test.i1258$label)
       truth
predict    0    1    2    3    4    5    6    7    8    9
      0    0    0    0    0    0    0    0    0    0    0
      1    0 1344    5    0    0   12    0    0   19    0
      2    0   11 1140    0    0   19    0    0   26    0
      3    0    0    0    0    0    0    0    0    0    0
      4    0    0    0    0    0    0    0    0    0    0
      5    0    2   16    0    0 1062    0    0   46    0
      6    0    0    0    0    0    0    0    0    0    0
      7    0    0    0    0    0    0    0    0    0    0
      8    0    6   24    0    0   33    0    0 1041    0
      9    0    0    0    0    0    0    0    0    0    0
```

5. Using 5-fold cross validation of linear kernel on full dataset, the cost parameter has been tuned for 56 mins in which the best cost is 0.1.

```
> start_time=Sys.time()
> options(warn=-1)
> tune.out.all=tune(svm,label~.,data=mnist.train,kernel="linear",tunecontrol=tc,ran
ges=list(cost=c(0.01,0.1,1,10,100)))
> end_time=Sys.time()
> end_time-start_time
Time difference of 55.55482 mins
> summary(tune.out.all)

Parameter tuning of ‘svm’:

- sampling method: 5-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.0626

- Detailed performance results:
   cost      error  dispersion
1 1e-02 0.06373333 0.003079051
2 1e-01 0.06260000 0.003001389
3 1e+00 0.06903333 0.003795831
4 1e+01 0.07366667 0.003717451
5 1e+02 0.07656667 0.003724170
```

The misclassification error on the test data is $1-\frac{1108+1341+1112+1136+1115+1023+1148+1183+1004+1058}{12000}=6.43\%$ and the confusion matrix is:

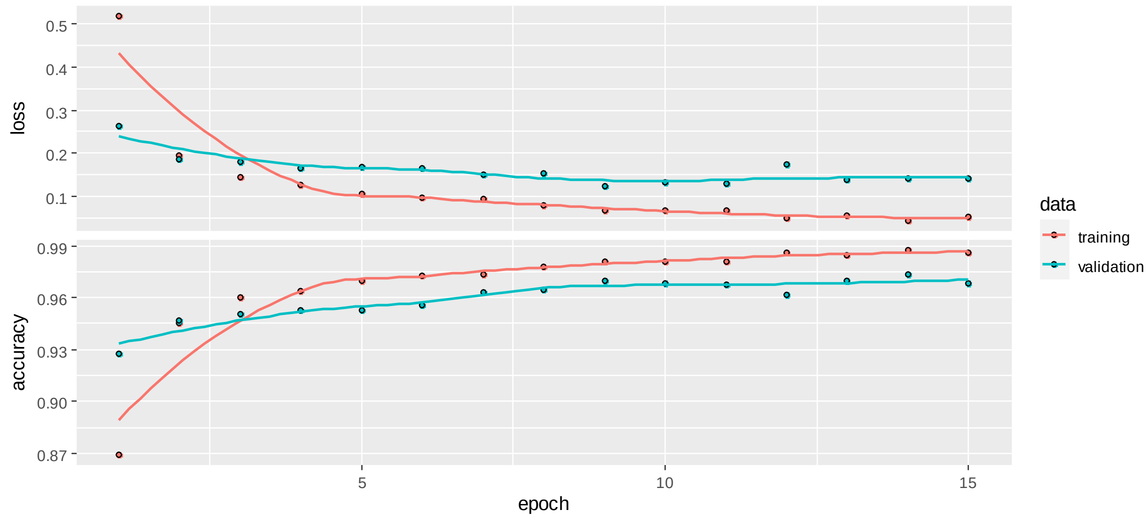
```
> bestmod_all=tune.out.all$best.model
> ypred=predict(bestmod_all ,mnist.test)
> table(predict=ypred, truth=mnist.test$label)
       truth
predict    0    1    2    3    4    5    6    7    8    9
      0 1108    0    5    1    2    7   10    1    5    4
      1    0 1341    2    7    3   12    3    3   16    6
      2    2    8 1112   38   12    6   12   13   22    8
      3    1    2    5 1136    0   33    0    4   24    9
      4    1    0   24    0 1115    3    8    7    5   26
      5   10    2    3   39    3 1023   13    5   35    6
      6   10    1    6    3    7   17 1148    0    2    0
      7    0    2    9    7    5    1    1 1183    3   29
      8    6    6   16   19    1   18    4    2 1004    7
      9    2    1    3   12   27    6    1   46   16 1058
```

Part IV. Additional Bonus

Neural network could be used for this dataset. A Multi-Layer Perceptron (MLP) with 6 hidden layer was trained as below:

```
> library(dplyr)
> library(keras)
> library(tensorflow)
> library(yardstick)
>
> data_train=read.csv("train_resized.csv", header=TRUE)
> data_test=read.csv("test_resized.csv", header=TRUE)
>
> train_x=as.matrix(data_train[,-1])
> test_x=as.matrix(data_test[,-1])
> train_y=to_categorical(data_train$label, num_classes = 10)
> test_y=data_test$label
>
> tf$random$set_seed(1)
> start_time=Sys.time()
> model=keras_model_sequential(name = "MLP_MNIST")
> layer_dense(model,units = 512, activation = "relu", input_shape = ncol(train_x), name = "Hidden_1")
> layer_dense(model,units = 256, activation = "relu", name = "Hidden_2")
> layer_dense(model,units = 128, activation = "relu", name = "Hidden_3")
> layer_dense(model,units = 64, activation = "relu", name = "Hidden_4")
> layer_dense(model,units = 32, activation = "relu", name = "Hidden_5")
> layer_dense(model,units = 16, activation = "relu", name = "Hidden_6")
> layer_dense(model,units = 10, activation = "softmax", name = "Out")
Model
Model: "MLP_MNIST"
________________________________________________________________________________
Layer (type)                          Output Shape                      Param #
================================================================================
Hidden_1 (Dense)                      (None, 512)                       74240
________________________________________________________________________________
Hidden_2 (Dense)                      (None, 256)                       131328
________________________________________________________________________________
Hidden_3 (Dense)                      (None, 128)                       32896
________________________________________________________________________________
Hidden_4 (Dense)                      (None, 64)                        8256
________________________________________________________________________________
Hidden_5 (Dense)                      (None, 32)                        2080
________________________________________________________________________________
Hidden_6 (Dense)                      (None, 16)                        528
________________________________________________________________________________
Out (Dense)                           (None, 10)                        170
================================================================================
Total params: 249,498
Trainable params: 249,498
Non-trainable params: 0
________________________________________________________________________________

> compile(model, loss="categorical_crossentropy", optimizer=optimizer_adam(lr=0.001), metrics="accuracy")
> history=fit(model, x=train_x, y=train_y, epochs=15, batch_size=32, validation_split=0.2, verbose=1)
Epoch 1/15
750/750 [==============================] - 3s 3ms/step - loss: 1.1758 - accuracy: 0.7621 - val_loss: 0.2616 - val_accuracy: 0.9278
Epoch 2/15
750/750 [==============================] - 3s 3ms/step - loss: 0.1966 - accuracy: 0.9442 - val_loss: 0.1845 - val_accuracy: 0.9467
Epoch 3/15
750/750 [==============================] - 2s 3ms/step - loss: 0.1408 - accuracy: 0.9595 - val_loss: 0.1795 - val_accuracy: 0.9508
Epoch 4/15
750/750 [==============================] - 3s 3ms/step - loss: 0.1316 - accuracy: 0.9635 - val_loss: 0.1651 - val_accuracy: 0.9527
Epoch 5/15
750/750 [==============================] - 2s 3ms/step - loss: 0.1015 - accuracy: 0.9713 - val_loss: 0.1687 - val_accuracy: 0.9528
Epoch 6/15
750/750 [==============================] - 3s 3ms/step - loss: 0.0836 - accuracy: 0.9753 - val_loss: 0.1652 - val_accuracy: 0.9555
Epoch 7/15
750/750 [==============================] - 2s 3ms/step - loss: 0.0828 - accuracy: 0.9762 - val_loss: 0.1496 - val_accuracy: 0.9633
Epoch 8/15
750/750 [==============================] - 3s 3ms/step - loss: 0.0679 - accuracy: 0.9807 - val_loss: 0.1533 - val_accuracy: 0.9643
Epoch 9/15
750/750 [==============================] - 2s 3ms/step - loss: 0.0653 - accuracy: 0.9811 - val_loss: 0.1227 - val_accuracy: 0.9697
Epoch 10/15
750/750 [==============================] - 3s 3ms/step - loss: 0.0534 - accuracy: 0.9852 - val_loss: 0.1334 - val_accuracy: 0.9683
Epoch 11/15
750/750 [==============================] - 2s 3ms/step - loss: 0.0625 - accuracy: 0.9814 - val_loss: 0.1299 - val_accuracy: 0.9675
Epoch 12/15
750/750 [==============================] - 2s 3ms/step - loss: 0.0480 - accuracy: 0.9860 - val_loss: 0.1738 - val_accuracy: 0.9613
Epoch 13/15
750/750 [==============================] - 2s 3ms/step - loss: 0.0524 - accuracy: 0.9856 - val_loss: 0.1377 - val_accuracy: 0.9700
Epoch 14/15
750/750 [==============================] - 2s 3ms/step - loss: 0.0413 - accuracy: 0.9881 - val_loss: 0.1422 - val_accuracy: 0.9732
Epoch 15/15
750/750 [==============================] - 2s 3ms/step - loss: 0.0520 - accuracy: 0.9867 - val_loss: 0.1416 - val_accuracy: 0.9685
> end_time=Sys.time()
> end_time-start_time
Time difference of 39.05762 secs
> plot(history)
```



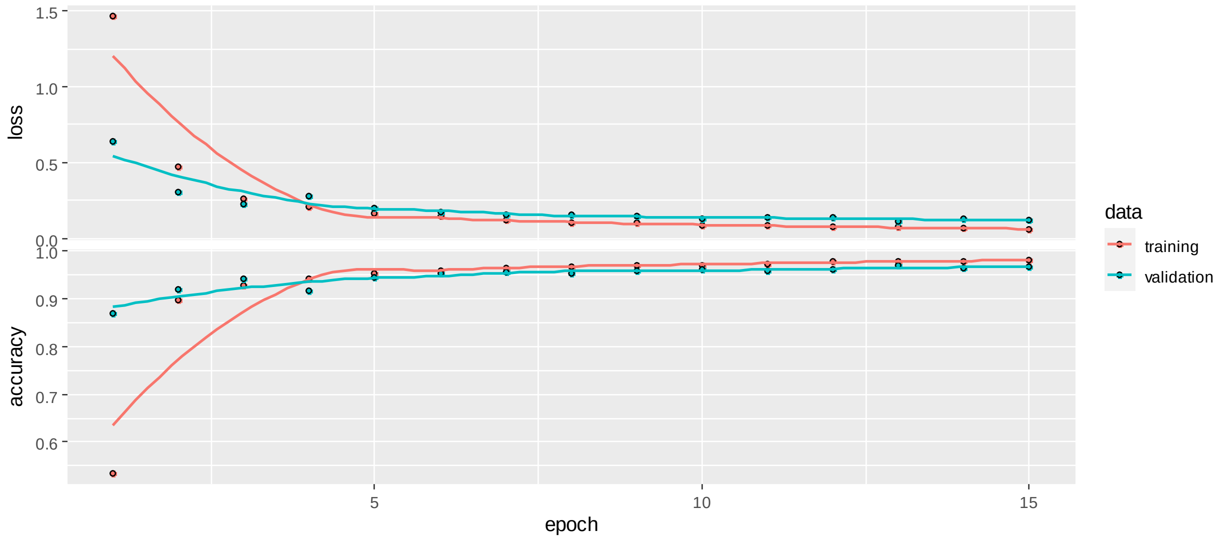
The misclassification error of the test data is 2.9% and the training time of the model is 39 seconds. The confusion matrix is:

```
> pred_test=predict_classes(model, test_x)
> error_mlp=1-accuracy_vec(truth = as.factor(data_test$label), estimate = as.factor(pred_test))
> error_mlp
[1] 0.029
> table(prediction = as.factor(pred_test), truth = as.factor(data_test$label))
          truth
prediction    0    1    2    3    4    5    6    7    8    9
         0 1120    0    4    0    3    3    2    1    1    3
         1    0 1340    2    1    1    2    4    1   11    2
         2    0    9 1148    4    0    0    1    6    5    2
         3    1    3    7 1234    0   25    0    0   20    5
         4    0    2    3    0 1125    0    2    1    1    8
         5    2    0    0    6    1 1075    9    0    8    2
         6    7    0    1    0    7    4 1177    0    1    0
         7    0    4   13   10    4    1    0 1242    5   13
         8    7    5    6    3    0    9    3    0 1075    2
         9    3    0    1    4   34    7    2   13    5 1116
```

The performance of neural network outperforms svm in part III. However, there are much more hyper-parameters need to be set for neural network (like number of hidden layers, number of neurons in each layer, activation function etc.) and it is hard to reach the optimal set of hyper-parameters. I have also trained a CNN model but the performance is similar to that of the MLP. The reason may be the dataset is not large enough.

```
> train_x_cnn=array_reshape(train_x, dim=c(nrow(train_x), 12, 12, 1))
> test_x_cnn=array_reshape(test_x, dim=c(nrow(test_x), 12, 12, 1))
>
> tf$random$set_seed(1)
> model1=keras_model_sequential(name = "CNN_Mnist")
> layer_conv_2d(model1,filters=32, kernel_size=c(2,2), padding="same", activation="relu", input_shape=c(12,12,1))
> layer_max_pooling_2d(model1, pool_size=c(2,2))
> layer_conv_2d(model1, filters=32, kernel_size=c(2,2), padding="same", activation="relu", input_shape=c(12,12,1))
> layer_max_pooling_2d(model1, pool_size=c(2,2))
> layer_conv_2d(model1, filters=32, kernel_size=c(2,2), padding="same", activation="relu", input_shape=c(12,12,1))
> layer_max_pooling_2d(model1, pool_size=c(2,2))
> layer_flatten(model1)
> layer_dense(model1,units=16, activation="relu")
> layer_dense(model1,units=10, activation="softmax", name="Output")
> model1
Model
Model: "CNN_Mnist"
________________________________________________________________________________________________________
Layer (type)                              Output Shape                          Param #
========================================================================================================
conv2d_3 (Conv2D)                         (None, 12, 12, 32)                    160
________________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)            (None, 6, 6, 32)                      0
________________________________________________________________________________________________________
conv2d_4 (Conv2D)                         (None, 6, 6, 32)                      4128
________________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)            (None, 3, 3, 32)                      0
________________________________________________________________________________________________________
conv2d_5 (Conv2D)                         (None, 3, 3, 32)                      4128
________________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)            (None, 1, 1, 32)                      0
________________________________________________________________________________________________________
flatten_1 (Flatten)                       (None, 32)                            0
________________________________________________________________________________________________________
dense_1 (Dense)                           (None, 16)                            528
________________________________________________________________________________________________________
Output (Dense)                            (None, 10)                            170
========================================================================================================
Total params: 9,114
Trainable params: 9,114
Non-trainable params: 0
________________________________________________________________________________________________________

> compile(model1, loss="categorical_crossentropy", optimizer=optimizer_adam(lr=0.001), metrics="accuracy")
> history=fit(model1, x=train_x_cnn, y=train_y, epochs=15, batch_size=32, validation_split=0.2, verbose=1)
Epoch 1/15
750/750 [==============================] - 7s 8ms/step - loss: 2.1869 - accuracy: 0.3098 - val_loss: 0.6360 - val_accuracy: 0.8702
Epoch 2/15
750/750 [==============================] - 6s 8ms/step - loss: 0.5400 - accuracy: 0.8891 - val_loss: 0.3081 - val_accuracy: 0.9188
Epoch 3/15
750/750 [==============================] - 7s 9ms/step - loss: 0.2684 - accuracy: 0.9244 - val_loss: 0.2251 - val_accuracy: 0.9408
Epoch 4/15
750/750 [==============================] - 7s 9ms/step - loss: 0.2027 - accuracy: 0.9443 - val_loss: 0.2769 - val_accuracy: 0.9177
Epoch 5/15
750/750 [==============================] - 7s 9ms/step - loss: 0.1698 - accuracy: 0.9520 - val_loss: 0.1983 - val_accuracy: 0.9440
Epoch 6/15
750/750 [==============================] - 9s 12ms/step - loss: 0.1423 - accuracy: 0.9596 - val_loss: 0.1752 - val_accuracy: 0.9537
Epoch 7/15
750/750 [==============================] - 8s 11ms/step - loss: 0.1154 - accuracy: 0.9666 - val_loss: 0.1554 - val_accuracy: 0.9552
Epoch 8/15
750/750 [==============================] - 9s 12ms/step - loss: 0.1000 - accuracy: 0.9689 - val_loss: 0.1566 - val_accuracy: 0.9547
Epoch 9/15
750/750 [==============================] - 9s 12ms/step - loss: 0.0922 - accuracy: 0.9714 - val_loss: 0.1466 - val_accuracy: 0.9592
Epoch 10/15
750/750 [==============================] - 7s 10ms/step - loss: 0.0806 - accuracy: 0.9736 - val_loss: 0.1328 - val_accuracy: 0.9610
Epoch 11/15
750/750 [==============================] - 9s 12ms/step - loss: 0.0772 - accuracy: 0.9756 - val_loss: 0.1395 - val_accuracy: 0.9585
Epoch 12/15
750/750 [==============================] - 6s 9ms/step - loss: 0.0739 - accuracy: 0.9786 - val_loss: 0.1376 - val_accuracy: 0.9612
Epoch 13/15
750/750 [==============================] - 9s 12ms/step - loss: 0.0716 - accuracy: 0.9781 - val_loss: 0.1136 - val_accuracy: 0.9692
Epoch 14/15
750/750 [==============================] - 9s 13ms/step - loss: 0.0640 - accuracy: 0.9804 - val_loss: 0.1321 - val_accuracy: 0.9647
Epoch 15/15
750/750 [==============================] - 7s 9ms/step - loss: 0.0572 - accuracy: 0.9827 - val_loss: 0.1193 - val_accuracy: 0.9673
> plot(history)
```




```
> pred_test=predict_classes(model1, test_x_cnn)
> error_cnn=1-accuracy_vec(truth = as.factor(data_test$label), estimate = as.factor(pred_test))
> error_cnn
[1] 0.03141667
> table(prediction = as.factor(pred_test), truth = as.factor(data_test$label))
          truth
prediction    0    1    2    3    4    5    6    7    8    9
         0 1119    0    7    0    4    1    1    1    8    5
         1    0 1347    3    2    3    0    1    2    9    1
         2    3    6 1145   21    3    0    1    5    8    1
         3    0    2    2 1198    0   13    0    5    4    1
         4    0    1    3    0 1134    0    4    4    1   11
         5    0    0    0    7    0 1079    1    0    6    6
         6   13    2    0    0    3   19 1188    0    3    0
         7    0    0    8    8    0    0    0 1211    0    3
         8    5    5   16   21    0   13    4    4 1087   10
         9    0    0    1    5   28    1    0   32    6 1115
```