Analyzing PML Activity Data Set

YSB

September 15, 2017

Preliminary Analysis of the data

First, let's load packages and read the dataset from the files. We assume dataset files are located in the same directory as this markdown document.

```
#Load Required Packages
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.3
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.3
```

```
dtrain = read.csv("./pml-training.csv")
dtest = read.csv("./pml-testing.csv")

print("No of missing values:")
```

```
## [1] "No of missing values:"
```

```
missing_values_cnt = sum(is.na(dtrain))
print(missing_values_cnt)
```

```
## [1] 1287472
```

```
print("Total No of values:")
```

```
## [1] "Total No of values:"
```

```
total_values_cnt = dim(dtrain)[1]*dim(dtrain)[2]
print(total_values_cnt)
```

```
## [1] 3139520
```

```
print("Percentage of missing data:")
```

```
## [1] "Percentage of missing data:"
```

```
print(100*missing_values_cnt/total_values_cnt)
```

```
## [1] 41.00856
```

```
print("Percentage of observations with complete data")
```

```
## [1] "Percentage of observations with complete data"
```

```
print(100*sum(complete.cases(dtrain))/dim(dtrain)[1])
```

```
## [1] 2.069106
```

```
dtrain = dtrain[,-1]
dtest = dtest[,-1]
```

The first column “X” in the dataset is just an index and hence, we remove it from further analysis. Also, as seen above, the dataset has significant amount of missing values and hence, we will impute the missing values by using the mean value of the respective columns. But before that, we analyze which columns are important with the given data so that we impute only columns that we need. This would save us unnecessary computations.

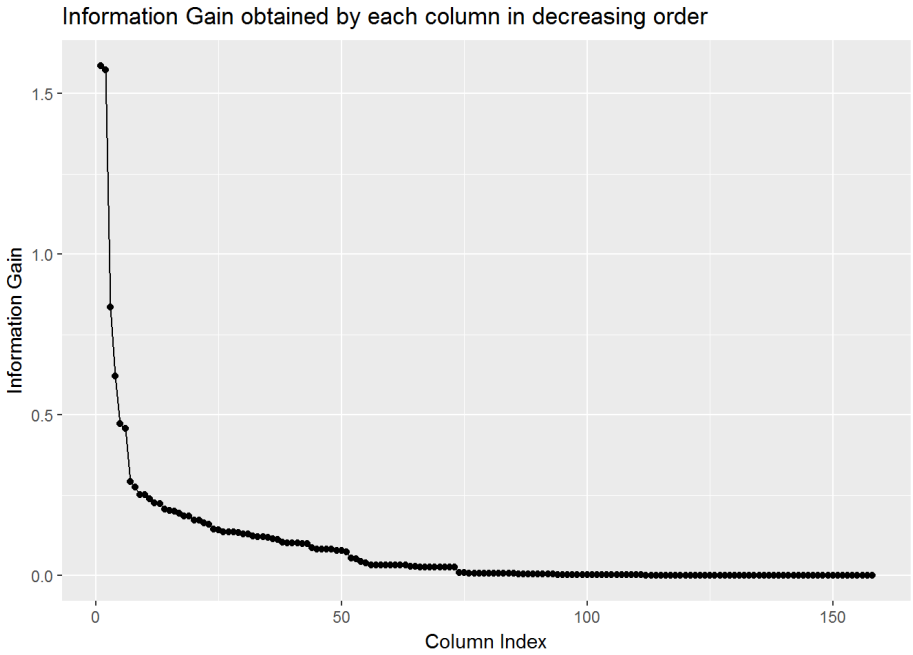
Analysis of importance of each column using Information Gain

We calculate the information gain for each column to find out important columns. We then plot the information gain given by each column in decreasing order of the gain.

```
library(FSelector)
```

```
## Warning: package 'FSelector' was built under R version 3.3.3
```

```
IG = information.gain(classe~.,data=dtrain)
qplot(x=1:dim(IG)[1],y=IG[cutoff.k(IG,k=dim(IG)[1]),])+geom_line()+labs(title="Information Gain obtained by each column in d
ecreasing order")+xlab("Column Index")+ylab("Information Gain")
```




```
igFeatures = cutoff.k(IG,k=20)
training = dtrain[c(igFeatures,"classe")]
print("No of missing values:")
```

```
## [1] "No of missing values:"
```

```
sum(is.na(training))
```

```
## [1] 0
```

From the above plot, we can see that 20 columns are good enough w.r.t. information gain. Further columns do not add much information. Hence, we select top 20 columns for further processing.

Also, As we can see, there are no missing values in the reduced dataset and hence, this can save us from effort of imputing columns. We proceed with dividing the dataset for further analysis.

Preprocess the dataset

We divide data into 25% training and 75% validation dataset. Note that, this is not a classical split of 80-20. In fact, we are giving only 25% data for training. However, this is again due to limited computation power of the machine being used. We would explain later that classifier used here is powerful enough to generalize even with less data. Further, since the dataset is large, we have good enough number of training samples. Out of 75% validation data, we split it further into 50-50 as validation and testing data. We keep testing data only for final evaluation. We are also scaling the dataset into 0-1 range for better analysis.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.3
```

```
## Loading required package: lattice
```

```
dim(dtrain)
```

```
## [1] 19622   159
```

```
smallTrInd = createDataPartition(dtrain$classe,p=0.25,list=F)

training2 = dtrain[c(igFeatures,"classe")]
training2 = training2[smallTrInd,]

remaining = dtrain[c(igFeatures,"classe")]
remaining = remaining[-smallTrInd,]

smallValInd = createDataPartition(remaining$classe,p=0.25,list=F)

validation = remaining[smallValInd,]
testing = remaining[-smallValInd,]

unseen_data = dtest[igFeatures]

pp=preProcess(training2,method="range")
tr_scaled = predict(pp,training2)
tr_scaled=tr_scaled[,-3]

val_scaled = predict(pp,validation)
val_scaled=val_scaled[,-3]

test_scaled = predict(pp,testing)
test_scaled = test_scaled[,-3]

unseen_scaled = predict(pp,unseen_data)
unseen_scaled = unseen_scaled[,-3]
```

Training and validating of the classifier

Here, we use Support Vector Machine (svm) classifier. This classifier is well known for its generalization ability. Hence, even with 25% data, we expect it to perform well. We also use RBF or Gaussian kernel. This setting requires us to choose 2 parameters: cost parameter of SVM and gamma parameter of kernel. SVM allows inbuilt cross-validation which we set to 3. Gamma parameter for a normalized data could lie within range of 1 to dimensionality (or no of columns) of the dataset. For cost, we try exponentially increasing values like: 0.01,0.03,0.1,0.3, etc. Further, we do another level of validation using our own validation dataset kept separately. We find the accuracy on the validation dataset for all the combinations of above two parameters. We note down the best model using this.

```
nsv=c()
acc=c()
val_acc=c()
model_no=c()
degree_vals=1:19
cost_vals=c(0.01,0.03,0.1,0.3,1,3,10,30)
#degree_vals=2
#cost_vals=c(0.01,0.03)

best_model_degree = 0
best_model_cost = 0
best_val_acc = -1
best_model = NULL
n=0

for(j in degree_vals){

for(i in cost_vals){ m1=svm(x=tr_scaled[,-20],y=tr_scaled$classe,cross=3,scale = F,cost=i,gamma=1/j)
  nsv=c(nsv,m1$tot.nSV)
  acc=c(acc,m1$tot.acc)
  n=n+1
  model_no=c(model_no,n)

  val_pred=predict(m1,val_scaled[,-20])
  m1_val_acc=sum(val_pred==val_scaled$classe)*100/length(val_pred)
  val_acc=c(val_acc,m1_val_acc)
  if(m1_val_acc>best_val_acc){
    best_model = m1
    best_model_degree = j
    best_model_cost = i
    best_val_acc = m1_val_acc

  }
}
}

print("Following is the best model using validation accuracy:")
```

```
## [1] "Following is the best model using validation accuracy:"
```

```
print("Cost")
```

```
## [1] "Cost"
```

```
print(best_model_cost)
```

```
## [1] 30
```

```
print("Degree")
```

```
## [1] "Degree"
```

```
print(best_model_degree)
```

```
## [1] 1
```

```
print("Model")
```

```
## [1] "Model"
```

```
print(best_model)
```

```
##
## Call:
## svm.default(x = tr_scaled[, -20], y = tr_scaled$classe, scale = F,
##     gamma = 1/j, cost = i, cross = 3)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  30
##       gamma:  1
##
## Number of Support Vectors:  1770
```

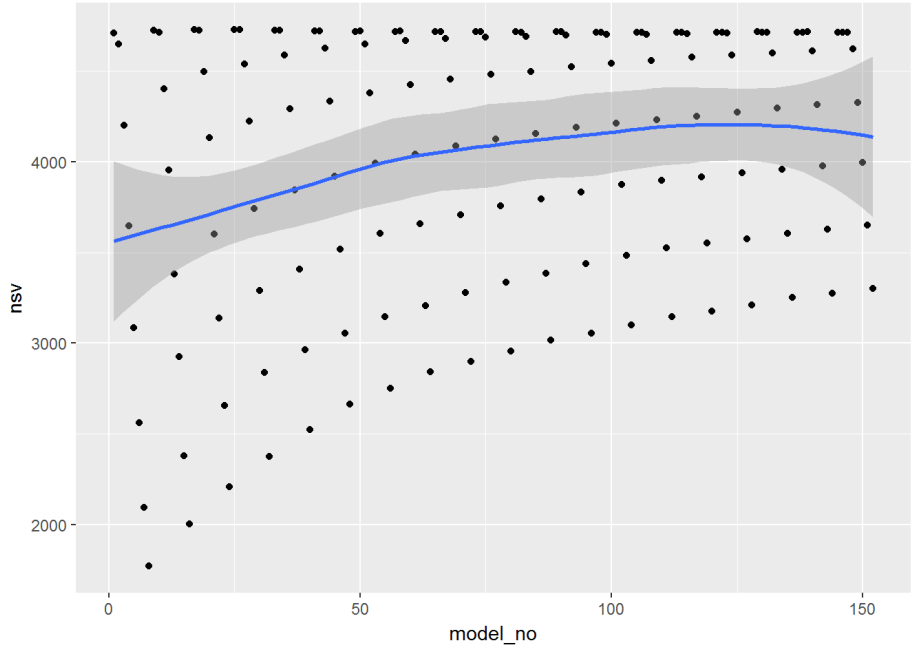
```
saveRDS(nsv,"nsv.rds")
saveRDS(acc,"acc.rds")
saveRDS(val_acc,"val_acc.rds")
saveRDS(best_model,"best_model.rds")
```

Effect of SVM parameters

Let's study the effect of various SVM parameters. First one is number of support vectors obtained. It is expected that, a good model has less number of support vectors. Following plot shows number of support vectors for each model. As cost and gamma increase, so does support vectors. The next plot shows the effect of model on the validation accuracy. As can be seen, there are some models with high accuracy. Final plot shows, as training accuracy increaes, so does the validation accuracy. Hence, we can claim, our model is not overfit to training data.

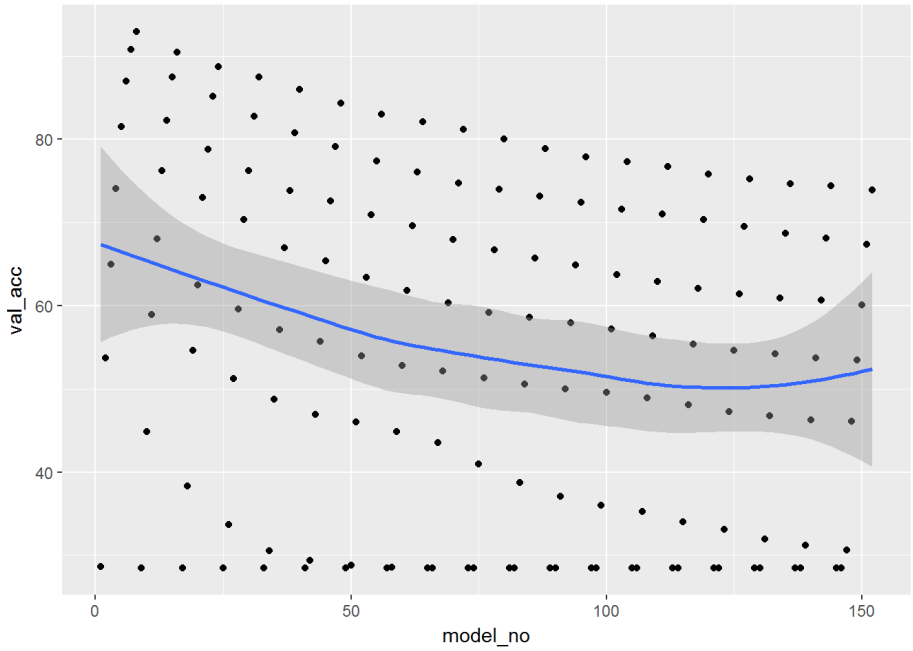
```
qplot(x=model_no,nsv)+geom_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```



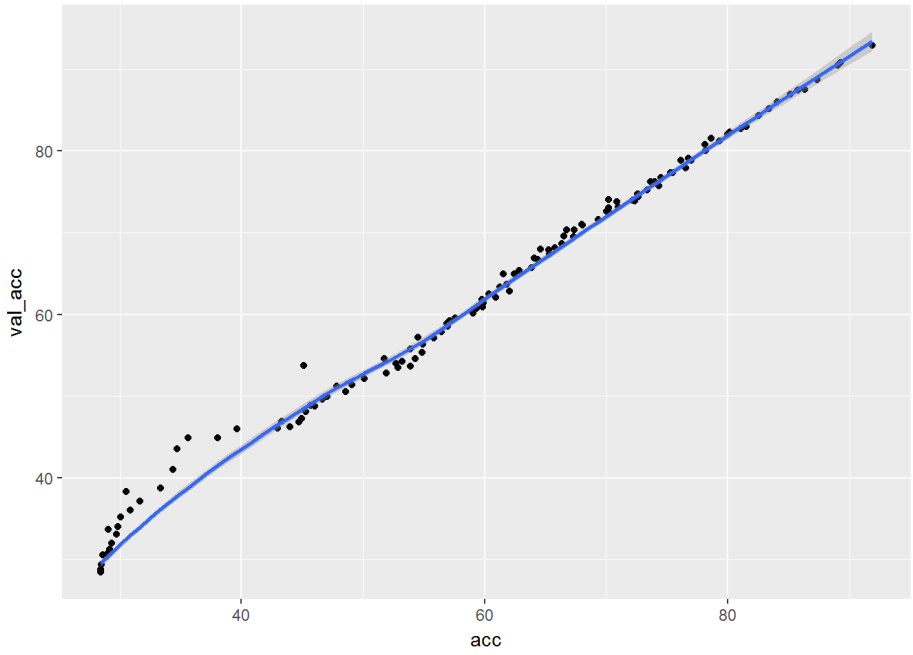

```
qplot(x=model_no,val_acc)+geom_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```




```
qplot(x=acc,y=val_acc)+geom_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```




```
print("Validation Accuracy of the best model:")
```

```
## [1] "Validation Accuracy of the best model:"
```

```
print(best_val_acc)
```

```
## [1] 92.9367
```

Final accuracy on the testing dataset

Let’s check the performance of model on the testing data that we kept aside.

```
test_pred=predict(best_model,test_scaled[,-20])
test_acc=sum(test_pred==test_scaled$classe)*100/length(test_pred)
print("Testing Accuracy of the best model:")
```

```
## [1] "Testing Accuracy of the best model:"
```

```
print(test_acc)
```

```
## [1] 93.83723
```

Predictions on unseen data

Finally we provide bellow the predictions on 20 unseen observations given:

```
r   unseen_pred=predict(best_model,unseen_scaled[,-20])   print(unseen_pred)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20    ##  B  A  C  A  A  E  D  B  A  A  B  C  B  A  E  E  A  B  B  B    ## Levels:
```

Conclusions

PML dataset has many missing values.But when we analyzed it using information gain, it was found that the columns that are important do not have any missing values. We have used SVM classifier to perform classification of the dataset. We found that SVM is powerful classifier and even with 25% training data it performed pretty well. We have achieved validation accuracy above 92% and test accuracy above 93%. For the unseen test data the labels are not available. However, we have provided predictions here.