1. Data pre-processing steps and the chosen evaluation method and measure(s)

Import data

```
library(caret)
library(tidyverse)
train.df=read_csv("C:\\Users\\Arnold\\OneDrive\\R_Python_working_directory\\IST 707 Data Analytics\\Kaggle-digit-train.csv")
```

Convert labels to factors

```
train.df$label=factor(train.df$label)
```

Split training data for fitting model and validation.

```
fit.idx=createDataPartition(train.df$label,p = .5,list = F)
fit.df=train.df[fit.idx,]
val.df=train.df[-fit.idx,]
```

Check the summary to identify some other problems

```
summary(fit.df[,c(1,2,sample(3:785,8))])
```

```
##      label         pixel0     pixel689         pixel315
##  1      :112   Min.   :0   Min.   :  0.00   Min.   :  0.00
##  7      :105   1st Qu.:0   1st Qu.:  0.00   1st Qu.:  0.00
##  3      :104   Median :0   Median :  0.00   Median :  0.00
##  2      :100   Mean   :0   Mean   : 17.87   Mean   : 42.21
##  9      :100   3rd Qu.:0   3rd Qu.:  0.00   3rd Qu.:  9.00
##  0      : 99   Max.   :0   Max.   :255.00   Max.   :255.00
##  (Other):384
##     pixel407        pixel666          pixel371         pixel534
##  Min.   :  0.0   Min.   :  0.000   Min.   :  0.00   Min.   :  0.0000
##  1st Qu.:  0.0   1st Qu.:  0.000   1st Qu.:  0.00   1st Qu.:  0.0000
##  Median :189.0   Median :  0.000   Median :  0.00   Median :  0.0000
##  Mean   :142.9   Mean   :  1.834   Mean   : 46.56   Mean   :  0.4313
##  3rd Qu.:253.0   3rd Qu.:  0.000   3rd Qu.: 23.25   3rd Qu.:  0.0000
##  Max.   :255.0   Max.   :253.000   Max.   :255.00   Max.   :223.0000
##
##     pixel443         pixel190
##  Min.   :  0.00   Min.   :  0.00
##  1st Qu.:  0.00   1st Qu.:  0.00
##  Median :  0.00   Median :  0.00
##  Mean   : 15.41   Mean   : 23.92
##  3rd Qu.:  0.00   3rd Qu.:  0.00
##  Max.   :254.00   Max.   :255.00
##
```

A lot of pixels are mostly 0, & some are even all 0.

Remove pixels with all 0's, because they provide no values.

```
fit.df=fit.df[,c(T,colSums(fit.df[,-1])>0)]
```

2. Build kNN, SVM, and Random Forest models.

The training control method

```
ctr=trainControl(method = 'cv',number = 3,allowParallel = T)
```

Fitting KNN model

```
(knn=train(label~.,fit.df,method='knn',trControl=ctr,tuneGrid=expand.grid(k=1
:6)))
```

```
## k-Nearest Neighbors
##
## 1004 samples
##  614 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 670, 669, 669
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   1  0.8824888  0.8693479
##   2  0.8496172  0.8327891
##   3  0.8655465  0.8504937
##   4  0.8705336  0.8560273
##   5  0.8615694  0.8460540
##   6  0.8625704  0.8471542
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```
val.df$knn.predict=predict(knn,val.df)
knn.score=postResample(pred = val.df$knn.predict,obs = val.df$label)[1]
```

Create dataframe for models comparison

```
mod.com=data.frame(Model='KNN',`Test Accuracy`=knn.score,row.names = NULL)
```

Fitting SVM Linear model

```
(lsvm=train(label ~ .,fit.df,method = "svmLinear",trControl = ctr,tuneLength=
expand.grid(C=seq(0.0001,1,length.out = 3)),scale=F))
```

```
## Support Vector Machines with Linear Kernel
##
## 1004 samples
```

```
##  614 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 671, 667, 670
## Resampling results:
##
##   Accuracy   Kappa
##   0.8705157  0.8560777
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
val.df$lsvm.predict=predict(lsvm,val.df)
lsvm.score=postResample(pred = val.df$lsvm.predict,obs = val.df$label)[1]
```

I tried a lot of different C values, but they all have same results, so I'll try kernal tricks.

Add a row to models comparison dataframe

```
mod.com=data.frame(Model='SVM Linear',`Test Accuracy`=lsvm.score,row.names =
NULL) %>% rbind(mod.com)
```

Fitting Kernal SVM model

```
(ksvm=train(label ~ ., data =
fit.df,tuneGrid=expand.grid(C=seq(1.75,4,length.out =
6)),method= "svmRadialCost",trControl=ctr,scale=F))
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1004 samples
##  614 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 669, 671, 668
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   1.75  0.9133306  0.9036647
##   2.20  0.9183028  0.9091941
##   2.65  0.9183028  0.9091941
##   3.10  0.9183028  0.9091945
##   3.55  0.9192978  0.9102998
##   4.00  0.9163038  0.9069714
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 3.55.
```

```
val.df$ksvm.predict=predict(ksvm,val.df)
ksvm.score=postResample(pred = val.df$ksvm.predict,obs = val.df$label)[1]
```

Add a row to models comparison dataframe

```
mod.com=data.frame(Model='Kernal SVM',`Test Accuracy`=ksvm.score,row.names =
NULL) %>% rbind(mod.com)
```

Fitting Random Forest model

```
(rf=train(label ~ ., data = fit.df,tuneGrid=expand.grid(mtry=8:11),method=
"rf",trControl=ctr))

## Random Forest
##
## 1004 samples
##  614 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 670, 670, 668
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    8    0.8854969  0.8726999
##    9    0.8924532  0.8804326
##   10    0.8974313  0.8859689
##   11    0.8854969  0.8727005
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 10.

val.df$rf.predict=predict(rf,val.df)
rf.score=postResample(pred = val.df$rf.predict,obs = val.df$label)[1]
```

Add a row to models comparison dataframe

```
mod.com=data.frame(Model='Random Forest',`Test Accuracy`=rf.score,row.names =
NULL) %>% rbind(mod.com)
```

3. Report

The generated models and their performance.

```
mod.com

##           Model Test.Accuracy
## 1 Random Forest     0.9079910
## 2    Kernal SVM     0.9198702
## 3    SVM Linear     0.8840619
## 4           KNN     0.8840863
```

```
resamples(list(Random.Forest = rf, KNN = knn, SVMLinear = lsvm,SVMRBF =
ksvm)) %>%
  summary()

##
## Call:
## summary.resamples(object = .)
##
## Models: Random.Forest, KNN, SVMLinear, SVMRBF
## Number of resamples: 3
##
## Accuracy
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## Random.Forest 0.8869048 0.8925542 0.8982036 0.8974313 0.9026946 0.9071856
## KNN           0.8656716 0.8731343 0.8805970 0.8824888 0.8908973 0.9011976
## SVMLinear     0.8532934 0.8643322 0.8753709 0.8705157 0.8791269 0.8828829
## SVMRBF        0.9044776 0.9101968 0.9159159 0.9192978 0.9267080 0.9375000
##
## Kappa
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## Random.Forest 0.8742589 0.8805345 0.8868101 0.8859689 0.8918239 0.8968376
## KNN           0.8506790 0.8589391 0.8671992 0.8693479 0.8786824 0.8901655
## SVMLinear     0.8369758 0.8492006 0.8614255 0.8560777 0.8656287 0.8698319
## SVMRBF        0.8938214 0.9001851 0.9065488 0.9102998 0.9185391 0.9305293
```

They are all very good models based on the test accuracy. We see the best performing model is the Kernal SVM. We also see the cross validation results supporting our conclusion.The results make sense. Kernal SVM has the best performance, because of the nature of our data. We have different hand written numbers to classify. For any particular number, the hand written pattern should be fairly close to each other. Since they are close to each other, it should fairly easy to seperate them with hyper planes. We have kernal SVM performing better than linear SVM, because kernal SNM has more dimensions than linear SVM I think. As for Random Forest and KNN, they both perform good as well, but just not as good as kernal SVM.Compare to the test accuracy of 70% for Naive Bayes, and 80% for Decision Tree, the models we built this time perform much better.