Applied Data Science: Midterm Project

Group 8

13 Mar 2019

```
n.prop <- c(0.02,0.05,0.1)
iterations <- 3
```

```
train.file <- "../Data/MNIST-fashion training set-49.csv"
test.file <- "../Data/MNIST-fashion testing set-49.csv"
dt_train <- fread(input = train.file, verbose = FALSE)
dt_test <- fread(input = test.file, verbose = FALSE)
```

```
# Convert the "label" variable from character to factor and see the distribution of each lev
dt_train$label <- as.factor(dt_train$label)
dt_test$label <- as.factor(dt_test$label)

table(dt_train$label)
```

```
 Ankle boot         Bag        Coat       Dress    Pullover      Sandal 
       6000        6000        6000        6000        6000        6000 
      Shirt     Sneaker T-shirt/top     Trouser 
       6000        6000        6000        6000 
```

```
table(dt_test$label)
```

```
 Ankle boot         Bag        Coat       Dress    Pullover      Sandal 
       1000        1000        1000        1000        1000        1000 
      Shirt     Sneaker T-shirt/top     Trouser 
       1000        1000        1000        1000 
```

```
# Check the shape of the two datasets
dim(dt_train)
```

```
[1] 60000    50
```

```
dim(dt_test)
```

```
[1] 10000    50
```

```
func_sample <- function(data,n,iter){
  # Sample from original data
  # Each sample is with `n` size, and iterates for `iter` times
  # Return a list contains `iter` data.table
  size <- n*iter
  len <- nrow(data)
  if (size >= len){
    the.rows <- sample(x = 1:len, size = size, replace = TRUE)
    }
  else{
    the.rows <- sample(x = 1:len, size = size, replace = FALSE)
    }
  data.sample<-list()
  for (i in 1:iter){
    data.sample<-list.append(data.sample,data[the.rows[((i-1)*n+1):(i*n)],])
  }
  return(data.sample)
}

func_split.label.feature<-function(data){
  dt_label <- data[,"label"]
  dt_feature <- data[,-"label"]
```

```
  return(list(dt_label,dt_feature))
}

func_sample.name<-function(data,list_sample.length,list_size){
  # Generate Sample Names, Assign Sample Names w/ Samples, Return Sample Names
  list_name<-c()
  for (sample.length in list_sample.length){
    for (num in list_size){
      dt_sample<-func_sample(data,sample.length,length(list_size))[num][[1]]
      name<-paste("dat",sample.length,num,sep = "_")
      assign(name, dt_sample,envir = .GlobalEnv)
      list_name<-c(list_name,name)
    }
  }
  return(list_name)
}

func_model.pred.convt <- function(model_pred_old){
  # Convert the GBM prediction results by match the max probability outcomes with the associ
    model_pred_old <- data.frame(model_pred_old)
    categories <- c("Ankle boot", "Bag", "Coat", "Dress", "Pullover",
                    "Sandal", "Shirt", "Sneaker", "T-shirt/top", "Trouser")
    model_pred_new <- list()
    for(i in 1:nrow(model_pred_old)){
      model_pred_new <- append(model_pred_new, categories[which.max(model_pred_old[i, ])])
    }
    return(unlist(data.table(model_pred_new)))
}

func_grade <- function(model.name,data.name,time.diff,accurancy){
  # Generate the result table with final grade
  time.diff <- as.numeric(x=time.diff, units="secs")
  time.grade <- round(min(1,time.diff/60), 4)
  row.num <- nrow(get(data.name))
  sample.grade <- round(row.num/nrow(dt_train), 4)
  misclassify <- round(1-accurancy, 4)
  grade <- round(0.25*time.grade + 0.25*sample.grade + 0.5*misclassify, 4)
  disp.row <- list("Model Name"=model.name, "Sample Size"=row.num, "Name"=data.name,
                   "A-Propt"=sample.grade, "B-Runtime"=time.grade, "C-Inaccu"=misclassify, '
  return(disp.row)
}

func_accuracy <- function(truth, pred) {
  # generate the confusion matrix
  cmat <- table(truth, pred)
  sum(diag(cmat))/sum(cmat)
}

round.numerics <- function(x, digits = 4){
  if(is.numeric(x)){x <- round(x = x, digits = digits)}
  return(x)
}
```

```
# Randomly generate samples from training set
n.values <- n.prop * nrow(dt_train)
n.values
```

```
[1] 1200 3000 6000
```

```
list_sample.names <- func_sample.name(dt_train, n.values, c(1, 2, 3))
list_sample.names
```

```
[1] "dat_1200_1" "dat_1200_2" "dat_1200_3" "dat_3000_1" "dat_3000_2"
[6] "dat_3000_3" "dat_6000_1" "dat_6000_2" "dat_6000_3"
```

Introduction

In this project, the team of Group 8 performed a grand tour of 10 machine learning techniques on a simplified image recognition task. Upon building various model functions, training and predicting, parameter tuning, and performance evaluation, we obtained hands-on coding and debugging experience plus further algorithm understanding. Detailed model explanation and comparison will be demonstrated in the Model and Discussion sections.

Models

Model 1: Multinomial Logistic Regression

```
func_mlr <- function(data.name){
  require(nnet)

  toc <- Sys.time()
  model.mlr <- multinom(label ~ ., data = get(data.name),trace=FALSE)
  pred.mlr <- predict(model.mlr, newdata = dt_test[,-1])
  tic <- Sys.time()

  accu.mlr <- mean(pred.mlr == dt_test$label)

  return(func_grade("Multinomial Logistic Regression",data.name,tic-toc,accu.mlr))
}
```

```
# Display results for the 9 samples
result_mlr <- data.table()
for (dat in list_sample.names){result_mlr <- rbind(result_mlr, func_mlr(dat))}
datatable(result_mlr, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Multinomial Logistic Regression	1200	dat_1200_1	0.02	0.0252	0.2621	0.1424
Multinomial Logistic Regression	1200	dat_1200_2	0.02	0.0319	0.2755	0.1507
Multinomial Logistic Regression	1200	dat_1200_3	0.02	0.0305	0.3068	0.166
Multinomial Logistic Regression	3000	dat_3000_1	0.05	0.0708	0.2431	0.1518
Multinomial Logistic Regression	3000	dat_3000_2	0.05	0.0719	0.2222	0.1416
Multinomial Logistic Regression	3000	dat_3000_3	0.05	0.0729	0.2351	0.1483
Multinomial Logistic Regression	6000	dat_6000_1	0.1	0.2288	0.2292	0.1968
Multinomial Logistic Regression	6000	dat_6000_2	0.1	0.2629	0.2591	0.2203
Multinomial Logistic Regression	6000	dat_6000_3	0.1	0.2657	0.238	0.2104

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: Multinomial logistic regression is similar to the logistic regression with more than 2 responses. Model the log odds as a linear combination of predictors.

Advantages: Multinomial logistic regression doesn't require the assumption about the distribution of predictors. It has a high accuracy of fitting the train data.

Disadvantages: When there exists collinearity or small sample size, the coefficients become unstable.

Model 2: Ensemble Models

```
func_ensemble<-function(data.name){
  require(caret)
  require(kernlab)

  data=get(data.name)

  # Without Cross Validation
  fitControl.base <- trainControl(method="none",savePredictions = TRUE,classProbs=TRUE)
  fitControl.top <- trainControl(method="none")

  toc <- Sys.time()

  # Base Model 1: Random Forest
```

```
  model.base.rf<-train(make.names(label)~. ,data = data, method="rf",trControl=fitControl.ba
  train.base.rf<-predict(model.base.rf, newdata = data[,-1], type="prob")
  test.base.rf<-predict(model.base.rf, newdata = dt_test[,-1], type="prob")

  # Base Model 2: Multinomial Logistic Regression
  model.base.logit<-train(make.names(label)~. ,data = data, method="multinom",trControl=fitC
  train.base.logit<-predict(model.base.logit, newdata = data[,-1], type="prob")
  test.base.logit<-predict(model.base.logit, newdata = dt_test[,-1], type="prob")

  # Process Output of Base Models as Input of Top Model
  # Average Probability Outputs of Two Base Models
  train.top.input<-cbind(data[,1],(train.base.rf+train.base.logit)/2)
  train.top.input$label<-make.names(train.top.input$label)
  test.top.input<-cbind(dt_test[,1],(test.base.rf+test.base.logit)/2)
  test.top.input$label<-make.names(test.top.input$label)

  # Top Model: SVM with RBF Kernel
  model.top.svm<-train(label~., data = train.top.input, method="svmRadial",trControl=fitCont
  top.pred<-predict(model.top.svm, newdata = test.top.input[,-1])

  tic <- Sys.time()

  summary.top<-confusionMatrix(as.factor(unlist(test.top.input[,1])), top.pred)
  accu.ensemble<-unname(summary.top$overall["Accuracy"])

  return(func_grade("Ensemble Models",data.name,tic-toc,accu.ensemble))
}
```

```
# Display results for the 9 samples
result_ensemble <- data.table()
for (dat in list_sample.names){result_ensemble <- rbind(result_ensemble, func_ensemble(dat))
datatable(result_ensemble, rownames = FALSE)
```

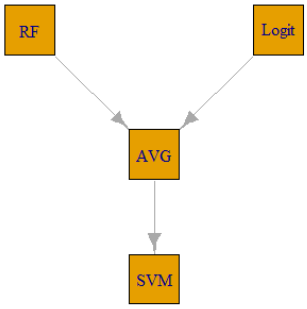
Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Ensemble Models	1200	dat_1200_1	0.02	0.2492	0.1984	0.1665
Ensemble Models	1200	dat_1200_2	0.02	0.2576	0.2009	0.1698
Ensemble Models	1200	dat_1200_3	0.02	0.2265	0.2018	0.1625
Ensemble Models	3000	dat_3000_1	0.05	0.5002	0.179	0.227
Ensemble Models	3000	dat_3000_2	0.05	0.428	0.1791	0.209
Ensemble Models	3000	dat_3000_3	0.05	0.4949	0.1808	0.2266
Ensemble Models	6000	dat_6000_1	0.1	1	0.169	0.3595
Ensemble Models	6000	dat_6000_2	0.1	1	0.165	0.3575
Ensemble Models	6000	dat_6000_3	0.1	0.973	0.1683	0.3524

Showing 1 to 9 of 9 entries Previous 1 Next

Idea: We use Random Forest and Logistic Regression as our base models, each of them outputs probability of predictions. Then we average those probabilities and feed it to top SVM model.

```
library(igraph)
node <- c("RF","Logit","SVM","AVG")
edge <- data.frame("From"=c("AVG","RF","Logit"),"To"=c("SVM","AVG","AVG"))
net <- graph_from_data_frame(d=edge, vertices=node, directed=T)
a <- layout_with_sugiyama(net)
plot(net,vertex.shape="square",vertex.size=40,layout=a$layout[,1:2])
```



Procedure: 1. Train RF and Logit on train data, then use them to predict on both train data and test data, output prediction prabability only. 2. Average the train prediction probability as a new train data. 3. Average the test prediction probability as a new test data. 4. Train SVM on new train data. 5. Test SVM on new test data.

Model Explaination: As the result shows, for ensembled model we have a higher accuracy comparing to RF and Logit, however, not much changes than single SVM. This might due to that SVM has reached its limits so combining other models as input would hardly increase its performance. Moreover, we obtain a dramatically increase in training time, especially for large data set. As a result, the (bad) grade of ensembled model is higher than others.

Parameters Choice: RF, Logit and SVM paramters are the same as their individual models (explained in individual models). This is to keep consistent in making comparisons.

Advantages: 1. Reduce over-fitting: Combining no regularization models with regularization models could avoid over-fitting. 2. Dimension reduction: Our input is a 49 pixel data and if we train our model in base layer, our top layer only takes 10 features as input. A time consuming model should be put into top layer thus faster model could reduce its input dimension. 3. It could be customized to solve different problems.

Disadvantages: 1. Lack of interpretability: Hard for people to explain the model. Specifically in this model, we cannot provide why we average two probabilities instead of other methods. 2. Too many parameters: Each individual model has its own parameters so ensembled model has even more, how to choose them is a problem, cross-validation could be a choice but the cost is high. 3. Computationally Expensive.

Model 3: Classification Tree

```
func_rpart <- function(data.name, cp = 0.001){
  require(rpart)

  toc <- Sys.time()
  model_fit <- rpart(formula = label ~ ., data = get(data.name), cp = cp)
  model_pred <- predict(model_fit, newdata = dt_test[, -1], type = "class")
  tic <- Sys.time()

  accu.rpart <- mean(model_pred == dt_test$label)

  return(func_grade("Classification Tree",data.name,tic-toc,accu.rpart))
}
```

```
# Display results for the 9 samples
result_rpart <- data.table()
for (dat in list_sample.names){result_rpart <- rbind(result_rpart, func_rpart(dat))}
datatable(result_rpart, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Classification Tree	1200	dat_1200_1	0.02	0.0091	0.302	0.1583
Classification Tree	1200	dat_1200_2	0.02	0.0095	0.3207	0.1677
Classification Tree	1200	dat_1200_3	0.02	0.0086	0.3254	0.1698
Classification Tree	3000	dat_3000_1	0.05	0.0263	0.281	0.1596

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Classification Tree	3000	dat_3000_2	0.05	0.0276	0.2816	0.1602
Classification Tree	3000	dat_3000_3	0.05	0.0257	0.2874	0.1626
Classification Tree	6000	dat_6000_1	0.1	0.0615	0.2467	0.1637
Classification Tree	6000	dat_6000_2	0.1	0.0368	0.2561	0.1622
Classification Tree	6000	dat_6000_3	0.1	0.0329	0.2546	0.1605

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: A decision tree takes the entire dataset as input, and then calculates entropy of target variable as well as predictor attributes. (The entropy controls how a Decision Tree decides to split the data and draws its boundaries.) Then, it calculates the Information Gain (IG) of all attributes, and chooses the attribute with highest information gain as the root node. By repeating the same process on every branch till the decision node of each branch is finalized, it returns the classification results.

Parameters Choice: `cp` represents the complexity parameter, which sets the factor of the overall lack of fit after each split. It saves computing time by pruning off splits that are not that worthwhile. In this case, as we chose a relatively small sample size, the `cp` should be a smaller value to secure good prediction results. Yet a too small `cp` will increase computing time. Thus, after tested `cp = c(0.01, 0.001, 0.0001)`, we defined `cp = 0.001`.

Advantages: 1. Decision trees can be used for both classification and regression problems. 2. They are very intuitive - could be easily visulized and explained to business function teams. 3. Feature selection in decision tree is completed automatically - the top few nodes on which the tree is split

Disadvantages: Without proper pruning or limiting tree growth, they tend to overfit the training data.

Model 4: Random Forest

```
func_rf <- function(data.name, ntree = 50, mtry = 7){
  require(randomForest)

  toc <- Sys.time()
  model_fit <- randomForest(formula = label ~ ., data = get(data.name), ntree = ntree, mtry
  model_pred <- predict(model_fit, newdata = dt_test[, -1])
  tic <- Sys.time()

  accu.rf <- mean(model_pred == dt_test$label)

  return(func_grade("Random Forest",data.name,tic-toc,accu.rf))
}
```

```
# Display results for the 9 samples
result_rf <- data.table()
for (dat in list_sample.names){result_rf <- rbind(result_rf, func_rf(dat))}
datatable(result_rf, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Random Forest	1200	dat_1200_1	0.02	0.0052	0.2209	0.1168
Random Forest	1200	dat_1200_2	0.02	0.0049	0.2114	0.1119
Random Forest	1200	dat_1200_3	0.02	0.0047	0.2121	0.1122
Random Forest	3000	dat_3000_1	0.05	0.014	0.1876	0.1098
Random Forest	3000	dat_3000_2	0.05	0.0118	0.1896	0.1102
Random Forest	3000	dat_3000_3	0.05	0.012	0.1946	0.1128
Random Forest	6000	dat_6000_1	0.1	0.0286	0.1687	0.1165
Random Forest	6000	dat_6000_2	0.1	0.0303	0.1742	0.1197
Random Forest	6000	dat_6000_3	0.1	0.029	0.1718	0.1182

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: A random forest is an ensemble learning method that is built up upon a number of decision trees by randomly selecting k features from total n features where k<<n. Among the k features, it calculates the node d using the best split point, then splits the node into daughter nodes using the best split, and then repeats the previous two steps until leaf nodes are finalized. The model builds the whole forest by repeating the previous steps for n number of times to create n number of trees. In this case, we are solving a multiclass classification problem. The model returns the final results by majority vote with the bagging method.

Parameters Choice: 1. `ntree` represents the number of trees to grow. When tuning this parameter, we chose `ntree = c(10, 50, 100, 200)`, and ran the model. The results showed that when ntree was larger than 50, the prediction accuracy just increased slightly while the runtime increased more. Thus, to get a better score, we selected `ntree = 50`; 2. `mtry` represents the number of variables randomly sampled as candidates at each split. Here we set the `mtry = 7` which is the default parameter value (sqrt(p)) in random forest. p is the number of variables, which is 49 in this case. As we tested, increasing `mtry` would not improve the total score. 3. The `type` value is ignored here because random forest could recognize the class of the target label (which is converted to factor), and select the "classification" `type` automatically.

Advantages: 1. It can be used for both classification and regression problems. Additionally, randomForest will default to classification or regression depending on the class of the target. 2. Normally randomForest returns good prediction performance with very hand and easy to use algorithm. It searches for the best feature among a random subset of features, which results in a wide diversity that results in a better and more robust model. 3. As long as there are enough trees in the forest, the classifier wonâ€ t overfit the model.

Disadvantages: 1. To get a more accurate prediction, we need more trees. Yet a large number of trees leads to time inefficiency. 2. It's hard to visualize the results. Also some of the random methods may not be easily explained to nontechnical people.

Model 5: Gradient Boosting Machine

```
func_gbm <- function(data.name, ntree = 150){
  require(gbm)

  toc <- Sys.time()
  model_fit <- gbm(formula = label ~ ., data = get(data.name), distribution = "multinomial",
                   n.trees = ntree, interaction.depth = 1)
  model_pred <- predict(model_fit, newdata = dt_test[, -1], type = "response", n.trees = ntr
  model_pred <- func_model.pred.convt(model_pred)
  tic <- Sys.time()

  accu.gbm <- mean(model_pred == dt_test$label)

  return(func_grade("Gradient Boosting Machine",data.name,tic-toc,accu.gbm))
}
```

```
# Display results for the 9 samples
result_gbm <- data.table()
for (dat in list_sample.names){result_gbm <- rbind(result_gbm, func_gbm(dat))}
datatable(result_gbm, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Gradient Boosting Machine	1200	dat_1200_1	0.02	0.0768	0.2338	0.1411
Gradient Boosting Machine	1200	dat_1200_2	0.02	0.0952	0.23	0.1438
Gradient Boosting Machine	1200	dat_1200_3	0.02	0.1368	0.2308	0.1546
Gradient Boosting Machine	3000	dat_3000_1	0.05	0.2381	0.2155	0.1798
Gradient Boosting Machine	3000	dat_3000_2	0.05	0.2439	0.2116	0.1793
Gradient Boosting Machine	3000	dat_3000_3	0.05	0.2329	0.2182	0.1798
Gradient Boosting Machine	6000	dat_6000_1	0.1	0.4029	0.2102	0.2308

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Gradient Boosting Machine	6000	dat_6000_2	0.1	0.4882	0.2063	0.2502
Gradient Boosting Machine	6000	dat_6000_3	0.1	0.4874	0.2052	0.2494

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: Similar to random forest, Gradient Boosting Machine (GBM) is also an ensemble learning method - it predicts by combining the outputs from individual trees. The key difference is that random forests train each tree independently while GBMs build trees one at a time, where each new tree helps to correct errors made by previously trained tree.

Parameters Choice: 1. `ntree` represents the number of trees. When tuning this parameter, we selected `ntree = c(20, 50, 100, 150, 200)` to test the prediction accuracy and the computing time, and finally chose `ntree = 150` because it got the lowest score. 2. For cross validation, we set `cv.folds = 0`, same as the default value, because cv is really computationally expensive while it didn't add much to the prediction accuracy. 3. `distribution = "multinomial"` was defined in this case since we have multi-classifications. 4. `interaction.depth` represents the maximum depth of each tree. Since the sub-trees here should be shallow to save running time, we tested `interaction.depth = c(1, 2, 3)` and found out that depth increase just returned slightly accuracy increase but large runtime increase. Thus, we set `interaction.depth = 1`.

Advantages: 1. It can be used for both classification and regression problems. 2. Often provides reasonably good predictive accuracy.

Disadvantages: GBM generally takes longer running time because of the fact that the trees are built sequentially.

Model 6: Support Vector Machine

```
func_svm<-function(data.name){
  # RBF Kernel
  # However, test performance would not be affected. Avoid over-fitting!

  # Runtime Starts
  toc<-Sys.time()

  # Model
  model.svm<-svm(label~.,data=get(data.name),type="C-classification", kernel="radial",cost=1

  # Accurancy in test data set
  pred.svm<-predict(model.svm, dt_test[,-1])

  # Runtime Ends
  tic<-Sys.time()

  accu.svm<-classAgreement(table(pred = pred.svm, true = dt_test$label))$diag

  return(func_grade("SVM",data.name,tic-toc,accu.svm))
}
```

```
# Display results for the 9 samples
result_svm <- data.table()
for (dat in list_sample.names){result_svm <- rbind(result_svm, func_svm(dat))}
datatable(result_svm, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
SVM	1200	dat_1200_1	0.02	0.0636	0.1906	0.1162
SVM	1200	dat_1200_2	0.02	0.0601	0.1867	0.1134
SVM	1200	dat_1200_3	0.02	0.0608	0.1915	0.116
SVM	3000	dat_3000_1	0.05	0.1734	0.1681	0.1399
SVM	3000	dat_3000_2	0.05	0.1766	0.1705	0.1419
SVM	3000	dat_3000_3	0.05	0.1739	0.168	0.14

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
SVM	6000	dat_6000_1	0.1	0.3621	0.1506	0.1908
SVM	6000	dat_6000_2	0.1	0.255	0.1488	0.1632
SVM	6000	dat_6000_3	0.1	0.4146	0.1511	0.2042

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: SVM performs to maximize the distance of support vectors of each class. For non-linear seperable data, it uses kernel trick. Intuitively, kernel is to map the data to a higher dimension and trying to seperate them as much as possible, then it maps the hyperplane back to data dimension.

Parameters Choice: 1. `C-classification` also known as `C-SVM` means that the penalty on misclassification could be ranged from 0 to infinity. `nu-SVM` penalizes misclassification of support vectors and it's harder to optimise. Specifically, `C-SVM` is used for binary classification and here it applies `one-against-one` approach for multi-classification. Hence we choose `C-SVM`; 2. `cost` stands for the penalty. Increase `cost` value would dramatically increase training performance, however, over-fitting could appear. `cost=10` is a moderate punishment to avoid over-fitting 3. `kernel` here we choose `RBF`, it maps our data to infinite dimension for hyperplane to seperate. Choosing `RBF` is the most common choice but the training time might increase.

Advantages: 1. Regularization: SVM uses regularization to avoid over-fitting. It penalize the number of features' weights. Here in `LIBSVM` it uses `L2-Regularization` by default. 2. Adaptability: SVM is defined as a convex optimisation problem so it won't stuck in local minima. 3. Extensibility: Kernel trick can be applied to all kinds of problems.

Disadvantages: 1. Kernel function is sometimes hard to pick. 2. Training time is quite long especially for large data set. 3. Hard to interpret the trained variable weights and individual impact.

Model 7: Neural Networks

```
func_nn<-function(data.name){

  fitControl <- trainControl(method = "none")

  # Runtime Starts
  toc<-Sys.time()

  # Model
  model.nn<-train(label~.,data=get(data.name), method = 'nnet', trControl = fitControl, preF
  pred.nn<-predict(model.nn, newdata=dt_test[, -1])

  # Runtime Ends
  tic<-Sys.time()

  accu.nn<-unname(confusionMatrix(pred.nn, dt_test$label)$overall[1])

  return(func_grade("Neural Networks",data.name,tic-toc,accu.nn))
}
```

```
# Display results for the 9 samples
result_nn <- data.table()
for (dat in list_sample.names){result_nn <- rbind(result_nn, func_nn(dat))}
datatable(result_nn, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Neural Networks	1200	dat_1200_1	0.02	0.034	0.2464	0.1367
Neural Networks	1200	dat_1200_2	0.02	0.0733	0.2421	0.1444
Neural Networks	1200	dat_1200_3	0.02	0.0862	0.2409	0.147
Neural Networks	3000	dat_3000_1	0.05	0.1529	0.2116	0.1565
Neural Networks	3000	dat_3000_2	0.05	0.1475	0.2031	0.1509
Neural Networks	3000	dat_3000_3	0.05	0.1465	0.2034	0.1508
Neural Networks	6000	dat_6000_1	0.1	0.2832	0.1934	0.1925
Neural Networks	6000	dat_6000_2	0.1	0.1501	0.1945	0.1598

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Neural Networks	6000	dat_6000_3	0.1	0.165	0.1839	0.1582

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: Neural networks performs like a "Break Apart and Group Up". It takes pixel data from input and using different math operations to extract features. Noise signals could be filtered and key features are aggregated together. Hence it captures the most important information of signals and keeps high accuracy in preditions.

Parameters Choice: 1. `trControl=trainControl(method = "none")` : ignoring cross-validation. After testing with and without cross-validation, we see minor changes in accuracy but major changes in training time. Hence we skip cross-validation. 2. `size=10` the number of hidden layer units. We only use 1 hidden layer here. Generally more hidden layer means better feature extraction, but the cost is more time and worse feature aggregation (image features are more dispersed in hidden layers). Comparing to 49 pixels, `size=10` could be a moderate setting. 3. `decay=0.05` decay controls the weights of regularization, similar in SVM model. If decay is too high, our model focuses more on keeping regularization term low instead of tracking training performance. 0.05-0.1 should be a good range for `decay` .

Advantages: 1. Neural networks is a non-linear model, which is important for complex data, for example, the image pixel data. This feature is functional for solving real life problems. 2. It captures unseen data structures. Unlike many statistical models which uses mean, standard divation etc., to summarize data structure, its hidden layers extract features not by certain formulas but by a complex connection of raw input.

Disadvantages: 1. Black box: it's hard to tell what is going on between hidden layers. One way is to print out hidden layers' outputs as images, to see what features are collected, but for humans it's hard to recognize anything. 2. Data size. To train a decent neural networks requires a huge amount of data and the cost of computation is high. 3. Parameter choice: it's hard to explain why some hyperparameters perform well and somes are bad. This is a result-oriented parmater choosing.

Model 8: Ridge Regression

```
# Ridge, alpha=0

func_ridge<-function(data.name){

  dat<-get(data.name)
  x<-as.matrix(dat[,-1])
  y<-as.list(dat[,1])$label

  # Runtime Starts
  toc<-Sys.time()

  # Train for 100 lambda, %Dev > 0.5, good.
  model.ridge<-glmnet(x,y, alpha=0, nlambda=100, standardize=TRUE,family="multinomial")

  # Obtain best lambda
  s<-which.min(model.ridge$lambda)
  pred.ridge <- predict(model.ridge, as.matrix(dt_test[,-1]),type="class")
  accu.ridge <- func_accuracy(as.list(dt_test[,1])$label,pred.ridge[,s])

  # Runtime Ends
  tic<-Sys.time()

  return(func_grade("Ridge",data.name,tic-toc,accu.ridge))
}
```

```
# Display results for the 9 samples
result_ridge <- data.table()
for (dat in list_sample.names){result_ridge <- rbind(result_ridge, func_ridge(dat))}
datatable(result_ridge, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Ridge	1200	dat_1200_1	0.02	0.2207	0.2357	0.178
Ridge	1200	dat_1200_2	0.02	0.2105	0.2278	0.1715
Ridge	1200	dat_1200_3	0.02	0.2117	0.2199	0.1679
Ridge	3000	dat_3000_1	0.05	0.2925	0.2244	0.1978

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Ridge	3000	dat_3000_2	0.05	0.1979	0.2243	0.1741
Ridge	3000	dat_3000_3	0.05	0.275	0.228	0.1952
Ridge	6000	dat_6000_1	0.1	0.3596	0.2249	0.2274
Ridge	6000	dat_6000_2	0.1	0.2697	0.2227	0.2038
Ridge	6000	dat_6000_3	0.1	0.2374	0.2246	0.1966

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: Ridge regression is one type of linear regression with the coefficients estimated by minimizing $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2$, where $\lambda \geq 0$ is a tuning parameter.[]

Parameters Choice: 1. `alpha = 0` is for the elastic-net mixing parameter α, with range $\alpha \in [0, 1]$. $\alpha = 0$ is the ridge. 2. `nlambda = 100` represents the numbers of tuning parameters we trained. Default is 100. 3. `standardize = TRUE` is a logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize = TRUE. 4. `family = "multinomial"` was defined in this case since we have multi-classifications.

Advantages: Ridge regression has the effect of shrinking the estimates of all coefficients towards zero to decrease the likelihood of overfitting the training data. As λ increases, the variance of the predictions decrease and bias increase slightly.

Disadvantages: Ridge regression includes all predictors in the final model. This might be a problem of interpreting the model with all predictors.

Model 9: Lasso Regression

```
# Lasso, alpha=1
library(glmnet)

func_lasso<-function(data.name){

  dat<-get(data.name)
  x<-as.matrix(dat[,-1])
  y<-as.list(dat[,1])$label

  # Runtime Starts
  toc<-Sys.time()

  # Train for 100 lambda, %Dev > 0.5, good.
  model.lasso<-glmnet(x,y, alpha=1, nlambda=100, standardize=TRUE,family="multinomial")

  # Obtain best lambda
  s<-which.min(model.lasso$lambda)
  pred.lasso <- predict(model.lasso, as.matrix(dt_test[,-1]),type="class")
  accu.lasso <- func_accuracy(as.list(dt_test[,1])$label,pred.lasso[,s])

  # Runtime Ends
  tic<-Sys.time()

  return(func_grade("Lasso",data.name,tic-toc,accu.lasso))
}
```

```
# Display results for the 9 samples
result_lasso <- data.table()
for (dat in list_sample.names){result_lasso <- rbind(result_lasso, func_lasso(dat))}
datatable(result_lasso, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Lasso	1200	dat_1200_1	0.02	0.5509	0.2456	0.2655
Lasso	1200	dat_1200_2	0.02	0.5796	0.2429	0.2714
Lasso	1200	dat_1200_3	0.02	0.4823	0.2393	0.2452
Lasso	3000	dat_3000_1	0.05	0.9843	0.1995	0.3583

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Lasso	3000	dat_3000_2	0.05	0.7076	0.2026	0.2907
Lasso	3000	dat_3000_3	0.05	0.9162	0.2029	0.343
Lasso	6000	dat_6000_1	0.1	1	0.1897	0.3698
Lasso	6000	dat_6000_2	0.1	0.7875	0.1946	0.3192
Lasso	6000	dat_6000_3	0.1	0.6809	0.1876	0.289

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: In the regularized regression models, Lasso Regression is one of the ways to perform variable selection to choose a small number of variables for inclusion in the model. The lasso regression estimates the coefficients by minimizing $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|$.

Parameters Choice: 1. `alpha = 0` is for the elastic-net mixing parameter α, with range $\alpha \in [0, 1]$. $\alpha = 1$ is the lasso. 2. `nlambda = 100` represents the numbers of tuning parameters we trained. Default is 100. 3. `standardize = TRUE` is a logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize = TRUE. 4. `family = "multinomial"` was defined in this case since we have multi-classifications.

Advantages: The lasso performs variable selection by forcing some of the coefficient estimates to be zero when the tuning parameter λ is large. And the model from lasso is easier to interpret with fewer variables.

Disadvantages: One of the disadvantages of Lasso is that if there are some variables are highly correlated, the lasso tends to arbitrarily select one from them.

Model 10: Linear Discriminant Analysis

```
# Function for the model
func_lda <- function(data.name){
  require(MASS)
  toc <- Sys.time()
  model_fit <- lda(label~.,data = get(data.name))
  model_pred <- predict(model_fit,dt_test)
  tic <- Sys.time()
  accu.lda <- mean(model_pred$class == dt_test$label)
  return(func_grade("Linear Discriminant Analysis",data.name,tic-toc,accu.lda))
}
```

```
# Display results for the 9 samples
result_lda <- data.table()
for (dat in list_sample.names){result_lda <- rbind(result_lda, func_lda(dat))}
datatable(result_lda, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Linear Discriminant Analysis	1200	dat_1200_1	0.02	0.0026	0.2364	0.1238
Linear Discriminant Analysis	1200	dat_1200_2	0.02	0.0016	0.2351	0.123
Linear Discriminant Analysis	1200	dat_1200_3	0.02	0.001	0.2405	0.1255
Linear Discriminant Analysis	3000	dat_3000_1	0.05	0.0013	0.2368	0.1312
Linear Discriminant Analysis	3000	dat_3000_2	0.05	0.0014	0.2324	0.129
Linear Discriminant Analysis	3000	dat_3000_3	0.05	0.0013	0.2435	0.1346
Linear Discriminant Analysis	6000	dat_6000_1	0.1	0.0021	0.2304	0.1407

Model Name	Sample Size	Name	A-Propt	B-Runtime	C-Inaccu	Grade
Linear Discriminant Analysis	6000	dat_6000_2	0.1	0.0021	0.2371	0.1441
Linear Discriminant Analysis	6000	dat_6000_3	0.1	0.0021	0.2323	0.1417

Showing 1 to 9 of 9 entries Previous 1 Next

Model Explaination: The underlying assumption of linear discriminant analysis is that all predictors are from a multivariate normal distribution with a mean vector and a common covariance matrix. Given each class, model the distribution of the variables as a multivariate normal. Then use Bayes rule to estimate the likelihood of each class given the values of the predictors.

Advantages: When there are more than 2 responses or sample size is small, LDA is preferred and more stable than the logistic regression.

Disadvantages: LDA requires an assumption of predictors drawn from a multivariate normal distribution with a common covariance matrix. If the assumptions are not met, the model may perform worse than that from the logistic regression.

Scoreboard

```
# Combine all the results togerther
aggre_result <- rbind(result_mlr, result_ensemble, result_rf, result_gbm, result_rpart,
                      result_nn, result_ridge, result_lasso, result_svm, result_lda)

# Average out, round numerics, and set order by the grade
aggre.variables <- c("A-Propt", "B-Runtime", "C-Inaccu", "Grade")
aggre_result <- aggre_result[, lapply(.SD, 'mean'), .SDcols = aggre.variables, keyby = c("Mo
aggre_result <- aggre_result[, lapply(.SD, 'round.numerics')]
setorder(aggre_result, Grade)

# Display final scoreboard
datatable(aggre_result, rownames = FALSE)
```

Show 10 entries Search:

Model Name	Sample Size	A-Propt	B-Runtime	C-Inaccu	Grade
Random Forest	3000	0.05	0.0126	0.1906	0.1109
Random Forest	1200	0.02	0.0049	0.2148	0.1136
SVM	1200	0.02	0.0615	0.1896	0.1152
Random Forest	6000	0.1	0.0293	0.1716	0.1181
Linear Discriminant Analysis	1200	0.02	0.0017	0.2373	0.1241
Linear Discriminant Analysis	3000	0.05	0.0013	0.2376	0.1316
SVM	3000	0.05	0.1746	0.1689	0.1406
Linear Discriminant Analysis	6000	0.1	0.0021	0.2333	0.1422
Neural Networks	1200	0.02	0.0645	0.2431	0.1427
Gradient Boosting Machine	1200	0.02	0.1029	0.2315	0.1465

Showing 1 to 10 of 30 entries Previous 1 2 3 Next

Discussion

In the results of our models, the best machine learning model for this project is the SVM model with sample size 1200. (The runtime of each model may vary a bit each time we knit the report.) The model with the lowest inaccurate rate is the SVM with sample size 6000. Among all the models, the LDA models use the least amount of time. In the group of models with sample size 1200, the SVM wins the lowest inaccurate rate and points. In the group of models with sample size 3000, the SVM wins the highest accuracy and the RandomForest gets the lowest points. In the group of models with sample size 6000, the SVM wins the lowest inaccuracy and the RandomForest gets the lowest points. If the high accuracy is required, like in the healthcare industry, then the SVM model with a large sample size would be a powerful model of

classification. In some other industries, like social media requiring prediction in real time, RandomForest might be a better model by using a relatively little amount of time to train models.

References

1. https://stackoverflow.com/questions/15031338/subscript-out-of-bounds-general-definition-and-solution
2. https://stat.ethz.ch/R-manual/R-devel/library/rpart/html/rpart.control.html
3. https://github.com/zalandoresearch/fashion-mnist
4. http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/
5. https://stats.stackexchange.com/questions/312897/c-classification-svm-vs-nu-classification-svm-in-e1071-r
6. https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
7. http://www.simafore.com/blog/bid/62333/4-key-advantages-of-using-decision-trees-for-predictive-analytics
8. https://www.quora.com/What-are-the-advantages-disadvantages-of-using-Gradient-Boosting-over-Random-Forests
9. http://uc-r.github.io/gbm_regression#proscons