# Introduction to Ensemble Methods

## Comment on Model Selection Methods

For problem set 5, you read the article *The garden of forking paths: Why multiple comparisons can be a problem, even when there is no "fishing expedition" or "p-hacking" and the research hypothesis was posited ahead of time*, by Gelman and Loken.

That article outlined the following challenges:

- $p$-values for hypothesis tests are only valid if the analysis was specified before looking at the data.
- $p$-values for hypothesis tests are only valid if the model is correctly specified
- In most practical data analysis tasks, it is not feasible to correctly specify the model without looking at the data (we need to get the right transformations, the right interactions, etc - and for any problem of reasonable complexity, it's impossible to do that without looking at plots).

The article suggested two primary strategies for moving forward:

1. Pre-publication replications: An experiment is conducted twice.
   - The results from the first experiment will be used for model exploration and selection.
   - The results from the second experiment will then be analyzed with the selected model from the first stage analysis.
   - The $p-$-values from the second analysis will be more reliable.
2. Presenting results from a wide variety of models that *could have been* analyzed. We will be looking to see whether there is consensus in the inferences obtained from these different models.

**Important point for today:** Analyzing and presenting results from multiple different models is valuable in an inference setting.

## Ensembles: Combine predictions/classifications from multiple different models

Theme for the next few weeks: often, combining results from multiple different models is also valuable in a prediction or classification setting.

### Motivating Example

This example is inspired by https://mlwave.com/kaggle-ensembling-guide/

Setting: Classification with 2 classes ($Y_i = 0$ or 1)

Suppose I have three different classifiers which make the correct prediction for test set observations with probability 0.7, indepdendently.

Let's consider the behavior of the **majority vote** classifier, which makes the prediction that is made by at least 2 out of the 3 classifiers. What is the probability that the prediction from the majority vote classifier is correct?

Define the following events:

- $A$ is the event that the first classifier is correct. P(A) = 0.7
- $B$ is the event that the second classifier is correct. P(B) = 0.7
- $C$ is the event that the third classifier is correct. P(C) = 0.7

$$
\begin{aligned}
&P(\text{At least two classifiers are correct}) = P(\text{Exactly two classifiers are correct}) + P(\text{Three classifiers are correct}) \\
&\quad = P(A \text{ and } B \text{ and } C^c) + P(A \text{ and } B^c \text{ and } C) + P(A^c \text{ and } B \text{ and } C) + P(A \text{ and } B \text{ and } C) \\
&\quad = 0.7 * 0.7 * 0.3 + 0.7 * 0.3 * 0.7 + 0.3 * 0.7 * 0.7 + 0.7 * 0.7 * 0.7 \\
&\quad = 0.784
\end{aligned}
$$

So, the majority vote classifier does better than any individual classifier.

**Caveat**: Predictions from good classifiers are usually correlated, so the above calculations are an extreme case.

What would happen if the three classifiers all made identical predictions (total lack of independence)?

- Either all three classifiers would be correct, or all three classifiers would be incorrect.
- Majority vote classifier would have exactly the same performance as any individual model.

**Conclusion: Ensembles work best if they're combining predictions from uncorrelated models.**

## Real Example: Ionosphere radar data

This example is adapted from a discussion at https://burakhimmetoglu.com/2016/12/01/stacking-models-for-improved-predictions/

Our data set for today is published at https://archive.ics.uci.edu/ml/datasets/ionosphere. Here's a description from that website:

> This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. See the paper for more details. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.
>
> Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this databse are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.
>
> Attribute Information:
>
> - All 34 are continuous
> - The 35th attribute is either "good" or "bad" according to the definition summarized above. This is a binary classification task.

```
library(readr)
library(dplyr)
library(ggplot2)
library(gridExtra)
library(purrr)
library(glmnet)
library(caret)

# read in data
ionosphere <- read_csv("http://www.evanlray.com/data/UCIML/ionosphere/ionosphere.data", col_names = FALSE)

# X2 was all 0's
ionosphere <- ionosphere %>% select(-X2)

# Convert prediction target to a factor
ionosphere$X35 <- factor(ionosphere$X35)


## Initial train/test split ("estimation"/test) and cross-validation folds
set.seed(63770)
tt_inds <- caret::createDataPartition(ionosphere$X35, p = 0.8)
ionosphere_train <- ionosphere %>% slice(tt_inds[[1]])
ionosphere_test <- ionosphere %>% slice(-tt_inds[[1]])

crossval_val_fold_inds <- caret::createFolds(
  y = ionosphere_train$X35, # response variable as a vector
  k = 10 # number of folds for cross-validation
)
```

```
get_complementary_inds <- function(x) {
  return(seq_len(nrow(ionosphere_train))[-x])
}
crossval_train_fold_inds <- map(crossval_val_fold_inds, get_complementary_inds)


## Function to calculate error rate
calc_error_rate <- function(observed, predicted) {
  mean(observed != predicted)
}
```

### Individual Methods

### Logistic Regression

```
logistic_fit <- train(
  form = X35 ~ .,
  data = ionosphere_train,
  family = "binomial", # this is an argument to glm
  method = "glm", # method for fit
  trControl = trainControl(method = "cv", # evaluate method performance via cross-validation
    number = 10, # number of folds for cross-validation
    index = crossval_train_fold_inds, # I'm specifying which folds to use, for consistency across methods
    indexOut = crossval_val_fold_inds, # I'm specifying which folds to use, for consistency across methods
    returnResamp = "all", # return information from cross-validation
    savePredictions = TRUE, # return validation set predictions from cross-validation
    classProbs = TRUE) # return validation set predicted class probabilities from cross-validation
)

logistic_fit$results
```

```
##   parameter  Accuracy     Kappa AccuracySD  KappaSD
## 1      none 0.8722906 0.7171562 0.07416491 0.166649
```

```
head(logistic_fit$pred)
```

```
##   pred obs            b            g rowIndex parameter Resample
## 1    g   g 0.0005368858 9.994631e-01       18      none   Fold01
## 2    g   g 0.0001349143 9.998651e-01       22      none   Fold01
## 3    b   b 0.9842897951 1.571020e-02       23      none   Fold01
## 4    g   g 0.0021638496 9.978362e-01       28      none   Fold01
## 5    b   b 1.0000000000 2.220446e-16       42      none   Fold01
## 6    g   g 0.0014000061 9.986000e-01       43      none   Fold01
```

```
logistic_fit$pred %>%
  group_by(Resample) %>%
  summarize(accuracy = mean(pred == obs)) %>%
  ungroup() %>%
  summarize(accuracy = mean(accuracy))
```

```
## # A tibble: 1 x 1
##   accuracy
##      <dbl>
## 1    0.872
```

### KNN

```
knn_fit <- train(
  form = X35 ~ .,
```

```
  data = ionosphere_train,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "cv",
    number = 10,
    index = crossval_train_fold_inds, # I'm specifying which folds to use, for consistency across methods
    indexOut = crossval_val_fold_inds, # I'm specifying which folds to use, for consistency across methods
    returnResamp = "all",
    savePredictions = TRUE,
    classProbs = TRUE),
  tuneGrid = data.frame(k = 1:20)
)

knn_fit$results
```

```
##     k  Accuracy     Kappa AccuracySD   KappaSD
## 1   1 0.8435961 0.6312602 0.05555387 0.1416257
## 2   2 0.8330049 0.6048754 0.04617159 0.1140735
## 3   3 0.8115764 0.5436372 0.05498370 0.1417965
## 4   4 0.7971675 0.5012242 0.04764045 0.1287739
## 5   5 0.7901478 0.4836172 0.06346634 0.1727569
## 6   6 0.7758621 0.4402555 0.07124431 0.1974507
## 7   7 0.7757389 0.4418288 0.05617063 0.1554953
## 8   8 0.7757389 0.4362940 0.05863946 0.1722108
## 9   9 0.7613300 0.3962108 0.04566366 0.1411818
## 10 10 0.7507389 0.3680442 0.04525739 0.1433910
## 11 11 0.7541872 0.3724594 0.06021259 0.1834820
## 12 12 0.7577586 0.3807042 0.06547342 0.1907690
## 13 13 0.7578818 0.3803979 0.06066707 0.1848983
## 14 14 0.7507389 0.3565872 0.06350085 0.1892079
## 15 15 0.7543103 0.3672147 0.06445188 0.1920880
## 16 16 0.7471675 0.3401043 0.06670248 0.2066319
## 17 17 0.7435961 0.3310751 0.05848508 0.1849466
## 18 18 0.7364532 0.3079044 0.06187939 0.1990674
## 19 19 0.7364532 0.3085801 0.05954504 0.1899028
## 20 20 0.7328818 0.2967558 0.06204108 0.1979344
```

**Trees**

```
rpart_fit <- train(
  form = X35 ~ .,
  data = ionosphere_train,
  method = "rpart",
  trControl = trainControl(method = "cv",
    number = 10,
    index = crossval_train_fold_inds, # I'm specifying which folds to use, for consistency across methods
    indexOut = crossval_val_fold_inds, # I'm specifying which folds to use, for consistency across methods
    returnResamp = "all",
    savePredictions = TRUE,
    classProbs = TRUE),
  tuneLength = 10
)

rpart_fit$results
```

```
##            cp  Accuracy     Kappa AccuracySD   KappaSD
## 1  0.00000000 0.8541872 0.6687683 0.04551359 0.1060588
## 2  0.05610561 0.8932266 0.7673936 0.04763338 0.1017781
## 3  0.11221122 0.8932266 0.7673936 0.04763338 0.1017781
```

```
## 4  0.16831683 0.8932266 0.7673936 0.04763338 0.1017781
## 5  0.22442244 0.8259852 0.5956364 0.08523190 0.2135394
## 6  0.28052805 0.7866995 0.4879269 0.05450905 0.1320280
## 7  0.33663366 0.7866995 0.4879269 0.05450905 0.1320280
## 8  0.39273927 0.7866995 0.4879269 0.05450905 0.1320280
## 9  0.44884488 0.7866995 0.4879269 0.05450905 0.1320280
## 10 0.50495050 0.7652709 0.4220733 0.06487394 0.1892912
```

**Test set predictions from each of the 3 methods above:**

```
logistic_preds <- predict(logistic_fit, newdata = ionosphere_test)
calc_error_rate(ionosphere_test$X35, logistic_preds)
```

```
## [1] 0.1
```

```
knn_preds <- predict(knn_fit, newdata = ionosphere_test)
calc_error_rate(ionosphere_test$X35, knn_preds)
```

```
## [1] 0.1
```

```
rpart_preds <- predict(rpart_fit, newdata = ionosphere_test)
calc_error_rate(ionosphere_test$X35, rpart_preds)
```

```
## [1] 0.08571429
```

**Ensemble Methods**

**Majority Vote**

```
majority_vote_preds <- ifelse(
  (logistic_preds == "g") + (knn_preds == "g") + (rpart_preds == "g") >= 2,
  "g",
  "b"
)

calc_error_rate(ionosphere_test$X35, majority_vote_preds)
```

```
## [1] 0.07142857
```

**Mean of Class Probabilities from Individual Methods**

```
logistic_class_probs <- predict(logistic_fit, newdata = ionosphere_test, type = "prob")
logistic_prob_g <- logistic_class_probs[, 2]

knn_class_probs <- predict(knn_fit, newdata = ionosphere_test, type = "prob")
knn_prob_g <- knn_class_probs[, 2]

rpart_class_probs <- predict(rpart_fit, newdata = ionosphere_test, type = "prob")
rpart_prob_g <- rpart_class_probs[, 2]

mean_prob_g <- (logistic_prob_g + knn_prob_g + rpart_prob_g) / 3
mean_prob_preds <- ifelse(
  mean_prob_g >= 0.5,
  "g",
  "b"
)
calc_error_rate(ionosphere_test$X35, mean_prob_preds)
```

```
## [1] 0.07142857
```

**Stacking: Fit a model to combine predicted class membership probabilities**

- Some methods might be better than others; we should give them more weight.
- We can use training set performance to determine how much to weight them.
- We must cross-validate: otherwise, we'd give too much weight to models that overfit the training data

**Process:**

Estimation:

1. Get cross-validated predictions for each "stage 1" or "component" model (this was done above)
2. Create a new data set where the explanatory variables are the cross-validated predictions from the component models
3. Fit a "stage 2" model to predict the response based on the component model predictions

Prediction for test set:

4. Re-fit each component model to the full training data set, make predictions for the test set from each component model (this was done above)
5. Create a new data set where the explanatory variables are the test set predictions from the component models
6. Predict using the stage 2 model fit from step 3 and the data frame created in step 5.

```
# Step 2: data set with component model predictions as explanatory variables
ionosphere_train <- ionosphere_train %>%
  mutate(
    logistic_prob_g = logistic_fit$pred %>%
      arrange(rowIndex) %>%
      pull(g),
    knn_prob_g = knn_fit$pred %>%
      filter(k == 1) %>%
      arrange(rowIndex) %>%
      pull(g),
    rpart_prob_g = rpart_fit$pred %>%
      filter(abs(cp - 0) < 10^(-5)) %>%
      arrange(rowIndex) %>%
      pull(g)
  )

# Step 3: fit model using component model predictions as explanatory variables
stacking_logistic_fit <- train(
  form = X35 ~ logistic_prob_g + knn_prob_g + rpart_prob_g,
  data = ionosphere_train,
  family = "binomial",
  method = "glm"
)

# Step 5: Assemble data frame of test set predictions from each component model
stacking_test_x <- data.frame(
  logistic_prob_g = logistic_prob_g,
  knn_prob_g = knn_prob_g,
  rpart_prob_g = rpart_prob_g
)

# Step 6: Stacked model predictions
stacking_preds <- predict(stacking_logistic_fit, stacking_test_x)

# Calculate error rate
calc_error_rate(ionosphere_test$X35, stacking_preds)
```

```
## [1] 0.04285714
```

### Stacking via KNN

- We could also use other methods for the second stage model.

```
ionosphere_train <- ionosphere_train %>%
  mutate(
    logistic_prob_g = logistic_fit$pred %>%
      arrange(rowIndex) %>%
      pull(g),
    knn_prob_g = knn_fit$pred %>%
      filter(k == 1) %>%
      arrange(rowIndex) %>%
      pull(g),
    rpart_prob_g = rpart_fit$pred %>%
      filter(abs(cp - 0) < 10^(-5)) %>%
      arrange(rowIndex) %>%
      pull(g)
  )

stacking_knn_fit <- train(
  form = X35 ~ logistic_prob_g + knn_prob_g + rpart_prob_g,
  data = ionosphere_train,
  method = "knn"
)

# Assemble data frame of test set predictions from each
# component model (these were obtained in the previous part)
stacking_test_x <- data.frame(
  logistic_prob_g = logistic_prob_g,
  knn_prob_g = knn_prob_g,
  rpart_prob_g = rpart_prob_g
)

# Stacked model predictions
stacking_preds <- predict(stacking_knn_fit, stacking_test_x)
calc_error_rate(ionosphere_test$X35, stacking_preds)
```

```
## [1] 0.04285714
```

### Notes about relative performance of these methods

- The results with the seed I have set make the stacking approaches look amazing - but in different runs I did as I was developing this, relative performance of the ensemble approaches here varied.
- In general, stacking is the best of these ensemble approaches in terms of expected value of performance, if the methods' performance is not all equal and we have enough training set data.
- Note that in this example we had only 3 stage 1/component models. In general, ensembles are most useful when we have:
  - A large number of models that are "diverse"/uncorrelated/predictions are close to independent
  - Enough training data available to reliably tell which component models are best and how to combine their predictions effectively.