Model performance

Tad Dallas

Contents

What do we mean by 'performance'?

So we fit a model. We have some attempt at understanding a system, or at least some response variable given our set of predictors. But what we want to know is "is this model doing a good job?". Significant predictor variables do not inherently mean that these predictors are informative or predictive of our response. Estimates of model performance attempt to quantify the amount of variance explained (e.g., R^2) or the relationship between predicted and empirical values of the response variable.

How do we measure model performance?

So we have our predictions that were made on data that the model hasn't seen... so it reduces the chances that we would overfit. Now we want to assess the performance of the model. We can do so in a whole bunch of ways. A helpful way to think about model performance is to think about how we define model performance. Do we want a model whose predictions most closely match the data? Do we want a model is quite certain in it's predictions (whether the predictions are wrong or right).

Calibration: Calibration refers to the statistical consistency between distributional predictions and true values. So are the range of predictions in alignment with the range and distribution of the empirical data.

This one is tricky to understand, but it basically argues that in a well-calibrated model, the distribution of the output predictions should roughly match the distribution of true values. So if a model predicts that 10 out of 100 points will have between 0.7 and 0.8 suitability, a well-calibrated model will have around 70% to 80% of the values in this range as true occurrence points.

Discrimination: discrimination refers to the ability of the model to rank values correctly. This makes the most sense when thinking about binary data, where the truth is 0/1 and the prediction takes on some continuous value.

One example of this would be AUC (the area under the receiver operating characteristic). This gets at the balance between the true positive rate (1's correctly predicted as 1's) and the false positive rate (0's falsely predicted to be 1's). True positive rate is commonly referred to as *sensitivity* and false positive rate is commonly referred to as *1-specificity*. We won't go into the details here, but the wikipedia for AUC is pretty solid, as is this explainer.

Precision: precision (also called sharpness) measures the uncertainty in model predictions for a given prediction

In a traditional view of precision, we can imagine the classic accuracy v. precision dartboard example, where accurate points tend to be in the center of the board, but precise points will be grouped together anywhere on the board. Precision in this case relates to the ability of the model to predict values that coincide with 0 or 1, such that an estimate of 0.5 is the least precise.

Accuracy: accuracy estimates the distance between predicted values and true values, typically ignoring the uncertainty in predicted values.

So one measure of accuracy for our simple example above would simply be the squared difference between prediction and truth (in order to make all values positive).

```
preds2 <- data.frame(truth=runif(100), pred=runif(100))
acc <- (preds2$truth - preds2$pred)**2
```

This results in a vector that gets at the relative error in each predicted point. We can also calculate a measure of performance for the entire model by measuring root mean squared error (RMSE). Here, we just take the square root of the average model accuracy as defined above as the squared difference.

```
sqrt(mean(acc))
```

```
## [1] 0.4395527
```

What is interesting here though is that these measures would not be comparable across models, and don't really have clear bounds or ways to say that if RMSE is below a certain value, then the model is good. This is why we may prefer a measure of model performance that is either relative to an uninformed model (or some baseline model containing a few variables) or when the performance measure is bounded between certain values (e.g., a correlation between predicted and actual values will always be bound between -1 and 1).

This was obviously a simple example of how we could calculate one aspect of model performance, but there are plenty of others. For binary response data, there is a good discussion of model performance measures in https://doi.org/10.1002/ecm.1370. Otherwise, a good resource for thinking about model building and testing is Hastie, Tibshirani, and Friedman *Elements of Statistical Learning* (https://web.stanford.edu/~hastie/ElemStatLearn/).

Another consideration is that sometimes the response variable we are attempting to model is not a continuous variable (meaning that some model performance measures like the correlation coefficient above would not be appropriate). In many cases, the goal of a regression or classification model is to predict a binary output (e.g., survival models are a good example of this, as are species distribution models). The next section will explore model performance measures for binary data, while also discussing the importance of cross-validation and model transference.

The importance of the train/test split for predictive models

Oftentimes when we want to identify important variables, we use all of the data to get the best fit model, and then craft the narrative around the effects of certain variables in the model (i.e., which variables were significantly related to the response variable?). Fitting models in this way can sometimes tell us about how much collective variation the predictors explained (e.g., R^2 measures), but these will not tell us about the predictive ability of the model. The reason for this is that model performance estimates measured on data the model has already 'seen' creates bias, in that a model could be incredibly *overfit* and look like a really good model, when it actually has no predictive worth. *Overfitting* occurs when you have complex models that greedily try to capture as much variation in the response variable as possible, but in doing so, they cannot be used to extrapolate to new data. We talked about overfitting previously in the `modeling` lectures.

To avoid overfitting in predictive models, we can simply split the data into two chunks, using one chunk for model fitting and the other as an independent test case. Let's work through an example of how you can approach this train/test split in the context of another logistic model.

Model performance for continuous output

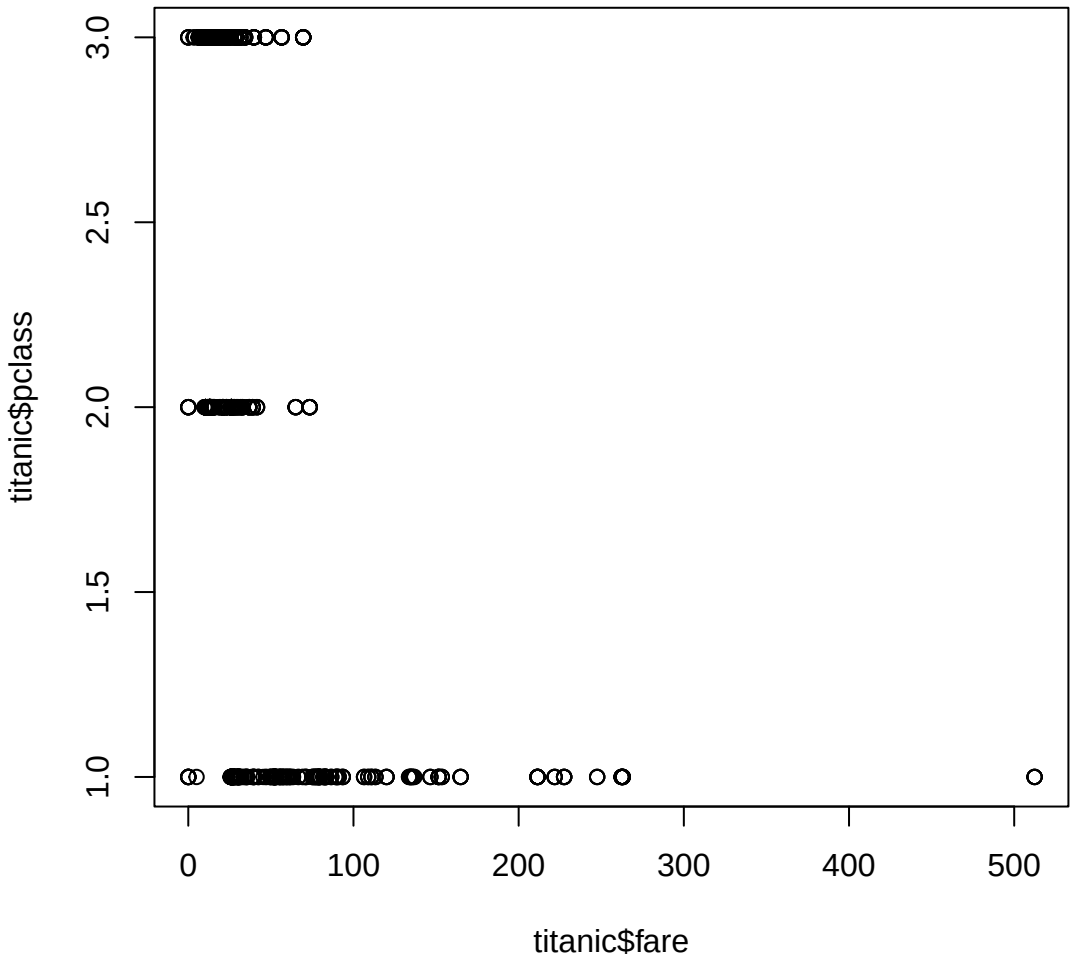
Let's continue to explore model performance and the importance of the train/test split for a model with a continuous response variable. Here, our goal is to develop a predictive model of the price of the fare for people on the Titanic. I know it's a bit of a dark dataset to use, but it's weirdly common as a beginner dataset. We'll explore some biological data as well.

```
titanic <- read.csv('titanic3.csv')
titanic$age[which(is.na(titanic$age))] <- mean(titanic$age, na.rm=TRUE)
```

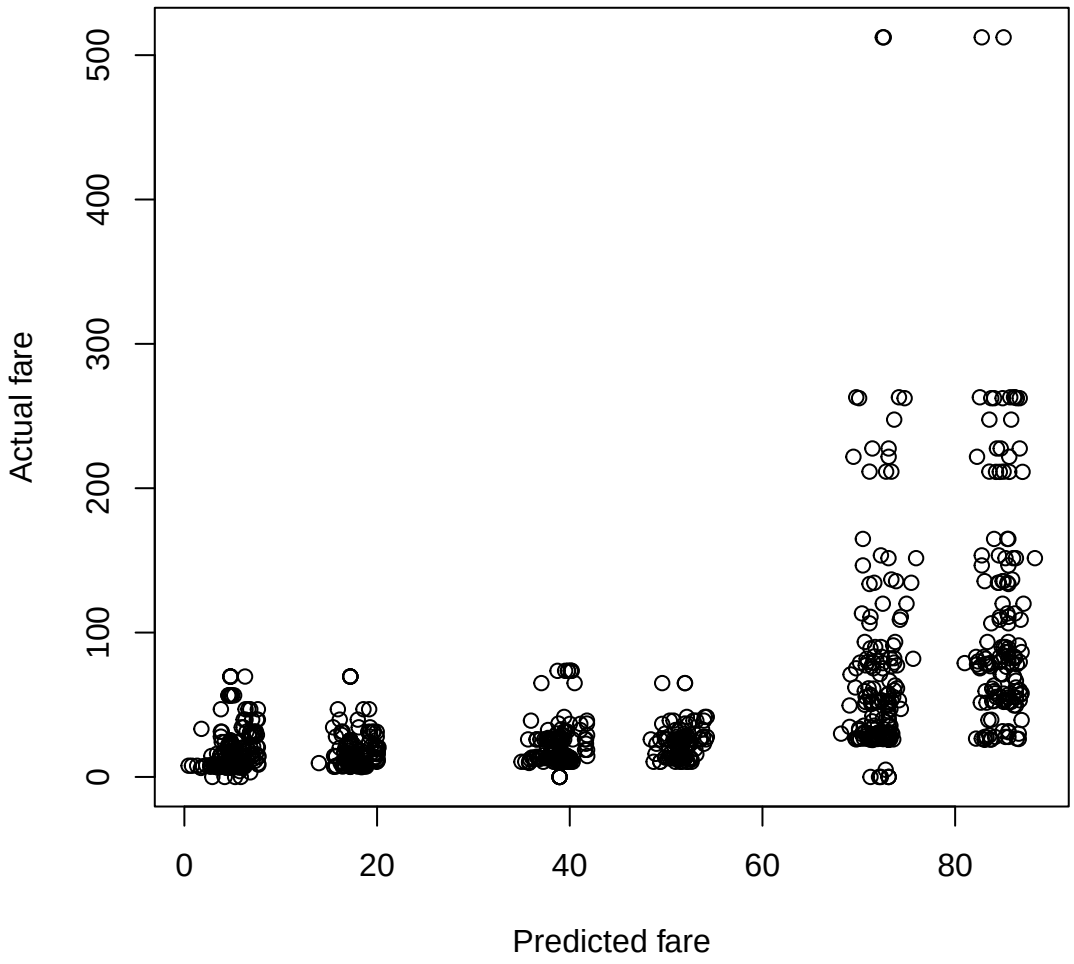
```r
titanic$fare[which(is.na(titanic$fare))] <- mean(titanic$fare, na.rm=TRUE)


titanicMod0 <- glm(fare ~ sex+age+pclass, data=titanic, family='gaussian')

summary(titanicMod0)
```

```
##
## Call:
## glm(formula = fare ~ sex + age + pclass, family = "gaussian",
##     data = titanic)
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 122.63133    5.46893  22.423  < 2e-16 ***
## sexmale     -12.43795    2.48950  -4.996 6.64e-07 ***
## age          -0.09844    0.09870  -0.997    0.319
## pclass      -34.15610    1.52710 -22.367  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1808.245)
##
##     Null deviance: 3501400  on 1308  degrees of freedom
## Residual deviance: 2359760  on 1305  degrees of freedom
## AIC: 13538
##
## Number of Fisher Scoring iterations: 2
```

```r
plot(titanic$fare, titanic$pclass)
```




```
plot(x=predict(titanicMod0, type='response'), y=titanic$fare,
  ylab='Actual fare', xlab='Predicted fare')
```




```
cor.test(predict(titanicMod0), y=titanic$fare)
```

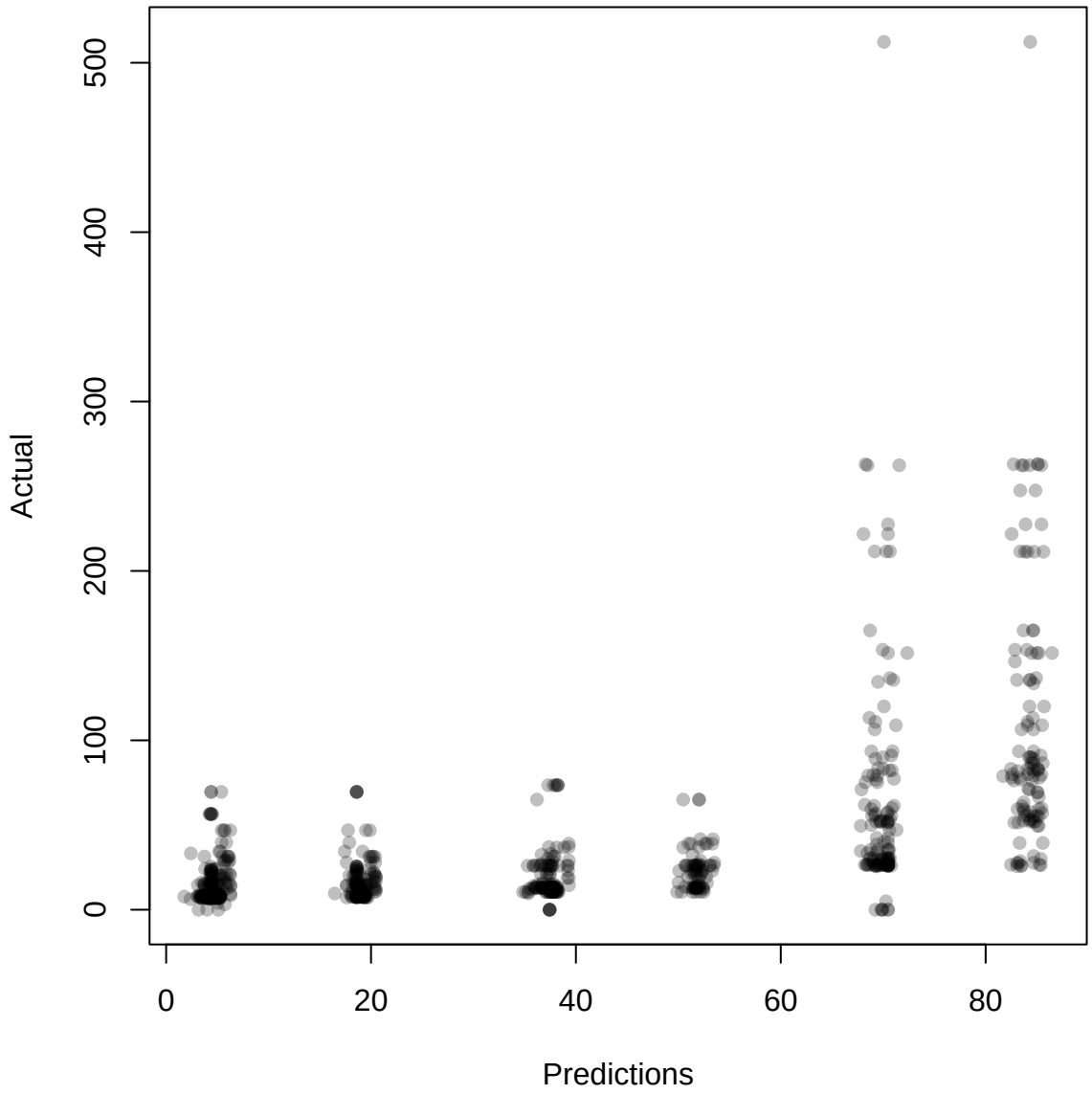
```
##
##  Pearson's product-moment correlation
##
## data:  predict(titanicMod0) and titanic$fare
## t = 25.146, df = 1307, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5333289 0.6064298
## sample estimates:
##       cor
## 0.5710102
```

Is this a good model?

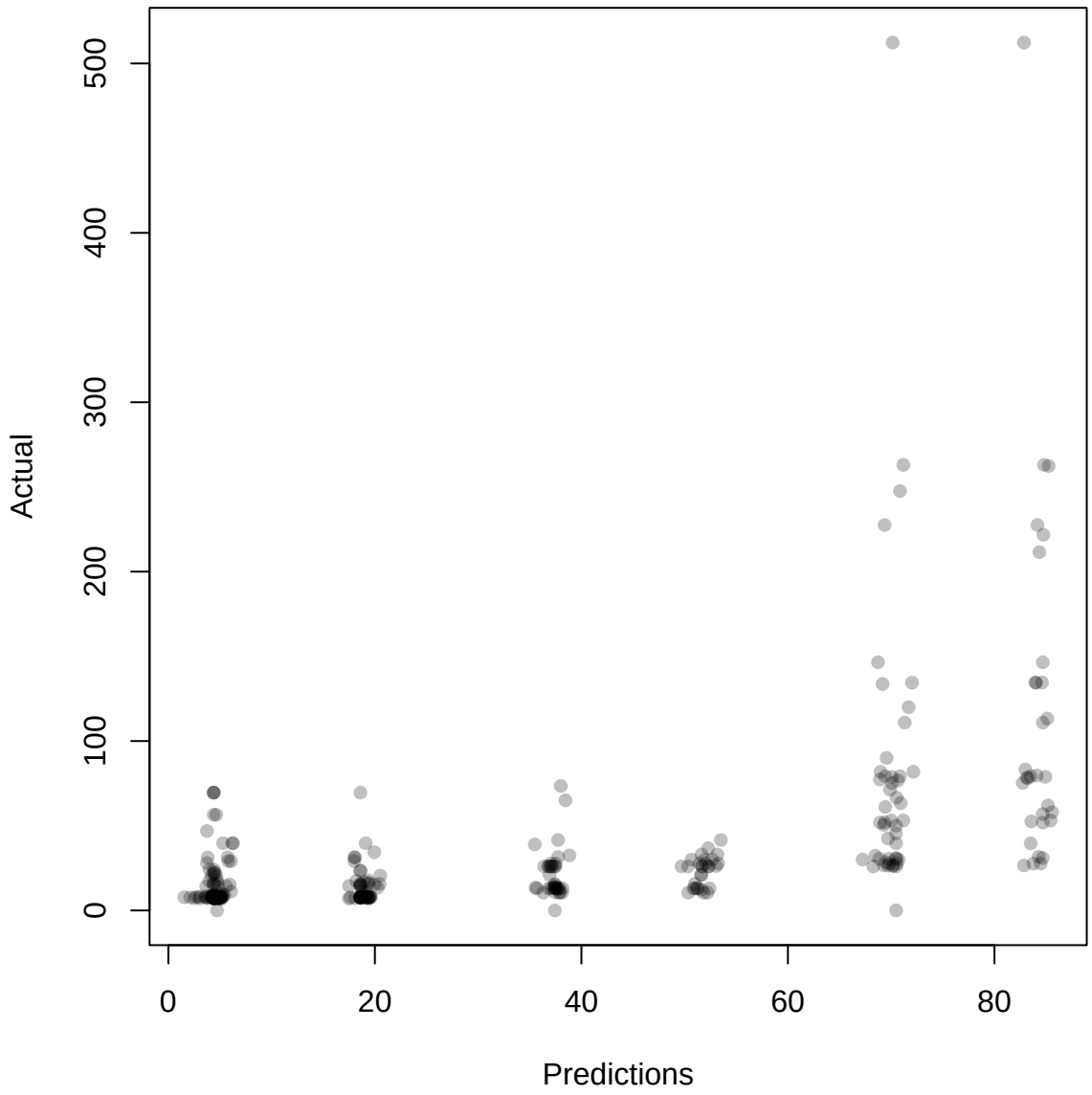
Let's explore how it does on a test set versus the training data

```
inds <- sample(1:nrow(titanic), round(0.75 * nrow(titanic)), replace=FALSE)
train <- titanic[inds, ]
test <- titanic[-inds, ]

titanicMod1 <- glm(fare ~ sex+age+pclass, data=train, family='gaussian')
```

```
par(mar=c(4,4,0.5,0.5))
plot(predict(titanicMod1, type='response'), train$fare,
  xlab='Predictions', ylab='Actual', pch=16, col=adjustcolor(1,0.25))
```




```
# predicting to the test set
par(mar=c(4,4,0.5,0.5))
plot(predict(titanicMod1, type='response', newdata=test), test$fare,
  xlab='Predictions', ylab='Actual', pch=16, col=adjustcolor(1,0.25))
```




```
# correlation coefficients for train and test
cor.test(predict(titanicMod1, type='response'), train$fare)
```

```
##
##  Pearson's product-moment correlation
##
## data:  predict(titanicMod1, type = "response") and train$fare
## t = 22.637, df = 980, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5433175 0.6255868
## sample estimates:
##     cor
## 0.58596
```

```
cor.test(predict(titanicMod1, newdata=test, type='response'), test$fare)
```

```
##
##  Pearson's product-moment correlation
```

```
##
## data:  predict(titanicMod1, newdata = test, type = "response") and test$fare
## t = 11.44, df = 325, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4537175 0.6088836
## sample estimates:
##       cor
## 0.5358091
```

I'm focusing on correlation coefficients of predicted versus actual values as a good measure of model performance, but there are plenty of others. Many of them may be more appropriate when the user only really cares about how much variance the model explains. That is, we become less worried about the train/test split and the ability of the model to predict, and care more about the ability of the model to capture the variance (or deviance) in the data.

One way we can do this is to explore the relationship between residual deviance and null deviance. Here, we define deviance as a measure of error, where lower deviance means a better fit to the data. Null deviance is the amount of error given an intercept-only model. This should always be a larger value than the deviance.

```
with(summary(titanicMod1), 1 - deviance/null.deviance)
```

```
## [1] 0.3433491
```

This measure is bounded between 0 and 1, with larger values corresponding to more variance explained. This balance between deviance and null deviance is incredibly common, as the measure above is referred to as R^2.

> Let's think about how to make it better. Work in small groups to develop your own predictive model of fare. The goal here is to maximize model performance (10-15 minutes)

Using ROCR to get at standard performance measures for classification models

In this section, we will build a model of the probability of surviving the sinking of the titanic. The predictors include things like sex, age, fare, ticket number, class, and cabin.

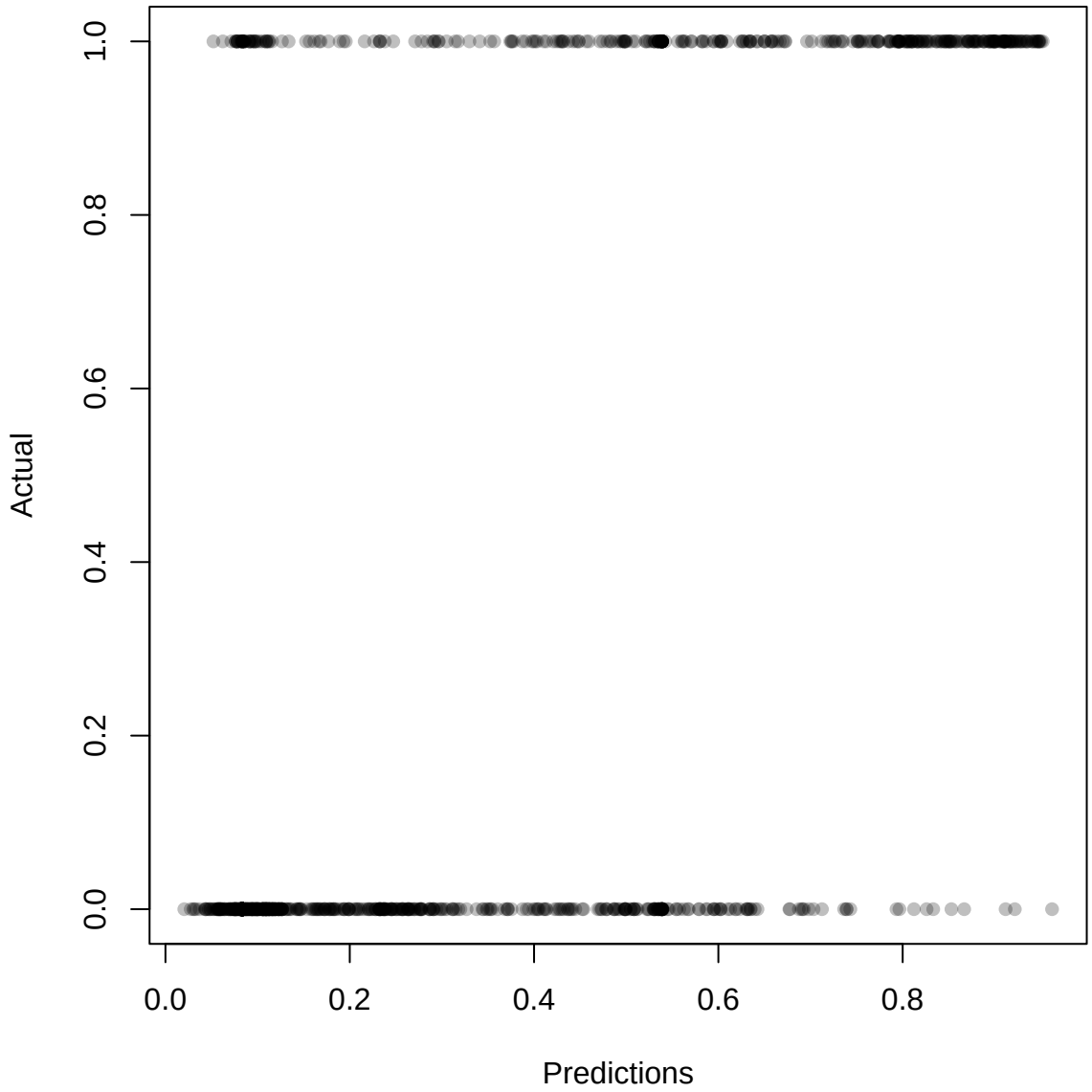
```
inds <- sample(1:nrow(titanic), round(0.75 * nrow(titanic)), replace=FALSE)
train <- titanic[inds, ]
test <- titanic[-inds, ]
```

```
titanicMod <- glm(survived ~ sex+age+fare+pclass, data=train, family='binomial')

summary(titanicMod)
```

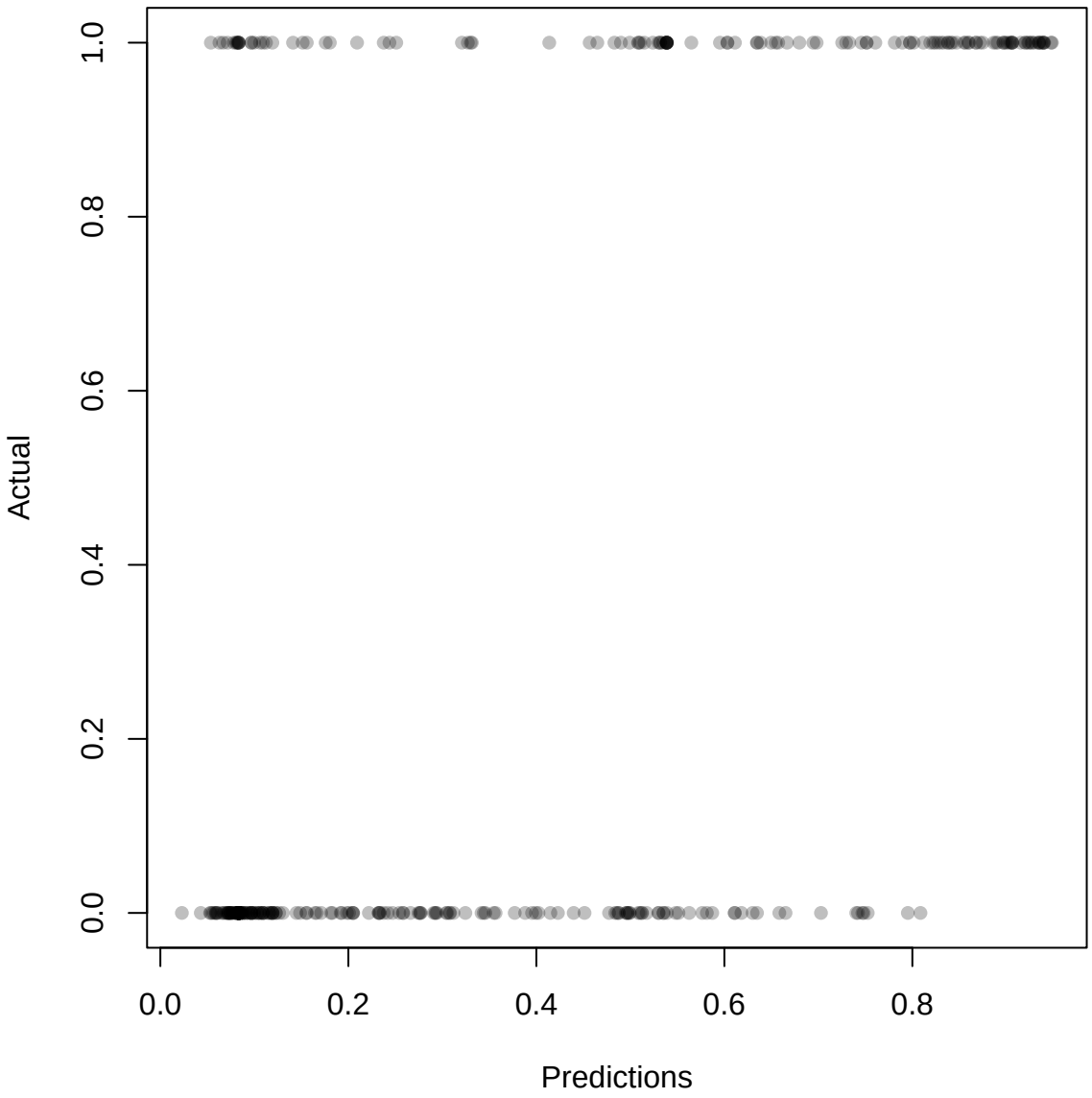
```
##
## Call:
## glm(formula = survived ~ sex + age + fare + pclass, family = "binomial",
##     data = train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.815407   0.508983   9.461  < 2e-16 ***
## sexmale     -2.552205   0.178186 -14.323  < 2e-16 ***
## age         -0.033419   0.007006  -4.770 1.84e-06 ***
## fare        -0.001973   0.002217  -0.890    0.373
## pclass      -1.215314   0.135720  -8.955  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1299.05  on 981  degrees of freedom
## Residual deviance:  905.74  on 977  degrees of freedom
## AIC: 915.74
##
## Number of Fisher Scoring iterations: 4
```

```
# visualizing predictions on the training set
par(mar=c(4,4,0.5,0.5))
plot(predict(titanicMod, type='response'), train$survived,
  xlab='Predictions', ylab='Actual', pch=16, col=adjustcolor(1,0.25))
```




```
# predicting to the test set
par(mar=c(4,4,0.5,0.5))
plot(predict(titanicMod, type='response', newdata=test), test$survived,
  xlab='Predictions', ylab='Actual', pch=16, col=adjustcolor(1,0.25))
```



So let's put some numbers behind these plots. We'll use the `ROCR` package to calculate some performance measures.

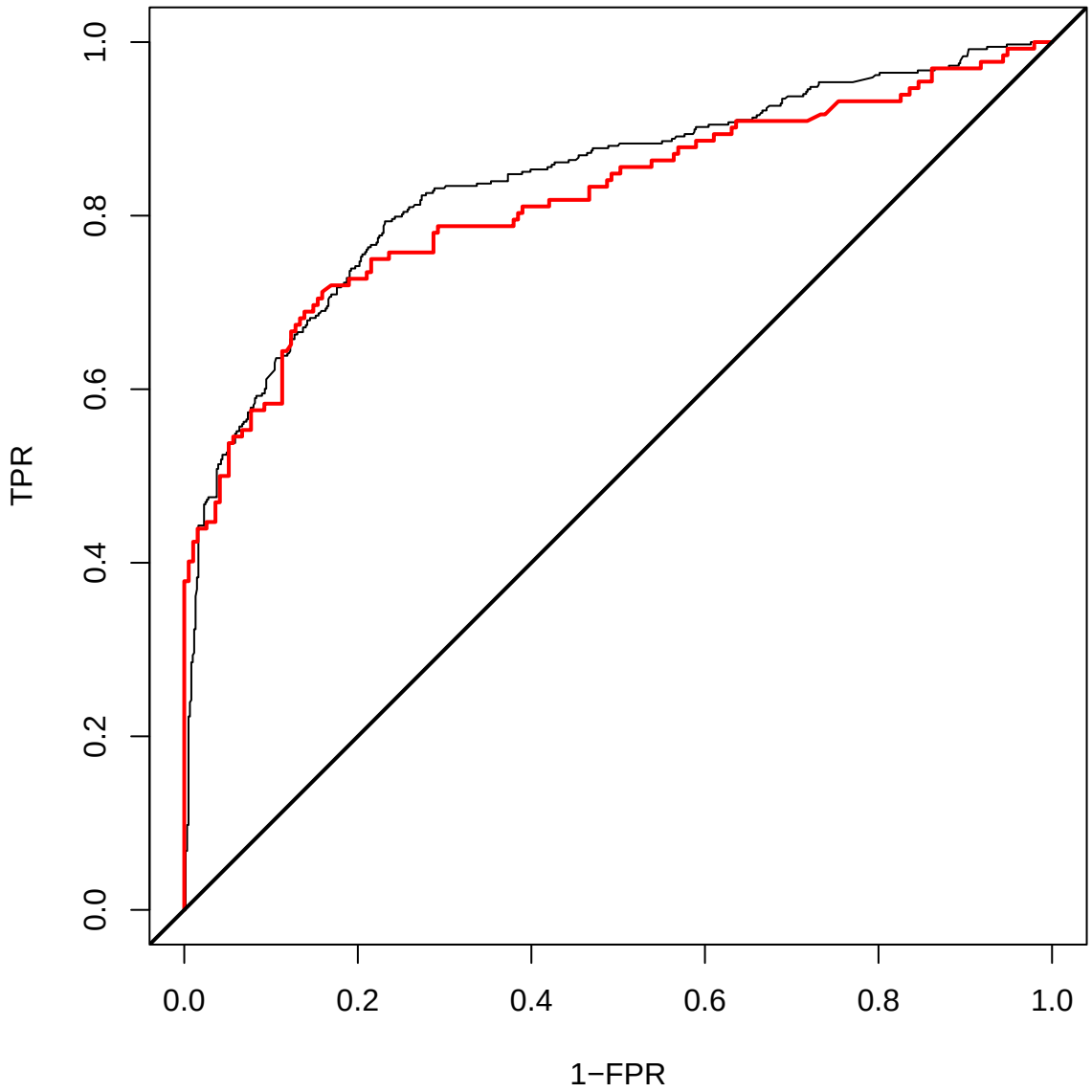
```
library(ROCR)

predTrain <- prediction(predictions=predict(titanicMod, type='response'),
  labels=train$survived)

predTest <- prediction(predictions=predict(titanicMod, type='response', newdata=test),
  labels=test$survived)


## discrimination (AUC)
as.numeric(performance(predTrain, 'auc')@y.values)
```

```
## [1] 0.8392579
```

```
as.numeric(performance(predTest, 'auc')@y.values)
```

```
## [1] 0.8184926
```

```
par(mar=c(4,4,0.5,0.5))
plot(unlist(performance(predTrain, 'tpr')@y.values)~
  unlist(performance(predTrain, 'fpr')@y.values),
  pch=16, type='l', xlab='1-FPR', ylab='TPR')
lines(unlist(performance(predTest, 'tpr')@y.values)~
  unlist(performance(predTest, 'fpr')@y.values),
  col='red', lwd=2)
abline(a=0,b=1, lwd=2)
```



This is actually not a terribly overfit model, but we still see that the model performance is higher when estimated on the data used to train the model than on the holdout test set.

> Can you calculate differences in accuracy between the two predictions above (i.e., `predTrain`, and `predTest`)?

Another useful visualization is the *confusion matrix*. This can start to tell you if the model is worse off at distinguishing false from true, or true from false. This can be visualized as data going into 1 of four quadrants. Here, the rows represent your predictions, and the columns the actual values. So the top left box corresponds to the case where you predicted a positive and the true value was positive (good job!). Same with the lower right, except you predicted a negative right. The other two situations are the situations that form the basis for lots of the measures of model performance. The top right case corresponds to the case where you predicted a positive but the value was negative (false positives; also called Type I error). The lower left case is where you predict a negative but it is actually positive (false negatives; also called Type II errror).

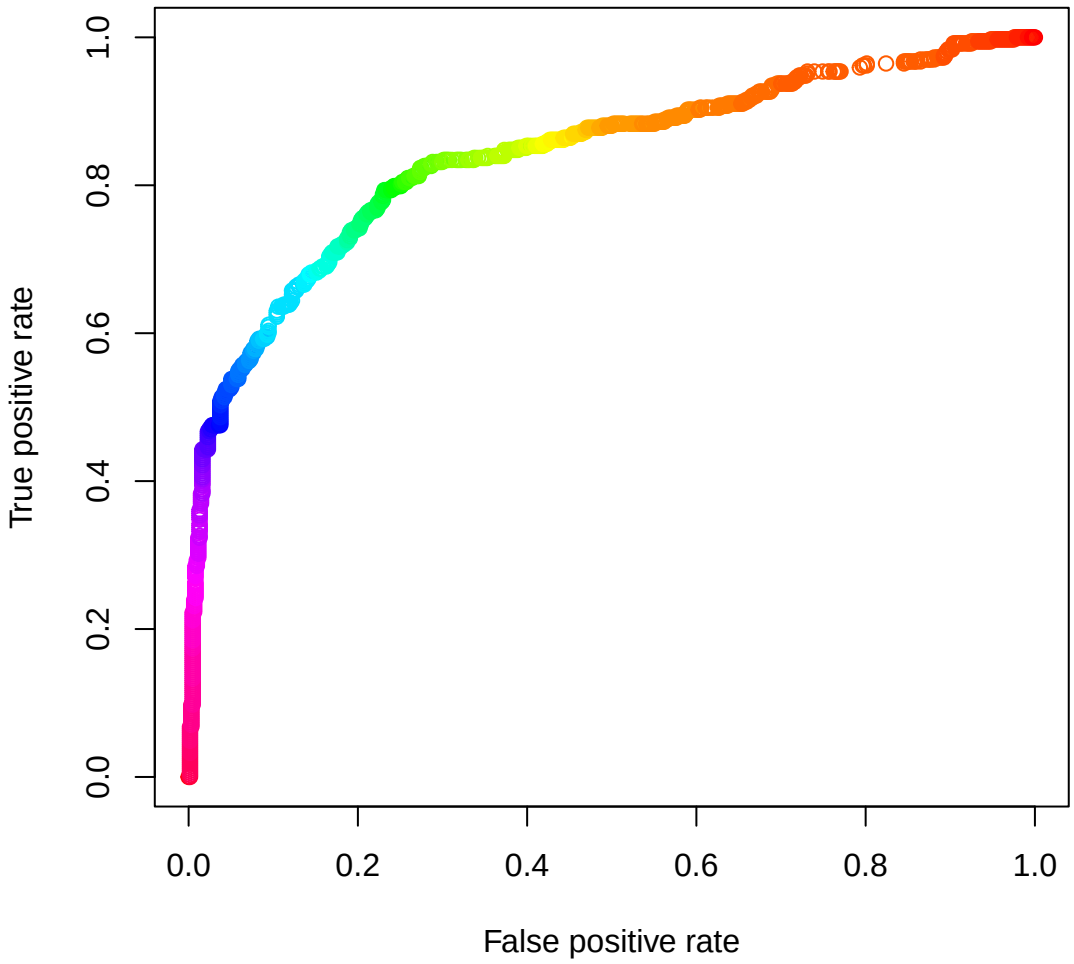
> Rows are predictions, columns are actual values.

Positive	Negative
Positive | TP | FP |
Negative | FN | TN |

But the thing about our predictions currently is that we have binary truth and continuous predictions, right? That is, our model does not output either 0 or 1, but a value in between 0 and 1. This is the fundamental difference between this being a regression task (continuous output) versus a classification task (binary output). This also speaks to the underlying structure of many measures of model performance (e.g., AUC) that are based on the confusion matrix.

> But given that measures like AUC need to be in the confusion matrix format, how do we get AUC values when our model predictions are continuous?

```
aucCurveTrain <- performance(predTrain, measure = "tpr", x.measure = "fpr")
aucCurveTest <- performance(predTest, measure = "tpr", x.measure = "fpr")

thresh <- unlist(aucCurveTrain@alpha.values)
thresh[1] <- 1

plot(unlist(aucCurveTrain@x.values),
  unlist(aucCurveTrain@y.values),
  ylab='True positive rate',
  xlab='False positive rate',
  col=rainbow(100)[cut(thresh, 100)])
```



In this space of TP and FP, what would a poorly performing model look like?

sessionInfo

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 22.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.10.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/New_York
```

```
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ROCR_1.0-11    geodata_0.5-8  terra_1.7-39   maps_3.4.1     rgbif_3.7.7
## [6] gbm_2.1.8.1    rmarkdown_2.23
##
## loaded via a namespace (and not attached):
##  [1] viridis_0.6.3     utf8_1.2.3        generics_0.1.3    xml2_1.3.5
##  [5] stringi_1.7.12    lattice_0.21-8    httpcode_0.3.0    digest_0.6.33
##  [9] magrittr_2.0.3    evaluate_0.21     grid_4.3.1        fastmap_1.1.1
## [13] plyr_1.8.8        jsonlite_1.8.7    Matrix_1.6-0      whisker_0.4.1
## [17] crul_1.4.0        tinytex_0.45      survival_3.5-5    urltools_1.7.3
## [21] gridExtra_2.3     httr_1.4.6        fansi_1.0.4       viridisLite_0.4.2
## [25] scales_1.2.1      oai_0.4.0         codetools_0.2-19  lazyeval_0.2.2
## [29] cli_3.6.1         rlang_1.1.1       triebeard_0.4.1   munsell_0.5.0
## [33] splines_4.3.1     yaml_2.3.7        parallel_4.3.1    tools_4.3.1
## [37] dplyr_1.1.2       colorspace_2.1-0  ggplot2_3.4.2     curl_5.0.1
## [41] vctrs_0.6.3       R6_2.5.1          lifecycle_1.0.3   stringr_1.5.0
## [45] pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.3      data.table_1.14.8
## [49] glue_1.6.2        Rcpp_1.0.11       xfun_0.39         tibble_3.2.1
## [53] tidyselect_1.2.0  highr_0.10        knitr_1.43        htmltools_0.5.5
## [57] compiler_4.3.1
```