Lecture 10:
Cross-Validation for Classification
STAT 452, Spring 2021

- We can also use cross-validation to evaluate the predictive performance of classification models.

- The predictive performance of a classifier can be summarized using the so-called **confusion matrix**, which is a contingency table that compares the actual and predicted values of the categorical response variable on the test set data.

- Using the confusion matrix we can compute classification metrics such as the accuracy, sensitivity, and specificity.

Example: 2016 Election Data

- To illustrate cross-validation for logistic regression we will use a data set that contains results for US counties for the 2016 presidential election.
- The response variable is `trump_win`, a binary variable that is 1 if Trump won the county, and 0 otherwise.
- The predictor variable we will consider for this demonstration is `obama_pctvotes`, the percent of votes cast for Obama in each county in 2012. The data set also contains other demographic predictor variables obtained from the US Census.

```
# read in data from URL
> county_votes <- readRDS(url("https://ericwfox.github.io/data/county_votes16.rds"))

> head(county_votes)
  state          county clinton_pctvotes trump_pctvotes obama_pctvotes pct_pop65
1    AL Autauga County            23.96          73.44          26.58      13.8
2    AL Baldwin County            19.57          77.35          21.57      18.7
3    AL Barbour County            46.66          52.27          51.25      16.5
4    AL    Bibb County            21.42          76.97          26.22      14.8
5    AL  Blount County             8.47          89.85          12.35      17.0
6    AL Bullock County            75.09          24.23          76.31      14.9
  pct_black pct_white pct_hispanic pct_asian highschool bachelors income trump_win
1      18.7      77.9          2.7       1.1       85.6      20.9 53.682         1
2       9.6      87.1          4.6       0.9       89.1      27.7 50.221         1
3      47.6      50.2          4.5       0.5       73.7      13.4 32.911         1
4      22.1      76.3          2.1       0.2       77.5      12.1 36.447         1
5       1.8      96.0          8.7       0.3       77.0      12.1 44.145         1
6      70.1      26.9          7.5       0.3       67.8      12.5 32.033         0

> dim(county_votes)
[1] 3112   14

> table(county_votes$trump_win)
   0    1
 488 2624

> table(county_votes$trump_win) / nrow(county_votes)
        0         1
0.1568123 0.8431877
```

Variable Descriptions:

- `trump_win`: indicator variable (1=Trump won, 0=Trump lost)
- `obama_pctvotes`: percent of votes cast for Obama in 2012
- `pct_pop65`: percent over 65 years
- `pct_black`: percent black
- `pct_white`: percent white
- `pct_hispanic`: percent hispanic
- `pct_asian`: percent asian
- `highschool`: percent high school graduate or higher
- `bachelors`: percent with Bachelor's degree or higher
- `income`: per capita income in the past 12 months (in thousands of dollars)

```
# randomly split data into a 70% training and 30% test set
> set.seed(999)
> n <- nrow(county_votes); n
[1] 3112
> round(0.7*n)
[1] 2178

> train_index <- sample(1:n, 2178)
> county_votes_train <- county_votes[train_index, ]
> county_votes_test <- county_votes[-train_index, ]
```

```
# fit model using training data
> glm1 <- glm(trump_win ~ obama_pctvotes, family = "binomial",
              data=county_votes_train)

> summary(glm1)
Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    19.64711    1.20927   16.25   <2e-16 ***
obama_pctvotes -0.36312    0.02289  -15.87   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

We can use a 0.5 probability threshold to classify points in the test set. If the predicted probability is greater than 0.5, classify as a Trump win (1). Otherwise, if the predicted probability is less than 0.5, classify as a Trump loss (0).

```
# make predictions for probabilities on test set
> probs1 <- predict(glm1, newdata = county_votes_test,
                    type = "response")

# use 0.5 probably threshold to make classifications
> preds1 <- ifelse(probs1 > 0.5, 1, 0)
> head(data.frame(probs1, preds1), n=15)
        probs1 preds1
1  0.999954392      1
8  0.999433422      1
9  0.928154844      1
12 0.916188372      1
16 0.999975665      1
18 0.779429928      1
20 0.999995040      1
22 0.999999405      1
23 0.999909081      1
24 0.003424933      0
27 0.998055339      1
29 0.999988482      1
31 0.999997962      1
33 0.043763807      0
45 0.994136952      1
```

Next make the so-called **confusion matrix**. The confusion matrix tabulates the predicted versus actual results on the test set. There are 934 counties in the test set. So we see that the model correctly predicted Trump losing in 125 counties, and correctly predicted Trump winning in 763 counties. There are also some misclassifications: the model predicted Trump losing in 22 counties that he actually won, and the model predicted Trump winning in 24 counties that he actually lost.

```
# make confusion matrix
> tb <- table(prediction = preds1,
              actual = county_votes_test$trump_win)

> addmargins(tb)
          actual
prediction   0   1 Sum
       0   125  22 147
       1    24 763 787
       Sum 149 785 934
```

We can use the confusion matrix to compute various classification metrics (accuracy, sensitivity, and specificity) on the test set.

```
> tb <- table(prediction = preds1,
              actual = county_votes_test$trump_win)
> addmargins(tb)
          actual
prediction   0   1 Sum
       0   125  22 147
       1    24 763 787
       Sum 149 785 934

# Accuracy (percent correctly classified)
> (125 + 763) / 934
[1] 0.9507495

# Sensitivity (percent of Trump wins (1) correctly classified)
> 763 / 785
[1] 0.9719745

# Specificity (percent of Trump losses (0) correctly classified)
> 125 / 149
[1] 0.8389262
```

Null Model

- The **null model** classifies every observation to the majority class.
- For the example, the null model would predict that Trump wins every county.

  ```
  > table(county_votes_test$trump_win)
    0   1
  149 785
  ```

  On the test set this gives the following results:

  - Accuracy = percent correctly classified = 785 / (785 + 149) = 0.84
  - Sensitivity = percent of Trump wins (1) correctly classified = 1
  - Specificity = percent of Trump losses (0) correctly classified = 0

- The logistic regression model, with `obama_pctvotes` as a predictor, performs substantially better than the null model.

Null Model

- It is always important to compare the performance of a classification model against a null model that just predicts the majority class.
- For example, if we have a highly imbalanced response variable that is 98% 1's and 2% 0's, then a model which predicts all 1's would yield 98% accuracy!

Your Turn

(a) Randomly split the `county_votes` data frame into a 70% training and 30% test set.

(b) Fit a logistic regression model with `bachelors` (percent of county's population with Bachelor's degree or higher) as a predictor using the training data.

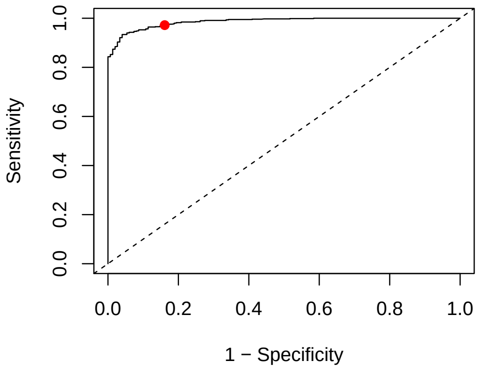
(c) Make a confusion matrix and calculate the accuracy, sensitivity, and specificity on the test set.

ROC Curve

- One issue with the classification metrics we have just considered is that they depended on using a 0.5 probability threshold.
- One popular graphical method to evaluate how well the model does at classifying, regardless of the choice of threshold, is the **receiver operating characteristic (ROC)** curve. The name comes from communication theory.
- The ROC curve is a plot of `sensitivity` versus `1-specificity` for a variety of probability thresholds between 0 and 1.
- In the context of the election data, the `sensitivity`, or true positive rate, is the percent of Trump wins (1) correctly classified. Whereas, `1-specificity`, or false positive rate, is the percent of Trump losses (0) that are misclassified as Trump wins (1).

ROC Curve

ROC curve for the logistic regression model for predicting Trump winning counties in the test set data. Note that the red point corresponds to the 0.5 probability threshold ($1 - \texttt{specificity} = 24/149 = 0.161$, and $\texttt{sensitivity} = 763/785 = 0.972$).



ROC Curve

- The point (0,1) in the top-left corner represents perfect classification. That is, it corresponds to correctly classifying all the response data in the test set.
- In general, the closer the ROC curve is to the top-left corner, or (0,1) point, the better the model performs at classification.
- On the other hand, a model that makes random classifications would have an ROC curve that lies close to the diagonal, or 1-1 line.
- For the election data, the ROC curve is close to the top-left corner, and thus the logistic regression model appears to be performing well at classification.

ROC Curve

I used the `pROC` package to plot the ROC curve (there are other R packages one could use). Here's the code:

```
> library(pROC)
> roc_obj <- roc(county_votes_test$trump_win, probs1)
> plot(1 - roc_obj$specificities, roc_obj$sensitivities, type="l",
       xlab = "1 - Specificity", ylab = "Sensitivity")

> # plot red point corresponding to 0.5 threshold
> points(x = 24/149, y = 763/785, col="red", pch=19)
> abline(0, 1, lty=2) # 1-1 line
```

Note that the second argument of `roc()` are the predicted **probabilities** on the test set.

AUC

- Another popular classification metric is the **area under the ROC curve (AUC)**.
- The closer the AUC is to 1 the better the model performs at classification. While a model that performs no better than random guessing would have an AUC close to 0.5.
- One reason the AUC is such a popular metric is because it does not depend on a probability threshold.
- For the election data, the logistic regression model has an AUC that is close to 1. We can compute the AUC using the `pROC` package:

```
> auc(roc_obj)
Area under the curve: 0.9863
```

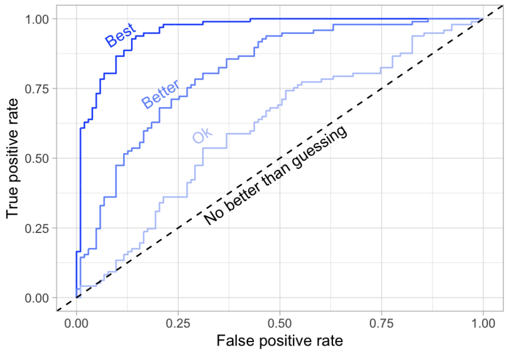


Figure: ROC Curve

From Chapter 2, Section 2.7, of *Hand-On Machine Learning*