Test 1

Mike Silva

2020-11-11

Available Online at: https://github.com/mikeasilva/CUNY-SPS/blob/master/DATA622/Test1.rmd (https://github.com/mikeasilva/CUNY-SPS/blob/master/DATA622/Test1.rmd)

Preliminaries

This test is an extension of the work done in Homework 1. I will be extending it with bagging and leave one out cross-validation (LOOCV), and adding in the results into the evaluation of the algorithms.

Load Packages

```
library(dplyr)
library(tidyr)
library(kableExtra)
library(caret)
library(e1071)
library(class)
library(ROCR)
library(stringr)
```

Read in the data

```
df <- read.csv("https://github.com/mikeasilva/CUNY-SPS/raw/master/DATA622/HW1.csv", stringsAsFac
tors = TRUE)
```

Data Prep

Since we will be using KNN we will need to normalize `X`. In order to get a fair comparison, we will used the same processed data to see how the algorithms preform.

```
normalize <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}

adjusted_df <- df
adjusted_df$X <- normalize(adjusted_df$X)
```

We also need to transform `Y` into dummy variables.

```
adjusted_df <- adjusted_df %>%
  mutate(Y_a = ifelse(Y == "a", 1, 0),
         Y_b = ifelse(Y == "b", 1, 0),
         Y_c = ifelse(Y == "c", 1, 0),
         Y_d = ifelse(Y == "d", 1, 0),
         Y_e = ifelse(Y == "e", 1, 0)) %>%
  select(-Y)
```

In order to evaluate the ability of the of the algorithms to generalize, the data needs to be divided into training and test sets. Since there is only 36 observations, I will be reserving 20% for the evaluation.

```
set.seed(123)
in_training_set <- createDataPartition(adjusted_df$label, p = 0.8, list = FALSE, times = 1)
training_df <- adjusted_df[in_training_set,]
test_df  <- adjusted_df[-in_training_set,]
# These subsets will help with training and evaluation
training_df_without_label <- training_df %>% select(-label)
test_df_without_label  <- test_df %>% select(-label)
training_df_label <- training_df$label
test_df_label <- test_df$label
```

Helper Functions

I will use the following helper functions. This first one takes a logistic regression model and returns predictions for the `data` data.frame

```
get_lr_yhat <- function(lr_model, data, label_col_name, threshold = 0.5){
  data_levels <- levels(data[[label_col_name]])
  cols_to_keep <- label_col_name != names(data)
  data <- data[,cols_to_keep]
  lr_yhat <- predict(lr_model, data, type = "response")
  lr_yhat <- as.factor(ifelse(lr_yhat <= threshold, data_levels[1], data_levels[2]))
  return(lr_yhat)
}
```

The second will take the the ground truth labels and predicted labels and create metrics for evaluation.

```
evaluate_algo <- function(ground_truth, yhat, algo){
  cm <- confusionMatrix(table(ground_truth, yhat))
  cm_table <- cm$table
  tpr <- cm_table[[1]] / (cm_table[[1]] + cm_table[[4]])
  fnr <- 1 - tpr
  fpr <- cm_table[[3]] / (cm_table[[3]] + cm_table[[4]])
  tnr <- 1 - fpr
  accuracy <- cm$overall[[1]]
  for_auc <- prediction(c(yhat), ground_truth)
  auc <- performance(for_auc, "auc")
  auc <- auc@y.values[[1]]
  return(data.frame(Algo = algo, AUC = auc, ACCURACY = accuracy, TPR = tpr, FPR = fpr, TNR = tn
r, FNR = fnr))
}
```

Now for the extensions for the test. This third function will be used to create a bootstrap sample. I will also make the assumption that we will use 100 boostrap samples in evaluating the performance of bagging.

```
get_bootstrap_sample <- function(data, bootstrap_proportion = 0.6, sample_with_replacement = TRU
E){
  n_bootstrap_observations <- round(nrow(data) * bootstrap_proportion, 0)
  return(data[sample(nrow(data), n_bootstrap_observations, replace = sample_with_replacement),])
}

n_bags <- 100
```

I will also need a way to aggregate/average the metrics

```
average_metrics <- function(metrics, algo){
  avg_metrics <- metrics %>%
  select(-Algo) %>%
  colMeans(na.rm = TRUE) %>%
  t() %>%
  data.frame("Algo" = c(algo), .)
  return(avg_metrics)
}
```

Baseline Models

First I will train the models following what was previously done in HW1 and collect the metrics on both the training and test data sets

Logistic Regression

```
lr_model <- glm(label ~ ., data = training_df, family = "binomial")
summary(lr_model)
```

```
Call:
glm(formula = label ~ ., family = "binomial", data = training_df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6882  -0.8067  -0.5294   0.7799   2.1677

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.8063     1.2429  -0.649   0.5165
X            -1.3152     1.2670  -1.038   0.2992
Y_a           0.8276     1.4489   0.571   0.5679
Y_b           1.0500     1.4752   0.712   0.4766
Y_c           3.0900     1.6563   1.866   0.0621 .
Y_d           2.4556     1.7455   1.407   0.1595
Y_e          -0.1275     1.5916  -0.080   0.9361
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 40.381  on 29  degrees of freedom
Residual deviance: 32.471  on 23  degrees of freedom
AIC: 46.471

Number of Fisher Scoring iterations: 4
```

Capacity to Learn

The sample data are too small (30) to have any confidence in the Logistic Regression model, so I will be throwing this one out. Bagging and LOOCV will not resolve this issue.

```
capacity_to_learn <- data.frame("Algo" = c("LR", "LR with Bagging", "LR with LOOCV"), AUC=c(NA), ACCURACY=c(NA), TPR=c(NA), FPR=c(NA), TNR=c(NA), FNR=c(NA))
```

Capacity to Generalize

Again the sample data are too small to use to make meaningful inferences using Logistic Regression. Bagging and LOOCV will not help and will be excluded from the metrics.

```
capacity_to_generalize <- data.frame("Algo" = c("LR", "LR with Bagging", "LR with LOOCV"), AUC=c(NA), ACCURACY=c(NA), TPR=c(NA), FPR=c(NA), TNR=c(NA), FNR=c(NA))
```

Naive Bayes

I will repeat the same process for Naive Bayes.

Capacity to Learn

```
nb_model <- naiveBayes(label ~ ., data = training_df)
training_nb_yhat <- predict(nb_model, training_df_without_label)
capacity_to_learn <- rbind(capacity_to_learn, evaluate_algo(training_df_label, training_nb_yhat,
"NB"))
```

Capacity to Generalize

```
nb_yhat <- predict(nb_model, test_df_without_label)
capacity_to_generalize <- rbind(capacity_to_generalize, evaluate_algo(test_df_label, nb_yhat, "N
B"))
```

KNN

Finally I will repeat the process for KNN where k=3 and k=5

Capacity to Learn

```
training_knn3_yhat <- knn(training_df_without_label, training_df_without_label, training_df_labe
l, k = 3)
capacity_to_learn <- rbind(capacity_to_learn, evaluate_algo(training_df_label, training_knn3_yha
t, "KNN3"))

training_knn5_yhat <- knn(training_df_without_label, training_df_without_label, training_df_labe
l, k = 5)
capacity_to_learn <- rbind(capacity_to_learn, evaluate_algo(training_df_label, training_knn5_yha
t, "KNN5"))
```

Capacity to Generalize

```
knn3_yhat <- knn(training_df_without_label, test_df_without_label, training_df_label, k = 3)
capacity_to_generalize <- rbind(capacity_to_generalize, evaluate_algo(test_df_label, knn3_yhat,
"KNN3"))

knn5_yhat <- knn(training_df_without_label, test_df_without_label, training_df_label, k = 5)
capacity_to_generalize <- rbind(capacity_to_generalize, evaluate_algo(test_df_label, knn5_yhat,
"KNN5"))
```

Models With Bagging

Now to try bagging the Naive Bayes, and KNN models. I will use the models trained on the bootstrap samples to predict the test data and collect metrics on the capacity to learn and generalize, which will be aggregated.

```
the_bags <- list()
for (i in 1:n_bags){
  bag_df <- get_bootstrap_sample(training_df)
  bag_df_without_label <- bag_df %>% select(-label)
  bag_df_label <- bag_df$label
  the_bags[[i]] <- bag_df

  # Train the Naive Bayes Model
  bag_nb_model <- naiveBayes(label ~ ., data = bag_df)
  training_bag_nb_yhat <- predict(bag_nb_model, bag_df_without_label)
  ## Evaluate the capacity to learn
  bag_capacity_to_learn <- evaluate_algo(bag_df_label, training_bag_nb_yhat, paste("NB Bag", i))
  if(exists("nb_bag_capacity_to_learn")){
    nb_bag_capacity_to_learn <- rbind(nb_bag_capacity_to_learn, bag_capacity_to_learn)
  } else {
    nb_bag_capacity_to_learn <- bag_capacity_to_learn
  }
  ## Evaluate the capacity to generalize
  bag_nb_test_yhat <- predict(bag_nb_model, test_df_without_label)
  bag_capacity_to_generalize <- evaluate_algo(test_df_label, bag_nb_test_yhat, paste("NB Bag",
 i))
  if(exists("nb_bag_capacity_to_generalize")){
    nb_bag_capacity_to_generalize <- rbind(nb_bag_capacity_to_generalize, bag_capacity_to_genera
lize)
  } else {
    nb_bag_capacity_to_generalize <- bag_capacity_to_generalize
  }

  # Train the KNN3 Model
  ## Evaluate the capacity to learn
  training_bag_knn3_yhat <- knn(bag_df_without_label, bag_df_without_label, bag_df_label, k = 3)
  bag_capacity_to_learn <- evaluate_algo(bag_df_label, training_bag_knn3_yhat, paste("KNN3 Bag",
i))
  if(exists("knn3_bag_capacity_to_learn")){
    knn3_bag_capacity_to_learn <- rbind(knn3_bag_capacity_to_learn, bag_capacity_to_learn)
  } else {
    knn3_bag_capacity_to_learn <- bag_capacity_to_learn
  }
  ## Evaluate the capacity to generalize
  bag_knn3_test_yhat <- knn(bag_df_without_label, test_df_without_label, bag_df_label, k = 3)
  bag_capacity_to_generalize <- evaluate_algo(test_df_label, bag_knn3_test_yhat, paste("KNN3 wit
h Bagging", i))
  if(exists("knn3_bag_capacity_to_generalize")){
    knn3_bag_capacity_to_generalize <- rbind(knn3_bag_capacity_to_generalize, bag_capacity_to_ge
neralize)
  } else {
    knn3_bag_capacity_to_generalize <- bag_capacity_to_generalize
  }

  # Train the KNN5 Model
  ## Evaluate the capacity to learn
  training_bag_knn5_yhat <- knn(bag_df_without_label, bag_df_without_label, bag_df_label, k = 5)
  bag_capacity_to_learn <- evaluate_algo(bag_df_label, training_bag_knn5_yhat, paste("KNN5 with
```

```
 Bagging", i))
  if(exists("knn5_bag_capacity_to_learn")){
    knn5_bag_capacity_to_learn <- rbind(knn5_bag_capacity_to_learn, bag_capacity_to_learn)
  } else {
    knn5_bag_capacity_to_learn <- bag_capacity_to_learn
  }
  ## Evaluate the capacity to generalize
  bag_knn5_test_yhat <- knn(bag_df_without_label, test_df_without_label, bag_df_label, k = 5)
  bag_capacity_to_generalize <- evaluate_algo(test_df_label, bag_knn5_test_yhat, paste("KNN5 wit
h Bagging", i))
  if(exists("knn5_bag_capacity_to_generalize")){
    knn5_bag_capacity_to_generalize <- rbind(knn5_bag_capacity_to_generalize, bag_capacity_to_ge
neralize)
  } else {
    knn5_bag_capacity_to_generalize <- bag_capacity_to_generalize
  }

}
```

Now to average the metrics:

```
# Naive Bayes
capacity_to_learn <- rbind(capacity_to_learn, average_metrics(nb_bag_capacity_to_learn, "NB with
Bagging"))
capacity_to_generalize <- rbind(capacity_to_generalize, average_metrics(nb_bag_capacity_to_gener
alize, "NB with Bagging"))

# KNN3
capacity_to_learn <- rbind(capacity_to_learn, average_metrics(knn3_bag_capacity_to_learn, "KNN3
 with Bagging"))
capacity_to_generalize <- rbind(capacity_to_generalize, average_metrics(knn3_bag_capacity_to_gen
eralize, "KNN3 with Bagging"))

# KNN5
capacity_to_learn <- rbind(capacity_to_learn, average_metrics(knn5_bag_capacity_to_learn, "KNN5
 with Bagging"))
capacity_to_generalize <- rbind(capacity_to_generalize, average_metrics(knn5_bag_capacity_to_gen
eralize, "KNN5 with Bagging"))
```

Models With LOOCV

Now to examine how the models preform using LOOCV

```
training_loocv_nb_yhat <- c()
training_loocv_knn3_yhat <- c()
training_loocv_knn5_yhat <- c()
loocv_test_label <- c()
loocv_dfs <- list()

for (i in 1:nrow(training_df)){
  # Leave One Out
  loocv_test <- training_df[i,]
  loocv_test_without_label <- loocv_test %>% select(-label)
  loocv_test_label <- c(loocv_test_label, loocv_test$label)
  loocv_training_df <- training_df[-c(i),]
  loocv_training_df_without_label <- loocv_training_df %>% select(-label)
  loocv_training_df_label <- loocv_training_df$label
  loocv_dfs[[i]] <- loocv_training_df

  # Train the Naive Bayes Model
  loocv_nb_model <- naiveBayes(label ~ ., data = loocv_training_df)
  ## Evaluate the capacity to learn
  ### Note: This will be done once we have gone over the whole training data set
  training_loocv_nb_yhat <- c(training_loocv_nb_yhat, predict(loocv_nb_model, loocv_test_without
_label))
  ## Evaluate the capacity to generalize
  loocv_nb_test_yhat <- predict(loocv_nb_model, test_df_without_label)
  loocv_capacity_to_generalize <- evaluate_algo(test_df_label, loocv_nb_test_yhat, paste("NB wit
h LOOCV", i))
  if(exists("nb_loocv_capacity_to_generalize")){
    nb_loocv_capacity_to_generalize <- rbind(nb_loocv_capacity_to_generalize, loocv_capacity_to_
generalize)
  } else {
    nb_loocv_capacity_to_generalize <- loocv_capacity_to_generalize
  }

  # Train the KNN3 Model
  ## Evaluate the capacity to learn
  ### Note: This will be done once we have gone over the whole training data set
  training_loocv_knn3_yhat <- c(training_loocv_knn3_yhat, knn(loocv_training_df_without_label, l
oocv_test_without_label, loocv_training_df_label, k = 3))
  ## Evaluate the capacity to generalize
  loocv_knn3_test_yhat <- knn(loocv_training_df_without_label, test_df_without_label, loocv_trai
ning_df_label, k = 3)
  loocv_capacity_to_generalize <- evaluate_algo(test_df_label, loocv_knn3_test_yhat, paste("KNN3
with LOOCV", i))
  if(exists("knn3_loocv_capacity_to_generalize")){
    knn3_loocv_capacity_to_generalize <- rbind(knn3_loocv_capacity_to_generalize, loocv_capacity
_to_generalize)
  } else {
    knn3_loocv_capacity_to_generalize <- loocv_capacity_to_generalize
  }

  # Train the KNN5 Model
  ## Evaluate the capacity to learn
  ### Note: This will be done once we have gone over the whole training data set
```

```
  training_loocv_knn5_yhat <- c(training_loocv_knn5_yhat, knn(loocv_training_df_without_label, l
oocv_test_without_label, loocv_training_df_label, k = 5))
  ## Evaluate the capacity to generalize
  loocv_knn5_test_yhat <- knn(loocv_training_df_without_label, test_df_without_label, loocv_trai
ning_df_label, k = 5)
  loocv_capacity_to_generalize <- evaluate_algo(test_df_label, loocv_knn5_test_yhat, paste("KNN5
with LOOCV", i))
  if(exists("knn5_loocv_capacity_to_generalize")){
    knn5_loocv_capacity_to_generalize <- rbind(knn5_loocv_capacity_to_generalize, loocv_capacity
_to_generalize)
  } else {
    knn5_loocv_capacity_to_generalize <- loocv_capacity_to_generalize
  }

}
```

Now to average the metrics:

```
# Naive Bayes
capacity_to_learn <- rbind(capacity_to_learn, evaluate_algo(loocv_test_label, training_loocv_nb_
yhat, "NB with LOOCV"))
capacity_to_generalize <- rbind(capacity_to_generalize, average_metrics(nb_loocv_capacity_to_gen
eralize, "NB with LOOCV"))

# KNN3
capacity_to_learn <- rbind(capacity_to_learn, evaluate_algo(loocv_test_label, training_loocv_knn
3_yhat, "KNN3 with LOOCV"))
capacity_to_generalize <- rbind(capacity_to_generalize, average_metrics(knn3_loocv_capacity_to_g
eneralize, "KNN3 with LOOCV"))

# KNN5
capacity_to_learn <- rbind(capacity_to_learn, evaluate_algo(loocv_test_label, training_loocv_knn
5_yhat, "KNN5 with LOOCV"))
capacity_to_generalize <- rbind(capacity_to_generalize, average_metrics(knn5_loocv_capacity_to_g
eneralize, "KNN5 with LOOCV"))
```

Summary and Analysis

We have evaluated 9 algorithm's capacity to learn and generalize. The following tables summarize their performance:

Table 1. Capacity to Learn

Algo	AUC	ACCURACY	TPR	FPR	TNR	FNR
KNN3	0.7777778	0.8000000	0.6666667	0.2000000	0.8000000	0.3333333
KNN3 with Bagging	0.7427832	0.8061111	0.6495349	0.1958171	0.8041829	0.3504651
KNN3 with LOOCV	0.5972222	0.6333333	0.7368421	0.4444444	0.5555556	0.2631579
KNN5	0.6805556	0.7333333	0.7727273	0.1666667	0.8333333	0.2272727

Algo	AUC	ACCURACY	TPR	FPR	TNR	FNR
KNN5 with Bagging	0.6856171	0.7661111	0.6917903	0.1956817	0.8043183	0.3082097
KNN5 with LOOCV	0.6388889	0.7000000	0.8095238	0.2000000	0.8000000	0.1904762
LR	NA	NA	NA	NA	NA	NA
LR with Bagging	NA	NA	NA	NA	NA	NA
LR with LOOCV	NA	NA	NA	NA	NA	NA
NB	0.6944444	0.7333333	0.7272727	0.2500000	0.7500000	0.2727273
NB with Bagging	0.7186606	0.6944444	0.5581284	0.2589672	0.7410328	0.4418716
NB with LOOCV	0.6388889	0.6666667	0.7000000	0.4000000	0.6000000	0.3000000

Table 2. Capacity to Generalize

Algo	AUC	ACCURACY	TPR	FPR	TNR	FNR
KNN3	0.5000000	0.5000000	0.6666667	0.6666667	0.3333333	0.3333333
KNN3 with Bagging	0.5675000	0.5583333	0.5846801	0.5259470	0.4740530	0.4153199
KNN3 with LOOCV	0.4291667	0.4055556	0.5477778	0.6794444	0.3205556	0.4522222
KNN5	0.7500000	0.6666667	0.5000000	0.5000000	0.5000000	0.5000000
KNN5 with Bagging	0.6175000	0.6433333	0.6931667	0.4435417	0.5564583	0.3068333
KNN5 with LOOCV	0.7458333	0.7222222	0.6233333	0.4166667	0.5833333	0.3766667
LR	NA	NA	NA	NA	NA	NA
LR with Bagging	NA	NA	NA	NA	NA	NA
LR with LOOCV	NA	NA	NA	NA	NA	NA
NB	0.3750000	0.3333333	0.5000000	0.7500000	0.2500000	0.5000000
NB with Bagging	0.4812500	0.4366667	0.4762626	0.6281787	0.3718213	0.5237374
NB with LOOCV	0.3916667	0.3500000	0.4933333	0.7222222	0.2777778	0.5066667

Discussion

There are 30 rows in the training set. When doing LOOCV the model is trained on 29 rows of data. This subsample is going to be very similar to the full training set. The bagging on the other hand will be significantly different. There are 18 rows of data and since it is sampled with replacement a same observation could be

selected multiple times. Consequently the underlying data is different. We can see this by looking at the share of records that are labeled “BLACK.”

```
get_share_black <- function(df){
  df %>% filter(label == "BLACK") %>% nrow() / nrow(df)
}
```

The share of the training data set that is labeled black is 0.6. Let’s compare this to the shares in the bags.

```
share_black <- c()
for(i in 1:n_bags){
  share_black <- c(share_black, get_share_black(the_bags[[i]]))
}

summary(share_black)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.2778  0.5000  0.5556  0.5972  0.7222  0.8889 
```

You can see that the share of observations that are black range from 28% to 89%. This is very different than what we observe with the LOOCV data set:

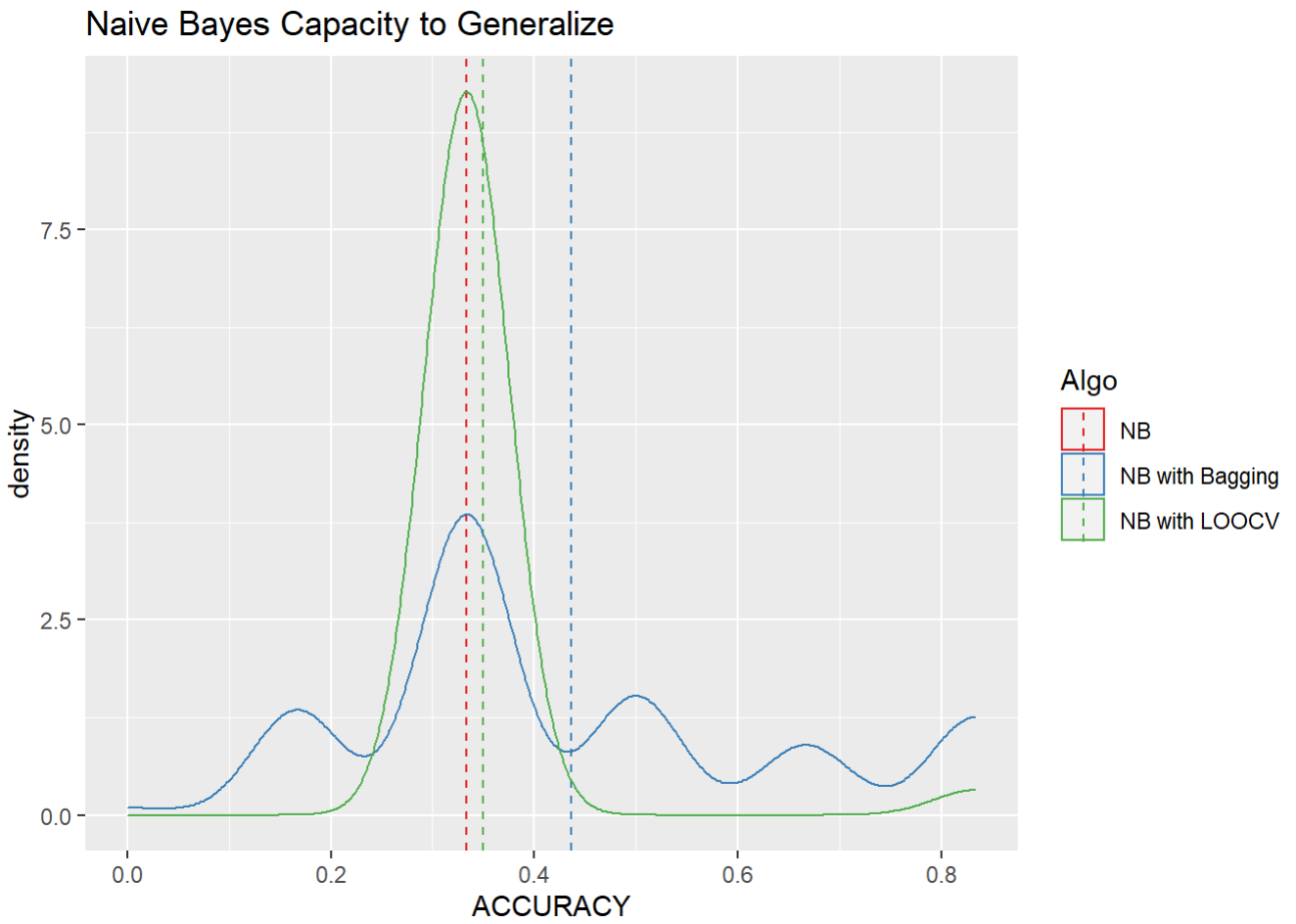
```
share_black <- c()
for(i in 1:nrow(training_df)){
  share_black <- c(share_black, get_share_black(loocv_dfs[[i]]))
}

summary(share_black)
```

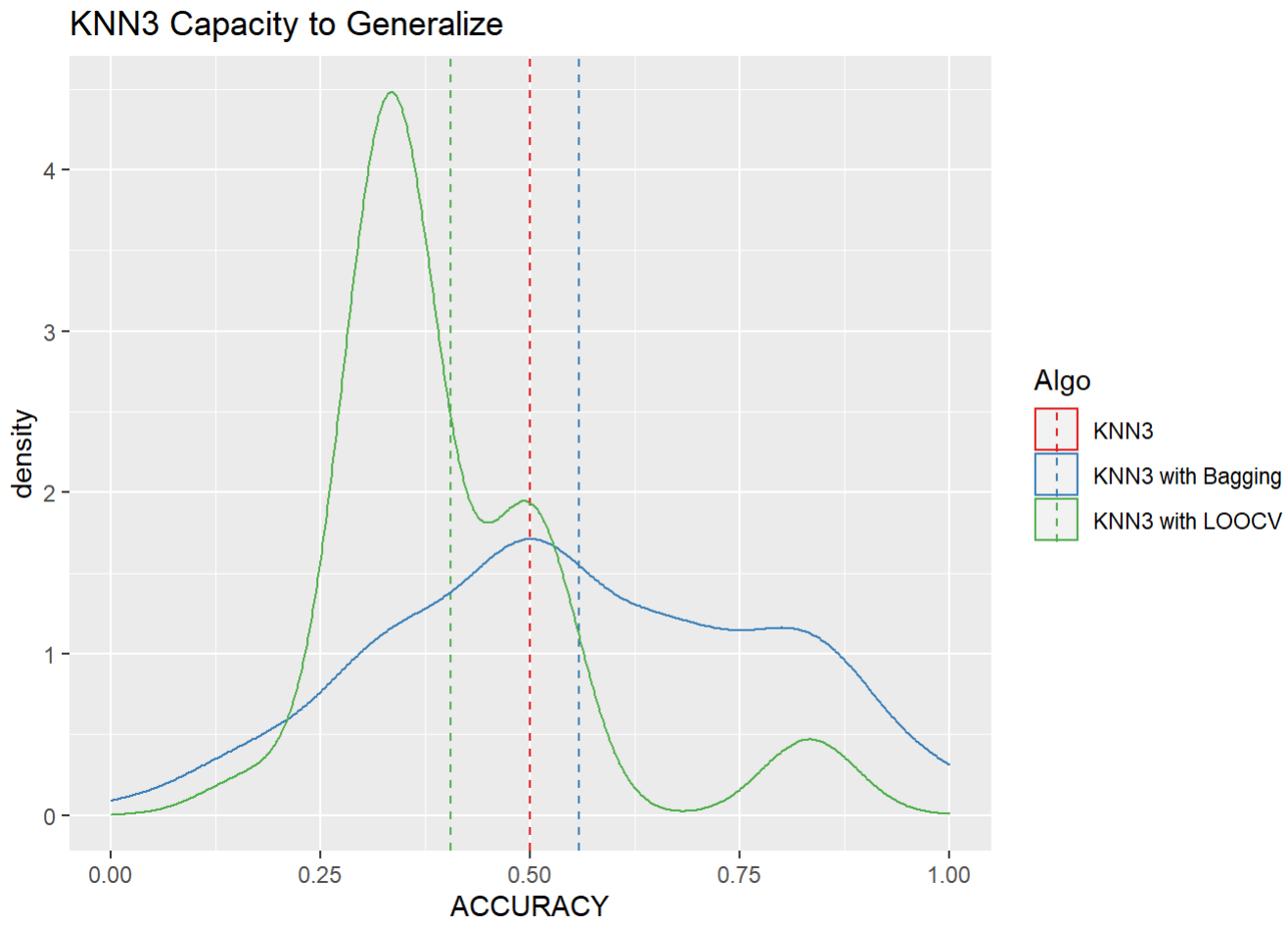
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5862  0.5862  0.5862  0.6000  0.6207  0.6207 
```

This is a lot closer to the full training set. Consequently a model trained on the bootstrap sample is likely to have different results if it is a weak learner, than the full training set. Some of the models will do a better job at generalizing than others due to the composition of the data. This can be seen by looking at a metric like the accuracy of the model. The following images represent the mean accuracy on the test dataset as a dashed vertical line. For the bagging and LOOCV runs the distribution of the accuracy is also shown:

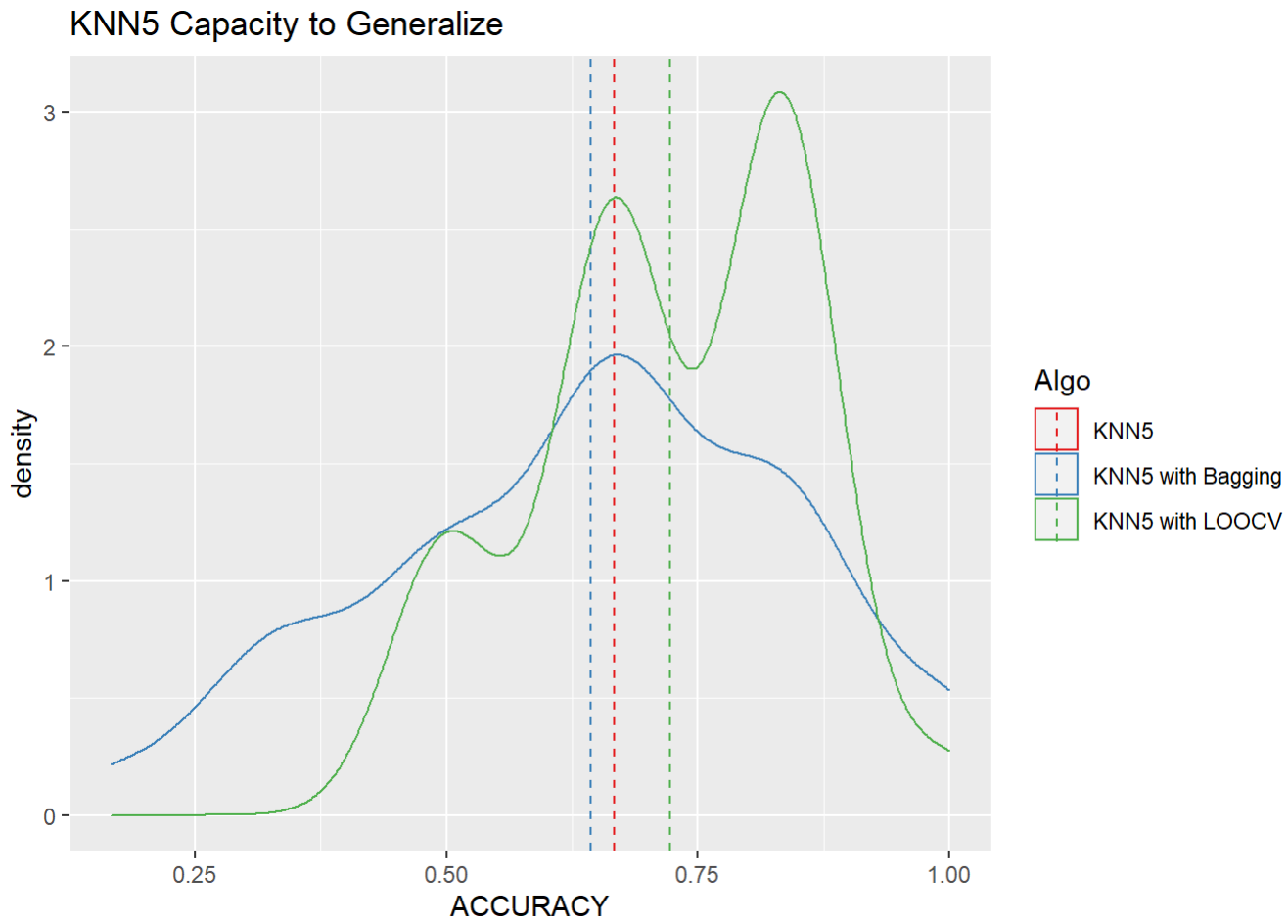
```
vline_df <- capacity_to_generalize %>% filter(str_detect(Algo, "NB"))
ggplot() + geom_vline(data=vline_df, aes(xintercept=ACCURACY, color=Algo), linetype="dashed") +
 geom_density(aes(x=ACCURACY, color="NB with Bagging"), data=nb_bag_capacity_to_generalize) + ge
om_density(aes(x=ACCURACY, color="NB with LOOCV"), data=nb_loocv_capacity_to_generalize) + ggtit
le("Naive Bayes Capacity to Generalize") + scale_color_brewer(palette = "Set1")
```




```
vline_df <- capacity_to_generalize %>% filter(str_detect(Algo, "KNN3"))
ggplot() + geom_vline(data=vline_df, aes(xintercept=ACCURACY, color=Algo), linetype="dashed") +
 geom_density(aes(x=ACCURACY, color="KNN3 with Bagging"), data=knn3_bag_capacity_to_generalize)
 + geom_density(aes(x=ACCURACY, color="KNN3 with LOOCV"), data=knn3_loocv_capacity_to_generaliz
e) + ggtitle("KNN3 Capacity to Generalize") + scale_color_brewer(palette = "Set1")
```




```
vline_df <- capacity_to_generalize %>% filter(str_detect(Algo, "KNN5"))
ggplot() + geom_vline(data=vline_df, aes(xintercept=ACCURACY, color=Algo), linetype="dashed") +
 geom_density(aes(x=ACCURACY, color="KNN5 with Bagging"), data=knn5_bag_capacity_to_generalize)
 + geom_density(aes(x=ACCURACY, color="KNN5 with LOOCV"), data=knn5_loocv_capacity_to_generaliz
e) + ggtitle("KNN5 Capacity to Generalize") + scale_color_brewer(palette = "Set1")
```



Now I hasten to note that all of these figures hare very specific to the initial split in the data which is controlled by the random number seed. So the statements I will make at this point are specific to this run. If I would have used another random number seed the results could be vastly different.

When I completed HW-1 I noted the the KNN family of models preformed the best. KNN3 with bagging out preformed KNN3 on the whole training data. As previously discussed it is due to the variation in the bootstrap samples. When it comes to the ability to generalize KNN5 with LOOCV out preformed KNN5 and KNN3 with bagging out preformed KNN3. The accuracy increases above the “baseline” algorithm. NB improved the ability to generalize through both bagging and LOOCV. This suggests that the performance of these weak learners is highly dependent on the data, but when averaged together the group outperforms the single model.