Hao-Chun, Niu Final Project for Practical Machine Learning Course

Brief Introduction

In order to accurately predict the classes based on WLE data set, I will come up with several classification methods. The entire process could be divided into three main steps. First, I will read the training data sets, divide the training data sets into one sub-training data set and one sub-testing data set, and clean out the redundant variables. Second, due to the fact that there are too many useful variables within the data set, I will conduct the principle component analysis to transform the data set. Lastly, I will create three different classification models, Random Forest, Boosting with GBM, and SVM, and assess the performance of each model.

1. Data Preparation and Tidy

The raw data have about 19622 observations and 160 variables. Thus, the data is quite complicated.

```
## [1] 19622   160
```

Yet, the data includes a lot of redundant variables. For instance, the variables that include “max_” in their column names calculate the maximum value of the previous variables. Thus, they are redundant. At the end, I discard all those variables that contain “min_”, “max_”, “var_”, “avg_”, “amplitude_”, “skewness_”, “kurtosis_”, “stddev_” in their column names and only 55 variables remain.

```
## Note: Using an external vector in selections is ambiguous.
## ℹ Use `all_of(non)` instead of `non` to silence this message.
## ℹ See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
## [1] 19622    55
```

As for our y, there are totally 5 levels (“A”, “B”, “C”, “D”, “E”) in the classe variable. The A classe has the largest proportion, about 30 percent. As for the rest of the classes, they each take about 18 percent.

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

```
##
##         A         B         C         D         E
## 0.2843747 0.1935073 0.1743961 0.1638977 0.1838243
```

Next, before I subset the training data set into sub-training and sub-testing, I check the correlation among the continuous variables to discard the ones that are highly correlated with others. Eventually, I find 7 variables that are corrrelated with others, and the correlation rates are larger than 0.9. Then, I remove those 7 columns and get my final data set.

```
correlation<-cor(df[,-c(1,55)])
high_cor<-findCorrelation(correlation,cutoff = 0.9)
sort(high_cor)
```

```
## [1]  2  9 10 11 19 32 34
```

Then, I remove those 7 columns and get my final data set with 19622 observations and 48 variables.

```
## [1] 19622    48
```

Lastly, I divide the data set into a sub-training data set and sub-testing test. The sub-training data set has 70 percent of the observations (13737), and the sub-testing data set has the rest of the 30 percent (5885).

```
inTrain<-createDataPartition(df$classe,p=0.7,list = F)
testing<-df[-inTrain,]
training<-df[inTrain,]
dim(training)
```

```
## [1] 13737    48
```

```
dim(testing)
```

```
## [1] 5885   48
```

2.Principal Component Analysis (PCA)

From the result of the first stage, I notice that there are still too many variables. Therefore, I decide to conduct PCA to reduce the complexity of the data set. First, I transform the training data set. According to the result, to capture more than 90 percent of the variance, I have to create 20 principle components.

```
preProc<-preProcess(training[,-48],method = "pca",thresh = 0.9)
preProc
```

```
## Created from 13737 samples and 47 variables
##
## Pre-processing:
##   - centered (46)
##   - ignored (1)
##   - principal component signal extraction (46)
##   - scaled (46)
##
## PCA needed 20 components to capture 90 percent of the variance
```

```
trained<-predict(preProc,training[,-48])
trained<-mutate(trained,"classe"=training$classe)
head(trained)
```

```
##   new_window      PC1       PC2      PC3       PC4       PC5          PC6
## 1         no 3.666849 -3.373472 1.486573 -1.150250 -1.918547  0.012105381
## 2         no 3.690407 -3.383096 1.481931 -1.158107 -1.931590  0.005070210
## 3         no 3.722657 -3.371196 1.476829 -1.159165 -1.945240  0.002499528
## 4         no 3.697508 -3.357677 1.478492 -1.144930 -1.924584  0.006700895
## 5         no 3.757981 -3.390052 1.449340 -1.227190 -2.012603 -0.015460736
## 6         no 3.665499 -3.355889 1.423984 -1.083112 -1.913640  0.022547330
##         PC7       PC8          PC9      PC10      PC11      PC12     PC13
## 1 -2.131973 -1.777732 -0.053884461 -1.265878 -1.529015 -1.973316 2.135474
## 2 -2.121348 -1.752886 -0.006077687 -1.261358 -1.449571 -1.971237 2.122668
## 3 -2.126766 -1.755321 -0.013940270 -1.239055 -1.471325 -1.947323 2.141069
## 4 -2.112589 -1.758953 -0.017742990 -1.243713 -1.480572 -1.957754 2.129368
## 5 -2.104572 -1.687942 -0.009699362 -1.242755 -1.383665 -1.976202 2.118974
## 6 -2.166082 -1.744908 -0.040141079 -1.257233 -1.591366 -1.949532 2.173216
##        PC14       PC15      PC16       PC17       PC18      PC19      PC20
## 1 -1.173162 -0.5861177 0.6406187 -0.4517338 -0.5432780 -1.145821 0.9180276
## 2 -1.186626 -0.6047573 0.6588707 -0.4810660 -0.5397072 -1.137729 0.9115459
## 3 -1.192071 -0.6189772 0.6587668 -0.4704078 -0.5261220 -1.133077 0.9092840
## 4 -1.183625 -0.6109110 0.6550056 -0.4655692 -0.5278887 -1.137128 0.8971099
## 5 -1.208912 -0.5945775 0.6643559 -0.5316878 -0.5320436 -1.126741 0.9191054
## 6 -1.152705 -0.6259224 0.6337127 -0.3971657 -0.5185909 -1.137327 0.9090854
##   classe
## 1      A
## 2      A
## 3      A
## 4      A
## 5      A
## 6      A
```

Next, I do the exactly same transformation to the testing data set.

```
head(tested)
```

```
##   new_window      PC1       PC2      PC3       PC4       PC5          PC6
## 1         no 3.705708 -3.399376 1.464137 -1.210510 -1.993221 -0.008009132
## 2         no 3.702217 -3.368580 1.467718 -1.158292 -1.960488  0.006272289
## 3         no 3.740054 -3.300900 1.463345 -1.164295 -2.001670 -0.005353764
## 4         no 3.674962 -3.359774 1.461541 -1.151366 -1.933733  0.013926057
## 5         no 3.709784 -3.363665 1.469768 -1.171304 -1.959358  0.009999067
## 6         no 3.652325 -3.394200 1.454604 -1.160489 -1.960096  0.012720947
##         PC7       PC8          PC9      PC10      PC11      PC12     PC13
## 1 -2.105158 -1.722030  0.004424205 -1.253714 -1.418895 -1.965907 2.112245
## 2 -2.102786 -1.729151  0.022068659 -1.258699 -1.451283 -1.924214 2.134411
## 3 -2.141806 -1.701153  0.018235003 -1.228018 -1.434184 -1.980182 2.128884
## 4 -2.144377 -1.745617 -0.002966516 -1.259749 -1.510831 -1.957871 2.149882
## 5 -2.146092 -1.749989 -0.013910351 -1.244533 -1.481886 -1.963118 2.135225
## 6 -2.136439 -1.736631 -0.031378764 -1.249907 -1.543077 -1.985350 2.134087
##        PC14       PC15      PC16       PC17       PC18      PC19      PC20
## 1 -1.222296 -0.6093307 0.6834205 -0.5120288 -0.5285952 -1.136678 0.9326119
## 2 -1.212412 -0.6351389 0.6828268 -0.4825919 -0.5077323 -1.132379 0.9167139
## 3 -1.154868 -0.5092902 0.6034002 -0.5356616 -0.5308244 -1.113993 0.9361998
## 4 -1.165945 -0.6184323 0.6385451 -0.4377927 -0.5278454 -1.138624 0.9047118
## 5 -1.180422 -0.6202130 0.6480624 -0.4612458 -0.5214264 -1.135169 0.9012760
## 6 -1.187828 -0.6117118 0.6359872 -0.4496551 -0.5143538 -1.136534 0.9091891
##   classe
## 1      A
## 2      A
## 3      A
## 4      A
## 5      A
## 6      A
```

3-1. Modeling: Random Forest

The first model I am going to create is the random forest model. I will use all 21 predictors to predict the outcome, classe. Besides, I use Bootstrap as my resampling method with 25 repeats. The performance is outstanding. To be more specific, the final model has more than 95 percent accuracy rate.

```
model_rf<-train(classe~.,data = trained,method="rf")
model_final_rf<-model_rf$finalModel
model_rf
```

```
## Random Forest
##
## 13737 samples
##    21 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9616289  0.9514649
##   11    0.9497167  0.9364042
##   21    0.9352229  0.9180694
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

The result below shows the detail of the final model. The final model creates 500 trees with 2 variables use at each node. In addition, the OOB estimate of error rate is less than 3 percent. In general, this model is pretty effective.

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.6%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3860   15   17   11    3  0.01177675
## B   39 2566   42    7    4  0.03461249
## C    6   38 2327   22    3  0.02879800
## D    6    7   95 2140    4  0.04973357
## E    2   10   10   16 2487  0.01504950
```

Lastly, I predict the testing data, using the random forest model. The result is showed below. Based on the confusion matrix, the accuracy rate is above 95 percent. Besides, almost all the sensitivity and specificity rates for all 5 level of classe reach 95 percent. Hence, the model has accurately predicted the testing data.

```
pred_rf<-predict(model_rf,tested)
freq_rf<-table(tested$classe,pred_rf)
confusionMatrix(freq_rf)
```

```
## Confusion Matrix and Statistics
##
##    pred_rf
##        A    B    C    D    E
##   A 1656    2    8    7    1
##   B   21 1101   12    2    3
##   C    5    7 1004    7    3
##   D    1    0   32  930    1
##   E    0    4   10    8 1060
##
## Overall Statistics
##
##                Accuracy : 0.9772
##                  95% CI : (0.9731, 0.9809)
##     No Information Rate : 0.286
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9712
##
##  Mcnemar's Test P-Value : 2.082e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9840   0.9883   0.9418   0.9748   0.9925
## Specificity            0.9957   0.9920   0.9954   0.9931   0.9954
## Pos Pred Value         0.9892   0.9666   0.9786   0.9647   0.9797
## Neg Pred Value         0.9936   0.9973   0.9872   0.9951   0.9983
## Prevalence             0.2860   0.1893   0.1811   0.1621   0.1815
## Detection Rate         0.2814   0.1871   0.1706   0.1580   0.1801
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9898   0.9902   0.9686   0.9840   0.9940
```

3-2. Modeling: Boosting (gbm)

Next, I am going to create a boosting model using the gbm method. I will also use all 21 predictors to predict the outcome, classe, and use Bootstrap as my resampling method. The performance is fine but way worse than the random forest model. The final model's accuracy rate only reaches about 80 percent.

```
model_gbm<-train(classe~.,data = trained,method="gbm",verbose=FALSE)
model_final_gbm<-model_gbm$finalModel
model_gbm
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    21 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.5628133  0.4402425
##   1                  100      0.6106580  0.5042632
##   1                  150      0.6359008  0.5372644
##   2                   50      0.6579597  0.5654038
##   2                  100      0.7198706  0.6448901
##   2                  150      0.7512469  0.6849105
##   3                   50      0.7109167  0.6334589
##   3                  100      0.7696329  0.7082356
##   3                  150      0.8024456  0.7499095
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

The result below shows the detail of the final model. The final values used for the model are n.trees = 150, interaction.depth=3.

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 21 predictors of which 20 had non-zero influence.
```

Next, I predict the testing data, using the boosting (gbm) model. The result is printed below. According to the confusion matrix, around 80 percent of the observations are predicted accurately. In addition, the average of model's sensitivity rates is about 80 percent, and the average of model's specificity rates is about 95 percent.

```
pred_gbm<-predict(model_gbm,tested)
freq_gbm<-table(tested$classe,pred_gbm)
confusionMatrix(freq_gbm)
```

```
## Confusion Matrix and Statistics
##
##    pred_gbm
##        A    B    C    D    E
##   A 1444   55   62   88   25
##   B  138  835   96   50   20
##   C   66   75  851   30    4
##   D   50   30  115  753   16
##   E   40   85   64   50  843
##
## Overall Statistics
##
##                Accuracy : 0.8031
##                  95% CI : (0.7927, 0.8132)
##     No Information Rate : 0.2953
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7507
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8308   0.7731   0.7163   0.7755   0.9284
## Specificity            0.9445   0.9367   0.9627   0.9571   0.9520
## Pos Pred Value         0.8626   0.7331   0.8294   0.7811   0.7791
## Neg Pred Value         0.9302   0.9484   0.9306   0.9557   0.9865
## Prevalence             0.2953   0.1835   0.2019   0.1650   0.1543
## Detection Rate         0.2454   0.1419   0.1446   0.1280   0.1432
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.8877   0.8549   0.8395   0.8663   0.9402
```

3-3. Modeling: SVM

The last but not least, I create a SVM model as well. To be more specific, I use all 21 predictors to predict the outcome, classe. Because I do not use Bootstrap to resample the data, I only create one model.

```
model_svm<-svm(classe~.,data = trained)
model_svm
```

```
##
## Call:
## svm(formula = classe ~ ., data = trained)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  6687
```

Next, I predict the testing data, using the SVM model, and print the result below. The performance is better than the boosting (gbm) model but slightly worse than the random forest model. The accuracy rate reaches more than 90 percent. Besides, both the average sensitivity rate and specificity rate reach more than 90 percent.

```
pred_svm<-predict(model_svm,tested,type="class")
freq_svm<-table(tested$classe,pred_svm)
confusionMatrix(freq_svm)
```

```
## Confusion Matrix and Statistics
##
##    pred_svm
##        A    B    C    D    E
##   A 1610   23   34    6    1
##   B   79  988   57    5   10
##   C    4   45  942   33    2
##   D    7    9  102  842    4
##   E    3   18   22   38 1001
##
## Overall Statistics
##
##                Accuracy : 0.9147
##                  95% CI : (0.9073, 0.9217)
##     No Information Rate : 0.2894
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8921
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9454   0.9123   0.8142   0.9113   0.9833
## Specificity            0.9847   0.9686   0.9822   0.9754   0.9834
## Pos Pred Value         0.9618   0.8674   0.9181   0.8734   0.9251
## Neg Pred Value         0.9779   0.9800   0.9558   0.9833   0.9965
## Prevalence             0.2894   0.1840   0.1966   0.1570   0.1730
## Detection Rate         0.2736   0.1679   0.1601   0.1431   0.1701
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9650   0.9404   0.8982   0.9433   0.9833
```

Summary:

In conclusion, among all three methods, the random forest model is the most powerful one. However, the model takes a long time to calculate. Hence, to be practical, the most useful model would be the SVM model, which takes a much less of computing time but only has a slightly smaller accuracy rate.