3/2/2023

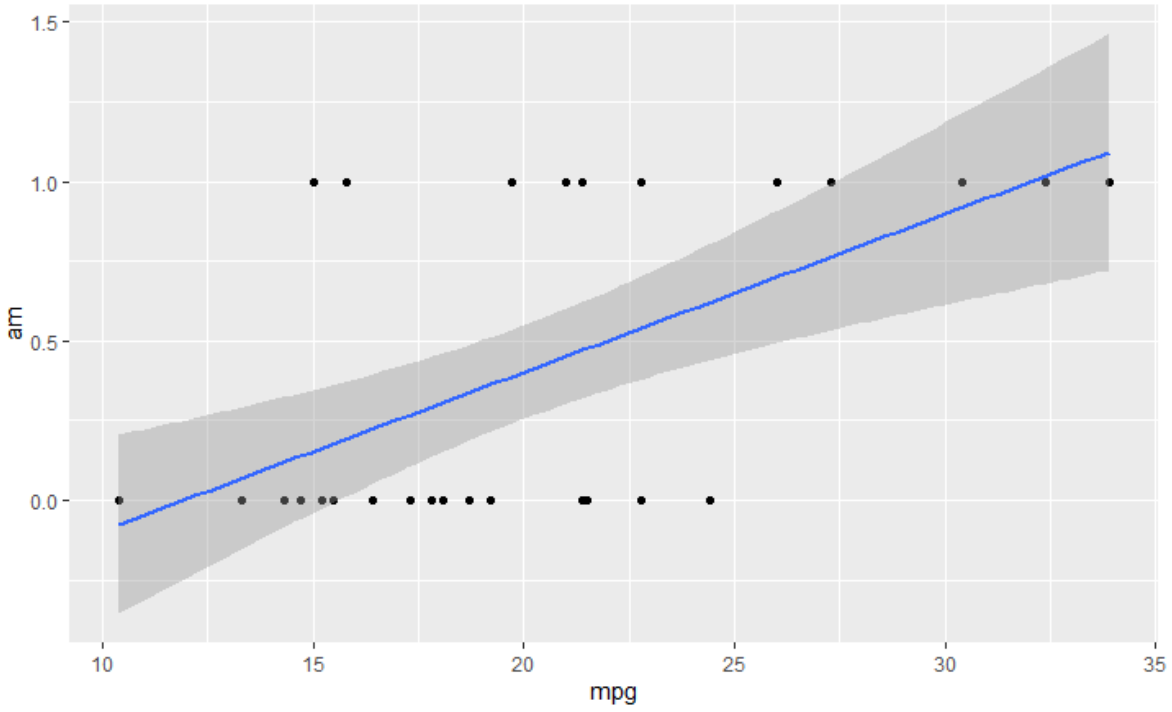
Logistic Regression and Classification models

We've looked at cases with ANOVA where were modeled a numerical response variable with categorical or factored explanatory variables, and with regression so far, we've looked at modeling a numerical response variable with other numerical explanatory variables. We want to extend our general linear models now to a third case, where we model a categorical response variable with numerical explanatory variables. We have discussed some of the potential issues with categorical variables, ordinal variables and discrete variables in the context of being an explanatory variable. Some of these issues will also have to be considered in the case where they act as response variables: what do we do if the prediction falls between our discrete levels? How do we interpret that? What if the levels are unordered? What if the jump between levels is nonuniform?

We will begin with the case of just two levels, and then we will briefly discuss strategies and potential pitfalls for situations where there are more than two levels.

When there are just two possible categorical responses, we call this binary classification. We use the term classification to indicate that our goal is to bin the data, classify it, into groups. Our model is binary because we have just two possible outcomes. You may recall a similar situation when we discussed Bernoulli variables (e.g. binomial distribution) that had only two outcomes, success and failure. This is generally how we will think of our categorical variable in this instance as well. We will call 1 a success, and 0 a failure; a 1 is in the category of interest, and 0 is not in the category of interest.

Suppose that we naively say, well, probability is continuous variable between 0 and 1, so let's just make a linear regression function on the data and interpret it as a probability? The graph below shows data from mtcars with am (American made) as the response variable and mpg as the explanatory variable, and a linear model plotted on the data using traditional simple linear regression.



You can see from the graph that the line doesn't fit the data particularly well, and we have potential problems at both ends: the regression model is predicting negative probabilities on the left end, and probabilities greater than 1 on the right end. How would we interpret that? Could we just round the predictions? If less than 50%, then it predicts 0, and if greater than 50%, it predicts 1? Certainly, there might be some value here, but given the sensitivity of the model to random variation, there would be a lot of volatility in the predictions for values around the 50% mark depending on the specifics of the data collected and the specific coefficients in the model.

There is some merit in this approach, but how can we improve it?

One way would be to accommodate more predictive values than just between 0 and 1: to use more of the number line. One way to achieve this is to use odds, rather than probabilities.

Recall from last semester that odds for an event are $\frac{p(x)}{1-p(x)}$: the probability of the event, divided by the probability the event will not happen. When the probability of the event is very high, the complement is very small, so the odds are large. When the probability of an event is small, the complement is large and the odds are small (greater than zero, but small). If we used odds as the model instead of a probability, then we expand the range of possible values from 0 to infinity. The odds = 1 is when the probability is evenly split (50% both for and against). The range of values for the top half of the range of probabilities is 1 to infinity, versus the bottom half of probabilities is only from 0 to 1. How do we make that better?

Taking the logarithm of the odds function would extend the values from 1 to infinity into the set 0 to infinity (since ln(1)=0), and the values from 0 to 1 would be stretched into negative infinity to 0. This evenly balances the range of outcomes for both halves and means that every predicted value has a meaning. Then we can apply linear regression to the resulting values.

$$\ln\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x$$

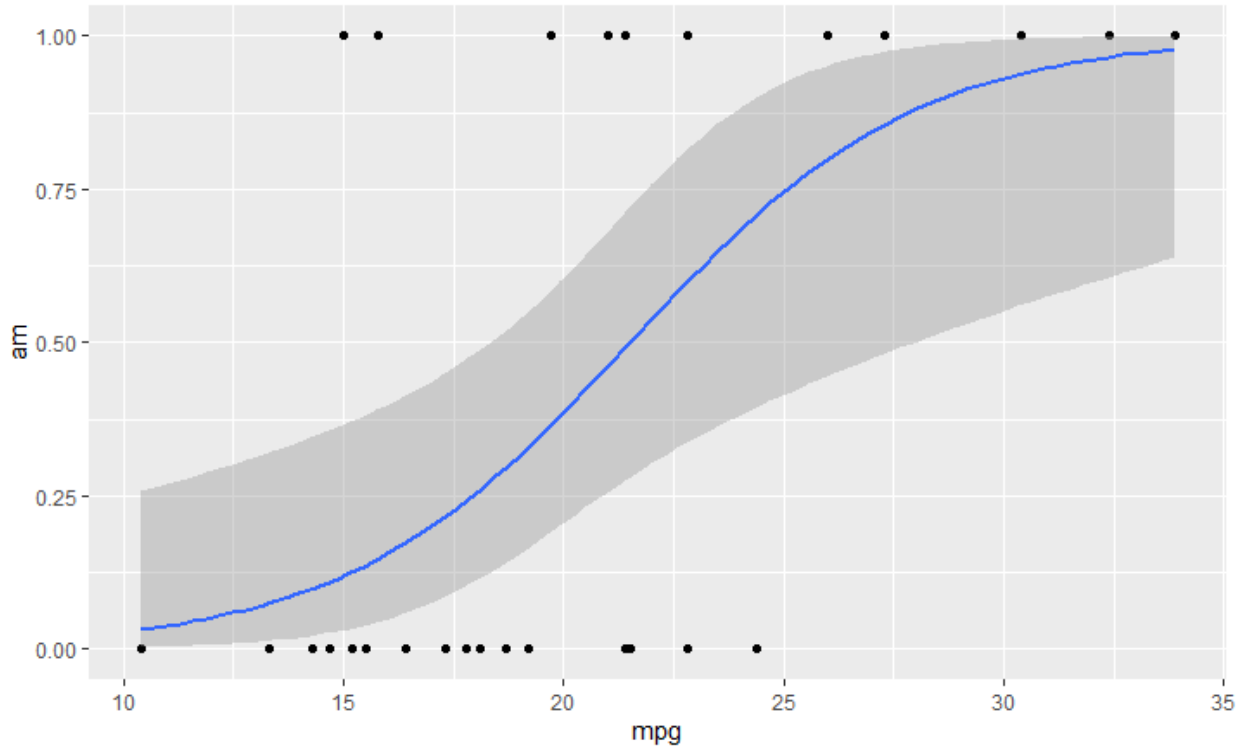
We can rewrite this as

$$\frac{p(x)}{1-p(x)} = e^{\beta_0+\beta_1 x}$$

And if we then solve for $p(x)$ we then obtain what is referred to as the logit equation.

$$p(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

The resulting curve is sometimes described as an S-curve, with a sharp increase around 50% probability and asymptotes at 0 and 1. A logit curve model of our American made variable is shown in the graph below.



This function eliminates the issues we had constructing a simple linear regression on this same data, and uses the glm() function in R to analyze the model for the same things we were able to analyze in our other linear models. For instance, we can conduct hypothesis tests on the full model, or on individua coefficients in the model. After transforming the variables, our model has similar assumptions to traditional regression, with errors on the coefficients approximately normal, and so forth.

Let's look at the output of the model analysis.

Call:
glm(formula = am ~ mpg, family = "binomial", data = mtcars)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.5701	-0.7531	-0.4245	0.5866	2.0617

Coefficients:

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-6.6035	2.3514	-2.808	0.00498 **
mpg	0.3070	0.1148	2.673	0.00751 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

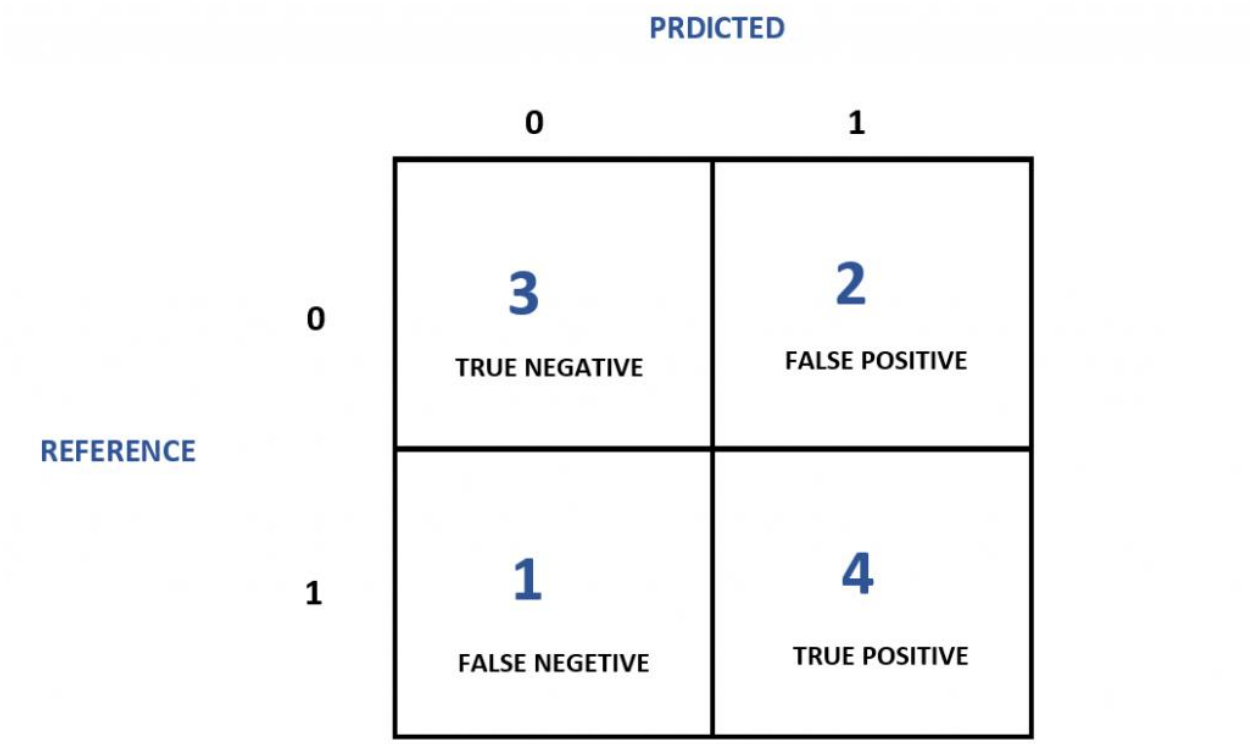
(Dispersion parameter for binomial family taken to be 1)

Null deviance: 43.230 on 31 degrees of freedom

Residual deviance: 29.675 on 30 degrees of freedom
AIC: 33.675

Number of Fisher Scoring iterations: 5

Outputs of the model are values between 0 and 1. These are interpreted as probabilities. The most common classification scheme is, therefore, that predictions from the logistic equation that are 0.50 or larger are interpreted as class 1, and values less than that are interpreted as class 0 (following a traditional rounding scheme). In R, you can have the output delivered as the probabilities, or as the classifications. However, there may be reasons to deviate from this standard scheme. It may be that given the classifications, that a false positive or a false negative is more dangerous than the other. In such a case, we can adjust the minimum value needed for the classification break point, either by adjusting it downward (to avoid false negatives) or adjusting it upward (to avoid false positives). This is fairly uncommon when the stakes are low but is somewhat more common in a medical context (for example), where a false positive might trigger further testing to resolve the issue.



We can think of the false positives as a Type I error, and the false negatives as a Type II error (recall from hypothesis testing).

Because the predictions, if they are incorrect, are off by an integer (once the classification scheme is applied), standard residuals are not the usual mechanism to test the quality of the model. Instead, a table called a confusion matrix is typically used. In a confusion matrix, there are typically 4 cells (in a 2x2 matrix), where one cell is the correctly classified 0s, one cell is the incorrectly classified 0s, one cell is the incorrectly classified 1s, and one cell is the correctly classified 1s. Adding the two cells together for correct classifications, and divided by the total number of observations in the data set, provides a

percentage that indicates the quality of the model. The higher the percentage of correct classifications the better.

For example, suppose the following is the confusion matrix generated from our analysis.

		Predicted	
		0	**1**
Actual	**0**	30	12
	1	8	56

The total number of observations is $113 = (30 + 12 + 8 + 56)$. The total number of correct classifications is $86 = (30 + 56)$. So the accuracy of our model is 76% ($\frac{86}{113}$). This is pretty low, actually. We'd generally prefer it much higher.

The error rate is 1 minus the accuracy. Here, that is $1 - 0.76 = 24\%$.

Ideally, we'd want to employ classification models (this or another type) when they perform as well as or better than human classifiers.

There are two additional metrics we can calculate from the confusion matrix: recall and precision.

Recall is the ratio of the true positive to all the observations that should have been positive (the false negatives). In the example above that is $\frac{56}{8+56} = \frac{56}{64} = 87.5\%$.

Precision is the ratio of the true positives to all those that the model predicted were positive. In the example above that is $\frac{56}{12+56} = \frac{56}{68} = 82.3\%$.

If models have high recall and low precision or vice versa, it can be a challenge to compare the models, so another measure is computed, called the F-score for this purpose. Let R be the recall value and P the precision.

$$F = \frac{2RP}{R + P}$$

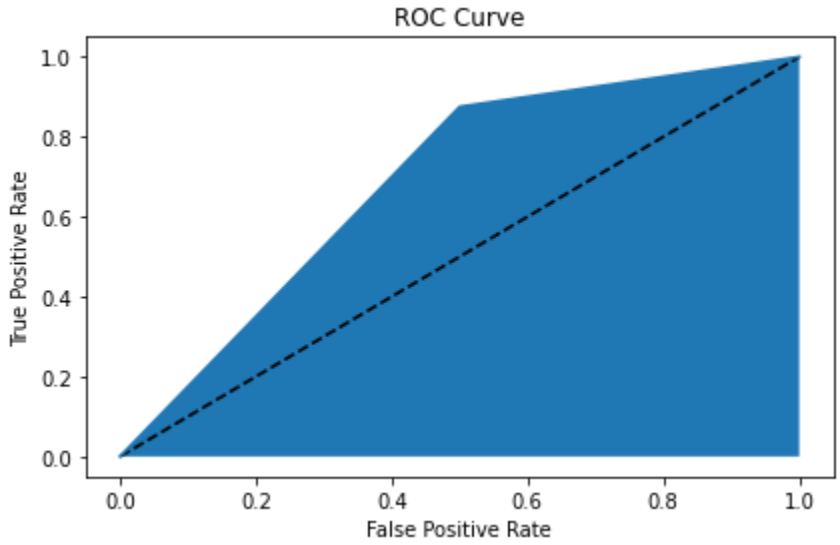
For the table above, that would give us:

$$F = \frac{2(0.875)(0.823)}{0.875 + 0.823} = 0.84848\ldots$$

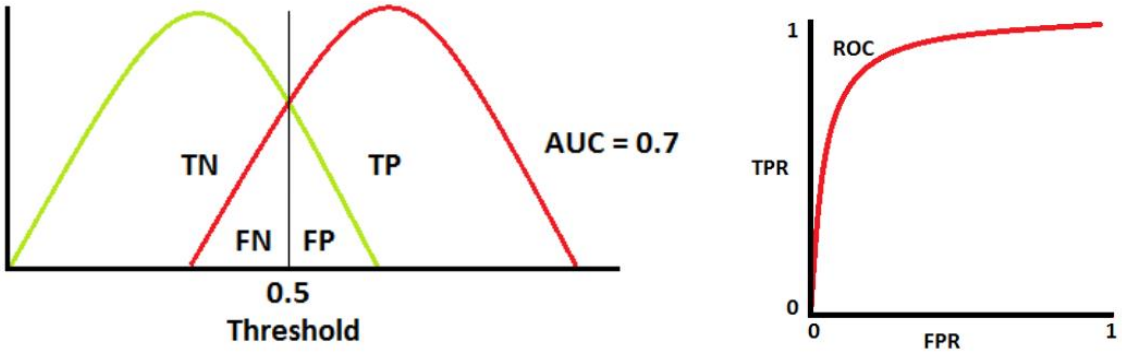
Since our recall and precision were both high, we got a value similar to them both, but this value is most useful when the recall and precision are most different.

One potential problem with classification that can occur is masking: if the number of observations that are one of the classes greatly outnumbers the other class, the model may conclude that it is better to guess everything is in one class, rather than put anything in the second class. This may produce higher overall accuracy, even though everything in the smaller class is predicted incorrectly. Sometimes this can create extremely problematic results, particularly when the classes are related to race, or if the data used for the training has some other kind of bias built in. The model will then replicate that bias. It may be necessary to adjust the balance of the data to make the errors more fair, though this may sacrifice overall accuracy. Classification models are not value-free.

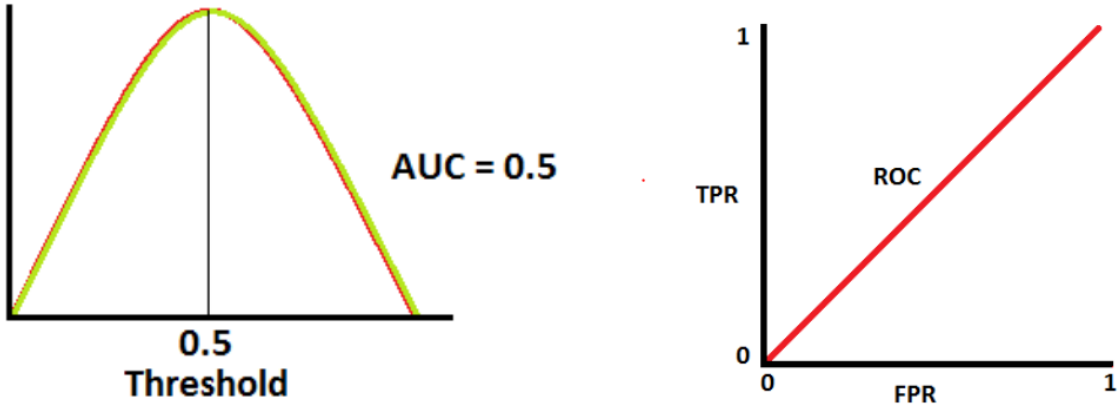
It was noted that it's possible to change the threshold for the classifier from the default of 0.50. How do we decide what it should be? For that, we can use an ROC curve. The curve is plotted by setting the threshold at different levels and seeing how the classifier behaves in each case. The AUC, or area under the curve, is a measure of how good the classifier is overall. One can think of it as a measure of the separability of the two classes. The higher the AUC, the better the classifier is. Here is an example of an ROC plot (this one was built in Python). There is a sharp break at 0.5, which is not that uncommon. The curve allows you to see the trade-off for adjusting the threshold. You can catch more true positives or true negatives, but generally not both.



A model similar to the one with the output in the table above (with approximately 70% accuracy), produces an ROC curve like this:

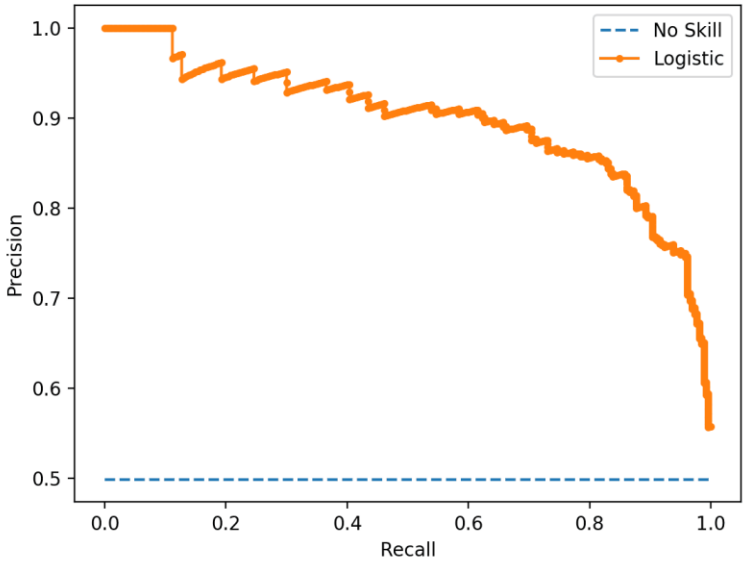


TN is the true negatives (true 0), TP is the true positives (True 1), FN is the false negatives, and FP are the false positives. If the classifier had higher accuracy, then the initial rise of the curve can be even more sharp. If the classifier has a lower accuracy overall, say, 50% (chance), the slope will be a straight line with a slope of 1.



If the accuracy is even lower, the concavity flips, but if it's really that bad (less than chance), there is no point in using the model at all.

Alternatively, you can compare precision and recall (values we calculated above). In this case, the "no skill" classifier (just chance) looks like a flat horizontal line. The curve shows the trade off between precision and recall, which, as we saw above, relate the ratio of true positives to actual positives or predicted positives.



We've considered the case of binary outputs. What if we don't have a binary output? What can be done then?

If the values are ordinal, one option to consider is predicting values with a traditional linear model, and having some mechanism (such as rounding) to deal with predictions between the discrete categories. This has some of the problems we discussed previously with the 0-1 classifier.

Another alternative is to create multiple binary classifiers. Suppose you have three categories. One model would treat class 1 as success and classes 2 and 3 as failure. One model would treat class 2 as success and class 1 and 3 as failure. If the model predicts failure for both class 1 and class 2, then it is designated as class 3. One can do this with the entire dataset (this would require a plan to handle what if the prediction is both class 1 and class 2), or you could create the second model only from the failures of the first model. One potential problem, here, though, is that one might end up creating a masking effect, depending on how many classes there are, and their relative sizes.

Logistic regression is not the only possible method of doing classifications. We will briefly discuss other methods later in the course when we review machine learning more broadly, but many other classification methods have a similar limitation of being primarily a binary classifier, so some schemes for those methods can also be applied to logistic regression.

References:

1. https://faculty.ksu.edu.sa/sites/default/files/probability_and_statistics_for_engineering_and_the_sciences.pdf
2. https://www.geeksforgeeks.org/how-to-plot-a-logistic-regression-curve-in-r/
3. https://stats.oarc.ucla.edu/r/dae/logit-regression/
4. https://www.digitalocean.com/community/tutorials/confusion-matrix-in-r
5. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
6. https://www.kaggle.com/code/vithal2311/auc-roc-curve-confusion-matrix-explained-in-detail

7. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/
8. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/