STSCI4740 Final Project

Chengcheng Ji (cj368) Yanheng Li (yl2696)

Jiaqi Wang (jw2479) Zhiyuan Zhong (zz449)

Introduction

In this report, we are going to test different models to identify which variables can define the type of a Pokémon and predict whether a Pokémon is legendary or not. The first part of our report is going use tree-based model and multinomial logistic regression to perform type classification. For the tree-based model, we also apply tree pruning, bagging and random forest algorithm to improve accuracy and performed lasso algorithm in logistic regression to select important features. The test error rate and model simplicity are our criteria to select models and results indicate that random forest performed the best. Next, in the second part of our report, we use logistic regression, linear discriminant analysis, quadratic discriminant analysis, KNN model, SVM, Random Forest and AdaBoosting Models to predict whether a Pokémon is legendary or not. Since KNN model has the smallest test error rate, we decide to choose KNN model as our best model to predict legendary.

Data Description

The Pokémon dataset contains 800 observations with 13 variables: Name, Type.1, Type.2, Total, HP, Attack, Defense, Sp..Atk, Sp..Def, Speed, Generation, Legendary. Within 13 variables, there are 4 qualitative variables: Name, Type.1, Type.2 and Legendary.

Part 1. Which variables can define the type of a Pokémon

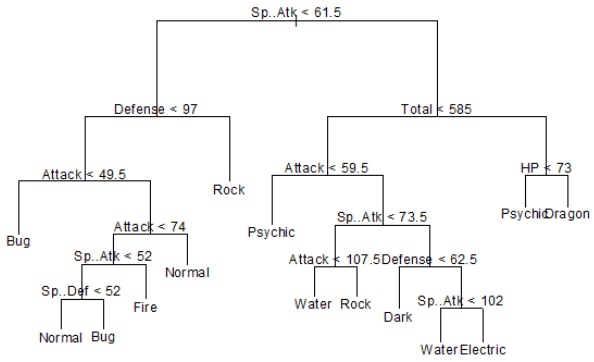
1. Data preparation

Considering that some Pokémon has two types, we treat Type.1 and Type.2 separately and try to identify features that are important to define each of them. We first perform data imputation and encoded the NA values in Type.2 which indicate these Pokémon only have 1 type into “NoType2”. We set 75% data as training data and use the remaining 25% as test data to check the performance of the models we used. Since some types are rare, when splitting the data, we need to pay much attention to include all types of both training and test data.

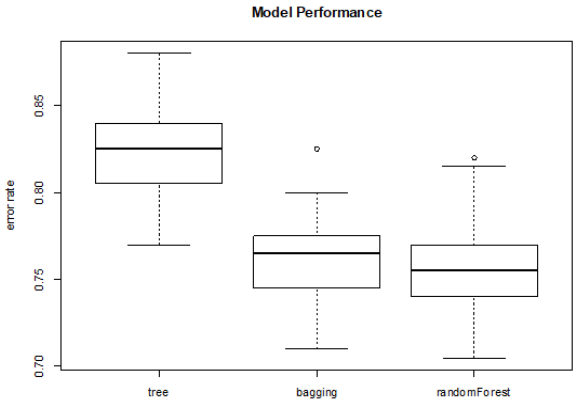
2. Decision Tree, Bagging, and Random Forests

Since tree model can be graphically displayed and it can be easily interpreted by people even without statistical background, we start with training tree models for Type.1 and Type.2 to visualize which features are useful.

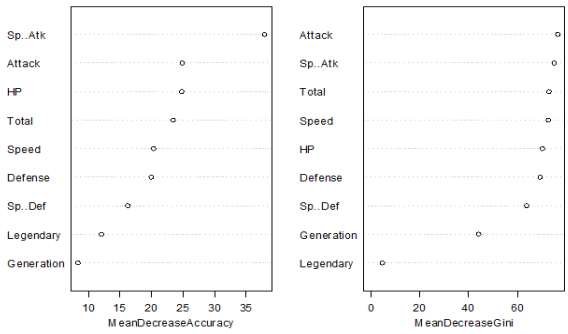
2.1 Type 1



The figure above is the simple classification tree of Type.1, and it suggests Sp..Atk, Sp..Def, Attack, Defense, Hp and Total are important variables. However, the performance is not desirable. There are 18 types for Type.1, but the decision tree can only classify 9 types, and the accuracy rate is around 20% which is just a little better than a random guess (1/18 = 5.6%). Therefore, we try to aggregate many decision trees and use bagging and random forest to improve model accuracy. The box plots of the error rate of the three models are shown below:

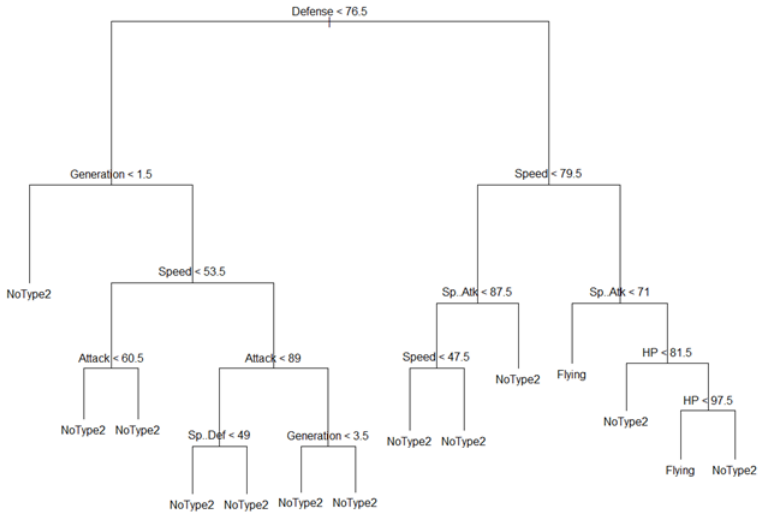


As we can see from the figure on the left, random forest model performs much better than decision trees and slightly better than bagging. Therefore, we plot the importance of each variable in the random forest model with smallest error rate.



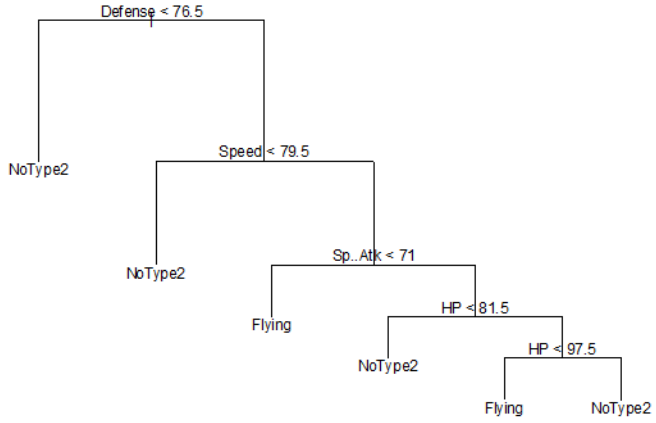
From the figure on the left, we could conclude that the top four most important variables to define Type.1 are Sp..Atk, Total, HP and Attack if we use MeanDecreaseAccuracy as the criteria.

2.2 Type 2

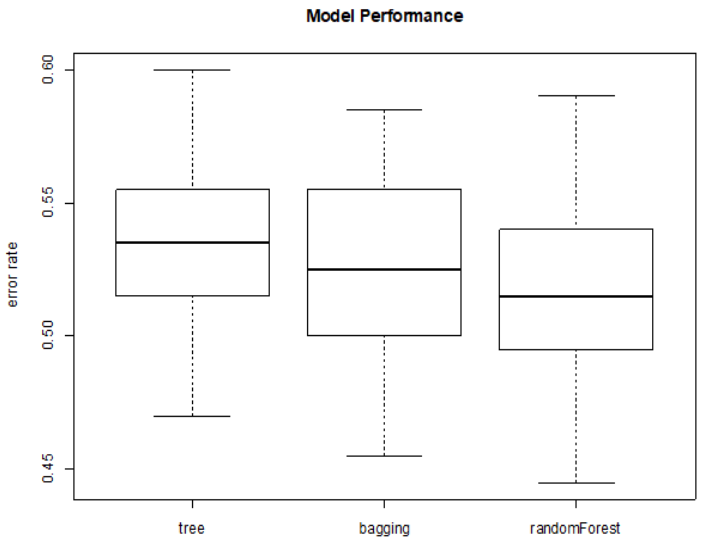


The figure above is the classification tree for Type.2. The test error rate is around 50% which is much better than the classification of Type.1, but this may result from the large number of NoType2 in Type.2. We find that there are too many terminal nodes with NoType2, so we try to prune the tree to get a smaller tree with fewer splits.

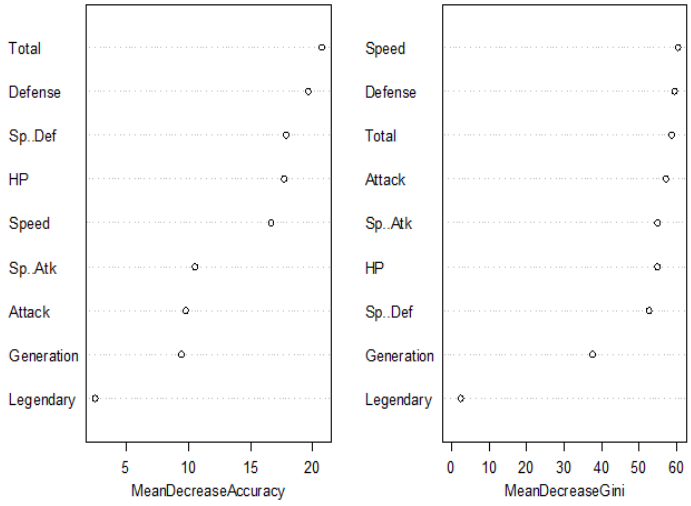
The test error is almost the same after pruning, but nodes decrease to 6 which make the model simpler. However, this model can only classify 2 types out of 19. Even though most Notype2 are classified correct which lead to a relatively lower test error rate, the model performance is still bad. Therefore, bagging and random forest models are also performed to improve the model performance. The box plots of test error rate for three models are shown below.



The test error is almost the same after pruning, but nodes decrease to 6 which make the model simpler. However, this model can only classify 2 types out of 19. Even though most Notype2 are classified correct which lead to a relatively lower test error rate, the model performance is still bad. Therefore, bagging and random forest models are also performed to improve the model performance. The box plots of test error rate for three models are shown below.



From the figure on the left, we could random forest still performs better than simple decision tree model and bagging. Then we plot the importance of each variable in random forest model with smallest error rate.



From the graph, we could conclude that the top five important variables for defining Type.2 are Total, Defense, Sp..Def, HP and Speed if we use MeanDecreaseAccurary as the criteria.

3. Shrinkage Methods

In this section, we use Lasso to train a multinomial logistic regression model to perform variable selection. Recall that lasso has the effect of shrinking some of the coefficients estimated to be exactly 0. Hence, similar to best subset selection, the lasso performs feature selection.

3.1 Type 1

For different types of Type 1, the lasso model gives different important features. Since there are 18 types in total, in order to better visualize the important variables for each type, we put them in the table as shown below

	Total	HP	Attack	Defense	Sp..Atk	Sp...Def	Speed	Generation	Legendary
Bug		*			*	*			
Dark			*				*	*	
Dragon	*		*					*	*
Electric		*	*		*		*		
Fairy							*		
Fighting			*	*	*				
Fire				*	*				
Flying									*
Ghost				*	*			*	
Grass					*		*	*	*
Ground			*		*	*			*
Ice									
Normal		*		*	*		*	*	*
Poison					*			*	
Psychic			*	*	*	*			*
Rock			*	*	*		*		
Steel				*					
Water		*	*			*		*	
Total	1	4	8	7	11	4	6	7	6

From the table above, it is clear that variables are different for defining each type in the lasso model. But based on the total times that each feature appears, we could have a general idea which features are important. In this case, Sp..Atk, Attack, Defense and Generation, speed and legendary are the top six important features for defining Type 1. This is a little different compared with the random forest model, and we will compare the test error rate in the later section.

3.2 Type 2

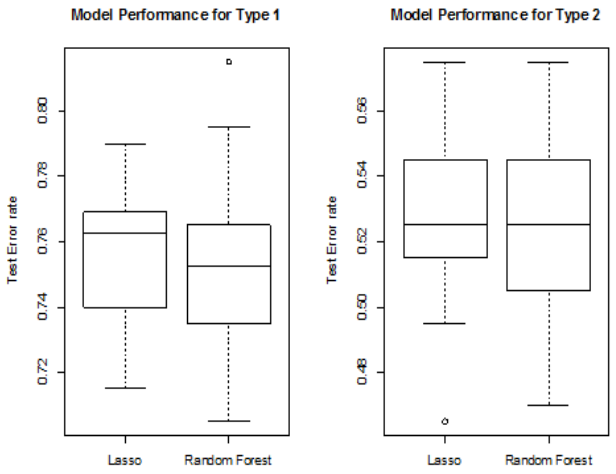
Similar to Type 1, we get the important features for each type in Type.2 and put them in a table as shown below

	Total	HP	Attack	Defense	Sp..Atk	Sp...Def	Speed	Generation	Legendary
Bug									
Dark			*						
Dragon	*				*				*
Electric									
Fairy			*						
Fighting			*				*		
Fire					*				
Flying				*	*		*	*	
Ghost									
Grass					*			*	
Ground		*			*	*	*	*	
Ice		*	*						*
Normal									
Notype2				*	*			*	
Poison				*	*			*	
Psychic				*	*	*		*	
Rock				*	*	*	*		
Steel				*					
Water									
Total	1	2	4	6	9	3	4	6	2

From the table above, we could conclude that the top 5 important features are Attack, Defense, Sp..Atk, Speed, Generation which are also different compared with the results from random forest model, and we will compare these two models in the later section.

4. Model Comparison and Justification

In this section, the test error rate of random forest model and lasso model for each Type will be used to assess which model is the best. The mean test error rate for each model is shown in the following graph:



We find that for type 1 random forest has better performance than lasso, while for type 2, random forest and lasso have similar performance. Therefore, for defining type 1, random forest is our choice. Sp..Atk, Total, HP and Attack are the four important features that could define Type 1. For Type 2, we still choose random forest as the model since it has relatively higher chance to get smaller test error. As a result, Total, Defense, Sp..Def, HP and Speed are five features that could define Type 2.

Part 2. Predict whether a Pokémon is legendary or not

1. Data Preparation

1.1 Problem description

Legendary is a binary variable. We are using the other 12 variables to predict whether a Pokémon is a legendary Pokémon.

1.2 Data preprocessing:

Some Pokémon are the evolution version of others, which is presented in the ID and Name. However, the unique ID is 721 compared to total 800 Pokémon. Though these id and names may contain some information, if we do the one-hot encoding, the resulting 721 features will be too hard to train limited to such few data. In conclusion, we decide to drop ID and Name.

In type2, there are much missing data which means this Pokémon is a single type Pokémon. We decide to fill these data with a new type “vacant”. Then for type1 and type2 we do one-hot encoding for them.

We keep the other numeric variables. Though Total is the sum of some other variables, it's a strong power conclusion of Pokémon. Also, some tree-based algorithm doesn't care collinearity.

For methods related with distance like SVM,KNN, we do data standardization.

1.3 Feature engineering

After encoding, we have 45 features. We calculate the absolute correlation of each variable with Legendary then sort them in order. The result is as following:

	Variable	Absolute_Correlation
38	Total	0.501758383
42	Sp..Atk	0.448907256
43	Sp..Def	0.363937117
40	Attack	0.345407976
44	Speed	0.326715295
39	HP	0.273619558
41	Defense	0.246376800
3	Type.1_Dragon	0.219463294
15	Type.2_Psychic	0.166624728
8	Type.1_Flying	0.108647046

Numeric variables have strong correlation with Legendary as these numbers decide general power of one Pokémon. It's interesting that the types dragon, psychic, and flying have also strong correlation, which is the same as what we feel when playing Pokémon games. Finally, we decide the 10 features which have the strongest correlation with Legendary as our final features for model training.

Actually, variables like attack can be divided into bins to better capture the distribution. For simplicity, since it doesn't necessary improve the performance, we don't do that here.

2. Model Training

In the 800 examples, there're 65 legendary and 735 non-legendary examples. It's actually an unbalanced dataset. Here we use simple prediction error rate to measure our final performance, though F1 and AUC may be more appropriate for this problem.

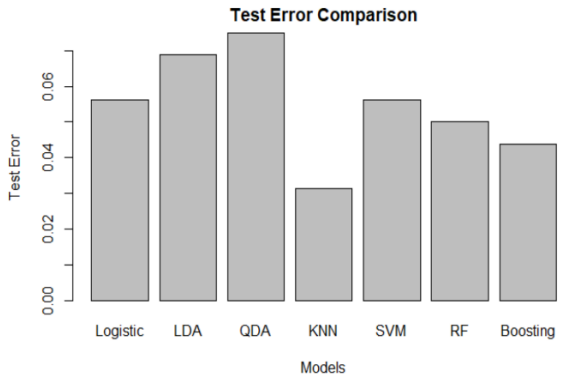
We split the data into 60/20/20 for train/validation/test set. For validation, we use 5-fold cross validation.

We start with looking at some classification models. We use different models to figure out the test error rate and compare which model perform the best. Under the classification models, we use logistic regression, Linear Discriminant Analysis(LDA), Quadratic Discriminant Analysis(QDA) and K-Nearest Neighbors(KNN) to predict Legendary. From the logistic regression model, we use the full model and get the test error rate is 5.625%. In LDA and QDA models, collinearity exist when we use all predictors in the model. We delete the qualitative

variables Type.1, Type.2 and Generation in the model. The test error rate for LDA is 6.875% and the test error rate for QDA is 7.500%. Lastly, by looking at the KNN model, we first select the K with the smallest test error. The best K we select is when K=1. The corresponding test error rate for K=1 is 3.125%. Then we look at the Support Vector Machine (SVM), we use radial kernel and tune the parameter cost and gamma and get the test error rate is 5.625%. By using the Random Forest, we tune the parameter ntree and get the test error rate is 5.000%. Finally, we look at the AdaBoosting model, we tune the parameter mfinal and get the test error rate is 4.375%.

3. Model Comparison and Justification

We create a bar-plot as below to visualize the test-error rate comparison for each model.



It is clearly to see from the plot above that the KNN model has the smallest test error rate and we decide to use KNN model as our best model to predict Legendary.

Conclusion

The results from our report suggest that 1) Random Forest has better performance for classifying both Type 1 and Type 2. Sp...Atk, Total, HP and Attack are the most important variables that could define Type 1 and Total, Defense, Sp..Def, HP and Speed are the variables that could be used to classify Type 2. 2) KNN with K=1 has the smallest test error rate of 3.125%, which is the best model to predict Legendary. For further research, we would recommend to try different method, such as F1 and AUC instead of test-error rate to justify model.

Acknowledgement

We would like to show our warm thank to Dr. Ning and Mr. Cao's assistance throughout the semester and kind suggestions for our report.

Reference

[1] G. James, D. Witten, T.Hastie, R. Tibshirani: An Introduction to Statistical Learning, ISSN 1431-875X

Appendix: R-Code

Part 1 Defining Type
0.data preprocessing
```{r}
pokemon <- read.csv("Pokemon.csv", header = TRUE,stringsAsFactors = TRUE, na.strings = "")
attach(pokemon)
Type.2 <- as.character(Type.2)
Type.2[is.na(Type.2)] <- "NoType2"
Type.2 <- as.factor(Type.2)
##Convert NA as a new lelve of Type.2 and encoded it as NoType2
pokemon$Type.2 <- Type.2
set.seed(1)
train = sample(1:nrow(pokemon),0.75*nrow(pokemon))
train.data = pokemon[train,]
test.data = pokemon[-train,]

#Make sure all the Types are included in both training and testing subset
summary(train.data$Type.1)
summary(test.data$Type.1)
summary(train.data$Type.2)
summary(test.data$Type.2)
```

1.Tree-based model
1.1 simple decision tree for type 1 and type 2
```{r}
#Type 1
library(tree)
tree.pokemon1 = tree(Type.1 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, train.data)
tree.pred1 = predict(tree.pokemon1, test.data, type = "class")
plot(tree.pokemon1, main = "Type.1")
text(tree.pokemon1)
error.rate <- mean(tree.pred1 != test.data$Type.1)
error.rate

#Type 2
tree.pokemon2 = tree(Type.2 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, train.data)
tree.pred2 = predict(tree.pokemon1, test.data, type = "class")
plot(tree.pokemon1, main = "Type.2")
text(tree.pokemon1)
error.rate2 <- mean(tree.pred1 != test.data$Type.2)
error.rate2
```

1.2 Tree pruning for type 2
```{r}
set.seed(88)
cv.pokemon = cv.tree(tree.pokemon2, FUN = prune.misclass)
cv.pokemon

par(mfrow=c(1,2))
plot(cv.pokemon$size, cv.pokemon$dev, type = 'b')

#The best tree size from the results is 1, but we cannot get a tree with size 1
#Thus we choose the second least dev with size 6
prune.pokemon = prune.misclass(tree.pokemon2, best=6)
par(mfrow=c(1,1))
plot(prune.pokemon)
text(prune.pokemon, pretty =0)

prune=predict(prune.pokemon,test.data,type="class")
table(prune,test.data$Type.2)
mean(prune != test.data$Type.2)
```

1.3 Bagging and random forest performance comparison
```{r}
##type 1
tree.err <- array()
bagging.err <- array()
rf.err <- array()
for (i in 1:50){
  set.seed(i)
  train = sample(1:nrow(pokemon),0.75*nrow(pokemon))
  train.data = pokemon[train,]
  test.data = pokemon[-train,]

  ##Tree
  tree.pokemon1 = tree(Type.1 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, train.data)
  tree.pred1 = predict(tree.pokemon1, test.data, type = "class")
  tree.err[i] <- mean(tree.pred1 != test.data$Type.1)

  ##bagging
  library(randomForest)
  bagging.type.1 <- randomForest(formula = Type.1 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, data=train.data,
                                 mtry=9,
                                 importance = TRUE,
                                 ntree = 500
```

```
  )
  bagging.pred <- predict(bagging.type.1, newdata = test.data)
  bagging.err[i] <- mean(bagging.pred != test.data$Type.1)

  ##randomForest
  rf.type.1 <- randomForest(formula = Type.1 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, data=train.data,
                            mtry=3,
                            importance = TRUE,
                            ntree = 500
  )
  rf.pred <- predict(rf.type.1, newdata = test.data)
  rf.err[i] <- mean(rf.pred != test.data$Type.1)
}

boxplot(tree.err, bagging.err, rf.err, names = c("tree", "bagging", "randomForest"), main =
"Model Performance", ylab = "error rate")

#type 2
tree.err2 <- array()
bagging.err2 <- array()
rf.err2 <- array()

for (i in 1:50){
  set.seed(i)
  train = sample(1:nrow(pokemon),0.75*nrow(pokemon))
  train.data = pokemon[train,]
  test.data = pokemon[-train,]

  ##Tree
  tree.pokemon2 = tree(Type.2 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, train.data)
  tree.pred2 = predict(tree.pokemon2, test.data, type = "class")
  tree.err2[i] <- mean(tree.pred2 != test.data$Type.2)

  ##bagging
  library(randomForest)
  bagging.type.2 <- randomForest(formula = Type.2 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, data=train.data,
                               mtry=9,
                               importance = TRUE,
                               ntree = 500
  )
  bagging.pred2 <- predict(bagging.type.2, newdata = test.data)
  bagging.err2[i] <- mean(bagging.pred2 != test.data$Type.2)
```

```
  ##randomForest
  rf.type.2 <- randomForest(formula = Type.2 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, data=train.data,
                        mtry=3,
                        importance = TRUE,
                        ntree = 500
  )
  rf.pred2 <- predict(rf.type.2, newdata = test.data)
  rf.err2[i] <- mean(rf.pred2 != test.data$Type.2)
}

boxplot(tree.err2, bagging.err2, rf.err2, names = c("tree", "bagging", "randomForest"), main =
"Model Performance", ylab = "error rate")
```

1.4 importance plot of random forest
```{r}
#Type 1
#random forest is the most accurate one
#using the rf model with smallest error rate to get the importance of each feature
which.min(rf.err)
set.seed(7)
train = sample(1:nrow(pokemon),0.75*nrow(pokemon))
train.data = pokemon[train,]
test.data = pokemon[-train,]
rf.type.1 <- randomForest(formula = Type.1 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, data=train.data,
                        mtry=3,
                        importance = TRUE,
                        ntree = 500
)
rf.pred <- predict(rf.type.1, newdata = test.data)
mean(rf.pred != test.data$Type.1)
varImpPlot(rf.type.1)

#Type 2
#random forest is the most accurate one for Type 2 as well
#using the rf model with smallest error rate to get the importance of each feature
which.min(rf.err2)
set.seed(6)
train = sample(1:nrow(pokemon),0.75*nrow(pokemon))
train.data = pokemon[train,]
test.data = pokemon[-train,]

rf.type.2 <- randomForest(formula = Type.2 ~
Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legendary, data=train.data,
```

```
                    mtry=3,
                    importance = TRUE,
                    ntree = 500
)
rf.pred2 <- predict(rf.type.2, newdata = test.data)
mean(rf.pred2 != test.data$Type.2)
varImpPlot(rf.type.2)
```

1.5 lasso multinomial logistic regression
```{r}
#Type 1
grid=10^seq(10,-2,length=100)
x.train=model.matrix(~Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legen
dary-1,train.data)
y.train1=train.data$Type.1
x.test=model.matrix(~Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legend
ary-1,test.data)
library(glmnet)
cv.lasso = cv.glmnet(x.train,y.train1,alpha=1,family = c("multinomial"),lambda=grid, nfolds
=10)
cv.lasso
best.lasso = cv.lasso$lambda.min
best.lasso
glm.fit = glmnet(x.train,y.train1,family = c("multinomial"), alpha=1,lambda=best.lasso)
coef(glm.fit)
ls.fit = predict(glm.fit, s = best.lasso, newx = x.test, type='class' )
ls.fit
#Type 2
y.train2=train.data$Type.2
cv.lasso2 = cv.glmnet(x.train,y.train2,alpha=1,family = c("multinomial"),lambda=grid, nfolds
=10)
cv.lasso2
best.lasso2 = cv.lasso2$lambda.min
best.lasso2
glm.fit2 = glmnet(x.train,y.train,family = c("multinomial"), alpha=1,lambda=best.lasso2)
coef(glm.fit)
ls.fit = predict(glm.fit2, s = best.lasso2, newx = x.test, type='class' )
ls.fit
```

1.6 lasso and random forest performance comparison
```{r}
lasso.err1 <- array()
lasso.err2 <- array()
```

```
for (i in 1:50){
  set.seed(i)
  train = sample(1:nrow(pokemon),0.75*nrow(pokemon))
  train.data = pokemon[train,]
  test.data = pokemon[-train,]

x.train=model.matrix(~Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legen
dary-1,train.data)
  y.train1=train.data$Type.1
  y.train2=train.data$Type.2

x.test=model.matrix(~Total+HP+Attack+Defense+Sp..Atk+Sp..Def+Speed+Generation+Legend
ary-1,test.data)

  cv.lasso1 = cv.glmnet(x.train,y.train1,alpha=1,family = c("multinomial"),lambda=grid, nfolds
=5)
  cv.lasso1
  best.lasso1 = cv.lasso1$lambda.min
  glm.fit1 = glmnet(x.train,y.train1,family = c("multinomial"), alpha=1,lambda=best.lasso1)
  ls.fit1 = predict(glm.fit1, s = best.lasso1, newx = x.test, type='class' )
  lasso.err1[i] <- mean(ls.fit1 != test.data$Type.1)

  cv.lasso2 = cv.glmnet(x.train,y.train2,alpha=1,family = c("multinomial"),lambda=grid, nfolds
=5)
  cv.lasso2
  best.lasso2 = cv.lasso2$lambda.min
  glm.fit2 = glmnet(x.train,y.train2,family = c("multinomial"), alpha=1,lambda=best.lasso2)
  ls.fit2 = predict(glm.fit2, s = best.lasso2, newx = x.test, type='class' )
  lasso.err2[i] <- mean(ls.fit2 != test.data$Type.2)
}

par(mfrow=c(1,2))
boxplot(lasso.err1,rf.err, main = "Model Performance for Type 1", ylab = "Test Error rate",
names = c("Lasso","Random Forest"))
boxplot(lasso.err2,rf.err2, main = "Model Performance for Type 2", ylab = "Test Error rate",
names = c("Lasso","Random Forest"))
```
part 2 legendary prediction
0. package preparation

```{r,warning=FALSE,message = F}
packages.used=c("dummies","e1071","ROCR","randomForest","adabag","DMwR","knitr","split
stackshape",'caret')
# check the packages needed
packages.needed=setdiff(packages.used,
                        intersect(installed.packages()[,1],
```

```
                              packages.used))
# install
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}

library(dummies)
library(ROCR)
library(e1071)
library(randomForest)
library(adabag)
library(DMwR)
library(knitr)
library(splitstackshape)
library(caret)

set.seed(1)

data<-data.frame(read.csv("Pokemon.csv",header=T,as.is=T))
# use as.is=T to read the data to avoid forcing character type data to become factor type
head(data)
str(data)
```
1. data preprocessing

```{r}
data1<-data

## for column one
## change the colname to be 'id'
colnames(data1)[1]<-'id'

## delete column one and two
data1 <- data1[,seq(3,ncol(data1))]

## for column four, fill the missing value with vacant
data1['Type.2'][data1['Type.2'] == ''] <- 'vacant'

## for column three & four, check their distinct level which is 18,19, so we make them one-hot encoding
length(unique(data1$Type.1))
length(unique(data1$Type.2))
length(unique(data1$Generation))
data1 <- dummy.data.frame(data1, names = 'Type.1',sep="_")
data1 <- dummy.data.frame(data1, names = 'Type.2',sep="_")
```

```
## for last column, make it to be two level factor
data1$Legendary<- ifelse(data1$Legendary == 'True',1,0)


## preview the data preprocessed
head(data1)
str(data1)
```
2. feature engineering

```{r,warning = F}
## make sure the data preprocessed has no NA
sum(is.na(data1))
```

```{r}
## check the correlation between the dependent variable and independent variables.
legendary<- ifelse(data$Legendary == 'True',1,0)
corr<-NULL
for(i in 1:(ncol(data1)-1))
{
  corr[i]<-cor(data1[,i],legendary)
}
mat<-as.data.frame(cbind(colnames(data1)[1:(ncol(data1)-1)],corr))
mat$corr<-as.numeric(as.character(mat$corr))
colnames(mat)<-c('variable',"absolute_correlation")
selected = as.numeric(rownames(mat[order(abs(mat[,2]),decreasing=T),][0][1:20,]))
data1 = data1[,c(selected,46)]
```


```{r}
## standardarize

data1_sub<-apply(data1[, 1:(ncol(data1)-1) ],2,scale)
data2<-as.data.frame(cbind(data1_sub,data1$Legendary))
colnames(data2)[21]="Y"
head(data2)
str(data2)
```


```{r}
## split the data into 60/20/20 train/validation/test
dim(data2)
```

```
test_index<-sample(1:800,800 * 0.2)
test<-data2[test_index,]
train<-data2[-test_index,]
folds <- createFolds(c(1:640), k = 5, list = FALSE, returnTrain = FALSE)
dim(test)
dim(train)
```

3. modeling
```{r}
table(data2$Y)
# asymmetric class sizes
```
3.1 SVM

choose 5-fold cross validation
since it's a classification problem, let type = "C-classification"
due to asymmetric class sizes, we set class.weights according to the ratio the legendary being different levels

```{r}

for (gamma in c(0.01,0.1,0.15)){
  error = 0
  for (i in c(1:5)){
  model<-svm(Y~.,data=train[folds!=i,],type = "C-classification",
         cost=10,kernel="radial",gamma=gamma,scale = F)
  pred<-predict(model,train[folds==i,])
  error = error + sum(pred!=train$Y[folds==i])/128.
  }

  print(gamma)
  print(error/5)
}
model<-svm(Y~.,data=train,type = "C-classification",
         cost=10,scale = F)
pred<-predict(model,test)
print(paste0("error is ", sum(pred!=test$Y)/160.))
```
```{r}
library(class)
Error = c()
for (k in c(1:100)){
  error = 0
  for (i in c(1:5)){
  model<-knn(train[folds!=i,],train[folds==i,],
```

```
            train$Y[folds!=i],k=k)
  error = error + mean(model!=train$Y[folds==i])
  }

  Error = cbind(Error,error/5)
}
k=which(Error==min(Error))
print(k)
model <-knn(train,test,train$Y,k=1)
print(paste0("error is ", mean(model!=test$Y)))
```

3.2 RF

```{r}
set.seed(1)
for (ntree in c(50,100,200,300)){
  error = 0
  for (i in c(1:5)){
  model<-randomForest(as.factor(Y)~.,data=train[folds!=i,],ntree=100)
  pred<-predict(model,train[folds==i,])
  error = error + sum(pred!=train$Y[folds==i])/128.
  }

  print(ntree)
  print(error/5)
}
model<-randomForest(as.factor(Y)~.,data=train,ntree=ntree)
pred<-predict(model,test)
print(paste0("error is ", sum(pred!=test$Y)/160.))
```
3.3 boosting

```{r}
set.seed(1)
train$Y = as.factor(train$Y)
for (mfinal in c(100,200,300)){
  error = 0
  for (i in c(1:5)){
  model<-boosting(Y~.,data=train[folds!=i,],mfinal=mfinal)
  pred<-predict(model,train[folds==i,])
  error = error + sum(pred!=train$Y[folds==i])/128.
  }

  print(mfinal)
  print(error/5)
```

```
}
model<-boosting(Y~.,data=train,mfinal=100)
pred<-predict(model,test)
print(paste0("error is ", sum(pred$class!=test$Y)/160.))
```
###LDA

```{r}
library(MASS)
{
  error = 0
  for (i in c(1:5)){
fit.lda=lda(Y~HP+Attack+Defense+Sp..Atk+Sp..Def+Speed,data=train[folds!=i,])
  pred.lda=predict(fit.lda,train[folds==i,])
  error = error + sum(pred.lda$class!=train$Y[folds==i])/128.
  }

  print(error/5)
}
fit.lda=lda(Y~HP+Attack+Defense+Sp..Atk+Sp..Def+Speed,data=train)
pred.lda=predict(fit.lda,test)
testerror_LDA=sum(pred.lda$class!=test$Y)/160.
print(paste0("error is ", sum(pred.lda$class!=test$Y)/160.))
```

###QDA
```{r}
library(corpcor)
cor2pcor(cov(data2))
```

```{r}
library(MASS)
{
  error = 0
  for (i in c(1:5)){
  fit.qda=qda(Y~HP+Attack+Defense+Sp..Atk+Sp..Def+Speed,data=train[folds!=i,])
  pred.qda=predict(fit.qda,train[folds==i,])
  error = error + sum(pred.qda$class!=train$Y[folds==i])/128.
  }

  print(error/5)
}
fit.qda=qda(Y~HP+Attack+Defense+Sp..Atk+Sp..Def+Speed,data=train)
pred.qda=predict(fit.qda,test)
testerror_QDA=sum(pred.qda$class!=test$Y)/160.
```

```
print(paste0("error is ", sum(pred.qda$class!=test$Y)/160.))
```

###Logistic
```{r}
{ error = 0
  for (i in c(1:5)){
  fit.logit=glm(Y~.,data=train[folds!=i,],family="binomial")
  glm.probs=predict(fit.logit,train[folds==i,],type="response")
  glm.pred=rep(0,length(glm.probs))
  glm.pred[glm.probs>0.5]=1
  error = error + sum(glm.pred!=train$Y[folds==i])/128.
  }

  print(error/5)
}

fit.logit=glm(Y~.,data=train,family="binomial")
summary(fit.logit)
glm.probs=predict(fit.logit,test,type="response")
glm.pred=rep("0",length(glm.probs))
glm.pred[glm.probs>0.5]="1"

testerror_logit=sum(glm.pred!=test$Y)/160.
print(paste0("error is ", sum(glm.pred!=test$Y)/160.))
```
###KNN
```{r}
library(class)
Error = c()
for (k in c(1:100)){
  error = 0
  for (i in c(1:5)){
  model<-knn(train[folds!=i,],train[folds==i,],
             train$Y[folds!=i],k=k)
  error = error + mean(model!=train$Y[folds==i])
  }

  Error = cbind(Error,error/5)
}
k=which(Error==min(Error))
print(k)
model <-knn(train,test,train$Y,k=1)
testerror_KNN=mean(model!=test$Y)
print(paste0("error is ", mean(model!=test$Y)))
```

###Plot for Comparison
```{r}
library(graphics)
Test_Error=c(testerror_logit,testerror_LDA,testerror_QDA,testerror_KNN,0.05625,0.05,0.04375
)
Test_Error
barplot(Test_Error,main="Test Error Comparison",xlab="Models",ylab="Test Error",
        names.arg=c("Logistic","LDA","QDA","KNN","SVM","RF","Boosting"))
```