Predictive Modeling Basics

Applied Machine Learning in R
Pittsburgh Summer Methodology Series

Lecture 2-B July 20, 2021

Overview

Plan for Day 2-B

First will be a lecture on **Performance Metrics** (one of my favorite topics)

Second will be a **lecture** on training, evaluation, and interpretation in R

- We will **adapt familiar (statistical) algorithms** to predictive modeling
- *This will ease the transition to ML and highlight its similarities with classical statistics*
- We will also **foreshadow future topics** (e.g., regularized linear models and tuning)

Finally, we will have a Live Coding Activity and related Hands-on Activity

Performance Metrics

Metrics for Supervised Regression

- **Distance** between predicted and trusted values
- **Correlation** between predicted and trusted values

Metrics for Supervised Classification

- **Confusion matrix** between predicted and trusted classes
- Compare predicted **class probabilities** to trusted classes

Metrics for Regression

Classic Distance Metrics for Regression

Root Mean Squared Error (RMSE)

- Based on squared loss
- Penalizes severe errors harsher, Sensitive to outliers
- Ranges from 0 to $+\infty$, lower is better

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2}$$

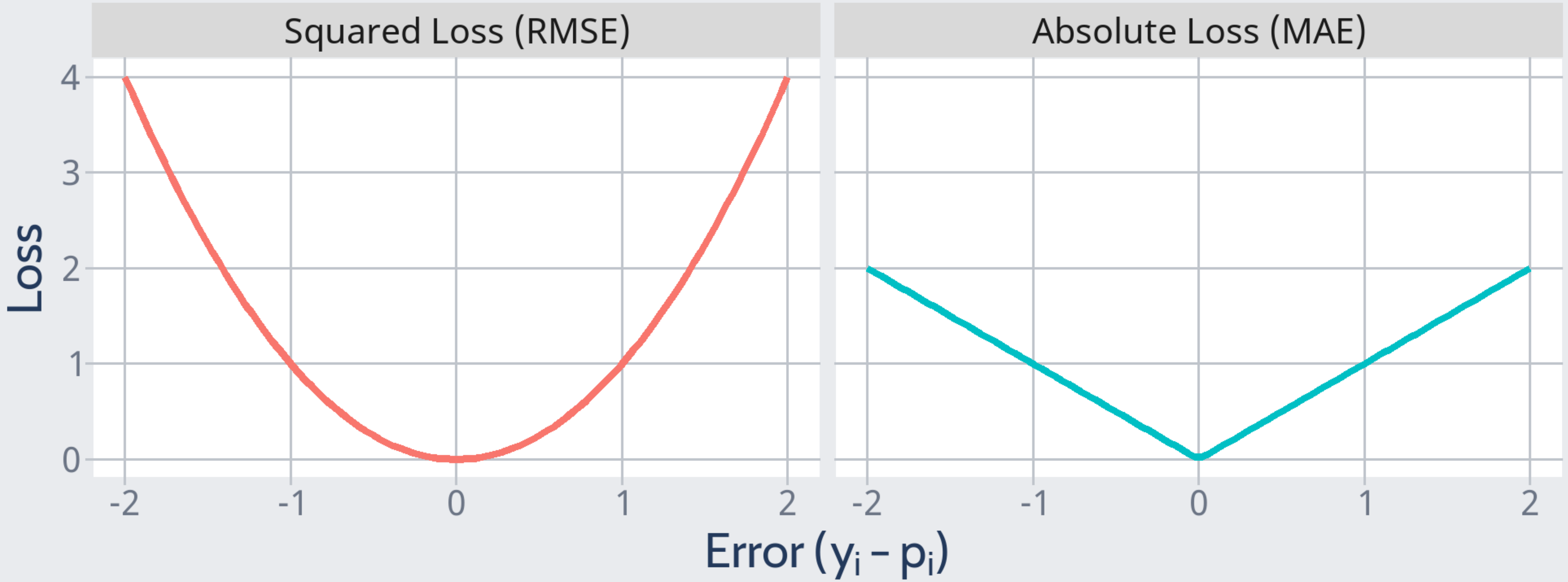
Mean Absolute Error (MAE)

- Based on absolute loss
- Penalizes error consistently, Robust to outliers
- Ranges from 0 to $+\infty$, lower is better

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - p_i|$$

[1] Note that, here, we will refer to the trusted labels as y and the predicted labels as p.

Visualizing Regression Loss Functions



Correlation Metrics for Regression

R-Squared $(R^2$ or RSQ)

- Calculated in ML as the **squared correlation** between the predictions and labels
- Ranges from 0 to 1, higher is better

$$R^2 = \left(\frac{\mathrm{cov}(y,p)}{\sigma_y \sigma_p}\right)^2 = \left(\frac{\sum(y_i - \bar{y})(p_i - \bar{p})}{\sqrt{\sum(y_i - \bar{y})^2}\sqrt{\sum(p_i - \bar{p})^2}}\right)^2$$

Caution!

- RSQ is a measure of *consistency* (i.e., linear association) and not distance
- RSQ can become unstable or undefined when data variability is low
- RSQ can become unstable when applied in small samples (e.g., test sets)

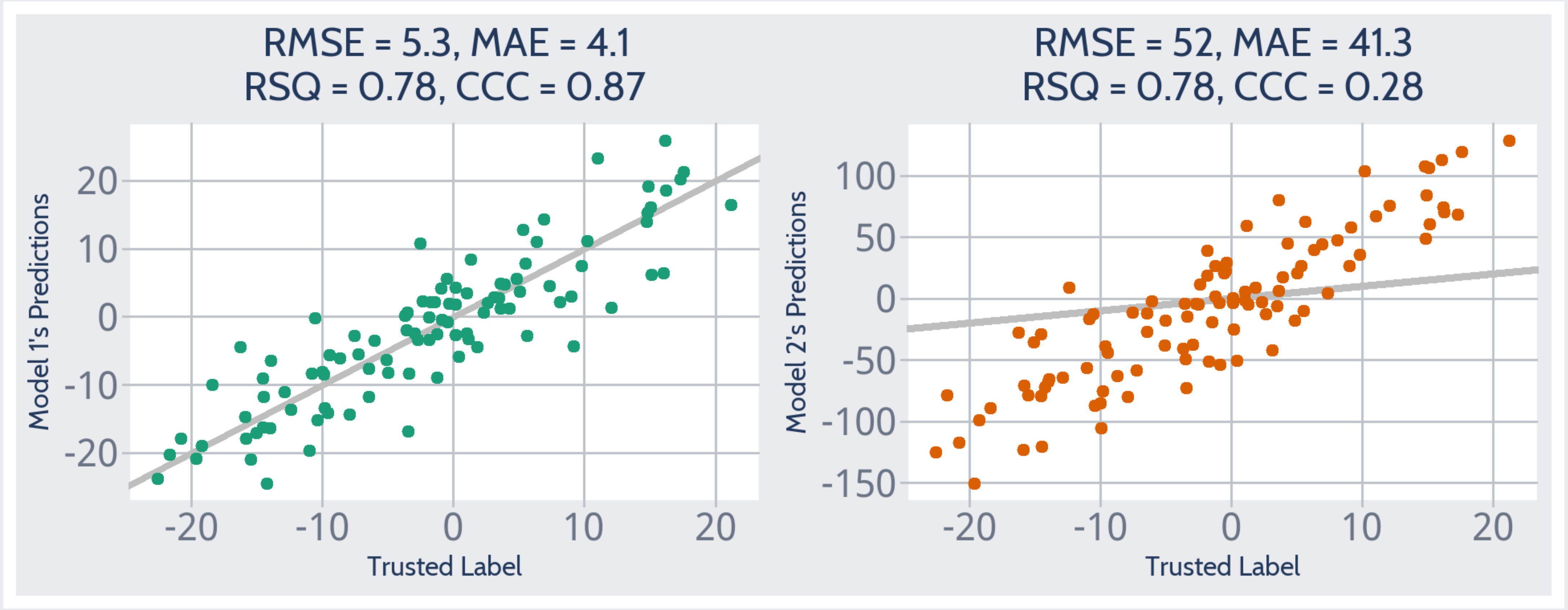
Advanced Correlation Metrics for Regression

Concordance Correlation Coefficient (CCC)

- Combines both accuracy (distance) and consistency (correlation) information
- Very similar to certain formulations of the intraclass correlation coefficient
- Ranges from -1 to $+1$, where higher is better

$$CCC = \frac{2\rho_{yp}\sigma_y\sigma_p}{\sigma_y^2 + \sigma_p^2 + (\mu_y - \mu_p)^2} = \frac{\frac{2}{n}\sum(y_i - \bar{y})(p_i - \bar{p})}{\frac{1}{n}\sum(y_i - \bar{y})^2 + \frac{1}{n}\sum(p_i - \bar{p})^2 + (\bar{y} - \bar{p})^2}$$

Comparing Regression Performance Metrics



Metrics for Classification

(Based on Predicted Classes)

Confusion Matrix Metrics

	Trusted = No $(y = 0)$	**Trusted = Yes** $(y = 1)$
Predicted = No $(p = 0)$	True Negatives (TN)	False Negative (FN)
Predicted = Yes $(p = 1)$	False Positive (FP)	True Positive (TP)

$$\text{Accuracy} = \frac{TN + TP}{n}$$

Ranges from 0 to 1, higher is better

Accuracy can be misleading when the classes are highly imbalanced (e.g., more 0s than 1s)

Confusion Matrix Metrics

With imbalanced classes, predicting the larger class will often be right "by chance"

To "correct" accuracy for imbalanced classes, **Cohen's Kappa** is often used

$$\kappa = \frac{\text{Accuracy} - \text{Chance}}{1 - \text{Chance}}$$

Chance agreement is estimated using the observed class probabilities from y and p

$$\text{Chance} = \Pr(y = 0) \cdot \Pr(p = 0) + \Pr(y = 1) \cdot \Pr(p = 1)$$

Kappa also ranges from 0 to 1 (technically -1 to 1), higher is better

Paradoxically, Kappa can be overly conservative when classes are very imbalanced.

Additional Confusion Matrix Metrics

	$y = 0$	$y = 1$
$p = 0$	True Negatives (TN)	False Negative (FN)
$p = 1$	False Positive (FP)	True Positive (TP)

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_1 \text{ Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

All range from 0 to 1, higher is better

F_1 does not consider TN, so only use it when detecting negatives is not important for your application.

Confusion Matrix Metrics Examples

Balanced	$y = 0$	$y = 1$
$p = 0$	101	54
$p = 1$	33	105

Imbalanced	$y = 0$	$y = 1$
$p = 0$	256	11
$p = 1$	6	2

$$\begin{aligned} \text{Accuracy} &= 0.70 \\ \kappa &= 0.41 \\ \\ \text{Sensitivity} &= 0.66 \\ \text{Specificity} &= 0.75 \\ \text{Balanced Accuracy} &= 0.71 \\ \\ \text{Precision} &= 0.76 \\ \text{Recall} &= 0.66 \\ F_1 &= 0.71 \end{aligned}$$

$$\begin{aligned} \text{Accuracy} &= 0.94 \\ \kappa &= 0.16 \\ \\ \text{Sensitivity} &= 0.15 \\ \text{Specificity} &= 0.98 \\ \text{Balanced Accuracy} &= 0.57 \\ \\ \text{Precision} &= 0.25 \\ \text{Recall} &= 0.15 \\ F_1 &= 0.19 \end{aligned}$$

Multiclass Performance Strategies

With more than two classes, you can make a larger (e.g., 3×3) confusion matrix)

	y = **Healthy**	y = **Depression**	y = **Mania**
p = **Healthy**	100	3	7
p = **Depression**	30	25	20
p = **Mania**	10	1	10

Macro-averaging: compute the standard binary metric for each class separately (using a one-vs-rest procedure) and then calculate the average metric score across classes

Micro-averaging: compute a confusion matrix for each class separately (one-vs-rest), add these matrices together, and calculate the binary metric from the summed matrix

Metrics for Classification

(Based on Class Probabilities)

Class Probability Metrics for Classification

Some classifiers estimate **the probability of each class** as their prediction (p_{ij})

If we consider a higher estimated probability as higher "confidence"

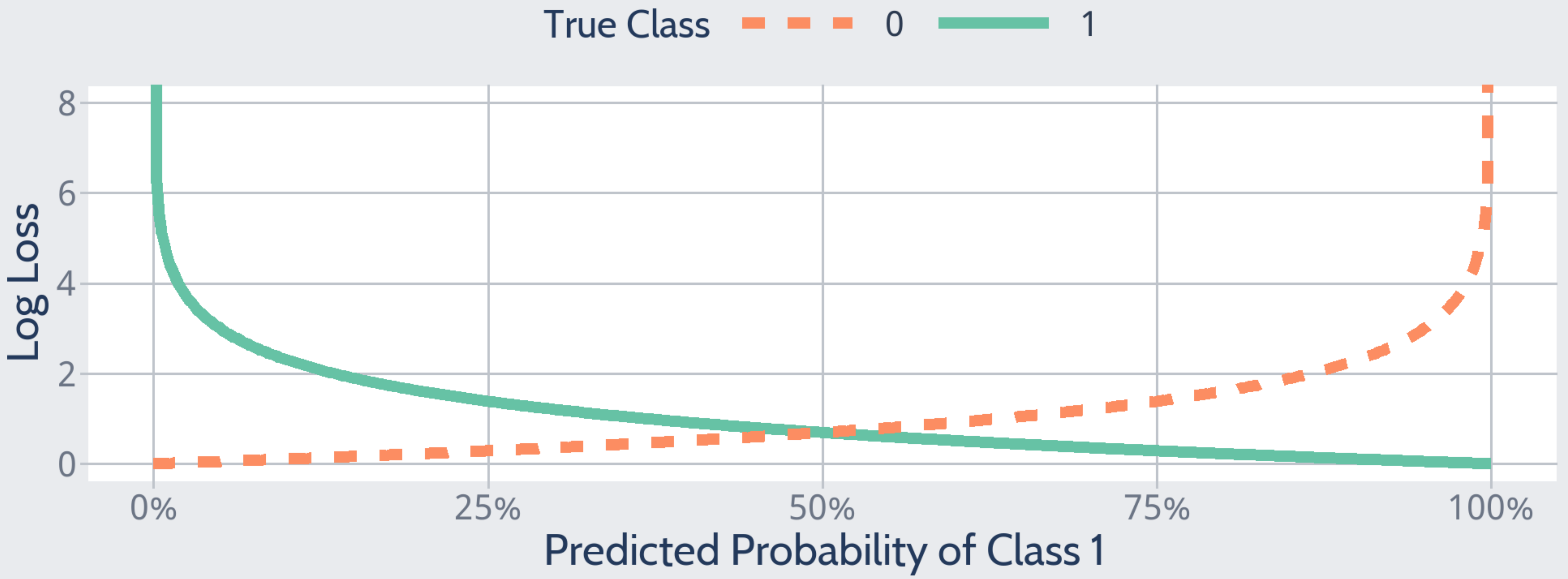
- We can **reward** the classifier for being more **confident when correct**...
- ...and **penalize** the classifier for being more **confident when wrong**

This gives rise to the Logistic or **Log Loss**, which can be summed or averaged

$$L_{log}(Y, P) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{q}\left(y_{ij}\log(p_{ij})\right)$$

$y_{ij} \in \{0, 1\}$ is a binary indicator of whether observation i is truly in class j
$p_{ij} \in (0, 1)$ is the estimated probability that observation i is in class j

Visualizing Log Loss in Binary Classification



Performance Curves

- When a classifier outputs class probabilities, we can choose any **decision threshold**
- We might naturally consider any probability over 50% positive and all others negative
- But we could choose a threshold more conservative (e.g., 75%) or liberal (e.g., 25%)

- **Performance curves** plot the characteristics of different decision thresholds
- This gives us an overview of how the classification system performs in general

- There are many performance curves[1], so we'll use the original as an example
- Finally, the **area under the curve (AUC)** is often used as a performance metric

[1] Popular options include ROC curves, precision-recall curves, gain curves, and lift curves.

Receiver Operating Characteristic (ROC) Curves

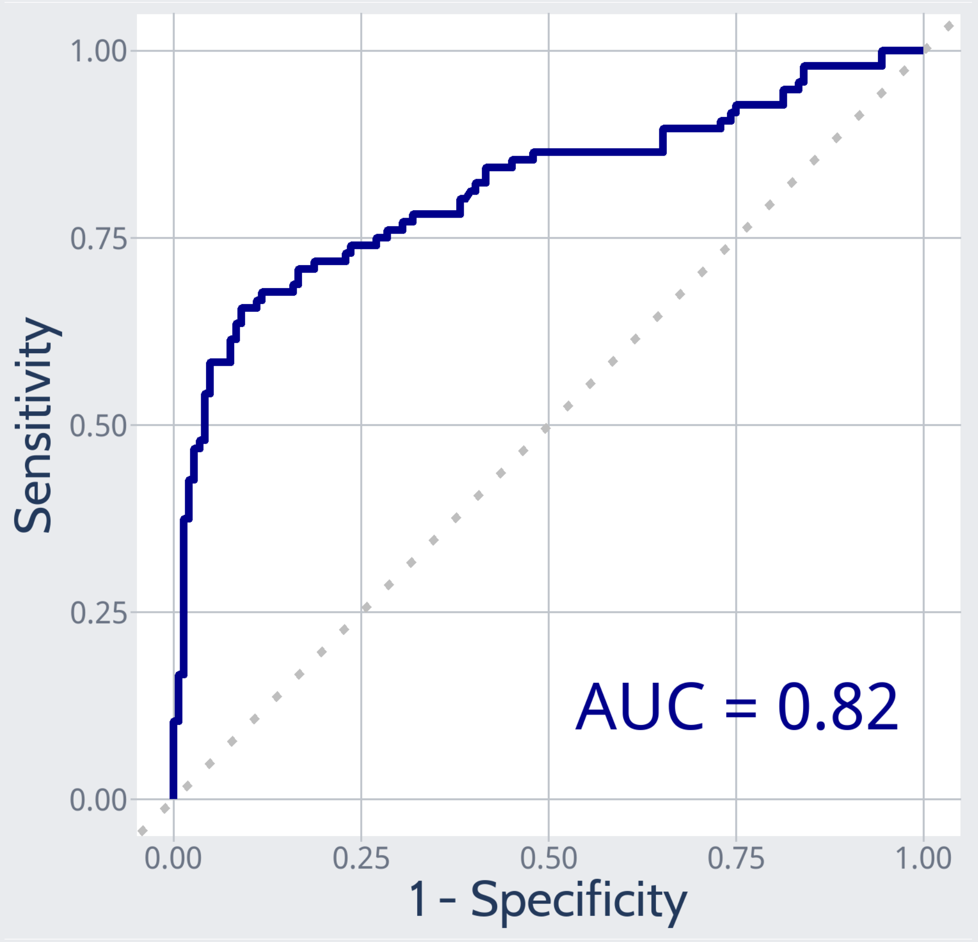
Each point in a ROC curve corresponds to a possible decision threshold

The performance metrics compared for each point are Sensitivity and Specificity

Better curves are closer to the top-left

The area under the ROC curve (AUC-ROC) ranges from 0.5 to 1.0, higher is better.

AUCROC is the probability that a random positive example has a higher estimate than a random negative example.



Comprehension Check #1

Bindi trains Model [A] to predict how many kilometers each bird will migrate this year and Model [B] to predict whether or not it will reproduce this year.

1. Which combination of performance metrics would be appropriate to use?

a) Log Loss for [A] and CCC for [B]

b) Precision for [A] and Recall for [B]

c) MAE for [A] and Balanced Accuracy for [B]

d) None of the above

2. Which combination of performance scores should Bindi hope to see?

a) RMSE = 531.6 and AUC-ROC = 0.04

b) RMSE = 1129.7 and AUC-ROC = 0.04

c) RMSE = 531.6 and AUC-ROC = 0.88

d) RMSE = 1129.7 and AUC-ROC = 0.88

Training and Cross-Validation

`caret::train()`

{caret} standardizes the syntax to train over 200 different ML algorithms

It also plays nicely with the {recipes} package we learned for preprocessing

The train() function will handle model **training**, **resampling**, and **tuning**

Because LM and GLM have no hyperparameters, we don't need tuning (yet!)

Today, we will **focus on training** and just use resampling to estimate performance

We will point to where tuning would be configured but leave that for tomorrow

Main Arguments to `train()`

Argument	Description
x	A {recipe} object with variable roles and preprocessing steps
data	A data frame to be used for training (prior to prep and bake)
method	A string indicating the algorithm to train (e.g., "lm" or "glm")

To train any of the 200+ supported algorithms, you just need to change `method`

This makes it *super easy* to implement new algorithms and explore the world of ML!

Be sure you understand an algorithm before trying to publish a paper using it.

Additional Arguments to `train()`

Argument	**Description**
trControl	Controls the resampling procedure used during tuning
metric	Controls which metric to optimize during tuning
tuneGrid	Control which specific tuning values to compare
tuneLength	Control how many tuning values to automatically compare

These arguments are largely used to configure tuning (discussed tomorrow)

The training set will be resampled and different tuning values will be compared

The "best" tuning values will be used to train a **final model** using all the training data

Resampling options

The `trControl` argument can be configured by `trainControl()`:

Argument	**Description**
method	Controls the type of resampling (e.g., "cv", "repeatedcv", "boot")
number	Controls the number of folds in cv and iterations in boot
repeats	Controls the number of repetitions in repeatedcv

"cv" will perform resampling through k-fold cross-validation

"repeatedcv" will repeat k-fold cross-validation multiple times

"boot" will perform resampling through bootstrapping

Pseudo-code for training with resampling

```r
# Configure resampling options
resampling_options <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3
)

# Train model from recipe
trained_model <- train(
  x = my_recipe,
  data = my_training,
  method = "lm",
  trControl = resampling_options
)
```

Comprehension Check #2

```r
# Part 1
my_recipe <-
  my_data %>%
  recipe(outcome ~ .) %>%
  step_center(all_numeric()) %>%
  step_zv(all_predictors()) %>%
  prep(training = my_training)

# Part 2
trained_model <- train(
  x = my_recipe,
  data = my_training_set,
  method = "cv"
)
```

1. What was the main mistake made in Part 1?

a) The predictors should be listed in `recipe()`

b) Numeric predictors cannot be centered

c) `step_zv()` only works for numeric predictors

d) The recipe should not be prepped yet

2. What was the main mistake made in Part 2?

a) `data` should be `my_testing_set`

b) `x` should be `my_data`

c) "cv" is not a method for `train()`

d) Forgot to add the `number` argument

Applied Example

Let's put what we just learned into practice in R

Let's use {caret} and {recipes} to train a regression model on the `titanic` data

- We will load in and split the data
- We will create a recipe for feature engineering
- We will train LM to predict each passenger's fare (how much they paid)

Applied Example

```r
# Read in data
titanic <- read_csv("titanic.csv")

# Create a training and testing set (stratified by fare)
set.seed(2021)
fare_index <- createDataPartition(titanic$fare, p = 0.75, list = FALSE)
fare_train <- titanic[fare_index, ]
fare_test <- titanic[-fare_index, ]
```

```r
# Check sizes
dim(fare_train)
#> [1] 723   7
dim(fare_test)
#> [1] 240   7
```

Applied Example

```r
# Create a preprocessing recipe (don't prep or bake)
fare_recipe <-
  titanic %>%
  recipe(fare ~ .) %>%
  step_rm(survived) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_nzv(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())
```

Applied Example

```r
set.seed(2021)

# Configure resampling
fare_tc <- trainControl(method = "cv", number = 10)

# Train the model using the recipe, data, and method
fare_lm <- train(
  fare_recipe,
  data = fare_train,
  method = "lm",
  trControl = fare_tc
)
```

Model Evaluation

Training Set Performance

The object created by `train()` will contain lots of information

We can view a summary of training set performance in the `$results` field

- Each row corresponds to one combination or set of hyperparameters
- Columns define the **hyperparameter values** and the set's **performance scores**
- Results will include the Mean and SD of each metric across resamples

```
fare_lm$results
```

intercept	RMSE	Rsquared	MAE	RMSESD	RsquaredSD	MAESD
TRUE	41.31	0.48	20.96	12.84	0.1	2.83

[1] Because there are no real hyperparameters for LM, there is only one row in `results`.

Test Set Predictions

To evaluate performance on the test set, we need two things:

1. Predictions from the model on the test set (e.g., values, classes, or probabilities)
2. Trusted labels on the test set (after any preprocessing steps from {recipes})

We can compare these predicted and trusted labels using {caret} or {yardstick}

- {caret} has basic performance metrics built in and is straightforward to use
- {yardstick} offers many more options but requires some coding to work with {caret}

Test Set Predictions

To get predictions from the final model on new data[1], we can use `predict()`

Argument	**Description**
object	A trained model object created by `train()`
newdata	A data frame with the same features (prior to baking)
type	Return raw classes ("raw") or probabilities ("prob")?

```
fare_pred <- predict(fare_lm, newdata = fare_test)
glimpse(fare_pred)
#>  num [1:240] 114 103.6 101.6 93.7 92.9 ...
```

[1] The same process is used for both evaluating performance and deploying the model in the real world.

Test Set Labels

To get labels from the final model in the format expected by R, we can use `bake()`

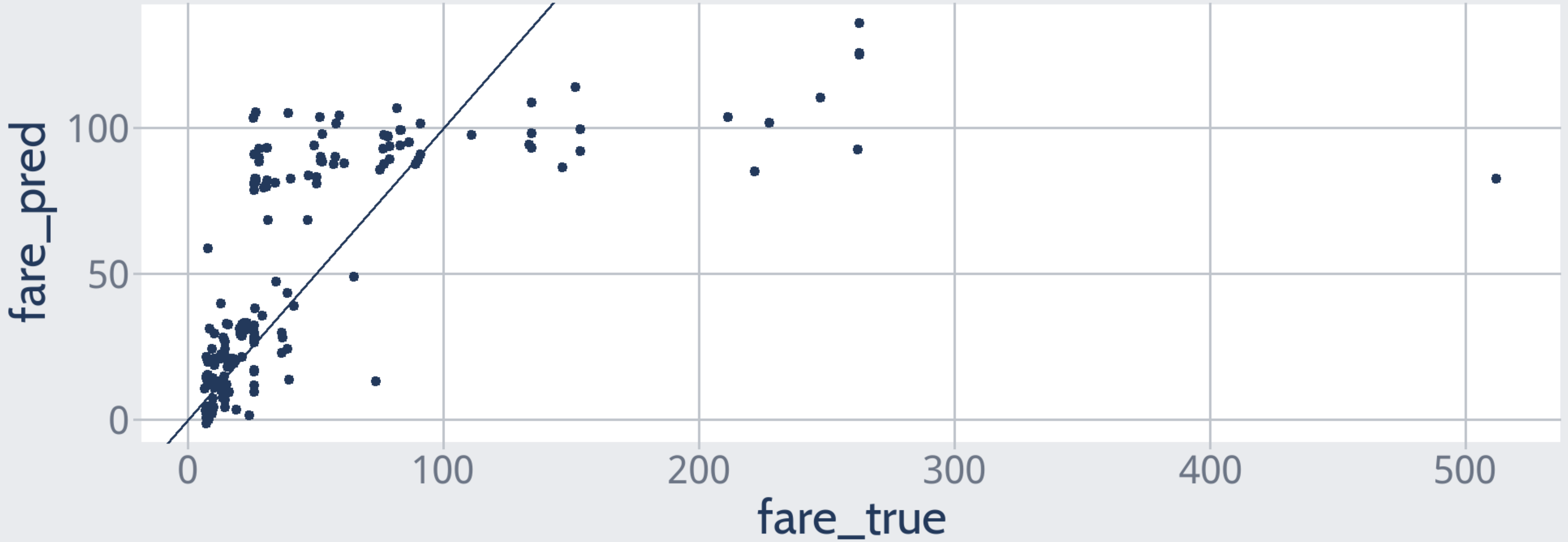
```
fare_test_baked <-
  fare_recipe %>%
  prep(training = fare_train) %>%
  bake(new_data = fare_test)

fare_true <- fare_test_baked$fare
glimpse(fare_true)
#>  num [1:240] 151.6 51.5 227.5 78.8 76.3 ...
```

A few points of potential confusion to avoid:

1. Use `train()` and `predict()` before baking the recipe (i.e., let them do it)
2. `caret::predict()` uses `newdata`, whereas `recipes::bake()` uses `new_data`

Visualizing Predicted and Trusted Labels

```
qplot(x = fare_true, y = fare_pred) + geom_abline()
```



Estimating Test Set Performance

To estimate performance in {caret}, use `postResample()`

Argument	Description
pred	A vector of predicted labels
obs	A vector of trusted labels

```
postResample(pred = fare_pred, obs = fare_true)
#>       RMSE   Rsquared        MAE 
#> 44.2923446  0.4240302 21.5320223
```

This function will give basic results (e.g., Accuracy) for classification too

More Metrics from {yardstick}

The {yardstick} package provides lots of performance metrics

For regression, I like the **Huber Loss** distance metric and **CCC** correlation metric

```r
# Huber Loss (blends RMSE and MAE)
yardstick::huber_loss_vec(truth = fare_true, estimate = fare_pred)
#> [1] 21.04084
```

```r
# Concordance Correlation Coefficient
yardstick::ccc_vec(truth = fare_true, estimate = fare_pred)
#> [1] 0.593931
```

For classification, I like the **MCC** class metric and **Log Loss** probability metric

Model Interpretation

Variable Importance

Predictive **accuracy** is emphasized in ML over interpretability and inference

- The main goal of most applied ML studies is to **quantify performance**

However, some algorithms can provide insight into their decision-making

- As a model usually used for inference, LM has strong interpretability
- We can examine the model coefficients (intercept and slopes)

```
fare_lm$finalModel %>% coefficients()
```

(Intercept)	age	sibsp	parch	pclass_X2nd	pclass_X3rd	sex_male
99.59	-1.78	5.36	8.53	-71.78	-80.65	-9.88

Variable Importance

However, these coefficients have different units and levels of uncertainty

Instead, for LM, we use the absolute value of each coefficient's t-statistic

```
varImp(fare_lm, scale = FALSE)
#> lm variable importance
#>
#>                Overall
#> pclass_X3rd     19.028
#> pclass_X2nd     15.867
#> parch            4.941
#> sibsp            3.120
#> sex_male         2.824
#> age              1.002
```

```
varImp(fare_lm, scale = TRUE)
#> lm variable importance
#>
#>                Overall
#> pclass_X3rd     100.00
#> pclass_X2nd      82.47
#> parch            21.85
#> sibsp            11.75
#> sex_male         10.11
#> age               0.00
```

[1] Other algorithms have different ways to estimate variable importance, but `varImp()` will take care of it.

Live Coding

Now that we have explored the prediction of passenger fare, let's predict survival

Training, evaluating, and interpreting a **classification** model is very similar but...

- We will be using GLM (logistic regression) instead of LM as our **new algorithm**
- We will have some **different performance metrics** to calculate and interpret
- We can explore the **raw class predictions** and **estimated class probabilities**

Finally, as a hands-on activity, you will classify `satisfaction` in the `airsat` dataset

Hands-on Activity

Modify the Live Coding example code to achieve the following goals:

1. Read in the `airsatisfaction.csv` file (same as used in Activity 2-A)
2. Create a 75% training and 25% testing set stratified by `satisfaction`
3. Create a recipe predicting `satisfaction` with the following steps:
    - `step_nzv(all_predictors())`
    - `step_normalize(all_numeric_predictors())`
    - `step_pca(all_numeric_predictors(), threshold = 0.9)`
    - `step_dummy(all_nominal_predictors())`
4. Train a GLM with this recipe with 10-fold cross-validation repeated 3 times
5. Estimate Accuracy and Kappa in the training and testing sets

BONUS 1: Examine and plot the model's variable importance. What effect does PCA have on interpretation?

BONUS 2: Construct a confusion matrix and ROC curve for the model