Classification models

MODELING WITH TIDYMODELS IN R

David Svancer
Data Scientist

datacamp

Predicting product purchases

Classification models predict categorical outcome variables

- Predicting product purchases

purchased	total_time	total_visits
yes	800	3
yes	978	7
no	220	4
no	124	5
yes	641	4

Classification algorithms

Goal: Create distinct, non-overlapping regions along set of predictor variable values

- Predict the same categorical outcome in each region

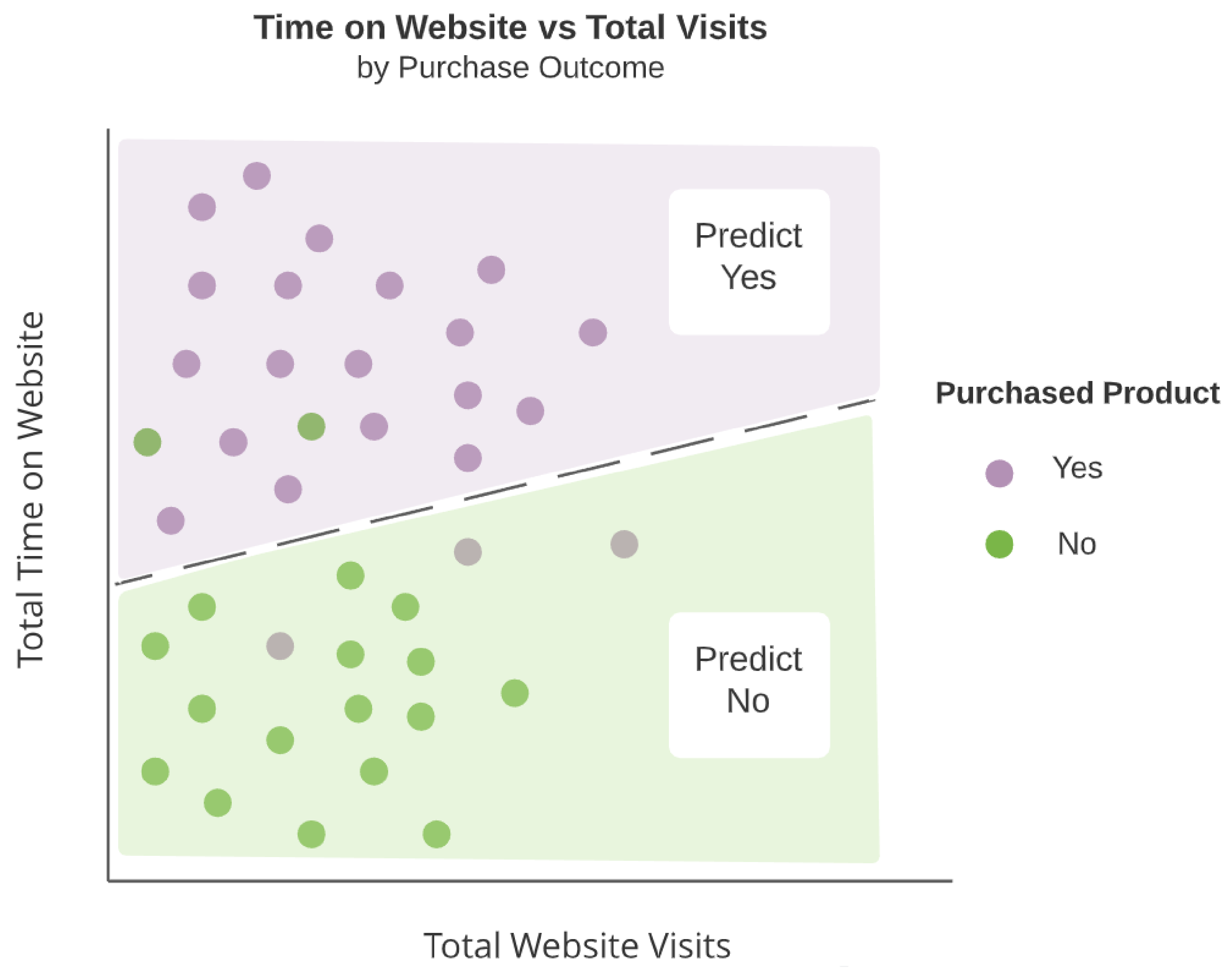
Classification algorithms

Goal: Create distinct, non-overlapping regions along set of predictor variable values

- Predict the same categorical outcome in each region

Logistic Regression

- Popular classification algorithm which creates a *linear* separation between outcome categories



Lead scoring data

```
leads_df
```

```
# A tibble: 1,328 x 7
   purchased total_visits total_time pages_per_visit total_clicks lead_source    us_location
   <fct>            <dbl>      <dbl>           <dbl>        <dbl> <fct>          <fct>
 1 yes                  7       1148            7              59 direct_traffic west
 2 no                   8        100            2.67           24 direct_traffic west
 3 no                   5        228            2.5            25 email          southeast
 4 no                   7        481            2.33           21 organic_search west
 5 no                   4        177            4              37 direct_traffic west
 6 no                   2       1273            2              26 email          midwest
 7 no                   3        711            3              28 organic_search west
 8 no                   3        166            3              32 direct_traffic southeast
 9 no                   3          7            3              23 organic_search west
10 no                   6        562            6              48 organic_search southeast
# ... with 1,318 more rows
```

Data resampling

First step in fitting a model

- Create data split object with `initial_split()`
- Create training and test datasets with `training()` and `testing()`

```
leads_split <- initial_split(leads_df,
                             prop = 0.75,
                             strata = purchased)

leads_training <- leads_split %>%
  training()

leads_test <- leads_split %>%
  testing()
```

Logistic regression model specification

Model specification in `parsnip`

- `logistic_reg()`
  - General interface to logistic regression models in `parsnip`
  - Common engine is 'glm'
  - Mode is 'classification'

```
logistic_model <- logistic_reg() %>%
  set_engine('glm') %>%
  set_mode('classification')
```

Model fitting

Once model is specified, the `fit()` function is used for model training

- Pass model object to `fit()`
- Specify model formula
- Provide training data, `data`

```
logistic_fit <- logistic_model %>%
  fit(purchased ~ total_visits + total_time,
      data = leads_training)
```

Predicting outcome categories

The `predict()` function

- `new_data` specifies dataset on which to predict new values
- `type`
  - 'class' provides categorical predictions

Standardized output from `predict()`

1. Returns a tibble
2. When `type` is 'class', returns a factor column named `.pred_class`

```
class_preds <- logistic_fit %>%
  predict(new_data = leads_test,
          type = 'class')

class_preds
```

```
# A tibble: 332 x 1
   .pred_class
   <fct>
 1 no
 2 yes
 3 no
 4 no
 5 yes
 # ... with 327 more rows
```

Estimated probabilities

Setting `type` to 'prob' provides estimated probabilities for each outcome category

```
prob_preds <- logistic_fit %>%
  predict(new_data = leads_test,
          type = 'prob')

prob_preds
```

The `predict()` function will return a tibble with multiple columns

- One for each category of the outcome variable
- Naming convention is `.pred_{outcome_category}`

```
# A tibble: 332 x 2
   .pred_yes .pred_no
       <dbl>    <dbl>
 1    0.134     0.866
 2    0.729     0.271
 3    0.133     0.867
 4    0.0916    0.908
 5    0.598     0.402
# ... with 327 more rows
```

Combining results

For model evaluation with the `yardstick` package, a results tibble will be needed

```
leads_results <- leads_test %>%
  select(purchased) %>%
  bind_cols(class_preds, prob_preds)
```

The outcome variable from the test dataset and prediction tibbles can be combined with `bind_cols()`

```
leads_results
```

```
# A tibble: 332 x 4
   purchased .pred_class .pred_yes .pred_no
   <fct>     <fct>           <dbl>    <dbl>
 1 no        no             0.134    0.866
 2 yes       yes            0.729    0.271
 3 no        no             0.133    0.867
 4 no        no             0.0916   0.908
 5 yes       yes            0.598    0.402
# ... with 327 more rows
```

Telecommunications data

```
telecom_df
```

```
# A tibble: 975 x 9
   canceled_service cellular_service avg_data_gb avg_call_mins avg_intl_mins internet_service   contract    months_with_company monthly_charges
   <fct>            <fct>                  <dbl>         <dbl>         <dbl>         <fct>        <fct>                   <dbl>           <dbl>
 1 yes              single_line            7.78            497           127         fiber_optic    month_to_month              7            76.4
 2 yes              single_line            9.04            336            88         fiber_optic    month_to_month             10            94.9
 3 no               single_line           10.3             262            55         fiber_optic    one_year                   50           103.
 4 yes              multiple_lines         5.08            250           107         digital        one_year                   53            60.0
 5 no               multiple_lines         8.05            328           122         digital        two_year                   50            75.2
 6 no               single_line            9.3             326           114         fiber_optic    month_to_month             25            95.7
 7 yes              multiple_lines         8.01            525            97         fiber_optic    month_to_month             19            83.6
 8 no               multiple_lines         9.4             312           147         fiber_optic    one_year                   50            99.4
 9 yes              single_line            5.29            417            96         digital        month_to_month              8            49.8
10 no               multiple_lines         9.96            340           136         fiber_optic    month_to_month             61           106.
# ... with 965 more rows
```

Let's practice!

MODELING WITH TIDYMODELS IN R

Assessing model fit

MODELING WITH TIDYMODELS IN R

David Svancer
Data Scientist

datacamp

Binary classification

Outcome variable with two levels

- **Positive class**
  - Event of interest to predict
  - "yes" in `purchased` variable
- **Negative class**
  - "no"
- In `tidymodels` outcome variable needs to be a factor
  - First level is positive class
  - Check order with `levels()`

```
leads_df
```

```
# A tibble: 1,328 x 7
  purchased total_visits  ...   us_location
   <fct>          <dbl>   ...      <fct>
 1 yes                7   ...      west
 2 no                 8   ...      west
 3 no                 5   ...      southeast
# ... with 1,325 more rows
```

```
levels(leads_df[['purchased']])
```

```
[1] "yes" "no"
```

Confusion matrix

Matrix with counts of all combinations of actual and predicted outcome values

Correct Predictions

- True Positive (TP)
- True Negative (TN)

Classification Errors

- False Positive (FP)
- False Negative (FN)

Predicted \ Truth	Positive (+)	Negative (-)
Positive (+)	TP	FP
Negative (-)	FN	TN

Classification metrics with yardstick

Creating confusion matrices and other model fit metrics with yardstick

- Requires a tibble of model results which contain:
  - True outcome values
    - `purchased`
  - Predicted outcome categories
    - `.pred_class`
  - Estimated probabilities of each category
    - `.pred_yes`
    - `.pred_no`

```
leads_results
```

```
# A tibble: 332 x 4
   purchased .pred_class .pred_yes .pred_no
   <fct>     <fct>           <dbl>    <dbl>
 1 no        no             0.134     0.866
 2 yes       yes            0.729     0.271
 3 no        no             0.133     0.867
 4 no        no             0.0916    0.908
 5 yes       yes            0.598     0.402
 6 no        no             0.128     0.872
 7 yes       no             0.112     0.888
 8 no        no             0.169     0.831
 9 no        no             0.158     0.842
10 yes       yes            0.520     0.480
# ... with 322 more rows
```

Confusion matrix with yardstick

The `conf_mat()` function

- Tibble of model results
- `truth` - column with true outcomes
- `estimate` - column with predicted outcomes

Logistic regression on `leads_df`

- Correctly classified 252 out of 332 customers (76%)
- 46 false negatives
- 34 false positives

```
conf_mat(leads_results,
         truth = purchased,
         estimate = .pred_class)
```

```
          Truth
Prediction yes  no
       yes  74  34
       no   46 178
```

Classification accuracy

The `accuracy()` function

- Takes same arguments as `conf_mat()`
- Calculates classification accuracy

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- `yardstick` functions always return a tibble
  - `.metric` - type of metric
  - `.estimate` - calculated value

```
accuracy(leads_results,
         truth = purchased,
         estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.759
```

Sensitivity

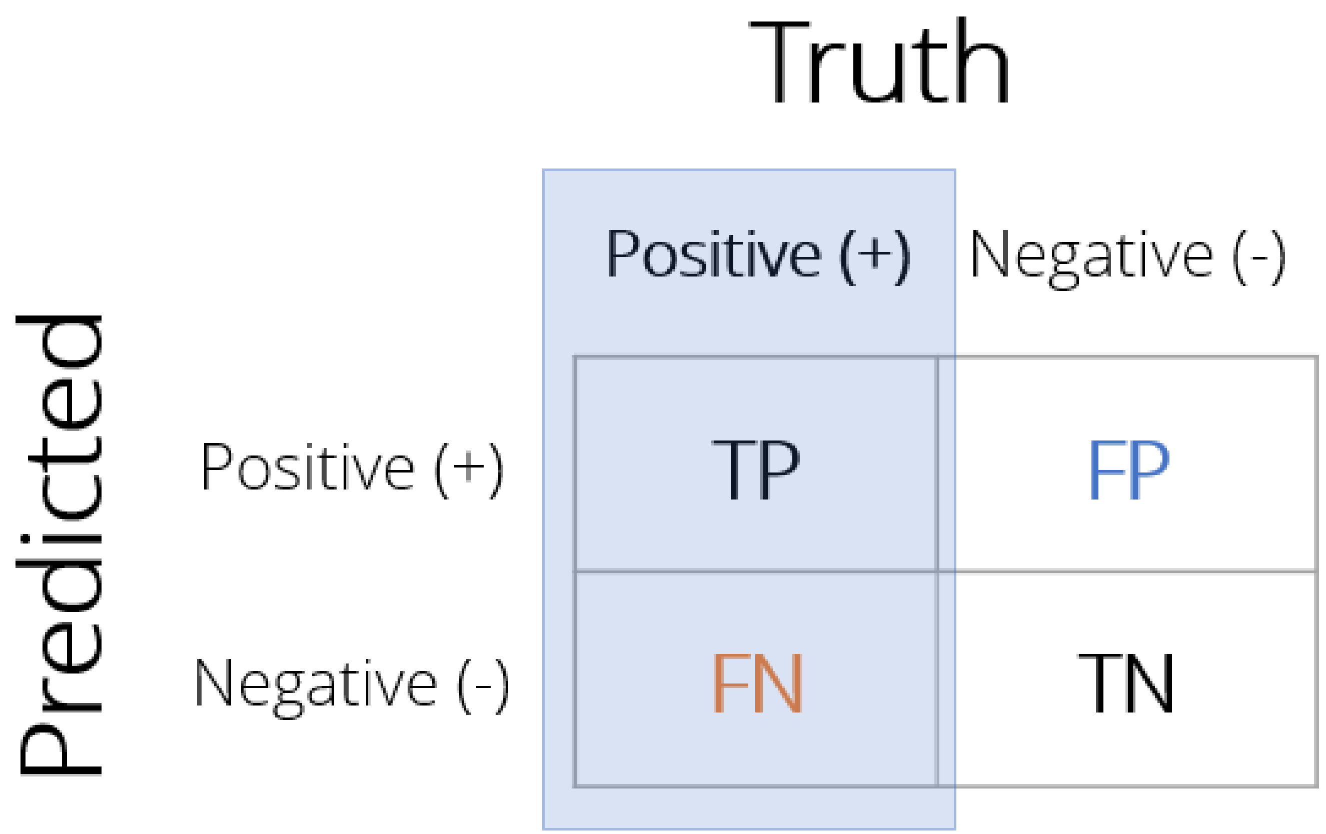
In many cases *accuracy* is not the best metric

- `leads_df` data
  - Classifying all as 'no' gives 64% accuracy

Sensitivity

Proportion of all positive cases that were correctly classified

- Of customers *who did purchase*, what proportion did our model predict correctly?
  - Lower false negatives increase sensitivity



$$\frac{TP}{TP + FN}$$

Calculating sensitivity

The `sens()` function

- Takes same arguments as `conf_mat()` and `accuracy()`
- Returns sensitivity calculation in `.estimate` column

```
sens(leads_results,
     truth = purchased,
     estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.617
```

Specificity

Specificity is the proportion of all negative cases that were correctly classified

- Of customers who *did not purchase*, what proportion did our model predict correctly?
  - Lower false positives increase specificity

1 - Specificity

- Also called the *false positive rate* (FPR)
- Proportion of false positives among true negatives

Truth

Predicted	Positive (+)	Negative (-)
Positive (+)	TP	FP
Negative (-)	FN	TN

$$\frac{TN}{TN + FP}$$

Calculating specificity

The `spec()` function

- Takes same arguments as `sens()`
- Returns specificity calculation in `.estimate` column

```
spec(leads_results,
     truth = purchased,
     estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.840
```

Creating a metric set

User-defined metric sets

- `metric_set()` function
  - Creates user-defined metric function with selected `yardstick` metrics
  - Pass `yardstick` metric function names into `metric_set()`
  - Use custom function to calculate metrics

```
custom_metrics <-
  metric_set(accuracy, sens, spec)
```

```
custom_metrics(leads_results,
               truth = purchased,
               estimate = .pred_class)
```

```
# A tibble: 3 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.759
2 sens     binary         0.617
3 spec     binary         0.840
```

Many metrics

Binary classification metrics

- Wide variety of binary classification metrics
  - `accuracy()`, `kap()`, `sens()`, `spec()`, `ppv()`, `npv()`, `mcc()`, `j_index()`, `bal_accuracy()`, `detection_prevalence()`, `precision()`, `recall()`, `f_meas()`
- Pass results of `conf_mat()` to `summary()` to calculate all

https://yardstick.tidymodels.org/reference

```
conf_mat(leads_results, truth = purchased,
         estimate = .pred_class) %>%
  summary()
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.759
 2 kap                  binary         0.466
 3 sens                 binary         0.617
 4 spec                 binary         0.840
 5 ppv                  binary         0.685
 6 npv                  binary         0.795
 7 mcc                  binary         0.468
 8 j_index              binary         0.456
 9 bal_accuracy         binary         0.728
10 detection_prevalence binary         0.325
11 precision            binary         0.685
12 recall               binary         0.617
13 f_meas               binary         0.649
```

Let's practice!

MODELING WITH TIDYMODELS IN R

Visualizing model performance

MODELING WITH TIDYMODELS IN R

David Svancer
Data Scientist

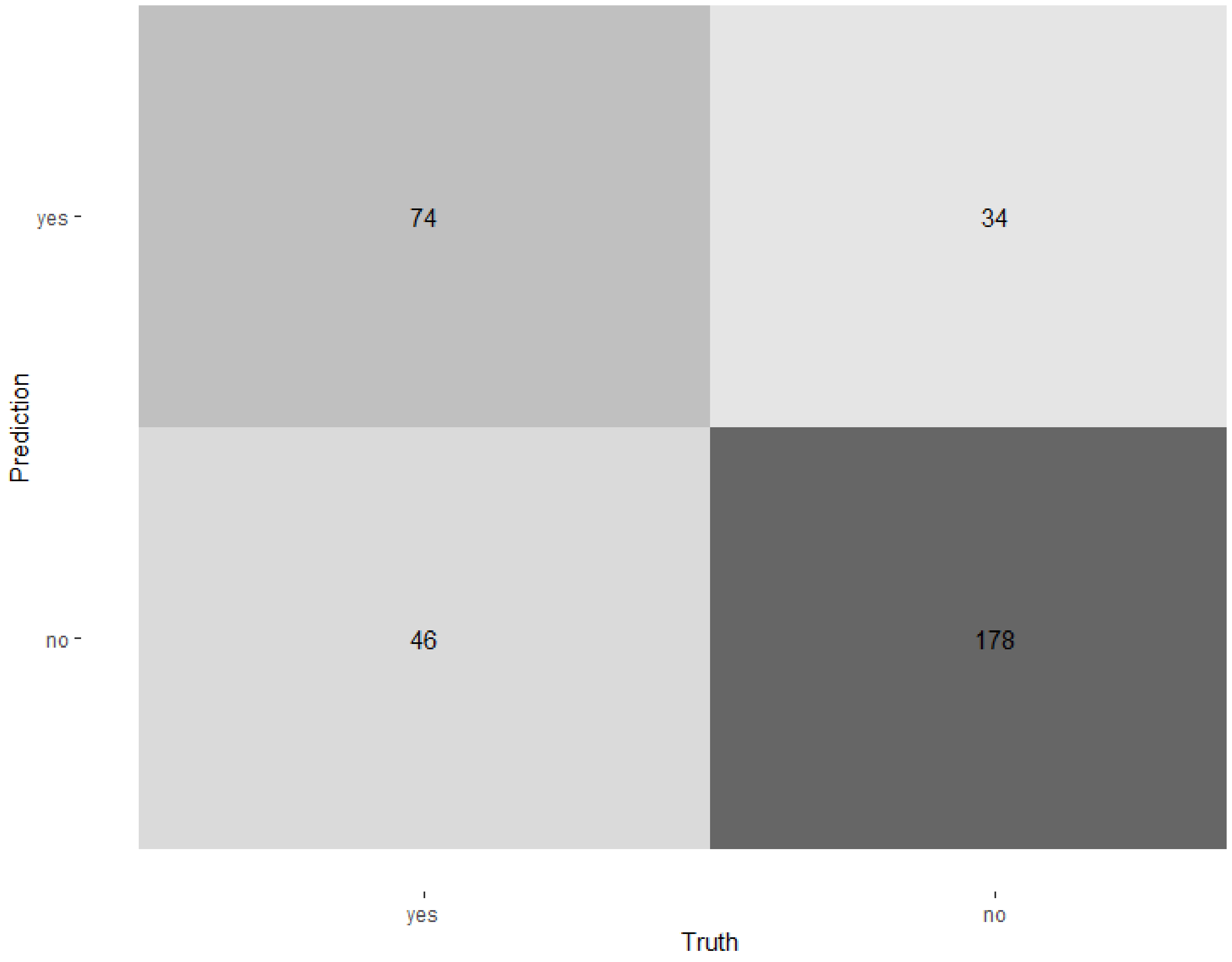
datacamp

Plotting the confusion matrix

Heatmap with `autoplot()`

- Pass confusion matrix object into `autoplot()`
- Set `type` to 'heatmap'
- Visualize the most prevalent counts

```
conf_mat(leads_results,
         truth = purchased,
         estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```

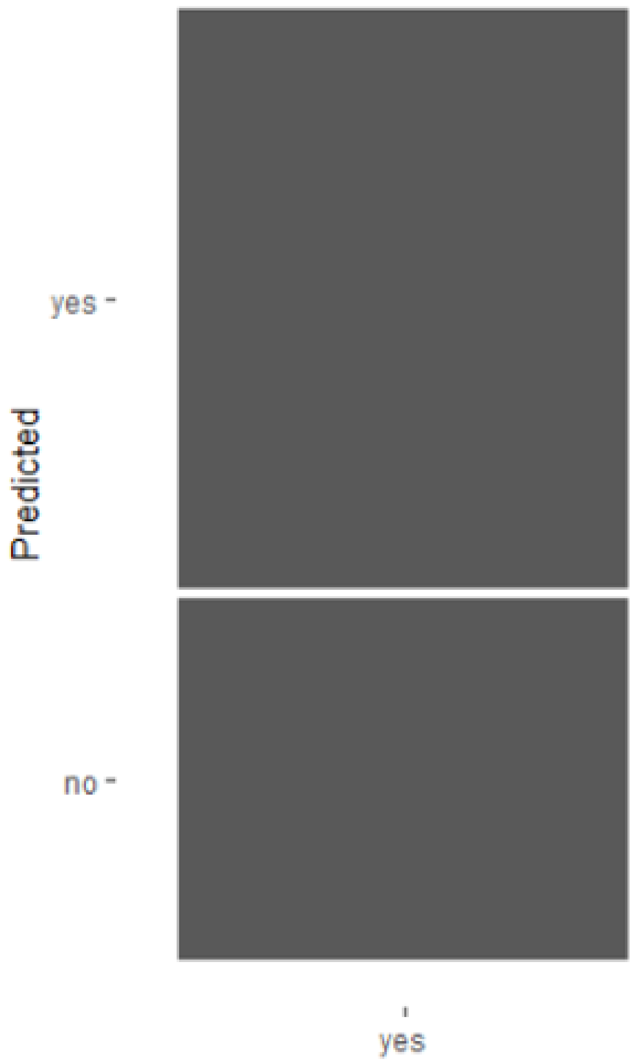


Mosaic plot

Mosaic with `autoplot()`

- Set `type` to 'mosaic'
- Each vertical bar represents 100% of actual outcome value in column
- Visually displays
  - sensitivity

```
conf_mat(leads_results,
         truth = purchased,
         estimate = .pred_class) %>%
  autoplot(type = 'mosaic')
```

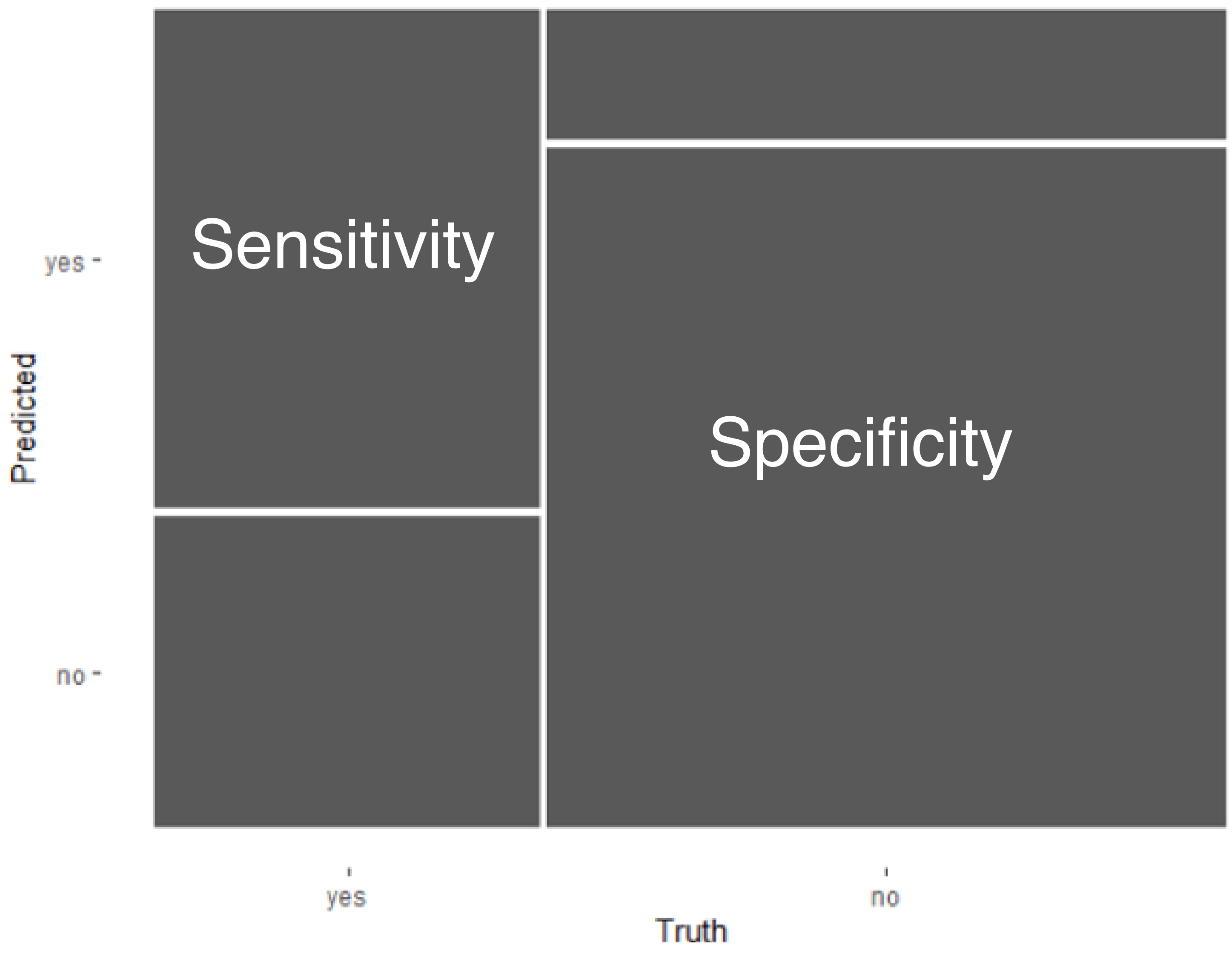


Mosiac plot

Mosaic with `autoplot()`

- Set `type` to 'mosaic'
- Each vertical bar represents 100% of actual outcome value in column
- Visually displays
  - sensitivity
  - specificity

```
conf_mat(leads_results,
         truth = purchased,
         estimate = .pred_class) %>%
  autoplot(type = 'mosaic')
```



Probability thresholds

Default probability threshold in binary classification is 0.5

- If the estimated probability of the positive class is greater than or equal to 0.5, the positive class is predicted

`leads_results`

- If `.pred_yes` is greater than or equal to 0.5 then `.pred_class` is set to 'yes' by the `predict()` function in `tidymodels`

```
leads_results
```

```
# A tibble: 332 x 4
   purchased .pred_class .pred_yes .pred_no
   <fct>     <fct>           <dbl>    <dbl>
 1 no        no             0.134     0.866
 2 yes       yes            0.729     0.271
 3 no        no             0.133     0.867
 4 no        no             0.0916    0.908
 5 yes       yes            0.598     0.402
 6 no        no             0.128     0.872
 7 yes       no             0.112     0.888
 8 no        no             0.169     0.831
 9 no        no             0.158     0.842
10 yes       yes            0.520     0.480
# ... with 322 more rows
```

Exploring performance across thresholds

How does a classification model perform across a range of thresholds?

- Unique probability thresholds in the `.pred_yes` column of the test dataset results
  - Calculate specificity and sensitivity for each

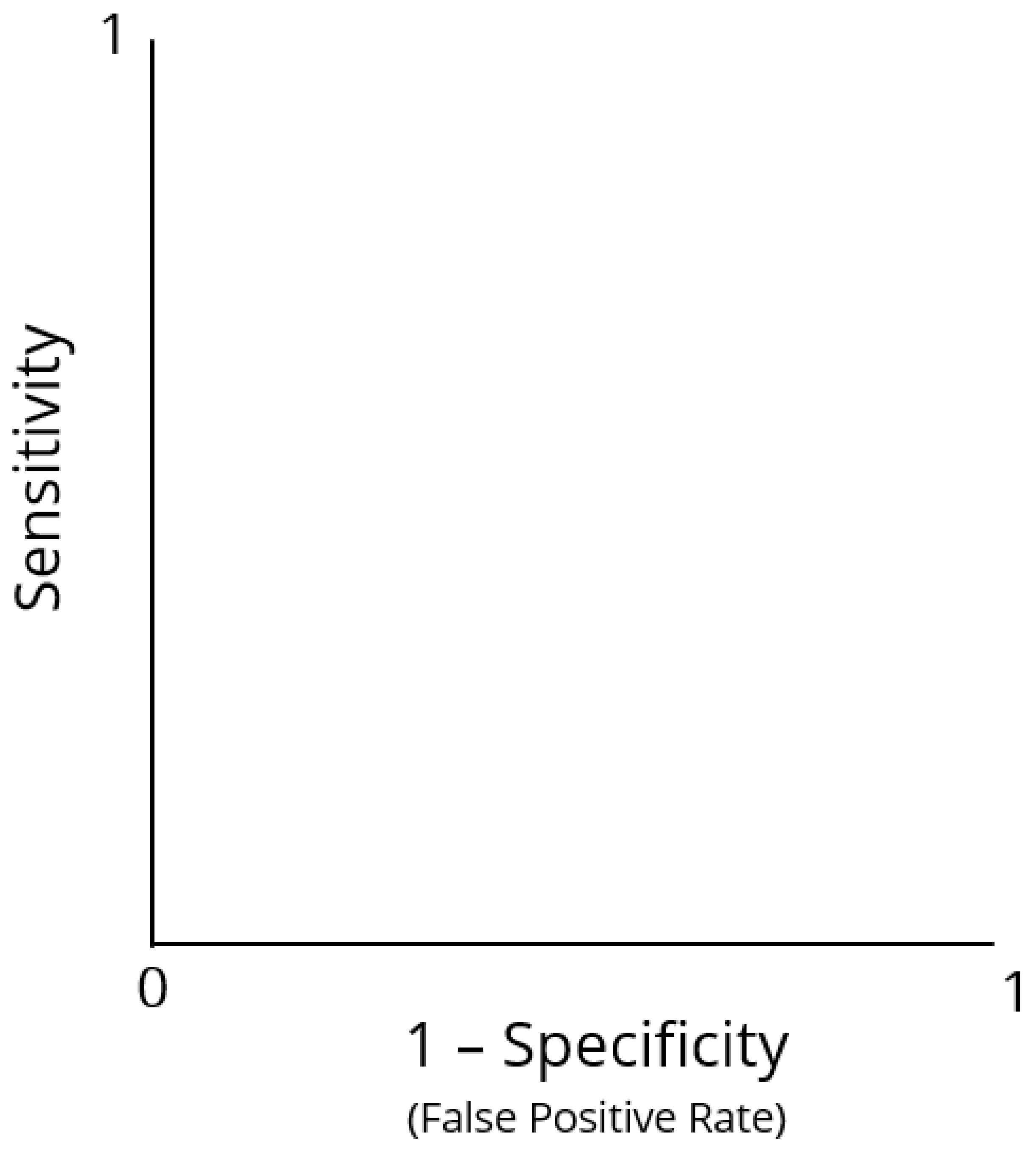
threshold	specificity	sensitivity
0	0	1
0.11	0.01	0.98
0.15	0.05	0.97
...	...	...
0.84	0.89	0.08
0.87	0.94	0.02
0.91	0.99	0
1	1	0

Visualizing performance across thresholds

Receiver operating characteristic (ROC) curve

- Used to visualize performance across probability thresholds

- Sensitivity vs. (1 - specificity) across unique thresholds in test set results

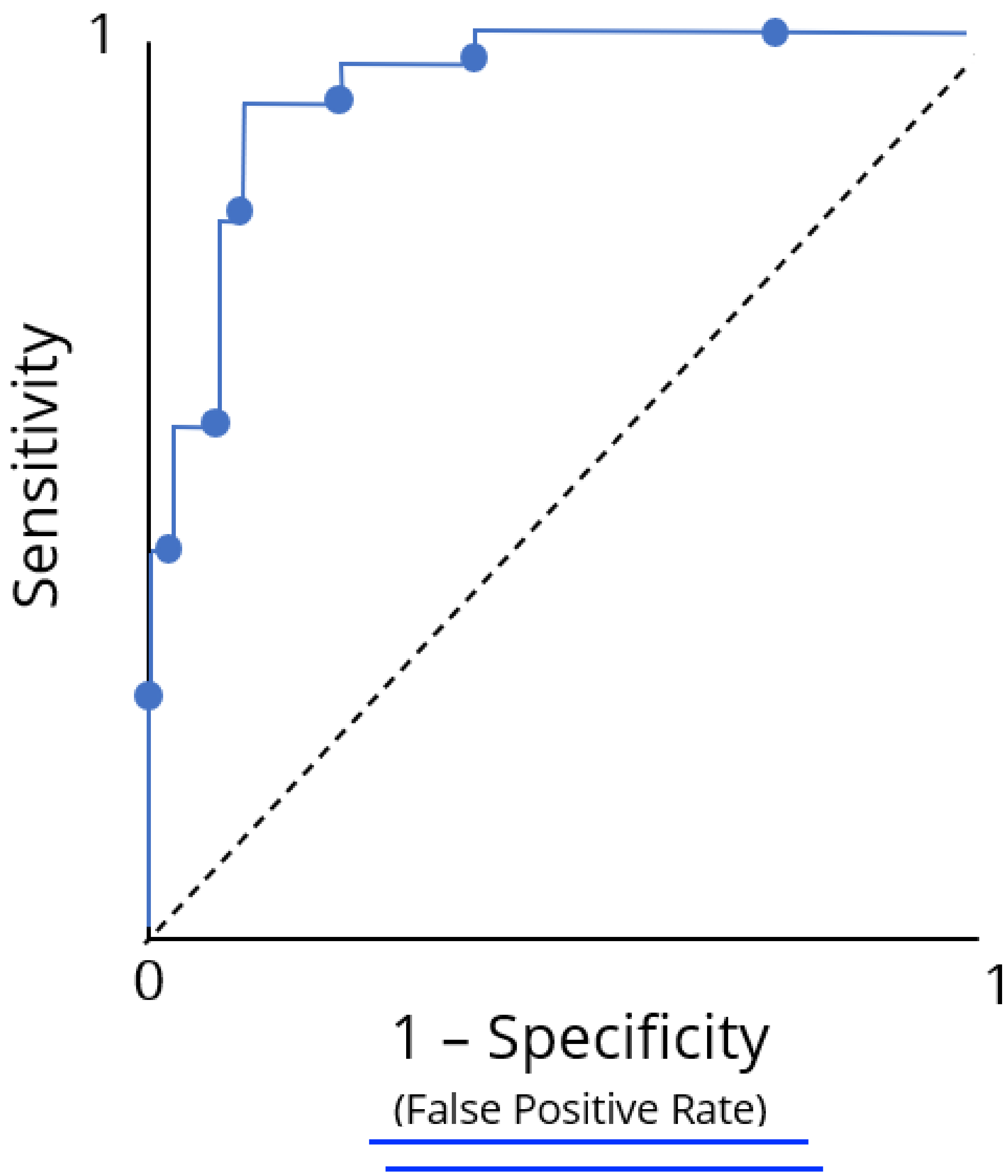


Visualizing performance across thresholds

Receiver operating characteristic (ROC) curve

- Used to visualize performance across probability thresholds

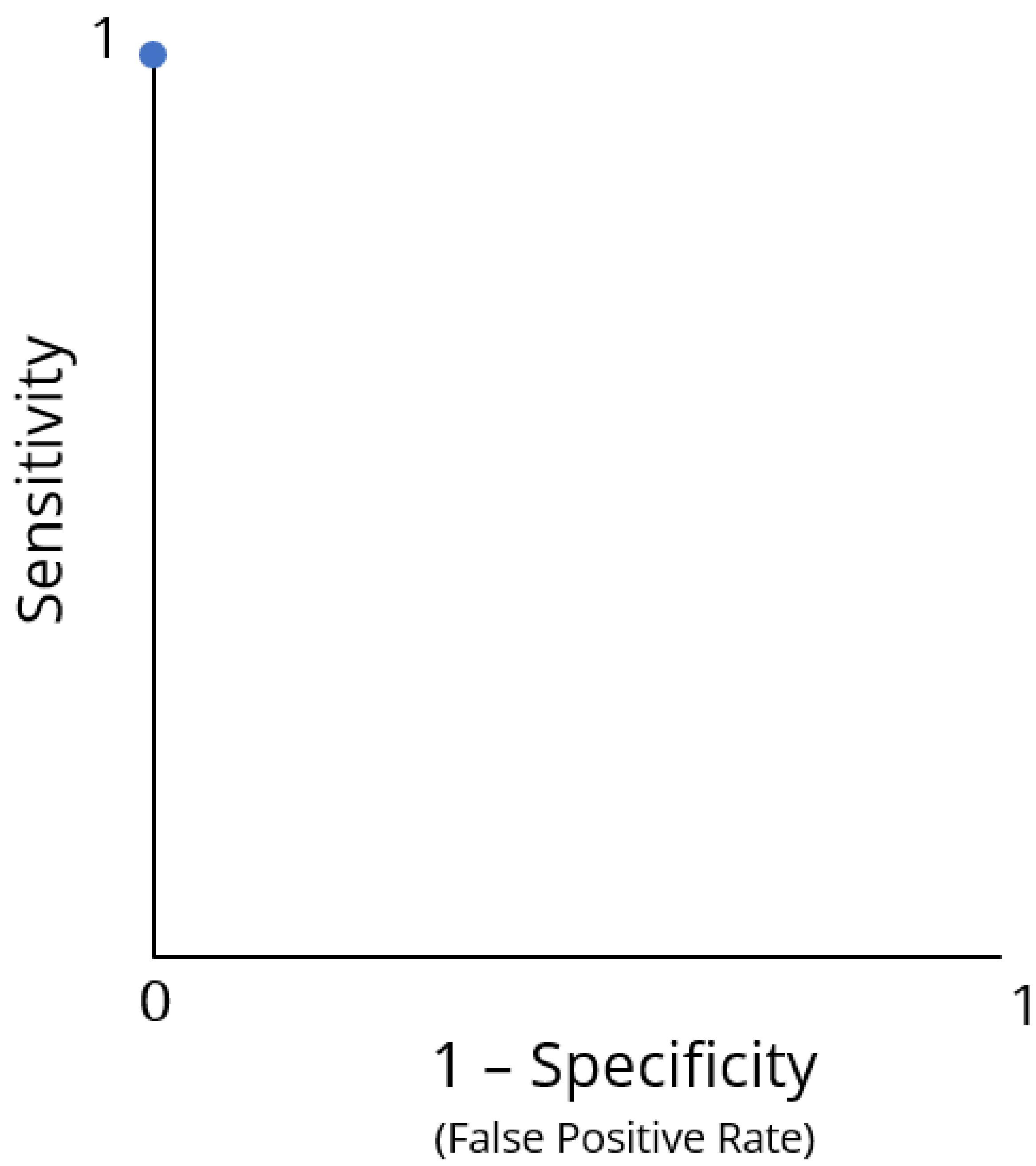
- Sensitivity vs (1 - specificity) across unique thresholds in test set results
    - Proportion correct among actual positives vs. proportion **incorrect** among actual negatives



ROC curves

Optimal performance is at the point (0, 1)

- Ideally, a classification model produces points close to left upper edge across all thresholds



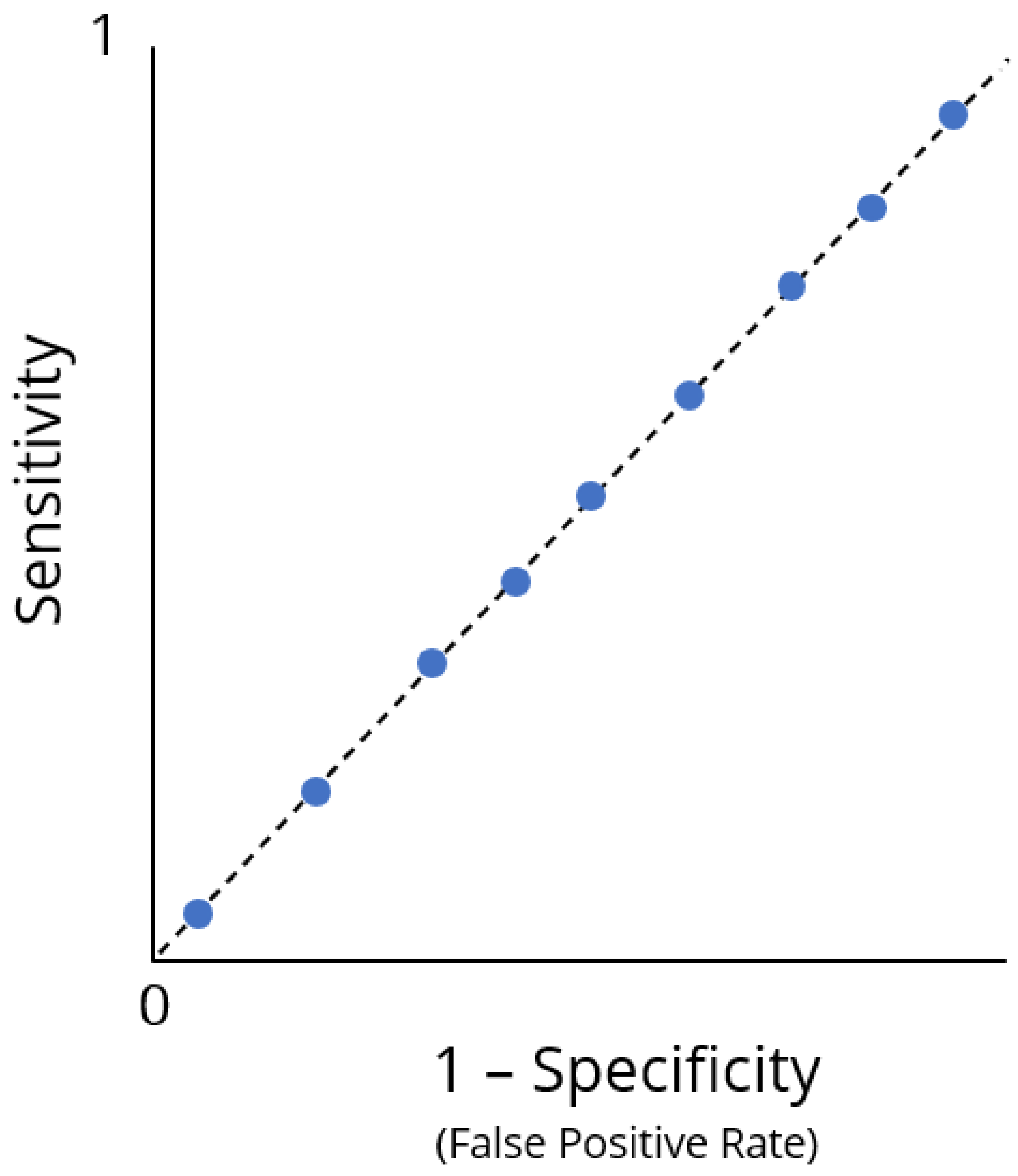
ROC curves

Optimal performance is at the point (0, 1)

- Ideally, a classification model produces points close to left upper edge across all thresholds

Poor performance

- Sensitivity and (1 - specificity) are equal across all thresholds
  - Corresponds to a classification model that predicts outcomes based on the result of randomly flipping a fair coin

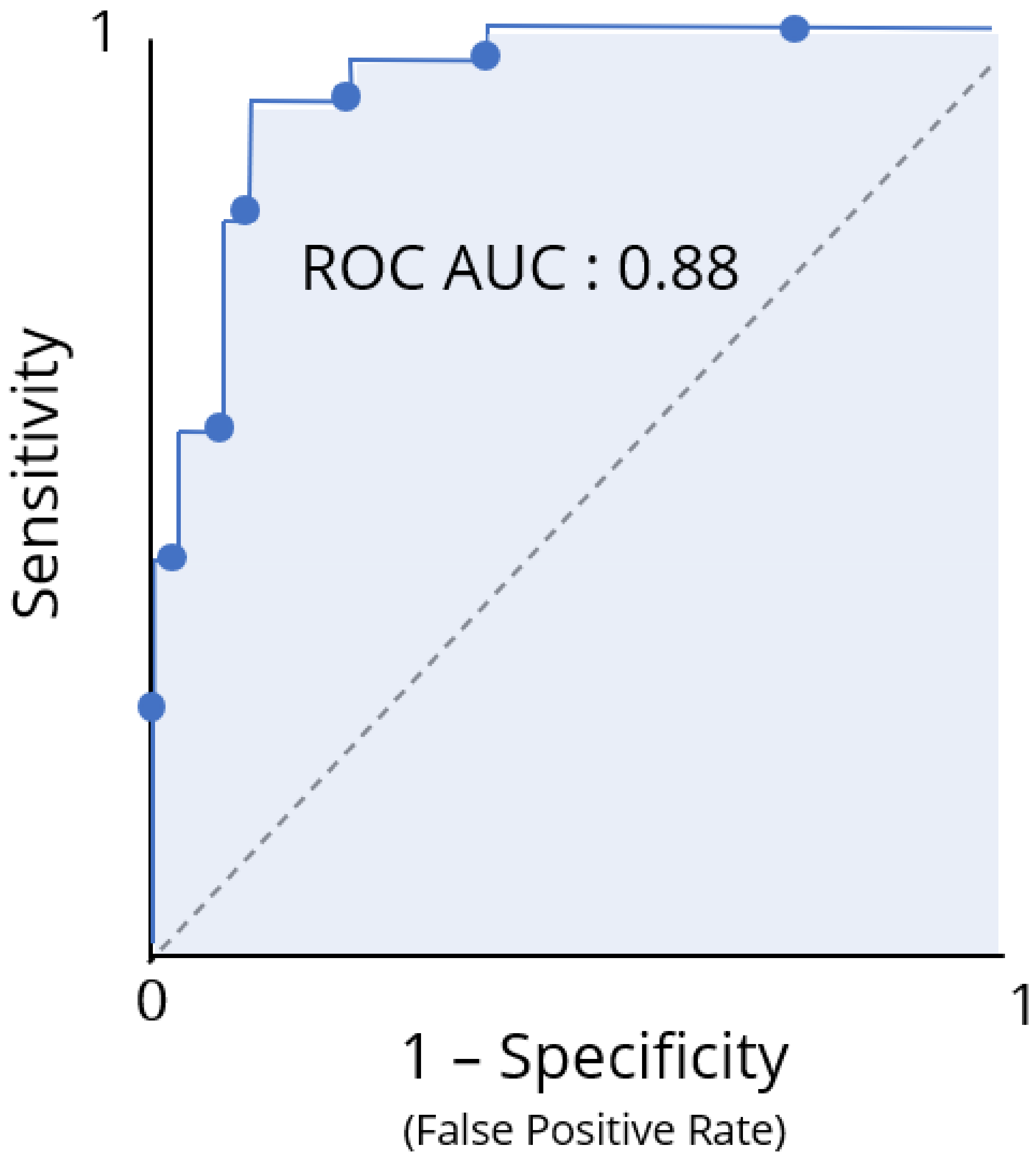


Summarizing the ROC curve

The area under the ROC curve (ROC AUC) captures the ROC curve information of a classification model in a single number

Useful interpretation as a letter grade of classification performance

- A - [0.9, 1]
- B - [0.8, 0.9)
- C - [0.7, 0.8)
- D - [0.6, 0.7)
- F - [0.5, 0.6)



Calculating performance across thresholds

The `roc_curve()` function

- Takes a results tibble as the first argument
- `truth` column with true outcome categories
- Column with estimated probabilities for the positive class
  - `.pred_yes` in `leads_results` tibble
- Returns a tibble with specificity and sensitivity for all unique thresholds in `.pred_yes`

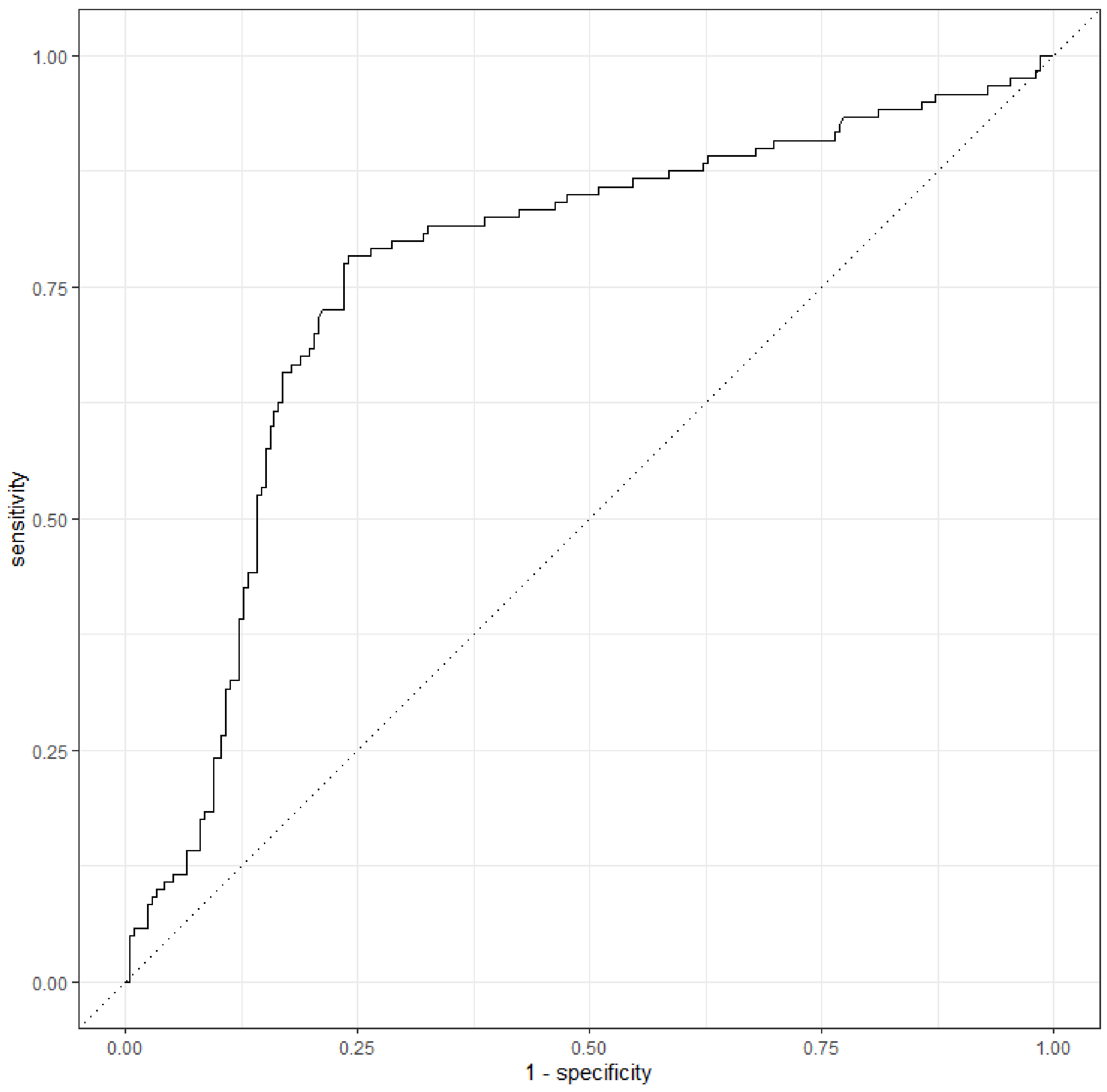
```
leads_results %>%
  roc_curve(truth = purchased, .pred_yes)
```

```
# A tibble: 331 x 3
   .threshold specificity sensitivity
        <dbl>       <dbl>       <dbl>
 1     -Inf       0           1
 2     0.0871     0           1
 3     0.0888     0.00472     1
 4     0.0893     0.00943     1
 5     0.0896     0.0142      1
 6     0.0902     0.0142      0.992
 7     0.0916     0.0142      0.983
 8     0.0944     0.0189      0.983
# ... with 323 more rows
```

Plotting the ROC curve

Passing the results of `roc_curve()` to the `autoplot()` function returns an ROC curve plot

```
leads_results %>%
  roc_curve(truth = purchased, .pred_yes) %>%
  autoplot()
```



Calculating ROC AUC

The `roc_auc()` function from `yardstick` will calculate the ROC AUC

- Tibble of model results
- `truth` column
- Column with estimated probabilities for the positive class

```
roc_auc(leads_results,
        truth = purchased,
        .pred_yes)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>      <chr>        <dbl>
1 roc_auc    binary       0.763
```

Let's practice!

Automating the modeling workflow

MODELING WITH TIDYMODELS IN R

David Svancer
Data Scientist

datacamp

Streamlining the workflow

The `last_fit()` function

- Also accepts classification models
- Speeds up the modeling process
- Fits the model to the training data and produces predictions on the test dataset

Similar to using `fit()`, the first steps include:

- Creating a data split object with `rsample`
- Specifying a model with `parsnip`

```
leads_split <- initial_split(leads_df,
                             strata = purchased)

logistic_model <- logistic_reg() %>%
  set_engine('glm') %>%
  set_mode('classification')
```

Fitting the model and collecting metrics

The `last_fit()` function

- `parsnip` model object
- Model formula
- Data split object

```
logistic_last_fit <- logistic_model %>%
  last_fit(purchased ~ total_visits + total_time,
           split = leads_split)

logistic_last_fit %>%
  collect_metrics()
```

The `collect_metrics()` function calculates metrics using the test dataset

- Accuracy and ROC AUC by default

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>      <chr>        <dbl>
1 accuracy   binary       0.759
2 roc_auc    binary       0.763
```

Collecting predictions

`collect_predictions()`

- Creates a tibble with all necessary columns for `yardstick` functions
- Actual and predicted outcomes with the test data
- Estimated probability columns for all outcome categories

```
last_fit_results <- logistic_last_fit %>%
  collect_predictions()
```

```
last_fit_results
```

```
# A tibble: 332 x 6
   id               .pred_yes .pred_no .row .pred_class purchased
   <chr>                <dbl>    <dbl> <int>    <fct>       <fct>
 1 train/test split    0.134    0.866     2      no          no
 2 train/test split    0.729    0.271    17      yes         yes
 3 train/test split    0.133    0.867    21      no          no
 4 train/test split    0.0916   0.908    22      no          no
 5 train/test split    0.598    0.402    24      yes         yes
# ... with 327 more rows
```

Custom metric sets

The `metric_set()` function

- `accuracy()`, `sens()`, and `spec()`
  - Require `truth` and `estimate` arguments
- `roc_auc()`
  - Requires `truth` and column of estimated probabilities

The `custom_metrics()` function will need all three, with `.pred_yes` as the last argument

```
custom_metrics <- metric_set(accuracy, sens,
                             spec, roc_auc)
```

```
custom_metrics(last_fit_results,
               truth = purchased,
               estimate = .pred_class,
               .pred_yes)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.759
2 sens     binary         0.617
3 spec     binary         0.840
4 roc_auc  binary         0.763
```

Let's practice!

MODELING WITH TIDYMODELS IN R