What Is Multicollinearity?

Multicollinearity is a problem that many a data scientist will come across in the problems they solve. It's a situation where the explanatory variables are nearly perfectly correlated with other. In this situation, it becomes difficult to use linear regression via OLS or gradient descent because the technique cannot accurately estimate the regression coefficients, often resulting in inflated values for these parameters. This is because it's hard to distinguish the effect of one explanatory variable from another, and subsequently each explanatory variable's effect on the response variable. As a product of multicollinearity, we observe the value of the regression coefficients changing, sometimes drastically, each time we initialize a linear regression algorithm. Ultimately, this renders traditional linear regression as a less preferable method for handling data that exhibits these types of patterns.

Testing for Multicollinearity

Very highly positive regression coefficients are one of the first tell-tale signs of multicollinearity. In addition to this, we should calculate the correlation of all the explanatory variables with each other. Correlation coefficients of $.95 \leq \rho \leq 1$ should also raise red flags in the mind of a data scientist. Specifically, though, there is a statistic that we can use to determine whether we most definitely have multicollinearity in our data set, called variance inflation factor.

Variance Inflation Factor (VIF)

The VIF statistic is calculated on a range from $0 \leq VIF \leq \infty$. Typically, the rule of thumb is that any VIF score that is > 5 indicates multicollinearity, and any score above 10 indicates severe multicollinearity. The statistic is calculated by regressing a given explanatory variable against the others and then using the result to calculate the coefficient of determination, yielding the following:

$$VIF_j = \frac{1}{1 - R_j^2}, \text{ Where } j = 1, \ldots, k$$

Ridge Regression

To combat multicollinearity specifically, ridge regression was developed and is a useful technique. Relevant to our discussions of norms earlier (L1 versus L2), ridge regression uses an L2 norm to achieve an optimal solution. Here is the equation for ridge regression:

$$\arg\min_{\beta} \|y - X\beta\|_2^2 + \lambda \|\beta\|_2$$

One of the key distinctions in ridge regression is the tuning parameter λ, which determines the degree to which the regression coefficients shrink. The technique gets the name *ridge* due to the fact that the L2 norm forms a spherical or circular shaped region where the optimal solutions for the regression coefficients exist are chosen along the "ridge" of this shape. Visually, this often looks like Figure 3-9.

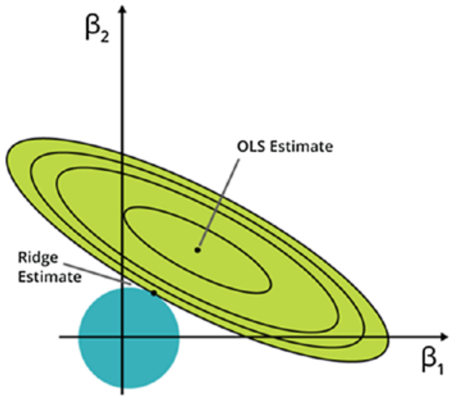


Figure 3-9. *Ridge regression OLS estimates*

Least Absolute Shrinkage and Selection Operator (LASSO)

Lasso is very similar to ridge regression except LASSO performs variable selection while regressing the explanatory and response variables. The key differentiation between LASSO and ridge regression is the fact that LASSO uses the L1 norm rather than the L2 norm, giving the selection region a square or cubic shape depending on the dimensionality of the data. In Figure 3-10, we can see the LASSO OLS estimate:

$$\arg\min_{\beta} \|y - X\beta\| + \lambda \|\beta\|$$

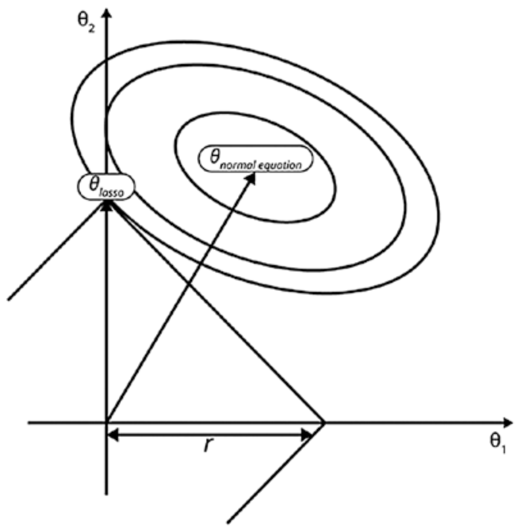


Figure 3-10. *LASSO regression*

Comparing Ridge Regression and LASSO

Both methods are highly useful for instances in which your data suffers from multicollinearity, but in instances where you're seeking to fit data as you would in a simple linear regression, these methods should be avoided, and the gradient methods along with the OLS method given earlier should be used. If you don't have more than one explanatory variable, these methods won't be of use to you. Although that's unlikely to be the case in practical terms most times, it's important to remember nonetheless.

Evaluating Regression Models

Beyond just building regression models, we need to find a way to determine how accurately the results yielded from a model are, and ultimately choose the best one on a case-by-case basis. In the case of regression, a useful method of evaluating machine learning models is by bootstrapping. Typically, *bootstrapping* involves running different regression models over several iterations using a data set that's smaller than the original, and with the original observations in randomized order, and then sampling several statistics and comparing their values relative to the other models' values. The process is as follows:

1. Build several models.
2. Collect sample statistics that we use as evaluators of each model over N iterations of the experiment.

3. Sample each of these evaluators and collect statistics upon each iteration, such as:
    a. Mean
    b. Standard deviation
    c. Max
    d. Min
4. Evaluate the results and pick the model that's most effective given your objectives and situational constraints.

The rest of this section covers the evaluators you should pick during bootstrapping.

Coefficient of Determination (R^2)

As described in Chapter 2, the coefficient of determination is what we use to evaluate how accurately a model explains variability in y through the variability in x. The higher the R^2 value, the better. That said, generally speaking, "good" R^2 values should be in the following range: $.70 \le R^2 \le .95$. Anything lower than .70 should be viewed as generally unacceptable, and anything higher than .95 should be examined to see if there is overfitting in the model. Although this won't change across a given iteration very much, we still should evaluate this objectively across models.

Mean Squared Error (MSE)

The MSE measures the distance of a given predicted value of y from the average value of the actual response variable. Our objective with any regression model is to minimize this statistic as much as possible, so we will want to pick the model that has the lowest MSE relative to the others being examined. This will be the evaluator that shows the most variance across models and should be the one that gives us the most inferential power with respect to which model we should choose.

Standard Error (SE)

In the case of a regression model, we would probably measure the standard error of a given model. The objective we should have should be to have a standard error that is as close to 0 as possible. Highly negative or highly positive standard error values are generally undesirable.

Classification

Moving beyond the case of predicting specific values, our data observations often belong to some class that we would like to label them as such. We refer to this paradigm of problems as *classification problems*. To introduce readers to these types of problems, we begin by addressing the most elementary of these algorithms: logistic regression.

Logistic Regression

In addition to regression, one of the important tasks of machine learning is classification of an observation. Although there are multinomial classification algorithms, we will start by examining a *binary classifier*, a method often used as a baseline for the remainder of the classifiers. Logistic regression gets its name from the function that powers it, known as the *logistic function*, illustrated in Figure 3-11.

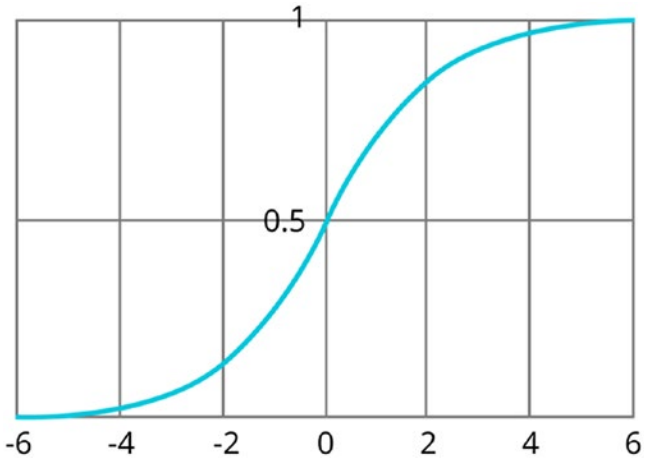


Figure 3-11. *Visualization of logistic function*

The function itself reads this this:

$$f(x) = \left(\frac{1}{1+e^{-x}} \right)$$

The intuition behind how we classify an observation is simple: we set a threshold for a given *f(x)* value and then classify it as a 1 if it meets or exceeds this threshold and a 0 if otherwise. In many contexts, the *x* variable will be replaced by a linear regression formula, in which we model the data. As such, the equation for *f(x),* or the log odds, will be

$$\pi = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$$

where π = log odds and π^* is the given threshold. As for the threshold we establish, that depends on what we would like to maximize: accuracy, sensitivity, or specificity.

- *Sensitivity/recall*: The ability of a binary classifier to detect true positives:

$$True\ Positive\ Rate = \frac{True\ Positives}{Positives} = \frac{True\ Positives}{True\ Positive + False\ Negatives}$$

- *Specificity*: The ability of a binary classifier to detect true negatives:

$$Specificity = \frac{True\ Negative}{Negatives} = \frac{True\ Negatives}{True\ Negatives + False\ Positives}$$

- *Accuracy*: The ability of a binary classifier to accurately classify both positives and negatives:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Positives + Negatives}$$

In certain contexts, it may be more advantageous to magnify any of these statistics, but that's all relative to where these algorithms are being applied. For example, if you were testing the probability of a phone battery combusting, you would probably want to be certain that false negatives are minimized as much as possible. But if you were trying to detect the probability that someone is going to find a match on a dating website, you probably would want to maximize true positives. The relationship between the tradeoff of these predictive abilities is most easily exemplified using an ROC curve, which shows how altering the value of π^* affects the classification statistics of the model.

Receiver Operating Characteristic (ROC) Curve

The ROC curve initially was used during World War II for the purposes of radar detection, but its uses were soon considered for other fields, statistics being one of them. The ROC curve displays the ability of a binary classifier to accurately detect true positives and simultaneously check how inaccurate it is by displaying its false positive rate. This is shown in Figure 3-12.

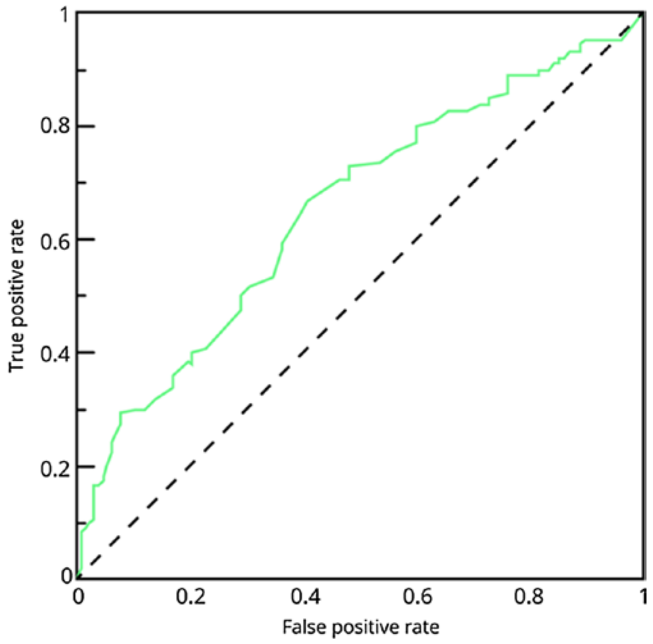
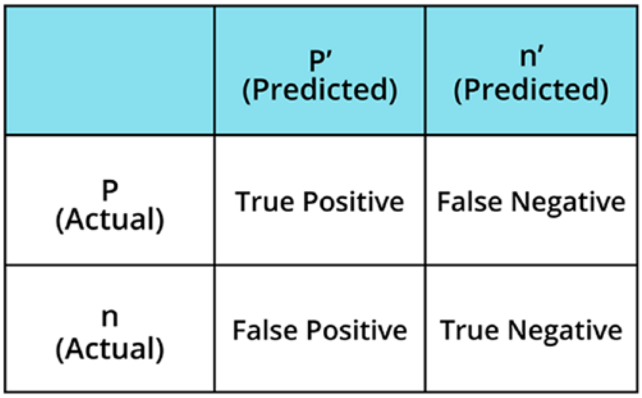


Figure 3-12. *Example ROC curve plot*

In the context of logistic regression, the evaluation of any specific model, given a specific threshold for π, is ultimately determined by the *area under the curve*, or AUC. The vertex of the plot is the .50 AUC score, which indicates that the model, should its score be this, is no better at classifying than a random guess. Ideally, this AUC score would be as close to one, but we generally accept anything ≥ .70 as acceptable.

Confusion Matrix

Another method of evaluating classification models is the *confusion matrix*, a graphical representation of the classifiers predictions against the actual labels of a given observations. From this visualization, we derive the values for the statistics listed previously that ultimately help us accurately evaluate a classification model's performance. Figure 3-13 shows a visual example of a confusion matrix.



	P′ (Predicted)	n′ (Predicted)
P (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

Figure 3-13. *Confusion matrix*

Interpreting the values within a confusion matrix often is a subjective task that is up to the reader to determine. In some instances, false positives, such as determining whether users should buy a product or not, will not be as detrimental to solving the problem at hand. In other cases, such as determining whether a car engine is faulty, false positives may actually be detrimental. Readers should be conscious of the task they are performing and tune the model to limit the false positives and/or false negatives accordingly.

Limitations to Logistic Regression

Logistic regression can only predict discrete outcomes. It requires many of the assumptions necessary for ordinary linear regression, and overfitting of data can become quite common. In addition to this, classification with logistic regression works best when we have data that is clearly separable. For these reasons, in addition to the fact that there are more sophisticated techniques available, it is a common modeling practice to consider the logistic regression model to be the baseline by which we juxtapose other classification methods and observe the nuances.

Moving forward, we will look at a simple example using logistic regression. This data set will be referenced in later chapters, and in Chapter 10 in detail, for those who are curious about the process by which this model was produced.:

```
#Code Redacted, please check github!
#Logistic Regression Model
lr1  <- glm(data[,1] ~ data[,2] + data[,3] + data[,4] + data[,5] + data[,6]
+ data[,7],
            family = binomial(link = "logit"), data = data)

#Building Random Threshold
y_h <- ifelse(lr1$fitted.values >= .40, 1, 0)
```

```
#Construct ROC Curve
roc(response = data[,1], predictor = y_h, plot=TRUE, las=TRUE,
    legacy.axes=TRUE, lwd=5,
    main="ROC for Speed Dating Analysis", cex.main=1.6, cex.axis=1.3,
    cex.lab=1.3)
```

The ROC curve for our model is shown in Figure 3-14.

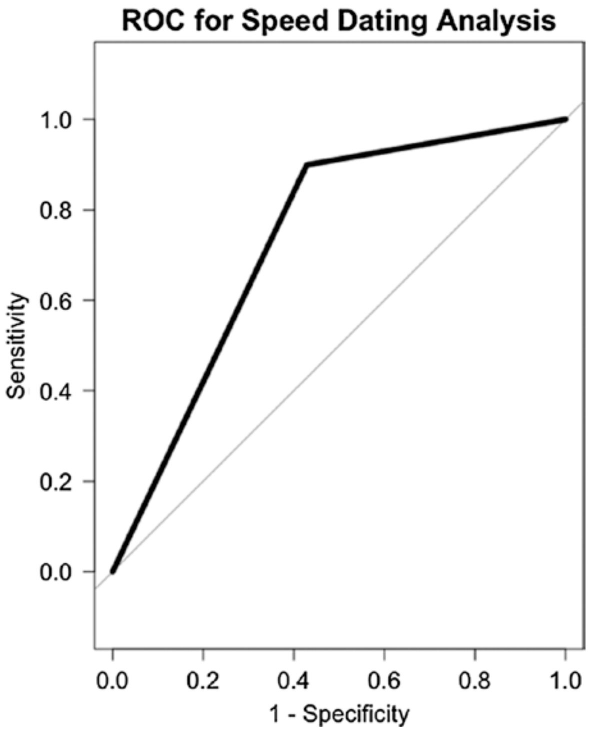


Figure 3-14. *ROC curve for logistic regression example*

Using the preceding code, we have an area under the curve of .7353. Given the threshold that we set before, this model's performance would be considered acceptable, but it should likely undergo more tuning.

Support Vector Machine (SVM)

Among the more sophisticated machine learning models available, *support vector machines* are a binary classification method that has more flexibility than the logistic regression model in that they can perform non-linear classification. This is performed via its kernel functions, which are equations that orthogonally project the data onto a new feature space, and the classification of the objects are performed as a product of two hyperplanes constructed by a norm (See Chapter 2).

In the case of liner SVMs, we take in as our inputs a response variable, Y, and an explanatory variable(s), x. We orthogonally project this data into feature space, such that we form the hyperplane that separates the data points. The size of these hyperplanes is determined by the Euclidean norm of the weights, or w, vector in addition to an upper bound and lower bound, respectively denoted as the following:

$$wx + b = 1$$

$$wx + b = -1$$

We keep reiterating this process until we have reached a norm of w that maximizes the separation between the two classes. The separation of the classes is maximized by minimizing ||w||, being that the size of the hyperplane is given by the following:

$$\frac{2}{\|w\|}$$

The following constraints also prevent us from allowing observations to fall in between the two hyperplanes:

$$wx + b \geq 1,\ \text{if y} = 1$$

$$wx + b \leq -1,\ \text{if y} = -1$$

The observations that ultimately fall on the boundaries of the hyperplane are the most important, as they are the "support vectors" that define the separation between classifications. This transformation is shown in Figure 3-15.

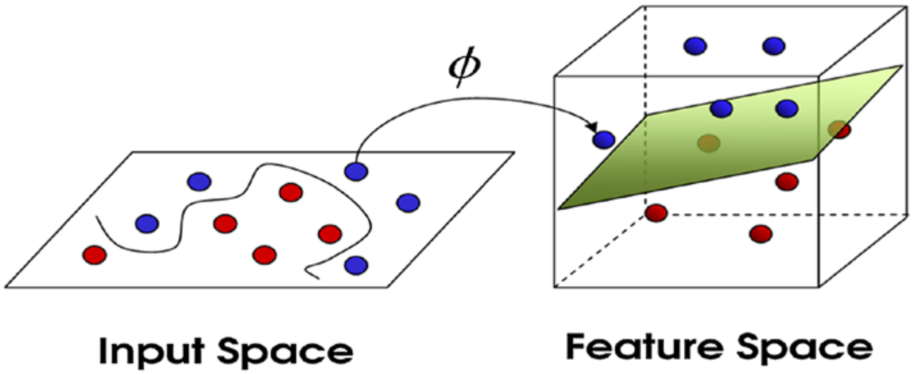


Figure 3-15. *Orthogonal transformation of data via kernel function*