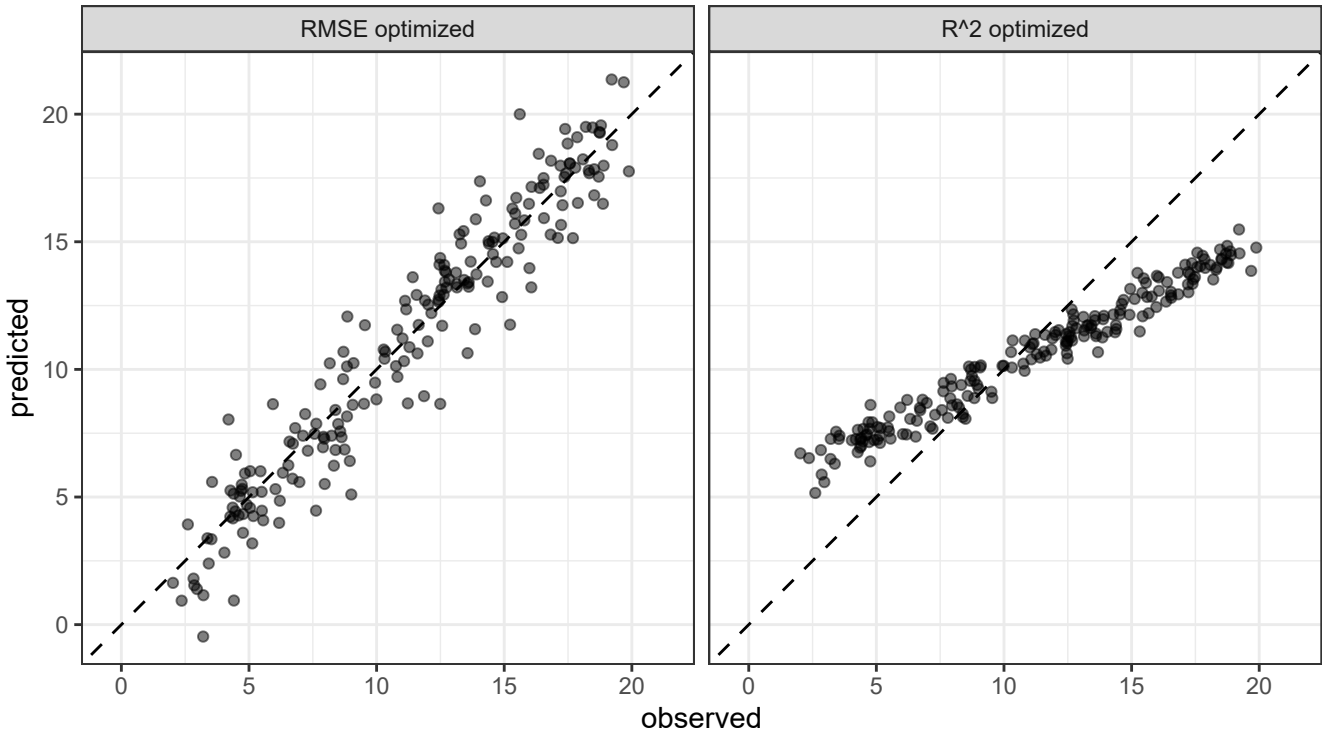
9 Judging model effectiveness

Once we have a model, we need to know how well it works. A quantitative approach for estimating effectiveness allows us to understand the model, to compare different models, or to tweak the model to improve performance. Our focus in tidymodels is on *empirical validation*; this usually means using data that were not used to create the model as the substrate to measure effectiveness.

> The best approach to empirical validation involves using *resampling* methods that will be introduced in Chapter 10. In this chapter, we will use the test set for illustration purposes and to motivate the need for empirical validation. Keep in mind that the test set can only be used once, as explained in Section 5.1.

The choice of which metrics to examine can be critical. In later chapters, certain model parameters will be empirically optimized and a primary performance metric will be used to choose the best *sub-model*. Choosing the wrong method can easily result in unintended consequences. For example, two common metrics for regression models are the root mean squared error (RMSE) and the coefficient of determination (a.k.a. R^2). The former measures *accuracy* while the latter measures *correlation*. These are not necessarily the same thing. This figure demonstrates the difference between the two:



A model optimized for RMSE has more variability but has relatively uniform accuracy across the range of the outcome. The right panel shows that there is a tighter correlation between the observed and predicted values but this model performs poorly in the tails.

This chapter will largely focus on the **yardstick** package. Before illustrating syntax, let's explore whether empirical validation using performance metrics is worthwhile when a model is focused on inference rather than prediction.

9.1 PERFORMANCE METRICS AND INFERENCE

The effectiveness of any given model depends on how the model will be used. An inferential model is used primarily to understand relationships, and typically is discussed with a strong focus on the choice (and validity) of probabilistic distributions and other generative qualities that define the model. For a model used primarily for prediction, by contrast, predictive strength is primary and

concerns about underlying statistical qualities may be less important. Predictive strength is usually focused on how close our predictions come to the observed data, i.e., fidelity of the model predictions to the actual results. This chapter focuses on functions that can be used to measure predictive strength. However, our advice for those developing inferential models is to use these techniques *even when the model will not be used with the primary goal of prediction*.

A longstanding issue with the practice of inferential statistics is that, with a focus purely on inference, it is difficult to assess the credibility of a model. For example, consider the Alzheimer's disease data from Craig–Schapiro et al. (2011) when 333 patients were studied to determine the factors that influence cognitive impairment. An analysis might take the known risk factors and build a logistic regression model where the outcome is binary (impaired/non-impaired). Let's consider predictors for age, sex, and the Apolipoprotein E genotype. The latter is a categorical variable with the six possible combinations of the three main variants of this gene. Apolipoprotein E is known to have an association with dementia (Jungsu, Basak, and Holtzman 2009).

A superficial, but not uncommon, approach to this analysis would be to fit a large model with main effects and interactions, then use statistical tests to find the minimal set of model terms that are statistically significant at some pre-defined level. If a full model with the three factors and their two- and three-way interactions were used, an initial phase would be to test the interactions using sequential likelihood ratio tests (Hosmer and Lemeshow 2000).

- When comparing the model with all two-way interactions to one with the additional three-way interaction, the likelihood ratio tests produces a p-value of 0.888. This implies that there is no evidence that the 4 additional model terms associated with the three-way interaction explain enough of the variation in the data to keep them in the model.
- Next, the two-way interactions are similarly evaluated against the model with no interactions. The p-value here is 0.0382. This is somewhat borderline, but, given the small sample size, it would be prudent to conclude that there is evidence that some of the 10 possible two-way interactions are important to the model.
- From here, we would build some explanation of the results. The interactions would be particularly important to discuss since they may spark interesting physiological or neurological hypotheses to be explored further.

While shallow, this analysis strategy is common in practice as well as in the literature. This is especially true if the practitioner has limited formal training in data analysis.

One missing piece of information in this approach is how closely this model fits the actual data. Using resampling methods, discussed in Chapter 10, we can estimate the accuracy of this model to be about 73.3%. Accuracy is often a poor measure of model performance; we use it here because it is commonly understood. If the model has 73.3% fidelity to the data, should we trust the conclusions produced by the model? We might think so until we realize that the baseline rate of non-impaired patients in the data is 72.7%. This means that, despite our statistical analysis, the two-factor model appears to be *only 0.6% better than a simple heuristic that always predicts patients to be unimpaired*, irregardless of the observed data.

> The point of this analysis is to demonstrate the idea that **optimization of statistical characteristics of the model does not imply that the model fits the data well.** Even for purely inferential models, some measure of fidelity to the data should accompany the inferential results. Using this, the consumers of the analyses can calibrate their expectations of the results of the statistical analysis.

In the remainder of this chapter, general approaches for evaluating models via empirical validation are discussed. These approaches are grouped by the nature of the outcome data: purely numeric, binary classes, and three or more class levels.

9.2 REGRESSION METRICS

Recall from Section 6.3 that tidymodels prediction functions produce tibbles with columns for the predicted values. These columns have consistent names, and the functions in the **yardstick** package that produce performance metrics have consistent interfaces. The functions are data frame-based, as opposed to vector-based, with the general syntax of:

```
function(data, truth, ...)
```

where `data` is a data frame or tibble and `truth` is the column with the observed outcome values. The ellipses or other arguments are used to specify the column(s) containing the predictions.

To illustrate, let's take the model from Section 8.8. The `lm_wflow_fit` object was a linear regression model whose predictor set was supplemented with an interaction and spline functions for longitude and latitude. It was created from a training set (named `ames_train`). Although we do not advise using the test set at this juncture of the modeling process, it will be used to illustrate functionality and syntax. The data frame `ames_test` consists of 588 properties. To start, let's produce predictions:

```
ames_test_res <- predict(lm_fit, new_data = ames_test %>% select(-Sale_Price))
ames_test_res
#> # A tibble: 588 × 1
#>   .pred
#>   <dbl>
#> 1  5.07
#> 2  5.17
#> 3  5.28
#> 4  5.05
#> 5  5.51
#> 6  5.42
#> # … with 582 more rows
```

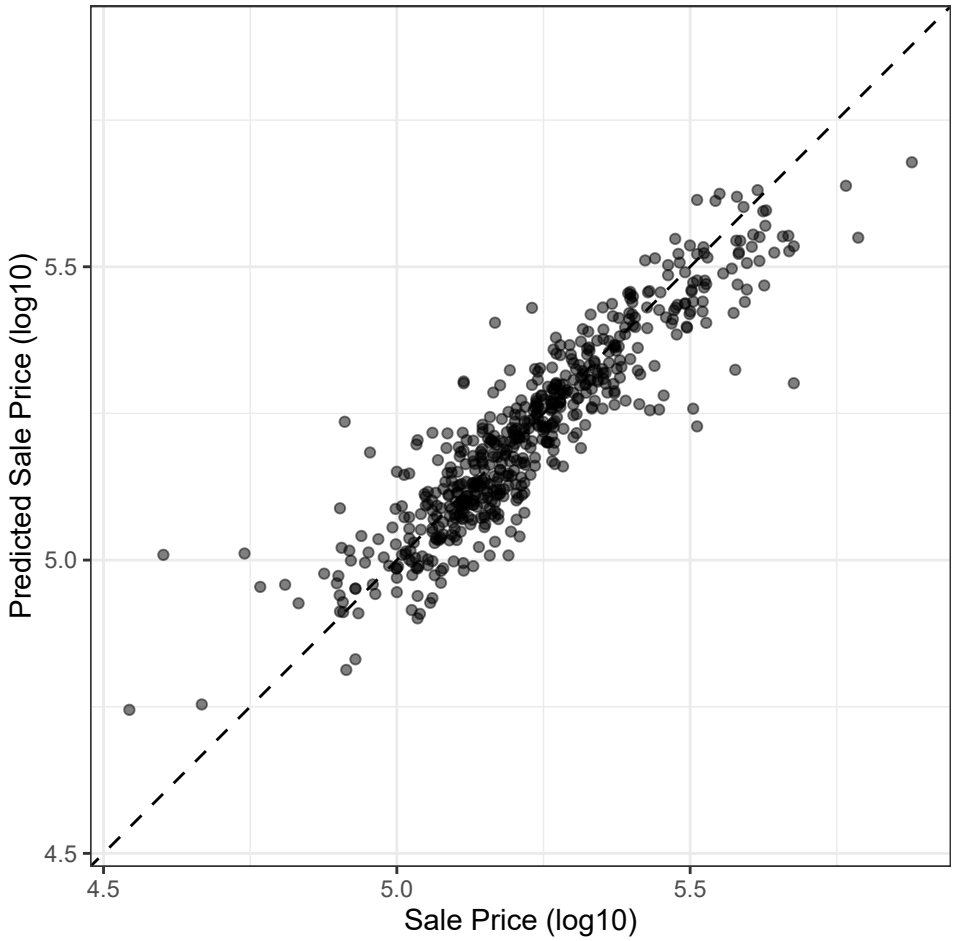
The predicted numeric outcome from the regression model is named `.pred`. Let's match the predicted values with their corresponding observed outcome values:

```
ames_test_res <- bind_cols(ames_test_res, ames_test %>% select(Sale_Price))
ames_test_res
#> # A tibble: 588 × 2
#>   .pred Sale_Price
#>   <dbl>      <dbl>
#> 1  5.07       5.02
#> 2  5.17       5.24
#> 3  5.28       5.28
#> 4  5.05       5.06
#> 5  5.51       5.60
#> 6  5.42       5.33
#> # … with 582 more rows
```

Note that both the predicted and observed outcomes are in log10 units. It is best practice to analyze the predictions on the transformed scale (if one were used) even if the predictions are reported using the original units.

Let's plot the data before computing metrics:

```
ggplot(ames_test_res, aes(x = Sale_Price, y = .pred)) +
  # Create a diagonal line:
  geom_abline(lty = 2) +
  geom_point(alpha = 0.5) +
  labs(y = "Predicted Sale Price (log10)", x = "Sale Price (log10)") +
  # Scale and size the x- and y-axis uniformly:
  coord_obs_pred()
```



There is one property that is substantially over-predicted.

Let's compute the root mean squared error for this model using the `rmse()` function:

```
rmse(ames_test_res, truth = Sale_Price, estimate = .pred)
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 rmse    standard      0.0754
```

The output above shows the standard format of the output of **yardstick** functions. Metrics for numeric outcomes usually have a value of "standard" for the `.estimator` column. Examples with different values for this column are shown in the next sections.

To compute multiple metrics at once, we can create a *metric set*. Let's add R^2 and the mean absolute error:

```
ames_metrics <- metric_set(rmse, rsq, mae)
ames_metrics(ames_test_res, truth = Sale_Price, estimate = .pred)
#> # A tibble: 3 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 rmse    standard      0.0754
#> 2 rsq     standard      0.826
#> 3 mae     standard      0.0546
```

This tidy data format stacks the metrics vertically.

> The **yardstick** package does *not* contain a function for adjusted R^2. This commonly used modification of the coefficient of determination is needed when the same data used to fit the model are used to evaluate the model. This metric is not fully supported in tidymodels because it is always a better approach to compute performance on a separate data set than the one used to fit the model.

9.3 BINARY CLASSIFICATION METRICS

To illustrate other ways to measure model performance, we will switch to a different example. The **modeldata** package contains example predictions from a test data set with two classes ("Class1" and "Class2"):

```
data(two_class_example)
str(two_class_example)
#> 'data.frame':    500 obs. of  4 variables:
#>  $ truth    : Factor w/ 2 levels "Class1","Class2": 2 1 2 1 2 1 1 1 2 2 ...
#>  $ Class1   : num  0.00359 0.67862 0.11089 0.73516 0.01624 ...
#>  $ Class2   : num  0.996 0.321 0.889 0.265 0.984 ...
#>  $ predicted: Factor w/ 2 levels "Class1","Class2": 2 1 2 1 2 1 1 1 2 2 ...
```

The second and third columns are the predicted class probabilities for the test set while `predicted` are the discrete predictions.

For the hard class predictions, there are a variety of **yardstick** functions that are helpful:

```
# A confusion matrix:
conf_mat(two_class_example, truth = truth, estimate = predicted)
#>           Truth
#> Prediction Class1 Class2
#>     Class1    227     50
#>     Class2     31    192

accuracy(two_class_example, truth = truth, estimate = predicted)
#> # A tibble: 1 × 3
#>   .metric  .estimator .estimate
#>   <chr>    <chr>          <dbl>
#> 1 accuracy binary         0.838

# Matthews correlation coefficient:
mcc(two_class_example, truth, predicted)
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 mcc     binary         0.677

# F1 metric:
f_meas(two_class_example, truth, predicted)
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 f_meas  binary         0.849
```

For binary classification data sets, these functions have a standard argument called `event_level`. The *default* is that the **first** level of the outcome factor is the event of interest.

> There is some heterogeneity in R functions in this regard; some use the first level and others the second to denote the event of interest. We consider it more intuitive that the first level is the most important. The second level logic is borne of encoding the outcome as 0/1 (in which case the second value is the event) and unfortunately remains in some packages. However, tidymodels (along with many other R packages) *require* a categorical outcome to be encoded as a factor and, for this reason, the legacy justification for the second level as the event becomes irrelevant.

As an example where the second class is the event:

```
f_meas(two_class_example, truth, predicted, event_level = "second")
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 f_meas  binary         0.826
```

In the output above, the `.estimator` value of "binary" indicates that the standard formula for binary classes will be used.

There are numerous classification metrics that use the predicted probabilities as inputs rather than the hard class predictions. For example, the receiver operating characteristic (ROC) curve computes the sensitivity and specificity over a continuum of different event thresholds. The predicted class column is not used. There are two **yardstick** functions for this method: `roc_curve()` computes the data points that make up the ROC curve and `roc_auc()` computes the area under the curve.

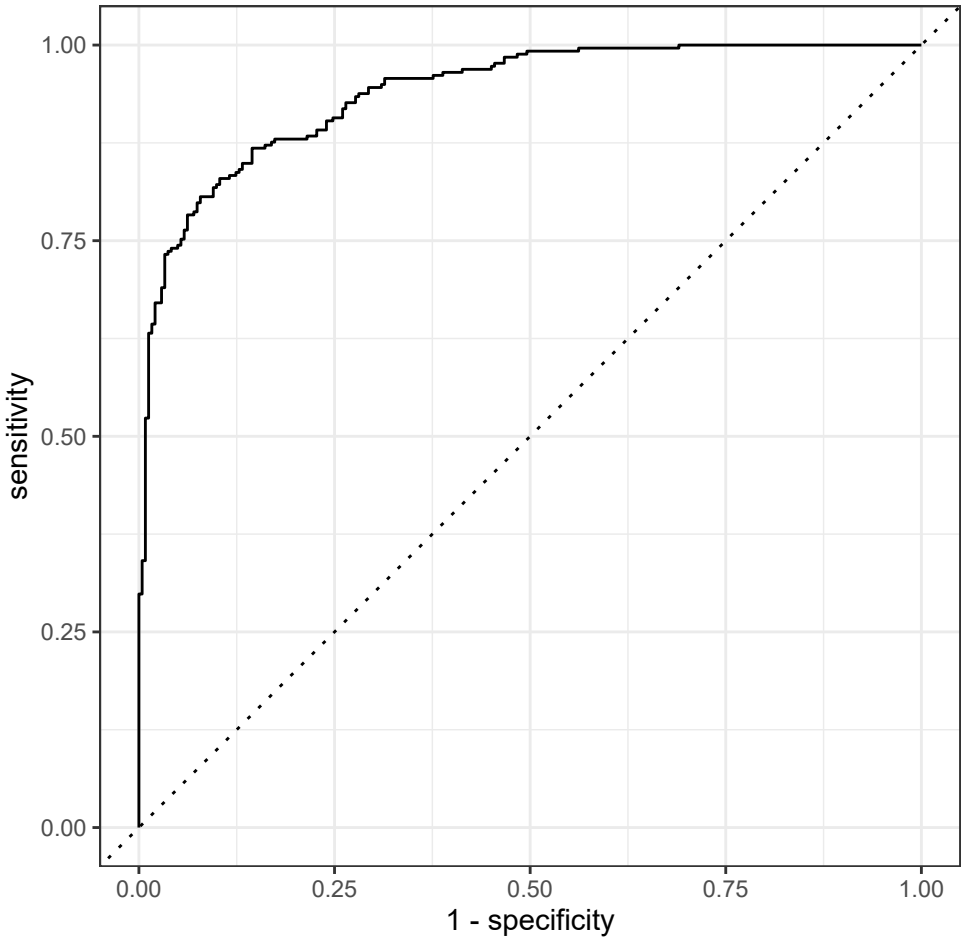
The interfaces to these types of metric functions use the `...` argument placeholder to pass in the appropriate class probability column. For two-class problems, the probability column for the event of interest is passed into the function:

```
two_class_curve <- roc_curve(two_class_example, truth, Class1)
two_class_curve
#> # A tibble: 502 × 3
#>   .threshold specificity sensitivity
#>        <dbl>       <dbl>       <dbl>
#> 1 -Inf           0                1
#> 2    1.79e-7     0                1
#> 3    4.50e-6     0.00413          1
#> 4    5.81e-6     0.00826          1
#> 5    5.92e-6     0.0124           1
#> 6    1.22e-5     0.0165           1
#> # … with 496 more rows

roc_auc(two_class_example, truth, Class1)
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 roc_auc binary         0.939
```

The `two_class_curve` object can be used in a `ggplot` call to visualize the curve. There is an `autoplot()` method that will take care of the details:

```
autoplot(two_class_curve)
```



There are a number of other functions that use probability estimates, including `gain_curve()`, `lift_curve()`, and `pr_curve()`.

9.4 MULTI-CLASS CLASSIFICATION METRICS

What about data with three or more classes? To demonstrate, let's explore a different example data set that has four classes:

```
data(hpc_cv)
str(hpc_cv)
#> 'data.frame':    3467 obs. of  7 variables:
#>  $ obs     : Factor w/ 4 levels "VF","F","M","L": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ pred    : Factor w/ 4 levels "VF","F","M","L": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ VF      : num  0.914 0.938 0.947 0.929 0.942 ...
#>  $ F       : num  0.0779 0.0571 0.0495 0.0653 0.0543 ...
#>  $ M       : num  0.00848 0.00482 0.00316 0.00579 0.00381 ...
#>  $ L       : num  1.99e-05 1.01e-05 5.00e-06 1.56e-05 7.29e-06 ...
#>  $ Resample: chr  "Fold01" "Fold01" "Fold01" "Fold01" ...
```

As before, there are factors for the observed and predicted outcomes along with four other columns of predicted probabilities for each class. These data also include a `Resample` column. These results are for out-of-sample predictions associated with 10-fold cross-validation (discussed in Chapter 10). For the time being, this column will be ignored.

The functions for metrics that use the discrete class predictions are identical to their binary counterparts:

```
accuracy(hpc_cv, obs, pred)
#> # A tibble: 1 × 3
#>   .metric  .estimator .estimate
#>   <chr>    <chr>          <dbl>
#> 1 accuracy multiclass     0.709

mcc(hpc_cv, obs, pred)
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 mcc     multiclass     0.515
```

Note that, in these results, a "multiclass" `.estimator` is listed. Like "binary", this indicates that the formula for outcomes with three or more class levels was used. The Matthews correlation coefficient was originally designed for two classes but has been extended to cases with more class

levels.

There are methods for using metrics that are specific to outcomes with two classes for data sets with more than two classes. For example, a metric such as sensitivity measures the true positive rate which, by definition, is specific to two classes (i.e., “event” and “non-event”). How can this metric be used in our example data?

There are wrapper methods that can be used to apply sensitivity to our four-class outcome. These options are macro-, macro-weighted, and micro-averaging:

- Macro-averaging computes a set of one-versus-all metrics using the standard two-class statistics. These are averaged.
- Macro-weighted averaging does the same but the average is weighted by the number of samples in each class.
- Micro-averaging computes the contribution for each class, aggregates them, then computes a single metric from the aggregates.

See Wu and Zhou (2017) and Opitz and Burst (2019).

Using sensitivity as an example, the usual two-class calculation is the ratio of the number of correctly predicted events divided by the number of true events. The “manual” calculations for these averaging methods are:

```
class_totals <-
  count(hpc_cv, obs, name = "totals") %>%
  mutate(class_wts = totals / sum(totals))
class_totals
#>   obs totals class_wts
#> 1  VF   1769   0.51024
#> 2   F   1078   0.31093
#> 3   M    412   0.11883
#> 4   L    208   0.05999

cell_counts <-
  hpc_cv %>%
  group_by(obs, pred) %>%
  count() %>%
  ungroup()

# Compute the four sensitivities using 1-vs-all
one_versus_all <-
  cell_counts %>%
  filter(obs == pred) %>%
  full_join(class_totals, by = "obs") %>%
  mutate(sens = n / totals)
one_versus_all
#> # A tibble: 4 × 6
#>   obs   pred      n totals class_wts  sens
#>   <fct> <fct> <int>  <int>     <dbl> <dbl>
#> 1 VF    VF     1620   1769    0.510  0.916
#> 2 F     F       647   1078    0.311  0.600
#> 3 M     M        79    412    0.119  0.192
#> 4 L     L       111    208    0.0600 0.534

# Three different estimates:
one_versus_all %>%
  summarize(
    macro = mean(sens),
```

```
    macro_wts = weighted.mean(sens, class_wts),
    micro = sum(n) / sum(totals)
  )
#> # A tibble: 1 × 3
#>   macro macro_wts micro
#>   <dbl>     <dbl> <dbl>
#> 1 0.560     0.709 0.709
```

Thankfully, there are easier methods for obtaining these results:

```
sensitivity(hpc_cv, obs, pred, estimator = "macro")
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 sens    macro          0.560
sensitivity(hpc_cv, obs, pred, estimator = "macro_weighted")
#> # A tibble: 1 × 3
#>   .metric .estimator     .estimate
#>   <chr>   <chr>              <dbl>
#> 1 sens    macro_weighted     0.709
sensitivity(hpc_cv, obs, pred, estimator = "micro")
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 sens    micro          0.709
```

For metrics using probability estimates, there are some metrics with multi-class analogs. For example, Hand and Till (2001) determined a multi-class technique for ROC curves. In this case, *all* of the class probability columns must be given to the function:

```
roc_auc(hpc_cv, obs, VF, F, M, L)
#> # A tibble: 1 × 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 roc_auc hand_till      0.829
```

Macro-averaging is also available:

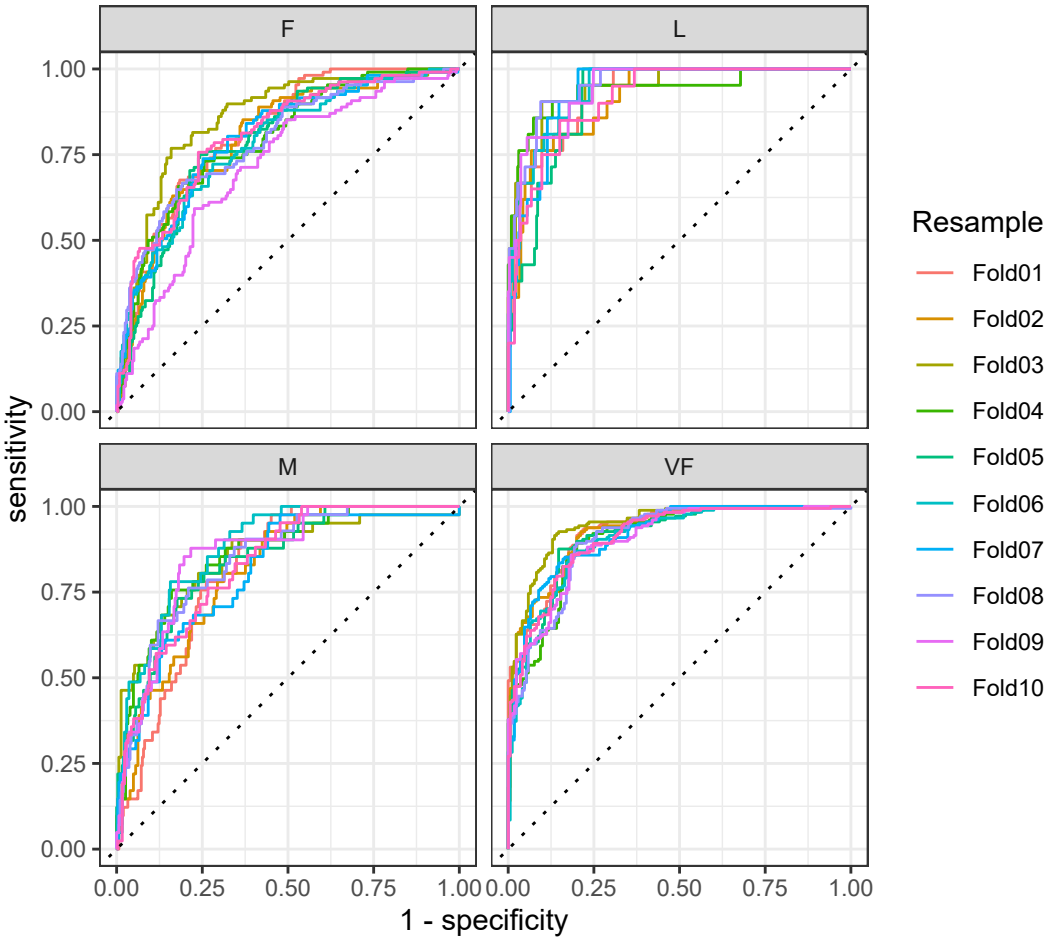
```
roc_auc(hpc_cv, obs, VF, F, M, L, estimator = "macro_weighted")
#> # A tibble: 1 × 3
#>   .metric .estimator     .estimate
#>   <chr>   <chr>              <dbl>
#> 1 roc_auc macro_weighted     0.868
```

Finally, all of these performance metrics can be computed using **dplyr** groupings. Recall that these data have a column for the resampling groups. Passing a grouped data frame to the metric function will compute the metrics for each group:

```
hpc_cv %>%
  group_by(Resample) %>%
  accuracy(obs, pred)
#> # A tibble: 10 × 4
#>   Resample .metric  .estimator .estimate
#>   <chr>    <chr>    <chr>          <dbl>
#> 1 Fold01   accuracy multiclass     0.726
#> 2 Fold02   accuracy multiclass     0.712
#> 3 Fold03   accuracy multiclass     0.758
#> 4 Fold04   accuracy multiclass     0.712
#> 5 Fold05   accuracy multiclass     0.712
#> 6 Fold06   accuracy multiclass     0.697
#> # … with 4 more rows
```

The groupings also translate to the `autoplot()` methods:

```
# Four 1-vs-all ROC curves for each fold
hpc_cv %>%
  group_by(Resample) %>%
  roc_curve(obs, VF, F, M, L) %>%
  autoplot()
```



This can be a quick visualization method for model effectiveness.

9.5 CHAPTER SUMMARY

Functions from the **yardstick** package measure the effectiveness of a model using data. The primary interface is based on data frames (as opposed to having vector arguments). There are a variety of regression and classification metrics and, within these, there are sometimes different estimators for the statistics.

REFERENCES

Craig–Schapiro, R, M Kuhn, C Xiong, E Pickering, J Liu, T Misko, R Perrin, et al. 2011. "Multiplexed Immunoassay Panel Identifies Novel CSF Biomarkers for Alzheimer's Disease Diagnosis and Prognosis." *PLoS ONE* 6 (4): e18850.

Hand, D, and R Till. 2001. "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." *Machine Learning* 45 (August): 171–86.

Hosmer, D, and Sy Lemeshow. 2000. *Applied Logistic Regression*. New York: John Wiley; Sons.

Jungsu, K, D Basak, and D Holtzman. 2009. "The Role of Apolipoprotein E in Alzheimer's Disease." *Neuron* 63 (3): 287–303.

Opitz, J, and S Burst. 2019. "Macro F1 and Macro F1." http://arxiv.org/abs/1911.03347.

Wu, X, and Z Zhou. 2017. "A Unified View of Multi-Label Performance Measures." In *International Conference on Machine Learning*, 3780–8.