06 - Classification

ml4econ, HUJI 2021

Itamar Caspi

April 18, 2021 (updated: 2021-04-17)

Packages and setup

Use the {pacman} package that automatically loads and installs packages if necessary:

```r
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
)
```

Set a theme for `ggplot` (Relevant only for the presentation)

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

And set a seed for replication

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

Outline

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 denote the outcome of a COVID-19 test, where

$$y_i = \begin{cases} 1 & \text{if positive,} \\ 0 & \text{if negative,} \end{cases}$$

where the values 1 and 0 are chosen for simplicity.[1]

Two types of questions we might ask:

1. What is the probability of being positive?
2. Can we classify an individual as positive/negative?

[*] It is common to find a $\{1, -1\}$ notation for binary outcomes in the ML literature.

Israeli COVID-19 tests data

The The Isreali Ministry of Health provides information on more than 100,000 COVID-19 test results. Our aim here is to predict which person will be classified as "positive", i.e. infected by the virus, based on his symptoms and characteristics.

Outcome variable: `corona_result`

Features:

- Symptoms
  - `cough`
  - `fever`
  - `sore_throat`
  - `shortness_of_breath`
  - `head_ache`
- Characteristics
  - `age_60_and_above`
  - `gender`

Read and examine the data

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

```r
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, ...
## $ fever               <dbl> 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, ...
## $ sore_throat         <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 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, ...
## $ head_ache           <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ corona_result       <chr> "negative", "negative", "negative", "positive", "ne...
## $ age_60_and_above    <chr> "No", "No", "Yes", "Yes", "Yes", "Yes", "No", "No",...
## $ gender              <chr> "male", "male", "male", "male", "female", "male", "...
```

Note that since $n = 107,542$ and $p = 7$, we should not worry much about overfitting.

Preprocessing

We'll now define all variables, outcome and features, as factors:

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

and extract the outcome and features as matrices (for later use with `glmnet`):

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

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

Raw detection frequencies

How are 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 is an example of **class imbalance** (the distribution of examples across the known classes is skewed), which is a typical feature of classification problems.

Measuring classification accuracy

What does MSE mean in the context of classification problems?

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

In words: In this case, MSE measures the **missclassifcation rate**, i.e., the ratio between the number of missclassifications and the total number of observations.

Classification accuracy is the total number of correct predictions divided by the total number of predictions made for a dataset.

Clearly,

$$accuracy = 1 - missclasification.$$

Are missclasification/accuracy rates useful? Think imbalanced outcome.

A naive classifier

Our naive "model" says: "classify everyone as being negative"

```r
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 the model is $98,586/107,542 = 91.67\%$!

Pretty impressive! Or is it?

This naive classifier lacks the ability to discern one class versus the other, and more importantly, it fails to identify infected individuals - the thing we really care about!

The Confusion Matrix

Beyond accuracy – other measures of performance

The **confusion matrix** is a table that categorizes predictions according to whether they match the ground truth.

		Truth	**Truth**
		Negative	Positive
Prediction	Negative	*True negative* (TN)	*False negative* (FN)
Prediction	Positive	*False positive* (FP)	*True positive* (TP)

Note that $TP + TN + FP + TP = N$, where N is the number of observations. Accuracy in this case is defined as $(TN + TP)/N$.

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

Types of classification errors

False positive rate: The fraction of negative examples that are classified as positive, $0/98,586 = 0\%$ in example.

False negative rate: The fraction of positive examples that are classified as negative, $8,956/8,956 = 100\%$ in example.

Can we do better?

A perfect classifier

Here is a simple example. Let’s assume we have a sample of 100 test results, and exactly 20 of them are labeled "positive". If our classifier was perfect, the confusion matrix would look like this:

		Truth	**Truth**
		Negative	Positive
Prediction	Negative	80	0
Prediction	Positive	0	20

That is, our classifier has a 100% accuracy rate, zero false positive and zero false negative.

The realistic classifier

Now, here is a classifier that makes some errors:

		Truth	**Truth**
		Negative	Positive
Prediction	Negative	70	10
Prediction	Positive	5	15

In this example, 10 persons with the pathogen were classified as Negative (not infected), and 5 persons without the pathogen were 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 goal is to predict $\Pr(y_i = 1|\mathbf{x}_i)$.

Let p_i denote the probability of seeing $y_i = 1$ 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 regression of y_i on $\mathbf{x}_i$ ignores the discreteness of the dependent variable and does not constrain predicted probabilities to be between zero and one.

Logitic regression model

A more appropriate model is the **logit model** or **logistic regression model** specifies as

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

where $\Lambda(\cdot)$ is the logistic cdf. As such, the model imposes the restriction that $0 \leq p_i \leq 1$.

Odds-ratio

Note that

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

Taking logs yields

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

The above is useful representation of the logistic regression model. The LHS is called the log **odds ratio** (or relative risk.)

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

The likelihood function

Likelihood refers to the probability of seeing the data given parameters.

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

taking (natural) logs yields the **log likelihood**

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

In estimation, we want to make the above as big as possible (hence, maximum likelihood estimation, MLE).

Deviance

Another usefule conceppt is the **deviance**, a generalization of the concept of "least squares" to general linear models (such as logit), and is a measure of the distance between data and fit.

The relationship between deviance and likelihood is given by

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

The constant wrapps terms that relate to the likelihood of the "perfect" model and we can mostly ignore it.

Deviance and estimation

In estimation, we want to make deviance as *small* as possible.

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

This is the what R's `glm` function minimizes for logistic regressions.

(**NOTE:** In linear models, the deviance is porportional to the RSS)

Penalized logistic regression

We can also minimized the deviance subject to a standard lasso type (ℓ_1 norm) penalty on β:

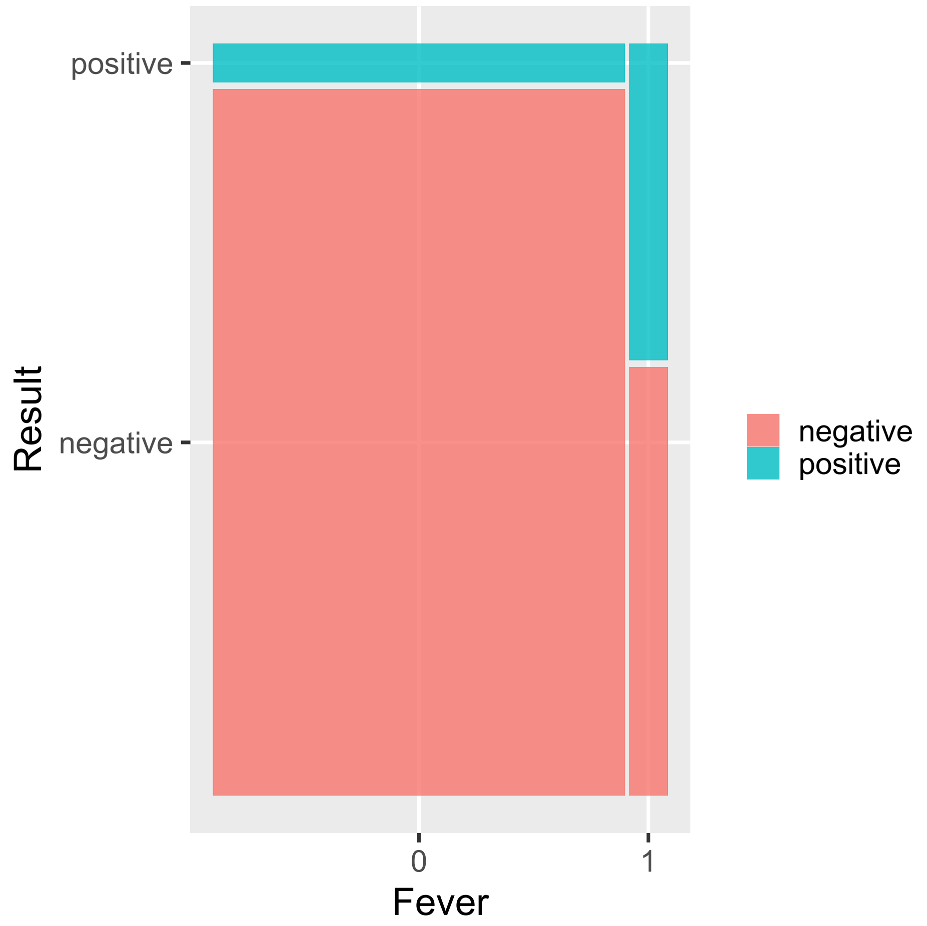
$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} \left[\frac{1}{N}\sum_{i=1}^{N}\log\left(1+e^{(\beta_0+x_i'\beta)}\right) - y_i\cdot(\beta_0+x_i'\beta)\right] + \lambda\|\beta\|_1$$

where again, the penalty is on the sum of the absolute values of β (no including the intercept.)

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

There is some evidence that having fever is associated with being "positive".

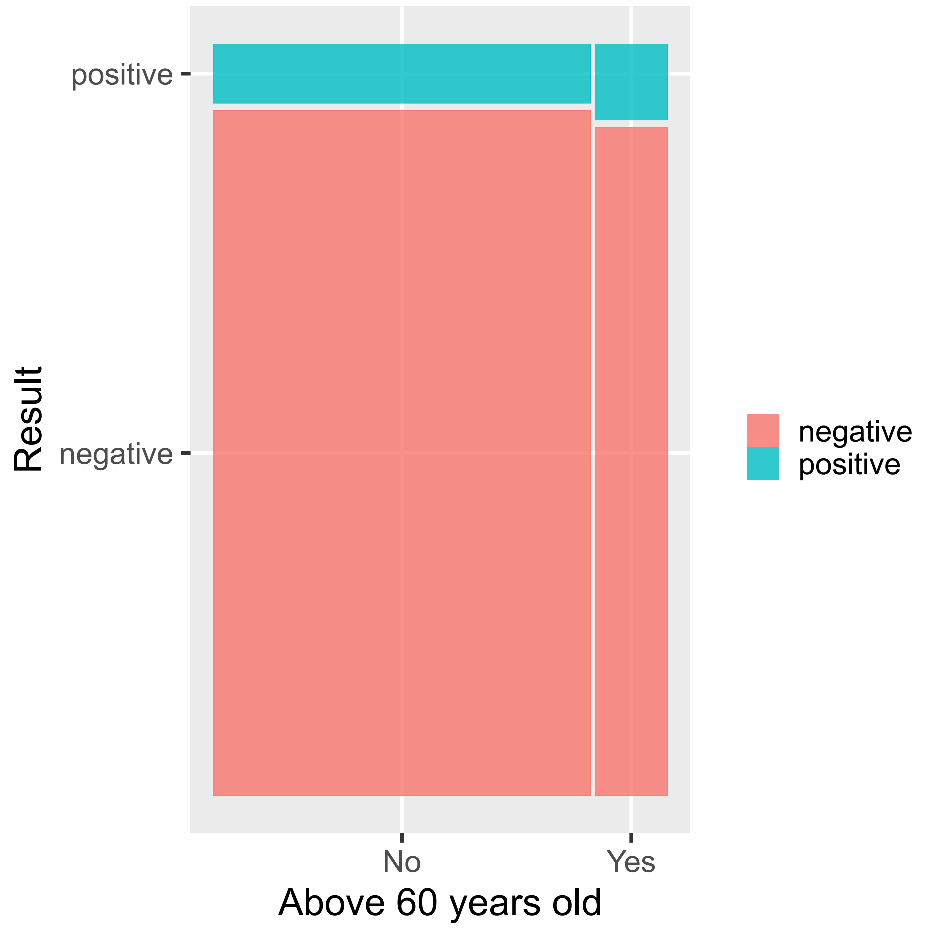
```
covid %>%
  ggplot() +
  geom_mosaic(
    aes(x = product(corona_result, fever),
        fill = corona_result)
  ) +
  labs(
    x = "Fever",
    y = "Result",
    fill = ""
  )
```



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

We will estimate the model using base R's `glm` (stands for generalized linear model) function:

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

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

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

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

Model output

The `tidy()` and `glance()` functions from the `{broom}` package provides tidy summary of the output from `glm` objects:

```
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 7
##   null.deviance df.null  logLik    AIC    BIC deviance df.residual
##           <dbl>   <int>   <dbl>  <dbl>  <dbl>    <dbl>       <int>
## 1        61666.  107541 -20726. 41468. 41544.   41452.      107534
```

Generate predictions

The `augment()` function (also from `{broom}`) augments the original dataframe with the fitted values (and standard errors)

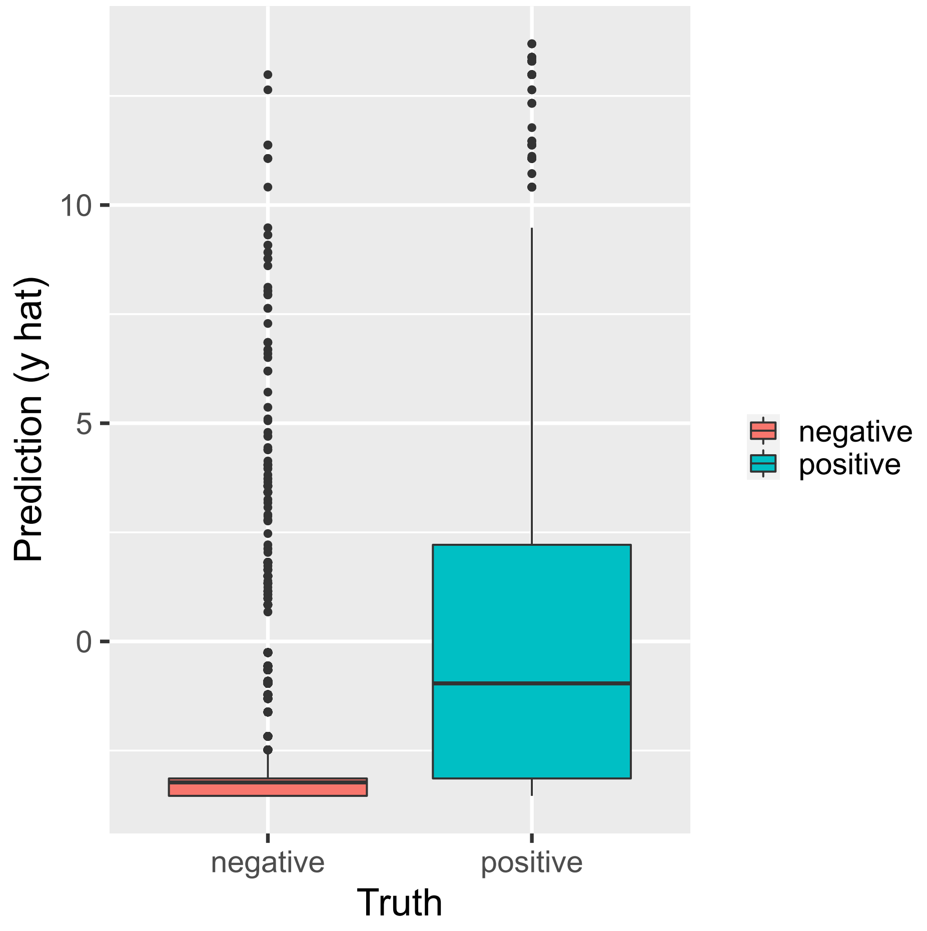
```
covid_pred <-
  logit_model %>%
  augment()

covid_pred
```

```
## # A tibble: 107,542 x 15
##    corona_result cough fever sore_throat shortness_of_br~ head_ache age_60_and_above
##    <fct>         <fct> <fct> <fct>       <fct>            <fct>     <fct>
##  1 negative      1     0     0           0                0         No
##  2 negative      1     1     0           0                0         No
##  3 negative      0     1     0           0                0         Yes
##  4 positive      1     1     0           0                0         Yes
##  5 negative      1     0     0           0                0         Yes
##  6 positive      1     1     0           0                0         Yes
##  7 negative      0     0     0           0                0         No
##  8 negative      1     1     1           0                1         No
##  9 negative      1     0     1           0                0         No
## 10 negative      1     1     1           1                0         No
## # ... with 107,532 more rows, and 8 more variables: gender <fct>, .fitted <dbl>,
## #   .se.fit <dbl>, .resid <dbl>, .hat <dbl>, .sigma <dbl>, .cooksd <dbl>,
## #   .std.resid <dbl>
```

Model predictions (in sample)

The figure on the right shows the resulting in-sample fit. There appears to be little overlap between probabilities for the true positives and the true negatives.

```r
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/negative we first need to set a **classification rule** (cut-off), i.e., a probability p^* above which we classify an individual as positive.

For illustration, we'll set $p^* = 0.8$:

```r
class_rule <- 0.8
```

This means that whenever $\hat{y}_i > 0.8$, we would 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've seen, classifying everyone as "negative" $(p^* = 1)$, fails to be specific, i.e., it fails to identify any positive results (what we really care about!):

Sensitivity: The fraction of positive examples that are classified as positive ("true positive rate"), $98,586/98,586 = 100\%$ in example.

Specificity: The fraction of negative examples yhat are classified as negative ("true negative rate"), $0/8,956 = 0\%$ in example.

Note that in general,

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

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

Our model's confusion matrix

The function `cnf_mat()` from the `{yardstick}` package provides easy access to a model's confusion matrix and the implied 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
```

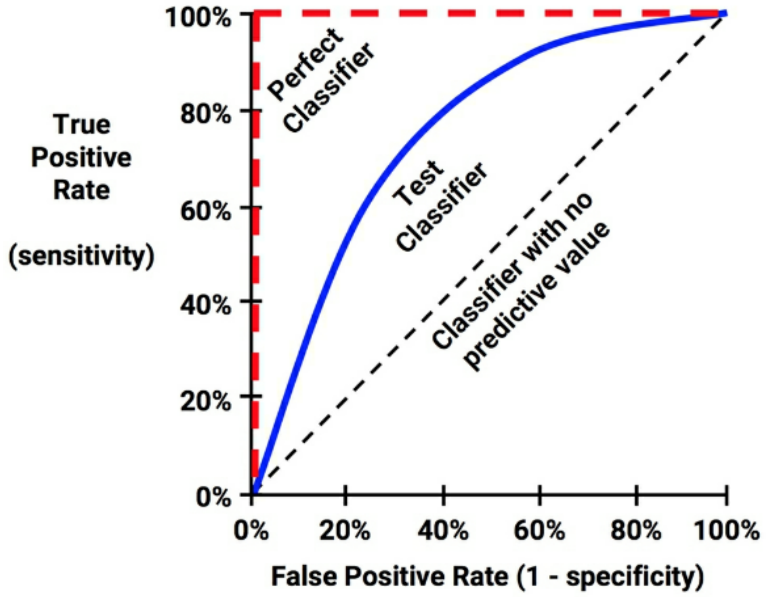
```
covid_conf_mat%>%
  summary() %>%
  filter(.metric %in% c("accuracy", "sens", "spec"))
  mutate("1-.estimate" = 1 - .estimate)
```

```
## # 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, for `class_rule = 0.8`, the model is highly sensitive but not so specific. Clearly, changing the rule would change 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. By doing so, it highlights the trade-off between false-positive and true-positive error rates as the classifier threshold is varied.



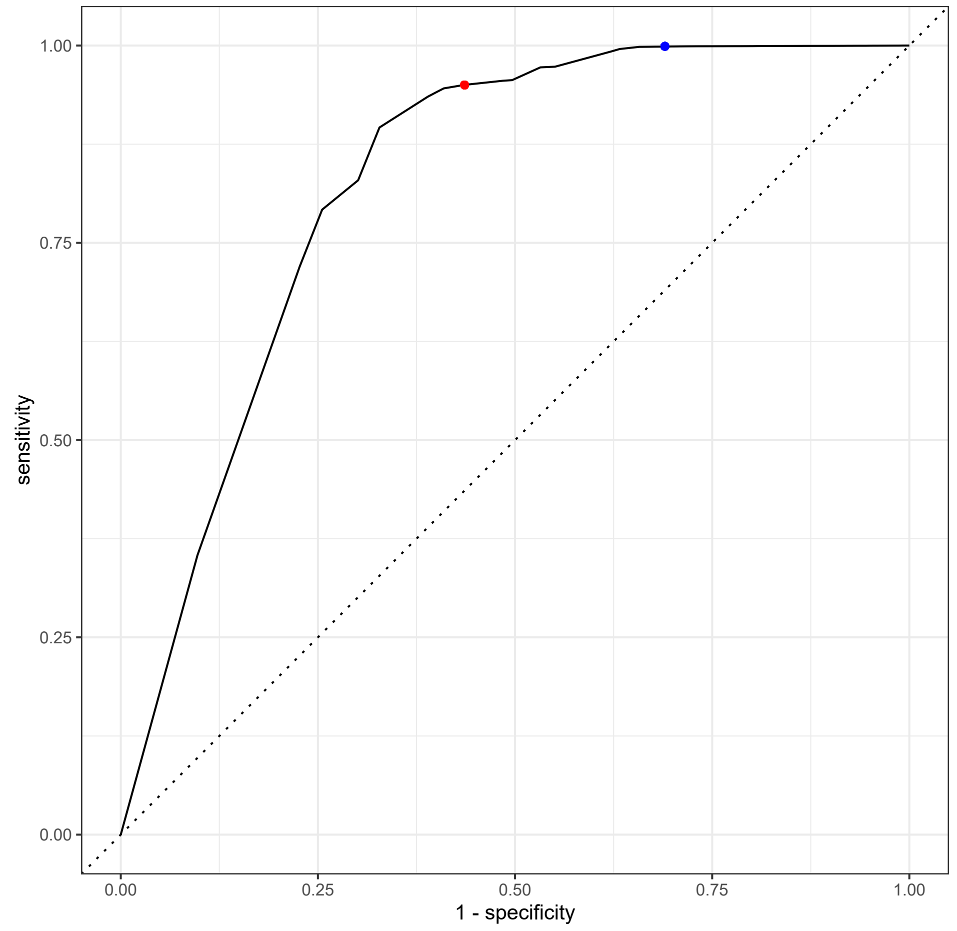
Source: "Machine Learning with R: Expert techniques for predictive modeling"

Our model's ROC curve

On the left, you can see our model's ROC curve, plotted using the `roc_curve()` function. The red and blue dots correspond to two cut-offs, 0.8 and 0.2, respectively.

```r
covid_pred %>%
  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`.

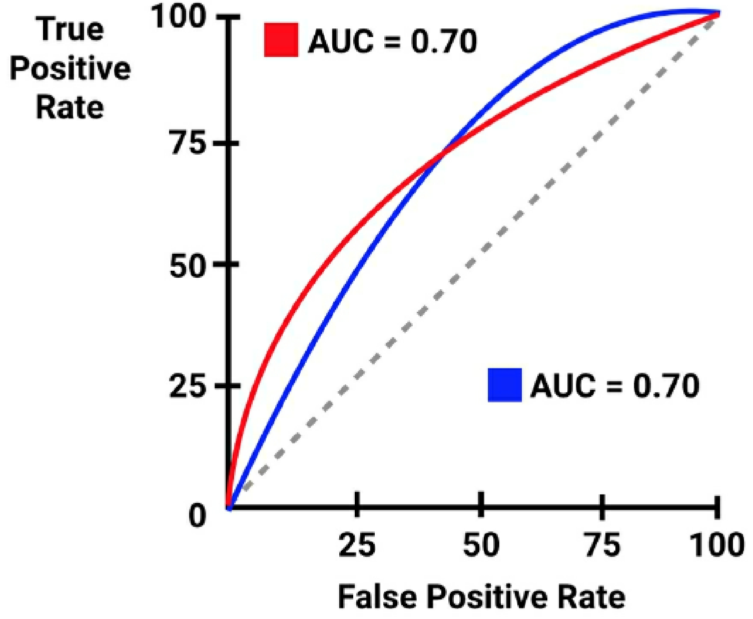


Area under the curve (AUC)

- Ranking of classifiers can be made based on the area under the ROC curve (AUC).
- For example, a perfect classifier has `auc=1` and a classifier with no discriminate value has `auc=0.5`.
- Nevertheless, identical `auc` values can result from two different ROC curves. Thus, qualitative examination is warrant.

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

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.827
```



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

AUC and cross-validation

When it comes to classification tasks, it is sometimes more reasonable to tune the penalty parameter based on classification performance metrics (and not on, say, deviance.)

For example, we can use the `cv.glmnet()` function while setting the `type.measure = "auc"` in order to tune based on auc values

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

or set `type.measure = "class"` to tune based on the misclassification rate.

Multiclass Classification

Multiclass outcomes

- Each observation belongs to one of $j = 1, \ldots, G$ Classes (groups)
- Outcome variable

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

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

- Conditional probability

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

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

Multinomial regression model

For each class we model the outcome 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: There is no explicit base class here since regularized solutions are not equivariant under base changes, and regularization automatically eliminates the redundancy

Likelihood and deviance

Given 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 is the total number of observations.

Taking logs and multiplying by -2 yields the multinomial deviance

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

Regularization

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

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

Similar to the binomial case, we can minimized 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`) data frame has 214 rows and 10 columns.

The data include for each of 214 shards of glass, measurements on the refractive index (`RI`) and 8 measurements of chemical composition by weight of oxide (percentage) for elements `Na`, `Mg`, `Al`, `Si`, `K`, `Ca`, `Ba`, and `Fe`.

The fragments were originally classed into seven types:

- `WinF`: window float glass
- `WinNF`: window non-float glass
- `Veh`: vehicle window glass
- `Con`: containers
- `Tabl`: tableware
- `Head`: vehicle headlamps

Out objective is to classify new data to one the above six types.

Load and inspect the `fgl` data

The `fgl` data comes with 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  -0.39   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 0.570  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
```

Tidy the data using `pivot_longer`

The following code chunk transforms the date from wide to long format using the `pivot_longer()` function from the `{tidyr}` package (this will come in handy soon when we plot the data.):

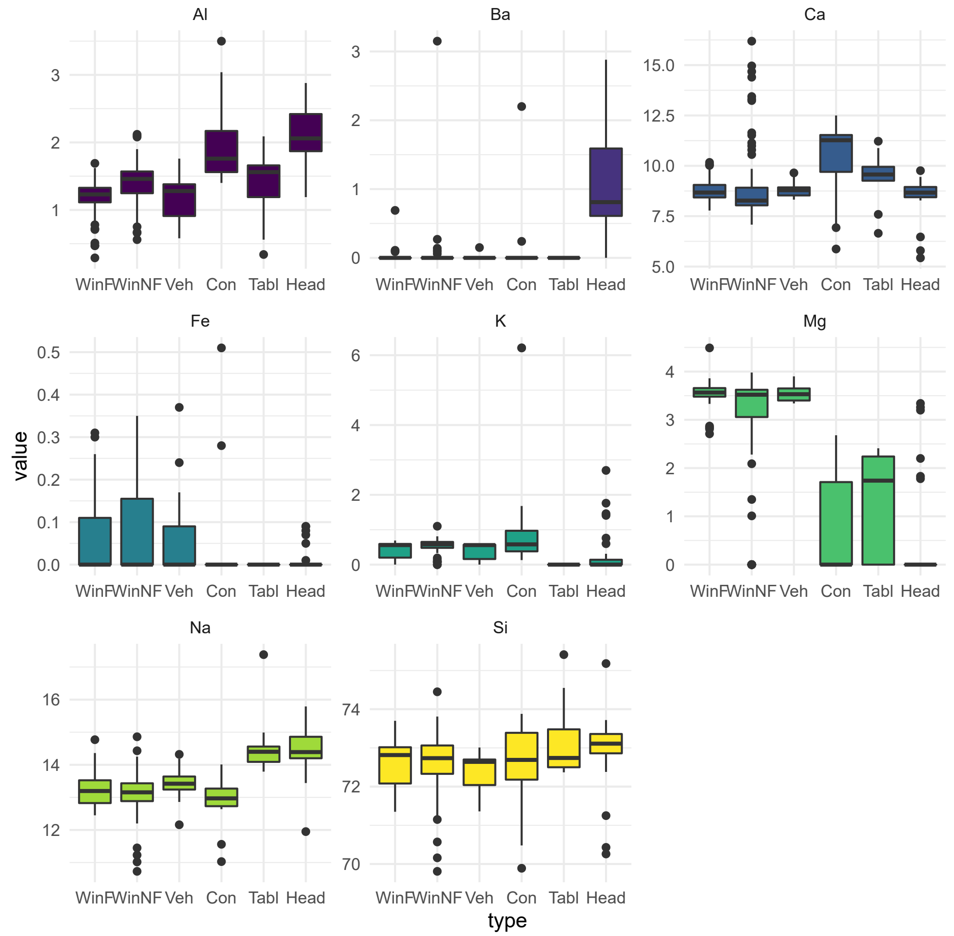
```
fgl_long <-
  fgl_wide %>%
  pivot_longer(-type, names_to = "feature", values_to = "value")

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      -0.39
## # ... with 1,916 more rows
```

Distribution of (some) feature values by glass type

```r
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 are clear discriminators, e.g., `Ba` is barely present in all glass types but `Head`.



Preprocess the data (some interactions)

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

```r
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
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>   <dbl>   <dbl>
## 1  3.01   13.6  4.49  1.1   71.8 0.06   8.75     0  0    WinF    41.1    13.5 
## 2 -0.39   13.9  3.6   1.36  72.7 0.48   7.83     0  0    WinF    -5.42   -1.40
## 3 -1.82   13.5  3.55  1.54  73.0 0.39   7.78     0  0    WinF   -24.6    -6.46
## 4 -0.340  13.2  3.69  1.29  72.6 0.570  8.22     0  0    WinF    -4.49   -1.25
## 5 -0.580  13.3  3.62  1.24  73.1 0.55   8.07     0  0    WinF    -7.70   -2.10
## 6 -2.04   12.8  3.61  1.62  73.0 0.64   8.07     0  0.26 WinF   -26.1    -7.36
## # ... with 6 more variables: RI_x_Al <dbl>, RI_x_Si <dbl>, RI_x_K <dbl>,
## #   RI_x_Ca <dbl>, RI_x_Ba <dbl>, RI_x_Fe <dbl>
```

Prepare input to `glmnet`

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

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

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

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

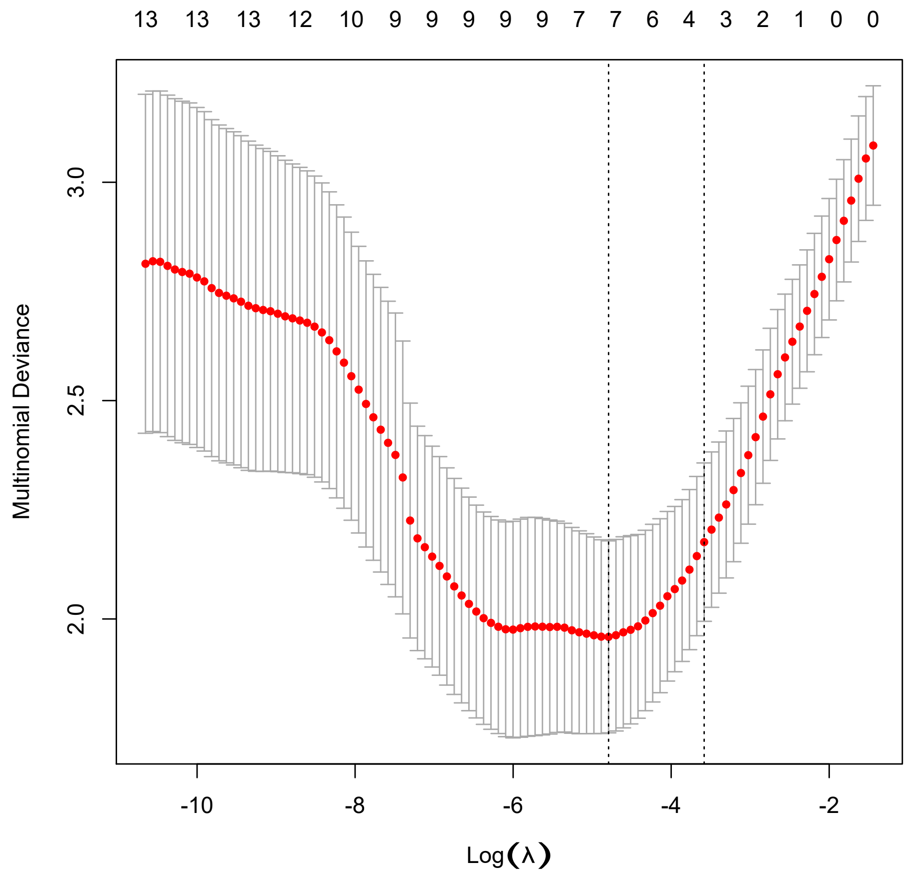
Cross-validation using `glmnet`

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 pf belonging to each class

```r
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 rearrange `class` and `prob` as a nice `tibble`:

```
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 print the multiclass confusion matrix using the `conf_mat()` function (from `{yardstick}`):

```
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 54 observations as `WinF` out of all predicted `WinF` ($54/80 = 67.5\%$ precision rate.)

On the other hand, the model correctly predicted 54 `WinF`out of the number of actual `WinF`, 64, which is $54/70 = 77.1\%$.

Multiclass ROC curve(s)

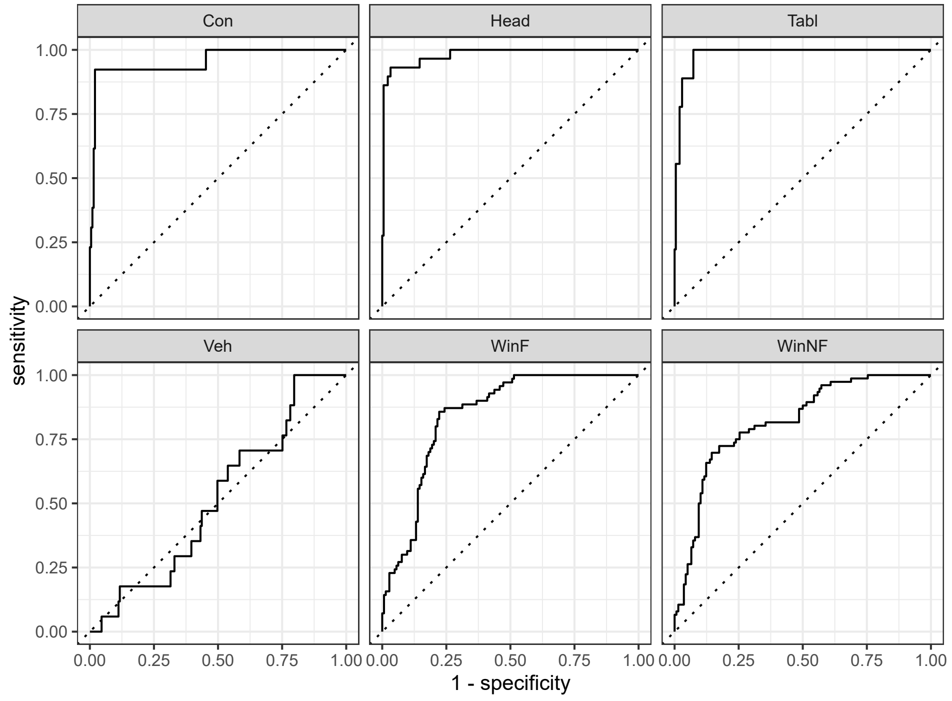
A one-vs-all approach is often taken to calculate multiple ROC curves.

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

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

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

See how the model fails to distinguish between `Veh` and the others, whereas classifying as `Tabl` is almost perfect.



Multiclass ROC-AUC

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

Calculating the multiclass AUC value can be done using the `roc_auc` function from the `{yardstick}` function:

```
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      0.867
```

`slides::end()`

Source code

References

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

Lantz, Brett. Machine Learning with R: Expert techniques for predictive modeling, 3rd Edition (p. 333). Packt Publishing.

Taddy, Matt. B*usiness Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions*. McGraw-Hill Education.