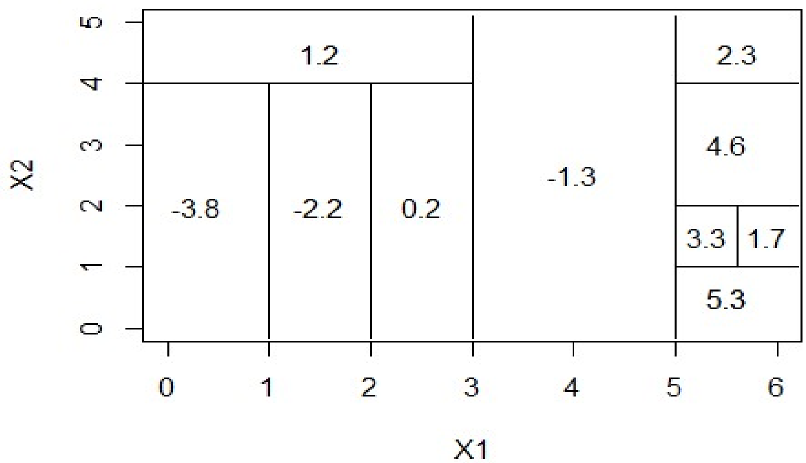
HW6 MSA 8150

Anutida Sangkla

3/31/2021

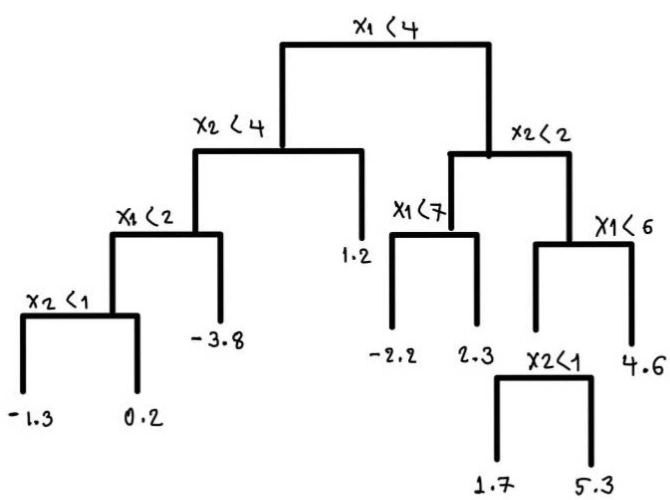
Question 1

part (a) Create a partitioned diagram



part (b) Sketch the tree

We have the following paths to terminal nodes:



Question 2

part (a) Reading the data

```
Hitters <- read.csv("Hitters2.csv", header=TRUE,  sep=",")
str(Hitters)

## 'data.frame':    263 obs. of  20 variables:
##  $ AtBat    : int  315 479 496 321 594 185 298 323 401 574 ...
##  $ Hits     : int  81 130 141 87 169 37 73 81 92 159 ...
##  $ HmRun    : int  7 18 20 10 4 1 0 6 17 21 ...
##  $ Runs     : int  24 66 65 39 74 23 24 26 49 107 ...
##  $ RBI      : int  38 72 78 42 51 8 24 32 66 75 ...
##  $ Walks    : int  39 76 37 30 35 21 7 8 65 59 ...
##  $ Years    : int  14 3 11 2 11 2 3 2 13 10 ...
##  $ CAtBat   : int  3449 1624 5628 396 4408 214 509 341 5206 4631 ...
##  $ CHits    : int  835 457 1575 101 1133 42 108 86 1332 1300 ...
##  $ CHmRun   : int  69 63 225 12 19 1 0 6 253 90 ...
##  $ CRuns    : int  321 224 828 48 501 30 41 32 784 702 ...
##  $ CRBI     : int  414 266 838 46 336 9 37 34 890 504 ...
##  $ CWalks   : int  375 263 354 33 194 24 12 8 866 488 ...
##  $ League   : chr  "N" "A" "N" "N" ...
##  $ Division : chr  "W" "W" "E" "E" ...
##  $ PutOuts  : int  632 880 200 805 282 76 121 143 0 238 ...
##  $ Assists  : int  43 82 11 40 421 127 283 290 0 445 ...
##  $ Errors   : int  10 14 3 4 25 7 9 19 0 22 ...
##  $ Salary   : num  475 480 500 91.5 750 ...
##  $ NewLeague: chr  "N" "A" "N" "N" ...

train = Hitters[1:200,]
dim(train)

## [1] 200  20

test = Hitters[201:263,]
dim(test)

## [1] 63 20
```

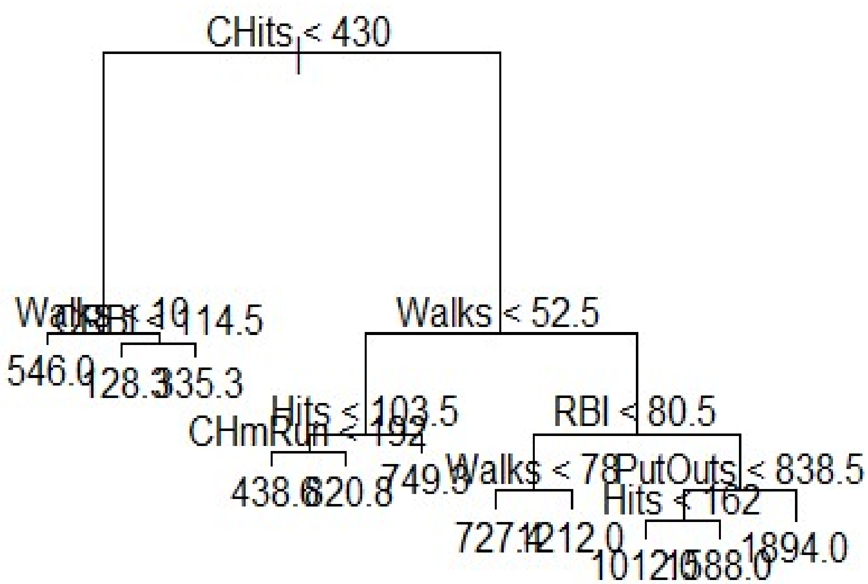
part (b)

Fit a decision tree

```
library(tree)

## Warning: package 'tree' was built under R version 4.0.4

set.seed(1)
tree.hitters = tree(Salary~., train)

## Warning in tree(Salary ~ ., train): NAs introduced by coercion

summary(tree.hitters)
```

```
##
## Regression tree:
## tree(formula = Salary ~ ., data = train)
## Variables actually used in tree construction:
## [1] "CHits"   "Walks"   "CRBI"    "Hits"    "CHmRun"  "RBI"     "PutOuts"
## Number of terminal nodes:  11
## Residual mean deviance:  66230 = 12520000 / 189
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -620.80 -113.90  -18.32    0.00   73.95 1581.00
```

```r
# A decision tree
plot(tree.hitters)
text(tree.hitters)
```




```r
# The test MSE
y_pred = predict(tree.hitters, newdata = test)
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by
coercion
```

```r
y_test = test$Salary
tree.mse = mean((y_test - y_pred)^2)
sprintf('%s = %10.3f', 'The test MSE of the decision tree', tree.mse)
```

```
## [1] "The test MSE of the decision tree =  63509.141"
```

The **test MSE of the decision tree is 63509.141.**

part (c)

Fit a model using baging method

```
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.0.4

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

set.seed(1)
bag.hitters = randomForest(Salary~., train, mtry = 19, importance = TRUE)
bag.hitters

##
## Call:
##  randomForest(formula = Salary ~ ., data = train, mtry = 19, importance =
TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 19
##
##           Mean of squared residuals: 102465.3
##                     % Var explained: 54.27

# The test MSE
y_pred = predict(bag.hitters, newdata = test)
y_test = test$Salary
bag.mse = mean((y_test - y_pred)^2)
sprintf('%s = %10.3f', 'The test MSE of the model using bagging method',
bag.mse)

## [1] "The test MSE of the model using bagging method =  51720.318"
```

The **test MSE of the model using bagging method is 51720.318.**

part (d)

Fit a random forest with m = 5

```
library(randomForest)
set.seed(1)
rf.hitters = randomForest(Salary~., train, mtry = 5, importance = TRUE)
rf.hitters

##
## Call:
##  randomForest(formula = Salary ~ ., data = train, mtry = 5, importance =
TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
```

```
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 100191.8
##                     % Var explained: 55.28
```

```r
# The test MSE
y_pred = predict(rf.hitters, newdata = test)
y_test = test$Salary
rf.mse = mean((y_test - y_pred)^2)
sprintf('%s = %10.3f', 'The test MSE of the model using random forest
method', rf.mse)
```

```
## [1] "The test MSE of the model using random forest method =  50125.268"
```

The **test MSE of the model using random forest method = 50125.268.**

part (e)

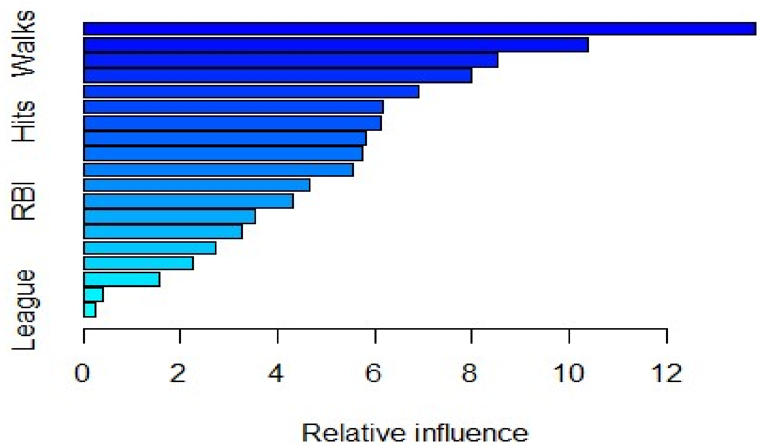
Fit a model using boosting technique

```r
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.0.4
```

```
## Loaded gbm 2.1.8
```

```r
set.seed(1)

# Since we can use boosting technique to fit a model if variable is character
type, we change them to factor.
train$League = as.factor(train$League)
train$Division = as.factor(train$Division)
train$NewLeague = as.factor(train$NewLeague)
test$League = as.factor(test$League)
test$Division = as.factor(test$Division)
test$NewLeague = as.factor(test$NewLeague)


boost.hitters = gbm(Salary~., train, distribution = 'gaussian', n.trees =
10000, shrinkage = 0.001, interaction.depth = 4, verbose = F)
summary(boost.hitters)
```




```
##                 var    rel.inf
## CHmRun       CHmRun 13.8448034
## Walks         Walks 10.3858366
## CRBI           CRBI  8.5447607
## CAtBat       CAtBat  7.9704334
## CWalks       CWalks  6.8888374
## PutOuts     PutOuts  6.1811218
## Hits           Hits  6.1231163
## CRuns         CRuns  5.8323080
## Years         Years  5.7559594
## CHits         CHits  5.5595170
## Assists     Assists  4.6674158
## RBI             RBI  4.2950680
## HmRun         HmRun  3.5188978
## AtBat         AtBat  3.2510105
## Runs           Runs  2.7158146
## Errors       Errors  2.2655348
## Division   Division  1.5664998
## NewLeague NewLeague  0.3794429
## League       League  0.2536219
```

```
# The test MSE
y_pred = predict(boost.hitters, newdata = test, n.trees = 10000)
y_test = test$Salary
boost.mse = mean((y_test - y_pred)^2)
sprintf('%s = %10.3f', 'The test MSE of the model using boosting technique',
boost.mse)
```

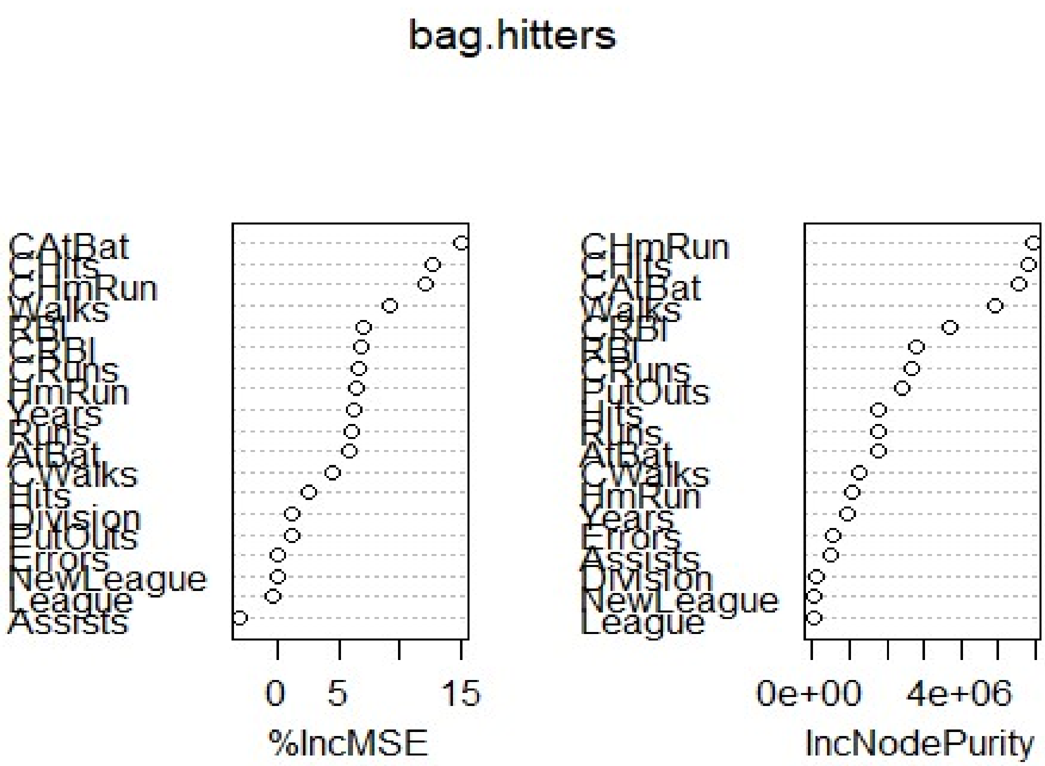
```
## [1] "The test MSE of the model using boosting technique =  56466.186"
```

The test MSE of the model using boosting technique = 56466.186.

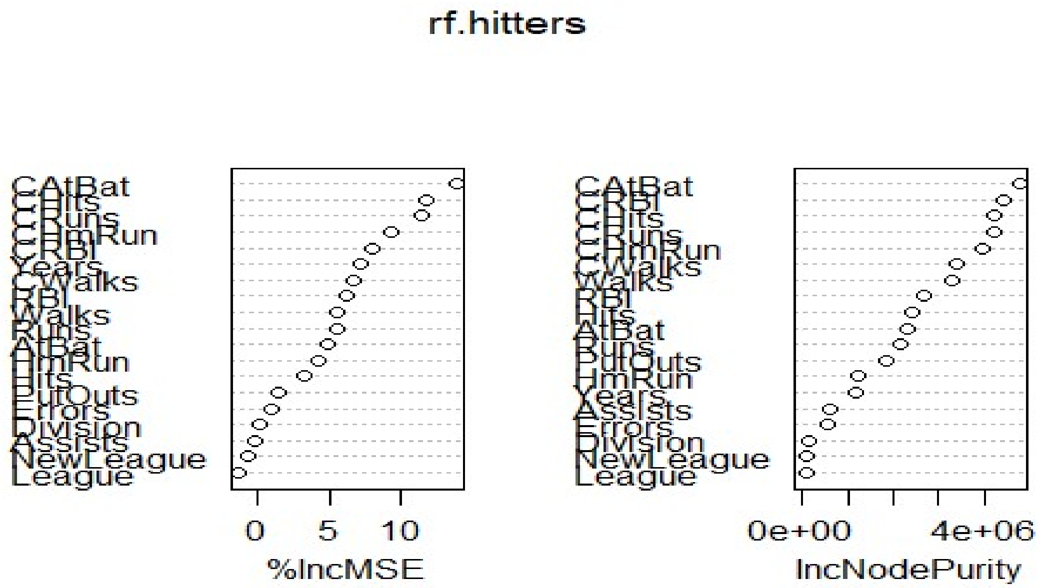
part (f)

Plot the feature importance

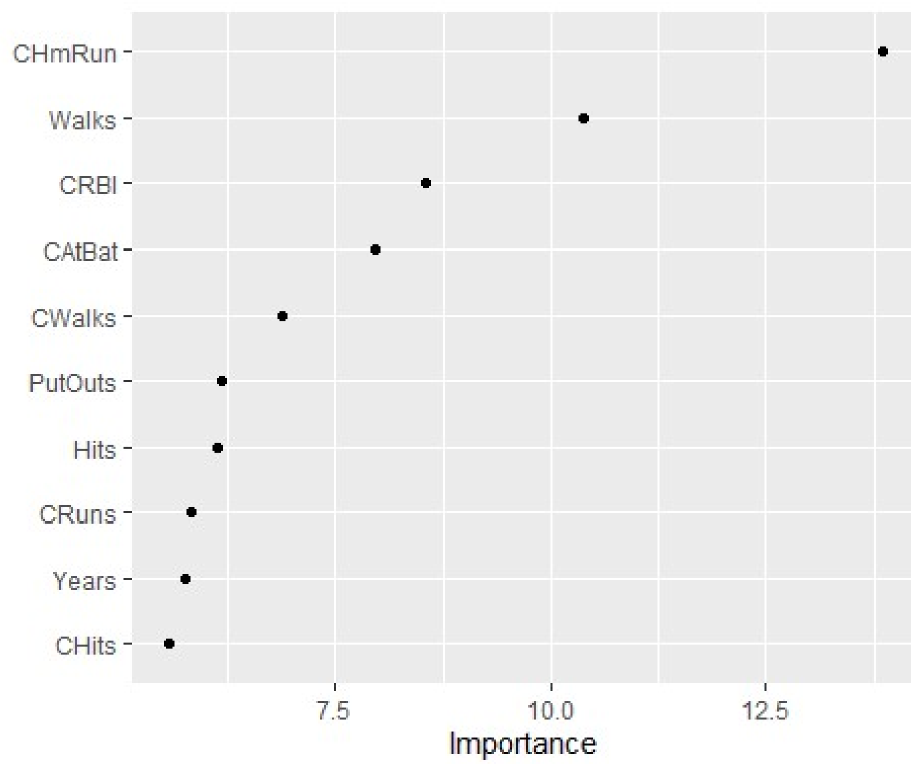
```
library(vip)

## Warning: package 'vip' was built under R version 4.0.4

##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
##
##     vi

# the feature importance plot for part (c)
varImpPlot(bag.hitters)
```



```r
# the feature importance plot for part (d)
varImpPlot(rf.hitters)
```




```r
# the feature importance plot for part (f)
vip(boost.hitters, geom = "point")
```



Question 3

```
# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# reshape to be [samples][pixels][width][height]
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(25, (7, 7), activation='relu', input_shape=(28, 28,1)))
    model.add(MaxPooling2D((4, 4)))
    model.add(Dropout(0.25))
    model.add(Conv2D(15, (4, 4),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(196, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # compile model
    model.compile(optimizer= 'adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# build the model
model = define_model()
# Fit the model:
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=7, batch_size=200)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Large CNN Error: %.2f%%" % (100-scores[1]*100))
print("Large CNN Accuracy: %2.2f%%" % (scores[1]*100))
```

```
Epoch 1/7
300/300 [==============================] - 14s 48ms/step - loss: 0.8665 - accuracy: 0.7003 - val_loss: 0.1877 - val_accurac
y: 0.9450
Epoch 2/7
300/300 [==============================] - 15s 49ms/step - loss: 0.3556 - accuracy: 0.8841 - val_loss: 0.1266 - val_accurac
y: 0.9619
Epoch 3/7
300/300 [==============================] - 14s 48ms/step - loss: 0.2719 - accuracy: 0.9121 - val_loss: 0.1032 - val_accurac
y: 0.9719
Epoch 4/7
300/300 [==============================] - 20s 65ms/step - loss: 0.2351 - accuracy: 0.9241 - val_loss: 0.0893 - val_accurac
y: 0.9726
Epoch 5/7
300/300 [==============================] - 16s 54ms/step - loss: 0.2053 - accuracy: 0.9338 - val_loss: 0.0789 - val_accurac
y: 0.9777
Epoch 6/7
300/300 [==============================] - 17s 58ms/step - loss: 0.1870 - accuracy: 0.9387 - val_loss: 0.0756 - val_accurac
y: 0.9773
Epoch 7/7
300/300 [==============================] - 20s 68ms/step - loss: 0.1741 - accuracy: 0.9445 - val_loss: 0.0745 - val_accurac
y: 0.9775
Large CNN Error: 2.25%
Large CNN Accuracy: 97.75%
```

The test accuracy is ~98% and the test error is ~2%. What we can observe for this is the error would be around ~1-2% and the accuracy would be around ~98-99%.