Supplemental Information: CH3MS-RF

Emily Barnes Franklin

1/29/2022

Supplemental Information: R Code for Implementation of Ch3MS-RF

(Chemical Characterization by Chromatography-MS Random Forest Modeling)

Custom Functions

Here we define custom functions that will be useful later in the analysis.

```
# left join, but filling all empty cells with "False" instead of NA
left_joinF <- function(x, y, fill = FALSE){
  z <- left_join(x, y)
  tmp <- setdiff(names(z), names(x))
  z <- replace_na(z, setNames(as.list(rep(fill, length(tmp))), tmp))
  z
}

# left join, but filling all empty cells with "0" instead of NA
left_join0<- function(x, y, fill = 0){
  z <- left_join(x, y)
  tmp <- setdiff(names(z), names(x))
  z <- replace_na(z, setNames(as.list(rep(fill, length(tmp))), tmp))
  z
}

#r squared
rsq <- function (x, y) cor(x, y) ^ 2

#Out of sample R squared, for evaluating prediction performance
# with a training and test set
OSR2 <- function(predictions, train, test) {
  SSE <- sum((test - predictions)^2)
  SST <- sum((test - mean(train))^2)
  r2 <- 1 - SSE/SST
  return(r2)
}

# function for formatting mass spectral parsing
trim.leading <- function (x)  sub("^\\s+", "", x)
```

Reading In Data Files

In this section, the necessary data files for analysis are read in. The first of these files is a .csv file containing the names, retention indexes, and mass spectra of all of the external standard (or training dataset) components. The mass spectra are in the form of a semicolon separated string. The second file is a .csv file containing the names, formulae, functional group categories, and vapor pressures of the known training data set. Functional group categories are required to allow the external standard data set to be split into training and test subsets while maintaining a proportional representation of different functional group types between the sets. Note that any properties of interest could be added to this file and substituted into the training and property prediction framework described below. Finally, we provide an example unknown sample .csv for property prediction. The required fields in this file are Name (unique identifier for each unknown compound), retention index, and mass spectrum.

```
# First required file: .csv file containing mass spectra, see template file
# for formatting
MS_table_o <- read_csv("EBF_GoldsteinGroup_ExternalStandard_GoA2018_ms.csv")
# Second required file: .csv containing relevant properties of compounds
# (both those needed to split training and test datasets and properties
# that are the intended predicted result)
Prop_table_o <- read_csv("EBF_GoldsteinGroup_ExternalStandard_GoA2018_properties.csv")

# Third resuired file: .csv of retenion index and mass spectra of unknown sample \
#compounds of interest
Unk_table_o <- read_csv("EBF_GoldsteinGroup_Template_Unknown_Dataset.csv")
```

Parsing Mass Spectra

The principle peaks and the mass differences between key peaks are the key indicators of functional groups that exert the greatest influence on the calibration factor of the compound in question. In this section, the top 10 peaks (by intensity) are identified for each compound. Then, the top 40 peaks (the 40 peaks which appear in the most compounds from the external standard list) are turned into factors, for which each compound is represented by 0 if the peak is not in the top 10 for that compound or an intensity value between 0 and 1000 if that peak is one of the top 10 peaks for that compound.

```
p.n <- 10 #Number of major peaks to consider from each mass spectrum when making list of most comm
on peaks
p.l.l <- 40 # number of common peaks to turn into features


ES_table_t = MS_table_o %>%
    dplyr::mutate(MS = strsplit(as.character(MS), ";")) %>%
    unnest(MS) %>%
    filter(MS != "") %>%
    dplyr::mutate(MS2 = trim.leading(MS))



ES_table_3 = separate(data = ES_table_t, col = MS2, into = c("mz", "intensity"), sep = " ")

ES_table_full = ES_table_3 %>%
  dplyr::mutate(intensity = as.numeric(intensity)) %>%
  dplyr::mutate(mz = as.numeric(mz)) %>%
  arrange(desc(intensity)) %>%
  arrange(desc(Name)) %>%
  group_by(Name) %>%
  dplyr::mutate(my_ranks = order(order(intensity, decreasing=TRUE))) %>%
  ungroup()

ES_names <- MS_table_o %>%
  dplyr::select("Name")

ES_table_trimmed = ES_table_full %>%
  filter(my_ranks <= p.n)

mz_tab <- as.data.frame(table(ES_table_trimmed$mz))
mz_tab <- mz_tab %>%
  dplyr::mutate(mz = as.character(Var1)) %>%
  dplyr::mutate(mz = as.numeric(mz))
mz_tab <- mz_tab %>%
  arrange(desc(Freq)) %>%
  dplyr::mutate(freq_ranks = order(order(Freq, decreasing = TRUE)))

ES_table_ranked <- ES_table_trimmed %>%
  left_join(mz_tab) %>%
  filter(freq_ranks <= p.l.l)
```

```
## Joining, by = "mz"
```

```
ES_common_mz = ES_table_ranked %>%
  dplyr::select(Name, mz, intensity) %>%
  dplyr::mutate(mz = paste("mz_", mz, sep = "_")) %>%
  dplyr::mutate(mz_obs = TRUE) %>%
  dplyr::mutate(mz_intensity = intensity)

peak_list.es <- ES_common_mz %>%
  dplyr::select(mz) %>%
  dplyr::mutate(istop40 = TRUE) %>%
  distinct(mz, istop40, .keep_all = TRUE)

wide_mz = ES_common_mz %>%
  dplyr::select(Name, mz, mz_obs) %>%
  spread(mz, mz_obs, fill = FALSE)

wide_mz <- ES_names %>%
  left_joinF(wide_mz)
```

```
## Joining, by = "Name"
```

```
wide_mz_i = ES_common_mz %>%
  dplyr::select(Name, mz, mz_intensity) %>%
  spread(mz, mz_intensity, fill = 0)

wide_mz_i <- ES_names %>%
  left_join0(wide_mz_i)
```

```
## Joining, by = "Name"
```

Next, the losses are determined. The mass differences between the top 5 peaks for each compound are evaluated and turned into their own list. The 20 losses/neutral mass differences which appear most frequently become true/false factors for inclusion in future analysis.

```
l.n <- 5 # number of major peaks to consider when identifying losses between major peaks
l.l.l <- 20 # number of common neutral mass differences to convert into features

unique_names = unique(ES_table_trimmed$Name)

x = seq(1, l.n, 1)
y = x

d2 <- expand.grid(x = x, y = y, KEEP.OUT.ATTRS = FALSE)
d2 <- d2 %>%
  filter(y > x)

xt = d2$x

yt = d2$y

names_losses = as.character(rep(unique_names, each = length(d2$x)))
losses_vec = rep(999, times = length(names_losses))
loss_num = rep(seq(1, length(xt), 1), times = length(unique_names))

losses <- data.frame(names_losses, losses_vec, loss_num)
losses = losses %>%
  dplyr::mutate(names_losses = as.character(names_losses))



for(i in 1:nrow(losses)) {
  #i = 1

  name_t = names_losses[i]
  loss_num_t = losses$loss_num[i]

  upper_index = xt[loss_num_t]
  lower_index = yt[loss_num_t]

  df_t = ES_table_trimmed %>%
    filter(Name == name_t)

  upper_mz = df_t$mz[upper_index]
  lower_mz = df_t$mz[lower_index]


  losses$losses_vec[i] = abs(upper_mz - lower_mz)

}

arranged_mz <- ES_table_trimmed %>%
  arrange(desc(mz)) %>%
  arrange(desc(Name)) %>%
  dplyr::mutate(loss_fast = 0)

for(i in 2:nrow(arranged_mz)){
```

```
  arranged_mz$loss_fast[i]= arranged_mz$mz[i-1]-arranged_mz$mz[i]
}

arranged_mz.t <- as.data.frame(table(arranged_mz$loss_fast)) %>%
  dplyr::mutate(num_loss = as.character(Var1)) %>%
  dplyr::mutate(num_loss = as.numeric(num_loss)) %>%
  filter(num_loss > 0) %>%
  arrange(desc(Freq)) %>%
  dplyr::mutate(id = row_number())


losses = losses %>%
  left_join(arranged_mz.t, by = c("losses_vec"= "num_loss"))

losses_trimmed = losses %>%
  filter(id <= l.l.l)



ES_common_losses = losses_trimmed %>%
  dplyr::select(names_losses, losses_vec) %>%
  dplyr::mutate(losses_vec = paste("loss_", losses_vec, sep = "_")) %>%
  dplyr::mutate(Loss_obs = TRUE) %>%
  distinct()

loss_list.es <- ES_common_losses %>%
  dplyr::select(losses_vec) %>%
  dplyr::mutate(istop20 = TRUE) %>%
  distinct(losses_vec, istop20, .keep_all = TRUE)

wide_losses = ES_common_losses %>%
  spread(losses_vec, Loss_obs, fill = FALSE) %>%
  right_join(ES_names, by = c("names_losses"= "Name")) %>%
  rename(Name = names_losses)

wide_losses[is.na(wide_losses)] <- FALSE
```

This process is repeated for the data set of unknown compounds, as the data sets must be identically formatted for the property prediction to proceed smoothly. The peaks and mass differences from the known data set are again the features, as generating new features for the unknown compounds would create a lack similarity between the training data and the predictive data.

```
unk_table_t = Unk_table_o %>%
    dplyr::mutate(MS = strsplit(as.character(MS), ";")) %>%
    unnest(MS) %>%
    filter(MS != "") %>%
    dplyr::mutate(MS2 = trim.leading(MS))




unk_table_3 = separate(data = unk_table_t, col = MS2, into = c("mz", "intensity"), sep = " ")

unk_table_full = unk_table_3 %>%
  dplyr::mutate(intensity = as.numeric(intensity)) %>%
  dplyr::mutate(mz = as.numeric(mz)) %>%
  arrange(desc(intensity)) %>%
  arrange(desc(Name)) %>%
  group_by(Name) %>%
  dplyr::mutate(my_ranks = order(order(intensity, decreasing=TRUE))) %>%
  ungroup()

unk_names <- Unk_table_o %>%
  dplyr::select("Name")

unk_table_trimmed = unk_table_full %>%
  filter(my_ranks <= p.n)

mz_tab.a <- as.data.frame(table(unk_table_trimmed$mz))

mz_tab.a <- mz_tab.a %>%
  dplyr::mutate(mz = as.character(Var1)) %>%
  dplyr::mutate(mz = as.numeric(mz))
mz_tab.a <- mz_tab.a %>%
  arrange(desc(Freq)) %>%
  dplyr::mutate(freq_ranks = order(order(Freq, decreasing = TRUE)))

unk_table_ranked <- unk_table_trimmed %>%
  left_join(mz_tab.a)
```

```
## Joining, by = "mz"
```

```
# note- here selecting only the MZ of the most common ES list so that the
#model will be usable
unk_common_mz = unk_table_ranked %>%
  dplyr::select(Name, mz, intensity) %>%
  dplyr::mutate(mz = paste("mz_", mz, sep = "_")) %>%
  dplyr::mutate(mz_obs = TRUE) %>%
  dplyr::mutate(mz_intensity = intensity) %>%
  right_join(peak_list.es) %>%
  filter(istop40 == TRUE) %>%
  dplyr::select(-istop40)
```

```
## Joining, by = "mz"
```

```
wide_mz.a = unk_common_mz %>%
  dplyr::select(Name, mz, mz_obs) %>%
  spread(mz, mz_obs, fill = FALSE) %>%
  filter(!is.na(Name))

wide_mz.a <- unk_names %>%
  left_joinF(wide_mz.a)
```

```
## Joining, by = "Name"
```

```
wide_mz_i.a = unk_common_mz %>%
  dplyr::select(Name, mz, mz_intensity) %>%
  spread(mz, mz_intensity, fill = 0) %>%
  filter(!is.na(Name))

wide_mz_i.a <- unk_names %>%
  left_join0(wide_mz_i.a)
```

```
## Joining, by = "Name"
```

```
# moving on to losses of the sample/unknown compounds

unique_names.a = unique(unk_table_trimmed$Name)

x = seq(1, l.n, 1)
y = x

d2 <- expand.grid(x = x, y = y, KEEP.OUT.ATTRS = FALSE)
d2 <- d2 %>%
  filter(y > x)

xt = d2$x

yt = d2$y

names_losses.a = as.character(rep(unique_names.a, each = length(d2$x)))
losses_vec.a = rep(999, times = length(names_losses.a))
loss_num.a = rep(seq(1, length(xt), 1), times = length(unique_names.a))

losses.a <- data.frame(names_losses= names_losses.a, losses_vec = losses_vec.a, loss_num = loss_nu
m.a)
losses.a = losses.a %>%
  dplyr::mutate(names_losses = as.character(names_losses))



for(i in 1:nrow(losses.a)) {
  #i = 1

  name_t = names_losses.a[i]
  loss_num_t = losses.a$loss_num[i]

  upper_index = xt[loss_num_t]
  lower_index = yt[loss_num_t]

  df_t = unk_table_trimmed %>%
    filter(Name == name_t)

  upper_mz = df_t$mz[upper_index]
  lower_mz = df_t$mz[lower_index]


  losses.a$losses_vec[i] = abs(upper_mz - lower_mz)

}

arranged_mz.a <- unk_table_trimmed %>%
  arrange(desc(mz)) %>%
  arrange(desc(Name)) %>%
  dplyr::mutate(loss_fast = 0)

for(i in 2:nrow(arranged_mz.a)){
```

```
  arranged_mz.a$loss_fast[i]= arranged_mz.a$mz[i-1]-arranged_mz$mz[i]
}

arranged_mz.t.a <- as.data.frame(table(arranged_mz.a$loss_fast)) %>%
  dplyr::mutate(num_loss = as.character(Var1)) %>%
  dplyr::mutate(num_loss = as.numeric(num_loss)) %>%
  filter(num_loss > 0) %>%
  arrange(desc(Freq)) %>%
  dplyr::mutate(id = row_number())

losses.a = losses.a %>%
  left_join(arranged_mz.t.a, by = c("losses_vec"= "num_loss"))

losses_trimmed.a = losses.a


unk_common_losses = losses_trimmed.a %>%
  dplyr::select(names_losses, losses_vec) %>%
  dplyr::mutate(losses_vec = paste("loss_", losses_vec, sep = "_")) %>%
  dplyr::mutate(Loss_obs = TRUE) %>%
  distinct() %>%
  right_join(loss_list.es) %>%
  filter(istop20 == TRUE) %>%
  dplyr::select(-istop20)
```

```
## Joining, by = "losses_vec"
```

```
wide_losses.a = unk_common_losses %>%
  spread(losses_vec, Loss_obs, fill = FALSE) %>%
  filter(!is.na(names_losses)) %>%
  right_join(unk_names, by = c("names_losses"= "Name")) %>%
  rename(Name = names_losses)

wide_losses.a[is.na(wide_losses.a)] <- FALSE
```

Parsing Chemical Formulae

Now that the mass spectra have been parsed, we parse the chemical formulae of the external standard compounds to generate potential properties of interest for modeling.

```
Element <- c("C", "H", "O", "N", "S")
d.e <- data.frame(Element = Element)

# creating columns for the number of atoms of each element to parse chemical formulae
ES_info_all.form <-  Prop_table_o %>%
  dplyr::mutate(Cnum = 0) %>%
  dplyr::mutate(Hnum = 0) %>%
  dplyr::mutate(Onum = 0) %>%
  dplyr::mutate(Nnum = 0) %>%
  dplyr::mutate(Snum = 0)

for(i in 1:nrow(ES_info_all.form)){

  #i = 10
  form.t <- ES_info_all.form$ChemFormula[i]

  form.t1 <- data.frame(makeup(form.t))
  setDT(form.t1, keep.rownames = TRUE)[]
  names(form.t1)<- c("Element", "num")

  form.t2 <- d.e %>%
    left_join0(form.t1)

  form.t3 <- data.frame(form.t2[,-1])
  rownames(form.t3) <- form.t2[,1]

  form.t4 <- data.frame(t(form.t3))
  names(form.t4) <-c("C", "H", "O", "N", "S")

  ES_info_all.form$Cnum[i]= form.t4$C[1]
  ES_info_all.form$Hnum[i]= form.t4$H[1]
  ES_info_all.form$Onum[i]= form.t4$O[1]
  ES_info_all.form$Nnum[i]= form.t4$N[1]
  ES_info_all.form$Snum[i]= form.t4$S[1]
}

ES_info_all.prop <- ES_info_all.form %>%
  dplyr::mutate(O_C_ratio = Onum/Cnum) %>%
  dplyr::mutate(H_C_ratio = Hnum/Cnum) %>%
  dplyr::mutate(OSc = (2*O_C_ratio)-H_C_ratio)

pf <- names(ES_info_all.prop)
# creating a list for easy removal of non predictive features later
Predictive_Features <- pf[pf!="Retention_Index"]
```

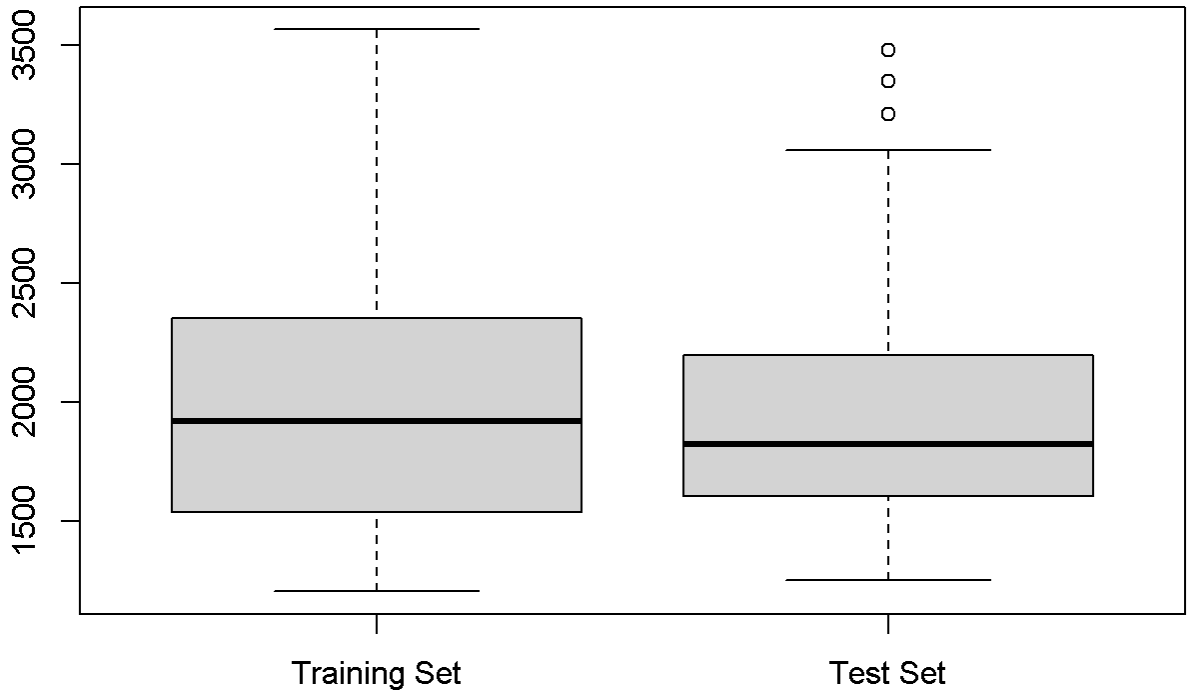
Preparing training and test sets

Here we identify the property of interest, prepare a data tables with only the relevant predictive information and the property of interest, and split into test and training sets. In this example case, we select carbon number as the property of interest to predict using the model.

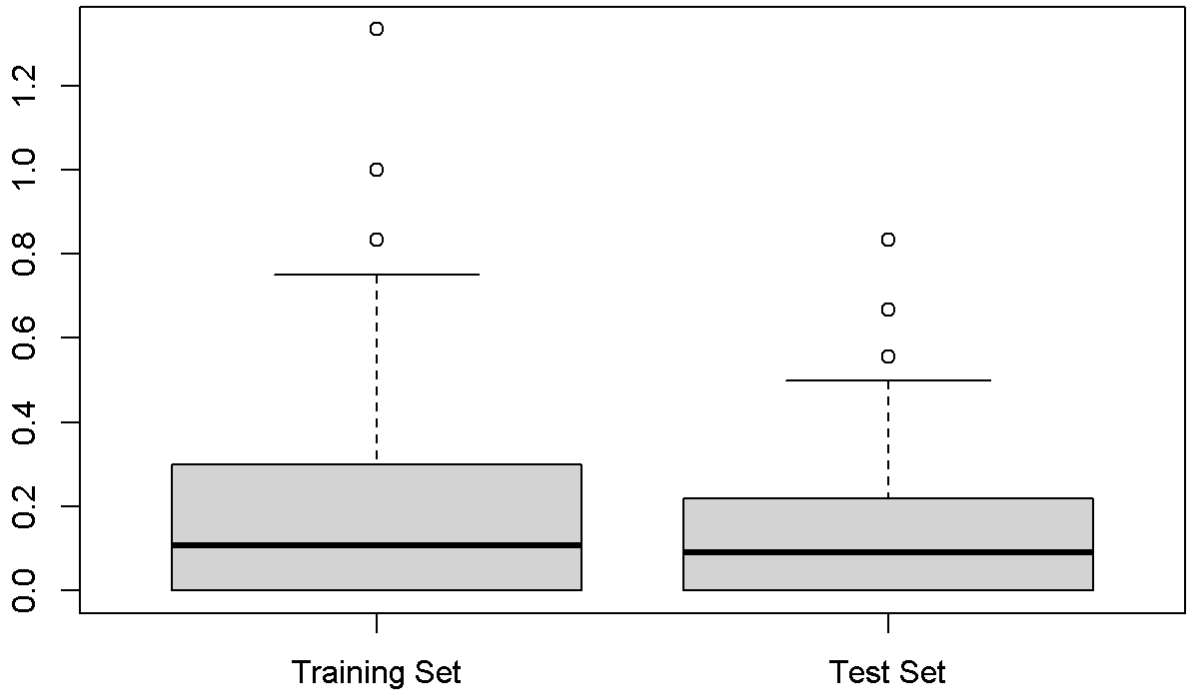
```
ES_info_all <- ES_info_all.prop %>%
  mutate(Prop = Cnum) %>% #Cnum can be substituted for any other property of interest, including v
apor pressure, O:C, OSc, etc.
  left_join(wide_mz_i) %>%
  left_join(wide_losses) %>%
  filter(!is.na(Prop)) # limiting training and test sets to compounds for which we have known prop
erties for the training compounds
```

```
## Joining, by = "Name"
## Joining, by = "Name"
```

```
# now splitting standard compounds into training and test sets
split.ratio = .8 #setting split to 80:20- this can be adjusted depending on external standard size
set.seed(111)
split = sample.split(ES_info_all$Func_Group_Cat_3, SplitRatio = 0.8)

ES_info_all.train <- filter(ES_info_all, split == TRUE)
ES_info_all.test <- filter(ES_info_all, split == FALSE)

#Visualizations to identify how similarly distributed the training and test sets are

boxplot(ES_info_all.train$Retention_Index, ES_info_all.test$Retention_Index, names = c("Training S
et", "Test Set"))
```



```
boxplot(ES_info_all.train$O_C_ratio, ES_info_all.test$O_C_ratio, names = c("Training Set", "Test S
et"))
```




```
# if the test set is outside the bounds of the training, a different split should be selected as t
he predictive performance will not be good.
# The same check will be applied to the unknown sample compounds.
# training set appears to cover the bounds of the test set fairly well, proceeding

# now removing all of the other predictive properties so that the data frames are ready for the mo
del
ES.train <- ES_info_all.train %>%
  dplyr::select(-Predictive_Features)
```

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

```
ES.test <- ES_info_all.test %>%
  dplyr::select(-Predictive_Features)
```

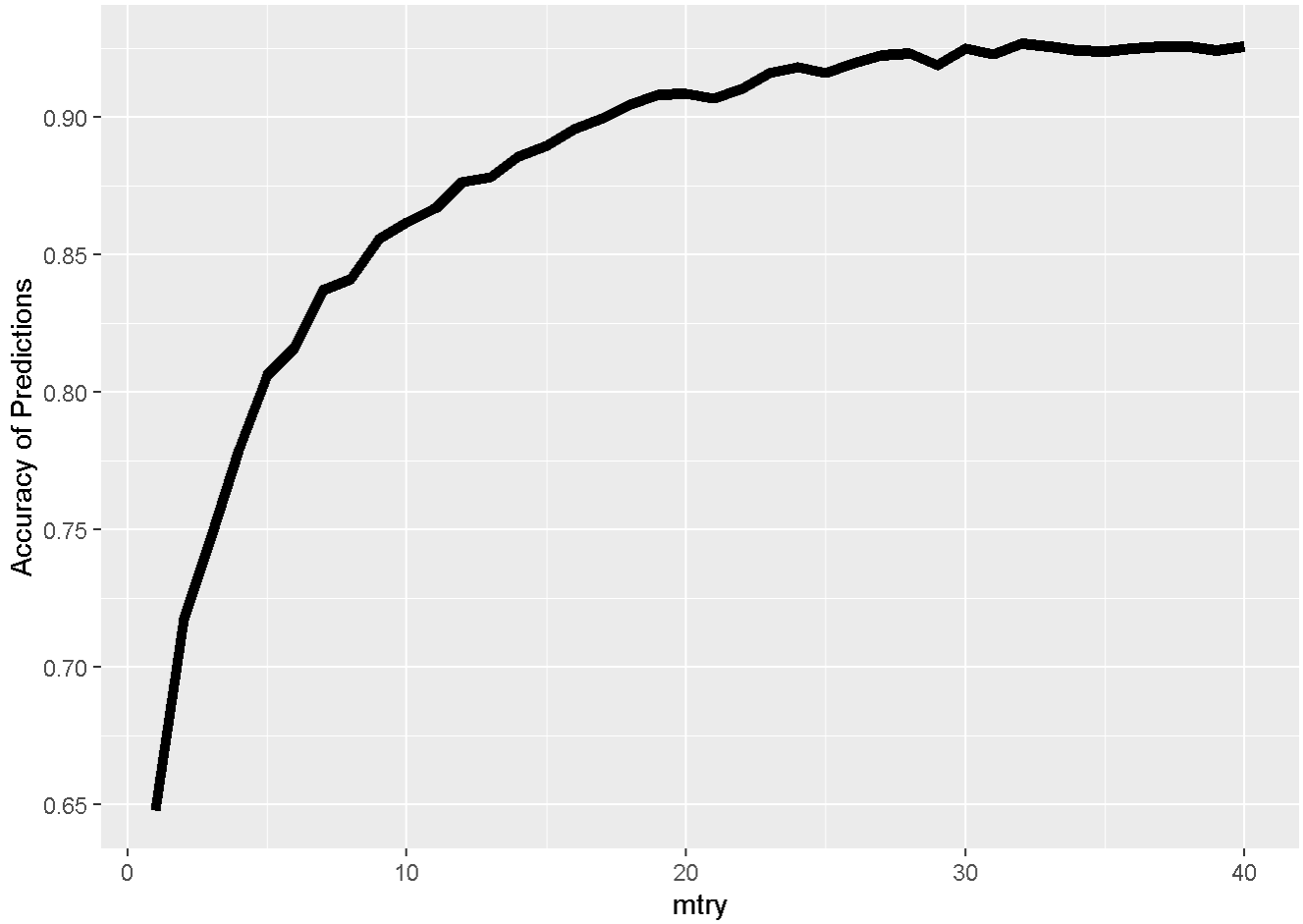
Random Forest Modeling of Selected Property

Next, we use 5-fold cross validation to identify the optimal number of features for feature restriction in creating diverse trees, identify the optimal model, predict the properties for the test set, and evaluate prediction performance.

```
# training random forest model on the training ES subset
rf.mtry.max = 40 # setting the maximum number of features for mtry feature restriction

# Proceeding with 5-fold cross validation
set.seed(144)
train.rf = train(Prop ~., data = ES.train, method = "rf", tuneGrid = data.frame(mtry=seq(1, rf.mtr
y.max, 1)), trControl = trainControl(method = "cv", number = 5), metric = "RMSE", na.action = na.e
xclude)

best.rf = train.rf$finalModel
best.rf
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = min(param$mtry, ncol(x)))
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 40
##
##           Mean of squared residuals: 5.038802
##                     % Var explained: 90.64
```

```
rf.plot <- ggplot(train.rf$results, aes(x=mtry, y=Rsquared)) + geom_line(lwd=2) +
  ylab("Accuracy of Predictions")
rf.plot
```




```
Cat.mm = as.data.frame(model.matrix(Prop ~., data = ES.test))

set.seed(124)
pred.best.rf = predict(best.rf, newdata = Cat.mm, type = "class")

# creating a field for the predicted property for the test set
ES.test$Prop.p <- pred.best.rf

#out of sample R2
OSR2(ES.test$Prop.p, ES.train$Prop, ES.test$Prop)
```

```
## [1] 0.8997875
```

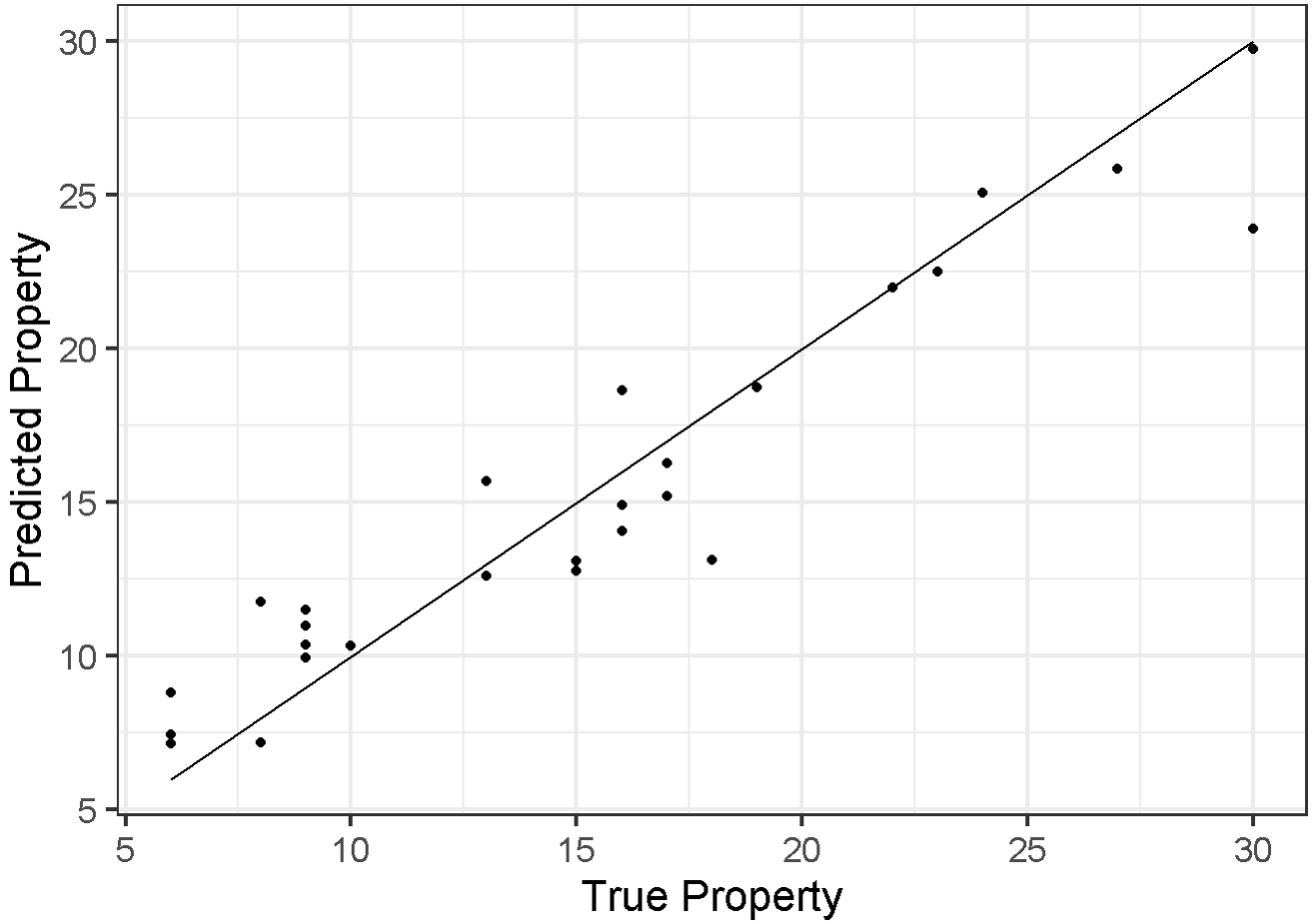
```
# mean average error
MAE(ES.test$Prop.p, ES.test$Prop)
```

```
## [1] 1.731485
```

```
# correlation coefficient of real and predicted values
rsq(ES.test$Prop.p, ES.test$Prop)
```

```
## [1] 0.9080001
```

```
line <- data.frame(x = c(min(ES.test$Prop), max(ES.test$Prop)),
                   y = c(min(ES.test$Prop), max(ES.test$Prop)))

# visualizing the distribution of true vs predicted properties
p.c <- ES.test %>%
  ggplot(aes(x = Prop, y = Prop.p))+
  geom_point()+
  geom_line(data = line, aes(x = x, y = y))+
  xlab("True Property")+
  ylab("Predicted Property")+
  theme_bw(base_size = 16)

p.c
```



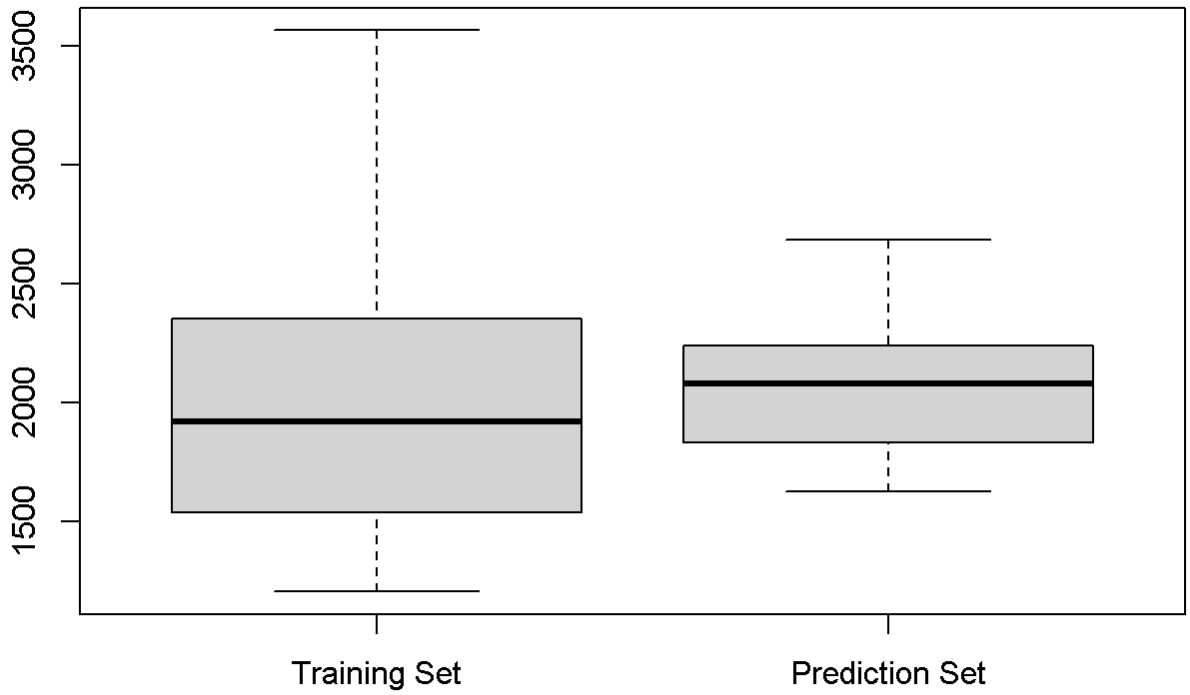
##

Predicting Properties of Unknown Compounds Next, we evaluate and prepare a dataframe for property prediction on the unknown compound dataset

```
Unk_table_mod.n <- Unk_table_o %>%
  dplyr::select(Name, Retention_Index) %>%
  left_join(wide_mz_i.a) %>%
  left_join(wide_losses.a)
```

```
## Joining, by = "Name"
## Joining, by = "Name"
```

```
boxplot(ES_info_all.train$Retention_Index, Unk_table_mod.n$Retention_Index, names = c("Training Se
t", "Prediction Set"))
```



```r
# clear here that the prediction/unknown data set lies within the bounds of the training data set, ok to proceed. If significant outliers exist, the training set should be altered to include more representative compounds or the outliers should be removed from prediction and not utilized to make conclusions about the data set.

Unk_table_mod <- Unk_table_mod.n %>%
  dplyr::select(-Name) %>%
  dplyr::mutate(Prop = -999) # creating a dummy variable to make the model matrix

Cat.mm.unk = as.data.frame(model.matrix(Prop ~., data = Unk_table_mod))

pred.best.unk <- predict(best.rf, newdata = Cat.mm.unk, type = "class")

Unk_table_mod.n$Prop.p <- pred.best.unk

# tidy final output table with the names and predicted properties of the sample compounds
Unk_table_tidy <- Unk_table_mod.n %>%
  dplyr::select(Name, Prop.p)

#exporting the output table for ease of use
#write.csv(Unk_table_tidy, "Example_Output_Table.csv")
```