06 - Classification

ml4econ, HUJI 2023

Itamar Caspi

May 14, 2023 (updated: 2023-05-14)

Managing Packages

Use the {pacman} package for *effortless loading and installation* of necessary packages. This happens automatically if they are not already in your system.

```
if (!require("pacman")) install.packages("pacman")

pacman::p_load(
  tidyverse,   # for data wrangling and visualization
  tidymodels,  # for modeling
  knitr,       # for displaying nice tables
  here,        # for referencing folders and files
  glmnet,      # for estimating lasso and ridge
  ggmosaic     # for tidy mosaic plots
)
```

Setting Up Themes and Seed

To maintain a consistent style for `ggplot` throughout the presentation, we set a theme as follows:

```
theme_set(theme_grey(20))
```

To ensure reproducibility of results, we also set a seed:

```
set.seed(1203)
```

Outline

- Binary Classification Problems
- The Confusion Matrix
- The Logistic Regression Model
- Sensitivity Specificity Trade-off
- Multiclass classification

Binary Classification Problems

Bill Gates on Testing for COVID-19

"Basically, there are two critical cases: anyone who is symptomatic, and anyone who has been in contact with someone who tested positive. Ideally both groups would be sent a test they can do at home without going into a medical center. Tests would still be available in medical centers, but the simplest is to have the majority done at home. **To make this work, a government would have to have a website that you go to and enter your circumstances, including your symptoms. You would get a priority ranking, and all of the test providers would be required to make sure they are providing quick results to the highest priority levels.** Depending on how accurately symptoms predict infections, how many people test positive, and how many contacts a person typically has, you can figure out how much capacity is needed to handle these critical cases. For now, most countries will use all of their testing capacity for these cases." - Bill Gates.

Source: "The first modern pandemic by Bill Gates"

Binary Classification

Let y_i represent the outcome of a COVID-19 test, defined as:

$$y_i = \begin{cases} 1 & \text{if the test result is positive,} \\ 0 & \text{if the test result is negative.} \end{cases}$$

Here, the values 1 and 0 are chosen for their simplicity.[1]

There are two types of questions that we might ask given this context:

1. What is the probability of a test returning a positive result?
2. Can we classify an individual's test result as positive or negative?

[*] It's common to encounter a $\{1, -1\}$ notation for binary outcomes in machine learning literature.

Israeli COVID-19 Test Data

The Israeli Ministry of Health provides data on more than 9 million COVID-19 test results. Our objective in this context is to predict whether a person will be classified as "positive", i.e., infected by the virus, based on their symptoms and characteristics.

Outcome variable: `corona_result`

Features include:

- Symptoms: `cough`, `fever`, `sore_throat`, `shortness_of_breath`, and `head_ache`
- Characteristics: `age_60_and_above`, `gender`

Read and examine the data

```
covid_raw <- here("06-classification/data","covid_proc.csv") %>%
  read_csv()
```

```
covid_raw %>% glimpse()
```

```
## Rows: 107,542
## Columns: 8
## $ cough               <dbl> 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0,~
## $ fever               <dbl> 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,~
## $ sore_throat         <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ shortness_of_breath <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,~
## $ head_ache           <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ corona_result       <chr> "negative", "negative", "negative", "positive", "negativ~
## $ age_60_and_above    <chr> "No", "No", "Yes", "Yes", "Yes", "Yes", "No", "No", "No"~
## $ gender              <chr> "male", "male", "male", "male", "female", "male", "male"~
```

Given that the total number of observations n is 107,542 and the number of predictor variables p is 7, we don't need to be overly concerned about overfitting.

Preprocessing

Now, we will categorize all variables, including the outcome and features, as factors:

```
covid <- covid_raw %>%
  mutate_all(as_factor)
```

Additionally, we will extract the outcome and features as matrices for future use with the `glmnet` function:

```
x <- covid %>%
  select(-corona_result) %>%
  model.matrix(~ .-1, data = .)

y <- covid %>% pull(corona_result) %>% as_factor()
```

Raw Detection Frequencies

How are the test results distributed?

```
covid %>%
  group_by(corona_result) %>%
  count()
```

```
## # A tibble: 2 x 2
## # Groups:   corona_result [2]
##   corona_result     n
##   <fct>         <int>
## 1 negative      98586
## 2 positive       8956
```

This distribution demonstrates an instance of **class imbalance**. This occurs when the distribution of examples across the known classes is skewed, a common feature in classification problems.

Measuring Classification Accuracy

What does MSE (Mean Squared Error) imply in the context of classification problems?

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{\{y_i \neq \hat{y}_i\}}$$

To put it in words: In this context, MSE measures the **misclassification rate**, which is the ratio of the number of misclassifications to the total number of observations.

Classification accuracy is the ratio of the total number of correct predictions to the total number of predictions made for a dataset.

Evidently,

$$\mathbf{accuracy} = 1 - \mathbf{misclassification\ rate.}$$

Is the misclassification rate or accuracy rate useful? Consider the context of imbalanced outcomes.

A Naive Classifier

Our naive "model" strategy is to "classify everyone as being negative".

```
covid %>%
  mutate(corona_result = as_factor(corona_result)) %>%
  mutate(.fitted_class = factor("negative", levels = c("negative", "positive"))) %>%
  conf_mat(corona_result, .fitted_class)
```

```
##           Truth
## Prediction negative positive
##   negative    98586     8956
##   positive        0        0
```

The accuracy of this model is $98,586/107,542 = 91.67$!

This seems quite impressive! Or is it?

Despite its high accuracy, this naive classifier lacks the ability to distinguish between classes, and more crucially, it fails to identify infected individuals - the aspect we truly care about!

The Confusion Matrix

Beyond Accuracy – Other Measures of Performance

The **confusion matrix** is a table that categorizes predictions based on whether they align with the actual outcomes.

		Actual	**Actual**
		Negative	Positive
Predicted	Negative	*True Negative* (TN)	*False Negative* (FN)
Predicted	Positive	*False Positive* (FP)	*True Positive* (TP)

The total number of observations N is the sum of all these categories, i.e., $TP + TN + FP + FN = N$. Accuracy in this context is defined as $(TN + TP)/N$.

Note: The confusion matrix can be extended to include multiclass outcomes.

Types of Classification Errors

False Positive Rate: This is the proportion of negative examples that are incorrectly classified as positive. In our example, this is $0/98,586 = 0\%$.

False Negative Rate: This is the proportion of positive examples that are incorrectly classified as negative. In our example, this is $8,956/8,956 = 100\%$.

The question then arises: Can we improve upon these rates?

A Perfect Classifier

Consider a simple example. Let's say we have a sample of 100 test results, with exactly 20 of them labeled as "positive". If our classifier was perfect, the confusion matrix would look as follows:

		Actual	**Actual**
		Negative	Positive
Predicted	Negative	80	0
Predicted	Positive	0	20

In this scenario, our classifier achieves 100% accuracy with zero false positives and zero false negatives.

The Realistic Classifier

Now, let's consider a classifier that makes some errors:

		Actual	Actual
		Negative	Positive
Predicted	Negative	70	10
Predicted	Positive	5	15

In this example, 10 individuals with the pathogen were incorrectly classified as Negative (not infected), and 5 individuals without the pathogen were incorrectly classified as Positive (infected).

Logistic Regession Model

First Things First: The Linear Probability Model

Consider a dependent variable $y_i \in \{0, 1\}$. Given a vector of features $\mathbf{x}_i$, the aim is to predict $\Pr(y_i = 1|\mathbf{x}_i)$.

Let p_i denote the probability of observing $y_i = 1$ given $\mathbf{x}_i$, i.e.,

$$p_i \equiv \Pr(y_i = 1|\mathbf{x}_i)$$

The linear probability model specifies that

$$p_i = \mathbf{x}_i' \boldsymbol{\beta}$$

However, an OLS (Ordinary Least Squares) regression of y_i on $\mathbf{x}_i$ overlooks the discrete nature of the dependent variable and does not limit the predicted probabilities to the range between zero and one.

Logistic Regression Model

A more appropriate model for this case is the **logit model** or **logistic regression model**, which is specified as follows:

$$p = \Lambda(\mathbf{x}'\boldsymbol{\beta}) = \frac{\exp(\mathbf{x}'\boldsymbol{\beta})}{1 + \exp(\mathbf{x}'\boldsymbol{\beta})}$$

Here, $\boldsymbol{\Lambda}(\cdot)$ represents the logistic cumulative distribution function (CDF). Consequently, the model enforces the restriction that $\mathbf{0} \leq p_i \leq \mathbf{1}$.

Odds-Ratio

Note that

$$\frac{p}{1-p} = \exp\left(\mathbf{x}'\boldsymbol{\beta}\right)$$

Taking the natural logarithm of both sides yields

$$\ln\left(\frac{p}{1-p}\right) = \mathbf{x}'\boldsymbol{\beta}$$

This equation provides a useful representation of the logistic regression model. The left-hand side (LHS) is known as the log **odds ratio** (or relative risk).

Therefore, we can say that the logistic regression model is linear in terms of the log odds-ratio.

The Likelihood Function

Likelihood refers to the probability of observing the given data for certain parameter values.

$$\begin{aligned}\text{Likelihood} &= \prod_{i=1}^{n} \Pr(y_i|\mathbf{x}_i) \\ &= \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} \\ &= \prod_{i=1}^{n} \left(\frac{\exp\left(\mathbf{x}_i'\beta\right)}{1+\exp\left(\mathbf{x}_i'\beta\right)}\right)^{y_i} \left(\frac{1}{1+\exp\left(\mathbf{x}_i'\beta\right)}\right)^{1-y_i}\end{aligned}$$

Taking the natural logarithm of both sides results in the **log likelihood**:

$$\log(\text{Likelihood}) = \sum_{i=1}^{N} \left[\log\left(1+e^{(\beta_0+x_i'\beta)}\right) - y_i \cdot \left(\beta_0 + x_i'\beta\right)\right]$$

During estimation we aim to maximize this value, hence maximum likelihood estimation (MLE).

Deviance

Another useful concept is the **deviance**, which generalizes the concept of "least squares" to encompass general linear models like the logit model.

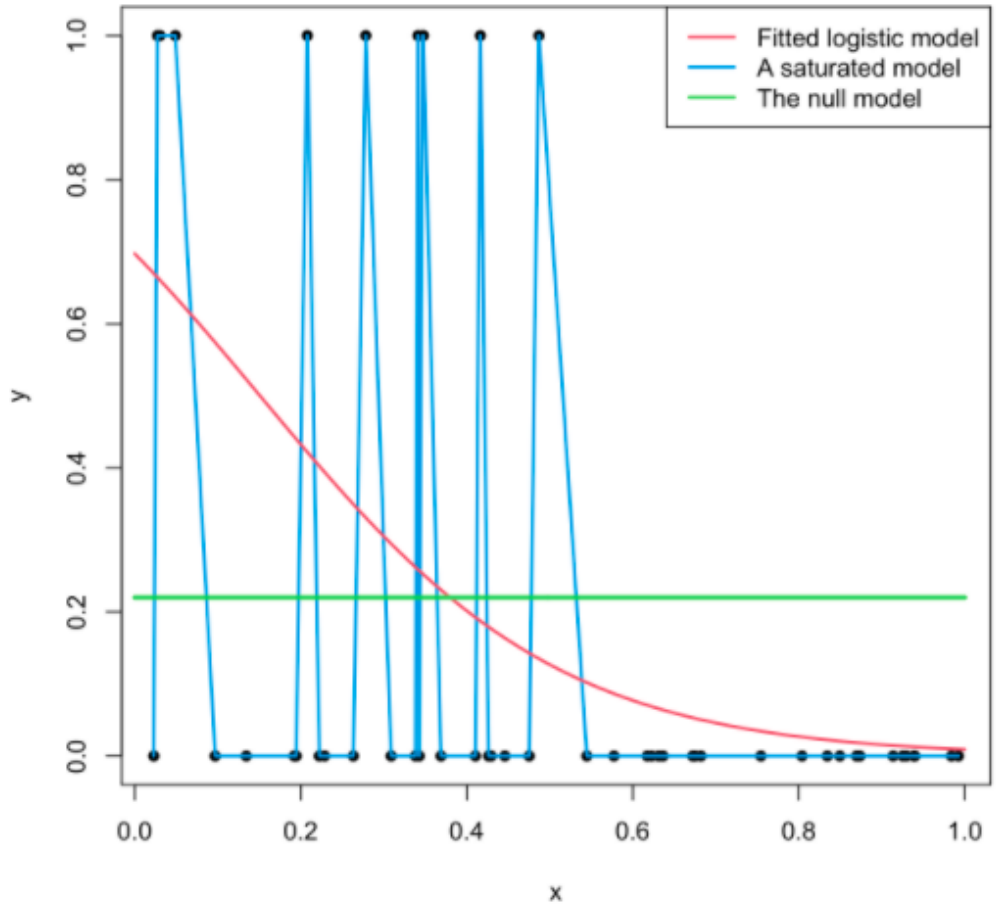
It serves as a measure of the discrepancy between the observed data and the fitted model.

The relationship between deviance and likelihood is given by:

$$\text{Deviance} = -2 \times \log(\text{Likelihood}) + \text{Constant}$$

The constant term includes terms related to the likelihood of a "perfect" model and is usually not of major concern and can be disregarded for practical purposes.

Illustration of the Deviance



Source

Deviance and Estimation

In the estimation process, our goal is to minimize the deviance.

$$\begin{aligned}\text{Deviance} &= -2\sum_{i=1}^{N}\left[\log\left(1+e^{(\beta_0+x_i'\beta)}\right)-y_i\cdot(\beta_0+x_i'\beta)\right]+\text{Constant}\\ &\propto \sum_{i=1}^{N}\left[\log\left(1+e^{(\beta_0+x_i'\beta)}\right)-y_i\cdot(\beta_0+x_i'\beta)\right]\end{aligned}$$

This is what R's `glm` function minimizes when performing logistic regressions.

NOTE: In linear models, the deviance is proportional to the Residual Sum of Squares (RSS).

Penalized Logistic Regression

Alternatively, we can minimize the deviance with an ℓ_1 norm penalty on β, known as a standard lasso-type penalty:

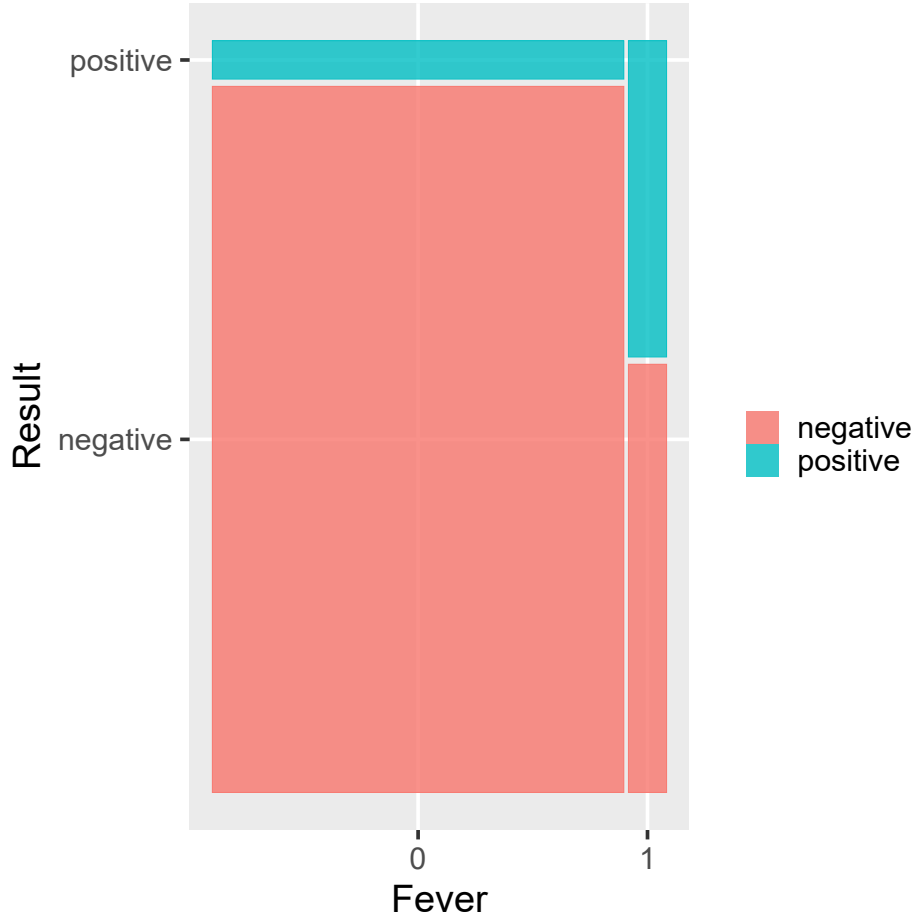
$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} \left[\frac{1}{N} \sum_{i=1}^{N} \log\left(1 + e^{(\beta_0 + x_i'\beta)}\right) - y_i \cdot \left(\beta_0 + x_i'\beta\right) \right] + \lambda \|\beta\|_1$$

Here, the penalty is applied to the sum of the absolute values of β (excluding the intercept).

Back to the Data: Can We Do Better Than Being "Naive"?

There is some evidence suggesting that having a fever is associated with testing positive.

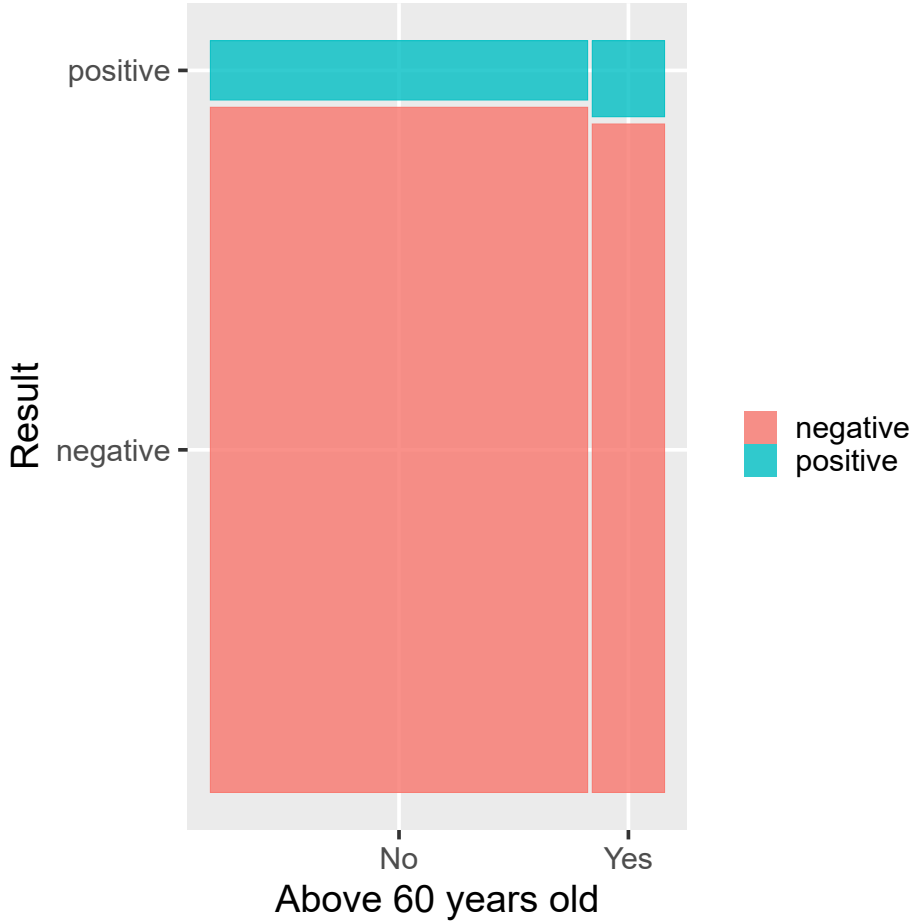
```
covid %>%
  ggplot() +
  geom_mosaic(
    aes(x = product(corona_result, fever),
        fill = corona_result)
  ) +
  labs(
    x = "Fever",
    y = "Result",
    fill = ""
  )
```



Back to the data: can we do better than being "naive"?

and some evidence for an association with age (above 60)

```
covid %>%
  ggplot() +
  geom_mosaic(
    aes(x = product(corona_result, age_60_and_above)
        fill = corona_result)
  ) +
  labs(
    x = "Above 60 years old",
    y = "Result",
    fill = ""
  )
```



Estimating the Model using R

To estimate the model, we will use base R's `glm` function, which stands for generalized linear model:

```
logit_model <- glm(
  corona_result ~ .,
  data = covid,
  family = "binomial"
)
```

Alternatively, we can estimate the regularized version of the model using `glmnet` with `family = "binomial"`:

```
logit_model_lasso <- cv.glmnet(x, y, family = "binomial")
```

SPOILER ALERT: `cv.glmnet` selects all features.

Model Output

To obtain a tidy summary of the output from `glm` objects, we can use the `tidy()` and `glance()` functions from the `{broom}` package:

```
logit_model %>% tidy()
```

```
## # A tibble: 8 x 5
##   term                 estimate std.error statistic   p.value
##   <chr>                   <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)            -3.23     0.0224    -144.  0
## 2 cough1                  0.656    0.0353      18.6 4.62e- 77
## 3 fever1                  1.92     0.0371      51.8 0
## 4 sore_throat1            4.38     0.119       36.7 2.00e-294
## 5 shortness_of_breath1    4.21     0.138       30.4 1.41e-203
## 6 head_ache1              5.35     0.139       38.6 0
## 7 age_60_and_aboveYes     0.399    0.0343      11.6 2.83e- 31
## 8 genderfemale           -0.308    0.0279     -11.0 2.34e- 28
```

```
logit_model %>% glance()
```

```
## # A tibble: 1 x 8
##   null.deviance df.null  logLik    AIC    BIC deviance df.residual   nobs
##           <dbl>   <int>   <dbl>  <dbl>  <dbl>    <dbl>       <int>  <int>
## 1        61666.  107541 -20726. 41468. 41544.   41452.      107534 107542
```

Generate Predictions

To generate predictions, we can use the `augment()` function from the `{broom}` package. It augments the original dataframe with fitted values and standard errors:

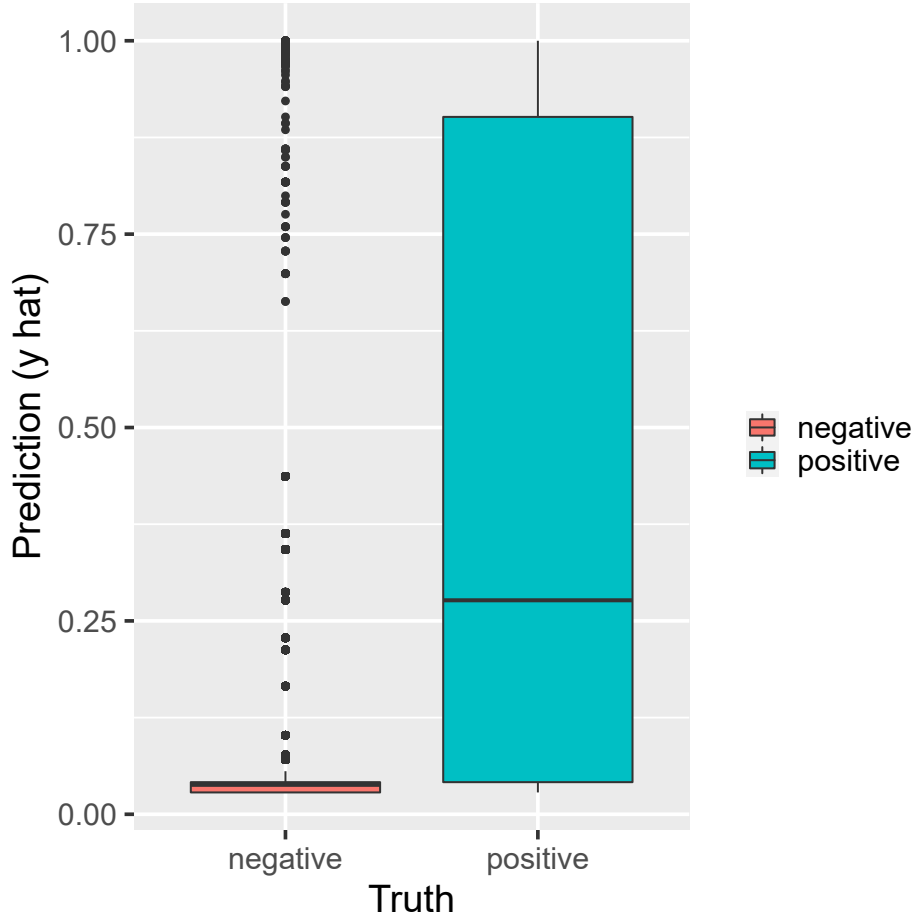
```
covid_pred <-
  logit_model %>%
  augment(type.predict = "response")

covid_pred
```

```
## # A tibble: 107,542 x 14
##    corona_~1 cough fever sore_~2 short~3 head_~4 age_6~5 gender .fitted .resid .std.~6
##    <fct>     <fct> <fct> <fct>   <fct>   <fct>   <fct>   <fct>    <dbl>  <dbl>   <dbl>
##  1 negative  1     0     0       0       0       No      male    0.0709 -0.383  -0.383
##  2 negative  1     1     0       0       0       No      male    0.342  -0.916  -0.916
##  3 negative  0     1     0       0       0       Yes     male    0.287  -0.822  -0.823
##  4 positive  1     1     0       0       0       Yes     male    0.437   1.29    1.29
##  5 negative  1     0     0       0       0       Yes     female  0.0770 -0.400  -0.400
##  6 positive  1     1     0       0       0       Yes     male    0.437   1.29    1.29
##  7 negative  0     0     0       0       0       No      male    0.0381 -0.279  -0.279
##  8 negative  1     1     1       0       1       No      female  1.00   -4.19   -4.19
##  9 negative  1     0     1       0       0       No      female  0.817  -1.84   -1.85
## 10 negative  1     1     1       1       0       No      female  1.00   -3.91   -3.91
## # ... with 107,532 more rows, 3 more variables: .hat <dbl>, .sigma <dbl>,
## #   .cooksd <dbl>, and abbreviated variable names 1: corona_result, 2: sore_throat,
## #   3: shortness_of_breath, 4: head_ache, 5: age_60_and_above, 6: .std.resid
```

Model Predictions (In Sample)

The figure on the right illustrates the resulting in-sample fit. It is evident that there is little overlap between the predicted probabilities for true positives and true negatives.

```
covid_pred %>%
  ggplot(aes(x = corona_result,
             y = .fitted,
             fill = corona_result)) +
  geom_boxplot() +
  labs(
    x = "Truth",
    y = "Prediction (y hat)",
    fill = ""
  )
```



Sensitivity Specifisity Trade-off

Classification Rule

To classify individuals as positive or negative, we need to establish a **classification rule** or cut-off. This rule involves setting a probability threshold p^* above which we classify an individual as positive.

For illustration purposes, we will set $p^* = 0.8$:

```
class_rule <- 0.8
```

This implies that whenever $\hat{y}_i > 0.8$, we classify individual i as positive.

QUESTION: Is this rule overly aggressive or passive?

Classification under the rule

```
covid_pred <- logit_model %>%
  augment(type.predict = "response") %>%
  mutate(
    .fitted_class = if_else(.fitted < class_rule, "negative", "positive"),
    .fitted_class = as_factor(.fitted_class)
  ) %>%
  select(corona_result, .fitted, .fitted_class)

covid_pred
```

```
## # A tibble: 107,542 x 3
##    corona_result .fitted .fitted_class
##    <fct>           <dbl> <fct>
##  1 negative       0.0709 negative
##  2 negative       0.342  negative
##  3 negative       0.287  negative
##  4 positive       0.437  negative
##  5 negative       0.0770 negative
##  6 positive       0.437  negative
##  7 negative       0.0381 negative
##  8 negative       1.00   positive
##  9 negative       0.817  positive
## 10 negative       1.00   positive
## # ... with 107,532 more rows
```

Sensitivity-Specificity Trade-off

As we have observed, classifying everyone as "negative" ($p^* = 1$) fails to achieve specificity, meaning it fails to correctly identify any positive results, which is what we truly care about!

Sensitivity: the fraction of positive examples that are correctly classified as positive ("true positive rate"). In this example, it is $98,586/98,586 = 100\%$.

Specificity: the fraction of negative examples that are correctly classified as negative ("true negative rate"). In this example, it is $0/8,956 = 0\%$.

Note that in general,

$$\text{false negative rate} = 1 - \text{specificity}$$

$$\text{false positive rate} = 1 - \text{sensitivity}$$

Our model's confusion matrix

The {yardstick} package provides the conf_mat() function, which offers convenient access to a model's confusion matrix and the associated performance statistics.

```
covid_conf_mat <-
  covid_pred %>%
  conf_mat(corona_result, .fitted_class)

covid_conf_mat
```

```
##            Truth
## Prediction negative positive
##   negative    98455     6179
##   positive      131     2777
```

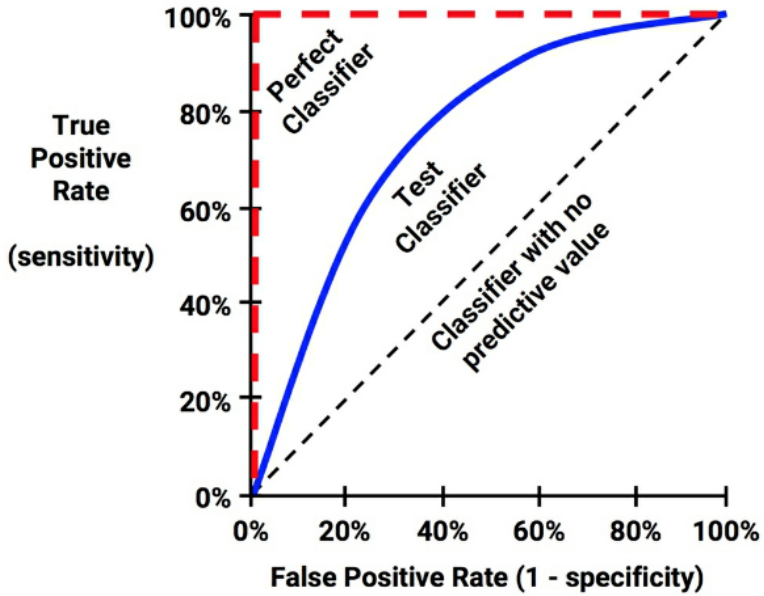
```
covid_conf_mat%>%
  summary() %>%
  filter(.metric %in% c("accuracy", "sens", "spec"))
  mutate("1-.estimate" = 1 - .estimate)
```

```
## # A tibble: 3 x 4
##   .metric  .estimator .estimate `1-.estimate`
##   <chr>    <chr>          <dbl>         <dbl>
## 1 accuracy binary         0.941       0.0587
## 2 sens     binary         0.999       0.00133
## 3 spec     binary         0.310       0.690
```

As we can see, when class_rule = 0.8, the model exhibits high sensitivity but relatively lower specificity. It is evident that modifying the classification rule would alter the model's classification properties.

Visualizing the Sens-Spec Trade-off with ROC Curves

A **Receiver Operating Characteristic (ROC)** curve plots sensitivity against 1-specificity. This graphical representation showcases the trade-off between false-positive and true-positive error rates as the classifier threshold is adjusted.



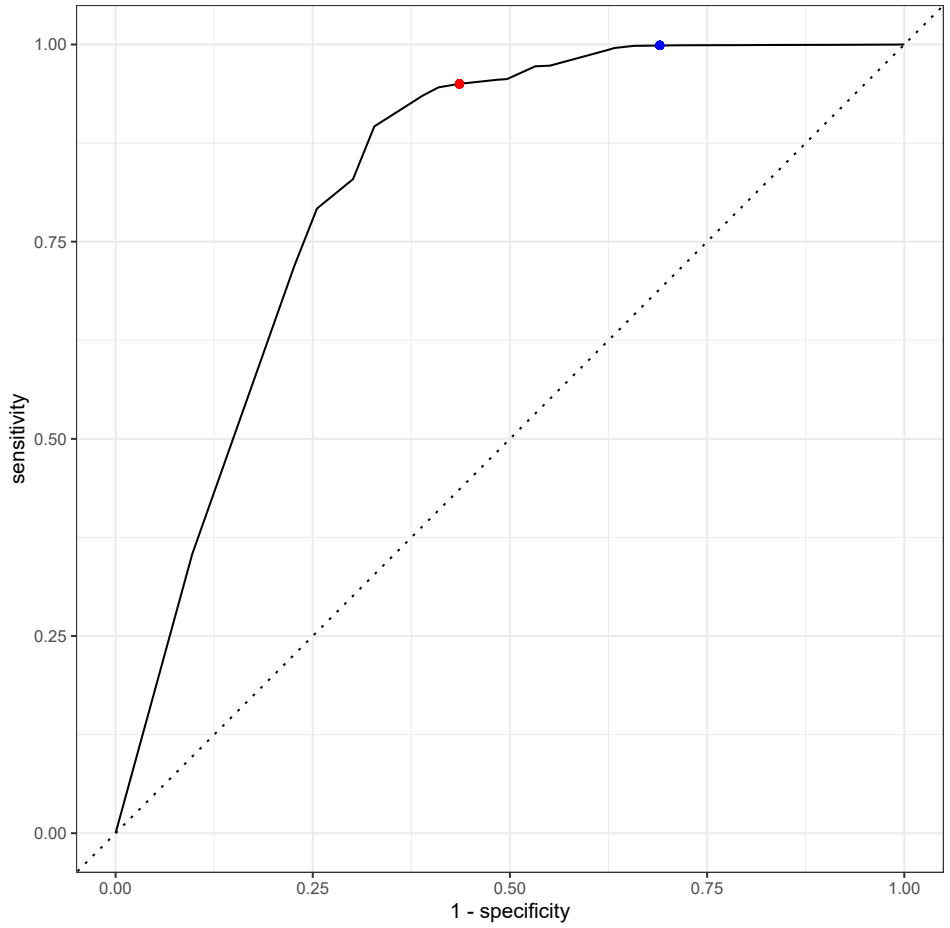
Source: "Machine Learning with R: Expert techniques for predictive modeling"

Our model's ROC curve

On the left side, we can observe the ROC curve of our model, generated using the `roc_curve()` function. The red and blue dots represent two different cut-offs, namely 0.8 and 0.2, respectively.

```
covid_pred %>%
  mutate(.fitted = 1-.fitted) %>%
  roc_curve(corona_result, .fitted) %>%
  autoplot() +
  geom_point(
    aes(x = 0.690, y = 0.999),
    color = "blue"
  ) + # 0.8 threshold
  geom_point(
    aes(x = 0.436, y = 0.950),
    color = "red"
  ) # 0.2 threshold
```

Note that we've used `.fitted` instead of `.fitted_class`.

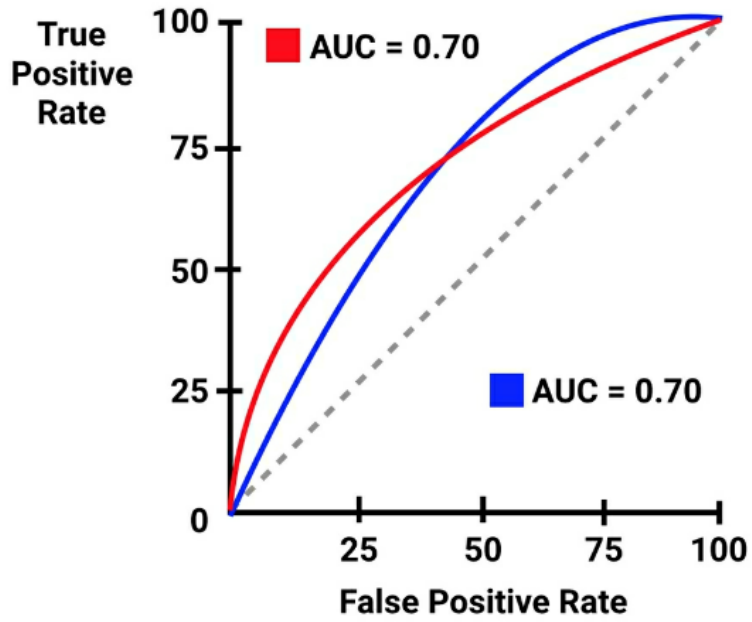


Area under the Curve (AUC)

- Classifiers can be ranked based on the area under the ROC curve (AUC).
- For instance, a perfect classifier has an auc of 1, while a classifier with no discriminative value has an auc of 0.5.
- However, it's important to note that two different ROC curves can yield identical auc values. Therefore, qualitative examination is warranted.

```
covid_pred %>% roc_auc(corona_result, .fitted)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.173
```



Source: "Machine Learning with R: Expert techniques for predictive modeling"

AUC and Cross-Validation

When dealing with classification tasks, it is often more reasonable to tune the penalty parameter based on classification performance metrics rather than, for example, deviance.

For instance, we can utilize the `cv.glmnet()` function and set `type.measure = "auc"` to perform tuning based on AUC values.

```
logit_model <- cv.glmnet(
  x, y,
  family = "binomial",
  type.measure = "auc"
)
```

or alternatively, you can set `type.measure = "class"` to perform tuning based on the misclassification rate.

Multiclass Classification

Multiclass Outcomes

- Each observation is assigned to one of $j = 1, \ldots, G$ classes or groups.
- The outcome variable is represented as

$$y = (y_1, \ldots, y_G)$$

  where $y_j = 1$ if the outcome belongs to the j^{th} class and zero otherwise.

- The conditional probability is defined as

$$p_j \equiv \Pr\big(y_j = 1 | \mathbf{x}\big), \qquad \text{for } j = 1, \ldots, G$$

  In other words, p_g represents the probability that y belongs to class g, given the input $\mathbf{x}_i$.

Multinomial Regression Model

For each class, the outcome is modeled as

$$p_j = \frac{\exp(\mathbf{x}'\boldsymbol{\beta}_j)}{\sum_{g=1}^{G} \exp(\mathbf{x}'\beta_g)}, \qquad \text{for } j = 1, \ldots, G$$

where $\sum_{j=1}^{G} p_j = 1$.

NOTE: In this case, there is no explicit base class since regularized solutions are not equivariant under base changes, and regularization automatically eliminates redundancy.

Likelihood and Deviance

Given the probabilities p_{ij} for $y_{ij} = 1$, the probability of the observed data is proportional to

$$\prod_{i=1}^{N} \prod_{j=1}^{G} p_{ij}^{y_{ij}}$$

where N represents the total number of observations.

By taking the logarithm and multiplying by -2, we obtain the multinomial deviance:

$$-2 \sum_{i=1}^{N} \sum_{j=1}^{G} y_{ij} \log(p_{ij})$$

Regularization

Let K represent the length of $\boldsymbol{\beta}$, i.e., the number of features in the model.

The coefficient matrix $\mathbf{B} = [\boldsymbol{\beta}_1 \cdots \boldsymbol{\beta}_G]$ contains $K \times G$ elements: G coefficients, one per class, multiplied by the number of features K.

Similar to the binomial case, we can minimize the deviance subject to a standard lasso-type (ℓ_1 norm) penalty on β:

$$\min_{\mathbf{B} \in \mathbb{R}^{K \times G}} \left\{ -\frac{2}{N} \sum_{i=1}^{N} \sum_{j=1}^{G} y_{ij} \log p_{ij} + \lambda \sum_{i=1}^{K-1} \sum_{j=1}^{G} |\beta_{ij}| \right\}$$

where $p_{ij} = \Lambda(\mathbf{x}_i' \beta_j)$, and the intercepts are unregularized.

Illustration: Forensic Glass Data

The forensic glass (`fgl`) dataset consists of 214 rows and 10 columns.

The dataset includes measurements for each of the 214 glass shards, such as the refractive index (`RI`) and eight measurements of chemical composition by weight of oxide (percentage) for elements `Na`, `Mg`, `Al`, `Si`, `K`, `Ca`, `Ba`, and `Fe`.

The glass shards were originally classified into seven types:

- `WinF`: Window float glass
- `WinNF`: Window non-float glass
- `Veh`: Vehicle window glass
- `Con`: Containers
- `Tabl`: Tableware
- `Head`: Vehicle headlamps

Our objective is to classify new data into one of the six types mentioned above.

Load and Inspect the `fgl` Data

The `fgl` dataset is included in the `{MASS}` library.

```
fgl_wide <-
  MASS::fgl %>%
  as_tibble()
```

```
head(fgl_wide)
```

```
## # A tibble: 6 x 10
##       RI    Na    Mg    Al    Si     K    Ca    Ba    Fe type
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
## 1  3.01   13.6  4.49  1.1   71.8  0.06  8.75     0  0    WinF
## 2 -0.390  13.9  3.6   1.36  72.7  0.48  7.83     0  0    WinF
## 3 -1.82   13.5  3.55  1.54  73.0  0.39  7.78     0  0    WinF
## 4 -0.340  13.2  3.69  1.29  72.6  0.57  8.22     0  0    WinF
## 5 -0.580  13.3  3.62  1.24  73.1  0.55  8.07     0  0    WinF
## 6 -2.04   12.8  3.61  1.62  73.0  0.64  8.07     0  0.26 WinF
```

Tidying the Data with `pivot_longer`

The code chunk below uses the **`pivot_longer()`** function from the **`{tidyr}`** package. This function *transforms the data* from wide to long format, preparing it for later plotting:

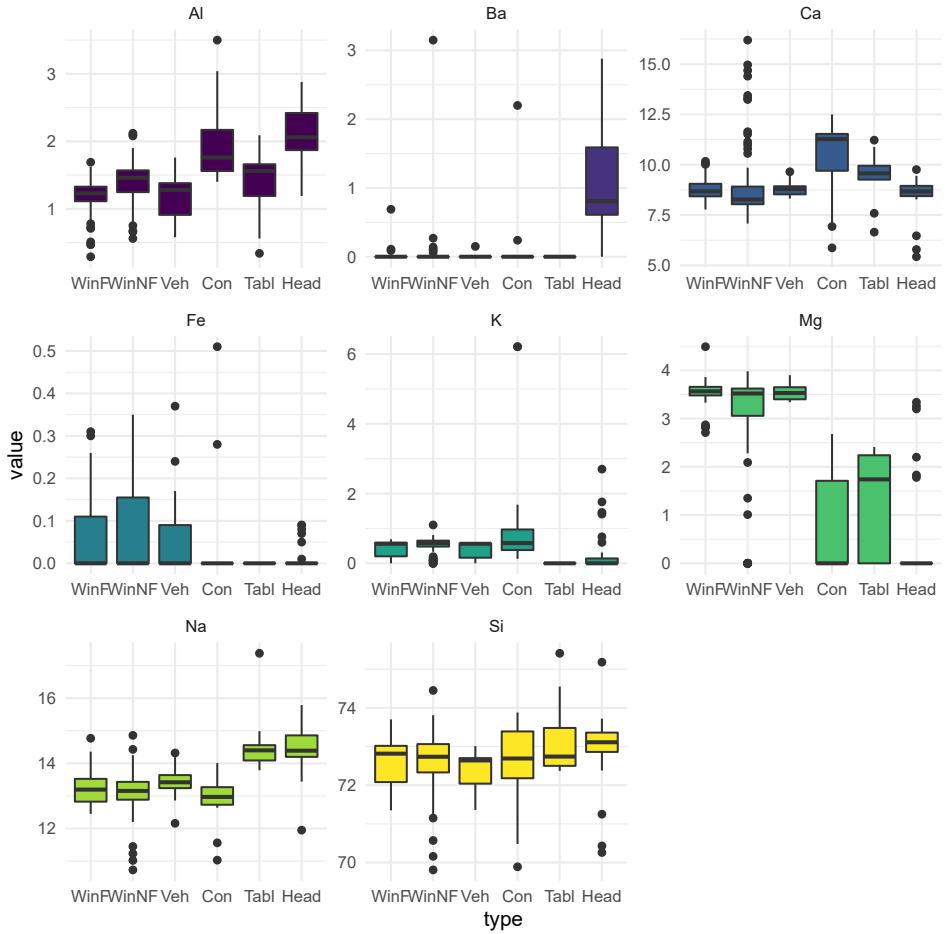
```
fgl_long <-
  fgl_wide %>%
  pivot_longer(-type, names_to = "feature", values_to = "value")

fgl_long
```

```
## # A tibble: 1,926 x 3
##    type  feature  value
##    <fct> <chr>    <dbl>
##  1 WinF  RI        3.01
##  2 WinF  Na       13.6
##  3 WinF  Mg        4.49
##  4 WinF  Al        1.1
##  5 WinF  Si       71.8
##  6 WinF  K         0.06
##  7 WinF  Ca        8.75
##  8 WinF  Ba        0
##  9 WinF  Fe        0
## 10 WinF  RI       -0.390
## # ... with 1,916 more rows
```

Distribution of (some) feature values by glass type

```
fgl_long %>%
  filter(feature != "RI") %>%
  ggplot(aes(x = type, y = value, fill = feature)) +
  geom_boxplot() +
  facet_wrap(~ feature, scales = "free") +
  theme_minimal() +
  scale_fill_viridis_d() +
  theme(legend.position = "none")
```

Some of the features serve as clear discriminators. For example, the element Ba is present in all glass types except for Head, where it is barely present.



Preprocess the Data (Including Interactions)

To make the feature set more "interesting," we add interactions with `RI`:

```
fgl_interact <-
  recipe(type ~ ., data = fgl_wide) %>%
  step_interact(~ all_predictors() * RI) %>%
  step_zv(all_predictors()) %>%
  prep() %>%
  juice()

head(fgl_interact)
```

```
## # A tibble: 6 x 18
##       RI    Na    Mg    Al    Si     K    Ca    Ba    Fe type  RI_x_Na RI_x_Mg RI_x_Al
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>   <dbl>   <dbl>   <dbl>
## 1  3.01   13.6  4.49  1.1   71.8  0.06  8.75     0  0    WinF    41.1    13.5    3.31
## 2 -0.390  13.9  3.6   1.36  72.7  0.48  7.83     0  0    WinF    -5.42   -1.40  -0.530
## 3 -1.82   13.5  3.55  1.54  73.0  0.39  7.78     0  0    WinF   -24.6    -6.46  -2.80
## 4 -0.340  13.2  3.69  1.29  72.6  0.57  8.22     0  0    WinF    -4.49   -1.25  -0.439
## 5 -0.580  13.3  3.62  1.24  73.1  0.55  8.07     0  0    WinF    -7.70   -2.10  -0.719
## 6 -2.04   12.8  3.61  1.62  73.0  0.64  8.07     0  0.26 WinF   -26.1    -7.36  -3.30
## # ... with 5 more variables: RI_x_Si <dbl>, RI_x_K <dbl>, RI_x_Ca <dbl>,
## #   RI_x_Ba <dbl>, RI_x_Fe <dbl>
```

Prepare Input for `glmnet`

Before fitting the model, we need to transform the data into outcome and feature matrices.

```
y <-
  fgl_interact %>%
  pull(type)

x <-
  fgl_interact %>%
  select(-type) %>%
  as.matrix()
```

Note that `y` is a one-dimensional *factor*.

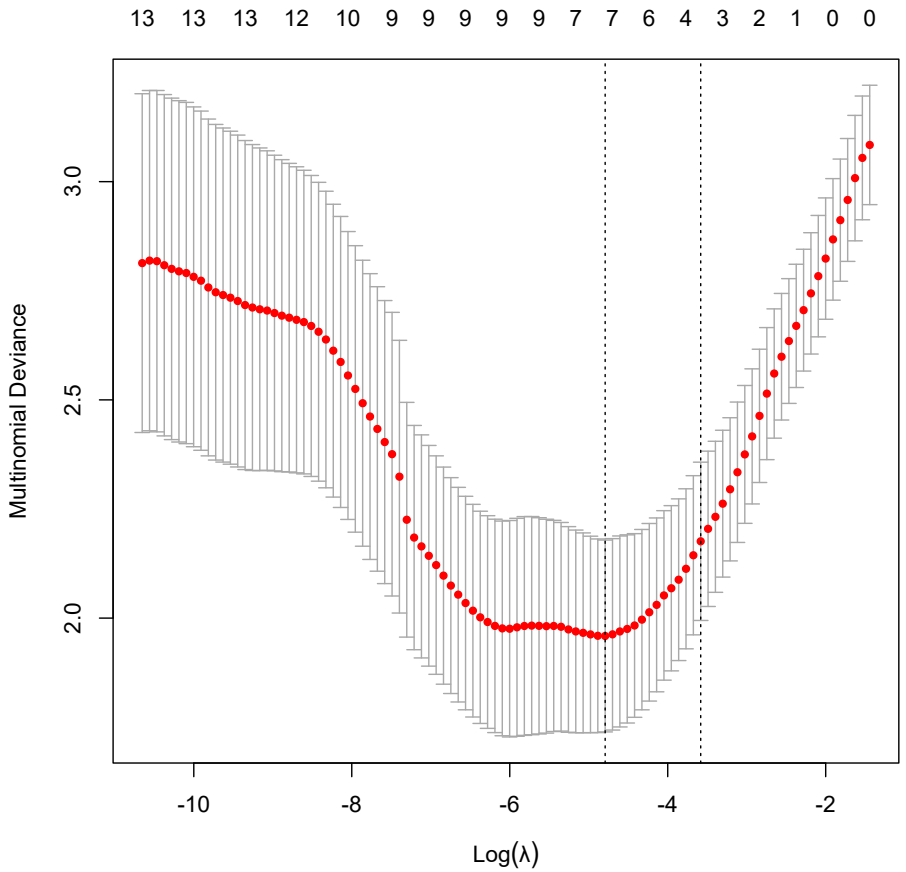
Cross-Validation using `glmnet`

We can estimate the regularized version of the model using `glmnet` with `family = "multinomial"`:

```
fit <- cv.glmnet(
  x = x,
  y = y,
  family = "multinomial"
)
```

and plot the cross-validation results using `plot`

```
plot(fit)
```



Multiclass Prediction

The following code chunk extracts the predicted class and predicted probabilities for each class:

```
class <-
  fit %>%
  predict(newx = x, s = "lambda.1se", type = "class")

prob <-
  fit %>%
  predict(newx = x, s = "lambda.1se", type = "response") %>%
  as_tibble()
```

Maximum Probability Rule

We can organize `class` and `prob` as a tidy `tibble` using the following code:

```
fgl_pred <-
  fgl_wide %>%
  select(type) %>%
  mutate(
    class = class[,1],
    class = factor(class, levels = levels(type))
  ) %>%
  bind_cols(prob)
```

Predicted class is determined using the *maximum probability rule*.

```
fgl_pred %>% sample_n(5)
```

```
## # A tibble: 5 x 8
##   type  class WinF.1 WinNF.1  Veh.1  Con.1  Tabl.1  Head.1
##   <fct> <fct>  <dbl>   <dbl>  <dbl>  <dbl>   <dbl>   <dbl>
## 1 Veh   WinF  0.417    0.417 0.0915 0.0283 0.0238  0.0225
## 2 WinNF WinF  0.483    0.349 0.0898 0.0201 0.0352  0.0225
## 3 Head  Con   0.131    0.297 0.102  0.374  0.0132  0.0824
## 4 WinNF WinNF 0.0229   0.899 0.0170 0.0508 0.00268 0.00780
## 5 WinNF WinNF 0.319    0.528 0.0841 0.0302 0.0175  0.0217
```

Multiclass Confusion Matrix

We can generate the multiclass confusion matrix using the `conf_mat()` function from the `{yardstick}` package:

```
fgl_pred %>%
  conf_mat(type, class)
```

```
##            Truth
## Prediction WinF WinNF Veh Con Tabl Head
##      WinF    53    18   8   0    0    1
##      WinNF   17    58   9   9    6    2
##      Veh      0     0   0   0    0    0
##      Con      0     0   0   3    0    1
##      Tabl     0     0   0   0    1    0
##      Head     0     0   0   1    2   25
```

For example, our model correctly classified 53 observations as `WinF` out of all the predicted `WinF` samples, resulting in a precision rate of $53/80 = 66.3\%$.

Similarly, the model correctly predicted 58 `WinNF` samples out of the actual number of `WinNF` samples, which is 76, resulting in a recall rate of $58/76 = 76.3\%$.

Multiclass ROC Curve(s)

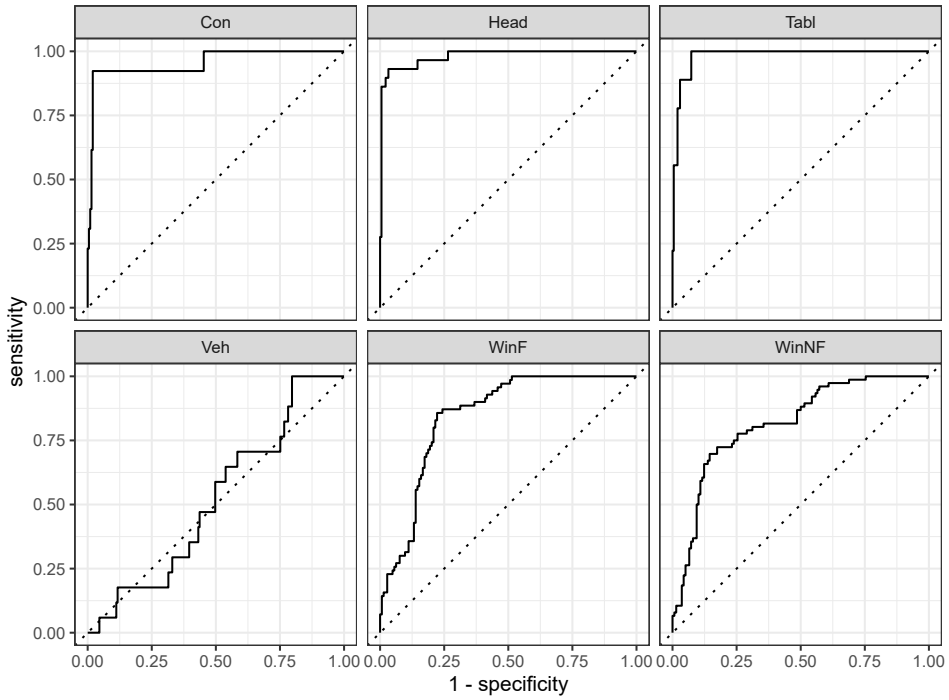
A common approach is to use a one-vs-all strategy to calculate multiple ROC curves.

We can plot multiclass ROC curves using the `roc_curve` function from the `{yardstick}` package:

```
fgl_pred %>%
  roc_curve(type, WinF.1:Head.1) %>%
  autoplot()
```

where `WinF.1:Head.1` represents the model's fitted probabilities.

The model clearly struggles to differentiate between `Veh` and the other classes, while its classification for `Tabl` is nearly perfect.



Multiclass ROC-AUC

Hand and Till (2001) extended the definition of ROC-AUC to the case of more than two classes by averaging pairwise comparisons.

To calculate the multiclass AUC value, you can utilize the `roc_auc` function from the `{yardstick}` package:

```
fgl_pred %>%
  roc_auc(type, WinF.1:Head.1)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.858
```

`slides::end()`

 Source code

References

Hand, D. J., & Till, R. J. (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems". *Machine Learning*, 45(2), 171-186.

Lantz, B. (2020). *Machine Learning with R: Expert techniques for predictive modeling* (3rd ed.). Packt Publishing.

Taddy, M. (2020). *Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions*. McGraw-Hill Education.